INOTAI LÁSZLÓ
LÁZÁR LÁSZLÓ

IBM PC XT/AT rendszerprogramozás II.

A BIOS és bevezető a hardver közvetlen programozásába

Novotrade Kiadó

INOTAI LÁSZLÓ
LÁZÁR LÁSZLÓ

IBM PC XT/AT rendszerprogramozás II.

A BIOS és bevezető a hardver közvetlen programozásába

Novotrade Kiadó

Írta : Inotai László
Lázár László

Lektorálta : Ila László

A kiadásért felel Rényi Gábor, a Novotrade Rt. vezérigazgatója
Budapest 1991

Tipográfia és fedélterv: Erdősi Zoltán, fotó: Ács László

Szedte az Anteus Kft.

Készült a Szegedi Nyomdában.: Felelős vezető: Kónya Antal megbízott igazgató

ISBN 963 585 109 X (I., II., III. kötet)
ISBN 963 585 111 1 (II. kötet)

TARTALOMJEGYZÉK

1. BIOS

Az IBM PC/XT/AT számítógépek működését vezérlő rendszerprogramok második nagy csoportját a BIOS néven ismert rendszerprogramok alkotják. A BIOS az angol Basic Input/Output System kifejezés kezdőbetűiből képzett betűszó, a kifejezés magyar nyelvű megfelelője „be- és kiviteli alaprendszer". A BIOS egyik fő feladata, hogy közvetlen kapcsolatot tartson a számítógép központi egysége és a hozzá csatlakozó legfontosabb külső egységek, pl. billentyűzet, képernyő, lemezmeghajtó(k) stb. között.

A BIOS tehát alapvetően azokat az – általunk eszközmeghajtóknak nevezett – perifériavezérlő programokat tartalmazza, amelyek ezeknek a perifériáknak a működését vezérlik.

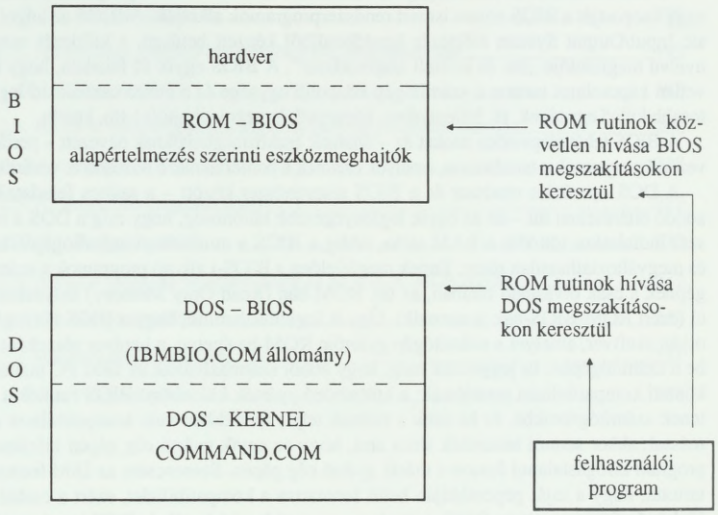
A DOS operációs rendszer és a BIOS alaprendszer között – a sajátos feladataikból adódó eltéréseken túl – az az egyik leglényegesebb különbség, hogy míg a DOS a rendszer indításakor töltődik a RAM tárba, addig a BIOS a mindenkori számítógép állandó és megváltoztathatatlan része. Ennek megfelelően a BIOS-t alkotó programok a számítógépnek a csak olvasható tárában, az ún. ROM-ban (Read Only Memory) helyezkednek el (ezért ROM-BIOS-nak is nevezik). Úgy is fogalmazhatunk, hogy a BIOS lényegében olyan szoftver, amelyet a számítógép gyártója ROM-ba égetve, a hardver részeként épít be a számítógépbe. Itt jegyezzük meg, hogy ebből származhatnak az IBM PC utánzatok közötti kompatibilitási problémák: a különböző gyártók különböző BIOS rutinokat építenek számítógépeikbe, és ha ezek a rutinok teljes mértékben nem kompatibilisek egymással, akkor semmi biztosíték sincs arra, hogy az egyik gyártó cég gépén kifejlesztett program kifogástalanul fusson a másik gyártó cég gépén. Szerencsére az IBM fontosnak tartotta, hogy a saját gépcsaládján belül fenntartsa a kompatibilitást, ezért a család különböző gépein az egyes BIOS rutinok azonos módon hívhatók (a DOS-hoz hasonlóan ezek is megszakításokon keresztül; ezeket nevezik BIOS megszakításoknak).

A BIOS, illetve a megszakításokon keresztül hívható BIOS rutinok fő célja az, hogy megbízható és állandó kapcsolódást hozzanak létre a hardver és felhasználója között. Az állandóság biztosítja azt, hogy a gépre írt programok hardverfüggetlenek lehetnek: a kapcsolódási pontok akkor sem változnak, ha a hardveren belül változtatások történnek. Így a programozónak a munkája során ezekhez a változásokhoz nem kell igazodnia (de az előnyeit élvezheti).

Az előző fejezetekben részletesen bemutatott DOS operációs rendszer is ezeket a BIOS megszakításokat veszi igénybe az alapvető be- és kiviteli műveletek elvégzéséhez.

Ugyanígy egy felhasználói programnak is lehetősége van arra, hogy kihasználja a BIOS megszakítások által nyújtott lehetőségeket. Annak eldöntése, hogy egy program mikor használja a DOS és mikor a BIOS szolgáltatásokat, alapos és több szempontra kiterjedő mérlegelést igényel. Itt csak röviden térünk ki erre: ha azt akarjuk, hogy a programunk teljes mértékben hardverfüggetlen legyen, azaz minden olyan számítógépen fusson, amelynek operációs rendszere az MS-DOS, akkor BIOS megszakításokat nem hívhatunk. Ekkor viszont le kell mondanunk arról, hogy kihasználjuk a BIOS által a perifériák – mindenekelőtt a képernyő és a soros interfész – kezelésében nyújtott nagyobb rugalmasságot. A legtöbb program kompromisszumot köt: a hardverfüggetlenség egy részéről lemond a nagyobb sebesség és rugalmasság javára.

A ROM-BIOS, DOS-BIOS és a felhasználói program közötti kapcsolatokat az 1. ábra mutatja. A DOS-BIOS által hozott új BIOS megszakítások (a vektor felülírásával) lehetővé teszik a ROM-BIOS továbbfejlesztését.



1. ábra: A ROM – BIOS, a DOS – BIOS és a felhasználói program kapcsolódásai

A rendelkezésre álló BIOS megszakítások:

Megszakítás száma	Megszakítás szerepe
16 – 10(h)	Képernyőkezelő funkciók
17 – 11(h)	Konfiguráció megállapítása
18 – 12(h)	Tárkapacitás megállapítása
19 – 13(h)	Hajlékony- és merevlemezes műveletek
20 – 14(h)	Soros interfészre vonatkozó műveletek
21 – 15(h)	Kazetta műveletek
22 – 16(h)	Billentyűzetkezelő műveletek
23 – 17(h)	Nyomtatókezelő műveletek
25 – 19(h)	Rendszer újraindítása (Ctrl-Alt-Del)
26 – 1A(h)	Dátum és idő megállapítása

Könyvünknek ez a része alapvetően e BIOS megszakítások szerepét, meghívásuk módját és használatukat ismerteti.

1.1. A rendszer indítása

Az I. kötet 6.3 alfejezetében (A DOS betöltése) azt vizsgáltuk, hogy mi történik a számítógép bekapcsolását követően. Ott azt írtuk, hogy amikor bekapcsoljuk a gépet, vagy a Ctrl-Alt-Del billentyűk egyidejű lenyomásával újra indítjuk (reset), akkor elindul egy, a ROM-BIOS-ban, az FFFF0(h) abszolút címen tárolt program végrehajtása. Azt is írtuk, hogy ezen a címen egy gépi kódú ugró utasítás (JMP) áll, amely a vezérlést egy belső hardver rutinnak adja át. Most ennek a rutinnak a feladatát nézzük meg.

Maga a rutin számos részrutinból épül fel. Ezeket az IBM PC szakirodalom összefoglaló néven POST (Power On Self Test), azaz magyarul bekapcsolási öntesztelő rutinoknak nevezi. A rutin a BIOS verziótól függően különböző ROM címeken kezdődhet, de a feladata azonos.

Először természetesen magát a processzort, a processzor regisztereit és néhány utasítás végrehajtását vizsgálja meg. Ha ennek során hibát talál, akkor „szó nélkül" leáll (mivel egy hibás processzor meg sem tud szólalni). Ha a processzor működését rendben találta, akkor rátér a ROM elemek vizsgálatára, amelyek helyes működését egy ellenőrző összeg képzésével állapítja meg. Ezt követően megvizsgálja az alaplemezen lévő integrált áramköröket (beleértve a RAM elemeket is), majd rátér a perifériák vizsgálatára (billentyűzet, képernyő, lemezmeghajtók stb.). E vizsgálatok során különböző táblákat hoz létre (így például a megszakítási vektortáblát), és a táblákba, valamint a BIOS változókba beírja a megfelelő kezdeti értékeket.

Ennek a bekapcsolási öntesztelő rutinnak a végrehajtása teljes egészében a BIOS „belügye", hiszen a DOS operációs rendszer még nincs a tárban. Ennek betöltésére a BIOS a 19(h) megszakítást hívja meg, amely természetesen a felhasználó rendelkezésére is áll (ezt a megszakítást váltja ki a Ctrl-Alt-Del billentyűk egyidejű lenyomása: mint tud-

juk, így indítható újra a rendszer). Ez a ROM-ban lévő betöltő (bootstrap) rutin most megnézi, talál-e valamelyik meghajtóban olyan lemezt, amelyről az operációs rendszert (ami nemcsak a DOS lehet) betöltheti. Ha talált, akkor betölti az első szektort, és ezzel a maga részéről elvégezte a feladatát. A többi már nem az ő dolga (ld. A DOS betöltése című fejezetet).

1.2. Alapvető hardverjellemzők és a konfiguráció megállapítása

Mint ismeretes, az IBM PC gépcsalád tagjai alulról felfelé kompatibilisek egymással, vagyis egy PC/XT-re írt program minden további nélkül fut egy PC/AT-n, de fordítva már nem biztos. Ha ugyanis a program olyan utasításokat is tartalmaz, amely csak az AT alapját képező 80286-os mikroprocesszor utasításkészletében szerepelnek, akkor ez a program a 8088-as processzorra épülő XT gépen nem fog működni. Hasonló a helyzet akkor is, ha egy program meghatározott konfigurációt igényel (pl. adott méretű tár, színes-grafikus képernyőcsatoló stb.), de ez nem áll rendelkezésre. A program ebben a környezetben nagy valószínűséggel nem fog működni.

Azért, hogy az ilyen eseteket elkerüljük, a programnak mindjárt az induláskor illik megvizsgálnia a hardver adottságokat, és amennyiben ezt nem találja kielégítőnek, erről a felhasználót tájékoztatnia.

A következőkben azt vizsgáljuk meg, hogy miként kérdezhetők le ezek a hardver jellemzők és a gép konfigurációja.

1.2.1. A BIOS verziója

A ROM-ban lévő BIOS program, csakúgy mint általában más programok, különböző adatokat is tartalmaz. Ezek egy része állandó, ezért a ROM-ban helyezhetők el, más részük változó, ezért a RAM-ba kerülnek. ROM-ban lévő állandó adat például a BIOS verziószáma és a számítógép típusának kódjele.

A korábbiakban már volt arról szó, hogy az FFFF0(h) abszolút címen egy gépi kódú utasítás áll, amely egy – többnyire távoli, azaz 5 bájtos – ugrást ír elő. Ezt az öt bájtot követő, 8 bájt hosszú tárterületen általában a BIOS verziószáma, ill. a verzió kibocsátásának dátuma áll, az angol nyelvterületen szokásos hónap/nap/év ábrázolásmódnak megfelelően. Ezt a dátumot nagyon egyszerűen elolvashatjuk, ha – pl. a DOS részét alkotó DEBUG programmal – a képernyőre iratjuk az F000:FFF0 tárcímen kezdődő 16 bájtot. Eredményül pl. ez jelenhet meg:

F000:FFF0 EA 5B E0 00 F0 30 39 2F-32 34 2F 38 37 FF FE 38 j[‘.p09/24/87.~8

Az első 5 bájt a JMP F000:E05B ugróutasítást, a következő 8 bájt pedig az 1987. szeptember 24-i dátumot tartalmazza.

A BIOS természetesen ezen túlmenően még tartalmazza a gyártó cég nevét és a verziószámát is, ennek helye azonban gyártótól függően különböző tárcímeken lehet.

1.2.2. A PC típusa

Egy program számára a BIOS kibocsátásának dátumánál vagy a verziószámánál sokkal fontosabb információ az, hogy a PC család melyik tagján fut. Amint már említettük, ha egy program olyan szolgáltatásokat akar igénybe venni, amelyek csak a PC/AT-n állnak rendelkezésre (pl. a valós idejű óra vagy az 1 Mbájt fölötti tárterület használata), akkor először meg kell győződnie arról, hogy a befogadó gép valóban AT típusú.

A gép típusát a ROM-BIOS az FFFFE(h) abszolút címen tartalmazza, az alábbi kódolás szerint:

FB(h): PC/XT (későbbi típusok)
FC(h): PC/AT, PC/XT-286
FE(h): PC/XT (korábbi típusok)
FF(h): PC

A fenti értékek csak eredeti IBM gyártmányú PC-k esetében garantálhatók. Az utánzatoknál az azonosító bájt értéke az itt megadottól eltérhet. Általánosságban az mondható, hogy amennyiben az azonosító bájt értéke FC(h), akkor a gép 80286-os processzorra épült AT vagy XT, egyébként 8088-as alapú XT.

1.2.3. A PC konfigurációja

Az előző fejezetben említettük, hogy a számítógépben a bekapcsolását követően lefut a bekapcsolási öntesztelő rutin, amely sorban megvizsgálja a rendszerben lévő ROM és RAM elemeket, valamint a csatlakoztatott perifériákat. A vizsgálat elvégzéséhez természetesen számba kell vennie azt, hogy milyen elemeket kell megvizsgálnia. Egy megszakításon keresztül a felhasználói program is hozzáférhet ezekhez az információkhoz, és így megállapíthatja, hogy milyen az a konfiguráció, amelyben futhat.

A megszakítás száma 11(h), a meghívásához semmiféle paramétert, illetve funkciószámot sem kell átadni. A megszakítás visszatérésekor az AX regiszterben kapjuk vissza azt a szót, amelybe a rendszer az indításakor észlelt konfigurációra vonatkozó információkat beírta.

A konfiguráció szó egyes bitjeinek jelentése a PC, PC/XT, valamint az AT esetében némileg eltér. Ezért ahhoz, hogy az egyes bitek jelentését helyesen értelmezhessük, először a gép típusát kell megállapítani.

A bitek jelentése PC és PC/XT esetén:

0.bit: értéke 1, ha a rendszer egy vagy több hajlékonylemezes meghajtót tartalmaz
1.bit: nem használt
2–3. bit: nem használt (az első PC-knél az alaplemezen elhelyezett RAM tár méretét adta meg max. 64 kbájtig).
4–5. bit: video üzemmód a rendszer indításakor
00 = nem használt
01 = 40*25 karakter, színes kártya
10 = 80*25 karakter, színes kártya
11 = 80*25 karakter, monokróm kártya
6–7. bit: a rendszerben lévő hajlékonylemezes meghajtók számát adja meg, ha a 0. bit értéke 1:
00 = 1 hajlékonylemezes meghajtó
01 = 2 hajlékonylemezes meghajtó
10 = 3 hajlékonylemezes meghajtó
11 = 4 hajlékonylemezes meghajtó
8. bit: értéke 0, ha a rendszerben van egy DMA áramkör
9–11.bit: a csatlakoztatott RS232 kártyák száma
12. bit: értéke 1, ha a rendszerhez játékadapter csatlakozik
13. bit: nem használt
14–15. bit: a csatlakoztatott nyomtatók számát adják meg.

A bitek jelentése PC/AT esetén:

0. bit: értéke 1, ha a rendszer egy vagy több hajlékonylemezes meghajtót tartalmaz
1. bit: értéke 1, ha a rendszer matematikai társprocesszort tartalmaz
2–3. bit: nem használt
4–5. bit: video üzemmód a rendszer indításakor
00 = nem használt (vagy EGA)
01 = 40*25 karakter, színes kártya
10 = 80*25 karakter, színes kártya
11 = 80*25 karakter, monokróm kártya
6–7. bit: a rendszerben lévő hajlékonylemezes meghajtók számát adja meg, ha a 0. bit értéke 1:
00 = 1 hajlékonylemezes meghajtó
01 = 2 hajlékonylemezes meghajtó
10 = 3 hajlékonylemezes meghajtó
11 = 4 hajlékonylemezes meghajtó
8. bit: nem használt
9–11. bit: a csatlakoztatott RS232 kártyák száma
12–13. bit: nem használt
14–15. bit: a csatlakoztatott nyomtatók számát adják meg.

Felhívjuk a figyelmet arra, hogy ez a funkció azt a video üzemmódot szolgáltatja, amelyet a rendszer a bekapcsolásakor észlelt. A mindenkori aktuális üzemmód a 10(h) megszakítás 0F(h) funkciójával állapítható meg.

1.2.4. A RAM tár nagysága

Az előbb ismertetett, a konfigurációt lekérdező, 11(h) megszakítás meghívásával csak az alaplemezen lévő RAM tár mérete állapítható meg. A rendszerben lévő teljes tárkapacitás lekérdezéséhez a 12(h) megszakítást kell meghívni. A PC és az XT esetén a számítógép bekapcsolásakor beinduló öntesztelő rutin megvizsgálja a RAM méretére vonatkozó DIP-kapcsolók (Dual-In-Line Package) állását, és ezek alapján állapítja meg a rendelkezésre álló tárméretet. A 12(h) megszakítás az így megállapított tárméretet olvassa le. Ebből következik, hogy ez – a kapcsolók beállításától függően – nem szükségszerűen egyezik meg a rendszerben fizikailag is meglévő tárkapacitással. Az AT esetén a megszakítás a fizikai tár méretére vonatkozó információt egy 64 bájtos RAM tárból olvassa ki (ez a tár nem a központi tár része, hanem egy ettől független integrált áramkörben helyezkedik el, amely a külön telepről táplált, valós idejű órát is tartalmazza). A 12(h) megszakítással azonban mindkét esetben csak az 1 Mbájt alatti RAM tár mérete kérdezhető le. Ez PC és PC/XT esetén nem jelent megszorítást, hiszen a 8088-as processzor címtartománya amúgy is csak 1 Mbájtig terjed. Más a helyzet az AT esetén, mert az ebben lévő 80286-os processzorral akár 16 Mbájt tárterület is megcímezhető, így ezek a gépek az 1 Mbájtos határ fölött is tartalmazhatnak RAM elemeket. Ezek méretét egy másik megszakítással lehet kérdezni (erre majd a későbbiekben térünk ki).

A 12(h) megszakítás visszatérésekor az AX regiszterben szolgáltatja a leolvasott tárméretet, 1 kbájtos egységekben.

A következő pontban olyan programpéldákat mutatunk be, amelyek az előzőekben ismertett hardverjellemzőkről és a rendszer konfigurációjáról adnak tájékoztatást.

1.2.5. Programpéldák

Az előzőekben leírtaknak megfelelően a konfiguráció főbb paramétereinek megállapítását bemutató programpéldák elsőként az F000:FFFE címen elhelyezkedő bájtot olvassák be. Ennek tartalma alapján határozható meg a gép processzorának típusa (80286 vagy 8088). A géptípusnak jelentősége van az 1 Mbájt feletti tárterületek szempontjából is, ebben a címtartományban csak az AT tartalmazhat bővítést.

A programok a 11(h), 12(h) és (AT esetén) 15(h) megszakításokat használják fel a konfigurációs paraméterek megállapítására. Az 1 Mbájt alatt rendelkezésre álló tárterület meghatározása célszerűen a külön erre a célra szolgáló 12(h) megszakítással történik, ez a konfigurációs szóval szemben nem csupán az alaplemezen elhelyezkedő, hanem a bővítéseket is figyelembe véve kiadódó összes tárterületet adja eredményül. A további esetleges bővítést a programok AT esetén a 15(h) megszakítás 88(h) funkciójával hatá-

