FELFÖLDI JÓZSEF
SZ. LUKÁCS JÁNOS

# GÉPI KÓDÚ PROGRAMOZÁS

## ENTERPRISE 128

Felföldi József – Sz. Lukács János

# GÉPI KÓDÚ PROGRAMOZÁS

## ENTERPRISE 128

NOVOTRADE Rt.

Lektorálta: Bányi László
Szerkesztette: Stankovics János
SCAN: GAFZ, 2003

A kiadásért felel RÉNYI GÁBOR, a Novotrade Rt. vezérigazgatója
Budapest, 1990

Műszaki szerkesztő: Erdősi Zoltán

Szedte: NOVOTRADE Rt. – TYPOART Kiadványszerkesztő Iroda

Készült a Somogy megyei Nyomdaipari Vállalat kaposvári üzemében
Felelős vezető: Mike Ferenc igazgató

ISBN 963 585 047 6

# Tartalomjegyzék

# Bevezetés

Minden személyi számítógéppel ismerkedő, a gépet kreatívan használni akaró alkalmazó eljut — előbb vagy utóbb — arra a szintre, hogy érezni kezdi a BASIC programozási rendszer korlátait. Akár az elérhető működési sebességet érzi túl kicsinek, akár olyan különleges hatások keltik fel a figyelmét, amelyekkel gyári programokban, esetleg más gépeken találkozott — a felhasználók jelentős része érdeklődni kezd a gépi kódú programozás iránt.

Így van ez az ENTERPRISE esetén is. Ki kell emelni azonban, hogy nemigen találkozhattunk még a home computer (házi számítógép) kategóriájú, nálunk is elterjedt gépek között ennyire komfortos, a látványos, dinamikus programozást ilyen mértékben segítő BASIC értelmezővel.

Mire számíthat az, aki rászánja magát erre az útra, a gép mikroprocesszorának közvetlen utasításokkal való vezérlésére, programozására? Először bizony meglehetősen sok buktatóra, nem éppen kellemes meglepetésre. Az a baki, amit a BASIC rendszer ilyen-olyan (pl. szintaktikai) hibának minősített, a gépi kódú program futtatása során sajnos gyakran a rendszer összeomlásához vezethet, esetleg lefagy a gép (olyan végtelen ciklusba kerül, amiből csak a kikapcsolás menti ki). A gépi kódú programokat tehát sokkal fegyelmezettebben, körültekintőbben kell megírni, a tárban elhelyezni, majd futtatni, mint pl. a BASIC programokat. De a haszna is megvan a többletenergia ráfordításának. Ha csodákat nem is várhatunk a programok gépi szinten való megírásától (hiszen adott a fizikai háttér, az ún. hardver: a processzor működési sebessége, címzési kapacitása, a képmegjelenítés technikája stb.), azt mindenképpen mondhatjuk, hogy így sokkal hatékonyabb programokat tudunk készíteni.

Mit jelent ez a hatékonyság? Három fő területet lehet kiemelni:

1. A gépi kódú program jóval tömörebb, mint a BASIC. Egy-két kilobájtnyi gépi programmal már egészen összetett feladatokat lehet megoldani.

2. A futási sebesség egyes területeken nagyságrendekkel nagyobb, mint a hasonló feladatot megoldó BASIC program esetén. Van persze más magas szintű nyelv is (pl. a PASCAL), ami egyesíti a programozási komfortot és a nagy működési sebességet, de a mikroprocesszor közvetlen programozásával a lehető leggyorsabb programfutás érhető el.

3. A közvetlen programozással semmiféle olyan szoftverkorlátozást nem kötelező figyelembe vennünk, amit a rendszer tervezői — pl. az egységes kezelés érdekében — szabtak. Hogy egy példát mondjunk erre: a képmegjelenítést végző — hallatlanul sokoldalú — NICK-chip lehetővé tenne akár pont-so-

ronként különböző video üzemmódokat (szöveges vagy grafikus módok, színválaszték stb.), de az operációs rendszer beépített VIDEO kezelője csak karaktersoronként engedélyez módváltást. Ez is sokkal több a COMMODORE vagy ZX gépek lehetőségeinél. Mindenesetre igen vonzó lehetőség, hogy gépi kódú programozással elérhetjük a fizikai lehetőségek elvi határait.
Mindezeken kívül vannak a szoftvernek olyan területei, amelyek elvileg is elérhetetlenek magas szintű nyelvekből (ilyen pl. a megszakítási rendszer), amelyek programozása csakis gépi kódban képzelhető el.

A gépi szintű programozáshoz rendelkezésre álló szakirodalom meglehetősen felemás. Míg bőven találunk a Z80-as mikroprocesszor utasításkészletével, programozásával foglalkozó könyveket, cikkeket, addig ezek ENTERPRISE gépen való alkalmazásával alig foglalkozik kiadvány. Nagyon sokat segített ezen a helyzeten a számítógép operációs rendszerének (EXOS 2.1-es verzió) leírása, amely nagyon sok hasznos információt nyújt az operációs rendszer felhasználásával kapcsolatban. Példákat azonban nem közöl, így a gépével most ismerkedő Olvasó számára viszonylag nehezen követhető az igényes, magas színvonalú leírás. Sokaknak hiányzik — tapasztalataink szerint — a gép működési logikájának leírása is, annak követése, hogy mi miért és hogyan történik a gépben. Nem foglalkozik a leírás a BASIC értelmezővel sem (hiszen ez már felhasználói program), pedig ennek bővítése, rutinjainak felhasználása ugyancsak hálás területe a gépi kódú programozásnak.

Mindezek alapján érthető, hogy könyvünket ezen hiányok pótlására, az érezhető igények kielégítésére szántuk. Mivel feltevésünk szerint az országban elterjedt többezer gép tulajdonosa — a BASIC-et már többé-kevésbé ismerő, de a gépi kódú programozással most ismerkedő fiatal — veszi elsősorban kezébe a könyvet, az 1. fejezetben röviden összefoglaljuk a gépi kódú programozáshoz szükséges alapismereteket. Ezután végigkövetjük előbb a háttérrendszer — az EXOS — majd a BASIC rendszer működését, felhasználását (sok példával, végig az alkalmazást, saját elképzeléseink megvalósítását, rutinjaink beillesztését tartva szem előtt).

A kényszerűség néha pozitív mellékhatásokkal is jár. A szakirodalom hiánya arra ösztönzi a felhasználókat, hogy kísérletezve, próbálkozva fedezze fel gépe tulajdonságait. Azt tanácsoljuk az Olvasónak, hogy kövesse azt az utat, amit — többek között — mi is követni kényszerültünk. Kísérletezzen, próbálja ki a példákat, a változtatási lehetőségeket, azok hatásait. Ez az út talán egy kicsit időigényes, nem zárja ki a tévedés lehetőségét sem, de minden bizonnyal a gép alapos megismeréséhez vezet.

# 1. A gépi kódú programozás alapjai

Mint a bevezetőben írtuk, az ENTERPRISE személyi számítógépek mikroprocesszoráról — a Z80-asról —, annak programozásáról igen sok színvonalas szakkönyv jelent meg. Ezen a területen tehát nincs szükség hiánypótlásra. A következő összefoglalással inkább azt szeretnénk biztosítani, hogy összeállításunk önmagában is használható legyen, akár a gépével, a számítástechnika e területével most ismerkedő felhasználó számára is. Aki tehát jól ismeri a Z80-as utasításait, alkalmazását, át is lapozhatja ezt a fejezetet — leglábbis első nekifutásra.

A gépi kódú programozás iránt kétségtelenül sokakban meglevő idegenkedést — tapasztalataink szerint — az alapfogalmak tisztázatlansága is erősíti. Mindenekelőtt foglaljuk össze a gépi szintű programozás alapvető fogalmait, kifejezéseit.

## 1.1 Alapfogalmak

A számítógép működése során minden utasítást, adatot a különböző memóriák tárolnak. A memóriák elemi egységei, cellái egyetlen információt, egy *bitet* képesek megőrizni, aminek értéke 0 vagy 1 lehet. Valójában két jól megkülönböztethető fizikai állapotról van szó (pl. nincs feszültség — van feszültség), amit a 0 és 1 kódokkal jelölünk. A memória *törlése* azt jelenti, hogy 0 állapotba hozzuk, *beírása* pedig az 1 állapotba billentést jelenti. Az olyan memóriát, amelynek rekeszeibe tetszés szerint írhatunk 0 vagy 1 értéket, RAM-nak nevezzük (Random Access Memory — tetszés szerint elérhető, azaz írható, olvasható memória). Vannak olyan memóriák is, amelyek celláinak állapotát gyárilag beégették, ezek a ROM-ok (Read Only Memory — csak olvasható memóriák). Ezek alkalmasak a gépet működtető alapprogramok, belső adatok (pl. PI) stb. tárolására.

A memória bitjei — a Z80-as fizikai sajátosságainak megfelelően — 8-as csoportokba, ún. bájtokba (gyakori a byte írásmód is) vannak rendezve. Egyetlen utasítással egy ilyen 8-bites blokk olvasható vagy írható. Egy bájt értéke már természetesen sokkal többféle lehet: az alkotó bitek értékétől függően 00000000-tól 11111111-ig 256 különféle értéket vehet fel. Jól látható, hogy a bájtok megadására, kezelésére kínálkozó egyik segédeszköz a kettes számrendszer. A bájtok által képviselt *bináris* (kettes számrendszerbeli) érték meghatározásánál a legutolsó bitet tekintjük a legalacsonyabb helyi értékűnek (ez a 0. bit), és előre haladva az egyes bitek egyre magasabb helyi értéket képviselnek. A bájtot alkotó legnagyobb értékű bit lesz a 7. Szokásos még az LSB (Least Significant Bit — legkisebb értékű bit), ill. az MSB (Most Significant Bit — legnagyobb értékű bit) megjelölés is.

Az n. bit értéke a bájtban $2^n$:

BIT 0: 1
BIT 1: 2
BIT 2: 4
BIT 3: 8
BIT 4: 16
BIT 5: 32
BIT 6: 64
BIT 7: 128

Így lesz pl. a 00000101 bájt értéke decimálisan (azaz a megszokott tizes számrendszerben) $1 \cdot 2^2 + 1 \cdot 2^0 = 5$. A bináris—decimális átalakítást segíti az ENTERPRISE BASIC-je (BIN(x)), de a decimális számok kettes számrendszerbe alakításához nekünk kell segédrutint készítenünk:

```
100 DEF BIN$(X)
110   LET B$=""
120   LET B=128
130   FOR N=0 TO 7
140     LET BN=INT(X/B)
150     LET B$=B$&STR$(BN)
160     LET X=X+B*(BN=1)
170     LET B=B/2
180   NEXT
190   LET BIN$=B$
200 END DEF
210 !
220 INPUT PROMPT "Atirando szam? ":SZ
230 IF SZ<0 OR SZ>255 THEN 220
240 PRINT "Binaris alak: ":BIN$(SZ)
250 GOTO 220
```

Érdemes ezt a rutint (vagy hasonlót) beírni a gyakorlás lehetőségén kívül azért is, mert jól használhatjuk majd a gépi kódú programok nyomkövetésénél.

A bináris alak nagyon praktikus akkor, ha egy-két kiemelt bit értékét kell figyelnünk, vagy pl. a grafikában, ahol a megjelenő alakzat pontjai egy az egyben megfeleltethetőek a bitek értékeinek, de túlzottan körülményes az átlagos számolásokhoz, kódok, címek megadásához stb. A gépi kódú programozásban igen elterjedt a bájt felépítését továbbra is tükröző, de a binárisnál jóval tömörebb tizenhatos számrendszerbeli *(hexadecimális)* megadás. A hexadecimális számjegyek jelölésére — jobb híján — a decimális jegyeket (0—9) és az A—F betűket használjuk:

| Decimális | Bináris | Hexadecimális |
|---|---|---|
| 0 | 0000 | 0 |
| 1 | 0001 | 1 |
| 2 | 0010 | 2 |
| 3 | 0011 | 3 |
| 4 | 0100 | 4 |
| 5 | 0101 | 5 |
| 6 | 0110 | 6 |
| 7 | 0111 | 7 |
| 8 | 1000 | 8 |
| 9 | 1001 | 9 |
| 10 | 1010 | A |
| 11 | 1011 | B |
| 12 | 1100 | C |
| 13 | 1101 | D |
| 14 | 1110 | E |
| 15 | 1111 | F |

Egy hexadecimális jegy tehát 4 bit minden lehetséges értékét képes leírni. A bájt értékét ezek szerint két hexadecimális jegy adja meg. Az első a tizenhatosok számát, a második az egyesek számát tartalmazza (mint ahogy a tízes számrendszerben pl. 35 esetén 3 a tízesek száma, 5 az egyesek száma). Így pl. az A5 hexadecimális szám értéke tízes számrendszerben;

$$10 \cdot 16^1 + 5 \cdot 16^0 = 165$$

A továbbiakban az egyértelműség érdekében a hexadecimális alakot a szám után írt H betűvel jelöljük. Például:

$$32H = 3 \cdot 16^1 + 2 \cdot 16^0 = 50$$

Megjegyezzük, hogy a téma szakirodalmában gyakori a #32 vagy $32 típusú jelölés is. A 0 és 255 közötti számok hexadecimális alakját (00H—FFH) az 1. függelék tartalmazza.

A memóriák elérésére, megcímzésére a Z80-as processzornak 16 vezetéke van, amelyek mindegyike 0 vagy 1 állapotú lehet. Egy-egy memóriahely címét (ahol 1-1 bájt adat van elhelyezve) ezek szerint egy 16 bites bináris szám (az ún. "szó", "word") határozza meg. A processzor tehát $2^{16} = 65\ 536$ memóriahelyet tud megkülönböztetni, megcímezni (és oda adatot írni vagy onnan olvasni). Hogy az ENTERPRISE gépek esetén ennél jóval nagyobb memóriát is képes kezelni, az már egy technikai trükknek köszönhető (erről a következő fejezetben részletesebben is írunk).

Hogyan lehet egy címet megadni? Alapvetően természetesen egy 16 bites bináris

számmal. A előbbiek alapján logikusan következik, hogy jóval praktikusabb, tömörebb két bájttal (4 hexadecimális jeggyel) megadni ugyanazt az értéket. Például:

$$1100\ 0010\ 0101\ 0100B = C254H$$

(a bináris értéket B-vel jelöljük)

Az első 8 bitet szokás felső (High) bájtnak, a második 8 bitet alsó (Low) bájtnak nevezni. A felső bájt tulajdonképpen a kétszázötvenhatosok számát adja. Így az előbbi cím decimális értéke:

$$C2H = 194$$
$$54H = 84$$
$$\text{cím} = 194 \cdot 256 + 84 = 49\,748$$

A decimális—hexadecimális oda-vissza konverzióra feltétlenül érdemes rutint készíteni, olyan gyakran lesz szükség az átszámításra:

```
100 DEF H$(N)=CHR$(INT(N/16)+48-7*
    (INT(N/16)>9))&CHR$(MOD(N,16)+
    48-7*(MOD(N,16)>9))
110 DEF W$(WORD)=H$(INT(WORD/256))&
    H$(MOD(WORD,256))
120 DEF DEC(H$)
130   LET D=0:LET Q=1
140   FOR N=LEN(H$) TO 1 STEP-1
150     LET D=D+ORD(HEX$(H$(N:N)))*Q
160     LET Q=Q*16
170   NEXT
180   LET DEC=D
190 END DEF
```

```
PRINT H$(127)
7F
ok
PRINT W$(16383)
3FFF
ok
PRINT DEC("F"),DEC("FF"),DEC("FFFF")
 15          255          65535
ok
```

Az egyszerűség (és gyorsaság) érdekében a H$(N), ill. W$(N) rutin nem ellenőrzi az argumentum értéktartományát (0—255, ill. 0—65 535). Szükség esetén ezt a hívónak kell megtennie.

A bit, a bájt és az összetartozó kétbájtos érték a gépi kódú programozás leggyakrabban használt egységei. Nagyobb memóriatartományok meghatározására gyakori még a kilobájt (kbyte, 1024 bájt) egység is.

A számítógép minden műveletet (a gyökvonástól az ellipszisrajzolásig) ilyen

egy- és kétbájtos adatok mozgatásával, rajtuk és közöttük elemi műveletek elvégzésével valósít meg. Tekintsük át az alapműveleteket is.

A bájtok között elvégzett minden (logikai, aritmetikai) művelet a bitek közötti műveletekből származtatható. Ezeket a legegyértelműbben az ún. igazságtáblázattal adhatjuk meg: az összes lehetséges esethez megadjuk a művelet eredményét.

Elsőként nézzük a logikai műveleteket, amelyeknél a művelet eredménye mindig 1 bit:

AND: logikai ÉS művelet

| *a* | *b* | *x* = *a* AND *b* |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

OR: logikai VAGY művelet

| *a* | *b* | *x* = *a* OR *b* |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

XOR: logikai KIZÁRÓ VAGY művelet

| *a* | *b* | *x* = *a* XOR *b* |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

NOT: komplementálás, invertálás

| *a* | *x* = NOT *a* vagy $x = \bar{a}$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

A bájtok közötti műveletet az azonos helyi értékű bitek között külön-külön elvégzett műveletek valósítják meg.

A műveletek használhatóságáról majd az utasításkészlet tárgyalása során ejtünk még néhány szót. Az ún. aritmetikai műveletek haszna egészen nyilvánvaló, a számtani alapműveletekről van szó:

összeadás $x = a + b$

| $a$ | $b$ | $x$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 10 |

Az utolsó sorra érdemes felhívni a figyelmet: 1+1 = 0 és marad 1. A túlcsorduló 1-es már a magasabb helyi értékű bitek értékeit növeli (hasonlóan, mint a tízes számrendszerbeli összeadásnál). Több bites számok (pl. két bájt) összeadásánál először a legkisebb helyi értékű biteket kell összeadni, majd sorra a nagyobb értékűeket, mindig hozzáadva az összeadandó bitekhez az előző összeadás átvitelét is. Magától értetődő, hogy — ha az összeg nagyobb 255-nél — a legmagasabb helyi értékű bitekről is keletkezhet átvitel. Valójában tehát a 8 bites számok összeadásához szükség van még egy 9. bitre is, amely az esetleges átvitelt, a túlcsordulást jelzi. Mint látni fogjuk, a processzor rendelkezik is ilyen bittel.

kivonás $x = a - b$

A kivonás az első problematikus művelet. A nehézségét az adja, hogy az eredmény negatív is lehet, ami nem illik bele a számábrázolás eddigi rendszerébe. Ha szükség van a negatív számok kezelésére is — márpedig ez a számítógép alkalmazások legtöbbjében elengedhetetlen —, meg kell egyezni a negatív értékek ábrázolásának módjában.

A Z80-as mikroprocesszornál választott megoldás az ún. kettes komplemens ábrázolás: az egybájtos számok tartományában a –b érték helyett a 256 – b értéket tárolja és használja a rendszer. Ez már újra egy bájton ábrázolható pozitív érték. Az így megadható számok értéke –128, és +127 között lehet:

| Előjeles érték | Tárolt érték | |
|---|---|---|
| | decimálisan | hexadecimálisan |
| –128 | 256 – 128 = 128 | 80H |
| –127 | 256 – 127 = 129 | 81H |
| . | | |
| . | | |
| . | | |
| –1 | 256 – 1 = 255 | FFH |
| 0 | 0 | 00H |
| 1 | 0 | 01H |
| . | | |
| . | | |
| . | | |
| 127 | 127 | 7FH |

Ez a megoldás több szempontból praktikus:

— a szám előjele az első bit (MSB) alapján egyszerűen megállapítható (ha ez 1, azaz a tárolt érték nagyobb mint 127, a szám negatív);
— a műveletek következetesen alkalmazhatók:

| | | |
|---|---|---|
| 07H | | 07H |
| –03H | helyett | +FDH |
| 04H | | 104H |

az egy bájton kapott 04H érték a helyes végeredmény;
— a kettes komplemens értéket egyszerűen megkaphatjuk a szám invertálásával (ez az 1-es komplemens) + 1 hozzáadásával:

$$(-b) \text{ tárolt értéke: } 256 - b = \overline{b} + 1.$$

Most már csak az a kérdés, hogy ha pl. egy memóriacímről FEH értéket olvasunk be, honnan tudjuk, hogy ezt 254-nek vagy –2-nek kell-e tekintenünk. Ha tömören akarunk válaszolni, azt mondhatnánk, hogy sehonnan. Valójában ezt csak az érték előélete szabhatja meg. Lényegében mi döntjük el, hogy egy művelet operandusait és eredményét előjeles számoknak vagy egybájtos pozitív egészeknek tekintjük-e. A processzor mindkét értéktartományt egyformán kezeli, figyeli és jelzi mindkét esetre a határátlépéseket (átvitel, túlcsordulás), lehetőséget adva ezzel a tetszés szerinti kiértékelésre.

Az elmondottakból logikusan következik, hogy ha kétbájtos értékek esetén akarunk előjeles számokat ábrázolni, –nn helyett ennek kettes komplemense (65 536 – nn) a megadandó érték (így –32 768 és 32 767 közötti számokat ábrázolhatunk).

Az eddigiek szerint a memóriaterület egy-egy címén tárolt egybájtos adat jelenthet 0 és 255 közötti egész értéket, vagy –128 és +127 közötti előjeles számot. Ezenkívül a rendszer még két kódolási formát alkalmaz az adatok tárolásánál, feldolgozásánál.
— A tízes számrendszerbeli számok ábrázolására az ENTERPRISE általában az ún. *BCD-kódrendszert* alkalmazza (Binary Coded Decimal = binárisan kódolt decimális). Ennek lényege, hogy a tárolt érték hexadecimális alakja magukat a számjegyeket adja meg (a két hexadecimális karakter két számjegyet):

| Kódolandó szám | Tárolt érték |
|---|---|
| 0 | 00H |
| 1 | 01H |
| . | |
| . | |
| . | |
| 9 | 09H |
| 10 | 10H |
| 11 | 11H |
| . | |
| . | |
| . | |

Ez a kódolási forma tehát csak annyiban tér el a már megismert hexadecimálistól, hogy tiltja annak A—F karaktereit. Így a lehetséges 256 különböző érték helyett egy memóriacímen csak 100-féle értéket tárolhatunk (0—99), de a memóriapazarlásért kárpótol minket az aritmetikai műveletek, a kijelzés egyszerűsége, sebessége, pontossága.

A következő program egy, az operációs rendszer által BCD-ben kezelt értéket olvas és ír ki folyamatosan: az óra másodperc értékét. A decimális alakon jól láthatók a BCD logika ugrásai (a tiltott állapotok átlépései), a hexadecimális kiírás viszont tökéletesen követhető:

```
100 DEF H$(N)=CHR$(INT(N/16)+48-7*
    (INT(N/16)>9))&CHR$(MOD(N.16)+
    48-7*(MOD(N,16)>9))
110 LET CIM_SEC=49010
120 TEXT
130 LET SEC=SPEEK(255,CIM_SEC)
140 PRINT AT 1,1:SEC,H$(SEC)
150 GOTO 130
```

— A szövegek, karakterek tárolásának alapja az ENTERPRISE-on az *ASCII kódrendszer.* Itt egy bájton egyetlen karaktert tárolunk, pontosabban annak kódját. Az alfanumerikus jegyek (számjegyek, matematikai jelek, az angol ABC betűi) nemzetközi szabvány szerinti kódjait (pl. A kódja 41H = 65Dec) az ENTERPRISE speciális karaktereinek kódjai egészítik ki.

A kódokhoz tartozó karakterek kísérletként közvetlenül is megjeleníthetők a képernyő legfelső sorában (az állapotsorban) a programfutás idejére (leállás után a BASIC rendszer felülírja ezt a területet):

```
100 LET CIM=48830
110 LET K=64
120 FOR I=0 TO 31
130   POKE CIM+I,K+I
140 NEXT
150 GOTO 150
```

Próbálja ki a rutint különböző K kezdőértékekkel! A kódokhoz rendelt karaktereket az 1. függelék is tartalmazza.

Az általános fogalmak összefoglalása után nézzük meg, hogy maga a Z80-as mikroprocesszor milyen feltételeket biztosít a gépi kódú programozáshoz.

Bár a processzor működésének lényege a külső memóriacímek olvasása, tartalmuk átírása, módosítása, de ennek segítésére jó néhány belső memóriarekesze (ún. regisztere) is van. Ezek a regiszterek általában 8-bitesek, de egy részüket párban — mint 16-bites regisztert — is képes használni a processzor.

*Regiszterek:*

— A 8-bites aritmetikai és logikai műveletek alapja az A-regiszter, az ún. akkumulátor. Ebbe kell tölteni a műveletek előtt az (egyik) operandust és ez tartalmazza a művelet eredményét.
— A flagregiszter (F) a művelet eredményére jellemző egybites információkat tartalmaz. Hat értékes bitje:

BIT 0: C (Carry = átvitel)
— értéke 1, ha az elvégzett művelet során átvitel keletkezett a 8. bitről a 9-re és fordítva (pl. A0H + B0H után).
BIT 1: N (kivonásjelző)
— belső használatú bit; az elvégzett művelet típusát jelzi.
BIT 2: P/V (paritás vagy túlcsordulás)
— logikai műveletek után: az akkumulátor paritását jelzi (0, ha *A*-ban páros számú bit 1-es),
— aritmetikai műveletek után: a kettes komplemens számábrázolási tartományátlépését jelzi (a 7. bitről a 8-ra és fordítva).
BIT 4: H (Half-Carry = fél-átvitel)
— ugyancsak belső használatú bit az alsó 4 bitről keletkezett átvitel jelzésére; a BCD számábrázolási módban az értékek korrekciójához használja a processzor.
BIT 6: Z (Zero)
— értéke 1, ha a művelet eredménye 0.
BIT 7: S (Sign, előjel)
— értéke 1, ha a művelet eredménye negatív (a kettes és egyes komplemens ábrázolási logikában ez tulajdonképpen az eredmény 7. bitjének másolata).

A 3. és 5. bit közvetlenül nem kérdezhető le, gyakorlatilag nincs kihasználva. Két dolgot már most érdemes kiemelni a jelzőbitekre vonatkozóan:
— A bitek a művelet eredményét, és nem az akkumulátor értékét tükrözik (pl. az összehasonlító utasításban (l. CP) az akkumulátor értéke nem változik, de a státusz lehet Z, C stb.).

— Sok művelet nem változtatja meg a jelzőbiteket (vagy nem mindegyiket). A ROM-programokban is gyakran látunk majd olyan esetet, hogy egy művelet után egy jóval későbbi utasítás kérdez rá a művelet eredményére.

*A regiszterkészlet további elemei:*

— Általános célú regiszterek: B, C, D, E, H és L. Ezek mint BC, DE és HL regiszterpárok is használhatók. Az univerzális felhasználhatóság mellett a B, C regisztereknek számlálóként lesz külön jelentőségük (ciklusok), míg a HL regiszterpárt sok utasítás használja 16-bites akkumulátorként, ill. a külső memória megcímzésére, a cím H (felső) és L (alsó) bájtjának megadására. Az A és F, valamint a B, C, D, E, H, L regiszterekből még egy teljes készlete van a processzornak; ezek a háttérregiszterek (AF', B'—L'), amelyek teljesen egyenértékűek az elsődleges megfelelőikkel és egyetlen utasítással kicserélhetők azokkal (l. EX AF, AF' ill. EXX utasítások). A processzor mindig az éppen „előtérbe hozott" regiszterekkel dolgozik.

— A IX és IY 16-bites indexregiszterek elsősorban a memória (pontosabban egy 256 bájtnyi memóriaintervallum) címzésére használhatók. Megjegyezzük, hogy (bár ez általában nem szerepel a leírásokban) az indexregiszterek alsó, ill. felső 8 bitje — az L, ill. H regiszterekre vonatkozó adatmozgató, logikai és aritmetikai utasítások kiterjesztéseként — külön is használható. Az IX és IY regiszterek — HL kiterjesztéseként — 16-bites akkumulátorként is használhatók.

— A veremmutató (SP = Stack-Pointer) ugyancsak 16-bites és egy külső RAM-területre kell mutatnia. Ez a terület, az ún. verem igen jól használható nagyobb regiszterigény esetén a belső regiszterek külső memóriába mentésére (l. PUSH és POP utasítások). A processzor szubrutinok hívása esetén itt tárolja a visszatérési címet. Az SP regiszterpár mindig a használható veremterület fölé (az utoljára letárolt érték alsó bájtjára) mutat. Értékét a processzor automatikusan beállítja a veremműveleteknél.

A további regiszterekkel nekünk közvetlenül nemigen lesz dolgunk. Csak felsorolásszerűen:

— a programszámláló (PC = Program Counter) 16-bites, mindig a végrehajtandó utasításra mutat. Ennek értékét módosítják az ugróutasítások (CALL, JP, JR, RST, DJNZ, RET).

— Interrupt regiszter (I)
8-bites, csak IM2-es módban használatos (l. ott).

— Frissítő regiszter (R = Refresh regiszter).
7 bites; értéke minden utasítás beolvasása után 1-gyel nő.
A hardver rendszer a memóriaterület folyamatos pásztázására (és ezzel szüntelen frissítésére) használja. Mivel egy gyorsan pörgő érték, akár véletlenszám generálásához is felhasználható.

Az utasítások áttekintése előtt (a bemutatandó példák érdekében) nézzük meg, hogy miként lehet egyáltalán az ENTERPRISE-on gépi kódú rutinokat betölteni és futtatni. Itt — éppen az egyszerűség érdekében — a BASIC rendszeren belül maradunk.

A program tárolásának, a kódok memóriába töltésének viszonylag kényelmes eszköze a CODE utasítás. Ez egy füzér (string) karaktereit helyezi el a memóriában. A tárolás kezdőcímét a 021C/DH BASIC rendszerváltozó tartalmazza. Ez alaphelyzetben a BASIC munkaterület kezdete. Óvatosan kell tehát bánni az utasítással, hiszen a BASIC program elejét felülírhatjuk vele. Ezt akadályozza meg egy a program elején kiadott ALLOCATE n utasítás, ami *n* bájttal hátracsúsztatja a programot, helyet csinálva a gépi kódú rutinunknak. Az utasítás törli a változókat, tehát valóban a program elején praktikus kiadni. A lefoglalt hellyel nem érdemes különösebben takarékoskodni. Az összecsúszás veszélye miatt, a bővítések, alakítások érdekében mindig lefoglalunk legalább 100 bájtot (ezért ne is csodálkozzon az Olvasó, ha egy néhány bájtos rutin előtt ALLOCATE 100 utasítást lát).

A 021C/DH rendszerváltozó (decimálisan 540/1) közvetlen átírásával a tárolási cím máshová is állítható.

Például:

```
POKE 540, 0
POKE 541, 48
```

utasítások után a következő CODE már a 3000H (12288Dec) címtől tárolja a bájtokat. Ezt is használjuk, ha pl. rögzített címen lévő adatmezőre vagy rutinra van szükség. Ez azért is fontos lehet, mert a rendeltetésszerűen elhelyezett gépi kódú rutin nincs teljes biztonságban a BASIC munkaterület kezdetén: a program szerkesztéskor visszacsúszhat erre a területre, felülírva a rutint. Ez nyilván megengedhetelen egy olyan gépi kódú program esetén, amit az operációs rendszer hív a BASIC-től függetlenül (pl. megszakításkezelő).

A példaként írt 3000H kezdőcím praktikus, mert külön védelem nélkül is viszonylag biztonságban van a BASIC terület közepén (de csak ha nem használunk nagy memóriaigényű 0-s BASIC programot).

A 0—16 383 címtartományban (nulláslap) kisebb szabad területet találhatunk még a 0180H címtől (ha BASIC-ben maradunk, 128 bájt) vagy néhány bájtot a memória legelején is (0000—0004). A programok elhelyezésére — legalábbis az egyszerűbb esetekben — használjuk ezt a nulláslapot, hiszen a memóriakonfiguráció további tartományairól nincs mindig biztos információnk (l. pl. 2. fejezet).

A CODE utasítás — mint írtuk — egy füzért vár operandusként

CODE = karakterkifejezés

alakban.

Például a

CODE = CHR$(6) & "VIDEO:"

utasítás a szöveg hosszát, majd karaktereit helyezi el az aktuális tárolási címtől a memóriában (így várja ezt pl. a csatorna megnyitáskor a rendszer).
Sokat egyszerűsít a beírás munkáján a HEX$ függvény, amely az argumentumában megadott (vesszővel elválasztott) hexadecimális értékeket alakítja bináris számokká és fűzi karaktersorozatba. A

CODE = HEX$ ("21,00,00,C9")

utasítás végrehajtása után pl. a 33,0,0,201 (decimális) értékeket találjuk a memóriában. Ha egy változónevet is megadunk a CODE után, az értelmező a változóba tölti a füzér aktuális tárolási címét. Esetünkben pl.:

```
ALLOCATE 3
ok
CODE M1=HEX$("01,02")
ok
CODE M2=HEX$("03")
ok
PRINT M1,M2
 4827     4829
ok
```

Praktikus ezt a lehetőséget kihasználni, hiszen gyakran szükség lehet a rutinok, adatmezők kezdőcímére (pl. szubrutinhívásnál). A kezdőcímek felhasználását is támogatja az értelmező a WORD$ függvénnyel: ez az argumentumot (0—65 535 közötti érték) alakítja kétbájtos egésszé

```
A$=WORD$(12288)
ok
PRINT LEN(A$)
 2
ok
PRINT ORD(A$(1:1)),ORD(A$(2:2))
 0         48
ok
```

Mint a megadott esetben (3000H) is látható, a füzérben elöl az alsó bájt áll (0) azt követi a felső bájt (48Dec = 30H). Ez a forma — mint látni fogjuk — általános a gépi kódú programokban a címek megadására. A memóriában elhelyezett gépi kódú program futtatására a USR függvény alkalmas. Hívható pl.:

CALL USR(cím,hl)

alakban, de alkalmazható akár

PRINT 12 · USR(cím,hl) + 5

formában is. Az argumentumba természetesen a rutin kezdőcímét kell írni, valamint egy 0—65 535 közötti paramétert. Az értelmező a cím meghívása előtt a paraméter értékét (hl) a processzor HL regiszterébe tölti. Ezt felhasználhatjuk bemenő adatként rutinunkban. Ugyanígy vissza is adhatunk egy értéket (HL-ben) az értelmezőnek, amit az majd függvényértéknek tekint. A meghívott rutint "RET" utasítással kell zárnunk (C9H kód). Ennek hatására tér vissza a vezérlés a BASIC értelmezőhöz. A BASIC a visszaadott értéket előjeles kettes komplemens számnak tekinti. Jó tudni még azt is, hogy a felhasználói rutin meghívásakor a 3. lapra (a C000H—FFFH tartományba) az 1. ROM-szegmens van lapozva (erről bővebben írunk a 2. fejezetben), tehát a megadott tartományban ennek a szegmensnek a rutinjait, adatait találjuk.

## 1.2 Az utasításkészlet

A Z80-as mikroprocesszor többszáz utasítást különböztet meg és hajt végre. Ez csak úgy lehetséges, hogy a gyakori, egyszerű utasítások egybájtos kódjai mellett több-bájtos utasításkódok is vannak. Hogy ne vesszünk el ebben a kódtengerben, most általában nem is foglalkozunk az utasítások kódjaival (ezt az 1. és 2. Függelék táblázataiban megtalálja az Olvasó). Bár a gépi kódú program valójában kódok egymásutánját jelenti, mi az áttekinthetőség érdekében egyszerű, a funkcióra utaló néhány betűs neveket adunk meg: a gépi kódú programozás assembly nyelvének ún. mnemonikjait. A kódokkal való megadást mindenhol példaszerűen tekintjük át. Ebben a fejezetben minden kódot és operandust hexadecimális alakban adunk meg, ezért ezt most külön nem jelöljük.

### 1.2.1 Adatmozgató és -cserélő utasítások

A processzor talán legalapvetőbb utasítástípusa nagy területet ölel fel. A regisztereket konstanssal (vagy egy másik regiszter tartalmával) feltöltő utasítások a BASIC "LET" parancsával analógak (LET A = n, LET A = B stb.). Ugyanakkor a külső memóriát célzó adatmozgató utasításokban találhatjuk a "POKE" és "PEEK" analógiáit.

*8 bites adatmozgatás — (LOAD)*

Egy regisztert vagy memóriacímet tölt fel egy bájttal. Az érték forrása változatlan marad. A forrás és a cél igen sokféle lehet. Tekintsük át az első utasításnál részletesen a megadható módokat, hiszen ezzel ismerhetjük meg a processzor címzésmódjait.

Először azokat az utasítástípusokat vegyük sorra, amelyek a A regiszterbe töltenek egy bájtot:

LD A,B: A B regiszter tartalmát tölti A-ba. Ugyanígy a többi általános célú regiszter is megadható forrásként.

LD A,n: A-ba tölti n értékét (00 ≤ n ≤ FF). n értékét az utasítás kódja után kell megadni. Jelen esetben

3E,n

LD A,(nn): A-ba tölti az nn memóriacím (0000 ≤ nn ≤ FFFF) tartalmát. Az A = PEEK (nn) utasítással analóg. A zárójel mindig azt jelenti, hogy a benne levő érték memóriacímként szolgál, az operandus az adott memóriacím tartalma. Az utasítás kódja után először a cím alsó bájtját kell írni, majd a felső bájtját. Például LD A,(0180) esetén:

3A,80,01

LD A,(HL): ugyancsak a külső memória tartalmával tölti fel az A-t, de itt a memória címét a HL regiszterpár tartalmazza. Például, ha HL = 020AH (az aktuális BASIC programszámot tároló rendszerváltozó címe) az utasítás végrehajtása után A-ban az aktuális program száma lesz.

LD A,(IX+d): A-t az IX+d memóriacímről tölti fel. Itt d egy előjeles egész (–128Dec ≤ d ≤ +127Dec). Ezzel az utasítással olvashatjuk be egy memóriatömb d. elemét (ahol az IX indexregiszter a tömb címét tartalmazza). Megadásakor az utasítás kétbájtos kódja után kell megadni d értékét:

DD,7E,d

Az előbbiekkel analóg módon fordított irányú adatmozgatást is kérhetünk:

LD B,A
LD (nn),A
LD (HL),A
LD (IX+d),A

Ezekkel az utasításokkal természetesen az A regiszter tartalmát tölthetjük a célregiszterbe, ill. a memóriacímre.

A következő táblázat összefoglalja a 8-bites LD utasítás lehetséges (x) címzésmódjait:

| | A | B | C | D | E | H | L | (HL) | (BC) | (DE) | (IX+d) | (IY+d) | (nn) | n |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LD A, | x | x | x | x | x | x | x | x | x | x | x | x | x | x |
| LD B, | x | x | x | x | x | x | x | x | | | x | x | | x |
| LD C, | x | x | x | x | x | x | x | x | | | x | x | | x |
| LD D, | x | x | x | x | x | x | x | x | | | x | x | | x |
| LD H, | x | x | x | x | x | x | x | x | | | x | x | | x |
| LD L, | x | x | x | x | x | x | x | x | | | x | x | | x |
| LD (HL), | x | x | x | x | x | x | x | | | | | | | x |
| LD (BC), | x | | | | | | | | | | | | | |
| LD (DE), | x | | | | | | | | | | | | | |
| LD (IX+d), | x | x | x | x | x | x | x | | | | | | | x |
| LD (IY+d), | x | x | x | x | x | x | x | | | | | | | x |
| LD (nn), | x | | | | | | | | | | | | | |

Valamint

```
LD A,I
LD A,R
LD I,A
LD I,R
```

Fontos hangsúlyozni, hogy a LD utasítások nem változtatják meg az F regiszter bitjeit, kivéve a LD A,I és LD A,R utasításokat (l. az 1.2.9 pontot). Attól tehát, hogy kiadunk egy

```
LD A,00
```

utasítást, még nem lesz Z a státusz.

Nézzünk egy-két elemi példát az utasítások alkalmazására. Egyelőre írjuk a mnemonikok mellé az utasítások kódjait is:

```
M1      LD H,L          65
        LD L,00         2E 00
        RET             C9
```

Ez a rutin áttölti a HL regiszterpár alsó bájtját a felsőbe, tehát tulajdonképpen 256-tal szorozza azt. BASIC programban betöltve és futtatva:

```
100 ALLOCATE 100
110 CODE M1=HEX$("65,2E,00,C9")
120 INPUT PROMPT "L=? ":L
130 PRINT USR(M1,L)
140 GOTO 120
```

```
START
L=? 1
 256
L=? 2
 512
L=? 100
 25600
L=? 12E
-32768
```

A 2. rutin a HL-ben megadott címről olvas be egy bájtot és ezt adja vissza a BASIC-nek (tehát tulajdonképpen a BASIC "PEEK" utasítását hajtja végre):

| | | |
|---|---|---|
| M2 | LD A,(HL) | 7E |
| | LD L,A | 6F |
| | LD H,00 | 26 00 |
| | RET | C9 |

```
100 ALLOCATE 100
110 CODE M2=HEX$("7E,6F,26,00,C9")
120 INPUT PROMPT "CIM=? ":CIM
130 LET A=USR(M2,CIM)
140 PRINT "GEPI:";A,"PEEK:":PEEK(CIM)
150 GOTO 120
```

```
START
CIM=? 512
GEPI: 163       PEEK: 163
CIM=? 0
GEPI: 0         PEEK: 0
CIM=? 49152
GEPI: 178       PEEK: 178
```

*16-bites adatmozgató utasítások*

A kétbájtos adatátvitel a regiszterpárok tartalmát együtt mozgatja. Mnemonikja ugyancsak LD.
Típusai:

LD HL,nn: a HL regiszterpárba tölti az nn kétbájtos értéket: L-be az alsó bájt, H-ba a felső bájt kerül.
Például a

LD HL,021C

kódjai: 21, 1C, 02

és végrehajtása után

L = 1C
H = 02

LD HL,(nn): a zárójel itt is azt jelenti, hogy a regiszterpárt memória tartalmával tölti fel a processzor:
L-be az nn memóriacím tartalmát
H-ba az nn+1 cím tartalmát

LD SP,HL: a regiszterpárból regiszterpárba töltés csak néhány esetben működő címzési mód

LD (nn),HL: a regiszterpár memóriába töltése a
POKE nn,L
POKE nn+1,H
utasításokkal analóg módon történik

Itt is foglaljuk össze táblázatosan a megengedett címzésmódokat:

*2. táblázat*

| | BC | DE | HL | SP | IX | IY | nn | (nn) |
|---|---|---|---|---|---|---|---|---|
| LD BC, | | | | | | | x | x |
| LD DE, | | | | | | | x | x |
| LD HL, | | | | | | | x | x |
| LD SP, | | | x | | x | x | x | x |
| LD IX, | | | | | | | x | x |
| LD IY, | | | | | | | x | x |
| LD (nn), | x | x | x | x | x | x | | |

A 16-bites adatmozgató utasítások közé tartozik két különleges csoport: a PUSH és a POP utasítás. Az utasítások célja egy regiszterpár taralmának külső memóriába (a verembe) töltése, ill. visszaolvasása onnan. Specialitásuk abban áll, hogy a külső memóriára mutató regiszterpár (SP) értéke automatikusan állítódik minden művelet után:

PUSH HL: H értéke az (SP-1) címre
L értéke az (SP-2) címre
SP új értéke SP-2

POP HL: L-be az (SP) cím tartalmát tölti
H-ba az (SP+1) cím tartalmát tölti
SP új értéke SP+2

A verem tehát egy olyan tárterület, amelyet felülről lefelé vesz igénybe a processzor, miközben az SP veremmutató mindig a szabad terület fölé mutat.

Amellett, hogy a processzor számára más műveletnél (szubrutinhívások) nélkülözhetetlen a verem, a most megismert utasításokkal is igen jól használható:
— regisztertartalmak tárolására (pl. egy olyan rutin előtt, amely maga is használja a regisztereket)
— regiszterpárok cseréjére, csak így lehet pl. hozzájutni az F regiszter teljes tartalmához (pl. a
PUSH AF
POP HL
utasítások után F tartalma L-be kerül)
A PUSH és POP utasítások az

AF, BC, DE, HL, IX, IY

regiszterpárokra alkalmazhatók.

A processzor státuszát (F) a 16-bites adatmozgató utasítások egyike sem változtatja meg.

Először itt is nézzünk két igen egyszerű példát az utasítások alkalmazására. Az M3 rutin a képernyő első sorának tárolási címét olvassa be a sorparaméter táblából.

```
M3      LD HL,(B914)    2A 14 B9
        RET             C9
```

A cím felhasználható közvetlen képernyőíráshoz. Ehhez előbb a VIDEO cím tartománya alapján meg kell határoznunk a tárolószegmens számát (erről pl. a 2.1 alfejezetben lesz bővebben szó). Az utolsó sor a bal felső sarokba ír egy A betűt:

```
100 ALLOCATE 100
110 CODE M3=HEX$("2A,14,B9,C9")
120 TEXT 40
130 PRINT
140 LET VID_C=USR(M3,0)
150 IF VID_C<0 THEN LET VID_C=65536+VID_C
160 PRINT "Video-cim:" VID_C
170 LET SGM=252+INT(VID_C/16384)
180 SPOKE SGM,VID_C,65
```

A második rutin felhasználásával lekérdezhetjük az állapotsor tetszőleges karakterét. Ehhez az IX indexregisztert is felhasználjuk, tehát gondoskodnunk kell a regiszter BASIC-beli értékének tárolásáról, majd visszaállításáról:

```
M4      PUSH IX         DD E5
        LD IX,BEBE      DD 21 BE BE
        LD L,(IX+0)     DD 6E 00
        POP IX          DD E1
        RET             C9
```

A BASIC program a gépi rutin a LD L,(IX+0) utasításába, az index helyére írja be a kiolvasandó karakter sorszámát. Így a futtatáskor ezen a részen mindig a konkrét indexérték (az indexregiszterbe töltött báziscímhez képest relatív eltolás) lesz (pl. LD L,(IX+2)). A program a visszakapott kódot — a színinformáció letakarása után — karakterként ki is írja.

```
100 ALLOCATE 100
110 CODE M4=HEX$("DD,E5,DD,21,BE,BE,
    DD,6E,00,DD,E1,C9")
120 INPUT PROMPT "Hanyadik karaktert
    olvassam? ":D
130 POKE M4+8,D
140 LET C$=CHR$(USR(M4,0) BAND 127)
150 PRINT D ". karakter: " C$
160 PRINT
170 GOTO 120
```

```
START
Hanyadik karaktert olvassam? 0
 0 . karakter: I

Hanyadik karaktert olvassam? 1
 1 . karakter: S

Hanyadik karaktert olvassam? 23
 23 . karakter: 0

Hanyadik karaktert olvassam? -6
-6 . karakter: C
```

Az indexregiszter a megadott értékkel az állapotsor "IS-BASIC ..." szövegének első karakterére mutat, így adódnak a kiolvasott karakterek. A (–6) érték a futtatáskor éppen érvényes CAPS üzemmódszöveg C karakterét adta vissza.

Az előző rutinokat tulajdonképpen csak a kódbeírás, paraméterátadás, rutinmeghívás bemutatására szántuk. Ezek még nem végeztek olyan feladatot, amit BASIC-ben meg ne tudtunk volna oldani. Írjunk most egy olyan rutint a megismert utasítások alkalmazásával, amelyet jól tudunk majd használni a gépi kódú utasítások funkcióinak, sajátosságainak bemutatásához, megértéséhez.

A gépi kódú programozás egyik — sokakat távoltartó — problémája a folyamatok követhetetlensége. Egy jó segédprogram (monitor) egyik legfőbb szolgáltatása éppen az, hogy lépésenként tudjuk az egyes utasítások regiszterekre és jelzőbitekre való hatását követni. Lényegében arról van szó, hogy bármely utasítás után ki tudjuk íratni a (számunkra érdekes) regiszterek tartalmát.

Ezt gyári monitorprogram nélkül, saját rutin segítségével is megoldhatjuk. Írjunk egy olyan befejezést későbbi programrészleteinkhez, amely a regiszterek tartalmát a memória rögzített helyére tölti. Innen majd BASIC-ben kiolvashatjuk és kiírhatjuk a minket érdeklő paramétereket, jelzőket.

A feladatot legegyszerűbben a veremműveletek segítségével oldhatjuk meg. Nem szabad megfeledkeznünk arról, hogy a rutin befejezése előtt visszaállítsuk az SP veremmutató eredeti tartalmát:

```
M5      LD (0190),SP    ED 73 90 01
        LD SP,0190      31 90 01
        PUSH AF         F5
        PUSH BC         C5
        PUSH DE         D5
        PUSH HL         E5
        LD SP, (0190)   ED 7B 90 01
        RET             C9
```

Természetesen SP visszaolvasása előtt szükség esetén más regisztereket is kivihetünk (pl. PUSH IX). A rutin futtatása után a memóriaterületről kiolvashatjuk a Z80-as regiszterek visszatérés előtti tartalmát:

```
HEX     DEC
0191    401 : SP felső bájt
0190    400 : SP alsó bájt
018F    399 : A
018E    398 : F
018D    397 : B
018C    396 : C
C18B    395 : D
018A    394 : E
0189    393 : H
0188    392 : L
```

A vizsgálandó rutin előtt praktikus a flagregisztert is törölni (így egyértelmű a vizsgált szakasz hatása). Erre egy lehetséges megoldás:

```
LD BC,0000      01 00 00
PUSH BC         C5
POP AF          F1
.
.
.
```

Itt egy 0-val feltöltött regiszterpár tartalmát írjuk át a verem közvetítésével AF-be.

A betöltő BASIC program tartalmazza az előkészítő (250. sor) és a befejező programszakaszokat (270. sor). Mivel a CODE utasítás folyamatosan tárolja a bájtokat a memóriában, a program így is életképes, de valódi haszna a két szakasz közé, a 260. sorba beírt utasítás- vagy rutinrészlet tesztelésében van.

A program végrehajtó része a státusz könnyebb kiértékelése érdekében bitekre bontja a flagregisztert (szükség esetén ezt a többi regiszterrel is megtehetjük). A már ismert decimális—bináris átalakító rutint kissé módosítottuk, hogy az F regiszter határozatlan bitjei helyett egy-egy pont jelenjen csak meg. A további regisztereket — jelen formában — hexadecimális alakban írja ki a program.

```
100 ALLOCATE 100
110 DEF H$(N)=CHR$(INT(N/16)+48-7*
    (INT(N/16)>9))&CHR$(MOD(N,16)+
    48-7*(MOD(N,16)>9))
120 DEF BIN$(X)
130   LET B$=""
140   LET B=128
150   FOR N=0 TO 7
160     LET BN=INT(X/B)
170     IF N<>2 AND N<>4 THEN LET B$
        =B$&STR$(BN)
180     IF N=2 OR N=4 THEN LET B$=B$
        &"."
190     LET X=X+B*(BN=1)
200     LET B=B/2
210   NEXT
220   LET BIN$=B$
230 END DEF
240 !
250 CODE M=HEX$("01,00,00,C5,F1")
260 !
270 CODE VEGE=HEX$("ED,73,90,01,31,
    90,01,F5,C5,D5,E5,ED,7B,90,01,
    C9")
280 !
290 CALL USR(M,0)
300 LET F=PEEK(398)
310 PRINT ,,'SZ.H.PNC"
320 PRINT "   F:";,,BIN$(F)
330 PRINT "   A=";H$(PEEK(399))
340 PRINT "B,C=" H$(PEEK(397)) ","
    H$(PEEK(396))
350 PRINT "D,E=" H$(PEEK(395)) ","
    H$(PEEK(394))
360 PRINT "H,L=" H$(PEEK(393)) ","
    H$(PEEK(392))
```

Például a

LD DE, 3322 (11,22,33)

utasítás hatása:

```
260 CODE =HEX$("11,22,33")
START
                SZ.H.PNC
  F:            00.0.000
  A=00
B,C=00,00
D,E=33,22
H,L=00,00
ok
```

Az A, F, B, és C regiszter az előkészítő szakaszban töltődik fel 0-val, H és L értéke a CALL USR (M,0) hívás paramétere (0), a D = 33H, C = 22H értékek pedig a tesztelt utasítás eredményei.

*Adatcserélő utasítások*

16-bites regiszterpárok tartalmát cserélik meg egyetlen utasítással.

EX DE,HL: a két regiszterpár tartalma felcserélődik, az összes többi regiszter (F is) változatlan marad

EX (SP),HL: HL-be a veremtár aktuális tartalma kerül (mint egy POP HL utasításnál) de ugyanakkor HL régi tartalma a verembe töltődik (mint egy PUSH HL utasítás esetén). SP értéke nem változik
Az utasítás EX (SP),IX
és EX (SP),IY
alakban is használható

EX AF,AF': kicseréli az akkumulátort és a flagregisztert a háttérregiszter-készlet megfelelő rekeszeivel.
Az utasítás kiválóan használható pl. egy sokszor használt bemenő paraméter (vagy ami az operációs rendszerben igen gyakori: egy jellemző státusz) ideiglenes — veremtől független — tárolására és ismételt elővételére a megfelelő pillanatban

EXX: egyetlen utasítással megcseréli a
BC — BC'
DE — DE'
HL — HL'
regiszterpárokat; igen gyors adattárolásra alkalmas.

A regisztercserék után a processzor az eddigi háttérregiszterekkel dolgozik, míg az eredeti regiszterek értéke megőrződik a háttérben.

*Blokkmozgató utasítások*

A Z80-as mikroprocesszor talán leghatékonyabb utasításai a blokkutasítások. A gépi kódú programozásban gyakori feladat egy kisebb-nagyobb memóriaterület feltöltése egy rögzített adatterületről vagy egy adott bájttal. A Z80-as processzornál ezt egyetlen utasítással is elérhetjük.

LDI: a blokkmozgató utasítás egylépéses változata; a HL memóriacím tartalmát a DE memóriacímre tölti, majd 1-gyel növeli HL és DE, csökkenti BC értékét

A blokkmozgató utasításoknál HL mindig a forrás, a DE a cél címe, BC pedig az átviendő bájtok számát adja meg

LDIR: az LDI utasítást automatikusan ismétli mindaddig, amíg BC értéke nullára nem csökken. Ez a parancs tehát a HL címen kezdődő, BC hosszúságú memóriaterület tartalmát másolja át a DE kezdőcímű memóriaterületre

LDD: ez az ugyancsak egylépéses utasítás analóg az LDI-vel, de ez az adatátvitel után csökkenti HL és DE értékét (és persze BC-t is)

LDDR: az LDD utasítást ismétli, amíg BC = 0 nem lesz. Analóg tehát az LDIR utasítással, de ez felülről lefelé haladva viszi át az HL címen végződő, BC hosszúságú memóriaterület tartalmát a DE végcímű memóriaterületre

A mnemonikok betűjelei a funkciókra utalnak:

I: Increment — növelés
D: Decrement — csökkentés
R: Repeat — ismétlés

Az F regiszter jelzőbitjeit a négy utasítás azonos módon állítja:

— az N és a H bit törlődik,
— a P/V bit itt egy speciális jelentést kap: akkor válik 0-vá, ha BC értéke 0 lett a végrehajtáskor (ez az LDIR, LDDR utasításoknál mindig így van)
— a többi bit változatlan.

Alkalmazási példaként vigyük át az 1. szegmens E70FH címén kezdődő üzenet szövegét az állapotsorba (a BEBEH címtől). A szöveg ("1985 Intelligent Software Ltd") hossza 29 (1DH).

```
M6      LD HL,E70F      21 0F E7
        LD DE,BEBE      11 BE BE
        LD BC,001D      01 1D 00
        LDIR            ED B0
        .               .
        .               .
        .               .
```

A rutint illesszük be az előzőekben megismert regiszterkiíratő programba és úgy futtassuk le:

```
  260 CODE M6=HEX$("21,0F,E7,11,BE,BE,
      01,1D,00,ED,B0")
START
                  SZ.H.PNC
  F:              00.0.000
  A=00
B,C=00,00
D,E=BE,DB
H,L=E7,2C
```

(A programot célszerű végtelen ciklusban tartani, pl. egy 370 GOTO 370 sorral, hogy ne íródjon felül az állapotsor.)

A futtatás után természetesen

BC = 0
DE = BEDB = BEBE + 1D
HL = E72C = E70F + 1D

A forrás- és célmutató (HL, ill. DE) az átvitt, ill. feltöltött terület utáni bájtra mutat, tehát szükség esetén folytatható az átvitel, esetleg más területre (vagy területről).

Az LDIR és LDDR utasítások különbsége akkor válik lényegessé, ha a forrásterület és a feltöltendő blokk átlapolódik (van közös területük). Ekkor ui. előfordulhat, hogy az utasítás "beleír" a forrásterületbe. Hogy ezt elkerüljük:

— ha DE < HL, az LDIR utasítást alkalmazzuk,
— ha DE > HL, válasszuk az átvitelre az LDDR-t.

Ezt természetesen nem kötelező alkalmazni. Ha pl. egy terület adott bájttal való feltöltése a célunk, nagyon is jól jön ez a tulajdonság: a karaktert csak az első címre kell betöltenünk, és DE = HL + 1 választással az LDIR ezt a karaktert töltögeti végig a blokkba:

```
M7:     LD HL,BEBE      21 BE BE
        LD (HL),2A      36 2A
        PUSH HL         E5
        POP DE          D1
        INC DE          13
        LD BC,001D      01 1D 00
        LDIR            ED B0
        .               .
        .               .
        .               .
```

Az állapotsor elejére a * karakter ASCII kódját (2AH) töltjük. A még ismeretlen INC DE utasítás a DE = DE + 1 műveletet végzi el.

```
  260 CODE M7=HEX$("21,BE,BE,36,2A,E5,
      D1,13,01,1D,00,ED,B0")
START
                  SZ.H.PNC
  F:              00.0.000
  A=00
B,C=00,00
D,E=BE,DC
H,L=BE,DB
```

## 1.2.2 Logikai utasítások

A logikai alapműveleteket végzik el az akkumulátor és egy másik 8-bites adat (regiszter- vagy memóriatartalom) bitjei között. Az eredmény az A regiszterben képződik.

AND B — logikai és az A és B bitjei között
OR B — logikai VAGY az A és B bitjei között
XOR B — logikai KIZÁRÓ VAGY az A és B bitjei között

A megengedett címzésmódok:

*3. táblázat*

| | A | B | C | D | E | H | L | (HL) | (IX+d) | (IY+d) | n |
|---|---|---|---|---|---|---|---|---|---|---|---|
| AND | x | x | x | x | x | x | x | x | x | x | x |
| OR | x | x | x | x | x | x | x | x | x | x | x |
| XOR | x | x | x | x | x | x | x | x | x | x | x |

Értelmes tehát pl. az OR (HL), de nem létező az OR(nn) utasítás.

A logikai utasítások hatása a jelzőbitekre:

— a C és N bit törlődik (0);
— a H bit
  • AND után 1-be áll,
  • OR és XOR után törlődik;
— az S és a Z bit az A művelet utáni értékének megfelelően áll be:

$$Z = 1, \text{ ha } A = 0,$$
$$S = 1, \text{ ha } A \geq 80H.$$

A logikai utasítások egyik gyakori alkalmazási területe a grafika. Például a HL címen lévő bájt bitjei felelnek meg a képpontoknak. Ekkor

— OR (HL)
  LD (HL),A — kigyújtja azokat a pontokat, ahol A bitjei 1 állapotúak;
— AND (HL)
  LD (HL),A — törli azokat a pontokat, ahol A bitjei 0 állapotúak;
— XOR (HL)
  LD (HL),A — átbillenti azokat a pontokat, ahol A bitjei 1 állapotúak.

Például: (HL) = 00110011
A= 10101010
OR : 10111011
AND : 00100010
XOR : 10011001

Az AND utasítást jól használhatjuk egy adat minket érdeklő bitjeinek kiemelésére, maszkolására, az OR A utasítást igen gyakran alkalmazzák a ROM-rutinok, általában csak az átvitelbit (C) törlését várva tőle, a XOR A utasítás a legegyszerűbb módja az A = 0 értékadásnak stb.

### 1.2.3 Aritmetikai utasítások

*8-bites utasítások*

A Z80-as processzor az összeadást és a kivonást tudja elvégezni. A művelet egyik operandusa mindig az A-regiszter. Az eredmény is az akkumulátorban képződik. Típusai (ugyancsak a B operandus példáján):

ADD A,B: az A = A + B műveletet végzi el. Ha az eredmény meghaladja az ábrázolási tartományt, jelzi az átvitelt (az F regiszter C (carry) bitje 1)

ADC A,B: az összeadásnál figyelembe veszi az átvitelbit (C) előző állapotát is: A = A + B + Carry

SUB B: A = A – B

SBC A,B: A = A – B – Carry

A lehetséges operandusok mind a négy utasításra:

| A | B | C | D | E | H | L | (HL) | (IX+d) | (IY+d) | n |
|---|---|---|---|---|---|---|---|---|---|---|
| x | x | x | x | x | x | x | x | x | x | x |

A műveletek akkumulátorban keletkező egybájtos eredménye mellett fontos a jelzőbitek figyelembevétele is:

C = 1, ha az eredmény túllépett az ábrázolási tartomány határán, azaz
összeadásnál: ha A + x > 255
kivonásnál: ha A – x < 0

P/V: az aritmetikai műveleteknél túlcsordulásjelzőként viselkedik (V), azaz V = 1, ha a kettes komplemens számábrázolási tartományból (–128 ≤ x ≤ 127) kieső eredmény keletkezett

Z = 1, ha a művelet után A = 0

S = 1, ha a művelet után A ≥ 80H, azaz (kettes komplemens ábrázolásnál) negatív

N = 0, ha összeadás történt (ADD, ADC)

N = 1, ha kivonást hajtottunk végre (SUB, SBC)

H = 1, ha a BCD számábrázolási tartományban történt túlcsordulás (ezt az információt a DAA korrekciós utasítás veszi figyelembe)

Érdemes egy kicsit gyakorolni a regiszterkiíratót programunkkal az aritmetikai műveleteket, hatásukat a jelzőbitekre. Például:

```
M8:     LD A,40         3E 40
        ADD A,50        C6 50
        .
        .
        .
```

Csak az érintett regisztereket íratva ki:

```
 260 CODE M8=HEX$("3E,40,C6,50")
START
                SZ.H.PNC
F:              10.0.100
A=90
```

Az eredmény 40H + 50H = 90H helyes, ha pozitív egészeknek tekintjük az értékeket (C = 0), de hibás (P = 1, itt helyesebb lenne V = 1-et írni), ha előjeles számokként értelmezzük az operandusokat és az eredményt, hiszen

$$40H + 50H \neq -70H$$

Ugyanakkor A = 40H, n = E0H esetén

```
 260 CODE M8=HEX$("3E,40,C6,E0")
START
                SZ.H.PNC
F:              00.0.001
A=20
```

Az eredmény tehát hibás (C = 1) pozitív számábrázolásnál, hiszen

$$40H + E0H = 120H \neq 20H$$

de helyes (P/V = 0) előjeles számábrázolásnál, hiszen

$$40H + (-20H) = 20H$$

Speciális aritmetikai utasításnak tekinthető az összehasonlító utasítás (CP), amely a SUB utasítással azonos módon állítja be a jelzőbiteket, de az eredményt nem tárolja (az akkumulátor változatlan marad). Például:

CP B : F regiszter bitjeinek beállítása az (A–B) eredménynek megfelelően. A használható operandusok is megegyeznek a fenti aritmetikai műveletek címzésmódjaival. Például a CP A utasítás Z státuszt állít be.

Az utasítás elsősorban karakter keresésére, azonosítására használható. Léteznek tömbkereső változatai is (amelyek analógak a tömbátviteli utasításokkal):

```
CPI: egy lépés:  CP (HL)
                 HL = HL + 1
                 BC = BC - 1
```

CPIR: automatikusan ismétli az összehasonlítást, végigpásztázva a HL-lel megcímzett, BC hosszúságú memóriaterületet. Lényeges különbség, hogy a keresés leáll a blokk vége előtt, ha közben megtalálta a keresett karaktert (A = (HL) volt). Ezt a Z = 1 jelzőbit mutatja

CPD: egy lépés: CP (HL)
HL = HL – 1
BC = BC – 1

CPDR: automatikus ismétlés felülről lefelé, míg meg nem találja a keresett karaktert (vagy a blokk végére nem ér)

A jelzőbitek:
P/V = 0, ha BC = 0 kilépéskor (blokk végéig ért)
Z = 1, ha A = (HL) volt
S és H a CP művelet eredményének megfelelő
N = 1
C változatlan
A CP utasítások leggyakrabban a feltételes elágazások (l. az 1.2.6 pontot) kérdései (IF A = N...).

Igen praktikus, hogy a processzor képes az operandusok közvetlen (akkumulátortól független) növelésére és csökkentésére. Például:

INC B: 1-gyel növeli a B regiszter tartalmát,
DEC B: 1-gyel csökkenti a B regiszter tartalmát.

Az operandusok köre megegyezik az aritmetikai utasítások címzésmódjaival, kivéve a nyilván értelmetlen INC n típust:

| A | B | C | D | E | H | L | (HL) | (IX+d) | (IY+d) |
|---|---|---|---|---|---|---|---|---|---|
| x | x | x | x | x | x | x | x | x | x |

A jelzőbitek:

— fontos (és talán kissé ellentmondásos is), hogy az INC, DEC utasítások *nem változtatják* az átvitelbitet (C-t),
— az S, Z, H, P/V bitek értéke a művelet eredményének megfelelően áll be,
— N = 0 INC esetén
N = 1 DEC esetén,
A két utasítás elsősorban számlálásra, ciklusszervezésre használható.

*16-bites aritmetikai utasítások*

A Z80-as processzor kétbájtos operandusokon is képes az elemi aritmetikai műveletek elvégzésére. Ilyenkor az alsó bájtról keletkező átvitelt automatikusan figyelembe veszi a felső bájtok összeadásánál, kivonásánál. A művelet egyik operandusa mindig a HL regiszterpár (ill. ennek kiterjesztéseként az ADD utasításnál IX vagy IY) és az eredményt is az első operandusban kapjuk.

Típusai:

ADD HL,BC: HL = HL + BC
ADC HL,BC: HL = HL + BC + carry
SBC HL,BC: HL = HL – BC – carry

Címzésmódjai:

| | | BC | DE | HL | SP | IX | IY |
|---|---|---|---|---|---|---|---|
| ADD | HL, | x | x | x | x | | |
| ADD | IX, | x | x | | x | x | |
| ADD | IY, | x | x | | x | | x |
| ADC | HL, | x | x | x | x | | |
| SBC | HL, | x | x | x | x | | |

Az utasítások hatása a jelzőbitekre:

ADD:

—C = 1, ha az eredmény átlépett a kétbájtos tartomány határán (HL + xx > FFFFH);
—N = 0, H határozatlan;
—a többi jelzőbit változatlan.

ADC és SBC:

—C = 1, ha HL + xx > FFFFH
vagy HL - xx < 0000H;
—P/V: túlcsordulásjelző, a kétbájtos előjeles kettes komponens számábrázolási tartomány (–32 768 ≤ x ≤ 32 767) határának átlépését jelzi;
—S: előjelbit
S = 1, ha HL ≥ 8000H (azaz előjeles számnak tekintve negatív);
—Z = 1, ha az összeg 0;
—N = 0 összeadásnál (ADC);
N = 1 kivonásnál (SBC);
—H: határozatlan.

Itt is nézzünk meg egy-két esetet. Például:

```
M9      LD HL,7FE0      21 EO 7F
        LD DE,0040      11 40 00
        ADC HL,DE       ED 5A
        .
        .
        .
```

Általában a C-bit törléséről is gondoskodni kell az ADC utasítás előtt (ha nem éppen az átvitel figyelembevétele a célunk), pl. egy OR A utasítással. Itt ezt elhagyhattuk, a bevezető szakasz törölte az átvitelbitet.
Az adott értékekkel:

```
  260 CODE M9=HEX$("21,E0,7F,11,40,00,
      ED,5A")
START
                  SZ.H.PNC
  F:              10.1.100
  A=00
B,C=00,00
D,E=00,40
H,L=80,20
```

Itt is mi dönthetjük el, hogy milyen számábrázolási tartományban dolgozunk és ennek megfelelően helyes-e az eredmény, történt-e átvitel.

Kétbájtos pozitív egészként az eredmény elfogadható, hiszen C = 0,

$$7FE0H + 0040H = 8020H$$

Előjeles számnak tekintve azonban már hibás az eredmény, hiszen

$$7FE0H + 0040H \neq -7FE0H$$

Ezt a P/V = 1 érték jelzi.

HL = FFE0H és DE = 0040H értékekkel viszont:

```
  260 CODE M9=HEX$("21,E0,FF,11,40,00,
      ED,5A")
START
                  SZ.H.PNC
  F:              00.1.001
  A=00
B,C=00,00
D,E=00,40
H,L=00,20
```

Itt átvitel keletkezett (C = 1), tehát pozitív egészként az eredmény hibás:

$$FFE0H + 0040H = 10020H \neq 0020H$$

A P/V = 0 érték viszont azt jelzi, hogy az előjeles számok tartományában helyes a művelet eredménye:

$$(-0020H) + 0040H = 0020H$$

A természetes aritmetikai alkalmazásokon kívül a fenti utasítások alkalmasak pl. két regiszterpár értékének összehasonlítására is.

Például az

```
OR A
SBC HL,DE
ADD HL,DE
```

utasítások elvégzése után a regisztertartalmak változatlanok, de az átvitelbit:

C = 1, ha DE > HL
C = 0, ha DE ≤ HL

A processzor a 16-bites regiszterpárokon is képes az inkrementálás (növelés) és dekrementálás (csökkentés) elvégzésére.

Például:

INC BC: BC = BC + 1
DEC BC: BC = BC – 1

A lehetséges operandusok a két műveletre:

| | BC | DE | HL | SP | IX | IY |
|---|---|---|---|---|---|---|
| INC | x | x | x | x | x | x |
| DEC | x | x | x | x | x | x |

A jelzőbiteket a 16-bites INC és DEC utasítások *nem változtatják* (nincs tehát átvitel (carry) az FFFFH→0000H átlépésnél sem!).

*Kiegészítő, általános célú utasítások*

DAA: az akkumulátor tartalmát BCD alakúra konvertálja az összeadások (ADD, ADC, INC) vagy kivonások (SUB, SBC, DEC, NEG) után.
Például a
LD A,08 3E 08
ADD A,04 C6 04
utasítások után az akkumulátorban a binárisan elvégzett összeadás eredménye van (0C):

```
 260 CODE M10=HEX$("3E,08,C6,04")
START
                EZ.H.PNC
F:              00.0.000
A=0C
```

Ha kiegészítjük a rutinrészletet egy DAA utasítással:

```
M11     LD A,08         3E 08
        ADD A,04        C6 04
        DAA             27
        .
        .
        .
```

az átalakított eredmény (12H) a tízes számrendszerben elvégzett összeadás jegyeit adja:

```
 260 CODE M11=HEX$("3E,08,C6,04,27")
START
                SZ.H.PNC
 F:             00.1.100
 A=12
```

Jelzőbitek:

— C: a 7. bitről keletkező átvitelt jelzi
— Z, S, H: a művelet utáni A-értéknek megfelelő
— P/V: paritást jelöl.
— N: nem változik.

CPL: az akkumulátor tartalmát komplementálja: $A = \overline{A}$ (másképpen $A = 255 - A$, egyes komplemens)
Jelzőbitek:
— N és H = 1
— a többi bit nem változik

NEG: megfordítja az akkumulátor előjelét
$A = -A$ (másképpen: $A = 256 - A$, kettes komplemens)
Jelzőbitek: a kivonás műveletnek megfelelően állítódnak. Az adott esetben C mindig 1 (kivéve, ha A = 0 volt), és P/V mindig 0, kivéve ha A = 80H volt. Erre az eredmény ugyancsak 80H, ami hibás, hiszen

$$-(-128) \neq -128$$

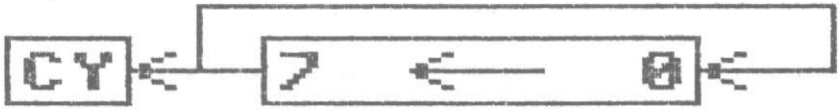
SCF: (Set Carry Flag): 1-be állítja az átvitelbitet (C). Ezenkívül az N és H jelzőbitek törlődnek

CCF: (Complement Carry Flag): átbillenti a C-bitet. Ezenkívül N = 0, H határozatlan a művelet után. Az átvitelbit törlésére nincs közvetlen utasítás, de más egybájtos utasítással elérhető (pl. OR A).

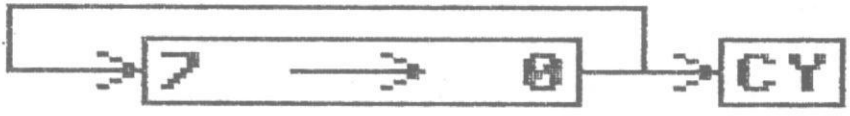
## 1.2.4 Bitléptető és -rotáló utasítások

A bájtot alkotó bitsort léptetik balra vagy jobbra, pontosabban a magasabb vagy alacsonyabb helyi értékek felé. Attól függően, hogy mi történik a szélen kilépő bittel, ill. milyen bit lép be a szélen, több típusa van.

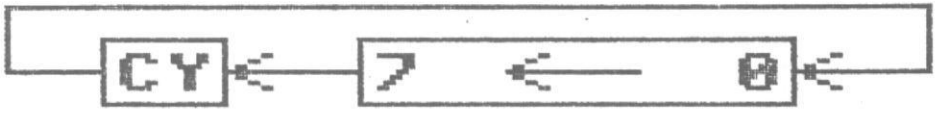
RLC B: a B bitjeit balra forgatja; a 7. bitről kilépő érték visszakerül a 0. bitre és egyúttal az átvitelbitbe is ez íródik:



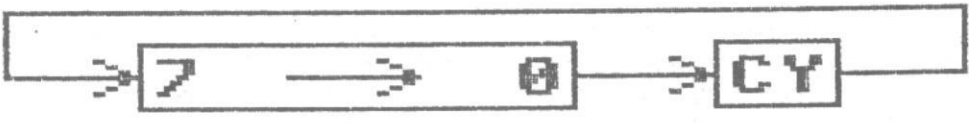

RRC B: jobbra forgatás; 0. bitről kilépő érték visszakerül a 7. bitre és egyúttal az átvitelbitbe is:



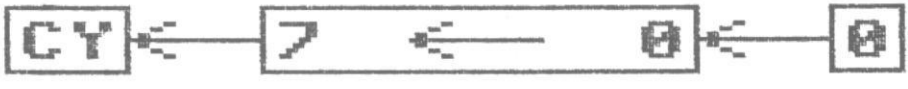

RL B: balra forgatás az átvitelbiten keresztül; a 7. bitről kilépő érték C-be kerül, de az átvitelbit előző értékét a 0. bitre tölti:



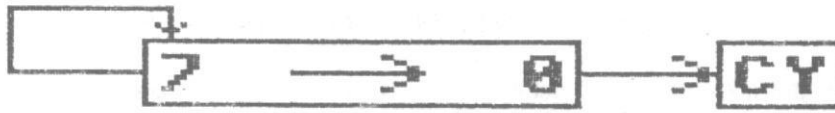

RR B: ugyanez jobbra forgatással; a 0. bitről kilépő érték kerül C-be, de C előző értéke a 7. bitre töltődik:




SLA B: a B regiszter bitjeinek balra léptetése; a kicsorduló 7. bit az átvitelbitbe kerül, a 0. bitbe 0-t tölt az utasítás:




SRA B: jobbra léptetés; a 0. bitről kilépő érték az átvitelbe kerül. A 7. bit értéke változatlan marad:




SRL B: jobbra léptet, mint az előző utasítás, de a 7. bitre itt 0 kerül:


Valamennyi utasítás a következő operandusokkal alkalmazható:

| A | B | C | D | E | H | L | (HL) | (IX+d) | (IY+d) |
|---|---|---|---|---|---|---|---|---|---|
| x | x | x | x | x | x | x | x | x | x |

Jelzőbitek:

— az N és H bitek törlődnek,
— a többi jelzőbit az operandus művelet utáni értékének felel meg, P/V paritást jelez.

Az első 4 utasításnak van egy egyszerűbb (csak az akkumulátorra használható, egybájtos, kevesebb jelzőbitet kezelő)-változata is:

RLCA
RRCA
RLA
RRA

Ezek funkcionálisan az RLC A, RRC A, RL A ill. RR A utasításokkal egyeznek meg, de érdemben csak az átviteljelző bitet állítják be (emellett itt is törlődik N és H).

A biteltoló utasítások elsődleges alkalmazási területe az aritmetika, a kettővel való szorzás, osztás. Egyértelmű az analógia, ha belegondolunk, hogy pl. tízes számrendszerben is úgy szorzunk tízzel, hogy a számjegy karaktereit egy helyi értékkel balra léptetjük, az utolsó helyre pedig 0-t írunk.

Több bájtos számok osztásánál, szorzásánál egyszerűen fel lehet használni az előző bájtról keletkezett átvitelt is. Osszuk el példaként a HL bemenő paramétert 2-vel:

```
M12     SRL H       CB 3C
        RR L        CB 1D
        .
        .
        .
```

A regiszterkiíratót programot az osztórutin beírásán túl egy kissé ki is kell egészíteni az osztandó paraméter beviteléhez:

```
260 CODE M12=HEX$("CB,3C,CB,1D")

285 INPUT PROMPT "HL=? ":HL
290 CALL USR(M,HL)

START
HL=? 256
                SZ.H.PNC
  F:            10.0.000

H,L=00,80

START
HL=? 9
                SZ.H.PNC
  F:            00.0.001

H,L=00,04
```

Az első bevitt érték 256 (100H), fele valóban megfelel a HL = 0080 kiírásnak. A második értékkel a helyes túlcsordulásjelzést szemléltetjük: 9/2 = 4,5, ennek felel meg a

HL = 0004
CY = 1

kijelzés.

Az utasítások másik gyakori alkalmazási területe a grafika, a képpontok jobbra-balra görgetése lehet.

A léptető utasítások között két olyan is van, amely a BCD számok műveleteit segíti. Ezek nem biteket, hanem BCD számjegyeket (félbájtokat) léptetnek. Az operandus itt csak (HL) lehet (tehát a HL értéke által meghatározott memóriarekesz), átvitelként pedig az A regiszter alsó félbájtja (0—3. bit) szolgál.

RLD: balra forgatja HL BCD-jegyeit:

— (HL) alsó félbájtja a felsőbe kerül,
— (HL) felső félbájtja A alsó felébe kerül,
— A alsó félbájtja (HL) alsó 4 bitjére kerül,

RRD: fordított irányú forgatás: a félbájtok "forgalma":

— A alsó → (HL) felső
— (HL) felső → (HL) alsó
— (HL) alsó → A alsó


Jelzőbitek:

C nem változik,
N és H törlődik,
Z és S az akkumulátor tartalmának megfelelően áll be,
P/V az A paritását jelzi.

## 1.2.5 Bitkezelő utasítások

A Z80-as mikroprocesszor képes egybájtos operandusok tetszőleges bitjének lekérdezésére, törlésére vagy beírására. Nézzük meg példaként a B regiszter 3. bitjére vonatkozó utasítások mnemonikjait:
BIT 3,B: az F regiszter Z bitjét a B regiszter 3. bitjének megfelelően állítja be:

Z = 1, ha BIT 3,B = 0 (Z státusz)
Z = 0, ha BIT 3,B = 1 (NZ státusz)

RES 3,B: törli (0-ra állítja) a kijelölt bitet
SET 3,B: beírja (1-re állítja) a kijelölt bitet

A lehetséges operandusok mindhárom utasításra azonosak:

| | A | B | C | D | E | H | L | (HL) | (IX+d) | (IY+d) |
|---|---|---|---|---|---|---|---|---|---|---|
| BIT n, | x | x | x | x | x | x | x | x | x | x |
| RES n, | x | x | x | x | x | x | x | x | x | x |
| SET n, | x | x | x | x | x | x | x | x | x | x |

A kijelölt bit sorszáma (n) 0 és 7 közötti érték lehet.

Jelzőbitek:

a BIT utasítás:
a Z bitet értelemszerűen állítja
N = 0, H = 1
P/V és S határozatlan
C változatlan
a RES és SET utasítások nem változtatják a jelzőbiteket

Az utasítások a különböző jelzőfunkciójú bitek tesztelésén, beállításán túl jól használhatók pl. egy memóriacím tartományának megállapítására. A 2. fejezetben foglalkozunk részletesebben a memóriakiosztás, a lapozás kérdéseivel. Mint látni fogjuk, a 64 kbájtnyi címzési tartományt négy 16 kbájtos lapra osztjuk fel:

| BIT 15,CÍM | BIT 14,CÍM | Címtartomány DEC | HEX |
|---|---|---|---|
| 0 | 0 | 0—16 383 | 0000—3FFF |
| 0 | 1 | 16 384—32 767 | 4000—7FFF |
| 1 | 0 | 32 768—49 151 | 8000—BFFF |
| 1 | 0 | 49 152—65 535 | C000—FFFF |

Természetesen a tényleges lekérdezés (vagy beállítás) a kétbájtos cím felső bájtjainak 7. és 6. bitjén történik.

## 1.2.6 Ugróutasítások

A BASIC GOTO analógiája megtalálható a gépi kódú utasítások között is. Létezik feltétel nélküli ugróutasítás, amely egyszerűen a processzor programszámlálóját (PC) írja át, így a következő utasítást már az új memóriacímről olvassa a Z80-as, de a legnagyobb jelentősége a feltételes ugróutasításoknak van. Eddig számtalan utasítást láttunk, amely megfelelő módon beállította a státuszregiszter jelzőbitjeit, de most találkozunk az első olyan utasításcsoporttal, amely lekérdezni is képes ezeket a biteket (legalábbis egy részüket).

JP nn: feltétel nélküli ugrás az nn címre.
A következő végrehajtandó utasítást az nn címről fogja beolvasni a processzor

JP cc,nn: feltételes ugrás az nn címre.
Itt cc a processzor státuszára vonatkozó rövidítés lehet:
az elugrás feltétele.
A megadható feltételek és az érintett jelzőbitek

NZ (Non Zero): ugrás, ha Z = 0
Z (Zero): ugrás, ha Z = 1

| | | |
|---|---|---|
| NC | (Non Carry): | ugrás, ha C = 0 (átvitel nincs) |
| C | (Carry): | ugrás, ha C = 1 (átvitel van) |
| PO | (Parity Odd): | ugrás, ha P/V = 0 (páratlan paritás) |
| PE | (Parity Even): | ugrás, ha P/V = 1 (páros paritás) |
| P | (Positive): | ugrás, ha S = 0 (pozitív előjel) |
| M | (Negative): | ugrás, ha S = 1 (negatív előjel) |

Például a JP Z,C000 utasítás hatására a processzor — bárhol tartott is a programban — a C000H címen folytatja a végrehajtást, de csak akkor, ha a státusz az utasítás beolvasásakor Z volt (zero, pl. egy kivonás eredményeként). Ez az utasítás tehát a BASIC

IF A = 0 THEN GOTO cím

analógiája. Ezzel az utasításcsoporttal lehet tehát feltételes elágazásokat létrehozni a gépi kódú programban, ami gyakorlatilag minden programhoz nélkülözhetetlen.

Az ugróutasítások másik csoportja nem a folytatás címét adja meg (két bájton), hanem az előre vagy visszafelé átlépendő utasítások számát. Ezek a *relatív* ugróutasítások. Mivel ezek argumentuma egyetlen bájt (egy előjeles kettes komplemens számnak tekintett érték), ezért így az utasítást követő CIM körüli (CIM – 128, CIM + 127) intervallumba lehet csak ugrani. Ez a kis hatósugár egy erős korlátozó tényező, de mégis gyakran alkalmazzuk ezt az ugrástípust a relatív címzésmód előnyei miatt:

— az operandus csak 1 bájt, így helyet takaríthatunk meg
— még fontosabb, hogy ennél a címzésmódnál nem kell tudnunk a cél abszolút címét a memóriában, az ilyen utasításokkal megírt programok tehát a memória tetszőleges címén elhelyezhetők, futtathatók (relokálhatók).

A relatív ugrások is lehetnek feltétel nélküliek vagy feltételhez kötöttek, de itt kevesebb feltételt adhatunk meg:

JR n: (Jump Relative): relatív ugrás
JR NZ,n: relatív ugrás, ha Z = 0
JR Z,n: relatív ugrás, ha Z = 1
JR NC,n: relatív ugrás, ha C = 0
JR C,n: relatív ugrás, ha C = 1

Az ugrás relatív címét elemi módon az átlépendő bájtok leszámolásával határozhatjuk meg (visszafelé ugrásnál a JR utasítás, az operandus és a következő utasítás 3 bájtját is figyelembe kell venni).

Például:

```
            JR CIM          18 03
            LD HL,0000      21 00 00
CIM         LD A,(HL)       7E
```

illetve

```
CIM     INC HL          23
        LD A,(HL)       7E
        CP 40           FE 41
        JR NZ,CIM       20 FA
        RET             C9
```

Az utóbbi rutinban (amely egyébként a memóriát pásztázza végig a belépési HL címtől egy A karaktert keresve) az átlépendő bájtok száma 6 (visszafelé: C9, FA, 20, 41, FE, 7E), tehát a relatív ugráscím:

$$n = -6$$

Kettes komplemens ábrázolásban:

$$n = 256 - 6 = 250 = FAH$$

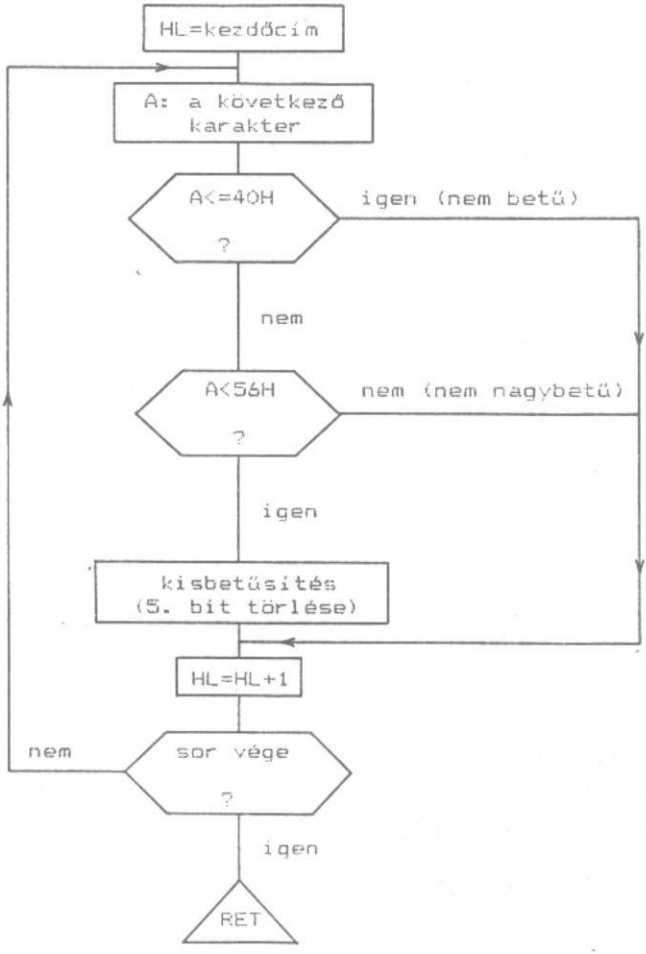
Egy speciális — és igen hasznos — relatív ugróutasítás a DJNZ (Decrement andJump if Non Zero), amely igen egyszerű ciklusszervezést tesz lehetővé:

DJNZ n : B = B – 1, majd relatív ugrás, ha B nem lett 0.

Mindezeken kívül a Z80-as processzor ismer egy igen hatékony ugrástípust is, amikor az ugrás abszolút címét egy regiszterpár tartalmazza. Így lehet kiszámított ugrásokat végezni, vagy egy ugrástáblázatból kiolvasott rutincímre ugrani:

JP (HL): ugrás a HL címre (PC = HL)
JP (IX): PC = IX
JP (IY): PC = IY

Alkalmazási példaként írjuk át az állapotsor üzenetszövegének nagybetűs karaktereit (pl. 32 bájt hosszon) kisbetűkre. Egy blokkdiagram jól foglalja össze a megoldandó feladatokat:




Ennek megfelelően a gépi rutin:

| | | |
|---|---|---|
| M11 | LD HL,BEBE | 21 BE BE |
| | LD B,20 | 06 20 |
| CIM | LD A,(HL) | 7E |
| | AND 7F | E6 7F |
| | CP 40 | FE 40 |
| | JR C, NEXT | 38 06 |
| | CP 5B | FE 5B |
| | JR NC, NEXT | 30 02 |
| | SET 5,(HL) | CB EE |
| NEXT | INC HL | 23 |

```
        DJNZ CIM        10 F0
        RET             C9
```

BASIC betöltővel:

```
100 ALLOCATE 100
110 CODE M=HEX$("21,BE,BE,06,32,7E,
    E6,7F,FE,40,38,06.FE,5B,30,02,
    CB,EE,23,10,F0,C9")
120 CALL USR(M.0)
130 GOTO 130
```

## 1.2.7 Szubrutin utasítások

Egy összetettebb feladat áttekinthető, strukturált megoldásának feltétele a szubrutinok (a program más részeiből tetszés szerint hívható alprogramok) rendszerbe illeszthetősége. A Z80-as processzornál a szubrutin hívását a CALL, a szubrutinból való visszatérést a RET utasítások szolgálják. Mindkét utasítás alkalmazható feltétel nélkül vagy státuszfeltételhez kötötten.

CALL nn: meghívja a memória nn címén kezdődő alprogramot.
Ehhez:

— a veremtárba tölti a CALL-t követő utasítás címét (ide kell majd visszatérnie),
— a programszámlálóba tölti nn értékét, tehát a következő utasítást már az alprogram területéről olvassa be.

RET: visszatér az alprogramból a hívóhoz. Ehhez visszaolvassa a veremtárból a CALL-t követő utasítás címét és ezt tölti a programszámlálóba. Ez lesz tehát a következő végrehajtott utasítás.

A feltételes utasítások:

CALL cc,nn
RET cc

ahol cc a JP-utasításnál megismert 8-féle státuszfeltétel valamelyikének rövidítése:

| | NZ | Z | NC | C | PO | PE | P | M |
|---|---|---|---|---|---|---|---|---|
| cc: | x | x | x | x | x | x | x | x |

Például a RET PO utasítás hatására a végrehajtás visszatér a veremben tárolt címre, ha az F regiszterben P/V = 0 (ami az előző művelet függvényében jelentheti azt, hogy az operandus páratlan paritású, de azt is, hogy érvényes előjeles egész).

A visszatérési cím az előbbiek szerint egy szubrutin futása alatt a veremtár

legutoljára betárolt eleme. Ez a szubrutinkezelési mód a verem igen fegyelmezett használatát követeli meg. Egy pár nélküli PUSH vagy POP utasítás a rendszer összeomlását okozhatja, hiszen a legközelebbi RET hatására a Z80-as mindenképpen visszatérési címnek értelmezi a verem tetején talált kétbájtos értéket.

Másrészt azonban programjainkban fel is használhatjuk ezt az információt: kiolvashatjuk, hogy honnan hívták a rutinunkat, onnan akár paramétereket is olvashatunk be (így működnek pl. a RST 30H (EXOS) és RST 10H (BASIC) funkcióhívások), esetleg tudatosan meg is változtathatjuk a visszatérési címet.

Anélkül, hogy túlzottan előre akarnánk ugrani (a BASIC rendszerhez), elmondhatjuk, hogy a sokszor használt USR(CIM,HL) gépi kódú programhívó függvény egy olyan alprogram címével (mint visszatérési címmel) lép ki a felhasználói rutinba, amely alaphelyzetbe állítja a BASIC munkaterületét, majd HL értékét (mint előjeles kétbájtos konstanst) tölti oda. Így lehetséges, hogy pl. a PRINT USR ( ) hívás esetén a PRINT utasítás ezt az értéket írja ki. Ha ezt a visszatérési címet egyszerűen töröljük a tárból (egy POP utasítással), a BASIC munkaterületen a mi rutinunkban betöltött érték maradhat mint kiírandó (vagy máshogy felhasználható) érték. Például az 1. szegmens CA0BH rutinja a PI értékét tölti a munkaterületre. Használjuk fel ezt a ROM-programot szubrutinként:

```
M12     CALL CA0B     CD 0B CA
        POP HL        E1
        RET           C9
```

A program futtatása után PI értékét írja ki a rendszer:

```
100 ALLOCATE 100
110 CODE M=HEX$("CD,0B,CA,E1,C9")
120 PRINT USR(M,0)

START
 3.141592654
ok
```

Folytatva a csoport további utasításainak bemutatását, két olyan RET típusú utasítással találkozunk (RETI és RETN), amelyeket az ENTERPRISE nem használ. Ezek a megszakításból, ill. nem maszkolható megszakításból való visszatérést kiváltó utasítások az alapfunkción túl még belső jelzőket is állítanak, ami az ENTERPRISE megszakítási rendszerében érdektelen.

A processzor rendelkezik néhány egybájtos szubrutinhívó utasítással is. Ezek a memóriaterület legelején, rögzített címen kezdődő rutinokat hívnak meg, hatásuk egyébként azonos a CALL utasításéval.
Például:

—RST 30H = CALL 0030H
A megengedett RST-utasítások (hexadecimális címekkel):

| | 00 | 08 | 10 | 18 | 20 | 28 | 30 | 38 |
|---|---|---|---|---|---|---|---|---|
| RST | x | x | x | x | x | x | x | x |

Ezeket a hívásokat igen hatékonyan használja az operációs rendszer: az RST 30H utasítást az EXOS sajátította ki, az RST 08H, 10H, 18H, 20H utasításokat pedig a BASIC. Segítségükkel igen egyszerűen hívhatók a rendszerek nagybonyolultságú funkciói.

### 1.2.8 Input/Output (I/O) utasítások

A Z80-as processzor a 64 kbájtnyi memóriaterület elérése mellett 65 536 különböző Input/Output cím — ún. port — olvasására, írására is képes az IN és OUT utasításokkal. Az ENTERPRISE azonban fizikailag lényegesen kevesebb I/O portot használ. Ilyenek pl. a nagy bonyolultságú kiegészítő áramkörök (a NICK és a DAVE chip) különböző regiszterei. Ilyen regisztereket használ a gép az elérhető memóriaterület kiterjesztéséhez (a memória 16 kbájtos szegmenseinek a Z80-as címtartományába való lapozásához), a nyomtató kezeléshez, a videokijelzés bizonyos funkcióihoz, billentyűzet letapogatáshoz stb. Ezek címzéséhez bőségesen elegendő 1 bájt (256 I/O cím). A továbbiakban így adjuk meg az utasításokat (a Z80-as teljes I/O címzési lehetőségeit bármelyik Z80-as szakirodalom tartalmazza).

IN A,(n): az A regiszterbe olvassa az n című port adatát.
IN r,(C): bármelyik regiszterbe (r lehet: A, B, C, D, E, H, L) olvashatjuk az adatot. A port címét a C regiszter tartalma adja.
OUT (n),A: az n című portra adja az akkumulátor tartalmát.
OUT (C),r: a C tartalmának megfelelő portra adja az r regiszter tartalmát (r lehet A, B, C, D, E, H, L).

Jelzőbitek:
az IN A,(n), OUT (n),A és OUT (C),r utasítások nem változtatják a státuszt, míg az IN r,(C) utasítás végrehajtása után a Z, P/V (paritás), S és H bitek a beolvasott értéknek felelnek meg; N = 0 és C változatlan.

Az I/O utasításoknak van (a tömbmozgató utasításokkal analóg) adatsorozat átvitelét segítő változatuk is. Ezekben a kijelölt port címét mindig a C regiszter tartalmazza, míg az adat a HL címre kerül, ill. a HL cím tartalmát viszi ki a processzor a portra.
Számlálóként a B regiszter szolgál.

INI: C port adatát a HL címre írja
B = B – 1
HL = HL + 1
INIR: az INI utasítást automatikusan ismétli, amíg B = 0 nem lesz
IND: analóg az INI-vel, csak HL = HL – 1

INDR: az IND utasítást automatikusan ismétli, amíg B = 0 nem lesz
Ugyanezek az utasítások adatkivitelre:
OUTI: a HL cím tartalmát a C portra viszi
B = B – 1
HL = HL + 1
OTIR: ismételt OUTI B = 0-ig
OUTD: (HL) tartalma a C portra
B = B – 1
HL = HL – 1
OTDR: ismételt OUTD B = 0-ig
Jelzőbitek:
Z = 1, ha B = 0 a művelet után
P/V, S és H határozatlan
C változatlan
N = 1

Alkalmazási példaként olvassuk be a billentyűzetmátrixot. Ehhez a mátrix tíz során ak számát sorra kiírjuk a B5H portra, mert ezzel jelöljük ki az olvasandó sort, majd az onnan beolvasott adatot (minden 0 bit egy lenyomott billentyűnek felel meg) kiírjuk az állapotsorba (ott természetesen az érték valamilyen karakterként jelenik majd meg):

```
M13     LD HL,BEBE
        LD C,00
        LD B,0A
CIKL    LD A,C
        OUT (B5),A
        IN A,(B5)
        CPL
        LD (HL),A
        INC C
        INC HL
        DJNZ CIKL
        RET
```

BASIC betöltővel:

```
100 ALLOCATE 100
110 CODE M=HEX$("21,BE,BE,0E,00,06,
    0A,79,D3,B5,DB,B5,2F,77,0C,23,
    10,F5,C9")
120 CALL USR(M,0)
130 GOTO 120
```

## 1.2.9 A processzor vezérlőutasításai

NOP: üres utasítás, a processzor nem hajt végre semmilyen műveletet (a gépi kódú programban helykitöltésre, egy ütemnyi késleltetésre használható).