rozzák meg, mely az AX regiszterben az 1 Mbájt feletti tárterület méretét adja vissza 1 Kbájtos egységekben. A képernyőn a futás során megjelenő összes többi információ közvetlenül a 11(h) megszakítás által szolgáltatott konfigurációs szóból származik, az egyes konfigurációs komponensek a megfelelő bit(ek) szűrésével nyerhetők abból. A programok korlátozott terjedelmük miatt a konfigurációs szónak csak az AT és XT esetén megegyező jelentésű elemeit dolgozzák fel.

```
1000 '╔═══════════════════════════════════════════════════════════════════╗
1005 '
1010 '                        K O N F I G U R . B A S
1020 '
1030 '  Funkciója : a főbb konfigurációs paraméterek megjelenítése.
1035 '
1040 '╚═══════════════════════════════════════════════════════════════════╝
1050 '
1060 DEFINT Q                                   'Q "dummy" egész érték.
1070 KEY OFF : CLS
1080 PRINT : PRINT
1090 PRINT "FIGYELMEZTETÉS :" : PRINT
1100 PRINT "A 60000-es címtől kezdődően betöltött megszakításhívó ru-"
1110 PRINT "tin felülírását elkerülendő a GWBASIC-et GWBASIC /M:60000"
1120 PRINT "paranccsal kell indítania. Ha nem így hívta a GWBASIC-et,"
1130 PRINT "akkor üsse le az <s> billentyűt, egyébként egy tetszőle-"
1140 PRINT "ges másik billentyűt !"
1150 '
1160 Z$ = ""
1170 WHILE Z$ = ""                              'Helytelen indítás
1180    Z$ = INKEY$                             'esetén a program
1190 WEND                                       'leállítása.
1200 IF Z$ = "s" OR Z$ = "S" THEN CLS:END
1210 '
1220 GOSUB 9000                                 'Megszakításhívó rutin
1230                                            'betöltése.
1240 CLS
1250 '
2000 '
2010 '          P R O G R A M T Ö R Z S
2020 '
2030 '
2040 DEF SEG = &HF000                           'BIOS szegmensre váltás.
2050 EZEGYATX = 0
2060 IF PEEK(&HFFFE) = &HFC THEN EZEGYATX = 1
2070 '
2080 LOCATE 5,1
2090 PRINT"A PC konfiguráció legfontosabb paraméterei:"
2100 PRINT
2110 PRINT"┌──────────────────────────────────────────────────────────┐"
2120 PRINT"│                                                          │"
2130 PRINT"│   A PC típusa                :                           │"
2140 PRINT"│   A mikroprocesszor típusa   :                           │"
2150 PRINT"│   RAM-memória mérete         :                           │"
2160 PRINT"│   RAM bővítés mérete         :                           │"
2170 PRINT"│   Videoüzemmód               :                           │"
2180 PRINT"│   Hajlékonylemez egység      :                           │"
2190 PRINT"│   RS232 kártya               :                           │"
2200 PRINT"│   Nyomtató kártya            :                           │"
2210 PRINT"│                                                          │"
2220 PRINT"└──────────────────────────────────────────────────────────┘"
2230 '
2300 LOCATE  9,31                               'Ha a F000:FFFE bájt
2310 IF PEEK(&HFFFE) <> &HFF THEN GOTO 2330     'tartalma
2320    PRINT "PC" : GOTO 2400                  '         FF : normál PC,
2330 IF PEEK(&HFFFE) <> &HFE THEN GOTO 2350     '         FE : PC/XT,
2340    PRINT "XT" : GOTO 2400                  '         FC : PC/AT,
2350 IF PEEK(&HFFFE) <> &HFC THEN GOTO 2380     'egyéb esetekben a gép-
2360    PRINT "AT" : GOTO 2400                  'típus nem azonosítható.
2370 '
2380 PRINT "a F000:FFFE bájt alapján nem azonosítható"
2390 '
```

```
2400 DEF SEG                                        'BASIC szegmensre váltás.
2410 LOCATE 10,31
2420 IF EZEGYAT% <> 1 THEN GOTO 2440                'A PC és PC/XT procesz-
2430    PRINT "80286" : GOTO 2500                   'szora INTEL 8088, az
2440 PRINT "8088"                                   'AT-é INTEL 80286.
2450 '
2500 LOCATE 11,30                                   'Tárkapacitás 1 MB alatt.
2510 MSZX = &H12                                    '12(h) megszakítás.
2520 '
2530 CALL RCIM(MSZX,RAMHIX,RAMLOX,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q)
2540 PRINT RAMHIX*256+RAMLOX; " KB"
2550 '
2600 LOCATE 12,30                                   'AT esetén 1 MB feletti
2610 IF EZEGYAT% <> 1 THEN GOTO 2700                'memória megállapítás
2620 MSZX = &H15                                    'fun., 15(h) megszakítás.
2630 RAMHIX = &H88
2640 CALL RCIM(MSZX,RAMHIX,RAMLOX,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q)
2650 PRINT RAMHIX*256+RAMLOX; " KB 1 MB felett"
2670 '
2700 LOCATE 13,3'                                   'Konfiguráció beolvasás
2710 MSZX = &H11                                    '11(h) megszakítással.
2720 '
2730 CALL RCIM(MSZX,KONFHIX,KONFLOX,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q)
2740 IF (KONFLOX AND 48) <>  0 THEN GOTO 2760
2750    PRINT "definiálatlan" : GOTO 2810
2760 IF (KONFLOX AND 48) <> 16 THEN GOTO 2780
2770    PRINT "40*25 kar. színes kártya" : GOTO 2810
2780 IF (KONFLOX AND 48) <> 32 THEN GOTO 2800
2790    PRINT "80*25 kar. színes kártya" : GOTO 2810
2800 PRINT "80*25 kar. mono kártya"
2810 LOCATE 14,30                                   'A konfiguráció
2820 PRINT INT(KONFLOX/64)+1                        'a 11(h) megszakí-
2830 LOCATE 15,30                                   'tás által beállí-
2840 PRINT INT(KONFHIX/2) AND 3                     'tott AX regisz-
2850 LOCATE 16,30                                   'terből olvasható
2860 PRINT INT(KONFHIX/64)                          'ki.
2870 LOCATE 20, 1
2880 PRINT "Nyomjon meg egy billentyűt ..."
2890 '
3000 Z$ = ""
3010 WHILE Z$ = ""                                  'Kiszállás előtt egy
3020    Z$ = INKEY$                                 'billentyű megnyomá-
3030 WEND                                           'sára várakozás.
3040 CLS
3050 END
3060 '
9000 '
9010 '  A MEGSZAKíTASHíVó RUTIN INICIALIZALASA
9020 '     BE: -
9030 '     KI: RCIM a Rutin CIMe
9040 '
9050 '
9060 RCIM = 60000!                  'A rutin kezdőcíme a BASIC szegmensben.
9070 DEF SEG                        'A BASIC szegmens beállítása.
9080 RESTORE 9140
9090 FOR IX = 0 TO 160                                  'A rutin betöltése a
9100    READ XX : POKE RCIM + IX, XX                    'tárba.
9110 NEXT
9120 RETURN
9130 '
9140 DATA    85, 139, 236,  30,   6, 139, 118,  30, 139,   4, 232, 140
9150 DATA     0, 139, 118,  12, 139,  60, 139, 118,   8, 139,   4,  61
```

```
9160 DATA  255, 255, 117,   2, 140, 216, 142, 192, 139, 118,  28, 138
9170 DATA   36, 139, 118,  26, 138,   4, 139, 118,  24, 138,  60, 139
9180 DATA  118,  22, 138,  28, 139, 118,  20, 138,  44, 139, 118,  18
9190 DATA  138,  12, 139, 118,  16, 138,  52, 139, 118,  14, 138,  20
9200 DATA  139, 118,  10, 139,  52,  85, 205,   0,  93,  86, 156, 139
9210 DATA  118,  12, 137,  60, 139, 118,  28, 136,  36, 139, 118,  26
9220 DATA  136,   4, 139, 118,  24, 136,  60, 139, 118,  22, 136,  28
9230 DATA  139, 118,  20, 136,  44, 139, 118,  18, 136,  12, 139, 118
9240 DATA   16, 136,  52, 139, 118,  14, 136,  20, 139, 118,   8, 140
9250 DATA  192, 137,   4,  88, 139, 118,   6, 137,   4,  88, 139, 118
9260 DATA   10, 137,   4,   7,  31,  93, 202,  26,   0,  91,  46, 136
9270 DATA   71,  66, 233, 108, 255
```

```
/*
/*  ┌──────────────────────────────────────────────────────────────┐
/*  │                                                              │
/*  │                        K O N F I G U R . C                   │
/*  │                                                              │
/*  │  Funkciója : a PC konfiguráció megjelenítése.                │
/*  │                                                              │
/*  │  FONTOS    : a memóriakezelő "PEEKB" a PEEKPOKE.ASM forrásál-│
/*  │              lományban helyezkedik el, ezért a LINK parancs  │
/*  │              helyes formája a következő :                    │
/*  │                                                              │
/*  │                LINK KONFIGUR PEEKPOKE                        │
/*  │                                                              │
/*  └──────────────────────────────────────────────────────────────┘
/*

extern short int PEEKB();     /* Az assembler rutin beszerkesztendő. */

#include <dos.h>
#include <io.h>

#define  FALSE    0
#define  TRUE     1
#define  MM     0x07                          /* Normál megjelenítés. */

#define byte unsigned char

/*  ┌──────────────────────────────────────────────────────────┐
/*  │ AKTOLDAL : AZ AKTUALIS KÉPERNYŐOLDAL MEG-                │
/*  │            HATAROZASA                                    │
/*  └──────────────────────────────────────────────────────────┘

byte AKTOLDAL()

{
union REGS regiszter;                    /* Regiszter union deklaráció. */

regiszter.h.ah = 0x0F;                        /* Video olvasás funkció. */
int86(0x10, &regiszter, &regiszter);          /* 10(h) megszakítás.     */
return(regiszter.h.bh);                       /* Aktuális képernyőoldal. */
}

/*  ┌──────────────────────────────────────────────────────────┐
/*  │ KURPOZ : A KURZOR POZÍCIONALASA A KIVA-                  │
/*  │          LASZTOTT KÉPERNYŐOLDALON                        │
/*  └──────────────────────────────────────────────────────────┘

void KURPOZ(oldal, oszlop, sor)
int oldal;
int oszlop;
int sor;

{
union REGS regiszter;                    /* Regiszter union deklaráció. */

regiszter.h.ah = 0x02;                        /* Kurzor beállítása az */
regiszter.h.bh = oldal;                       /* aktuális oldalon az  */
regiszter.h.dh = sor;                         /* átadott sor- és osz- */
regiszter.h.dl = oszlop;                      /* loppozícióra.        */
int86(0x10, &regiszter, &regiszter);          /* 10(h) megszakítás.   */
}

/*  ┌──────────────────────────────────────────────────────────┐
/*  │ GORGETES_FEL : MEGADOTT KÉPERNYŐTARTOMANY                │
/*  │                FELFELÉ GORGETÉSE, ill.                   │
/*  │                TORLÉSE                                   │
/*  └──────────────────────────────────────────────────────────┘
```

```
void GORGETES_FEL(gsor, szin, bfo, bfs, jao, jas)

int gsor;             /* (G)örgetendő (sor)ok száma.                  */
int szin;             /* A törölt terület (szín)e.                    */
int bfs ;             /* Képernyőtartomány (b)al  (f)első (s)ora.     */
int bfo ;             /*                   (b)al  (f)első (o)szlopa.  */
int jas ;             /*                   (j)obb (a)lsó  (s)ora.     */
int jao ;             /*                   (j)obb (a)lsó  (o)szlopa.  */

{
union REGS regiszter;                 /* Regiszter union deklaráció.  */

regiszter.h.ah = 0x06;                /* Görgetés funkció.            */
regiszter.h.al = gsor;                /* Görgetendő sorok száma.      */
regiszter.h.bh = szin;                /* Az üres sorok színe.         */
regiszter.h.ch = bfs ;                /* Ablak-                       */
regiszter.h.cl = bfo ;                /*        koor-                 */
regiszter.h.dh = jas ;                /*             dinã-            */
regiszter.h.dl = jao ;                /*                  tãk.        */

int86(0x10, &regiszter, &regiszter);          /* 10(h) megszakítás.   */
}

/*  ┌──────────────────────────────────────┐                          */
/*  │ CLRSCR : KÉPERNYŐ TÖRLÉSE            │                          */
/*  └──────────────────────────────────────┘                          */

void CLRSCR()

{
GORGETES_FEL(0, NM, 0, 0, 79, 24);        /* Törlés görgetéssel.      */
KURPOZ(AKTOLDAL(), 0, 0);                 /* Kurzor "home" pozícióba. */
}

/*  ┌──────────────────────────────────────┐                          */
/*  │ KIIRATAS : A PC KONFIGURÁCIÓ MEGÁLLAPÍTÁ-                      */
/*  │            SA ÉS MEGJELENÍTÉSE       │                          */
/*  └──────────────────────────────────────┘                          */

void KIIRATAS()

{
union REGS regiszter;                 /* Regiszter union deklaráció.  */
byte  EZ_EGY_AT;

EZ_EGY_AT = (PEEKB(0xF000, 0xFFFE) == 0xFC) ? TRUE : FALSE;

printf("");
printf("A PC konfigurációs legfontosabb paraméterei:");
printf("  ┌──────────────────────────────────────────────────────┐");
printf("  │                                                      │");
printf("  │   A PC típusa                 :                      │");
printf("  │   A mikroprocesszor típusa    :                      │");
printf("  │   RAM-memória mérete          :                      │");
printf("  │   RAM bővítés mérete          :                      │");
printf("  │   Videoüzemmód                :                      │");
printf("  │   Hajlékonylemez egység       :                      │");
printf("  │   RS232 kártya                :                      │");
printf("  │   Nyomtató kártya             :                      │");
printf("  │                                                      │");
printf("  └──────────────────────────────────────────────────────┘");
printf("Nyomjon meg egy billentyűt ...");
```

```
KURPOZ(AKTOLDAL(), 30, 9);
switch(PEEKB(0xF000, 0xFFFE))                    /* Ha a F000:FFFE bájt      */
   {                                             /* tartalma                 */
   case 0xFF : printf("PC");                     /*          FF : normál PC, */
               break;                            /*          FE : PC/XT,     */
   case 0xFE : printf("XT");                     /*          FC : PC/AT,     */
               break;                            /* egyéb esetekben a gép-   */
   case 0xFC : printf("AT");                     /* típus nem azonosítható.  */
               break;
   default   : printf("a F000:FFFE bájt alapján nem azonosítható");
               break;
   }
KURPOZ(AKTOLDAL(), 30, 10);
if (EZ_EGY_AT)
   printf("INTEL 80286");                 /* A PC és PC/XT processzora  */
else                                      /* INTEL 8088, az AT-é 80286. */
   printf("INTEL 8088");
KURPOZ(AKTOLDAL(), 30, 11);
int86(0x12, &regiszter, &regiszter);
printf("%d KB",regiszter.x.ax);
KURPOZ(AKTOLDAL(), 30, 12);
if (EZ_EGY_AT)
   {
   regiszter.h.ah = 0x88;                                /* 1 MB feletti mem.  */
                                                         /* megállapítása fun. */
   int86(0x15, &regiszter, &regiszter);                  /* 15(h) megszakítás. */
   printf("%d KB iMB felett", regiszter.x.ax);
   }
else
   printf("csak PC/AT esetén lehetséges");
KURPOZ(AKTOLDAL(), 30, 13);
int86(0x11, &regiszter, &regiszter);                     /* 11(h) megszakítás */
switch(regiszter.x.ax & 48)                              /* a konfiguráció    */
   {                                                     /* megállapítására.  */
   case  0 : printf("definiálatlan");
             break;
   case 16 : printf("40*25 kar. színes kártya");
             break;
   case 32 : printf("80*25 kar. színes kártya");
             break;
   case 48 : printf("80*25 kar. mono kártya");
             break;
   }
KURPOZ(AKTOLDAL(), 30, 14);                              /* A konfiguráció    */
printf("%d", (regiszter.x.ax >> 6 & 3) + 1);             /* a 11(h) megszakí- */
KURPOZ(AKTOLDAL(), 30, 15);                              /* tás által beállí- */
printf("%d",  regiszter.x.ax >> 9 & 0x03);               /* tott AX regisz-   */
KURPOZ(AKTOLDAL(), 30, 16);                              /* terből olvasható  */
printf("%d",  regiszter.x.ax >> 14);                     /* ki.               */
printf("");
}

/*                                                                          */
/*  ┌──────────────────────────────────────────────────────────────────┐    */
/*  │                          F Ő P R O G R A M                       │    */
/*  └──────────────────────────────────────────────────────────────────┘    */

void main()

{
CLRSCR();
KIIRATAS();
getch();                                 /* Várakozás egy billentyűre. */
CLRSCR();
}
```

```
{ ╔═══════════════════════════════════════════════════════════════════════╗ }
{ ║                                                                       ║ }
{ ║                          K O N F I G U R . P A S                      ║ }
{ ║                                                                       ║ }
{ ║  Funkciója : a PC konfiguráció megjelenítése.                         ║ }
{ ║                                                                       ║ }
{ ╚═══════════════════════════════════════════════════════════════════════╝ }

program KONFIGUR;

type dosreg = record
                ax, bx, cx, dx, bp, di, si, ds, es, flags : integer;
              end;

var z : char;                  { Csak a várakozási állapotokhoz szükséges. }

{ ┌──────────────────────────────────────────────────┐ }
{ │ KIIRATAS : A PC KONFIGURÁCIÓ MEGALLAPíTASA       │ }
{ │            ÉS MEGJELENíTÉSE                      │ }
{ └──────────────────────────────────────────────────┘ }

procedure KIIRATAS;

var EZ_EGY_AT : boolean;
    regiszter : dosreg;                  { Regiszter rekord deklaráció. }

begin
 if mem[$F000:$FFFE] = $FC then EZ_EGY_AT := true              { AT-e  }
                           else EZ_EGY_AT := false;            { gép ? }
gotoxy(1, 5);
writeln('A PC konfiguráció legfontosabb paramétereI:');
writeln;
writeln('┌──────────────────────────────────────────────────────────────────────┐');
writeln('│                                                                      │');
writeln('│   A PC típusa                :                                       │');
writeln('│   A mikroprocesszor típusa   :                                       │');
writeln('│   RAM-memória mérete         :                                       │');
writeln('│   RAM bővítés mérete         :                                       │');
writeln('│   Videoüzemmód               :                                       │');
writeln('│   Hajlékonylemez egység      :                                       │');
writeln('│   RS232 kártya               :                                       │');
writeln('│   Nyomtató kártya            :                                       │');
writeln('│                                                                      │');
writeln('└──────────────────────────────────────────────────────────────────────┘');

gotoxy(31, 9);                         { Ha a F000:FFFE bájt tartalma }
case mem[$F000:$FFFE] of               {          FF : normál PC,     }
   $FF : write('PC');                  {          FE : PC/XT,         }
   $FE : write('XT');                  {          FC : PC/AT,         }
   $FC : write('AT');                  { egyéb esetekben a géptípus   }
                                       { nem azonosítható.            }

   else write('a F000:FFFE bájt alapján nem azonosítható')
end;

gotoxy(31, 10);
if EZ_EGY_AT then write('INTEL 80286')    { A PC és PC/XT processzora  }
             else write('INTEL 8088');    { INTEL 8088, az AT-é 80286. }

gotoxy(31, 11);
intr($12, regiszter);
write(regiszter.ax);
write(' KB');
```

```
gotoxy(31, 12);
if (EZ EGY_AT) then
   begin
   regiszter.ax := $8800;                    { 1 MB feletti mem.  }
   intr($15, regiszter);                     { megállapítása fun. }
   write(regiszter.ax);                      { 15(h) megszakítás. }
   write(' KB 1 MB felett');
   end
else
   write('csak PC/AT esetén lehetséges');

gotoxy(31, 13);
intr($11, regiszter);
case regiszter.ax and 48 of                  { 11(h) megszakítás }
    0 : writeln('definiálatlan');            { a konfiguráció    }
   16 : writeln('40*25 kar. színes kártya'); { megállapítására. }
   32 : writeln('80*25 kar. színes kártya');
   48 : writeln('80*25 kar. mono kártya');
end;

gotoxy(31, 14);                              { A konfiguráció    }
write((regiszter.ax shr 6 and 3) + 1);       { a 11(h) megszakí- }
gotoxy(31, 15);                              { tás által beállí- }
write(regiszter.ax shr 9 and 3);             { tott AX regisz-   }
gotoxy(31, 16);                              { terből olvasható  }
write(regiszter.ax shr 14);                  { ki.               }

gotoxy(1, 20);
write('Nyomjon meg egy billentyűt ...');
end;

{                                            }
{          FŐPROGRAM                         }
{                                            }

begin
   clrscr;
   KIIRATAS;
   read(kbd, z);                  { Várakozás egy billentyűre. }
   clrscr;
end.
```

1.3. A képernyő vezérlése

A számítógépen futó programok minősítésénél az objektív szempontok (a futás gyorsasága, a számítások pontossága, a program mérete stb.) mellett igen nagy súllyal esnek latba az olyan, viszonylag szubjektív szempontok is, mint például a képi megjelenítés gyorsasága, a képernyő felépítése, áttekinthetősége, a grafika és a színek használata stb. A felhasználó egy programról az első benyomását általában a képernyőn keresztül szerzi. Aligha szorul bizonyításra, hogy azonos tudású programok közül hamarabb „elkel" a „szebbik". A következőkben azokat a BIOS megszakításokat, ill. funkciókat vizsgáljuk meg, amelyek ebben segítik a programozót.

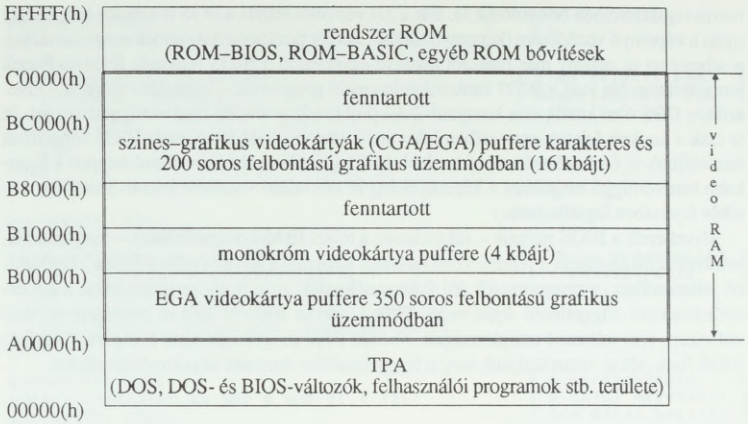
A képernyőt vezérlő rutinok mindegyikét a 10(h) megszakításon keresztül kell meghívni. A meghívó program és a rutinok közötti adatcsere most is a processzor meghatá-

rozott regisztereiben bonyolódik le. Bár a 2.0 verziótól felfelé a DOS is számos lehetőséget nyújt a képernyő vezérlésére (képernyő törlése, kurzor pozícionálása, színek megválasztása), a sebességet és néhány más funkciót tekintve egyértelmű a BIOS funkciók fölénye. Ennek természetesen ára van: a BIOS funkciókat használó programok – legalábbis részben – elveszítik a DOS által kínált, és a kompatibilitási problémákat feloldó hardverfüggetlenséget. Itt is csak a konkrét feladat ismeretében lehet arról dönteni, hogy DOS vagy BIOS funkciókat használjunk-e. (A hardverhez legközelebb álló, leghatékonyabb, következésképpen a leginkább hardverfüggő megoldást a képernyőkártyák közvetlen vezérlése jelenti. Ezzel egy későbbi fejezetben foglalkozunk.)

Mivel ezek a BIOS rutinok – feltételezve a teljes IBM-kompatibilitást – megfelelő illeszkedést biztosítanak a rendszerben használt képernyőkártyák (videokártyák) különböző jellemzőihez, a programozók jól felhasználhatják arra, hogy programjaikat a képernyőkártyáktól függetlenül írják meg. Ehhez persze ismerni kell a rendszerben lévő videokártyá(k) jellemző tulajdonságait. Mielőtt tehát megvizsgálnánk a képernyővezérlő BIOS funkciókat, ismerkedjünk meg a leggyakrabban használt képernyőkártyákkal.

1.3.1. Képernyőkártyák

Előljáróban megjegyezzük, hogy a képernyőkártyák olyan, nyomtatott áramkörű lemezek, amelyek a számítógép alaplapján lévő bővítő csatlakozók valamelyikébe helyezhetők be. Feladatuk az, hogy a számítógéphez csatlakoztatott megjelenítő egységen (video display, monitor, képernyő) vizuális formában megjelenítsék a kártya RAM tárában lévő szöveges és/vagy grafikus információt. A megjelenítést végző integrált áramkört video adapternek, a képernyőkártyát pedig videokártyának is nevezik. A képi információt tároló RAM, amelybe, ill. amelyből a programok úgyanúgy írhatnak, ill. olvashatnak, mint a gép belső, operatív RAM-jába, nem az alaplapon, hanem a képernyőkártyán helyezkedik el (ezért video RAM-nak is nevezik). A video RAM-nak a videokártya típusától függően, a gép tervezői által meghatározott címtartományban kell elhelyezkednie. A video RAM tárbeli elhelyezkedését és felosztását a 2. ábra mutatja.



2. ábra: A video RAM helye és felosztása

Az IBM PC/XT/AT számítógépekben használatos képernyőkártyák alapvetően két fő csoportra oszthatók: monokróm és színes-grafikus kártyákra. (Vannak „közbenső" kártyák is: ezek képesek a grafikus megjelenítésre, de a színeket nem ismerik.) A következőkben a képernyőkártyák különböző típusaival ismerkedünk meg.

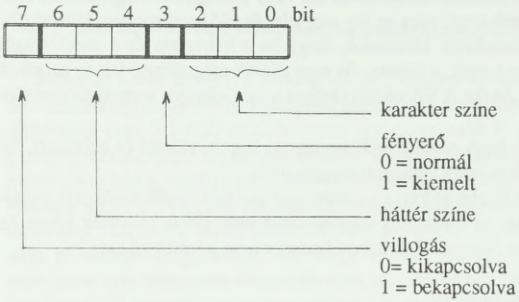
Monokróm képernyőkártya

A monokróm képernyőkártyával egy monokróm monitoron szöveges (karakteres) információk jeleníthetők meg. A monokróm jelző itt azt jelenti, hogy a képernyőre kerülő karakterek csak egyféle színűek lehetnek. A gyakorlatban a karakterek általában fekete háttérben zöld vagy sárga színűek. A monokróm kártyákkal egy képernyőoldalon 25 sorba és 80 oszlopba írhatók karakterek. Azt mondjuk, hogy a videokártya felbontása 25 sor * 80 oszlop. Ezzel a kártyával tehát összesen 25 * 80 = 2000 karakter jeleníthető meg. Képernyőoldalon a képernyőn egyidejűleg megjeleníthető karakterek összességét értjük (mint majd látni fogjuk, vannak olyan videokártyák, amelyekkel egyidejűleg több képernyőoldal is létrehozható; de egy időben természetesen mindig csak egy látható közülük). A sorok és oszlopok számozása a képernyő bal felső sarkában, 0-val kezdődik. Ennek megfelelően 0 – 24. sorról és 0 – 79. oszlopról beszélünk.

A 2000 karakterpozíció mindegyikén az IBM standard karakterkészletét alkotó 256 karakter egyike jeleníthető meg (a karakterkészletet a III. kötet tartalmazza). Egy karakter megjelenítéséhez 2 bájtnyi információ szükséges: a video RAM-ban az alacsonyabb (páros) címen lévő bájt a megjelenítendő karakter ASCII-kódját, a rákövetkező (páratlan) címen lévő bájt pedig a megjelenítés módját határozza meg. Ez utóbbi bájtot attribútum bájtnak nevezik. Abból, hogy egy képernyőpozíción lévő karakter megjelenítéséhez

2 bájtra van szükség, következik, hogy a monokróm kártyának egy képernyőoldalnyi információ tárolásához 2 * 2000 = 4000 bájt nagyságú címterületre van szüksége. Amint a 2. ábrán látható, ez a címterület a B0000(h) abszolút tárcímen kezdődik, és 4 kbájtot foglal el (a 4 kbájt = 4096 bájt tárterület 96 bájttal több, mint a 4000 bájtos tárigény; ez a 96 bájt kihasználatlan marad).

A 3. ábra az attribútum bájt bitjeinek jelentését, a 4. ábra pedig az értelmezhető bitkombinációkat, illetve az ezeknek megfelelő megjelenítési módokat mutatja be.



3. ábra: Az attribútum bájt bitjeinek jelentése monokróm képernyőkártya esetén

bitkombinációk								érték		normál	inverz	kiemel	villog	aláhúz
7	6	5	4	3	2	1	0	dec.	hexa					
0	0	0	0	0	0	0	1	1	01(h)	*	–	–	–	*
0	0	0	0	0	1	1	1	7	07(h)	*	–	–	–	–
0	0	0	0	1	0	0	1	9	09(h)	*	–	*	–	*
0	0	0	0	1	1	1	1	15	0F(h)	*	–	*	–	–
0	1	1	1	0	0	0	0	112	70(h)	–	*	–	–	–
1	0	0	0	0	0	0	1	129	81(h)	*	–	–	*	*
1	0	0	0	0	0	1	0	130	82(h)	*	–	–	*	–
1	0	0	0	1	1	1	1	143	8F(h)	*	–	*	*	–
1	1	1	1	0	0	0	0	240	F0(h)	–	*	–	*	–

4. ábra: Az attribútum bájt értelmezhető bitkombinációi és a megfelelő megjelenítési módok monokróm képernyőkártya esetén

Az attribútum bájt felépítése meglehetősen szokatlan: mind a karakter, mind a háttér színére 3–3 bitet foglal le, jóllehet tudjuk, hogy szín nem is választható meg (maximum a normál és az inverz megjelenítés közül választhatunk). Ennek az a magyarázata, hogy karakteres üzemmódban a színes megjelenítés is ugyanezt az attribútum bájtot használja a színek kijelzésére, ahol a most feleslegesnek tűnő bitek önálló szerephez jutnak. Egy – látszólag felesleges – bit azonban monokróm kép esetén is ki van használva: bizonyos bitkombinációk esetén a 0. bit 1 értéke a vonatkozó karaktert aláhúzva jeleníti meg; ez olyan képesség, amellyel a színes megjelenítés nem rendelkezik.

Az 4. ábrában az „értelmezhető" bitkombináció kifejezést használtuk. Ezen azt értjük, hogy az ábrán fel nem tüntetett bitkombinációk vagy a megadott megjelenítési módok valamelyikével egyeznek meg, vagy az így megjelenítendő karakter nem látható (fekete alapon fekete színű karakter). Hihetnénk, hogy ha a háttérszín és a karakterszín bitjei rendre mind 0-k vagy 1-esek, a képernyőn nem jelenik meg látható jel; ez azonban csak akkor igaz, ha a bitek értéke 0. Ellenkező esetben a megjelenítés normál módon (fekete alapon) történik.

Az ábrából az is látszik, hogy normál módban nincs kiemelt, villogó és aláhúzott, inverz módban pedig kiemelt és/vagy aláhúzott megjelenítés.

Megjegyezzük, hogy az itt elmondottak a színes-grafikus képernyőkártyákra is vonatkoznak, ha azok monokróm és karakteres üzemmódban működnek. (Kivételt képez ez alól az, hogy színes-grafikus kártyával aláhúzott karakter nem jeleníthető meg.)

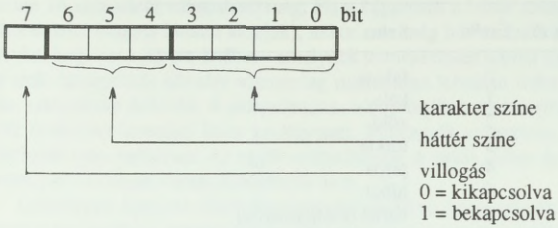
Színes-grafikus képernyőkártya

A színes-grafikus képernyőkártya (angol nevén Color/Graphics Adapter, CGA) az IBM karakterkészletébe tartozó karakterek színes megjelenítésén túl grafikus képek megjelenítésére is képes.

Karakteres (szöveges) módban kétféle felbontással dolgozhat: 80 * 25 vagy 40 * 25 karakteres felbontással. 40 * 25-ös felbontás esetén a karakterek szélessége kétszeresére nő. Az 2. ábrán látható, hogy a CGA kártya által használt video RAM a B8000(h) – BC000(h) között, 16 kbájtos címterületen helyezkedik el. Ebből könnyen kiszámítható, hogy 80 * 25 karakteres felbontás esetén ezen a tárterületen egyidejűleg 4 képernyőoldalnyi (4 * 4 kbájt), 40 * 25-ös felbontás esetén pedig 8 képernyőoldalnyi (8 * 2 kbájt) információ építhető fel. (Mivel a felhasználói programok ehhez a tárterülethez – amely egyébként ugyanolyan RAM, mint a többi – közvetlenül hozzáférhetnek, az egyes képernyőoldalakon megjelenítendő információk szoftverből előre felépíthetők. Ez azt jelenti, hogy amíg az egyik képernyőoldal aktív, más szóhasználattal aktuális, azaz látható, a többi a tárban a felhasználó számára láthatatlanul kialakítható, és már a kész kép kerülhet a képernyőre. Ez a technika, amelyet tártérképes vagy tárba ágyazott megjelenítési technikának (angolul memory mapped display technique) neveznek, rendkívül gyors képváltásokat tesz lehetővé.)

A sorok és oszlopok számozása most is a képernyő bal felső sarkából, a 0,0 koordinátájú karakterpozícióból indul.

A színes-grafikus videokártyánál az attribútum bájt elsődlegesen a karakter és a háttér színének megválasztására szolgál. Az attribútum bájt felépítését és az egyes bitek jelentését az 5. ábra mutatja.



5. ábra: Az attribútum bájt bitjeinek jelentése színes-grafikus képernyőkártya esetén

Amint az ábrán látható, a karakter (előtér) színe 4 bit kombinációjával állítható be, ami elméletileg összesen 16 különböző színt jelent. Ezzel szemben a háttér színének beállításához csak 3 bit áll rendelkezésre, így ehhez csak 8 szín közül lehet választani. (Megjegyezzük, hogy egyes szakirodalmi források a karakter legmagasabb helyértékű színbitjét (a bájt 4. bitjét) a monokróm kártya attribútum bájtjához hasonlóan itt is fényerőbitnek tekintik. A kétféle megközelítés eredménye azonos: a karakter színét alapvetően az alsó 3 bit (0 – 2. bitek) határozza meg, a negyedik bit ezeket a színeket csak módosítja pl. kékről világoskékre. Az, hogy a negyedik bit bekapcsolásával létrehozott módosított szín mennyivel fényesebb mint az, amelyiknél a 4. bit értéke 0, egyrészt magától a színtől, másrészt pedig a megjelenítő monitor minőségétől is függ.)

A 4 biten létrehozható bitkombinációkat és az ezeknek megfelelő színeket a 6. ábra tartalmazza. A színek három alapszín összegző keverésével állnak elő. Mindegyik színnek egy bit felel meg az attribútum bájtban. A karakter színére vonatkozóan pl.:

2. bit = piros (R)
1. bit = zöld (G)
0. bit = kék (B)

Bitkombináció	Érték	Szín
0000	0	fekete
0001	1	kék
0010	2	zöld
0011	3	türkiz
0100	4	piros
0101	5	bíbor
0110	6	barna (esetleg sárga)
0111	7	világosszürke (esetleg fehér)
1000	8	sötétszürke (esetleg fekete)
1001	9	világoskék
1010	10	világoszöld
1011	11	világostürkiz
1100	12	világospiros
1101	13	világosbíbor
1110	14	sárga (esetleg világossárga)
1111	15	fehér (esetleg kiemelt fehér)

6. ábra: Az attribútum bájt színbitjeinek kombinációi és a színek színes-grafikus képernyőkártya esetén

A színes-grafikus kártya monokróm kártyaként is használható (pl. ha a monitorunk nem színes). Ebben az esetben az attribútum bájt jelentése ugyanaz, mint monokróm kártya esetén azzal a megszorítással, hogy aláhúzott karakterek nem jeleníthetők meg.

Grafikus üzemmódban ugyancsak kétféle felbontás, az ún. nagy felbontás és közepes felbontás közül lehet választani: a nagy felbontásnál 640 * 200, a közepes felbontásnál 320 * 200 képpont jeleníthető meg a képernyőn. (Az elnevezésekből logikusan következik, hogy létezik az ún. kis felbontás – 160 * 100 képpont – is, azonban ezt a ROM-BIOS nem támogatja.) A képpontok számozása mindkét esetben a képernyő bal felső sarkában, a 0,0 koordinátájú képpontpozícióból indul.

Mivel grafikus üzemmódban minden egyes képpontot külön-külön kell kezelni, a szöveges módban értelmezett karakter mint előre definiált képpontegyüttes itt nem értelmezhető. Ebből következik, hogy a hozzájuk tartozó attribútum bájt sem értelmezhető. Karakterek természetesen grafikus üzemmódban is megjeleníthetők, de ehhez a karaktereket alkotó minden egyes képpont megjelenítéséről külön-külön kell gondoskodni.

A nagy felbontású, 640 * 200 képpontos módban a színek megválasztása a képpontok jelentős tárigénye (16 kbájt) miatt igen korlátozott: csak a háttér és a karakter színének megkülönböztetésére van lehetőség, ami egyet jelent a monokróm megjelenítéssel.

A közepes felbontású, 320 * 200 képpontos módban – a kisebb tárigény (8 kbájt) következtében – a rendelkezésre álló 16 kbájtos video RAM-on belül minden egyes képpont megjelenítéséhez két bit használható. Mivel két biten négy különböző érték ábrázolható, lehetővé válik, hogy egy képpont négy különböző szín valamelyikén jelenjen meg. A négy érték egyike – egy porton keresztül (amelyről a későbbiekben még bőven lesz szó) – a háttér színének megválasztását teszi lehetővé. A porton keresztül a háttér

színe 16 szín közül választható meg. Ha a képpontot a háttér színétől meg akarjuk különböztetni (vagyis láttatni akarjuk), akkor erre még három érték áll rendelkezésünkre. Azért, hogy ezt a három értéket – amelynek természetesen három szín feleltethető meg – a szűk tárkapacitás ellenére viszonylag rugalmasan lehessen felhasználni, az IBM két, ún. színpalettát definiált. A színpalettát az előbb említett porton (illetve a megfelelő BIOS funkción) keresztül lehet kiválasztani. Mindegyik színpaletta három-három, előre definiált színt tartalmaz. Az egyik színpalettával a türkiz, bíbor és fehér, a másikkal a zöld, piros és sárga színek jeleníthetők meg.

Lehetséges, hogy az előző fejtegetések egy kicsit bonyolultnak tűnnek (mi tagadás, nem is egyszerűek). Ezért röviden foglaljuk össze az elmondottakat (a későbbiekben ennek még hasznát vesszük). A közepes felbontású üzemmódban egy képpont négy különböző színnel ábrázolható: ezek egyike a háttér színét adja meg (16 lehetséges szín közül), a másik három pedig azt, hogy a képpont két, előre meghatározott, de általunk kiválasztott palettához tartozó három szín melyikén jelenjen meg.

Megemlítjük még az EGA (Enhanced Graphics Adapter) betűszóval rövidített, bővített grafikus tulajdonságokkal rendelkező videokártyát, amely 640 * 350 képpont felbontású, 16 színű grafikus megjelenítést tesz lehetővé. Az EGA kártya mind a CGA, mind a monokróm kártyával felfelé kompatibilis.

Amint már említettük, a színes-grafikus kártyák video RAM-ja a B8000(h) tárcímen kezdődik (az EGA kártyáé 350 soros felbontás esetén az A0000(h) címen). A kártyák által elfoglalt tárterület nem lapolja át a monokróm videokártya által igénybe vett tárterületet. Ez azt jelenti, hogy a kétféle kártya egymás működését nem zavarja, vagyis egy rendszer egyidejűleg monokróm és színes-grafikus videokártyát is tartalmazhat, és ezek közül mindig az igény szerinti lehet az aktív.

Monokróm Hercules-grafikus képernyőkártya

A Hercules-grafikus képernyőkártya szöveges üzemmódban megegyezik az IBM monokróm képernyőkártyájával azzal a különbséggel, hogy az utóbbitól eltérően nem egy, hanem két képernyőoldalt képes megjeleníteni. A nevéből következik, hogy természetesen grafikus módban is működtethető: két képernyőoldalon oldalanként 720 * 348 képpont felbontású grafika jeleníthető meg. Mivel azonban ez a kártya a grafikus üzemmódban nem IBM-kompatibilis, a grafikus képességei BIOS funkciókkal nem használhatók ki. A BIOS a Hercules-grafikus kártyát szöveges üzemmódban ugyanúgy kezeli, mint az egyetlen képernyőoldalon 80 * 25 karakter megjelenítésére képes, normál monokróm videokártyát.

1.3.2. A BIOS képernyővezérlő funkciói

A BIOS valamennyi képernyővezérlő funkciója a 16 – 10(h) számú megszakításon keresztül hívható meg. A következőkben az alábbi funkciókkal ismerkedünk meg:

0	–	00(h)	video üzemmód beállítása
1	–	01(h)	kurzor alakjának meghatározása
2	–	02(h)	kurzor pozícionálása
3	–	03(h)	kurzor pozíciójának kiolvasása
4	–	04(h)	fénytoll pozíciójának kiolvasása
5	–	05(h)	aktuális képernyőoldal kiválasztása
6	–	06(h)	szövegsorok felfelé görgetése (scroll)
7	–	07(h)	szövegsorok lefelé görgetése (scroll)
8	–	08(h)	karakter/szín kiolvasása
9	–	09(h)	karakter/szín beírása
10	–	0A(h)	karakter beírása
11	–	0B(h)	keret- és háttérszín kiválasztása (0. alfunkció)
11	–	0B(h)	színpaletta kiválasztása (1. alfunkció)
12	–	0C(h)	grafikus pont beírása
13	–	0D(h)	grafikus pont kiolvasása
14	–	0E(h)	karakter beírása
15	–	0F(h)	video üzemmód megállapítása
19	–	13(h)	karakterlánc kiírása (csak AT)

A paraméterek átadása természetesen most is a processzor regisztereiben történik. Bár az egyes funkciókat különböző paraméterekkel kell meghívni, és az általuk szolgáltatott eredmények is különbözők, van néhány általános szabály, amely az átadandó paramétereket és az ezeket átadó regisztereket egymáshoz rendeli.

A processzor a 10(h) megszakítás meghívásakor a funkció számát minden esetben az AH regiszterben várja. Ha a funkciónak egy karaktert vagy egy képpontot kell a képernyőn megjelenítenie, akkor az erre vonatkozó értéket az AL regiszterbe kell beírni.

Ha a meghívandó funkció szöveges üzemmódban képernyő koordinátákat vár, akkor az X koordináta értékét (oszlop száma) a DL, az Y koordináta értékét (sor száma) pedig a DH regiszterbe kell írni. Grafikus üzemmódban az X koordinátát a CX regiszternek, az Y koordinátát pedig a DX regiszternek kell tartalmaznia. Ha a funkció meghívásánál több képernyőoldal közül választhatunk, akkor a kívánt oldal számát a BH regiszterben kell átadni.

Elsősorban az assembly nyelven programozók számára fontos tudnivaló, hogy a megszakításokból való visszatérés után csak a BX, CX, DX regiszterek és a szegmensregiszterek tartalma nem változik meg. Ezért a megszakítások meghívása előtt a többi regiszter tartalmát – amennyiben azokra a visszatérést követően is szükség lesz – a verembe kell menteni.

A 00(h) funkció a video üzemmód, illetve – amennyiben a számítógépben több videokártya van – az aktív kártya kiválasztására szolgál. A funkció meghívásához az AH regiszterbe a 00(h) értéket, az AL regiszterbe pedig a kívánt video üzemmód kódját kell beírni. Természetesen csak olyan üzemmód választható meg, amelyet a PC-ben lévő vi-

deokártya támogat. Az egyes üzemmódoknak az alábbi kódszámok felelnek meg (zárójelben az a kártya szerepel, amely az illető üzemmódot támogatja):

kód	video üzemmód
0	40*25 karakter, szöveg, fekete/fehér (színes-grafikus)
1	40*25 karakter, szöveg, színes (színes-grafikus)
2	80*25 karakter, szöveg, fekete/fehér (színes-grafikus)
3	80*25 karakter, szöveg, színes (színes-grafikus)
4	320*200 grafikus képpont, 4 szín (színes-grafikus)
5	320*200 grafikus képpont, 4 szín (színes-grafikus) (színek fekete/fehérben)
6	640*200 grafikus képpont, 2 szín (színes-grafikus)
7	80*25 karakter, szöveg, fekete/fehér (monokróm kártya) (monokróm kártya belső üzemmódja)

Monokróm képernyőkártya esetén a video üzemmód megválasztásának valójában nincs jelentősége, mivel itt csak egyetlen üzemmód, a 80 * 25 karakter felbontású, szöveges üzemmód létezik, és ez mindig aktív.

A 00(h) funkció párja a 15 – 0F(h) funkció, amellyel a video üzemmód lekérdezhető. A funkció meghívásához az AH regiszterbe a 0F(h) értéket kell írni. A funkció a visszatérésekor az aktuális video üzemmód kódját az AL regiszterben adja vissza. Amint említettük, monokróm kártya esetén ennek a kódja mindig 7. A funkció az üzemmód kódja mellett az AH regiszterben az egy sorban megjeleníthető karakterek számát, a BH regiszterben pedig az aktuális képernyőoldal számát adja vissza.

Miután kiválasztottuk a kívánt video üzemmódot, rátérhetünk a megjelenítés megtervezésére. A szöveges üzemmódban a megjelenítés egyik leggyakrabban használt segédeszköze a kurzor, amely a képernyőn a kiírandó karakter helyét határozza meg. A kurzor helyzetét a 2 – 02(h) funkció meghívásával állíthatjuk be. Az AH regiszterbe a funkció számát, a DH, illetve DL regiszterekbe pedig annak a képernyő pozíciónak a sor-, illetve oszlopszámát kell beírni, ahová a kurzort el akarjuk helyezni. A BH regiszterbe annak a képernyőoldalnak a számát kell beírni, amelyiken a kurzort pozícionálni akarjuk. Ennek során figyelembe kell venni, hogy bár a karakteres üzemmódban mindegyik – létező – képernyőoldalhoz tartozik egy kurzor, csak egyetlen villogó kurzor van, amely mindig az aktív képernyőoldalon látható. Ezért a villogó kurzor pozíciója ezzel a funkcióval csak akkor állítható be, ha a BH regiszterben lévő érték az aktív képernyőoldalnak felel meg.

A 2 – 02(h) számú funkcióval szemben a 3 – 03(h) funkció a kurzor aktuális pozíciójának kiolvasására szolgál. Az egyes regiszterek szereposztása az előző funkcióéhoz hasonló: az AL regiszterbe a funkció számát, a BH regiszterbe pedig annak a képernyőoldalnak a számát kell írni, amelyen a leolvasandó kurzor található. Itt is ügyelni kell arra, hogy egy valóban létező oldal számát adjuk meg.

Monokróm képernyőkártya esetén az oldal száma csak 0 lehet, mert ezzel csak egyetlen képernyőoldal kezelhető (ennek a sorszáma pedig 0). A funkció meghívását követően, a 10(h) megszakítás visszatérésekor a kurzorra vonatkozóan 4 regiszterben kapunk vissza értékeket: a DH és DL regiszterek a megadott képernyőoldalon található kurzor pozícióját (sor, oszlop), míg a CH és CL regiszterek az aktuális képernyőoldaltól független, villogó kurzor alakjára vonatkozó információt, nevezetesen ennek a villogó kurzor-

nak a karaktermátrixon belüli kezdő és végsorát tartalmazzák. Ennek megértéséhez tudni kell, hogy egy karaktert a színes-grafikus képernyőkártya 8, a monokróm kártya pedig 14 egyedi, a képernyőt pásztázó elektronsugár által meghatározott soron helyez el.

Ezeket az értékeket, vagyis a villogó kurzor alakját az 1 – 01(h) funkció segítségével a programozó is meghatározhatja. Ehhez az AH regiszterbe 1-et, a CH és CL regiszterekbe pedig a villogó kurzornak a mátrixon belüli kezdő és végsorát kell beírni. Arra ügyelni kell, hogy az egyes értékek a kártyának megfelelő értékek (azaz monokróm kártya esetén 0 – 13, színes-grafikus kártya esetén 0 – 7) között legyenek, és a kezdősor ne legyen nagyobb a végsornál, mert különben a villogó kurzor láthatatlanná válik.

A képernyőoldalak az 5 – 05(h) funkció segítségével választhatók ki. Az aktíválandó (vagyis megjelenítendő) oldal kiválasztásához az AH regiszterbe a funkció számát, az AL regiszterbe pedig az illető oldal számát kell beírni, majd természetesen meg kell hívni a 16 – 10(h) ROM-BIOS megszakítást. Tekintettel arra, hogy a monokróm videokártya csak egy oldal kezelésére alkalmas, ebben az esetben az 5-ös funkció meghívásának nincs értelme. Színes-grafikus kártya esetén az üzemmódtól függően a kiválasztható képernyőoldal sorszáma 0 – 7 között (40 * 25 karakteres felbontás), illetve 0 – 3 között (80 * 25 karakteres felbontás) lehet.

A videoüzemmód megválasztása és a kurzor pozícionálása után következhet egy (vagy több) karakter megjelenítése. Erre a célra a BIOS különböző funkciókat bocsát rendelkezésünkre. Ezek a funkciók – amelyek a 9 – 09(h), 10 – 0A(h), 14 – 0E(h), illetve az AT esetén még a 19 – 13(h) sorszámokkal hívhatók meg, – természetesen bizonyos szolgáltatásokban különböznek egymástól. Ilyen különbség például az, hogy miként dolgozzák fel az ún. vezérlő karaktereket, vagy hogy a karakter beírását követően megváltozik-e a kurzor aktuális pozíciója. Csak emlékeztetőül megjegyezzük, hogy a vezérlő karaktereknek megfelelő kódok a szabványos ASCII-kódtáblázatban szereplő kódok, amelyek csak annyiban különböznek a többi „normál" kódtól, hogy ezeket a BIOS/DOS meghatározott funkciói nem a nekik megfelelő karakterekként jelenítik meg, hanem – a távközlésben kialakult hagyományoknak megfelelően – bizonyos, hozzájuk rendelt vezérlési feladatokat hajtanak végre. Ilyen vezérlő karakter az

- ún. „bell" (ASCII-kódja 7), amely a számítógép hangszóróján keresztül egy figyelmeztető, sípoló hangot ad ki,
- a törlő/visszaléptető (angolul backspace) karakter (ASCII-kódja 8), amely a kurzort egy pozícióval balra lépteti, és az ott talált karaktert törli,
- a soremelő (angolul line feed, LF) karakter (ASCII-kódja 10), amelynek hatására a kurzor a következő sorba kerül,
- a kocsi vissza (angolul carriage return, CR) karakter (ASCII-kódja 13), amely a kurzort a sor elejére állítja.

A karakterek megjelenítésére szolgáló valamennyi funkcióra vonatkozik, hogy ezek mind szöveges, mind grafikus üzemmódban használhatók, bár a megjelenítés módja a kétféle üzemmódban alapvetően eltér egymástól. Szöveges üzemmódban a karakterek megjelenítése viszonylag egyszerű. A 0 – 255 közötti, ún. kibővített ASCII-kódoknak megfelelő karaktereket – pontosabban fogalmazva az ezeket megrajzoló képpontokat, vagy más kifejezéssel bitmintát – ugyanis egy, a videokártyán elhelyezett ROM-elem, az ún. karaktergenerátor tartalmazza. A funkcióknak csak a megjelenítendő karakter

ASCII-kódját kell a video RAM megfelelő címeire beírni, a karaktert megrajzoló képpontok megjelenítését már a videokártyán lévő vezérlő áramkörök végzik. Más a helyzet a grafikus üzemmódban, hiszen itt az egyes képpontokat programból, külön-külön kell kezelni. Ehhez adnak segítséget a rendszerprogramok úgy, hogy a 0 – 127 közötti ASCII-kódoknak megfelelő karakterek bitmintáit a ROM-BIOS tartalmazza, a 128 – 255 közötti kódoknak megfelelő karakterek bitmintái pedig a DOS GRAFTABL parancsával a gép központi RAM-jába tölthetők.

Azt a táblát, amely ezen utóbbi bitmintákat tartalmazza, grafikus karaktergeneráló táblának nevezik, és a kezdőcímét a megszakítási vektortábla 0000:007C – 0000:007F címei tartalmazzák: az alsó két bájt a tábla ofszetcímét, a felső kettő a szegmenscímét adja meg. Mivel ez a cím adattáblára, és nem egy végrehajtható megszakítási rutinra mutat, az egyébként szokásos INT utasítással való meghívása egészen biztosan hibát okoz.

Az, hogy ez a tábla a RAM-ban van, egyúttal azt is jelenti, hogy a benne lévő bitmintákat programból megváltoztathatók, és ezáltal a képernyőn speciális, az IBM szabványos karakterkészletében nem szereplő karakterek is képernyőre írhatók. Mivel grafikus üzemmódban egy karakter egy 8 * 8 képpontból álló mátrixon ábrázolható, egy karakter megrajzolásához 8 bájtra van szükség. Könnyen kiszámítható, hogy a tábla a RAM-ban összesen 1024 bájtot, azaz 1 kbájtot foglal el. A táblában a karakterek a kódszámuk szerinti sorrendben helyezkednek el, így a 0 – 7. bájt a 128-as, a 8 – 15. bájt a 129-es stb. ASCII-kódú karakter bitmintáját tartalmazza. Az egyes karaktereket megrajzoló, 0-tól 7-ig számozott bájtok a 8 * 8 –as mátrix sorainak felelnek meg. Az egyes soroknak megfelelő bájtok értéke attól függ, hogy az illető sorban melyik bitet kell be-, ill. kikapcsolni.

Amint erről már volt szó, a képernyőre a 9 – 09(h), 10 – 0A(h), 14 – 0E(h), ill. PC/AT esetén még a 19 – 13(h) számú funkciókkal írhatók karakterek. Először vizsgáljuk meg a 9-es és a 10-es funkciót. A két funkció között csak annyi a különbség, hogy míg a 10-es funkció a karakter színét nem változtatja meg (vagyis ugyanolyan lesz a színe, mint az előzőleg megjelenített karakteré), addig a 9-es funkcióval a karakter színe (ill. attribútuma) is meghatározható. A karakter megjelenítésekor a kurzort egyik funkció sem lépteti a következő pozícióra, úgyhogy ismételt meghívásukkor a megadott karakter mindig ugyanazon a helyen jelenik meg.

Ha összefüggő szöveget akarunk a képernyőre írni, akkor a kurzort minden egyes karakter megjelenítése után a 2-es funkció meghívásával a következő pozícióra kell állítani.

Az előzőekben említett vezérlő karaktereket mind a 9-es, mind a 10-es funkció normál karakterekként értelmezi, és a képernyőn megjeleníti. A megszakítás meghívásához a funkció számát az AH, a megjelenítendő karakter ASCII-kódját pedig az AL regiszterbe kell írni. A BH regiszternek most is a kívánt képernyőoldal számát kell tartalmaznia. A CX regiszterben megadható egy ún. ismétlési tényező: a funkció a karaktert annyiszor jeleníti meg – ezúttal rendre egymást követő képernyőpozíciókon – , amekkora ez a tényező. Grafikus üzemmódban azonban ügyelni kell arra, hogy az így megjelenítendő karakterek elférjenek az aktuális képernyősorban. Ellenkező esetben az eredmény kiszámíthatatlan. Az ismétlési tényező használatával nem csak időt, hanem tárhelyet is megtakaríthatunk. Ha a karaktert csak egyszer kell a képernyőre írni, akkor a CX-ben 1-

nek kell lennie. (A CX tartalma a visszatéréskor nem dekrementálódik, úgyhogy ez ciklus szervezésére nem használható fel!)

Mint említettük, a 9-es funkcióval a megjelenítendő karakter színe, ill. attribútuma is beállítható. A kívánt értéket a BL regiszternek kell tartalmaznia.

Lássuk most a 14 – 0E(h) sorszámú funkció használatát. Ez a funkció úgy jeleníti meg a képernyőn a karaktert, ahogyan egy írógép vagy távíró a papírra ír (ezért TTY, azaz teletype funkciónak is nevezik): a karakter kiírása után a kurzor automatikusan a következő képernyőpozícióra lép. Ha a kurzor elérte a képernyő utolsó oszlopát, akkor a következő sor elejére áll, és itt folytatódik a kiírás. Ha „betelt" az oldal, akkor a képernyő tartalma egy sorral felfelé gördül, aminek következtében a legfelső sor eltűnik, és alul egy üres sor lép be. A kiírás most ennek a sornak az elején folytatódik.

A funkció – jellegéből adódóan – az előzőekben említett 4 vezérlő karaktert nem írja ki, hanem észlelésükkor végrehajtja a hozzájuk rendelt műveletet (pl. sort emel és az új sor elejére állítja a kurzort). A 10-es funkcióhoz hasonlóan ezzel a funkcióval – szöveges módban – a karakter színe nem állítható be, hanem „örökli" az illető képernyőpozíció színét. Grafikus üzemmódban a karakter színét meghatározó kódot a BL regiszterben kell átadni.

A megszakítás meghívásához a funkció számát az AH, a kiírandó karakter ASCII-kódját az AL, a képernyőoldal számát pedig a BH regiszterben kell megadni. (Ha az aktuális video üzemmód több képernyőoldalt tud kezelni – pl. színes-grafikus videokártya és szöveges mód –, akkor a funkcióval nemcsak az aktív képernyőoldalra lehet írni!)

A 14-es funkció bővített változatának tekinthető a 19 – 13(h) sorszámú funkció, amelyet csak az IBM PC/AT gépben lévő ROM-BIOS tud végrehajtani. (A funkció azokon az XT gépeken is meghívható, amelyekben EGA videokártya van, ugyanis ennek saját, e funkciót is támogató ROM-BIOS rendszerprogramja van.) A funkció segítségével egy adott képernyőoldal adott pozíciójától kezdődően egy pufferben megadott karakterlánc négy különböző módon jeleníthető meg. Az egyes megjelenítési módokat a 0 – 3. sorszámok azonosítják, amelyek egyrészt aszerint csoportosíthatók, hogy a karakterek attribútum bájtjait a karakterláncot tartalmazó puffer vagy egy processzorregiszter tárolja-e, másrészt viszont aszerint, hogy a karakterlánc kiírását követően aktualizálódik-e a kurzor pozíciója vagy nem. Vegyük sorra a különböző eseteket:

- A 0 – 1. sorszámú módban a puffernek csak a karakterláncot alkotó ASCII-kódokat kell tartalmaznia. A funkció a karakterekhez tartozó attribútum bájtot a BL regiszterből olvassa be, és „ékeli" az ASCII-kódok közé. A video RAM-ba természetesen már az ASCII-kódok és az attribútum bájtok felváltva íródnak be. A funkció meghívásakor a CX regiszterbe csak a ténylegesen megjelenítendő karakterek számát kell beírni. A két megjelenítési mód között az a különbség, hogy míg a 0. sorszámú módban a karakterlánc megjelenítését követően a kurzor pozíciója nem változik, az 1-es módban a kurzor a kiírást követő első képernyő pozícióra kerül.
- A 2 – 3. sorszámú módban a puffernek felváltva kell tartalmaznia a karakterláncot alkotó ASCII-kódokat és a hozzájuk tartozó attribútum bájtokat. Ebből következik, hogy az így létrejövő karakterlánc hossza a ténylegesen megjelenítendő karakterek számának kétszerese; ennek ellenére a funkció meghívásakor a CX regiszterbe csak a a ténylegesen megjelenítendő karakterek számát kell beírni. A kurzor pozí-

cionálását illetően e két megjelenítési mód között is ugyanaz a különbség, mint az előző kettő között: a 2-es módban a kurzor pozíciója a megjelenítést követően nem változik, a 3-asban viszont aktualizálódik.

A regiszterek szereposztása: az AH regiszternek a funkció számát, az AL-nek a megjelenítés módjának fent ismertetett sorszámát (0 – 3), a BH-nak a képernyőoldal számát (ami nem feltétlenül az aktív!), a BL-nek az attribútum bájt értékét (csak a 0 – 1. módban), a CX-nek a karakterek számát, a DH/DL regisztereknek a kiírás kezdő pozícióját (DH = sor, DL = oszlop), az ES:BP regiszterpárnak pedig a karakterláncot tartalmazó puffer szegmens- és ofszetcímét kell tartalmaznia.

Az előző néhány bekezdésben azokkal a funkciókkal foglalkoztunk, amelyek segítségével karakterek írhatók a képernyőre. E művelet fordítottját, azaz egy karakternek és a hozzá tartozó attribútum bájtnak egy adott képernyőpozícióról való elolvasását a 8-as funkcióval végezhetjük el. Ehhez a funkció számát az AH, a képernyőoldal számát pedig a BH regiszterbe kell beírni. A megszakításból való visszatéréskor az AH regiszter a megadott képernyőoldal aktuális kurzorpozícióján lévő karakter szín/attribútum bájtját, az AL regiszter pedig a karakter ASCII-kódját adja vissza.

Ez a funkció grafikus üzemmódban is meghívható: ekkor a funkció a karakter ASCII-kódját nem a video RAM-ból olvassa ki, hanem az aktuális kurzorpozíción lévő bitmintát a – részben a ROM-BIOS-ba beépített, részben a RAM-ba (a DOS GRAFTABL parancsával) betöltött grafikus karakterkészlet bitmintáival hasonlítja össze. Ha az összehasonlítás sikeres volt, a funkció az AL-ben a megtalált karakter ASCII-kódját, ellenkező esetben a 0 értéket adja vissza.

Következzen most két további, igen érdekes funkció leírása. E két, a 6-os és 7-es sorszámmal meghívható funkció lehetőséget ad arra, hogy segítségükkel a képernyőnek vagy a képernyő egy általunk megadott részének – egy ún. képernyőablaknak – a tartalmát töröljük, ill. fel- vagy lefelé görgessük (közismert angol szóhasználattal ezt a műveletet „scroll"-ozásnak nevezik.) Előljáróban megjegyezzük, hogy ezek a műveletek csak az aktuális képernyőoldalon hajthatók végre.

Nézzük meg először a felfelé görgető, 6-os funkció működését. A funkció számát természetesen most is az AH regiszterben kell átadni. Az AL regiszterben a felfelé gördítendő sorok számát kell megadni. Ha ez az érték 0, akkor a funkció a képernyőablakban nem gördíti a sorokat, hanem az ablakot szóköz karakterekkel (ASCII-kódja 32) tölti fel, azaz törli a képernyőablakot. A BH regiszterbe most nem a képernyőoldal számát, hanem a belépő üres sor(ok) színét, ill. attribútumát kell megadni (mint mondtuk, a művelet csak az aktív képernyőoldalon végezhető el). A képernyőablakot a sarokpontjaival kell kijelölni a következők szerint:

CH = ablak bal felső sarkának képernyő sora
CL = ablak bal felső sarkának képernyő oszlopa
DH = ablak jobb alsó sarkának képernyő sora
DL = ablak jobb alsó sarkának képernyő oszlopa

A 6-os funkció ellentettje a 7-es funkció, amellyel ugyanúgy görgethető, ill. törölhető egy megadott képernyőablak tartalma, mint a 6-ossal. A különbség mindössze annyi, hogy a 7-es funkcióval a sorok nem fel-, hanem lefelé gördíthetők.

Amennyiben fénytoll is csatlakozik a számítógéphez, akkor ennek a képernyő pozíciója a 4 – 04(h) funkció segítségével olvasható ki. A funkció meghívásához csak az AH regiszterben kell átadni a funkció számát. A megszakításból való visszatéréskor az AH regiszter tartalma jelzi, hogy a leolvasás sikeres volt vagy nem. Ha az AH-ban 0-t kapunk vissza, akkor ez azt jelenti, hogy a leolvasást meg kell ismételni. Ha a visszakapott érték 1, akkor a processzor regiszterei a következők szerint tartalmazzák a fénytoll pozícióját:

DH = fénytoll képernyő sora (szöveg mód)
DL = fénytoll képernyő oszlopa (szöveg mód)
CH = fénytoll képernyő sora (grafikus mód)
BX = fénytoll képernyő oszlopa (grafikus mód)

A visszakapott koordináták értelmezésénél természetesen figyelembe kell venni a mindenkori video üzemmód sor/oszlop felbontását.

Végezetül vizsgáljuk meg a grafikus üzemmódban rendelkezésre álló 11 – 0B(h), 12 – 0C(h) és 13 – 0D(h) számú funkciók használatát. Ezek a funkciók természetesen csak akkor használhatók, ha a gép színes-grafikus videokártyát (is) tartalmaz, és ez az aktív kártya.

A 11-es funkcióval a 320 * 200 képpont felbontású grafikus üzemmódban a keret- és a háttér színe, illetve a színpaletta választható meg. Ettől függően beszélünk 0. vagy 1. alfunkcióról: ha a BH regiszterben a 0 értéket adjuk át, akkor a funkció a BL regiszterben lévő színkódnak megfelelően a háttér- és a keret színét állítja be (0. alfunkció). 16 szín közül lehet választani úgy, hogy a BL regiszter tartalma 0 – 15 közötti érték lehet. Ezzel az alfunkcióval szöveges módban a keret színét határozhatjuk meg (ugyanis, mint már láttuk, ebben az üzemmódban minden karakterpozíció háttérszínét a hozzá tartozó attribútum bájt határozza meg).