HALT: felfüggeszti a processzor működését egy megszakításkérés érkezéséig. Elsősorban perifériaorientált rendszerben (pl. egy mérésvezérlő központi egységben) használják, ahol a processzor egyetlen feladata a külső eszköz kiszolgálása.

Maga a megszakításkérés a processzor INT vagy NMI csatlakozási pontjára (lábára) érkező impulzus. A kérést a processzor az első esetben csak akkor fogadja el, ha előzőleg a program engedélyezi a megszakítást, a második esetben pedig feltétel nélkül (NMI = Non Mascable Interrupt — nem maszkolható megszakítás).

EI: engedélyezi a megszakításokat.

DI: letiltja a további maszkolható megszakításokat (a következő EI utasításig).

A két utasítás természetesen csak a maszkolható megszakításokra vonatkozik. A pillanatnyi állapotot egy belső flag őrzi, amelynek értéke 1, ha a megszakítás engedélyezett, 0, ha tiltott. Az LD A,I vagy LD A,R utasítással olvashatjuk a státuszregiszterbe (P/V bit).

Hogy mit jelent az, hogy a processzor elfogadja a megszakításkérést, azt a pillanatnyi megszakítás üzemmód határozza meg. Maszkolható megszakításkérésnél először letiltja a további megszakításokat, majd a kérést a legutóbbi üzemmód-beállító utasítás szerint kezeli:

IM 0: (0-s megszakítás mód) a CPU azt az utasítást hajtja végre, amit a kiszolgálást kérő eszköz az adatbuszra ad.

IM 1: a processzor egy RST 38H (azaz CALL 0038H) utasítást hajt végre.

IM 2: a CPU egy olyan szubrutint hív meg, amelynek címét egy memóriabeli táblázat tartalmazza. A táblázat adott elemének címét részben a megszakítást kérő eszköz határozhatja meg:
— cím felső bájt: I regiszter,
— cím alsó bájt: a periféria adhatja a buszra.

Nem maszkolható megszakításkérésnél a processzor egy RST 66H utasítást hajt végre.

Az ENTERPRISE megszakítás rendszere az IM 1-es módot használja. Ez ugyan önmagában nem a leghatékonybb (hiszen minden megszakításkérést a 0038H szubrutin szolgál ki), de a hardver megoldás még az IM 2-es módnál is nagyobb hatásfokúvá teszi. A megszakításkérések ui. nem közvetlenül, hanem egy kiegészítő áramkör (a DAVE chip) közvetítésével jutnak a processzorra. A megszakítás forrását a DAVE regiszterei alapján szoftver úton lehet megállapítani, sőt — ami minőségileg újat jelent — külön-külön is le lehet tiltani az egyes források kéréseit. A megszakítás rendszerről a 2.4 alfejezetben írunk bővebben.

* * *

A Z80-as mikroprocesszor megismert utasításkészletével nekivághatunk a gépi kódú programozásnak. Az eddig közölt példák megmutatták, hogy a közvetlen, kódokkal való programozás meglehetősen körülményes (főleg a relatív ugrások számítása nehézkes). Néhány tucat, esetleg 1—200 utasítás fölött már feltétlenül érdemes a gépi kódú programozást igen megkönnyítő segédprogramot, *assemblert* alkalmazni.

A mintapéldákban eddig is megadtuk a kódsorozat mellett az utasítások mnemonikjait. Például egy az akkumulátorba az A kódját töltő, majd vissztérő rutin gépi kódban:

```
3E 41 C9
```

Ennek *assembly* nyelvű kifejezése:

```
LD A,41H
RET
```

Az assembly nyelven megírt programot gépi kódok sorozatára lefordító segédprogram az assembler. Valójában egy jó assembler lényegesen többet nyújt a mnemonikok kódokkal való helyettesítésénél. A relatív ugrásokat, szubrutinhívásokat stb. segítő címkéket használhatunk, operandusként paraméterek is megengedettek, sőt az aritmetikai műveletek egy részét is használhatjuk stb. A nyújtott szolgáltatásokat mindig a kiválasztott assembler leírása adja meg. Mi a talán legelterjedtebb SIMON (editor, assembler és monitor) programot használtuk. Ennek szabályait a 9. Függelékben foglaljuk össze. Most nézzünk meg egy-két olyan szintaktikai szabályt, amely a következő fejezetek mintapéldáinak, leírásainak megértéséhez szükséges.

— Az assembler (megfelelő beállítással) a memória kijelölt címére tölti a lefordított kódokat. A kívánt kezdőcím az ORG direktívával adható meg.
Például:

```
ORG 0180H
```

— Egy sorba csak egy utasítást írhatunk. Az első pozíción csak címke kezdődhet. Ha nincs címke, legalább egy szóköznek kell állnia a Z80-as utasítás előtt.
Például:

```
        LD B,255
LOOP    LD (HL),A
        INC HL
        DJNZ LOOP
        .
        .
        .
```

— A numerikus operandusokat a SIMON assemblere decimálisan értelmezi. Ha hexadecimális értéket akarunk megadni, ezt a konstans után írt H betűvel kell

jelezni. Egy megkötés: konstans csak számjeggyel kezdődhet (ez különbözteti meg a címkétől). Innen a néha körülményesnek tűnő alak:

LD HL, 0BEB8H

— A programokba tetszés szerint írhatunk — pontosvessző után — megjegyzéseket.

A működés bemutatására, magyarázatára megadott ROM-részleteket ugyancsak a SIMON programmal disassembláltuk; fordítottuk vissza a kódsorozatot jól követhető assembly listává. Ezekben a listákban viszont a SIMON minden értéket (címet és operandust) hexadecimálisan ad meg, külön H jelzés nélkül.

Ezek után ismerkedjünk meg a gép két fő rendszerének felépítésével, működésével: a beépített operációs rendszer (az EXOS) és a leggyakrabban használt felhasználói program, az IS-BASIC gépi kódú programjainknak környezetet adó munkaterületeivel, változóival, általunk is felhasználható rutinjaival.

# 2. A háttérrendszer

## 2.1 A memóriaszervezés

Kezdjük a gép megismerését a memóriaszervezés tárgyalásával, hiszen ez minden későbbi működési, felhasználási információ alapja.

Az ENTERPRISE számítógép egyik fontos tulajdonsága, hogy igen nagy memóriaterületet kezel: akár 4 Mbájtot (256 · 16 384 bájtot). Ugyanakkor a gép központi egysége (a Z80-as mikroprocesszor) 16 címvezetékével csak $2^{16} = 64$ kbájtot (azaz 4 · 16 384 bájtot) képes megkülönböztetni. Ezt az ellentmondást csak hardver segédlettel lehet feloldani: a CPU címzési tartományát 4 — egyenként 16 kbájtos — intervallumra, ún. lapokra osztjuk fel és ezekre a területekre a gép nagy bonyolultságú perifériaáramköre (a DAVE-chip) lapozza az elérhető 256 db 16 kbájtos memóriaszelet bármelyikét (az ún. szegmenseket).

A különböző memóriaáramkörök áramkörbe csatlakoztatásával egyértelműen rögzítődik azok szegmensszáma. A szegmensszám tehát kötött fizikai paraméter: adott szegmenshez mindig (bármilyen lapozásnál) ugyanazok a memóriarekeszek tartoznak.

Azt, hogy a CPU a 4 memóriatartományban — mint ablakon — melyik szegmenst látja, a DAVE-chip 4 írható, olvasható regisztere határozza meg:

176. regiszter (B0H): 0. LAP
177. regiszter (B1H): 1. LAP
178. regiszter (B2H): 2. LAP
179. regiszter (B3H): 3. LAP

A lapregiszterek mindegyike tetszőleges értékre beállítható a Z80-as CPU OUT utasításával (annak sincs akadálya, hogy akár mind a 4 lapon ugyanazt a szegmenst lássa a mikroprocesszor). Itt jegyezzük meg, hogy ugyanez a lapozás elvégezhető a BASIC OUT utasításával is, de ezt csak az 1. és 2. LAP-on használhatjuk, a 0. és 3. LAP átírása a rendszer összeomlását okozhatja.

Az egyes lapokhoz tartozó címtartományok:

0. LAP: 0—16 383 (0000H—3FFFH)
1. LAP: 16 384—32 767 (4000H—7FFFH)
2. LAP: 32 768—49 151 (8000H—BFFFH)
3. LAP: 49 152—65 535 (C000H—FFFFH)

Az előbbiek szerint a

```
LD A,01H
OUT (B2H),A
```

utasítások végrehajtása után az 1. szegmens lapozódik a 2. LAP-ra: a 32 768 és 49 151 címtartományba eső minden írási vagy olvasási utasítást a DAVE-chip automatikusan az 1. szegmens megfelelő rekeszeihez irányít. Ez a lapozási technika az ENTERPRISE működtető szoftverének, operációs rendszerének egyik legmarkánsabb sajátossága, és egyben — sajnos — működési sebességének korlátozója is. Szerencsére (mint látni fogjuk) a szoftver sok eleme támogatja ezt a tárkezelést.

A következő kérdés, hogy milyen szegmenseken találunk ténylegesen tárakat, és milyen típusúak azok. Általánosan igaz, hogy a legalacsonyabb sorszámú szegmenseken a ROM-ok találhatóak: azok a tárak, amelyek a gépet működtető szoftvert, a gyárilag beégetett operációs rendszert, a BASIC értelmezőt stb. tartalmazzák. A saját fejlesztésű (vagy magnetofonról beolvasott) programokat, változó adatokat, képeket stb. befogadó RAM-memóriák felülről lefelé (255-től) számozódnak.

A továbbiakban példaként választott géptípus — a 128 kbájt RAM-mal felszerelt angol és német nyelvű verzió — esetén konkrétan a következő szegmenseket találjuk a gépben (csatlakoztatott IS-BASIC bővítő modul esetén):

0. szegmens (00H): belső ROM
   (elsősorban EXOS rendszerprogramok; video, hang, szerkesztő és billentyűzetkezelők)
1. szegmens (01H): belső ROM
   (matematikai rutinok; nyomtató, kazettás magnetofon, soros hálózati kezelők; szövegszerkesztő stb.)
4. szegmens (04H): külső ROM BASIC-BRD
   (elsősorban a német nyelvű verzióhoz szükséges kiegészítéseket tartalmazza)
5. szegmens (05H): külső ROM BASIC-UK
   (BASIC értelmező)

A német (BRD) és angol (UK) BASIC szegmensek sorszáma gépenként (szériától függően) eltérő lehet, akár meg is cserélődhet. Ezt az operációs rendszer hidegindításkor figyelembe veszi. Mivel egyes alkalmazásoknál szükséges a konkrét szegmenssszámok ismerete, javasoljuk az Olvasónak, hogy — pl. a következő fejezetek alapján — állapítsa meg saját gépe szegmenskiosztását. Mi a továbbiakban a választott példa szegmenssszámait használjuk.

248. szegmens (F8H)
249. szegmens (F9H)
250. szegmens (FAH)
251. szegmens (FBH)  RAM-szegmensek (8 · 16 k)
252. szegmens (FCH)
253. szegmens (FDH)
254. szegmens (FEH)
255. szegmens (FFH)

A felsorolt RAM-szegmensek közül a felső 4-nek (FCH—FFH) kiemelt jelentőséget ad az a tény, hogy a videoáramkör (a képernyőn való megjelenítést végző NICK-chip) csak ezt a tartományt látja, ezt a 64 kbájtos memóriaterületet képes elérni.

A NICK-chip által használt videocím kötött, lapozástól független. Adott videocímen mindig ugyanaz a memória érhető el (akkor is, ha a Z80-as számára az adott szegmens be sincs lapozva!):

| Videocím | Szegmens |
|---|---|
| 0000—3FFFH | FCH |
| 4000—7FFFH | FDH |
| 8000—BFFFH | FEH |
| C000—FFFFH | FFH |

## 2.2 Az operációs rendszer szerkezete

A továbbiakban ismerkedjünk meg a háttérszoftver szerkezetével, a gép működését szervező, irányító operációs rendszer működési logikájával.

A szoftver lelke az EXOS (EXtendable Operating System — bővíthető operációs rendszer). A 2.1-es EXOS változat funkcionális ismertetését a "Műszaki leírás" adja meg. Mi más szemmel tekintjük át a rendszert. Két legfontosabb — gépi kódú programozásnál alapvető jelentőségű — szempontunk az lesz, hogy

— milyen környezetet, hátteret épít fel számunkra az EXOS;
— hogyan használhatjuk fel a rendszer rutinjait, funkcióit (akár közvetlenül is, a kényelmes, de időigényes főágak megkerülésével, átlépésével).

Mindenekelőtt tekintsük át az EXOS alapfeladtait.

1. Bekapcsolása után inicializálja a rendszert (ezt kissé részletesebben áttekintjük a 2.3.1 pontban):
   — ellenőrzi a különböző memóriaterületeket és típusokat (ROM vagy RAM) az eredménynek megfelelően könyvtárazza azokat;
   — kijelzésre alkalmas állapotot hoz létre a VIDEO áramkör (NICK-chip) számára;
   — alapértéket ad a rendszerváltozóknak, RAM-ba helyez bizonyos nélkülözhetetlen rendszerrutinokat;
   — listát készít a rendszer-ROM-ban és a bővítőkben talált perifériakezelőkről.

2. Lehetőséget ad a bővítőben csatlakoztatott alkalmazói programnak a vezérlés átvételére (ha nincs ilyen program, a szövegszerkesztő kapja meg a vezérlést).

3. A háttérben továbbra is ellát minden olyan feladatot (az alkalmazói programban kiadott funkcióhívások útján), amely a memóriaszervezéssel, a perifériák írásával, olvasásával és a csatornaműveletekkel kapcsolatos.

A két belső ROM-ban tárolt alapszoftver a következőket tartalmazza:
— egy inicializáló szakaszt a rendszer egyértelmű alaphelyzetbe állításához;
— egy sokrétű funkcióhívás kezelőt, ennek főágát és alprogramjait, köztük a megszakításkezelővel;
— a géphez csatlakozó be- és kiviteli alapelemek kezelőit:
a *0. szegmensen:*
- VIDEO (képernyőkezelés)
- SOUND (hangkezelés)
- KEYBOARD (billentyűzetkezelés)
(ezt a feladatot a német BASIC-ROM átveszi a német karakterkezelés érdekében)
- EDITOR (szerkesztés)
(ezt is felülírja a BASIC-BRD szegmens a BASIC-ben)

az *1. szegmensen:*
- PRINTER (nyomtatókezelés)
- TAPE (magnetofonkezelő)
- SERIAL (soros adatátviel)
- NET (hálózatkezelő)

— egy szövegszerkesztőt (Word Processor);
— ezen kívül tartalmaz még adatokat is; változók alapértékeit, 10-es szorzótáblát, üzenetek (és hibaüzenetek) szövegét, a karakterkészletet, a matematikai rutinok egy részét stb.

## 2.3 Az EXOS működése

Az EXOS — operációs rendszer létére — szokatlanul sokoldalú, meglepően sok feladatot lát el. Ezek között vannak természetes alapfeladatok, pl. a különböző memóriaterületek ellenőrzése, alaphelyzetbe állítása stb., de — különösen a képernyőkezelés területén — ellát olyan feladatokat is, amelyek egy alkalmazói programnak is becsületére válnának. Elég azt megemlíteni, hogy ez a rendszer tartalmaz pl. ellipszisrajzoló vagy a zárt grafikus alakzatokat kitöltő alprogramokat is (a VIDEO perifériakezelőben). Tulajdonképpen a gép — igen komfortos — BASIC-jének alig van olyan (perifériákkal kapcsolatos) utasítása, amit ne támogatna az EXOS.

Ez a tény nagyon leegyszerűsíti a gépi kódú programozás bizonyos területeit is, hiszen bonyolult feladatokat oldhatunk meg egyetlen funkcióhívással (pl. képernyőlap kijelzése) anélkül, hogy akár a végrehajtó rutinok, akár az érintett memóriaterületek címeit ismernünk kellene.

Az EXOS bemutatása során — bár nem ROM-fejtés közlése a célunk — néhány belső rutint is elemzünk, részben a profi gépi kódú programozás tanulságai érdekében, részben a felhasználás feltételeinek megismeréséhez. A megértéshez, felhasználáshoz példákat is adunk, általában BASIC-ben felépítve, hogy a gép elé leülő, a rendszerrel most ismerkedő Olvasó azonnal kipróbálhassa, ellenőrizhesse a leírtakat.

Ez annál is fontosabb, mert a forgalmazott gépek között több változat is előfordult. Az operációs rendszer alaptulajdonsága — nevéhez méltóan — a kiterjeszthetőség, bővíthetőség. A személyi számítógépek körében szinte páratlan mértékben képes memóriamérethez, verzióhoz alkalmazkodni. Ez a nagyfokú rugalmasság természetesen azzal jár, hogy a memóriakiosztás is képlékeny. Igazán hordozható, minden verzióban működő programot tehát csak az EXOS rendszer, a funkcióhívások felhasználásával építhetünk fel. Egy adott gépen dolgozva azonban mindenképpen praktikus a gép megismerése, feltérképezése, hiszen a konkrét adatok birtokában gyakran sokkal hatékonyabban (rövidebb programokkal és gyorsabban) oldhatjuk meg a feladatokat. A mi példáink — mint leírtuk — a 128 kbájtos, angol/német változatra vonatkoznak. A saját gépén az Olvasónak feltétlenül érdemes a közös báziscímekből kiindulva megállapítani (és rögzíteni) a konkrét adatokat és címeket.

### 2.3.1 Inicializálás

A bekapcsolás után az EXOS hidegindítási főága hozza létre azt a jól definiált hátteret, amelyben minden felhasználói program dolgozhat. Gépi kódú programot írva számunkra is ez lehet a munkánk bázisa, ez szabja meg a lehetőségeket és a korlátokat. Kövessük tehát végig a rendszer felépítésének folyamatát, természetesen azokat a részeket, elemeket kiemelve, amelyek valamilyen szempontból fontosak lesznek a későbbiekben.

A feszültség rákapcsolásakor a processzor a 0. szegmens legelső utasításait kezdi végrehajtani. Még a 0. LAP-on letiltja a megszakításokat és egyszer s mindenkorra az 1. megszakítási módba kapcsolja a processzort (részletesebben l. a 2.4 alfejezetben). Ezután elugrik a 3. LAP-ra, ahol — bekapcsolás után — szintén a 0. szegmens van belapozva, és most már itt folytatja a futást.

Érdemes megjegyezni, hogy a továbbiakban a különböző rendszerprogramok tartózkodási helye szinte kizárólag a 3. LAP lesz. Ide lapoz be minden programot, kezelőt is az EXOS, mielőtt átadná nekik a vezérlést. Ezért mi is a 3. LAP-on értelmezett címeket használjuk, pl. egy disassemblált ROM-részletben. Így lesz a 0. szegmens legelső bájtja a C000H című utasítás, a szegmens végcíme pedig a FFFFH.

*Memóriaellenőrzés és könyvtárazás*

A 3. LAP-on elvégzett első érdemi funkció azonnal érdekes lehet a felhasználó számára. A 0. LAP-ra lapozott 4. szegmensen (ide, a bővítő cartridge-ban lévő EPROM-ba elvileg saját beégetett programunkat is tehettük) ellenőrzi, hogy nem a TEST-ROM szöveggel (54H, 45H, 53H, 54H, 5FH, 52H, 4FH, 4DH bájt-sorozat) kezdődik-e. Ha igen, azonnal el is ugrik a szöveget követő bájtra. Az EXOS rugalmasságára jellemző, hogy már az inicializálás pillanatában hajlandó átadni a vezérlést a felhasználónak, aki így akár egy teljesen más karakterű gépet is kihozhat az adott hardverből.

Mi azonban az ENTERPRISE alaphelyzetét, memóriakiosztását szeretnénk megismerni. Kövessük tehát tovább a főágat, a hidegindítás folyamatát.

Az EXOS külön ellenőrzi a később fontos célra szánt 255. szegmenst (FFH). Az ellenőrzés módszerét érdemes áttekinteni a később valamennyi RAM-szegmens vizsgálatára felhasznált rövid ROM-részletben. A vizsgált szegmens az 1. LAP-on van (4000H—7FFFH címtartomány), a teszteléshez felhasznált értékek a 0. szegmens (ROM) C000H—FFFFH bájtjai.

```
C20E  21 00 40     LD   HL,4000     ;vizsgált szegmens (RAM)
C211  01 00 C0     LD   BC,C000     ;mintaszegmens (ROM)
C214  0A           LD   A,(BC)      ;minta
C215  77           LD   (HL),A      ;a RAM-ba
C216  BE           CP   (HL)        ;tárolódott?
C217  C0           RET  NZ          ;vissza, ha nem RAM (NZ)
C218  2F           CPL              ;komplementer
C219  77           LD   (HL),A      ;tárolása
C21A  23           INC  HL          ;következő RAM
C21B  03           INC  BC          ;  és minta
C21C  CB 78        BIT  7,B         ;3. LAP-cím felső bit
C21E  20 F4        JR   NZ,C214     ;tovább, ha cím<=FFFFH
```

Szegmens vége: visszafelé újra megvizsgálja a letárolt értékeket:

```
C220  0B           DEC  BC          ;következő minta
C221  2B           DEC  HL          ;   és RAM-cím
C222  CB 74        BIT  6,H         ;RAM-cím>=4000H?
C224  C8           RET  Z           ;kész az ellenőrzés, ha
                                     vége a szegmensnek (Z)
C225  0A           LD   A,(BC)      ;minta
C226  86           ADD  A,(HL)      ;+ saját komplementere
C227  3C           INC  A           ;+ 1 (=0, ha jól tárolt)
C228  77           LD   (HL),A      ;0 -> RAM
C229  C0           RET  NZ          ;RAM-hiba, ha nem 0
C22A  18 F4        JR   C220        ;tovább ellenőriz
```

Azt láthatjuk, hogy a tesztelés utolsó mozzanataként minden RAM-címre 0-t tölt a rutin. A RAM-szegmensek tehát *törölve* kerülnek ki az ellenőrzésből.

Ha a szegmens hibátlan, ez lesz a rendszerszegmens, ez tartalmazza majd a rendszerváltozókat, a báziscímeket, az EXOS futása alatt a Z80-as-vermet stb. Érthető tehát, hogy ha hibát talált az ellenőrzés, a rendszer ezen a ponton lefagy; pörgő keretszínnel végtelen ciklusba kerül:

```
C0AF  3D           DEC  A           ;az összes szín előfordul
C0B0  D3 81        OUT  (81),A      ;keretszín beállítása
C0B2  18 FB        JR   C0AF        ;végtelen ciklusban
```

Ezt igen könnyen kipróbálhatjuk, akár BASIC-ben is:

```
OUT 177,0              ! A 0. SZG. az 1. LAP-ra
CALL USR (16559,0)     ! 40AFH belépési cím
```

(itt még szerencsére segít a RESET gomb).

A Z80-as veremmutató (Stack Pointer) az ellenőrzés alatt ideiglenesen a még ellenőrzetlen 254. szegmensbe kerül, de a továbbiakban (és általában az EXOS-ban) SP = B217H-ról indul a 255. (rendszer-) szegmensben.

A további ellenőrzés már kijelzés mellett folyik, de ennek még vannak előfeltételei: ebben a szakaszban még nincs meg a NICK-chip működéséhez nélkülözhetetlen sorparaméter-tábla (LPT), sőt a karaktergenerátor (a kijelezhető karakterek alakját, pontmátrixát tároló memóriaterület) sem. Mindezt a VIDEO perifériakezelő inicializálás rutinjának meghívásával (0.SZG. D260H belépési cím) hozza létre az EXOS.

A rutin hívása után a képernyő sötét, a képernyő legfelső sorában (az ún. állapotsorban) megjeleníthető memóriaterület pedig a belapozott rendszerszegmens BEB8H címén kezdődik. Ide tölti be a program közvetlenül a kijelzendő karaktereket, miután az állapotsort láthatóvá tette.

A következő lépésben a saját ROM-jait (0. és 1. szegmens) ellenőrzi az EXOS (ellenőrző összeg 0). Természetesen itt sem léphet tovább, ha hibát talál (ebben az esetben az "INTERNAL CHEKSUM ERROR" feliratot ír az előkészített állapotsorba).

Ezek után a RAM-szegmensek ellenőrzése következik, a már megismert módon. Felülről lefelé (a 254.-től a 8.-ig) végigvizsgál minden szegmenst (részletesen csak azokat, ahol legalább az első címen írható memóriát talált). Ez a kulcsa annak, hogy tetszőleges memóriakiterjesztés esetén korrekt RAM-könyvtárat készíthessen a rendszer.

Eközben az állapotsorban az

"EXOS 2.1 TESTING ERROR"

felirat látható (ill. az "ERROR" remélhetőleg nem, hiszen jó RAM-ok esetén ez fekete színű). A látható sor végére viszont kiírja (hexadecimális alakban) az ellenőrzött szegmens számát.

Az ellenőrzés során számlálja a jó és rossz RAM-okat és elkészíti a jó RAM-szegmensek táblázatát: az ABD0H címtől lefelé sorra bejegyzi minden jónak talált RAM szegmensszámát. Az eljárás végére ki is alakul néhány fontos rendszerváltozó értéke (zárójelben a vizsgált verzióban, hibamentes esetben adódó értékek):

BF9A/BH: a RAM-szegmensek táblázatának kezdőcíme (ABD0H)
BF9FH: a szabad szegmensek száma (6)
BFA2H: a rendszerszegmensek száma (1)
BFA3H: az összes működő RAM-szegmensek száma (8)
BFA4H: a hibás RAM-szegmensek száma (0)
BFFCH: a 0. LAP szegmense; USER-P0 (F8H)

A RAM-tábla: ABCAH ABD0H

F9H, FAH, FBH, FCH, FDH, FEH, FFH

A szabad és rendszerszegmensek kezelésekor az EXOS nem veszi figyelembe a legalsó jó RAM-ot (esetünkben a 248. (F8H) szegmens), ami igencsak kötött felhasználású: ez lesz majd a 0. LAP gyakorlatilag állandóan belapozott szegmense. Így érthető, hogy a rendszer leállásához vezet, ha az ellenőrzés nem talál legalább még egy jó szegmenst a már ellenőrzött 255. szegmens mellé.

Ezzel alakul ki a felsorolt változók értéke: a talált nyolc RAM-szegmensből hetet vehet figyelembe az EXOS (F9H-FFH), ebből a legmagasabb sorszámút (FFH) a rendszer már igénybe vette, tehát hat szegmens marad a felhasználó számára szabadon.

Mielőtt továbbmennénk, álljunk meg egy pillanatra egy terminológiai problémánál. Az előbbiekben megismert RAM-szegmens táblázathoz hasonlóan szinte minden táblázat, lánc felülről lefelé épül fel. Egy táblázat utáni bájt címe tehát (a táblázat végcíme —1).

Hogy elkerüljük a fogalmakban rejlő ellentmondást — és ezzel a félreértéseket —, a továbbiakban az előtti, utáni megjelölések helyett az adott esetekben az alatti, fölötti kifejezéseket használjuk (a "fölső" alatt természetesen mindig a magasabb címet értve).

*A ROM-szegmensek könyvtárazása*

Az inicializálás folyamatában a RAM-ok ellenőrzése után a következő lépés a ROM-ok nyilvántartásba vétele. Először a — tizenhatos számrendszerben — kerek értékű szegmenseket (F0H, E0H, ...10H) vizsgálja végig a rendszer: ellenőrzi, hogy EXOS-ROM karakterekkel (45H, 58H, 4FH, 53H, 5FH, 52H, 4FH, 4DH bájtokkal) kezdődik-e az adott szegmens.

Ha igen, könyvtárazza azt. A RAM-szegmensek táblázata alatt felépített ROM-szegmens táblázatban 4—4 bájt tartozik minden bejegyzett ROM-hoz: a legfelső bájt a ROM-szegmens száma, alá majd később jegyzi be a rendszer a ROM számára esetleg kiosztott RAM-terület szegmenssszámát és címét.

Ha — pl. címdekódolási okok miatt — több szegmenssszámon is megjelenik az adott ROM, az EXOS csak a legalsó számot veszi fel a nyilvántartásba.

Ezekre a kerek címekre vagy a 7.—4. szegmensre kell tehát dekódolni (és a fenti bájtokkal kezdeni) az IS-BASIC melletti további kiterjesztésre szánt esetleges EPROM-okat.

A következő lépésben a 7.—4. szegmenseket ellenőrzi és könyvtárazza a rendszer (esetünkben), végül bejegyzi a táblázatba az 1. szegmenssszámot is.

Az elkészített táblázat ebben a fázisban a vizsgált alapkiépítésű gép esetén:

```
ABB9H 00             ; tábla vége (alsó bájt)
ABBAH 00 00 00 01    ; 1. szegmens
ABBEH 00 00 00 04    ; 4. szegmens
ABC2H 00 00 00 05    ; 5. szegmens
ABC6H 00 00 00 00    ; a tábla felső vége
```

Az EXOS beállítja a tábla két végét jelző rendszerváltozókat is:

BF9C/DH: a ROM-tábla legfelső bájtjára mutat (ABC9H)
BF97/8H: a ROM-tábla legalsó bájtjára mutat (ABB9)

Egyelőre ezzel azonos értékűre állítja a rendszer munkaterületének kezdőcímét is (BF95/6H, értéke ekkor ABB9H).

A táblázatok elkészítése után az EXOS alapértéket ad néhány rendszerváltozónak:

BFE8H : LV-TAPE (2, azaz 40 mV)
BFD5H : BAUD-SER (0FH, azaz 9600 bit/s)
BFE7H : PROTECT (FFH, azaz védett)

*Rendszerrutinok kiépítése*

Az inicializálás következő lépése a rendszer működése szempontjából alapvető fontosságú két RAM-terület feltöltése:

— a rendszerszegmensre másolja (a B217H címtől) a 0. ROM C3AAH—C3E4H területét
— a 0. LAP-ra lapozott F8H-as szegmensre másolja (a 0030H címtől) a C3E7H—C40FH ROM-területet.

A nulláslapon kialakított terület lesz az EXOS n funkcióhívások belépési és visszatérési ága.

A rendszerszegmens B217H—B33DH területén kialakított alprogram az EXOS lapozási technikájának alaprutinja. Mint már említettük, a különböző rendszerek programjai általában a 3. LAP-on futnak. Ha tehát át akarunk térni egy másik szegmensre (vagy meg akarjuk hívni annak valamelyik alprogramját), meg kell oldanunk a lapozás alapproblémáját: az OUT (B3H),A utasítás nem lehet a 3. LAP-on futó program része, hiszen a végrehajtás után a következő bájtot (utasítást) már az átlapozott szegmensről olvassa be a CPU. Nincs tehát időnk a JP vagy CALL utasítás kiadására. Az átlapozásnak feltétlenül másik szegmensen kell történnie.

Az EXOS erre a célra a 2. LAP-on látszó rendszerszegmenst — abban az előbb bemásolt rutint — használja:

```
                                    az A-ban hozott szegmenst
B217  D3 B3       OUT   (B3),A      :lapozza a 3. LAP-ra
B219  08          EX    AF.AF'      :átadandó reg. és státusz
B21A  CD 25 B2    CALL  B225        :lényegében  CALL(HL)
B21D  08          EX    AF,AF'      :
B21E  AF          XOR   A           :a 0. szegmenst
B21F  D3 B3       OUT   (B3),A      :lapozza vissza
B221  08          EX    AF,AF'      :visszaadott reg.+státusz
B222  C9          RET               :

B223  D3 B3       OUT   (B3),A      :hozott SZG. a 3. LAP-ra
B225  E9          JP    (HL)        :a célprogramra ugrik

B226  7C          LD    A,H         :eredeti 3. LAP
B227  D3 B3       OUT   (B3),A      :visszalapozása
B229  7D          LD    A,L         :eredeti A
B22A  2A 7C BF    LD    HL,(BF7C)   :eredeti HL
B22D  C3 51 00    JP    0051        :EXOS visszatérési
                                     főágra
```

```
B230  57 72 69 74 74 65 6E 20  Written
B238  62 79 3A A0 CD F2 EC A0  by: Mrl
B240  C2 D4 A0 CE CD D6 A0 C7  BT NMV G
B248  CE C8 A0 C3 C7 C5 A0 C1  NH CGE A
B250  C5 CC                    EL
```

B217H belépésekor A-ban és HL-ben a célrutin szegmensszáma és belépési címe van, AF'-ben paramétert és státuszt adhatunk át, a visszaadott paramétert és státuszt az AF-ben kapjuk meg. A többi regiszter szabadon használható.

A B223H belépés tetszőleges JP(HL), a B226H-B22DH szakasz pedig az EXOS n funkcióhívások visszatérési ágát szolgálja ki.

A rutinokkal együtt a rendszerszegmensbe másolt további bájtok copyright jellegű üzenetet hordoznak: a BF30H-BF51H területen a "Written by: Mrl BT NMV GNH CGE AEL" szöveg található, amit meg is jeleníthetünk az állapotsorban a POKE 49119,42 utasítással (l. a 2.4 pontot).

A lapozó rutin alapvető fontossága alapján érthető, hogy a rendszer fokozottan védi ezt a memóriaterületet: megváltoztatása azt eredményezi, hogy még a melegindítási kísérlet is (RESET) hidegindításba fordul. Már kevésbé indokolt a terület végének (a szövegtartománynak) ez a nagyfokú védelme. Ellentmondásban is áll a gép — egyébként minden ízében érezhető — felhasználócentrikusságával, hiszen ezzel egy igen kényelmesen használható képernyőüzenet területet tettek problematikussá.

A rendszer felépítésének folytatásához az EXOS átugrik az 1. szegmens C00DH címére az előbbiekben tárgyalt lapozórutin segítségével, miután az egyébként sokoldalú belépési ponthoz a hidegindítást biztosító paramétert állított be (A = 6).

Az 1. szegmensen a vezérlés azonnal az E40FH címre, innen — A értéke alapján — az E66EH címre adódik. Itt folytatódnak a hidegindítás funkciói.

Itt a rendszer mindenekelőtt egy EXOS 0 funkcióhívást ad ki, C = 20H RESET-jelzővel. A funkcióhívásokról a következő pontban írunk részletesebben, de mivel ez a hívás hozza létre azt a háttérállapotot, memóriakiosztást, amivel a felhasználó gazdálkodhat, kövessük itt is a hidegindítás folyamatát!

A funkcióhívások közös bevezető szakasza után az EXOS 0 végrehajtása (újra a 0. szegmensen) a C653H címen kezdődik. A rendszer mindenekelőtt törli (0-ra állítja) a

BFF8/9 RST-ADDR
003D/EH SOFT-ISR
BFED/EH USER-ISR

rendszerváltozókat. Ezután a végrehajtás szétválik a RESET-jelzők alapján. Esetünkben (C = 20H) a C23AH rutin következik.

*A RAM-igények kielégítése*

Az első érdemi funkció, amit itt a hidegindítás során elvégez, további EXOS

változók alapértékének beállítása (ezeket a funkcióhívás leírásánál adjuk meg). Ezután viszont a RAM-felhasználást alapvetően érintő lépés következik: az EXOS végigkérdezi az előbbiekben felépített ROM-táblázat bekönyvtározott ROM-jait, hogy igényelnek-e RAM-ot a rendszertől. Mivel ez fontos lépés a memóriatérkép kialakításában, másrészt az igénylés és a kielégítés módja is fontos lehet a gépi kódban programozó számára, tekintsük át az eljárást. Az áttekinthetőség érdekében azt az utat kövessük, amit a rendszer az eddig tárgyalt alapkiépítés esetén jár be:

```
C2E3  2A 9C BF     LD   HL,(BF9C)   ;ROM-szegmensek táblája
C2E6  E5           PUSH HL          ; (felső byte)
C2E7  E1           POP  HL          ;verem -> báziscím
C2E8  2B           DEC  HL          ;-1
C2E9  2B           DEC  HL          ;-2
C2EA  2B           DEC  HL          ;-3
C2EB  2B           DEC  HL          ;-4
C2EC  7E           LD   A,(HL)      ;a következő ROM
C2ED  B7           OR   A           ; szegmensszáma
C2EE  C8           RET  Z           ;vissza, ha tábla vége
C2EF  E5           PUSH HL          ;báziscím -> verem
C2F0  0E 07        LD   C,07        ;akciókód: RAM-igény
C2F2  21 0A C0     LD   HL,C00A     ;ROM belépési pont
C2F5  CD 17 B2     CALL B217        ;belapozza és meghívja
C2F8  79           LD   A,C         ;visszaadott C
C2F9  B7           OR   A           ;ha 0:kell RAM
C2FA  20 EB        JR   NZ,C2E7     ;következő ROM-ra, ha
                                     ez nem igényel RAM-ot
C2FC  78           LD   A,B         ;kért RAM típusa
C2FD  42           LD   B,D         ;B=kért RAM nagysága MSB
C2FE  4B           LD   C,E         ;C=     "            LSB
C2FF  0F           RRCA             ;BIT 1,A -> CY
C300  F5           PUSH AF          ; (hol kér RAM-ot?)
C301  30 19        JR   NC,C31C     ;elugrik, ha az 1. LAP-on

Az igényelt RAM-ot a 2. LAP-on kéri (a rendszerszegmensben):

C303  2A 95 BF     LD   HL,(BF95)   ;rendszerterület kezdő címe
C306  B7           OR   A           ;NC státusz
C307  ED 42        SBC  HL,BC       ;szabad ter. kezdő címe -
                                     - igényelt RAM nagysága
C309  38 11        JR   C,C31C      ;(a 2. LAP-on nem fér el)
C30B  11 E8 83     LD   DE,83E8     ;03E8H hely kell még a
                                      rendszernek is
C30E  ED 52        SBC  HL,DE       ;így is elfér a 2. LAP-on?
C310  38 0A        JR   C,C31C      ;elugrik, ha nem
C312  19           ADD  HL,DE       ;(BF95)-BC visszaállítás
C313  22 95 BF     LD   (BF95),HL   ;rendszerterület új kezdőc.
C316  F1           POP  AF          ;RAM-típus?
C317  0F           RRCA             ;az 1. LAP-on is megfelel?
C318  3E FF        LD   A,FF        ;rendszerszegmens
C31A  18 17        JR   C333        ;

C333  30 04        JR   NC,C339     ;tovább, ha a 2. LAP-on kell

C339  EB           EX   DE,HL       ;DE= a kapott RAM kezdő címe
C33A  E1           POP  HL          ;ROM-tábla: a szegmens címe
C33B  2B           DEC  HL          ;-1
```

```
C33C  77        LD   (HL),A     ;a kapott szegmens száma
C33D  2B        DEC  HL         ;-2
C33E  72        LD   (HL),D     ;RAM kezdőcím HI
C33F  2B        DEC  HL         ;-3
C340  73        LD   (HL),E     ;RAM kezdőcím LO
C341  18 A8     JR   C2EB       ;a ROM-tábla következő elemére
```

A rutinból is látható, hogy a táblázatban tárolt ROM-ok belépési pontja a C00AH cím. A táblából megállapított szegmensszámú és a B217H lapozórutin által a 3. LAP-ra kapcsolt és meghívott ROM-ok a C = 7 akciókód hatására megadhatják a RAM-igényüket (a DE regiszterpárban a nagyság, B-ben a helye: B = 1 esetén a 2. LAP-on, B = 2 esetén az 1. LAP-on). Ha megvizsgáljuk a bejegyzett ROM-ok (sorrendben az 5., 4., 1) belépési pontjait (erről még lesz szó a továbbiakban), megállapíthatjuk, hogy közülük csak a BASIC-BRD (esetünkben a 4.) szegmens reagál a C = 7 akciókódra, innen C = 0: kell RAM; B = 0: a 2. LAP-on; DE = 04B3H értékekkel tér vissza a rendszer).

Ezek szerint a ROM-okat lekérdező ciklusból csak a 4. szegmensnél lép ki a rendszer, és mivel ez a rendszerszegmensben kér RAM-ot, a C303H programrészletben kapja meg. A rendszerterület kezdő címe (BF95/6H) belépés előtt a ROM-táblák aljára mutatott (ABB9H), a kiosztott RAM lefoglalása után tehát a munkaterület az ABB9H – 04B3H = A706H cím alatt kezdődik. Ezzel alakul ki az érintett rendszerváltozók BASIC-ben is lekérdezhető értéke, valamint a ROM-tábla megváltozott eleme:

ABBEH 06 A7 FF 04

*A beépített perifériakezelők könyvtárazása*

Az EXOS 0 inicializálási folyamatában ezután a periférialánc felépítése következik. A rendszerszegmensben elkészülő lánc elemeit, a periférialeírókat a ROM-okban tárolt perifériainformációk (álleírók) alapján alakítja ki az EXOS.

Minden ROM-szegmens elején a C008/9H címen található az a kétbájtos cím, ami az adott ROM első perifériájának adatmezőjére mutat (a cím az 1. LAP-on értelmezett cím kell legyen, mint látni fogjuk). Ez az adatmező is tartalmaz egy pointert (mutatót) a következő perifériára stb., a lánc végéig, amit a pointer 0000 értéke mutat (ha az adott ROM-ban nincs perifériakezelő, már a C008/9H címen 0 van). Lássunk egy konkrét példát (az EXOS leírás terminológiájával) az álleírók most figyelembe vett elemeire: a 0. szegmens C008/9H címén az 5780H pointer érték található. Ez a VIDEO periféria álleírójának méretmeghatározó (SIZE) mezőjére mutat. Az 1. LAP címtartományában az álleíró:

| Cím | Érték | Funkció | |
|---|---|---|---|
| 576F | 6B7B | (NEXT) | mutató a következő periféria (SOUND) álleírójára |
| 5771 | FFFE | (RAM) | a leíróhoz szükséges RAM (itt: 0 bájt, l. a 2.5 alfejezetet) |
| 5773 | 00 | (TYPE) | típus (itt 0-nak kell lennie) |
| 5774 | 20 | (IRQFLAG) | milyen megszakításokat kezel (itt: VIDEO) |
| 5775 | 01 | (FLAGS) | VIDEO periféria jelző (máshol ez 0) |
| 5776 | 5781 | (TAB) | belépési pont táblázat címe |
| 5778 | 00 | (TAB-SEG) | táblázat szegmense (értékét felülírja az EXOS) |
| 5779 | 00 | (UNIT-COUNT) | lehet-e több periféria is ezen a néven (nem) |
| 577A | 05 | (NAME) | a perifériánév hossza |
| 577B | | | VIDEO a perifériánév karakterei |
| 5780 | 0D | (SIZE) | a leíróba átmásolandó bájtok száma (TYPE-tól (SIZE-1-ig) |

Az EXOS a munkaterület kezdőcímétől (BF95H értéke, eddig követett példánkban A706H) lefelé tárolja a leírókat. A leírók alatti szabad terület kezdőcímét (BF93H értéke) minden lépésben a befejezett leíró alá állítja.

Először a 0. szegmens periférialáncát követi (és könyvtárazza) végig, majd a ROM-táblázat alapján végigvizsgálja az összes — bejegyzett — ROM-ot és azok perifériáit is nyilvántartásba veszi. Az eljárás érdekessége, hogy ebben az esetben alulról felfelé veszi sorra a ROM-szegmenseket (0-1-4-5 sorrendben), tehát a magasabb sorszámú bővítőknek mindig lehetőségük van felülírni a beépített kezelőket.

Egy leíró bejegyzését a rendszer a következő lépésekben végzi:

— a vizsgált ROM-szegmenst az 1. LAP-ra lapozza;
— beolvassa a láncmutatót (a SIZE mezőre mutat). Ha a mutató értéke 0, a következő szegmenst vizsgálja;
— az álleíró elemeit (a típusbájttól a nevéig) áttölti a munkaterületre a BF93H aktuális értéke alá;
— beolvassa a ROM-ból az álleíró RAM-igényét, és ennyi helyet szabadon hagy a bejegyzett leíró és az alá írandó 3-bájtos mutató alatt. BF93H-at az utolsó lefoglalt bájtra állítja;
— beolvassa a BFC0H báziscím alatti 3 bájtot és helyére az éppen bejegyzett leíró szegmensszámát (báziscím – 1) és tárolási címét tölti (báziscím –3/–2 a leíró TYPE-bájtjára mutat). A báziscím alól kiolvasott pointert letárolja a bejegyzett leíró alá. Mivel a legelső bejegyzés előtt a BFC0H báziscím alatti bájtok értéke 0 volt (a hidegindításnál minden RAM törlődött), egy későbbi bejegyzésnél pedig itt az előzőleg felvett leíró tárolási címe és szegmensszáma található, az eljárás folytatásával egy lánc alakul ki. A láncban a BFC0H báziscím alatti pointer a legutoljára felvett leíróra mutat, e leíró előtti pointer az előtte felvett leíróra stb. a legelőször felvett leíróig, aminek pointere 0;
— ezután — ha a leíró UNIT-COUNT eleme 0, azaz nem lehet több leíró ilyen

néven — végigvizsgálja a már bejegyzett leírókat a láncban és, ha az éppen felvett perifériára talál másik bejegyzést, érvényteleníti azt (a TYPE-bájtot FF-re állítja). Ilyen bejegyzést találhatunk a vizsgált konfigurációnál is (mint a fejezet végén, az ellenőrző programmal látni fogjuk);

— a leíró felvételének befejező lépéseként a TAB-SEG mezőre azt a szegmensszámot írja, amelyen megtalálta az érintett periféria álleíróját. A felhasználónak tehát egy ilyen perifériakezelő (és annak álleírója) elkészítésekor — a rendszert kifejlesztők jól érezhető koncepciójának megfelelően — akár nem is kell tudnia, konkrétan melyik szegmensre kerül az adott részszoftver.

A láncszemek felfűzését jól követhetjük a ROM — minden láncépítési folyamatban hívott — rutinján:

```
C636  2B          DEC  HL          ;báziscím-1
C637  46          LD   B,(HL)      ;előző szegmensszám
C638  77          LD   (HL),A      ;helyére az új érték
C639  2B          DEC  HL          ;báziscím-2
C63A  7E          LD   A,(HL)      ;előző cím HI
C63B  72          LD   (HL),D      ;helyére az új érték
C63C  2B          DEC  HL          ;báziscím-3
C63D  4E          LD   C,(HL)      ;előző cím LO
C63E  73          LD   (HL),E      ;helyére az új érték
C63F  EB          EX   DE,HL       ;HL: az új láncelem címe

Az új láncelem alá beírja a báziscím alól kiemelt pointert:

C640  2B          DEC  HL          ;cím-1
C641  70          LD   (HL),B      ;előző szegmensszám
C642  2B          DEC  HL          ;cím-2
C643  77          LD   (HL),A      ;előző cím HI
C644  2B          DEC  HL          ;cím-3
C645  71          LD   (HL),C      ;előző cím LO
C646  C9          RET              ;
```

Ezt az eljárást folytatja az EXOS egy ROM-on belül addig, amíg a periférialánc végére nem ér (NEXT = 0000H), a ROM-ok során pedig addig, amíg el nem éri a ROM-táblázat végét, amíg a ROM-szegmens száma helyéről 0-t nem olvas be.

*A csatornaterület*

A felülről lefelé kiépített periférialánc alatt kezdődhet az igazi EXOS munkaterület: a csatornák leíró és puffermemóriái. A periférialánc legalsó bájtjára a BF93H redszerváltozó mutat. Erre állítja az EXOS a BFBDH alatti 3-bájtos mutatót (cím + szegmens). A BFBDH báziscím tehát a felhasználók által felépített csatornalánc kiindulási pontja lesz. Maga az EXOS 0 a hidegindítás folyamatában nem nyit tényleges csatornákat, csak egy töredék leírót (az alapértelmezés szerinti 0-s csatornához tartozó néhány bájtot) helyez el, ami alá kerülhet majd a hozzárendelt periféria mutatója és a munkaterület. Egyelőre ide — mint az EXOS munkaterület alsó határára — állítja be a rendszer a BF91/2H rendszerváltozó, az EXOS határ értékét (0—16 k közötti értékre transzformálva).

Ezzel lényegében kialakult a rendszermemória. Az EXOS legalábbis már csak az alapértelmezésű periféria nevét (TAPE) helyezi el rögzített munkaterületen (BF18H: hossz, BF19H-tól a karakterek), és a megszakítás engedélyezést állítja be a BFC5H rendszerváltozóban és a DAVE-chip B4H-es regiszterén.

A felhasználócentrikus logikára jellemző módon azonban még megadja a lehetőséget a perifériáknak, majd a rendszerbővítőknek a saját inicializálásra is.

A perifériakezelők — funkciójuknak, működési logikájuknak megfelelően — különböző feladatokat látnak el, amikor az EXOS meghívja belépési-cím táblázatuk 12. (inicializálás) címét. A VIDEO-kezelő pl. újra beállítja a sorparaméter-táblát (LPT), az 'EDITOR' csak egy bájttal jelzi saját munkaterületén, hogy megtörtént az inicializálás, a nyomtató-kezelő pedig semmit nem csinál.

A rendszerbővítők lekérdezésénél egy olyan báziscímhez is fordul az EXOS, amiről eddig még nem volt — hidegindításnál nem is lehetett — szó. A BFC3H báziscím alá írja majd a rendszer a magnetofonról beolvasott, vagy általunk kreált bővítők (pl. PASCAL, ASMON stb.) láncának mutatóját. A bővítők ui. ugyanolyan láncba fűzhetők, mint a periférialeírók vagy a csatornák. Mivel az így könyvtárazott bővítők státusza egészen különleges a gép életében, ezzel a kérdéssel még érdemes foglalkozni.

Az EXOS C = 8 akciókóddal (Inicializálás) először ezeket a bővítőket kérdezi le (de természetesen a hidegindítás alatt a bővítők lánca még üres), majd a ROM-táblázat alapján végigkérdezi a ROM-szegmenseket is. Az alapkonfigurációban egyedül a 4. szegmens lát el tényleges feladatokat ennél a hívásnál (előkészíti a német nyelvű hibaüzenetek kiírását, a német karakterkészletnek megfelelően állítja be a karaktergenerátort stb.).

Ezzel tulajdonképpen be is fejeződött a rendszer inicializálása; kialakult a jellemző memóriatérkép. Ez a rendszer azonban még nem képes kapcsolatot tartani a környezetével, nincsenek meg a gép és perifériái (pl. billentyűzet, képernyő stb.) közötti utak: a csatornák. Ezeket a felhasználónak kell igényei szerint kialakítani. Ezt teszi meg — ideiglenesen — az 1. szegmens is a bejelentkezés érdekében, miután az inicializálás után ide tér vissza a rendszer.

*Bejelentkezés*

A csatornaparaméterek beállítása, a csatornák megnyitásának módja számunkra is érdekes lehet. Kövessük hát ezt az eljárást.

A felhasznált F745H rutin a HL címen kezdődő táblázat alapján dolgozik. A 80H-nál kisebb beolvasott értékeket EXOS változó sorszámnak tekinti és beírja a következő táblaelemet. A nagyobb értékeket csatornaszámként kezeli és megnyitja azt arra a perifériára, amelynek nevére a tábla következő két eleme mutat:

```
F745  08        EX   AF,AF'      ;
F746  4E        LD   C,(HL)      ;érték beolvasás
F747  CB 79     BIT  7,C         ;EXOS változó?
F749  20 09     JR   NZ,F754     ;ugrik, ha csatorna

F74B  06 01     LD   B,01        ;"írás" jelző
```

```
F74D  23          INC  HL          ;következő adatra
F74E  56          LD   D,(HL)      ;D: írandó érték
F74F  23          INC  HL          ;következő adatra
F750  F7 10       EXOS 10          ;EXOS változó írása
F752  18 F2       JR   F746        ;olvasást folytat

Csatornanyitás

F754  79          LD   A,C         ;A: csatornaszám
F755  23          INC  HL          ;következő adatra
F756  5E          LD   E,(HL)      ;név címe LO
F757  23          INC  HL          ;
F758  56          LD   D,(HL)      ;név címe HI
F759  23          INC  HL          ;
F75A  FE FF       CP   FF          ;csatornaszám=FF?
F75C  28 27       JR   Z,F785      ;  (itt nem)
F75E  F7 01       EXOS 01          ;csatornanyitás
F760  C8          RET  Z           ;vissza
```

Ezt a rutint felhasználva állítja be az E69DH rutin a bejelentkezéshez szükséges paramétereket, megnyit egy hardver szöveglapot az "1985 Intelligent..." üzenethez, egy 256 színű grafikus lapot a nagybetűs, villogó "ENTERPRISE" felirathoz és a billentyűzetcsatornát, hogy kiléphessünk ebből a végtelen ciklusból:

```
E69D  21 3E E7    LD   HL,E73E     ;tábla kezdőcím
E6A0  3C          INC  A           ;
E6A1  CD 45 F7    CALL F745        ;98H csatorna
E6A4  C0          RET  NZ          ; (szöveglap)
E6A5  CD 45 F7    CALL F745        ;97H csatorna
E6A8  C0          RET  NZ          ; (grafikus)
E6A9  CD 45 F7    CALL F745        ;96H csatorna
E6AC  C0          RET  NZ          ; (billentyű)
E6AD  3E 98       LD   A,98        ; szöveglap
E6AF  01 01 01    LD   BC,0101     ; első két sora
E6B2  11 15 02    LD   DE,0215     ; kijelzés
E6B5  F7 0B       EXOS 0B          ; a 21. sortól
E6B7  C0          RET  NZ          ;
E6B8  3E 97       LD   A,97        ; grafikus lap
E6BA  01 01 01    LD   BC,0101     ; első két sora
E6BD  11 0A 02    LD   DE,020A     ; kijelzés
E6C0  F7 0B       EXOS 0B          ; a 10. sortól
E6C2  C0          RET  NZ          ;

"1985 Intelligent Software Ltd" kijelzés

E6C3  21 06 E7    LD   HL,E706     ;szövegre mutat
E6C6  0E 98       LD   C,98        ;szöveglap
E6C8  CD F9 E6    CALL E6F9        ;kiírás
E6CB  C0          RET  NZ          ;
E6CC  3E 96       LD   A,96        ;billentyűzet
E6CE  F7 09       EXOS 09          ;karakterolvasás
E6D0  B1          OR   C           ;volt billentyű?
E6D1  C8          RET  Z           ;vissza, ha igen
```

```
Grafikus lap beállítása:

E6D2  21 2C E7     LD   HL,E72C     ;kiírandó bájtokra
E6D5  CD F7 E6     CALL E6F7        ;kurzorpozíció
E6D8  C0           RET  NZ          ; beállitás (0.71)
E6D9  7E           LD   A,(HL)      ;"ENTERPRISE" szöveg
E6DA  B7           OR   A           ;következő karaktere
E6DB  28 EF        JR    Z,E6CC     ;ismétlés, ha vége
E6DD  E5           PUSH HL          ;szövegcím verembe
E6DE  21 04 01     LD   HL,0104     ;munkaterület
E6E1  77           LD   (HL),A      ;karakter pufferbe
E6E2  2B           DEC  HL          ;elé:
E6E3  ED 5F        LD   A,R         ;véletlen érték
E6E5  77           LD   (HL),A      ;pufferbe
E6E6  2B           DEC  HL          ;elé:
E6E7  36 49        LD   (HL),49     ;"I": tintaszín
E6E9  2B           DEC  HL          ;elé:
E6EA  36 1B        LD   (HL),1B     ;"ESC"
E6EC  2B           DEC  HL          ;elé:
E6ED  36 04        LD   (HL),04     ;"szöveg" hossza
E6EF  CD F7 E6     CALL E6F7        ;karakter kiírása
                                     véletlen színnel
E6F2  E1           POP  HL          ;szövegcím vissza
E6F3  C0           RET  NZ          ;
E6F4  23           INC  HL          ;következő betűre
E6F5  18 E2        JR   E6D9        ;     vissza

Szöveg kiírása pufferből a csatornára:

E6F7  0E 97        LD   C,97        ;csatornaszám
E6F9  46           LD   B,(HL)      ;szöveg hossza
E6FA  23           INC  HL          ;következő kar.
E6FB  C5           PUSH BC          ;
E6FC  79           LD   A,C         ;A: csatornaszám
E6FD  46           LD   B,(HL)      ;B: irandó érték
E6FE  23           INC  HL          ;következő kar.-re
E6FF  F7 07        EXOS 07          ;karakter írása
E701  C1           POP  BC          ;
E702  C0           RET  NZ          ;
E703  10 F6        DJNZ E6FB        ;végig a szövegen
E705  C9           RET              ;
```

A felhasznált táblázatok:

```
E73E  16 00 17 00 18 28 19 02   MODE_VID=0    :szöveglap
                                COLR_VID=0    :2 szín
                                X_SIZ_VID=40  :szélesség
                                Y_SIZ_VID=2   :magasság
E746  98 34 FE                  98h csatorna  :"VIDEO:"
E749  16 01 17 03               MODE_VID=1    :HIRES gr.
                                COLR_VID=3    :256 szín
E74D  97 34 FE                  97h csatorna  :"VIDEO:"
E750  96 4C FE                  96h csatorna  :"KEYBOARD"
```

```
6706  25 1B 6F 20 20 20 20 80  ...     .
670E  20 31 39 38 35 20 49 6E   1985 In
6716  74 65 6C 6C 69 67 65 6E  telligen
671E  74 20 53 6F 66 74 77 61  t Softwa
6726  72 65 20 4C 74 64        re Ltd


672C  06 1B 41 00 00 47 00 45  .....E
6734  4E 54 45 52 50 52 49 53  NTERPRIS
673C  45 00                    E
```

A "VIDEO:" és "KEYBOARD:" szövegek az 1. lap FE34H, ill. FE4CH címén találhatóak. A rutinokat felhasználni kívánó Olvasónak jó tudni, hogy ezek mellett le vannak tárolva az "EDITOR:" (FE3BH) és "printer:" (FE43H) periférianevek is.

A megnyomott billentyű hatására a rendszer lezárja az ideiglenesen megnyitott csatornákat, majd (0-t írva a műveleti kód tárolóba (BF78H)) felkínálja a vezérlést a bővítőknek: egy EXOS 26 hívást hajt végre.

Ennek hatására az EXOS C=1 (Hidegindítás) kóddal meghívja minden rendszerbővítő, majd minden ROM belépési pontját (C00AH). Ekkor az alkalmazói program átveheti a vezérlést. Ha minden bővítőből visszatérne a rutin (változatlan C-regiszterrel), a sor végén ott áll az 1. szegmens, annak WP alkalmazói programja (a szövegszerkesztő), ami csak erre a hívásra vár (ha nincs bővítőmodul a gépben, akkor éppen ez a helyzet áll elő: a bejelentkezés után a szövegszerkesztő kezd dolgozni).

A vizsgált kiépítésben azonban a ROM-tábla első szegmense (az 5. BASIC-ROM) azonnal átveszi a vezérlést és kialakítja a BASIC rendszert.

Ezzel végre végetért az inicializálás (legalábbis az EXOS-é), a szó ettől a pillanattól az alkalmazóé.

Mielőtt azonban továbblépnénk, beszéljünk még egy inicializálás vonatkozású folyamatról: a melegindításról. A felhasználó beavatkozásával ui. a tárgyalt indítási folyamat — a felépített rendszer és a felhasználói memóriák védelmében — lényegesen enyhébb irányba is fordítható. Ehhez az alkalmazónak egy 0-tól különböző értéket (praktikusan saját inicializáló belépési címét) kell betölteni a RST-ADDR rendszerváltozóba (BFF8/9H, 49144/5). A RESET gomb megnyomására a hidegindítási pontról elinduló programág ui. — mielőtt még "drasztikusabb" lépéseket tenne — megvizsgálja ezt a címet. Ha ez nem 0 (valamint a védett rendszerterületet (B217H—B251H) is épnek találja), akkor csak egy lényegesen kevesebb változással járó EXOS 0-t hajt végre, C = 10 RESET-jelzővel. Előtte PUSH utasítással tárolja a talált RST-ADDR címet (így majd az EXOS 0 végén álló 'RET' hatására a felhasználó melegindítási rutinjára fog ugrani) és — ideiglenesen — 0-t ír a RST-ADDR cím helyére. Ezért, ha mégegyszer megnyomjuk a RESET gombot, mielőtt a melegindítással újraéled a rendszer, ez már valódi hidegindítást vált ki.

Befejezésként, az eddigiek szemléltetésére, rendszerezésére nézzünk meg egy-két programot, példát.

A 3. Függelékben összefoglalóan megadjuk a memória megismert térképét, a fontosabb rendszerváltozókat, mutatókat, de valószínűleg nem fog ártani egy konkrét példán (az eddig vizsgált verziójú gépen) megvizsgálni és értelmezni a ténylegesen kialakuló rendszermemóriát.

Ehhez készítsünk egy DUMP programot, egy olyan segédeszközt, amely (az áttekinthetőség érdekében hexadecimális formában) kiírja egy tetszőleges memóriaterület tartalmát. A program felhasználja a hexa→decimális és decimális→hexa konvertáló rutinokat (l. pl. az 1. fejezetet):

```
100 PROGRAM "DUMP"
110 NUMERIC D
120 DEF H$(N)=CHR$(INT(N/16)+48-7*
    (INT(N/16)>9))&CHR$(MOD(N,16)+
    48-7*(MOD(N,16)>9))
130 DEF DEC(H$)
140   LET D=0:LET Q=1
150   FOR N=LEN(H$) TO 1 STEP-1
160     LET D=D+ORD(HEX$(H$(N:N)))*Q
170     LET Q=Q*16
180   NEXT
190 END DEF
200 INPUT PROMPT "KEZDŐCIM ? ":A$
210 CALL DEC(A$)
220 LET KC=D
230 INPUT PROMPT "VEGCIM ? ":A$
240 CALL DEC(A$)
250 LET VC=D
260 INPUT PROMPT "SZEGMENS ? ":A$
270 CALL DEC(A$)
280 LET SG=D
290 FOR N=KC TO VC STEP 8
300   LET A$=H$(N/256)&H$(MOD(N,
      256))
310   PRINT N;" ";A$;"  ";
320   FOR I=0 TO 7
330     LET DT=SPEEK(SG,N+I)
340     PRINT H$(DT);" ";
350   NEXT
360   PRINT
370 NEXT
380 END
```

A program hexadecimális alakban kéri a kiírandó memóriaterület kezdő- és végcímét, valamint szegmensszámát. A kiíratásnál a kezdőcímek decimális alakját is megadja, így egy-egy cím közvetlenül is írható, olvasható.

Alkalmazásként próbáljuk ki programunkat az előzőekben felépített rendszerterületen, a rendszerváltozók tartományában:

```
START
KEZDőCIM ? BF90
VEGCIM ? BFF8
SZEGMENS ? FF
 49040   BF90   00 C7 13 5C A6 06 A7 B9
 49048   BF98   AB 00 D0 AB C9 AB 00 06
 49056   BFA0   00 00 01 08 00 6A D0 61
 49064   BFA8   FF 01 D7 5C FF 67 5C 66
 49072   BFB0   FF 00 00 00 61 19 00 00
 49080   BFB8   00 00 5C 66 FF 61 66 FF
 49088   BFC0   00 00 00 C1 E9 14 00 20
 49096   BFC8   00 00 00 01 00 00 FF 03
 49104   BFD0   1E 00 00 00 14 0F 00 00
 49112   BFD8   00 FF FF 00 00 28 18 00
 49120   BFE0   00 00 66 69 08 18 00 FF
 49128   BFE8   02 FF FF 00 BA 00 00 00
 49136   BFF0   CC 0E 00 00 00 B9 B8 BE
 49144   BFF8   04 01 3B AC F8 04 FF 01
ok
```

Értelmezzük együtt ezt az — első látásra talán áttekinthetetlen — adattömeget, megadva a fontosabb változók, pointerek funkcióját:

| Cím | Jelenlegi érték | Funkció, magyarázat |
|---|---|---|
| BF91/2 | 13C7 | az EXOS határértéke 0—3FFF közé transzformálva (a 2. LAP-on ténylegesen 93C7 a határ) (a csatornaterület alja) |
| BF93/4 | A65C | a periférialeíró terület alja (innen indul a csatornaterület) |
| BF95/6 | A706 | a ROM-ok részére kiutalt RAM-terület alja (ez alatt indul a periférialeírók területe) |
| BF97/8 | ABB9 | a ROM-tábla alja |
| BF9A/B | ABD0 | a RAM-táblában annak a szegmensnek a címe, amelyikben az EXOS határ van |
| BF9C/D | ABC9 | a ROM-tábla teteje (a RAM-tábla alatti cím) |

A rendszerszegmens verem alatti, viszonylag állandó táblaterülete és az ez alatti mozgó csatornaterület a 8 hibátlan RAM-szegmenssel rendelkező, angol/német BASIC modullal felszerelt gép esetén a következőképpen alakul:

| | | |
|---|---|---|
| | ABDO | RAM-tábla (8 bájt) |
| | ABCA | |
| (BF9C) | ABC9 | ROM-tábla ($4 \cdot 4 + 1 = 17$ bájt) |
| (BF97) | ABB9 | |
| | ABB8 | ROM-szegmens (4. szeg. 04B3H = 1203 bájt) pufferterülete |
| (BF95) | A706 | |
| | A705 | Periférialeíró terület (170 bájt) |
| (BF93) | A65C | |
| | A65B | Csatornaterület (igény szerint) |
| (!BF91) | 93C7 | |
| | 8000 | Szegmenshatár |

Tovább vizsgálva a rendszerváltozókat

| Cím | Jelenlegi érték | Funkció, magyarázat |
|---|---|---|
| BF9E | 0 | a megosztott szegmens száma (most 0, mert nincs ilyen: a felhasználó még nem kérte a rendszerszegmens kiutalását) |
| BF9F | 6 | a szabad szegmensek száma (az F8 szegmens kötött felhasználású, az FF-et pedig az EXOS használja) |
| BFA0 | 0 | a felhasználónak kiutalt szegmensek száma (egyelőre a BASIC elfér az F8 szegmensen) |
| BFA1 | 0 | perifériáknak kijelölt szegmensek száma |
| BFA2 | 1 | a rendszerszegmensek száma (az EXOS egyelőre elfér az FF szegmensen) |
| BFA3 | 8 | a működő RAM-szegmensek száma |
| BFA4 | 0 | a hibás szegmensek száma |

(A fenti 7 adat a rendszerállapotként lekérdezhető adatsor). Ezután a 3 legutóbb használt (legkönnyebben elérendő) csatorna adatait találjuk:

| | | |
|---|---|---|
| BFA5 | 6AH | Csatornaszám + 1 (#105: KEYBOARD) |
| BFA6/7 | 61D0H | Csatornaleíró címe a 0. LAP-on |
| BFA8 | FFH | A csatornaleíró szegmense |
| | | |
| BFA9 | 01H | |
| BFAA/B | 5CD7H | #0: EDITOR |
| BFAC | FFH | |
| | | |
| BFAD | 67H | |
| BFAE/F | 665CH | #102: VIDEO |
| BFB0 | FFH | |

A különböző láncok pointerei:

1. Csatornalánc

| | | |
|---|---|---|
| BFBA/B | 665CH | az első leíró címe a láncban (1. LAP) |
| BFBC | FFH | a leíró szegmense |

Báziscím: BFBD

2. Periférialánc

| | | |
|---|---|---|
| BFBD/E | 6661H | az első leíró címe |
| BFBF | FFH | szegmense |

Báziscím: BFC0

3. Bővítőlánc (jelenleg üres)

| | | |
|---|---|---|
| BFC0/1 | 0000H | belépési cím |
| BFC2 | 00H | bővítő szegmense (itt: lánc vége) |

Báziscím: BFC3

A láncokat egy külön programmal érdemes végigkövetni, de egyelőre nézzük végig a rendszerváltozókat:

| | | | |
|---|---|---|---|
| BFC5 | 14H | IRQ-ENABLE-STATE: | 1 Hz-es és videomegszakítás engedélyezve |
| BFC6 | 0 | | nem használt cím, az 1. EXOS változót (FLAG-SOFT-IRQ) a BFF2H címen tárolja és kezeli a rendszer |
| BFC7 | 20H | CODE-SOFT-IRQ | (még nem törlődött a STOP kód) |
| BFC8 | 0 | DEF-TYPE | (nem file-kezelő periféria, magnetofon) |
| BFC9 | 0 | DEF-CHAN | (a #FF csatornahívás ezt használja) |

| | | | |
|---|---|---|---|
| BFCA | 0 | TIMER | (nincs időzítés) |
| BFCB | 1 | LOCK-KEY | (CAPS állapot) |
| BFCC | 0 | CLICK-KEY | (engedélyezett a billentyűhang) |
| BFCD | 0 | STOP-IRQ | (a STOP szoftver megszakítást okoz) |
| BFCE | 255 | KEY-IRQ | (a többi billentyű nem okoz megszakítást) |
| BFCF | 3 | RATE-KEY | (billentyűismétlési sebesség kb. 17/s) |
| BFD0 | 30 | DELAY-KEY | (első késleltetés 0,6 s) |
| BFD1 | 0 | TAPE-SND | (magnetofonhang engedélyezett) |
| BFD2 | 0 | WAIT-SND | |
| BFD3 | 0 | MUTE-SND | (a belső hangszóró aktív) |
| BFD4 | 20 | BUF-SND | |
| BFD5 | 15 | BAUD-SER | (9600 baud átviteli sebesség) |
| BFD6 | 0 | FORM-SER | |
| BFD7 | 0 | ADDR-NET | (hálózati gépszám alapérték) |
| BFD8 | 0 | NET-IRQ | (vett adat szoftvermegszakítást okoz) |
| BFD9 | FFH | CHAN-NET | |
| BFDA | FFH | MATCH-NET | |
| BFDB | 0 | MODE-VID | (hardver szövegmód) |
| BFDC | 0 | COLR-VID | (két szín) |
| BFDD | 40 | X-SIZ-VID | (az aktuális VIDEO-lap 40 karakter széles) |
| BFDE | 24 | Y-SIZ-VID | (24 sor magas) |
| BFDF | 0 | ST-FLAG | (állapotsor kijelezhető) |
| BFE0 | 0 | BORD-VID | keretszín fekete |
| BFE1 | 1 | BIAS-VID | |
| BFE2 | 66H | VID-EDIT | (a szerkesztő videocsatorna száma) |
| BFE3 | 69H | KEY-EDIT | (a szerkesztő billentyűzetcsatornája) |
| BFE4 | 8 | BUF-EDIT | (a szerkesztő puffere 2 kbájt) |
| BFE5 | 18H | FLAG-EDIT | |
| BFE6 | 0 | SP-TAPE | (gyors szalagátvitel) |
| BFE7 | FFH | PROTECT | (védelem aktív) |
| BFE8 | 2 | LV-TAPE | |
| BFE9 | FFH | REM1 | (1. magnetofon távvezérlés kikapcsolva) |
| BFEA | FFH | REM2 | (2. magnetofon távvezérlés kikapcsolva) |
| BFEB | 0 | SPRITE | |
| BFEC | BAH | RANDOM-IRQ | (pillanatnyi érték a kiolvasáskor) |
| BFED/E | 0000 | USR-ISR | (nincs felhasználói megszakítási rutin) |
| BFEF | 0 | CRDISP-FLAG | (bejelentkezési üzenet engedélyezve) |
| BFF0/1 | 0ECC | SECOND-COUNTER | (másodpercszámláló pillanatnyi érték) |
| BFF2 | 0 | FLAG-SOFT-IRQ | (nincs megszakításkérés) |
| BFF3 | 0 | PORTB5 | |
| BFF4/5 | B900 | LP-POINTER | (a sorparaméter-tábla kezdőcíme 47360Dec) |
| BFF6/7 | BEB8 | ST-POINTER | (az állapotsor címe) |

| | | | |
|---|---|---|---|
| BFF8/9 | 0104 | RST-ADDR | (a BASIC által beállított melegindítási cím) |
| BFFA/B | AC3B | STACK-LIMIT | |
| BFFC | F8H | USR-P0 | az EXOS-t hívó |
| BFFD | 04H | USR-P1 | memóriakonfiguráció |
| BFFE | FFH | USR-P2 | |
| BFFF | 01H | USR-P3 | |

Ez a rengeteg rendszerváltozó szinte ingerel a kísérletezésre, próbálgatásra. Mi biztatjuk is erre az Olvasót, hiszen — mint a bevezetőben is írtuk — ez a legbiztosabb módja a gép megismerésének. Biztatásként nézzünk egy-két példát együtt is az EXOS memóriaszervezésére!

Nézzük meg külön a szegmenskiutalások nyilvántartását (a rendszerállapot-területet):

```
START
KEZDöCIM ? BF9E
VEGCIM ? BF9F
SZEGMENS ? FF
 49054  BF9E  00 06 00 00 01 08 00 6A
ok
```

Tehát 6 szabad szegmens van. Nyissunk meg most egy új BASIC programot (EDIT), majd lapozzunk vissza a DUMP programhoz, hogy megnézzük, milyen változások történtek:

```
EDIT 1
ok
10 REM Csak azert, hogy lefoglaljuk!
EDIT 0
ok
START
KEZDöCIM ? BF9E
VEGCIM ? BF9F
SZEGMENS ? FF
 49054  BF9E  00 05 01 00 01 08 00 6A
ok
```

Látható, hogy az EXOS egy szabad szegmenst kiutalt a felhasználó (jelen esetben a BASIC rendszer) számára, és be is könyvelte ezt a változást.

Lássunk most egy olyan változást, amely az EXOS területét érinti. Például egy nagy grafikus lap nyitása sok helyet igényel, és ezt az EXOS biztosítja:

```
DISPLAY GRAPHICS
ok
DISPLAY TEXT
ok
START
KEZDöCIM ? BF9E
VEGCIM ? BF9F
SZEGMENS ? FF
 49054  BF9E  00 04 01 00 02 08 00 6A
ok
```

Ismét csökkent tehát a szabad szegmensek száma, és a lefoglalt szegmenst saját magának utalta ki az EXOS. Ha nagyobb területet íratunk ki, azt is láthatjuk, hogy most előbbre (a RAM-tábla FEH szegmensére) mutat a BF9A/B (hiszen ebbe a szegmensbe került az EXOS határ).

Egy másik érdekes területe a rendszermemória "böngészésének" a különböző láncok követése. A lánc felépítési logikáját láttuk e fejezetben, de nézzük meg a 0. szegmens rutinján, hogyan keresi vissza maga a rendszer az elemeket:

```
C647  2B          DEC  HL          ;báziscím-1
C648  7E          LD   A,(HL)      ;A: szegmensszám
C649  2B          DEC  HL          ;báziscím-2
C64A  46          LD   B,(HL)      ;B: cím HI
C64B  2B          DEC  HL          ;báziscím-3
C64C  6E          LD   L,(HL)      ;L: cím LO
C64D  60          LD   H,B         ;HL: a láncelem címe
C64E  B7          OR   A           ;a szegmensszám 0?
C64F  C8          RET  Z           ;lánc vége, ha igen (Z)
C650  D3 B1       OUT  (B1),A      ;belapozza a láncelemet
                                    tároló szegmenst
C652  C9          RET              ;
```

A báziscím ismeretében mi is követhetjük ezt a logikát, akár egy BASIC programmal is.

```
100 PROGRAM "LANC"
110 NUMERIC D
120 DEF H$(N)=CHR$(INT(N/16)+48-7*
    (INT(N/16)>9))&CHR$(MOD(N,16)+
    48-7*(MOD(N,16)>9))
130 DEF DEC(H$)
140   LET D=0:LET Q=1
150   FOR N=LEN(H$) TO 1 STEP-1
160     LET D=D+ORD(HEX$(H$(N:N)))*Q
170     LET Q=Q*16
180   NEXT
190 END DEF
195 !
200 LET SGE=255
210 INPUT PROMPT "BAZISCIM?":A$
220 CALL DEC(A$)
230 LET SG=SPEEK(SGE,D-1)
240 IF SG=0 THEN STOP
250 LET HL=SPEEK(SGE,D-3)+256*
    SPEEK(SGE,D-2)
260 LET A$=H$(HL/256)&H$(MOD(HL,
    256))
270 PRINT H$(SG);"/";A$;": ";
280 CALL LEIRO
290 LET SGE=SG
300 GOTO 220
305 !
310 DEF LEIRO
320   FOR N=HL-3 TO HL+7
330     LET DT=SPEEK(SG,N)
340     PRINT H$(DT);" ";
```

```
350   NEXT
360   CALL NEV
370 END DEF
375 !
380 DEF NEV
390   FOR N=1 TO DT
400     PRINT CHR$(SPEEK(SG,HL+7+
        N));
410   NEXT
420   PRINT
430 END DEF
```

A program a hexadecimális alakban beírható báziscím alól kiolvassa a következő láncelem szegmensszámát (a lánc végét jelzi, ha ez 0) és tárolási címét (HL), azaz a láncelem pointerét. Kiírja a pointert, majd a periférialeíró elemeit és a periféria nevét. Ezután a pointer értékét tekinti báziscímnek és így keresi a következő elemet. A LEIRO rutin ilyen formában kifejezetten a periférialeíró kiolvasására alkalmas.

A futtatás eredménye:

```
START
BAZISCIM ? BFC0
FF/6661: 74 66 FF 00 20 00 EF 5F 04 00 08 KEYBOARD
FF/6674: 85 66 FF 00 08 00 36 4D 04 00 06 EDITOR
FF/6685: 93 66 FF 00 80 00 D3 6D 01 00 03 NET
FF/6693: A4 66 FF 00 00 00 A7 6D 01 00 06 SERIAL
FF/66A4: B3 66 FF 00 00 00 BB 67 01 00 04 TAPE
FF/66B3: C5 66 FF 00 00 00 69 67 01 00 07 PRINTER
FF/66C5: D6 66 FF FF 00 00 82 70 00 00 06 EDITOR
FF/66D6: E9 66 FF FF 20 00 55 70 00 00 08 KEYBOARD
FF/66E9: F9 66 FF 00 20 00 7C 6B 00 00 05 SOUND
FF/66F9: 61 66 00 00 20 01 81 57 00 00 05 VIDEO
```

A rutin a pointer által címzett adat (a leíró TYPE-eleme) előtti 3 értéket is kiírja. Így pl. az 1. sor elemei

— az első 3 adat (FF/6674) mutató az EDITOR perifériára
— 4.: TYPE (0, tehát ez a bejegyzés érvényes)
— 5.: IRQFLAG (20H, tehát a videomegszakításokat kezeli)
— 6.: FLAGS (0, nem VIDEO típus)
— 7/8.: TAB (04/5F8F, a billentyűzetkezelő belépési pontjainak táblázata)
— 9.: TAB-SEG
— 10.: UNIT-COUNT (0, nem lehet több ilyen nevű leíró)
— 11.: NAME (8, a név betűinek száma)

A TYPE-elemek áttekintésével látható, hogy a 0. szegmens beépített KEYBOARD és EDITOR kezelőit a bővítőben csatlakoztatott 4. szegmens felülírta, érvénytelenítette.

Ez a táblázat minden perifériaművelet alapja, tehát igen praktikusan felhasználható az esetleges átírásokra. Most csak egy durva beavatkozással szemléltetjük ezt a lehetőséget: érvénytelenítsük pl. a PRINTER-leírót, azaz írjunk 0-tól különböző

értéket a TYPE mezőre. A mi példánkban a PRINTER-leíró (és egyben a TYPE adat) címe FF/66B3, azaz 255/26291. Írjuk be

SPOKE 255,26291,1

Ezután minden nyomtatóművelet "Gerät nicht vorhanden" ("Device does not exist") hibajelzéssel áll le, amíg vissza nem állítjuk az érvényes típust.

SPOKE 255,26291,0

A hasonló láncba felfűzött csatornák végigkövethetők ugyanezzel a programmal, de a leírók kilistázásához alakítsuk át a LEIRO rutint, hogy csak a pointer által megadott cím előtti 16 bájtot írja ki:

```
310 DEF LEIRO
320   FOR N=HL-16 TO HL-1
330     LET DT=SPEEK(SG,N)
340     PRINT H$(DT);" ";
350   NEXT
360   PRINT
370 END DEF
```

A BASIC bejelentkezés után a következő (alapértelmezésűnek tekinthető) csatornák élnek a gépen:

```
START
BAZISCIM ? BFBD
FF/665C: 00 00 F9 66 FF 00 00 80 04 C0 00 67 00 DC 61 FF
FF/61DC: 00 C0 61 FF FF 00 00 0C 00 0C 00 00 01 D0 61 FF
FF/61D0: 00 00 61 66 FF 00 00 10 00 10 00 6A 00 C0 61 FF
FF/61C0: 00 00 E9 66 FF 00 00 D9 04 D9 04 68 00 E7 5C FF
FF/5CE7: 00 00 B3 66 FF 00 00 10 00 10 00 69 00 D7 5C FF
FF/5CD7: 00 00 74 66 FF 00 00 10 09 10 09 01 00 97 55 00
```

A csatornákról még lesz szó, most a 16-bájtos csatornaleírók három elemére hívjuk fel a figyelmet (a kapott lista értelmezéséhez):

— A pointer által megadott tárolási cím előtt (itt a sorok utolsó 3 adata) itt is a következő elem mutatóját találjuk.

— A csatornaszámot (a belső kezeléshez 1-gyel megnövelve) a (báziscím – 5) mezőn (hátulról az 5.) tárolja az EXOS

— A –12., –13., –14. elemek — már az alakjukból is látszik, hogy ezek is pointerek — a csatornához rendelt periféria leírójára mutatnak.

Hogy még itt is kísérletezzünk egy kicsit, nyissunk meg egy csatornát, és figyeljük meg, hogyan jelenik ez meg a láncban.

```
10 OPEN #5:"PRINTER:"

START
BAZISCIM?BFBD

FF/665C: 00 00 F9 66 FF 00 00 80 04 C0 00 67 00 DC 61 FF
FF/61DC: 00 C0 61 FF 00 00 00 0C 00 0C 00 00 01 D0 61 FF
FF/61D0: 00 00 61 66 FF 00 00 10 00 10 00 6A 00 C0 61 FF
FF/61C0: 00 00 E9 66 FF 00 00 D9 04 D9 04 68 00 E7 5C FF
FF/5CE7: 00 00 B3 66 FF 00 00 10 00 10 00 69 00 D7 5C FF
FF/5CD7: 00 00 74 66 FF 00 00 10 09 10 09 01 00 C7 53 FF
FF/53C7: 00 00 B3 66 FF 00 00 10 00 10 00 06 00 C7 53 00
```

Az 5. csatornát (6-os belső értékkel) a kialakult lánc aljára vette föl az EXOS (ezzel érdemes egy kicsit kísérletezni: mi történik kisebb vagy nagyobb pufferigényű csatorna nyitásakor, egy csatorna lezárásakor stb.).

A pointer alapján megfigyelhető, hogy a hozzárendelt periféria (FF/66B3) valóban a PRINTER.

Érdemes még egybevetni az új láncelem adatait a rendszermemóriáról készült legelső DUMP-pel. Ott láthattuk, hogy az igénybevett rendszerterület alsó határa az EXOS határ; (BF91) (13C7) volt, ill. 1. LAP címként értve: 53C7H. Látható, hogy az új láncelemet éppen ettől a címtől illesztette a rendszermemóriához az EXOS (és a határt nyilván lefelé mozgatta).

Befejezésül próbálkozzunk egy-két olyan beavatkozással, aminek még hasznát is vehetjük a továbbiakban. Vessünk egy pillantást a rendszer által használható memóriaterületek könyvtárára, a ROM- és RAM-szegmensek táblázatára. A fejezetben leírtak és a rendszerváltozók aktuális értéke alapján a ROM-tábla alsó címe (BF97/8H értéke) ABB9H, a RAM-tábla felső címe pedig ABD0H (ez rögzített érték).

```
START
KEZDöCIM ? ABB9
VEGCIM ? ABD0
SZEGMENS ? FF
 43961   ABB9   00 00 00 00 01 06 A7 FF
 43969   ABC1   04 00 00 00 05 00 00 00
 43977   ABC9   00 F9 FA FB FC FD FE FF
ok
```

Láthatjuk a bejegyzett ROM-ok szegmensszámát (1,4,5), előttük a számukra kiutalt RAM-terület mutatóját (itt csak a 4. szegmens kért RAM-ot; az FFH szegmens A706H címén kezdődő területet kapta), majd a RAM szegmensszámokat.

Hogy szemléltessük, milyen nagy biztonsággal kezeli a rendszer a rendelkezésére álló memóriákat, próbáljunk ki egy drasztikus beavatkozást. Próbáljuk meg egyszerűen kitörölni az egyik RAM-szegmenst a táblázatból.

Nézzük meg ismét a rendszerállapotot és a RAM-táblázatot:

```
START
KEZDöCIM ? BF9E
VEGCIM ? BF9F
SZEGMENS ? FF
 49054  BF9E  00 06 00 00 01 08 00 6A
ok
START
KEZDöCIM ? ABC9
VEGCIM ? ABD0
SZEGMENS ? FF
 43977  ABC9  00 F9 FA FB FC FD FE FF
ok
```

Írjunk 0-t pl. az FCH szegmens helyére:

POKE 43981,0

majd lássuk újra a két területet:

```
START
KEZDöCIM ? BF9E
VEGCIM ? BFA4
SZEGMENS ? FF
 49054  BF9E  00 05 01 00 01 08 00 6A
ok
START
KEZDöCIM ? ABC9
VEGCIM ? ABD0
SZEGMENS ? FF
 43977  ABC9  00 00 F9 FA FB FD FE FF
ok
```

Láthatjuk, hogy az EXOS túltette magát az összezavart táblázattal elékerülő problémán. A START után elénk kerülő memóriakép már ismét egy egészséges táblát mutat, csak éppen hiányzik a listáról a kiiktatott szegmens. A rendszerállapot azt mutatja, hogy a változásnak megfelelően csökkent (5-re) a szabad szegmensek száma is. A hiányzó szegmenst az EXOS mint "felhasználónak kiutaltat" könyvelte el.

Ha BASIC programok nyitásával (alulról) vagy grafikus lapok nyitásával (felülről) megterheljük a memóriakezelést, meggyőződhetünk róla, hogy az EXOS valóban nem számol a törölt szegmenssel. Igen egyszerű módon — szinte hályogkovács módjára — hozzájutottunk tehát egy olyan szegmenshez, amely általunk szabadon használható, a rendszer viszont nem bolygatja a továbbiakban (a készítők persze nem ezt a módszert képzelték el a szegmensigényléshez; erről is lesz szó a következő pontban).

Reméljük, hogy kellőképpen felkeltettük az Olvasó érdeklődését. Szinte kínálkozik a következő lépés: a ROM-tábla! Abba nem nyúlhatnánk bele? Dehogynem, csak egy kicsit óvatosabban. Azonnal végrehajtható, előkészületet nem igénylő próbálkozásként iktassuk be a (rendszertől az imént elorzott) FCH szegmenst a BASIC-BRD

(4.) ROM-szegmens helyére (lemondva persze ezzel a német verzió minden előnyéről). Előzőleg csak annyit tegyünk meg, hogy egy 'RET' utasítást helyezzünk az új ál-ROM belépési pontjára, hiszen valójában még semmilyen program nem fogadja a hozzá forduló EXOS-t:

```
SPOKE 252,10,201
```

Ezután már következhet a szegmensszámcsere:

```
POKE 43969,252
```

majd adjunk ki egy olyan utasítást, amely inicializálja a rendszert (pl. :BASIC).

Az újonnan induló rendszer már nem tartalmazza a 4. szegmens szolgáltatásait (erről meggyőződhetünk pl. a periférialánc kiolvasásával vagy egy :HELP utasítással), tartalmazhat viszont mindent, amit mi írunk a szegmensbe. Ha meggondoljuk, hogy ezzel a lépéssel egy teljes értékű ROM-hoz jutottunk, ami részt vehet a rendszer inicializálásában, saját perifériái lehetnek (amik megmaradnak akkor is, ha egy másik alkalmazói programba, pl. szövegszerkesztőbe lépünk) és ráadásul — az előéletnek köszönhetően — még egy védett, könnyen elérhető RAM-területtel is rendelkezhetünk, beláthatjuk, hogy a lehetőségeknek valóban csak a fantázia szab határt.

## 2.3.2 A funkcióhívások

Az EXOS rendkívül kényelmes lehetőséget ad a felhasználónak a rendszerszolgáltatások elérésére. Minden EXOS funkció — az inicializálástól a memóriagazdálkodáson át a perifériakezelők által megvalósított csatornaműveletekig — elérhető egy kétbájtos gépi kódú utasítással. Még ha hozzávesszük is ehhez, hogy a műveletek esetleges paramétereit (pl. kiíratandó szöveg kezdőcímét, igényelt memóriaterület nagyságát stb.) természetesen be kell állítanunk a hívás előtt a megfelelő regiszterekben, nyilvánvaló, hogy ennél tömörebb gépi kódú programírási lehetőséget semmilyen rendszer nem kínálhat. Minden alkalmazói program (pl. a BASIC értelmező is) rendszeresen él a funkcióhívások lehetőségével. Mi sem mellőzzük ezeket. Annyi mindent elvégez a rendszer a funkcióhívások végrehajtása során — végrehajtó rutinok és pufferterületek kikeresése, memóriaellenőrzés és rendezés, lapozás stb. —, hogy fölmerülhet a kérdés: érdemes-e egyáltalán a funkcióhívás megkerülésének, a végrehajtó rutinok közvetlen hívásának gondolatával foglalkozni. Nos, ha valóban mindazt meg kell csinálnunk, amit az EXOS elvégez, akkor nem érdemes! Lehetnek azonban olyan esetek (pl. egy kialakított képernyőlap és állandósult csatornaállapot), amikor a báziscímek egyszeri megkeresése után, a végrehajtó rutinok címeinek ismeretében túl körülményesnek (és főleg időigényesnek) tűnik a hosszadalmas ágak bejárása minden egyes műveletnél. Ez az érzésünk még valószínűleg fokozódik is, ha végigkövetjük azt az utat, ahogy az EXOS a

funkcióhívás kiadásától eljut a konkrét végrehajtásig. Mégis érdemes ezt az elemzést megtennünk, ha mélyebben meg akarunk ismerkedni gépünkkel. Akkor is ismernünk kell a működés egyes részleteit, ha saját perifériakezelőt akarunk a rendszerhez illeszteni, amit viszont majd az EXOS hív a funkcióhívások során.

Ha fel akarjuk használni a háttérrendszer szolgáltatásait, feltétlenül ismernünk kell az alkalmazás korlátait és lehetőségeit, feltételeit és várható eredményeit. Ezekről a kérdésekről magas színvonalon ír az EXOS műszaki leírása is, mi viszont a működés követésével és konkrét információkkal próbálunk meg újat mondani. A következőkben a programozástechnikai szempontból talán legérdekesebb bevezető és záró szakaszt a ROM rutinjainak elemzésével követjük, a kevésbé áttekinthető, hosszadalmasabb ágakat pedig funkcionálisan írjuk le.

Kezdjük az alapproblémával: kétbájtos utasításról beszéltünk, ugyanakkor sokféle összetett funkcióról. A gépi kódú programozással esetleg most ismerkedő Olvasó talán ellentmondást érez a két állítás között. A rendszer kifejlesztőinek valóban egy trükköt kellett alkalmazniuk a megoldáshoz. A továbbiakban "EXOS n"-ként emlegetett funkcióhívás valójában egy RST 30H Z80-as utasítás, amelyet a memóriában az argumentum: n követ (n értéke 0 és 11 vagy 16 és 34 közötti lehet). Az egybájtos utasítással hívható szubrutinnak (a 0030H címen kezdődő programnak) kell majd az őt hívó memóriaterületről az argumentumot beolvasni, és az értéknek megfelelően különböző funkciókat végrehajtani. Bonyolítja a helyzetet, hogy a hívások különböző csoportjait eltérő módon kell kezelni: egy részüket közvetlenül végrehajtja az EXOS, a csatornaműveleteket viszont a perifériakezelőknek kell átadni. Látható tehát, hogy a formai egyszerűségnek ára van, mind helyfoglalásban (ez a kevésbé érdekes, hiszen a gép készítői szokatlanul bőkezűen bántak a memóriával), mind futásidőben.

Hogy a továbbiakban tudjunk mire hivatkozni, megadjuk a funkcióhívások egyszerű felsorolását:

EXOS 0 EXOS inicializálás
EXOS 1 Csatorna megnyitása
EXOS 2 Csatorna létrehozása
EXOS 3 Csatorna lezárása
EXOS 4 Csatorna megszüntetése
EXOS 5 Karakter olvasása
EXOS 6 Blokk olvasása
EXOS 7 Karakter írása
EXOS 8 Blokk írása
EXOS 9 Csatornakészenlét olvasása
EXOS 10 Csatornaállapot beállítása és olvasása
EXOS 11 Speciális funkció
EXOS 16 EXOS változó olvasása, írása vagy átbillentése
EXOS 17 Csatorna átirányítása olvasáshoz
EXOS 18 Csatorna átirányítása íráshoz
EXOS 19 Alapértelmezés szerinti periférianév beállítása

EXOS 20 Rendszerállapot lekérdezése
EXOS 21 Periféria felvétele
EXOS 22 EXOS határ olvasása
EXOS 23 Felhasználói határ beállítása
EXOS 24 Szegmensigénylés
EXOS 25 Szegmens-felszabadítás
EXOS 26 Rendszerbővítők letapogatása
EXOS 27 Csatornapuffer kijelölése
EXOS 28 Hibakódok értelmezése
EXOS 29 Modul betöltése
EXOS 30 Áthelyezhető modul betöltése
EXOS 31 Idő beállítása
EXOS 32 Idő olvasása
EXOS 33 Dátum beállítása
EXOS 34 Dátum olvasása

Ezek után lássuk, mi történik, ha az alkalmazó programja egy EXOS hívást tartalmaz:

```
(hívó)     F7    RST 30
(hívó+1)   n     DEFB n          ; funkciószám
(hívó+2)   CC                    ; tetszőleges kód, az alkalmazói program folytatása
```

A processzor a 0030H címre ugrik, az őt követő (a processzor feltevése szerint a visszatérési) cím pedig a verembe kerül.

```
0030  F3          DI              ;megszakítás letiltás
0031  32 5A 00    LD   (005A),A   ;eredeti A tárolása
0034  B7          OR   A          ;NC státusz
0035  E3          EX   (SP),HL    ;HL: a RST 30H utasítást
0036  18 07       JR   003F       ;    követő cím (hívó+1)

003F  7E          LD   A,(HL)     ;funkcióhívás argumentum
0040  23          INC  HL         ;visszatérési cím (hívó+2)
0041  E3          EX   (SP),HL    ;verembe
0042  32 59 00    LD   (0059),A   ;argumentum tárolása
0045  DB B3       IN   A,(B3)     ;hívó 3. LAP-szegmense
0047  32 55 00    LD   (0055),A   ;munkaterületre
004A  3E 00       LD   A,00       ;EXOS szegmens
004C  D3 B3       OUT  (B3),A     ;a 3. LAP-ra
004E  C3 10 C4    JP   C410       ;elugrik a végrehajtásra
```

A vezérlés tehát a 0. ROM-szegmensre kerül. Itt
— tárolja a hívó 2. LAP szegmensét és HL regiszterét,
— a belapozott rendszerszegmensről beolvassa a jelzőbájtot.

Ennek kettős funkciója van. Ha 7. bitje 1, az EXOS verem rendezett. További bitjein csak akkor van 0-tól különböző érték, ha valamilyen perifériaművelet közben

történt a hívás. Ez lesz majd a tiltó feltétel bizonyos (perifériák számára tilos) funkcióknál (pl. csatornanyitás, -zárás).

```
C410  DB B2        IN   A,(B2)       ;pillanatnyi 2. LAP
C412  32 54 00     LD   (0054),A     ;tárolása
C415  3E FF        LD   A,FF         ;rendszerszegmens
C417  D3 B2        OUT  (B2),A       ;a 2. LAP-ra
C419  22 7C BF     LD   (BF7C),HL    ;eredeti HL letárolás
C41C  21 79 BF     LD   HL,BF79      ;EXOS flag-byte
C41F  7E           LD   A,(HL)       ;(alapérték 80H)
C420  CB 7F        BIT  7,A          ;veremrendezés kell?
C422  CB FE        SET  7,(HL)       ;"verem rendben"
C424  20 07        JR   NZ,C42D      ;átugorja, ha nem kell
C426  ED 73 7A BF  LD   (BF7A),SP    ;pillanatnyi SP tárolás
C42A  31 17 B2     LD   SP,B217      ;EXOS verem
C42D  F5           PUSH AF           ;flag, státusz -> verem
C42E  17           RLA               ; (A) <- CY
C42F  B7           OR   A            ;NC státusz
C430  1F           RRA               ;hívási státusz:
                                        CY=0, ha EXOS n
                                        CY=1, ha megszakítás
                                        NZ, ha periféria hív
```

A kialakuló státuszban az átvitelbit csak akkor lehet 1, ha a vezérlés máshonnan (konkrétan a 0038H címen induló megszakításkezelőről) került ide. Ez a bit téríti majd el a közös előkészítés után a megszakítás rutint.