Ha a BH regiszterben viszont az 1 értéket adjuk át, akkor a 11-es funkció a BL regiszter tartalmától függően a rendelkezésre álló két színpaletta egyikét választja ki (1. alfunkció).

A színpaletták színei:

0. számú színpaletta: zöld, piros, sárga (BL = 0)
1. számú színpaletta: türkiz, bíbor, fehér (BL = 1)

Miután beállítottuk a grafikus üzemmódot (0. funkció) és kiválasztottuk a kívánt háttérszínt és a színpalettát, már csak a grafikus képpont (ismert nevén pixel) megjelenítése van hátra. Erre szolgál a 12 – 0B(h) funkció. A megjelenítendő képpont koordinátáit a CX, ill. a DX regiszterben kell megadni. A CX regiszter a képpont X irányú koordinátáját (oszlopszám), a DX pedig az Y irányú koordinátáját (sorszám) tartalmazza. A lehetséges értékek természetesen a mindenkori video üzemmódtól függnek. A képpont színének kódját az AL regiszterbe kell beírni. Ez az érték monokróm Hercules-grafikus videokártyánál, ill. a színes-grafikus kártya 640*200 képpontos üzemmódjában csak 0 (képpont kioltva) vagy 1 (képpont bekapcsolva) lehet. Ezzel szemben a színes-grafikus kártya 320*200 képpontos felbontása esetén 4 szín közül választhatunk, vagyis az AL regiszterbe a 0 – 3 érték írható be. Amint erről a II. kötet Képernyőkártyák című szaka-

szában már volt szó, ez a 4 szín nem választható meg egymástól teljesen függetlenül. Ha a regiszterbe a 0 értéket írjuk, akkor a képpont színe az a háttérszín lesz, amelyet az előbb említett 11-es funkcióval beállítottunk (vagyis a képpont nem jelenik meg, hanem „beleolvad" a háttérbe). Az 1 – 3 értékekkel három szín valamelyike állítható be attól függően, hogy a rendelkezésre álló két lehetséges színpaletta közül előzőleg melyiket választottuk ki.

Ha a 0. számú színpaletta aktív, akkor a színértékek jelentése:

1 = zöld, 2 = piros, 3 = sárga.

Ha az 1. számú színpaletta aktív, akkor a színértékek jelentése:

1 = türkiz, 2 = bíbor, 3 = fehér.

A most ismertetett funkció ellentettje a 13 – 0D(h) funkció, amellyel egy grafikus képpont (pixel) színértéke kérdezhető le. A funkció meghívásához a AH regiszterben a funkció számát, a CX és DX regiszterekben pedig a leolvasandó képpont X, illetve Y irányú koordinátáit kell átadni. A megszakítás visszatérésekor az AL regiszter az illető képpont színértékét adja vissza. A színérték jelentése ugyanaz, mint a 12 – 0C(h) funkciónál.

1.3.3. Példaprogramok

A képernyőkezelő funkciókból mutatunk be egy csokorra valót programpéldáinkban. A bemutatott funkciók köre korántsem teljes, nem is ez volt a célunk a példák megírásánál, hanem közelebb hozni a BIOS-on keresztül történő képernyőkezelést. Kompatibilitási okokból mindegyik program egyetlen képernyőoldalra dolgozik és egyik sem tartalmaz grafikus képernyőkezelő funkciókat. Meggyőződésünk, hogy példáink alapján, amenyiben munkájához erre szüksége lenne, bárki csekély energiabefektetéssel ki tudja egészíteni a programokat a hiányzó funkciókkal.

Turbo Pascal és C nyelven a BIOS képernyőkezelő szolgáltatásainak használata sebességi követelmények alapján is felmerülhet, ezeken a nyelveken ezzel a módszerrel a nyelv által biztosított képernyőkezelésnél gyorsabb működés érhető el. Korántsem igaz ez a BASIC nyelvre, amelynél az amúgy is lassú PRINT parancs lecserélése BIOS funkciókra tovább csökkenti a program sebességét. Mindhárom nyelvben egyértelműen előnyös viszont a képernyőkezelő BIOS funkciók használata néhány (egyik nyelvnél kevesebb, másiknál több) a nyelv által nem támogatott szolgáltatás létrehozására. A felhasználói programokban például elegáns megjelenítés érhető el ablaktechnika használatával, amihez viszont (többek között) feltétlenül szükséges az ablakok tartalmának görgethetősége két irányban, ill. törlése. Az erre szolgáló nyelvi eszközök alapkiépítettségben egyik fent említett nyelven sem állnak rendelkezésre.

A BIOS funkciók használatának természetesen hátrányai is vannak. Az egyik lehetséges hátrány (a körülmények dönthetik el, hogy valóban az-e) a csökkenő hordozhatóság. A másik hátrány viszont egyértelmű, ez a numerikus értékek megjelenítével kapcsolatos. Míg a magasszintű nyelvek kelléktárából nem hiányzik a numerikus értékek kezelésével, megjelenítésével kapcsolatos általában igen gazdag eszközrendszer (helyértékek kezelése, pontosság stb.), addig a BIOS funkciók kizárólag ASCII karakterekkel dolgoznak.

Ennek az a következménye, hogy numerikus értékek használata esetén a felhasználónak kell a karakterlánc <–> numerikus átalakítását elvégeznie, ami egyrészt körülményesebbé teszi, másrészt lassítja a programot.

A programpéldák demonstrációs része igen egyszerű, a grafikus keretkarakterekkel feltöltött képernyőn 6 ablakot nyitottunk, amelyek mindegyikében folyamatosan növekvő (0 és 9 közötti) számértékek jelennek meg, de jobbról balra a növelés sebessége mindig a felére csökken. Maguk az ablakok a növekedéssel szinkronban fel-, ill. legördülnek. A képernyőkezelő megszakítás felhasznált funkcióinak (kurzor pozícionálása, aktuális képernyőoldal meghatározása, görgetési műveletek, képernyő törlése, karakter kiírása stb.) használatát a programokban elhelyezett megjegyzések világítják meg.

A BASIC nyelvű programváltozattal kapcsolatban ismét hangsúlyozzuk, hogy a megszakítást hívó gépi kódú eljárást konstans értékekkel nem lehet meghívni, a paramétereket EGÉSZ típusú változókban kell átadni és ezekben a változókban kapjuk vissza az eredményt is.

```
1000 '
1005 '
1010 '                    K E P E R N Y O . B A S
1020 '
1030 '  Funkciója : a videomegszakítások szemléltetése néhány egyéb
1040 '              célra is jól használható rutin segítségével.
1050 '
1060 '
1070 '
1080 DEFINT Q                                   'Q "dummy" egész értékű.
1090 KEY OFF : CLS
1100 PRINT : PRINT
1110 PRINT "FIGYELMEZTETÉS :" : PRINT
1120 PRINT "A 60000-es címtől kezdődően betöltött megszakításhívó ru-"
1130 PRINT "tin felülírását elkerülendő a GWBASIC-et GWBASIC /M:60000"
1140 PRINT "paranccsal kell indítania. Ha nem így hívta a GWBASIC-et,"
1150 PRINT "akkor üsse le az <s> billentyűt, egyébként egy  tetszőle-"
1160 PRINT "ges másik billentyűt !"
1170 '
1180 Z$ = ""
1190 WHILE Z$ = ""                              'Helytelen indítás
1200    Z$ = INKEY$                             'esetén a program
1210 WEND                                       'leállítása.
1220 IF Z$ = "s" OR Z$ = "S" THEN CLS:END
1230 '
1240 GOSUB 9000                                 'Megszakításhívó rutin
1250                                            'betöltése.
1260 CLS
1270 '
2000 '
2010 '          P R O G R A M T Ö R Z S
2020 '
2030 '
2040 OLDAL%=0
2050 SZIN%=7
2060 FOR SOR%=0 TO 24                           'Ciklus oszlopokra és
2070   FOR OSZLOP%=0 TO 79                      'sorokra.
2080     KAR$=CHR$(179+(SOR%*80+OSZLOP%) MOD 45) 'Grafikus karakter.
2090     GOSUB 3000                             'Kurzorpozicionálás.
2100     GOSUB 6000                             'Karakter kiírás.
2110   NEXT
2120 NEXT
2130 '
2140 GSOR% = 3                                  '2 x 6 db kép-
2150 BFO%=10: BFS%= 1: JAO%=69: JAS%= 3: GOSUB 4000   'ernyőablak
2160                                            'megnyitása
2170 BFO%= 4: BFS%= 5: JAO%=10: JAS%= 7: GOSUB 4000   'görgetéssel.
2180 BFO%=17: BFS%= 5: JAO%=23: JAS%= 7: GOSUB 4000
2190 BFO%=30: BFS%= 5: JAO%=36: JAS%= 7: GOSUB 4000
2200 BFO%=43: BFS%= 5: JAO%=49: JAS%= 7: GOSUB 4000
2210 BFO%=56: BFS%= 5: JAO%=62: JAS%= 7: GOSUB 4000
2220 BFO%=69: BFS%= 5: JAO%=75: JAS%= 7: GOSUB 4000
2230 '
2240 GSOR% = 14
2250 BFO%= 4: BFS%= 9: JAO%=10: JAS%=22: GOSUB 4000
2260 BFO%=17: BFS%= 9: JAO%=23: JAS%=22: GOSUB 4000
2270 BFO%=30: BFS%= 9: JAO%=36: JAS%=22: GOSUB 4000
2280 BFO%=43: BFS%= 9: JAO%=49: JAS%=22: GOSUB 4000
2290 BFO%=56: BFS%= 9: JAO%=62: JAS%=22: GOSUB 4000
2300 BFO%=69: BFS%= 9: JAO%=75: JAS%=22: GOSUB 4000
2310 '
2320 SZIN%=&H70
```

```
2330 OSZLOPX=13 : SORX=2
2340 T$="Görgetés a leggyorsabb ablakban  300x, jelenleg : "
2350 GOSUB 7000
2360 OSZLOPX= 5 : SORX=6 : T$="1/32" : GOSUB 7000
2370 OSZLOPX=18 : SORX=6 : T$="1/16" : GOSUB 7000
2380 OSZLOPX=31 : SORX=6 : T$=" 1/8" : GOSUB 7000
2390 OSZLOPX=44 : SORX=6 : T$=" 1/4" : GOSUB 7000
2400 OSZLOPX=57 : SORX=6 : T$=" 1/2" : GOSUB 7000
2410 OSZLOPX=70 : SORX=6 : T$="  1 " : GOSUB 7000
2420 '
2430 SZINX=7                                        'Ciklikus görgetés
2440 FOR IX = 0 TO 299                              'és kiíratás, az
2450     OSZLOPX=64 : SORX=2 : T$=STR$(IX)          'egyes ablakokban
2460     GOSUB 7000                                 'eltérõ sebességgel.
2470 '
2480     GSORX=1 : BFOX=69 : BFSX=10 : JAOX=75 : JASX= 21
2490     GOSUB 4000
2500     OSZLOPX=72 : SORX= 21
2510     GOSUB 3000
2520     KAR$=CHR$(48+IX MOD 10)
2530     GOSUB 6000
2540     IF (IX MOD 2 <> 0) THEN GOTO 2890
2550          GSORX=1 : BFOX=56 : BFSX=10 : JAOX=62 : JASX= 21
2560          GOSUB 5000
2570          OSZLOPX=59 : SORX= 10
2580          GOSUB 3000
2590          KAR$=CHR$(48+(IX \ 2) MOD 10)
2600          GOSUB 6000
2610     IF (IX MOD 4 <> 0) THEN GOTO 2890
2620          GSORX=1 : BFOX=43 : BFSX=10 : JAOX=49 : JASX= 21
2630          GOSUB 4000
2640          OSZLOPX=46 : SORX= 21
2650          GOSUB 3000
2660          KAR$=CHR$(48+(IX \ 4) MOD 10)
2670          GOSUB 6000
2680     IF (IX MOD 8 <> 0) THEN GOTO 2890
2690          GSORX=1 : BFOX=30 : BFSX=10 : JAOX=36 : JASX= 21
2700          GOSUB 5000
2710          OSZLOPX=33 : SORX= 10
2720          GOSUB 3000
2730          KAR$=CHR$(48+(IX \ 8) MOD 10)
2740          GOSUB 6000
2750     IF (IX MOD 16 <> 0) THEN GOTO 2890
2760          GSORX=1 : BFOX=17 : BFSX=10 : JAOX=23 : JASX= 21
2770          GOSUB 4000
2780          OSZLOPX=20 : SORX= 21
2790          GOSUB 3000
2800          KAR$=CHR$(48+(IX 6) MOD 10)
2810          GOSUB 6000
2820     IF (IX MOD 32 <> 0) THEN GOTO 2890
2830          GSORX=1 : BFOX= 4 : BFSX=10 : JAOX=10 : JASX= 21
2840          GOSUB 5000
2850          OSZLOPX= 7 : SORX= 10
2860          GOSUB 3000
2870          KAR$=CHR$(48+(IX 2) MOD 10)
2880          GOSUB 6000
2890 NEXT
2900 CLS
2910 KEY ON
2920 END
2930 '
```

```
3000 '
3010 '  A KURZOR POZíCIONÁLÁSA A KIVÁLASZTOTT
3020 '          KÉPERNYŐOLDALON
3030 '     BE : OLDAL% képernyőoldal
3040 '          OSZLOP% képernyőoszlop
3050 '          SOR% képernyősor
3060 '
3070 '
3080 FUN%=&H2                                  ' Kurzorbeállítás funkció.
3090 MSZ%=&H10                                 ' 10(h) megszakítás.
3100 '
3110 CALL RCIM(MSZ%,FUN%,0,OLDAL%,0,0,0,SOR%,OSZLOP%,0,0,0,0)
3120 RETURN
3130 '
4000 '
4010 '  KÉPERNYŐ FELFELE GÖRGETÉSE
4020 '     BE : GSOR% a Görgetendő SOROK száma
4030 '          BFO% a Bal Felső Oszlop
4040 '          BFS% a Bal Felső Sor
4050 '          JAO% a Jobb Alsó Oszlop
4060 '          JAS% a Jobb Alsó Sor
4070 '          SZIN% a törölt terület SZÍNe
4080 '     KI : -
4090 '  Ha GSOR% 0, akkor a terület törlődik.
4100 '
4110 '
4120 FUN%=&H6                                  ' Felfelé görgetés funkció.
4130 MSZ%=&H10                                 ' 10(h) megszakítás.
4140 '
4150 CALL RCIM(MSZ%,FUN%,GSOR%,SZIN%,0,BFS%,BFO%,JAS%,JAO%,0,0,0,0)
4160 RETURN
4170 '
5000 '
5010 '  KÉPERNYŐ LEFELE GÖRGETÉSE
5020 '     BE : GSOR% a Görgetendő SOROK száma
5030 '          BFO% a Bal Felső Oszlop
5040 '          BFS% a Bal Felső Sor
5050 '          JAO% a Jobb Alsó Oszlop
5060 '          JAS% a Jobb Alsó Sor
5070 '          SZIN% a törölt terület SZÍNe
5080 '     KI : -
5090 '  Ha GSOR% 0, akkor a terület törlődik.
5100 '
5110 '
5120 MSZ%=&H10                                 ' 10(h) megszakítás.
5130 FUN%=&H7                                  ' Lefelé görgetés funkció.
5140 '
5150 CALL RCIM(MSZ%,FUN%,GSOR%,SZIN%,0,BFS%,BFO%,JAS%,JAO%,0,0,0,0)
5160 RETURN
5170 '
6000 '
6010 '  EGY KARAKTER KIÍRÁSA ADOTT ATTRIBÚTUMMAL
6020 '  TETSZ. KÉPERNYŐOLDALRA
6030 '     BE : KAR$ a karakter
6040 '          OLDAL% a megjelenítés oldala
6050 '          SZIN% a megjelenítés színe
6060 '     KI : -
6070 '
6080 '
6090 FUN%=&H9                                  ' Karakter megjelenítés funkció.
6100 MSZ%=&H10                                 ' 10(h) megszakítás.
6110 N1%=0                                     ' Egy megjelenítés szükséges.
```

```
6120 N2% =1
6130 KOD%=ASC(KAR$)
6140 CALL RCIM(MSZ%,FUN%,KOD%,OLDAL%,SZIN%,N1%,N2%,Q,Q,Q,Q,Q,Q)
6150 RETURN
6160 '
7000 '
7010 '  KARAKTERLÁNC KIÍRÁSA ADOTT KÉPERNYŐPOZÍ-
7020 '  CIÓTÓL KEZDVE ADOTT OLDALRA
7030 '     BE : T$ a karakterlánc
7040 '          OLDAL% a megjelenítés oldala
7050 '
7060 '
7070 GOSUB 3000                          ' Kurzor pozícionálás.
7080 FOR J%=1 TO LEN(T$)                 ' Kar. lánc feldolg. ciklusban.
7090     KAR$=" "
7100     GOSUB 6000
7110     KOD%=ASC(MID$(T$,J%,1))         ' Az akt. karakter dekódolása.
7120     FUN%=&HE                        ' Karakter írás funkció.
7130     MSZ%=&H10                       ' 10(h) megszakítás.
7140 '
7150     CALL RCIM(MSZ%,FUN%,KOD%,OLDAL%,Q,Q,Q,Q,Q,Q,Q,Q,Q)
7160 NEXT
7170 RETURN
7180 '
9000 '
9010 '  A MEGSZAKÍTÁSHÍVÓ RUTIN INICIALIZÁLÁSA
9020 '     BE: -
9030 '     KI: RCIM a Rutin CÍMe
9040 '
9050 '
9060 RCIM = 60000!                       'A rutin kezdőcíme a BASIC szegmensben.
9070 DEF SEG                             'A BASIC szegmens beállítása.
9080 RESTORE 9140
9090 FOR I% = 0 TO 160                                  'A rutin betöltése a
9100     READ X% : POKE RCIM + I%, X%                   'tárba.
9110 NEXT
9120 RETURN
9130 '
9140 DATA    85, 139, 236,  30,   6, 139, 118,  30, 139,   4, 232, 140
9150 DATA     0, 139, 118,  12, 139,  60, 139, 118,   8, 139,   4,  61
9160 DATA   255, 255, 117,   2, 140, 216, 142, 192, 139, 118,  28, 138
9170 DATA    36, 139, 118,  26, 138,   4, 139, 118,  24, 138,  60, 139
9180 DATA   118,  22, 138,  28, 139, 118,  20, 138,  44, 139, 118,  18
9190 DATA   138,  12, 139, 118,  16, 138,  52, 139, 118,  14, 138,  20
9200 DATA   139, 118,  10, 139,  52,  85, 205,   0,  93,  86, 156, 139
9210 DATA   118,  12, 137,  60, 139, 118,  28, 136,  36, 139, 118,  26
9220 DATA   136,   4, 139, 118,  24, 136,  60, 139, 118,  22, 136,  28
9230 DATA   139, 118,  20, 136,  44, 139, 118,  18, 136,  12, 139, 118
9240 DATA    16, 136,  52, 139, 118,  14, 136,  20, 139, 118,   8, 140
9250 DATA   192, 137,   4,  88, 139, 118,   6, 137,   4,  88, 139, 118
9260 DATA    10, 137,   4,   7,  31,  93, 202,  26,   0,  91,  46, 136
9270 DATA    71,  66, 233, 108, 255
```

```
/*                                                                    */
/*  ┌──────────────────────────────────────────────────────────────┐  */
/*  │                                                              │  */
/*  │                        K E P E R N Y O . C                   │  */
/*  │                                                              │  */
/*  │  Funkciója : a videomegszakítások szemléltetése néhány egyéb │  */
/*  │              célra is jól használható rutin segítségével.    │  */
/*  │                                                              │  */
/*  └──────────────────────────────────────────────────────────────┘  */

#include <dos.h>
#include <io.h>

#define  NM     0x07                    /* Normál megjelenítés.      */
#define  IM     0x70                    /* Inverz megjelenítés.      */
#define  EM     0x0F                    /* Megnövelt fényerő.        */

#define  FALSE    0
#define  TRUE     1

#define byte unsigned char

/*  ┌──────────────────────────────────────────────────────────────┐  */
/*  │ KURPOZ : A KURZOR POZÍCIONÁLÁSA A KIVÁ-                      │  */
/*  │          LASZTOTT KÉPERNYŐOLDALON                            │  */
/*  └──────────────────────────────────────────────────────────────┘  */

void KURPOZ(oldal,oszlop, sor)
int oldal;
int oszlop;
int sor;

{
union REGS regiszter;                  /* Regiszter union deklaráció. */

regiszter.h.ah = 2;                    /* Kurzor beállítása az       */
regiszter.h.bh = oldal;                /* aktuális oldalon az        */
regiszter.h.dh = sor;                  /* átadott sor- és osz-       */
regiszter.h.dl = oszlop;               /* loppozícióra.              */
int86(0x10, &regiszter, &regiszter);   /* 10(h) megszakítás.         */
}

/*  ┌──────────────────────────────────────────────────────────────┐  */
/*  │ AKTOLDAL : AZ AKTUÁLIS KÉPERNYŐOLDAL MEG-                    │  */
/*  │            HATÁROZÁSA                                        │  */
/*  └──────────────────────────────────────────────────────────────┘  */

byte AKTOLDAL()

{
union REGS regiszter;                  /* Regiszter union deklaráció. */

regiszter.h.ah = 0x0F;                 /* Video olvasás funkció.     */
int86(0x10, &regiszter, &regiszter);   /* 10(h) megszakítás.         */
return(regiszter.h.bh);                /* Aktuális képernyőoldal.    */
}

/*  ┌──────────────────────────────────────────────────────────────┐  */
/*  │ GÖRGETÉS_FEL : MEGADOTT KÉPERNYŐTARTOMÁNY                    │  */
/*  │                FELFELÉ GÖRGETÉSE, ill. TÖRLÉSE               │  */
/*  └──────────────────────────────────────────────────────────────┘  */

void GORGETES_FEL(gsor, szin, bfo, bfs, jao, jas)
```

```
int gsor;              /* (G)örgetendõ (sor)ok száma.                    */
int szin;              /* A törölt terület (szín)e.                      */
int bfs ;              /* Képernyõtartomány (b)al  (f)elsõ (s)ora.       */
int bfo ;              /*                   (b)al  (f)elsõ (o)szlopa.    */
int jas ;              /*                   (b)al  (a)lsó  (s)ora.       */
int jao ;              /*                   (j)obb (a)lsó  (o)szlopa.    */

{
union REGS regiszter;                  /* Regiszter union deklaráció.     */

regiszter.h.ah = 0x06;                 /* Görgetés funkció.               */
regiszter.h.al = gsor;                 /* Görgetendõ sorok száma.         */
regiszter.h.bh = szin;                 /* Az üres sorok színe.            */
regiszter.h.ch = bfs ;                 /* Ablak-                          */
regiszter.h.cl = bfo ;                 /*       koor-                     */
regiszter.h.dh = jas ;                 /*            dináták.             */
regiszter.h.dl = jao ;                 /*                                 */

int86(0x10, &regiszter, &regiszter);           /* 10(h) megszakítás.      */
}

/*
/*  ┌──────────────────────────────────────────┐                          */
/*  │ GORGETES_LE : MEGADOTT KÉPERNYÕTARTOMÁNY │                          */
/*  │               LEFELE GÖRGETÉSE, ill.TÖRLÉSE                         */
/*  └──────────────────────────────────────────┘                          */

void GORGETES_LE(gsor, szin, bfo, bfs, jao, jas)

int gsor;              /* (G)örgetendõ (sor)ok száma.                    */
int szin;              /* A törölt terület (szín)e.                      */
int bfs ;              /* Képernyõtartomány (b)al  (f)elsõ (s)ora.       */
int bfo ;              /*                   (b)al  (f)elsõ (o)szlopa.    */
int jas ;              /*                   (b)al  (a)lsó  (s)ora.       */
int jao ;              /*                   (j)obb (a)lsó  (o)szlopa.    */

{
union REGS regiszter;                  /* Regiszter union deklaráció.     */

regiszter.h.ah = 0x07;                 /* Görgetés funkció.               */
regiszter.h.al = gsor;                 /* Görgetendõ sorok száma.         */
regiszter.h.bh = szin;                 /* Az üres sorok színe.            */
regiszter.h.ch = bfs ;                 /* Ablak-                          */
regiszter.h.cl = bfo ;                 /*       koor-                     */
regiszter.h.dh = jas ;                 /*            dináták.             */
regiszter.h.dl = jao ;                 /*                                 */

int86(0x10, &regiszter, &regiszter);           /* 10(h) megszakítás.      */
}

/*
/*  ┌──────────────────────────────────────────┐                          */
/*  │ KARIR : EGY KARAKTER KIÍRÁSA ADOTT ATTRI-│                          */
/*  │         BÚTUMMAL TETSZ. KÉPERNYÕOLDALRA  │                          */
/*  └──────────────────────────────────────────┘                          */

void KARIR(oldal, kar, szin)
int  oldal;                            /* A kiírandó karakter és a hozzá  */
char kar;                              /* tartozó attribútum, ill. szín,  */
int  szin;                             /* valamint a megjelenítés oldala. */

{
union REGS regiszter;                  /* Regiszter union deklaráció.     */
```

```
regiszter.h.ah = 9;                    /* Karaktermegjelenítés funkció.    */
regiszter.h.al = kar;                  /* A megjelenítendő karakter.       */
regiszter.h.bh = oldal;                /* Az aktuális képernyőoldal.       */
regiszter.h.bl = szin;                 /* A megjelenítés színe.            */
regiszter.x.cx = 1;                    /* Egyszer történik megjelenítés.   */

int86(0x10, &regiszter, &regiszter);           /* 10(h) megszakítás.       */
}

/*  ┌──────────────────────────────────────────────────┐                    */
/*  │ IR : KARAKTERLÁNC KIÍRÁSA ADOTT KÉPER-           │                    */
/*  │      NYŐPOZÍCIÓTÓL KEZDVE ADOTT OLDALRA          │                    */
/*  │ A kar. lánc végét '\0' karakter jelzi.           │                    */
/*  └──────────────────────────────────────────────────┘                    */

void IR(oldal, oszlop, sor, szin, txt)
int  oldal;
int  oszlop;
int  sor;                              /* A megjelenítés koordinátái,      */
int  szin;                             /* a kiírandó kar.lánc és a hozzá   */
char *txt;                             /* tartozó attribútum, ill. szín.   */

{
union REGS regiszter;                        /* Regiszter union deklaráció. */

KURPOZ(oldal, oszlop, sor);                  /* Kurzor pozícionálása.       */
regiszter.h.ah = 14;                         /* Karakter írás funkció.      */
regiszter.h.bh = oldal;                      /* Akt. képernyőoldal beáll.   */
while (*txt)                                 /* Kar.lánc írása '\0'-ig.     */
   {
   KARIR(oldal, ' ', szin);
   regiszter.h.al = *txt++;                        /* A kiírandó karakter.  */
   int86(0x10, &regiszter, &regiszter);            /* 10(h) megszakítás.    */
   }
}

/*  ┌──────────────────────────────────────────────────┐                    */
/*  │ CLRSCR : KÉPERNYŐ TÖRLÉSE                        │                    */
/*  └──────────────────────────────────────────────────┘                    */

void CLRSCR()

{
GORGETES_FEL(0, KM, 0, 0, 79, 24);           /* Törlés görgetéssel.         */
KURPOZ(AKTOLDAL(), 0);                       /* Kurzor "home" pozícióba.    */
}

/*  ┌──────────────────────────────────────────────────┐                    */
/*  │                 F Ő P R O G R A M                │                    */
/*  └──────────────────────────────────────────────────┘                    */

void main()

{
int i, j, k, l;
char ngorg[5];

CLRSCR();
for (i = 0; i < 25; i++)                           /* Képernyő feltöltése   */
   for (j = 0; j < 80; j++)                        /* graf. karakterekkel.  */
   {
   KURPOZ(0, j, i);
   KARIR(0, 179+(i*80+j)%45, KM);
   }

GORGETES_LE( 3, KM, 10, 1, 69,  3);
```

```
GORGETES_LE( 3, NM,  4, 5, 10,  7);          /* 2 x 6 db képernyõ- */
GORGETES_LE( 3, NM, 17, 5, 23,  7);          /* ablak megnyitása   */
GORGETES_LE( 3, NM, 30, 5, 36,  7);          /* görgetéssel.       */
GORGETES_LE( 3, NM, 43, 5, 49,  7);
GORGETES_LE( 3, NM, 56, 5, 62,  7);
GORGETES_LE( 3, NM, 69, 5, 75,  7);
GORGETES_LE(14, NM,  4, 9, 10, 22);
GORGETES_LE(14, NM, 17, 9, 23, 22);
GORGETES_LE(14, NM, 30, 9, 36, 22);
GORGETES_LE(14, NM, 43, 9, 49, 22);
GORGETES_LE(14, NM, 56, 9, 62, 22);
GORGETES_LE(14, NM, 69, 9, 75, 22);

IR(0,13, 2, IM, "Görgetés a leggyorsabb ablakban 1000x, jelenleg : ");

IR(0, 5, 6, IM, "1/32");
IR(0,18, 6, IM, "1/16");
IR(0,31, 6, IM, " 1/8");
IR(0,44, 6, IM, " 1/4");
IR(0,57, 6, IM, " 1/2");
IR(0,70, 6, IM, "  1 ");

for (i = 0; i < 1000 ; i++)                  /* Ciklikus görgetés  */
   {                                         /* és kiíratás, az    */
   sprintf(ngorg, "%4d", i);                 /* egyes ablakokban   */
   IR(0, 63, 2, IM, ngorg);                  /* eltérõ sebességgel.*/

   GORGETES_FEL(1, NM, 69, 10, 75, 21);
   KURPOZ(0,72,21);
   KARIR (0,  48 + i %  10, EM);
   if (i % 2 == 0)
      {
      GORGETES_LE (1, NM,56, 10, 62, 21);
      KURPOZ(0,59,10);
      KARIR (0,  48 + (i / 2) % 10, EM);
      }
   if (i % 4 == 0)
      {
      GORGETES_FEL(1, NM,43, 10, 49, 21);
      KURPOZ(0,46,21);
      KARIR (0,  48 + (i / 4) % 10, EM);
      }
   if (i % 8 == 0)
      {
      GORGETES_LE (1, NM,30, 10, 36, 21);
      KURPOZ(0,33,10);
      KARIR (0,  48 + (i / 8) % 10, EM);
      }
   if (i % 16 == 0)
      {
      GORGETES_FEL (1, NM,17, 10, 23, 21);
      KURPOZ(0,20,21);
      KARIR (0,  48 + (i / 16) % 10, EM);
      }
   if (i % 32 == 0)
      {
      GORGETES_LE (1, NM, 4, 10, 10, 21);
      KURPOZ(0,7,10);
      KARIR (0,  48 + (i / 32) % 10, EM);
      }
   for (l = 0; l < 4000 ; l++)
      ;
   }
CLRSCR();
}
```

```
{ ┌──────────────────────────────────────────────────────────┐ }
{ │                                                          │ }
{ │                     K E P E R N Y O . P A S              │ }
{ │                                                          │ }
{ │  Funkciója : a videomegszakítások szemléltetése néhány egyéb │ }
{ │              célra is jól használható rutin segítségével. │ }
{ │                                                          │ }
{ └──────────────────────────────────────────────────────────┘ }

program VIDEO;

const NM        = $07;                          { Normál megjelenítés. }
      IM        = $70;                          { Inverz megjelenítés. }
      EM        = $01;                          { Megnövelt fényerő.   }

type dosreg     = record
                  ax, bx, cx, dx, bp, di, si, ds, es, flags : integer;
                  end;
     Karlanc    = string[80];

{ ┌──────────────────────────────────────────────┐ }
{ │ KURPOZ : A KURZOR POZíCIONÁLÁSA A KIVÁ-      │ }
{ │          LASZTOTT KÉPERNYŐOLDALON            │ }
{ └──────────────────────────────────────────────┘ }

procedure KURPOZ(oldal,oszlop,sor : integer);

var regiszter : dosreg;                 { Regiszter rekord deklaráció. }

begin
   regiszter.ax := 2 shl 8;                         { Kurzor beállítása az }
   regiszter.bx := oldal shl 8;                     { aktuális oldalon az  }
   regiszter.dx := sor shl 8 + oszlop;              { átadott sor- és osz- }
                                                    { loppozícióra.        }
   intr($10, regiszter);                            { 10(h) megszakítás.   }
end;

{ ┌──────────────────────────────────────────────┐ }
{ │ GORGETES_FEL : MEGADOTT KÉPERNYŐTARTOMÁNY    │ }
{ │          FELFELE GÖRGETÉSE, 111. TÖRLÉSE     │ }
{ └──────────────────────────────────────────────┘ }

procedure GORGETES_FEL(gsor,            { (G)örgetendő (sor)ok száma.    }
                       szin,            { A törölt terület (szín)e.      }
                                        { Képernyőtartomány              }
                       bfs,             {      (b)al  (f)első (s)ora.    }
                       bfo,             {      (b)al  (f)első (o)szlopa. }
                       jas,             {      (b)al  (a)lsó  (s)ora.    }
                       jao:integer);    {      (j)obb (a)lsó  (o)szlopa. }

var regiszter : dosreg;                 { Regiszter rekord deklaráció.   }

begin
   regiszter.ax := 6 shl 8 + gsor;      { Görgetés funkció+ sorok száma. }
   regiszter.bx := szin shl 8;          { Az üres sorok színe.           }
   regiszter.cx := bfs shl 8 + bfo;     { AblakKoor      -               }
   regiszter.dx := jas shl 8 + jao;     {                        dináták. }
   intr($10, regiszter);                { 10(h) megszakítás.             }
end;

{ ┌──────────────────────────────────────────────┐ }
{ │ GORGETES_LE : MEGADOTT KÉPERNYŐTARTOMÁNY     │ }
{ │          LEFELE GÖRGETÉSE, 111. TÖRLÉSE      │ }
{ └──────────────────────────────────────────────┘ }
```

```
procedure GORGETES_LE (gsor,          { (G)örgetendő (sor)ok száma.    }
                       szin,          { A törölt terület (szín)e.      }
                                      { Képernyőtartomány              }
                       bfs,           {      (b)al  (f)első (s)ora.    }
                       bfo,           {      (b)al  (f)első (o)szlopa. }
                       jas,           {      (b)al  (a)lsó  (s)ora.    }
                       jao:integer);  {      (j)obb (a)lsó  (o)szlopa. }

var regiszter : dosreg;               { Regiszter rekord deklaráció.   }

begin
   regiszter.ax := 7 shl 8 + gsor;    { Görgetés funkció+ sorok száma. }
   regiszter.bx := szin shl 8;        { Az üres sorok színe.           }
   regiszter.cx := bfs shl 8 + bfo;   { Ablakkoor      -               }
   regiszter.dx := jas shl 8 + jao;   {                      dináták.  }
   intr($10, regiszter);              { 10(h) megszakítás.             }
end;

{ ┌──────────────────────────────────────────────┐ }
{ │ KARIR : EGY KARAKTER KIÍRÁSA ADOTT ATTRI-    │ }
{ │         BÚTUMMAL TETSZ. KÉPERNYŐOLDALRA      │ }
{ └──────────────────────────────────────────────┘ }

procedure KARIR(oldal :integer;     { A kiírandó karakter és a hozzá  }
                kar   : char;       { tartozó attribútum, ill. szín,  }
                szin  :integer);    { valamint a megjelenítés oldala. }

var regiszter : dosreg;               { Regiszter rekord deklaráció. }

begin
   regiszter.ax := 9 shl 8 + ord(Kar);    { Karakter megjelenítés fun+}
                                          { a megjelenítendő karakter.}
   regiszter.bx := oldal shl 8 + szin;    { Az aktuális képernyőoldal+}
                                          { a megjelenítés színe.     }
   regiszter.cx := 1;                     { Csak egy megjelenítés.    }
   intr($10, regiszter);                  { 10(h) megszakítás.        }
end;

{ ┌──────────────────────────────────────────────┐ }
{ │ IR : KARAKTERLANC KIÍRÁSA ADOTT KÉPER-       │ }
{ │      NYŐPOZÍCIÓTÓL KEZDVE ADOTT OLDALRA      │ }
{ └──────────────────────────────────────────────┘ }

procedure IR(oldal,                   { A megjelenítés koordinátái, a  }
             oszlop,                  { a kiírandó kar.lánc és a hozzá }
             sor,                     { tartozó attribútum, ill. szín. }
             szin : integer;
             txt : karlanc);

var regiszter : dosreg;               { Regiszter rekord deklaráció. }
    i         : integer;

begin
   KURPOZ(oldal, oszlop, sor);                { Kurzor pozícionálása. }

   for i := 1 to length(txt) do
   begin
      KARIR(oldal, ' ', szin);                { Karakter írás funkció.}
      regiszter.ax := 14 shl 8 + ord(txt[i]);
      regiszter.bx := oldal shl 8;
      intr($10, regiszter);
   end;
end;
```

```
{
{   F Ő P R O G R A M : MEGJELENíTŐ CIKLUS                    }
{                                                            }

var  i, j, k, l : integer;

begin
clrscr;
for i := 0 to 24 do                                  { Képernyő feltöltése }
   for j := 0 to 79 do                               { graf. karakterekkel. }
      begin
      KURPOZ(0, j, i);
      KARIR(0, chr(179+(i*80+j) mod 45), NM);
      end;

GORGETES_LE( 3, NM, 1, 10,  3, 69);

GORGETES_LE( 3, NM, 5,  4,  7, 10);                  { 2 x 6 db képernyő- }
GORGETES_LE( 3, NM, 5, 17,  7, 23);                  { ablak megnyitása   }
GORGETES_LE( 3, NM, 5, 30,  7, 36);                  { görgetéssel.       }
GORGETES_LE( 3, NM, 5, 43,  7, 49);
GORGETES_LE( 3, NM, 5, 56,  7, 62);
GORGETES_LE( 3, NM, 5, 69,  7, 75);

GORGETES_LE(14, NM, 9,  4, 22, 10);
GORGETES_LE(14, NM, 9, 17, 22, 23);
GORGETES_LE(14, NM, 9, 30, 22, 36);
GORGETES_LE(14, NM, 9, 43, 22, 49);
GORGETES_LE(14, NM, 9, 56, 22, 62);
GORGETES_LE(14, NM, 9, 69, 22, 75);

IR(0,13, 2, IM, 'Görgetés a leggyorsabb ablakban 1000x, jelenleg : ');

IR(0, 5, 6, IM, '1/32');
IR(0,18, 6, IM, '1/16');
IR(0,31, 6, IM, ' 1/8');
IR(0,44, 6, IM, ' 1/4');
IR(0,57, 6, IM, ' 1/2');
IR(0,70, 6, IM, '  1 ');

for i := 0 to 999 do                                 { Ciklikus görgetés  }
   begin                                             { és kiíratás, az    }
   KURPOZ(0,63,2);                                   { egyes ablakokban   }
   write(i:4);                                       { eltérő sebességgel. }

   GORGETES_FEL(1, NM, 10, 69, 21, 75);
   KURPOZ(0,72,21);
   KARIR (0,  chr(48 + i mod 10), NM);
   if (i mod 2 = 0) then
      begin
      GORGETES_LE (1, NM,10, 56, 21, 62);
      KURPOZ(0,59,10);
      KARIR (0,  chr(48 + (i div 2) mod 10), NM)
      end;
   if (i mod 4 = 0) then
      begin
      GORGETES_FEL(1, NM,10, 43, 21, 49);
      KURPOZ(0,46,21);
      KARIR (0,  chr(48 + (i div 4) mod 10), NM);
      end;
   if (i mod 8 = 0) then
      begin
```

```
      GORGETES_LE (1, KM,10, 30, 21, 36);
      KURPOZ(0,33,10);
      KARIR (0,  chr(48 + (1 div 8) mod 10), KM);
      end;
   if (1 mod 16 = 0) then
      begin
      GORGETES_FEL (1, KM,10, 17, 21, 23);
      KURPOZ(0,20,21);
      KARIR (0,  chr(48 + (1 div 16) mod 10), KM);
      end;
   if (1 mod 32 = 0) then
      begin
      GORGETES_LE (1, KM,10, 4, 21, 10);
      KURPOZ(0,7,10);
      KARIR (0,  chr(48 + (1 div 32) mod 10), KM);
      end;
   for l := 0 to 4000 do
      begin
      end
   end;
clrscr;
end.
```