Ezután a hívó rendszerállapotának további mentése és a saját visszatérési ág előkészítése következik:

```
C431  08           EX   AF,AF'       ;státusz háttérbe
C432  F5           PUSH AF           ;és verembe
C433  2A FE BF     LD   HL,(BFFE)    ;USR_P2,P3
C436  E5           PUSH HL           ;verembe
C437  2A 54 00     LD   HL,(0054)    ;a hívó P2,P3 SZG.-ei
C43A  22 FE BF     LD   (BFFE),HL    ;letárolás
C43D  21 FD BF     LD   HL,BFFD      ;USR_P1 címe
C440  7E           LD   A,(HL)       ;USR_P1
C441  F5           PUSH AF           ;verembe
C442  DB B1        IN   A,(B1)       ;pillanatnyi P1
C444  77           LD   (HL),A       ;letárolás
C445  2A 7C BF     LD   HL,(BF7C)    ;eredeti HL vissza
C448  E5           PUSH HL           ;verembe
C449  FD E5        PUSH IY           ;  "
C44B  DD E5        PUSH IX           ;  "
C44D  3E C9        LD   A,C9         ;"RET"
C44F  32 57 00     LD   (0057),A     ;a visszatérési ágba
C452  67           LD   H,A          ;
C453  08           EX   AF,AF'       ;hívási statusz
C454  38 5C        JR   C,C4B2       ;megszakítás kezelésre,
                                      ha CY=1
C456  3A 5A 00     LD   A,(005A)     ;eredeti A
C459  08           EX   AF,AF'       ;háttérbe
```

Ennyi előkészítés után az EXOS végre előveszi a tárolt funkciókódot és elkezdi a szétválogatást. Az egyetlen funkció, amit azonnal lekérdez és minden feltétel nélkül végre is hajt, a rendszerváltozók elérése (EXOS 16):

```
C45A  3A 59 00     LD   A,(0059)    ;EXOS argumentum
C45D  FE 10        CP   10          ;EXOS 16?
C45F  CA F9 C8     JP    Z,C8F9     ;végrehajtásra
```

Az összes további funkciót már csak akkor végzi el, ha a beállított CY bit sértetlenül túléli a saját visszatérési ágban való megfordulást:

```
C462  37           SCF              ;CY=1
C463  CD 56 00     CALL 0056        ;visszatérési ág
C466  CD 80 C5     CALL C580        ;szétválogatás
```

A C580H rutin ui. — mint látni fogjuk — mindenekelőtt a CY bitet ellenőrzi, és ha törlődött, ILLFN hibára fut. De vajon mi történhet ezzel a bittel a visszatérési ágban? Alapesetben itt csak egy EI:RET utasítássort talál, tehát nem változik. Ha azonban éppen megszakításkezelés folyik, a kezelő átírta ezt a címet OR A:RET-re. Megszakítások kezelése közben tehát az EXOS nem fogad el funkcióhívásokat (kivéve az EXOS változók írását, olvasását).

Egyelőre fogadjuk el, hogy megtörténik a funkcióhívások szétválogatása és végrehajtása. Most azt nézzük meg, hogyan tér vissza a rendszer a hívóhoz. Mindenekelőtt visszaállítja a hívó regisztereit és LAP-konfigurációját:

```
C469  F3           DI               ;megszakitás letiltás
C46A  26 B7        LD   H,B7        ;"OR A"
C46C  6F           LD   L,A         ;EXOS visszaadott A
C46D  22 54 00     LD   (0054),HL   ;visszatérési ágba
C470  DD E1        POP  IX          ;veremből
C472  FD E1        POP  IY          ;  "
C474  E1           POP  HL          ;  "
C475  22 7C BF     LD   (BF7C),HL   ;ideiglenes tárolás
C478  21 FD BF     LD   HL,BFFD     ;USR_P1 címe
C47B  7E           LD   A,(HL)      ;hívó P1
C47C  D3 B1        OUT  (B1),A      ;belapozza
C47E  F1           POP  AF          ;USR_P1 veremből
C47F  77           LD   (HL),A      ;vissza
C480  2A FE BF     LD   HL,(BFFE)   ;hívó P2,P3
C483  E3           EX   (SP),HL     ;USR_P2,P3 veremből
C484  22 FE BF     LD   (BFFE),HL   ;vissza
C487  E1           POP  HL          ;hívó P2,P3
C488  F1           POP  AF          ;hívási státusz
C489  08           EX   AF,AF'      ;
C48A  F1           POP  AF          ;flagbyte
C48B  20 22        JR   NZ,C4AF     ;átlépi a szoftver-
                                     megszakítás kezelést
C48D  32 79 BF     LD   (BF79),A    ;belépési értékre
```

Ezután pedig ellát egy — a felhasználó számára igen érdekes — feladatot: a szoftver megszakítás kezelését (pontosabban indítását). A rutin a BFF2H rendszerváltozó (FLAG-SOFT-IRQ) értékét teszteli. Ide pl. valamelyik megszakí-

táskezelő (billentyűzet, hálózat) írhatott be értéket. Ha a rendszer 0-tól különböző értéket talál itt, és a 003D/E címen is értelmes (0-tól különböző) érték (a felhasználói szoftver megszakításkezelő címe) van, úgy változtatja meg saját visszatérési ágát, hogy onnan majd egy JP nn-re kerüljön:

```
C490  3A F2 BF    LD   A,(BFF2)    ;FLAG_SOFT_IRQ
C493  B7          OR   A           ; értéke
C494  28 15       JR    Z,C4AB     ;átlépi, ha nincs IRQ
C496  32 C7 BF    LD   (BFC7),A    ;CODE_SOFT_IRQ beáll.
C499  AF          XOR  A           ;A=0
C49A  32 F2 BF    LD   (BFF2),A    ;törli a FLAG_...-et
C49D  E5          PUSH HL          ;
C49E  2A 3D 00    LD   HL,(003D)   :felhasználói megsza-
                                    kítási rutin címe
C4A1  7C          LD   A,H         :
C4A2  B5          OR   L           :HL=0?
C4A3  E1          POP  HL          ;
C4A4  28 05       JR    Z,C4AB     ;átlépi, ha igen
C4A6  3E 18       LD   A,18        ;"JR"
C4A8  32 57 00    LD   (0057),A    ;a visszatérési ágba
                                    ("JR 003C" beállítás)
```

Egy kis kitérőként próbáljuk ki ezt a lehetőséget:

```
 10 PRINT 1
 20 POKE 49138,33
 30 GOTO 10

START
 1
 1

*** STOP key pressed.
20 POKE 49138,33
```

A szoftvermegszakítás egy EXOS funkcióhívás után történik meg, tehát a rendszer a második kiírásnál vette észre a megszakítási kódot. A megszakítás BASIC-ben való kezeléséről később lesz szó.

Térjünk vissza a funkcióhíváshoz és kövessük a befejezés lépéseit:

```
C4AB  ED 7B 7A BF LD   SP,(BF7A)   ;SP belépési érték
C4AF  C3 26 B2    JP   B226        ;

B226  7C          LD   A,H         ;hívó P3
B227  D3 B3       OUT  (B3),A      ;visszalapozás
B229  7D          LD   A,L         ;hívó P2
B22A  2A 7C BF    LD   HL,(BF7C)   ;eredeti HL vissza
B22D  C3 51 00    JP   0051        ;visszatérési ágra
```

A visszatérési ág változhat az előzményektől függően. Egy átlagos funkcióhívásnál már csak a hívó 2. LAP szegmensének visszalapozása van hátra a teljes rendszerállapot visszaállításához. Itt kapja meg az A egy ügyes trükkel azt az értéket, amit a végrehajtó rutin adott vissza (a hibakódot vagy 0-t), és a C46DH című utasítással tárolt le a rendszer éppen a LD A,n utasítás argumentumaként.

```
0051  D3 B2        OUT   (B2),A      ;hívó P2 visszalapozás
0053  3E XX        LD    A,XX        ;EXOS visszaadott A
0055  B7           OR    A           ;státusz
0056  FB           EI                ;megszakítás eng.
0057  C9           RET               ;vissza a hívóhoz
```

Ha szoftvermegszakítás történt, a rutin legvége másként alakul:

```
0057  18 E3        JR    003C        ;
003C  C3 cc cc     JP    cccc        ;felhasználói megsza-
                                      kítási rutinra
```

Áttekintettük tehát az EXOS funkcióhívás főágát, működési logikáját. Nem láttuk viszont még a funkciók tényleges szétválogatását (kivéve az EXOS 16-ot). Térjünk tehát vissza ehhez a mozzanathoz.

A rutin először (az EXOS 0 kiemelése mellett) a hibás argumentumokat szűri ki (a törölt CY-bit jelentésével már foglalkoztunk).

```
C580  30 20        JR    NC,C5A2     ;(FE) hiba, ha NC
C582  B7           OR    A           ;EXOS argumentum (n)
C583  CA 53 C6     JP    Z,C653      ;EXOS 0, végrehajtásra
C586  FE 0C        CP    0C          ;n<12?
C588  DA 49 CB     JP    C,CB49      ;ha igen, elugrik
C58B  FE 23        CP    23          ;n>34?
C58D  30 10        JR    NC,C59F     ;(FF) hiba, ha igen
C58F  D6 11        SUB   11          ;A=n-17
C591  38 0C        JR    C,C59F      ;(FF) hiba, ha 11<n<17
```

Mint látható, az EXOS 1—11 hívásokat egy csoportban kezeli a rendszer. Az EXOS 0 végrehajtási címe C653H, a 17—34 közötti funkcióhívások belépési pontjait pedig egy ROM-táblázatból határozza meg a program:

```
EXOS 17-34 belépési címek megállapítása (0<=A<=17 alapján)

C593  21 0A C0     LD    HL,C00A     ;belépési cím táblázat
C596  87           ADD   A,A         ;A=2*A
C597  85           ADD   A,L         ;
C598  6F           LD    L,A         ;HL=HL+2*A
C599  7E           LD    A,(HL)      ;belépési cím LO
C59A  23           INC   HL          ;HI címe
C59B  66           LD    H,(HL)      ;belépési cím HI
C59C  6F           LD    L,A         ;HL: belépési cím
C59D  AF           XOR   A           ;A=0
C59E  E9           JP    (HL)        ;végrehajtó rutinra

C59F  3E FF        LD    A,FF        ;".IFUNC" hiba
C5A1  C9           RET               ;nem megengedett EXOS kód

C5A2  3E FE        LD    A,FE        ;".ILLFN" hiba
C5A4  C9           RET               ;nem hozzáférhető hívás
```

A táblázat:

```
C00A  82 C6 81 C6 E5 C7 08 C8
C012  98 C6 F2 C6 FC C6 0E C7
C01A  8E C7 18 C8 5E CD 3A C9
C022  5F C9 2C CB 3F CB 40 CB
C02A  3D CB 3E CB
```

Ezzel tehát megismertük a 0. és 16—34. funkcióhívások belépési pontjait. Foglaljuk össze, milyen feltételekkel indul ezeknek a rutinoknak a végrehajtása. A regiszterek általában eredeti értékükkel lépnek be. Kivétel ez alól a HL és az A, ezeket használta a rendszer a belépésig. Fontos viszont, hogy az AF' regiszter milyen értéket tartalmaz. A'-ben a hívás előtti akkumulátorértéknek kell lenni (pl. csatornaszám), F' pedig a C430H-n leírt hívási státusz.

Keményebb dió az 1.—11. funkciók csoportjának kezelése, a végrehajtókhoz való eljuttatás követése. Néhány dolgot érdemes összefoglalóan elmondani a perifériakezelőkről és a csatornákról, hogy jobban megérthessük a rendszer működését. Mi a közös az egy csoportban kezelt funkcióhívásokban? A felsorolásból láthatjuk, hogy ezek kivétel nélkül csatornákkal kapcsolatos műveleteket végeznek: nyitás, zárás, kivitel, beolvasás, vezérlés. Nyilvánvaló hogy a csatornaművelet önmagában értelmetlen dolog, a csatornáknak mindig valamilyen perifériához kell kapcsolódniuk. Ha pl. egy karaktert írunk a 104-es csatornába, ennek csak akkor van értelme, ha ehhez a számhoz egy perifériakezelő kötődik (jelen esetben a PRINTER), ami tudja, hogy mit kell kezdenie a kapott karakterrel.

Természetes tehát, hogy a csatornák élete a csatornaszám és perifériakezelő egymáshoz rendelésével (és az ezt nyilvántartó leíró elkészítésével) kezdődik és az összekapcsolás megszüntetésével fejeződik be. Ezt a funkciót még el tudná végezni belső rutinjaiban is az EXOS (pusztán az előző pontban megismert periférialánc adatai alapján), de a rendszer a csatornanyitási művelet során nemcsak ezt a (16-bájtos) leírót hozza létre. A csatornaműveletek egyszerűsítésére az EXOS közvetlenül a leíró alá illeszti a periféria által az adott esetben igényelt RAM-területet: a csatornapuffert (aminek felső határát meg is adja a perifériakezelőnek — mint látni fogjuk — minden híváskor az IX regiszterpárban). Azt pedig, hogy az adott csatorna létezéséhez, működtetéséhez mekkora pufferterületre van szükség, ismét csak a hozzárendelt perifériakezelő tudhatja. A PRINTER-kezelő pl. nem igényel külön RAM-ot a munkájához, a VIDEO rendszer viszont a legkülönbözőbb méretű puffereket kérheti, amit saját maga számol ki a kijelzési módot és a videolap méreteit meghatározó rendszerváltozók alapján.

Ezek alapján a csatornanyitási és -zárási műveleteket is a perifériakezelőkre kell bízni — legalábbis ami a szükséges pufferméret meghatározását illeti. A puffer kiutalását és nyilvántartásba vételét természetesen ismét a — minden memóriamenedzselési feladatot magára vállaló — rendszerprogramtól kell kérni (EXOS 27), majd a megkapott RAM-területtel ismét a kezelő rendelkezhet saját igényei szerint (pl. bejegyezheti a megnyitott videolap paramétereit).

A rendkívül felhasználócentrikus EXOS több lehetőséget is ad tetszőleges perifériakezelők létrehozására és rendszerbe illesztésére. Az egységes elérhetőség érdekében azonban ezeknek a kezelőknek — bizonyos pontjaikon — szabványosnak kell lenniük. Ilyen a periféria álleírója, amivel a 2.5 alfejezetben még foglalkozunk, és a kezelő végrehajtó rutinjainak belépési pontjait is egy egységes táblázatba kell rendezni. Ez a táblázat — kétbájtos címek egymásutánja — adott sorrendben tartalmazza a csatornaműveletek teendőit ellátó rutincímeket (és még néhány más alprogramot). Az EXOS 1, 2, ... stb. funkciók tényleges végrehajtóinak címe ezen táblázat 1., 2. stb. eleme. Az általában közvetlenül az álleíró után elhelyezett táblázat kezdőcímét a leíró TAB eleme adja meg (3. LAP címként).

Ahhoz, hogy az EXOS n funkcióhívás ebből a táblázatból megállapíthassa a végrehajtó kezelő alprogram címét, előbb a csatornaszám (a funkció hívásakor az A regiszterben megadott érték) alapján meg kell keresnie a hozzárendelt periférialeírót. Erre szolgál — többek között — a csatornaleírók lánca.

Mint az előbb már utaltunk rá, egy csatorna megnyitásakor a szükséges paramétereket az EXOS egy 16-bájtos csatornaleíróban könyvtárazza be. Ezeket a leírókat láncszerűen kapcsolja egymáshoz (erről az előbbi pontban már írtunk, és ki is olvastuk az aktuális leírólláncot). A BFBDH báziscím alatti pointer az első csatornaleíró fölé mutat. Ehhez az új báziscímhez képest a –1., –2. és –3. elem a következő csatornaleíró mutatója: a leírót tartalmazó szegmens számát (–1.) és címét (LO:–3., HI:–2.) adja meg. A –5. bájt a belső csatornaszám (a hívó által megadott csatornaszám + 1). Ezután a rendszer belső paramétereket tárol (pl. az igényelt területek nagyságát a puffer mozgatásához), majd egy újabb pointer következik: a csatornához rendelt periféria mutatója; szegmensszám a –12., leírócím a –13. és –14. bájton.

Hogy egy konkrét példát is lássunk, vizsgáljuk meg az előző fejezetben példaként kiíratott csatornalánc első leíróját. A BFBDH bázisszám alatti pointer: FF/665C. Az e cím előtti 16 bájt 00 00 F9 66 FF 00 00 80 04 C0 00 67 00 DC 61 FF.

A leíró említett elemei

665C – 01 FF
665C – 02 61
665C – 03 DC: a következő leíró báziscíme: FF/61DC
665C – 05 67: a leíró a 66H csatornához tartozik (ez a BASIC #102-es VIDEO csatornája)
665C – 0C FF
665C – 0D 66
665C – 0E F9: a periférialeíró pointere.

A kijelölt periféria (ugyancsak az előző fejezet példájaként kiírt periférialánc alapján) a VIDEO.

A pointer alatti két bájtot ugyancsak a rendszer használja a csatornaátirányítások bejegyzésére. Mint az EXOS 17 és 18 funkcióknál látni fogjuk, az adott csatorna

helyett használandó másodlagos csatorna számát a báziscím –15. (átirányítás olvasáshoz), ill. báziscím –16. (átirányítás íráshoz) címen jegyzi be a rendszer.

A csatornaterületeket a rendszermemória alján építi fel az EXOS. Ha a perifériakezelő az adott csatorna nyitásához nem igényel RAM-ot, a 16-bájtos leíró alsó címe lesz az új EXOS határ. Ha puffer is tartozik a csatornához, az itt kezdődik (az EXOS a (báziscím –16) értéket adja meg a kiutalt pufferterület báziscímeként az IX regiszterben). A következő csatornanyitáskor az új leíró már a lefoglalt csatornapuffer alatt kap helyet.

A csatornakezelés meggyorsításához még egy fogást alkalmaz az operációs rendszer. A rendszerváltozók területén, a BFA5H címtől kezdődően néhány bájtot fenntart azoknak a csatornáknak a paramétereihez, amelyekkel a legutóbb foglalkozott (és így feltételezhetően a leggyakrabban használt, leggyorsabban elérendő csatornák). Minden csatornához 4 bájtot rendel: a csatorna (belső) számát és a leíró pointerét. Három ilyen blokkot kezel ezen az aktuális csatornaterületen. Ha egy új csatornával kell foglalkoznia, egy blokkal (4 bájttal) feljebb tolja ezt a tartományt (ezzel a legrégebben használt csatornaparaméter blokkja kiesik) és felveszi az új csatorna paramétereit. Ez a 3 blokk éppen elég a számítógép működési idejének jelentős részét kitevő szerkesztési műveletekhez (EDITOR, KEYBOARD, VIDEO), így az alapcsatornákat nem kell a leíróláncban keresgélnie, mindig kéznél vannak (ezt az állapotot láthattuk az előző pont végén a rendszerváltozókról készített DUMP értelmezésénél).

Ezek után kövessük végig: milyen lépéseket tesz meg az EXOS, amíg a vezérlést átadja a funkcióhívást végrehajtó perifériakezelőnek.

— Verembe menti a háttérregisztereket.
— A csatorna nyitás-zárás funkcióknál ellenőrzi, hogy nem egy perifériakezelőből jött-e a hívás (hívási státusz!), ez ui. tiltott funkció.
— A hívott csatornaszámot (az akkumulátorban átadott értéket) megnöveli 1-gyel. Ez lesz a belső csatornaszám. Ha a hívás az FFH csatornára vonatkozott, helyettesíti azt a DEF—CHAN rendszerváltozó (BFC9H) aktuális értékével (kivéve, ha az is FFH volt, ez az eset ICHAN hibát eredményez).
— Ezután a funkciókezelés két részre válik szét.

1. A csatornanyitási, létrehozási funkció (EXOS 1, 2) esetén
   — a BFBDH báziscímről induló csatornaláncot végignézi, nincs-e már megnyitva a kérdéses csatorna. Ha megtalálja, CHANX hibajelzéssel tér vissza;
   — ezután a csatornához rendelt periféria nevével foglalkozik. A DE regiszterpárt erre a névre (pontosabban a név előtti hosszbájtra) állította a hívó, saját memóriakonfigurációjának bármely LAP-ján. Az EXOS a DE címet az 1. LAP-ra transzformálja és (a USER-P1, ..., P3 rendszerváltozók alapján) idelapozza a nevet tartalmazó szegmenst.
   A név karaktereit az első kettőspontig — nagybetűsítve és ellenőrizve — átviszi a BF54H címen kezdődő pufferbe. Ha a név 0 hosszúságú vagy nem tartalmaz kettőspontot (ilyen pl. az OPEN #1:"ADATFILE" típusú megnyitás),

az inicializálás során beállított alapértelmezésű eszköznevet (esetünkben: TAPE) írja a pufferbe (l. még: EXOS 19);

— a következő lépésben a pufferben tárolt periférianevet keresi meg a periférialáncban. Ha nem talál ilyen eszköznevet, ".NODEV" hibával tér vissza;

— munkaterületre tölti a funkciók elvégzéséhez szükséges paramétereket:

BF84: FLAG; 1, ha VIDEO periféria 0, egyébként
BF85: belső csatornaszám
BF86/7: a periférialeíró címe
BF88: a periférialeíró szegmense
BF89: jelzőbájt; FFH, amíg nincs puffer igényelve a csatornához;

— az előkészítés után meghívja az adott perifériakezelő megfelelő (1. vagy 2.) rutinját. A meghívás konkrét módját érdemes végigkövetni a 0. szegmens — minden csatornakezelő funkcióhívásban alkalmazott — rutinján. A rutin a veremméret és a perifériaérvényesség ellenőrzésével kezdődik.

```
C53E  01 19 53    LD   BC,5319     ;verem alsó határ
                                     (-ACECH)
C541  08          EX   AF,AF       ;funkciószám verembe
C542  7E          LD   A,(HL)      ;periférialeíró
C543  B7          OR   A           ;"TYPE"=0?
C544  3E FA       LD   A,FA        ;".NODEV"
C546  C0          RET  NZ          ;hiba, ha nem

C547  C5          PUSH BC          ;
C548  E3          EX   (SP),HL     ;HL=alsó határ
C549  39          ADD  HL,SP       ;aktuális veremhatár
C54A  3E FC       LD   A,FC        ;".STACK"
C54C  E1          POP  HL          ;
C54D  D0          RET  NC          ;hiba, ha SP a megenge-
                                    dett határ alatt van
```

A rendszer ezután beállítja a perifériakezelőnek átadandó IY és B' paramétereket, majd a belépési pont-táblázatból beolvassa az A-ban hozott funkciószámhoz tartozó címet.

```
C54E  44          LD   B,H         ;
C54F  4D          LD   C,L         ;BC: periférialeíró
C550  FD 21 00 40 LD   IY,4000     ;
C554  FD 09       ADD  IY,BC       ;IY: leíró címe P2-n
C556  DB B1       IN   A,(B1)      ;A: a leíró szegmense
C558  F5          PUSH AF          ;verembe
C559  23          INC  HL          ; (IRQ_FLAG)
C55A  23          INC  HL          ; (FLAGS)
C55B  23          INC  HL          ;TAB (alsó byte)
C55C  4E          LD   C,(HL)      ;C: táblacím LO
C55D  23          INC  HL          ;TAB (felső byte)
C55E  46          LD   B,(HL)      ;B: táblacím HI
C55F  23          INC  HL          ;SEG
```

```
C560  7E          LD   A,(HL)      ;a tábla szegmense
C561  D3 B1       OUT  (B1),A      ;az 1. LAP-ra
C563  08          EX   AF,AF'      ;A: funkciószám (n)
C564  87          ADD  A,A         ;A=2*n
C565  6F          LD   L,A         ;
C566  26 00       LD   H,00        ;HL=2*n
C568  09          ADD  HL,BC       ;HL: n. táblaelem c.
C569  7E          LD   A,(HL)      ;rutincím LO
C56A  23          INC  HL          ;
C56B  66          LD   H,(HL)      ;rutincím HI
C56C  6F          LD   L,A         ;HL: n. rutin címe
```

Az előkészítés után meghívja a perifériakezelő végrehajtó rutinját a B217H lapozórutin közvetítésével. Előtte átállítja a hívási státuszt meghatározó BF79H jelzőbájtot, hiszen ezután perifériaprogram fut. Ez a bájt egészen a visszatérésig (amikor újra alapértékre áll) tilt bizonyos funkcióhívásokat.

```
C56D  7B          LD   A,E         ;A: a csatornaleíró
C56E  D3 B1       OUT  (B1),A      ; szegmense 1. LAP-ra
C570  C1          POP  BC          ;B: perif.leíró szegm.
C571  7A          LD   A,D         ;A: csatornaszám
C572  08          EX   AF,AF'      ;A: végrehajtó rutin
                                      szegmense
C573  E5          PUSH HL          ;
C574  D9          EXX              ;
C575  21 79 BF    LD   HL,BF79     ;jelzőbyte címe
C578  34          INC  (HL)        ;"ettől kezdve perif."
C579  E3          EX   (SP),HL     ;HL: végrehajtó rutin
                                      címe
C57A  CD 17 B2    CALL B217        ;belapozás a 3. LAP-ra
                                      és végrehajtás
C57D  E1          POP  HL          ;HL=BF79
C57E  35          DEC  (HL)        ;jelzőbyte alapérték
C57F  C9          RET              ;
```

A perifériakezelő végrehajtó rutinjából való visszatérés után — az 1. és 2. funkcióhívások szétválogató ágának befejezéseként még ellenőrzi a BF89H jelzőbájton (amit maga állított FFH-re a kezelő hívása előtt), hogy nem feledkezett-e meg a periféria csatornanyitásért felelős rutinja a pufferigénylésről (EXOS 27 hívás). Ez akkor is kötelező, ha a periféria nem igényel puffert, mert a csatornaleírót is ez a funkcióhívás hozza létre, majd — jelezve, hogy feladatát ellátta — 0-ra állítja a fenti jelzőbájtot. Erre tehát nekünk is ügyelnünk kell egy esetleges új periféria létrehozásakor.

2. A fennmaradó csatornaműveleteket (EXOS 3-11) másképp készíti elő rendszer.
   — Mindenekelőtt megkeresi a szóban forgó csatorna leíróját. Először az aktuális csatornák között keres (BFA5H-tól). Ha itt nincs bejegyezve az adott csatorna, végigvizsgálja a BFBDH báziscímről induló csatornaláncot is (ha itt sem találja, ICHAN hibával visszatér). Ha megtalálja a leírót, azt bevezeti az aktuális területre is, mint legfrissebb csatornát.

— A csatorna leírójában ellenőrzi, hogy a funkcióhívásnak megfelelő irányban (írás vagy olvasás) nincs-e átirányítva az adott csatorna egy másik csatornaszámra. Ha igen, újrakezdi a keresést erre a másodlagos csatornára.

A csatornakeresés mechanizmusának mi is hasznát vehetjük. Nézzük meg, hogyan csinálja ezt a 0. ROM CD3AH rutinja:

```
CD3A  0E 05       LD   C,05        ;csatornaszám helye
CD3C  21 BD 7F    LD   HL,7FBD     ;csatornalánc báziscím
                                     az 1. LAP-on
CD3F  08          EX   AF,AF'      ;keresett csatornaszám
                                     háttérbe
CD40  3E FF       LD   A,FF        ;rendszerszegmens
CD42  D3 B1       OUT  (B1),A      ;az 1. LAP-ra
CD44  08          EX   AF,AF'      ;
CD45  08          EX   AF,AF'      ;
CD46  22 81 BF    LD   (BF81),HL   ;b.cím munkaterületre
CD49  32 83 BF    LD   (BF83),A    ;szegmensszám   "
CD4C  CD 47 C6    CALL C647        ;pointer kiolvasása
CD4F  06 00       LD   B,00        ;
CD51  C8          RET  Z           ;vissza, ha lánc vége
CD52  0C          INC  C           ;        (Z)
CD53  0D          DEC  C           ;
CD54  C8          RET  Z           ;vissza, ha C=0 volt
                                     (Z)
CD55  08          EX   AF,AF'      ;A: keresett csat.szám
CD56  ED 42       SBC  HL,BC       ;csatornaszám címe
CD58  BE          CP   (HL)        ;a leiróban=A?
CD59  09          ADD  HL,BC       ;
CD5A  20 E9       JR   NZ,CD45     ;tovább keres, ha nem

Ha megtalálta a keresett csatornaszámot:

CD5C  08          EX   AF,AF'      ;NZ státusz
CD5D  C9          RET              ;
```

Ha a rutinból (Z) státusszal tér vissza a program, nem találta meg a keresett csatornaszámot. Ha viszont megtalálta, HL-ben a leíró báziscíme, A-ban a szegmense található. Érdemes megjegyezni, hogy ezt a pointert a BF81/2H, ill. BF83H munkaterületre is letárolta a keresőrutin.

Az előkészítés befejezéseként az EXOS beállítja a végrehajtó rutinnak átadandó paramétereket. Az IX regiszterbe a csatornaleíró báziscíme –16 értéket tölt. Így IX a leíró legalsó bájtjára mutat, alatta kezdődik a csatorna RAM-területe. A perifériaművelet kezelése során tehát IX-ben van az adott csatornához kiutalt puffer báziscíme. Ezt tehát feltétlenül be kell állítanunk egy esetleges közvetlen hívás előtt is.

Ezután a leíróból beolvassa a csatornához rendelt periféria báziscímét, és a már megismert módon (a C53EH rutin felhasználásával) belapozza és meghívja a belépési címtáblázat funkcióhívásnak megfelelő sorszámú rutinját (tehát pl. EXOS 7 esetén a 7. rutint).

Ezzel gyakorlatilag be is fejezte az EXOS a csatorna I/O műveletek kezelését. A csatornát lezáró (EXOS 3,4) funkcióhívásoknál még elvégzi a memóriarendezési

feladatokat, majd a háttérregiszterek visszaállítása után innen is rátér a rendszerhívások — már megismert — visszatérési ágára.

Foglaljuk végül össze, milyen feltételeket biztosít, milyen paramétereket ad át az EXOS a funkcióhívások végrehajtóinak, a perifériakezelőknek:

— A memóriakonfiguráció:
  0. LAP: a nulláslap szegmense (esetünkben F8H)
  1. LAP: a csatornaleíró szegmense
  2. LAP: az FFH rendszerszegmens
  3. LAP: a perifériakezelő szegmense

— A regiszterek:
  A: a csatornaszám
  BC: eredeti érték a hívótól
  DE: eredeti érték a hívótól
  HL: a használt (a meghívott) rutin címe
  B': a futó perifériakezelő leírójának szegmensszáma
  IY: a periférialeíró báziscíme (előtte lehet a perifériának az inicializálás folyamán kiutalt RAM)
  IX: a csatorna RAM báziscíme; a használható terület ez alatt indul az (IX—1) címen

A kezelőtől az EXOS nem várja el egyik regiszter megőrzését sem, de maga megőrzi és a hívónak továbbadja a kezelőtől visszakapott A, BC és DE regisztereket.

Végül a funkcióhívások konkrét áttekintése előtt még egy közös vonatkozású megjegyzés. A leírás során többször említettük, hogy adott esetben az EXOS valamilyen hibával tér vissza. Valójában maga az EXOS nem ad semmilyen hibaüzenetet, kizárólag az A visszaadott értékében (ill. a státuszban) jelzi, ha valami rendellenesség történt. A hívó feladata, hogy a visszatérés után ellenőrizze a státuszt és NZ esetén megtegye a szükséges lépéseket. A hibakódokat a 4. Függelékben soroljuk fel.

Ezek után vegyük sorra az EXOS funkcióhívásokat, konkrét adatokkal, példákkal segítve az alkalmazásukat. A példákat általában BASIC-ben építjük fel, hogy segédeszköz nélkül, könnyen kipróbálhatók legyenek. A kimenő adatok között nem említjük külön a minden hívásnál visszaadott (esetleges hibára utaló) státuszt.

*EXOS 0: EXOS inicializálás*

Belépési pont: C653H (az EXOS rutinok belépési címei a 0-s szegmensben vannak, ezt külön nem adjuk meg).

Bemenő adat: a C regiszter, amelynek bitjei az inicializálás különböző lépéseit váltják ki és a BF78H rendszerváltozó, amely ugyancsak jelzőbájtként szolgál.

Kimenő adat: nincs.

Funkció: A jelzők értékétől függetlenül törli a melegindítási címet (RST-ADDR; BFF8/9H), a felhasználói megszakítási rutin címét (USER-ISR; valamint a felhasználói szoftver megszakítási rutin címét (SOFT-ISR; 003D/EH).

A további végrehajtás a C regiszter bitjeitől függően elágazik:

— ha BIT 7,C = 1: teljes hidegindítást végez (egyenértékű a RESET gomb kétszeri megnyomásával).

A továbbiakat tekintsük át a legkevésbé drasztikus variációtól:

— ha minden bit 0: visszatér a hívóhoz (ilyenkor tehát csak a fenti belépési címeket törli).

A megszakítás a hívóhoz való visszatéréskor (most és a további alternatíváknál is) tiltott.

— Ha BIT 6,C = 1: alaphelyzetbe állítja a szegmenskiosztást (felszabadítja a felhasználónak kiutalt szegmenseket).
BASIC-ben: nem törli a változókat, a megnyitott csatornákat, de törli a 0-tól különböző számon megnyitott programokat.

— Ha bármelyik további bit 1:
hidegindítás utáni szegmenskiosztást hoz létre.
— Nullát tölt a

PORTB5
FLAG—SOFT—IRQ
REM1
REM2
ST—FLAG
NET—IRQ
WAIT—SND
STOP—IRQ
TIMER
DEF—CHAN
DEF—TYPE

EXOS változókba és a
BF89H (csatornapuffer-igénylés)
BFA5H (aktuális csatorna)
BFBCH (csatornalánc mutató)
címekre.

— FFH alapértéket tölt a

NET—IRQ
MATCH—NET
CHAN—NET

EXOS változókba

— (csak ha a BF78H jelzőbájt 0 volt): teljesíti a ROM-táblában levő szegmensek RAM-igényét (és 1-re állítja a jelzőbájtot),
— (csak ha BIT 5,C = 1 volt): újraépíti a ROM-szegmensek periféria láncát,
— törli a csatornaterületet,
— újra TAPE-re állítja az alapértelmezésű eszköz nevét,
— beállítja a video- és a 1 Hz-es megszakítások engedélyezését,
— végighívja a perifériák, majd a RAM és ROM rendszerbővítők Inicializálás rutinjait.

Felhasználás: minden olyan esetben, amikor egy újonnan induló rendszer tiszta lappal akar kezdeni. Így biztosíthatja az egyértelmű rendszermemóriát az alkalmazói program induláskor. Mi is használhatjuk ezt a funkcióhívást, ha olyan beavatkozást végeztünk a rendszer területén, hogy a változás végigvitelét az EXOS-ra kell bíznunk.

Erre egy példát is adunk a következőkben.

A funkció közvetlen hívása nehezen megoldható (a sajátos visszatérési út miatt), de nem is igazán indokolt: a rendszerinicializálás nem az a terület, ahol a megtakarítható mikro- (vagy akár milli-) szekundumoknak jelentősége lenne. Szokásos úton való hívása viszont egyszerű:

```
LD C,XX
EXOS 00
RET
```

BASIC-ből betöltött gépi kódú rutinnal:

```
100 ALLOCATE 100
110 CODE M=HEX$("0E,XX,F7,00,C9")
120 CALL USR(M,0)
```

Természetesen "XX" helyére az inicializálási lépéseket kiváltó biteket tartalmazó hexadecimális értéket kell írni. A kísérletezve megismerés jegyében javasoljuk az Olvasónak, hogy próbálja ki a hívást különféle beavatkozások után a 00, 10, 20, 40, 60, 80 értékekkel (ez utóbbival csak akkor, ha nincs a gépben féltett program).

Hogy az első komolyabb beavatkozásunkat megtegyük (aminek még sok hasznát vehetjük a továbbiakban), próbáljuk meg saját igényeinknek megfelelően átalakítani a memóriatérképet. Építsünk be a ROM-szegmensek táblázatába egy új elemet valamelyik RAM-szegmens számával. Egy ilyen könyvtárazott ál-ROM előnyeiről már írtunk a 2.3.1 pontban, de ott csak egy igen drasztikus lépéssel, 4. szegmens feláldozásával, értünk el ilyen hatást. Ha valaki nem akar lemondani a 4. szegmens szolgáltatásairól (pl. mert a billentyűzet adott kiosztását szokta meg), vagy egyszerűen nincs is a gépén ilyen helyettesíthető szegmens (egynyelvű verzió), fölvetheti a hidegindítás folyamatának követése alapján: nem lehetne-e a ROM-táblát annyival lejjebb tolni, hogy a tetejére beírhassuk a felvenni kívánt szegmens számát, az így kialakult új táblahatártól pedig újra felépíttetni a rendszermemória ez alatti részeit (ROM-szegmensek RAM-területe, periférialánc, csatornalánc).

Nos, mindennek semmi akadálya, csak a konkrét végrehajtáshoz figyelmesen szemügyre kell venni a beavatkozás előtti rendszerállapotot (kinek-kinek a saját gépén). Az érintett változóterület listája (a 2.3.1 pontban ismertetett DUMP programmal):

```
START
KEZDŐCIM ? BF90
VEGCIM ? BFBF
SZEGMENS ? FF
 49040   BF90   00 C7 13 5C A6 06 A7 B9
 49048   BF98   AB 00 D0 AB C9 AB 00 06
 49056   BFA0   00 00 01 08 00 6A D0 61
 49064   BFA8   FF 01 D7 5C FF 67 5C 66
 49072   BFB0   FF 00 00 00 61 19 00 00
 49080   BFB8   00 00 5C 66 FF 61 66 FF
ok
```

A RAM- és ROM-táblák területe pedig:

```
START
KEZDŐCIM ? ABB0
VEGCIM ? ABD0
SZEGMENS ? FF
 43952   ABB0   E5 2D E6 00 00 00 00 7B
 43960   ABB8   7E 00 00 00 00 01 06 A7
 43968   ABC0   FF 04 00 00 00 05 00 00
 43976   ABC8   00 00 F9 FA FB FC FD FE
 43984   ABD0   FF DD B1 00 00 00 00 00
ok
```

Esetünkben az ABB9H—ABC9H közötti bájtokat kellene eltolni lefelé (helyet csinálva egy új 4-bájtos táblaelemnek). A konkrét címek ismeretében ezt egyszerűen megoldhatjuk, de nem különösebben bonyolult az általános (a mozgatandó terület határait és méretét a rendszerváltozók alapján megállapító) megoldás sem.

Milyen változókat kell figyelembe venni:

BF97/8H: a mozgatandó terület alsó címe (ROM-tábla alja)

BF9C/DH: a mozgatandó terület felső címe + 1 (RAM-tábla alja).

A két cím különbsége adja a mozgatandó terület nagyságát. Ezek alapján felírhatjuk a program mozgató részét.

A blokk eltolása után már csak két feladatunk maradt:

— a bejegyezni kívánt szegmens számát (példánkban FCH) beírjuk a volt legfelső elem helyére,

— a ROM-tábla új alsó címére állítjuk a megfelelő rendszerváltozókat, majd meghívjuk az EXOS 0 funkciót, hogy végigvezesse a memóriában a változásokat. A blokkmozgatással belecsúsztunk a 4. szegmensnek kiutalt RAM-területbe, ezért úgy állítjuk be a funkcióhívás jelzőit, hogy a ROM-ok RAM-igényét is újra elégítse ki.

Ezek után a feladatokat ellátó assembly program:

```
ORG 3000H          ;rutin tárolási címe
LD DE,(0BF97H)     ;ROM-tábla alja
LD HL,(0BF9CH)     ;ROM-tábla teteje+1
OR A               ;NC státusz
SBC HL,DE          ;ROM-tábla hossza
PUSH HL            ;
POP BC             ;BC=mozgatandó hossz
EX DE,HL           ;
PUSH HL            ;
LD DE,4            ;
SBC HL,DE          ;ROM-tábla alja - 4
EX DE,HL           ;DE: új alsó cím
POP HL             ;HL: régi alsó cím
PUSH DE            ;
LDIR               ;blokkmozgatás
LD DE,5            ;
SBC HL,DE          ;legfelső szegm. címe
LD (HL),0FCH       ;ál-ROM bejegyzése
POP HL             ;HL: tábla alja
LD (0BF97H),HL     ;bejegyzés a rendszer-
LD (0BF99H),HL     ;    változókba
XOR A              ;A=0
LD (0BF78H),A      ;a jelzőbyte-ba
LD C,60H           ;RESET-jelzők
EXOS 00H           ;EXOS inicializálás
RET
```

ill. ugyanez BASIC betöltővel:

```
100 POKE 540,0
110 POKE 541,48
120 CODE M=HEX$("ED,5B,97,BF,2A,9C,
    BF,B7,ED,52,E5,C1,EB,E5,11,04,
    00,ED,52,EB,E1,D5,ED,B0,11,04,
    00,ED,52,36,FC,E1,22,97,BF,22,
    95,BF,AF,32,78,BF,0E,60,F7,00,
    C9")
130 SPOKE 252,10,201
140 CALL USR(M,0)
```

A rutin tárolására választott 3000H cím a BASIC terület közepén van ugyan, de — éppen ezért — kis programok esetén különösebb védelem nélkül is megfelel a rutinok ideiglenes tárolására.

Az EXOS 0 kiváltotta inicializálási folyamatban már az újonnan bejegyzett ROM-szegmens is részt vesz, ezért ne feledkezzünk meg a rutin végrehajtása előtt a szegmens belépési pontját (C00AH a 3. LAP-on) megfelelően kialakítani. Ennek legegyszerűbb módját láthatjuk a BASIC betöltőprogram 130. sorában, ami — egyelőre — egy RET utasításkódot ír a szegmens 10. bájtjára.

Az EXOS 0 hívás törli a melegindítási címet, szoftvermegszakítási címet. Hogy egyértelművé tegyük a BASIC alaphelyzetet, a rutin futtatása után adjunk ki egy BASIC utasítást. Ez ugyan törli a betöltőprogramot is, de az már ellátta a feladatát.

Ellenőrizzük a rendszermemória területeit, változóit, hogy rendben megtörtént-e a változás a teljes rendszerben.

A rendszerváltozók:

```
START
KEZDőCIM ? BF90
VEGCIM ? BFBF
SZEGMENS ? FF
 49040  BF90  00 C3 13 58 A6 02 A7 B5
 49048  BF98  AB 00 D0 AB C9 AB 00 06
 49056  BFA0  00 00 01 08 00 6A CC 61
 49064  BFA8  FF 01 D3 5C FF 67 58 66
 49072  BFB0  FF 00 00 00 65 19 00 00
 49080  BFB8  00 00 58 66 FF 5D 66 FF
ok
```

Örömmel láthatjuk, hogy a ROM-tábla alatti területekhez tartozó összes cím, mutató értéke 4-gyel csökkent. Lejjebb csúszott a periférialánc, a csatornaterület és — természetesen — az EXOS határ is. Maga a megváltoztatott ROM-tábla:

```
START
KEZDőCIM ? ABB0
VEGCIM ? ABD0
SZEGMENS ? FF
 43952  ABB0  00-00 00 7B 7E 00 00 00
 43960  ABB8  00 01 02 A7 FF 04 00 00
 43968  ABC0  00 05 00 00 00 FC 00 00
 43976  ABC8  00 00 F9 FA FB FC FD FE
 43984  ABD0  FF DD B1 00 00 00 00 00
ok
```

Ez tartalmazza (az 5. szegmens előző helyén) a bejegyezni kívánt ál-ROM-ot (FCH), a többi blokk pedig lefelé tolódott. A 4. szegmens blokkjában jól látható, hogy a rendszer újra kiutalta számára az igényelt RAM-területet (természetesen 4-gyel kisebb kezdőcímmel).

Végül ellenőrizzük, hogy a periférialánc hogyan épült újra. A 2.3.1 pontban leírt LANC program futtatásával (az ott kapott listával egybevetve) láthatjuk, hogy a ROM-tábla valamennyi perifériája beláncolódott a hidegindítással analóg módon, csak kisebb báziscímekkel:

```
START
BAZISCIM? BFC0
FF/665D: 70 66 FF 00 20 00 8F 5F 04 00 08 KEYBOARD
FF/6670: 81 66 FF 00 08 00 36 4D 04 00 06 EDITOR
FF/6681: 8F 66 FF 00 80 00 D3 6D 01 00 03 NET
FF/668F: A0 66 FF 00 00 00 A7 6D 01 00 06 SERIAL
FF/66A0: AF 66 FF 00 00 00 BB 67 01 00 04 TAPE
FF/66AF: C1 66 FF 00 00 00 69 67 01 00 07 PRINTER
FF/66C1: D2 66 FF FF 00 00 82 70 00 00 06 EDITOR
FF/66D2: E5 66 FF FF 20 00 55 70 00 00 08 KEYBOARD
FF/66E5: F5 66 FF 00 20 00 7C 6B 00 00 05 SOUND
FF/66F5: 5D 66 00 00 20 01 81 57 00 00 05 VIDEO
```

A könyvtárazott ROM-szegmens most már a rendszerhez tartozik kikapcsolásig. Ha komoly feladatokat akarunk vele elláttatni, akkor persze a bejegyzett szegmenst

védeni kell, meg kell akadályozni, hogy a rendszer felhasználhassa (pl. az EXOS 24 funkcióhívás segítségével).

Egyszerűbb kísérletezéshez azonban így is felhasználhatjuk a szegmenst. Érdekességként — a géppel való ismerkedést segítendő — írjunk pl. a belépési pontra egy egyszerű rutint, amely a hívási kód értékét (C regiszter) minden hozzáforduláskor kiírja az állapotsorba (az egyszerűség kedvéért egy nem törlődő, szabadon használható karakterhelyre).

A feladat megoldása igen egyszerű:

```
LD   HL,0EBDH      ;állapotsor 6. byte
LD   A,30H         ;ASCII "0"
ADD  A,C           ;+ hívási kód
LD   (HL),A        ;karakter az állapotsorba
RET                ;
```

BASIC betöltővel:

```
100 ALLOCATE 100
110 CODE M=HEX$("C9,21,BD,BE,3E,30,
    81,77,C9")
120 FOR N=0 TO 10
130   SPOKE 252,10+N,PEEK(M+N)
140 NEXT
```

A 252. szegmens belépési pontjára áttöltött rutin továbbra is RET-tel kezdődik. A kijelzési funkció így egy

SPOKE 252,10,0

utasítással aktivizálható és

SPOKE 252,10,201

utasítással kapcsolható ki. Érdemes kipróbálni a különböző — bővítőket is hívó — utasításokat (:HELP; :SZÖVEG; hibát okozó EXOS hívás, pl. nem létező csatornára írás; a RESET gomb megnyomása stb.). Az indítási kódokról még lesz szó a 2.6 alfejezetben.

Még figyelemfelkeltőbb a ROM-ok lekérdezése, ha valamilyen hanghatással is jár. Erre igen egyszerűen felhasználható pl. a BASIC PING utasítás végrehajtó rutinja (5. szegmens, D3BFH cím), csak azt a problémát kell megoldani, hogy a 3. LAP-on hívott bővítő hogyan hívja egy másik szegmens rutinját ugyanezen a lapon. Az egyik lehetséges megoldás, hogy a bővítő programja ellapozza saját magát (az 1. LAP-ra) és ott folytatja a futást:

```
        LD   A,0FCH          ;saját szegmenszám
        OUT  (0B1H),A        ;1. LAP-ra lapozza magát
        JP   (CIM-8000H)     ;a folytatásra ugrik
CIM     PUSH DE              ;
        PUSH BC              ;
        LD   A,5             ;BASIC szegmens
        OUT  (0B3H),A        ;a 3. LAP-ra
        CALL 0D3BFH          ;"PING" rutin
        POP  BC              ;
        POP  DE              ;
        RET                  ;
```

Ennek megfelelően kiegészítve a BASIC betöltőt:

```
100 ALLOCATE 100
110 CODE M=HEX$("C9.21,BD,BE,3E,30,
    31,77,3E,FC,D3.B1,C3.19,40,D5,
    C5,3E,05,D3,B3,CD,BF.D3,C1,D1,
    C9")
120 FOR N=0 TO 26
130   SPOKE 252,10+N,PEEK(M+N)
140 NEXT
```

Jó kísérletezést!

*EXOS 1-11*

A csatornakezelő utasítások belépési pontjai csak közvetve adhatóak meg, hiszen ezeket — mint láttuk — az EXOS előkészítése után a csatornához rendelt perifériakezelők hajtják végre. A perifériakezelők belépési pontjait az 5. Függelék tartalmazza. A közvetlen hívás lehetőségével itt csak az EXOS 7 esetén foglalkozunk, ahol ennek a legnagyobb jelentősége lehet.

*EXOS 1: csatorna megnyitása*

*EXOS 2: csatorna létrehozása*

A két hívásról együtt beszélhetünk, hiszen (a TAPE kivételével) a perifériakezelők is azonos módon kezelik ezeket.
Bemenő adatok:

A: csatornaszám (0—254)
DE: a csatornához rendelt periféria nevére mutat (pontosabban a név előtti hosszbájtra)
EXOS: változók a végrehajtó periféria igénye szerint

A periférianév a hívó lapkonfigurációjában bármelyik LAP-on lehet, azt az EXOS megtalálja az előkészítés során. A név végén kettőspontnak kell állnia, különben a rendszer file-névnek tekinti és periférianévként a TAPE-t állítja be (alapértelmezésű periférianév).

Használatára egy egyszerű példát mutat a következő BASIC rutin:

```
100 ALLOCATE 100
110 CODE NEV=CHR$(8)&"PRINTER:"
120 CODE M=HEX$("11")&WORD$(NEV)&
    HEX$("3E,01,F7,01,C9")
130 CALL USR(M;0)
140 PRINT #1:"SZöVEG"
```

Ez a LD DE,NEV
LD A,1
EXOS 1
RET
gépi programot tölti a memóriába és hívja meg. A rutin futtatása után ugyanúgy írhatunk a nyomtatóra az #1-es csatornán, mint ha OPEN #1:"PRINTER:" BASIC utasítással nyitottuk volna meg. A név beírása helyett persze kihasználhatjuk azt is, hogy a ROM-szegmensekben is megtalálhatók a nyitáshoz szükséges periférianevek (mint ezt már említettük a 2.3.1 pontban):

```
100 ALLOCATE 100
120 CODE M=HEX$("11,43,FE,3E,01,F7,
    01,C9")
130 CALL USR(M,0)
140 PRINT #1:"SZöVEG"
```

Ebben az esetben az 1. szegmens FE43H címén tárolt névre állítottuk DE-t, felhasználva azt a tényt, hogy a USR függvény kiértékelésekor az 1. szegmens van belapozva a 3. LAP-on.

Ilyen névtárolási pontok:

1. szegmens:

FE34: VIDEO:
FE3B: EDITOR:
FE43: PRINTER:
FE4C: KEYBOARD:

5. szegmens:

F9E5: EDITOR:
FA98: KEYBOARD:
FADF: PRINTER:
FB16: SOUND:
FB50: VIDEO:

A példaként választott PRINTER perifériakezelőhöz viszonylag egyszerűen nyitható csatorna. Itt ui. nem kell előkészíteni a megnyitást az EXOS változók területén. A VIDEO kezelő viszont ezekből a változókból szerez tudomást a megnyitandó csatorna (VIDEO lap) méretéről, kijelzési módjáról. A MODE—VID, COLR—VID, X—SIZ—VID és Y—SIZE—VID EXOS-változók beállítására és a csatorna nyitására

már láttunk példát a 2.3.1 pontban, a bejelentkezési üzenet előkészítésénél. Most gyorsítsuk egy kicsit ezt az eljárást a változók közvetlen beírásával! Nyissunk meg egy 32·16 karakternyi méretű grafikus lapot 16 színű módban az EXOS 1 funkcióhívás segítségével (pl. az 5. csatornára), majd — egyelőre BASIC-ben — jelezzük ki az új lapot és rajzoljunk is rá, hogy ellenőrizzük a beállított paramétereket.

```
100 ALLOCATE 100
110 CODE NEV=CHR$(6)&"VIDEO:"
120 CODE M=HEX$("11")&WORD$(NEV)&
    HEX$("21,01,02,22,DB.BF,21,20,
    10.22,DD,BF,3E,05,F7,01,C9")
130 CALL USR(M,0)
140 SET BORDER 10
150 DISPLAY #5:AT 5 FROM 1 TO 10
160 PLOT #5:500,400,ELLIPSE 150.150,
```

A változás a PRINTER-csatorna nyitásához képest:

```
...
LD  HL,0201H     ;
LD  (0BFDBH),HL  ;MODE_VID=1   (grafikus)
                  COLR_VID=2   (16 szín)
LD  HL,1020H     ;
LD  (0BFDDH),HL  ;X_SIZ_VID=32
                  Y_SIZ_VID=16
...
```

Megjegyzendő, hogy ha ezt a közvetlen módszert egy olyan programban alkalmazzuk, ahol határozatlan a 2. LAP tartalma, a rendszerszegmenst is be kell lapozni a változók letárolása előtt.

Végül még egy megjegyzést. A TAPE perifériakezelő megkülönbözteti az EXOS 1 és EXOS 2 hívásokat. Egy nem létező csatorna megteremtése, pufferfoglalás stb. az EXOS 2 funkcióhívással kérhető, míg az EXOS 1 egy már létező csatornát nyit meg az adatforgalomnak.

*EXOS 3: csatorna lezárása*
*EXOS 4: csatorna megszüntetése*

Bemenő adat: A a csatorna száma (0—254)

A két funkciót minden beépített perifériakezelő azonos módon kezeli. Maguk a periférák csak — a saját működésükhöz tartozó — részleges funkciókat látnak el (jelzőbájtok nullázása, a VIDEO-kezelőnél a lezárt csatornához tartozó éppen kijelzett sorok letakarása stb.). A nyilvántartásból való törlést, az ezzel járó memóriarendezést az EXOS végzi.

A funkció előkészítés nélkül hívható:

```
LD A,CSAT
EXOS 3
```

BASIC betöltővel (egyben ellenőrizve is, hogy valóban megtörtént-e a csatorna lezárása):

```
  100 ALLOCATE 100
  110 OPEN #1:"VIDEO:"
  120 CODE M=HEX$("3E,01,F7,03,C9")
  130 CALL USR(M,0)
  140 PRINT #1:"OK?"
START

*** Kanal nicht vorhanden.
140 PRINT #1:"OK?"
```

A futás hibajelzéssel áll le a szóban forgó csatornára írni próbáló utasításnál, igazolva ezzel a lezárást.

Ha olyan csatornát próbálunk meg lezárni, amelyik nincs is nyitva (nincs benne a csatornaláncban), az EXOS hibakóddal tér vissza. Ellenőrizzük ezt a kódot, felhasználva, hogy a HL értékét a USR kiértékelő rutinja visszaadja a BASIC-nek. Az EXOS-ból való visszatérés után írjuk a hibakódot hordozó A értékét HL-be:

```
LD H,0
LD L,A
```

```
  100 ALLOCATE 100
  110 CODE M=HEX$("3E,01,F7,03,26,00,
      6F,C9")
  120 PRINT "Hibakod:";USR(M,0)
START
Hibakod: 251
ok
```

A kapott hibakód (FBH) ICHAN ("csatorna nem létezik") hibát jelez (l. a 4. Függeléket).

*EXOS 5: karakterolvasás*

Bemenő adat: A: csatornaszám
Kimenő adat: B: az olvasott érték

Az olvasott értékre csak a csatornák egy részénél illik a karaktermegnevezés, pl. grafikus módú csatornánál a VIDEO-kezelő a kurzorkoordináta pontjának palettaszínét adja vissza a hívás eredményeként.

Fel kell hívni a figyelmet, hogy egyes perifériák (a kezelőkön belül) várakoznak, amíg a karakter nincs készen az olvasásra (pl. a KEYBOARD kezelő vár addig, amíg meg nem nyomunk egy billentyűt).

Az első felhasználási példánk legyen a billentyűleolvasás. Az alapértelmezésben megnyitott 69H csatornáról olvassunk be egy karaktert és értékét adjuk vissza a BASIC-nek:

```
LD A,69H
EXOS 5
LD H,0
LD L,B
RET
```

A BASIC programban írjuk ki ezt az értéket, mint a megnyomott billentyű kódját és térjünk vissza az újabb karakter olvasására:

```
100 ALLOCATE 100
110 CODE M=HEX$("3E,69,F7,05,26,00,
    68,C9")
120 PRINT "A lenyomott billentyü
    kodja:";USR(M,0)
130 GOTO 120
```

A rutinnal kiolvashatunk minden billentyűkódot (különböző shiftelési módok mellett). Figyelemre méltó a funkcióbillentyűk hatása. Azt láthatjuk, hogy a kezelő (saját pufferjéből) a billentyűhöz rendelt összes karaktert kiadja (egyenként) a megismételt olvasási funkcióhívások hatására (és akkor áll le újabb várakozásra, amikor a puffere kiürült).

Igen érdekes a funkció felhasználása egy magnetofonról beolvasható file letapogatására:

```
100 ALLOCATE 100
110 OPEN #1:""
120 CODE M=HEX$("3E,01,F7,05,26,00,
    68,C9")
130 PRINT USR(M,0)
140 GOTO 130
```

A program elindítása után állítsuk a kazettát egy tetszőleges rögzített file-ra és indítsuk el a lejátszást.

Befejezésül lássunk még egy példát a funkcióhívás felhasználására grafikus VIDEO-csatorna esetén. A program beolvassa — és a képernyő alsó sávjában kiírja — a pillanatnyi kurzorkoordináták palettaszínét.

```
100 ALLOCATE 10
110 CODE M=HEX$("3E.65,F7,05.26,00,
    68,C9")
120 LET X,Y=100
130 GRAPHICS
140 PRINT AT 2,1:"Palettaszin:"
150 PLOT 200,200,ELLIPSE 100.100,
160 PLOT 350,200,ELLIPSE 100.100,
170 SET INK 1
180 PLOT 200,200,PAINT
190 SET INK 2
200 PLOT 275,200,PAINT
210 SET INK 3
220 PLOT 350,200.PAINT
```

```
230 PLOT X,Y,
240 LET P=USR(M,0)
250 PRINT AT 2,15:P
260 SET INK 3+3*(P=3)
270 PLOT X,Y
280 SET INK P
290 PLOT X,Y
300 GET A$
310 IF A$="" THEN 360
320 IF ORD(A$)=176 THEN LET Y=Y+4
330 IF ORD(A$)=180 THEN LET Y=Y-4
340 IF ORD(A$)=184 THEN LET X=X-4
350 IF ORD(A$)=188 THEN LET X=X+4
360 GOTO 230
```

A villogó pontkurzor a beépített botkormánnyal vezérelhető pontonként a képernyő adott helyére. A rajzolás és a kurzorbeállítás egyelőre a BASIC program feladata, ezek gépi kódú megoldására a 8. Függelék ad információt.

*EXOS 6: blokk olvasása*

Bemenő adatok:

A: csatornaszám
DE: puffercím
BC: az olvasandó blokk hossza

Kimenő adatok:

DE: a puffer következő szabad helyére mutat
BC: a még olvasandó karakterek száma (ha hiba történt olvasás közben)

A DE-t egy szabad RAM-terület kezdőcímére kell állítani (itt is tetszőleges LAP-on). Az EXOS ettől kezdve tárolja az olvasott blokkot. A visszaadott értékek olvasási hiba esetén használhatók fel a továbbolvasáshoz.

A funkció szemléltetésére alakítsuk át az előző pont valamelyik példaprogramját. A kiolvasás szemmel követhető lesz, ha pufferterületként az állapotsorban kijelzett memóriaterületet jelöljük ki (amíg a program fut, ezt tetszőlegesen használhatjuk, csak a leállás — pl. STOP — hatására írja felül az operációs rendszer).

A pufferterületet hívás előtt törölni is kell. Így az assembly rutin:

```
        LD  HL,0BEBEH   ;puffer kezdőcím
        PUSH HL         ;verembe
        LD  B,20H       ;32 byte hossz
L1      LD  (HL),20H    ;"SPACE" bufferbe
        INC HL          ;
        DJNZ  L1        ;32 hely törlése
        POP DE          ;DE: puffercím
        LD  BC,0020H    ;BC: 32 (hossz)
        LD  A,1         ;csatornaszám=1
        EXOS 6          ;blokkolvasás
        RET             ;
```

Használjuk ezt a rutint pl. a kazettafile beolvasására:

```
100 ALLOCATE 100
110 OPEN #1:""
120 CODE M=HEX$("21,BE,BE,E5,06,20,
    36,20,23,10,FB,D1,01,20,00,3E,
    01,F7,06,C9")
130 CALL USR(M,0)
140 PING
150 GOTO 130
```

A rutinból való visszatérést hangadás jelzi, majd továbbolvassuk a csatornát. Érdemes más csatornák olvasásával is próbálkozni.

*EXOS 7: karakter írása*

Bemenő adatok:

A: csatornaszám

B: az írandó karakter

Az EXOS számunkra legsokoldalúbban használható hívása (a blokkíró EXOS 8-cal együtt). A számítógép szinte minden látható és hallható kijelzési, sőt adatátviteli funkciója elérhető ezen keresztül. A kiírandó karakter (az EXOS főágában előkészített egyéb paraméterekkel együtt) a perifériakezelőhöz kerül. Itt dől el — a kezelő feladatának és a kivitt értéknek megfelelően —, hogy a kiírt bájt (bájtsorozat) a képernyő egy adott részén jelenik-e meg alfanumerikus karakterként, vagy magnetofonkazettára kerül-e, esetleg egy grafikus alakzat kitöltését indítja-e el stb. A funkció értelmezése tehát csak egy-egy adott periféria esetén lehetséges.

A hangkezelő (SOUND) pl. a képzendő hangok paramétereit és vezérlőkaraktereket vár ilyen formában. Van azonban egy olyan karakter (a 7-es) amelynek kiírása egymagában kiváltja egy alapértelmezésű hang megszólalását (PING):

```
LD A,67H
LD B,7
EXOS 7
RET
```

BASIC-ből betöltve és elindítva:

```
100 ALLOCATE 100
110 CODE M=HEX$("3E,67,06,07,F7,07,
    C9")
120 CALL USR(M,0)
```

Ezt is felhasználhatjuk gépi kódú programjainkban.

Ezek után nézzünk egy példát a perifériakezelő közvetlen hívására. Természetesen olyan példát, amelyben kézzel fogható ennek előnye. Legyen a feladat pl. 255 db azonos karakter (mondjuk A) kiírása a képernyőre.

A feladat az EXOS 7 hívással könnyen megoldható:

```
        LD   B,FFH        ;számláló
C1      PUSH BC           ;verembe
        LD   A,66H        ;csatornaszám
        LD   B,51H        ;karakter (Q)
        EXOS 7            ;kiíratás
        POP  BC           ;számláló vissza
        DJNZ C1           ;ismétlés 255-ször
        RET               ;
```

A BASIC betöltő programot kiegészítettük, hogy a rutin futásidejét mérhetővé tegyük. A 140. sor folyamatosan átírja a kiírandó karakter kódját (így jobban tudjuk követni a képernyőn a folyamatot).

```
100 ALLOCATE 100
110 CODE M=HEX$("06,FF,C5,3E,66,06,
    51,F7,07,C1,10,F6,C9")
120 TIME "00:00:00"
130 FOR N=65 TO 90
140   POKE M+6,N
150   PRINT AT 1,1:
160   CALL USR(M,0)
170   PRINT AT 10,1:
180 NEXT
190 PRINT "Idő: ":TIME$
```

Ezek után térjünk rá a VIDEO perifériakezelő kiíratő rutinjának (0/D4D0H, l. 5. Függelék) közvetlen hívására.

Az EXOS főág követésével megállapítottuk, hogy a végrehajtó rutinnak átadandó, előkészítendő paraméter a kiíratandó karakteren kívül (B) a szóban forgó csatorna pufferének báziscíme (IX) is. Ugyanitt láttuk azt is, hogy milyen módon keresi ki az EXOS a csatorna leíróját. Ehhez a kereséshez használjuk mi is a 0. szegmens CD3AH rutinját. Ez az alprogram a BFCDH báziscímmel induló csatornaláncból kikeresi a csatornát a belső csatornaszámot tartalmazó A alapján. Belapozza a leíró szegmensét az 1. LAP-ra és HL-ben a leíró báziscímét adja vissza (ill. Z státusszal tér vissza, ha nem találja a keresett csatornát a láncban).

Mivel tudjuk, hogy maga a leíró 16 bájt hosszú, a leíró báziscíméből már könnyen meg tudjuk határozni a közvetlenül alatta kezdődő puffer báziscímét. Ezt IX-be töltve már hívhatjuk a végrehajtó rutint. Megjegyezzük, hogy — az egyszerűség kedvéért — itt most elhagytuk az EXOS által ugyancsak átadott periférialeíró pointer előkészítését, (B', IY), ezt ui. nem igényli az adott végrehajtó rutin.

A következő rutin az előbbi lépések szerint oldja meg a feladatot:

```
        PUSH IX           ;BASIC 'IX" mentés
        IN   (A),0B3H     ;pillanatnyi 3. LAP
        PUSH AF           ;verembe
        XOR  A            ;A=0
        OUT  (0B3H),A     ;3. LAP: 0. ROM
        LD   A,67H        ;!belső csatornaszám!
        CALL 0CD3AH       ;csatornát keres
        JR   Z,L2         ;végére, ha nincs
        LD   BC,0FFF0H    ;-16
```

```
        ADD   HL,BC       ;leíró báziscím-16
        PUSH  HL          ;
        POP   IX          ;IX=puffer báziscím
        LD    B,0FFH      ;számláló
L1      PUSH  BC          ;verembe
        LD    B,51H       ;karakter (Q)
        CALL  0D4D0H      ;VIDEO kiírás
                           végrehajtó rutin
        POP   BC          ;számláló vissza
        DJNZ  L1          ;ismétlés 255-ször
L2      POP   AF          ;eredeti 3. LAP
        OUT   (0B3H),A    ;visszalapozása
        POP   IX          ;BASIC "IX" vissza
        RET               ;
```

ill. ennek BASIC betöltője és hívója:

```
100 ALLOCATE 100
110 CODE M=HEX$("DD,E5,DB,B3,F5,AF,
    D3,B3,3E,67,CD,3A,CD,28,12,01,
    F0,FF,09,E5,DD,E1,06,FF,C5,06,
    51,CD,D0,D4,C1,10,F7,F1,D3,B3,
    DD,E1,C9")
120 TIME "00:00:00"
130 FOR N=65 TO 90
140   POKE M+26,N
150   PRINT AT 1,1:
160   CALL USR(M,0)
170   PRINT AT 10,1:
180 NEXT
190 PRINT "Idő: ":TIME$
```

A BASIC program a gépi kódokat tartalmazó (110) és a kiírandó karakter kódját változtató (140) sortól eltekintve megegyezik a feladat előző megoldásának programjával. Így összemérhető a két futásidő. Az EXOS funkcióhívással dolgozó rutin futásideje kb. 2—3-szor akkora, mint a közvetlen hívó második megoldásé. Nyilvánvaló tehát, hogy — legalábbis egyes feladatoknál — érdemes a formailag rendkívül egyszerű EXOS hívás helyett egy kicsit többet fáradni a közvetlen hívással.

*EXOS 8: blokk írása*

Bemenő adatok:

A: csatornaszám
DE: a kiírandó blokk (szöveg) kezdőcíme
BC: a kiírandó bájtok száma

Kimenő adatok:

BC: a még ki nem írt karakterek száma (ha hiba történik kiírás közben)
DE: a még hátralevő szöveg címe

Nagyobb tömegű adat, összetartozó karaktercsoport stb. ideális kiviteli eszköze. Egyszerű alkalmazási példaként írassuk az 1. szegmens egyik copyright-üzenetét az aktuális csatornára:

```
100 ALLOCATE 100
110 CODE M=HEX$("11,80.F5,01,21,00,
    3E,FF,F7,08,C9")
120 CALL USR(M,0)

START
 1985 Intelligent Software Ltd.
c>
```

*EXOS 9: csatornakészenlét olvasása*

Bemenő adat:
A: csatornaszám
Kimenő adat:
C: készenlétjelző bájt

Olyan csatorna esetén praktikus az alkalmazása, amely olvasásnál várakozna a készenlétig (pl. KEYBOARD). Így várakozás nélkül is információt szerezhetünk arról, van-e olvasható karakter a perifériakezelő pufferében.
A készenléti státuszt C hordozza:

C = 0: a periféria készen (van olvasható karakter)
C = 1: a periféria nincs kész (nincs olvasható karakter)
C = FFH: file vége

A BASIC rendszerben a GET utasítás használja pl. ezt a hívást, mielőtt megkísérli az olvasást (nehogy bennragadjon a kezelőben).

A készenléti jelzőt írja ki a következő program az 1. sorba a billentyűzet csatornájáról a

```
LD A,69H
EXOS 9
LD H,0
LD L,C
RET
```

gépi rutin felhasználásával. A kész karaktert folyamatosan ki is olvassuk, mert különben egyetlen billentyű leütése után csak az állandó készenlétet látnánk.

```
100 ALLOCATE 10
110 CODE M=HEX$("3E,69.F7,09,26,00,
    69,C9")
120 LET KOD=USR(M,0)
130 PRINT AT 1,1:KOD;
140 IF KOD=0 THEN
150   GET A$
160   PRINT A$;
170 END IF
180 GOTO 120
```

*EXOS 10: csatornaállapot megadása és olvasása*

Az alapkiépítésű gépen nem használt funkció: egyetlen beépített perifériakezelő sem fogadja (NOFN hibakódot küldenek vissza). Saját perifériakezelőnkben tetszés szerint használhatjuk (a funkció pl. lemezkezelő bővítés esetén nyer értelmet).

*EXOS 11: speciális funkció*

Bemenő adatok:
A: csatornaszám
B: a speciális funkció kódja
C, DE: átadható paraméterek a funkció igénye szerint
Kimenő adatok:
BC, DE: tartalmazza a funkció által esetleg visszaadott értékeket
A perifériák vezérlőfunkciója. A beépített perifériák egy része (SERIAL, TAPE, PRINTER) nem kezeli ezt a hívást (ISPEC — Érvénytelen speciális funkcióhívás — hibakódot küld válaszként). A többi kezelő elfogadja a hívást a B értékének egy-egy tartományában:

| | |
|---|---|
| 1—4: | VIDEO |
| 8—9: | KEYBOARD |
| 16—17: | NET |
| 24—26: | EDITOR |

A várhatóan leggyakrabban hívott képernyő-, ill. billentyűzetkezelés esetén nézzük meg egy kicsit részletesebben az elérhető funkciókat és hívásuk módját:

*B = 1: videolap kijelzése*
BASIC-ben
DISPLAY #csat: FROM kezdősor TO végsor AT képernyősor utasítással elvégezhető funkciót látja el: a képernyő adott sorától kezdődően kijelzi a szóban forgó csatornán felépített videolap kezdő- és végsor közötti részét.
Paraméterei:

A: kijelzendő csatornaszám
C: kezdősor
D: kijelzendő sorok száma (végsor – kezdősor + 1)
E: képernyősor

Lássunk egy rövid rutint a funkció alkalmazására:

```
LD  A,65H       ;grafikus csatorna
LD  C,L         ;kezdősor BASIC-ből
LD  B,1         ;funkciókód: DISP
LD DE,0A05H     ;D: 10 kijelzendő sor
                 E: az 5. sortól
EXOS 11         ;funkcióhívás
RET             ;
```

Egy kis BASIC kiegészítéssel jól látható a funkció hatása. A kijelzendő laprészlet (10 sor) kezdő sorszámát BASIC-ből adjuk meg (a USR második argumentuma L-be, onnan a rutinban C-be kerül).

```
 10 PROGRAM "DISP"
100 ALLOCATE 100
110 CODE DISP=HEX$("3E,65,4D,06,01,
    11,05,0A,F7,0B,C9")
120 DISPLAY GRAPHICS
130 PLOT 600,350,ELLIPSE 400,300,
140 PLOT 0,700,
150 PRINT #101:"Felső resz:"
160 PLOT 0,100,
170 PRINT #101:"Also resz:"
180 DISPLAY TEXT
200 CALL USR(DISP,1)
210 WAIT 1
220 CALL USR(DISP,11)
230 WAIT 1
240 GOTO 200
```

A következő két speciális funkcióhívás az adott csatornához tartozó videolap paramétereit olvassa ki és adja vissza a hívónak.

*B = 2: üzemmód lekérdezése*

Kimenő adatai:

B: a videolap szélessége karakterekben (X)
C: a videolap magassága karakterekben (Y)
D: a kijelzési mód
E: a színmód

A négy regiszterben visszaadott érték tulajdonképpen az

X—SIZ—VID
Y—SIZ—VID
MODE—VID és
COLOR—VID

EXOS változók értéke a szóban forgó csatorna megnyitásakor.

*B = 3: tárolási cím lekérdezése*

Kimenő adatai:

BC: fő tárolási cím

DE: másodlagos tárolási cím

A video adatok tárolásáról (és közvetlen eléréséről) a 8. Függelékben írunk még. Példaként most alkalmazzuk ez utóbbi két speciális funkcióhívást TEXT 80 üzemmódban az alapértelmezésű VIDEO-csatorna lekérdezésére (66 H). A visszakapott paramétereket tároljuk a BASIC által nem használt 0180H—0187H területeken (decimálisan 384—391)

```
LD  A,66H       ;VIDEO-csatorna
LD  B,2         ;"mód és méret" kód
EXOS 11         ;lekérdezés
LD (0180H),BC   ;Y -> 0180H
                 X -> 0181H
LD (0182H),DE   ;COLR -> 0182H
                 MODE -> 0183H
LD  A,66H       ;VIDEO-csatorna
LD  B,3         ;"tár.cím" kód
EXOS 11         ;lekérdezés
LD (0184H),BC   ;fő RAM címe->0184/5H
LD (0186H),DE   ;2. RAM címe->0186/7H
RET
```

A BASIC betöltőprogramban egyúttal kiolvassuk és ki is íratjuk a funkcióhívás tárolt eredményeit.

```
100 ALLOCATE 100
110 CODE M=HEX$("3E,66,06,02,F7,0B,
    ED,43,80,01,ED,53,82,01")
120 CODE =HEX$("3E,66,06,03,F7,0B,
    ED,43,84,01,ED,53,86,01")
130 TEXT 80
140 CALL USR(M,0)
150 PRINT "A"
200 FOR N=384 TO 391
210   PRINT N;PEEK(N)
220 NEXT
```

Értelmezzük együtt a futtatással kapott adatokat:

```
A
 384  24
 385  80
 386  0
 387  2
 388  156
 389  154
 390  28
 391  222
ok
PRINT SPEEK(254,154*256+156)
 24
ok
PRINT SPEEK(255,222*256+28)
 65
ok
```

— sorok száma (Y—SIZ—VID) : 24
— karakterek száma a sorban (X—SIZ—VID) : 80
(TEXT 80!)
— színmód (COLR—VID) : 0
(két szín)
— kijelzési mód (MODE—VID) : 2
(szoftver szövegmód, ez is a "TEXT 80"-nak felel meg)
— fő tárolási cím (jelen esetben a bittérkép tárolási kezdő címe):
LBS: 156=9CH
MSB: 154=9AH
— másodlagos tárolási cím (jelen esetben az ASCII-kódokból felépített térkép)
LBS: 28 =1CH
MSB: 222= DEH

Itt is felhívjuk a figyelmet, hogy a két tárolási cím VIDEO cím. Az adatokat tároló szegmens száma a cím MSB tartományából határozható meg.
1. 80H < 9AH < BFH a fő cím az FEH szegmensben van
2. C0H < DEH < FFH a másodlagos cím az FFH szegmensben van

Ezt a két kezdőcímet kérdezi le a fenti képernyőmásolat két sora. A fő címen lévő érték 24 (00011000) a képernyő bal felső sarkába kiíratott A karakter pontmátrixának felső sora. A másodlagos cím tartalma (65) ennek az A betűnek az ASCII kódja.

*B = 4: karakterkészlet-inicializálás*

A funkciónak külön be- és kimenő adata nincs. Még a megadott csatornaszám is csak annyiban kritikus, hogy VIDEO perifériához megnyitott csatorna legyen (ez biztosítja, hogy a hívás a VIDEO kezelőhöz jusson el).

A ROM-adatmezőből — hidegindítási folyamathoz hasonlóan — újratölti a RAM karaktergenerátor területet. Hatása — a hívott csatornától függetlenül — valamennyi videocsatornára kiterjed.

A következő két speciális funkcióhívás a billentyűzetkezelőé:

*B = 8: funkcióbillentyűk programozása*

Bemenő adatok:

A: a billentyűzetcsatorna száma (BASIC alapértelmezésben 69H)
C: a programozni kívánt funkcióbillentyű száma – 1 (0—7, ill. shiftelt gombokhoz 8—15)
DE: a beprogramozandó szöveg előtti hosszbájtra mutat

Példaként írjuk át a F8 funkciógombot (C = 7) úgy, hogy megnyomására AUTO üzemmódba kerüljön a gép, és a 100-as sorba írja is be a programjainkban igen gyakran használt ALLOCATE utasítást, de argumentum nélkül:

```
100 ALLOCATE 100
110 CODE SZOVEG=HEX$("10,A5")&"AUTO"
    &HEX$("A1,0D")&"ALLOCATE "
120 CODE FK=HEX$("3E,69,01,07,08,11"
    )&WORD$(SZOVEG)&HEX$("F7,0B,C9")
130 CALL USR(FK,0)
```

A beprogramozott szövegben a hosszbájt 10H (16 karakter következik), az első kiírt bájt (A5H) visszatöröl a sor elejéig. Ezután következik az AUTO szöveg, törlés a sor végéig, majd a sor bevitele (0DH = ENTER). Az ezt követő ALLOCATE karakterekkel befejezzük a beprogramozott szöveget, nem zárjuk le a sort, így a kurzor a szöveg utáni pozíción marad; a programozó beírhatja a lefoglalni kívánt bájtok számát.

*B = 9: a botkormányok olvasása*

Bemenő adatok:

C: botkormány-választás
C = 0: belső
C = 1: külső 1.
C = 2: külső 2.

Kimenő adat: C = olvasott érték
A botkormány és a tűzgomb (a beépített joystick esetén SPACE) pozícióját a C regiszter bitjei reprezentálják (1, ha megnyomott):

```
JOBB:  BIT 0,C
BAL:   BIT 1,C
LE:    BIT 2,C
```

FÖL: BIT 3,C
TÜZ: BIT 4,C

Végül a további (várhatóan ritkábban igényelt) speciális funkcióhívások (részleteket az EXOS leírás tartalmaz):

NET
B = 16: kiviteli puffer ürítése (FLUSH)
B = 17: pufferek törlése (CLEAR)

EDITOR
B = 24: lapszélmargók beállítása
B = 25: dokumentum betöltése
B = 26: dokumentum kivitele

*EXOS 16: EXOS változó olvasása, írása, átbillentése*

Belépési pont: C8F9H
Bemenő adatok:

B: az elvégzendő művelet kódja
B = 0: olvasás
B = 1: írás
B = 2: átbillentés
C: az EXOS változó száma
D: beírandó érték

Kimenő adat:
D: a változó értéke

Ez az egyetlen EXOS hívás, amit még megszakításkezelés közben is elfogad a rendszer. Tulajdonképpen rögzített címen kezdődő változó táblázat adatainak kezelését végzi.

Ha a változó száma 39-nél nagyobb, E-ben a változó számával, C-ben az "EXOS változó" kóddal (4) végigkérdezi a rendszerbővítőket, lehetőséget adva így új változók bevezetésére, kezelésére.

```
C8F9  21 69 C4    LD   HL,C469     ;visszatérési cím
C8FC  E5          PUSH HL          ;verembe
C8FD  79          LD   A,C         ;változó száma
C8FE  D6 28       SUB  28          ;-40
C900  30 2C       JR   NC,C92E     ;bővítőkhöz, ha n>39
```

A 0 és 39 közötti (alapértelmezésű) változó hívása esetén megállapítja a hozzá tartozó táblázatcímet: HL = BFC5H + n. Az 1. változót külön kezeli (FLAG = SOFT

= IRQ), ezt a BFF2H címen keresi és manipulálja. Az EXOS változók táblájának azonos nevű eleme voltaképpen csak ennek másolata lesz.

```
C=02  F5          PUSH AF          ;
C=03  21 F2 BF    LD   HL,BFF2     ;FLAG_SOFT_IRQ
C=06  79          LD   A,C         ;változószám
C=07  FE 01       CP   01          ;=1?
C=09  28 03       JR    Z,C90E     ;cím marad, ha igen
C=0B  C6 C5       ADD  A,C5        ;egyébként
C=0D  6F          LD   L,A         ;HL=BFC5+n (tábla)
```

A kiszámított címen elvégzi a kért műveletet:

```
C90E  05          DEC  B           ;funkciókód?
C90F  FA 18 C9    JP    M,C918     ;ha B=0 volt (olv.)
C912  28 03       JR    Z,C917     ;ha B=1 volt (írás)
C914  7E          LD   A,(HL)      ;változó értéke
C915  2F          CPL              ;átbillentés
C916  57          LD   D,A         ;
C917  72          LD   (HL),D      ;új érték változóba
C918  56          LD   D,(HL)      ;(vissza)-olvasás
```

REM 1 és REM 2 — hívó esetén (36., ill. 37. változó) a megfelelő PORT-okon is beállítja az aktuális értéknek megfelelő szinteket, majd — a RET hatására — elugrik a belépéskor PUSH utasítással tárolt címre: az EXOS funkcióhívások visszatérési ágára:

```
C919  F1          POP  AF          ;kód-40
C91A  D6 FC       SUB  FC          ;kód-36
C91C  28 02       JR    Z,C920     ;ha kód=36
C91E  FE 01       CP   01          ;kód=37?
C920  CC 25 C9    CALL  Z,C925     ;REM1, REM2 esetén
                                    PORT-okat állít
C923  AF          XOR  A           ;A=0 státusz
C924  C9          RET              ;(vissza az EXOS
                                     befejezésre)
```

A funkció közvetlenül is hívható (természetesen a visszatérési címet verembe töltő bevezető utasítások átlépésével, a C8FDH címen), de igazi jelentősége inkább a változók közvetlen elérésének van. Az ehhez szükséges címeket (és az inicializálás utáni értékeket) a 3. Függelék tartalmazza. Az alkalmazásra már láttunk példát az EXOS 1, 2 tárgyalásakor.

*EXOS 17: csatorna átirányítása olvasáshoz*

Belépési pont: C682H

*EXOS 18: csatorna átirányítása íráshoz*

Belépési pont: C681H
A két hívás együtt tárgyalható, a rendszer is együtt kezeli ezeket. Mindkét hívás

feladata a főcsatorna átállítása olyan módon, hogy ezután a rávonatkozó hívások a másodlagos csatornára irányítódjanak.
Bemenő adatok:

A: a főcsatorna száma

B: a másodlagos csatorna száma (a C = FFH érték törli az átirányítást)

A két funkció érdekessége, hogy — bár csatornákat érintő utasításokról van szó — a rendszer nem a csatornautasítások között (1—11) kezeli őket. Ennek oka, hogy ilt a perifériakezelőket nem kell bevonni a végrehajtásba. A rendszer a hívások hatására csak a főcsatornák leírójának egy-egy bájtját állítja át:

```
C681  37           SCF              ;CY=1, ha EXOS 18
                                     CY=0, ha EXOS 17
C682  59           LD   E,C         ;E: másodlagos csat.
C683  F5           PUSH AF          ;státusz verembe
C684  08           EX   AF,AF'      ;A: fő csatornaszám
C685  3C           INC  A'          ;belső csatornaszám
C686  C4 3A CD     CALL NZ,CD3A     ;keresi a láncban
                                     HL=leíró báziscím
C689  3E FB        LD   A,FB        ;".ICHAN" hibakód
C68B  28 09        JR    Z,C696     ;hiba, ha A=FFH
                                     vagy nincs megnyitva
C68D  0E 0F        LD   C,0F        ;C=15
C68F  F1           POP  AF          ;státusz vissza
C690  ED 42        SBC  HL,BC       ;HL=cím-15, ha E.17
                                        cím-16, ha E.18
C692  1C           INC  E           ;2. csat. belső szám
C693  73           LD   (HL),E      ;a leíróba
C694  AF           XOR  A           ;A=0 státusz
C695  C9           RET              ;
```

Az átirányítás megvalósítása tulajdonképpen — mint láttuk — a csatornahívásokat előkészítő EXOS főágban történik. Így ezt a funkciót a szóban forgó csatornaleíró közvetlen manipulálásával is kiválthatjuk.

A két funkcióhívás a BASIC

"CAPTURE FROM főcsatorna TO mellékcsatorna", ill.

"REDIRECT FROM főcsatorna TO mellékcsatorna"

gépi megvalósítója. Segítségükkel az igen praktikus átirányítás (pl. a képernyőkezelésre tervezett program átállítása egyetlen utasítással nyomtatókezelőre) gépi programjainkban is egyszerűen elérhető, pl. az említett esetre:

```
...
LD  A,66H     :VIDEO-csatornáról
LD  C,68H     :PRINTER-csatornára
EXOS 18       :átirányítás
...
```

*EXOS 19: az alapértelmezésű periférianév beállítása*

Belépési pont: C7E5H

Bemenő adatok:

C: a periféria típuskódja

C = 0: nem file-kezelő (magnetofon)

C = 1: file-kezelő (lemezegység)

DE: a perifériánév előtti hosszbájtra mutat (a hívó memóriakiosztásában bármelyik LAP-on)

Mint a csatornakezelő EXOS hívások előkészítésénél láttuk, ha a csatornanyitáskor megadott perifériánév 0 hosszúságú vagy nem tartalmaz kettőspontot, a rendszer az alapértelmezésű periféria nevével helyettesíti azt (TAPE).

Ezt a nevet írhatjuk át az EXOS 19 hívással. A DE-vel megcímzett új név bármelyik könyvtárazott periféria neve lehet (a szövegvégi kettőspont itt nem kell).

A hívás a nagybetűsített és a karakter tartományában ellenőrzött szöveget rögzített című pufferbe tölti:

BF18H: hossz

BF19—: karakterek

Ennek ismeretében akár közvetlenül is lekérdezhető vagy átírható az alapértelmezésű perifériánév.

Használata praktikus lehet pl. abban az esetben is, ha igen gyakran nyitunk meg csatornát egy adott perifériához. A következő program VIDEO-ra írja át az alapértelmezésű nevet.

```
100 ALLOCATE 100
110 CODE NEV=CHR$(5)&"VIDEO"
120 CODE M=HEX$("0E,00,11")&WORD$
    (NEV)&HEX$("F7,13,C9")
130 CALL USR(M,0)
```

A program futtatása után

OPEN#n :""

alakban nyithatunk meg videocsatornákat.

*EXOS 20: rendszerállapot lekérdezése*

Belépési pont: C808H

Bemenő adat:

DE: a puffer címe, ahová az EXOS töltheti az információt hordozó bájtokat

Kimenő adatok:

B: az EXOS verzió száma (BCD alak; esetünkben 21H)

pufferben az állapotadatok

Az egyszerű rutin a rendszerváltozó-terület 7 bájtját átmásolja a hívó pufferébe:

```
C808   D5          PUSH DE          :puffercim verembe
C809   CD 3A D2    CALL D23A        :pufferszg. 1.LAP-ra
C80C   21 9E BF    LD   HL,BF9E     :változóterület címe
C80F   0E 07       LD   C,07        :bájtok száma
C811   CD 00 C8    CALL C800        :7 bájtot átmásol
                                       a pufferbe
C814   06 21       LD   B,21        :verziószám
C816   D1          POP  DE          :puffercim vissza
C817   C9          RET              :
```

Az átadott adatokkal már találkoztunk a hidegindítás folyamatának követésénél. Nézzük most meg az EXOS hívás segítségével a mi verziónk pillanatnyi paramétereit. Írjuk a programunkat egy 0-tól különböző programszámon (pl. EDIT 1), hogy ennek a hatását is láthassuk. Pufferként ismét a 0180H kezdőcímet adjuk meg.

```
100 ALLOCATE 100
110 CODE M=HEX$("11,80,01,F7,14,26,
'   00,68,C9")
120 LET B=USR(M,0)
130 PRINT "A verzioszam:";B
135 PRINT "A rendszerallapot:"
140 FOR N=0 TO 6
150   PRINT N;PEEK(384+N)
160 NEXT
```

A futás eredménye:

```
START
A verzioszam: 33
A rendszerallapot:
 0  0
 1  5
 2  1
 3  0
 4  1
 5  8
 6  0
ok
```

A verziószámban természetesen csak hexadecimális alakban (21H) ismerhetjük fel a 2.1-es verziószámot. A többi adat rendre:

— 0. A megosztott szegmens száma (jelenleg nincs ilyen).
— 1. A szabad szegmensek száma (5: a 8 jó szegmensből 1 rendszerszegmens, 1 nulláslap, 1 kiutalva).
— 2. A felhasználónak kiutalt szegmensek száma (itt 1, az a szegmens, amiben éppen dolgozunk az 1. BASIC programmal).
— 3. A perifériáknak kiutalt szegmensek száma (itt nincs ilyen).
— 4. A rendszer részére kijelölt szegmensek száma (jelenleg 1, az FFH szegmens).
— 5. Az összes (működő) RAM-szegmens száma (8).
— 6. A hibás RAM-szegmensek száma (0).

*EXOS 21: új periféria felvétele*

Belépési pont: C698H
Bemenő adatok:
DE: a bevezetni kívánt periféria leírójára mutat (TYPE)
BC: az igényelt RAM nagysága

Ha a felhasználó egy saját perifériát szeretne rendszerbe illeszteni, az eszköz és a kezelő paramétereit kötött formájú leíróban kell összefoglalni. Erről az inicializálás során volt már szó, és egy konkrét példa kapcsán foglalkozunk is még vele a 2.5 alfejezetben.

A hidegindítás során a ROM-szegmensek beépített perifériáinak leíróit a rendszerszegmensbe másolja át az EXOS. Az EXOS 21 funkcióhívással utólag beillesztendő leíró viszont megmarad saját szegmensében, csak a címmutatók állításával illeszti be a rendszer a leírók láncába. Ezért az ilyen perifériakezelőnek feltétlenül RAM-ban kell lennie. Eltér az inicializálás beláncolási műveletétől az a momentum is, hogy itt a perifériához igényelt RAM méretét nem a leíró (ill. álleíró) tartalmazza, hanem a beláncolás kérésekor kell megadni BC-ben.

Tekintsük át, milyen lépéseket követ az EXOS egy felhasználói periféria felvételekor:

— Az 1. LAP-ra lapozza a leíró szegmensét (azt a szegmenst, amelyre DE mutatott a híváskor).
— Ellenőrzi a leíró TYPE elemét. Csak TYPE = 0 esetén folyhat tovább a felvétel.
— Ellenőrzi a kért RAM nagyságát. Ha ez 0, a beláncolás egyáltalán nem foglal helyet a rendszer-RAM-ban. Ha viszont kér memóriát az eszköz, azt a csatornapuffer kiutalásának módszerével kapja meg (EXOS 27).
— Beláncolja a perifériát: a BFC0H báziscím alatti pointer helyére az új láncelem mutatóját (címét és szegmensszámát) tölti, a kiemelt pointert pedig a most beláncolt leíró elé írja. Ezek szerint a bevezetett leíró előtt is (legalább) 3 szabad RAM-helynek kell lennie!
— Ha a leíró UNIT-COUNT eleme 0 (nem lehet több ilyen nevű eszköz), érvényteleníti a lánc régebbi, azonos nevű perifériához tartozó leíróit (TYPE = FFH-t állít).

Befejezésül az igényelt — és kiutalt — pufferterület címével IX-ben meghívja a belépési címtáblázat 12. rutinját (INICIALIZÁLÁS). Ekkor van lehetősége a perifériakezelőnek a maga számára mindig elérhető helyen letárolni a kiutalt RAM-terület címét.

A perifériakezelő felépítésére és beláncolására a 2.5 alfejezetben kidolgozunk egy példát.

*EXOS 22: az EXOS határ kiolvasása*

Belépési pont: C6F2H
Kimenő adatok:
DE: az EXOS határ (0000H és 3FFFH közé transzformált érték)

C: a megosztott szegmens száma (0, ha nincs megosztott szegmens)
A rutin egyszerűen a — már ismert — rendszerváltozó értékeket olvassa ki és adja vissza:

```
C6F2  3A 9E BF     LD   A,(BF9E)    ;a megosztott szg. száma
C6F5  4F           LD   C,A         ;C-be
C6F6  ED 5B 91 BF  LD   DE,(BF91)   ;DE: EXOS határ
C6FA  AF           XOR  A           ;A=0 státusz
C6FB  C9           RET              ;
```

A címek ismeretében az értékek közvetlenül is elérhetők.

*EXOS 23: a felhasználói határ beállítása*

Belépési pont: C6FCH
Bemenő adat:
DE: a beállítani kívánt határ (0000 és 3FFFH közé transzformálva)
Ha van megosztott szegmens és a megadott felhasználói határ címe nem nagyobb az EXOS határnál, a rendszer bejegyzi DE értékét a rendszerváltozóba:

```
C6FC  3A 9E BF     LD   A,(BF9E)    ;a megosztott szg. száma
C6FF  B7           OR   A           ;van megosztott szg.?
C700  3E F4        LD   A,F4        ;".ISEG" hibakód
C702  C8           RET  Z           ;hiba, ha nincs

C703  2A 91 BF     LD   HL,(BF91)   ;HL= EXOS határ
C706  ED 52        SBC  HL,DE       ; - felhasználói határ
C708  3E F3        LD   A,F3        ;".IBOUND" hibakód
C70A  D8           RET  C           ;hiba, ha DE>HL
C70B  AF           XOR  A           ;A=0 egyébként
C70C  18 1C        JR   C72A        ;

C72A  ED 53 8F BF  LD   (BF8F),DE   ;felhasználói határt
C72E  C9           RET              ;       letárol
```

Ez utóbbi két hívás az EXOS memóriamenedzselésének fontos eleme. Ha a felhasználó (alulról felfelé tolódó) határa és a rendszer által (felülről lefelé) igénybevett terület határa — az EXOS határ — egy szegmensbe kerül, ez a két érték (pontosabban a köztük lévő különbség) korlátozza az elvégzendő műveleteket. Ezek kezelését, ellenőrzését minden hordozható, változó környezetben is alkalmazni kívánt felhasználói programnak el kell végeznie.

A BASIC értelmezőnek — nem tudván, hogy milyen memóriakiosztásba kerül — gyakorlatilag minden méretnövelő művelet (sorbeírás, változó tárolása stb.) előtt meg kell győződnie arról, hogy nem ütközik határokba. Fel kell készülnie arra is, hogy a megosztott szegmensben dolgozik, ahol lépésről-lépésre változik a két lefoglalt terület határa. Belátható, hogy ez a kötelezettség egy erős sebességkorlátozó tényező. Ha saját gépünk ismert kiosztású memóriáiban dolgozunk (esetleg egy lefoglalt szegmensben), sok időt — és energiát — takaríthatunk meg, elkerülve a folyamatos ellenőrzések kényszerét.

Belépési pont: C70EH

Bemenő adat: (közvetlen felhasználásnál) AF'-ben a hívási státusz, ez informálja a rendszert, hogy a szegmensigénylő egy felhasználói program (Z) vagy egy perifériakezelő (NZ).

Kimenő adatok:

C: a kiutalt szegmens száma

DE: a szegmensben szabadon használható bájtok száma

A (státusz): 7FH, ha megosztott szegmenst utalt ki az EXOS

A szegmensigénylés, lefoglalás a memóriagazdálkodás alapfunkciója. Számunkra is fontos lehet a kiutalás módja, ennek nyilvántartása. Kövessük hát a funkció végrehajtását.

A szegmensgazdálkodás alapja az a RAM-táblázat, amit az inicializálás során épített föl a rendszer, az ABD0H címtől lefelé bejegyezve a hibátlannak talált RAM-szegmensek számát (l. a 2.3.1 pontot).

Ebben a táblázatban tehát (mint láttuk is) alaphelyzetben, 128 K-s gép estén az ABCAH és ABD0H címek között növekvő sorrendben találjuk az F9H—FFH szegmensszámokat (a kötött nulláslap-szegmenssel (F8H) nem számol a rendszer).

A nyilvántartás másik része a rendszerváltozó területen folyik:

— a BF9A/BH tartalma arra a szegmensre mutat a táblázatban, amelyikben a felülről lefelé terjeszkedő EXOS határa éppen található (alaphelyzetben ABD0H)

— BF9EH: a megosztott szegmens száma, ha már van ilyen, azaz ki kellett utalni a felhasználónak a rendszer által is használt szegmens alját (alapérték 0: nincs megosztott szegmens)

— BF9FH: a szabad szegmensek száma (alapérték a nulláslap-szegmens és az eleve a rendszernek kiutalt FFH szegmens leszámítása után 6)

— BFA0H: a felhasználónak kiutalt szegmensek száma (alapérték 0)

— BFA1H: a perifériáknak kiutalt szegmensek száma (alapérték 0)

— BFA2H: a rendszer által igénybevett szegmensek száma (ide tartoznak a csatornapuffer területek, pl. a képernyőmemóriák stb.) (alapérték 1: az FFH szegmens)

Ezután lássuk, hogyan él a rendszer ezekkel az eszközökkel? Mindenekelőtt ellenőrzi, hogy van-e még szabad szegmens, amit kiutalhat a hívónak:

```
C70E  21 9F BF     LD   HL,BF9F     ;
C711  7E           LD   A,(HL)      ;A: szabad szg.-ek száma
C712  B7           OR   A           ;   =0?
C713  20 1A        JR   NZ,C72F     ;ugrik, ha van még sza-
                                      bad szegmens
```

Ha nincs szabad szegmens, megkísérli a rendszer által már használatba vett memóriát (EXOS határ szegmense) kiutalni. Ezt viszont perifériának nem utalja

ki, csak felhasználói programnak. Másrészt — természetesen — ezt csak akkor teheti meg, ha nem volt a kérdéses szegmens már előzőleg is megosztott.

```
C715  08          EX   AF,AF'      ;EXOS hívási státusz
C716  20 3A       JR   NZ,C752     ;hiba, ha periféria hív
                                     (".NOSEG")
C718  2A 9A BF    LD   HL,(BF9A)   ;RAM-tábla: EXOS határ
C71B  7E          LD   A,(HL)      ;határt tartalmazó szg.
C71C  21 9E BF    LD   HL,BF9E     ;HL: megosztott szg.
                                      számára mutat
C71F  BE          CP   (HL)        ;megegyeznek?
C720  28 30       JR    Z,C752     ;".NOSEG" hiba, ha már
                                      meg volt osztva
C722  77          LD   (HL),A      ;megosztott szg. száma
C723  4F          LD   C,A         ;C: kiutalt szg. száma
C724  3E 7F       LD   A,7F        ;A: ":SHARE" kód
C726  ED 5B 91 BF LD   DE,(BF91)   ;DE: EXOS határ
C72A  ED 53 8F BF LD   (BF8F),DE   ;a rendszerváltozót is
C72E  C9          RET              ;     beállítja
```

Miután láthatjuk, megosztott szegmens kiutalása esetén DE az EXOS határ alatt szabadon használható bájtok számát tartalmazza és A = 7EH.

Ha még van szabad szegmens, a rendszer azt utalja ki a hívónak. Ekkor nem tesz különbséget a felhasználó és a perifériakezelő — mint igénylő — között, csak annyiban, hogy hová könyveli be a kiutalt szegmenst.

```
C72F  47          LD   B,A         ;B: szabad szg.-ek száma
C730  35          DEC  (HL)        ;(szabad szg.-ek száma)-1
C731  23          INC  HL          ;BFA0
C732  23          INC  HL          ;BFA1
C733  08          EX   AF,AF'      ;EXOS hívási státusz
C734  7E          LD   A,(HL)      ;periféria-szegmensek sz.
C735  2B          DEC  HL          ;BFA0
C736  28 02       JR    Z,C73A     ;ugrik, ha nem periféria
C738  AF          XOR  A           ;átlépendő szegmensek sz.
C739  23          INC  HL          ;

C73A  34          INC  (HL)        ;(hívónak kiutalt szeg-
                                     mensek száma)+1
```

Ha felhasználó kérte a memóriát és a perifériáknak már van kiutalva szegmens, az EXOS átrendezi a RAM-táblát: a most kiutalt felhasználószegmenst a perifériaszegmensek elé teszi:

```
C73B  2A 9A BF    LD   HL,(BF9A)   ;RAM-tábla: EXOS határ
C73E  2B          DEC  HL          ;előtte a szabad szg-ek
C73F  10 FD       DJNZ C73E        ;
C741  4F          LD   C,A         ;BC: átlépendő szg-ek sz.
C742  B7          OR   A           ;    =0?
C743  7E          LD   A,(HL)      ;a legalsó szabad szegm.
C744  28 06       JR    Z,C74C     ;(nem kell átrendezni)
C746  54          LD   D,H         ;
C747  5D          LD   E,L         ;DE: kiutalt szg. címe
```

```
C748  2B          DEC  HL          ;HL: felső perifériaszg.
C749  ED B8       LDDR             ;feltolja a prifériáknak
                                    kiutalt szegmenseket
C74B  12          LD   (DE),A      ;eléjük a most kiutalt
C74C  4F          LD   C,A         ;C=kiutalt szg. száma
C74D  11 00 40    LD   DE,4000     ;DE=16 kbyte használható
C750  AF          XOR  A           ;A=0 státusz
C751  C9          RET              ;
```

A RAM-szegmensek táblázatának a szerkezete tehát a következő:

— felülről (ABD0H-tól) lefelé a rendszer által igénybevett szegmenseket találjuk bejegyezve;

— a legalsó (félig használt) elemre (BF9A/BH) mutat, ez alatt vannak a szabad szegmensek — (BF9F);

— alattuk először a perifériáknak kiutalt szegmenseket — (BFA1) db;

— majd a felhasználóknak kiutalt szegmenseket — (BFA0) db

találjuk.

Ezt a szerkezetet jól példázza az ASMON assembler és monitorprogram beolvasása és elindítása után kialakult helyzet.

```
ABC8  00 00 FB FC F9 FA FD FE
ABD0  FF DD B1 00 00 00 00 00


BF98  AB F9 CF AB C9 AB 00 01
BFA0  02 02 02 08 00 71 5C 66
```

A rendszerváltozó-terület és a RAM-tábla DUMP-ja alapján megállapíthatjuk, hogy a rendszer 2 szegmenst vett igénybe: FFH-t és FEH-t.

Az EXOS határ az FEH szegmensben van. Az egyetlen szabad szegmens az ez alatti FDH. A többit már kiutalta az EXOS:

— kettőt a perifériáknak (pontosabban a program beolvasásakor és elhelyezésekor a TAPE-nek), az F9H és FAH szegmenseket;

— ugyancsak kettőt a felhasználónak (ami most az ASMON program); az FBH és FCH szegmenseket.

Alapállapotú gépnél a táblázat szabad RAM-jai nagyság szerinti sorban állnak és a rendszer is sorban utalja ki azokat az egymás utáni kérésekre. Ezt illusztrálja a következő program, amelynek gépi kódú rutinja meghívja az EXOS 24 funkciót és annak kimenő adatait pufferbe tölti:

```
EXOS 24          ;szegmensigénylés
LD  HL,0180H     ;puffercím
LD  (HL),A       ;hibakód pufferbe
INC HL           ;
LD  (HL),C       ;szegmensszám
INC HL           ;
LD  (HL),E       ;méret LO
INC HL           ;
LD  (HL),D       ;méret HI
RET
```

Maga a program ciklikusan újabb és újabb szegmenseket kér, majd kiírja az EXOS-tól kapott paraméterek értékét:

```
LIST
  100 ALLOCATE 100
  110 CODE M=HEXA("F7,18,21,80,01,77,
      23,71,23,73,23,72,C9")
  120 LET S=1
  130 PRINT S;". hivas:"
  140 CALL USR(M,0)
  150 LET KOD=PEEK(384)
  160 IF KOD>0 AND KOD<>127 THEN GOTO
      230
  170 PRINT "A kapott szegmens szama:"
      ;PEEK(385)
  180 IF KOD=127 THEN PRINT "
      Megosztott!"
  190 PRINT "A felhasznalhato bajtok
      szama:";PEEK(386)+256*PEEK(387)
  200 PRINT
  210 LET S=S+1
  220 GOTO 130
  230 PRINT "Elfogyott a memoria!"
  240 END
ok
```

A futtatás eredménye (a vizsgált kiépítésű gépnél):

```
START
 1 . hivas:
A kapott szegmens szama: 249
A felhasznalhato bajtok szama: 16384

 2 . hivas:
A kapott szegmens szama: 250
A felhasznalhato bajtok szama: 16384

 3 . hivas:
A kapott szegmens szama: 251
A felhasznalhato bajtok szama: 16384

 4 . hivas:
A kapott szegmens szama: 252
A felhasznalhato bajtok szama: 16384

 5 . hivas:
A kapott szegmens szama: 253
A felhasznalhato bajtok szama: 16384

 6 . hivas:
A kapott szegmens szama: 254
A felhasznalhato bajtok szama: 16384

 7 . hivas:
A kapott szegmens szama: 255
Megosztott!
A felhasznalhato bajtok szama: 5063

 8 . hivas:
Elfogyott a memoria!
ok
```

Belépési pont: C78EH
Bemenő adatok:
C: a felszabadítandó szegmens száma
AF': az EXOS hívási státusz (l. pl. EXOS 24)

Ezzel a hívással az EXOS 24 segítségével lefoglalt konkrét szegmenst lehet felszabadítani a rendszer vagy más felhasználó számára. Hogyan oldja ezt meg az EXOS? Nyilván az előbbi hívásnál megismert rendszerváltozók és RAM-tábla alapján.

Itt még egy mutatót használ a rendszer:

— BF9C/DH tartalma a RAM-szegmensek táblázata alatti cím.

A rendszer mindenekelőtt ellenőrzi, hogy a megosztott szegmens felszabadítását kéri-e a hívó. Ha igen, a művelet egyszerű: 0-t ír a "megosztott szegmens száma" változóba (BF9EH) és a felhasználói határt is 0-ra állítja (BF8F = 90H). Más (teljes) szegmens esetén több lépést kell megtenni:

— megkeresi a táblában a felszabadítandó szegmenst (hibát jelez, ha az nem is tartozott a hívóhoz);
— a kiemelt szegmens helyére lecsúsztatja a táblázat többi elemét;
— a felszabadított szegmens számát újra bejegyzi a szabad szegmensek közé, úgy rendezve a táblát — a szabad szegmensek között —, hogy ne boruljon fel a szegmensszám szerinti sorrend;
— eközben természetesen rendet rak a könyvelésében is, növeli a szabad szegmensek számát és csökkenti a hívónak (felhasználó vagy periféria) kiutaltakét.

A funkcióhívás felhasználása (a természetes eseten kívül, amikor visszaadunk a rendszernek egy már nem használt memóriablokkot) akkor is praktikus lehet, ha nem tetszőleges szegmens lefoglalására van szükségünk, nem arra, amit az EXOS kiutal számunkra az első híváskor.

Gyakran szükséges pl. egy VIDEO-szegmens lefoglalása (FCH—FFH), egy olyan szegmensé, amelyben a rendszertől háborítatlanul dolgozhatunk és munkánk eredményét megjeleníthetjük a képernyőn. A rendszer viszont először a legalacsonyabb számú szegmenseket utalja ki kérésünkre (EXOS 24). Természetesen lehet ismételni a szegmensigénylést, mindaddig amíg videoszegmenst nem kapunk, de ez nagy memóriaterület fölösleges lefoglalását jelenti. Ekkor alkalmazhatjuk a tárgyalt funkcióhívást a fölösleges szegmensek felszabadítására.

A következő gépi kódú rutin ezt a feladatot látja el. Addig ismétli a szegmensigénylést, amíg a kiutalt memóriablokk száma 251-nél nagyobb nem lesz. Eközben a kapott szegmensszámokat letárolja egy ideiglenes pufferbe (0180H-tól). Ha VIDEO-szegmenst kapott, rátér a rutin második részére (L2 címke), amelyben a pufferből visszaolvasott számú szegmenseket sorra felszabadítja:

```
        LD   HL,0180H   ;puffercím
L1      PUSH HL         ;verembe
        EXOS 24         ;szegmensigénylés
        POP  HL         ;puffer
        LD   (HL),C     ;kapott szg. pufferbe
        LD   A,0FBH     ;
        CP   C          ;VIDEO-szegmens?
        JR   C,L2       ;kilép, ha igen
        INC  HL         ;puffer köv. cím
        JR   L1         ;ismétlés

L2      LD   HL,0180H   ;puffercím
L3      LD   C,(HL)     ;szegm.szám pufferből
        LD   A,0FBH     ;
        CP   C          ;VIDEO-szegmens?
        RET  C          ;vége, ha igen
        PUSH HL         ;
        EXOS 25         ;felszabadítás
        POP  HL         ;puffercím
        INC  HL         ;növelése
        JR   L3         ;vissza a következő
                         szegmensre
```

A BASIC betöltőprogram a rutin hívása után ki is írja a RAM-tábla területét:

```
100 ALLOCATE 100
110 CODE M=HEX$("21,80,01,E5,F7,18,
    E1,71,3E,FB,B9,38,03,23,18,F3,
    21,80,01,4E,3E,FB,B9,D8,E5,F7,
    19,E1,23,18,F4")
120 CALL USR(M,0)
200 FOR N=0 TO 7
210   PRINT N;PEEK(43977+N)
220 NEXT
```

A táblázatban látható, hogy a végül is lefoglalva maradt FCH szegmens (decimálisan 252) került legalulra a többi felszabadítása után.

```
START
 0  0
 1  252
 2  249
 3  250
 4  251
 5  253
 6  254
 7  255
ok
```

*EXOS 26: rendszerbővítők lekérdezése*

Belépési pont: C818H
Bemenő adatok:
DE: az utasításszövegre mutat
(BF78H): funkciójelző bájt

A funkció alapfeladata egy utasításszöveg körbeadása a rendszerbővítőknek, lehetőséget adva ezzel a szöveg felismerésére és magyarázó üzenet kiírására vagy a vezérlés átvételére. Ha azonban a BF78H jelzőbájt értéke 0, a rendszer — a szöveg figyelmen kívül hagyásával — azonnal a bővítőkhöz fordul, C = 1 értékkel, hidegindítást várva tőlünk.

A funkció hívás a gépi kódú programjainkból talán kisebb jelentőségű (bár így lehet pl. visszaadni a BASIC rendszernek a kezelést), annál inkább fel kell készülnünk arra, hogy saját (bővítőként megírt és beillesztett) programrendszerünket hívja így az EXOS. Nézzük meg tehát, mik a hívás körülményei.

A funkcióhívás kiadásakor DE-t egy utasításszöveg előtti hosszbájtra kell állítani (a hívó memóriakiosztásának tetszőleges helyén).

Ez a szöveg lehet pl.

a) HELP

b) HELP BASIC

c) BASIC

A három különböző típusú szövegre három különböző reakciót vár a bővítőtől az EXOS:

a) minden megadni kívánt információt írjon ki a bővítő,

b) a HELP-et követő utasításhoz tartozó információt írja ki a bővítő (ha felismeri az adott utasítást),

c) vegye át a vezérlést a bővítő, ha a megadott utasítás hozzá tartozik.

A funkcióhívás végrehajtó rutinjának első szakasza az utasításszöveg előkészítését végzi. Felismeri, ha a szöveg HELP-pel kezdődik (az első szóközig vizsgál), a további (bővítő által vizsgálandó) szövegrészt nagybetűsítve pufferbe tölti (a BF8B/CH tartalma által kijelölt címtől). Ezután hívja meg a bővítőket ténylegesen körbekérdező C8ABH rutint a következő paraméterekkel:

B: a felismerendő utasításszöveg hossza

DE: puffercím (a felismerendő szöveg előtti hosszbájtra mutat)

C: akciókód:

C = 3, ha az eredeti szöveg HELP-pel kezdődött

C = 2 egyébként (parancsszöveg)

Ugyanide lép a rendszer C = 1 értékkel, ha a (BF78H) jelzőbájt értéke 0 volt (és általában C egyéb értékeivel más pontokról is).

Hogyan történik a bővítők lekérdezése?

Első lépésben (a háttérregiszterek tárolása után) a RAM-bővítőket kérdezi le az EXOS a BFC3H báziscím alapján. A bővítők pointerei (cím és szegmens) ugyanolyan láncba vannak fűzve, mint ahogy a periféria- vagy csatornaleíróknál láttuk:

```
C8AB  D9          EXX             ;háttérregiszterek
C8AC  C5          PUSH BC         ;mentése
C8AD  D5          PUSH DE         ;
C8AE  E5          PUSH HL         ;
C8AF  D9          EXX             ;
C8B0  21 C3 BF    LD   HL,BFC3    ;bővítőlánc báziscím
C8B3  0C          INC  C          ;akciókód
C8B4  0D          DEC  C          ; =0?
```

```
C8B5  28 3C       JR    Z,C8F3     ;végére, ha törlődött
C8B7  C5          PUSH  BC         ;
C8B8  CD 47 C6    CALL  C647       ;bővítő pointer
                                    HL: belépési cím
                                    A:  szegmens
C8BB  C1          POP   BC         ;
C8BC  28 09       JR    Z,C8C7     ;ROM-okra, ha vége
                                     a bővítőláncnak
C8BE  E5          PUSH  HL         ;belépési cím
C8BF  CB FC       SET   7,H        ;3. LAP-ra
C8C1  CD 17 B2    CALL  B217       ;bővítő hívása
C8C4  E1          POP   HL         ;báziscím vissza
C8C5  18 EC       JR    C8B3       ;a következő
                                    bővítőre
```

Ha a bővítőlánc végére ért, következik a ROM-szegmensek lekérdezése a ROM-táblázat alapján. A belépési cím itt minden esetben a C00AH, és az átadott paraméterek köre bővül: IY a szóban forgó ROM-szegmensnek kiutalt RAM-terület kezdőcímére mutat (a hidegindítás követésénél láttuk, hogy a példaként vizsgált kiépítésű gép esetén csak a 4. szegmens kért és kapott ilyen puffert).

```
C8C7  2A 9C BF    LD    HL,(BF9C)  ;ROM-tábla címe
C8CA  2B          DEC   HL         ;
C8CB  2B          DEC   HL         ;
C8CC  2B          DEC   HL         ;következő elem
C8CD  E5          PUSH  HL         ;báziscíme verembe
C8CE  0C          INC   C          ;akciókód
C8CF  0D          DEC   C          ; =0?
C8D0  28 20       JR    Z,C8F2     ;végére, ha törlődött
C8D2  D9          EXX              ;
C8D3  E1          POP   HL         ;tábla báziscím
C8D4  2B          DEC   HL         ;
C8D5  7E          LD    A,(HL)     ;ROM-szegmens száma
C8D6  B7          OR    A          ; =0?
C8D7  28 1B       JR    Z,C8F4     ;ugrik, ha tábla vége
C8D9  08          EX    AF,AF'     ;szg.-szám háttérbe
C8DA  2B          DEC   HL         ;kiutalt RAM-puffer
C8DB  7E          LD    A,(HL)     ;szegmenssz áma
C8DC  D3 B1       OUT   (B1),A     ;puffer az 1. LAP-ra
C8DE  2B          DEC   HL         ;puffercím
C8DF  46          LD    B,(HL)     ;LO
C8E0  2B          DEC   HL         ;puffercím
C8E1  4E          LD    C,(HL)     ;HI
C8E2  C5          PUSH  BC         ;
C8E3  FD E1       POP   IY         ;IY: RAM-puffer címe
C8E5  E5          PUSH  HL         ;táblacím verembe
C8E6  D9          EXX              ;paraméterek vissza
C8E7  08          EX    AF,AF'     ;A: ROM szegmense
C8E8  FE FF       CP    FF         ;törölt elem?
C8EA  21 0A C0    LD    HL,C00A    ;belépési cím
C8ED  C4 17 B2    CALL  NZ,B217    ;meghívása
C8F0  18 DC       JR    C8CE       ;vissza a következő
                                    ROM-szegmensre
C8F2  E1          POP   HL         ;veremrendezés
C8F3  D9          EXX              ;háttérregiszterek
C8F4  E1          POP   HL         ;visszaállítása
C8F5  D1          POP   DE         ;
C8F6  C1          POP   BC         ;
C8F7  D9          EXX              ;
C8F8  C9          RET              ;
```

Látható, hogy a bővítők sorra kérdezésének kritériuma C > 0 értéke. Az akciókódot tehát változatlanul kell hagynunk saját bővítőrutinjainkban, ill. — szükség esetén — törlésével leállíthatjuk a további közbelépéseket. Figyelemre méltó az is, hogy a letapogatás a RAM-bővítőkkel indul, tehát egy megfelelően felépített saját bővítő meghatározója lehet a rendszernek.

A rendszerbővítő felépítésével, beláncolásával egy konkrét példa kapcsán foglalkozunk.

Befejezésként felsoroljuk milyen akciókódokkal fordulhat a rendszer a bővítőkhöz:

C = 1: hidegindítás
C = 2: parancsszöveg
C = 3: HELP-szöveg
C = 4: EXOS változó (n > 39 sorszám esetén)
C = 5: hibakód magyarázata
C = 6: modulbetöltés
C = 7: RAM-kijelölés
C = 8: inicializálás

*EXOS 27: csatornapuffer igénylése*

Belépési pont: CD5EH
Bemenő adatok:
DE: az egy szegmensben igényelt puffer mérete
BC: annak a puffernek a mérete, amely szegmenshatárt is átléphet
Kimenő adat:
IX: a kiutalt RAM-terület báziscíme (a puffer fölé mutat); a puffer szegmense az 1. LAP-on van
Ez a funkcióhívás csak a perifériakezelőkből, az EXOS 1—11 kezelése közben meghívott végrehajtó rutinokból adható ki. Igényli a munkaterület (EXOS által elvégzett) előkészítését, ezért nem hívhatjuk felhasználói programunkból sem közvetlenül, sem az EXOS-on keresztül.

A BC-ben megadott pufferméretet csak akkor veszi figyelembe, ha a szóban forgó periféria leírójának FLAG eleme 1 (a beépített perifériák közül ez csak a VIDEO esetén van így). Erre tekintettel kell lenni, ha esetleg saját perifériánkban vállalkozunk a szegmenshatárt átlépő pufferterület kezelésére.

A funkcióhívás az igényelt RAM-terület egyszerű lefoglalásán kívül (ami az egyszerű EXOS határ mozgatástól a meglehetősen összetett átrendezésekig, akár egész szegmensek áttöltéséig terjedhet) létrehoz egy leírót is (amint arról a csatornaműveletek — EXOS 1—11 — előkészítésénél már volt szó).

*EXOS 28: hibakód értelmezése*

Belépési pont: C93AH
Bemenő adatok:

A: a magyarázandó hibakód

DE: a puffer címe, ahová az értelmező szöveget töltheti a program

A funkcióhívás B-ben a magyarázandó hibakóddal és C = 5 akciókóddal (Hibakód magyarázata) meghívja a rendszerbővítőket sorra kérdező (az EXOS 26-nál megismert) rendszerrutint. Mint ott láttuk, az EXOS addig hívja a rendszerbővítőket, amíg az egyikből C = 0-val nem tér vissza, vagy amíg a sor végére nem ér.

Ha tehát egy bővítő magyarázni kívánja az adott hibakódot, törölnie kell a C regisztert. A DE regisztert az adott értelmező szöveget tartalmazó memóriára állítja (a szöveg előtti hosszbájtra), míg B-ben az üzenet szegmensszámát adja vissza. A bővítőből való visszatérés után az EXOS rutin gondoskodik a magyarázó szöveg felhasználói pufferbe töltéséről, Visszatérés előtt DE-t a belépési értékre állítja vissza, tehát az üzenet hosszára mutat.

Ezt a funkciót egyszerűen fel lehet használni az EXOS hibakódok magyarázó szövegeinek kiíratására. Ehhez először egy puffercímet adunk meg (példánkban 0180H), A-ba töltjük a felhasználó által megadott magyarázandó kódot, majd meghívjuk az EXOS 28 rutint. A kapott üzenetet a blokkírás rendszerrutinnal íratjuk ki (ehhez C-be a hosszat kell írnunk, DE-t a szöveg első karakterére állítjuk). A szöveg rendezettsége érdekében a tételeket kocsivissza, soremelés karakterekkel zárjuk le:

```
LD   DE,0180H ;üzenetpuffer címe
LD   A,L      ;A: hibakód
EXOS 28       ;értelmezésre
LD   A,(DE)   ;szöveg hossza
LD   B,0      ;
LD   C,A      ;BC=szöveg hossza
INC  DE       ;DE: első karakterre
LD   A,0      ;EDITOR csatorna
PUSH AF       ;csatornaszám verembe
EXOS 8        ;blokkírás
POP  AF       ;A: csatornaszám
PUSH AF       ;
LD   B,0AH    ;írandó karakter
EXOS 7        ;kiírás (soremelés)
POP  AF       ;csatornaszám
LD   B,0DH    ;írandó karakter
EXOS 7        ;(kocsivissza)
RET           ;
```

A rutint betöltő és felhasználó BASIC program sorra kéri az EXOS-tól a hibakódok magyarázatát. A 4. Függelék tartalmazza a futás eredményét német, ill. angol módban (vagyis az EXOS által értelmezett hibakódok magyarázatát).

```
100 ALLOCATE 100
110 DEF H$(N)=CHR$(INT(N/16)+48-7*
    (INT(N/16)>9))&CHR$(MOD(N,16)+
    48-7*(MOD(N,16)>9))
120 CODE M=HEX$("11,80,01,7D,F7,1C,
    1A,06,00,4F,13,3E,00,F5,F7,08,
    F1,F5,06,0D,F7,07,F1,06,0A,F7,
    07,C9")
130 FOR L=255 TO 207 STEP-1
140   PRINT L;"(";H$(L);")  ";
150   CALL USR(M,L)
160 NEXT
```

*EXOS 29: modul betöltése*

Belépési pont: C95FH
Bemenő adatok:
DE: a felhasználó pufferének címe (ahová a modul fejrészét töltheti a rendszer)
B: a csatorna száma
Kimenő adatok:
A = 0, ha az EXOS (vagy egy bővítő) betöltötte a modult
A = ECH (".NOMOD"), ha a betöltött modul "File vége" típusú
A = EEH (".TYPE"), ha az (egyébként ENTERPRISE formátumú) modul típusbájtját nem ismerte föl sem az EXOS, sem egy rendszerbővítő
A = EFH (".ASCII"), ha a modul nem ENTERPRISE formátumú (vagy típusbájtja 0), ill. megfelelő hibakód, ha beolvasási vagy elhelyezési hiba történt
B: a modul típusa, ha még nem történt meg a beolvasás
A funkcióhívást a megfelelő csatorna (pl. az alapértelmezésben a TAPE perifériához rendelt 6AH olvasásra való megnyitása után adhatja ki a felhasználó egy filesoron következő moduljának beolvasására.

Az EXOS rutin a modult — típusától függően — vagy maga tölti be, vagy a rendszerbővítőket hívja C = 6 (Modul betöltése) akciókóddal. Ha azok sem töltik be, a felhasználó visszakapja a vezérlést. Az ENTERPRISE file-ok szerkezetét, típusait, betöltésüket az EXOS leírás tárgyalja igen részletesen.

Az utolsó 5 EXOS hívás (30.—34.) belépési pontja tulajdonképpen a 0. szegmensen van, de itt csak minimális előkészítés történik. A tényleges végrehajtó rutinok ezekben az esetekben az 1. szegmensen találhatók.

*EXOS 30: áthelyezhető modul betöltése*

Belépési pont: 0/CB2CH
Végrehajtó rutin: 1/E52BH
Bemenő adatok:
B: csatornaszám
DE: kezdőcím a betöltéshez

A funkció egy adott modultípus (2-es) betöltését végzi. Akkor szükséges a hívása, ha az előző (EXOS 29) hívás nem töltötte be a modult és B = 2 típuskóddal tért vissza. A pufferben visszakapott fejrész 2. (LO) és 3. (HI) bájtja a betöltendő modul méretét adja. A felhasználónak kell gondoskodnia arról, hogy megadott kezdőcím fölött elférjen a betöltött egység. A betöltés után azonnal meg kell hívni az új programot az inicializálás érdekében.

A további funkcióhívások az 1 Hz-es megszakítások által léptetett óra/naptár rendszer írását, olvasását végzik.

*EXOS 31: az óra beállítása*

Belépési pont: 0/CB3FH
Végrehajtó rutin: 1/E43CH
Bemenő adatok: (BCD kódban):
C: óra (0—23)
D: perc (0—59)
E: másodperc (0—59)
A funkcióhívás mindenekelőtt ellenőrzi a bevitt értékek érvényességét, majd a rögzített rendszerváltozókba tölti az óra, perc, másodperc adatokat:

```
E43C  7B           LD   A,E          ;A=másodperc
E43D  06 60        LD   B,60         ;max. érték+1
E43F  CD 17 E5     CALL E517         ;A<60? (hiba, ha nem)
E442  7A           LD   A,D          ;A=perc
E443  06 60        LD   B,60         ;max. érték+1
E445  CD 17 E5     CALL E517         ;A<60?
E448  79           LD   A,C          ;A=óra
E449  06 24        LD   B,24         ;max érték+1
E44B  CD 17 E5     CALL E517         ;A<24?
E44E  F3           DI                ;
E44F  ED 53 72 BF  LD   (BF72),DE    ;az adatokat
E453  79           LD   A,C          ;   beírja a
E454  32 74 BF     LD   (BF74),A     ;rendszerváltozókba
E457  FB           EI                ;
E458  AF           XOR  A            ;A=0 státusz
E459  C9           RET               ;
```

Az ellenőrzéshez felhasznált rutin azt vizsgálja, lehet-e az A regiszter tartalma BCD adat, majd azt, hogy az érték kisebb-e, mint a B-ben megadott határérték. Ha bármelyik kérdésre nemleges választ kap, hibakóddal (EBH, .TIME) tér vissza, de nem az órabeállító rutinhoz (azt a visszatérési címet kiemeli a veremből), hanem közvetlenül az EXOS főághoz.

Nézzük meg ezt a rutint is, hiszen a gépi kódú programozási gyakorlatban is szükséges lehet az ilyen jellegű adatok ellenőrzése:

```
E517  F5          PUSH AF          :ellenőrzendő érték
E518  FE A0       CP   A0          :1. "karakter" BCD?
E51A  30 0A       JR   NC,E526     :hiba, ha nem
E51C  E6 0F       AND  0F          :2. "karakter"
E51E  FE 0A       CP   0A          :BCD?
E520  30 04       JR   NC,E526     :hiba, ha nem
E522  F1          POP  AF          :ellenőrzendő érték
E523  B8          CP   B           :A<max.érték+1?
E524  D8          RET  C           :vissza, ha igen

Ha hibás volt az érték:

E525  F5          PUSH AF          :
E526  F1          POP  AF          :
E527  F1          POP  AF          :visszatérési cím
                                       törlése
E528  3E EB       LD   A,EB        :'.TIME' hibakód
E52A  C9          RET              :vissza az EXOS-ba
```

Az EXOS 30 funkció végrehajtó rutinjához visszatérve, láthatjuk, hogy az óraadatokat tartalmazó rendszerválozók (az FFH szegmensben):

49010 (BF72H): másodperc
49011 (BF73H): perc
49012 (BF74H): óra

Ezen címek ismeretében az óraadatok közvetlenül is beállíthatók. Ügyelnünk kell a BCD formára!
Például: a 23 órát a

POKE 49012,35

utasítással írhatjuk be.

Az ilyen módon bevitt adatokat nem ellenőrzi a rendszer, ez lehetőséget ad egészen szokatlan időértékek beállítására is.

*EXOS 32: az óra olvasása*

Belépési pont: 0/CB40H
Végrehajtó rutin: 1/E424
Kimenő adatok (BCD kódban):
C: óra (0—23)
D: perc (0—59)
E: másodperc (0—59)

Az igen egyszerű végrehajtó rutin az előbb megismert rendszerváltozók értékeit olvassa a regiszterbe:

```
E424  F3          DI              ;megszakítás letiltása
E425  ED 5B 72 BF LD   DE,(BF72)  ;E=másodperc
                                   D=perc
E429  3A 74 BF    LD   A,(BF74)   ;A=óra
E42C  4F          LD   C,A        ;C=óra
E42D  FB          EI              ;megsz. engedélyezés
E42E  AF          XOR  A          ;A=0 státusz
E42F  C9          RET             ;
```

Az egyetlen momentum, amire itt fel kell hívni a figyelmet az, hogy a kiolvasás letiltott megszakítások mellett történik. Ez szükséges, hiszen az értékeket a megszakításrendszer lépteti (az éppen futó programtól függetlenül), és ha éppen két érték olvasása közben lépne az óra, az adatsor következetlen lenne (előfordulhatna pl., hogy az előző óra 59. perce és a közben átfordult következő óra lenne a kiolvasott érték).

Az óraadat kiolvasására a BASIC TIMES változó a legjobb példa, a közvetlen kiolvasás lehetőségét pedig a megszakítás-programozásnál használjuk ki ("Óra az állapotsorban").

*EXOS 33: a dátum beállítása*

Belépési pont: 0/CB3DH
Végrehajtó rutin: 1/E45AH
Bemenő adatok: (BCD kódban)

C: év (0—99)
D: hónap (1—12)
E: nap (1—a beállított hónapnak megfelelő max. érték)

A végrehajtó rutin feladata annyival összetettebb az EXOS 31-ben megismertnél, hogy

— a hónap és nap értékeire a 0-t sem fogadhatja el
— a nap értékére elfogadható maximális adatot az aktuális hónap alapján külön meg kell határoznia.

```
E45A  79          LD   A,C        ;A=év
E45B  06 FF       LD   B,FF       ;nincs felső határ
E45D  CD 17 E5    CALL E517       ;BCD adat?
E460  7A          LD   A,D        ;A=hónap
E461  06 13       LD   B,13       ;max. érték+1
E463  CD 14 E5    CALL E514       ;0<A<13?
E466  CD E5 E4    CALL E4E5       ;B=a hónap napjainak
                                     száma
E469  7B          LD   A,E        ;A=nap adat
E46A  04          INC  B          ;max. érték+1
E46B  CD 14 E5    CALL E514       ;0<A<max+1?
E46E  F3          DI              ;
E46F  ED 53 75 BF LD   (BF75),DE  ;az adatokat
E473  79          LD   A,C        ; beírja a
E474  32 77 BF    LD   (BF77),A   ;rendszerváltozókba
E477  FB          EI              ;
E478  AF          XOR  A          ;A=0 státusz
E479  C9          RET             ;
```

Az értékek ellenőrzéséhez itt is az EXOS 31-ben megismert rutint alkalmazza a rendszer, csak a hónap és nap értékeknél ezelőtt még 0-ra is ellenőriz.

A naptár adatait tároló rendszerváltozók ezek szerint:

49013 (BF75H): év (00-tól; a BASIC ehhez még 1980H-t hozzáad, így a naptár kezdőéve 1980)

49014 (BF76H): hónap

49015 (BF77H): nap

Sajátos következetlenség a rendszer részéről, hogy míg bevitelkor — természetesen — tiltja a 0. hónap, 0. nap beírását, addig ő maga megfeledkezik az értékek inicializálásáról (ezért indul a naptár 1980:00:00-ról).

Programozástechnikailag érdekes feladat a hónap napjainak megállapítása, különös tekintettel a szökőévekre. Kövessük hát — az utolsó elemzett EXOS rutinként — a rendszer módszerét.

A különböző hónapok napszámait egy táblázatban tárolja a program (E502H-tól): a hónap sorszámának megfelelő táblaelem tartalmazza az értékeket. Az adatsor készítésekor csak arra kellett ügyelni, hogy a sorszám BCD érték, így a szeptemberi adat a 9., az októberi viszont a 16. kell legyen (10H = 16Dec).

Külön kell figyelni a februári értékre, ami általában 28, de szökőévben (4-gyel osztható évszám esetén) 29 lesz. Ezt az esetet egy ügyes fogással figyeli a rendszer. A szökőévek:

— Ha az évtizedek száma páros (BIT 4,D = 0), akkor BIT 1,D = BIT 0,D =0 (0, 4, 8). Ebben az esetben az AND 13H maszkolás tehát nullát eredményez.

— Ha az évtizedek száma páratlan (BIT 4,D = 1), akkor BIT 1,D = 1 és BIT 0,D = 0 (2,6). Ebben az esetben az előbbi maszkolás eredménye: 12H. Ezt lehet 0-ba billenteni XOR 12H-val. Ezek után már lépésről lépésre követhető a rendszer működése:

```
E4E5  D5          PUSH DE        ;adatok verembe
E4E6  C5          PUSH BC        ;  "       "
E4E7  5A          LD   E,D       ;hónap
E4E8  16 00       LD   D,00      ;DE=hónap-sorszám
E4EA  21 01 E5    LD   HL,E501   ;tábla elé mutat
E4ED  19          ADD  HL,DE     ;HL=táblacím
E4EE  79          LD   A,C       ;A=év
E4EF  E6 13       AND  13        ;0, ha páros évti-
                                   zed és szökőév
E4F1  28 02       JR    Z,E4F5   ;
E4F3  EE 12       XOR  12        ;0, ha páratlan év-
                                   tized és szökőév
E4F5  D6 01       SUB  01        ;C, ha szökőév
                                  NC egyébként
E4F7  7E          LD   A,(HL)    ;táblából napok sz.
E4F8  CB 7F       BIT  7,A       ;"február" jelzőbit?
E4FA  28 02       JR    Z,E4FE   ;marad az érték, ha
                                     nem február
E4FC  CE A8       ADC  A,A8      ;80+A8+CY=28 vagy 29
E4FE  C1          POP  BC        ;adatok vissza
E4FF  47          LD   B,A       ;B: a hónap napjai-
                                     nak száma
E500  D1          POP  DE        ;adatok vissza
E501  C9          RET            ;
```

```
E502  31  ;01: január
E503  80  ;02: február (jezőbit beállítva)
E504  31  ;03: március
E505  30  ;04: április
E506  31  ;05: május
E507  30  ;06: június
E508  31  ;07: július
E509  31  ;08: augusztus
E50A  30  ;09: szeptember
E50B  00  ;0A: -
...
E510  00  ;0F: -
E511  31  ;10: október
E512  30  ;11: november
E513  31  ;12: december
```

*EXOS 34: a dátum olvasása*

Belépési pont: 0/CB3EH
Végrehajtó rutin: 1/E430H
Kimenő adatok: (BCD kódban)
C: év (0—99, 1980-tól számítható)
D: hónap
E: nap
A funkcióhívás végrehajtása tökéletesen analóg az idő olvasásával (EXOS 32), csak a kiolvasott rendszerváltozók módosulnak az előző pontnak megfelelően.

## 2.4 A megszakítások programozása

### 2.4.1 A megszakítási rendszer

A megszakítások programozása — az operációs rendszer megszakítási rutinjainak kisebb-nagyobb módosítása — nem elemi, de mindenképpen igen hálás területe a gépi kódú programozásnak. A gép szokásos üzeme alatt, láthatatlanul a háttérből kiváltott képi hatások vagy hanghatások egyrészt igen vonzóak, másrészt gyakran egyedüli megoldásai egy-egy feladatnak (l. pl. STOPPER).

Sok személyi számítógépen találkozhatunk olyan — nyilvánvalóan a megszakítási rendszer által működtetett — hatásokkal, mint pl. a munka közben folyamatosan járó (és a képernyő adott helyén megjelenő) digitális óra, vagy a munkánkat kellemesen kísérő — de attól teljes egészében független — háttérzene stb. Egy

ilyen megoldás természetesen minőségileg is többet jelent, mint pl. az óraadat egyszerű lekérdezése. Mégis sok felhasználó visszariad az ilyen jellegű feladatoktól, annak ellenére, hogy — leglábbis egy bizonyos szintig — ez sem bonyolultabb, mint a gépi kódú programozás más területei. Szerencsére az ENTERPRISE — mint látni fogjuk — szinte tálcán kínálja a lehetőséget a saját megszakítási rutin beillesztésére. Igaz ugyanakkor (és ez némileg indokolja az idegenkedést), hogy a belső megszakítási rendszer módosítása, a beépített rutinok kiegészítése, esetleg megkerülése az operációs rendszer mélyebb ismeretét igényli. Ezért mindenekelőtt tekintsük át az EXOS megszakításkezelését:

— milyen megszakításokat kezel a rendszer,
— milyen feladatokat lát el eközben,
— hol ad lehetőséget a felhasználónak a belenyúlásra, beavatkozásra (ez számunkra a legfontosabb).

A gép bekapcsolásakor, inicializálásakor az EXOS a végrehajtott legelső utasításokkal az 1-es megszakítási módba kapcsolja a CPU-t:

```
C000   F3       DI
C001   ED 56     IM 1
       ...
```

Ezt az üzemmódot sem az operációs rendszer, sem a BASIC értelmező nem változtatja meg a működés során. Saját programunkkal beállíthatnánk ugyan az IM0-s vagy IM2-es módot is, de a hardvertámogatást ehhez megadó speciális interfész nélkül ennek nyilván nem lenne értelme.

A gép működése során tehát a processzor olyan üzemmódban dolgozik, amelyben bármilyen megszakításkérésre egy RST 38H utasítás végrehajtásával reagál:

Az éppen végrehajtott utasítást követő cím — mint visszatérési cím — PUSH utasítással történő elmentése után a 0038H (decimálisan 56) címen folytatja a végrehajtást.

Mielőtt elemeznénk a rutin működését, ennél a pontnál álljunk meg néhány megjegyzésre. A 0038H cím *RAM-területen* van (a PAGE-0-ra lapozott F8H szegmensben). A memória tartalma tehát tetszés szerint módosítható. Átírhatjuk mindjárt az első bájtot (értéke alaphelyzetben 245, azaz PUSH AF), pl. RET-re (201). Próbáljuk meg beírni:

```
POKE 56,201
```

Ezt mindenesetre akkor tegyük, ha nincs a gépben féltett adatunk, programunk, a hatás ui. meglehetősen kellemetlen. A rendszer nem hajthatja végre a megszakítási rutinokat — azonnal visszairányítjuk —, nem kezeli a billentyűzetet a szokásos módon, azaz egyszerűen nem tudunk beírni semmilyen utasítást. Szerencsére — hacsak nem változtattuk meg a rendszermemória kényesebb területeit — a RESET gomb még melegindítást vált ki: föléled a rendszer.

Az mindenesetre nyilvánvaló, hogy ezen a ponton eltéríthetjük a megszakítás-

kezelést. Akár az egész megszakítási rendszert sajátunkkal helyettesíthetjük (természetesen a játékszabályok betartásával). Azt is megmutatta ez a kísérlet, hogy a rendszer a megszakításkezelés leállítása után sem omlik össze, ép marad a kijelzés, sőt mint erről meggyőződhetünk, ha a fenti utasítást programban adjuk ki a BASIC program is tovább működik, leglábbis a legtöbb utasítás végrehajtható. Természetesen nem várhatunk működő órát az adott körülmények között, sőt a GET és INPUT sem működik. Hasznos lehet viszont ez az utasítás, ha egy időigényes programrészlet előtt adjuk ki, majd a rutin lefutása után visszaállítjuk a memóriacím eredeti tartalmát. Nyilvánvaló, hogy a program futása felgyorsul, ha a másodpercenként sokezer (kizárólag a megszakításokat kezelő) utasítást nem kell elvégeznie. Próbálja ki ezt pl. a következő egyszerű program futtatásával:

```
100 !POKE 56,201
110 FOR N=1 TO 500
120   PRINT AT 1,1:N
130 NEXT
140 POKE 56,245
```

A program futási ideje ebben a változatban kb. 40 s. Ha a megszakításokat leállítjuk

```
100 POKE 56,201
```

átírással, a végrehajtás kb. 30 s-ig tart.

A kb. 25%-os időmegtakarításon túl további előny, hogy a megszakítások kizárásával pontosabbá válik a BASIC utasítások végrehajtási ideje, megszűnik a futási idő esetlegessége.

Ez a módszer használható akkor is, amikor a rendszerbe való beavatkozás (pl. egy több bájtos adat vagy cím kiolvasása, ill. beírása) miatt le kellene tiltanunk a megszakításokat. Józan ésszel azt gondolnánk, ehhez elég egy saját gépi kódú rutinban a 'DI' (Disable megszakítási) Z80-as utasítást kiadni. Ez azonban nem vezet eredményre, hiszen az EXOS a futás során a megszakítási letiltást rendszeresen feloldja. Ezt a feladatot is a fenti "ál-letiltás" fogással oldhatjuk tehát meg legegyszerűbben.

Térjünk ezek után vissza az EXOS megszakításkezeléséhez. Milyen főbb feladatai vannak ennek a rendszernek?

— A processzor pillanatnyi állapotának tárolása (regiszterek, státusz).
— A megszakítást okozó forrás megkeresése.
— A forrásnak megfelelő feladatok elvégzése. Ezek egyrészt általános feladatok, amiket az EXOS végez el, de a beépített és a felhasználó által bevezetett perifériák is meg kell, hogy kapják a szót.
— A kezelés elvégzése után a tárolt paraméterek visszaállítása, a megszakítás elfogadásakor automatikusan letiltott további megszakítások engedélyezése, visszatérés a félbehagyott programhoz.

Hogyan látja el ezeket a funkciókat az EXOS?

A 0038H-as címen induló rutin — bebillentett átvitelbittel — az EXOS leggyakrabban használt rutinján, a RST 30-on folytatja futását:

```
0038  F5          PUSH AF          ;A regiszter elmentése
0039  37          SCF              ;CY=1, ez különbözteti
                                    meg a
003A  18 09       JR   0045        ;többi EXOS hívástól


0045  DB B3       IN   A,(B3)      ;P3 pillanatnyi SZG-e
0047  32 55 00    LD   (0055),A    ;átmeneti tárolóba
004A  3E 00       LD   A,00        ;a 0. szegmenst
004C  D3 B3       OUT  (B3),A      ;lapozza a 3.LAP-ra
004E  C3 10 C4    JP   C410        ;rendszer-ROM-ra ugrik
```

A 0. ROM-szegmens C410H címétől egy szakaszon együtt folyik a megszakításkezelés elökészítése az 'EXOS n' végrehajtásával.

Mivel a megszakításkezelés során beírt átvitelbittel ér ide a program (l. 0039H), a C454H pontnál a végrehajtás elválik az EXOS főágtól.

A 0. szegmens C4B2H címén induló tényleges megszakításkezelő mindenekelőtt a verembe tárolja a még biztonságba nem helyezett regisztereket:

```
C4B2  C5          PUSH BC          ;a további regiszterek
C4B3  D5          PUSH DE          ;verembe mentése
C4B4  D9          EXX              ;
C4B5  C5          PUSH BC          ;
C4B6  D5          PUSH DE          ;
C4B7  E5          PUSH HL          ;
```

A továbbiakban a saját igényeinek megfelelően állítja be a visszatérési szakaszt:

```
C4B8  3E B7       LD   A,B7        ;"OR A" kódja
C4BA  32 56 00    LD   (0056),A    ;a visszatérési ágba
```

Bármilyen megszakítás történt, feltétel nélkül növeli 1-gyel a RANDOM—IRQ rendszerváltozót (ez a változó véletlenszámként való felhasználásának alapja):

```
C4BD  21 EC BF    LD   HL,BFEC     ;RANDOM_IRQ címe
C4C0  34          INC  (HL)        ;növeli a változót
```

A következő szakaszban a kezelő a megszakítást kiváltó okkal foglalkozik. Az EXOS leírás szerint a rendszerben alapvetően négyféle megszakítás történhet:

— hangmegszakítás,
— 1 Hz-es megszakítás,
— videomegszakítás (50 Hz),
— külső (hálózat) megszakítás.

Ezek mindegyikét a DAVE chip kezeli, közvetíti. Az egyes források megszakításkérését külön engedélyezhetjük, vagy tilthatjuk a B4H-es kimeneti ponton. Az EXOS ezt az IRQ—ENABLE—STATE rendszerváltozó megfelelő beírásával végzi el (ennek az eljárásnak a megváltoztatása könnyen a rendszer összeomlását okozhatja).

Amennyiben bármelyik engedélyezett forrás megszakítást kér, a DAVE egységesen továbbítja ezt a mikroprocesszor felé. A megszakítás lehetséges forrásait a szoftverből beolvasható B4H-es pont egyes bitjei reprezentálják:

BIT 1: hang
BIT 3: 1 Hz
BIT 5: video
BIT 7: külső

Adott esetben az a forrás váltotta ki az IRQ-t (Interrupt Request — megszakításkérés), amelyik bit értéke 1. Ezt vizsgálja a rutin folytatása:

```
C4C1  DB E4       IN   A,(B4)      ;a megszakítás forrása
C4C3  E6 AA       AND  AA          ;10101010 maszkolás
C4C5  47          LD   B,A         ;regiszterbe
C4C6  ED 44       NEG              ;-A
C4C8  A0          AND  B           ;egyetlen 1-es marad
C4C9  57          LD   D,A         ;D: megszakítási kód
```

A továbbiakban tehát a D regiszterben áll rendelkezésünkre az IRQ forrásának információja az előbbiek szerint.

Ennek alapján a megszakításkezelő minden videomegszakításnál (azaz másodpercenként 50-szer) aktualizálja a keretszínt a BORD-VID rendszerváltozó beírásával és a NICK-chip FIXBIAS regiszterét a BIAS—VID, a MUTE—SND és a SPRITE rendszerváltozók alapján.

```
C4CA  CB 6A       BIT  5,D         ;VIDEO megszakítás?
C4CC  C4 16 C5    CALL NZ,C516     ;rutinra, ha igen
```

Ez az oka annak, hogy — bár a keretszín a 81H (129Dec) ponton közvetlenül is beírható — az így átírt színű keret legfeljebb egy villanásnyira jelenik meg.

Ezután érkezik el a megszakításkezelő ahhoz a ponthoz, amely számunkra talán a legfontosabb. Beolvassa a USER—ISR rendszerváltozót (a rendszerszegmens BFED/EH címének tartalmát) és ha ott nullától különböző értéket talál, azt a felhasználói megszakítási rutin címének tekinti.

```
C4CF  2A ED BF    LD   HL,(BFED)   ;HL= USER_ISR
C4D2  7C          LD   A,H         ;
C4D3  B5          OR   L           ;HL=0000?
C4D4  D5          PUSH DE          ;kód verembe
C4D5  C4 25 B2    CALL NZ,B225     ;felhasználói rutin
                                    meghívása (ha van)
C4D8  D1          POP  DE          ;kód vissza
```

A B225H címen (az EXOS főág részeként) egy JP(HL) utasítás van, tehát a megszakításkezelő lényegében egy CALL USR—ISR hívást hajt végre.

Mivel a USER—ISR értéket tetszés szerint beállíthatjuk — akár BASIC-ből, akár gépi kódú rutinunkban az FF szegmens BFED/EH (decimálisan 49133/4) címén — ez a vizsgálat és elágazás adja a legkényelmesebb lehetőséget saját megszakítási rutinunk rendszerbe illesztésére. Foglaljuk össze tehát, milyen státuszban van a rendszer a rutin meghívásakor, mit használhatunk fel és mit kell megőriznünk.

*Memóriakonfiguráció:*

PAGE-0: az F8H-as RAM nulláslap-szegmens (mint mindig)
PAGE-1: megmarad a megszakítás pillanatában belapozott szegmens, tehát határozatlannak tekinthető
PAGE-2: az FFH-s RAM rendszerszegmens
PAGE-3: a 00H-s ROM-szegmens (ahol az EXOS megszakításkezelő fut)

Ha azt akarjuk, hogy rutinunk vége (RET) után a szokásos módon folytatódjon a megszakításkezelés, ezt a konfigurációt meg is kell őriznünk (vagy vissza kell állítani a visszatérés előtt), kivéve a PAGE-1 állapotát, amit a továbbiakban sem tekint határozottnak az EXOS, és vissza is állítja eredeti állapotát a megszakításkezelés végén. Mindezekből az is következik, hogy a felhasználói megszakítási rutin belépési címének a nulláslapon kell lennie (esetleg PAGE-2-n, a rendszerszegmensben, de itt könnyen ütközhetünk az EXOS-szal).

*Regiszterek:*

Az EXOS a processzor valamennyi regiszterét (a vesszős és az indexregisztereket is) elmentette, mire ehhez a ponthoz ért. Ebből a szempontból tehát szabad kezünk van: egyik regiszter tartalmát sem kell megőriznünk.
Hasznos információt a D regiszter tartalmaz: a fentiek szerint a megszakítás forrására utal. HL már a meghívott rutinra mutat, ezt esetleg szintén fel lehet használni.

A verem tetején a visszatérési cím — esetünkben C4D8H — található. Ezt akár le is emelhetjük innen (POP), ha meg akarjuk kerülni a megszakításkezelés ezután következő részeit. Ezzel mindenesetre körültekintőnek kell lenni, hiszen —mint látni fogjuk — hátra van még a szinte minden felhasználásban nélkülözhetetlen billentyűzetkezelés, másrészt az EXOS-szal megegyező módon kell rendbe tenni kilépés előtt a memóriakonfigurációt, regisztereket stb.

Ide tartozik még, hogy kilépéskor a megszakítások le vannak tiltva és nekünk sem szabad engedélyezni azokat ('EI') a kezelő vége előtt.

Ezen információk birtokában már bátran vállalkozhatunk kisebb megszakítási rutin elkészítésére, beillesztésére (mint arra a továbbiakban adunk is néhány példát). Egyelőre azonban nézzük tovább a ROM-ot, tekintsük át, milyen feladatokat lát még el az EXOS megszakításkezelője.

Minden 1 másodperces megszakításnál lépteti az órát és (ha szükséges) a naptárat:

```
C4D9  AF        XOR  A          ;A=0
C4DA  CB 5A     BIT  3,D        ;1 Hz-es megszakítás?
C4DC  C4 3C CB  CALL NZ,CB3C    ;óra, naptár léptetés,
```

Ezután következik a perifériák lekérdezése: át kívánják-e venni a vezérlést az adott típusú megszakításnál.

```
C4DF  21 C0 BF     LD   HL,BFC0     ;HL: báziscím
C4E2  CD 47 C6     CALL C647        ;a következő láncelem
C4E5  28 19        JR   Z,C500      ;elugrik, ha lánc vége
C4E7  5F           LD   E,A         ;E: a leíró szegmense
C4E8  7E           LD   A,(HL)      ;A: leíró TYPE eleme
C4E9  B7           OR   A           ;A=0?
C4EA  20 F6        JR   NZ,C4E2     ;tovább keres, ha ez
                                     törölt leíró
C4EC  23           INC  HL          ;IRQ_FLAG cime
C4ED  7E           LD   A,(HL)      ;A=IRQ_FLAG
C4EE  A2           AND  D           ;(D: megszakitási kód)
C4EF  2B           DEC  HL          ;TYPE címe
C4F0  E5           PUSH HL          ;regiszterek
C4F1  D5           PUSH DE          ;   verembe
C4F2  7E           LD   A,(HL)      ;A=0
C4F3  01 E5 53     LD   BC,53E5     ;memóriaellenőrzéshez
C4F6  C4 41 C5     CALL NZ,C541     ;hívja a kezelő megsza-
                                    kitás rutinját, ha kéri
C4F9  D1           POP  DE          ;regiszterek
C4FA  E1           POP  HL          ;   vissza
C4FB  7B           LD   A,E         ;leíró szegmense
C4FC  D3 B1        OUT  (B1),A      ;az 1. LAP-ra
C4FE  18 E2        JR   C4E2        ;következő láncelemre
```

Az eljárás alapja az a leírólánc, amiben az EXOS tárolja a létező perifériák adatait (l. inicializálás). Az első leíróra a BFC0H báziscím (49088D) mutat (pontosabban: BFBFH-n a leírót tároló szegmens száma van, BFBD/EH tartalma a leíró címe a PAGE-1-en).

Ez a cím ugyancsak bázisnak tekinthető: az előtte lévő 3 bájt a következő láncelemre mutat. A lánc mindaddig tart, amíg a következő leíró szegmensszámaként 0-t nem találunk. Ezt a láncot vizsgálja végig a rutin, ugyanúgy, ahogy mi is megtettük ezt a 2.3.1 pontban a LANC programmal.

Tekintsük most át még egyszer a futtatás eredményeként megkapott periférialáncot.

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

74 66 FF 00 20 00 BF 5F 04 00 08 KEYBOARD
85 66 FF 00 08 00 36 4D 04 00 06 EDITOR
93 66 FF 00 80 00 D3 6D 01 00 03 NET
A4 66 FF 00 00 00 A7 6D 01 00 06 SERIAL
B3 66 FF 00 00 00 BB 67 01 00 04 TAPE
C5 66 FF 00 00 00 69 67 01 00 07 PRINTER
D6 66 FF FF 00 00 82 70 00 00 06 EDITOR
E9 66 FF FF 20 00 55 70 00 00 08 KEYBOARD
F9 66 FF 00 20 00 7C 6B 00 00 05 SOUND
00 00 00 00 20 01 81 57 00 00 05 VIDEO
```

Az EXOS leírás megadja az egyes bájtok jelentését (ezzel a 2.5 alfejezetben foglalkozunk részletesebben). Számunkra most a TYPE (0.) és IRQ—FLAG (1.) bájtok a legérdekesebbek. A láncban csak azok a leírók érvényesek, ahol TYPE = 0. Látható, hogy az inicializálás során felvett 0. szegmensbeli KEYBORD és EDITOR kezelőt mintegy felülírta a későbbiek során vizsgált 4. szegmens ugyanilyen nevű perifériája.

A periférialeírók 1. eleme, az IRQ—FLAG az, amelyet a rendszer összehasonlít a megszakítás forrását reprezentáló D regiszterrel. Megegyezés esetén, — ha a periféria igényt tart az adott megszakítás kezelésére — meghívja annak megszakításkezelőjét (ehhez belapozza a periféria szegmensét PAGE-3-ra, majd a belépési pont táblázat 0. címére ugrik). Ez a harmadik (és egyben utolsó) pont, ahol módosíthatjuk az alapértelmezésű megszakítási rendszert. Akár a perifériakezelők paramétereit változtatjuk meg, akár új perifériát veszünk fel a rendszerbe, lehetőségünk van a beépített megszakításkezelés körének szűkítésére, bővítésére.

Tekintsük át az alaphelyzetben (angol és német nyelvű változat) kiépített periférialáncot. A táblázat alapján megállapíthatjuk, hogy

— az 1 Hz-es megszakítás (BIT 3) kezelésére az EDITOR tart igényt
— a videomegszakítás (BIT 5) kezelését a billentyűzet, a hang és a VIDEO periféria kéri,
— a külső megszakításra (BIT 7) — természetesen — a hálózat (NET) regál.

Az EDITOR a megszakításkezelés során gyakorlatilag az adott (német) BASIC verzió belapozottságán őrködik. Számunkra elsősorban a videomegszakítások érdekesek; az, hogy mi minden történik a háttérben, minden ötvened másodpercben.

KEYBOARD (4/DFC2):

— Figyeli a CTRL, SHIFT, ALT, HOLD gombokat, beállítja és kijelzi az állapotsor szélén az üzemmódot.
— Beolvassa a billentyűzetmátrixot. A billentyűkésleltetést és ismétlési gyakoriságot meghatározó DELAY—KEY és RATE—KEY rendszerváltozók (és saját számlálói) függvényében fogad el és vesz fel a pufferbe billentyűt.
— Ha a KEY—IRQ rendszerváltozó értéke 0 (a billentyűleütés szoftvermegszakítást okozhat), a BFF2 címre (FLAG—SOFT—IRQ) jelzőkód ír (STOP esetén 21H, alapbillentyűnél 20H).
— Elfogadott billentyűnél hangot ad, ha a CLICK—KEY rendszerváltozó értéke 0 (ehhez nem használja a 'SOUND' kezelőt, hanem közvetlenül programozza a DAVE-chipet).

SOUND: (0/EBD6):

A hangkezelő a videomegszakítások alatt (tehát másodpercenként 50-szer) aktualizálja a hangparamétereket (a burkológörbéknek megfelelően).

VIDEO (0/D33D):

A videomegszakítás-kezelő az állapotsor kijelzéséről gondoskodik. A sorparaméter-tábla (LPT) 0. sorában beállítja a margókat és a megjelenítendő memóriaterület kezdőcímét. Az ST—FLAG rendszerváltozó (FF/BFDFH cím) értéke szerint:

— 0: IS BASIC üzenet és információk (kezdőcím: FEB8H videocím, ami számunkra az FFH rendszerszegmens BEB8H címe),
— 2AH (42dec): a készítők monogramját tartalmazó üzenet (F230H, ill. FF(B230H),
— egyéb: a kijelzést kikapcsolja a bal margó 3FH-ra való állításával.

A perifériák lekérdezésével véget is ér a hardvermegszakítások kezelése. A

továbbiakban az IRQ—ENABLE—STATE rendszerváltozó alapján aktualizálja a DAVE—chip megszakításengedélyező regisztereit, visszaállítja az EXOS visszatérési szakasz és a regiszterek belépés előtti értékét, majd visszaugrik az EXOS főágra.

```
C500  3A C5 BF     LD   A,(BFC5)    ;IRQ_ENABLE_STATE
C503  B2           OR   D           ;(D: megszakítási kód)
C504  D3 B4        OUT  (B4),A      ;engedélyező reg.be
C506  3E FB        LD   A,FB        ;"EI" kódja
C508  32 56 00     LD   (0056),A    ;a visszatérési ágba
C50B  E1           POP  HL          ;regiszterek
C50C  D1           POP  DE          ;   vissza
C50D  C1           POP  BC          ;
C50E  D9           EXX              ;
C50F  D1           POP  DE          ;
C510  C1           POP  BC          ;
C511  26 F1        LD   H,F1        ;"POP HL" kódjával
C513  C3 6D C4     JP   C46D        ;ugrik az EXOS főágra
```

A főág a belépés előtti memóriakonfiguráció visszalapozása és a regiszterek visszaállítása után még egy megszakítás vonatkozású funkciót is ellát visszatérés előtt: a szoftver-megszakítás kezelését. Ezt bármelyik megszakításkezelő alprogram (pl. billentyűzet) vagy EXOS "n" végrehajtó rutin kiválthatja azzal, hogy 0-tól különböző értéket tölt a rendszerszegmens BFF2H címére (FLAG—SOFT—IRQ). Ha az EXOS a zárószakaszban nullától különböző értéket talál, beírja ezt, mint szoftver megszakítási kódot a CODE—SOFT—IRQ rendszerváltozóba (BFC7H). 0-t ír a FLAG-..be (megteremtve ezzel a következő megszakítás lehetőségét), majd ellenőrzi a nulláslap 3D/3EH címét: a felhasználói szoftvermegszakítási rutin címét (SOFT—SR). Ezt a címet a felhasználó tetszés szerint beállíthatja, pl. az adott BASIC verzióban, alapértéke 00FFH).

Nullától különböző rutincím esetén az EXOS átírja a nulláslapon saját visszatérési szakaszának legvégét: a szokásos C9H ('RET') helyére 18H, E3H, ("JR E3H") kerül. Ezután ugrik a kezelő (a rendszerszegmensen lévő EXOS betéten keresztül) a visszatérési szakaszra. Alaphelyzetben innen végre visszakerül a hívóhoz, míg szoftvermegszakítás esetén egy 'JP SOFT—ISR' utasításra ugrik és a felhasználói szoftver megszakításkezelőn át tér vissza vagy fejezi be a futást hibajelzéssel.

Itt is egyszerű lehetőségünk adódik tehát a beavatkozásra: a 3D/3EH címekre beírhatjuk saját szoftver megszakításkezelőnk címét. Ebben tetszés szerint dönthetünk a további teendőkről. Például a STOP-billentyű lenyomása által kiváltott szoftver-megszakítási esetén (CODE—SOFT—IRQ=21H) egyszerűen figyelmen kívül hagyjuk a megszakítást, míg a többi esetben a beépített kezelőre ugrunk.

Foglaljuk össze azokat a pontokat, ahol beavatkozhatunk a hardvermegszakítások kezelésébe:

— Belépési pont: 0038H
  itt a teljes megszakításkezelés téríthető vissza, vagy saját kezelőre.
— Felhasználói megszakítási rutin. Címe a USER-ISR rendszerváltozóba (FFH szegmens, BFED/EH) írandó.
— A perifériakezelők megszakítási rendszerének módosítása a leíróbeli paraméterek vagy címek átírásával, saját perifériakezelő beláncolásával.

Hasznosítsuk most már néhány példában a megszerzett információkat. Első alkalmazásként lássunk egy elemi, de esetenként igen hasznos beavatkozást.

## 2.4.2 Állapotsor saját használatra

A videokezelő — mint láttuk — minden ötvened-másodpercben kijelzi az állapotsorban a megfelelő memóriaterület tartalmát. A kijelzett információ (billentyűzetüzemmód, programszám stb.) igen hasznos, de egyes esetekben legalább ilyen hasznos lenne saját üzenetünk kiírása. Írhatunk ugyan a másik kijelezhető területre (a készítők monogramjainak helyére) is, de erre az EXOS meglehetősen kényes; védett területnek tekinti. Egyetlen bájt átírása is azzal jár, hogy a RESET gomb egyszeri megnyomása is hidegindítást vált ki. Akadályozzuk meg tehát a videokezelő munkáját, és máris szabadon bánhatunk az állapotsorral. Ez az eddigi ismeretek alapján igen egyszerű: mindössze azt a bitet kell törölnünk a periférialeíróban, ami alapján a kezelő reagál a videomegszakításokra.

Az átírandó cím a periférialánc táblázatából állapítható meg. A VIDEO leíró báziscímét az előző leíró (SOUND) 'NEXT' paramétere adja. Esetünkben ez: 255. szegmens, cím HI = 66H = 102Dec, LO = F9H = 249Dec.
A báziscím tehát

$$BC = 249 + 256 \cdot 102 = 26\,361$$

Az IRQ—FLAG címe: BC + 1, esetünkben 26 362.

A cím tartalma alaphelyzetben 20H (32Dec), azaz BIT 5 = 1 (a kezelő a videomegszakításra reagál).
Írjuk be tehát:

```
SPOKE 255,26362,0
```

és miénk az állapotsor (persze előbb ki-ki ellenőrizze a konkrét címeket saját gépén a fenti lépésekkel). Az így megszerzett lehetőség felhasználására láthatunk példát a következő megszakítás alkalmazásban.

## 2.4.3 Óra az állapotsorban

Sok személyi számítógépen láttunk már olyan programokat, amelyek hatására megjelenik a gép digitális órája a képernyő valamelyik zugában (általában ott, ahol más információt nem zavar), és folyamatosan jár, miközben mi dolgozunk a gépen: programot írunk vagy futtatunk stb.

Ennek a feladatnak a megoldása a megszakítási rendszer ismeretében szinte magától értetődik: be kell építeni egy felhasználói megszakításkezelő rutint, amely az 1 Hz-es megszakítások alatt kiolvassa a gépi órát és — a szokott formátumban — kijelzi a képernyőn. Különösen bátran indul neki a megoldásnak, aki a TIMES függvény (ill. változó) kiértékelését ismerve tudja, hogy az óraadat munkaregiszterbe töltése egyetlen ROM-rutin hívásával elérhető. Senkinek sem javasoljuk azonban ezt a magát szinte kínáló megoldást! Az 1. szegmenst C530H címén indító rutin alkalmazása ugyan problémamentes lenne, de a rutin lelke — az óra-, perc- és másodpercadatokat a C, D és E regiszterbe töltő EXOS 20H rutin — két buktatót

is rejt. A minden EXOS hívásnál fellépő alapproblémát az utolsó példa kapcsán tárgyaljuk, de itt maga a végrehajtó rutin is alkalmazhatatlan megszakításkezelőben. Nézzük meg, miért! Az 1. szegmens E424H címén kezdődő rutin rendkívül egyszerű:

```
E424  F3          DI              ; megszakítás letiltás
E425  ED 5B 72 BF LD   DE,(BF72)  ; D=perc; E=s
E429  3A 74 BF    LD   A,(BF74)   ; A=óra
E42C  4F          LD   C,A        ; C=óra
E42D  FB          EI              ; megszakítás eng.
E42E  AF          XOR A           ; A=0; EXOS vissza-
E42F  C9          RET             ;       téréshez
```

Azonnal szembeötlik az alkalmazást kizáró ok: a rutin vége előtt kiadott 'EI' utasítás. Ez nyilván tilos egy megszakításkezelő kellős közepén. Szerencsére nem akkora ez a rutin, hogy ne írhatnánk be közvetlenül is saját kezelőnkbe. Az így megszerzett adatokkal problémamentesen felhasználhatjuk a TIME$ rutin hátralevő (az óraadatokat füzér alakba konvertáló és a munkaregiszterbe töltő) részét. Ezután már csak az a probléma, hogy hová írjuk ki az eredményt. Átvihetnénk pl. a mindenkori első sorba, amelynek címét a sorparaméter-táblából állapíthatjuk meg (B914/5H). Ez az út járható ugyan, de a képernyő görgetésekor egy-egy sorban "szemétként" megmaradhatnak a régebben odaírt óraadatok. Ez lehet ugyan szórakoztató, de előbb-utóbb bosszantóvá válik.

A legegyszerűbb megoldás a rendszerszegmens B230H címére írni (l. a következő példát), de ezzel elveszítjük a RESET gomb adta biztonságot. Alkalmazzuk tehát az előzőekben tárgyalt fogást: tiltsuk le a video megszakításkezelőt és ezután bármilyen videomemóriát (FCH—FFH szegmensek) megjeleníthetünk az állapotsorban. Mivel kevés helyre van szükségünk, egy érdekes megoldásként felhasználhatjuk a sorparaméter-tábla (LPT) legvégén álló, ki nem jelzett szinkronizáló sorainak cím- és palettabájtjait. Ennek a területnek a végét (BB20H) kell megadnunk munkaregiszterként, ha azt szeretnénk, hogy a TIME$ rutin közvetlenül a pufferbe töltse az órafüzér karaktereit.

Ezek után elkészíthető a megszakításkezelő rutin:

```
USR_ISR ORG 0180H          ;
        BIT  3,D           ; 1 Hz-es megszakítás?
        JR   NZ,IR_1Hz     ;
        RET                ; vissza, ha nem
IR_1Hz  LD   DE,(0BF72H)   ; D=perc; E=s
        LD   A,(0BF74H)    ; A=óra
        LD   C,A           ; C=óra
        IN   A,(0B3H)      ; pillanatnyi 3. LAP
        PUSH AF            ; szegmense verembe
        LD   A,01          ; 1. szegmenst
        OUT  (0B3H),A      ; lapozza a 3. LAP-ra
        LD   HL,(0228H)    ; HL: munkaregiszter
        PUSH HL            ; aktuális érték verembe
        LD   HL,0BB20H     ; VIDEO puffercím
        LD   (0228H),HL    ; mint munkaregiszter
        CALL 0C532H        ; TIMES -> puffer
        POP  HL            ;
        POP  AF            ;
        OUT  (0B3H),A      ; 3. LAP  vissza
        LD   (0228H),HL    ; alapérték vissza
        RET
```

A rutin futását BASIC-ből elő kell készíteni, sőt az egész rutint is érdemes BASIC-ben beírni és indítani. Ezt végzi el a következő program.

```
100 !VIDEO leiro cime
110 !print 102*256+249
120 ! 26361
130 !print speek(255,26362)
140 ! 32 (IRQ_FLAG=20H)
150 !spoke 255,26362,0
160 POKE 47364,24 !PUFFER LSB
170 POKE 47365,251 !PUFFER MSB
180 POKE 47362,35 !bal margo
190 POKE 47363,43 !jobb margo
200 !USR_ISR
210 POKE 540,128
220 POKE 541,1
230 CODE M=HEX$("CB,5A,20,01,C9,ED,
    5B,72,BF,3A,74,BF,4F,DB,B3,F5,
    3E,01,D3,B3,2A,28,02,E5,21,20,
    BB,22,28,02,CD,32,C5,E1,F1,D3,
    B3,22,28,02,C9")
240 CODE MB=HEX$("21,ED,BF,F3,36,80,
    23,36,01,FB,C9")
250 CALL USR(MB,0)
```

A program első része az állapotsor megszerzésének lépéseit mutatja (REM-sorokban) az előző pont gondolatmenete alapján. A 160. sor ténylegesen végrehajtandó, természetesen az adott gépen érvényes címmel. A további sorok az FB18H videocím (FF szegmens, BB18H cím) kijelzésére állítják az állapotsort, a sor margóit pedig 8 karakter kijelzésére.

Ezután történik a felhasználói megszakításrutin beírása a memóriába és beillesztése a megszakításkezelésbe. Itt két dolog magyarázatot igényel:

1. Hol tároljuk a rutint a memóriában?

BASIC programról lévén szó, a legegyszerűbb az lett volna, ha ALLOCATE-tel helyet foglalunk a rutin számára a program előtt, és oda töltjük. Általában ez tökéletes megoldás, de — megszakításrutinról lévén szó — valamit hangsúlyoznunk kell. A BASIC munkaterület elejére elhelyezett gépi rutin nincs teljes biztonságban. Bizonyos esetekben (pl. program fejlesztéskor) a programszöveg visszacsúszhat a lefoglalt területre, felülírva ezzel az operációs rendszer által sűrűn meghívott rutint. Ez szinte biztosan a rendszer összeomlásához vezet. Ezért helyeztük el a rutint a BASIC által nem használt 0180H címtől kezdődően.

2. Hogyan illesztjük be a rutint a megszakítási rendszerbe?

A feladat igen egyszerűnek látszik: be kell írni a USR—ISR változóba (BFED/EH, ill. 49133/4Dec) a rutin címének alsó és felső bájtját (esetünkben 80H, 01H, ill. 128Dec, 1Dec). Valójában ezt nem tehetjük meg két egymás utáni POKE utasítással, mert ha a kettő között megszakításkérés érkezik, az EXOS a félig érvényes című rutinra ugrik. Két megoldást alkalmazhatunk: a megszakítást az átírás idejére letiltó gépi kódú programot, vagy a megszakításokat "eltérítő" BASIC utasításokat:

```
..10 POKE 56,201
..20 POKE 49133,128
..30 POKE 49134,1
..40 POKE 56,245
```

A példaprogramban az első megoldást alkalmaztuk:

```
MB      LD HL,0BFEDH  : USR_ISR változó
        DI            ;
        LD (HL),80H   : LO beírás
        INC HL        ;
        LD (HL),01H   : HI beírás
        EI
        RET
```

A kijelzett óra azonos a TIME$ változó értékével, a TIME utasítással bármikor beállítható. A kijelzés fennmarad akkor is, ha az elindítás után a "hordozó" programot töröljük a tárból, saját programot írunk, futtatunk stb. Ha szükséges, a BASIC állapotsort bármikor visszakaphatjuk, ha visszaállítjuk a nullázott IRQ-FLAG (l. 160. sor) értékét 32-re, újra lehetővé téve ezzel a VIDEO periféria megszakítás-kezeléseit.

## 2.4.4 Stopper

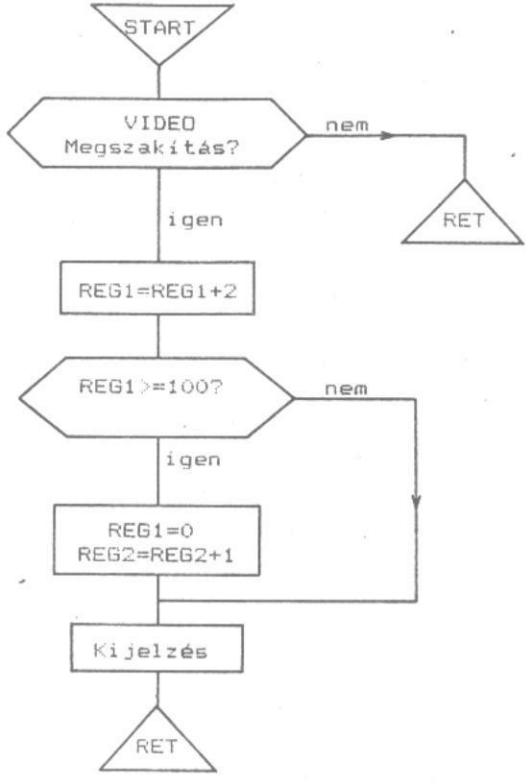
Az előző példán felbátorodva sokakban felmerülhet a kérdés: ha ilyen könnyen beavatkozhatunk a megszakítási rendszerbe, miért elégedünk meg a gép 1 másodperc felbontású belső órájával. Sok játékhoz, reakcióidő vagy futásidő méréshez, esetleg komolyabb oktatástechnikai, mérési alkalmazáshoz nélkülözhetetlen egy sokkal finomabb lépésközű — pl. századmásodperces óra.

Az ENTERPRISE megszakítási rendszerének ismeretében megállapíthatjuk, hogy ha nem is századmásodperces, de 0,02 s felbontású időmérést minden probléma nélkül beépíthetünk. Ennyit tesz lehetővé az 50 Hz-es VIDEO megszakítás.

Tekintsük át, milyen feladatokat kell ellátnia megszakításkezelőnknek!

Ha megelégszünk 99,98 s-ig való méréssel, két számlálóra lesz szükségünk a századmásodpercek (REG1) és a másodpercek számára (REG2).

Foglaljuk egy egyszerű blokkdiagramba a teendőket!




A két regiszterhez válasszuk a 0000 és 0001 memóriacímet (bármilyen szabadon használható RAM megfelel). A számlálást végezzük BCD rendszerben. Ez egyrészt jobban megfelel a decimális kijelzési logikának, mint a bináris érték, másrészt igen egyszerű a 99→00-ba való átfordulás detektálása.

Az óra állását jelezzük ki ##,## alakban. A BCD értékek 2 számjegyes ASCII alakra konvertálásához jól felhasználhatjuk a TIME$ és DATE$ rutinok alprogramját az 1. szegmens C56FH címén. Itt is felmerül az előző példa kérdése: hol és hogyan jelezzük ki a stopperünket? Ebben az esetben válasszuk az egyszerűbb megoldás nagyobb kockázat kompromisszumát (legalább erre is látunk példát): használjuk a rendszerszegmens B230H címen kezdődő "copyright" területét. A kockázat természetesen csak abban áll, hogy a készítők monogramjaiba való belenyúlás miatt a RESET gomb egyszeri megnyomása is hidegindítást eredményez. Ha ez megengedhetetlen, használja az előző rutin kijelzési megoldását!

Ezután lássuk a felhasználói megszakítási rutin assembly listáját:

```
USR_ISR ORG 0180H       ;
        BIT  5,D        ; 50 Hz-es megszakitás?
        RET  Z          ; vissza; ha nem
IR_VID  LD   HL,0000    ; munkaregiszter
        LD   A,(HL)     ; REG1: 1/50-ed sec.
        INC  A          ; számlál
        DAA             ;     BCD korrekció
        INC  A          ; -""-
        DAA             ;      -""-
        LD   (HL),A     ; REG1=REG1+2
        JR   NZ,DISP    ; kijelzésre, ha REG1<100
        CCF             ;
        INC  HL         ;
        LD   A,(HL)     ; REG2: másodperc
        INC  A          ; számlál
        DAA             ; BCD korrekció
        LD   (HL),A     ; REG2=REG2+1
DISP    IN   A,(0B3H)   ; pillanatnyi 3.LAP
        PUSH AF         ; szegmense verembe
        LD   A,01       ; az 1. szegmenst
        OUT  (0B3H),A   ; lapozza a 3.LAP-ra
        LD   A,(0000)   ; A: századmásodperc
        LD   HL,0B23AH  ; HL: puffercim
        CALL 0C56FH     ; konverzió "##" alakra
        DEC  HL         ; elé
        LD   (HL),2CH   ; ":"
        LD   A,(0001)   ; A: másodperc
        CALL 0C56FH     ; konverzió "##" alakra
        POP  AF         ; szegmens-szám
        OUT  (0B3H),A   ; 3.LAP alapérték
        RET
```

A rutin működése nem igényel különösebb magyarázatot; lépésről-lépésre követi az előbbiekben tárgyalt megoldási logikát. Megírásához használhatunk assemblert, de nem igényel különösebb energiabefektetést a BASIC-ből való betöltés sem. A program a gépi rutin betöltése előtt kijelzi és törli (szóközökkel tölti fel) a kiválasztott memóriaterületet. Betöltés előtt itt is (mint az előző pontban) egy gépi rutint alkalmazhatunk a felhasználói megszakítási rutin címének beállítására (USR—ISR). Az eltérés az, hogy itt a stopper leállítására is szükség van. A bekapcsoló (MB) és kikapcsoló (MK) rutin teljesen analóg, csak MK nullát tölt az USR-ISR rendszerváltozóba.

```
100 POKE 49119,42
110 FOR N=0 TO 33
120   POKE 45616+N,32
130 NEXT
140 POKE 540,128
150 POKE 541,1
160 CODE M=HEX$("CB,6A,C8,21,00,00,
    7E,3C,27,3C,27,77,20,06,3F,23,
    7E,3C,27,77,DB,B3,F5,3E,01,D3,
    B3,3A,00,00,21,3A,B2,CD,6F,C5,
    2B,36,2C,3A,01,00,CD,6F,C5,F1,
    D3,B3,C9")
170 CODE MB=HEX$("F3,36,80,23,36,01,
    FB,C9")
```

```
180 CODE MK=HEX$("F3,36,00.23,36,00,
    FB,C9")
190 DEF STOPPER
200   CALL USR(MK,49133)
210   POKE 0,0
220   POKE 1,0
230   CALL USR(MB,49133)
240   FOR T=1 TO 548
250   NEXT
260   CALL USR(MK,49133)
270 END DEF
```

A program végén található STOPPER rutin egy egyszerű példa a megszakítási rutin felhasználására: a 250—260. programrészlet futásidejét méri (természetesen ennek helyére akármi mást is beírhatunk). A mérést a CALL STOPPER utasítással lehet indítani.

Érdekesebb alkalmazást tesz lehetővé a reflexidőt mérő következő program. A bevezető rész hasonló az előzőhöz: a 0,02 másodperces számláló megszakítási rutint tölti be. Mivel azonban itt nincs szükség a folyamatos kijelzésre, elhagyhatjuk az állapotsort előkészítő sorokat. Maga a gépi kódú rutin is egyszerűbb lett ezzel, hiszen abban is elhagyható a kijelző funkció: az előző assembly listában a DISP címke már a visszatérési pont lehet (RET). Ezzel egyben visszakaptuk a RESET gombot is.

```
110 POKE 540,128
120 POKE 541,1
130 CODE M=HEX$("CB,6A,C8,21,00,00,7E,3C,27,3C,
    27,77,20,06,3F,23,7E,3C,27,77,C9")
140 CODE MB=HEX$("F3,36,80,23,36,01,FB,C9")
150 CODE MK=HEX$("F3,36,00,23,36,00,FB,C9")
160 CALL REAKCIO_IDO
170 END
180 DEF REAKCIO_IDO
190   LET J,H=0
200   LET MIN=2:LET MAX=0
210   TEXT
220   PRINT AT 5,5:"  Nyomjon meg egy gombot, ha"
230   PRINT
240   PRINT "                    '*'"
250   PRINT
260   PRINT "    jelenik meg a - - jelek között!"
270   PRINT AT 15,18:"-   -"
280   WAIT 2
290   DO
300     CALL USR(MK,49133)
310     POKE 0,0
320     POKE 1,0
330     LET N=RND(5)
340     LET A$=CHR$(40+N)
350     PING
360     PRINT AT 15,20:A$
370     CALL USR(MB,49133)
380     IF PEEK(1)>=1 THEN GOTO 300
390     GET A$
400     IF A$="" THEN GOTO 380
```

```
410      CALL USR(MK,49133)
420      IF N=2 THEN CALL OK
430      IF N<>2 THEN CALL HIBA
440    LOOP
450 END DEF


460 DEF HIBA
470    SOUND
480    PRINT "HIBA!          "
490    LET H=H+1
500    PRINT AT 1,1:"Hiba:";H
510    WAIT 2
520 END DEF


530 DEF OK
540    SOUND PITCH 45
550    LET IDO=PEEK(1)+PEEK(0)/100
560    PRINT "id!=";IDO;"sec     "
570    LET J=J+1
580    PRINT AT 1,23:"Jo reakcio:";J
590    IF IDO<MIN THEN LET MIN=IDO
600.   IF IDO>MAX THEN LET MAX=IDO
610    PRINT AT 2,22:"Max. id!:";MAX
620    PRINT AT 3,22:"Min. id!:";MIN
630    WAIT 2
640 END DEF
```

Az akár játékként is használható program véletlenszerűen írja ki a 40—44 ASCII kódú karakterek valamelyikét a képernyő adott helyére és egyidejűleg elindítja a stoppert is (360. és 370.). A "játékosnak" csak a 'csillag' karakterre (CHRS(42)) szabad reagálnia, de arra minél gyorsabban, bármelyik billentyű lenyomásával. Ekkor a program leállítja a stoppert, majd kiértékeli és kijelzi a reakció helyességét és idejét. A futásidőt közvetlenül a számlálóból olvassa ki (550).

A HIBA és OK rutinok a kijelzésen kívül "adminisztrálják" is az elkövetett hibák (más karakterre való reagálások) és a jó reakciók számát, a futás alatt mért maximális és minimális reflexidőt. Ezeket a részleteket tehát érdemes még egy kissé "feldíszíteni", de ezt bizonnyal az Olvasóra bízhatjuk.

## 2.4.5 Hangkeltés a megszakítási rutinban

Utolsó alkalmazási példaként próbáljunk meg hangot is kelteni megszakításkezelő rutinunkban. Nem tűzünk ki magunk elé túl nagy célt (pl. zenét a háttérben), annak viszont semmi akadálya nincs, hogy — az 1 Hz-es megszakítást felhasználva — másodpercenként valamilyen jellegzetes géphangot adjunk ki. Azt ne kérdezze meg persze a kedves Olvasó, hogy mire jó egy ketyegő számítógép (hacsak nem egy várt vagy hívatlan vendég meghökkentése és teljes összezavarása a cél, közvetlen hasznát nemigen tudnánk megadni). Fontos lehet ugyanakkor az itt tárgyalt módszer, ahogyan megszakításkezelőből EXOS hívást indítunk.

A PING BASIC utasítás rendkívül egyszerű végrehajtó rutinja alapján (egyetlen EXOS hívásból áll), kézenfekvő megoldás lehetne a hangot ennek a rutinnak a hívásával (vagy másolásával) kelteni:

```
D3BF  3E 67       LD   A,67      : SOUND csatorna
D3C1  06 07       LD   B,07      : irandó kód
D3C3  F7 07       EXOS 07        : kiirás
D3C5  C9          RET
```

Ennél egyszerűbb feladat nincs is — gondolhatnánk! Be is írhatjuk az előző példák alapján a néhány utasításos felhasználói megszakításkezelőt — végül is nem "száll el" a gép, csak éppen semmit sem hajt végre. Pedig nem a programban van a hiba! Ha a rutint CALL USR ( ) utasítással hívjuk meg, azonnal megszólal a várt hang.

Mi az a lényeges változás a megszakítási rutinból való hívás esetén, ami miatt az EXOS nem hajtja végre a "karakterírás" funkciót (sőt, ha ellenőrizzük az akkumulátor hívás utáni tartalmát, kiderül, hogy az EXOS-ból FEH hibakóddal, .ILLFN, azaz "nem hozzáférhető EXOS hívás" hibával tért vissza).

A magyarázathoz adott minden információ, csak figyelmesen át kell néznünk az EXOS n végrehajtás menetét és a megszakításkezelést.

A válasz kulcsa a funkciókódok szerinti szétválogatásban, ill. az azt közvetlenül megelőző szakaszban van:

```
C462  37          SCF            : CY=1
C463  CD 56 00    CALL 0056      : visszatérési szakasz
C466  CD 80 C5    CALL C580      : szétválogatás
...
C580  30 20       JR NC,C5A2     : hiba, ha közben
...                                törlődött a CY
...
C5A2  3E FE       LD   A,FE      : .ILLFN hibakód
C5A4  C9          RET            : végrehajtás nélkül
                                   hibajelzéssel vissza
```

A 0056/57H címen általában egy EI:RET utasításpár áll (az előző EXOS hívás így fejezte be). Az EXOS megszakításkezelő azonban a 0056 címre egy OR A utasításkódot töltött (C4B8/AH).

Ez az utasítás törli az átvitelbitet, ha a felhasználói megszakítási rutinból adunk ki funkcióhívást és emiatt szalad a jelzett ágra a program. Megszakításkezelés során tehát nem lehet EXOS n utasítást kiadni. És ha mégis szükségünk van rá? Akkor persze lehet egy-két trükkel próbálkozni.

A megoldás első lépése egyszerű: át kell írni a bajt okozó bájtot pl. 00-ra (a NOP nem fogja törölni a CY-t). A megszakítási feladat elvégzése után persze vissza kell állítani az eredeti OR A értéket (B7H), hogy a hátralévő kezelőknek szabályos memóriabeállítást hagyjunk hátra.

Ezzel a fogással már működik az EXOS hívás, de néha egészen sajátos jelenségeket tapasztalhatunk. Ennek oka nyilván az, hogy az EXOS n szokása

szerint végrehajtás közben EI-re állította a sokat emlegetett 0056-os bájtot, és visszatérés előtt végre is hajtotta a megszakításengedélyezést, a megszakításkezelés közepén). Hogy javítsunk a helyzeten, tiltsuk le újra a megszakításokat közvetlenül az EXOS hívás után. Tegyünk meg még egy lépést, hogy a megmaradt minimális kockázatot is kizárjuk. A fejezet legelején megismert fogással (a 0038H megszakítás belépési cím ideiglenes átírásával) térítsük el az esetleg mégis becsúszó megszakításokat. A leírtakat a következő assembly rutin valósítja meg lépésről lépésre:

```
        ORG 0180H
USR_ISR BIT  3,D          ; 1 Hz-es megszakítás?
        JR   NZ,IR_1Hz    ;
        RET               ; vissza, ha nem
IR_1Hz  LD   A,00         ; "NOP"
        LD   (0056H),A    ; az EXOS ágba
        LD   A,0C9H       ; "RET"
        LD   (0038H),A    ; az IR belépésre
        LD   A,67H        ; SOUND csatorna
        LD   B,07H        ; írandó kód
        EXOS 07H          ; kiírás
        DI                ; INT letiltás
        LD   A,0F5H       ; "PUSH AF"
        LD   (0038H),A    ; eredeti érték
        LD   A,0B7H       ; "OR A"
        LD   (0056H),A    ; eredeti érték
        RET
```

Ezt a gépi kódú rutint tölti be a BASIC program, majd a megszokott módon beilleszti a megszakítási rendszerbe mint felhasználói megszakítási rutint:

```
100 POKE 540,128
110 POKE 541,1
120 CODE M=HEX$("CB,5A,20,01,C9")
130 CODE A=HEX$("3E,00,32,56,00,3E,
    C9,32,38,00,3E,67,06,07,F7,07,
    F3,3E,F5,32,38,00,3E,B7,32,56,
    00,C9")
140 CODE B=HEX$("F3,36,80,23,36,01,
    FB,C9")
200 CALL USR(B,49133)
```

Ezzel a megoldással más EXOS funkcióhívásokat is kiadhatunk (pl. karakterírás a képernyőre stb.).

Befejezésül még egy nagyon rövid példa arra, hogy nem mindig érdemes erőltetni az EXOS-on keresztüli hívást. Ha az előző feladatban csak a másodpercenkénti hangkeltés a cél, praktikusabb lehet a közvetlen hangkeltés. Hogy tényleg rövid legyen a példa, használjuk a billentyűzetkezelő TICK-hangot keltő rutinját (0. szegmens ECBAH). Ha az ECBCH címre lépünk be, és az A regiszternek különböző értékeket adunk, még kísérletezhetünk is a hanghatásokkal. A végrehajtandó feladat nagyon egyszerű: A-nak értéket adunk és meghívjuk a ROM-rutint. A program adataival egy óraketyegés jellegű hang szólal meg minden másodpercben. Próbálja

visszafejteni a gépi kódú rutint és keresse meg azt a listaelemet, amelynek átírásával megváltoztatható a hang jellege!