1.4. A hajlékonylemezes meghajtó vezérlése

Az IBM PC piacra kerülésekor, az 1980-as évek legelején – állítólag – léteztek olyan típusok, amelyek az adatok, ill. programok tárolására kazettás magnetofonkészüléket használtak. Ezek a kazettás egységek azonban a gyakorlatban sohasem terjedtek el, és gyártásukat rövid időn belül meg is szüntették. A ma piacon lévő IBM PC/XT/AT számítógépek elsődleges háttértárolóként kizárólag hajlékony- és/vagy merevlemezes lemezmeghajtókat használnak. Bár a kétféle lemezmeghajtó BIOS megszakításokon keresztüli vezérlése sok tekintetben azonos, mégis célszerűnek látszik, hogy ezeket külön fejezetben tárgyaljuk. Ebben a fejezetben a hajlékonylemezes meghajtó (ismert angol nevén floppy disk drive) BIOS-ból történő vezérlésével ismerkedünk meg.

A hajlékony mágneslemez (angolul floppy disk vagy diskette) – amit ebben a fejezetben röviden csak lemeznek fogunk nevezni – adattárolási formátumaival az I. kötet DOS-ról szóló részének 6.12. Lemezkezelés című alfejezetében már részletesen volt szó, ezért ezeket itt csak röviden összefoglaljuk.

Az IBM PC/XT gépek lemezmeghajtóiban a DOS operációs rendszer alatt használatos lemezek mindkét oldala 40-40 sávra (angolul track), egy-egy sáv pedig 9 szektorra (sector) van felosztva. (A teljesség kedvéért megjegyezzük, hogy a kétoldalas lemezműveletekre csak a DOS 1.1 verziójától felfelé van lehetőség, és a 2.0-nál korábbi verziók a sávokat nem 9, hanem csak 8 szektorra osztották fel.) Egy szektor a DOS alapértelmezése szerint 512 bájtot tartalmaz. Ezek alapján könnyen kiszámítható, hogy a kétoldalas, 40 sávos, 9 szektoros lemez kapacitása 360 kbájt. Az IBM PC/AT gépekbe beépített hajlékonylemezes meghajtók 80 sávra, és sávonként 15 szektorra osztott lemezeket is képesek kezelni. E lemezek kapacitása 1.2 Mbájt. Hangsúlyozzuk, hogy a lemezeket a DOS

osztja fel ilyen adattárolási egységekre (a FORMAT parancs végrehajtásakor). Amint majd látni fogjuk, a BIOS funkciókkal ettől eltérő felosztás is létrehozható. Ez annyiban lehetővé teszi a lemezek másolás elleni védelmét, hogy az adatok olyan formátumban helyezhetők el a lemezen, amelyet a DOS nem tud értelmezni. Más kérdés, hogy ez egyrészt nem túl rafinált védelem, másrészt az így tárolt program más, DOS alatt futó gépeken nem olvasható be.

A lemezmeghajtók BIOS-ból történő vezérlésénél még egy apró, de lényeges körülményt figyelembe kell vennünk: míg a DOS a BIOS-szal megegyezően a lemezoldalakat (azaz a lemez egyik és másik oldalát író/olvasó fejeket) a 0 – 1., a sávokat pedig a 0 – 39. ill. 0 – 79. sorszámokkal azonosítja, addig a DOS a szektorokat 0-tól, a BIOS pedig 1-től kezdődően számozza. Ez a gyakorlatban azt jelenti, hogy egy DOS alatt formált lemez betöltő (boot) szektora a DOS számozása szerint a lemez 0. oldalán, a 0. sávon és a 0. szektoron helyezkedik el (ezt tekintjük a lemez 0. logikai szektorának). Ezzel szemben ugyanezt a szektort a BIOS a 0. oldal 0. sávjának 1. szektoraként tartja számon. Ennek megfelelően a 0. oldal 0 sávján a 0. szektorhoz való hozzáférést a BIOS tehát érvénytelennek is tekinti, és végrehajtásakor hibaüzenetet küld.

A könnyebb érthetőség céljából a 7. ábrán bemutatjuk, hogy a DOS „szabvány" szerint (40 sáv, 9 szektor) felosztott kétoldalas lemezen hogyan helyezkednek el az egyes sávok és szektorok a BIOS felosztása szerint.

	0. oldal szektorok									1. oldal szektorok								
	1	2	3	4	5	6	7	8	9	1	2	3	4	5	6	7	8	9
0. sáv																		
1. sáv																		
2. sáv																		
.																		
.																		
.																		
37. sáv																		
38. sáv																		
39. sáv																		

7. ábra: Egy hajlékonylemez BIOS szerinti felosztása

Ha már itt tartunk, térjünk vissza egy kicsit a DOS-hoz, és elevenítsük fel a lemezfelosztásról ott leírtakat. Eközben sort kerítünk arra is, hogy ismét foglalkozzunk a lemezek logikai és fizikai felosztásával kapcsolatos tudnivalókkal.

A DOS a lemezen a szektorokat sávfolytonosan alakítja ki: a szektorok formálását a 0. oldal 0. sávján kezdi, majd az 1. oldal ugyancsak 0. sávján folytatja. Miután mindkét

oldal 0. sávján létrehozta a szektorokat, rátér a 0. oldal 1. sávjára. Azt követően, hogy itt is kialakította a szektorokat, munkáját a másik oldal 1. sávján folytatja, és így tovább. Ennek megfelelően történik a lemez egyes szektorainak logikai sorszámozása is: a DOS a szektorokat a 0. oldal (fej) 0. sávjának 1. szektorától kezdődően (az ábra szerinti esetben) az 1. oldal 39. sávjának utolsó (kilencedik) szektoráig folytonosan sorszámozza. Így a sorszámozás a lemez logikai 0. szektorától (ami fizikailag a 0. oldal, 0. sáv első szektorának felel meg) a logikai 719. szektoráig (fizikailag az 1. oldal 39. sáv utolsó szektoráig) terjed.

Míg a DOS a lemez egyes szektorait az előzőekben részletezett logikai sorszámok szerint azonosítja (amivel a programozót mentesíti az adathordozó fizikai adottságaitól és felosztásától függő mechanikus átszámításoktól), a BIOS a lemez kezelésében ennél szigorúbb és puritánabb: elvárja, hogy a programozó ismerje annak az adathordozónak a fizikai felépítését, amelyen műveletet akar végezni, és természetesen azt is, hogy közölje, a kívánt műveletet a lemez mely részén óhajtja végrehajtani. Vagyis: ahhoz, hogy egy lemezműveletet végrehajtsunk, nem elég, ha csak a műveletet definiáljuk (az állományról, mint az adatok valamiféle célszerű tárolási lehetőségéről a BIOS még csak nem is hallott), hanem azt is meg kell adnunk, hogy a BIOS a műveletet a lemez melyik oldalán, melyik sávján és mely szektora(in) végezze el.

Ennek megfelelően a lemez egy meghatározott részének – BIOS szintű – azonosításához legalább három paraméterre van szükség: a lemezoldal (író/olvasó fej), a sáv és a szektor megadására. Ehhez jön még egy negyedik paraméter: mivel a BIOS megengedi, hogy az egyes sávokon belül a szektorok különböző méretűek legyenek, ugyanakkor a BIOS legkisebb „hozzáférési egysége" egy szektor, azt is meg kell adni, hogy mekkora ez a hozzáférési egység, azaz a szektorméret. Ezt a négy, egyenként egy-egy bájton tárolt paramétert, amelyet a szakirodalom a

C (Cylinder = hajlékonylemezek esetén a sáv megfelelője),
H (Head = író/olvasó fej, azaz lemezoldal),
R (Record = a szektor megfelelője) és az
N (Number = szektorméret bájtokban, kódolva)

betűkkel jelöl, és mint adatszerkezetet, „address field"-nek nevez (magyarul címmezőnek, címjelölésnek vagy szektorazonosítónak mondható), a BIOS a lemezen megformált minden egyes szektor adatbájtjai elé felírja. Az ezekben a címjelölésekben tárolt paraméterek teszik lehetővé, hogy a BIOS a kért szektort lemezoldal, sáv és szektorméret szerint – vagyis fizikailag – azonosítsa és megtalálja.

Ennyi bevezető után lássuk azokat a funkciókat, amelyeket a BIOS a hajlékonylemezes meghajtó vezérléséhez rendelkezésünkre bocsát. Alapvetően 6 ilyen funkció van, de ezek száma a PC/AT megjelenésekor további 3-mal bővült. Mindegyik funkció a 19 – 13(h) megszakításon keresztül hívható meg, a funkció számát pedig mindegyik esetben az AH regiszterbe kell beírni. A funkciók:

0 – 00(h) reset
1 – 01(h) státusz olvasása
2 – 02(h) olvasás
3 – 03(h) írás
4 – 04(h) verifikálás
5 – 05(h) formálás
21 – 15(h) meghajtó típusának megállapítása (csak AT)
22 – 16(h) hajlékonylemez cseréjének megállapítása (csak AT)
23 – 17(h) hajlékonylemez formátumának meghatározása (csak AT)

A 0. számú funkció alapállapotba állítja a lemezmeghajtót, és az író/olvasó fejeket a 0. sávra pozícionálja (elterjedt angol nevén ezt reset funkciónak nevezik). Ez a művelet a rendszer minden egyes újraindításakor automatikusan végrehajtódik, de akkor is el kell végezni, ha valamelyik lemezműveletet végző funkció hibajelzéssel tér vissza. A funkció meghívásához az AH regiszterbe a 0 értéket kell beírni. A DL regiszterben, amely elvileg a meghajtó azonosító számát tartalmazza, tulajdonképpen bármilyen érték állhat: a funkció ugyanis valamennyi csatlakoztatott hajlékonylemezes meghajtón elvégzi a reset műveletet. A különbség csak annyi, hogy ha ez az érték 0 – 127, akkor a művelet csak a hajlékonylemezes, ha pedig 128 – 255, akkor a merevlemezes meghajtóra is vonatkozik. A funkció – csakúgy mint az 1 – 5. sorszámú funkciók – a végrehajtását követően ugyancsak az AH regiszterbe egy értéket ír vissza. Ez az érték arról ad felvilágosítást, hogy a művelet végrehajtása során történt-e hiba, és ha igen, akkor ez milyen jellegű. A 0 – 5. funkciók sikeres végrehajtását az átvitelbit 0 értéke jelzi, és ebben az esetben az AH tartalma is 0. Ha a végrehajtás során hiba lépett fel, akkor az átvitelbit értéke 1 lesz, az AH regiszterbe pedig a hiba kódja kerül.

Az 1. számú funkció azt teszi lehetővé, hogy úgy is leolvashassuk ezt a hibakódot, hogy ehhez előzőleg ne kelljen valamilyen lemezműveletet végrehajtanunk. A hibakód ekkor az utoljára végrehajtott lemezművelet utáni állapotot tartalmazza. Ebből következik, hogy a lemezmeghajtó DL-ben átadott azonosító számának (értéke akár 0 – 127 is lehet) itt sincs jelentősége. A visszaadott hibakód helyére vonatkozóan engedtessék meg némi bizonytalanság: A különböző szakirodalmi források ebben a dologban eltérőek: egyes források szerint a művelet sikerességét az átvitelbit jelzi, mások erről nem tesznek említést. Egyes források szerint a hibakód az AH, mások szerint az AL-be kerül. A saját, meglehetősen kevés számú gépen szerzett tapasztalatok szerint a művelet sikeres végrehajtását az átvitelbit 0 értéke jelezte, a hibakódot pedig mind az AH, mind az AL regiszter tartalmazta.

Az egyes hibakódok jelentése:

01(h) : nem megengedett funkciószám
02(h) : címjelölést nem találta
03(h) : írási kísérlet írásvédett lemezre
04(h) : a keresett szektort nem találta
06(h) : lemezcsere az utolsó lemezművelet után (csak AT)
08(h) : DMA túlcsordulás
09(h) : adatátvitel a szegmenshatáron túl
10(h) : olvasási hiba
20(h) : lemezvezérlő hiba
40(h) : sávot nem találta
80(h) : időtúllépés, lemezmeghajtó nem válaszolt (Time Out)

Ha az egyes BIOS funkciók meghívásakor a fenti hibák valamelyike fellép, akkor ez még nem jelenti feltétlenül azt, hogy a lemezvezérlő, a lemezmeghajtó vagy a lemez hibás. A lemez olvasását végző funkció pl. az első kísérletre sokszor térhet vissza a 80(h) kódszámú hibakóddal (időtúllépés), aminek egyszerűen csak az az oka, hogy a funkció végrehajtásakor a lemezmeghajtó motorja még nem érte el a hibátlan olvasáshoz szükséges forgási sebességet. Ekkor a lemezmeghajtót a 0. funkcióval alapállapotba kell hozni, és a műveletet legalább háromszor meg kell ismételni, mielőtt valóban hardverhibára gyanakodnánk.

A lemez olvasásához a 2. számú funkciót kell meghívni. Ezzel a funkcióval a lemez bármelyik szektora – beleértve a betöltő szektort, a FAT-ot, ill. a katalógust tartalmazó, ún. rendszerterület szektorait is – beolvasható a számítógép operatív tárába. A funkció meghívásához meg kell adni, hogy a beolvasandó szektor a lemezen hol helyezkedik el (hajlékonylemezek esetén a szektorok logikai sorszáma többnyire követi a fizikai elhelyezkedésük szerinti sorrendet), továbbá a tárban ki kell jelölni egy puffert, amelybe a funkció a szektor(ok) tartalmát beolvashatja. A megadandó paraméterek és az ezeket átadó regiszterek:

DL = lemezmeghajtó száma (0 – 3)
DH = lemezoldal száma (0 vagy 1)
CH = sáv száma (0 – 39, ill. PC/AT-nél 0 – 79 is)
CL = szektor száma (1 – 9, ill. PC/AT-nél 1 – 15)
AL = olvasandó szektorok száma
ES = a puffer szegmenscíme
BX = a puffer ofszetcíme

Mivel ezzel a funkcióval egyidejűleg egynél több szektor is beolvasható, a lemezen tárolt adatok lényegesen gyorsabban olvashatók be, mint a megfelelő DOS funkcióval. Míg a DOS egy lemezfordulat során csak egy szektort olvas be, ez a BIOS funkció a megadott lemezoldalon és sávon található valamennyi szektort egy fordulat alatt képes beolvasni. A beolvasandó szektorok számát az AL regiszterben kell átadni.

A művelet végrehajtásának sikerét most is az átvitelbit értéke jelzi: ha ez az érték 0, akkor a művelet sikeres volt, és az AH regiszterbe 00 íródik. Ellenkező esetben az AH-ban a hiba kódját kapjuk vissza.

Az olvasási funkcióhoz hasonlóan kell a 3. számú, írási funkciót is meghívni. A megadandó paraméterek, és az ezeket átadó regiszterek:

DL = lemezmeghajtó száma (0 – 3)
DH = lemezoldal száma (0 vagy 1)
CH = sáv száma (0 – 39, ill. PC/AT-nél 0 – 79 is)
CL = szektor száma (1 – 9, ill. PC/AT-nél 1 – 15)
AL = írandó szektorok száma
ES = a puffer szegmenscíme
BX = a puffer ofszetcíme

Az olvasáshoz hasonlóan íráskor is egy funkcióhívással egyidejűleg több szektor is lemezre írható. Itt is érvényes az a megszorítás, hogy egy művelettel egyidejűleg csak a megadott lemezoldal és sáv szektoraira lehet adatokat írni. A hibajelzés is azonos az olvasási funkciónál leírtakkal.

A 4. sorszámú funkció segítségével ellenőrizhető, hogy az adatok lemezre írása, ill. beolvasása rendben lezajlott-e. A vizsgálat, amit verifikálásnak is nevezünk, nem a tárban és a lemezen lévő adatokat hasonlítja össze egymással, hanem a szektorok bájtjaiból egy számítási eljárással, az ún. ciklikus redundanciavizsgálattal (CRC, cyclic redundancy check) képzett vizsgálóösszegeket ellenőrzi. Ez az ellenőrzés DOS szintről is elvégezhető a VERIFY ON (verifikálás BE) parancs kiadásával. Ugyancsak ezt a funkciót használja a DOS akkor, ha a COPY parancsot a /V paraméterrel adjuk ki. Tekintettel azonban arra, hogy a lemezműveletek igen megbízhatók, és a verifikálás jelentősen megnöveli a végrehajtási időt, a felhasználói programok ezt a funkciót általában nem használják.

A regiszterek szereposztása ugyanaz mint a 2., ill. 3. funkciónál azzal a különbséggel, hogy az AH regiszterbe most természetesen 4-et kell írni. Az ES:BX regiszterpár megadását illetően az egyes szakirodalmi források eltérnek egymástól. Az IBM ROM-BIOS szerint sem az XT, sem az AT esetén nem kell megadni a regiszterpárt. Ebből következően a funkció nem tudja a CRC-t ellenőrizni, hanem csak a szektor bájtok olvashatóságát vizsgálja.