```
100 POKE 540,128
110 POKE 541,1
120 CODE =HEX$("CD,5A,20,01,C9,3E,
    48,CD,BC,EC,C9")
130 CODE MB=HEX$("F3,36,80,23,36,01,
    FB,C9")
140 CALL USR(ME,49133)
```

## 2.5 Perifériakezelők

Mint azt már láttuk, a számítógép és környezete között kapcsolatot teremtő eszközök (pl. billentyűzet, képernyő, magnetofon, stb.) kezelőprogramjait egyéges leíróik alapján tartja nyilván és működteti az operációs rendszer. Ezek a leírók láncot alkotnak a memóriában; ez a lánc rögzített címről indul és minden elem tartalmazza a következő leíró címadatait.

A 2.3.1 pontban a rendszer felépülésének részeként elemeztük, hogy miként alakítja ki az EXOS ezt a periférialáncot és hogyan keres meg benne egy-egy leírót (a LANC programmal ezt magunk is végigkövettük). Tudjuk tehát, hogy a periférialánc báziscíme (a rendszerszegmensben) BFC0H, ez alatt található az első leíró pointere: a leírót tartalmazó szegmens száma (báziscím – 1) és tárolási címe (báziscím – 2: felső bájt; báziscím – 3: alsó bájt). A tárolási cím egyben báziscím is: alatta ugyanilyen módon a következő láncelem pointerét találjuk. A lánc végét a (báziscím – 1) = 0 érték jelzi.

Hogyan kerülhet egy eszközleíró a láncba?

Egyrészt az inicializálás folyamatában automatikusan, ha a kezelő egy ROM-szegmens bizonyos formai követelménynek eleget tévő része. Ekkor az EXOS a ROM-ból a rendszerszegmensbe másolja a leírót alkotó bájtokat.

A másik mód, hogy egy felhasználói program kéri az általa (RAM-területen) szabványosan kialakított leíró nyilvántartásba vételét (egy EXOS 21 funkcióhívással).

A két lehetőség közötti különbség elsősorban a leíróhoz esetleg igényelt rendszer-RAM igénylési és kiutalási módjában van. A másik lényeges különbség a funkcionálisan egyébként gyakorlatilag azonos leírók között, hogy egy, a periférialánc törlésével járó rendszerinicializálás után (pl. ha a BASIC-ből a szövegszerkesztőbe lépünk) csak a ROM-leírók lánca épül automatikusan újra. Ez fontos lehet akkor, ha pl. egy szövegszerkesztőből is használni kívánt különleges printer-kezelőt hozunk létre. Számunkra elsősorban a második mód adott a saját kezelő beillesztésére, de (éppen az előbbiekben tárgyalt szempont miatt) keressük a módot az első lehetőség kihasználására is.

A saját perifériakezelő beillesztéséhez mindenképpen nélkülözhetetlen a leíró ismerete. A periférialánc egy-egy eleme a következő formájú:

(Báziscím – 3): alsó bájt
(Báziscím – 2): felső bájt a következő leíró címe (az 1./LAP tartományban)
(Báziscím – 1): a következő leíró szegmense
(Báziscím): TYPE - érvényes leírónál 0
(Báziscím + 1): IRQFLAG; bitjei azt jelzik, hogy a periféria kíván-e foglalkozni egy-egy megszakításkéréssel (1, ha igen)

BIT 1: hangmegszakítás (hang–IRQ)
BIT 3: 1 Hz-es IRQ
BIT 5: video (50 Hz-es) IRQ
BIT 7: külső (hálózat) IRQ

(Báziscím + 2): FLAGS; 1, ha a periféria kezeli a szegmenshatáron átnyúló puffert is (VIDEO)
0, ha csak egy szegmensen belüli puffert fogad el
(Báziscím + 3): TAB—LO bájt
(Báziscím + 4): TAB—HI bájt; a kezelő belépési ponttáblázatának kezdőcíme (az 1./LAP tartományban)
(Báziscím + 5): TAB—SEG — a táblázatot tartalmazó szegmens száma
(Báziscím + 6): UNIT—COUNT — 0, ha nem lehet több periféria ilyen néven; ebben az esetben az EXOS törli az előzőleg már esetleg felvett, ilyen nevű kezelőket a láncból (TYPE = FFH állítással)
(Báziscím + 7): NAME — a periférianév hossza
Báziscím + 8-tól a név karakterei következnek (NAME darab).

A báziscím előtti pointer 3 bájtját az EXOS írja be a beláncoláskor, a leíró további elemeit viszont nekünk kell előkészítenünk.

Az eszközkezelők másik szabványos része a végrehajtó rutinok belépési címeit tartalmazó táblázat (amelyre a leíró TAB pointere mutat). Ezeket a rutinokat az EXOS hívja az illető perifériával kapcsolatos csatornaműveletek vagy más feladatok végrehajtásához.

A híváskor bizonyos paramétereket átad a kezelőnek, mint azt a 2.3.2 pontban (csatornaműveletek előkészítése) láttuk. Az IY regiszterpár a leíró címét tartalmazza (a 2. LAP tartományában), B'-ben van a leírót tartalmazó szegmens száma. Csatornaműveleteknél IX a pufferterület fölé mutat, A-ban a belső csatornaszám (hívott csatornaszám + 1) található.

A végrehajtó rutinok belépési pontjainak táblázata már 3. LAP-címként adja meg a címek alsó és felső bájtjait (hiszen a kezelő — mint minden más rendszerprogram is — a 3. LAP-on fut majd). Tekintsük át röviden a táblázat elemeinek funkcióit (a beépített kezelők példáin utalva az elvégezhető vagy elvárt feladatokra)!

0. (TAB + 0/1): Megszakítás

Az IRQ—FLAG beállított bitjei alapján hívja meg az EXOS a megszakításkezelés során. Itt figyeli pl. a KEYBOARD a billentyűmátrixot 1/50 másodpercenként, itt újítja fel a VIDEO az állapotsor kijelzési paramétereit stb.

1. (TAB + 2/3): Csatorna megnyitása

A kezelőnek DE-ben (és esetleg BC-ben) a csatorna számára igényelt puffer nagyságával ki kell adnia egy EXOS 27 hívást. A funkcióhívás után a kezelő visszakapja a vezérlést (IX a kiutalt puffer fölé mutat) így módja van bejegyezni a csatorna paramétereit a pufferterületre (pl. a VIDEO ekkor írja be a videolap méretével, üzemmódjával, tárolási címeivel kapcsolatos adatokat). A puffert nem igénylő nyomtató-kezelő viszont csak az EXOS hívást adja ki (DE = 0000-val).

2. (TAB + 4/5): Csatorna létrehozása

Általában megegyezik az 1. rutinnal (l. 2.3.2 pont, EXOS 1,2).

3. (TAB + 6/7): Csatorna lezárása

A VIDEO-kezelő letakarja (a margók állításával) az aktuális képernyőn a lezárt csatornához tartozó sorokat. A többi kezelő belső logikájának megfelelően egyszerűbb feladatokat lát el (jelzőbájt törlése, munkaterület nullázása stb). A pufferterület tényleges felszabadítását az EXOS végzi el.

4. (TAB + 8/9): Csatorna megszüntetése

Általában megegyezik a 3. rutinnal (l. 2.3.2 pont, EXOS 3,4)

5. (TAB +10/11): Karakter olvasása

Kifejezetten perifériaspecifikus művelet. Azok a kezelők, amelyek nem olvasnak, "NOFN" hibakódot állítanak be az akkumulátorban (pl. printer)..

6. (TAB + 12/13): Blokk olvasása

A szokásos paramétereken kívül még az olvasandó karakterek számát (BC) és a felhasználó puffercímét (DE) is megkapja a kezelő. Az 5. rutin felhasználásával tölti a pufferbe a sorban olvasott karaktereket. Ha saját kezelőnkben vállalkozunk a blokkolvasásra, számolnunk kell a lapozás problémájával: a 0., 2. és 3. LAP tartalma kötött (nulláslap szegmens, rendszerszegmens és a kezelő szegmense). A csatornapuffer és a felhasználói célpuffer egyaránt az 1. LAP-on érhető el. Bonyolítja a problémát, hogy a pufferterület szegmenshatárt is átléphet.

7. (TAB + 14/15): Karakter írása

8. (TAB + 16/17): Blokk írása

A 6. rutinnal analóg feladatot kell ellátnia a kezelőnek.

9. (TAB + 18/19): Csatornakészenlét olvasása

A C regiszterben kell visszaadni a készenléti állapotot:

00H, ha a karakter olvasható,
FFH, ha file-vége,
01H egyébként.

A beépített kezelők közül pl. a VIDEO feltétel nélkül C = 0-t állít, a KEYBOARD viszont csak akkor, ha van karakter a pufferben (volt megnyomott billentyű).

10. (TAB + 20/21): Csatornaállapot beállítása vagy olvasása

A beépített kezelők .NOFN hibakóddal válaszolnak erre a hívásra. Saját kezelőnkben tetszés szerint használhatjuk (ha nincs lemezkezelő a rendszerben).

11. (TAB + 22/23): Különleges funkció

Az egyes perifériákhoz szóló célhívás (l. 2.3.2 pont, EXOS 11). Saját kezelőnkben a 63 fölötti speciális funkciószámokat használhatjuk.

12. (TAB + 24/25): Inicializálás

A kezelők saját, egyértelmű alaphelyzetüket állíthatják be. A VIDEO pl. ebben a rutinban hozza létre a kijelzés alapjául szolgáló sorparaméter-táblát és a karaktergenerátort, a SOUND a DAVE-chip regisztereinek alapértékeit állítja be egy ROM-tábla alapján.

A felhasználói programból EXOS hívással rendszerbe illesztett perifériakezelő esetén külön jelentősége van ennek a rutinnak. A beláncolást kérő EXOS 21 hívás befejezéseként ui. az operációs rendszer meghívja az újonnan beillesztett kezelő 12. (Inicializálás) rutinját, miközben IX a perifériához kért és az EXOS által kiutalt pufferterület fölé mutat. Ilyenkor kell megjegyeznie a kezelőnek a kapott RAM-terület címét.

13. (TAB + 26/27): Puffermozgatás

Valójában nem a kezelőnek kell a puffert mozgatnia, csak lehetőséget kap a rendszer által pl. egy csatorna lezárása utáni memóriarendezés során eltolt csatornapuffer címváltozásainak könyvelésére. Ehhez a kezelő megkapja a csatornapuffer új báziscímét (IX) és az eltolás mértékét (BC). Ezek alapján határozza meg pl. a VIDEO-kezelő az érintett videolap új kijelzési címeit (és tárolja a puffer részét képező paraméterterületen).

Az 5. Függelékben megadjuk a beépített perifériakezelők leíróit és belépési pontjait. Ezek egy része — a 2.3.2 pontban leírt feltételek betartásával — közvetlenül is hívható, másrészt egy disassemblerrel követve egy-egy rutin felépítését, jó ötleteket kaphatunk saját kezelőink elkészítéséhez.

A megismert leírót és belépési címtáblázatot kell tehát elkészítenünk, mielőtt saját kezelőnk beláncolását kérnénk. Az EXOS 21 funkcióhívás előtt DE-t a leíró TYPE elemének címére kell állítanunk, míg BC-ben megadhatjuk a rendszerszegmensben esetleg kért RAM-puffer nagyságát. A hívás során az EXOS érintetlenül hagyja a leírót abban a szegmensben, amelyikben felépítettük, csak beilleszti a periférialáncba. Ehhez viszont fontos, hogy a TYPE elem előtt még legyen 3 szabad RAM-hely (a következő láncelem pointerének beírására). Az így beláncolt kezelő csak egy újrainicializálás során törlődik (l. EXOS 0).

Az elmondottakat leginkább egy konkrét példánál lehet átlátni. Alkalmazásként készítsünk el és illesszünk a rendszerbe egy saját perifériakezelőt. A leíró és kezelő

ROM-szegmensben való felépítésével és beillesztésével egy külön példában foglalkozunk (l. 2.5.2 pont).

### 2.5.1 Nyomtatás saját karakterkészlettel

Építsünk fel egy olyan nyomtatókezelőt (programrendszert és leírót), amely a nyomtató grafikus üzemmódjának felhasználásával a kiírt szöveget az ENTERPRISE karakterkészletével jeleníti meg. Ha ezzel egyszerűen nyomtathatóvá tesszük a normál karakterkészleten túl a saját definiálású karaktereket is (ékezetes betűk, matematikai jelek, idegen karakterkészletek stb.), az egyszerű illusztráción túl jól használható lehetőséghez is jutunk. Először gondoljuk végig, milyen feladatokat kell ellátnia a kezelő 13 funkciórutinjának. A választott periféria eléggé egyszerű ahhoz, hogy a funkciók egy részénél (megszakítás (0), csatornalezárás (3) és megszüntetés (4), inicializálás (12) és puffermozgatás (13)) ne kelljen tényleges műveletet elvégezni, itt csak az A = 0 státuszt állítjuk be ("OK").

A funkciók másik csoportjának hívása az adott eszköznél hibának számít. A karakterolvasás (5), blokkolvasás (6), a csatornakészenlét olvasása (9) és a csatornaállapot beállítása (10) funkciókra ".NOFN" hibakóddal válaszolunk, ("HIBA"), az esetünkben nem kezelt speciális funkcióhívásból pedig A = EAH (".ISPEC") hibakóddal térünk vissza.

Nem törvényszerű, de nagymértékben leegyszerűsíti a kezelést, ha nem végezzük el a blokkírás funkciót sem (ezzel lényegében csak a "LIST # csat." lehetőséget veszítjük el), de ez a probléma is megkerülhető (COPY). Szükség esetén a beépített nyomtató-kezelő 6. rutinjának (l. 5. Függelék) elemzésével kiegészíthető a saját kezelő is.

Érdemi funkciót tehát csak a csatornanyitás (1), ill. — létrehozás (2) és természetesen a karakterírás (7) hívásokra kell ellátnunk.

A csatornanyitáshoz mindössze egy EXOS 27 hívást adunk ki DE = 0000-val, hiszen nem igénylünk csatornapuffert (a kivitelhez szükséges pufferterület a saját RAM-szegmensben lehet).

A karakterírás funkciót alaposabban át kell gondolni. Az alapműveletet (egy bájt kivitele a nyomtatóra) nyugodtan elleshetjük a belső nyomtató-kezelőtől, ennél jobban úgysem tudnánk megoldani (ez a felépített assembly program BYTE-KI rutinja).

Ez a rutin azonban a mi esetünkben csak végrehajtó alprogramként használható. Kivételt a nyomtatót vezérlő karakterek jelentenek: a kocsivissza (CR) vagy soremelés (LF) kódokat természetesen nem kell grafikusra alakítani. Kezelőnk a 32 alatti írandó kódok számára átlátszó, azokat közvetlenül a BYTE-KI rutinnak adja át.

Az írható karakterek esetén viszont a karakter kódja helyett az ENTERPRISE karakter-RAM (rendszerszegmens, B480H kezdőcím) adatai alapján meghatározott bájtsorozatot kell a nyomtatóra küldeni. A grafikus üzemmód az EPSON (és velük kompatíbilis) nyomtatóknál egy vezérlőkarakter-sorozattal váltható ki. Az alapfelbontáshoz (ESC"K"): 1BH, 4BH, LO, HI

A LO és HI adatok a grafikusként értelmezendő bájtok számát adják meg (LO: alsó bájt, HI: felső bájt, max. érték: 480).

Esetünkben vigyük ki a grafikus adatokat karakterenként. Ez 8 adatbájtot jelent: LO = 8, HI = 0. A vezérlőkódokat követő bájtok egy-egy grafikus pontoszlop adatait adják a nyomtatónak: a legfelső bit (128-as helyi érték) a legfelső pontot jelenti.

A karaktergenerátor adatai viszont a megjelenítendő pontmátrix sorait adják. Az első sor adatának címe: a karakter-RAM kezdete + a karakter kódja. A következő sorok adatai 128-asával növekvő címeken találhatók.

A soradatok oszlopadatokká alakítása kezelőnk feladata. Ehhez a rutin egy 8-bájtos adatpuffert használ (ami az átalakítás után egyben a grafikus kivitel adatmezője is).

A konverzió kulcsa a balra görgető RL utasítás: a sorban megjelenítendő biteket tartalmazó adatot ciklikusan balra léptetjük és a CY-be kicsordult bitet a puffer egymás utáni bájtjaiba léptetjük be. A 8 bit 8 oszlopbájtba kerül és ahogy egyre újabb soradatokat konvertálunk, egyre feljebb lépnek, míg végül kialakul a 8 adatbájt. Itt jegyezzük meg, hogy az ENTERPRISE karaktermátrixai 9 pontsorból állnak, de a nyomtatóknak csak 8 tűje vezérelhető az adat 8 bitjével. Programunk a felső 8 pontot jeleníti meg, de — a karakter-RAM kezdőcímének eltolásával — átállhatunk az alsó 8 bitre is.

Ezek után felépíthetjük a progamrendszert a felsorolt feladatoknak megfelelően:

```
        ORG  0C000H

OK      XOR A               ;A=0 (nincs hiba)
        RET

HIBA    LD  A,0E7H          ;".NOFN" hibakód
        RET

IRO         EQU  OK

OPEN    XOR A               ;A=0
        LD  D,A             ;DE=0000
        LD  E,A             ;(nincs pufferigény)
        EXOS 27             ;kötelező hívás
        RET

OPEN2       EQU OPEN
CLOSE       EQU OK
CLOSE2      EQU OK
OLV         EQU HIBA
BLOKK_OLV   EQU HIBA
BLOKKIRAS   EQU HIBA
ALL_OLV     EQU HIBA
ALL_IRAS    EQU HIBA

SPEC    LD  A,0EAH          ;".ISPEC" hibakód
        RET

INIC        EQU OK
PUF_MOZG    EQU OK
```

```
IRAS    LD  A,B             ;irandó karakter
        CP  20H             ;alfanumerikus?
        JP  C,BYTE_KI       ;azonnal kiviszi, ha
                              vezérlőkarakter
        LD  E,B             ;E:irandó karakter kódja
        XOR A               ;A=0
        LD  D,A             ;D=0
        LD  HL,PUFFER+4     ;adatpuffer
        LD  IY,0B480H       ;karakter_RAM kezdőcim
        ADD IY,DE           ;+00nn: irandó karakter
        LD  B,8             ;8 pontsort olvas
C1      PUSH BC             ;számláló verembe
        LD  A,(IY+0)        ;pontmátrix adat
        LD  B,8             ;8 bitet figyel
        LD  HL,PUFFER+4     ;adatpuffer
C2      RLA                 ;MSB -> CY
        RL  (HL)            ;CY -> (HL)
        INC HL              ;adatpuffer köv. cím
        DJNZ C2             ;következő bit kivitele
        LD  BC,0080H        ;következő mátrixadat
        ADD IY,BC           ;címe
        POP BC              ;sorszámláló vissza
        DJNZ C1             ;köv. pontsorra


PUF_KI  LD  HL,PUFFER       ;puffercím
        LD  B,12            ;4 vezérlőkar.+ 8 adat
C3      PUSH BC             ;számláló verembe
        LD  B,(HL)          ;kiirandó bájt
        CALL BYTE_KI        ;nyomtatóra
        INC HL              ;következő puffercím
        POP BC              ;számláló vissza
        DJNZ C3             ;a következő bájtra
        XOR A               ;A=0 (".OK")
        RET                 ;

BYTE_KI LD  A,B             ;kiirandó bájt
        OUT (0B6H),A        ;PORT-ra
C4      LD  A,(0BFF2H)      ;FLAG_SOFT_IRQ
        CP  20H             ;volt "STOP"?
        LD  A,0E5H          ;".STOP" hibakód
        RET Z               ;

        IN  A,(0B6H)        ;PORT adat
        BIT 3,A             ;"READY"?
        JR  NZ,C4           ;vissza, ha a PRINTER
                              nem kész
        LD  A,(0BFF3H)      ;PORTB5
        OR  10H             ;STROBE=1
        OUT (0B5H),A        ;kivitel
        AND 0EFH            ;STROBE=0
        OUT (0B5H),A        ;kivitel
        XOR A               ;".OK"
        RET                 ;


LEIRO   DEFB    0,0,0       ;3 hely a láncoláshoz
TYPE    DEFB    0,0,0       ;TYPE,IRQFLAG,FLAGS
        DEFW    TAB-8000H   ;TAB_LO,TAB_HI
```

```
        DEFB    0FBH,0      ;TAB_SEG,UNIT_COUNT
NAME    DEFB    10,"GRFPRINTER"

;a belépési pontok táblázata

TAB     DEFW    IRQ
        DEFW    OPEN
        DEFW    OPEN2
        DEFW    CLOSE
        DEFW    CLOSE2
        DEFW    OLV
        DEFW    BLOKK_OLV
        DEFW    IRAS
        DEFW    BLOKKIRAS
        DEFW    ALL_OLV
        DEFW    ALL_IRAS
        DEFW    SPEC
        DEFW    INIC
        DEFW    PUF_MOZG


PUFFER  DEFB 1BH,4BH,8,0,0,0,0,0,0,0,0,0
```

A rutinok után kialakított leíró elemei is a fenti logikának felelnek meg:

TYPE = 0: érvényes
IRQFLAG = 0: nem kezelünk megszakítást
FLAGS = 0: nem VIDEO típus
UNIT-COUNT = 0: nem lehet több ilyen nevű kezelő

A leíró TAB elemeivel foglalkozzunk egy kicsit részletesebben. A rutinokat és táblázatokat az 1. fejezetben említett ASMON assemblerében írtuk meg. Az ORG C000H utasítás miatt az ASMON úgy fordítja le az assembly programot, hogy a címkéket a 3. LAP C000H címétől számítja (OK = C000H, HIBA = C002H stb.). Így a belépési cím táblázatba helyesen, 3. LAP-címként kerülnek az értékek. A leíró TAB elemének azonban az 1. LAP-on kell megadnia a táblázat kezdőcímét; ezt szolgálja a –8000H eltolás.

A táblázat szegmense (a fenti listában FBH) természetesen mindig az a szegmens lesz, amelyet valamilyen módon lefoglaltunk és amelybe ténylegesen betöltöttük a kezelőt.

Ha ténylegesen a memóriába akarjuk tölteni a kezelő rutinjait és táblázatait, ügyelni kell arra, hogy ezt nem végezhetjük el a 3. LAP-on (ott maga az ASMON fut). Kénytelenek vagyunk pl. 1. LAP-ra lapozni a kiválasztott szegmenst és 8000H MEMORY OFFSET értékkel végeztetni a fordítást.

Szokás szerint itt is megadjuk a BASIC betöltőprogramot:

```
100 ALLOCATE 100
110 CODE SZEGMENS=HEX$("F7,18,79,D3,
    B1,C9")
120 CODE BELANCOLAS=HEX$("11,6A,40,
    01,00,00,F7,15,C9")
130 CALL USR(SZEGMENS,0)
140 POKE 540,0
150 POKE 541,64
160 CODE OK=HEX$("AF,C9")
170 CODE HIBA=HEX$("3E,E7,C9")
180 CODE OPEN=HEX$("AF,57,5F,F7,1B,
    C9")
190 CODE SPEC=HEX$("3E,EA,C9")
200 CODE IRAS=HEX$("78,FE,20,DA,49,
    C0,58,AF,57,21,9D,C0,FD,21,80,
    B4,FD,19,06,08,C5,FD,7E,00,06,
    08,21,9D,C0,17,CB,16,23,10,FA,
    01,80,00,FD,09,C1,10,E9")
210 CODE PUF_KI=HEX$("21,99,C0,06,
    0C,C5,46,CD,49,C0,23,C1,10,F7,
    AF,C9")
220 CODE BYTE_KI=HEX$("78,D3,B6,3A,
    F2,BF,FE,20,3E,E5,C8,DB,B6,CB,
    5F,20,F2,3A,F3,BF,F6,10,D3,B5,
    E6,EF,D3,B5,AF,C9")
230 CODE LEIRO=HEX$("00,00,00")
240 CODE TYPE=HEX$("00,00,00,7D,40")
250 CODE =CHR$(IN(177))
260 CODE =HEX$("00,0B")&"
    GRAFPRINTER"
270 CODE TAB=HEX$("00,C0,05,C0,05,
    C0,00,C0,00,C0,02,C0,02,C0,0E,
    C0,02,C0,02,C0,02,C0,0B,C0,00,
    C0,00,C0")
280 CODE PUFFER=HEX$("1B,4B,08,00")
290 CALL USR(BELANCOLAS,0)
```

A program itt lényegesen többet végez el a tulajdonképpeni betöltésnél:

— egy EXOS hívással lefoglal egy szegmenst és ezt be is lapozza az 1. LAP-ra (100—130)

```
EXOS 24          F7 18
LD A,C           79
OUT (B1H),A      D3 B1
RET              C9
```

— a belapozott szegmens kezdőcímére állítja a CODE—mutatót, majd betölti a kezelő rutinjait és tábláit (140—280)

— egy újabb gépi rutinnal kéri a beírt kezelő beláncolását (120 és 290):

```
LD DE, 406AH     11 6A 40
LD BC,0000H      01 00 00
EXOS 21          F7 15
RET              C9
```

Itt DE a leíró TYPE elemének tárolási címe, BC = 0, mert nem igénylünk perifériapuffert.

A program lefuttatása után létezik a GRAFPRINTER nevű periféria, megnyithatunk hozzá egy csatornát:

OPEN #1: "GRAFPRINTER:"

utasítással, amit a szokásos módon használhatunk numerikus vagy szöveges adatok kiírására, a szöveges képernyő (pontosabban az EDITOR) COPY-zására stb:

```
π= 3.141592654

ok
OPEN #1:"GRAFPRINTER:"
ok
PRINT #1:"π=":PI
ok
COPY FROM #0 TO #1
```

(Az alternatív karakterkészlet legfelső sorának kinyomtatása a karakter-RAM címének előbbre állításával érhető el: LD IY,B400H).

Befejezésül két megjegyzés.

— A kiíró rutin viszonylagos egyszerűségének megvan a hátránya is. A kezelő az adatokat karakterenként viszi ki, emiatt a nyomtató is karakterenként (zajosan és lassan) nyomtat. Ha valakinek gyakran van szüksége erre a szolgáltatásra, érdemes egy kis munkával kiegészíteni a kezelőt. A lehetőség adott: a puffer tovább terjeszkedhet a RAM-ban, tehát mindaddig letárolhatjuk az adatbájtokat, amíg egy nyomtatóvezérlő karakter nem érkezik. A puffer fejrészében természetesen mindig növelni kell 8-cal a grafikusan értelmezendő adatok számát. Ez a számláló lehet egyben a mutató is a következő szabad pufferhelyre.

— Az előbbieknél is jobban alkalmazható megoldás lehet, ha kombináljuk a mi kezelőnket a nyomtatókat közel levélminőségű (NLQ) vezérlését célzó (több helyen publikált) ötletekkel. Semmi akadálya a nyomtató maximálisan finom vezérlésének és saját, nagyfelbontású (pl. 24x24-es) karakterkészlet létrehozásának. Ez természetesen nagy munka, de a saját karakterekkel elérhető NLQ nyomtatás lehetősége is nagy vonzerő.

## 2.5.2 Nyomtatóvezérlés a szövegszerkesztőből

Az ENTERPRISE beépített szövegszerkesztőjének egyik legnagyobb hiányossága, hogy nem küldhetünk a nyomtatóra vezérlőkaraktereket, így nem használhatjuk ki a nyomtató lehetőségeit, írásmódjait (betűtípus választás, aláhúzás, dőlt betű, vastagítás stb.).

Aki az előző pontot áttekintette, valószínűleg érzi: semmi akadálya, hogy a beépített PRINTER-kezelőt felváltsuk egy olyan kezelővel, amely a karakterkódok adott tartományában (pl. 127 fölött) a kód helyett egy belső táblázat vezérlőkaraktereit

küldené a nyomtatóra. Nem is a kezelő létrehozása a gond, hanem az, hogy az előző fejezet módszerével beláncolt perifériakezelő törlődik, ha a szövegszerkesztőbe lépünk.

Az inicializálással járó átlépésnél csak a ROM-szegmensek beépített kezelőiből épül újra a periférialánc automatikusan. Nem kell azonban feladnunk az ötletet, ha felidézzük, hogy a 2.3.2 pontban — az EXOS 0 alkalmazásaként — sikerült (az operációs rendszer megtévesztésével) egy tetszőleges RAM-szegmenst felvétetni a ROM-szegmensek nyilvántartásába. Ezt a szegmenst a továbbiakban teljes értékű ROM-szegmensként kezeli a rendszer; a benne kialakított kezelő ezután beépítettnek számít.

A beépített perifériakezelők beláncolása ugyancsak leíróik alapján történik, de ezek a belső leírók (álleírók) néhány pontban különböznek az előbbiekben megismertektől. A tulajdonképpeni leírón kívül az átmásolásukhoz, megkeresésükhöz szükséges információkat is tartalmaznak:

SIZE: a periféranév utáni bájton meg kell adni a leíró hosszát (a TYPE-tól a név utolsó karakteréig)

RAM—LO

RAM—HI: a TYPE elem előtt meg kell adni az igényelt RAM-terület nagyságát –2-től induló negatív értékkel: 10000H – 0002H – RAM

Ha nincs szükség pufferre, az alapérték –2 (azaz FFFEH)

RAM—LO = FEH

RAM—HI = FFH

NEXT—LO

NEXT—HI: a RAM előtti két bájton a szegmens következő beépített perifériakezelőjének címét kell megadni: a SIZE elem címét az 1. LAP tartományában. Ha nincs további kezelő, NEXT = 0000H

A szegmens első kezelőjének címét a 8/9. bájton kell letárolni (ugyancsak 1. LAP címként). Itt indul a kezelőlánc keresése az inicializálás során.

Az igényelt RAM-területet ebben az esetben közvetlenül a leíró alatt foglalja le az EXOS, tehát ha a kezelőnkhöz fordul a rendszer, a puffert egyszerűen, az IY-1, IY-2 stb. címeken érhetjük el.

A másik sajátosság a ROM-szegmensek kezelőinek beláncolásakor, hogy az EXOS a TAB-SEG elemhez nem veszi figyelembe az álleíró adatát, ide mindenképpen azt a szegmensszámot írja, ahol a kezelőt megtalálta.

Ezután a fentiek és az előző pont alapján könnyen felépíthetjük nyomtatókezelőnket (pl. NYOMTATO néven):

```
                ORG     0C008H
MUTATO  DEFW    SIZE-8000H

        ORG     0C010H
LEIRO   DEFB    0,0             ;NEXT: lánc vége
        DEFB    0FEH,0FFH       ;nincs RAM-igény
TYPE    DEFB    0,0,0           ;TYPE,IRQFLAG,FLAGS
        DEFW    TAB-8000H       ;TAB_LO,TAB_HI
        DEFB    0,0             ;TAB_SEG,UNIT_COUNT
NAME    DEFB    7,"PRINTER"     ;periférianév
SIZE    DEFB    0FH             ;15 bájt a TYPE-ig

Belépési pontok táblázata:,

TAB     DEFW    IRQ
        DEFW    OPEN
        DEFW    OPEN2
        DEFW    CLOSE
        DEFW    CLOSE2
        DEFW    OLV
        DEFW    BLOKK_OLV
        DEFW    IRAS
        DEFW    BLOKKIRAS
        DEFW    ALL_OLV
        DEFW    ALL_IRAS
        DEFW    SPEC
        DEFW    INIC
        DEFW    PUF_MOZG

OK      XOR  A
        RET
HIBA    LD   A,0E7H
        RET

IRQ     EQU OK
OPEN    XOR  A
        LD   D,A
        LD   E,A
        EXOS 27
        RET
OPEN2   EQU OPEN
CLOSE   EQU OK
CLOSE2  EQU CLOSE
OLV     EQU HIBA
BLOKK_OLV EQU HIBA
BLOKKIRAS EQU HIBA
ALL_OLV EQU HIBA
ALL_IRAS EQU HIBA
SPEC    LD   A,0EAH
        RET
INIC    EQU OK
PUF_MOZG EQU OK
```

```
IRAS    LD   A,B          :irandó karakter
        CP   80H          :A<128?
        JP   C,BYTE_KI    :kiírásra, ha igen
        SUB  80H          :vezérlőkód sorszám
        LD   C,A          :C=sorszám
        LD   HL,TABLA     :HL=tábla kezdőcím
        LD   D,0          :
        INC  C            ;
C1      DEC  C            :sorszám-1
        JR   Z,C2         :kiolvasásra, ha elértük a
                           keresett elemet
        LD   E,(HL)       :az adott elem hossza
        INC  E            :+ a hosszbájt
        ADD  HL,DE        :HL=a következő elem táblacíme
        JR   C1           :

C2      LD   B,(HL)       :hány bájtot vigyen ki
C3      INC  HL           :a következő táblacím
        LD   A,(HL)       :adat a táblából
        CALL BYTE_KI      :kiírás
        DJNZ C3           :ismétlés B-szer
        RET

BYTE_KI OUT  (0B6H),A
C4      LD   A,(0BFF2H)
        CP   20H
        LD   A,0E5H
        RET  Z
        IN   A,(0B6H)
        BIT  3,A
        JR   NZ,C4
        LD   A,(0BFF3H)
        OR   10H
        OUT  (0B5H),A
        AND  0EFH
        OUT  (0B5H),A
        XOR  A
        RET

A vezérlőkarakterek táblázata:

        ORG  0C100H
TABLA   DEFB 1,15  ;CONDENSED tipus
        DEFB 1,18  ;COND. törlés
        ...
(igény szerint tölthető fel)
```

A formai eltéréseken kívül az IRÁS funkcióban változtattunk lényegesen a kezelőn:

— a 128 alatti kódú karaktereket (alapjelkészlet, vezérlőkarakterek) változtatás nélkül viszi ki, ez a normál nyomtatófunkció;

— a 128-as és nagyobb kódú karakterek (ALT + betű) helyett egy táblázat adatait küldi a nyomtatóra.

A vezérlőkarakterek táblázatát saját igényeink szerint alakíthatjuk ki. Az első bájt a következő vezérlőkarakterek száma legyen.

A kezelő itt is létrehozható BASIC-ből is:

```
100 OUT 177,252
110 POKE 540,8
120 POKE 541,64
130 CODE MUTATO=HEX$("24,40")
140 POKE 540,16
150 CODE LEIRO=HEX$("00,00,FE,FF,00,
    00,00,25,40,00,00,08")&"
    NYOMTATO"
160 CODE SIZE=HEX$("10")
170 CODE TAB=HEX$("41,C0,46,C0,46,
    C0,41,C0,41,C0,43,C0,43,C0,4F,
    C0,43,C0,43,C0,43,C0,4C,C0,41,
    C0,41,C0")
180 CODE =HEX$("AF,C9,3E,E7,C9,AF,
    57,3F,F7,1B,C9,3E,EA,C9")
190 CODE IRAS=HEX$("78,FE,80,DA,6F,
    C0,D6,80,4F,21,00,C1,16,00,0C,
    0D,28,05,5E,1C,19,18,F8,46,23,
    7E,CD,6F,C0,10,F9,C9,D3,B6,3A,
    F2,BF,FE,20,3E,E5,C8,DB,B6,CB,
    5F,20,F2,3A,F3,BF,F6,10,D3,B5,
    E6,EF,D3,B5,AF,C9")
200 POKE 540,0
210 POKE 541,65
220 CODE TABLA=HEX$("01,0F,01,12")
230 POKE 540,0
240 POKE 541,48
250 CODE M=HEX$("ED,5B,97,BF,2A,9C,
    BF,B7,ED,52,E5,C1,EB,E5,11,04,
    00,ED,52,EB,E1,D5,ED,B0,11,04,
    00,ED,52,36,FC,E1,22,97,BF,22,
    95,BF,AF,32,78,BF,0E,60,F7,00,
    C9")
260 SPOKE 252,10,201
270 CALL USR(M,0)
```

A program első része az FCH szegmensbe tölti a táblákat és rutinokat (100—190). A 190—220 sorok a vezérlőkarakterek táblázatát kezdik el.

A befejező rész az FCH szegmenst építi be a ROM-szegmensek táblázatába. A hívással törlődnek bizonyos belépési címek (l. EXOS 0). Ezeket közvetlenül is visszaírhatjuk, de egyszerűbb egy :BASIC utasítás kiadása. Ez ugyan törli a programot is, de az már úgyis ellátta a feladatát.

A vezérlőkarakterek táblázata nyomtatónk és igényeink szerint teljessé tehető a 220. sor adatmezőjének folytatásával, de külön programmal is.

```
100 FOR N=0 TO 82
110   READ A
120   SPOKE 252,256+N,A
130 NEXT
140 DATA 1,15  !condensed          ALT+U
150 DATA 1,18  !cond. vissza       ALT+A
160 DATA 2,27,52 !dőlt             ALT+B
170 DATA 2,27,53 !dőlt vissza      ALT+C
180 DATA 2,27,77 !ELIT             ALT+D
190 DATA 2,27,80 !PICA             ALT+E
200 DATA 2,27,69 !fett             ALT+F
```

```
210 DATA 2,27.70 !fett vissza          ALT+G
220 DATA 2,27.71 !dupla                ALT+H
230 DATA 2,27,72 !dupla vissza         ALT+I
240 DATA 3,27.87,1 !széles             ALT+J
250 DATA 3,27,87,0 !széles vissza      ALT+K
260 DATA 3,27.45,1 !aláhúzott          ALT+L
270 DATA 3,27.45,0 !aláh. vissza       ALT+M
280 DATA 4,27.83,0,15 !felső index     ALT+N
290 DATA 4,27.84,0,18 !ind. vissza     ALT+O
300 DATA 4,27.83,1,15 !alsó index      ALT+P
310 DATA 4,27.84,0,18 !ind. vissza     ALT+Q
320 DATA 18,27,76,14,0,0,32,80,66,65,73,
    126,72,72,80,64,128,0,0 !spec.     ALT+R
```

```
                Mintaszöveg

A vezérlőkarakterek kiviteli lehetőségével az
EPSON nyomtatók minden szolgáltatását
elérhetjük a szövegszerkesztőből is.

 Megjegyzés:  A különböző kiemelési
              módokon kívül többféle
              karaktertípust használ-
              hatunk.
Még a műszaki szövegben gyakran szükséges indexek
sem okoznak problémát (H2SO4, e-E/kT stb.).

Lehetőség van saját (ékezetes vagy speciális: ₹)
karakterek definiálására is.
```

Az így létrehozott kezelő már a szövegszerkesztőből is használható. Mivel a rendszer alapértelmezésben a PRINTER-kezelőt használja, a nyomtatóutasítás (F3) után a "Press ENTER for parallel printer or type device-name:", ill. "Falls Parallel-Drucker, so ENTER. Sonst Vorsichtungsnamen eingeben:" kérdésre nekünk kell megadnunk a használandó kezelő nevét: NYOMTATO:.

Példánkban nem foglalkoztunk a felhasznált szegmens lefoglalásával. A szegmensigénylés (EXOS 24) egyszerű módja itt nem jelent megoldást, mert az inicializálással járó átlépés (WP) törli a felhasználói szegmeskiutalást is. Ha olyan nagy fájlokkal dolgozunk, hogy felmerülhet a memóriavédelem igénye, perifériának kiutalt szegmensként kell lefoglalnunk a munkaterületet, amit akár közvetlen beavatkozással is megvalósíthatunk a nyilvántartási logika alapján (l. EXOS 24 és EXOS 25).

## 2.6 Rendszerbővítők

A nagyobb, valamilyen rendszert alkotó programok rendszerbővítőként kapcsolódnak az ENTERPRISE operációs rendszeréhez. Ilyennek tekinthető az alapkiépítésű gépben a modulban csatlakoztatott BASIC, de ilyen pl. a külső adathordozóról beolvasott PASCAL is. Ezek a rendszerek ténylegesen az EXOS a bővítései: az

operációs rendszer szolgáltatásainak körét szélesítik ki. Az EXOS a nyilvántartásba vett bővítőket sok esetben körbekérdezi (ilyen hívás váltható ki pl. BASIC-ben a :HELP vagy :WP paranccsal), lehetőséget adva üzenetek kiírására, az EXOS hibák egyéni hibaüzeneteinek megadására vagy akár a vezérlés átvételére. Saját programunkat (programrendszerünket) is így tudjuk a legszorosabban csatolni a rendszerhez; az egyszerű hívási, aktivizálási lehetőség mellett mi magunk is így kaphatjuk a legtöbb szolgáltatást a rendszertől.

A rendszerbővítők két fő típusba sorolhatók. Az EXOS rendszerbővítőként kezeli az inicializálás során a ROM-könyvtárba felvett szegmenseket: az EXOS-ROM szöveggel kezdődő ROM-okat és az 1. szegmenst. A ROM-szegmensek táblázatába, mint az EXOS 0 tárgyalásakor vagy a 2.5.2 pontban láttuk, beilleszthetjük az általunk használt szegmenst és ezzel saját bővítőnket is.

A másik típust a RAM-ban lévő rendszerbővítők alkotják, amelyeket egy bővítőláncban tart nyilván az EXOS. Ebbe a láncba a beolvasott rendszerbővítők kerülnek be (PASCAL, ASMON stb.), de mi magunk is beláncolhatjuk bővítőnket (akár néhány memóriacím közvetlen módosításával is).

Foglaljuk össze a rendszerbővítők kialakításához szükséges információkat (a hívások módját és eseteit már láttuk a 2.3.1 és 2.3.2 pontban).

— A ROM-bővítőket és az ún. abszolút rendszerbővítőket a 3. LAP-on, a C00AH belépési címen hívja az EXOS (a közvetlen beláncolással más kezdőcím is letárolható).

— A rendszer 8 különböző esetben (8 különböző céllal) hívja a bővítőket C = 1,2...8 akciókóddal. Ezek egy részét (C = 2,3 esetleg C = 1) a rendszerbővítők maguk is kérhetik egy EXOS 26 funkcióhívással. A hívásokra nem kötelező válaszolni (a C nem kezelt értékeinél egy RET-re léphetünk), ha pedig elvégeztük a kért feladatot, ezt C = 0 állítással kell jeleznünk a rendszer számára (ekkor a körbekérdezés leáll). Ilyenkor A-ban is be kell állítanunk a hibakódot (alapesetben 0).

— A hívási memória és regiszterállapot:
— 0. LAP: nulláslap-szegmens (F8H)
— 1. LAP: határozatlan, tetszés szerint használható
— 2. LAP: rendszerszegmens (FFH)
— 3. LAP: a bővítőt tartalmazó szegmens
— C regiszter: akciókód
— B és DE: paraméterek

Ezek a regiszterek megőrzendőek, ha tovább adjuk a hívást az EXOS-nak. A többi regiszter tetszés szerint használható.

Az akciókódok

A továbbiakban a puszta felsoroláson túl utalunk a hívási körülményekre, a beépített egységek által ellátott feladatokra is.

C = 1: Hidegindítás

Az EXOS akkor adja körbe, amikor a vezérlés átvételét várja (pl. a bejelentkezés — villogó ENTERPRISE felirat — után). Először, mint a többi funkciónál is, a RAM-bővítőket hívja, majd sorra a ROM-okat (5., 4., 1. sorrendben). A felhasználói bővítőnek lesz tehát először lehetősége átvenni a vezérlést. Alapkiépítésben az 5. (BASIC) és az 1. szegmens (WP) fogadja a hívást (az utóbbihoz természetesen csak akkor jut el, ha nincs cartridge a gépben).

C = 2: Parancsfüzér

Az előbbi hívás célzott változata: egy adott szó (név) felismerésekor kell átvenni a vezérlést vagy elvégezni a funkciót. Ilyen hívás váltható ki pl. a BASIC ":UK" paranccsal. A bővítőbe lépéskor DE az azonosítandó szöveg előtti hosszbájtra mutat, B pedig az első szó hosszát adja meg. A beépített funkciók:

5. szegmens:

BASIC — a BASIC értelmező veszi át a vezérlést

4. szegmens:

VDUMP — a grafikus képernyő kivitele nyomtatóra
VSAVE — a grafikus képernyő tárolása
VLOAD — a grafikus képernyő beolvasása

(ez utóbbi két funkció hibás, nem működik a forgalomban lévő gépeken)

BRD — német mód

(a karakterkészlet, a billentyűkezelés, a hibaüzenetek németesítése)

UK — angol mód

A fentieken kívül a szegmens tartalmazza még a BASICX funkciót is, ami a BASIC funkcióhívások német kiterjesztését végzi (ez közvetlen hívásnál hibajelzést vált ki, mert nem állít C = 0-t) valamint a "KRAM"&CHR$(255) funkciót a billentyű-RAM inicializálására.

1. szegmens: WP — a szövegszerkesztő veszi át a vezérlést

C = 3: HELP — füzér

Az EXOS a bővítőben kezelt funkciók listájának megadását várja. BASIC-ből a :HELP paranccsal aktivizálható. Ha a HELP után is következik szöveg, ennek címét az előző pont szerint adja meg a rendszer. Ilyenkor csak az adott név felismerése esetén kell (esetleg részletesebb) információt kiírni. A segítő szöveg a 255. csatornára írható, praktikusan a blokkírás EXOS hívással. Alapkiépítésű gép általános HELP-listája az előbbi szolgáltatásokat tartalmazza, és további copyright jellegű üzenetet ír ki a BASIC,WP és az ISL1985 füzérekre.

```
:HELP
BASIC version 2.1
VDUMP Screen dump
VSAVE Screen save
VLOAD Screen load
BRD   German mode
UK    English mode
WP    version 2.1
ok
:HELP BASIC
BASIC version 2.1
      Enterprise IS-BASIC
       1985 Intelligent Software Ltd.
ok
:HELP WP
WP    version 2.1
      Enterprise Word Processor
       1985 Intelligent Software Ltd.

ok
:HELP ISL1985
Copyright INTELLIGENT SOFTWARE LTD

ok
```

C = 4: EXOS változó írása, olvasása, átbillentése

A hívást a rendszer az EXOS 16 funkcióhívás kiterjesztéseként hívja meg 39-nél nagyobb változószám esetén. A ROM-bővítők közül a 4. szegmens kezel a német verzió belapozottságával kapcsolatos jelzőfunkciókra két új EXOS változót (90H és 91H).

C = 5: Hibakódok magyarázata

Ez EXOS funkcióhívások utáni hibakód (A) értelmezésére hívott rutin. A bővítő a hibakódot B-ben kapja meg. Ha hibaüzenetet ad, DE-ben az üzenet címét, B-ben az üzenetet tartalmazó szegmens számát kell visszaadnia és C = 0-t kell állítania.

Az alapértelmezésű gépben először a 4. szegmens kap szót és a kódok egy részére német nyelvű üzenetet ad, ha belapozott a német verzió (egyébként az 1. szegmenshez jut a hívás, ami angol magyarázatokat küld). Ha saját bővítőt illesztünk a rendszerhez, az a ROM-szegmensek előtt kapja meg a hívást, így lehetőségünk van pl. magyar nyelvű hibaüzeneteket adni.

C = 6: Modul betöltése

Akkor hívja meg az EXOS, ha maga nem ismeri fel a beolvasott modulfejrészt. A ROM-bővítők egyike sem válaszol erre a hívásra.

C = 7: RAM-igénylés

Ezt a hívást a hidegindítás során kapják meg a ROM-bővítők, hogy RAM-puffert kérhessenek a rendszerszegmensben. Az alapkiépítésű gépben csak a 4. szegmens kér pufferkiutalást a billentyűzetkezeléshez. Ezt a 2.3.1 pontban tárgyaltuk.

C = 8: Inicializálás

A hidegindítás és az EXOS 0 inicializálás során kapja ezt a hívást minden bővítő. Itt végezheti el a hozzá tartozó rendszer egyértelmű alaphelyzetbe állítását. A ROM-szegmensek közül itt is csak a 4. szegmens reagál a hívásra a német karakterkészlet, stb. kialakításával.

*Rendszerbővítő kialakítása*

Az elmondottak gyakorlati bemutatására készítsünk egy egyszerű bővítőt és illesszük a rendszerbe. Illusztrációnak szánt kiterjesztésünk feladata legyen pl. az EXOS hibakódok magyar nyelvű értelmezése és az ezzel kapcsolatos alapműveletek elvégzése.

Bővítőnket a fentieknek megfelelően a szegmens C00AH címén (a 3. LAP tartományában C00AH belépési címmel) kezdjük el. Mint az assembly listában követhető, a vezérlés a C (akciókód) különböző értékeinél különböző programpontokra kerül. Itt most nem minden végrehajtó rutinban történik érdemleges művelet. Bővítőnk nem veszi át a vezérlést (nem kíván aktuális program lenni) így a hidegindítás kódjára (1) csak egy RET-tel válaszolunk. Ugyanígy azonnal visszatérünk az "EXOS változó"(4), "Modul betöltése"(6) és "RAM-igénylés"(7) hívásokból is (természetesen a bővítő ezeken a pontokon is igény szerint kiterjeszthető).

A ténylegesen ellátott funkciók a következők:

— "Help"(3) hívásra a rendszer megadja a szolgáltatás rövid megjelölését: "HUN Magyar mód".

— Ezt a neki címzett "Help" hívásra (:HELP HUN) kiegészíti: "HUN Magyar EXOS hibaüzenetek" és megadja a pillanatnyi üzemmódot is: "A jelenlegi mod:" BRD vagy UK vagy HUN.

— A magyar üzemmód a parancsfüzér (2) hívással aktivizálható (BASIC-ből a :HUN paranccsal). Ezt a módot a FLAG változó 0 értéke jelzi.

— Az érdemi funkció a "Hibakódok magyarázata" (5) hívásnál történik. Ha FLAG = 0, a rutin az üzenettáblából (TABLA) kikeresi az aktuális üzenetszöveget. Ha nincs ilyen, a hívást továbbadja a rendszernek.

— Az "Inicializálás" (8) rutin (BASIC és WP közti átlépés, RESET stb.) törli a magyar módot (FLAG = 01) az egyértelmű alaphelyzet érdekében.

A bővítő összeállítása és a rutinok működése a megjegyzésekkel ellátott assembly listán követhető. Kiegészítésként csak annyit jegyzünk meg, hogy a HP-TEXT a BRD, ill. UK módot a BASIC RST 10H rutin különbözőképpen betöltött részletéről ismeri fel.

A lista végén helyeztük el az üzenettáblát (ill. annak csak önkényesen kiválasztott két sorát). Egy-egy elem szerkezete: EXOS hibakód, karakterek száma, szöveg. A tábla természetesen tetszés szerint folytatható.

```
        ORG 0C00AH          :belépési cím

        LD  A,C             :akciókód
        DEC A               :C=1 volt?
        JR  Z,HIDEG         :-> hidegindítás
        DEC A               ;C=2?
        JR  Z,PARANCS       :-> parancsfüzér
        DEC A               :C=3?
        JR  Z,HELP          :-> Help-füzér
        DEC A               :C=4?
        JR  Z,VALTOZO       :-> EXOS változó
        DEC A               :C=5?
        JR  Z,HIBA          :-> hibakódok magyarázata
        DEC A               :C=6?
        JR  Z,MODUL         :-> modul betöltése
        DEC A               :C=7?
        JR  Z,RAM           :-> RAM-igénylés
        DEC A               ;C=8?
        JR  Z,INIC          :-> inicializálás
        RET                 : vissza, ha egyik sem
FLAG    DEFB 0              :mód-flag (FLAG=0, ha HUN)

HIDEG   RET                 :nincs funkció

PARANCS PUSH DE             ;regiszterek
        PUSH BC             :  verembe
        CALL AZONOS?        ;a füzér="HUN"?
        JR   NZ,VEGE        :vissza, ha nem
        XOR  A              ;HUN jelzőbájt
        LD   (FLAG),A       ;a flag-be
VEGE_OK POP  BC             :regiszterek
        POP  DE             :  vissza
        LD   C,0            :"akció elvégezve"
        XOR  A              :"hiba nincs"
        RET

VEGE    POP  BC             :regiszterek
        POP  DE             :  vissza
        RET                 :továbbadja a hívást

HELP    PUSH DE             :regiszterek
        PUSH BC             :  verembe
        LD   A,B            ;A=füzér hossza
        OR   A              :A=0? (":HELP"?)
        JR   NZ,HP_TEXT     ;további vizsgálatra, ha
                             a füzér folytatódik
        LD   A,255          ;aktuális csatorna
        LD   DE,SZOVEG      :kiírandó szöveg kezdőcíme
        LD   BC,S_HOSSZ     : és hossza
        EXOS 8              :blokkírás
        JR   VEGE           ;befejezésre ugrik
```

```
VALTOZO RET                 ;nincs funkció

HIBA    LD   A,(FLAG)       ;mód-flag
        OR   A              ;A=0? (HUN?)
        RET  NZ             ;nem kezeli, ha nem HUN mód
        PUSH DE             ;regiszterek
        PUSH BC             ;  verembe
        LD   DE,0           ;eltolt
        LD   HL,TABLA-1     ;üzenettábla kezdőcíme-1
HI1     INC  HL             ;üzenet első bájt
        ADD  HL,DE          ;a következő üzenet címe
        LD   A,(HL)         ;EXOS hibakód
        OR   A              ;A=?
        JP   Z,VEGE         ;tábla vége, ha igen
        INC  HL             ;üzenethossz címe
        LD   E,(HL)         ;E=üzenet hossza
        CP   B              ;ez a keresett kód?
        JR   NZ,HI1         ;a következő üzenetre, ha nem
        POP  BC             ;regiszterek
        POP  DE             ;  vissza
        LD   C,0            ;"akció elvégezve"
        IN   A,(0B3H)       ;saját szegmensszám
        LD   B,A            ;B-be (az üzenet szegmense)
        EX   DE,HL          ;DE: az üzenet kezdőcíme
        RET

MODUL   RET                 ;nincs funkció

RAM     RET                 ;nincs funkció

INIC    LD   A,255          ;"Magyar mód törölve"
        LD   (FLAG),A       ;kód a jelzőbájtba
        RET


AZONOS? INC  DE             ;füzér első karakter címe
        LD   HL,NEV         ;bővítőnév címe
        LD   B,N_HOSSZ      ;bővítőnév hossza
A1      LD   A,(DE)         ;füzérkarakter
        CP   (HL)           ;=névkarakter?
        RET  NZ             ;"nem azonos", ha eltérnek
        INC  DE             ;következő
        INC  HL             ;  karakterek címe
        DJNZ A1             ;végigvizsgál
        RET


HP_TEXT CALL AZONOS?        ;("HELP HUN"?)
        JR   NZ,VEGE        ;vissza, ha nem
        LD   A,255          ;aktuális csatorna
        LD   DE,UZENET1     ;üzenet első rész címe
        LD   BC,U_HOSSZ     ;  és hossza
        EXOS 8              ;blokkírás
        LD   A,(FLAG)       ;mód-flag
        OR   A              ;A=0?
        LD   DE,.HUN        ;" HUN" szövegre
        JR   Z,KIIRAS       ;ha HUN mód
                            ;(egyébként BRD vagy UK)
        LD   A,(00B8H)      ;RST 10h részlet
```

```
        CP    3EH          ;csak UK módban ilyen
        LD    DE,.UK       ;" UK " szövegre
        JR    Z,KIIRAS     ;ha UK mód
        LD    DE,.BRD      ;egyébként " BRD" szöveg
KIIRAS  LD    BC,6         ;hossz
        LD    A,255        ;aktuális csatorna
        EXOS  8            ;blokkírás
        JP    VEGE_OK      ;a befejezésre ugrik


SZOVEG  DEFB "HUN   Magyar mod",13,10
S_HOSSZ EQU 18

UZENET1 DEFB "HUN   Magyar EXOS-hibaüzenetek",13,10
        DEFB "      A jelenlegi mod: "
U_HOSSZ EQU 55

NEV     DEFB "HUN".
N_HOSSZ EQU  3

.HUN    DEFB " HUN",13,10
.BRD    DEFB " BRD",13,10
.UK     DEFB " UK ",13,10

TABLA   DEFB 255,24,"Nem megengedett EXOS kod"
        DEFB 251,20,"Csatorna nem letezik"
        DEFB 0
```

Ezt a bővítőt már nemcsak kényelmi szempontból érdemes egy jó assembler program segítségével megírni. Az ASMON szolgáltatása pl., hogy a lefordított bájtsorozat (OBJECT CODE) magnetofonra vihető 6-os modulfejjel (abszolút rendszerbővítő) l. a 9. Függeléket. A magnetofonról később beolvasott file számára az EXOS szegmenst foglal és automatikusan beláncolja a bővítő nyilvántartásba. A BASIC-ből felépített változatnál mindezt magunknak kell megtennünk.

```
100 ALLOCATE 100
110 CODE M=HEX$("3E,FF,D3,B2,21,79,
    BF,34,E5,F7,18,E1,35,79,D3,B1,
    26,00,6F,C9")
120 LET SZEGMENS=USR(M,0)
130 POKE 540,10
140 POKE 541,64
150 CODE BELEPES=HEX$("79,3D,28,17,
    3D,28,15,3D,28,26,3D,28,35,3D,
    28,33,3D,28,52,3D,28,50,3D,28,
    4E,C9")
160 CODE FLAG=CHR$(255)
170 CODE HIDEG=HEX$("C9")
180 CODE PARANCS=HEX$("D5,C5,CD,77,
    C0,20,0A,AF,32,24,C0,C1,D1,0E,
    00,AF,C9,C1,D1,C9")
190 CODE HELP=HEX$("D5,C5,78,B7,20,
    45,3E,FF,11,B4,C0,01,12,00,F7,
    08,18,EB")
200 CODE VALTOZO=HEX$("C9")
210 CODE HIBA=HEX$("3A,24,C0,B7,C0,
```

```
    D5,C5,11,00,00,21,11,C1,23,19,
    7E,B7,CA,37,C0,23,5E,B8,20,F4,
    C1,D1,0E,00,DB,B3,47,EB,C9")
220 CODE MODUL=HEX$("C9")
230 CODE RAM=HEX$("C9")
240 CODE INIC=HEX$("3E,FF,32,24,C0,
    C9")
250 ! SEGEDRUTINOK ES TABLAK
260 CODE AZONOS=HEX$("13,21,FD,C0,
    06,03,1A,BE,C0,13,23,10,F9,C9")
270 CODE HP_TEXT=HEX$("CD,77,C0,20,
    AD,3E,FF,11,C6,C0,01,37,00,F7,
    08,3A,24,C0,B7,11,00,C1,28,0D,
    3A,B8,00,FE,3E,11,0C,C1,28,03,
    11,06,C1,01,06,00,3E,FF,F7,08,
    C3,31,C0")
280 CODE SZOVEG="HUN...Magyar mod"&
    HEX$("0D,0A")
290 CODE UZENET1="
    HUN...Magyar.EXOS.hibauzenetek"&
    HEX$("0D,0A")
300 CODE =".....A.jelenlegi.mod:."
310 CODE NEV="HUN"
320 CODE HUN=".HUN"&HEX$("0D,0A")
330 CODE BRD=".BRD"&HEX$("0D,0A")
340 CODE UK=".UK."&HEX$("0D,0A")
350 CODE TABLA=HEX$("FF,18")&"Nem
    megengedett EXOS kod"
360 !...
370 CODE =HEX$("FB,14")&"Csatorna
    nem letezik"
380 !...
400 !BELANCOLAS
410 POKE 49088,10
420 POKE 49089,64
430 POKE 49090,SZEGMENS
```

A BASIC program középső része (150—370. sorok) betölti a bővítő rutinjait és tábláit az 1. LAP-ra lapozott szegmensbe. Ezzel kapcsolatban csak egy megjegyzésünk van: mivel a betöltő rész rögzített tábla- és szövegcímeket használ, a különböző kiírandó szövegeket pontosan a megadott formában (ill. azonos karakterszámmal) kell beírni. Ezért a kritikus 280—340. sorokba a szóközök helyére pontokat írtunk (így egyszerűbben leszámolható). A hibaüzenet-tábla már tetszés szerinti üzeneteket tartalmazhat a következő formában:

1. bájt: EXOS hibakód
2. bájt: az üzenet karaktereinek száma

szöveg

A bővítő csak azokra a hibakódokra ad magyar nyelvű üzenetet, amelyeket beépítettünk a táblába.

A program első és záró része két érdemi funkciót lát el: a szegmenslefoglalás és a rendszerbe illesztés műveletét.

A bővítő aktív marad egyes inicializálások (l. EXOS 0) után is, de a felhasználó számára lefoglalt szegmens felszabadul egy-egy ilyen műveletnél. Ilyenkor két lehetőség közül választhatunk.

1. Nem foglalkozunk a felhasznált szegmens lefoglalásával, mert — programunk méretének ismeretében — tudjuk, hogy a rendszer úgysem éri el a mi szegmensünket (pl. az FBH szegmenst).

2. Felhasználjuk, hogy a perifériáknak lefoglalt szegmensek nem szabadulnak fel nagyobb fokú inicializálás (pl. BASIC-ből WP-be lépés) során sem. Ehhez a rendszer mélyebb ismerete szükséges: az a módszer, ahogy az EXOS maga jelzi a perifériaműveleteket. Az operációs rendszer a szegmenskiutalás során — mint láttuk — közvetve a BF79H rendszerváltozó értéke alapján ismeri fel a perifériáktól érkező hívásokat. Ezek alapján egyszerű a rendszert becsapni:

— lapozzuk a rendszerszegmenst a 2. LAP-ra,
— állítsuk a fenti változót "periféria" jelzésre,
— kérjük az EXOS-tól egy szegmenskiutalását,
— állítsuk vissza a fenti változót,
— lapozzuk a kapott szegmenst az 1. LAP-ra,
— adjuk vissza a szegmenssźámot a BASIC-nek is.

```
LD A,FFH        3E FF
OUT (B2H),A     D3 B2
LD HL,BF79H     21 79 BF
INC (HL)        34
PUSH HL         E5
EXOS 18H        F7 18
POP HL          E1
DEC (HL)        35
LD A,C          79
OUT (B1H),A     D3 B1
LD H,0          26 00
LD L,A          6F
RET             C9
```

A szegmens lefoglalása és a bővítő betöltése után már csak a rendszerbe illesztés van hátra. Ha minden inicializálást (kivéve a hidegindítást) túlélő bővítőt szeretnénk, ágyazzuk be a szegmenst a ROM-könyvtárba (l. 2.5.2 pont).

Mi most a 410—430. sorokban a legegyszerűbb beláncolást alkalmaztuk: a BFC3H báziscím alá beírtuk a belépési cím alsó és felső bájtját (az 1. LAP tartományában!), majd a bővítő szegmensét. Ha már volt a rendszerben előzőleg RAM-bővítő, ezt a módszert ki kell egészíteni valódi láncolássá: a leírás előtt ki kell olvasni a báziscím alatti pointert és saját bővítőnk belépési címe előtt kell letárolni a 3 bájtot, a 4007H, 4008H, 4009H címekre.

A beillesztés után a bővítő aktívan kapcsolódik a rendszerhez, mint azt a következő képernyőmásolat is szemlélteti:

```
:HELP
HUN    Magyar mod
BASIC  version 2.1
VDUMP  Grafik ausdrucken
VSAVE  Grafik absoeichern
VLOAD  Grafik laden
BRD    Deutsche Arbeitsweise
UK     Englische Arbeitsweise
WP     version 2.1
ok

:HELP HUN
HUN    Magyar EXOS-hibauzenetek
       A jelenlegi mod:  BRD
ok
PRINT #1:1

*** Kanal nicht vorhanden.

:UK
ok
:HELP HUN
HUN    Magyar EXOS-hibauzenetek
       A jelenlegi mod:  UK
ok
PRINT #1:1

*** Channel does not exist.

:HUN
ok
:HELP HUN
HUN    Magyar EXOS-hibauzenetek
       A jelenlegi mod:  HUN
ok
PRINT #1:1

*** Csatorna nem letezik.
```

# 3. A BASIC rendszer

Az ENTERPRISE személyi számítógéphez modulként csatlakoztatható IS-BASIC értelmező tulajdonképpen felhasználói program (az előző fejezet szóhasználatával: rendszerbővítő), de kétségtelenül a legszélesebb körben alkalmazott a felhasználói programok körében. A legtöbb géptulajdonos éppen a BASIC rendszerből kiindulva, annak lehetőségeit kiterjesztve foglalkozik (vagy szeretne foglalkozni) a gépi kódú programozással. Ismerkedjünk meg tehát az EXOS után a BASIC felépülésével, szerkezetével, működésével is, hogy biztosan támaszkodhassunk rá mint környezetre, felhasználhassuk rutinjait, szolgáltatásait.

## 3.1 Inicializálás

A BASIC memóriatérképéről, felépüléséről itt is az inicializálás folyamatát követve nyerhetjük a legtöbb információt.

A BASIC mint rendszerbővítő, két esetben veszi át a vezérlést. A rendszer inicializálása után — mint láttuk — a hidegindítás akciókódja jár körbe a bővítők között, ill. nem jár körbe, mert a BASIC szegmens (rendszerünkben az 5-ös) ennek hatására aktuális programmá válik és elkezdi saját inicializálását. Ugyanez történik akkor is, ha 2-es akciókódot indít körbe egy másik felhasználói program (vagy akár maga a BASIC), méghozzá BASIC parancsfüzérrel (ez történik, ha kiadjuk a :BASIC parancsot vagy szövegszerkesztőben megnyomjuk az F8 gombot stb.). A füzér felismerése után a két eset "összetalálkozik" és egységesen hozza létre az értelmező alapállapotát:

```
C0A4  F7 00       EXOS 00          ;rendszer alaphelyzet
C0A6  31 00 28    LD   SP,2800     ;verem az 1. LAP-on
C0A9  FB          EI               ;megszakítás eng.
C0AA  CD C6 C0    CALL C0C6        ;rutinok a nulláslapra
C0AD  C3 8A C2    JP   C28A        ;folytatásra
```

Itt álljunk meg egy pillanatra. A meghívott rutin a nulláslapra másolja a ROM-ból a BASIC nélkülözhetetlen RST rutinjait, érintetlenül hagyva természetesen a hasonló módon odatöltött EXOS rutinokat (a RST 30H-t és a megszakításkezelő bevezetőjét). Az új RST rutinokkal — köztük az értelmező talán lefontosabb rutinrendszerével, a RST 10H BASIC funkcióhívással — külön pontban foglalkozunk. A hívott rutin ezután beírja a szoftver megszakításkezelőjének címét (00FFH) a SOFT-ISR változóba (a 003D/EH címekre), majd saját szegmensszámát (a 3. LAP-ról kiolvasva) a 00C5H címen tárolja. Ez a momentum fontos lesz még, ha egy RAM-ba töltött "fantom" BASIC-et akarunk a rendszerbe illeszteni.

A nulláslap inicializálása után az indítási főág a BASIC munkaterületeket készíti elő. Először nullázza a rendszerváltozó és táblaterületeket, majd betölti a működés alapjául szolgáló munkatáblákat:

```
C28A  21 00 02     LD   HL,0200     ;változóterület kezd.
C28D  01 54 0E     LD   BC,0E54     ;hossz
C290  AF           XOR  A           ;A=0
C291  77           LD   (HL),A      ;0-val feltölt
C292  ED A1        CPI              ;HL=HL+1
                                     BC=BC-1: BC=0?
C294  EA 91 C2     JP   PE,C291     ;tovább nulláz
C297  22 46 02     LD   (0246),HL   ;végcím változóba
C29A  DD 21 00 02  LD   IX,0200     ;BASIC bázis:=m
C29E  CD 63 ED     CALL ED63        ;HL=működő RAM szeg-
                                        mensek száma (n)
C2A1  DD 75 0C     LD   (IX+0C),L   ;változóterületre
C2A4  29           ADD  HL,HL       ;HL=2*n
C2A5  11 54 0E     LD   DE,0E54     ;RAM-térkép kezdőcím
C2A8  19           ADD  HL,DE       ;2*n bájtot kihagy
C2A9  CD 79 F5     CALL F579        ;utána tölti a token-
                                     és függvénytáblát
```

A kulcsszótáblával és a függvények táblázatával, mint a memóriatérkép igen lényeges elemeivel, a 3.2 alfejezetben foglalkozunk részletesen.

A táblák végén tulajdonképpen kezdődhet az aktív BASIC munkaterület. Be is tölti ezt a címet a program a megfelelő rendszerváltozókba, de ez még nem a végleges érték. Az értelmező most ui. meghívja a BASICX rendszerbővítőt, lehetőséget adva ezzel a német verziónak a szükséges változtatásokra:

```
C2AC  22 1E 02     LD   (021E),HL   ;munkaterület kezdőcím
C2AF  22 1C 02     LD   (021C),HL   ;        a
C2B2  22 20 02     LD   (0220),HL   ; rendszerváltozókba
C2B5  11 0A FA     LD   DE,FA0A     ;"BASICX" szöveg címe
C2B8  F7 1A        EXOS 1A          ;bővítők lekérdezése
```

Ez a változtatás gyakorlatilag a RST 10H rutin eltérítését jelenti: a 4. szegmens egy kis rutint ragaszt az imént kialakított táblaterület végéhez, amely a német verzióban sajátos kezelést igénylő BFH BASIC funkcióhívást (hibaüzenet kiírása) saját kezelőjére irányítja (l. 3.3 alfejezet)

A munkaterület kezdőcíme tehát néhány bájttal hátracsúszott (ezt be is jegyzi a bővítő), mire a vezérlés visszakerül a főágra. A következő lépések a törölt BASIC alapállapotot hozzák létre. Nulla program-hossznak megfelelően állítják be a programterület utáni BASIC változóterület mutatóit és a programterület kezdetére állítják a progamszöveg mutatókat (pl. a DATA-mutatót). Itt történik meg a LOGO alapirány (90 fok) beállítása és a szükséges csatornák megnyitása a BASIC alapértelmezés szerint:

KEYBOARD: 69H (105D)
VIDEO: 66H (102D)
EDITOR: 00H (0D)

SOUND: 67H (103D)
NYOMTATÓ: 68H (104D)

Az előkészítés befejezéseként a rutin kiírja az INFO sorokat, majd a grafikus lap alapértelmezéséhez GRAPHICS 4-nek megfelelő változókat állít be:

```
C2BA  CD 90 ED     CALL ED90        ;RAM-térkép vizsgálat
C2BD  CD C7 CC     CALL CCC7        ;(NEW)
C2C0  D7 92 07     BASIC 92,07      ;PI/2 -> MREG
C2C3     96 19 0E  BASIC 96,0E19    ;MREG -> LOGO irány
C2C6     00        BASIC 00         ;funkcióhívások vége
C2C7  CD D8 F1     CALL F1D8        ;csatornák init.
C2CA  CD 5B D8     CALL D85B        ;INFO kiírás
C2CD  3E 01        LD   A,01        ;GRAPH. alapértékek
C2CF  32 0F 02     LD   (020F),A    ;MODE=1 (nagyfelb.)
C2D2  32 10 02     LD   (0210),A    ;COLOR=1 (4 szín)
```

A BASIC-ként megadott utasításokat, a RST 10H által kiváltott funkciókat, egyelőre fogadjuk el az adtott formában. Ezekkel a 3.3 alfejezetben foglalkozunk. Az inicializálás utolsó mozzanataként még egy fontos lépés van hátra: a melegindítási címet 0104H-re állítja az értelmező, majd újra inicializálja a nulláslapon felépített RST rutinokat (ide már máshonnan is léphet a rendszer, tehát ez nem szükségtelen művelet):

```
C2D5  DB B2        IN   A,(B2)      ;pillanatnyi 2. LAP
C2D7  F5           PUSH AF          ;szegmense verembe
C2D8  3E FF        LD   A,FF        ;rendszerszegmens
C2DA  D3 B2        OUT  (B2),A      ;a 2. LAP-ra
C2DC  21 04 01     LD   HL,0104     ;melegindítási cím
C2DF  22 F8 BF     LD   (BFF8),HL   ;RST_ADDR változóba
C2E2  F1           POP  AF          ;szegmens vissza
C2E3  D3 B2        OUT  (B2),A      ;eredeti érték
C2E5  CD C6 C0     CALL C0C6        ;nulláslap inic.
C2E8  DD CB 00 C6  SET  0,(IX)      ;"ok" írható
```

Ezzel a BASIC rendszer létrehozta a megfelelő alapállapotot és a végrehajtás a beolvasási főágba kerül. Mielőtt rátérnénk az alapállapot (változóterületek és táblák), majd a főágak konkrét megismerésére, tegyünk még 1-2 megjegyzést az eddig elmondottakkal, azok alkalmazásával kapcsolatban.

Az egyik érdekes momentum, hogy az inicializálás folyamán a rendszernek nem kellett tudnia, hogy melyik szegmensen fut. Az EXOS azt a szegmenst lapozta a 3. LAP-ra, amelyikben a bővítőt találta, az pedig innen olvassa be (és tárolja egyébként a 00C5H címen) saját szegmensszámát. Ha tehát pl. az egész 5. szegmenst átmásoljuk egy RAM-szegmensbe, majd ezt bővítőként beláncoljuk a rendszerbe (l. a 2.6 alfejezetet), a :BASIC utasításra ez a RAM-BASIC építi fel a rendszert, ebben fut a teljes értelmező. Egy RAM-BASIC pedig hallatlanul érdekes terület a kísérletezések számára.

Hasonlóan érdekes változtatási lehetőségekhez jutunk akkor is, ha egy saját "BASIC" bővítőt illesztünk a rendszerhez, hiszen mint láttuk, ezt meghívja az értelmező az inicializálás során.

Végül ténylegesen alkalmazzuk is a megszerzett információkat egy komolyabb beavatkozásra. Szerezzünk a BASIC munkaterület előtt egy olyan RAM-területet, amelyet szabadon használhatunk, amely érintetlenül marad programtörlésekkor, beolvasáskor stb. Egy védett terület a mindig belapozott nulláslapon kincset ér további munkánk során. A megoldás egyszerű: állítsuk HL-t a munkaterület új kezdőcímére, majd lépjünk be az inicializálási főágba és bízzuk a rendszerre a változások végigvitelét:

```
LD HL,2000H     21 00 20
LD A,5          3E 05
OUT (B3H),A     D3 B3
JP C2ACH        C3 AC C2
```

BASIC-ből betöltve:

```
100 ALLOCATE 100
110 CODE M=HEX$("21,00,20,3E,05,D3,
    B3,C3,A2,C2")
120 CALL USR(M,0)
```

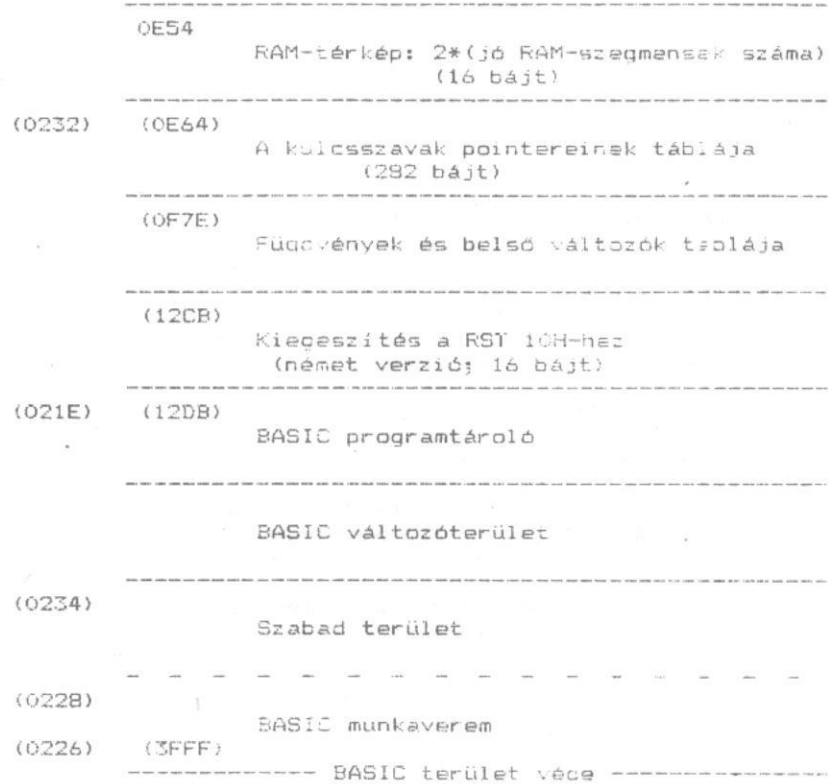
Az INFO paranccsal azonnal meggyőződhetünk róla, hogy a futtatás után (ami persze törli a rutint is) a BASIC kevesebb bájttal dolgozik: hátracsúszott. A programszöveg régi és új kezdőcíme közötti RAM most már a miénk (a következő inicializálásig). Használhatjuk pl. megszakítási rutin tárolására (amelyet igen praktikus a nulláslapon elhelyezni), a BASIC utasítások körének bővítésére stb.

## 3.2 Memóriafelosztás, változóterületek

Mint láttuk, az értelmező az inicializálás során alulról fölfelé építi fel a változóterületeit, tábláit. Vizsgáljuk tehát végig mi is ilyen sorrendben a különböző funkciójú tartományokat, méghozzá előbb egy átfogó térképen áttekintve, majd konkrétan is bemutatva az egyes területek határait, feladatait, szerkezetét. Példaként továbbra is a 128 kbájtos, angol/német verziót vizsgáljuk. Az egyes részeknél zárójelben megadott konkrét értékek, címek erre a változatra vonatkoznak.

A nulláslap elején felépített RST rutinokat külön vizsgáljuk a 3.3 alfejezetben. Most a 0200H címen kezdődő rendszerváltozó területtől tekintsük át a memória-térképet:

| | |
|---|---|
| 0200 | BASIC rendszerváltozók |
| 0248 | Szövegpuffer |
| 0347/8 | Mutató a szövegpuffer éppen feldolgozott elemére |
| 0379 | Munkaterület |
| 0446 | Puffer a hibaüzenetekhez |
| 0496 | Szövegpuffer az INPUT-hoz<br>Ideiglenes tároló memóriaeltolásoknál |
| 0595 | Numerikus puffer; munkaterület az elemzéseknél |
| 059E | Puffer a kiírandó számok ASCII alakjához |
| 05B2 | Numerikus puffer; munkaterület a számkonverziónál |
| 0DBA | Z80-as verem |
| 0DBB | Munkaterület (magnetofonműveletek) |
| 0DCB | Változók bázistáblája: 32*2 bájt |
| 0E0B | 1. rögzített numerikus munkaregiszter (REG1) 7 bájt |
| 0E12 | 2. rögzített numerikus munkaregiszter (REG2) 7 bájt |
| 0E19 | LOGO irány: 7 bájt (alapérték PI/2) |
| 0E20 | Általános munkaterület |




A térkép természetesen hexadecimális formában tartalmazza a határok címadatait. A táblázatban bal oldalt megadott adatok a rendszerváltozókra utalnak, amelyek az adott címet tartalmazzák, míg a zárójeles értékek az adott verzióban kialakuló címek, ill. adatok.

### 3.2.1 Rendszerváltozók

A változó területek, táblák kiíratásához jól használhatjuk — esetenként némi módosítással — a 2.3.1 pontban kidolgozott 2.9 számú DUMP-programot. Ezzel a rendszerváltozók területe:

```
START
KEZDŐCIM ? 0200
VEGCIM ? 0247
SZEGMENS ? F8
 512  0200  AB 00 00 00 01 00 00 00
 520  0208  00 00 00 00 08 00 00 01
 528  0210  01 00 00 00 24 15 09 15
 536  0218  00 00 DB 12 DB 12 DB 12
 544  0220  DB 12 00 00 00 00 FF 3F
 552  0228  CD 3F 0C 00 DB 12 E1 12
 560  0230  4B 02 64 0E D0 16 56 15
 568  0238  A9 17 B7 0D 0D 24 48 02
 576  0240  00 00 DD BE 00 00 11 02
ok
```

A változók egy része a feldolgozás, végrehajtás közben adódó paraméterek ideiglenes tárolására szolgál, többségüket viszont mi is használhatjuk gépi kódú programjainkban a BASIC rutinok hívásakor vagy saját bővítő készítésekor. Tekintsük át a fontosabb változók funkcióit (hexadecimális címekkel):

0200: Jelzőbájt. Bitjei az értelmező működésének különböző fázisaiban jelzik a végrehajtó rutinok, főágak számára a változásokat, üzemmódokat. Például a 4. bit a TRACE üzemmódot vezérli (ha 1, aktivizálódik a nyomkövető üzemmód), a 2. bit 1 értéke STOP-ot vált ki stb.

0201: Jelző; 0. bitje a szögek megadási módját jelöli ki (ha 0: radián, ha 1: fok)

0202: A BASIC sor elemzése során használt változó: a vizsgált elem típusa (erről még lesz szó a tokenizálás ismertetésénél és a BASIC 20H funkcióhívásnál)

0203: A vizsgált sorelem (pl.) változónév) hossza

0204: Jelzőbájt a sorelemzésnél. Műveleti jeleknél, sorvégnél a tárolt kód értéke

0205: A billentyűzetről beolvasott karakter ideiglenes tárolója

0206: A végrehajtás utáni teendők jelzőbájtja. A végrehajtó rutinok saját típusuknak megfelelően állítják be (vagy hagyják meg a 0 alapértéket). A kiváltott funkciók:

0: a következő bájton folytatja a végrehajtást, kettőspontot elfogad a sorban (pl. LET)

1: sorvégnek kell következnie

2: a következő sor elején folytatja a végrehajtást (pl. AUTO, DATA stb.)

3: a 0216/7H sorra ugrik (pl. GOTO)

4: az END DEF jelzi ezzel a visszatérést a végrehajtási főágba a szubrutinszerűen meghívott DEF alprogramról

0208: Trace csatorna. Közvetlenül is állítható

0209: Aktuális csatornaszám a kiíratásokhoz. Alapértéke 0 (EDITOR)

020A: Az aktuális program száma (alapérték: 0)

020B: A megosztott szegmens száma, ha már kiutalt ilyet az EXOS (alapérték: 0)

020C: Működő RAM-szegmensek száma (alapérték: 8). Ez maximálja a nyitható programok számát

020D: A megosztott szegmenst kapott program száma (alapérték: 0; nincs ilyen program)

020E: TEXT-mód jelző. Ha 0:40 karakteres mód, ha 2:80 karakteres mód. Közvetlenül is írható, hatása a következő TEXT-nél érvényesül

020F: A grafikus lap VIDEO MODE aktuális értéke (alapérték: 1; HIRES)

0210: A grafikus lap VIDEO COLOUR aktuális értéke (alapérték: 1; 4 szín)

0211: RND alapszám

0212: A futó program száma

0214/5: Pointer: a feldolgozás alatt álló BASIC sor vizsgált elemére mutat

0216/7: Pointer: a feldolgozás alatt álló sor elejére mutat

0218/9: Kétbájtos egész. A BASIC sorszámként értelmezhető számkonstans (pl. GOTO argumentum) tárolója

021A/B: Az aktuális program kezdőcíme. Esetünkben az érték: 12DB, tehát az inicializálás során felépített táblák utáni első szabad cím. A 0 program általában itt kezdődik, de innen hátratolhatja egy ALLOCATE utasítás vagy egy áthelyezhető modul betöltése. Az összes többi program a 4000H címen (az 1. LAP-on, egy másik szegmensben) kezdődik

021C/D: A CODE utasítás következő szabad pozíciója (a legközelebbi CODE ettől a címtől kezdve fogja tárolni a bájtokat). Közvetlen beállításának lehetőségét már eddig is számos példaprogramban alkalmaztuk. Alapértéke a BASIC programterület kezdete (12DB)

021E/F: A BASIC programterület kezdőcíme (12DB)

0220/1: A ALLOCATE-tel lefoglalható munkaterület kezdőcíme (általában megegyezik az előző értékkel)

0226/7: A BASIC terület felső határa. Programszámtól és mérettől függő érték. Esetünkben: 3FFF, a 0 LAP felső határa. Ha a program mérete (és a változók számára lefoglalt terület) nő, átlépheti a szegmenshatárt, ekkor a felső határ 7FFF, majd BFFF lesz (legalábbis ha másodszorra is teljes szegmenst kapott). Természetesen az 1. LAP-on induló, 0-tól különböző számú programnál már az alapérték 7FFF

0228/9: Pointer: a felülről lefelé bővülő BASIC munkaverem aljára, egyben aktuális elemére mutat

022A/B: A szükséges memóriaméret ideiglenes tárolója

022C/D: READ mutató: a következő olvasható elem sorának elejére mutat (alapérték a BASIC terület kezdőcíme)

022E/F: READ mutató: a következő adatra mutat

0232/3: A kulcsszótábla kezdőcíme (0E64)

0234/5: A szabad terület kezdőcíme. Ez adja a következő változó tárolási címét

0236/7: Ideiglenes kezdőcím a lokális változók tárolásához a DEF rutinok futása alatt

0238/9: A változók számára potenciálisan lefoglalt terület felső határa. Valójában innen számítható a szabad terület

023A/B: A stack-pointer (SP) ideiglenes tárolója a futás során

023C/D: A legutóbbi hiba kétbájtos egészként tárolt típusa (EXTYPE)

023E/F: A legutóbbi hibát kiváltó sor címe a memóriában. Esetünkben (0248H) a hiba az input-sorban történt, az előző változó aktuális értéke (240DH = 9229D) szerint ez egy STOP volt

0242/3: Puffercím a memóriába íráshoz. Az aktuális érték (BEDDH) éppen az állapotsor egyik pozíciója (végcím—2)

0246/7: Kétbájtos érték az RND számításhoz. A 0211 változóval együtt használja a rendszer

### 3.2.2 Pufferterületek

A feldolgozás, kiértékelés fázisaiban használt pufferek számunkra jórészt érdektelenek, a kezelőik magánügyei. Közvetlen felhasználásra leginkább az üzenet- és bemeneti puffereknél kerülhet sor. A 0248H címen kezdődő bemeneti pufferrel, a beírt sor feldolgozásával a tokenizálás során foglalkozunk. A hibaüzenet-puffert viszont bemutatjuk egy szándékosan kiváltott hiba után:

```
NEXT

*** Programm-Befehl im Direktmodus
unzulassig.
START
KEZDŐCIM ? 0446
VEGCIM ? 0476
SZEGMENS ? F8
 1094   0446   2A 20 50 72 6F 67 72 61    :. Progra
 1102   044E   6D 6D 2D 42 65 66 65 68    :mm-Befeh
 1110   0456   6C 20 69 6D 20 44 69 72    :l im Dir
 1118   045E   65 6B 74 6D 6F 64 75 73    :ektmodus
 1126   0466   20 75 6E 7A 75 6C 7B 73    : unzulas
 1134   046E   73 69 67 00 00 00 00 00    :sig.....
 1142   0476   00 00 00 00 00 00 00 00
ok
```

### 3.2.3 A változók bázistáblája

A rögzített (0DCBH) címen kezdődő táblázat 32 db kétbájtos címet tartalmaz. Ezek a címek 32 párhuzamosan induló változólánc elejére mutatnak. Innen a változó tárolására szolgáló terület első két bájtjaként egy újabb pointer: a következő láncelem címe olvasható ki. Így folytatódnak a láncok mindaddig, amíg a pointer helyén 0000-t nem találunk.

Ezt a láncrendszert a BASIC építi fel az inicializálás során a belső függvényekből és változókból (mint pl. az ABS, ill. a PI). Az értelmező ezekhez a láncokhoz fűzi hozzá az értékadó műveleteknél az újabb felhasználói változókat. A báziscímről kiemeli az előző pointert, beírja az új változó tárolási címét (ami a BASIC változóterület végén lesz), majd az új elem első két bájtjára betölti a báziscímről kimentett pointert. Így a lánc nem szakad meg, a legutóbb felvett változó kerül a legelső helyre (így azt találja meg legelőször az értelmező) és utána következnek a lánc eredeti elemei.

Azt, hogy a változó a párhuzamos ágak közül melyikbe kerül, a változó neve határozza meg. Lehetne dönteni a név első karaktere alapján is, de az értelmező egy bonyolultabb eljárással egy sajátos ellenőrzőösszegszerű bájtot (pontosabban 0 és 62 közötti páros számot) képez a név karaktereiből, mint az a következő rutinon követhető:

```
D1A2  2A 47 03     LD   HL,(0347)  ;keresett név címe
D1A5  AF           XOR  A          ;A=0
D1A6  DD 46 03     LD   B,(IX+03)  ;név hossza
D1A9  07           RLCA            ;görgetés
D1AA  AE           XOR  (HL)       ;és maszkolás
D1AB  23           INC  HL         ;a következő karakterre
D1AC  10 FB        DJNZ D1A9       ;       ugrik
D1AE  E6 3E        AND  3E         ;0<A<64 (páros) kód
D1B0  4F           LD   C,A        ;azonosító-od C-be
D1B1  C9           RET             ;
```

Az 1. szegmens rutinja adja azt az értéket tehát, amely meghatározza, hogy a 64-bájtos tábla melyik címéről, melyik ágán induljon el az értelmező, ha egy adott változót keres.

Ez a bázistábla az alapja minden változókeresésnek, a láncrendszerbe a BASIC minden pillanatnyilag létező változója fel van véve. Akár ki is írathatjuk az összes változó nevét (változó dump). Bár a változók tárolási struktúráját az illető területeknél tekintjük át, annyit közös tulajdonságként már most elmondunk, hogy a változót tároló elemben a két pointerbájt után egy típusbájt, majd a változó neve következik: elöl a név hossza, aztán a karakterek. Ezt használja fel a következő BASIC program, amely sorra fölfejti az összes láncot:

```
100 FOR C=0 TO 62 STEP 2
110   PRINT C;":";TAB(7);
120   LET BAZIS=3531+C
130   LET CIM=PEEK(BAZIS)+256*PEEK(BAZIS+1)
140   IF CIM=0 THEN GOTO 220
150   LET HOSSZ=PEEK(CIM+3)
160   FOR N=1 TO HOSSZ
170     PRINT CHR$(PEEK(CIM+3+N));
180   NEXT
190   PRINT " - ";
200   LET BAZIS=CIM
210   GOTO 130
220   PRINT
230 NEXT
```

A futás eredménye:

```
START
 0 :  COS -
 2 :  C - TRUNCATE - IP - HEX$ -
 4 :  SEC - RGB - LTRIM$ -
 6 :  BAZIS - RTRIM$ - RED - COT -
 8 :  COSH -
 10 : STR$ - MAXLEN - EXSTRING$ - ACOS -
 12 : SGN - POS - BLUE -
```

```
14 : N - HOSSZ - SIZE - REM - RAD - EXTYPE - BLACK -
16 : SIN - RND - CEIL -
18 : CIM - VERNUM - ROUND - ABS -
20 : TIME$ - LCASE$ - INKEY$ - BIN -
22 : VAL - PEEK -
24 :
26 : LBOUND - ASIN -
28 : TAN - ORD - IN - FP - DEG -
30 :
32 : USR -
34 : SPEEK - LOG2 - ATN -
36 : YELLOW -
38 : VER$ - EPS - ANGLE -
40 : PI - MIN - LOG - CSC -
42 : WORD$ - SINH -
44 : INT -
46 : MOD - MAX - JOY -
48 : LOG10 - DATE$ -
50 : TANH - EXLINE - CYAN -
52 : MAGENTA - LEN - FREE - EXP -
54 : UCASE$ - GREEN -
56 : UBOUND -
58 : CHR$ -
60 : SQR -
62 : WHITE - INF -
ok
```

A listában megtaláljuk a program változóit (C, BAZIS, CIM, HOSSZ, N) a 2, 6, 14 és 18 relatív című lánc elején, majd az összes beépített változót és függvényt. Ez zavaró lehet akkor, ha minket csak a program változói érdekelnek. Ekkor felhasználhatjuk, hogy a program változói mind a BASIC kezdete után tárolódnak, másrészt a lánc elejére épülnek be. A keresést tehát már akkor leállíthatjuk, ha a kiolvasott pointer (CIM) a BASIC kezdete (esetünkben 12DBH = 4827D) alá mutat:

```
140 IF CIM < 4827 THEN GOTO 220
```

### 3.2.4 RAM-térkép

A BASIC rendszer 16 bájton (más kiépítésben az érték: 2 · (működő RAM-szegmensek száma)) tárolja az összetartozó programszám/kiutalt szegmens adatokat:

```
START
KEZDŐCIM ? 0E54
VEGCIM ? 0E63
SZEGMENS ? F8
 3668  0E54  F9 01 00 00 00 00 00 00
 3675  0E5C  00 00 00 00 00 00 00 00
ok
```

A lista azt mutatja, hogy megnyitottuk az 1. programot (EDIT 1) és az a rendszertől az F9H szegmenst kapta. Ha a program átlép a szegmenshatáron, a

rendszer új elemként, 128-cal megnövelt programszámmal ezt a lapot is felveszi a térképre (természetesen a túlnyúló résznek kiutalt szegmens számával).

Ez a térkép az alapja az EDIT, CHAIN műveleteknek, a programok belapozásának stb.

### 3.2.5 Kulcsszótábla

A BASIC kulcsszavak végrehajtó rutinjait, legalábbis belépési pontjaikat, verziónkban az 5. szegmens tartalmazza. A belépési pontokat, a kulcsszavak nevét és típusbájtját egy táblázat foglalja össze, amely a szegmens F5A7 címén kezdődik. A paramétertábla elemei a következő paramétereket adják meg (az első elem példáján): F5A7: F8 CC 61 D0 53 08 ALLOCATE

Cím + 0/1: a tulajdonképpeni végrehajtó alprogram címe (1. rutin; CCF8H)

Cím + 2/3: a végrehajtást előkészítő alprogram címe (2. rutin; D061H)

Cím + 4: típusbájt; bitjei az adott funkció alkalmazhatóságát, elvárásait jelzik:

BIT 0 = 1: parancs módban használható
BIT 1 = 1: program módban használható
BIT 2 = 1: blokkvég utasítás
BIT 3 = 1: blokk-kezdet utasítás
BIT 4 = 1: THEN után használható
BIT 5 = 1: hívása előtt CLEAR-t kell végrehajtani
BIT 6 = 1: a kulcsszó utáni programszöveg tokenizálandó
BIT 7: nem használt

A bitek 0 értéke természetesen a fentiek ellenkezőjét jelzi, tehát pl. a GOTO (52H) nem használható parancsmódban, a REM (03H) után nem kell tokenizálni stb.

Az ALLOCATE a fentiek szerint parancs- és programmódban egyaránt használható, THEN után is állhat és igényli a követő szöveg (argumentum) tokenizálását:

cím + 5: a kulcsszó hossza (8)

cím + 6-tól a karakterek következnek.

Az értelmező nem közvetlenül ezen tábla alapján dolgozik, hanem a rugalmasságát, bővíthetőségét jelentősen megnövelő lépésként felépít egy táblázatot a nulláslapon (esetünkben a 0E64 címtől 0F7D-ig), amely sorban tartalmazza a kulcsszavak pointereit: a paramétertábla adott elemének címét és szegmensét (ez esetünkben mindig 5). A táblázat kezdőcímét a 0232/3H rendszerváltozó adja meg. A tábla két nulla bájttal kezdődik, majd a pointerek száma (tehát a rendszer által kezelt BASIC kulcsszavak száma) következik (5DH = 93D). Ezt a bevezetőt követik a 3-bájtos pointerek (itt tehát 93*3 bájt). Az első elem pl.:

0E67: A7 F5 05

Ez tehát az 5. szegmens F5A7 címére, a paramétertábla előbb látott ALLOCATE elemére mutat. A további táblaelemek innen számított relatív sorszáma (1.:ASK,

2:AUTO stb.) egyben a kulcsszavak azonosítókódja, tokenje is lesz az értelmező számára.

A tokentábla kiolvasására, a paraméterek kilistázására egy programot is érdemes készíteni:

```
100 DEF H$(N)=CHR$(INT(N/16)+48-7*
    (INT(N/16)>9))&CHR$(MOD(N,16)+
    48-7*(MOD(N,16)>9))
110 PRINT " token "
120 PRINT " D   H    1.cim 2.cim tp
     kulcsszo"
130 PRINT
140 LET TABLA=PEEK(562)+256*
    PEEK(563)
150 LET SZAM=PEEK(TABLA+2)
160 FOR N=0 TO SZAM-1
170   LET CIM=TABLA+(N+1)*3
180   LET LO=PEEK(CIM)
190   LET HI=PEEK(CIM+1)
200   LET SG=PEEK(CIM+2)
210   LET ROMCIM=HI*256+LO
220   PRINT N;TAB(5);H$(N);":   ";
230   FOR I=0 TO 5
240     PRINT H$(SPEEK(SG,ROMCIM+
        I));" ";
250   NEXT I
260   LET HOSSZ=SPEEK(SG,ROMCIM+5)
270   FOR I=1 TO HOSSZ
280     PRINT CHR$(SPEEK(SG,ROMCIM+
        5+I));
290   NEXT I
300   PRINT
310 NEXT N
```

A futás eredményét a 6. függelékben adjuk meg. Ez a tokentábla az értelmező működésének alapja, mind tokenizáláskor, mind végrehajtáskor vagy listázáskor ezt a táblát használja. Mivel ez RAM-ban van, egy biztos és kényelmes módot ad a BASIC bővítésére, a funkciók megváltoztatására stb. Erre természetesen még viszatérünk.

### 3.2.6 Függvénytábla

A beépített függvények és változók kiértékelő rutinjait is a ROM (jelen esetben az 1. szegmens) tartalmazza. Itt is megtalálható egy összesített táblázat a függvények paramétereitvel (típus, hossz, kiértékelő rutin címe). A rendszer ebből is egy RAM-táblát készít az inicializálás során (közvetlenül a tokentábla után), de ez merőben más típusú, mint a kulcsszavak pointer táblázata.

A függvénytáblában változó formában tárolja az értelmező a belső változók és függvények paramétereit is. A változólánc leendő pointeréhez szükséges 2 bájt kihagyása után a ROM-ból a táblába másolja a következő elemeket:

— típusbájt: 0CH, ha numerikus értéket ad (pl. ABS vagy LEN),
0DH, ha füzér értékű (pl. CHR$);
— a név hossza;
— a név karakterei;
— a kiértékelő rutin címe.

Ezután 1-et ír az elem végére (a kiértékelő rutin szegmense). A 0F7E—12CA területen így kialakított függvénytábla első eleme pl. 0F7E: 00 00 0C 03 41 42 53 21 C3 01

Az ABS nevű függvény tehát numerikus és kiértékelő rutinját az 1. szegmens C321H címén találjuk.

A teljes függvénytáblát a 7. függelékben adjuk meg (hiszen ez is igen jól használható terület a BASIC-be való belenyúláshoz).

A függvényeket, belső változókat tehát a BASIC változók formai szabályai szerint tárolja a rendszer. A formai egységességnek kezelésbeli egységesség az eredménye: a tokenizálás, tárolás során a függvényeket a felhasználói változóktól gyakorlatilag semmi nem különbözteti meg. A végrehajtásnál is egységes a keresés, de a kiértékelés már nyilván nem: a belső függvényeknél az adatmező helyén a kiértékeléshez használandó rutin címét tárolja az értelmező.

Az egységes keresés érdekében természetesen be is kell vezetni a függvénytábla elemeit a változóláncokba. Ezt meg is teszi a rendszer az inicializálás során (mint annak eredményét a változók bázistáblájának (0DCBH) vizsgálatakor láttuk).

### 3.2.7 A BASIC programtároló

A programterület a függvénytábla — és a német verzióban utána tárolt RST 10H kiegészítés — fölött kezdődik. A terület kezdőcímét a 021E/FH rendszerváltozó tartalmazza (12DBH). Alaphelyzetben ettől a címtől kezdi tárolni az értelmező a beírt (vagy beolvasott) BASIC programot, de lehetőség van arra is, hogy n bájtot lefoglaljunk a terület elején egy ALLOCATE n utasítással. Ekkor a program (vagy a leendő program) n bájttal hátracsúszik. Az aktuális program tényleges kezdőcímét a 021A/BH rendszerváltozó adja meg számunkra (és az értelmező számára is).

Ez a rendszerváltozó akkor is érvényes címet tartalmaz, ha egy 0-tól különböző programot lapozunk be, amelyet mindig a 4000H címtől tárol a rendszer.

Példaként nézzük meg a memóriakiíratő DUMP program első sorainak memóriabeli elhelyezését. A programban nincs ALLOCATE, így a programszöveget a 12DBH címtől kereshetjük:

```
START
KEZDŐCIM ? 12DB
VEGCIM ? 1556
SZEGMENS ? F8
 4827  12DB  0D 64 00 00 60 39 80 04   ;.........
 4835  12E3  44 55 4D 50 00 09 6E 00   ;DUMP....
 4843  12EB  80 60 30 21 44 00 6C 78   ;....D...
 4851  12F3  00 00 60 0D 42 48 24 08   ;.....H$.
 4859  12FB  21 4E 09 13 44 43 48 52   ;.N...CHR
 4867  1303  24 08 23 49 4E 54 08 21   ;$..INT..
 4875  130B  4E 0F C2 10 00 09 0B C2   ;N.......
 4883  1313  30 00 0D C2 07 00 0A 08   ;.........
 4891  131B  23 49 4E 54 08 21 4E 0F   ;.INT..N.
 4899  1323  C2 10 00 09 14 C2 09 00   ;.........
 4907  132B  09 09 06 44 43 48 52 24   ;....CHR$
STOP at line 350
ok
LIST
  100 PROGRAM "DUMP"
  110 NUMERIC D
  120 DEF H$(N)=CHR$(INT(N/16)+48-7*
      (INT(N/16)>9))&CHR$(MOD(N,16)+
      48-7*(MOD(N,16)>9))

STOP .
ok
```

A kiíratás során (az összehasonlítás érdekében) kilistáztuk a tárolt program első sorait is.

Az adott példán kövessük végig a programtárolás struktúráját a kezdőcímtől indulva:

CIM + 0: a programsor hossza (itt 13 bájt) A következő sor címe: CIM + HOSSZ CIM' = 12DBH + 0DH = 12E8H

CIM + 1: a programsorszám alsó bájtja

CIM + 2: a programsorszám felső bájtja, Esetünkben: LO = 64H = 100D

HI = 0

SORSZAM = 100

CIM + 3: skatulyabájt, az adott sor skatulyázási mélységét adja meg. Az értéket az előtte lévő blokk-kezdet utasítások (FOR, DO, ...) növelik, a blokkvég utasítások (NEXT, LOOP,...) csökkentik.

Az értelmező ennek alapján keresi meg a blokk-kezdethez tartozó blokkvéget, végzi a strukturált listázást.

CIM + 4: itt kezdődik a kódolt (tokenizált) programszöveg.

A tokenizálási szabályokkal most nem foglalkozunk (ez külön pont tárgya lesz), de aki összehasonlítja a tárolt bájtokat a programlistával és a függelék tokentáblájával, az maga is rájön a főbb törvényszerűségekre (pl. 39H a PROGRAM utasítás tokenje stb.).

CIM + HOSSZ – 1: a programsort egy 0 bájt zárja. Így mehetnénk végig a teljes programszövegen, mindaddig, amíg a következő sor kezdőcímén (a hosszbájt helyén) 0-t nem találunk. Ez jelzi a BASIC program végét.

## 3.2.8 Változóterület

Az értelmező a BASIC program futása során a tárolandó változókat (numerikus és füzérváltozókat, DEF rutinneveket stb.) a program utáni területen helyezi el. Egy-egy újabb elemet mindig a 0234/5H rendszerváltozó által mutatott szabad terület kezdetére tölt, de a hosszabb programfejlesztés és futás során kialakuló kép korántsem rendezett, nehezen áttekinthető (a nyúló-rövidülő program, a törlések után újra és újra bevezetett változók miatt az érintett memóriaterület meglehetősen kusza képet mutat). A fontos az — számunkra és az értelmező számára egyaránt —, hogy a változókeresés alapjául szolgáló bázistáblából induló változóláncok mindig tökéletesen egyértelműek. A memórialista elemzése helyett használjuk mi is ezt a lehetőséget a változótípusok bemutatására, a terület felhasználásának illusztrálására.

A változók tárolási címének megkeresésére fogadjuk el segédeszközként azt a gépi rutint (V-CIM), amelynek működését majd a BASIC funkcióhívások ismertetésekor mutatjuk be. A rutin a hívó BASIC szövegben az utasítás után írt változó tárolási címét keresi meg és adja vissza a BASIC-nek (a típusbájtra mutat).

Ami közös a változóterületen tárolt különböző típusú elemekben, az a már említett fejrész:

— két bájton a változólánc következő elemének címe,
— a változó típusbájtja,
— a név hossza, majd karakterei.

A típusbájt értékei:

00H: numerikus BASIC változó
01H: füzérváltozó
02H: numerikus tömb
03H: füzértömb
04H: numerikus felhasználói függvény (pl. DEF DEC(N))
05H: füzér értékű felhasználói függvény (pl. DEF H$(N))
0CH: belső numerikus függvény vagy változó (pl. SIN() vagy PI)
0DH: belső füzér értékű függvény vagy változó (pl. HEX$())

A tényleges információt a fejrészt követő bájtok tárolják, de már típusonként lényegesen különböző módon. Ezt tekintsük át konkrét példákon (megismerve közben az értelmező számábrázolási logikáját is) a következő program segítségével:

```
100 ALLOCATE 100
110 NUMERIC CIM
120 CODE V_CIM=HEX$("D7,BE,00,E5,D7,
    20,00,E1,C9")
130 !
140 !
150 LET A=5
160 LET CIM=USR(V_CIM,0) A
170 CALL FEJ
180 PRINT "A=";A;TAB(16);
```

```
190 CALL ADAT(6)
200 !
210 !
240 DEF FEJ
250   FOR N=CIM-2 TO CIM
260     PRINT H$(PEEK(N));" ";
270   NEXT
280   LET HOSSZ=PEEK(N)
290   FOR I=1 TO HOSSZ
300     PRINT CHR$(PEEK(N+I));
310   NEXT
320   LET CIM=N+HOSSZ
330   PRINT
340 END DEF
350 DEF ADAT(H)
360   FOR N=1 TO H
370     PRINT H$(PEEK(CIM+N));" ";
380   NEXT
390   PRINT
400 END DEF
410 DEF H$(N)=CHR$(INT(N/16)+48-7*
    (INT(N/16)>9))&CHR$(MOD(N,16)+
    48-7*(MOD(N,16)>9))
```

A program első szakasza betölti az említett V-CIM gépi rutint, a felhasználói rutinok pedig kiírják a standard fejet (pointer—típus—név), ill. az adatmező megadható számú bájtját (hexadecimálisan). Magát a futást a 150—200 szakasz néhány sora irányítja.

*Numerikus változó (00H típus)*
Az adott esetben a futtatás eredménye:

```
EB OF 00 A
A= 5            05 00 00 00 00 7F
ok
```

A fej: — pointer: 0FEB (a lánc következő eleme a COS belső függvény)
— típus: 00 (numerikus változó)
— név: A

Az adatmezőn a változó értékét bináris egészként látjuk viszont. A számábrázolás módját leginkább kísérletezve, különböző értékeket adva lehet követni. Figyeljünk meg pl. egy előjelváltást az adatmezőn. Ehhez a program aktív részét is át kell kissé alakítani:

```
150 LET A=3
160 LET CIM=USR(V_CIM.0) A
170 CALL FEJ
180 PRINT "A=";A;TAB(8);
190 CALL ADAT(6)
200 LET A=A-1
210 GOTO 180
```

```
START
EB 0F 00 A
A= 3    03 00 00 00 00 7F
A= 2    02 00 00 00 00 7F
A= 1    01 00 00 00 00 7F
A= 0    00 00 00 00 00 7F
A=-1    FF FF 00 00 00 7F
A=-2 .  FE FF 00 00 00 7F
A=-3    FD FF 00 00 00 7F

STOP at line 380
ok
```

Jól látható, hogy az adott esetekben az értelmező kétbájtos előjeles egészként ábrázolta az értékeket. Az adatsor végén álló 7FH kód ez a módot jelzi.

Teljesen más a tárolási mód törtszámoknál, nagy egészeknél, olyan esetekben, amikor eleve nem alkalmazható a kétbájtos előjeles ábrázolás:

```
150 LET A=1.234567891
160 LET CIM=USR(V_CIM.0) A
170 CALL FEJ
180 PRINT "A=";A;TAB(16):
190 CALL ADAT(6)
```

```
START
EB 0F 00 A
A= 1.234567891 91 78 56 34 12 40
ok
```

Feltűnő, hogy ha az első öt adatbájtot egymás elé illesztjük, megkapjuk a tárolt szám jegyeit. Ebben az esetban a tárolás tehát BCD alakban történik, méghozzá 5 bájton (10 decimális jegyen). A nagyságrend ábrázolási módját jól követhetjük a szám ismételt 10-es szorzásával:

```
150 LET A=1.234567891
160 LET CIM=USR(V_CIM.0) A
170 CALL FEJ
180 PRINT "A=";A;TAB(16):
190 CALL ADAT(6)
200 A=A*10
210 GOTO 180
```

```
START
EB 0F 00 A
A= 1.234567891 91 78 56 34 12 40
A= 12.34567891 91 78 56 34 12 41
A= 123.4567891 91 78 56 34 12 42
A= 1234.567891 91 78 56 34 12 43
A= 12345.67891 91 78 56 34 12 44
A= 123456.7891 91 78 56 34 12 45

STOP at line 380
ok
```

Mint látható, a tárolt alak egységesen $1.23.... \cdot 10^n$

ahol a szám nagyságrendjét — 10 kitevőjét — az adatmező 6. bájtja adja meg: kitevő = 6. bájt – 40H

Igaz ez 1-nél kisebb számokra is (itt természetesen a kitevő negatív):

```
150 LET A=1.234567891
160 LET CIM=USR(V_CIM.0) A
170 CALL FEJ
180 PRINT "A=";A:TAB(21):
190 CALL ADAT(6)
200 LET A=A/10
210 GOTO 180


START
EB 0F 00 A
A= 1.234567891        91 78 56 34 12 40
A= .1234567891        91 78 56 34 12 3F
A= 1.234567891E-02    91 78 56 34 12 3E
A= 1.234567891E-03    91 78 56 34 12 3D
A= 1.234567891E-04    91 78 56 34 12 3C

STOP at line 380
ok
```

Most már csak a negatív lebegőpontos számokra nézzünk példát:

```
150 LET A=-1.234567891
160 LET CIM=USR(V_CIM.0) A
170 CALL FEJ
180 PRINT "A=";A:TAB(21):
190 CALL ADAT(6)
200 LET A=A/10
210 GOTO 180


START
EB 0F 00 A
A=-1.234567891        91 78 56 34 12 C0
A=-.1234567891        91 78 56 34 12 BF
A=-1.234567891E-02    91 78 56 34 12 BE
A=-1.234567891E-03    91 78 56 34 12 BD
A=-1.234567891E-04    91 78 56 34 12 BC

STOP at line 380
ok
```

Mint látható, a negatív előjelet igen egyszerűen a kitevőbájt beírt 7. bitje jelzi.

Összefoglalva a számábrázolás módját:

— A kis egészeket a gép kétbájtos előjeles számként tárolja. Ezt a módot a kitevőbájt 7FH értéke jelzi. Ilyenkor az adatmező 1. (LO), 2. (HI) és 6. (flag) bájtja tartalmaz információt.

— Az előző módon nem ábrázolható számokat lebegőpontos alakban tárolja az értelmező: a szám 10 digitális jegyét 5 bájton, BCD alakban, míg a 6. bájt a kitevő és előjel információját hordozza.

— Ha a változó még nem kapott értéket (amire még nem láttunk példát), a tárolt elemet 7FH kitevőbájt és előtte egy FFH érték érvényteleníti.

Az egész típusú tárolás ábrázolási tartománya elvileg – 32 768 és 32 767 között van, azonban az értelmező, legalábbis a változó érték betárolásakor, csak a 10 000-nél kisebb abszolút értékű számokat ábrázolja ilyen módon (ezért 9999 a maximális BASIC sorszám). Az értékek megváltozásánál, a műveleteknél esetleg szükséges konverzió a két mód között automatikusan megtörténik a BASIC aritmetikai rutinjaiban.

*Füzérváltozó* (01H típus)

A karakterfüzérek tárolási módja lényegesen egyszerűbb:

```
150 LET A$="STRING"
160 LET CIM=USR(V_CIM.0) A$
170 CALL FEJ
180 PRINT "A$=";A$;TAB(14);
190 CALL ADAT(8)
200 !