Az 5. sorszámú funkcióval a lemez egy megadott sávját formálhatjuk. Egy új, még nem formált lemez egy-egy oldala egy egybefüggő mágnesréteget alkot, amely ilyen formájában még nem alkalmas arra, hogy az adatokat a rendszer számára értelmezhető szerkezetben tárolja (ezek az ún. „soft-sectored" lemezek). A BIOS az 5. számú funkció végrehajtásakor a lemez általunk megadott részén megfelelő információk felírásával létrehoz egy sávot, amelyet egy ugyancsak általunk megadott paraméter szerint szektorokra oszt fel. Mint tudjuk, a DOS FORMAT parancsa a lemezeket az alapértelmezés szerinti 40, ill. 80 sávra, sávonként 9, ill. 15 szektorra formálja, és a szektorok méretét 512 bájtra állítja be. A BIOS 19 – 13(h) megszakításának 5. funkciójával ezektől az értékektől eltérhetünk: a sávok, ill. a sávokon belüli szektorok száma a fenti értékeknél kisebb is lehet, egy szektor pedig – választásunk szerint – 128, 256, 512 vagy 1024 bájtot tartal-

mazhat. A sáv szabad megválasztása azt is jelenti, hogy olyan lemezformátum is létrehozható, amelynek különböző sávjai különböző számú szektorokból állnak. Annyi megszorítás azért van, hogy a funkcióval egyidejűleg egy teljes sáv formálható, de egy sávon belül a szektorok eltérő méretűek is lehetnek. Amint a fejezet elején említettük, azzal, hogy ezeket a paramétereket a funkció meghívásakor az adott korlátok között szabadon megválaszthatjuk, kialakíthatunk olyan lemezformátumot, amely bizonyos védelmet nyújt a lemez másolásával szemben. A nem DOS formátumú lemezeket ugyanis a DOS nem tudja beolvasni, így természetesen másolni sem képes. Ennek ellenére a DOS-tól eltérő lemezformátumok létrehozása nem ajánlott.

A funkció meghívásához a AH regiszterbe a funkció számát, a CH-ba a formálandó sáv sorszámát kell beírni. A meghajtó azonosító számát a DL, a lemezoldal számát pedig a DH regiszternek kell tartalmaznia.

A fenti adatokon kívül az ES:BX regiszterpárnak még egy pufferre kell mutatnia, amelynek a formálásra és a sáv/szektor azonosítására vonatkozó paramétereket kell tartalmaznia. Ezek a paraméterek alkotják az ún. címjelöléseket vagy címmezőket (address field), amelyek tartalma formáláskor minden egyes szektor elé felíródik. Az író és olvasó műveletek a címmezőben lévő információk alapján keresik meg az írásra vagy olvasásra kiválasztott szektort. Az egyes címmezők az alábbi – a fejezet elején már ismertetett – 4 bájtból állnak:

1. bájt :	formálandó sáv sorszáma (0 – 39/79)	(C)
2. bájt :	lemezoldal (fej) száma (0 vagy 1)	(H)
3. bájt :	formálandó szektorok száma (1 – 9/15)	(R)
4. bájt :	szektor mérete bájtokban	(N) a következők szerint:
	0 = 128 bájt/szektor	
	1 = 256 bájt/szektor	
	2 = 512 bájt/szektor (DOS szerinti alapértelmezés)	
	3 = 1024 bájt/szektor	

Amint látjuk, a címmezők első két bájtja ugyanazokat a paramétereket tartalmazza, mint a CH és a DH regiszter: a formálandó sáv sorszámát és a lemezoldal számát. Ennek ellenére ezeket az adatokat a címmezőknek is tartalmazniuk kell. Egy sáv formálásához a puffernek annyi 4 – 4 bájtos címmezőt kell tartalmaznia, ahány szektort a sávban el akarunk helyezni. Ezen belül az egyes címmezők első két bájtja nem változhat, mert egy funkcióhívással csak ugyanazon oldal egyetlen sávja formálható. A szektorokat viszont az 1-es sorszámtól kezdődően változó sorszámokkal kell ellátni.

A rendszer a lemezen a szektorokat fizikailag olyan sorrendben helyezi el, amilyen sorrendben az egyes szektorokat jelölő címmezők az ES:BX regiszterpár által kijelölt pufferben elhelyezkednek. Hajlékonylemez esetén a sávon belül a szektorokat általában folytonosan számozzák, ami azt jelenti, hogy a szektorok logikai sorszáma követi a fizikai elhelyezkedésüket. A merevlemezes meghajtóval foglalkozó fejezetben látni fogjuk, hogy ott a kétféle számozás eltérő.

A címmezőn belül a szektor méretét meghatározó bájt értéke szektoronként ugyancsak változhat, ami szintén felhasználható a lemez másolás elleni védelmére.

Nézzük meg a 8. ábrán, hogy miként kell elkészíteni egy szektorazonosító címjelölését. Az ábra olyan címmezőtáblát mutat be, amellyel egy lemez 0. oldalának (H = 0) 5. sávján (C = 5) 9 szektort (R = 1 – 9) formálhatunk meg, és mindegyik szektor 512 bájtos (N = 2).

A címmezők bájtjai

C	H	R	N
5	0	1	2
5	0	2	2
5	0	3	2
5	0	4	2
5	0	5	2
5	0	6	2
5	0	7	2
5	0	8	2
5	0	9	2

8. ábra: Egy sáv formálásához szükséges címmezőtábla szerkezete (példa)

Az eddigiekben említett paraméterekkel azt adtuk meg, hogy a funkció a formálandó szektor(ok)at a lemezen fizikailag hol hozza létre (oldal, sáv), valamint azt, hogy mekkora legyen a szektor. Ezen kívül azonban a BIOS-nak még számos más, olyan paraméterre is szüksége van, amelyek nélkülözhetetlenek a különböző lemezműveletek (írás, olvasás, verifikálás, formálás) helyes végrehajtásához. Ezeket a paramétereket azonban nem nekünk kell megadni, hanem a ROM BIOS a lemezmeghajtó és (részben a lemez) fizikai jellemzőinek megfelelően eleve tartalmazza. Ezek a paraméterek egy 11 bájt hosszúságú adatszerkezetet alkotnak, amelyet lemez paramétertáblának nevezünk (az angol nyelvű szakirodalom a disk base, ill. a diskette parameter table kifejezéseket használja). A kezdeti időkben a BIOS e paraméterek értékeit (pl. a fej beállását meghatározó időt) meglehetősen nagy „ráhagyással" állapította meg. Az újabb verziók – bizonyára a lemezmeghajtók biztonságos működésével kapcsolatban szerzett kellemes tapasztalatok alapján – a korábbi értékeket „megszigorították", és ezzel lényegesen meggyorsították a lemezműveleteket. Egy másik, nem kevésbé fontos változtatás az volt, hogy a BIOS a paramétertábla bájtjait a ROM-ból áttölti egy RAM területre, és ennek a területnek a címét beírja az 1E(h) megszakítási vektorba (a megszakítási vektortábla 0000:0078 – 007B címeire). Ezzel lehetővé válik, hogy ezekhez a paraméterekhez a DOS (és a programozó is) hozzáférjen, sőt akár módosíthassa is. Itt jegyezzük meg, hogy az újabb BIOS verziók által megállapított értékek feltehetően az optimális értékek, ezért ezeknek felhasználói programból való megváltoztatása semmi esetre sem ajánlott (egyes fejmozgatási értékek helytelen megválasztása a lemezmeghajtó meghibásodását is okozhatja).

A következőkben röviden ismertetjük a lemez paramétertábla 11 bájtjának jelentését:

0. bájt: Ez a bájt két paramétert tartalmaz. A felső félbájton (4 – 7. bit) álló érték azt a maximális időközt adja meg, amelyen belül az író/olvasó fejnek az egyik sávról egy másik sávra át kell lépnie (angol neve Step Rate Time, SRT). Az alsó félbájton (0 – 3. bit) lévő érték a fej felemeléséhez szükséges időt adja meg. A bájt szokásos értéke DF(h).

1. bájt: Ez a bájt a DMA-módra és a fej letevésének idejére vonatkozó adatokat tartalmaz.

2. bájt: Azt adja meg, hogy az utolsó lemezműveletet követően mennyi idő elteltével kell a lemezmeghajtó motorját kikapcsolni. A rendszer feltételezi, hogy egy lemezműveletet közvetlenül egy újabb követ, és ezért – hogy ne kelljen az újabb felpörgésre várni – nem kapcsolja ki azonnal a motort. A bájt értéke a rendszer ütemidejének (kb. 1/18 másodperc) egységében van megadva, vagyis a 18-as érték 1 másodpercet jelent. A bájt szokásos értéke 25(h) azaz 37, ami nagyjából két másodpercet jelent.

3. bájt: A szektorhossz kódja az előzőekben leírtak szerint. A DOS alapértelmezése szerint egy szektor hossza 512 bájt, aminek a 2-es kód felel meg. Ha a formálási művelet egy lemezen (vagy valamely sávján) ettől eltérő szektorméretet állított be, akkor a lemez (ill. a megfelelő sáv szektorai) csak úgy írhatók/olvashatók, ha ennek a bájtnak az értékét a formáláskor megadott értékre változtatjuk.

4. bájt: A lemez egy sávjában elhelyezhető szektorok maximális számát adja meg (szokásos értéke 09(h), ill. AT esetén 0F(h)).

5. bájt: Az egyes szektorok közötti (időben mért) távolságot (angolul gap) tartalmazza: azt adja meg, hogy a lemezmeghajtó egy szektor írás/olvasásakor mennyi idő elteltével várható a következő szektor.

6. bájt: Ez a bájt adja meg a szektor méretét azokra az esetekre, amikor a szektorhosszt a 3. bájt nem írja elő.

7. bájt: Ez az érték azt adja meg, hogy formáláskor az átmágnesezett hosszakat (= szektorokat) milyen hosszúságú „üres" mágneses szakaszok válasszák el egymástól. Ahhoz, hogy az írási/olvasási műveleteket elegendő biztonsággal végre lehessen hajtani, ennek az értéknek természetesen nagyobbnak kell lennie az 5. bájton lévő értéknél.

8. bájt: Annak a karakternek az ASCII-kódját tartalmazza, amellyel a lemez szektorai formáláskor feltöltődnek. A DOS alapértelmezés szerint az osztás (:) jelével tölti fel a szektorokat (F6(h) ASCII-kód).

9. bájt: Az író/olvasó fej mozgásának lecsengési idejét adja meg. Miután a fej az egyik sávról áttért egy másik sávra, némi időre van szüksége ahhoz, hogy az ebből a mozgásból származó kilengései megszűnjenek, és nyugalmi állapotból kezdhessen hozzá egy írási/olvasási műveletnek. A bájt értéke ezt az időt milliszekundumban adja meg.

10. bájt: A lemezmeghajtó motorjának felpörgéséhez szükséges időt adja meg 1/8 másodperces egységekben. A DOS ezt az értéket 2-re állítja be, ami 1/4 másodperces várakozási időt jelent.

Ezzel befejeztük a lemezmeghajtók vezérlésében fontos szerepet játszó lemez paramétertábla bájtjainak ismertetését.

Mint korábban már említettük, a BIOS a hajlékonylemezes meghajtó vezérlésére alapvetően 6 funkciót bocsát rendelkezésünkre. Az IBM a PC/AT számítógéppel egyidejűleg egy nagy kapacitású és nagy teljesítményű hajlékonylemezes meghajtót is bevezetett. Ez a meghajtó arra is képes, hogy a korábbi, PC és XT gépek meghajtóiban használt 320/360 kbájtos lemezek mellett – megtartva a lemez 5.25"-os átmérőjét – 1.2 Mbájt kapacitású lemezeket is kezeljen. Az AT meghajtója úgy éri el ezt a nagy kapacitást, hogy a lemezeket a korábbi 40 sáv helyett 80 sávra formázza, és egy-egy sávon 9 helyett 15 szektort hoz létre, változatlan szektorhossz (512 bájt) mellett. Ha ennek a meghajtónak a megnövelt kapacitását ki akarjuk használni, akkor az általuk kezelt lemezeknek már nem csak kétoldalasaknak (double sided, DS), hanem nagy írássűrűségűeknek (high density, HD) is kell lenniük. A kompatibilitással kapcsolatban általában elmondható, hogy míg a PC és XT meghajtóiban formált lemezeket az AT nagykapacitású lemezmeghajtója minden további nélkül képes írni és olvasni, az AT-ben 320/360 kbájtra formált lemezeket az XT meghajtója nem biztos, hogy írni, ill. olvasni tudja.

A megnövelt kapacitás mellett az új lemezmeghajtó még arra is képes, hogy felismerje, történt-e két lemezművelet között lemezcsere. Ennek a ténynek különösen az operációs rendszerek hatékonysága szempontjából van jelentősége: az operációs rendszer ugyanis egy lemezművelet elvégzése előtt a kezelésbe vett lemez minden lényeges információját beolvassa a belső tárba, hogy a műveletet a lemez formátumának, tartalmának stb. megfelelően végre tudja hajtani. Ha az operációs rendszernek lehetősége van arra, hogy egy lemezművelet végrehajtása előtt megállapítsa, hogy a műveletet ugyanazon a lemezen kell elvégeznie, amelyen a megelőzőt is végezte, akkor a lemezre vonatkozó, előbb említett információkat nem kell ismételten beolvasnia. Az így megtakarított idő jelentős sebességnövekedést tesz lehetővé. Másrészt a lemezcsere lekérdezése a biztonságot is növeli: ha két, ugyanazon lemezre (állományra) vonatkozó művelet között az operációs rendszer lemezcserét észlelt, akkor a felhasználói program ezt a tényt lekérdezheti, és ennek megfelelően reagálhat (pl. az előbb használt lemez visszahelyezését kéri). Így elkerülhető, hogy egy előző művelet eredményeként az operatív tár puffereiben lévő adatok véletlen csere folytán egy másik lemezre íródjanak.

Nézzük meg most azokat a funkciókat, amelyek csak a PC/AT gépeken állnak rendelkezésünkre. Ezek a funkciók egyébként az előzőekhez hasonlóan ugyancsak a 13(h) megszakításon keresztül hívhatók meg.

A 21 – 15(h) számú funkció a lemezmeghajtó típusának megállapítására szolgál. Meghívásához az AH regiszterben a funkció számát, a DL regiszterben pedig a lemezmeghajtó azonosító számát (0 vagy 1) kell megadni. A funkció a sikeres végrehajtást (átvitelbit = 0) követően az AH regiszterben visszaadott kóddal jelzi a művelet eredményét:

AH = 0 : nincs lemezmeghajtó
AH = 1 : hajlékonylemezes meghajtó, a meghajtó nem képes a lemezcsere felismerésére
AH = 2 : hajlékonylemezes meghajtó, a meghajtó képes a lemezcsere felismerésére
AH = 3 : merevlemezes meghajtó

Amint látjuk, ez a funkció csak azt tudja megállapítani, hogy a vizsgált lemezmeghajtó képes-e a lemezcsere felismerésére. Ahhoz, hogy magának a lemezcserének a tényét is megállapíthassuk, a 16(h) funkciót kell meghívnunk. A bemeneti paraméterek megegyeznek az előbbiekkel. Ha a funkció visszetérésekor az átvitelbit értéke 0, akkor ez azt jelenti, hogy a meghajtóban a legutolsó lemezművelet óta nem történt lemezcsere (AH tartalma ugyancsak 0). Ellenkező esetben (átvitelbit = 1) lemezcsere történt, és az AH tartalma 06. (Megjegyezzük, hogy azoknál a típusoknál, amelyeknél a lemezcserét figyelő vezeték aktív állapota a lemezmeghajtó nyílását lezáró kilincs állásától függ, már a kilincs kinyitása is lemezcserét jelent anélkül, hogy magát a lemezt a meghajtóból kivennénk.)

Végül a 23 – 17(h) funkcióval azt határozhatjuk meg, hogy egy lemezt milyen típusú meghajtóban milyen formátumra alakítsunk ki. Amint láttuk, magát a formálást a BIOS 13(h) megszakítás 5-ös funkciója végzi, de ezt megelőzően az AT lemezmeghajtójának azonosítására ezt a funkciót meg kell hívnunk, hogy megtudjuk, a formálást 320/360 kbájtos vagy 1.2 Mbájtos lemezegységben, 320/360 kbájtra vagy 1.2 Mbájtra kell-e végrehajtani. A funkció számát most is az AH, a meghajtó számát (0 vagy 1) a DL regiszterben kell megadni, míg a formálásra vonatkozó kódot az AL regiszternek kell tartalmaznia a következők szerint:

AL = 1 : formálás 320/360 kbájtra, 320/360 kbájtos meghajtóban
AL = 2 : formálás 320/360 kbájtra, 1.2 Mbájtos meghajtóban
AL = 3 : formálás 1.2 Mbájtra, 1.2 Mbájtos meghajtóban

1.4.1. Példaprogramok

A hajlékonylemezek kezeléséhez próbáltunk segítséget nyújtani egyszerű és rövid monitor programjainkkal. A BASIC, Turbo Pascal és C nyelven megírt programpéldák felépítése azonos. Az egyes monitorfunkciók tárgyalása előtt tekintsük át röviden a programokat felhasználói szempontból.

A monitor első lépésként kezdőértéket ad a választott lemezegység sorszámát és a formálási kódot tartalmazó változóknak (ezek alapértelmezése 0, ill. 3). A későbbiekben a monitor nem kérdez rá az egyes műveletek előtt ezekre a paraméterekre. Amennyiben mégis meg akarjuk változtatni értéküket, arra külön funkció van a főmenüben. Ezt követően a hajlékonylemez meghajtók inicializálása után a monitor prompttal jelzi mindegyik program, hogy kész parancs beolvasására és végrehajtására. Maga a „Segítség = ? >" prompt közvetlenül utal arra, hogy prompt szintről a „?" karakter beadásával bármikor segítség kérhető a parancsokról.

Segítséget kérve láthatjuk, hogy a közvetlenül a lemezt manipuláló parancsok köre (formálás, olvasás, szektor inicializálás) egy kivétellel teljes : hiányzik az írás művelet a főmenüből. Ha demonstrációs szemszögből nézzük a programokat, akkor ez a hiány látszólagos, hiszen egy szektor feltöltése azonos karakterekkel olyan speciális írási folyamat, amelynek során a lemezre írandó adatpuffer csak azonos karaktereket tartalmaz. Felhasználói oldalról viszont valóban nem lehet egy szektort tetszőleges tartalommal feltölteni. Ennek az az oka, hogy a programok nem tartalmaznak erre a célra kifejlesztett

saját kis szövegszerkesztőt, ami nélkül nincs igazán értelme külön írási művelet definiálásának.

A lemezkezelésre vonatkozó parancsok minden esetben bekérik a feldolgozni kívánt lemezoldalt és sávot. Harmadik paraméterként inicializálás és olvasás esetén a szektor sorszámát, formálásnál pedig a kialakítandó sávok számát olvassák be a programok. Ahogy már említettük, a lemezegység sorszáma és (AT esetén) a formálási kód, éppen mert ezeket viszonylag ritkán szükséges megváltoztatni, ilyenkor nem kerülnek bekérésre.

Most pedig ejtsünk néhány szót a fontosabb parancsokról. Az olvasás során a monitor beolvassa belső pufferébe a kért szektor tartalmát (itt jegyezzük meg, hogy az egyszerűség kedvéért a monitorok minden műveletben a DOS-ban szokásos 512 bájtos szektorhosszal dolgoznak, a formálás során is ekkora szektorok keletkeznek). A képernyőn a megjelenítés két lépésben történik, egy-egy lépés 256 bájtot jelenít meg. A bájtok sorszáma így a felső 256 bájt megjelenítésekor 256 bájtnyi eltolással értendő a szektor elejéhez képest. A hexadecimális megjelenítés mellett karakteresen is megjelennek az adatok, azzal a megszorítással, hogy 32-nél kisebb ASCII kód esetén csak egy pontot írnak ki a monitorok. A formálás által használt formálási kód értéke gép- és lemezfüggő és befolyásolja a formáláskor kialakítandó szektorok számát is. 360 Kbájtos lemezen 1 – 9, 1.2 Mbájtos hajlékonylemezen 1 – 15 szektor alakítható ki sávonként.

A BASIC nyelvű programváltozatban mint oly sokszor, most is problémát jelent a „szemétgyűjtés", ami az 512 bájtos (256 egész típusú változóból álló tömb) adatpuffer elcsúszását eredményezheti. Ezt elkerülendő az írási (inicializálási) és olvasási műveleteknél közvetlenül a BIOS funkció meghívása előtt kell meghatározni az adatpuffer ofszetcímét (az adatszegmens ofszetcíméhez az ES által tárolt szegmenscím adódik majd hozzá, ezért az ES regiszterben a funkciónak –1-et kell átadni, ami a BASIC adatszegmens DS címének ES-be történő másolását fogja eredményezni). A Turbo Pascal és C monitorok esetén a fenti probléma természetesen nem jelentkezik, ezeknek a változatoknak további előnye, hogy jól struktúráltak, tetszőleges hajlékonylemez-kezelő funkció könnyen átültethető belőlük egy másik programba.

```
1000 '
1005 '
1010 '                          L E M O N . B A S
1020 '
1030 '  Funkciója : alapvetõ hajlékonylemez monitor funkciók szem-
1040 '              léltetése egy egyszerû lemezmonitoron keresztül.
1050 '
1060 '
1070 '
1080 DEFINT Q                                   'Q "dummy" egész érték.
1090 KEY OFF : CLS
1100 PRINT : PRINT
1110 PRINT "FIGYELMEZTETÉS :" : PRINT
1120 PRINT "A 60000-es címtõl kezdõdõen betoltott megszakításhívó ru-"
1130 PRINT "tin felulírását elkerülendõ a GWBASIC-et GWBASIC /M:60000"
1140 PRINT "paranccsal kell indítania. Ha nem így hívta a GWBASIC-et,"
1150 PRINT "akkor usse le az <s> billentyût, egyébként egy  tetszõle-"
1160 PRINT "ges másik billentyût !"
1170 '
1180 Z$ = ""
1190 WHILE Z$ = ""                              'Helytelen indítás
1200    Z$ = INKEY$                             'esetén a program
1210 WEND                                       'leállítása.
1220 IF Z$ = "s" OR Z$ = "S" THEN CLS:END
1230 '
1240 GOSUB 9000                                 'Megszakításhívó rutin
1250                                            'betoltése.
1260 CLS
1270 '
2000 '
2010 '          P R O G R A M T o R Z S
2020 '
2030 '
2040 DIM V%[255]                    ' Adatvektor 512 kbájtos szektorhoz.
2050 DIM SZP%[29]                   ' Szektorokat leíró puffer.
2060 Q = 0
2070 EGYSEG% = 0                    ' Alapértelmezés : 0. meghajtó,
2080 LEMEZFORM% = 3                 '                  1,2 MB-os egység.
2090 '
2100 DEF SEG = &HF000               ' Géptípus megállapítása,ha FFFF:FFFE
2110 '                              ' tartalma FC(h), akkor AT a gép.
2120 '
2130 IF PEEK(&HFFFE)=&HFC THEN EZEGYAT% = - 1 ELSE EZEGYAT% = 0
2140 DEF SEG
2150 '
2160 GOSUB 3000                                 ' Alapállapot beállítása.
2170 GOSUB 4000                                 ' Hibaüzenet kiadása.
2180 PRINT
2190 INPUT "Segítség : ? > ",E$                 ' Felhasználói választás.
2200 '
2210 IF E$ = "?"              THEN GOSUB 5000 : GOTO 2180   ' Help.
2220 IF E$ = "r" OR E$ = "R" THEN GOTO  2160                ' Reset.
2230 IF E$ = "i" OR E$ = "I" THEN GOSUB 6000 : GOTO 2170    ' Inic.
2240 IF E$ = "f" OR E$ = "F" THEN GOSUB 7000 : GOTO 2170    ' Format.
2250 IF E$ = "o" OR E$ = "O" THEN GOSUB 8000 : GOTO 2170    ' Olvas.
2260 IF E$ = "p" OR E$ = "P" THEN GOSUB 8800 : GOTO 2180    ' Param.
2270 IF E$ = "q" OR E$ = "Q" THEN CLS : END                 ' Vége.
2280 PRINT "érvénytelen parancs ..."  : GOTO 2180
2290 '
3000 '
3010 '  A HAJLÉKONYLEMEZ VEZÉRLÕ ÉS A MEGHAJTÓ-
3020 '  EGYSÉGEK INICIALIZÁLÁSA
3030 '
3040 '
```

```
3050 FUNX = 0                                          ' RESET funkció.
3060 MSZX = &H13                                       ' 13(h) megszakítás.
3070 '
3080 CALL RCIM(MSZX,FUNX,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q)
3090 RETURN
3100 '
4000 '
4010 '  HIBAUZENET MEGJELENITESE, HA AZ UTOLSO
4020 '  LEMEZMOVELET NEM VOLT SIKERES
4030 '      BE : HIBAKOD
4040 '
4050 '
4060 IF FUNX = 0 THEN RETURN
4070 PRINT:PRINT "Hiba lépett fel : ";
4080 IF FUNX = &H1  THEN PRINT "nem megengedett funkciószám":GOTO 3000
4090 IF FUNX = &H2  THEN PRINT "címjelölés nem található"   :GOTO 3000
4100 IF FUNX = &H3  THEN PRINT "írásvédett lemez"           :GOTO 3000
4110 IF FUNX = &H4  THEN PRINT "szektor nem található"      :GOTO 3000
4120 IF FUNX = &H6  THEN PRINT "lemez kicserélve"           :GOTO 3000
4130 IF FUNX = &H8  THEN PRINT "DMA túlcsordulás"           :GOTO 3000
4140 IF FUNX = &H9  THEN PRINT "átvitel szegmenshatáron túl":GOTO 3000
4150 IF FUNX = &H10 THEN PRINT "olvasási hiba"              :GOTO 3000
4160 IF FUNX = &H20 THEN PRINT "lemezvezérlő hiba"          :GOTO 3000
4170 IF FUNX = &H40 THEN PRINT "sáv nem található"         :GOTO 3000
4180 IF FUNX = &H80 THEN PRINT "időtúllépés (Time Out)"     :GOTO 3000
4190 PRINT "nem azonosítható, kódja = ";FUNX;               :GOTO 3000
4200 '
5000 '
5010 '  A VALASZTHATÓ PARANCSOK MEGJELENITÉSE
5020 '  A KÉPERNYÖN
5030 '
5040 '
5050 PRINT
5060 PRINT"╔═══════════════════════════════════════════════════════════╗"
5070 PRINT"║                                                           ║"
5080 PRINT"║              M O N I T O R   P A R A N C S O K            ║"
5090 PRINT"║                                                           ║"
5100 PRINT"║                Q/q = Kilépés (Quit)                       ║"
5110 PRINT"║                ?   = Segítség                             ║"
5120 PRINT"║                O/o = Szektor (O)lvasás                    ║"
5130 PRINT"║                I/i = Szektor (I)nicializálás              ║"
5140 PRINT"║                R/r = Alapállapotba vezérlés - (R)eset     ║"
5150 PRINT"║                F/f = Sáv (F)ormálás                       ║"
5160 PRINT"║                P/p = Paraméter beállítás                  ║"
5170 PRINT"║                                                           ║"
5180 PRINT"╚═══════════════════════════════════════════════════════════╝"
5190 RETURN
5200 PRINT
5210 '
6000 '
6010 '  A KIVALASZTOTT SZEKTOR INICIALIZALASA ÉS
6020 '  FELTOLTÉSE EGY VALASZTOTT KARAKTERREL
6030 '
6040 '
6050 PRINT
6060 INPUT "Oldal : ", OLDALX
6070 INPUT "Sáv : ", SAVX
6080 INPUT "Szektor : ", VX
6090 INPUT "Inicializáló karakter : " ,Z$
6100 PRINT
6110 IF Z$ = "" THEN Z$ = CHR$(254)
6120 FOR IX = 0 TO 511 : POKE VARPTR(VX[0])+IX,ASC(Z$) : NEXT
6130 '
6140 FUNX =  3                                         ' írás funkció.
```

```
6150 MSZ% = &H13                                    ' 13(h) megszakítás.
6160 N%  =  1                                       ' írandó szektorok száma.
6170 SEG% = -1                                      ' BASIC DS ES-re átadva.
6180 OL% = VARPTR(V%[0]) AND 255                    ' Adatvektor ofszetcím
6190 OH% = INT(VARPTR(V%[0]) / 256)                 ' (L)ow és (H)igh bájtja.
6200 CALL RCIM(MSZ%, FUN%, N%, OH%, OL%, SAV%, V%, OLDAL%, EGYSEG%, Q, Q, SEG%, Q)
6210 RETURN
6220 '
7000 '
7010 '  512 BAJTos SZEKTOROK KIALAKíTASA A KIVA-
7020 '  LASZTOTT SAVON
7030 '
7040 '
7050 IF NOT(EZEGYAT%) THEN 7110                     ' Ha a gép egy AT, akkor :
7060 FUN% = &H17                                    ' Lemezform. beáll. funkció.
7070 MSZ% = &H13                                    ' 13(h) megszakítás.
7080 Q   = 0
7090 CALL RCIM(MSZ%, FUN%, LEMEZFORM%, Q, Q, Q, Q, Q, EGYSEG%, Q, Q, Q, Q)
7100 '
7110 PRINT
7120 INPUT "Oldal : ", OLDAL%
7130 INPUT "Sáv : ", SAV%
7140 INPUT "Szektorok száma : ", N%
7150 '
7160 IF N% > 15 THEN N% = 15                        ' Max. 15 szektor / sáv.
7170 '                                              ' Szektor leíró puffer
7180 FOR I% = 0 TO N%-1                             ' inicializálása:
7190 POKE VARPTR(SZP%[0])+I%*4, SAV%                '      sáv,
7200 POKE VARPTR(SZP%[0])+I%*4+1, OLDAL%            '      lemezoldal,
7210 POKE VARPTR(SZP%[0])+I%*4+2, I%+1              '      szektorsorszám,
7220 POKE VARPTR(SZP%[0])+I%*4+3, 2                 '      bájt / szektor.
7230 NEXT
7240 FUN% = 5                                       ' Formálás funkció.
7250 MSZ% = &H13                                    ' 13(h) megszakítás.
7260 SEG% = -1                                      ' BASIC DS ES-re átadva.
7270 OL% = VARPTR(SZP%[0]) AND 255                  ' Szektorpuffer ofszetcím
7280 OH% = INT(VARPTR(SZP%[0]) / 256)               ' (L)ow és (H)igh bájtja.
7290 Q = 0
7300 CALL RCIM(MSZ%, FUN%, N%, OH%, OL%, SAV%, Q, OLDAL%, EGYSEG%, Q, Q, SEG%, Q)
7310 RETURN
7320 '
8000 '
8010 '  A KIVALASZTOTT SZEKTOR BEOLVASASA ÉS MEG-
8020 '  JELENíTÉSE A KÉPERNYÖN
8030 '
8040 '
8050 PRINT
8060 INPUT "Oldal : ", OLDAL%
8070 INPUT "Sáv : ", SAV%
8080 INPUT "Szektor : ", S%
8090 FUN% = 2                                       ' Olvasás funkció.
8100 MSZ% = &H13                                    ' 13(h) megszakítás.
8110 N%  = 1                                        ' Olvasandó szektorok száma.
8120 SEG% = -1                                      ' BASIC DS ES-re átadva.
8130 OL%  = VARPTR(V%[0]) AND 255                   ' Adatvektor ofszetcím
8140 OH%  = INT(VARPTR(V%[0]) / 256)                ' (L)ow és (H)igh bájtja.
8150 Q = 0
8160 CALL RCIM(MSZ%, FUN%, N%, OH%, OL%, SAV%, S%, OLDAL%, EGYSEG%, Q, Q, SEG%, Q)
8170 IF FUN% <> 0 THEN RETURN
8180 '
8190 FOR H% = 0 TO 1
8200 CLS
```

```
8210 LOCATE 4,1
8220 PRINT "Oldal : "; OLDAL%; "Sáv : "; SAV%; "Szektor : "; S%;
8230 IF H% = 0 THEN PRINT " ═══   alsó 256 bájt" :GOTO 8250
8240 PRINT " ═══   felsõ 256 bájt"
8250 PRINT "┌──────────────────────────────────────────────────";
8260 PRINT "──────────────────────────────────────────────────┐"
8270 FOR I% = 0 TO 15
8280  '
8290 PRINT "| ";
8300 IF LEN(HEX$(I%*16)) = 1 THEN PRINT "0";
8310 PRINT HEX$(I%*16);" - ";
8320 IF LEN(HEX$(I%*16+15)) = 1 THEN PRINT "0";
8330 PRINT HEX$(I%*16+15);"  ";
8340 '
8350     FOR J% = 0 TO 15
8360        H$ = HEX$(PEEK(VARPTR(V%[0]) + H%*256 + I%*16 + J%))
8370        PRINT " ";
8380        IF LEN(H$) = 1 THEN PRINT "0";
8390        PRINT H$;
8400     NEXT J%
8410     PRINT "   ";
8420     FOR J% = 0 TO 15
8430        H$ = CHR$(PEEK(VARPTR(V%[0]) + H%*256 + I%*16 + J%))
8440        IF H$<CHR$(32) THEN PRINT "."; ELSE PRINT H$;
8450     NEXT J%
8460 '
8470 PRINT " |";
8480 NEXT I%
8490 PRINT "└──────────────────────────────────────────────────";
8500 PRINT "──────────────────────────────────────────────────┘"
8510 IF H%=1 THEN RETURN
8520 PRINT "Nyomjon meg egy billentyût ..."
8530 '
8540 Z$ = ""
8550 WHILE Z$ = ""                              ' Várakozás egy billentyûre.
8560     Z$ = INKEY$
8570 WEND
8580 '
8590 NEXT H%
8600 '
8800 '
8810 ' ┌──────────────────────────────────────┐
8820 ' │ MONITOR PARAMÉTEREK BEOLVASASA       │
8830 ' └──────────────────────────────────────┘
8840 INPUT "Meghajtó sorszáma (0-3) : ",EGYSEG%
8850 IF NOT(EZEGYAT%) THEN RETURN                'Lemezformátum csak
8860 PRINT                                       'AT-nél választható.
8870 PRINT "Választható formálási paraméterek :"
8880 PRINT "1 = 320/360-KB-os lemez / 320/360-KB-os meghajtó"
8890 PRINT "2 = 320/360-KB-os lemez / 1,2-MB-os meghajtó"
8900 PRINT "3 = 1,2-MB-os lemez / 1,2-MB-os meghajtó"
8910 PRINT
8920 INPUT "Formálási paraméter : ", LEMEZFORM%
8930 RETURN
8940 '
9000 '
9010 ' ┌──────────────────────────────────────┐
9020 ' │ A MEGSZAKÍTÁSHÍVÓ RUTIN INICIALIZÁLÁSA │
9030 ' │    BE: -                             │
9040 ' │    KI: RCIM a Rutin CÍMe             │
9050 ' └──────────────────────────────────────┘
9060 RCIM = 60000!              'A rutin kezdõcíme a BASIC szegmensben.
9070 DEF SEG                    'A BASIC szegmens beállítása.
9080 RESTORE 9140
```

```
9090 FOR I% = 0 TO 160                          'A rutin betoltése a
9100    READ X% : POKE RCIM + I%, X%            'tárba.
9110 NEXT
9120 RETURN
9130 '
9140 DATA    85, 139, 236,  30,   6, 139, 118,  30, 139,   4, 232, 140
9150 DATA     0, 139, 118,  12, 139,  60, 139, 118,   8, 139,   4,  61
9160 DATA   255, 255, 117,   2, 140, 216, 142, 192, 139, 118,  28, 138
9170 DATA    36, 139, 118,  26, 138,   4, 139, 118,  24, 138,  60, 139
9180 DATA   118,  22, 138,  28, 139, 118,  20, 138,  44, 139, 118,  18
9190 DATA   138,  12, 139, 118,  16, 138,  52, 139, 118,  14, 138,  20
9200 DATA   139, 118,  10, 139,  52,  85, 205,   0,  93,  86, 156, 139
9210 DATA   118,  12, 137,  60, 139, 118,  28, 136,  36, 139, 118,  26
9220 DATA   136,   4, 139, 118,  24, 136,  60, 139, 118,  22, 136,  28
9230 DATA   139, 118,  20, 136,  44, 139, 118,  18, 136,  12, 139, 118
9240 DATA    16, 136,  52, 139, 118,  14, 136,  20, 139, 118,   8, 140
9250 DATA   192, 137,   4,  88, 139, 118,   6, 137,   4,  88, 139, 118
9260 DATA    10, 137,   4,   7,  31,  93, 202,  26,   0,  91,  46, 136
9270 DATA    71,  66, 233, 108, 255
```

```
/*
/*
/*                          L E M O N . C
/*
/*   Funkciója : alapvetõ hajlékonylemez monitor funkciók szem-
/*               léltetése egy egyszerû lemezmonitoron keresztul.
/*
/*   FONTOS    : a memóriakezelõ "PEEKB" a PEEKPOKE.ASM forrásál-
/*               lományban helyezkedik el, ezért a LINK parancs
/*               helyes formája a következõ :
/*
/*               LINK LEMON PEEKPOKE
/*
/*
/*

extern short int PEEKB();     /* Az assembler rutin beszerkesztendõ. */

#include <dos.h>
#include <stdio.h>
#include <ctype.h>

#define  FALSE    0
#define  TRUE     1

#define  byte unsigned char

byte  STATUSTXT();

byte egyseg,                          /* A kiválasztott lemezegység.    */
     lemezform,                       /* A meghajtóegység típusa.       */
     EZ_EGY_AT;                       /* AT / egyéb megkülönböztetése.  */

/*
/*   RESET : A HAJLÉKONYLEMEZ VEZÉRLÕ ÉS A
/*           MEGHAJTÓEGYSÉGEK INICIALIZÁLÁSA
/*

byte RESET()

{
union REGS regiszter;                 /* Regiszter union deklaráció.    */

regiszter.h.ah = 0;                           /* RESET funkció,         */
regiszter.h.dl = 0;                           /* hajlékonylemezre.      */
int86(0x13, &regiszter, &regiszter);          /* 13(h) megszakítás.     */
return(regiszter.h.ah);                       /* Vissza : hibakód.      */
}

/*
/*   HL_OLVASAS : OLVASAS HAJLÉKONY LEMEZRÕL
/*

byte HL_OLVASAS(egyseg, oldal, sav, szektor, nszek, adatok)

byte egyseg,                          /* A kiválasztott lemezegység.    */
     oldal,                           /* A kiválasztott lemezoldal.     */
     sav,                             /* A kiválasztott sáv.            */
     szektor,                         /* Az elsõ beolvasandó szektor.   */
     nszek,                           /* Az olvasandó szektorok száma.  */
     *adatok;                         /* Bájtokat tartalmazó adatvektor.*/
```

```
{
union REGS regiszter;                    /* Regiszter union deklaráció. */

regiszter.h.ah = 2;                      /* Olvasás funkció.            */
regiszter.h.al = nszek;                  /* Szektorszám      -> AL.     */
regiszter.h.dh = oldal;                  /* Oldalszám        -> DH.     */
regiszter.h.dl = egyseg;                 /* Egység           -> DL.     */
regiszter.h.ch = sav;                    /* Sávsorszám       -> CH.     */
regiszter.h.cl = szektor;                /* Szektorsorszám -> CL.       */
regiszter.x.bx = (unsigned int) adatok;  /* Adatvektor offset.          */
int86(0x13, &regiszter, &regiszter);     /* 13(h) megszakítás.          */
return(regiszter.h.ah);                  /* Vissza : hibakód.           */
}

/*                                                                      */
/*  HL_IRAS : ÍRÁS A HAJLÉKONY LEMEZRE                                  */
/*                                                                      */

byte HL_IRAS(egyseg, oldal, sav, szektor, nszek, adatok)

byte egyseg,                             /* A kiválasztott lemezegység. */
     oldal,                              /* A kiválasztott lemezoldal.  */
     sav,                                /* A kiválasztott sáv.         */
     szektor,                            /* Az első felulírandó szektor.*/
     nszek,                              /* A felulírandó szektorok száma.*/
     *adatok;                            /* Bájtokat tartalmazó adatvektor.*/

{
union REGS regiszter;                    /* Regiszter union deklaráció. */

regiszter.h.ah = 3;                      /* írás funkció.               */
regiszter.h.al = nszek;                  /* Szektorszám      -> AL.     */
regiszter.h.dh = oldal;                  /* Oldalszám        -> DH.     */
regiszter.h.dl = egyseg;                 /* Egység           -> DL.     */
regiszter.h.ch = sav;                    /* Sávsorszám       -> CH.     */
regiszter.h.cl = szektor;                /* Szektorsorszám -> CL.       */
regiszter.x.bx = (unsigned int) adatok;  /* Adatvektor offset.          */
int86(0x13, &regiszter, &regiszter);     /* 13(h) megszakítás.          */
return(regiszter.h.ah);                  /* Vissza : hibakód.           */
}

/*                                                                      */
/*  HL_FORM : EGY SAV MEGFORMÁLÁSA                                      */
/*                                                                      */

byte HL_FORM(egyseg, oldal, sav, nszek, nbajt)

byte egyseg,                             /* A kiválasztott lemezegység. */
     oldal,                              /* A kiválasztott lemezoldal.  */
     sav,                                /* A kiválasztott sáv.         */
     nszek,                              /* Az formálandó szektorok száma.*/
     nbajt;                              /* Bájtok száma (kódolt) /szektor.*/
                                         /* Kódok : 0 =  128 bájt /szektor,*/
                                         /*         1 =  256 bájt /szektor,*/
                                         /*         2 =  512 bájt /szektor,*/
                                         /*         3 = 1024 bájt /szektor.*/

{
union  REGS regiszter;                   /* Regiszter union deklaráció. */
byte   i;
```

```
struct {                                /* Szektorformátum struktúra.    */
       byte sav;
       byte oldal;
       byte szektor;
       byte hossz;
       }
       format[15];

if (nszek <= 15)
   {
   for (i = 0; i < nszek; i++)                  /* Szektor leíró puffer */
      {                                         /* inicializálása:      */
      format[i].sav     = sav;                  /*      sáv,            */
      format[i].oldal   = oldal;                /*      lemezoldal,     */
      format[i].szektor = i+1;                  /*      szektorsorszám, */
      format[i].hossz   = nbajt;                /*      bájt / szektor. */
      }
   regiszter.h.ah = 5;                          /* Formálás funkció.    */
   regiszter.h.al = nszek;                      /* Szektorszám -> AL.   */
   regiszter.h.dh = oldal;                      /* Oldalszám   -> DH.   */
   regiszter.h.dl = egyseg;                     /* Egység      -> DL.   */
   regiszter.h.ch = sav;                        /* Sávsorszám  -> CH.   */
   regiszter.x.bx = (unsigned int) format;      /* Leíró táblázat címe. */
   int86(0x13, &regiszter, &regiszter);         /* 13(h) megszakítás.   */
   return(regiszter.h.ah);                      /* Vissza : hibakód.    */
   }
else return(0xFF);                              /* Meg nem engedett     */
}                                               /* paraméter.           */

/*  ┌──────────────────────────────────────────┐                          */
/*  │ HELP : A VALASZTHATÓ PARANCSOK MEGJELE-  │                          */
/*  │        NíTÉSE A KÉPERNYÖN                │                          */
/*  └──────────────────────────────────────────┘                          */

void HELP()

{
printf("");
printf("╔═══════════════════════════════════════════════════════════╗");
printf("║                                                           ║");
printf("║                M O N I T O R   P A R A N C S O K          ║");
printf("║                                                           ║");
printf("║                  Q/q : Kilépés (Quit)                     ║");
printf("║                  ?   : Segítség                           ║");
printf("║                  O/o : Szektor (O)lvasás                  ║");
printf("║                  I/i : Szektor (I)nicializálás            ║");
printf("║                  R/r : Alapállapotba vezérlés - (R)eset   ║");
printf("║                  F/f : Sáv (F)ormálás                     ║");
printf("║                  P/p : Paraméter beállítás                ║");
printf("║                                                           ║");
printf("╚═══════════════════════════════════════════════════════════╝");
printf("");
}

/*  ┌──────────────────────────────────────────┐                          */
/*  │ OLVASAS : A KIVALASZTOTT SZEKTOR BEOLVA- │                          */
/*  │           SÁSA ÉS MEGJELENíTÉSE A KÉP-   │                          */
/*  │           ERNYÖN                         │                          */
/*  └──────────────────────────────────────────┘                          */

void OLVASAS()
```

```
{
byte adatok[512];                                   /* Olvasás pufferje. */
int  h,i,j;
byte oldal,                                         /* A kiválasztott oldal.  */
     sav,                                           /* A kiválasztott sáv.    */
     szektor;                                       /* A beolvasandó szektor. */

printf("");
printf("Oldal : ");   scanf("%d", &i);  oldal  = (byte)i;
printf("Sáv : ");     scanf("%d", &i);  sav    = (byte)i;
printf("Szektor : "); scanf("%d", &i);  szektor = (byte)i;

if (STATUSTXT(HL_OLVASAS(egyseg, oldal, sav, szektor, 1, adatok)))
   {
   for (h = 0 ; h < 2; h++)
   {
   printf("");
   printf("Oldal : %d  Sáv : %d  Szektor : %d", oldal, sav, szektor);
   if (h==0) printf(" ═══   alsó 256 bájt");
   else      printf(" ═══   felső 256 bájt");
   printf("┌──────────────────────────────────────────────────┐");
   printf("└──────────────────────────────────────────────────┘");
   for (i = 0; i < 16; i++)
      {
      printf("| %2X - %2X  ", i*16, i*16+15);
      for (j = 0; j < 16; j++)
         if ((int)adatok[i*16+j] < 16)
            printf(" 0%1X", (int)adatok[i*16+j]);
         else
            printf(" %2X", (int)adatok[i*16+j]);
      printf("    ");
      for (j = 0; j < 16; j++)
         printf("%c",(int)adatok[i*16+j] < 32 ? '.' : adatok[i*16+j]);
      printf(" |");
      }
   printf("└──────────────────────────────────────────────────┘");
   printf("┌──────────────────────────────────────────────────┐");
   if (h==0)
      {
      printf("Nyomjon meg egy billentyűt ...");
      getch();
      }
   }
   }
}

/*  ┌──────────────────────────────────────────────┐                  */
/*  │ INIC : A KIVÁLASZTOTT SZEKTOR INICIALIZÁ-    │                  */
/*  │        LÁSA ÉS FELTÖLTÉSE EGY VÁLASZTOTT     │                  */
/*  │        KARAKTERREL                           │                  */
/*  └──────────────────────────────────────────────┘                  */

void INIC()

{
byte adatok[512];                   /* Inicializálás adatvektora.     */
byte oldal,                         /* A kiválasztott oldal.          */
     sav,                           /* A kiválasztott sáv.            */
     szektor;                       /* Az inicializálandó szektor.    */
char inikar;                        /* Az inicializáló karakter.      */
int  i;
```

```
printf("");
printf("Oldal : ");    scanf("%d", &i);  oldal   = (byte)i;
printf("Sáv : ");      scanf("%d", &i);  sav     = (byte)i;
printf("Szektor : "); scanf("%d", &i);  szektor = (byte)i;

printf("Inicializáló karakter : "); scanf("%1c", &inikar);
for (i = 0; i < 512; adatok[i++] = inikar)
   ;
STATUSTXT(HL_IRAS(egyseg, oldal, sav, szektor, 1, adatok));
}

/*  ------------------------------------------------------  */
/*  STATUSTXT : HIBAÜZENET MEGJELENÍTÉSE, HA                */
/*              AZ UTOLSÓ LEMEZMŰVELET NEM                  */
/*              VOLT SIKERES                                */
/*  ------------------------------------------------------  */

byte STATUSTXT(statusz)
byte statusz;                          /* Lemezművelet hibakódja. */

{
if (statusz)
   {
   printf("Hiba lépett fel : ");
   switch (statusz)
      {
      case 0x01 : printf("nem megengedett funkciószám");
                  break;
      case 0x02 : printf("címjelölés nem található");
                  break;
      case 0x03 : printf("írásvédett lemez");
                  break;
      case 0x04 : printf("szektor nem található");
                  break;
      case 0x08 : printf("DMA túlcsordulás");
                  break;
      case 0x09 : printf("adatátvitel szegmenshatáron túl");
                  break;
      case 0x10 : printf("olvasási hiba");
                  break;
      case 0x20 : printf("lemezvezérlő hiba");
                  break;
      case 0x40 : printf("sáv nem található");
                  break;
      case 0x80 : printf("időtúllépés (Time Out)");
                  break;
      case 0xFF : printf("hibás paraméter megadás");
                  break;
      default   : printf("nem azonosítható, kódja = %d ", statusz);
      }
   RESET();                                /* Hiba esetén RESET. */
   }
return(!statusz);
}

/*  ------------------------------------------------------  */
/*  FORMALAS : 512 BAJTos SZEKTOROK KIALAKÍ-                */
/*             TÁSA A KIVÁLASZTOTT SÁVON                    */
/*  ------------------------------------------------------  */

void FORMALAS()
```

```
{
byte oldal,                          /* A kivalasztott lemezoldal.      */
     sav,                            /* A formalando sav.               */
     nszek;                          /* A kialakitando szektorok szama. */
int  i;

printf("");
printf("Oldal : ");          scanf("%d", &i);  oldal = (byte)i;
printf("Sav : ");            scanf("%d", &i);  sav   = (byte)i;
printf("Szektorok száma : "); scanf("%d", &i);  nszek = (byte)i;

if (EZ_EGY_AT)
   {
   union REGS regiszter;             /* Regiszter union deklaráció.     */

   regiszter.h.ah = 0x17;            /* Lemezformátum beáll. funkció.   */
   regiszter.h.al = lemezform;
   regiszter.h.dl = egyseg;
   int86(0x13, &regiszter, &regiszter);        /* 13(h) megszakítás.    */
   }
STATUSTXT(HL_FORM(egyseg, oldal, sav, nszek, 2));
}

/*                                                                       */
/*  PARAM : MONITOR PARAMÉTEREK BEOLVASASA                               */
/*                                                                       */

void PARAM()

{
printf("");
printf("Meghajtó sorszáma (0-3) : ");
scanf ("%d", &egyseg);
if (EZ_EGY_AT)
   {
   printf("");
   printf("Választható formálási paraméterek :");
   printf("1 = 320/360-KB-os lemez / 320/360-KB-os meghajtó");
   printf("2 = 320/360-KB-os lemez / 1,2-MB-os meghajtó");
   printf("3 = 1,2-MB-os lemez / 1,2-MB-os meghajtó");
   printf("Formálási paraméter : ");
   scanf("%d", &lemezform);
   }
}

/*                                                                       */
/*                  F Ő P R O G R A M                                    */
/*                                                                       */

void main()

{
char menupont = '?';                 /* Felhasználói választás.         */

egyseg    = 0;                  /* Alapértelmezés : 0. meghajtó,        */
lemezform = 3;                  /*                  1,2 MB-os egység.   */

EZ_EGY_AT = (PEEKB(0xF000, 0xFFFE) == 0xFC) ? TRUE:FALSE; /* Típus.     */

STATUSTXT(RESET());             /* Elsőként RESET végrehajtása.         */
while (toupper(menupont) != 'Q')
   {
```

```
    switch(menupont = toupper(menupont))
       {
       case 'Q' : break;
       case '?' : HELP();
                  break;
       case 'O' : OLVASAS();
                  break;
       case 'I' : INIC();
                  break;
       case 'R' : STATUSTXT(RESET());
                  break;
       case 'F' : FORMALAS();
                  break;
       case 'P' : PARAM();
                  break;
       otherwise: printf("érvénytelen parancs ...");
       }
    printf("Segitség = ? > ");
    scanf(" %1c", &menupont);
    }
}
```

```
{  ┌──────────────────────────────────────────────────────────────────┐  }
{  │                                                                  │  }
{  │                      L E M O N . P A S                           │  }
{  │                                                                  │  }
{  │  Funkciója : alapvető hajlékonylemez monitor funkciók szem-      │  }
{  │              léltetése egy egyszerű lemezmonitoron keresztül.    │  }
{  │                                                                  │  }
{  └──────────────────────────────────────────────────────────────────┘  }

program LEMON;

type dosreg     = record
                  ax, bx, cx, dx, bp, di, si, ds, es, flags : integer;
                  end;

     szparam    = record
                  sav, oldal, szektor, hossz  : byte;
                  end;

                  { 1 szektor címmező paramétereinek rekordja. }

var hibakod       : byte;
    menupont      : string[1];
    egyseg,
    lemezform     : integer;
    EZ_EGY_AT     : boolean;
    z             : char;        { A várakozási állapotokhoz szükséges. }

{  ┌─────────────────────────────────────────┐  }
{  │ HL_OLVASAS : OLVASAS HAJLÉKONY LEMEZRŐL │  }
{  │         KI : HIBAKÓD                    │  }
{  └─────────────────────────────────────────┘  }

function HL_OLVASAS(egyseg,                 { A kiválasztott lemezegység.       }
                    oldal,                  { A kiválasztott lemezoldal.        }
                    sav,                    { A kiválasztott sáv.               }
                    szektor,                { Az első beolvasandó szektor.      }
                    nszek,                  { Az olvasandó szektorok száma.     }
                    adat_sz,
                    adat_of : integer           { Adatvektor szegmens-  }
                    ) : integer ;               { és ofszetcíme.        }

var regiszter : dosreg;                   { Regiszter rekord deklaráció. }

begin
   regiszter.ax := 2 shl 8 + nszek;             { Olvasás fun.     -> AH  }
                                                { Szektorszám      -> AL. }
   regiszter.dx := oldal shl 8 + egyseg;        { Oldalszám        -> DH. }
                                                { Egység           -> DL. }
   regiszter.cx := sav shl 8 + szektor;         { Sávsorszám       -> CH. }
                                                { Szektorsorszám -> CL.   }
                                                { Adatvektor              }
   regiszter.es := adat_sz;                     {    szegmenscím -> ES.   }
   regiszter.bx := adat_of;                     {    ofszetcím   -> BX.   }
   intr($13, regiszter);                        { 13(h) megszakítás.      }
   HL_OLVASAS  := hi(regiszter.ax);             { Vissza : hibakód, ami   }
end;                                            { AX HI bájtja.           }
```

```
{                                                          }
{  HL_IRAS : ÍRÁS A HAJLÉKONY LEMEZRE                      }
{       KI : HIBAKÓD                                       }
{                                                          }

function HL_IRAS(egyseg,                  { A kiválasztott lemezegység.    }
                 oldal,                   { A kiválasztott lemezoldal.     }
                 sav,                     { A kiválasztott sáv.            }
                 szektor,                 { Az első írandó szektor.        }
                 nszek,                   { Az írandó szektorok száma.     }
                 adat_sz, adat_of         { Adatvektor szegmens-           }
                 : integer) : integer;    { és ofszetcíme.                 }

var regiszter : dosreg;                   { Regiszter rekord deklaráció.   }

begin
   regiszter.ax := 3 shl 8 + nszek;       { Olvasás fun.    -> AH          }
                                          { Szektorszám     -> AL.         }
   regiszter.dx := oldal shl 8 + egyseg;  { Oldalszám       -> DH.         }
                                          { Egység          -> DL.         }
   regiszter.cx := sav shl 8 + szektor;   { Sávsorszám      -> CH.         }
                                          { Szektorsorszám  -> CL.         }
                                          { Adatvektor                     }
   regiszter.es := adat_sz;               {     szegmenscím -> ES.         }
   regiszter.bx := adat_of;               {     ofszetcím   -> BX.         }
   intr($13, regiszter);                  { 13(h) megszakítás.             }
   HL_IRAS      := hi(regiszter.ax);      { Vissza : hibakód, ami          }
end;                                      { AX HI bájtja.                  }

{                                                          }
{  HL_FORM : EGY SÁV MEGFORMÁLÁSA                          }
{                                                          }

function HL_FORM(egyseg,                  { A kiválasztott lemezegység.    }
                 oldal,                   { A formálandó lemezoldal, ill.  }
                 sav,                     { ezen belül a választott sáv.   }
                 nszek,                   { Szektorok száma a sávban.      }
                 hossz : integer          { A szektorok hossza kódolva.    }
                 ) : integer;

var regiszter : dosreg;                   { Regiszter rekord deklaráció.   }

    format    : array [1..15] of szparam; { Címmező paramétertömb.         }
    i         : integer;

begin
   for i := 1 to nszek do                 { Címmező paramétertömb feltöltése. }
      begin
      format[i].sav     := sav;           { Minden sáv címmezője           }
      format[i].oldal   := oldal;         { megegyezik a paramé-           }
      format[i].szektor := i;             { terként átadottal.             }
      format[i].hossz   := hossz;
      end;
   regiszter.ax := 5 shl 8 + nszek;       { Formálás fun.   -> AH          }
                                          { Szektorszám     -> AL.         }
   regiszter.dx := oldal shl 8 + egyseg;  { Oldalszám       -> DH.         }
                                          { Egység          -> DL.         }
   regiszter.cx := sav shl 8;             { Sávsorszám      -> CH.         }
                                          { Adatvektor                     }
   regiszter.es := seg(format[1]);        {     szegmenscím -> ES.         }
   regiszter.bx := ofs(format[1]);        {     ofszetcím   -> BX.         }
   intr($13, regiszter);                  { 13(h) megszakítás.             }
   HL_FORM      := hi(regiszter.ax);      { Vissza : hibakód, ami          }
end;                                      { AX HIGH bájtja.                }
```

```
{
{  UZENET : HIBAUZENET MEGJELENíTÉSE, HA
{           AZ UTOLSÓ LEMEZMOVELET NEM
{           VOLT SIKERES
{
```

```
procedure UZENET(statusz : integer);

begin
if (statusz <> 0) then
   begin
   writeln;
   write('Hiba lépett fel : ');
   case statusz of
      $01 : writeln('nem megengedett funkciószám');
      $02 : writeln('címjelölés nem található');
      $03 : writeln('írásvédett lemez');
      $04 : writeln('szektor nem található');
      $08 : writeln('DMA túlcsordulás');
      $09 : writeln('adatátvitel szegmenshatáron túl');
      $10 : writeln('olvasási hiba');
      $20 : writeln('lemezvezérlő hiba');
      $40 : writeln('sáv nem található');
      $80 : writeln('időtúllépés (Time Out)');
      $FF : writeln('hibás paraméter megadás');
      else  writeln('nem azonosítható, kódja = ', statusz)
   end
   end
end;
```

```
{
{  HELP : A VÁLASZTHATÓ PARANCSOK MEGJELE-
{         NíTÉSE A KÉPERNYŐN
{
```

```
procedure HELP;

begin
   clrscr;
   writeln('
   writeln('
   writeln('
   writeln('
   writeln('
   writeln('
   writeln('
   writeln('
   writeln('
   writeln('
   writeln('
   writeln('
   writeln;
end;
```

```
M O N I T O R   P A R A N C S O K

Q/q = Kilépés (Quit)
?   = Segítség
O/o = Szektor (O)lvasás
I/i = Szektor (I)nicializálás
R/r = Alapállapotba vezérlés - (R)eset
F/f = Sáv (F)ormálás
P/p = (P)araméter beállítás
```

```
function HEXVALUE(decvalue : integer) : char;

begin
if decvalue < 10
   then HEXVALUE := chr(decvalue + 48)
   else
```

```
        case decvalue of
        10 : HEXVALUE := 'A';
        11 : HEXVALUE := 'B';
        12 : HEXVALUE := 'C';
        13 : HEXVALUE := 'D';
        14 : HEXVALUE := 'E';
        15 : HEXVALUE := 'F'
        end;
end;

{ ┌──────────────────────────────────────────────┐ }
{ │ OLVASAS : A KIVÁLASZTOTT SZEKTOR BEOLVA-     │ }
{ │           SÁSA ÉS MEGJELENÍTÉSE A KÉP-       │ }
{ │           ERNYŐN                             │ }
{ └──────────────────────────────────────────────┘ }

procedure OLVASAS;

var adatok : array [1..512] of char; { Puffer a beolvasott adatoknak }
    oldal,                           { A kiválasztott lemezoldal.    }
    sav,                             { A kiválasztott sáv.           }
    szektor,                         { A beolvasandó szektor.        }
    h,i,j : integer;

begin
   writeln;
   write('Oldal : ');
   readln(oldal);
   write('Sáv : ');
   readln(sav);
   write('Szektor : ');
   readln(szektor);
   hibakod := HL_OLVASAS(egyseg, oldal, sav, szektor, 1,
              seg(adatok), ofs(adatok));
   if (hibakod = 0) then                          { Ha sikeres volt az }
      for h := 0 to  1 do                         { olvasás, akkor meg-}
      begin                                       { jelenítés.         }
      clrscr;
      gotoxy(1, 4);
      write('Oldal : ',oldal, '  Sáv : ',sav,'  Szektor : ',szektor);
      if h = 0 then writeln('  ======  alsó 256 bájt')
               else writeln('  ======  felsõ 256 bájt');
      write('┌──────────────────────────────────────────────┐');
      write('└──────────────────────────────────────────────┘');
      for i := 0 to 15 do
         begin
         write('│ ');
         write(HEXVALUE((i*16) mod 16), HEXVALUE((i*16) div 16));
         write(' - ');
         write(HEXVALUE((i*16+15) div 16),HEXVALUE((i*16+15) mod 16));
         write ('  ');
         for j := 1 to 16 do
            begin
            write(' ');
            write(HEXVALUE((integer(adatok[h*256+i*16+j])) div 16));
            write(HEXVALUE((integer(adatok[h*256+i*16+j])) mod 16))
            end;
         write('   ');
         for j := 1 to 16 do
            if adatok[h*256+i*16+j] < ' '
                     then write('.')
                     else write(adatok[h*256+i*16+j]);
```

```
            write(' |');
            end;
         write;
         write('└──────────────────────────────────────────┘');
         write('──────────────────────────────────────────');
         if h = 0 then
            begin
            write('Nyomjon meg egy billentyűt ...');
                read(kbd, z)                    { Várakozás egy billentyűre. }
            end
         end
      else UZENET(hibakod);
end;

{ ┌──────────────────────────────────────────────┐ }
{ │ INIC : A KIVÁLASZTOTT SZEKTOR INICIALIZÁ-    │ }
{ │        LÁSA ÉS FELTÖLTÉSE EGY VÁLASZTOTT     │ }
{ │        KARAKTERREL                           │ }
{ └──────────────────────────────────────────────┘ }

procedure INIC;

var adatok : array [1..512] of char;    { Inicializálás adatvektora.    }
    oldal,                              { A kiválasztott lemezoldal.    }
    sav,                                { A kiválasztott sáv.           }
    szektor : integer;                  { Az inicializálandó szektor.   }
    inikar  : char;                     { Az inicializáló karakter.     }
    i : integer;

begin
   writeln;
   write('Oldal : ');
   readln(oldal);
   write('Sáv : ');
   readln(sav);
   write('Szektor : ');
   readln(szektor);
   write('Inicializáló karakter : ');
   readln(inikar);
   for i := 1 to 512 do                 { Adatvektor feltöltése. }
      adatok[i] := inikar;
   UZENET(HL_IRAS(egyseg, oldal, sav, szektor, 1,
                  seg(adatok), ofs(adatok)));
end;

{ ┌──────────────────────────────────────────────┐ }
{ │ RESET : A HAJLÉKONYLEMEZ VEZÉRLŐ ÉS A        │ }
{ │         MEGHAJTÓEGYSÉGEK INICIALIZÁLÁSA      │ }
{ │    KI : HIBAKÓD                              │ }
{ └──────────────────────────────────────────────┘ }

function RESSET : integer;

var regiszter : dosreg;                 { Regiszter rekord deklaráció. }

begin
   regiszter.ax := 0;                   { RESET funkció,               }
   intr($13, regiszter);                { hajlékonylemezre.            }
   RESSET := hi(regiszter.ax);          { 13(h) megszakítás.           }
end;                                    { Vissza : hibakód.            }
```

```
{                                                                    }
{   FORMALAS : 512 BAJTos SZEKTOROK KIALAKÍ-                         }
{              TASA A KIVALASZTOTT SAVON                             }
{                                                                    }

procedure FORMALAS;

var regiszter : dosreg;                  { Regiszter rekord deklaráció.    }
    oldal,                               { A kiválasztott lemezoldal.      }
    sav,                                 { A formálandó sáv.               }
    nszek : integer;                     { A kialakítandó szektorok száma. }

begin
   writeln;
   write('Oldal : ');
   readln(oldal);
   write('Sáv : ');
   readln(sav);
   write('Szektorok száma : ');
   readln(nszek);
   if EZ_EGY_AT then
      begin
      regiszter.ax := $17 shl 8 + lemezform;     { Lemezformátum be-     }
      regiszter.dx := egyseg;                    { állítás funkció.      }
      intr($13, regiszter)                       { 13(h) megszakítás.    }
      end;
   UZENET(HL_FORM(egyseg, oldal, sav, nszek, 2));
end;

{                                                                    }
{   PARAM : MONITOR PARAMÉTEREK BEOLVASASA                           }
{                                                                    }

procedure PARAM;

begin
   writeln;
   write('Meghajtó sorszáma (0-3) : ');
   readln(egyseg);
   if EZ_EGY_AT then                         { Formálási paraméter csak }
      begin                                  { AT esetén választható.   }
      writeln;
      writeln('Választható formálási paraméterek :');
      writeln;
      writeln('1 = 320/360-KB-os lemez / 320/360-KB-os meghajtó');
      writeln('2 = 320/360-KB-os lemez / 1,2-MB-os meghajtó');
      writeln('3 = 1,2-MB-os lemez / 1,2-MB-os meghajtó');
      writeln;
      write('Formálási paraméter : ');
      readln(lemezform)
      end;
end;

{                                                                    }
{                       F Ő P R O G R A M                            }
{                                                                    }

begin
egyseg    := 0;                  { Alapértelmezés : 0. meghajtó,     }
lemezform := 3;                  {                  1,2 MB-os egység. }
```

```
clrscr;
if mem[$F000:$FFFE] = $FC then EZ_EGY_AT := true                { AT-e }
                          else EZ_EGY_AT := false;              { gép ? }

UZENET(RESSET);                          { Elsőként alapállapotba vezérlés. }
menupont := '?';
repeat
   case menupont of
      '?' : HELP;
      'O' : OLVASAS;
      'I' : INIC;
      'R' : UZENET(RESSET);
      'F' : FORMALAS;
      'P' : PARAM;
      else writeln('érvénytelen parancs ...');
   end;
   writeln;
   write('Segítség = ? > ');
   readln(menupont);
   if menupont = '' then menupont := ' ';
   menupont := upcase(menupont);
until (menupont = 'Q');
end.
```