START
8C 12 01 A$
A$=STRING     84 06 53 54 52 49 4E 47
```

Az adatmező első bájtja a lefoglalt bájtok számát adja (84H = 132D), a 2. bájt a ténylegesen tárolt karakterek száma (6). Ezt a karakterek ASCII kódja követi. A lefoglalt terület hátralévő része érdektelen.

Az inicializálatlan változót a 2. bájt FFH értéke jelzi.

*Numerikus tömb* (02H típus)

Itt is egy példán követhetjük a tárolás módját:

```
140 DIM A(3 TO 5,1 TO 2)
150 LET A(3,1)=1.23:LET A(4,1)=3.21:
    LET A(5,1)=PI
160 LET CIM=USR(V_CIM,0) A
170 CALL FEJ
180 !
190 CALL ADAT(79)
200 !


START
EB 0F 02 A
02 08 00 00 00 00 7F 03 00 00 00 00 7F
04 00 00 00 00 7F 01 00 00 00 00 7F 02
00 00 00 00 7F 00 00 00 30 12 40 00 00
00 00 FF 7F 00 00 00 10 32 40 00 00 00
00 FF 7F 00 00 00 00 FF 7F 00 00 00 00
FF 7F 54 26 59 41 31 40 00 00 00 00 FF
7F
```

Az adatmezőn az indexek számait (1 vagy 2) mégadó bájt (esetünkben 2) után 5 db 6-bájtos (szokásos alakú) numerikus érték következik:

— a tömb elemeinek száma (4 x 2 = 8),
— az 1. index alsó határa (3),
— az 1. indextartomány mérete (4),
— a 2. index alsó határa (1),
— a 2. indextartomány mérete (2).

Ez a bevezető rész teljes akkor is, ha nincs 2. index, csak akkor érdektelen a mező ezen része.

Ezután következnek sorra a tömb elemei (ugyancsak 6—6 bájton). Esetünkben a sorrend:

A(3,1) A(3,2)
A(4,1) A(4,2)
A(5,1) A(5,2)
A(6,1) A(6,2)

Most látunk példát az inicializálatlan (értéket nem kapott) elemek megjelölésére is (az 5. és 6. bájt: FF 7F)

*Füzértömb (03H típus)*

A fentivel megegyező bevezető rész után ugyanilyen sorrendben következnek a tömb elemei az egyszerű füzér adatmezőnek megfelelően.

*Felhasználói függvény (04H és 05H típus)*

```
150 DEF A=PI/180
160 LET CIM=USR(V_CIM,0) A
170 CALL FEJ
180 !
190 CALL ADAT(8)
200 !

START
EB 0F 04 A
00 00 00 00 FF 7F B4 13
```

Mint az egyszerű példa illusztrálja, az adatmező első része egy (a függvénytípusnak megfelelő) érvénytelenített változó mező (a névnek megfelelő lokális változó ideiglenes tárolására), amit egy pointer követ: a függvényt definiáló BASIC sor memóriabeli címe.

*Belső függvény vagy változó (0CH vagy 0DH típus)*

Ezek a változók a függvénytábla elemei. Mint ott láttuk, az adatmező 3 bájt: a kiértékelő rutin címe (LO,HI) és szegmense.

## 3.2.9 BASIC munkaverem

Az értelmező a belső változók, visszatérési címek stb. tárolására a BASIC terület felső részét használja, méghozzá veremszerűen, analóg módon a Z80-as veremkezelésével. Ez az analógia a GOSUB—RETURN utasításoknál a legerősebb: az értelmező a GOSUB-ot követő sor címét tölti a verembe, majd a RETURN hatására onnan olvassa vissza az utasításszámlálóba éppen úgy, mint a mikroprocesszor a CALL—RET utasításoknál.

Hasonló a memóriahasználat módja is: a BASIC is felülről lefelé növeli a vermet, az új elemet mindig az előző alá tölti és az aktuális elemre (egyben a szabad terület fölé) egy pointer, mondhatni BASIC STACK—POINTER mutat: a 0228/9H rendszerváltozó.

Az értelmező azonban sokféle elemet tárol ideiglenesen a veremben. Ezek közös tulajdonsága mindössze az, hogy mindegyik egy típusbájttal kezdődik. A futás során használt kódok:

*2: numerikus adat (9 bájt)*

A BASIC vermet az aritmetikai rutinok is használják. Ide töltik a konstansokat, többoperandusos műveletek adatait stb. Itt marad a kiértékelés végeredménye is, innen írja ki pl. a PRINT rutin. A veremben tárolt érték magját a szokásos 6 bájton tárolt érték adja (10 számjegy 5 bájton + a kitevő), de a számjegyek előtt és után még 1—1 bájtot tárol a rendszer a műveletek során várható túlcsordulások és alulcsordulások miatt.

Például PI értéke a BASIC veremben a következő módon tárolódik (hexadecimálisan):

02 00 54 26 59 41 31 00 40

*3: füzér (2 bájt + a karakterek)*

A karakterfüzérek a BASIC veremben is a szokásos módon tárolódnak (a típusbájttal kiegészülve).

Például a TEXT szöveg:

03 04 54 45 58 54

*4: a FOR—NEXT ciklus paraméterei (25 bájt)*

Az értelmező a program futása során a következő veremadat alapján dolgozik:

— típusbájt (4)
— a FOR sor címe a memóriában (2 bájt),
— a NEXT sor címe (2 bájt),
— a ciklusváltozó tárolási címe (2 bájt),
— a STEP érték numerikus adatként (9 bájt),
— a TO érték numerikus adatként (9 bájt).

Nézzünk meg egy egyszerű példát illusztrációként:

```
100 FOR I=0 TO 24
110   LET VEREM=PEEK(552)+256*
      PEEK(553)
120   PRINT H$(PEEK(VEREM+I));" ";
130 NEXT
140 PRINT
150 STOP
160 !
170 DEF H$(N)=CHR$(INT(N/16)+48-7*
    (INT(N/16)>9))&CHR$(MOD(N,16)+
    48-7*(MOD(N,16)>9))


START
04 DB 12 36 13 52 14 02 00 01 00 00 00
00 00 7F 02 00 18 00 00 00 00 00 7F

STOP at line 150
```

Esetünkben tehát a paraméterek:

— típusbájt (4),
— a FOR sor címe 12DBH (az első sor a programban),
— a NEXT sor címe 1336H,
— a ciklusváltozó (I) tárolási címe: 1452,
— a lépésköz szokásos értéke:
  01 00 00 00 00 7F = 1,
— a végérték:
  18 00 00 00 00 7F = 24 Dec.

A ciklus megkezdése után tehát csak a ciklusváltozó marad igazi BASIC változó (értékét bármikor átírhatjuk a ciklus futása közben is), míg a lépésköz és végérték adatai belépéskori aktuális értékükön tárolódnak a ciklus befejezéséig. A tárolás módjának ismeretében viszont közvetlen beírással megváltoztathatjuk ezeket az értékeket is.

*5: a DO-LOOP ciklus paraméterei (5 bájt)*
A verembe töltött elem:

— típusbájt (5),
— a blokk-kezdet (DO sor) memóriabeli címe (2 bájt),
— a blokk-vég (LOOP sor) címe (2 bájt).

Az értelmező itt egyéb paramétereket nem tárol, a ciklusfeltételeket a DO, ill. a LOOP kiértékelésénél veszi figyelembe.

*6: GOSUB paraméter (3 bájt)*

Itt az értelmező a típusbájt (6) után a GOSUB-ot követő BASIC sor címét tölti a munkaverembe. Ez okozza azt az ellentmondást, hogy bár a rendszer formálisan elfogad több utasítást is a GOSUB sorban (pl. GOSUB 100:PRINT X), a RETURN után ezt mégsem veszi figyelembe: a következő sor elején folytatja a végrehajtást.

*7: a felhasználói függvények (DEF rutinok) paraméterei (6 bájt)*

A DEF rutinok és sorok a BASIC futás és kiértékelés szerves részei, lényegében csak a változóterület kezelésében különböznek némileg a program egyéb rutinjaitól és értékadásaitól.

Ennek megfelelően hívásukkor az értelmező a munkaverembe tölti a változóterület pillanatnyi adatait és ezek alapján

— belső változóival a hívás előtti változóterület fölött dolgozik,

— a kiértékelés végén (END DEF) törli a határ fölötti, lokálisnak tekintett változókat és visszaállítja a változóterület eredeti adatait.

A felhasználói függvény alapvetően kétféle módon hívható meg:

1. A PRINT RUTIN vagy A = RUTIN típusú hívás egy értéket vár vissza a veremben

2. A CALL RUTIN hívás nem vár vissza adott értéket.

Ennek megfelelően a DEF paraméterblokk:

— típusbájt (7),

— a lokális változóterület határa (0236/7H) a belépéskor (2 bájt),

— jelzőbájt: 0. bitje 0, ha a kiértékelés utáni értéket a veremben kell hagyni (1 bájt),

— a szabad terület kezdete belépéskor (0234/5H).

*8: WHEN paraméterek (12 bájt)*

Mivel a kezelni kívánt hiba a blokkutasításoktól függetlenül történhet meg, a hiba esetén használandó HANDLER címét és a blokkparamétereket feltétlenül veremben kell tárolni. A WHEN blokkok egymásba skatulyázhatók, egy adott hibamélységhez adott kezelő tartozik, tehát a skatulyázási mélységet is fel kell venni a WHEN paraméter blokkba:

— típusbájt (8),

— skatulyabájt,

— a hiba esetén használandó HANDLER sor címe (2 bájt),

— belső munkaterület (6 bájt),

— a blokkvég (END WHEN) címe (2 bájt).

Az értelmező a BASIC verem különböző típusú elemeit típusuk alapján ismeri fel, így határozza meg a hosszukat, így keres közöttük és így törli a feleslegessé

vált elemeket. Ugyanakkor a gépi kódú programozásban — főként, ha a BASIC aritmetikát is fel kívánjuk használni — nélkülözhetetlen a verem saját célú igénybevétele. Különösen kell tehát ügyelnünk, hogy ne hagyjunk szemetet a veremben munkánk után (ezt BASIC funkcióhívások is segítik), vezérlésátadással járó rutinban pedig tekintettel kell lennünk az esetleg átlépett blokkhatárokra is. Ezért is adtuk meg a veremblokkok paramétereit viszonylag részletesen.

## 3.3 A BASIC Restart rutinok

Mint az inicializálási folyamatnál láttuk, a BASIC is felépíti a nulláslapon (a ROM-ból átmásolva) azokat a RST hívással aktivizálható alaprutinjait, amelyek nélkülözhetetlenek a működéséhez. Ezek funkciói:

RST 08H: a BASIC lapozórutinja. A 3. LAP-on futó értelmező programból ezen keresztül lehet egy másik szegmens ugyancsak 3. LAP-on futó rutinját meghívni.

RST 10H: a BASIC funkcióhívás. A programszövegben ezt követő bájt a kért funkció száma, mint az EXOS n esetén, de itt az egymás utáni bájtokat mindaddig funkciókódnak tekinti a rendszer, amíg a 00H kódhoz nem ér a sor.

RST 18H: az EXOS hibák kezelője. 9000D + A hibakóddal a RST 20H-ba fut.

RST 20H: a BASIC hibakezelő rutinja.

BASIC környezetben futó, esetleg azt bővítő gépi kódú programjainknál mindegyik hívás hasznos lehet (sőt a RST 10H a BASIC leghasznosabb rutinja lesz számunkra az általa elérhető rutinkönyvtár révén). Közülük a RST 08H működését feltételnül érdemes lépésenként is végigkövetni, már csak azért is, mert ezt egy közbenső belépéssel talán még praktikusabban használhatjuk, mint rendeltetésszerűen.

A rutin egy nagyon szellemes ötlettel oldja meg az azonos lapon, de különböző szegmensen futó rutinok át-, majd visszalapozását. A fogás lényege, hogy a célrutin címét egy mindig belapozott (nulláslap!) RAM-ba írt CALL nn utasítás argumentumába tölti és ezen keresztül hívja meg:

```
0008  32 68 00     LD    (0068),A   ;A munkaterületre (xx)
000B  22 F6 00     LD    (00F6),HL  ;HL munkater.-re (xxxx)
000E  18 4C        JR    005C       ;folytatásra ugrik

005C  E5           PUSH  HL         ;regiszterek
005D  F5           PUSH  AF         ;  verembe
005E  3A FA 00     LD    A,(00FA)   ;előző 3. LAP-paraméter
0061  67           LD    H,A        ;mentése
0062  DB B3        IN    A,(B3)     ;pillanatnyi 3.LAP (yy)
0064  32 FA 00     LD    (00FA),A   ;(yy) munkaterületre
0067  3E xx        LD    A,xx       ;munkaterületre tárolt
                                     érték visszaolvasása
0069  D3 B3        OUT   (B3),A     ;célrutin szegmense
006B  F1           POP   AF         ;
006C  E3           EX    (SP),HL    ;HL: belépési érték
006D  CD F5 00     CALL  00F5       ;meghívja a célrutint
0070  E3           EX    (SP),HL    ;visszaadott HL verem-
                                     be, 3. LAP-par. vissza
```

```
0071  CD F8 00     CALL 00F8        ;belépési 3. LAP vissza
0074  F5           PUSH AF          ;
0075  7C           LD   A,H         ;előző 3. LAP paraméter
0076  32 FA 00     LD   (00FA),A    ;munkaterületre
0079  F1           POP  AF          ;visszaadott A
007A  E1           POP  HL          ;visszaadott HL .
007B  C9           RET              ;


00F5  CD xx xx     CALL xxxx        ;meghívja a célrutint
00F8  F5           PUSH AF          ;mentés
00F9  3E 05        LD   A,yy        ;előző érték
00FB  D3 B3        OUT  (B3),A      ;visszalapozása
00FD  F1           POP  AF          ;visszaadott A
00FE  C9           RET              ;
```

A rutin lényegesen hatékonyabb, mint az EXOS által ugyancsak kiépített (B217H-) lapozórutin (amit most nem használhatunk, mert az a 0. szegmenst lapozza vissza kilépéskor).

Mint a listából kiolvasható, a RST 08H hívás bemenő paraméterei:

HL: a célrutin címe,

A: a célrutint tartalmazó szegmens.

A rutin minden regisztert megőriz és átad a célrutinnak (majd visszaad a hívónak), a futás végére a hívó szegmensét lapozza vissza.

Hatékonysága akkor használható ki igazán, ha mi töltjük a munkaterületekre rutinunkban a címet és a szegmenst (l. 0008H és 000BH sorok), majd A és HL tetszés szerinti értékével a 005CH címen hívjuk a lapozórutint. Sok praktikusan használható ROM-rutin igényli ui. az A és HL regisztereket is bemenő paraméterként.

A BASIC funkcióhívást elindító RST 10H rutin lépéseit funkcionálisan tekintsük át:

— elmenti a hívási regisztereket és memóriakonfigurációt,
— beolvassa a funkciószámot a hívó program következő bájtjáról,
— meghatározza a funkciószámhoz tartozó végrehajtó rutin címét és szegmensét,
— belapozza és meghívja a végrehajtó rutint,
— visszatér a következő funkciószám beolvasásához.

Ez a folyamat korlátlanul ismétlődhet mindaddig, amíg a beolvasott adat 00H nem lesz. Az ehhez tartozó végrehajtó rutin ui. visszaállítja a hívási regiszter- és memóriaállapotot, majd visszatér a hívóhoz. Ez a sorozatszerűen ismételhető funkcióhívás rendkívül hatékony aritmetikát tesz lehetővé, amiben könnyen építhetjük gépi kódú programjainkba a legbonyolultabb lebegőpontos műveletsorokat, sorfejtéseket stb. Több rutin segíti ugyanakkor az egyszerű beolvasást és kiíratást, szövegelemzést stb. A fontosabb, általánosan használható funkciókat a következő pontban írjuk le.

Most térjünk vissza a szétválogatás műveletéhez. A programszövegből beolvasott funkciószám értéke két tartományba eshet. A 00H—41H funkciókat a BASIC-UK szegmens (esetünkben az 5.) kezeli, végrehajtó rutinjaik belépési címeit a szegmens

C0E8H—C16BH táblázata tartalmazza. A 80H—C5H funkciók rutinjai az 1. szegmensben vannak, táblázatuk a szegmens elején található (C000H—C08BH). A két táblázatot DUMP-szerűen megadjuk, ebből az olvasó is kikeresheti egy-egy rutin címét:

```
5. szegmens  (BASIC 00H-41H)

C0E8  (00-03)  DE 00 01 F3 EF E8 5C E3
C0F0  (04-07)  CD EE 91 E9 CB E9 C7 E9
C0F8  (08-0B)  91 E5 8D E5 26 EE F9 E7
C100  (0C-0F)  01 E5 F8 E4 A5 E5 C8 E6
C108  (10-13)  E7 E9 B7 EA D1 E9 65 E8
C110  (14-17)  BD E8 8B EA E8 ED 79 E9
C118  (18-1B)  63 EA 6F EA 7B EA 0A E9
C120  (1C-1F)  25 E9 9C E7 B7 E7 3F E9
C128  (20-23)  59 F2 E4 E2 53 F2 E2 DE
C130  (24-27)  96 E0 9E FB BE EE 94 F0
C138  (28-2B)  97 F0 10 F0 C2 C6 1E E4
C140  (2C-2F)  07 E4 D4 D4 FD C3 58 C8
C148  (30-33)  44 C5 4D C5 D5 EA 7A EF
C150  (34-37)  19 EF D7 EF B5 FB 4B C7
C158  (38-3B)  66 FB DA EE DE C4 83 EC
C160  (3C-3F)  69 CE 68 D0 84 FB 35 E3
C168  (40-41)  93 E0 D9 DE


1. szegmens  (BASIC 80H-C5H)

C000  (80-83)  B2 D1 4D D0 7C D0 94 D0

C010  (88-8B)  E7 D0 AB D0 01 D1 89 D1
C018  (8C-8F)  67 D1 17 D1 53 D1 D9 D1
C020  (90-93)  61 C2 AC C2 8D C2 9F C2
C028  (94-97)  A4 C2 A9 C2 C6 C2 CC C2
C030  (98-9B)  D1 C2 E4 C2 F0 C2 ED C2
C038  (9C-9F)  F7 C2 FF C2 FC C2 06 C3
C040  (A0-A3)  E7 C2 8D C6 99 C3 68 C3
C048  (A4-A7)  86 CB 18 C5 9F CC F0 C5
C050  (A8-AB)  80 CA 86 C5 A7 C8 8D C9
C058  (AC-AF)  52 CC 8E CA 78 C7 17 CB
C060  (B0-B3)  B0 C7 20 C8 F9 CB D9 C1
C068  (B4-B7)  B9 C1 EF C1 F9 C1 F3 D1
C070  (B8-BB)  AC D2 30 D4 A8 D4 46 D5
C078  (BC-BF)  CF D5 48 DA 9D DA 4B DF
C080  (C0-C3)  46 D7 73 D7 B5 D7 C5 D7
C088  (C4-C5)  2E CD 12 D8
```

A szétválogatás menetébe a német verziójú gépeknél egy érdekes elem kerül. A német kiterjesztés bővítője az alaphelyzetben felépülő rutin néhány bájtját kiemeli és helyükre tölti a CALL nn utasítás kódjait (konkrétan a függvénytábla után beírt néhány bájtos kiegészítés címét adja meg). A kitérőben természetesen ellátja a kiemelt bájtok funkcióját és egy kód (a hibaüzeneteket kiíratő BFH funkció) esetén saját rutincímét és szegmensszámát adja meg, egyébként az eredeti módon folytatja a kiértékelést. Erre azért hívtuk fel a figyelmet, mert némi gyakorlat után az olvasóban is felmerülhet egy-egy funkció elfogásának szükségessége.

A *RST 18H* és *RST 20H* rutinok a felhasználó számára a hibakezelést teszik igen egyszerűvé. A RST 18H hívás bemenő paramétere az A (EXOS hibakód) és

```
EXOS n
RST 18H
```

alakban használható az esetleges EXOS hibák azonnali kiszűrésére. Ha nincs hiba, akkor visszatér és a következő utasításnál folytatja a végrehajtást.

A RST 20H bemenő paramétere a HL-ben megadott BASIC hibakód (ezek felsorolását a gépkönyv is megadja).

### 3.3.1 A BASIC funkcióhívások

Mielőtt elkezdenénk a fontosabb BASIC funkcióhívások leírását, térjünk ki két terminológiai jellegű megjegyzésre.

1. A RST 10H-n keresztüli funkcióhívások szintaxisa egy assembly programban:

```
RST 10H ;     funkcióhívás bekapcsolás
DEFB n ;      funkciószám
DEFB 00H ;    kikapcsolás
```

Mi saját használatra a következőkben a funkcióra jobban utaló

```
BASIC n
BASIC 00H
```

jelölést alkalmazzuk, de természetesen egy pl. ASMON-ban megírt programban az assembly (pontosabban a fordító assembler) szabályait kell betartanunk.

2. A funkcióhívások egyik fő területe az aritmetika. Mint a 3.2.9 pontban láttuk, a BASIC verem szolgál a kiértékelés során az adatok feldolgozás közbeni ideiglenes tárolására. A numerikus adatokat egy 9-bájtos, a füzéreket egy (2 + füzérhossz) bájtos blokk tárolja. A továbbiakban ezeket a blokkokat a rövidebb leírás érdekében egységesen munkaregisztereknek (rövidítve MREG) nevezzük. Mivel több argumentumos kiértékeléseknél, számításoknál gyakori a több veremblokk használata, szükség esetén az elsőként megnyitott blokkra MREG1-ként hivatkozunk, a továbbiakra pedig MREG2, MREG3... jelöléssel. MREG1 tehát fizikailag a legfölső blokk a létrehozott elemek közül (a verem felülről lefelé bővül).

Tekintsük át ezek után a gépi kódú programjainkban is jól használható, fontosabb hívások funkcióit. Az áttekinthetőség érdekében alkalmazási példákat majd a fejezet végén adunk.

BASIC 00H (00DE)
A funkcióhívás-sorozat kikapcsolása, a belépés előtti memóriakonfiguráció visszalapozása és visszatérés a hívóhoz. Minden funkcióhívás (sorozat) végén a 00H kódnak kell állnia

BASIC 01H (5/F301)
Gyakorlatilag megegyezik a RST 20H hívással: a HL kódú hibát kezeli

BASIC 02H (5/E8EF)
HL értékét egész típusú numerikus adatként a munkaverembe tölti:
HL → MREG

BASIC 03H (5/E35C)
A MREG tartalmát kiíratható füzérré alakítja a 059EH pufferben:

MREG → ASCII
A blokk a veremben marad.

BASIC 04H (5/EECD)
HL (előjeles kétbájtos egész) előjelét megfordítja
HL = –HL

BASIC 05H (5/E991)
A MREG-ben lévő szám előjelének vizsgálata. Eredménye a státusz:
Z, ha pozitív vagy 0
NZ, ha negatív

BASIC 06H (5/E9CB)
Megfordítja a MREG-ben lévő szám előjelét
MREG = – MREG

BASIC 07H (5/E9C7)
MREG = ABS (MREG)

BASIC 08H (5/E591)
A MREG-ben lévő szám 10-es komplemensét veszi (a BCD kivonás előkészítése). Csak lebegőpontos MREG-tartalom esetén alkalmazható

BASIC 09H (5/E58D)
Ha a MREG-ben lévő szám negatív, annak 10-es komplemensét veszi (BASIC 08H)

BASIC 0AH (5/EE26)

A MREG-ben lévő szám típusának vizsgálata. Eredmény:
Z, ha egész
NZ, ha lebegőpontos
A-ban a kitevőbájt

BASIC 0BH (5/E7F9)

A MREG tartalmát (egész típusúra alakítva) HL-be tölti.
MREG → HL
Az érékhatár túllépése esetén hibára ugrik. A rutin törli a kiolvasott blokkot a veremből. Érdekessége, hogy E7FAH belépésnél ki is értékeli a programszöveg következő elemét (pl. CAUSE EXCEPTION operandus)

BASIC 0CH (5/E501)

Összeadás:
MREG1 = MREG1 + MREG2
A művelet után a MREG2 törlődik a veremből, az aktuális elem az eredmény (MREG1). Ez a következő műveletekre is vonatkozik

BASIC 0DH (5/E4F8)

Kivonás:
MREG1 = MREG1 – MREG2

BASIC 0EH (5/E5A5)

Szorzás:
MREG1 = MREG1 · MREG2

BASIC 0FH (5/E6C8)

Osztás:
MREG1 = MREG1/MREG2

BASIC 10H (5/E9E7)

Összehasonlítja MREG1 és MREG2 tartalmát. A státusz a (MREG1 – MREG2) művelet eredményének megfelelően áll be:
Z, ha MREG1 = MREG2
P, ha MREG1 > MREG2
M, ha MREG1 < MREG2
A művelet után mindkét elem törlődik a tárból

BASIC 11H (5/EAB7)

A MREG tartalmát elosztja $10^A$-val (A az akkumulátor tartalma)

BASIC 12H (5/E9D1)

A MREG-ben lévő szám nagyságát vizsgálja
NZ, ha MREG<>0

Z, ha MREG = 0
Ez utóbbi esetben egész típusra állítja az elemet

BASIC 13H (5/E865)
Egészről lebegőpontos típusúra konvertálja a MREG tartalmát

BASIC 14H (5/E8BD)
Egy BASIC változó tartalmát viszi át (a HL címről) a MREG-be. Hívása előtt nyilván meg kell keresni a változó tárolási címét (pl. BASIC 8BH)
MREG = X

BASIC 15H (5/EA8B)
Egy blokkal előbbre is lemásolja a verem aktuális (numerikus vagy string) elemét:
MREG2 = MREG1

BASIC 16H (5/EDE8)
Törli a verem aktuális elemét (tetszőleges típus esetén). A BASIC verem Z80-as verem analógiában ez a POP utasításnak felel meg. Igen fontos hívás a verem rendbetételéhez. Valójában nem töröl semmit a veremben, csak a BASIC veremmutatót állítja a fölötte lévő elemre

BASIC 17H (5/E979)
Egész típusú 0-t tölt a MREG-be
MREG = 0
A típusbájt (02) utáni érték:
00 00 00 00 00 00 00 7F

BASIC 18H (5/EA63)
Lebegőpontos 0-t tölt a MREG-be:
MREG = 0.0
(00 00 00 00 00 00 00 3F)

BASIC 19H (5/EA6F)
Egész típusú 1-et tölt a MREG-be
MREG = 1
(00 01 00 00 00 00 00 7F)

BASIC 1AH (5/EA7B)
Lebegőpontos 1-et tölt a MREG-be
MREG = 1.0
(00 00 00 00 00 10 00 40)

BASIC 1BH (5/E90A)
A numerikus MREG tartalmát a BASIC változóba tölti (a HL címre):
X = MREG

BASIC 1CH (E/E925)
A füzér — MREG tartalmát a BASIC változóba tölti (a HL címre):
X$ = MREG

BASIC 1DH (5/E79C)
Az A értékét írja a MREG B-vel kijelölt számjegyébe. A = 0,...,9 lehet
B = 4 a legnagyobb helyi értékű digitet jelöli ki
B = 13 a legkisebb helyi érték (10 jegy)

BASIC 1EH (5/E7B7)
Az 1DH hívás fordítottja: A-ba olvassa a MREG-ben lévő szám B-vel kijelölt számjegyét

BASIC 1FH (5/E93F)
A MREG-ben lévő számot normálalakra konvertálja: az első értékes karaktert a legnagyobb helyi értékű digit helyére lépteti

BASIC 20H (5/F259)
A programszöveg következő elemének vizsgálata. A BASIC értelmező leggyakoribb hívása a sorok elemzése, végrehajtása közben.

A tokenizálással előkészített programszöveg következő (a 0214/5H rendszerváltozóval megcímzett) elemének (változó, konstans, kulcsszótoken, sorszám vagy jel)

— típuskódját a 0202H,
— hosszát a 0203H,
— jelek esetén kódját a 0204H

rendszerváltozóba tölti.

Ezenkívül

— a 0347/8H mutatót a bevezető kódot követő bájtra állítja (változó, ill. függvény esetén ez a név első karaktere), ez lesz majd a változókeresés alapja;

— számkonstans esetén az értéket beírja a 0596H pufferbe és — ha BASIC sorszámként értelmezhető 0 és 9999 közötti egész — a 0218/9H rendszerváltozóba is;

— az átlépett elem utáni címet állítja be következő aktuális címként a 0214/5H változóban;

— ha az elem típusbájtja 20H alatti (jel), az értéket A-ba is beolvassa (a sor végét A = 1 és Z státusz jelzi)

BASIC 21H (5/E2E4)
Zárójeles aritmetikai vagy füzér kifejezés szintaktikai elemzése

BASIC 22H (5/F253)
A következő sorelemnek A-nak kell lennie (sorvég, ")" jel, idézőjel stb., l. a 3.4.2 pontot.) Eltérő esetben szintaktikai hibát jelez, egyezésnél pedig a BASIC 20H-ra fut

BASIC 23H (5/DEE2)
Numerikus kifejezés kiértékelése. Az aritmetika alaprutinja. A 0214/5H rendszerváltozó által megcímzett (és BASIC 20H-val már megvizsgált) programszövegben induló, tetszőleges bonyolultságú aritmetikai kifejezést értékel ki. Az eredményt a MREG-be tölti. A rutin tetszőleges mélységben újrahívhatja saját magát is (pl. a zárójeles kifejezések kiértékelésénél, függvényértékek kiszámításánál)

BASIC 24H (5/E096)
Füzér kifejezés kiértékelése. Az eredmény itt is a MREG-be kerül (természetesen 03 típuskóddal)

BASIC 25H (5/FB9E)
Az INKEY$ alaprutinja: lekérdezi a billentyűzetcsatorna (69H) állapotát és ha van olvasható karakter, azt A-ba (és a 0205H ideiglenes tárolóba) tölti. Az A = 0 érték és státusz jelzi, ha nem volt karakter (nem nyomtunk meg billentyűt)

BASIC 26H (5/EEBE)
A (HL) címen lévő karakter kódját C-be olvassa és nagybetűsíti (ha belépéskor A = 40H volt) vagy kisbetűsíti (A = 60H esetén)

BASIC 27H (5/F094)
DE = 10 értékkel a következő funkcióra (BASIC 28H) fut

BASIC 28H (5/F097)
Ellenőrzi, hogy van-e még DE bájtnyi szabad hely a BASIC memóriában. Szükség esetén akár a BASIC újabb szegmensekre való kiterjesztésével is megpróbál helyet szerezni. Ha ez nem sikerül, vagy a Z80-as verem nőtt túlságosan nagyra, hibajelzésre fut

BASIC 29H (5/F010)
A BASIC munkavermet lecsúsztatja a változóterület végéhez, így a lehető legnagyobb területet felszabadítja a felülről lefelé terjeszkedő EXOS számára

BASIC 2AH (5/C6C2)
Az elemzett programszövegben átlépi a szóközöket (20H) és TAB (09H) karaktereket

BASIC 2BH (5/E41E)
A programszövegben talált ASCII formátumú számkonstanst alakítja egész vagy lebegőpontos formátumra és a 0595H pufferbe tölti (valamint a 0218/9H rendszerváltozóba is, ha az érték 0 és 9999 közötti egész)

BASIC 2CH (5/E407)
A 0595H pufferbe töltött számot átírja a verembe, a MREG-be

BASIC 2DH (5/D4D4)
Az RND alaprutinja; A-ban és HL-ben egy 3-bájtos véletlen értéket állít elő

BASIC 2EH (5/C3FD)
A HL sorszámot megkeresi az aktuális BASIC programban. Visszatéréskor HL a keresett sor első bájtjára mutat (ha nincs ilyen sor, HL a következő sor elejére áll és CY = 1)

BASIC 2FH (5/C858)
A HL számú BASIC sort kilistázza az aktuális (0209H) csatornára

BASIC 30H (5/C544)
Törli a progamból a HL számú sort, egyben üresre állítja a változóterületek mutatóit és a belső függvényekkel inicializálja a változóláncokat

BASIC 31H (5/C54D)
A BASIC-ben önállóan nem létező CLEAR rutin: törli a BASIC változókat, lezárja az 1—99 csatornákat stb.

BASIC 32H (5/EAD5)
Végigfut az aktuális BASIC programon és bevezeti a változóterületre a DEF és HANDLER sorok változóneveit (ennek alapján lehetséges pl., hogy programszerkesztés közben, törölt változók mellett is tudunk pl. LIST RUTIN-t kérni).

BASIC 33H (5/EF7A)
Végigfut a program hátralévő részén és a vizsgált sorral azonos skatulyázási mélységű sort keres (blokkvéget)

BASIC 34H (5/EF19)
A STOP végrehajtó rutinja.

BASIC 35H (5/EFD7)
Megkeresi a C kódú kulcsszó paraméter blokkját és HL-t a típusbájtra állítja. Visszatéréskor E-ben a típusbájt van

BASIC 36H (5/FBB5)

Egy sort olvas be az aktuális csatornáról a 0248H szövegpufferbe

BASIC 37H (5/C74B)

A 0248H szövegpufferből — tokenizálás után — beilleszti a beolvasott sort a BASIC programba, majd törli a változókat, lezárja az 1—99 csatornákat (BASIC 31H)

BASIC 38H (5/FB66)

Kiírja az A-ban lévő karaktert az aktuális csatornára. Mivel ez minden BASIC kiíratás alaprutinja, nézzük meg, hogyan működik:

```
FB66  C5          PUSH BC          ;regiszterek
FB67  D5          PUSH DE          ;  verembe
FB68  E5          PUSH HL          ;
FB69  F5          PUSH AF          ;
FB6A  47          LD   B,A         ;B=irandó karakter
FB6B  3A 09 02    LD   A,(0209)    ;A:aktuális csatorna
FB6E  FE FF       CP   FF          ;=255? (memóriába?)
FB70  20 0A       JR   NZ,FB7C     ;kiírásra, ha nem

Memóriába ír, ha az aktuális csatornaszám 255 volt:

FB72  2A 42 02    LD   HL,(0242)   ;puffercím
FB75  70          LD   (HL),B      ;karakter a pufferbe
FB76  23          INC  HL          ;következő pozíció
FB77  22 42 02    LD   (0242),HL   ;pointer beállítása
FB7A  AF          XOR  A           ;A=0
FB7B  21 F7 07    LD   HL,07F7     ;ezen az ágon érdek-
FB7E  DF          RST  18          ; telen utasítások

Ide lép, ha az aktuális csatorna nem 255 volt
(B:irandó karakter, A:csatornaszám):

FB7C F7 07        EXOS 07          ;karakter írása
FD7E DF           RST  18          ;hibaellenőrzés

FB7F  F1          POP  AF          ;regiszterek
FB80  E1          POP  HL          ;  visszaolvasása
FB81  D1          POP  DE          ;
FB82  C1          POP  BC          ;
FB83  C9          RET              ;
```

A BASIC aktuális csatornája a 0209H rendszerváltozó tartalma. Ha ez 255-től különböző, a kiíratás az EXOS-on keresztül történik. Ha viszont az érték 255 volt, a BASIC maga végzi el a kiírást: a karaktert a memóriába írja a 0242/3H pointer által mutatott címre, majd a következő pozícióra állítja a pointert.

Az aktuális csatorna általában a #0 (EDITOR), ami a kiírást a képernyőre irányítja. Az olyan utasítások, amelyek más perifériára írnak (pl. LPRINT), csak a 0209H rendszerváltozót állítják át céljuknak megfelelően (printerhez 68H = 104D) mielőtt a közös végrehajtó rutinra futnának

BASIC 39H (5/EEDA)

Kiírja HL értékét az aktuális csatornára. (ASCII alakban). A kiírás formáját és az értéktartományt a C regiszter és az átvitelbit határozza meg:

— NC státusz: 0—65 535 tartomány (5 jegy)

— C státusz: 0—9999 tartomány (4 jegy)

— a C regiszter az értékes jegyek előtti bevezető karakterek kódját adja (C = 20H:SPACE, C = 30H: "0")

BASIC 3AH (5/C4DE)

A tokenizálás alprogramja. A kódolandó forrásszöveg hátralévő részét mozgatja, hogy a tokenizált alak beilleszthető legyen

BASIC 3BH (5/EC83)

A tömbelem keresés alprogramja. Belépéskor HL a változó adatmező kezdetére mutat, C-ben a változó típuskódja van. Ha C tömbváltozót jelöl és B = 1, kiértékeli a megadott indexeket és HL-t a kijelölt elemre állítja

BASIC 3CH (5/CE69)

Csatornaargumentumot igénylő utasítások kiértékelő rutinja. A progamszöveg aktuális pozíciójától kiértékeli és az A-ba tölti a #n paramétert. Hibára ugrik, ha nem ilyen következik vagy értéktartománya nem megfelelő

BASIC 3DH (5/DO68)

A rutin a kódolt programszöveg hátralévő részének szintaktikai helyességét ellenőrzi

BASIC 3EH (5/FB84)

Kiírja a HL címen kezdődő füzért az aktuális csatornára (HL a karakterek számára mutat)

BASIC 3FH (5/E335)

Kiírja a MREG-ben lévő számot az aktuális csatornára

BASIC 40H (5/E093)

Ellenőrzi a programszövegben következő füzér kifejezés szintaktikai helyességét

BASIC 41H (5/DED9)

Ellenőrzi a programszövegben következő numerikus kifejezés szintaktikai helyességét

BASIC 80H (1/D1B2)

A HL címen tárolt változót bevezeti a változóláncba

BASIC 81H (1/D04D)

Tárolja a változó fejrészét a BASIC változóterület végére (0234/5H)-től. A csonka változót be is láncolja

BASIC 82H (1/D07C)
A változóterületre bevezetett numerikus fejrész után beírja az adatmezőt is (nem inicializált alapértékkel)

BASIC 83H (1/D094)
A változóterületre bevezetett füzér fejrész után lefoglalja az adatmezőt is (és nem inicializált alapértéket jegyez be)

A 84H—87H nem hívható funkciókódok (itt a belépési címtáblát másra használja a rendszer).

BASIC 88H (1/D0E7)
Tetszőleges típusú (0—3 kódú) BASIC változót bevezet a változóterületre (nem inicializált alapértékkel)

BASIC 89H (1/D0AB)
A program DEF és HANDLER típusú változóit bevezeti a változóterületre (hibára ugrik, ha már volt ilyen)

BASIC 8AH (1/D101)
A függvénykiértékelés közben talált referenciaparamétert vezeti be a változóterületre. Belépéskor HL az átadandó adatra mutat. A bejegyzett referenciaváltozót a típusbájt BIT 4 = 1 értéke jelzi

BASIC 8BH (1/D189)
Megkeresi a változóterülten azt a változót, amelynek nevére (az elemzett programszövegben) a 0347/8H pointer mutat. Ha nem talál ilyen változót, bevezeti azt nem inicializált alapértékkel. Visszatéréskor HL az adatmező elejére mutat, C a változó típusa

BASIC 8CH (1/D167)
Megkeresi a változó tárolási címét. Abban különbözik az előző (8BH) hívástól, hogy akkor is új elemként vezeti be a változóterületre, ha talált ugyan ilyen bejegyzést a láncban, de az belső függvény (vagy változó) volt

BASIC 8DH (1/D117)
A BASIC terület végén E bájtot lefoglal

BASIC 8EH (1/D153)
Megkeresi a változó tárolási címét. Hasonló a BASIC 8BH-hez, de nem vezet be új elemet, ha nem találja a változót. Visszatéréskor HL a típusbájtra mutat, a státusz

— NC, ha felhasználói változó;
— C, ha beépített változó.

BASIC 8FH (1/D1D9)
Inicializálja a változók bázistábláját (csak a beépített változókat és függvényeket hagyja meg a láncban)

BASIC 90H (1/C261)
A HL-ben megadott címtől újratölti a függvények és belső változók tábláját, majd az alapján inicializálja a változók bázistábláját (0DCBH-tól)

BASIC 91H (1/C2AC)
A HL címen tárolt numerikus értéket a MREG-be tölti. Analóg a BASIC 14H hívással, de az BASIC változó típusú (5 számjegybájt + kitevő) adatot tölt be a verembe, míg ez 6 számjegybájttal dolgozik. Mint láttuk, a verem numerikus adatblokkja 7 számjegybájtot tartalmaz (a változók 5 bájtját elöl-hátul kiegészíti egy-egy, a túlcsordulásokat felfogó bájttal). Ebből az alsó 6-ot mozgatja a BASIC 91H és a következő néhány funkcióhívás

BASIC 92H,n (1/C28D)
Kétbájtos funkcióhívás: a konstanstábla (1. ROM C08CH címtől) n. elemét átviszi a MREG-be. A hívó gépi kódból még egy bájtot beolvas, ezt tekinti argumentumnak. Hívása tehát assembly programból

```
RST 10H
DEFB 92,n
...
```

alakban érhető el

BASIC 93H, nn (1/C29F)
Hárombájtos funkcióhívás. A hívó program következő két bájtján adható meg az a cím, ahonnan a BASIC 91H hívással be kell olvasni a numerikus adatot a MREG-be:

```
RST 10H
DEFB 93H, nLO, nHI
...
```

BASIC 94H (1/C2A4)
Az 1. rögzített munkaregiszter (0E0BH) tartalmát tölti a MREG-be:
REG1 → MREG

BASIC 95H (1/C1A9)
A 2. rögzített munkaregiszter (0E12H) tartalmát tölti a MREG-be:
REG2 → REG

BASIC 96H,nn (1/C2C6)
Hárombájtos funkcióhívás: a MREG tartalmát a hívó gépi kódú programban következő két bájton megadható címre tölti (a BASIC 93H,nn hívás fordítottja)

BASIC 97H (1/C2CC)
A MREG tartamát az 1. rögzített munkaregiszterbe (0E0BH) tölti:
MREG →REG1
Az átmásolt elemet törli a veremből

BASIC 98H (1/C2D1)
A MREG tartalmát a 2. rögzített munkaregiszterbe (0E12H) tölti:
MREG → REG2
Az átmásolt elemet törli a veremből

BASIC 99H (1/C2E4)
Zárójeles numerikus kifejezés (függvény argumentum) kiértékelése. Az eredmény a MREG-be kerül

BASIC 9AH (1/C2F0)
A programszövegben "(" jelet vár, az utána következő numerikus kifejezést kiértékeli és a MREG-be tölti. Több argumentumos függvények (pl. SPEEK (SG,CIM)) első paraméterének kiértékelésére alkalmazható

BASIC 9BH (1/C2ED)
Folytatja a többtagú argumentum kiértékelését: ","-t vár, az utána következő numerikus kifejezést kiértékeli és a MREG-be tölti

BASIC 9CH (1/C2F7)
Zárójeles füzér kifejezés (füzérváltozós függvény argumentuma) kiértékelése. Az eredmény a MREG-be kerül

BASIC 9DH (1/C2FF)
"(" jelet vár, az utána következő füzér kifejezést kiértékeli és a MREG-be tölti (többtagú függvényargumentum első része)

BASIC 9EH (1/C2FC)
","-t vár, az utána következő füzérkijelzést kiértékeli és a MREG-be tölti

BASIC 9FH (1/C306)
Zárójelet ("(") és tömbváltozó nevet vár a programszövegben (egyébként hibát jelez). Kilépéskor HL a tömb adatmező kezdetére mutat

BASIC A0H (1/C2E7)
A programszövegben ")" jelet vár (többtagú argumentum vége)

BASIC A1H (1/C68D)
Az aktuális programban még használható bájtok számát tölti a munkaregiszterbe (FREE változó kiértékelő rutinja)

A következő rutinok (A2H—B2H) trigonometriai és más matematikai függvények kiértékelő eljárásai. A hasonló funkciójú BASIC függvények (pl. ATN) kiértékelő rutinjai a függvényargumentum meghatározása után hívják meg ezeket az alprogramokat.

BASIC A1H (1/C399)
Az *arc tg* függvény kiértékelő rutinja. Az argumentumot a MREG-ben várja és a függvényértéket (amelyet radiánban határoz meg) szintén oda tölti:
MREG = ATN (MREG)

BASIC A3H (1/C368)
Az *arc sin* függvény kiértékelő rutinja. Ellenőrzi a MREG-ben kapott argumentum értéktartományát ($-1 <= x <= +1$), egyébként hibajelzésre ugrik) és a radiánban meghatározott függvényértéket a MREG-be tölti
MREG = ASIN(MREG)
Az $\arcsin(x) = \text{arc tg} \frac{x}{\sqrt{1-x^2}}$
összefüggés felhasználásával számítja ki az értéket

BASIC A4H (1/CB86)
A *sin* függvény kiértékelő rutinja. Argumentumát — radiánban — a MREG-ben várja. Ellenőrzi az adat értékhatárait; meglehetősen misztikus módon csak az $ABS(x) <= 180/\pi$ értéket fogadja el. Ezután a

$$-\frac{\pi}{2} — +\frac{\pi}{2}$$

tartományba transzformálja az argumentumot (kihasználva a sin periodikusságát) és elvégzi a sorejtést. Az eredményt a MREG-be tölti:
MREG = SIN(MREG)

BASIC A5H (1/C518)
A *cos* függvény kiértékelő rutinja. Argumentumát a MREG-ben várja. MREG = MREG+PI/2 transzformáció után a *sin* rutinra (BASIC A4H) fut

BASIC A6H (1/CC9F)
A tg függvény kiértékelő rutinja a

$$\text{tg}\, x = \frac{\sin x}{\cos x} = \frac{\sin x}{\sin (x + \pi/2)}$$

azonosság felhaszálásával számítja ki a függvényértéket

BASIC A7H (1/C5F0)
Az $e^x$ függvény kiértékelő rutinja. Az argumentumát a MREG-ben várja és ide tölti a függvényértéket is:
MREG = EXP (MREG)

BASIC A8H (1/CA80)
A radiánban adott változót fokra számítja át, ha az aktuális üzemmód "fok":
MREG = 180/PI·MREG

BASIC A9H (1/C586)
A fokban adott változót radiánra számítja át, ha az aktuális üzemmód "fok":
MREG = PI/180·MREG

BASIC AAH (1/C8A7)
A tízes alapú logaritmus kiértékelő rutinja
MREG = LOG10(MREG)

BASIC ABH (1/C98D)
A *modulofüggvény* kiértékelő rutinja. Az első argumentumot (az osztandót) MREG1-ben, a második argumentumot (az osztót) MREG2-ben várja. Belépéskor a BASIC veremmutató MREG2-re mutat. Kiértékelés után a függvényértéket (maradék) MREG1-be tölti és a veremmutatót is erre állítja:
MREG1 = MOD(MREG1,MREG2)

BASIC ACH (1/CC52)
A rutin a MREG-ben megkapott argumentum négyzetgyökét határozza meg (előjelvizsgálatot maga a rutin nem végez, tehát feltétlenül pozitív — vagy 0 — értéket kell átadnunk).
MREG = SQR(MREG)

BASIC ADH (1/CA8E)
A rutin a MREG-ben megkapott numerikus érték reciprokát adja vissza:
MREG = 1/MREG

BASIC AEH (1/C778)
Az INT függvény kiértékelő alprogramja
MREG = INT(MREG)

BASIC AFH (1/CB17)
A ROUND függvény kiértékelésénél használt alprogram. Az első argumentumot (a kerekítendő számot) a MREG-ben várja, A-ban pedig a számjegyek kért számát (tizedespont előtt és után összesen) kell megadni: Az értékeket már nem ellenőrzi, tehát a megadott paramétereknek matematikailag feltétlenül értelmesnek kell lenniük.

BASIC B0H (1/C7B0)
A hatványozás ($x^y$) kiértékelő rutinja. Az alapértékét a MREG1-ben, a kitevőt MREG2-ben várja. Belépéskor a BASIC veremmutató MREG2-re mutat

BASIC B1H (1/C820)

AZ IP függvény kiértékelő rutinja: a MREG-ben kapott argumentum tizedespont utáni jegyeit 0-ra állítja:
MREG = IP(MREG)

BASIC B2H (1/CBF9)

A signumfüggvény kiértékelő rutinja. Argumentumát a MREG-ben kapja és az adat értéktartományától függően tölt +1-et, 0-t vagy –1-et a MREG-be

A következő 4 BASIC funkcióhívás a különböző számon megnyitott BASIC programok memóriaigénylésének és nyilvántartásának alaprutinja.

BASIC B3H (1/C1D9)

Felszabadítja (az EXOS-on keresztül) a C. szegmenst, törli a RAM-térképből az ehhez a szegmenshez kapcsolódó elemet. Ha a szegmens előzőleg megosztott volt, törli a 020BH rendszerváltozót is

BASIC B4H (1/C1B9)

Szegmenst kér a C sorszámú program számára az EXOS-tól. A kapott szegmenst a megadott programszámmal nyilvántartásba veszi a RAM-térképen. Ha megosztott szegmest kapott, a szegmens számát 020BH-be, a hozzá rendelt program számát 020DH-be is betölti.

Visszatéréskor a státusz:

— Z, ha teljes szegmenst kapott;
— NZ, P, ha megosztott szegmenst kapott;
— M, ha az EXOS már nem tudott szegmenst kiutalni.

BASIC B5H (1/CIEF)

Megkeresi a RAM-térképen (0E54H-tól) a C. szegmens bejegyzését. Visszatéréskor HL a tárolt elemre mutat:

(HL) a szegmens száma
(HL + 1)a szegmensen tárolt program száma

BASIC B6H (1/C1F9)

Megkeresi a RAM-térképen a C. program bejegyzését. Visszatéréskor a státusz

— C, ha nincs bejegyezve;
— NC, ha megtalálta. Ekkor HL a keresett elemre mutat:
(HL): a programot tároló szegmens száma,
(HL + 1): a keresett program száma.

BASIC B7H (1/D1F3)

A SET utasítás végrehajtó alprogramja. Hívása előtt az értelmező csak az aktuális csatornát állítja be. A rutin hívásakor a pointerek a programszöveg következő részére mutatnak

BASIC B8H (1/D2AC)

Az ASK, SET, TOGGLE után következő gépi feltételeket azonosítja. Visszatéréskor a státusz C, ha azonosította a kulcsszót.

A: csatornaszám a végrehajtáshoz (alapértelmezés). Pl. INK esetén A = 65H (grafikus csat.);

C: BIT 7,C = 0, ha a végrehajtás ESC szekvenciaként történhet (pl. PAPER);
BIT 7,C = 1, ha a kulcsszó EXOS változót jelöl (pl. TIMER). Ekkor C alsó bitje a változó számát adja.

BASIC B9H (1/D430)

BASIC sorszámtartományt értékel ki az elemzett sor következő részében (100 TO 200, FIRST — stb. típus), majd meghívja a BASIC BAH funkciót, azaz listáz vagy töröl az adott tartományban.

BASIC BAH (1/D4A8)

Listázza az aktuális BASIC program HL (kezdősorszám) és DE (végsorszám) közötti tartományát. Az elvégzendő műveletet a háttérstátusz (AF'!) határozza meg:

— listáz, ha C;
— töröl, ha NC.

BASIC BBH (1/D546)

Az AUTO parancs végrehajtó rutinja. Beolvassa a programszövegben esetleg következő paramétereket és

— kiírja az aktuális sorszámot,
— beolvas egy sort,
— tokenizálja,
— beépíti a programba.

Ezt ciklikusan ismétli mindaddig, amíg a STEP értéknek megfelelően növelt aktuális sorszám 9999 fölé nem nő vagy más módon (üres sor, hiba, STOP) ki nem lépünk az üzemmódból

BASIC BCH (1/D5CF)

A RENUMBER parancs végrehajtó programja

BASIC BDH (1/DA48)

A PRINT (és LPRINT) utasítás másodrutinja. Ellenőrzi a kulcsszó utáni programszöveg (AT x,y; #CSAT; USING stb.) szintaktikai helyességét

BASIC BEH (1/DA9D)

A PRINT utasítás végrehajtó programja. Hívása előtt az értelmező beállítja az aktuális csatorna értékét. A programszöveg pointerei a token utáni elemre mutatnak

BASIC BFH

Belépési címe

— UK módban 1/DF4B,

— BRD módban 4/EDE0.

Kiírja a HL-ban megadott hibakódhoz tartozó magyarázó szöveget a 0446H hibaüzenet-pufferbe. A 9000—9999 kódtartományban az EXOS 28 hívással a rendszertől kér értelmezést (de ezt is a fenti szegmensek adják meg, ha más bővítő nem válaszol előbb)

BASIC C0H (1/D746)

A WAIT utasítás végrehajtó programja

BASIC C1H (1/D773)

Végigfut az aktuális programon és a blokk-kezdet, ill. blokkvég utasítások alapján beírja a sorok skatulyázási mélységet jelző bájtjait

BASIC C2H (1/D7B5)

A TIME utasítás végrehajtó programja

BASIC C3H (1/D7C5)

A DATE utasítás végrehajtó programja

BASIC C4H (1/CD2F)

A MREG-ben lévő stringet nagybetűsíti vagy kisbetűsíti:
MREG = UCASE$(MREG), ha B = 60H
MREG = LCASE$(MREG), ha B = 40H

BASIC C5H (1/D812)

Beállítja a funkcióbillentyűk alapértelmezését (az 1. szegmens C842H címen kezdődő táblája alapján)

A felsorolást áttekintve láthatjuk, hogy a BASIC funkcióhívásokkal egy igen jól használható programkönyvtárhoz jutottunk. Egy olyan rutingyűjtemény ez, amelyben a matematikai kiértékelő rutinok (a számtani alapműveletektől a trigonometriai és egyéb függvényekig), a szintaktikai elemzést, sorfeldolgozást segítő alprogramok és praktikus BASIC segédprogramok (beolvasás, kiíratás stb.) rendkívül széles körét találhatjuk meg. Ráadásul, ami már igazán a szolgáltatások "non plus ultra"-ja: mindez egybájtos gépi kódokkal érhető el (nem számítva a funkcióhívások sorozatát bekapcsoló D7H — RST10H — és befejező 00H kódokat). Úgy használhatjuk tehát gépi kódú programjainkban akár a legbonyolultabb lebegőpontos műveletet is, mint ha egy hallatlanul megokosított processzort programoznánk. Legalábbis formailag, hiszen azzal, hogy igénybe vesszük a BASIC funkcióhívások lehetőségét, a végrehajtást tulajdonképpen az értelmezőre bízzuk, elfogadva a hívás idejére annak sebességbeli, szintaktikai és egyéb korlátait is.

Befejezésként lássunk egy-két példát a BASIC funkcióhívások alkalmazására. Természetesen a legjobb példa a felhasználásra a BASIC bővítések felépítése lesz, de szemléltetésként talán itt sem árt néhány egyszerű, BASIC-ben kialakított rutinban bemutatni a legfontosabb hívások alkalmazását.

Kezdjük az aritmetikával! A két legalapvetőbb rutin talán a konstans bevitele a MREG-be, ill. kiíratása az aktuális csatornára. Ezeket egyetlen rövid rutinban is bemutathatjuk:

```
RST 10          ; funkcióhívások bekapcsolása
BASIC 92,n      ; n. konstans MREG-be
BASIC 3F        ; kiíratás
BASIC 00        ; funkcióhívások vége
RET
```

Építsük fel a rutint BASIC-ből úgy, hogy a betöltendő konstans sorszáma helyére (kezdőcím+2) POKE utasítással sorra beírjuk a 0—42 értékeket:

```
100 ALLOCATE 100
110 CODE M=HEX$("D7.92.00.3F.00.C9")
120 FOR N=0 TO 42
130   POKE M+2,N
135   PRINT N;TAB(5);":";
140   CALL USR(M,0)
150   PRINT
160 NEXT
```

Érdemes áttekinteni a futás eredményét is, hiszen amellett, hogy a konstansok egy része a sorfejtések együtthatója, néhány értéket esetleg mi is felhasználhatunk rutinjainkban:

```
START
 0  : .5
 1  : 1
 2  : 9.99999999E62
 3  : 32768
 4  : 3.141592654
 5  : 57.29577951
 6  : 1.745329252E-02
 7  : 1.570796327
 8  : 1.047197551
 9  : .5235987756
 10 : 6.283185307
 11 : .223607
 12 : .894427
 13 : .316227766
 14 :-.1666666661
 15 : 8.333330721E-03
 16 :-1.984083282E-04
 17 : 2.752397107E-06
 18 :-2.386834641E-08
 19 :-1.440083449
 20 :-.7200268489
 21 : 4.320250389
 22 : 4.752225846
```

```
 23 : .2679491924
 24 : 1.732050808
 25 : .7320508076
 26 :-2.915681438
 27 : 3.163034916
 28 :-.6735816015
 29 :-10.07040695
 30 : 16.9669814
 31 :-8.190800455
 32 : ,8685889638
 33 : 2.302585093
 34 : 3.321928095
 35 : .8685889638
 36 : 1.151
 37 : 2.92546497E-04
 38 : 504.4648895
 39 : 14.00829976
 40 : 3.328736465E-02
 41 : 1008.929779
 42 : 112.0940811
ok
```

Néhány alapkonstans tehát

C0 = 0,5
C1 = 1
C2 = INF (a gépen ábrázolható max. érték)
C3 = 2 ^ 15
C4 = PI
C5 = 180/PI (1 radián értéke fokokban)
C6 = PI/180 (1 fok értéke radiánban)

(Megjegyezzük, hogy a gép a konstansokat 6 bájton — 12 számjeggyel — tárolja, de a kiíró rutin ezt automatikusan 10 jegyre kerekíti).

Ezek után végezzük el az alapműveleteket pl. az első két konstans között

```
a, 1 + 0,5: MA  RST 10
                BASIC 92,01      ; 1 → MREG1
                BASIC 92,00      ; 0,5→ MREG2
                BASIC 0C         ; MREG1 = MREG1 + MREG2
                BASIC 3F         ; kiíratás
                BASIC 00
                RET

b, 1 – 0,5: MB  RST 10
                BASIC 92,01
                BASIC 92,00
                BASIC 0D         ; MREG1 = MREG – MREG2
                BASIC 3F
                BASIC 00
                RET
```

Ezek analógiájára már könnyen elkészíthető a

c, $1 \cdot 0{,}5$

d, 1/0.5

számítás rutinja is. A BASIC betöltő, és a futtatás eredménye:

```
100 ALLOCATE 100
110 CODE MA=HEX$("D7.92.01.92.00.0C.
    3F.00.C9")
120 CODE MB=HEX$("D7.92.01.92.00.0D.
    3F.00.C9")
130 CODE MC=HEX$("D7.92.01.92.00.0E.
    3F.00.C9")
140 CODE MD=HEX$("D7.92.01.92.00.0F.
    3F.00.C9")
150 CALL USR(MA.0):PRINT
160 CALL USR(MB.0):PRINT
170 CALL USR(MC.0):PRINT
180 CALL USR(MD.0):PRINT.

START
 1.5
 .5
 .5
 2
ok
```

Így persze nemigen lehet megkülönböztetni a kivonás és a szorzás hatását, de van elég konstans a további kísérletezéshez.

Hogy egy elemi függvényt is használjunk, határozzuk meg fokokban, hogy melyik az a szög, amelynek szinusza 0,5:

```
RST 10
BASIC 92,00      ; 0,5→MREG
BASIC A3         ; MREG = ASIN(MREG)
BASIC 92,05      ; MREG2 = 180/PI
BASIC 0E         ; MREG = 180/PI · ASIN(0,5)
BASIC 3F
BASIC 00
RET
```

A programot betöltve és futtatva meggyőződhetünk a számítás helyességéről:

```
100 ALLOCATE 100
110 CODE M=HEX$("D7.92.00.A3.92.05.
    0E.3F.00.C9")
120 CALL USR(M.0):PRINT

START
 30
ok
```

Minőségileg újat jelent, ha túllépünk a belső konstansok körén és a BASIC szövegből adunk át paramétert rutinunknak. Ehhez egyelőre használjuk fel továbbra is a CALL USR hívást, és a feldolgozni kívánt programszöveget írjuk egyszerűen az utasítás után. Ez a nem túl elegáns, de jól működő megoldás azon alapul, hogy az értelmező elvégzi a CALL USR(CIM,HL) utasítás elemzését és a következő elem vizsgálata után lép be a felhasználói gépi rutinba. Az aktuális programszöveg pointerei az utasítást követő elemre mutatnak. Ezt használtuk ki a változóterület bemutatásánál — még magyarázat nélkül használt — V—CIM rutinban, amely a CALL USR (V—CIM,0)A alakban az utasítás után írt változó tárolási címét kereste meg. Most már jól követhető a rutin működése:

```
RST 10          ; funkcióhívás indul
BASIC 8E        ; HL = a változó tárolási címe
BASIC 00        ; hívás vége
PUSH HL         ; cím verembe
RST 10          ; funkcióhívás indul
BASIC 20        ; átlépi az elemet
BASIC 00        ; hívás vége
POP HL          ; cím vissza
RET
```

Egy másik alkalmazásként írjunk egy köbszámító rutint:

```
RST 10          ; funkcióhívás indul
BASIC 23        ; Következő szakasz kiértékelése
BASIC 15        ; MREG2=MREG1=X
BASIC 15        ; MREG3=MREG2=X
BASIC 0E        ; MREG2=MREG2·MREG3=X^2
BASIC 0E        ; MREG1=MREG1·MREG2=X^3
BASIC 3F        ; kiíratás
BASIC 00        ; hívás-sorozat vége
RET
```

A használatra a betöltőprogram ad példát:

```
100 ALLOCATE 100
110 CODE M=HEX$("D7,23,15,15,0E,0E,.
    3F,00,C9")
120 INPUT PROMPT "A szam: ":SZAM
130 PRINT "A szam köbe:";
140 CALL USR(M,0) SZAM
150 PRINT
160 GOTO 120
```

```
START
A szam: 2
A szam köbe: 8
A szam: 5
A szam köbe: 125
```

```
A szam: 10
A szam kobe: 1000
A szam:

STOP at line 120
ok
```

Befejezésül próbáljuk ki a listázó funkcióhívást is:

```
LD HL,0002      ; kezdő sorszám (2)
LD DE,0004      ; végsorszám (4)
OR A            ; CY=1 (LIST)
EX AF,AF'       ; háttérbe
RST 10          ; funkcióhívás indul
BASIC BA        ; LIST 2-4
BASIC 00        ; hívás vége
RET
```

Mindez a következőképpen működik a BASIC programban:

```
   1 '
   2 '
   3 '
   4 '
   5 '
 100 ALLOCATE 100
 110 CODE M=HEX$("21.02.00.11.04.00.
     37.08.D7.BA.00.C9")
 120 CALL USR(M.0)

START
   2 '
   3 '
   4 '
ok
```

## 3.4 A BASIC értelmező működése

Az értelmezőprogram, miután az inicializálás folyamán kiépítette saját hátterét (rendszerváltozók, csatornák stb.) gyakorlatilag egy végtelen ciklusban pörög:

— állandóan figyeli a billentyűzetet
— egy sor beírása (ENTER) után megkísérli BASIC utasításként vagy sorként értelmezni a beírt karakterek sorozatát
— eközben saját szabályai szerint elemzi és kódolja a felismert elemeket
— majd ha a sor megfelel a szintaktikai szabályoknak, elkezdi a végrehajtást.

Ez a legtöbb programnál előbb-utóbb végetér, és akkor az értelmező újra kezdi a ciklust a billentyűk figyelésével.

Az értelmező működése tehát három fő ágban:
— a beolvasási
— a tokenizálási (elemzés, kódolás)
— és a végrehajtási szakaszban foglalható össze.

Az ezekben elvégzett funkciókhoz, a használt rendszerváltozókhoz, pointerekhez nekünk is kapcsolódnunk kell, ha a gépi kódú programok írása során bővíteni (vagy akár csak használni) szeretnénk a BASIC rendszer szolgáltatásait. Ezért tekintsük át röviden (alkalmanként azért belenézve a működésbe) a három főágat.

### 3.4.1 Beolvasási főág

Az értelmező az inicializálás után közvetlenül a beolvasási főágba ér. Első teendője a BASIC által külön területen fenntartott Z80-as verem beállítása, majd az állapotsor üzenetszövegének kiíratása:

```
C2EC  31 BB 0D    LD   SP,0DBB     ;Z80 verem a BASIC-ben
C2EF  CD D1 C3    CALL C3D1        ;flag-állítás
C2F2  DB B2       IN   A,(B2)      ;2. LAP aktuális szg.
C2F4  F5          PUSH AF          ;verembe
C2F5  3E FF       LD   A,FF        ;rendszerszegmens
C2F7  D3 B2       OUT  (B2),A      ;a 2. LAP-ra
C2F9  32 09 02    LD   (0209),A    ;aktuális csatorna
                                    =255 (memóriába ír)
C2FC  2A F6 BF    LD   HL,(BFF6)   ;az állapotsor címe
C2FF  11 06 00    LD   DE,0006     ;az első 6 karaktert
C302  19          ADD  HL,DE       ;nem írja át
C303  22 42 02    LD   (0242),HL   ;puffercímként beír
C306  CD 00 CE    CALL CE00        ;köv. szöveg kiírása:

C309  14          DEFB 14          ;szöveghossz
C30A  ...         DEFB "IS-BASIC     program"

C31E  EB          EX   DE,HL       ;puffercím DE-be
C31F  DD 6E 0A    LD   L,(IX+0A)   ;L=aktuális prg. száma
C322  0E 20       LD   C,20        ;
C324  37          SCF              ;
C325  CD DA EE    CALL EEDA        ;kiírja L értékét
C328  CD 00 CE    CALL CE00        ;utána 7 szóközt ír:

C32B  07          DEFB 07          ;szöveghossz
C32C  ...         DEFB "       "

C333  F1          POP  AF          ;eredeti 2. LAP szg.
C334  D3 B2       OUT  (B2),A      ;visszalapozza
```

A végrehajtások, megindítás, hibakezelés után is mindig ide (pontosabban a C2F2H címre) lép a rendszer, valahányszor a beolvasási főág visszakapja a vezérlést.

Számunkra is hasznos lehet az itt is többször alkalmazott 5/CE00H rutin. Ez az aktuális csatornára (ez most éppen az állapotsor memóriaterülete) kiírja azt a szöveget, amely a hívó utasítást (CALL CE00H) követi (első a hosszbájt, utána a karakterek), majd a szöveget átlépve tér vissza a hívó rutinba.

Az értelmező többi üzenetváltása már az EDITOR csatornán (alapértelmezésben #0) történik. A továbbiakban tehát ezt állítja be az értelmező aktuális csatornaként, ellenőrzi és ide írja ki az "ok" üzenetet, ha megengedett (pl. programsor beírás után nem ír "ok"-t). A különböző üzemmódok jelzésére a 0200H rendszerváltozó, flagbájt szolgál (mint láttuk, az IX regiszter a BASIC-ben mindig ezt a változót címzi meg).

Ezután már következhet a beolvasás. Előtte még beállítja a progam a BASIC-ben szokásos EDITOR jelzőket, majd a sorbeolvasó rutinjára lép:

```
C336  AF           XOR  A          ;A=0
C337  32 09 02     LD   (0209),A   ;akt. csat. (EDITOR)
C33A  DD CB 00 8E  RES  1,(IX)     ;parancsmód
C33E  DD CB 00 46  BIT  0,(IX)     ;"ok" írható?
C342  28 0C        JR    Z,C350    ;átlépi, ha nem
C344  CD D0 F1     CALL  F1D0      ;EDITOR ellenőrzése
C347  CD 00 CE     CALL  CE00      ;"ok" kiíratás

C34A  05           DEFB  05        ;szöveghossz
C34B  6F           DEFB  "ok"
C34D  19           DEFB  19        ;sorvégig töröl
C34E  0D           DEFB  0D        ;CR (kocsivissza)
C34F  0A           DEFB  0A        ;LF (soremelés)

C350  DD CB 00 C6  SET  0,(IX)     ;"ok" írható
C354  CD D0 F1     CALL  F1D0      ;EDITOR ellenőrzés
C357  16 14        LD    D,14      ;NO_SOFT, AUTO_ERA
C359  01 20 01     LD    BC,0120   ;FLAG_EDIT írása
C35C  F7 10        EXOS  10        ;jelzők az EDITOR-ba
C35E  CD D6 C3     CALL  C3D6      ;sor beolvasás
```

Itt álljunk meg egy pillanatra. A beolvasási főág leglényegesebb momentumához érkeztünk: az értelmező most vár a felhasználó utasítására, ami a további teendőket dönti el. A kurzor villog, a képernyőn sorra megjelennek a leütött karakterek. Érdekes módon a vezérlés ilyenkor nem is a BASIC-é: az operációs rendszer EDITOR perifériakezelője gondoskodik a gép és kezelője kapcsolatáról, a szerkesztési lehetőségről. A vezérlés akkor kerül vissza a BASIC-hez, amikor — az utasítás, sor befejezésekor — lenyomjuk az ENTER billentyűt (esetleg a STOP-ot).

A beolvasó rutin először betölti az EDITOR-tól kapott karaktereket a szövegpufferbe (0248H:hossz, 0249H-tól a karakterek), majd bizonyos mértékig fel is dolgozza a puffer tartalmát:

```
C3D6  37           SCF             ;
C3D7  CD B6 FB     CALL  FBB6      ;egy sor a pufferbe
C3DA  D8           RET   C         ;vissza, ha STOP volt
C3DB  21 48 02     LD    HL,0248   ;puffer kezdőcím
C3DE  22 16 02     LD    (0216),HL ;pointerbe
C3E1  23           INC   HL        ;1. karakter címe
C3E2  22 14 02     LD    (0214),HL ;pointerbe
C3E5  CD A0 C5     CALL  C5A0      ;1. elem vizsgálata
C3E8  CD 4B C7     CALL  C74B      ;beépíti, ha prg.-sor
C3EB  30 E9        JR    NC,C3D6   ;újraolvas, ha nem volt
                                     feldolgozandc sor
C3ED  3A 04 02     LD    A,(0204)  ;1. elem típusa
C3F0  FE 02        CP    02        ;sor vége?
C3F2  38 E2        JR    C,C3D6    ;újraolvas, ha igen
C3F4  B7           OR    A         ;NC státusz
C3F5  C9           RET             ;
```

A sorfeldolgozással, a programszöveg elemeinek vizsgálatával, kódolásával (a következő pontban) külön is foglalkozunk, hiszen ez eredményezi a sokszor említett tokenizált szöveget. Most annyit figyeljünk meg, hogy a beolvasó rutinból ki sem lép az értelmező, ha csak egy üres sort írtunk be. Ha a beírt sor számmal kezdődik, azt — mint programsort — tokenizálás és szintaktikai elemzés után beépíti az értelmező az aktuális BASIC programba.

A beolvasási főágra csak akkor kerül vissza a vezérlés, ha a beírt sor egy feldolgozandó, majd végrehajtandó parancs (ill. ha STOP-ot nyomtunk meg).

Befejezésül az értelmező — felhasználva, hogy a beolvasó rutin már megvizsgálta a sor első elemének típusát — még különválaszt egy esetet: azonnal átadja a szöveget (egy EXOS hívással) a rendszerbővítőknek, ha az kettősponttal kezdődik (pl. :HELP vagy :BRD stb):

```
C361  38 ED       JR    C.C350     ;vissza, ha STOP volt
C363  FE 0A       CP    0A         ;az 1. elem "*"?
C365  28 F7       JR    Z.C35E     ;újraolvas, ha igen
C367  FE 10       CP    10         ;":"?
C369  20 11       JR    NZ.C37C    ;végrehajtásra, ha nem

Kettősponttal kezdődő sor (például :HELP) esetén:

C36B  3A 48 02    LD    A,(0248)   ;A=a sor hossza
C36E  ED 5B 14 02 LD    DE.(0214)  ;DE=1. karakter címe
C372  1B          DEC   DE         ;elé
 373  D6 03       SUB   03         ;hossz-3
C375  12          LD    (DE),A     ;szöveg hossza
C376  F7 1A       EXOS  1A         ;bővítőkhöz
C378  DF          RST   18         ;EXOS hibakezelés
C379  C3 F2 C2    JP    C2F2       ;főág elejére ugrik
```

A pufferben tehát most egy BASIC parancs vagy többutasításos sor van. Mielőtt követnénk az értelmezőt a végrehajtás folyamán, foglaljuk össze, hogy miként is elemez egy sort az értelmező, mit is jelentenek a különböző típusú elemek, hogyan néz ki végül egy tokenizált sor.

### 3.4.2 Sorelemzés, tokenizálás

Mindenekelőtt nézzünk meg néhány utasítást és próbáljuk elemekre bontani őket. Állapítsuk meg, hogy milyen elemeket lehet megkülönböztetni a szövegben.

1. PRINT 12
Az utasítás két eleme:

— PRINT: BASIC kulcsszó
— 12: numerikus konstans

2. LET SZAM = SIN(PI)+0,5
Itt már több elemet láthatunk:

— LET: BASIC kulcsszó
— SZAM: numerikus változó
— " = ": jel
— SIN: belső függvény
— "(": jel
— PI: belső változó
— ")": jel
— "+": jel
— 0,5: numerikus konstans

3. LET A$ = "NEV"
— LET: BASIC Kulcsszó
— A$: füzér változó
— " = ": jel
— "NEV": füzér "konstans"

4. GOTO 100
— GOTO: BASIC kulcsszó
— 100: olyan numerikus konstans, amely BASIC sorszámot jelöl

Ezekben a példákban az értelmező valamennyi megkülönböztetett elemtípusát láthattuk, és egyben a programszöveg szeletelési módját is: az értelmezőprogram ugyanilyen elemekben vizsgálja, kódolja, majd hajtja végre az utasításokat.

Mit is jelent a már sokszor emlegetett kódolás, tokenizálás? Nos, az értelmező nem eredeti formájában tárolja pl. a programban a beírt sorokat. Saját munkáját megkönnyítendő, előre bejegyzi a sorokba a különböző feldolgozást, végrehajtást igénylő elemek típuskódját, hosszát, konstansok esetén a bináris vagy BCD értéket stb. Ez persze valamelyest meghosszabítja a sorok bevitelét, de ez a veszteség sokszorosan megtérül a futásidőben.

Hány elemtípust kell végül is megkülönböztetni? Mint láthattuk, az utasítások elején mindig egy BASIC *kulcsszó* áll (kivétel ez alól a megengedett "A = ..." típusok, amelynél nyilván az értelmezőnek kell egy LET kulcsszót az értékadás elé képzelnie). Lehetnek a szövegben még *konstansok* (numerikusak vagy szövegesek), formailag egy csoportba sorolható felhasználói és belső *változók* vagy függvények és különböző *jelek*. Az IS-BASIC által használt típuskódok a következők:

| Típuskód | Elem |
|---|---|
| 00H | jel |
| 20H | numerikus változó vagy függvény |
| 40H | füzérváltozó vagy függvény |
| 60H | BASIC kulcsszó |
| 80H | füzérkonstans |
| A0H | BASIC sorszám (10000 alatti egész) |
| C0H | numerikus konstans |

Az értelmező tehát kódoláskor nem különözteti meg a változókat és függvényeket, de még a felhasználói vagy belső változatokat sem. Így könnyű a tokenizálás (a betűvel kezdődő karaktersorozatok automatikusan ebbe a csoportba kerülnek; még a kulcsszavak is az elemzés legelső stádiumában) és a kiértékeléskor válik majd szét a különböző típusok kezelése (a típuskódok alapján).

Látható, hogy a típuskódokban csak a felső 3 bit használt. Az elemekhez ragasztott típusjelző alsó 5 bitje tehát még további információkat hordozhat. Egyes típusoknál ki is használja ezt a rendszer (pl. a változónév hosszának beírására; ez maximálja a változók nevét $2^5 - 1 = 31$ karakterben). A feldolgozás során a típuskód AND E0H maszkolással, a kiegészítő információ AND 1FH maszkolással emelhető ki.

Tekintsük át ezek után, hogy az egyes elemtípusokat milyen formában kódolja az értelmező.

1. Jel (00H típuskód)

   Elem hossza: 1 bájt

   1.: típusjelző: 00H+ a jel kódja. A jel kódja a 21H—2FH ASCII kód tartományban egyszerűen az ASCII kód –20H, általában pedig a következő:

| kod HEX | kod DEC | jel | kod HEX | kod DEC | jel | kod HEX | kod DEC | jel |
|---|---|---|---|---|---|---|---|---|
| 01 | 1 | ! | 0A | 10 | * | 13 | 19 | = |
| 02 | 2 | " | 0B | 11 | + | 14 | 20 | > |
| 03 | 3 | # | 0C | 12 | , | 15 | 21 | <> |
| 04 | 4 | $ | 0D | 13 | - | 16 | 22 | <= |
| 05 | 5 | % | 0E | 14 | . | 17 | 23 | >= |
| 06 | 6 | & | 0F | 15 | / | 18 | 24 | [ |
| 07 | 7 | ' | 10 | 16 | : | 19 | 25 | \ |
| 08 | 8 | ( | 11 | 17 | ; | 1A | 26 | ] |
| 09 | 9 | ) | 12 | 18 | < | 1B | 27 | ↑ |

2. Numerikus változó vagy függvény (20H típuskód)

   Elem hossza: 1 + a név betűinek száma (n)

   1.: típusjelző: 20H + a változónév karaktereinek száma (1—31)

   2 – (n + 1).: a név karakterei (nagybetűsítve)

3. Füzérváltozó vagy függvény (40H típuskód)

   Elem hossza: 1 + a név betűinek száma (beleértve a "$" karaktert is)

   1.: típusjelző: 40H + a név karaktereinek szám (max. 31)

   2 – (n + 1).: a név karakterei (nagybetűsítve)

4. BASIC kulcsszó (60H típuskód)

   Elem hossza: 2 bájt

   1.: típusjelző: 60H

   2.: token: a kulcsszó kódja, tulajdonképpen a kulcsszótáblabeli sorszáma (l. pl. 6. Függelék)

5. Füzérkonstans (80H típuskód)
   Elem hossza: 2 + a füzér karaktereinek száma
   1.: típusjelző: 80H
   2.: hossz: a szöveg karaktereinek száma
   3 – (n + 2).: a karakterek (idézőjelek nélkül)

6. BASIC sorszám (A0H típuskód)
   Elem hossza: 3 bájt
   1.: típusjelző: A2H (A0H + 2)
   2.: LO: a sorszám alsó bájtja
   3.: HI: a sorszám felső bájtja

7. Numerikus konstans (C0H típuskód)
   Elem hossza 1 + 2 vagy 1 + 5
   1.: típusbájt: C2H, ha a konstans egész típusú (0 és 9999 között) C6H, ha a konstans lebegőpontos szám
   Egész típus esetén a további bájtok
   2.:LO: a konstans alsó bájtja
   3.:HI: a konstans felső bájtja
   lebegőpontos konstansnál:
   2.—6.: a konstans számjegyei BCD alakban
   7.: a szokásos kitevőbájt

Meg kell említeni, hogy a negatív előjelet külön jelként tárolja az értelmező (0DH típusjelző) és a következő konstanst pozitív értékként írja a kódolt szövegbe.

Felmerülhet a kérdés, hogy mi alapján különbözteti meg az értelmező az A0H és C0H típusú konstansokat a szövegben, mi a helyzet a REM-sorok kódolásával stb. Hogy ezekre, és hasonló kérdésekre válaszolni tudjunk, tekintsük át röviden egy programsor elemzésének folyamatát: azokat a lépéseket, amelyek az ENTER megnyomásától a tokenizált sor memóriába írásáig történnek.

Azt már tudjuk, hogy a beírt sor karakterei a 0248H pufferbe kerülnek. A vizsgálatkor, elemzéskor néhány pointer jelzi az aktuális pozíciót (ezek majd a végrehajtás közben is hasonlóan funkcionálnak):

— 0216/7H: a sor elejére mutat
— 0214/5H: a sor éppen feldolgozás alatt álló elemére mutat
— 0347/8H: a feldolgozott változónév (karaktersorozat) elejére mutat.

Ezek után nézzük a lépéseket (megadva a fontosabb rutinok címeit is, ezzel is segítve a ROM-ot elemezgető olvasó tájékozódását):

— Az értelmező a sorszámmal kapcsolatos bevezető elemzések után — megvizsgálja a sor első elemének típusát (C5A0H). Hibát vált ki, ha ez nem betűvel kezdődő karaktersorozat (formailag változónév, valójában kulcsszó).

— Összehasonlítja az első szót a BASIC kulcsszavakkal (EF48H). Ennek alapja

a kulcsszótábla (3.2.5 pont), amelynek pointerei a kulcsszavak alakját is tartalmazó paraméter blokkokra mutatnak. Azonosítás esetén kiolvassa a kulcsszó típusbájtját (B). C-ben az azonosított kulcsszó sorszáma (tokenje) van. Ha a tábla végéig nem találta meg a nevet, a LET tokenjét (28H) és típusbájtját állítja be.

Ellenőrzi (BIT 1,B), hogy az adott kulcsszó alkalmazható-e programban. A karaktersorozat helyére a kétbájtos kulcsszóelemet írja a sorba (ehhez szükség szerint csúsztatja el a hátralévő szövegrészt). Eddig minden sornál azonos a feldolgozás. A továbblépések viszont már erősen függenek az azonosított kulcsszótól.

— A típusbájt alapján megvizsgálja, hogy kell-e tokenizálni a sor kulcsszót követő szöveget (BIT 6,B). Ha igen, akkor ezt megteszi (C41DH), azaz a szöveg elemei helyett azok kódolt alakját írja a pufferbe.

— A következő lépés az általános tokenizálási eljárás utáni speciális kódolásra vagy a kulcsszónak megfelelő szintaktikai elemzésre ad lehetőséget. Az értelmező ui. meghívja (EFE0H) a kulcsszó 2. rutinját. A BASIC utasítások jelentős részénél ez nem lát el különösebb funkciót, néhány esetben azonban fontos feladata van. A RUN, GOTO és GOSUB 2. rutinja pl. az utasítást követő sorszám típusát ellenőrzi és megfelelő érték esetén (egész típus) a már tokenizált szövegben kijavítja a típusjelző bájtot C2H-ról A2H-re (íme itt a válasz a felmerült kérdésre!). Hasonlóan fontos a 2. rutin olyankor is, amikor formailag azonos nevekhez más tokeneket, végrehajtó rutinokat akar rendelni az értelmező (DEF, IF, END...). Például a DEF esetén a 2. rutin megvizsgálja, hogy egysoros, END nélküli definícióról van-e szó (ekkor meghagyja a tokenizáláskor beírt kódot) vagy DEF...END blokkról (ekkor a 0DH DEF-tokent kicseréli 0EH-ra). Az utasítások másik csoportjánál a rutin végigszalad a hátralévő részen és egy durva szintaktikai elemzést végez (pl. zárójelek), de ez már nem hívja meg pl. a függvények kiértékelő rutinjait, így azok szintaktikai szabályai a beíráskor nem érvényesülhetnek.

— Ezután az értelmező, ha a vizsgált kulcsszó megenged több utasítást a sorban (BIT 4,B) és ":" következik saját rekurzív hívásával folytatja a sorelemzést. Egyébként viszont a kulcsszóhoz tartozó szöveg után sorvéget vár.

Ezzel befejeződött a sor elemzése, tokenizálása. Így írja be az értelmező a programba az új sort (kiegészítve a bevezető hossz, sorszám, skatulya és lezáró 0 bájtokkal; l. 3.2.7 pont).

Ezek után már jól követhető, hogyan tokenizálódnak a fejezet elején példaként felírt utasítások. Hexadecimális kódokkal:

```
PRINT 12

 60   :token következik
 38   :PRINT token
 C2   :numerikus konstans következik (egész)
 0C   :LO=12 dec
 00   :HI=0   (konst.=12+256*0=12)
 00   :vége
```

A többi példa esetén (most már minimális kommentárral):

```
LET SZAM=SIN(PI)+0.5

 60 28 - 24 53 5A 41 4D - 13 - 23 53 49 4E -
   LET    4:  S  Z  A  M     =    3:  S  I  N

  - 08 - 22 50 49 - 09 - 0B - C6 00 00 00 00 50 63 - 00
     (    2:  P  I     )     +    6: 5.000000000   E-1


LET A1="NEV"

 60 28 - 42 41 24 - 13 - 80 03 4E 45 56 - 00
   LET    2:  A  $     =       3:  N  E  V


GOTO 100

 60 21 - A2 64 00 - 00
  GOTO    2: LO HI
```

### 3.4.3 Végrehajtás

A sorkódolást bemutató rövid kitérő után térjünk vissza a beolvasási főághoz; ahhoz a ponthoz, ahol kiderült: a beírt és pufferbe töltött sor első eleme nem sorszám, tehát a sor parancsmódban végrehajtandó. Ebben a fázisban a 0249H pufferbe töltött szöveg még nincs tokenizálva, csak az első elem típusát vizsgálta meg az értelmező.

A parancsmódú vágrehajtásnak is a kulcsszó felismeréssel kell kezdődnie, a következő lépésben pedig a sor hátralévő részét (a kulcsszótól függően) tokenizálni kell, hiszen a végrehajtó rutinok csak a kódolt paramétereket ismerik fel:

```
C37C  CD 48 EF     CALL EF48        ;kulcsszó azonosítás
                                     C:token, (HL):tipus
C37F  28 32        JR    Z,C3B3     ;LET-re, ha nem kulcs-
                                     szó az első elem
C381  CB 46        BIT   0,(HL)     ;parancsmódban jó?
C383  CA 4C F5     JP    Z,F54C     ;hibára, ha nem
C386  ED 5B 14 02  LD    DE,(0214)  ;aktuális elem címe
C38A  D5           PUSH  DE         ;regiszterek
C38B  E5           PUSH  HL         ;  verembe
C38C  C5           PUSH  BC         ;
C38D  CB 76        BIT   6,(HL)     ;kell tokenizálni?
C38F  C4 1D C4     CALL  NZ,C41D    ;kódol, ha kell
C392  C1           POP   BC         ;token vissza
```

Ezután az értelmező ugyanúgy meghívja az azonosított kulcsszó 2. rutinját, mint ahogy a programsor tokenizálásánál láttunk. A kulcsszó szintaktikájának megfelelő formai ellenőrzés, ill. az esetleges áttokenizálás után (pl. a RUN 2. rutinja írja át a kulcsszót követő sorszám típusbájtját A2H-re) kezdődhet a végrehajtás. A rutin

szükség esetén (pl. START, DELETE stb.) törli a változókat, 1—99. csatornákat, majd a sor következő elemének vizsgálata (BASIC 20H) után meghívja az azonosított kulcsszó 1. rutinját a végrehajtáshoz:

```
C39D  C5           PUSH BC          ;token verembe
C39E  CD E0 EF     CALL EFE0        ;2. rutin hívása
C3A1  C1           POP  BC          ;token vissza
C3A2  E1           POP  HL          ;típusbájt címe
C3A3  CB 6E        BIT  5,(HL)      ;kell CLEAR?
C3A5  C4 4D C5     CALL NZ,C54D     ;töröl, ha kell
C3A8  E1           POP  HL          ;aktuális elem címe
C3A9  22 14 02     LD   (0214),HL   ;vissza a pointerbe
C3AC  CD 59 F2     CALL F259        ;következő elem vissz.
C3AF  DD 36 09 00  LD   (IX+09),00  ;aktuális csatorna #0
C3B3  CD E7 EF     CALL EFE7        ;1. rutin: végrehajtás
```

Az utasítások végrehajtó rutinjait természetesen itt is a kulcsszótábla és a token alapján keresi meg az értelmező. A tábla manipulálása (megváltoztatása, bővítése) tehát hatással lesz a leírt végrehajtási folyamatra.

A parancsmódú végrehajtás befejezéseként az értelmező ellenőrzi a végrehajtó rutinok által beállított 0206H rendszerváltozót: a folytatás módjának jelzőbájtját. Ennek a programfutás során van jelentősége, parancsmódban a gép gyakorlatilag nem kezel többutasításos sorokat. Utolsó lépésként egy jelzőbitet kérdez le a rendszer:

```
C3B6  3A 06 02     LD   A,(0206)    ;folytatás-kód?
C3B9  FE 02        CP   02          ;0 vagy 1?
C3BB  18 10        JR   C3CD        ;

C3BD  DC 51 F2     CALL  C,F251     ;sor véget var
C3C0  DD CB 00 6E  BIT  5,(IX)      ;flag törlődött?
C3C4  C2 F2 C2     JP   NZ,C2F2     ;vissza a beolvasási
                                     főágra, ha nem

C3C7  DD CB 00 CE  SET  1,(IX)      ;programmód
C3CB  DD CB 00 EE  SET  5,(IX)      ;flag alap
C3CF  C9           RET              ;veremcímre ugrik
```

A vizsgált bit általában 1, így a végrehajtás befejezésével a vezérlés ismét a beolvasási főágra kerül. Ha azonban egy kulcsszó végrehajtó rutinja törli ezt a bitet, lehetséges a főág szubrutinként való hívása vagy a parancsmódban beírt kulcsszó végrehajtó rutinjából a verem tetején lévő címre ugrás (CONTINUE).

Ezzel áttekintettük a parancsmódú végrehajtás lépéseit. Valójában a számítógép legfontosabb tulajdonsága a progamfuttatás lehetősége. A progammódú végrehajtás menete némileg különbözik az eddig látottaktól.

Az alapot itt a már tokenizált, formailag ellenőrzött programszöveg adja. Nem lesz tehát szükség a 2. rutinok hívására. A másik lényeges különbség, hogy itt az utasítások, sorok szigorú tárolási szabályok szerint egymás után következő sorozatát kell feldolgozni. Az értelmezőnek csak ezen a láncon kell végighaladnia, sorra

hívja a talált tokenekhez tartozó végrehajtó rutinokat mindaddig, amíg a lánc végére nem ér (vagy az egyik rutin meg nem szakítja a végrehajtást). A ciklikus hívássorozatot végző rutin (CD68H, ide lép pl. a RUN végrehajtás is az esetleges paraméterek kiértékelése után)

— ellenőrzi a STOP-bitet (BIT 2,(IX)). Ha egy megszakítás vagy EXOS hívás után 1-be íródott ez a bit, elugrik a kezelésére;

— TRACE üzemmódban (BIT 4,(IX)) kiírja a kijelölt csatornára az éppen feldolgozott sor számát;

— beolvassa a kulcsszó tokenjét, majd megvizsgálja a következő elem típusát (BASIC 20H);

0206H = 0 jelzőbájtot állít,

— meghívja a kulcsszó (1.) végrehajtó rutinját. A végrehajtás után a kulcsszó rutinja által megfelelően beállított 0206H jelzőbájt alapján dönt a további tennivalókról:

— ha (0206H) = 0, a következő elemet vizsgálja, ":" esetén az adott sorban folytatja a végrehajtást,

— ha (0206H) = 1, sorvégnek kell következnie, különben hibát jelez,

— ha (0206H) = 2, nem veszi figyelembe a sor hátralévő részét, a pointereket (0216H, 0214H) a következő sor elejére állítva elölről kezdi a ciklust,

— ha (0206H) = 3, a pointereket a programszövegbe írt konstans (GOTO sorszám) szerinti BASIC sor elejére állítja és onnan folytatja a ciklus végrehajtást,

— ha (0206H) > 3, veremrendezés után RET-tel fejezi be a végrehajtást (programként meghívott DEF rutin végén itt tér vissza a hívóhoz.

Ha addig egy utasítás (STOP, END) ki nem váltja a megszakítást, a fenti ciklikus végrehajtás a progam végén szakad meg automatikusan (ha a következő sor elején 0 sorhosszúságot talál az értelmező). Ekkor a vezérlés néhány lépés után visszakerül a beolvasási főágba és újra kezdődhet a megismert működési folyamat.

## 3.4.4 Utasítások, függvények

Az előbbiekben végigkövettük, hogyan jut el az értelmező a kulcsszavakhoz tartozó végrehajtó rutinok meghívásáig. Nézzük meg most néhány, röviden leírható, konkrét esetben, hogy milyenek is ezek a rutinok.

Első példánk mintegy bemelegítésként egy olyan egyszerű utasítás legyen, amely semmilyen paramétert nem vár szövegkörnyezetétől és nem is ad vissza értéket. A PING utasítás egyetlen funkciója egy 7-es kód (ASCII BELL kód) írása a hangcsatornára:

```
D3BF  3E 07       LD    A,07        :SOUND csatorna
D3C1  06 07       LD    B,07        :irandó kód
D3C3  F7 07       EXOS  07          :kiírás
D3C5  C9          RET               :
```

Ez a rutin egyedülállóan egyszerű. A legtöbb utasítás paramétereket vár a működéséhez (vagy ad vissza). Ennek illusztrációjaként kövessük a POKE CIM, ADAT utasítás végrehajtását. Az egyébként igen egyszerű funkciót mindössze az bonyolítja kissé, hogy az értelmező az elfogadott paramétertartományban egységesen fölfelé kerekít:

```
D3C6  CD CB D3     CALL D3CB        ;paraméter kiértékelés
                                     HL=CIM, A=ADAT
D3C9  77           LD   (HL),A      ;végrehajtás
D3CA  C9           RET              ;

D3CB  CD E2 DE     CALL DEE2        ;BASIC 23H: CIM->MREG1
D3CE  21 00 80     LD   HL,8000     ;32768
D3D1  E5           PUSH HL          ;verembe
D3D2  CD EF E8     CALL E8EF        ;BASIC 02H: HL->MREG2
D3D5  CD F8 E4     CALL E4F8        ;BASIC 0DH: CIM-32768
D3D8  CD F9 E7     CALL E7F9        ;BASIC 0BH: MREG->HL
D3DB  D1           POP  DE          ;DE=32768
D3DC  19           ADD  HL,DE       ;CIM  0 és 65535 közé
D3DD  3E 0C        LD   A,0C        ;"," követi?
D3DF  CD 53 F2     CALL F253        ;ha nem, hiba
D3E2  C3 49 E8     JP   E849        ;2. paraméter kiér-
                                     tékelésére ugrik

E849  37           SCF              ;
E84A  E5           PUSH HL          ;CIM verembe
E84B  CD FB E7     CALL E7FB        ;BASIC 0BH: 2. param.
                                        ADAT->MREG->HL
E84E  24           INC  H           ;H=255?
E84F  28 04        JR   Z,E855      ;
E851  25           DEC  H           ;H=0?
E852  C4 BB F4     CALL NZ,F4BB     ;hiba, ha az ADAT>255
                                        vagy ADAT<-255
E855  7D           LD   A,L         ;A: ADAT
E856  E1           POP  HL          ;HL: CIM
E857  C9           RET              ;vissza a POKE-hoz
```

A programszövegben következő paraméterek kiértékelése, a megkívánt szintaxis (",") ellenőrzése tehát a végrehajtó rutin feladata. A meghívott aritmetikai rutin (BASIC 23H) viszont már önállóan kiértékeli az aritmetikai egységet, beleértve a függvényhívásokat is.

A BASIC egyik alapfunkciója az értékadás. Nézzük meg ezt a LET végrehajtásában. Mivel a többszörös értékadás (pl. LET A,B = 1) kissé komplikálja a futást, talán nem árt összefoglalni az értékadás vázát:

— a változó tárolási címének megkeresése (a változó létrehozása, ha még nem volt)

— " = " következik?

— a következő szakasz kiértékelése a változó típusának (numerikus vagy füzér) megfelelően

— az érték betöltése (a munkaregiszterből) a változó adatterületére.

Itt a többszörös értékadáshoz az értelmező mindaddig tárolja a Z80-as veremben (PUSH utasítással) az újabb és újabb változók tárolási címeit, amíg a sor végére (az " = " jelhez) nem ér. A beírandó kifejezés kiértékelése után az értéket sorra áttölti a veremből visszaolvasott tárolási címekre (itt a sor végét a PUSH utasítással először tárolt 0000 "előbukkanása" jelzi). Kövessük a végrehajtást pl. numerikus értékadás esetén:

```
D280  11 00 00     LD   DE,0000     ;sorozatvég jelző
D283  D5           PUSH DE          ;verembe
D284  15           DEC  D           ;D=FFH (első elem)
D285  CC 59 F2     CALL  Z,F259     ;BASIC 20H: típus?
D288  3A 02 02     LD   A,(0202)    ;elemtípus
D28B  BA           CP   D           ;=előzőek típusa?
D28C  28 0E        JR    Z,D29C     ;tovább, ha igen
D28E  14           INC  D           ;első változó?
D28F  C2 1C F5     JP   NZ,F51C     ;hiba, ha változott
                                      a típus
D292  57           LD   D,A         ;D=elemtípus
D293  FE 20        CP   20          ;numerikus változó?
D295  28 05        JR    Z,D29C     ;tovább, ha igen
D297  FE 40        CP   40          ;füzérváltozó?
D299  C2 1C F5     JP   NZ,F51C     ;hiba, ha egyik sem
D29C  D5           PUSH DE          ;típusjelző verembe
D29D  CD 7B EC     CALL EC7B        ;változó tárolási címe
D2A0  D1           POP  DE          ;
D2A1  E5           PUSH HL          ;tárolási cím verembe
D2A2  3A 04 02     LD   A,(0204)    ;a következő elem
D2A5  FE 0C        CP   0C          ;","? (újabb változó)
D2A7  28 DC        JR    Z,D285     ;ciklus elejére, ha
                                      újabb változó köv.
D2A9  3E 13        LD   A,13        ;"=" követi?
D2AB  CD 53 F2     CALL F253        ;hiba, ha nem
D2AE  7A           LD   A,D         ;típusjelző
D2AF  FE 20        CP   20          ;numerikus?
D2B1  20 0E        JR   NZ,D2C1     ;ha nem, füzérre
D2B3  CD E2 DE     CALL DEE2        ;BASIC 23H
                                      kiértékelés
D2B6  E1           POP  HL          ;HL= tárolási cím
D2B7  7C           LD   A,H         ;
D2B8  B5           OR   L           ;HL=0? (sorozat vége)
D2B9  CA E8 ED     JP    Z,EDE8     ;veremrendezésre, ha
                                      nincs több változó
D2BC  CD 0E E9     CALL E90E        ;kb. BASIC 1BH
                                      érték a változóba
D2BF  18 F5        JR   D2B6        ;ciklus elejére
```

Figyelemre méltó, hogy míg a belépéskor az elem típusvizsgálata után került a vezérlés a rutinra, a további elemek (változónevek) típusvizsgálatát már a végrehajtó rutinnak kell kérnie (BASIC 20H). A BASIC 1BH hívás az értéknek a változó adatterületére való áttöltése után törölné is az elemet a BASIC veremből. Ezért alkalmazza itt az értelmező a köztes belépést (E90EH), így az érték a sorozat végéig a munkaregiszterben marad.

A BASIC utasítások másik csoportja a végrehajtási sorrendet módosítja. Ezt az IS-BASIC-ben igen egyszerűen meg lehet tenni. A feladat lényegében az elemzési pozíciót meghatározó pointer bállítása a következő végrehajtandó sorra. Jó példa erre a GOSUB végrehajtó rutinja.

Itt egyben a BASIC verem használatára is látunk egy példát. A rutin a GOSUB sort követő BASIC sor címét írja a BASIC verembe 6-os típuskóddal, a célcímet pedig a (0216H) pointerbe írja. A tényleges vezérlésátadás már a programvégrehajtási főág feladata:

```
D1B5  11 04 00     LD   DE,0004     ;blokkméret+1
D1B8  CD 97 F0     CALL F097        ;van még hely?
D1BB  CD FA C3     CALL C3FA        ;BASIC 2EH: sorszám
                                     keresés a programban
D1BE  DC 25 F5     CALL  C,F525     ;hiba, ha nincs ilyen
D1C1  E5           PUSH HL          ;sor címe verembe
D1C2  2A 16 02     LD   HL,(0216)   ;GOSUB sor címe
D1C5  4E           LD   C,(HL)      ;GOSUB sor hossza
D1C6  06 00        LD   B,00        ;BC=sorhossz
D1C8  09           ADD  HL,BC       ;HL: a köv. sor címe
D1C9  EB           EX   DE,HL       ;DE=  -""-
D1CA  2A 28 02     LD   HL,(0228)   ;BASIC verem címe
D1CD  CD 06 DD     CALL DD06        ;elé irja DE-t
                                     (visszatérési cím)
D1D0  36 06        LD   (HL),06     ;elé GOSUB tipusbájt
D1D2  E1           POP  HL          ;célcím
D1D3  22 16 02     LD   (0216),HL   ;pointerbe
D1D6  DD 36 06 03  LD   (IX+06),03  ;folytatás kódja: a
                                     (0216) soron folytat
D1DA  C9           RET              ;
```

Hogy zárjuk a kört, nézzük meg az utasítás párját is. A RETURN rutinjának fordított feladata van: a BASIC veremből kiolvasott visszatérési címet kell hasonló módon pointerbe töltenie. A probléma mindössze az, hogy — mivel a RETURN is a végrehajtási sorrendet változtatja meg — előfordulhat, hogy egy megkezdett blokkból (pl. FOR-NEXT ciklusból) lépünk ki. Ezért a rutin visszaugrás előtt rendezi a BASIC vermet: törli a GOSUB blokk elé azóta beírt, nála kisebb típuskódú (1—5) elemeket. Így áll végül is a BASIC veremmutató (0228H) a GOSUB blokkra. Nem lehet viszont kilépni DEF rutinból vagy WHEN blokkból.

```
D82B  CD 51 F2     CALL F251        ;sorvéget vár
D82E  CD F3 ED     CALL EDF3        ;BASIC verem rendezés
D831  38 FB        JR    C,D82E     ;ciklikusan
D833  FE 06        CP   06          ;GOSUB blokk?
D835  C2 14 F5     JP   NZ,F514     ;hiba, ha nem
D838  CD ED DC     CALL DCED        ;blokk visszaolvasás
                                     HL: visszatérési cím
D83B  C3 D6 D1     JP   D1D6        ;pointerbe, folytatás
                                     a GOSUB utáni soron
```

Az értékadó és vezérlésátadó utasítástípusok után praktikus lenne a kiíratással is foglalkozni. A PRINT utasítás elemzése azonban meglehetősen helyigényes és

nehezen áttekinthető lenne, nem annyira az alapfunkció, mint inkább az IS-BASIC által megengedett sokféle paraméter, kiegészítés miatt (pl. AT, TAB, USING, vessző, pontosvessző stb.). A gépi kódú programjainkban is gyakran igényelt alapfeladat (numerikus érték vagy füzér kiíratása az aktuális csatornára) a BASIC funkcióhívások segítségével általában egyszerűen megoldható. Numerikus kifejezésre:

```
BASIC 23H     ; kiértékelés
BASIC 3FH     ; kiíratás
```

míg füzér kifejezés esetén:

BASIC 24H a kiértékelésre és
BASIC 3EH a kiíratásra.

A BASIC veremtől, programszövegtől független konkrét kiíratási feladatokat is több hívás segíti (pl. BASIC 38H, 39H).

A kifejezések kiértékelésének van egy olyan momentuma, amely minket is érdekelhet, ha függvénybővítéseket is szeretnénk készíteni. Nem láttuk még a beépített változók, függvények kiértékelő rutinjainak működését, ellátandó feladataikat.

Az alapalgoritmus jól követhető egy olyan egyszerű rutinon, mint pl. a PI kiértékelése. Mint a függvénytáblából megállapítható (l. 7. függelék), a rutin az 1. szegmens CA0BH címén kezdődik:

```
CA0B  D7          RST  10          ;funkcióhívás indul
CA0C  92 04       BASIC 92,04      ;PI -> MREG
CA0E  00          BASIC 00         ;funkcióhívás vége
CA0F  C9          RET              :
```

A rutin feladata röviden megadható: a kiértékelés eredményét a BASIC verembe kell tölteni (MREG). A feladat nyilván komplikáltabb olyan rutin esetén, amely paramétert, argumentumot is vár a kiértékeléshez, de a végeredmény akkor is ugyanez: az eredmény a MREG-ben. Nézzük meg pl. a PEEK (CIM) függvény kiértékelését (már csak a POKE utasítással való analógia miatt is):

```
C9CE  CD E4 C2    CALL C2E4        ;BASIC 99  HL=CIM
                                    argumentum kiért.
C9D1  CD DB C9    CALL C9DB        ;CIM kerekítés
C9D4  6E          LD   L,(HL)      ;L=PEEK(HL)
C9D5  26 00       LD   H,00        ;H=0
C9D7  D7          RST  10          ;funkcióhívás indul
C9D8  02          BASIC 02         ;HL -> MREG
C9D9  00          BASIC 00         ;hívás vége
C9DA  C9          RET              :
```

A címadat transzformációjához (kerekítéséhez) meghívott rutin ugyanazt az algoritmust követi, mint a POKE esetén.

A megszakításokkal foglalkozó fejezetben utalunk a TIME$ kiértékelésre. Nézzük most ezt meg:

```
C530  F7 20       EXOS 20          ;ora lekérdezése
C532  2A 28 02    LD   HL,(0228)   ;BASIC veremmutató
C535  7B          LD   A,E         ;A=sec
C536  CD 6F C5    CALL C56F        ;"ss" -> MREG
C539  7A          LD   A,D         ;A=min
C53A  CD 6C C5    CALL C56C        ;"mm:" -> MREG
C53D  79          LD   A,C         ;A=óra (h)
C53E  CD 6C C5    CALL C56C        ;"hh:" -> MREG
C541  18 1F       JR   C562        ;

C562  2B          DEC  HL          ;füzér elé
C563  36 08       LD   (HL),08     ;hosszbájt
C565  2B          DEC  HL          ;elé
C566  36 03       LD   (HL),03     ;füzér tipus
C568  22 28 02    LD   (0228),HL   ;mutató új érték
C56B  C9          RET              ;MREG="hh:mm:ss"
```

Füzér értékű függvény esetén is a MREG-ben hagyott érték a rutin futásának eredménye. Innen lehet kiírni vagy tovább feldolgozni.

A lebegőpontos aritmetika legszebb példáit a trigonometrikus függvények kiértékelő rutinjaiban láthatjuk. Kövessük az egyik legegyszerűbb, a tangensmeghatározó TAN rutin futását; azt a módot, ahogy gyakorlatilag egyetlen BASIC funkcióhívás sorral eljut az argumentumtól a végeredményig:

```
CC9B  D7          RST  10          ;funkcióhivás indul
CC9C  99          BASIC 99         ;argumentum -> MREG
CC9D  A9          BASIC A9         ;radiánra, ha kell
CC9E  00          BASIC 00         ;hívás vége

CC9F  D7          RST  10          ;funkcióhivás indul
CCA0  15          BASIC 15         ;MREG2=MREG1=x
CCA1  98          BASIC 98         ;MREG2 -> REG2
CCA2  A4          BASIC A4         ;MREG1=SIN(MREG1)=
                                       =SIN(x)
CCA3  95          BASIC 95         ;REG2 -> MREG2 (x)
CCA4  92          BASIC 92,07      ;MREG3=PI/2
CCA6  0C          BASIC 0C         ;MREG2=MREG2+MREG3
                                       =x+PI/2
CCA7  A4          BASIC A4         ;MREG2=SIN(MREG2)=
                                     =SIN(x+PI/2)=COS(x)
CCA8  0F          BASIC 0F         ;MREG1=MREG1/MREG2
CCA9  00          BASIC 00         ;hívás vége
CCAA  C9          RET              ;MREG=sin(x)/cos(x)
```

A rutin tényleges kiértékelő része a BASIC A6H funkcióhívás.

Befejezésül vizsgáljuk meg a gépi kódú progamozásban különös jelentőségű USR funkciót. Az ezzel hívott gépi rutinban esetenként pontosan ismerni kell a hívás körülményeit (LAP-konfiguráció, veremállapot stb.). Mivel a USR kiértékelő rutinja az 1. szegmensen van (CD3FH), az értelmező ezt lapozza a 3. LAP-ra a

végrehajtáskor. A felhasználói rutinba való kilépéskor tehát két biztos pontunk van a memóriakonfigurációban: a 3. LAP-on az 1. szegmens, míg a 0. LAP-on, mint mindig, a nulláslapszegmens (esetünkben az F8H). Az 1. LAP határozatlan (hacsak nem ott fut a BASIC programunk), a 2. LAP-on általában az FFH rendszerszegmens van (de ide is benyúlhat a BASIC).

Maga a rutin két szempontból is eltér az eddigiektől. Egyrészt két argumentuma is van (USR(CIM,ADAT)), így ennek kiértékelésére is látunk példát. Sokkal lényegesebb a másik specialitás: a rutin a felhasználói progam hívása előtt a verembe tölt egy visszatérési címet (CAECH). Így a felhasználói rutin RET utasítása nem a főágnak adja vissza a vezérlést, csak erre a címre lép. Ez igen fontos a paramétervisszaadás szempontjából, ui. akármilyen értéket helyez is rutinunk a munkaregiszterbe (kiíratáshoz vagy tovább feldolgozásra), ez a befejező szakasz rátölti a HL regiszterpár aktuális értékét. Visszatéréskor tehát ez a BASIC verem aktuális eleme, ezt írja ki pl. a PRINT USR(CIM,ADAT) utasítás. Ha mi saját adatunkat akarjuk visszaadni az értelmezőnek, ezt a befejezést el kell hagynunk, azaz a PUSH utasítással tárolt (el-PUSH-olt) visszatérési címet le kell emelnünk a veremből (POP). Ügyelni kell azonban arra, hogy ezzel mi vesszük magunkra a BASIC verem kezelésének felelősségét.

Ezek után kövessük a végrehajtást, ami egyébként magának a felhasználói rutinnak a hívására is a PUSH CIM:...:RET utasításpárt használja:

```
CD3F  21 EC CA    LD   HL,CAEC     ;visszatérési cím
CD42  E5          PUSH HL          ;verembe
CD43  D7          RST  10          ;funkcióhívás indul
CD44  9A          BASIC 9A         ;"(CIM" kiértékelése
CD45  0B          BASIC 0B         ;érték -> HL
CD46  00          BASIC 00         ;hívás vége
CD47  E5          PUSH HL          ;CIM verembe
CD48  D7          RST  10          ;funkcióhívás indul
CD49  9B          BASIC 9B         ;",ADAT" kiértékelése
CD4A  A0          BASIC A0         ;")" jelet vár
CD4B  0B          BASIC 0B         ;HL=ADAT
CD4C  00          BASIC 00         ;hívás vége
CD4D  C9          RET              ;elugrik a CIM-re

CAEC  D7          RST  10          ;funkcióhívás indul
CAED  02          BASIC 02         ;HL -> MREG
CAEE  00          BASIC 00         ;hívás vége
CAEF  C9          RET              ;
```

Látható, hogy a rutin nem védi (nem őrzi meg) a regisztereket (ezért pl. az IX megváltoztatása a BASIC összeomlásához vezethet), sem a LAP-konfigurációt, így a 3. LAP-on meg kell őriznünk az 1. szegmenst (vagy vissza kell lapoznunk visszatérés előtt).

# 4. Bővítési, kiterjesztési lehetőségek

Az ENTERPRISE operációs rendszerének kiterjesztésére számos példát láttunk a 2. fejezetben (ilyenek voltak pl. a megszakításkezelő rutinok, perifériakezelők, és természetesen az általános célú rendszerbővítő is). Ebben a fejezetben a BASIC rendszer bővítési lehetőségeit tárgyaljuk.

A két terület nem mindig válik élesen szét, hiszen pl. az EXOS rendszerbővítő is kiterjesztheti a BASIC szolgáltatásainak körét (jó példa erre a német verzió ":VDUMP" funkciója (képernyőmásolás) vagy a német nyelvű hibaüzenetek kiírása). Mivel a rendszerbővítőkről már volt szó a 2.6 alfejezetben, itt már csak a BASIC-en belüli bővítésekkel foglalkozunk.

Új utasításokat, függvényeket a gép komfortos BASIC nyelvén is kialakíthatunk (DEF rutinok), de ezek működési sebessége nyilván lényegesen elmarad a BASIC alapkészletben kínált beépített kulcsszavak és függvények végrehajtásától, kiértékelésétől. Egyes esetekben pedig (pl. parancsmódban használható GOTO) feltétlenül gépi kódú rutinra van szükség. A továbbiakban tehát a gépi kódban megírt, a BASIC rendszer lehetőségeit valamilyen irányban kiterjesztő, szolgáltatásait bővítő rutinok rendszerbe illesztéséről lesz szó. Maguk a konkrét funkciók elsősorban példaként szolgálnak, hiszen a bővítés irányát nyilván mindenkinél az igények szabják meg.

A gépi rutinok rendszerbe illesztésére, az értelmezőbe való belenyúlásra három — lényegesen különböző — módot tárgyalunk.

A legkevesebb változtatást, előkészítést a gépi kódú rutinok szokásos módon való beépítése (CODE vagy POKE) és hívása (CALL USR()) igényli. Nem magától értetődő azonban az utasításként vagy függvényként való alkalmazás különbsége, paraméterek átadása, visszaadása stb.

A másik megoldás a beépített utasítások és függvények nyilvántartásához kapcsolódik. A kulcsszótábla és a függvénytábla manipulálása (a végrehajtó rutinok megváltoztatása, kiegészítése, a nevek megváltoztatása, új funkciók hozzáfűzése) jelenti a BASIC bővítések valódi lehetőségét.

Harmadik módszerként — korántsem rendeltetésszerűen, de igen széles lehetőségekhez jutva — az egész értelmezőt RAM-ba helyezhetjük és ott tetszés szerint alakíthatjuk át.

# 4.1 USR rutinok

A BASIC programban felépített és szokásos módon hívott USR rutinok bővítésként való alkalmazására már láttunk példát (3.2.8 pont). Az ilyen típusú felhasználásnak két kritikus pontja van: a rutin tárolási területe és a paraméterkezelés.

A szokásos módon (ALLOCATE) lefoglalt területen tárolt gépi rutin jól használható a programban, de könnyen zavarba jöhetünk, ha parancsmódban, esetleg progamszerkesztés közben is szeretnénk élni a szolgáltatással. Ekkor ui. a program visszacsúszhat a lefoglalt területre, felülírva ezzel a gépi rutint, ami a rendszer összeomlásához vezethet a következő hozzáforduláskor. Az ilyen célra szánt rutint tehát a BASIC programtól független, lehetőleg védett területen kell elhelyezni.

A konkrét címre való betöltést a CODE utasítás báziscímének (021C/DH) beállításával érhetjük el:

POKE 540,LO
POKE 541,HI
ahol CIM = 256·HI+LO

A védettséget egyszerűbb esetben, ideiglenes megoldásként a BASIC mindkét végétől kellő távolságra lévő cím (pl. 3000H) választásával érhetjük el, valódi megoldást azonban a hely lefoglalása jelent. A lehetőségeket az korlátozza, hogy a rutinnak egy mindig belapozott szegmensen kell lennie. Ez gyakorlatilag csak a nulláslapra teljesül. Néhány tucat szabad bájtot a rendszerterület elején is találhatunk (0180H—0200H), de praktikusabb a BASIC munkaterület tetszés szerinti feltolása (erre láttunk példát a 3.1 alfejezet végén).

A paraméterkezelés nem jelent gondot, ha a rutin nem igényel (és nem ad vissza) más adatot, mint az értelmező által kezelt kétbájtos (előjeles egész) értéket (a HL-be töltött, majd futás után onnan kiolvasott adatot).

A más jellegű vagy további paraméterek bevitele a BASIC szintaktikának megfelelően folytatott progamszövegből lehetséges.

Például CALL USR(CIM,HL)x,y

A formailag kissé erőltetett, de hatékonyságában a bővítésekkel egyenértékű megoldás lehetőségét az adja, hogy a USR kiértékelés csak a befejező zárójelig ellenőrzi és értékeli ki a programszöveget (így nem veszi észre az egyébként szintaktikusan hibás szöveget), majd a következő elem típusvizsgálata (BASIC 20H) után lép a felhasználói rutinba. Ebben a rutinban a tervezett feldolgozás szerint ellenőrizhetjük és értékelhetjük ki a paramétermezőt és formai szempontból ismét helyes aktuális pozícióban (a sor végén) adjuk vissza a vezérlést az értelmezőnek.

BASIC utasításnak szánt rutinunkban akkor járunk el helyesen, ha a BASIC vermet alaphelyzetben adjuk vissza a rendszernek. Ha azonban függvénybővítésként

funkcionál a rutin, a futás eredménye a függvényérték. Ennek visszatéréskor a BASIC veremben kell maradnia. A 3.4.4 pontban (a USR kiértékelés elemzésekor) azt is bemutattuk, hogy miként lehet megakadályozni a USR befejező szakaszt, hogy HL értékét aktuális elemként a BASIC verembe tegye (egy pár nélküli POP utasítás volt a megoldás). Így a kiértékelés eredménye marad az aktuális MREG-ben, ezt írja ki (PRINT USR...) vagy dolgozza fel (A = B · USR...) az értelmező.

Ez a mód érezhető nehézkessége miatt elsősorban egy programon belüli ideiglenes célfeladatra, ill. az egyéb megoldások tesztelésére alkalmas. Az alkalmazásról eddig elmondottak szemléltetésére nézzünk néhány (függvény, ill. utasítás típusú) példát.

LINEPRT n (line-pointer)

Az n. BASIC sor tárolási címét adja meg. A megoldás a legegyszerűbb alkalmazásra mutat példát, hiszen a BASIC sorszám (0—9999) a USR természetes változójaként megadható és a kiértékelés eredménye is kétbájtos egész. A rutin megkapja HL-ben a keresett sorszámot az értelmezőtől és így is adhatja vissza a címet. A megoldás igen egyszerű a BASIC 2EH funkcióhívás felhasználásával:

```
LNPT    RST  10H          ;funkcióhívás indul
        DEFB 2EH          ;BASIC 2E: HL. sor keresése
        DEFB 00H          ;funkcóhívás vége
        RET               ;   HL=cim
```

A rutin sajátossága, hogy nem létező sorszám esetén is értelmes címet ad vissza (a következő létező sor címét, ill. a program végét). Így a vizsgálni kívánt sor keresése mellett egyszerűen megkaphatjuk az aktuális program kezdő- és végcímét is. Ha csak akkor szeretnénk 0-tól különböző címet, ha létező sor számát adtuk meg, felhasználhatjuk, hogy a BASIC 2EH után a státusz "C", ha nem találta a keresett sort:

```
LNPT2   RST  10H          ;funkcióhívás indul
        DEFB 2EH          ;BASIC 2E: HL. sor keresése
        DEFB 00H          ;funkcióhívás vége
        JR   NC,L1        ;ugrik, ha talált ilyen sort
        LD   HL,0000H     ;HL=0, ha nem volt ilyen sor
L1      RET               ;   HL=cím vagy 0
```

Az ideiglenes használatra szánt rutint az ALLOCATE-mezőn is elhelyezhetjük, de ha 0-tól eltérő számú programokban is használni szeretnénk, töltsük inkább rögzített címre, pl. a 3000H (12288D) címre. Nem túl hosszú 0-s BASIC program esetén itt biztonságban van:

```
100 POKE 540,0
110 POKE 541,48 !3000H
120 CODE LNPT=HEX$("D7,2E,00,C9")
130 CODE LNPT2=HEX$("D7,2E,00,30,03,21,00,00,C9")
140 DEF LINEPTR(L)=USR(LNPT,L)
150 DEF LINEPTR2(L)=USR(LNPT2,L)
```

```
PRINT LINEPTR(100)
 4827
Ok
PRINT LINEPTR(0),LINEPTR2(0)
 4827          0
Ok
PRINT "Hossz=";LINEPTR(9999)-LINEPTR(0)
Hossz= 186
Ok
```

A program a DEF sorok segítségével azt is eléri, hogy valóban függvényként hívhatjuk rutinunkat, amint arra a képernyőmásolat néhány példát is mutat.

Ha más programban (0-tól különböző programszámon) is használni kívánjuk a rutint, ott is kell írnunk egy hívás-sort (persze a rögzített címmel, hiszen az a program nem ismerheti az LNPT változót):

```
DEF LINEPTR(L) = USR(12288,L)
```

VARPTR X (variable-pointer)

Az argumentumban megadott változó tárolási címét keresi meg. A bővítések következő fokozata, amelyben az argumentum már nem adható meg a szokásos szintaktikai keretek között. Itt tehát a bevezetőben leírt módszerhez kell folyamodnunk: a felhasználói rutinba lépve tovább kell elemeznünk a progamszöveget. Magát a keresés-funkciót is egy funkcióhívás szolgálja (BASIC 8EH). A feladat egyik megoldását már megadtuk a változók ábrázolási módjának bemutatásakor (3.2.8 pont, ill. a BASIC funkcióhívások alkalmazási példái között. Ott pl. az A változó címét a

```
CIM = USR(V—CIM,0)A
```

hívással kaptuk meg. A rutin feladata ebben az esetben:

— a programszövegben következő változónév keresése (BASIC 8EH)

— a következő elem vizsgálata (BASIC 20H), hogy a sor végére jusson az értelmező.

Most tekintsük ezt a feladatot esettanulmánynak, amelyben megpróbáljuk túllépni a legegyszerűbb megoldás korlátait. Az idézett rutinnak ui. két hiányossága van, amely szerencsére az alkalmazások nagy részében nem okoz gondot:

— a keresett változónevet nem lehet füzérként, egy változóban megadni, hiszen a hívást a nevet tartalmazó sorelemnek kell követnie. Így nem lehet pl. INPUT-ban sem bekérni a nevet;

— a tárolási cím nagy BASIC programnál 32 767 fölött lehet. Ezt a címet negatívként értelmezi és írja ki az értelmező.

Mindkét probléma megoldható a funkcióhívások bőséges szolgáltatásaival. Az algoritmus a következő:

1. A programszövegben a rutint követő füzér kiértékelése. Az eredmény (a név karaktereit tartalmazó füzér) a BASIC verembe kerül (BASIC 24H).

2. A név nagybetűsítése (az értelmező is így tárol, tehát így azonosítható) (BASIC C4H).

3. A szövegelemzés pointereinek beállítása a verembe írt füzérre.

4. Tárolási cím megkeresése (BASIC 8EH).

5. A füzér (most már fölösleges) blokkjának törlése a veremből (BASIC 16H).
A további lépések a cím — teljes tartományban való — helyes visszaadását szolgálják.

6. A MREG-ben lévő cím előjelének vizsgálata. Ha pozitív (értsd: kisebb 32 768-nál), a megoldás kész.

7. Az értelmező által negatívként értelmezett cím transzformációja:
MREG = 65 536 + (–CIM)

8. A befejező lépés a már többször említett veremrendezés annak érdekében, hogy az értelmező a MREG tartalmát (és ne HL értékét) kapja vissza. Mindezt a következő assembly progam valósítja meg:

```
VP      RST  10H          ;funkcióhívás indul
        DEFB 24H          ;BASIC 24: string -> MREG
        DEFB 00H          ;funkcióhívás vége
        LD   B,60H        ;"nagybetű"
        RST  10H          ;funkcióhívás indul
        DEFB 0C4H         ;BASIC C4: MREG nagybetűsítése
        DEFB 00H          ;hívás vége
        LD   HL,(0228H)   ;MREG kezdőcím (típusbájt)
        INC  HL           ;(füzérhossz)
        LD   A,(HL)       ;A=hossz
        LD   (0203H),A    ;rendszerváltozóba
        INC  HL           ;első karakter címe
        LD   (0347H),HL   ;szöveg-pointerbe
        RST  10H          ;funkcióhívás indul
        DEFB 8EH          ;BASIC 8E: HL=tárolási cim
        DEFB 02H          ;BASIC 02: HL -> MREG
        DEFB 97H          ;BASIC 97: MREG -> REG1
        DEFB 16H          ;BASIC 16: füzér törlése a
                          ;          BASIC veremből
        DEFB 20H          ;BASIC 20: köv. elem vizsgálata
        DEFB 94H          ;BASIC 94: cím vissza MREG-be
        DEFB 05H          ;BASIC 05: előjelvizsgálat
        DEFB 00H          ;hívás vége
        JR   Z,L1         ;marad a cím, ha < 32768
        RST  10H          ;funkcióhívás indul
        DEFB 92H,03H      ;BASIC 92,03: 32768 -> MREG2
        DEFB 15H          ;BASIC 15: 32768 -> MREG3
        DEFB 0CH          ;BASIC 0C: (+): MREG2=65536
        DEFB 0CH          ;BASIC 0C: (+): MREG=(-CIM)+64k
        DEFB 00H          ;hívás vége
L1      POP  BC           ;veremrendezés
        RET               ;
```

A BASIC betöltő most a 0180H területen tárolja a rutint (éppen a feltételezett nagyméretű program, változómező miatt most nem használhatjuk a BASIC terület közepét). A program a bővítés alkalmazási módjára is mutat példát:

```
 10 LET BC=1
 20 LET A$="SAMU"
100 POKE 540,128
110 POKE 541,1  : 0180H
120 CODE VP=HEX$("D7,24,00,06,60,D7,
    C4,00,2A,28,02,23,7E,32,03,02,
    23,22,47,03,D7,8E,02,97,16,20,
    94,05,00,28,07,D7,92,03,15,0C,
    0C,00,C1,C9")
130 DEF VARPTR(NEV$)=USR(VP,0) NEV$
140 INPUT PROMPT "A keresett valtozo
    ? ":N$
150 PRINT "Cim:";VARPTR(N$)
160 GOTO 140
```

```
START
A keresett valtozo ? BC
Cim: 5168
A keresett valtozo ? a$
Cim: 5180
A keresett valtozo ? A
Cim: 0
A keresett valtozo ? VARPTR
Cim: 5474
A keresett valtozo ? SIN
Cim: 4591
```

Befejezésül nézzünk egy példát az utasítás típusú bővítésre is. Valósítsuk meg pl. a DPOKE CIM,ADAT utasítást, ahol az ADAT egy kétbájtos egész:

$$ADAT = 256 \cdot HI + LO$$

és a kért művelet tulajdonképpen:

POKE CIM,LO
POKE CIM + 1,HI

Ha lemondunk a kerekítési problémák precíz kezeléséről (az értelmező az egészrész vételekor egyszerűen elhagyja a törtrészt, ezzel a pozitív értékeknél lefelé kerekít, de a negatív (azaz 32 767 fölötti) értékeknél fölfelé), a feladat igen egyszerűen megoldható: a CIM a USR argumentumában átadható, az ADAT a következő programszövegből olvasható be és a feladat azonnal végrehajtható a paraméterekkel:

```
DP      PUSH HL         ;CIM verembe
        RST  10H        ;funkcióhívás indul
        DEFB 23H        ;ADAT kiértékelés
        DEFB 0BH        ;ADAT -> HL
        DEFB 00H        ;funkcióhívás vége
        POP  DE         ;DE=CIM
        EX   DE,HL      ;HL=CIM, DE=ADAT
        LD   (HL),E     ;POKE HL,LO
        INC  HL         ;
        LD   (HL),D     ;POKE HL+1,HI
        RET
```

Itt nem megoldható a közvetlen (DPOKE CIM,ADAT típusú) hívás. A beírás, beillesztés egyszerűségéért az alkalmazás formai nehézkességével (CALL DPOKE(CIM,ADAT) kell fizetnünk:

```
100 POKE 540,0
110 POKE 541,48
120 CODE DP=HEX$("E5,D7,23,0B,00,D1,
    EB,73,23,72,C9")
130 DEF DPOKE(CIM,ADAT)
140   CALL USR(DP,CIM) ADAT
150 END DEF

START
ok
CALL DPOKE(0,16383)
ok
PRINT PEEK(0),PEEK(1)
 255      63
ok
```

## 4.2 Utasítás- és függvénybővítések

### 4.2.1 A kulcsszótábla kiterjesztése

Az utasítások szintaktikai ellenőrzésének és végrehajtásának bázisa — mint láttuk — a kulcsszótábla. Az alaphelyzetben kiépülő tábla 93 kulcsszó pointereit tartalmazza RAM-területen (tehát megváltoztathatóan). Ezek a pointerek 3 bájtból állnak (l. 3.2.5 pont):

— az első két bájt a kulcsszóhoz tartozó paraméterblokk címe,

— a 3. bájt a végrehajtó rutint tartalmazó szegmens.

A kulcsszó paraméterblokkja

| | |
|---|---|
| 1.—2. bájt: | a végrehajtó rutin (1. rutin) címe, |
| 3.—4. bájt: | a szintaktikai elemzést, kiegészítő tokenizálást végző 2. rutin címe, |
| 5. bájt: | a típusbájt, |
| 6. bájt: | a kulcsszó hossza, |
| 7. bájttól: | a kucsszó karakterei (nagybetűk). |

A kulcsszótábla szolgál alapul mind a programszöveg elemzésénél (a kulcsszavak felismerésénél), mind a végrehajtásnál. A tábla kezelési módja lehetőséget ad újabb táblák rendszerbe láncolására. A legelső két bájtot ui. egy táblalánc pointerének tekinti az értelmező: ha itt 0000H-tól különböző értéket talál, az adott tábla végigfutása után az itt megadott címen keresi a következő táblát és ezen folytatja a keresést. Egy-egy tábla elemeinek (3-bájtos pointerek!) számát a tábla 3. bájtja adja meg. A legelső táblára a (0232/3H) rendszerváltozó mutat.

Alaphelyzetben egyetlen kulcsszótábla épül ki (a 0E64H—0F7DH tartományban). Az első két bájt 0: a lánc nem folytatódik. A 3. bájt értéke 5DH (93D), az alapértelmezésű kulcsszavak száma.

Mit kell még tudnunk ahhoz, hogy a táblát biztonsággal terjeszthessük ki? A memóriakonfiguráció bizonyos megkötéseit. Az értelmező adott (elemző, végrehajtó) szakaszainak működése során a 3. LAP-on az UK szegmens van belapozva. Ezen kívül csak a nulláslap állapota tekinthető biztosnak. Ezek szabják meg bővítéseink elhelyezését. Az esetleges újabb kulcsszótáblák, valamint a tábla pointerei által címzett paraméterblokkok ui. az előbbiek miatt csak a nulláslapra helyezhetők. A végrehajtó rutinok meghívása viszont a BASIC lapozórutinján keresztül történik, miközben a kulcsszótábla pointerének 3. bájtján megadott szegmens kerül a 3. LAP-ra. Amíg tehát egy bővítés pointerének a 0. LAP-on lévő paraméterblokkra kell mutatnia, a paraméterblokk által megadott két rutincím tetszés szerint lehet a nulláslapon (ekkor a pointer 3. bájtja érdektelen) vagy a 3. LAP-on (ekkor viszont a rutint ténylegesen tartalmazó szegmens számát kell megadni a pointerben).

Milyen módjai vannak tehát a BASIC utasítások megváltoztatásának, kiterjesztésének?

1. A meglévő tábla pointereit átcímezhetjük RAM-területre (0. LAP!) és itt új paraméterblokkot létrehozva megváltoztathatjuk az adott kulcsszó nevét, szolgáltatásait, vagy az eredeti funkció feláldozásával új utasítást is készíthetünk. Ez a kevesebb előkészületet igénylő, egyszerűbb megoldás.

2. Új kulcsszótáblát láncolhatunk az eredeti után (az eléláncolás is megoldható, ekkor az új kulcsszavakat találja meg először a rendszer, de ez felborítja az eredeti tokenhozzárendelést, ezért nem javasolható).

Ennyi bevezetés után nézzünk most már konkrét példákat az elmondottak szemléltetésére.

Első kísérletként írjunk új paraméterblokkot a LET kulcsszóhoz. Pontosabban írjuk először az eredeti blokkot a RAM-ba és címezzük erre a LET pointert, míg a szegmensszámot hagyjuk meg eredeti (5) értékén: maradjon az értelmező szegmense belapozva a LET-rutinok futása alatt. A kulcsszótábla kezdőcímét a rendszerváltozóból olvassuk ki, KT a 0. pointerre mutat, a LET pointere a 40. a táblában (LET token: 28H):

```
100 LET KT=PEEK(562)+256*PEEK(563)+3
110 LET LETPTR=KT+40*3
120 POKE 540,0
130 POKE 541,48
140 CODE =HEX$("80,D2,67,D2,53,03")&
    "LET"
150 POKE LETPTR,0
160 POKE LETPTR+1,48
```

A futtatás után a LET minden szempontból az eredeti tulajdonságokkal bír. Ellenőrizzük, hogy most már valóban a mi paraméterblokkunk határozza meg ezeket a tulajdonságokat: adjunk első lépésként új nevet (pl. LEGYEN) az utasításnak:

```
100 LET KT=PEEK(562)+256*PEEK(563)+3
110 LET LETPTR=KT+40*3
120 POKE 540,0
130 POKE 541,48
145 CODE =HEX$("80,D2,67,D2,53,06")&
    "LEGYEN"
150 POKE LETPTR,0
160 POKE LETPTR+1,48


START
ok
LIST 110
  110 LEGYEN LETPTR=KT+40*3
ok
LET A=123

*** nicht verstanden.
LEGYEN A=321
ok
PRINT A
 321
ok
```

A futtatás után — mint az a képernyőmásolatból is látható — a listában új néven jelenik meg az utasítás, a továbbiakban értelmetlen a LET karaktersorozat, viszont felismeri (és értékadásként végre is hajtja) az értelmező a LEGYEN utasítást. Ezzel az egyszerű módszerrel akár az egész kulcsszókészlet is átnevezhető.

Írjuk át ezek után a 2. rutin címét pl. a PING végrehajtó rutinjára (persze ezzel elveszítjük az elemzés közbeni szintaktikai ellenőrzést):

```
140 CODE =HEX$("80,D2,BF,D3,53,06")&
    "LEGYEN"
```

Ez az átírás jól illusztrálja az elemzés lépését, hiszen mindig egy PING hang hallatszik, valahányszor az értelmező a beírt sor elemzésekor meghívja a LET 2. rutint. Próbálja beírni az utasítást parancsmódban vagy egy programsorba, és vesse össze a hallottakat az eddigi ismereteivel!

E kis kísérletezés után írjunk most már valódi bővítéseket a rendszerbe. Ehhez foglaljunk le ténylegesen védett területet a nulláslapon, hogy a továbbiakban ne legyen gondunk a rutinok, táblák elhelyezésével. A gép bekapcsolása után a 3.1 alfejezetben bemutatott egyszerű rutinnal toljuk el pl. 2000H-ra a BASIC munkaterületét:

```
100 ALLOCATE 100
110 CODE M=HEX$("21,00,20,3E,05,D3,B3,C3,AC,C2")
120 CALL USR(M,0)
```

Ezután (a BASIC újrainicializálásáig) szabadon használhatjuk az eredeti munkaterület 2000H alatti részét (az adott verzióban 12DBH—1FFFH). A továbbiakban tárgyalt példák ilyen eltolt BASIC rendszerben értendők, a mintaprogramban alkalmazott címek is erre az esetre vonatkoznak!

Először írjunk meg két olyan bővítést, amely az eredeti utasításkészlet elemeinek működését változtatják meg. Nincs tehát szükség új kulcsszavakra, csak a végrehajtási címeket kell átirányítani (mint a LET esetében).

*Rugalmas POKE*

Írjuk át első lépésként a POKE utasítást úgy, hogy működhessen eredeti funkciójában is, de ha a megadott ADAT kétbájtos érték (nagyobb 255-nél), automatikusan az előző pontban bemutatott DPOKE funkciót hajtsa végre. A 2. rutin címét meghagyhatjuk eredeti értékén, de a végrehajtó rutint nekünk kell megírnunk:

```
POKE.   RST  10H        ;funkcióhívás indul
        DEFB 23H        ;BASIC 23: kiértékelés (CIM)
        DEFB 0BH        ;BASIC 0B: HL=CIM
        DEFB 00H        ;funkcióhívás vége
        PUSH HL         ;CIM verembe
        LD   A,0CH      ;"," kódja
        RST  10H        ;funkcióhívás indul
        DEFB 22H        ;BASIC 22: "," követi?
        DEFB 23H        ;BASIC 23: kiértékelés (ADAT)
        DEFB 0BH        ;BASIC 0B: HL=ADAT
        DEFB 00H        ;funkcióhívás vége
        POP  DE         ;DE=CIM
        EX   DE,HL      ;HL=CIM, DE=ADAT
        LD   (HL),E     ;POKE CIM,LO
        LD   A,D        ;A=HI
        OR   A          ;A=0? (ADAT < 256 ?)
        RET  Z          ;kész, ha igen
        INC  HL         ;
        LD   (HL),D     ;POKE CIM+1,HI
        RET             ;
```

A rutinban most sem foglalkoztunk a kerekítés problémájával. Ha ez szükséges, az argumentumok kiértékelése után beépíthető az eredeti POKE végrehajtásnál látott (3.4.4 pont) kerekítő algoritmus.

A BASIC betöltő az előbbi példával analóg módon ad új paramétertáblát a POKE utasításnak és címzi át a megfelelő pointert (a POKE tokenje: 36H = 54D). A rutin pikantériája, hogy a pointer megváltoztatása két lépésben történik (alsó és felső bájt, 170., 180. sorok) és a két lépés között ideiglenesen értelmetlen a POKE utasítás. Ezért használtuk ezekben a sorokban a SPOKE utasítást (a nulláslap szegmense esetünkben a 248.):

```
100 LET KT=PEEK(562)+256*PEEK(563)+3
110 LET POKEPTR=KT+54*3
120 POKE 540,0
130 POKE 541,19
140 CODE =HEX$("10,13,61,D0,53,04")&
    "POKE"
150 POKE 540,16
160 CODE =HEX$("D7,23,08,00,E5,3E,
    0C,D7,22,23,08,00,D1,EB,73,7A,
    B7,C8,23,72,C9")
170 SPOKE 248,POKEPTR,0
180 SPOKE 248,POKEPTR+1,19
```

A rutin futtatása után pl. a 120, 130. sorok helyett POKE 540,4864 is írható. A betöltés után a BASIC program törölhető, csak arra kell ügyelnünk, hogy a POKE paraméterblokk (1300H—) és a végrehajtó rutin (1310H—1324H) a továbbiak folyamán is megőrződjön.

*GOTO parancsmódban*

Az ENTERPRISE BASIC-jének bosszantó hiányossága (nem sok van!), hogy parancsmódban nem fogadja el a GOTO utasítást. Igen kellemetlen egy esetleg hosszú feldolgozás során nyert adattömeg elvesztése, ha valamilyen ok miatt (pl. egy meggondolatlan STOP) leáll a program futása és újraindítása csak RUN-nal lehetséges. Az ilyen esetek jelentős részében segíthet, ha — kiterjesztve a beépített GOTO hatáskörét — adattörlés nélkül léphetünk a kívánt sorra.

Annak közvetlen oka, hogy az értelmező parancsmódban nem fogadja el ezt az utasítást, a paraméterblokk típusbájtjában (52H) keresendő: törölt a "parancsmódban használható" jelentésű 0. bit. A paraméterblokk RAM-ba helyezésével ez a korlátozás kiküszöbölhető, de ez még nem oldja meg a problémát. Az így átírt utasítást ugyan elfogadja az értelmező, de parancsmódban nem hajt végre semmit. Ennek oka a GOTO végrehajtási logikájában található meg: a rutin nem ugrik el a keresett sorra, csak betölti annak címét a megfelő pointerbe (0216H) és úgy állítja be folytatásjelzőt (0206H = 3), hogy a programvégrehajtási főág a célsoron folytassa a futást. Parancsmódban azonban nem a programvégrehajtási főágba tér vissza a vezérlés a GOTO-ból, és így nem is éri el a célját.

Ezek alapján már egyszerű a megoldás. A 2. rutint meghagyva eredeti címén, az elvégzi a célsorszám előkészítését (A2H típuskód). Így nekünk nem kell a

kiértékeléssel foglalkoznunk, a sorszámot a 0218/9H rendszerváltozóban készen kapjuk. A fenti rendszerváltozók beállítása után el is ugorhatnánk a végrehajtási főágba. Az eredeti GOTO rutinnak azonban van egy olyan szolgáltatása, amit nekünk is fel kell használnunk: ha egy megkezdett FOR-NEXT vagy DO-LOOP blokkból a blokk határain kívül kell ugrani, törli a BASIC veremből az érvénytelenné vált elemeket. Hívjuk hát meg saját rutinunkban is az eredeti GOTO rutint:

```
GOTO.   CALL 0D1DF      :eredeti GOTO rutin hívása
        JP   0CFA4      :elugrik a programfőágba
```

A betöltőprogram futtatása után már parancsmódban és programban egyaránt használhatjuk (eredeti szintaktikája szerint) az új utasítást.

```
100 LET KT=PEEK(562)+256*PEEK(563)+3

200 LET GOTOPTR=KT+33*3
210 POKE 540,0
220 POKE 541,20
230 CODE =HEX$("10,14,5A,D0,53,04")&
    "GOTO"
240 POKE 540,16
250 CODE =HEX$("CD,DF,D1,C3,A4,CF")
260 POKE GOTOPTR,0
270 POKE GOTOPTR+1,20
```

Ha új utasításokat szeretnénk létrehozni, megtartva és kiegészítve az értelmező eredeti utasításkészletét, a kulcsszótáblát kell megtoldani. Egy-két új utasítás esetén talán megoldást jelent a ritkán használt utasítások feláldozása (ilyen lehet pl. a CAPTURE, az OK, az EXT — elérhető másképp is —, a REM (van helyette "!" stb.). Perspektívát azonban csak egy új kulcsszótábla készítése és rendszerbe illesztése (beláncolása) jelent. Ezért erre nézzünk most néhány példát.

*Rövidített kulcsszavak*

Hosszadalmas programírás közben bizonyára sokan gondolnak nosztalgiával a SPECTRUM és COMMODORE 1-2 billentyűs kulcsszó beírási lehetőségére. Terjesszük ki ez irányba az ENTERPRISE lehetőségeit is: bővítsük a rendszert olyan utasításokkal, amelyek minden szempontból megegyeznek az eredeti funkciókkal, csak a nevük más: 1-2 karaker, igény szerint. Készítsünk pl. egy L (értsd: LIST) és egy PR (értsd: PRINT) elemet.

Az új tokentábla első 2 bájtja pointer a következő láncelemre (most nincs ilyen, tehát 0,0), 3. eleme pedig a tábla kulcsszavainak száma (itt 2). Ezután következnek a 3-bájtos pointerek, majd a paraméterblokkok. Az új tábla beláncolásához a tábla kezdőcímét kell beírni az alaptábla pointerébe:

```
100 LET KT=PEEK(562)+256*PEEK(563)+3

310 POKE 540,0
320 POKE 541,21
330 CODE TOKENTABLA_2=HEX$("00,00,02")
340 CODE LIST.=HEX$("10,15,05")
350 CODE PRINT.=HEX$("20,15,05")
360 !
370 !
380 POKE 540,16
390 CODE L=HEX$("2E,C8,9C,CA,41,01")&"L"
400 POKE 540,32
410 CODE PR=HEX$("08,D4,01,D4,53,02")&"PR"
420 !
430 ! BELANCOLAS
440 !
450 POKE KT-3,0
460 POKE KT-2,21
```

Az új kulcsszavak könnyen (és gyorsan) használhatók, de megvan az a hátrányuk, hogy a listában is a rövid alakban jelennek meg és ami még nagyobb probléma, a velük írott programok csak a bővítő jelenlétében futnak (alaphelyzetű gépbe visszaolvasva hibát okoznak). Megoldható ez a gond is, ha a 2. rutint is mi írjuk meg, legalábbis a bevezető szakaszát és abban visszaírjuk az új token helyére az eredetit:

```
PR_2    LD   HL,(0214H) ;elemzési pointer
        DEC  HL         ;vissza a tokenre
        LD   A,38H      ;PRINT token
        LD   (HL),A     ;a PR token helyére
        JP   0D401H     ;elugrik a PRINT 2. rutinra
```

A kiegészítéshez szükséges új sorok és hatásuk:

```
410 CODE PR=HEX$("08,D4,30,15,53,02"
    )&"PR"
412 POKE 540,48
414 CODE RUT2=HEX$("2A,14,02,2B,3E,
    38,77,C3,01,D4")
```

```
10 PR 123
LIST 10
   10 PRINT 123
ok
```

Befejezésként illesszünk két képernyőkezelő bővítést a rendszerhez. A videoprocesszor programozása, a képernyőkezelés, a grafika EXOS-on keresztüli vagy közvetlen elérése rendkívül nagy terület. Egész kötetet lehetne megtölteni a témára vonatkozó információkkal, alkalmazási példákkal, trükkökkel. Ebben az összállításban csak ízelítőként mutatunk be néhány adalékot. A 8. Függelékben mindenesetre

összefoglaljuk a legfontosabb hívásokat, adatokat, a következőkben pedig adunk néhány a — képernyőkezelést megkönnyítő — utasítás- és függvénybővítést.

*POKE a video RAM-ba*

A videoszegmensek (FCH—FFH elérésének nehézkességét az adja, hogy míg a megjelenítést végző NICK-chip a 0—64 k címtartományban összefüggően látja ezeket (0000—3FFF:FC szegmens, ..., C000—FFFF:FF szegmens), addig mi csak szegmensként érjük el ezt a területet: a videocím értékéből kell meghatároznunk, hogy melyik szegmensről van szó. A videocím — pl. sorparamétertáblából kiolvasható — értéke alapján tehát nehézkes pl. POKE utasítással adatot vinni a képernyőterületre. Készítsük el ezért a

VPOKE CIM,ADAT

bővítést, amely a megadott videocím alapján meghatározza a hozzárendelt szegmensszámot és abba tölti az adatot.

A bővítés 2. rutinjaként alkalmazhatjuk a beépített POKE 2. rutinját (hasonló a szintaktika) és a végrehajtó rutin első szakasza is átvehető az előbbiekben megismert POKE kiterjesztésből.

Itt az adat értéktartománya korlátozott (0—255), ezt tehát ellenőriznünk kell (az esetleges hibajelzésre a BASIC RST 20H hibakezelőjét használjuk fel).

Ezután következik az érdemi feladat. A címtartományt a címbájt első két bitje határozza meg. Ezt kiemelve és 0—3 közé transzformálva megkapjuk a relatív szegmensszámot. A rutin ezt a szegmenst lapozza az 1. LAP-ra, és a cím 4000—7FFF közé transzformálása után ide írja be az adatot:

```
VPOKE.  RST  10H        ;funkcióhívás indul
        DEFB 23H        ;BASIC 23: kiértékelés (CIM)
        DEFB 0BH        ;BASIC 0B: HL=CIM
        DEFB 00H        ;funkcióhívás vége
        PUSH HL         ;CIM verembe
        LD   A,0CH      ;"," kódja
        RST  10H        ;funkcióhívás indul
        DEFB 22H        ;BASIC 22: "," következik?
        DEFB 23H        ;BASIC 23: kiértékelés (ADAT)
        DEFB 0BH        ;BASIC 0B: HL=ADAT
        DEFB 00H        ;hívás vége
        POP  DE         ;DE=CIM
        LD   A,H        ;ADAT felső bájt
        OR   A          ;A=0?
        JR   Z,L1       ;rendben, ha ADAT < 256
        LD   HL,03E8H   ;"érték nem megfelelő"
        RST  20H        ;hibára
L1      EX   DE,HL      ;HL=CIM, E=ADAT
        LD   A,H        ;CIM felső bájt
        AND  0C0H       ; 1100 0000 maszkolás
        RLCA            ; érték eltolás
        RLCA            ; 00 - 03 közé
        ADD  0FCH       ;0. videoszegmens + érték
        OUT  (0B1H),A   ;kijelölt szegmens az 1. LAP-ra
        RES  7,H        ;cím transzformálás
        SET  6,H        ;     az 1. LAP tartományába
        LD   (HL),E     ;POKE CIM,ADAT
        RET             ;
```

A bővítés beillesztéséhez szükséges kiegészítés a következő:

```
330 CODE TOKENTABLA_2=HEX$("00,00,
    03")

360 CODE VPOKE.=HEX$("40,15,05")

500 POKE 540,64
510 CODE VPOKE=HEX$("50,15,61,D0,53,
    05")&"VPOKE"
520 POKE 540,80
530 CODE VP_RUT=HEX$("D7,23,0B,00,
    E5,3E,0C,D7,22,23,0B,00,D1,7C,
    B7,28,04,21,E8,03,E7,EB,7C,E6,
    C0,07,07,C6,FC,D3,B1,CB,BC,CB,
    F4,73,C9")
```

A program futtatása után használható a VPOKE utasítás, amely pl.
VPOKE 1,255
esetén az FCH szegmens 1. címére tölt 255-öt, de
VPOKE 49153,255
esetén az adat az FEH szegmens 1. címére kerül (49153D = C001H).

*Téglalaprajzolás*

Befejező példánk egy grafikus bővítés: az alapértelmezésű (#101) grafikus csatornára egy téglalapot rajzol:

BOX X,Y

A téglalap bal alsó csúcsa az aktuális PLOT-pozícióba kerül, alapja X, magassága Y. A sugár ki- és bekapcsolását a bővítés nem vezérli, azt a szokásos módon kell beállítani.

A megoldás alapja a VIDEO perifériakezelő által támogatott

ESC R,xx,yy

karaktersorozat írás (relatív sugármozgás vízszintesen xx — kétbájtos érték —, függőlegesen yy lépéssel). Rutinunk hatékonysága érdekében felhasználtuk az UK (esetünkben az 5.) szegmens FE7CH rutinját, amely az aktuális csatornára sorra kiírja az

1BH(ESC),B,E,D,L,H

értékeket, ill. regisztereket.
Az algoritmus ezek után egyszerűen követhető.
Lényegében a

PLOT X0, Y0; X0, Y0 + yy; X0 + xx, Y0 + yy; X0 + xx, Y0; X0, Y0

utasítás menetét követjük.

```
50X     LD    A,65H       ;GRAPHICS csatorna
        LD    (0209H),A   ;beállítása aktuális csat.ként
        RST   10H         ;funkcióhívás indul
        DEFB  23H         ;BASIC 23: kiértékelés (X)
        DEFB  98H         ;BASIC 98: REG2=X
        DEFB  00H         ;hívás vége
        LD    A,0CH       ;"," kódja
        RST   10H         ;funkcióhívás indul
        DEFB  22H         ;BASIC 22: "," következik?
        DEFB  23H         ;BASIC 23: kiértékelés (Y)
        DEFB  97H         ;BASIC 97: REG1=Y
        DEFB  94H         ;BASIC 94: MREG=Y
        DEFB  0BH         ;BASIC 0B: HL=Y
        DEFB  00H         ;hívás vége
        LD    DE,0000H    ;DE=0
        PUSH  DE          ;konstans verembe
        LD    B,52H       ;"R": relatív sugármozgás
        PUSH  BC          ;kód verembe
        CALL  0FE7CH      ;ESC R,0,Y:   bal oldal
        RST   10H         ;funkcióhívás indul
        DEFB  95H         ;BASIC 95: MREG=X
        DEFB  0BH         ;BASIC 0B: HL=X
        DEFB  00H         ;hívás vége
        EX    DE,HL       ;DE=X
        POP   BC          ;B="R"
        POP   HL          ;HL=0
        PUSH  HL          ;konstansok
        PUSH  BC          ;  vissza a verembe
        CALL  0FE7CH      ;ESC R,X,0:   felső él
        RST   10H         ;funkcióhívás indul
        DEFB  94H         ;BASIC 94: MREG=Y
        DEFB  06H         ;BASIC 06: MREG=-Y
        DEFB  0BH         ;BASIC 0B  HL=-Y
        DEFB  00H         ;hívás vége
        POP   BC          ;B="R"
        POP   DE          ;DE=0
        PUSH  DE          ;konstansok
        PUSH  BC          ;  vissza a verembe
        CALL  0FE7CH      ;ESC R,0,-Y:  jobb oldal
        RST   10H         ;funkcióhivás indul
        DEFB  95H         ;BASIC 95: MREG=X
        DEFB  06H         ;BASIC 06: MREG=-X
        DEFB  0BH         ;BASIC 0B: HL=-X
        DEFB  00H         ;hívás vége
        EX    DE,HL       ;DE=-X
        POP   BC          ;B="R"
        POP   HL          ;HL=0
        CALL  0FE7CH      ;ESC R,-X,0:  alsó él
        XOR   A           ;A=0
        LD    (0209H),A   ;aktuális csatorna alapérték
        RET               ;
```

Az értékek megfelelő tárolását és beolvasását, előjelváltását — egészekről lévén szó — talán valamivel egyszerűbben is megoldhattuk volna, de így legalább az aritmetikai funkcióhívásokat is gyakoroltuk egy kicsit.

Itt is megadjuk a BASIC betöltőprogram előbbiekhez illeszkedő kiegészítését.

```
330 CODE TOKENTABLA_2=HEX$("00,00,
    04")

370 CODE BOX.=HEX$("80,15,05")

540 POKE 540,128
550 CODE BOX=HEX$("90,15,61,D0,53,
    03")&"BOX"
560 POKE 540,144
570 CODE BX_RUT=HEX$("3E,65,32,09,
    02,D7,23,98,00,3E,0C,D7,22,23,
    97,94,0E,00,11,00,00,D5,06,52,
    C5,CD,7C,FE,D7,95,0B,00,EB,C1,
    E1,E5,C5,CD,7C,FE,D7,94,06,0B,
    00,C1,D1,D5,C5,CD,7C,FE,D7,95,
    06,0B,00,EB,C1,E1,CD,7C,FE,AF,
    32,09,02,C9")
```

## 4.2.2 A függvénytábla kiterjesztése

A beépített függvények és változók nyilvántartásának alapja a függvénytábla. A tábla elemei lényegében változó jellegű blokkok, csak az adatmező speciális. A változó fejrésze a szokásos:

1.—2. bájt: pointer a változók láncba fűzéséhez;
3. bájt: típus; beépített változónál:
0CH, ha numerikus,
0DH, ha füzér értékű;
4. bájt: névhossz (n);
5.—től: a név karakterei.

Ezután az adatmező következik, ami a 0CH, 0DH kódú változóknál 3-bájtos: a kiértékelő rutin címe és szegmense.

Ilyen változóként kell tehát nekünk is létrehoznunk saját függvénybővítéseink paraméterblokkját, majd a blokkot be kell láncoltatni a változónyilvántartásba (ehhez felhasználhatjuk a BASIC 80H funkcióhívást).

Fölmerülhet persze a kérdés, hogy a meglévő függvények esetleges átírásával nem lehetne-e egyszerűbben elérni a célunkat. Természetesen itt is feláldozhatunk egy-két ritkán használt függvényt, változót (pl. a VERNUM-ért vagy a VER$-ért nemigen fáj senkinek a szíve) és a név, valamint a kiértékelő rutin címe céljainknak megfelelően állítható be. Az igazi megoldást azonban itt is a védett területen

felépített rutinok és paraméterblokkok beláncolása jelenti. Erre mutatunk a következőkben példát.

Először nézzük meg néhány illusztrációnak szánt, de esetenként jól használható függvénybővítés funkcióját, működését, majd a fejezet végén együtt foglalkozzunk a bővítők elhelyezésével és rendszerbe illesztésével.

*DPEEK*

Az előző pontban megvalósított DPOKE párjaként jól használható (pl. a gépi kódú programozással kapcsolatban is) a

DPEEK(CIM)

függvény, ami lényegében a

PEEK(CIM) + PEEK(CIM + 1)· 256

kétbájtos értéket adja vissza.

A feladat viszonylag egyszerűen megoldható. A CIM argumentum (itt zárójelben lévő numerikus kifejezés) kiértékelésére a BASIC 99H hívás alkalmas. A kiolvasott kétbájtos egész értéket az értelmező számunkra most kellemetlen módon előjelesként értékeli. Az értéktranszformációt a 4.1 alfejezetben már alkalmazott (VARPTR) módszerrel végezhetjük el:

```
DPEEK   RST  10H          ;funkcióhívás indul
        DEFB 99H          ;BASIC 99: kiértékelés (CIM)
        DEFB 0BH          ;BASIC 0B: HL=CIM
        DEFB 00H          ;hívás vége
        LD   E,(HL)       ;E=PEEK(CIM)
        INC  HL           ;
        LD   D,(HL)       ;D=PEEK(CIM+1)
        EX   DE,HL        ;HL=DPEEK(CIM)
        RST  10H          ;funkcióhívás indul
        DEFB 02H          ;BASIC 02: HL -> MREG
        DEFB 05H          ;BASIC 05: előjel?
        DEFB 00H          ;hívás vége
        JR   Z,L1         ;kész, ha az eredmény pozit.
        RST  10H          ;funkcióhívás indul
        DEFB 92H,03H      ;BASIC 92,03: MREG2=32768
        DEFB 15H          ;BASIC 15: MREG3=32768
        DEFB 0CH          ;BASIC 0C: MREG2=65536
        DEFB 0CH          ;BASIC 0C: MREG=65536+(-ADAT
        DEFB 00H          ;hívás vége
L1      RET               ;
```

*A kifejezést kiértékelő VAL*

Az értelmező meglehetősen könnyedén veszi ezt a funkciót. Igaz, hogy a legtöbb személyi számítógép hasonló módon értelmezi, és csak a számjegyeket képes értékként visszaadni. Az első műveleti jelnél vagy változónévnél leáll az elemzés. Például:

VAL("2*PI") = 2

Csak aki a ZX-SPECTRUM kiértékelő VAL-ját ismeri (ott VAL("2*PI") = 6.28..), az tudja, milyen sokat veszített ezzel a hiányossággal. INPUT-ban megadható matematikai formák, a program menetétől függően füzérként összefűzhető vagy szeletelhető függvények mind olyan lehetőségek, amelyeket csak egy tényleges VAL-funkcióval érhetünk el.

A feladat nehézsége abban áll, hogy a kiértékelés csak tokenizált szövegen történhet. A megoldás algoritmusa a következő lehet:

— A VAL() argumentum kiértékelése. Az eredmény egy füzér a BASIC veremben.
— A füzér átmásolása a szövegpufferbe.
— Az elemzési pointerek ráállítása a feldolgozni kívánt szövegre.
— Tokenizálás (a ROM rutin hívásával).

A szokásos kiértékelés az első elem vizsgálata után következhet. Az eredmény a kifejezés numerikus értéke a BASIC veremben.

Ezzel a feladatot lényegében megoldottuk. A gyakorlatban nem szabad megfeledkezni az elrontott pointerek értékének visszaállításáról, a BASIC verem rendezéséről sem:

```
VAL2    RST  10H          ;funkcióhívás indul
        DEFB 9CH          ;BASIC 9C: füzér kiértékelés
        DEFB 00H          ;hívás vége
        LD   HL,(0214H)   ;pointer aktuális érték
        PUSH HL           ;verembe
        LD   HL,(0203H)   ;változó aktuális érték
        PUSH HL           ;verembe
        LD   HL,(0228H)   ;MREG kezdőcím
        INC  HL           ;hosszra mutat
        LD   C,(HL)       ;C=a füzér hossza
        INC  HL           ;HL: az első karakterre mutat
        LD   B,00H        ;BC=karakterek száma
        LD   DE,0249H     ;DE: puffercím
        PUSH DE           ;verembe
        LDIR              ;pufferbe másolja a füzért
        XOR  A            ;A=0
        LD   (DE),A       ;a végére lezáró 0-t ír
        RST  10H          ;funkcióhívás indul
        DEFB 16H          ;BASIC 16: BASIC verem törlése
        DEFB 00H          ;hívás vége
        POP  HL           ;puffercím
        LD   (0214H),HL   ;elemzési pointerként
        LD   A,05H        ;BASIC szegmens
        OUT  (0B3H),A     ;a 3. LAP-ra
        CALL 0C41DH       ;tokenizálás
        LD   A,01H        ;1. szegmens
        OUT  (0B3H),A     ;vissza
        RST  10H          ;funkcióhívás indul
        DEFB 20H          ;BASIC 20: az első sorelem
                                  vizsgálata
        DEFB 23H          ;BASIC 23: kiértékelés
        DEFB 00H          ;hívás vége
        POP  HL           ;változó belépési érték
        LD   (0203H),HL   ;visszaállítás
        POP  HL           ;pointer belépési érték
        LD   (0214H),HL   ;vissza
        RET               ;
```

A rutint VAL2-ként fűzzük a rendszerhez, hogy szükség esetére megmaradjon az eredeti VAL is.

*VIDEO cím lekérdezése*

A VIDEO kezelőhöz megnyitott csatornák közvetlen elérésének alapja a képet tároló terület kezdő címe (0000—FFFF közötti videocím). Ennek lekérdezését az EXOS lehetővé teszi egy speciális funkcióhívással. Bővítsük a BASIC rendszert egy

V-CIM(N)

függvénnyel, amely az N. csatorna elsődleges videocímét adja vissza.

A feladat megoldása néhány lépés:

— Az argumentum kiértékelése és betöltése az A regiszterbe (csatornaszám). Tekintsünk el az értékellenőrzéstől, ne bonyolítsuk túlságosan ezt az egyszerű rutint (láttunk már rá példát).

— EXOS 0BH hívás B = 3 funkciókóddal. Ez a VIDEO-nak szóló ADDR hívás. Az eredmény: az elsődleges tárolási cím BC-ben.

— Az eredmény munkaregiszterbe töltése, az esetleges negatív (azaz 32 767 fölötti) érték transzformációja (l. pl. DPEEK).

```
V_CIM   RST  10H        ;funkcióhívás indul
        DEFB 99H        ;BASIC 99: kiértékelés
        DEFB 0BH        ;BASIC 0B: H=csatornaszám
        DEFB 00H        ;hívás vége
        LD   A,L        ;A: csatornaszám
        LD   B,03H      ;B: alfunkciószám: ADDR
        EXOS 0BH        ;speciális funkció (tár.cím)
        RST  18H        ;hibaellenőrzés
        PUSH BC         ;elsődleges tárolási cím verembe
        POP  HL         ;HL=V_CIM
        ...             ;értéktranszformáció, ha negatív
```

*PEEK a video RAM-ból*

Az előző pont VPOKE utasításának párjaként készítsünk egy

VPEEK(CIM)

függvénybővítést, amely a 0000—FFFF közötti videocím alapján megkeresi a címhez rendelt szegmenst és abból olvassa ki az adatot.

A videocím kezelés algoritmusát láttuk a VPOKE utasításnál. Ezzel a VPEEK kiértékelése egyszerűen megoldható:

```
VPEEK   RST  10H          ;funkcióhívás indul
        DEFB 99H          ;BASIC 99: kiértékelés (CIM)
        DEFB 0BH          ;BASIC 0B: HL=CIM
        DEFB 00H          ;hívás vége
        LD   A,H          ;felső bájt
        AND  0C0H         ; 1100 0000 maszkolás
        RLCA              ; érték eltolás
        RLCA              ; 00 és 03 közé
        ADD  0FCH         ;0. videoszegmens + érték
        LD   C,A          ;mentés
        IN   A,(0B1H)     ;pillanatnyi 1. LAP-szegmens
        PUSH AF           ;verembe
        LD   A,C          ;video_szegmens
        OUT  (0B1H),A     ;az 1. LAP-ra
        RES  7,H          ;cím transzformálás
        SET  6,H          ;  az 1. LAP tartományába
        LD   L,(HL)       ;PEEK
        LD   H,00H        ;HL=VPEEK(CIM)
        POP  AF           ;eredeti 1. LAP-szegmens
        OUT  (0B1H),A     ;visszalapozása
        RST  10H          ;funkcióhívás indul
        DEFB 02H          ;BASIC 02: ADAT -> MREG
        DEFB 00H          ;hívás vége
        RET               ;
```

Ezek után rátérünk a példaként kidolgozott bővítések elhelyezésére és beláncolására. Hangsúlyozzuk, hogy a következő megoldás (az adott címekkel) csak az előző pont szerint eltolt BASIC munkaterület esetén lehetséges. A betöltőprogram mind sorszámaiban, mind tárolási területeiben illeszkedik a 4.2.1 pont utasításbővítéseihez, azokkal egy időben beírható, használható (így válik kerekké pl. a V—CIM, VPEEK, VPOKE bővítéssor).

A program sorban betölti (az áttekinthetőség érdekében hexadecimálisan kerek címekre) a függvénybővítések végrehajtó rutinjait, majd paraméterblokkjukat. Ez utóbbiak tartalmazzák a neveket is, amelyeket tetszés szerint meg is változtathatunk (ügyelve a név előtti hosszbájtra). Befejezésként a program M gépi rutinja (HL-ben a paraméter blokkok címével) ismételten meghívja a beláncoló BASIC 80H funkciót.

```
1000 POKE 540,0
1010 POKE 541,22
1020 CODE DPEEK.=HEX$("D7,99,0B,00,
     5E,23,56,EB,D7,02,05,00,28,07,
     D7,92,03,15,0C,0C,00,C9")
1030 POKE 540,32
1040 CODE =HEX$("00,00,0C,05")&"
     DPEEK"&HEX$("00,16,01")
1050 POKE 540,48
1060 CODE VAL2.=HEX$("D7,9C,00,2A,14,
     02,E5,2A,03,02,E5,2A,28,02,23,
     4E,23,06,00,11,49,02,D5,ED,B0,
     AF,12,D7,16,00,E1,22,14,02,3E,
     05,D3,B3,CD,1D,C4,3E,01,D3,B3,
     D7,20,23,00,E1,22,03,02,E1,22,
     14,02,C9")
1070 POKE 540,128
```

```
1080 CODE =HEX$("00,00,0C,04")&"VAL2"
     &HEX$("30,16,01")
1090 POKE 540,144
1100 CODE V_CIM.=HEX$("D7,99,0B,00,
     7D,06,03,F7,0B,DF,C5,E1,D7,02,
     03,00,28,07,D7,92,03,15,0C,0C,
     00,C9")
1110 POKE 540,176
1120 CODE =HEX$("00,00,0C,05")&"V_
     CIM"&HEX$("90,16,01")
1130 POKE 540,192
1140 CODE VPEEK.=HEX$("D7,99,0B,00,
     7C,E6,C0,07,07,C6,FC,4F,DB,B1,
     F5,79,D3,B1,CB,BC,CB,F4,6E,26,
     00,F1,D3,B1,D7,02,00,C9")
1150 POKE 540,240
1160 CODE =HEX$("00,00,0C,05")&"
     VPEEK"&HEX$("C0,16,01")
1170 CODE M=HEX$("21,20,16,D7,80,00")
1180 CODE =HEX$("21,80,16,D7,80,00")
1190 CODE =HEX$("21,B0,16,D7,80,00")
1200 CODE =HEX$("21,F0,16,D7,80,00,
     C9")
1210 CALL USR(M,0)
```

A futtatás után használható a négy új függvény. Például GRAPHICS után

CIM = V—CIM(101)

a nagyfelbontású kép bal felső pontjának tárolási címe. A pont (ill. az azt tartalmazó bájt) lekérdezhető az X = VPEEK(CIM), írható a VPOKE CIM,X utasítással.

A listában látható paraméterblokkok (1040., 1080., 1120. és 1160. sorok) az első két bájtra (pointer) 0-t adnak meg. Ezt a két bájtot majd az értelmező tölti ki, létrehozva a folyamatos változóláncot (a beláncoláskor). Ezután viszont nem szabad újraindítani a programot, hiszen az újra nullázná a most már érvényes pointereket, megszakítva ezzel a kialakított láncokat.

Végül még egy megjegyzés. A bemutatott példákat a kötet szándékának megfelelően (minél kevesebb eszköz, minél egyszerűbb és gyorsabb beírás) BASIC-ből töltöttük és illesztettük a rendszerbe. A rutinokat, táblákat természetesen egy assembler program (pl. ASMON) segítségével is elkészíthetjük (ezt szolgálja a rutinok assembly listáinak megadása is) és egy betöltő, beláncoló rutinnal kiegészítve teljes bővítőprogramot nyerhetünk. Ha ezt az assembler lehetővé teszi, a lefordított programot felhasználói áthelyezhető modulként rögzíthetjük kazettán.

Az ilyen modult a BASIC a munkaterülete elé szúrja be (tehát hasonló területre, mint ahol mi tároltuk rutinjainkat).

# 4.3 RAM-értelmező

Mint a bevezetőben említettük, a rendszer inicilizálási logikája lehetőséget ad arra, hogy magát az értelmezőt (esetünkben az 5. szegmens tartalmát) is RAM-ba helyezzük, és ezt illesszük be a rendszerbe. Aki kísérletezni, játszani szeretne a lehetőségekkel, ennél többet nem is kaphatna a géptől.

Az áttöltéshez kérjünk egy szegmenst az EXOS-tól (méghozzá megtévesztve a rendszert, perifériaszegmegsként kérjük, hogy a kiutalás igazán tartós maradjon). A kapott szegmensre töltsük át az 5. szegmens tartalmát és az új BASIC szegmens számát írjuk be a ROM-táblába, az 5. szegmens nyilvántartási helyére. Ez a mi verziónknál az ABC5H cím (l. 2.3 alfejezet):

```
LD   A,0FFH       ;rendszerszegmens
OUT  (0B2H),A     ;a 2. LAP-ra
LD   HL,0BF79H    ;"periféria" jelzőbájt
INC  (HL)         ;beállitása
PUSH HL           ;
EXOS 18H          ;szegmens kérés
POP  HL           ;jelzőbájt címe
DEC  (HL)         ;törlés
LD   A,C          ;A: a kapott szegmens száma
LD   (0ABC5H),A   ;beírja a ROM-táblába
                   (az 5. szegmens helyére)
OUT  (0B1H),A     ;kapott szegmens az 1. LAP-ra
LD   A,05H        ;BASIC szegmens
OUT  (0B2H),A     ;a 2. LAP-ra
LD   HL,8000H     ;forrás: BASIC szegmens
LD   DE,4000H     ;cél: kiutalt RAM szegmens
LD   BC,4000H     ;hossz: 16384 bájt
LDIR              ;átmásolás a RAM-ba
RET               ;
```

Ugyanez BASIC-ből betöltve:

```
100 ALLOCATE 100
110 CODE M=HEX$("3E,FF,D3,B2,21,79,
    BF,34,E5,F7.18,E1,35,79,32,C5,
    A8,D3,B1,3E.05,D3,B2,21,00,80,
    11,00,40,01.00,40,ED,B0,C9")
120 CALL USR(M,0)
130 EXT "BASIC"
```

A program az áttöltés után BASIC inicializálást kér, azaz — a most már átírt ROM-tábla alapján — újraépül a BASIC rendszer. A BASIC funkcióhívások, a kulcsszórutinok mind-mind az új RAM-szegmenst használják, ami rendkívüli lehetőséget ad a változtatásokra, bővítésekre.

Hogy egy-két tippet is adjunk az első próbálkozásokhoz, írjuk át pl. az állapotsor üzenetszövegét tetszésünk szerint:

```
100 OUT 177,PEEK(197)
110 POKE 540,10
120 POKE 541,67
130 CODE ="RAM-BASIC"
```

Most már akár a szövegszerkesztőből való visszatéréskor is a mi kiírásunk jelenik meg az állapotsorban (de akár le is tilthatjuk ezt a funkciót a beolvasási főágban).

Most már közvetlenül is manipulálható a kulcsszavak paramétertáblája. Átírható pl. a LIST típusbájtja, amely megakadályozta az utasítás programbeli alkalmazását:

SPOKE PEEK(197),63357,67

Nagyobb változtatásoknál az egész értelmező elkészíthető (és háttértárolóra vihető) abszolút (6-os típusú) rendszerbővítőként is. Így egy a beolvasáskor automatikusan beláncolódó saját BASIC-et alakíthatunk ki.

A további lehetőségek az Olvasó fantáziáján, vállalkozókedvén múlnak. Jó ötleteket és kevés "elszállt" programot kívánunk a kísérletezéshez!

# FÜGGELÉK

# 1. Karakterek és Z80-as utasítások

A táblázat oszlopai:

— Dec: tízes számrendszerbeli érték
— Hex: az érték hexadecimális alakja
— Ch: az érték által kódolt karakter (CHR$). Ha BRD és UK módban különbözik a jel, elöl a német módú karakter áll.
Z80: a kódolt Z80-as utasítás — közvetlenül
— ha a kód CBH után áll
— ha a kód EDH után áll

| Dec | Hex | Z80 | CB után |
|---|---|---|---|
| 0 | 00 | NOP | RLC B |
| 1 | 01 | LD BC,nn | RLC C |
| 2 | 02 | LD (BC),A | RLC D |
| 3 | 03 | INC BC | RLC E |
| 4 | 04 | INC B | RLC H |
| 5 | 05 | DEC B | RLC L |
| 6 | 06 | LD B,n | RLC (HL) |
| 7 | 07 | RLCA | RLC A |
| 8 | 08 | EX AF,AF' | RRC B |
| 9 | 09 | ADD HL,BC | RRC C |
| 10 | 0A | LD A,(BC) | RRC D |
| 11 | 0B | DEC BC | RRC E |
| 12 | 0C | INC C | RRC H |
| 13 | 0D | DEC C | RRC L |
| 14 | 0E | LD C,n | RRC (HL) |
| 15 | 0F | RRCA | RRC A |
| 16 | 10 | DJNZ d | RL B |
| 17 | 11 | LD DE,nn | RL C |
| 18 | 12 | LD (DE),A | RL D |
| 19 | 13 | INC DE | RL E |
| 20 | 14 | INC D | RL H |
| 21 | 15 | DEC D | RL L |
| 22 | 16 | LD D,n | RL (HL) |
| 23 | 17 | RLA | RL A |
| 24 | 18 | JR d | RR B |
| 25 | 19 | ADD HL,DE | RR C |
| 26 | 1A | LD A,(DE) | RR D |
| 27 | 1B | DEC DE | RR E |
| 28 | 1C | INC E | RR H |
| 29 | 1D | DEC E | RR L |
| 30 | 1E | LD E,n | RR (HL) |
| 31 | 1F | RRA | RR A |

| Dec | Hex | Ch | Z80 | | CB után |
|---|---|---|---|---|---|
| 32 | 20 | | JR | NZ,d | SLA B |
| 33 | 21 | ! | LD | HL,nn | SLA C |
| 34 | 22 | " | LD | (nn),HL | SLA D |
| 35 | 23 | # £ | INC | HL | SLA E |
| 36 | 24 | $ | INC | H | SLA H |
| 37 | 25 | % | DEC | H | SLA L |
| 38 | 26 | & | LD | H,n | SLA (HL) |
| 39 | 27 | ' | DAA | | SLA A |
| 40 | 28 | ( | JR | Z,d | SRA B |
| 41 | 29 | ) | ADD | HL,HL | SRA C |
| 42 | 2A | * | LD | HL,(nn) | SRA D |
| 43 | 2B | + | DEC | HL | SRA E |
| 44 | 2C | , | INC | L | SRA H |
| 45 | 2D | - | DEC | L | SRA L |
| 46 | 2E | . | LD | L,n | SRA (HL) |
| 47 | 2F | / | CPL | | SRA A |
| 48 | 30 | 0 | JR | NC,d | |
| 49 | 31 | 1 | LD | SP,nn | |
| 50 | 32 | 2 | LD | (nn),A | |
| 51 | 33 | 3 | INC | SP | |
| 52 | 34 | 4 | INC | (HL) | |
| 53 | 35 | 5 | DEC | (HL) | |
| 54 | 36 | 6 | LD | (HL),n | |
| 55 | 37 | 7 | SCF | | |
| 56 | 38 | 8 | JR | C,d | SRL B |
| 57 | 39 | 9 | ADD | HL,SP | SRL C |
| 58 | 3A | : | LD | A,(nn) | SRL D |
| 59 | 3B | ; | DEC | SP | SRL E |
| 60 | 3C | < | INC | A | SRL H |
| 61 | 3D | = | DEC | A | SRL L |
| 62 | 3E | > | LD | A,n | SRL (HL) |
| 63 | 3F | ? | CCF | | SRL A |

| Dec | Hex | Ch | Z80 | | CB után | | ED, után | |
|---|---|---|---|---|---|---|---|---|
| 64 | 40 | § @ | LD | B,B | BIT | 0,B | IN | B,(C) |
| 65 | 41 | A | LD | B,C | BIT | 0,C | OUT | (C),B |
| 66 | 42 | B | LD | B,D | BIT | 0,D | SBC | HL,BC |
| 67 | 43 | C | LD | B,E | BIT | 0,E | LD | (nn),BC |
| 68 | 44 | D | LD | B,H | BIT | 0,H | NEG | |
| 69 | 45 | E | LD | B,L | BIT | 0,L | RETN | |
| 70 | 46 | F | LD | B,(HL) | BIT | 0,(HL) | IM | 0 |
| 71 | 47 | G | LD | B,A | BIT | 0,A | LD | I,A |
| 72 | 48 | H | LD | C,B | BIT | 1,B | IN | C,(C) |
| 73 | 49 | I | LD | C,C | BIT | 1,C | OUT | (C),C |
| 74 | 4A | J | LD | C,D | BIT | 1,D | ADD | HL,BC |
| 75 | 4B | K | LD | C,E | BIT | 1,E | LD | BC,(nn) |
| 76 | 4C | L | LD | C,H | BIT | 1,H | | |
| 77 | 4D | M | LD | C,L | BIT | 1,L | RETI | |
| 78 | 4E | N | LD | C,(HL) | BIT | 1,(HL) | | |
| 79 | 4F | O | LD | C,A | BIT | 1,A | LD | R,A |
| 80 | 50 | P | LD | D,B | BIT | 2,B | IN | D,(C) |
| 81 | 51 | Q | LD | D,C | BIT | 2,C | OUT | (C),D |
| 82 | 52 | R | LD | D,D | BIT | 2,D | SBC | HL,DE |
| 83 | 53 | S | LD | D,E | BIT | 2,E | LD | (nn),DE |
| 84 | 54 | T | LD | D,H | BIT | 2,H | | |
| 85 | 55 | U | LD | D,L | BIT | 2,L | | |
| 86 | 56 | V | LD | D,(HL) | BIT | 2,(HL) | IM | 1 |
| 87 | 57 | W | LD | D,A | BIT | 2,A | LD | A,I |
| 88 | 58 | X | LD | E,B | BIT | 3,B | IN | E,(C) |
| 89 | 59 | Y | LD | E,C | BIT | 3,C | OUT | (C),E |
| 90 | 5A | Z | LD | E,D | BIT | 3,D | ADC | HL,DE |
| 91 | 5B | Ä [ | LD | E,E | BIT | 3,E | LD | DE,(nn) |
| 92 | 5C | Ö \ | LD | E,H | BIT | 3,H | | |
| 93 | 5D | Ü ] | LD | E,L | BIT | 3,L | | |
| 94 | 5E | ^ | LD | E,(HL) | BIT | 3,(HL) | IM | 2 |
| 95 | 5F | _ | LD | E,A | BIT | 3,A | LD | A,R |

| Dec | Hex | Ch | Z80 | | CB után | ED után |
|---|---|---|---|---|---|---|
| 96 | 60 | ` ' | LD | H,B | BIT 4,B | IN H,(C) |
| 97 | 61 | a | LD | H,C | BIT 4,C | OUT (C),H |
| 98 | 62 | b | LD | H,D | BIT 4,D | SBC HL,HL |
| 99 | 63 | c | LD | H,E | BIT 4,E | LD (nn),HL |
| 100 | 64 | d | LD | H,H | BIT 4,H | |
| 101 | 65 | e | LD | H,L | BIT 4,L | |
| 102 | 66 | f | LD | H,(HL) | BIT 4,(HL) | |
| 103 | 67 | g | LD | H,A | BIT 4,A | RRD |
| 104 | 68 | h | LD | L,B | BIT 5,B | IN L,(C) |
| 105 | 69 | i | LD | L,C | BIT 5,C | OUT (C),L |
| 106 | 6A | j | LD | L,D | BIT 5,D | ADC HL,HL |
| 107 | 6B | k | LD | L,E | BIT 5,E | LD HL,(nn) |
| 108 | 6C | l | LD | L,H | BIT 5,H | |
| 109 | 6D | m | LD | L,L | BIT 5,L | |
| 110 | 6E | n | LD | L,(HL) | BIT 5,(HL) | |
| 111 | 6F | o | LD | L,A | BIT 5,A | RLD |
| 112 | 70 | p | LD | (HL),B | BIT 6,B | |
| 113 | 71 | q | LD | (HL),C | BIT 6,C | |
| 114 | 72 | r | LD | (HL),D | BIT 6,D | SBC HL,SP |
| 115 | 73 | s | LD | (HL),E | BIT 6,E | LD (nn),SP |
| 116 | 74 | t | LD | (HL),H | BIT 6,H | |
| 117 | 75 | u | LD | (HL),L | BIT 6,L | |
| 118 | 76 | v | HALT | | BIT 6,(HL) | |
| 119 | 77 | w | LD | (HL),A | BIT 6,A | |
| 120 | 78 | x | LD | A,B | BIT 7,B | IN A,(C) |
| 121 | 79 | y | LD | A,C | BIT 7,C | OUT (C),A |
| 122 | 7A | z | LD | A,D | BIT 7,D | ADD HL,SP |
| 123 | 7B | ä { | LD | A,E | BIT 7,E | LD SP,(nn) |
| 124 | 7C | ö \| | LD | A,H | BIT 7,H | |
| 125 | 7D | ü } | LD | A,L | BIT 7,L | |
| 126 | 7E | ß ~ | LD | A,(HL) | BIT 7,(HL) | |
| 127 | 7F | ⌂ | LD | A,A | BIT 7,A | |

| Dec | Hex | Ch | Z80 | CB után | ED után |
|---|---|---|---|---|---|
| 128 | 80 | Ç | ADD A,B | RES 0,B | |
| 129 | 81 | # | ADD A,C | RES 0,C | |
| 130 | 82 | H | ADD A,D | RES 0,D | |
| 131 | 83 | ð | ADD A,E | RES 0,E | |
| 132 | 84 | U | ADD A,H | RES 0,H | |
| 133 | 85 | á | ADD A,L | RES 0,L | |
| 134 | 86 | ø | ADD A,(HL) | RES 0,(HL) | |
| 135 | 87 | R | ADD A,A | RES 0,A | |
| 136 | 88 | é | ADC A,B | RES 1,B | |
| 137 | 89 | Ç | ADC A,C | RES 1,C | |
| 138 | 8A | à | ADC A,D | RES 1,D | |
| 139 | 8B | ↑ | ADC A,E | RES 1,E | |
| 140 | 8C | → | ADC A,H | RES 1,H | |
| 141 | 8D | ∩ | ADC A,L | RES 1,L | |
| 142 | 8E | ■ | ADC A,(HL) | RES 1,(HL) | |
| 143 | 8F | ƒ | ADC A,A | RES 1,A | |
| 144 | 90 | ╥ | SUB B | RES 2,B | |
| 145 | 91 | Σ | SUB C | RES 2,C | |
| 146 | 92 | ä | SUB D | RES 2,D | |
| 147 | 93 | ö | SUB E | RES 2,E | |
| 148 | 94 | ü | SUB H | RES 2,H | |
| 149 | 95 | à | SUB L | RES 2,L | |
| 150 | 96 | ø | SUB (HL) | RES 2,(HL) | |
| 151 | 97 | æ | SUB A | RES 2,A | |
| 152 | 98 | è | SBC A,B | RES 3,B | |
| 153 | 99 | ö | SBC A,C | RES 3,C | |
| 154 | 9A | Ü | SBC A,D | RES 3,D | |
| 155 | 9B | ↓ | SBC A,E | RES 3,E | |
| 156 | 9C | ← | SBC A,H | RES 3,H | |
| 157 | 9D | ¥ | SBC A,L | RES 3,L | |
| 158 | 9E | ← | SBC A,(HL) | RES 3,(HL) | |
| 159 | 9F | ■ | SBC A,A | RES 3,A | |

| Dec | Hex | Z80 | CB után | ED után |
|---|---|---|---|---|
| 160 | A0 | AND B | RES 4,B | LDI |
| 161 | A1 | AND C | RES 4,C | CPI |
| 162 | A2 | AND D | RES 4,D | INI |
| 163 | A3 | AND E | RES 4,E | OUTI |
| 164 | A4 | AND H | RES 4,H | |
| 165 | A5 | AND L | RES 4,L | |
| 166 | A6 | AND (HL) | RES 4,(HL) | |
| 167 | A7 | AND A | RES 4,A | |
| 168 | A8 | XOR B | RES 5,B | LDD |
| 169 | A9 | XOR C | RES 5,C | CPD |
| 170 | AA | XOR D | RES 5,D | IND |
| 171 | AB | XOR E | RES 5,E | OUTD |
| 172 | AC | XOR H | RES 5,H | |
| 173 | AD | XOR L | RES 5,L | |
| 174 | AE | XOR (HL) | RES 5,(HL) | |
| 175 | AF | XOR A | RES 5,A | |
| 176 | B0 | OR B | RES 6,B | LDIR |
| 177 | B1 | OR C | RES 6,C | CPIR |
| 178 | B2 | OR D | RES 6,D | INIR |
| 179 | B3 | OR E | RES 6,E | OTIR |
| 180 | B4 | OR H | RES 6,H | |
| 181 | B5 | OR L | RES 6,L | |
| 182 | B6 | OR (HL) | RES 6,(HL) | |
| 183 | B7 | OR A | RES 6,A | |
| 184 | B8 | CP B | RES 7,B | LDDR |
| 185 | B9 | CP C | RES 7,C | CPDR |
| 186 | BA | CP D | RES 7,D | INDR |
| 187 | BB | CP E | RES 7,E | OTDR |
| 188 | BC | CP H | RES 7,H | |
| 189 | BD | CP L | RES 7,L | |
| 190 | BE | CP (HL) | RES 7,(HL) | |
| 191 | BF | CP A | RES 7,A | |

| Dec | Hex | Z80 | CB után |
|---|---|---|---|
| 192 | C0 | RET NZ | SET 0,B |
| 193 | C1 | POP BC | SET 0,C |
| 194 | C2 | JP NZ,nn | SET 0,D |
| 195 | C3 | JP nn | SET 0,E |
| 196 | C4 | CALL NZ,nn | SET 0,H |
| 197 | C5 | PUSH BC | SET 0,L |
| 198 | C6 | ADD A,n | SET 0,(HL) |
| 199 | C7 | RST 00H | SET 0,A |
| 200 | C8 | RET Z | SET 1,B |
| 201 | C9 | RET | SET 1,C |
| 202 | CA | JP Z,nn | SET 1,D |
| 203 | CB | | SET 1,E |
| 204 | CC | CALL Z,nn | SET 1,H |
| 205 | CD | CALL nn | SET 1,L |
| 206 | CE | ADC A,n | SET 1,(HL) |
| 207 | CF | RST 08H | SET 1,A |
| 208 | D0 | RET NC | SET 2,B |
| 209 | D1 | POP DE | SET 2,C |
| 210 | D2 | JP NC,nn | SET 2,D |
| 211 | D3 | OUT (n),A | SET 2,E |
| 212 | D4 | CALL NC,nn | SET 2,H |
| 213 | D5 | PUSH DE | SET 2,L |
| 214 | D6 | SUB n | SET 2,(HL) |
| 215 | D7 | RST 10H | SET 2,A |
| 216 | D8 | RET C | SET 3,B |
| 217 | D9 | EXX | SET 3,C |
| 218 | DA | JP C,nn | SET 3,D |
| 219 | DB | IN A,(n) | SET 3,E |
| 220 | DC | CALL C,nn | SET 3,H |
| 221 | DD | | SET 3,L |
| 222 | DE | SBC A,n | SET 3,(HL) |
| 223 | DF | RST 18H | SET 3,A |

| Dec | Hex | Z80 | CB után |
|---|---|---|---|
| 224 | E0 | RET PO | SET 4,B |
| 225 | E1 | POP HL | SET 4,C |
| 226 | E2 | JP PO,nn | SET 4,D |
| 227 | E3 | EX (SP),HL | SET 4,E |
| 228 | E4 | CALL PO,nn | SET 4,H |
| 229 | E5 | PUSH HL | SET 4,L |
| 230 | E6 | AND n | SET 4,(HL) |
| 231 | E7 | RST 20H | SET 4,A |
| 232 | E8 | RET PE | SET 5,B |
| 233 | E9 | JP (HL) | SET 5,C |
| 234 | EA | JP (PE),nn | SET 5,D |
| 235 | EB | EX DE,HL | SET 5,E |
| 236 | EC | CALL PE,nn | SET 5,H |
| 237 | ED | | SET 5,L |
| 238 | EE | XOR n | SET 5,(HL) |
| 239 | EF | RST 28H | SET 5,A |
| 240 | F0 | RET P | SET 6,B |
| 241 | F1 | POP AF | SET 6,C |
| 242 | F2 | JP P,nn | SET 6,D |
| 243 | F3 | DI | SET 6,E |
| 244 | F4 | CALL P,nn | SET 6,H |
| 245 | F5 | PUSH AF | SET 6,L |
| 246 | F6 | OR n | SET 6,(HL) |
| 247 | F7 | RST 30H | SET 6,A |
| 248 | F8 | RET M | SET 7,B |
| 249 | F9 | LD SP,HL | SET 7,C |
| 250 | FA | JP M,nn | SET 7,D |
| 251 | FB | EI | SET 7,E |
| 252 | FC | CALL M,nn | SET 7,H |
| 253 | FD | | SET 7,L |
| 254 | FE | CP n | SET 7,(HL) |
| 255 | FF | RST 38H | SET 7,A |

# 2. A Z80-as utasítások kódjai

A táblázat a kódokat hexadecimális alakban adja meg. Az egyéb jelölések:

— n: egybájtos szám (0—255)

— d: előjeles (kettes komplemens) értéknek tekintett egybájtos szám (−128 és +127 között) a relatív ugrásokhoz és az indexeléshez

— nn: kétbájtos szám (0—65535); a kód után előbb az alsó bájtot (LO), majd a felső bájtot (HI) kell megadni

| Mnemonik | Kód |
|---|---|
| ADC A,(HL) | 8E |
| ADC A,(IX+d) | DD 8E d |
| ADC A,(IY+d) | FD 8E d |
| ADC A,A | 8F |
| ADC A,B | 88 |
| ADC A,C | 89 |
| ADC A,D | 8A |
| ADC A,E | 8B |
| ADC A,H | 8C |
| ADC A,L | 8D |
| ADC A,n | CE n |
| ADC HL,BC | ED 4A |
| ADC HL,DE | ED 5A |
| ADC HL,HL | ED 6A |
| ADC HL,SP | ED 7A |
| ADD A,(HL) | 86 |
| ADD A,(IX+d) | DD 86 d |
| ADD A,(IY+d) | FD 86 d |
| ADD A,A | 87 |
| ADD A,B | 80 |
| ADD A,C | 81 |
| ADD A,D | 82 |
| ADD A,E | 83 |
| ADD A,H | 84 |
| ADD A,L | 85 |
| ADD A,n | C6 n |
| ADD HL,BC | 09 |
| ADD HL,DE | 19 |
| ADD HL,HL | 29 |
| ADD HL,SP | 39 |
| ADD IX,BC | DD 09 |
| ADD IX,DE | DD 19 |
| ADD IX,IX | DD 29 |
| ADD IX,SP | DD 39 |
| ADD IY,BC | FD 09 |
| ADD IY,DE | FD 19 |
| ADD IY,IY | FD 29 |
| ADD IY,SP | FD 39 |
| AND (HL) | A6 |
| AND (IX+d) | DD A6 d |
| AND (IY+d) | FD A6 d |
| AND A | A7 |
| AND B | A0 |
| AND C | A1 |
| AND D | A2 |
| AND E | A3 |
| AND H | A4 |
| AND L | A5 |
| AND n | E6 n |

| Mnemonik | Kód |
|---|---|
| BIT 0,(HL) | CB 46 |
| BIT 0,(IX+d) | DD CB d 46 |
| BIT 0,(IY+d) | FD CB d 46 |
| BIT 0,A | CB 47 |
| BIT 0,B | CB 40 |
| BIT 0,C | CB 41 |
| BIT 0,D | CB 42 |
| BIT 0,E | CB 43 |
| BIT 0,H | CB 44 |
| BIT 0,L | CB 45 |
| BIT 1,(HL) | CB 4E |
| BIT 1,(IX+d) | DD CB d 4E |
| BIT 1,(IY+d) | FD CB d 4E |
| BIT 1,A | CB 4F |
| BIT 1,B | CB 48 |
| BIT 1,C | CB 49 |
| BIT 1,D | CB 4A |
| BIT 1,E | CB 4B |
| BIT 1,H | CB 4C |
| BIT 1,L | CB 4D |
| BIT 2,(HL) | CB 56 |
| BIT 2,(IX+d) | DD CB d 56 |
| BIT 2,(IY+d) | FD CB d 56 |
| BIT 2,A | CB 57 |
| BIT 2,B | CB 50 |
| BIT 2,C | CB 51 |
| BIT 2,D | CB 52 |
| BIT 2,E | CB 53 |
| BIT 2,H | CB 54 |
| BIT 2,L | CB 55 |
| BIT 3,(HL) | CB 5E |
| BIT 3,(IX+d) | DD CB d 5E |
| BIT 3,(IY+d) | FD CB d 5E |
| BIT 3,A | CB 5F |
| BIT 3,B | CB 58 |
| BIT 3,C | CB 59 |
| BIT 3,D | CB 5A |
| BIT 3,E | CB 5B |
| BIT 3,H | CB 5C |
| BIT 3,L | CB 5D |
| BIT 4,(HL) | CB 66 |
| BIT 4,(IX+d) | DD CB d 66 |
| BIT 4,(IY+d) | FD CB d 66 |
| BIT 4,A | CB 67 |
| BIT 4,B | CB 60 |
| BIT 4,C | CB 61 |
| BIT 4,D | CB 62 |
| BIT 4,E | CB 63 |
| BIT 4,H | CB 64 |
| BIT 4,L | CB 65 |

| Mnemonik | Kód |
|---|---|
| BIT 5,(HL) | CB 6E |
| BIT 5,(IX+d) | DD CB d 6E |
| BIT 5,(IY+d) | FD CB d 6E |
| BIT 5,A | CB 6F |
| BIT 5,B | CB 68 |
| BIT 5,C | CB 69 |
| BIT 5,D | CB 6A |
| BIT 5,E | CB 6B |
| BIT 5,H | CB 6C |
| BIT 5,L | CB 6D |
| BIT 6,(HL) | CB 76 |
| BIT 6,(IX+d) | DD CB d 76 |
| BIT 6,(IY+d) | FD CB d 76 |
| BIT 6,A | CB 77 |
| BIT 6,B | CB 70 |
| BIT 6,C | CB 71 |
| BIT 6,D | CB 72 |
| BIT 6,E | CB 73 |
| BIT 6,H | CB 74 |
| BIT 6,L | CB 75 |
| BIT 7,(HL) | CB 7E |
| BIT 7,(IX+d) | DD CB d 7E |
| BIT 7,(IY+d) | FD CB d 7E |
| BIT 7,A | CB 7F |
| BIT 7,B | CB 78 |
| BIT 7,C | CB 79 |
| BIT 7,D | CB 7A |
| BIT 7,E | CB 7B |
| BIT 7,H | CB 7C |
| BIT 7,L | CB 7D |
| CALL nn | CD lo hi |
| CALL C,nn | DC lo hi |
| CALL NC,nn | D4 lo hi |
| CALL Z,nn | CC lo hi |
| CALL NZ,nn | C4 lo hi |
| CALL M,nn | FC lo hi |
| CALL P,nn | F4 lo hi |
| CALL PE,nn | EC lo hi |
| CALL PO,nn | E4 lo hi |
| CCF | 3F |
| CP (HL) | BE |
| CP (IX+d) | DD BE d |
| CP (IY+d) | FD BE d |
| CP A | BF |
| CP B | B8 |
| CP C | B9 |
| CP D | BA |
| CP E | BB |
| CP H | BC |
| CP L | BD |

| Mnemonik | Kód |
|---|---|
| CP n | FE n |
| CPD | ED A9 |
| CPDR | ED B9 |
| CPI | ED A1 |
| CPIR | ED B1 |
| CPL | 2F |
| DAA | 27 |
| DEC (HL) | 35 |
| DEC (IX+d) | DD 35 d |
| DEC (IY+d) | FD 35 d |
| DEC A | 3D |
| DEC B | 05 |
| DEC BC | 0B |
| DEC C | 0D |
| DEC D | 15 |
| DEC DE | 1B |
| DEC E | 1D |
| DEC H | 25 |
| DEC HL | 2B |
| DEC IX | DD 2B |
| DEC IY | FD 2B |
| DEC L | 2D |
| DEC SP | 3B |
| DI | F3 |
| DJNZ d | 10 d |
| EI | FB |
| EX (SP),HL | E3 |
| EX (SP),IX | DD E3 |
| EX (SP),IY | FD E3 |
| EX AF,AF' | 08 |
| EX DE,HL | EB |
| EXX | D9 |
| IM 0 | ED 46 |
| IM 1 | ED 56 |
| IM 2 | ED 5E |
| IN A,(C) | ED 78 |
| IN A,(n) | DB n |
| IN B,(C) | ED 40 |
| IN C,(C) | ED 48 |
| IN D,(C) | ED 50 |
| IN E,(C) | ED 58 |
| IN H,(C) | ED 60 |
| IN L,(C) | ED 68 |
| INC (HL) | 34 |
| INC (IX+d) | DD 34 d |
| INC (IY+d) | FD 34 d |
| INC A | 3C |
| INC B | 04 |
| INC BC | 03 |
| INC C | 0C |

| Mnemonik | Kód |
|---|---|
| INC D | 14 |
| INC DE | 13 |
| INC E | 1C |
| INC H | 24 |
| INC HL | 23 |
| INC IX | DD 23 |
| INC IY | FD 23 |
| INC L | 2C |
| INC SP | 33 |
| IND | ED AA |
| INDR | ED BA |
| INI | ED A2 |
| INIR | ED B2 |
| JP nn | C3 lo hi |
| JP (HL) | E9 |
| JP (IX) | DD E9 |
| JP (IY) | FD E9 |
| JP C,nn | DA lo hi |
| JP NC,nn | D2 lo hi |
| JP Z,nn | CA lo hi |
| JP NZ,nn | C2 lo hi |
| JP M,nn | FA lo hi |
| JP P,nn | F2 lo hi |
| JP PE,nn | EA lo hi |
| JP PO,nn | E2 lo hi |
| JR d | 18 d |
| JR C,d | 38 d |
| JR NC,d | 30 d |
| JR Z,d | 28 d |
| LD NZ,d | 20 d |
| LD (BC),A | 02 |
| LD (DE),A | 12 |
| LD (HL),A | 77 |
| LD (HL),B | 70 |
| LD (HL),C | 71 |
| LD (HL),D | 72 |
| LD (HL),E | 73 |
| LD (HL),H | 74 |
| LD (HL),L | 75 |
| LD (HL),n | 36 n |
| LD (IX+d),A | DD 77 d |
| LD (IX+d),B | DD 70 d |
| LD (IX+d),C | DD 71 d |
| LD (IX+d),D | DD 72 d |
| LD (IX+d),E | DD 73 d |
| LD (IX+d),H | DD 74 d |
| LD (IX+d),L | DD 75 d |
| LD (IX+d),n | DD 36 d n |

| Mnemonik | Kód |
|---|---|
| LD (IY+d),A | FD 77 d |
| LD (IY+d),B | FD 70 d |
| LD (IY+d),C | FD 71 d |
| LD (IY+d),D | FD 72 d |
| LD (IY+d),E | FD 73 d |
| LD (IY+d),H | FD 74 d |
| LD (IY+d),L | FD 75 d |
| LD (IY+d),n | FD 36 d n |
| LD (nn),A | 32 lo hi |
| LD (nn),BC | ED 43 lo hi |
| LD (nn),DE | ED 53 lo hi |
| LD (nn),HL | 22 lo hi |
| LD (nn),IX | DD 22 lo hi |
| LD (nn),IY | FD 22 lo hi |
| LD (nn),SP | ED 73 lo hi |
| LD A,(BC) | 0A |
| LD A,(DE) | 1A |
| LD A,(HL) | 7E |
| LD A,(IX+d) | DD 7E d |
| LD A,(IY+d) | FD 7E d |
| LD A,(nn) | 3A lo hi |
| LD A,A | 7F |
| LD A,B | 78 |
| LD A,C | 79 |
| LD A,D | 7A |
| LD A,E | 7B |
| LD A,H | 7C |
| LD A,I | ED 57 |
| LD A,L | 7D |
| LD A,n | 3E n |
| LD A,R | ED 5F |
| LD B,(HL) | 46 |
| LD B,(IX+d) | DD 46 d |
| LD B,(IY+d) | FD 46 d |
| LD B,A | 47 |
| LD B,B | 40 |
| LD B,C | 41 |
| LD B,D | 42 |
| LD B,E | 43 |
| LD B,H | 44 |
| LD B,L | 45 |
| LD B,n | 06 n |
| LD BC,(nn) | ED 48 lo hi |
| LD BC,nn | 01 lo hi |
| LD C,(HL) | 4E |
| LD C,(IX+d) | DD 4E d |
| LD C,(IY+d) | FD 4E d |
| LD C,A | 4F |

| Mnemonik | Kód |
|---|---|
| LD C,B | 48 |
| LD C,C | 49 |
| LD C,D | 4A |
| LD C,E | 4B |
| LD C,H | 4C |
| LD C,L | 4D |
| LD C,n | 0E n |
| LD D,(HL) | 56 |
| LD D,(IX+d) | DD 56 d |
| LD D,(IY+d) | FD 56 d |
| LD D,A | 57 |
| LD D,B | 50 |
| LD D,C | 51 |
| LD D,D | 52 |
| LD D,E | 53 |
| LD D,H | 54 |
| LD D,L | 55 |
| LD D,n | 16 n |
| LD DE,(nn) | ED 5B lo hi |
| LD DE,nn | 11 lo hi |
| LD E,(HL) | 5E |
| LD E,(IX+d) | DD 5E d |
| LD E,(IY+d) | FD 5E d |
| LD E,A | 5F |
| LD E,B | 58 |
| LD E,C | 59 |
| LD E,D | 5A |
| LD E,E | 5B |
| LD E,H | 5C |
| LD E,L | 5D |
| LD E,n | 1E n |
| LD H,(HL) | 66 |
| LD H,(IX+d) | DD 66 d |
| LD H,(IY+d) | FD 66 d |
| LD H,A | 67 |
| LD H,B | 60 |
| LD H,C | 61 |
| LD H,D | 62 |
| LD H,E | 63 |
| LD H,H | 64 |
| LD H,L | 65 |
| LD H,n | 26 n |
| LD HL,(nn) | 2A lo hi |
| LD HL,nn | 21 lo hi |
| LD I,A | ED 47 |
| LD IX,(nn) | DD 2A lo hi |
| LD IX,nn | DD 21 lo hi |
| LD IY,(nn) | FD 2A lo hi |
| LD IY,nn | FD 21 lo hi |

| Mnemonik | Kód |
|---|---|
| LD L,(HL) | 6E |
| LD L,(IX+d) | DD 6E d |
| LD L,(IY+d) | FD 6E d |
| LD L,A | 6F |
| LD L,B | 68 |
| LD L,C | 69 |
| LD L,D | 6A |
| LD L,E | 6B |
| LD L,H | 6C |
| LD L,L | 6D |
| LD L,n | 2E n |
| LD SP,(nn) | ED 7B lo hi |
| LD SP,HL | F9 |
| LD SP,IX | DD F9 |
| LD SP,IY | FD F9 |
| LD SP,nn | 31 lo hi |
| LDD | ED A8 |
| LDDR | ED B8 |
| LDI | ED A0 |
| LDIR | ED B0 |
| NEG | ED 44 |
| NOP | 00 |
| OR (HL) | B6 |
| OR (IX+d) | DD B6 d |
| OR (IY+d) | FD B6 d |
| OR A | B7 |
| OR B | B0 |
| OR C | B1 |
| OR D | B2 |
| OR E | B3 |
| OR H | B4 |
| OR L | B5 |
| OR n | F6 n |
| OTDR | ED BB |
| OTIR | ED B3 |
| OUT (C),A | ED 79 |
| OUT (C),B | ED 41 |
| OUT (C),C | ED 49 |
| OUT (C),D | ED 51 |
| OUT (C),E | ED 59 |
| OUT (C),H | ED 61 |
| OUT (C),L | ED 69 |
| OUT (n),A | D3 n |
| OUTD | ED AB |
| OUTI | ED A3 |
| POP AF | F1 |
| POP BC | C1 |
| POP DE | D1 |
| POP HL | E1 |

| Mnemonik | Kód |
|---|---|
| POP IX | DD E1 |
| POP IY | FD E1 |
| PUSH AF | F5 |
| PUSH BC | C5 |
| PUSH DE | D5 |
| PUSH HL | E5 |
| PUSH IX | DD E5 |
| PUSH IY | FD E5 |
| RES 0,(HL) | CB 86 |
| RES 0,(IX+d) | DD CB d 86 |
| RES 0,(IY+d) | FD CB d 86 |
| RES 0,A | CB 87 |
| RES 0,B | CB 80 |
| RES 0,C | CB 81 |
| RES 0,D | CB 82 |
| RES 0,E | CB 83 |
| RES 0,H | CB 84 |
| RES 0,L | CB 85 |
| RES 1,(HL) | CB 8E |
| RES 1,(IX+d) | DD CB d 8E |
| RES 1,(IY+d) | FD CB d 8E |
| RES 1,A | CB 8F |
| RES 1,B | CB 88 |
| RES 1,C | CB 89 |
| RES 1,D | CB 8A |
| RES 1,E | CB 8B |
| RES 1,H | CB 8C |
| RES 1,L | CB 8D |
| RES 2,(HL) | CB 96 |
| RES 2,(IX+d) | DD CB d 96 |
| RES 2,(IY+d) | FD CB d 96 |
| RES 2,A | CB 97 |
| RES 2,B | CB 90 |
| RES 2,C | CB 91 |
| RES 2,D | CB 92 |
| RES 2,E | CB 93 |
| RES 2,H | CB 94 |
| RES 2,L | CB 95 |
| RES 3,(HL) | CB 9E |
| RES 3,(IX+d) | DD CB d 9E |
| RES 3,(IY+d) | FD CB d 9E |
| RES 3,A | CB 9F |
| RES 3,B | CB 98 |
| RES 3,C | CB 99 |
| RES 3,D | CB 9A |
| RES 3,E | CB 9B |
| RES 3,H | CB 9C |
| RES 3,L | CB 9D |

| Mnemonik | Kód |
|---|---|
| RES 4,(HL) | CB A6 |
| RES 4,(IX+d) | DD CB d A6 |
| RES 4,(IY+d) | FD CB d A6 |
| RES 4,A | CB A7 |
| RES 4,B | CB A0 |
| RES 4,C | CB A1 |
| RES 4,D | CB A2 |
| RES 4,E | CB A3 |
| RES 4,H | CB A4 |
| RES 4,L | CB A5 |
| RES 5,(HL) | CB AE |
| RES 5,(IX+d) | DD CB d AE |
| RES 5,(IY+d) | FD CB d AE |
| RES 5,A | CB AF |
| RES 5,B | CB A8 |
| RES 5,C | CB A9 |
| RES 5,D | CB AA |
| RES 5,E | CB AB |
| RES 5,H | CB AC |
| RES 5,L | CB AD |
| RES 6,(HL) | CB B6 |
| RES 6,(IX+d) | DD CB d B6 |
| RES 6,(IY+d) | FD CB d B6 |
| RES 6,A | CB B7 |
| RES 6,B | CB B0 |
| RES 6,C | CB B1 |
| RES 6,D | CB B2 |
| RES 6,E | CB B3 |
| RES 6,H | CB B4 |
| RES 6,L | CB B5 |
| RES 7,(HL) | CB BE |
| RES 7,(IX+d) | DD CB d BE |
| RES 7,(IY+d) | FD CB d BE |
| RES 7,A | CB BF |
| RES 7,B | CB B8 |
| RES 7,C | CB B9 |
| RES 7,D | CB BA |
| RES 7,E | CB BB |
| RES 7,H | CB BC |
| RES 7,L | CB BD |
| RET | C9 |
| RET C | D8 |
| RET NC | D0 |
| RET Z | C8 |
| RET NZ | C0 |
| RET M | F8 |
| RET P | F0 |
| RET PE | E8 |

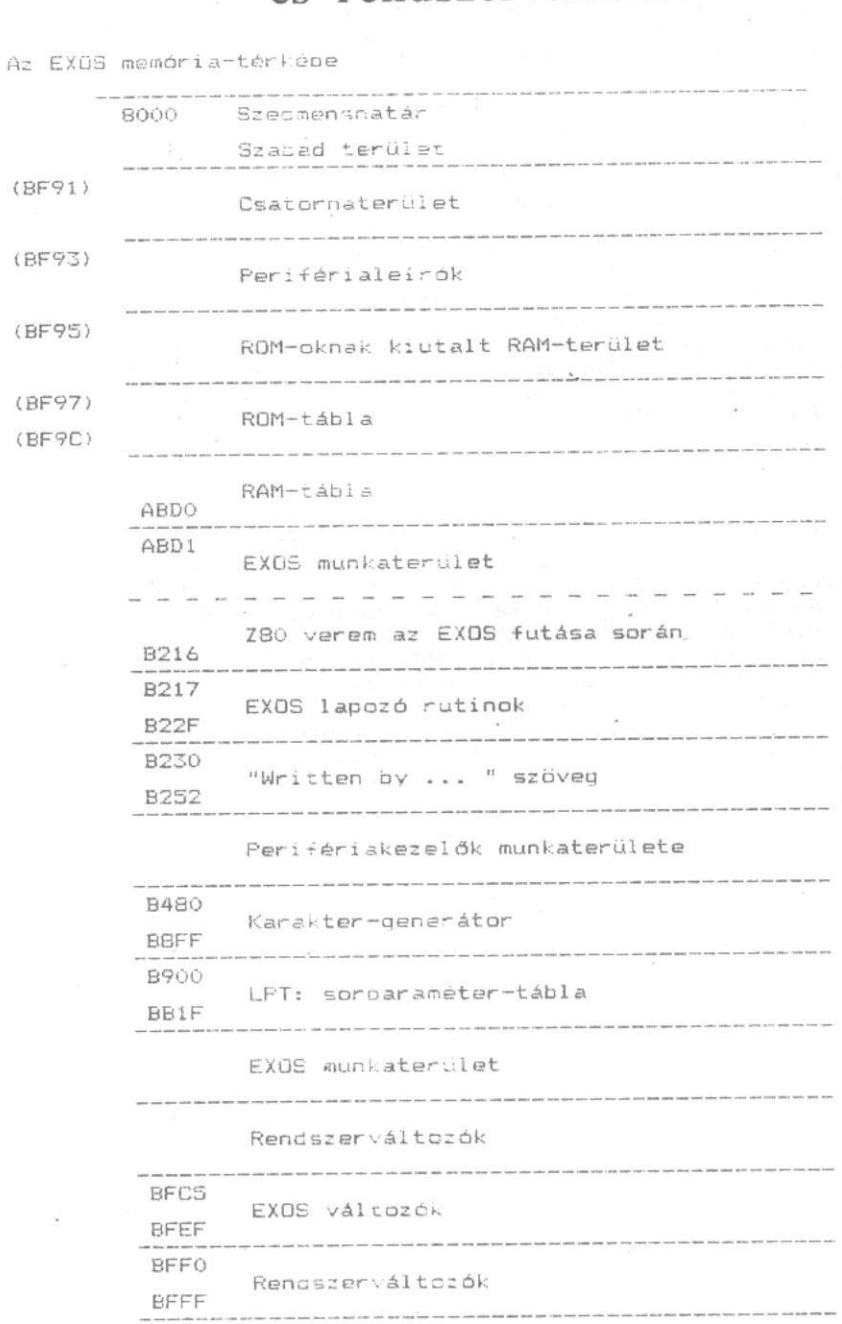
| Mnemonik | Kód |
|---|---|
| RET PO | E0 |
| RETI | ED 4D |
| RETN | ED 45 |
| RL (HL) | CB 16 |
| RL (IX+d) | DD CB d 16 |
| RL (IY+d) | FD CB d 16 |
| RL A | CB 17 |
| RL B | CB 10 |
| RL C | CB 11 |
| RL D | CB 12 |
| RL E | CB 13 |
| RL H | CB 14 |
| RL L | CB 15 |
| RLA | 17 |
| RLC (HL) | CB 06 |
| RLC (IX+d) | DD CB d 06 |
| RLC (IY+d) | FD CB d 06 |
| RLC A | CB 07 |
| RLC B | CB 00 |
| RLC C | CB 01 |
| RLC D | CB 02 |
| RLC E | CB 03 |
| RLC H | CB 04 |
| RLC L | CB 05 |
| RLCA | 07 |
| RLD | ED 6F |
| RR (HL) | CB 1E |
| RR (IX+d) | DD CB d 1E |
| RR (IY+d) | FD CB d 1E |
| RR A | CB 1F |
| RR B | CB 18 |
| RR C | CB 19 |
| RR D | CB 1A |
| RR E | CB 1B |
| RR H | CB 1C |
| RR L | CB 1D |
| RRA | 1F |
| RRC (HL) | CB 0E |
| RRC (IX+d) | DD CB d 0E |
| RRC (IY+d) | FD CB d 0E |
| RRC A | CB 0F |
| RRC B | CB 08 |
| RRC C | CB 09 |
| RRC D | CB 0A |
| RRC E | CB 0B |
| RRC H | CB 0C |
| RRC L | CB 0D |
| RRCA | 0F |
| RRD | ED 67 |

| Mnemonik | Kód |
|---|---|
| RST 00 | C7 |
| RST 08 | CF |
| RST 10 | D7 |
| RST 18 | DF |
| RST 20 | E7 |
| RST 28 | EF |
| RST 30 | F7 |
| RST 38 | FF |
| SBC A,(HL) | 9E |
| SBC A,(IX+d) | DD 9E d |
| SBC A,(IY+d) | FD 9E d |
| SBC A,A | 9F |
| SBC A,B | 98 |
| SBC A,C | 99 |
| SBC A,D | 9A |
| SBC A,E | 9B |
| SBC A,H | 9C |
| SBC A,L | 9D |
| SBC A,n | DE n |
| SBC HL,BC | ED 42 |
| SBC HL,DE | ED 52 |
| SBC HL,HL | ED 62 |
| SBC HL,SP | ED 72 |
| SCF | 37 |
| SET 0,(HL) | CB C6 |
| SET 0,(IX+d) | CB d C6 |
| SET 0,(IY+d) | CB d C6 |
| SET 0,A | CB C7 |
| SET 0,B | CB C0 |
| SET 0,C | CB C1 |
| SET 0,D | CB C2 |
| SET 0,E | CB C3 |
| SET 0,H | CB C4 |
| SET 0,L | CB C5 |
| SET 1,(HL) | CB CE |
| SET 1,(IX+d) | CB d CE |
| SET 1,(IY+d) | CB d CE |
| SET 1,A | CB CF |
| SET 1,B | CB C8 |
| SET 1,C | CB C9 |
| SET 1,D | CB CA |
| SET 1,E | CB CB |
| SET 1,H | CB CC |
| SET 1,L | CB CD |
| SET 2,(HL) | CB D6 |
| SET 2,(IX+d) | CB d D6 |
| SET 2,(IY+d) | CB d D6 |
| SET 2,A | CB D7 |
| SET 2,B | CB D0 |

| Mnemonik | Kód |
|---|---|
| SET 2,C | CB D1 |
| SET 2,D | CB D2 |
| SET 2,E | CB D3 |
| SET 2,H | CB D4 |
| SET 2,L | CB D5 |
| SET 3,(HL) | CB DE |
| SET 3,(IX+d) | DD CB d DE |
| SET 3,(IY+d) | FD CB d DE |
| SET 3,A | CB DF |
| SET 3,B | CB D8 |
| SET 3,C | CB D9 |
| SET 3,D | CB DA |
| SET 3,E | CB DB |
| SET 3,H | CB DC |
| SET 3,L | CB DD |
| SET 4,(HL) | CB E6 |
| SET 4,(IX+d) | DD CB d E6 |
| SET 4,(IY+d) | FD CB d E6 |
| SET 4,A | CB E7 |
| SET 4,B | CB E0 |
| SET 4,C | CB E1 |
| SET 4,D | CB E2 |
| SET 4,E | CB E3 |
| SET 4,H | CB E4 |
| SET 4,L | CB E5 |
| SET 5,(HL) | CB EE |
| SET 5,(IX+d) | DD CB d EE |
| SET 5,(IY+d) | FD CB d EE |
| SET 5,A | CB EF |
| SET 5,B | CB E8 |
| SET 5,C | CB E9 |
| SET 5,D | CB EA |
| SET 5,E | CB EB |
| SET 5,H | CB EC |
| SET 5,L | CB ED |
| SET 6,(HL) | CB F6 |
| SET 6,(IX+d) | DD CB d F6 |
| SET 6,(IY+d) | FD CB d F6 |
| SET 6,A | CB F7 |
| SET 6,B | CB F0 |
| SET 6,C | CB F1 |
| SET 6,D | CB F2 |
| SET 6,E | CB F3 |
| SET 6,H | CB F4 |
| SET 6,L | CB F5 |
| SET 7,(HL) | CB FE |
| SET 7,(IX+d) | DD CB d FE |
| SET 7,(IY+d) | FD CB d FE |
| SET 7,A | CB FF |
| SET 7,B | CB F8 |
| SET 7,C | CB F9 |
| SET 7,D | CB FA |
| SET 7,E | CB FB |
| SET 7,H | CB FC |
| SET 7,L | CB FD |

| Mnemonik | Kód |
|---|---|
| SLA (HL) | CB 26 |
| SLA (IX+d) | DD CB d 26 |
| SLA (IY+d) | FD CB d 26 |
| SLA A | CB 27 |
| SLA B | CB 20 |
| SLA C | CB 21 |
| SLA D | CB 22 |
| SLA E | CB 23 |
| SLA H | CB 24 |
| SLA L | CB 25 |
| SRA (HL) | CB 2E |
| SRA (IX+d) | DD CB d 2E |
| SRA (IY+d) | FD CB d 2E |
| SRA A | CB 2F |
| SRA B | CB 28 |
| SRA C | CB 29 |
| SRA D | CB 2A |
| SRA E | CB 2B |
| SRA H | CB 2C |
| SRA L | CB 2D |
| SRL (HL) | CB 3E |
| SRL (IX+d) | DD CB d 3E |
| SRL (IY+d) | FD CB d 3E |
| SRL A | CB 3F |
| SRL B | CB 38 |
| SRL C | CB 39 |
| SRL D | CB 3A |
| SRL E | CB 3B |
| SRL H | CB 3C |
| SRL L | CB 3D |
| SUB (HL) | 96 |
| SUB (IX+d) | DD 96 |
| SUB (IY+d) | FD 96 |
| SUB A | 97 |
| SUB B | 90 |
| SUB C | 91 |
| SUB D | 92 |
| SUB E | 93 |
| SUB H | 94 |
| SUB L | 95 |
| SUB n | D6 n |
| XOR (HL) | AE |
| XOR (IX+d) | DD AE |
| XOR (IY+d) | FD AE |
| XOR A | AF |
| XOR B | A8 |
| XOR C | A9 |
| XOR D | AA |
| XOR E | AB |
| XOR H | AC |
| XOR L | AD |
| XOR n | EE n |

# 3. Az EXOS memóriatérképe és rendszerváltozói



Rendszerváltozók és EXOS változók

BF8F: Felhasználói határ a megosztott szegmensben
BF91: EXOS határ (0000 és 3FFF közötti érték: a szabad bájtok száma a rendszerszegmensben)
BF93: A periférialeírók területének alsó határa
BF95: A ROM-oknak kiutalt RAM alsó határa
BF97: A ROM-tábla alsó határa
BF9A: A RAM-táblában az EXOS határt tartalmazó szegmensre mutat
BF9C: A ROM-tábla felső határa
BF9E: A megosztott szegmens száma (0, ha nincs ilyen)
BF9F: A szabad szegmensek száma
BFA0: A felhasználónak kiutalt szegmensek száma
BFA1: A perifériáknak kiutalt szegmensek száma
BFA2: A rendszerszegmensek száma
BFA3: A hibátlan RAM-szegmensek száma
BFA4: A hibás RAM-szegmensek száma
BFA5: Munkaterület az aktuális csatornák kereséséhez
BFBA: A csatornalánc első pointere (báziscím: BFBD)
BFBD: A periférialánc első pointere (báziscím: (BFC0)
BFC0: A bővítőlánc első pointere (báziscím: BFC3)

Az EXOS változók tárolási címe: BFC5 + a változó sorszáma (0 - 39)

BFED: USR_ISR: felhasználói megszakítási rutin címe
BFEF: CRDISP_FLAG: bejelenkezési üzenet engedélyezése
BFF0: SECOND_COUNTER: másodpercszámláló
BFF2: FLAG_SOF_IRQ: szoftver megszakításkérés jelző
BFF3: PORTB5: a B5H port aktuális értéke
BFF4: LP_POINTER: a sorparaméter-tábla kezdőcíme
BFF6: ST_POINTER: az állapotsor kezdőcíme
BFF8: RST_ADDR: felhasználói melegindítási cím
BFFA: STACK_LIMIT: veremhatár a belső ellenőrzéshez
BFFC: USR_P0: az EXOS hívás előtti LAP-konfiguráció
BFFD: USR_P1
BFFE: USR_P2
BFFF: USR_P3

# 4. EXOS hibakódok és üzenetek

Az EXOS 28 funkcióhívás eredménye UK és BRD módban

```
255 (FF)  Invalid EXOS function code
254 (FE)  EXOS function call not allowed
253 (FD)  Invalid EXOS string
252 (FC)  EXOS stack overflow
251 (FB)  Channel does not exist
250 (FA)  Device does not exist
249 (F9)  Channel exists
248 (F8)
247 (F7)  Insufficient memory
246 (F6)  Insufficient video memory
245 (F5)
244 (F4)
243 (F3)
242 (F2)  Unknown EXOS variable number
241 (F1)  Invalid device descriptor
240 (F0)  Unrecognised command string
239 (EF)  Invalid file header
238 (EE)  Unknown module type
237 (ED)  Invalid relocatable module
236 (EC)
235 (EB)  Invalid date or time value
234 (EA)  Invalid special function call
233 (E9)  Device in use
232 (E8)  Invalid unit number
231 (E7)  Call not supported by this device
230 (E6)  Invalid escape sequence
229 (E5)  STOP key pressed
228 (E4)  End of file
227 (E3)  Protection violation
226 (E2)  Function key string too long
225 (E1)  Envelope too big
224 (E0)  Envelope storage full
223 (DF)  Sound queue full
222 (DE)  Invalid video page size
221 (DD)  Invalid video mode
220 (DC)  Invalid display parameters
219 (DB)
218 (DA)  Invalid row number to scroll
217 (D9)  Invalid cursor coordinates
216 (D8)  Invalid beam position
215 (D7)  Cannot use both serial and network
214 (D6)  Network address not set
213 (D5)  Network link exists
212 (D4)  Editor video channel error
211 (D3)  Editor keyboard channel error
210 (D2)  Editor load file error
209 (D1)  Editor load file too big
208 (D0)  Cassette CRC error
207 (CF)  Copyright INTELLIGENT SOFTWARE LTD
```

255 (FF) ungültiger EXOS Funktions-Code
254 (FE) verbotener EXOS-Funktionsaufruf
253 (FD) ungültiger EXOS-String
252 (FC) EXOS-Stapel-Speicher Überlauf
251 (FB) Kanal nicht vorhanden
250 (FA) Gerät nicht vorhanden
249 (F9) Kanal schon vorhanden
248 (F8)
247 (F7) Speicherplatz zu klein
246 (F6) Video-Speicher-Platz zu klein
245 (F5)
244 (F4)
243 (F3)
242 (F2) unbekannte EXOS-Variablen-Number
241 (F1) ungültige Gerät-Bezeichnung
240 (F0) unbekannter Befehls-String
239 (EF) ungültiger Datei-Anfang
238 (EE) unbekannte Modulart
237 (ED) Modul nicht verlegbar
236 (EC) Ende der Modul-Datei
235 (EB) Datum oder Uhrzeit außerhalb des erlaubten Bereichs
234 (EA) ungültiger Aufruf einer Spezialfunktion
233 (E9) Gerät wird schon benutzt
232 (E8) Geräte-Nummer außerhalb des erlaubten Bereichs
231 (E7) Aufruf dieses Gerätes nicht möglich
230 (E6) ungültiges Steuerzeichen
229 (E5) STOP-Taste gedrückt
228 (E4) Datei-Ende
227 (E3) Schutz-Verletzung
226 (E2) Funktionstasten-String außerhalb des erlaubten Bereichs
225 (E1) Hüllkurve außerhalb des erlaubten Bereichs
224 (E0) Hüllkurvenstapel voll
223 (DF) Ton-Stapel voll
222 (DE) Bildschirm-Format außerhalb des erlaubten Bereichs
221 (DD) Bildschirm-Modus außerhalb des erlaubten Bereichs
220 (DC) Display-Parameter außerhalb des erlaubten Bereichs
219 (DB) ungültige Bildschirmseiten-Datei
218 (DA) zu scrollender Zeilenbereich außerhalb des erlaubten Bereichs
217 (D9) Cursor-Koordinaten außerhalb des erlaubten Bereichs
216 (D8) Grafik-Cursor-Koordinaten außerhalb des erlaubten Bereichs
215 (D7) Schnittstelle schon belegt
214 (D6) Netzwerk-Adresse nicht gesetzt
213 (D5) Netzwerk-Verbindung schon vorhanden
212 (D4) Editor: Video-Kanal-Fehler
211 (D3) Editor: Tastatur-Kanal-Fehler
210 (D2) Fehler in der zu ladenden Datei
209 (D1) Editor: Überlänge der zu ladenden Datei
208 (D0) Kassetten-Lese-Fehler
207 (CF) Copyright INTELLIGENT SOFTWARE LTD

# 5. A perifériakezelők belépési pontjai

Leíró:

FF/6661: 74 65 FF 00 20 00 8F 5F 04 00 08 KEYBOARD

Belépési címek (BRD szegmens):

| | | |
|---|---|---|
| 57282 | DFC2 | 0: Megszakítás |
| 58122 | E30A | 1: Csatorna megnyitása |
| 58122 | E30A | 2: Csatorna létrehozása |
| 57270 | DFB6 | 3: Csatorna lezárása |
| 57270 | DFB6 | 4: Csatorna megszüntetése |
| 58189 | E34D | 5: Karakter olvasása |
| 58508 | E48C | 6: Blokk olvasása |
| 58481 | E471 | 7: Karakter írása |
| 58481 | E471 | 8: Blokk írása |
| 58161 | E331 | 9: Csatornaállapot olvasása |
| 58481 | E471 | 10: Csatornaállapot beállítása |
| 58251 | E38B | 11: Különleges funkció |
| 57259 | DFAB | 12: Inicializálás |
| 57281 | DFC1 | 13: Puffermozgatás |

Leíró:

FF/6674: 85 65 FF 00 08 00 36 4D 04 00 06 EDITOR

Belépési címek (BRD szegmens):

| | | |
|---|---|---|
| 49895 | C2E7 | 0: Megszakítás |
| 52591 | CD6F | 1: Csatorna megnyitása |
| 52591 | CD6F | 2: Csatorna létrehozása |
| 52562 | CD52 | 3: Csatorna lezárása |
| 52562 | CD52 | 4: Csatorna megszüntetése |
| 53185 | CFC1 | 5: Karakter olvasása |
| 53193 | CFC9 | 6: Blokk olvasása |
| 53144 | CF98 | 7: Karakter írása |
| 53152 | CFA0 | 8: Blokk írása |
| 53109 | CF75 | 9: Csatornaállapot olvasása |
| 56755 | DDB3 | 10: Csatornaállapot beállítása |
| 52839 | CE67 | 11: Különleges funkció |
| 52565 | CD55 | 12: Inicializálás |
| 52567 | CD57 | 13: Puffermozgatás |

Leíró:

FF/6685: 93 88 FF 00 80 00 D3 6D 01 00 03 NET

Belépési címek ( 1 . szegmens):

| | | |
|---|---|---|
| 60959 | EE1F | 0: Megszakítás |
| 61285 | EF65 | 1: Csatorna megnyitása |
| 61285 | EF65 | 2: Csatorna létrehozása |
| 61445 | F005 | 3: Csatorna lezárása |
| 61445 | F005 | 4: Csatorna megszüntetése |
| 61515 | F04B | 5: Karakter olvasása |
| 61730 | F122 | 6: Blokk olvasása |
| 61742 | F12E | 7: Karakter írása |
| 61874 | F1B2 | 8: Blokk írása |
| 61886 | F1BE | 9: Csatornaállapot olvasása |
| 61913 | F1D9 | 10: Csatornaállapot beállítása |
| 61916 | F1DC | 11: Különleges funkció |
| 61984 | F220 | 12: Inicializálás |
| 61985 | F221 | 13: Puffermozgatás |

Leíró:

FF/6693: A4 66 FF 00 00 00 A7 6D 01 00 06 SERIAL

Belépési címek ( 1 . szegmens):

| | | |
|---|---|---|
| 61915 | F1DB | 0: Megszakítás |
| 61391 | EFCF | 1: Csatorna megnyitása |
| 61391 | EFCF | 2: Csatorna létrehozása |
| 61508 | F044 | 3: Csatorna lezárása |
| 61508 | F044 | 4: Csatorna megszüntetése |
| 61657 | F0D9 | 5: Karakter olvasása |
| 61736 | F128 | 6: Blokk olvasása |
| 61792 | F160 | 7: Karakter írása |
| 61880 | F1B8 | 8: Blokk írása |
| 61910 | F1D6 | 9: Csatornaállapot olvasása |
| 61913 | F1D9 | 10: Csatornaállapot beállítása |
| 61913 | F1D9 | 11: Különleges funkció |
| 61980 | F21C | 12: Inicializálás |
| 61915 | F1DB | 13: Puffermozgatás |

Leíró:

FF/66A4: B3 66 FF 00 00 00 BB 67 01 00 04 TAPE

Belépési címek ( 1 . szegmens):

| | | |
|---|---|---|
| 3141 | 0C45 | 0: Megszakítás |
| 59354 | E7DA | 1: Csatorna megnyitása |
| 59488 | E860 | 2: Csatorna létrehozása |
| 59634 | E8F2 | 3: Csatorna lezárása |
| 59634 | E8F2 | 4: Csatorna megszüntetése |
| 59664 | E910 | 5: Karakter olvasása |
| 59741 | E95D | 6: Blokk olvasása |
| 59751 | E967 | 7: Karakter írása |
| 59792 | E990 | 8: Blokk írása |
| 59802 | E99A | 9: Csatornaállapot olvasása |
| 59219 | E753 | 10: Csatornaállapot beállítása |
| 59351 | E7D7 | 11: Különleges funkció |
| 59871 | E9DF | 12: Inicializálás |
| 59821 | E9AD | 13: Puffermozgatás |

Leíró:

FF/6683: C5 66 FF 00 00 00 69 67 01 00 07 PRINTER

Belépési címek ( 1 . szegmens):

| | | |
|---|---|---|
| 3922 | 0F52 | 0: Megszakítás |
| 59272 | E788 | 1: Csatorna megnyitása |
| 59272 | E788 | 2: Csatorna létrehozása |
| 59306 | E7AA | 3: Csatorna lezárása |
| 59306 | E7AA | 4: Csatorna megszüntetése |
| 59219 | E753 | 5: Karakter olvasása |
| 59219 | E753 | 6: Blokk olvasása |
| 59278 | E78E | 7: Karakter írása |
| 58871 | E5F7 | 8: Blokk írása |
| 59219 | E753 | 9: Csatornaállapot olvasása |
| 59219 | E753 | 10: Csatornaállapot beállítása |
| 59269 | E785 | 11: Különleges funkció |
| 59307 | E7AB | 12: Inicializálás |
| 59307 | E7AB | 13: Puffermozgatás |

Leíró:

FF/66C5: D6 66 FF FF 00 00 82 70 00 00 06 EDITOR

Belépési címek ( 0 . szegmens):

| | | |
|---|---|---|
| 3666 | 0E52 | 0: Megszakítás |
| 61624 | F0B8 | 1: Csatorna megnyitása |
| 61624 | F0B8 | 2: Csatorna létrehozása |
| 61598 | F09E | 3: Csatorna lezárása |
| 61598 | F09E | 4: Csatorna megszüntetése |
| 62186 | F2EA | 5: Karakter olvasása |
| 62194 | F2F2 | 6: Blokk olvasása |
| 62145 | F2C1 | 7: Karakter írása |
| 62153 | F2C9 | 8: Blokk írása |
| 62117 | F2A5 | 9: Csatornaállapot olvasása |
| 49874 | C2D2 | 10: Csatornaállapot beállítása |
| 61859 | F1A3 | 11: Különleges funkció |
| 61601 | F0A1 | 12: Inicializálás |
| 61606 | F0A6 | 13: Puffermozgatás |

Leíró:

FF/66D6: E9 66 FF FF 20 00 55 70 00 00 08 KEYBOARD

Belépési címek ( 0 . szegmens):

| | | |
|---|---|---|
| 60687 | ED0F | 0: Megszakítás |
| 61303 | EF77 | 1: Csatorna megnyitása |
| 61303 | EF77 | 2: Csatorna létrehozása |
| 60682 | ED0A | 3: Csatorna lezárása |
| 60682 | ED0A | 4: Csatorna megszüntetése |
| 61349 | EFA5 | 5: Karakter olvasása |
| 53660 | D19C | 6: Blokk olvasása |
| 49874 | C2D2 | 7: Karakter írása |
| 49874 | C2D2 | 8: Blokk írása |
| 61332 | EF94 | 9: Csatornaállapot olvasása |
| 49874 | C2D2 | 10: Csatornaállapot beállítása |
| 61389 | EFCD | 11: Különleges funkció |
| 60682 | ED0A | 12: Inicializálás |
| 60686 | ED0E | 13: Puffermozgatás |

Leíró:

FF/66E9: F9 66 FF 00 20 00 7C 6B 00 00 05 SOUND

Belépési címek ( 0 . szegmens):

| | | |
|---|---|---|
| 60374 | EBD6 | 0: Megszakítás |
| 59464 | E848 | 1: Csatorna megnyitása |
| 59464 | E848 | 2: Csatorna létrehozása |
| 60410 | EBFA | 3: Csatorna lezárása |
| 60410 | EBFA | 4: Csatorna megszüntetése |
| 49874 | C2D2 | 5: Karakter olvasása |
| 49874 | C2D2 | 6: Blokk olvasása |
| 59532 | E88C | 7: Karakter írása |
| 53738 | D1EA | 8: Blokk írása |
| 49874 | C2D2 | 9: Csatornaállapot olvasása |
| 49874 | C2D2 | 10: Csatornaállapot beállítása |
| 60312 | EB98 | 11: Különleges funkció |
| 60410 | EBFA | 12: Inicializálás |
| 59514 | E87A | 13: Puffermozgatás |

Leíró:

FF/66F9: 00 00 00 00 20 01 81 57 00 00 05 VIDEO

Belépési címek ( 0 . szegmens):

| | | |
|---|---|---|
| 54077 | D33D | 0: Megszakítás |
| 54117 | D365 | 1: Csatorna megnyitása |
| 54117 | D365 | 2: Csatorna létrehozása |
| 54433 | D4A1 | 3: Csatorna lezárása |
| 54433 | D4A1 | 4: Csatorna megszüntetése |
| 54450 | D4B2 | 5: Karakter olvasása |
| 53656 | D198 | 6: Blokk olvasása |
| 54480 | D4D0 | 7: Karakter írása |
| 53734 | D1E6 | 8: Blokk írása |
| 54650 | D57A | 9: Csatornaállapot olvasása |
| 49874 | C2D2 | 10: Csatornaállapot beállítása |
| 54679 | D597 | 11: Különleges funkció |
| 53856 | D260 | 12: Inicializálás |
| 54866 | D652 | 13: Puffermozgatás |

# 6. A BASIC kulcsszavak paramétertáblázata

| token D | H | 1.cím | 2.cím | tp | kulcsszó |
|---|---|---|---|---|---|
| 0 | 00: | F8 CC | 61 D0 | 53 08 | ALLOCATE |
| 1 | 01: | 06 CE | 61 D0 | 53 03 | ASK |
| 2 | 02: | 23 C8 | 9C CA | 41 04 | AUTO |
| 3 | 03: | 34 CE | 61 D0 | 53 04 | CALL |
| 4 | 04: | 53 CE | 61 D0 | 53 07 | CAPTURE |
| 5 | 05: | AA DC | 61 D0 | 4E 04 | CASE |
| 6 | 06: | 5C CE | 61 D0 | 53 05 | CAUSE |
| 7 | 07: | 25 D4 | 44 D0 | 53 05 | CLEAR |
| 8 | 08: | E6 CF | 61 D0 | 53 05 | CLOSE |
| 9 | 09: | F0 CF | 61 D0 | 53 04 | CODE |
| 10 | 0A: | 4E CD | 61 D0 | 43 08 | CONTINUE |
| 11 | 0B: | 22 D0 | 61 D0 | 53 04 | COPY |
| 12 | 0C: | 9C CA | 9C CA | 02 04 | DATA |
| 13 | 0D: | 9C CA | 7F D0 | 42 03 | DEF |
| 14 | 0E: | E7 D9 | 9C CA | 4A 03 | DEF |
| 15 | 0F: | 3D C8 | 9C CA | 61 06 | DELETE |
| 16 | 10: | 04 EB | 61 D0 | 42 03 | DIM |
| 17 | 11: | 37 FC | 44 D0 | 53 07 | DISPLAY |
| 18 | 12: | B9 DC | 61 D0 | 4A 02 | DO |
| 19 | 13: | AC CE | 61 D0 | 53 05 | CHAIN |
| 20 | 14: | 97 C7 | 9C CA | 41 04 | EDIT |
| 21 | 15: | CD D9 | A3 D9 | 4E 04 | ELSE |
| 22 | 16: | CD D9 | 61 D0 | 4E 07 | ELSE IF |
| 23 | 17: | AD CF | A6 D0 | 42 03 | END |
| 24 | 18: | 5F DB | 9C CA | 46 07 | END DEF |
| 25 | 19: | 5B DB | 9C CA | 46 0B | END HANDLER |
| 26 | 1A: | 9C CA | 9C CA | 46 06 | END IF |
| 27 | 1B: | 9C CA | 9C CA | 46 0A | END SELECT |
| 28 | 1C: | A5 DE | 9C CA | 46 08 | END WHEN |
| 29 | 1D: | 8C FE | 44 D0 | 53 08 | ENVELOPE |
| 30 | 1E: | E1 D0 | 61 D0 | 52 04 | EXIT |
| 31 | 1F: | 60 DD | 61 D0 | 4A 03 | FOR |
| 32 | 20: | B5 D1 | 5A D0 | 52 05 | GOSUB |
| 33 | 21: | DF D1 | 5A D0 | 52 04 | GOTO |
| 34 | 22: | 8F FC | 61 D0 | 53 08 | GRAPHICS |
| 35 | 23: | E7 D9 | 61 D0 | 4A 07 | HANDLER |
| 36 | 24: | 9C CA | 9C CA | 02 05 | IMAGE |
| 37 | 25: | 4D D2 | 13 D2 | 42 02 | IF |
| 38 | 26: | B6 D9 | 9C CA | 4A 02 | IF |
| 39 | 27: | 12 D6 | F6 D4 | 52 05 | INPUT |
| 40 | 28: | 80 D2 | 67 D2 | 53 03 | LET |
| 41 | 29: | 16 D6 | FD D4 | 52 04 | LINE |
| 42 | 2A: | 2E C8 | 9C CA | 41 04 | LIST |
| 43 | 2B: | B5 CA | 9C CA | 61 04 | LOAD |
| 44 | 2C: | 0F DD | 61 D0 | 46 04 | LOOP |
| 45 | 2D: | 8D CB | 9C CA | 61 05 | MERGE |
| 46 | 2E: | A7 CC | 9C CA | 61 03 | NEW |
| 47 | 2F: | BC DD | 61 D0 | 46 04 | NEXT |
| 48 | 30: | 6A EB | 61 D0 | 42 07 | NUMERIC |
| 49 | 31: | 21 D3 | 9C CA | 53 04 | OPEN |

```
token
D   H    1.cím 2.cím tp    kulcsszó

50 32:  93 D3 61 D0 43 06 OPTION
51 33:  B4 F4 9C CA 41 02 OK
52 34:  B3 D3 61 D0 53 03 OUT
53 35:  82 FD 44 D0 53 04 PLOT
54 36:  C6 D3 61 D0 53 04 POKE
55 37:  E5 D3 61 D0 53 05 SPOKE
56 38:  08 D4 01 D4 53 05 PRINT
57 39:  9C CA 61 D0 42 07 PROGRAM
58 3A:  C6 D4 61 D0 52 09 RANDOMIZE
59 3B:  0F D6 F0 D4 52 04 READ
60 3C:  EE D7 61 D0 53 08 REDIRECT
61 3D:  9C CA 9C CA 03 03 REM
62 3E:  27 C8 9C CA 41 08 RENUMBER
63 3F:  06 D8 F6 D7 52 07 RESTORE
64 40:  24 D8 61 D0 42 05 RETRY
65 41:  2B D8 61 D0 52 06 RETURN
66 42:  7E CE F6 D7 53 03 RUN
67 43:  FC C9 9C CA 61 04 SAVE
68 44:  16 DC 61 D0 4A 06 SELECT
69 45:  54 D8 44 D0 53 03 SET
70 46:  28 FF 44 D0 53 05 SOUND
71 47:  78 CE 9C CA 61 05 START
72 48:  44 D9 61 D0 52 04 STOP
73 49:  5B D8 61 D0 51 04 INFO
74 4A:  3A EB 61 D0 42 06 STRING
75 4B:  08 FC 61 D0 53 04 TEXT
76 4C:  52 D9 61 D0 53 06 TOGGLE
77 4D:  6C D9 44 D0 53 05 TRACE
78 4E:  FE FB 9C CA 53 04 TYPE
79 4F:  D7 CB 9C CA 61 06 VERIFY
80 50:  48 DE 61 D0 4A 04 WHEN
81 51:  9C CA 9C CA 13 01 !
82 52:  2B C8 9C CA 41 05 LLIST
83 53:  05 D4 01 D4 53 06 LPRINT
84 54:  3E D8 61 D0 53 03 EXT
85 55:  38 D1 44 D0 53 03 GET
86 56:  29 D1 9C CA 53 05 FLUSH
87 57:  CF D2 9C CA 43 04 LOOK
88 58:  BF D3 9C CA 53 04 PING
89 59:  7B D0 9C CA 53 04 DATE
90 5A:  77 D0 9C CA 53 04 TIME
91 5B:  9F D9 9C CA 53 04 WAIT
92 5C:  9F D1 7C D1 42 02 ON
```

# 7. A beépített BASIC függvények és változók táblaterülete

| cím | ptr | tp | név | rutin | szg |
|---|---|---|---|---|---|
| 0F7E: | 00 00 | 0C | 03 ABS | 21 C3 | 01 |
| 0F88: | 00 00 | 0C | 04 ACOS | 26 C3 | 01 |
| 0F93: | 00 00 | 0C | 05 ANGLE | 30 C3 | 01 |
| 0F9F: | 00 00 | 0C | 04 ASIN | 62 C3 | 01 |
| 0FAA: | 00 00 | 0C | 03 ATN | 26 C4 | 01 |
| 0FB4: | 00 00 | 0C | 03 BIN | 2C C4 | 01 |
| 0FBE: | 00 00 | 0C | 05 BLACK | FA C4 | 01 |
| 0FCA: | 00 00 | 0C | 04 BLUE | 00 C5 | 01 |
| 0FD5: | 00 00 | 0C | 04 CEIL | 5D C4 | 01 |
| 0FE0: | 00 00 | 0D | 04 CHR$ | A2 C4 | 01 |
| 0FEB: | 00 00 | 0C | 03 COS | 14 C5 | 01 |
| 0FF5: | 00 00 | 0C | 04 COSH | 1F C5 | 01 |
| 1000: | 00 00 | 0C | 03 COT | 2A C5 | 01 |
| 100A: | 00 00 | 0C | 03 CSC | 8B CA | 01 |
| 1014: | 00 00 | 0C | 04 CYAN | 09 C5 | 01 |
| 101F: | 00 00 | 0D | 05 DATE$ | 43 C5 | 01 |
| 102B: | 00 00 | 0C | 03 DEG | 7F C5 | 01 |
| 1035: | 93 0F | 0C | 03 EPS | 91 C5 | 01 |
| 103F: | 14 10 | 0C | 06 EXLINE | D4 C5 | 01 |
| 104C: | 00 00 | 0C | 03 EXP | ED C5 | 01 |
| 1056: | 88 0F | 0D | 09 EXSTRING$ | B2 C5 | 01 |
| 1066: | BE 0F | 0C | 06 EXTYPE | E6 C5 | 01 |
| 1073: | 4C 10 | 0C | 04 FREE | 8D C6 | 01 |
| 107E: | 2B 10 | 0C | 02 FP | 7B C6 | 01 |
| 1087: | 00 00 | 0C | 05 GREEN | 03 C5 | 01 |
| 1093: | 00 00 | 0D | 04 HEX$ | EF C6 | 01 |
| 109E: | 7E 10 | 0C | 02 IN | 46 C7 | 01 |
| 10A7: | B4 0F | 0D | 06 INKEY$ | 58 C7 | 01 |
| 10B4: | 00 00 | 0C | 03 INF | 88 C6 | 01 |
| 10BE: | 00 00 | 0C | 03 INT | 75 C7 | 01 |
| 10C8: | 93 10 | 0C | 02 IP | 1B C8 | 01 |
| 10D1: | 00 00 | 0C | 03 JOY | 31 C8 | 01 |
| 10DB: | 9F 0F | 0C | 06 LBOUND | 4C C8 | 01 |
| 10E8: | A7 10 | 0D | 06 LCASE$ | 28 CD | 01 |
| 10F5: | 73 10 | 0C | 03 LEN | 7F C8 | 01 |
| 10FF: | 0A 10 | 0C | 03 LOG | 9B C8 | 01 |
| 1109: | AA 0F | 0C | 04 LOG2 | 92 C8 | 01 |
| 1114: | 1F 10 | 0C | 05 LOG10 | A4 C8 | 01 |
| 1120: | 00 00 | 0D | 06 LTRIM$ | 12 C9 | 01 |
| 112D: | F5 10 | 0C | 07 MAGENTA | 0C C5 | 01 |

```
cím    ptr  tp  név          rutin szg

113B: D1 10 0C 03 MAX        2D C9 01
1145: 56 10 0C 06 MAXLEN     42 C9 01
1152: FF 10 0C 03 MIN        6F C9 01
115C: 3B 11 0C 03 MOD        B4 C9 01
1166: 9E 10 0C 03 ORD        B5 C9 01
1170: 00 00 0C 04 PEEK       CE C9 01
117B: 09 11 0C 05 SPEEK      E7 C9 01
1187: 52 11 0C 02 PI         0B CA 01
1190: CA 0F 0C 03 POS        10 CA 01
119A: 66 10 0C 03 RAD        79 CA 01
11A4: 00 10 0C 03 RED        FD C4 01
11AE: 9A 11 0C 03 REM        96 CA 01
11B8: 20 11 0C 03 RGB        BB C4 01
11C2: D5 0F 0C 03 RND        A9 CA 01
11CC: 7E 0F 0C 05 ROUND      F0 CA 01
11D8: A4 11 0D 06 RTRIM$     54 CB 01
11E5: B8 11 0C 03 SEC        7C CB 01
11EF: C2 11 0C 03 SIN        82 CB 01
11F9: 00 00 0C 04 SINH       EB CB 01
1204: AE 11 0C 04 SIZE       0A CC 01
120F: 90 11 0C 03 SGN        F6 CB 01
1219: 45 11 0D 04 STR$       30 CC 01
1224: 00 00 0C 03 SQR        4B CC 01
122E: 66 11 0C 03 TAN        9B CC 01
1238: 3F 10 0C 04 TANH       AB CC 01
1243: E8 10 0D 05 TIME$      30 C5 01
124F: C8 10 0C 08 TRUNCATE   BB CC 01
125E: 00 00 0C 06 UBOUND     E8 CC 01
126B: B7 10 0D 06 UCASE$     25 CD 01
1278: 00 00 0C 03 USR        3F CD 01
1282: 70 11 0C 03 VAL        4E CD 01
128C: 35 10 0D 04 VER$       68 CD 01
1297: CC 11 0C 06 VERNUM     C1 CD 01
12A4: B4 10 0C 05 WHITE      0F C5 01
12B0: 00 00 0C 06 YELLOW     06 C5 01
12BD: F9 11 0D 05 WORD$      CE CD 01.
```

# 8. A képernyőkezelés fontosabb adatai

A videojelet a NICK-chip hozza létre. A megjelenítés alapja két memóriaterület: az, amelyik a megjelenítendő információt tartalmazza (adatmező) és az, amelyik a megjelenítés módját határozza meg (sorparaméter-tábla, LTP). Mindkét területnek a NICK-chip által elérhető (olvasható) memóriatartományban: az FCH—FFH RAM-szegmensekben kell lennie (video-RAM). A NICK-chip ezt a 4 (legfelső) szegmenst lapozás nélkül, mindig ugyanazokon a címeken látja:

0000H—3FFFH : FCH szegmens
4000H—7FFFH : FDH szegmens
8000H—BFFFH : FEH szegmens
C000H—FFFFH : FFH szegmens

A NICK-chip számára minden címet ilyen *videocímként* kell megadni.

Egy trükk a videocím meghatározásához: a szegmensszám alsó két bitje adja a videocím felső két bitjét.

A megfelelő szabályok szerint felépített sorparaméter-tábla nélkül semmilyen megjelenítés nem képzelhető el. A tábla a video-RAM bármely tartományában lehet, de csak 16-tal osztható címen kezdődhet: 12 értékes bitje van. Ezt a 12 bitet kell beírni — a tábla felépítése után — a NICK-chip regisztereibe:

— LPL (82H PORT): a 12 bites cím alsó 8 bitje
— LPH (83H PORT): BIT0—BIT3: a felső 4 bit

a további bitek a betöltést szinkronizálják (alaphelyzet: 1111 lehet)

A normál esetben felépülő LPT a B900H (47360D) címen kezdődik a 2. LAP-on lévő FFH rendszerszegmensben. Videocímként ez F900H (FFH szegmens!).

A sorparaméter-tábla 16-bájtos blokkokban tárolja egy-egy sor (1—255 pixelsor magas ablak) megjelenítéséhez szükséges mód, tárolási cím, szín stb. információkat a következők szerint (zárójelben az EXOS-ban kialakuló tipikus értékeket is megadjuk):

0. bájt (SC): a sor magasságát adja meg,
értéke: 256 – (pixelsorok száma)
(247, azaz 9 pont magasságú sorok)

1. bájt (MB): bitjei a megjelenítés módját vezérlik,

BIT 7 (VINT): ha értéke 1, a sor kiolvasásakor a NICK-chip megszakítást kér.
BIT 6, BIT 5 } színmód:

00: 2 szín
01: 4 szín
10: 16 szín
11: 256 szín

BIT 4 (VRES): karaktermódban 0, grafikus módban 1 (ha itt 0, a NICK-chip ismétli az előző sort)

BIT 3
BIT 2 } video mód:
BIT 1

000: a függőleges szinkronizációt biztosító kijelzés nélküli mód
001: nagyfelbontású grafikus mód
010: attributum grafikus mód
100: karaktermód (256 karakteres jelkészlet)
101: karaktermód (64-es karakterkészlet)
110: használatlan
111: kisfelbontású grafikus mód

BIT 0 (RELOAD) az LPT-cím újratöltését váltja ki a NICK-chipben (TEXT 40 módban a bájt értéke 8, GRAPHICS 4-nél 32H)

2. bájt (LM): bal margó (0—63)
+ BIT 7, BIT 6: kiegészítő színvezérlés karaktermódban

3. bájt (RM): jobb margó (0—63)
+BIT 7, BIT 6: kiegészítő színvezérlés karaktermódban
A margók értékét a NICK-chip RAM-kezelése korlátozza. A ténylegesen megjeleníthető terület:
LM > 8
RM < 54
(általában a margókat megadó alsó 6 bit: LM = 11, RM = 51)

4. bájt (LD1L)
5. bájt (LD1H) } elsődleges adatcím (a megjelenítendő információ videocíme)

6. bájt (LD2L)
7. bájt (LD2H) } másodlagos adatcím

8—15. bájt: az első 8 palettaszín
A második 8 palettaszínt (a 0—255 színtartomány 32 db 8-as csoportjának valamelyikét) a NICK-chip FIXBIAS regiszterének (80H PORT) felső 5 bitjére írt érték (0—31) határozza meg.

Az adott sor megjelenítésekor kialakuló képei a video adatmező(k) és a sor paraméterei együtt határozzák meg.

*Karaktermódban* a megjelenítendő karakter kódját az LD1 címről, az adott kódú karakter alakját, pontmátrixát az LD2 által mutatott karakter-RAM-ból olvassa ki. Pontosabban

$$LD2 = \frac{\text{a karakter-RAM videocíme}}{\text{a karakterkészlet mérete}}$$

(az EXOS által kialakított rendszerben $LD2 = \frac{F480H}{80H} = 01E9H$)

*Grafikus módban* az LD1 címen kezdődő adatmezőről kiolvasott érték bitjei (vagy bitcsoportjai) jelennek meg képpontonként.

Színképzés

A NICK-chip a színek kialakításához általában az adott sor palettaszíneit használja.
A karaktermódú megjelenítés alapvetően két színt használ (0: papír, 1: tinta)
A bal margó 6. és 7. bitje kiterjesztheti ezt a lehetőséget:
ha BIT 7, LM = 1, akkor szín = szín + 2 x(a megjelenítendő kód 7. bitje)
ha BIT 6, LM = 1, akkor szín = szín + 4 x(a megjelenítendő kód 6. bitje)
Grafikus módban:

— 2 szín esetén az adatmezőről kiolvasott bájt 0 bitjei a 0-s palettaszínnel, 1-es bitjei az 1-es palettaszínnel jelennek meg (8 pont);

— 4 szín: az adatbájt 2—2 bitje határozza meg a megjelenő 4 pont színét (0—3 palettaszín): (b3 és b7) (b2 és b6) (b1 és b5) (b0 és b4);

— 16 szín: az adatbájt 4—4 bitje határozza meg a megjelenő 2 pont színét (0—15): (b1, b5, b3, b7) (b0, b4, b2, b6);

— 256 szín: a palettaszínektől függetlenül a kiolvasott érték adja a megjelenő egyetlen pont színkódját.

A kétszínű grafikus képernyőre való többszínű karakteres írást támogatja a jobb margó (RM) 6. és 7. bitjének kiegészítő színvezérlése:
ha BIT 7, RM = 1, akkor szín = szín + 2 x(a kiolvasott bájt 7. bitje)
ha BIT 6, RM = 1, akkor szín = szín + 4 x(a kiolvasott bájt 0. bitje)

A színvezérlésre felhasznált biteket a rendszer lemaszkolja megjelenítéskor, így a karakterek soványabbak lesznek. A fentiek sajátos ötvözetét teszi lehetővé az attributum mód, amelyben az LD1-ről kiolvasott adat 8. bitje jelenik meg képpontként (2 szín), de a papír- és tintaszín az LD2 tartományról kiolvasott adat szerint karakterenként változhat:

papír: BIT 7-4
tinta: BIT 3-0

Ilyen sorparaméter-blokkból lehet összeállítani a teljes képernyőt meghatározó sorparaméter-táblát. Ez 312 pixelsor vezérlését jelenti.

A BASIC az EXOS-on keresztül 252 pixelsornyi képernyőt használ, a többi sor az alsó-felső keretet adja. A videoinformációkat tartalmazó blokkokat a függőleges szinkronizációt biztosító és az újraolvasást kiváltó zárószakasszal egészíti ki a rendszer 312 pixelsorra.

A leírtak szemléltetésére, saját tábla készítéséhez tanulságos lehet egy LPT megfigyelése. A következő sorparaméter-tábla pl. a GRAPHICS 4 utasítást követő állapotot mutatja.

```
B900  F7 08 0B 73 B8 FE E9 01  ;Állapotsor (TEXT 40)
B908  00 36 00 49 FF 24 2D 36  ;FF/BEB8 kezdőcim

B910  F7 32 0B 33 2C AC E9 01  ;1. grafikus sor
B918  00 24 49 DB 92 2D 36 FF
B920  F7 32 0B 33 FC AE E9 01  ;2. grafikus sor
B928  00 24 49 DB 92 2D 36 FF
.
.
.
BA40  F7 32 0B 33 9C E1 E9 01  ;20. grafikus sor
BA48  00 24 49 DB 92 2D 36 FF

BA50  F7 08 0B 73 24 E5 E9 01  ;21. sor (TEXT 40)
BA58  00 92 00 49 FF 24 2D 36
BA60  F7 08 0B 73 4C E5 E9 01  ;22. sor
BA68  00 92 00 49 FF 24 2D 36
.
.
.
BAB0  F7 08 3F 74 B8 FE E9 01  ;27. sor (TEXT 40)
BAB8  00 92 00 49 FF 24 2D 36

Zárószakasz:

BAC0  F2 92 3F 00 00 00 00 00  ; +14 pixelsor
BAC8  00 00 00 00 00 00 00 00
BAD0  FD 10 3F 00 00 00 00 00  ; +3 pixelsor
BAD8  00 00 00 00 00 00 00 00
BAE0  FE 10 06 3F 00 00 00 00  ; +2 pixelsor
BAE8  00 00 00 00 00 00 00 00
BAF0  FC 10 3F 1C 00 00 00 00  ; +4 pixelsor
BAF8  00 00 00 00 00 00 00 00
BB00  F0 12 06 3F 00 00 00 00  ; +16 pixelsor
BB08  00 00 00 00 00 00 00 00
BB10  EB 13 3F 00 00 00 00 00  ; +21 pixelsor
BB18  00 00 00 00 00 00 00 00     (RELOAD)
```

Az EXOS igen nagy mértékben támogatja a NICK-chip fentiek szerinti programozását. A funkcióhívások segítségével a BASIC ismert grafikus és szöveges szolgáltatásait (videolapok nyitását, megjelenítését, karakterek írását, grafikus műveleteket) egyszerűen elérhetjük gépi kódú programokból is.

A következőkben összefoglaljuk a képernyőkezelést támogató EXOS hívások főbb tudnivalóit.

— Csatornanyitás (EXOS 1 vagy EXOS 2)

Videolapokat VIDEO: perifériánévvel lehet nyitni. A létrehozandó lap méretét és módját a kezelő az EXOS változásból olvassa ki, ezért azokat hívás előtt megfelelően be kell állítani.

MODE VID: 0: hardver szövegmód (max. 42 karakter/sor, ezt használja a TEXT 40)
1: nagyfelbontású grafika (GRAPHICS)
2: szoftver szövegmód (max. 82 karakter/sor, ezt használja a TEXT 80)
4: kisfelbontású grafika
15: attributum mód

COLR VID: 0: 2 szín
1: 4 szín
2: 16 szín
3: 256 szín

X SIZ VID: a videolap szélessége (TEXT 40 módú karakterszámban megadva)
Y SIZ VID: a videolap magassága (karaktersorokban)

Fontos tudni, hogy a csatornanyitással a lap még nem jelenik meg a képernyőn (l. EXOS 11)

— Csatornazárás (EXOS 3 vagy EXOS 4)

A csatornamutatók törlésén kívül letakarja a képernyőn a lezárt csatornához tartozó sorokat.

— Karakterírás (EXOS 7)

A legsokoldalúbb képernyőkezelő funkcióhívás. Amellett, hogy bármilyen video- és színmódban kiírja a nyomtatható karaktereket a lapra (a megfelelő karaktermérettel), számos vezérlőutasítást is kezel. Ezek egy részét az ESC (CHR$ (27)) kód vezeti be.

*Általános vezérlőkarakterek*

0AH: soremelés (LF)
0DH: kurzor a sor elejére (CR)
1AH: laptörés, kurzor a kezdő pozícióba
1EH: kurzor a kezdő pozícióba
ESC K: karakter definiálása
A kiírandó bájtok:
CHR$(1BH); CHR$(4BH); CHR$(karakterkód); CHR$(D1); ....; CHR$(D9)
A D1—D9 adatok a karakter pontmátixát írják le a BASIC-ben szokásos módon.

ESC C: az első 8 palettaszín megadása
(1BH) (43H) (C1) .... (C8)
ESC c: egy palettaszín megadása:
(1BH) (63H) (N) (C)
Az N. palettaszín (0—7) értéke C lesz.
ESC I: tintaszín kiválasztása:
(1BH) (49H) (N)
A tintaszín az N. palettaszín lesz
ESC P: papírszín kiválasztása
(1BH) (50H) (N)
A papírszín az N. palettaszín lesz.
ESC =: kurzorpozíció beállítása
(1BH) (3DH) (Y+20H) (X+20H)
A kurzor az X. karaktersor Y. oszlopára áll. Ha valamelyik adat 0, abba az irányba nem mozdul a kurzor.

*Csak szöveglapon értelmezett vezérlőkarakterek*
08H: kurzor balra
09H: kurzor jobbra
0BH: kurzor fel
16H: kurzor le
19H: törlés a sor végéig
ESC?: kurzorpozíció olvasása
A következő két olvasott bájt (EXOS 5) a fentiek szerinti (Y + 20H) és (X + 20H) lesz
ESC.: kurzorkarakter beállítása
(1BH) (2EH) (N)
Az írás után az N kódú karakter lesz a lap kurzora
ESC M: kurzorszín beállítása
(1BH) (4DH) (N)
A kurzor színe az N. palettaszín lesz
ESC O: kurzorjelzés bekapcsolása
(1BH) (4FH)
ESC o: kurzorjelzés kikapcsolása
(1BH) (6FH)
ESC S: automatikus görgetés (scroll) bekapcsolása
(1BH) (53H)
ESC s: automatikus görgetés kikapcsolása
(1BH) (73H)
ESC V: fölfelé görgetés
(1BH) (55H) (K + 20H) (V + 20H)
A K—V tartományba eső sorok lépnek feljebb
ESC D: lefelé görgetés
(1BH) (44H) (K + 20H) (V + 20H)
A K—V tartomány sorai lépnek lejjebb

*Csak grafikus lapokra értelmezett vezérlőkarakterek*

ESC A: grafikus sugár pozicionálása
(1BH) (41H) (XLO) (XHI) (YLO) (YHI)
A sugár a kétbájtos X,Y koordinátákra kerül.

ESC @: sugárpozíció olvasása
(1BH) (40H)
A csatornáról olvasott 4 következő bájt a kurzorpozíciót adja

ESC R: relatív sugármozgás
(1BH) (52H) (XLO) (XHI) (YLO) (YHI)
A sugár vízszintesen X, függőlegesen Y pixelnyivel mozdul el.

ESC S: sugár bekapcsolása
(1BH) (53H)
A sugárpozíción az aktuális tintaszínnel képpont jelenik meg és a sugár a további mozgások során rajzol

ESC s: sugár kikapcsolása
(1BH) (73H)

ESC .: vonaltípus kiválasztása
(1BH) (2EH) (N)
N=1: folytonos vonal
N=2...14: szaggatott vonaltípusok

ESC M: vonalmód kiválasztása
(1BH) (4DH) (N)
N=0: felülírás
N=1: OR a régi és új videoadat között
N=2: AND
N=3: XOR

ESC a: attributum jelzőbájt beállítása
(1BH) (61H) (N)

ESC F: grafikus kitöltés (FILL)

ESC E: ellipszisrajzolás
(1BH) (45H) (XLO) (XHI) (YLO) (YHI)
A BASIC PLOT ELLIPSE X,Y utasításnak felel meg.

— *Blokkírás* (EXOS 8)
A karakterírás valamennyi funkciója használható

— *Karakterolvasás* (EXOS 5)

— *Blokkolvasás* (EXOS 6)
Az olvasott bájt alaphelyzetben
— szöveglapnál a kurzorpozíción álló karakterkódja
— grafikus lapnál a sugárpozíción álló képpont színe (palettaszín, ill. 255 színű módban a színkód)

Az ESC? ill. ESC@ írása után az olvasott bájtok a kurzor (sugár) pozícióját adják.

— *Speciális funkció* (EXOS 11)

Az alfunkciószámtól (B) függően elérhető funkciók:

B = 1: videolap kijelzése

B = 2: üzemmód és méret lekérdezése

B = 3: tárolási cím lekérdezése

B = 4: karakterkészlet inicializálására

Az egyes funkciókat, hívásuk módját és paramétereit a 2.3.2 pontban részletesen leírtuk.

# 9. Az ASMON (SIMON) assembler-monitor program használata

Röviden összefoglaljuk a könyv anyagának összeállításánál használt assembler-monitor (az 1.3 verziójú SemiSoft Asmon program) használatához szükséges tudnivalókat.

A program a beolvasás után ":ASMON" paranccsal indítható. Ekkor a monitorprogram jelentkezik be. A képernyő felső részén (az üzenetmezőn) ekkor a pillanatnyi státusz látható. Ebben az EXOS rendszerállapot mellett a pillanatnyi lapkiosztás is látható:

```
        EXOS version no. is 2.1
Page0=F8        01 Free segment(s)
Page1=FB        08 Working segment(s)
Page2=FC        00 Defective segment(s)
Page3=F9        08 Total RAM segment(s)


      Henrik Bendtsen 1985
```

A képernyő alsó részén az ASMON a regiszterek tartalmát és az általuk megcímzett memória környezetét jelzi ki.

A monitor a képernyő középső részén (INPUT-mező) várja a parancsot, illetve a hozzá tartozó címeket, adatokat.

Az értékeket az INPUT-sorban kijelzett számrendszerben kell beírni: hexadecimálisan (HEX) vagy tízes számrendszerben (DECI). A pillanatnyi üzemmód CTRL + F8-cal billenthető át.

A parancsok egykarakteresek és ENTER nélkül írhatók be.

Az elfogadott utasítások listáját kiírathatjuk az üzenetmezőre (H, azaz help):

```
A=Assemble              B=Breakpoints
C=Copy memory           D=Dump memory
E=Edit source           F=Fill memory
G=Go (CALL PC)          H=HELP function
I=Set func key          J=Swap registers
K=Read source           L=Disassemble
M=Modify memory         N=Number cruncher
O=Output port           P=Printer ON/OFF
Q=Query port            R=Read BIN file
S=Save BIN file         T=Trace memory
U=Untrace               V=View status
W=Write source          X=eXamine register
Y=Options Disas         Z=Options Assemble
[=Options Printer       \=Options Editor
]=Load Module           @=Symbol table
==Find string           :=EXOS string
```

Az *utasítások* a fenti sorrendben:

A: Lefordítja az Editor-ban megírt assembly programot a beállított opcióknak (l. Z) megfelelően

B: Töréspontokat (BREAKPOINT 1, illetve 2) helyez el a megadott címen (praktikusan a tesztelni kívánt gépi kódú programban). A rutin futtatásakor (G) erről a pontról visszatér a monitorba és visszaállítja az eredeti memóriatartalmat

C: A Start-tól az End-ig terjedő memóriatartományt másolja át a Destination (cél) kezdőcímű területre.

D: Kiírja az üzenetmezőre a Start kezdőcímű memóriaterület tartalmát hexadecimális dump formában. A listázás folyamatos, de ENTER-rel megszakítható. ENTER-re soronként, SPACE-re újra folyamatosan léphetünk tovább. Az üzemmódból STOP-pal vagy az ESC gombbal léphetünk ki.

E: Átlép Editor üzemmódba

F: Feltölti a memória Start-End tartományát a Value-ra megadott értékkel

G: Futtatja a Start címen kezdődő gépi kódú programot

H: Kiírja az utasítások listáját

I: a Key number-re megadott számú funkcióbillentyűt programozhatjuk a Key string szövegre (pl. G4000)

J: Az AF, BC, DE, HL regisztereket a háttérregiszterekre cseréli (ezt — ALT-REGS — felirat jelzi)

K: Beolvassa a File name nevű (lehet üres string is) assembly forrásprogramot az Editorba

L: Disassemblálja a Start címen kezdődő gépi kódú programot. Opciói beállíthatók (Y)

M: Módosíthatjuk a hexadecimális dump és karakterdump formában kiírt memóriatartományt. A kurzor ALT+F8-cal váltható át az egyik mezőből a másikba. Kilépés STOP-pal vagy ESC-pel.

N: Kiírja a Value-ra megadott érték (szám, kifejezés, címke stb.) hexadecimális, decimális, bináris és karakteres alakját. Alapértelmezése hexadecimális; az eltérő értéket jelezni kell (pl. 65D, 101B, "A" stb.)

O: Portállítás: lényegében egy Out (Port), Value utasítás. Ezzel változtathatjuk meg például (a B0—B3 portokon) a lapkiosztást

P: Nyomtatókimenet nyitás/zárás. ON állásban a dump, disassemblált lista stb. a nyomtatóra is kikerül

Q: Kiírja a Port periféria aktuális értékét (IN)

R: Betölti a Start-End memóriaterületre a File name nevű állományt

S: Kiírja a Start-End memóriatartományt File name néven bináris fájlként

T: Végrehajt a Start címen kezdődő gépi rutinban Steps lépést, miközben kijelzi a címeket és utasításokat

U: A fenti utasítással azonos funkció kijelzés nélkül

V: Kijelzi a státuszt (l. belépés)

W: File name néven kiírja az Editor forrásprogramját

X: Beállíthatjuk a Regiszterpair regisztereket Value értékre

Y: Disassembler (L) opciók

Memory offset: a címekhez hozzáadandó eltolási érték. Ezzel lehet pl. az 1. LAP-on tárolt programot úgy listázni, hogy a címek a végleges, 3. LAP-tartományban jelenjen meg (offset = 8000)

Addresses YES/NO

Az assembly szöveg előtt a cím és a kód listázását kapcsolja be/ki. Átkapcsolás bármelyik billentyűvel, beírás az ENTER-rel. A letiltással lehet pl. forrásszöveg formátumú listát kapni meglévő programokról (l. még [).

Z: Assembler (A) opciók

Lásd az Editor/Assembler leírásnál (a 9. Függelék további részében)

[ (BRD: Ä): PRINTER-opciók

Output to PRINTER/EDITOR

Editor választásnál a listák, dump-ok stb. a szövegszerkesztőbe kerülnek, és innen háttértárolóra vihetők, beépíthetők a saját forrásprogramba stb.

Use: (PRINTER:)

A nyomtató meghajtására használandó kezelő neve

Return send CR+LF/CR

Sorvégeken kiküldendő vezérlőkarakterek választása

Lines (72): oldalanként sorszám

Chars (72): soronkénti betűszám

Left margin (8): bal margó

Bottom margin (6): alsó margó

Header: fejléc-szöveg (az Asmon V 1.3 szöveggel és a lapszámmal együtt íródik ki)

\ (BRD:Ö): Editor opciók

Szükség esetén változtatható az editorpuffer kezdő- és végcíme:

Editor buffer start (0801)

Editor buffer end (BFFF)

] (BRD:Ü): Modul betöltése

@ (BRD:§): A forrásprogram lefordítása után kiírja a szimbólumok (címkék, változók) nevét és értékét

=: Megkeresi a Start címen kezdődő memóriaterületen a Search kérdésre megadott bájtsorozatot (számok és idézőjelek közötti karakterek vesszővel elválasztva)

:: EXOS parancsszöveg (pl.":BASIC")

*Editor és assembler*

A monitorból az E paranccsal léphetünk az editorba. Ez az üzemmód lényegében egy szövegszerkesztő, amelyben megírhatjuk az assembly forrásprogramot. Az editor szintaktikai ellenőrzést nem végez.

Az utasítássorokat sorszám nélkül kell beírni. A sor első pozícióján kezdődő szöveget az assembler szimbólumnak (címkének) tekinti. Az utasításokat praktikus az első tabulátorpozíciótól kezdeni.

Az utasítások számértékű operandusait (pl. LD A, n) igen sokféleképpen megadhatjuk:

— számkonstansként:
decimálisan: 65D,
hexadecimálisan: 41H
(a hexadecimális érték csak számmal kezdődhet, pl. 0FFH),
oktálisan: 101O,
binárisan: 01100101B
(az azonosító betű nélküli számokat alapértelmezésűnek tekinti, l. RADIX);
— karakterenként ("A"), értéke a karakter ASCII kódja;
— változóval (szimbólummal);
— a $ karakterrel (értéke a fordítás közbeni aktuális cím).

Az assembler bizonyos műveleteket is megenged és kiértékel az operandusokban (a BASIC szabályai szerint):

— összeadás, kivonás, szorzás, osztás;
— modulófüggvény (X MOD Y);
— logikai függvények:
X AND Y,
X OR Y,
X XOR Y,
— NOT X (egyes komplemens);
— speciális műveletek:
LOW X: X alacsonyabb helyiértékű bájtja,
HIGH X: X felső bájtja,
X SHRY: az X-et jobbra forgatja Y bittel,
X SHLY: az X-et balra forgatja Y bittel;
— kiértékeli a < = > relációkat is.

A Z80-as utasításokon kívül az *ASMON assembler* jó néhány *saját utasítást* (direktívát) is ismer.

ORG nn
A címszámláló kezdőértéke nn lesz (az assembler erre a címre tölti a lefordított gépi kódú programot). Általában a program első sora

DEFB n (vagy DB n)
Az operandus értékét tárolja az aktuális címen. Az n helyén egybájtos operandus, vagy ilyenek vesszővel elválasztott sorozata, valamint karaktersorozat állhat.
Például:

DEFB 07H,"PRINTER:"

DEFW nn (vagy DW nn)
A kétbájtos értéket tárolja a programban.

DEFM füzér
A karaktersorozat kódjait tárolja

DEFS nn: üresen hagy nn helyet

EQU: "szimbólum EQU nn" formában értéket ad a szimbólumnak, címkének

INCLUDE név: beolvassa és lefordítva a programba illeszti a kijelölt programot

EXOS n: az n. EXOS funkcióhívást váltja ki. Lényegében DEFB 0F7H,n
END: a forrásprogram vége

*További direktívák* a fordítás menetét vezérlik:

.PHASE nn:
Ideiglenesen az nn értékre állítja a címszámlálót

.DEPHASE
Visszaállítja a címszámlálót az eredeti ORG-nek megfelelő címtartományra. A két utasítás közötti programszakaszt a főprogramban helyezi el az assembler, de címei egy másik tartományban érvényesek (kihelyezhető rutin).

IF kifejezés
1. rutin
ELSE
2. rutin
ENDIF

Az assembler a BASIC IF kezelésnek megfelelő rutint fordítja le.

Kiegészítő feltétel-lehetőségek:
IFT (= IF TRUE, ha igaz...)
IFF (= IF FALSE, ha hamis...)
IF1 (ha a fordítás 1. menete folyik...)
IF2 (ha a fordítás 2. menete folyik..)
.LIST: bekapcsolja a listázást
A fordítás közben kiíródik a cím, a kód és a mnemonik.(forrásszöveg)
.XLIST: kikapcsolja a listázást
.SFCOND: feltételes elágazásnál csak a teljesülő (lefordított) ágat listázza
.LFCOND: minden ágat listáz
.TFCOND: átbillenti (TOGGLE) a feltételes listázás üzemmódját
TITLE füzér: fejlécszöveg a nyomtatónak
.PRINTX füzér: a fordítási listába beírandó szöveg (vagy érték, ha az első karakter a "%")
.COMMENT karakter: addig nem fordítja a forrásprogramot, amíg a megadott karakterhez nem ér
.RADIX n: az alapértelmezésű számrendszer alapja

Az ASMON-ban *makrókat* is definiálhatunk: olyan — általában rövid, de gyakran használt — programrészleteket, amelyekre programunkban nevükkel hivatkozhatunk. Fordítás közben az assembler a nevek helyére mindenhol a hivatkozott rutinrészletet tölti.

MACRO — makrokezdet
ENDM — makrovég
EXITM — makrobefejezés ENDM előtt (feltételes elágazásnál)

Például:

```
.PUSH MACRO
        PUSH HL
        PUSH DE
        PUSH BC
        ENDM
```

Ezután a

```
        .PUSH
        LD HL, 1234H
```

programrészlet a következőképpen fordítódik:

```
        PUSH HL
        PUSH DE
        PUSH BC
        LD HL, 1234H
        ...
```

A makrónak paraméterek is átadhatók:

```
        "név MACRO P1, P2,..."
```

Ezek a paraméterek operandusok vagy akár regiszternevek is lehetnek. Például:

```
MASZK MACRO P1, P2
        LD A, P1
        AND P2
        LD P1, A
        ENDM
```

Ennek felhasználásával a

```
        LD HL, (CIM)
        MASZK H, 0FH
        MASZK L, 0F0H
        ...
```

rutin a fordítás után:

```
        LD HL,(CIM)
        LD A,H
        AND 0FH
        LD H,A
        LD A,L
        AND 0F0H
        LD L,A
        ...
```

A programírást, szövegkezelést a funkcióbillentyűkkel hívható almenük segítik. A menükben a joystick-kel mozoghatunk, ENTER-rel választhatunk.

A *funkciók* röviden:

F1: Find string: szövegkeresés
Find & repl: szövegkeresés és-helyettesítés
Cont search: továbbkeresés
Repl string: szöveghelyettesítés
Repl & cont: továbbkeresés helyettesítés után

F2: Set tabmark: tabulátor beállítása a kurzorpozíción
Res tabmark: tabulátor törlése a kurzorpozíción
Clr tabmark: az összes tabulátor törlése
Stand tabmark: tabulátorok alaphelyzetbe

F3: Mark start: blokk-kezdet a kurzor helyén
Mark end: blokkvég a kurzor helyén
Copy block: kijelölt blokk másolása a kurzor helyére
Del block: kijelölt blokk törlése
Prnt block: kijelölt blokk nyomtatása
Clr marks: blokkjelzők törlése

F4: Load file: forrásszöveg beolvasása (MERGE)
Save file: forrásszöveg kivitele
Save block: kijelölt blokk kivitele
Kill text: forrásszöveg törlése

F6: Információ: Text: forrásszöveg hossza
Free: szabad puffer

F7: Help (szövegszerkesztő funkciók)

F8: visszatérés a monitorhoz

Az editorral megírt forrásprogramot a monitor A parancsa fordítja le két menetben (PASS 1, szimbólumtábla készítése, PASS2: fordítás).
A fordítás opcióit a Z utasítással adhatjuk meg:
Assembly listing OFF/ON
A fordítás közbeni lista készítését kapcsolhatjuk ki/be.
List conditions NO/YES
Feltételes elágazások listázása
Force Pass2 NO/YES
"YES" esetén akkor is megkísérli a 2. menetet a fordító, ha az elsőben hibát talált

Memory assembly NO/YES

"Yes" esetén az ORG címnek (és az offsetnek) megfelelően a memóriába töltődnek a lefordított gépi kódok (így készíthető futtatható rutin). Ebben az esetben a következő kérdés:

Memory offset

Megadható a címszámláláshoz hozzáadandó eltolási érték. Pl. egy rendszerbővítőnek, majd a 3. LAP-on kell futnia, de az assembler nem töltheti ide, hiszen saját maga is itt működik. Ilyenkor:

ORG C000H

offset = 8000H

Object file name:

Ha nevet adunk meg, az ASMON a lefordított programot háttértárolóra viszi. Ehhez adhatunk meg további információkat

EXOS module header NO/YES

A fájl elé kell-e EXOS fej

EXOS modul type

Megadható a kivitt fájl típusa (pl.: 6, abszolút rendszerbővítő). Az ASMON nem kezeli a relatív fájl-formát (2. és 7.)

A fentiek alapján könnyen elkészíthető pl. a 2.6. alfejezet rendszerbővítője ASMON-ban. Az ORG C00AH kezdőcím és a Memory offset = 8000H eltolás biztosítja, hogy a bővítő a C00AH címre fordítódjon, az

Object file name: "NEV"

EXOS module header: YES

EXOS module type: 6

opciók pedig azt, hogy abszolút rendszerbővítőként kerüljön kazettára. Beolvasáskor a bővítő automatikusan beláncolódik és így elláthatja a funkcióit.

182,– Ft

A könyv a gépi kódú programozás alapjait már ismerő olvasóknak bemutatja e terület Enterprise nyújtotta sajátosságait és lehetőségeit.
Főbb témakörök: Memóriafelosztás – Memóriakezelés – A BASIC operációs rendszer – Számábrázolás, aritmetika – Kiíratás, karakteres képernyőkezelés – Grafika – A gépi kódú programozás eszközei.

*

Enterprise 128-as tulajdonosoknak ajánljuk:
Brányi László: Enterprise 128-as ROM visszafejtése
Felföldi József–Sz. Lukács János: Tippek és trükkök

NOVOTRADE