1.5. A merevlemezes meghajtó vezérlése

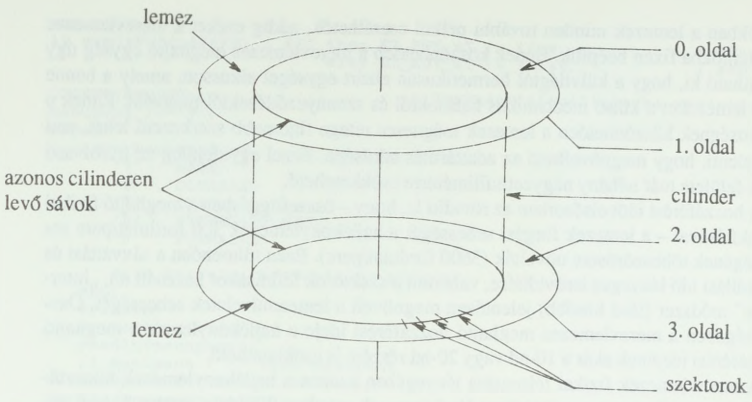
1983 márciusában, az IBM PC/XT megjelenésével új szakasz kezdődött a személyi számítógépek történetében: ez volt az az első mikroszámítógép, amelybe háttértárként hajlékonylemezes meghajtó mellett merevlemezes meghajtót is építettek. Érdekességként megjegyezzük, hogy a fejlesztés idején az IBM ezt az egységet a nálunk is általánosan ismert Winchester-tárnak nevezte. (Az elnevezést illetően nem állnak rendelkezésünkre megbízható dokumentumok. Azt olvastuk, hogy az IBM, amely az eredeti verziót kifejlesztette, a fejlesztési tervet a 3030-as kódszámmal látta el, és ez a szám történetesen megegyezett az egykor oly híres vadnyugati Winchester-puska 30–30-as típusszámával... Mindenesetre a merevlemezes meghajtók neve a jelenlegi angol szóhasználat szerint hard disk drive, ill. fixed disk driver – az utóbbit az IBM használja). A kétféle meghajtó között számos hasonlóság és különbség van: a különbség elsősorban a tárolási kapacitásban és a hozzáférési időben jelentős. Míg egy hajlékonylemezes meghajtó kapacitása általában 360 kbájt (nem megfeledkezve az AT-kben használható 1.2 Mbájtos lemezekről), addig már az első XT-kbe beépített merevlemezes meghajtók is minimum 10 Mbájt tárolására voltak képesek (ami majdnem 30 hajlékonylemez összkapacitásának felel meg!). Manapság a merevlemezek kapacitása már a 80 vagy akár a 160 Mbájtot is eléri. A kapacitás ilyen mértékű növelését számos, elsősorban fizikai jellemző megváltoztatása teszi lehetővé. Ezek közül ez egyik legkézenfekvőbb az, hogy a merevlemezes meghajtókban a meghajtó tengelyre nem egy, hanem több lemezt szerelnek fel, amelyek egymással – a tengely által létrehozott mechanikus kapcsolatnak megfelelően – fizikailag együtt forognak. Egy másik lényeges változtatás, hogy míg a hajlékonylemezes meg-

hajtókban a lemezek minden további nélkül cserélhetők, addig ezeket a merevlemezes meghajtókba fixen beépítik. Ennek köszönhetően a merevlemezes meghajtó egység úgy alakítható ki, hogy a külvilágtól hermetikusan elzárt egységet alkosson, amely a benne lévő lemezeket a külső mechanikus hatásoktól és szennyeződésektől megvédi. Ennek a védettségnek köszönhetően a lemezek mágneses rétege finomabb szerkezetű lehet, ami azt jelenti, hogy megnövelhető az adattárolás sűrűsége. Ezzel egyidejűleg az író/olvasó fejek felülete már néhány négyzetmilliméterre csökkenthető.

A hozzáférési időt elsősorban az rövidíti le, hogy – összefüggésben a meghajtó fizikai kialakításával – a lemezek forgási sebességét a hajlékonylemezek 300 fordulat/perc sebességének többszörösére növelték (3600 fordulat/perc). Ezen túlmenően a sávváltási és fejbeállási idő lényeges lerövidítése, valamint a szektorok felírásakor használt ún. „interleave" módszer (lásd később) jelentősen megnöveli a lemezműveletek sebességét. Összességében a merevlemezes meghajtó hozzáférési ideje a hajlékonylemezes meghajtó hozzáférési idejének akár a 10-ed vagy 20-ad részére is csökkenthető.

A merevlemezek fizikai felosztása lényegében azonos a hajlékonylemezek felosztásával: a merevlemezes meghajtóban lévő lemezek mindegyik oldala azonos számú sávokra, a sávok pedig ugyancsak azonos számú szektorokra vannak osztva. Az egymás fölött elhelyezkedő sávokat a merevlemezeknél cilindereknek (magyarul henger) nevezik. (Ha az egymás fölött lévő sávok egy-egy pontját gondolatban egy, a forgástengelylyel párhuzamos egyenessel összekötjük, akkor a sávok mentén így kialakuló egyenessereg egy hengerfelületet alkot. Így az egymás fölött elhelyezkedő sávok egyetlen közös hengert, azaz cilindert alkotnak. Egy cilinderen tehát annyi sáv van, ahány írható/olvasható lemezoldalt tartalmaz a meghajtó). A cilinderek számozása a lemezek külső kerületétől, a 0. sorszámmal kezdődik, és befelé halad. A cilinderek szokásos száma 10 Mbájtos (2 lemezes, azaz 4 oldalas) meghajtónál 306 (0 – 305), 20 Mbájtos (ugyancsak 4 oldalas) meghajtónál pedig 615 (0 – 614). Az egyes cilinderek lemezoldalanként 17 szektort tartalmaznak (fizikailag 1-től 17-ig számozva), a szektorok hossza pedig 512 bájt. A lemezoldalak (író/olvasó fejek) számozása 0-tól kezdődik.

A fentiek alapján egy szektort (fizikaliag) egy számhármassal lehet azonosítani, amelyben a számok a a cilinder, a lemezoldal és a szektor sorszámát adják meg. Megjegyezzük, hogy a szakirodalom egy része a cilindert és a sávot azonos értelemben használja. Logikusabbnak tűnik azonban az a meggondolás, amely szerint a cilindereket az előbb vázoltaknak megfelelően a lemezek forgástengelyére nézve koncentrikus hengerfelületekként képzeljük el. E felületek mindegyike annyi sávot tartalmaz, ahány lemezoldal (fej) a meghajtóban van. Eszerint – hogy a hajlékony- és a merevlemezek felosztása között ebben a tekintetben is egyrészt valamilyen analógiát találjunk, másrészt viszont a fogalmak használatát illetően következetesek is legyünk – mi különbséget teszünk a ci-linder és a sáv között: a cilinder az egymás fölött elhelyezkedő, azonos sorszámú sávokból áll, a sávnak ugyanaz a jelentése, mint amit a hajlékonylemezeknél megismertünk, és egy cilinder annyi sávból áll, ahány lemezoldal (fej) van a meghajtóban. Egy sáv azonosításához tehát a cilinderen kívül még a lemezoldal (fej) számát is meg kell adni. Az elmondottakat a 9. ábrán szemléltetjük.



9. ábra: Merevlemezek fizikai felosztása

A szektorok logikai számozása úgy történik, amint azt a hajlékonylemezeknél már láttuk, megjegyezve, hogy a sávfolytonosság itt természetesen úgy értendő, hogy az azonos cilinderhez tartozó sávok folytonosan követik egymást, és a sorszámozás csak az illető cilinder utolsó sávja után tér át a következő cilinderre. Ugyanakkor a merevlemezeknél a szektorok logikai sorszámozása – a hajlékonylemezektől eltérően – általában nem követi a fizikai elhelyekedésüket. Ennek magyarázatára a későbbiekben még visz - szatérünk.

Még egy, a hajlékonylemezeknél nem használható felosztásról kell szót ejteni. Ezt a felosztást a szakirodalom partícionálásnak nevezi, és azt jelenti, hogy egy merevlemez több, olyan önálló részre (partícióra) osztható, amelyek mindegyike különböző operációs rendszerekhez rendelhető hozzá. Így lehetőség van arra, hogy a merevlemez egyik része a DOS, míg egy másik része pl. a UNIX operációs rendszer alatt futó programokat, ill. magukat az operációs rendszereket tartalmazza. A merevlemez partícionálása a DOS FDISK nevű programjával végezhető el. A partíciókra vonatkozó legfontosabb jellemzőket (az egyes partíciók hossza, típusa, a kezdő és utolsó cilinder száma stb.) a merevlemez legelső sávján (0. oldal, 0. sáv) lévő 17 szektor tartalmazza. Ezt az adatterületet a merevlemez „master boot sector"-ának, más kifejezéssel a lemez partíciótáblájának nevezik.

Mielőtt rátérnénk a merevlemezes meghajtót vezérlő BIOS funkciók ismertetésére, felhívjuk a figyelmet arra, hogy e funkciók használata fokozott elővigyázatosságot igényel. Míg hajlékonylemez esetén lehetőségünk van arra, hogy egy üres, vagy nem különösebben fontos, ill. könnyen pótolható adatokat tartalmazó lemezen kísérletezzünk, ez merevlemez esetén általában nem lehetséges. Ha a merevlemezen akarunk valamilyen műveletet a BIOS-ból végrehajtani, akkor elegendő csak egy csöppnyi figyelmetlenség ahhoz, hogy olvasás helyett esetleg írjunk, nem is beszélve a lemez formálásáról! A BIOS a funkció végrehajtásakor nem kérdez vissza, hogy vajon tényleg azt a műveletet

akarjuk elvégezni, amit – esetleg hibásan – megadtunk, hanem szó nélkül és könyörtelenül végrehajtja. Merevlemezen az így bekövetkező kár jóvátehetetlen lehet.

Ezek után lássuk a rendelkezésünkre álló funkciókat, először csak felsorolva ezeket:

0	–	00(h)	reset (csak XT és AT)
1	–	01(h)	státusz olvasása (csak XT és AT)
2	–	02(h)	olvasás (csak XT és AT)
3	–	03(h)	írás (csak XT és AT)
4	–	04(h)	verifikálás (csak XT és AT)
5	–	05(h)	formálás (csak XT és AT)
8	–	08(h)	formátum lekérdezése (csak XT és AT)
9	–	09(h)	külső meghajtók illesztése (csak XT és AT)
10	–	0A(h)	bővített olvasás (csak XT és AT)
11	–	0B(h)	bővített írás (csak XT és AT)
13	–	0D(h)	reset (csak XT és AT)
16	–	10(h)	lemezmeghajtó üzemkész? (csak XT és AT)
17	–	11(h)	lemezmeghajtó újraállítása (csak XT és AT)
20	–	14(h)	meghajtóvezérlő vizsgálata (csak XT és AT)
21	–	15(h)	meghajtó típusának megállapítása (csak AT)

A funkciókhoz a BIOS 13(h) megszakításán keresztül lehet hozzáférni. Amint korábban már láttuk, ugyanezen a megszakításon keresztül vezérelhetők a hajlékonylemezes meghajtók is. Bármennyi közös tulajdonsága is van azonban a hajlékony- és merevlemezes meghajtóknak, a közöttük lévő különbségek miatt természetesen alapvetően más rutinokat igényel a kétféle típusú meghajtó vezérlése. A közös, 13(h) megszakítás használatát az teszi lehetővé, hogy a megszakítás mögött két, egymástól teljesen eltérő rutinokból álló modul található: az egyik a hajlékony-, a másik a merevlemezes meghajtók vezérlését végzi.

Mint tudjuk, az első PC-k csak hajlékonylemezes meghajtó(ka)t tartalmaztak, amelyeket a ROM-BIOS 13(h) megszakításán keresztül lehetett vezérelni. A merevlemezes egységek megjelenésekor az ehhez tartozó BIOS programot módosították. Amikor a gép bekapcsolásakor elinduló ROM rutinok észlelik a merevlemezes meghajtó meglétét, a 13(h) megszakítási vektor tartalmát úgy írják át, hogy az most azokra a BIOS rutinokra mutasson, amelyek a merevlemezes meghajtó vezérlő kártyáján lévő ROM-elemekben vannak, és mind a hajlékony-, mind a merevlemez vezérlésre képes modulokat tartalmazzák. Az eredeti lemezvezérlő rutinok kezdőcíme ekkor a 13(h) vektorból átkerül a 40(h) vektorba. Ezt követően a BIOS a 13(h) megszakítás meghívásakor már a merevlemezes meghajtó vezérlő kártyáján lévő rutinokat aktivizálja. Azt, hogy a kért funkció a merevlemezes vagy a hajlékonylemezes meghajtó(k)ra vonatkozik, a DL regiszterben lévő érték dönti el. Ha DL ≥ 80(h), akkor a BIOS a merevlemezes meghajtót vezérlő rutinokat aktivizálja, ellenkező esetben pedig meghívja a 40(h) megszakítást, amely most az eredeti hajlékonylemezes meghajtót vezérlő rutinokra mutat.

A merevlemezes meghajtót kezelő funkciók egyik közös jellemzője, hogy a megszakításból való visszatérésükkor a művelet sikeres végrehajtását az átvitelbit értékével jelzik. Sikeres művelet esetén a bit értéke 0, ellenkező esetben 1, és az AH regiszter a hibakódot tartalmazza.

Az egyes hibakódok jelentése:

01(h) :	nem megengedett funkciószám, vagy nem létező meghajtó
02(h) :	címjelölést nem találta
04(h) :	szektort nem találta
05(h) :	vezérlő reset hiba
07(h) :	hiba a vezérlő inicializálásában
09(h) :	DMA átviteli hiba: szegmenshatár átlépés
0A(h) :	hiba a szektorban
10(h) :	olvasási hiba
11(h) :	olvasási hiba ECC-vel kijavítva
20(h) :	lemezvezérlő hiba
40(h) :	sikertelen keresés
80(h) :	időtúllépés, lemezmeghajtó nem válaszolt (Time Out)
AA(h) :	meghajtó nem üzemkész
CC(h) :	írási hiba

Itt is érvényes az, amit a hajlékonylemezes meghajtó vezérlésével foglalkozó fejezetben a hibák előfordulásával kapcsolatban elmondtunk: Ha a fenti hibák valamelyike (az 1-es kivételével) fellép, akkor ez még nem jelenti feltétlenül azt, hogy a lemezvezérlő, a lemezmeghajtó vagy a lemez hibás. Hiba esetén a lemezmeghajtót a 0. funkcióval alapállapotba kell hozni (reset), és a műveletet meg kell ismételni.

Ha egy olvasási művelet a 11(h) hibakóddal tér vissza, akkor ez azt jelenti, hogy a funkció olyan olvasási hibát talált, amit az ún. ECC (Error Correcting Code) hibajavító algoritmussal ki tudott javítani. Az algoritmus a szektor bájtjaiból matematikai számításokkal egy négybájtos értéket képez, amelyet minden egyes szektor után felír a lemezre. Olvasási hiba esetén a lemezen tárolt ECC értékek alapján a hiba bizonyos határok között javítható.

A funkciók a paraméterek átadása és a regiszterek kiosztása tekintetében is számos közös vonást mutatnak. A funkció számát minden esetben az AH regiszterben, a meghajtó azonosító számát pedig a DL regiszterben kell átadni. Itt a 80(h) az első, a 81(h) pedig – ha van – a második meghajtót jelenti. A lemezoldal sorszámát (ami itt logikailag az író/olvasó fej sorszámának felel meg) a DH, a cilinder sorszámát pedig a CH regiszterbe kell beírni. Mivel azonban a 8 bites CH regiszterben maximum 256 sorszám tárolható, és az XT-nek már a legkisebb kapacitású, 10 Mbájtos merevlemeze is 306 cilindert tartalmaz, látható, hogy ennek befogadására a CH nem elegendő.

A rendszer tervezői azt a megoldást választották, hogy a cilinderek sorszámának átadásához a CH regiszter hosszát a CL regiszter – egyébként kihasználatlan – két legfelső, 6. és 7. bitjével megnövelték. Az így létrejött 8 + 2 = 10 biten most már 1024 cilindersorszám helyezhető el. A CL regiszter a fennmaradó alsó 6 bitjén (0 – 5. bit) annak a szektornak a (fizikai) sorszámát (1 – 17) tartalmazza, amelyen az illető műveletet el akarjuk végezni. (Sőt, mivel egy sáv összesen maximum 17 szektort tartalmazhat, ebből is csak az első 5 bit van kihasználva.) Itt is van lehetőségünk arra, hogy egy műveletet egyidejűleg több – amint majd látni fogjuk, maximum 128 logikailag folytonos sorszámozású – szektoron hajtsunk végre. Ebben az esetben a CL alsó 6 (pontosabban 5) bitjén

megadott sorszám a kezdő szektor sorszámát adja meg, míg a szektorok logikailag folytonosan sorszámozott „darabszámát" az AL regiszterben kell megadnunk. Az író és olvasó műveleteknél természetesen azt a pufferterületet is meg kell adnunk, amelyre a funkció írhat, ill. amelyről olvashat. A puffer szegmens- és ofszetcímét az ES:BX regiszterpárban kell megadnunk.

Az imént említett paraméterek megadása alól kivételt képez a 0, valamint a 13 – 0D(h) sorszámú funkció. Mindkét funkció arra szolgál, hogy segítségükkel – például egy hiba fellépése után – alapállapotba állítsuk a meghajtót. A két funkció között az a különbség, hogy míg a 0. sorszámú funkció a DL regiszter értékétől függően vagy csak a hajlékonylemezes meghajtót (DL 80(h)), vagy mind a hajlékony-, mind a merevlemezes meghajtót (DL 80(h)) alapállapotba állítja, addig a 13 – 0D(h) funkció csak a merevlemezes meghajtót viszi alapállapotba anélkül, hogy a hajlékonylemezes meghajtó állapotát megváltoztatná. E funkciók végrehajtásához csak a funkció kódját és a meghajtó azonosítóját kell megadni.

Az 1-es sorszámú funkcióval az utoljára végrehajtott lemezművelet állapota kérdezhető le anélkül, hogy ehhez egy külön műveletet végre kellene hajtani. A meghajtó azonosító számát most is a DL regiszterben kell átadni.

A 2-es sorszámú funkció egy (vagy több) szektor tartalmának beolvasására szolgál. Amint erről már volt szó, a funkció egyszeri meghívásával egyidejűleg maximum 128, logikailag folytonos sorszámozású szektor tartalma olvasható be. Ennek a megszorításnak az az oka, hogy a szektorokon tárolt bájtok az adatátvitel meggyorsítása céljából „kikerülik" a mikroprocesszort, és a DMA (Direct Memory Access) vezérlő integrált áramkör segítségével a lemezről közvetlenül a számítógép operatív tárába kerülnek. Mivel egy DMA áramkör „egyszuszra" csak 64 kbájtnyi adat átvitelére képes, és az így megcímzett terület is csak szegmenshatárokon kezdődhet, lényeges, hogy az ES:BX-ben megadott puffer hosszúsága beleférjen az ES-sel megadott kezdőcímű 64 kbájt hosszúságú szegmensbe. Ellenkező esetben a DMA áramkör hibát jelez.

Ha egyidejűleg több szektort akarunk beolvasni, akkor a beolvasandó szektorok számát az AL regiszterben kell megadni. Emlékezzünk arra, hogy a beolvasás – csakúgy, mint más, több szektort érintő műveletek – itt is sávfolytonosan történik: az olvasás a cilinder-, lemezoldal- és szektorsorszámmal megadott szektoron kezdődik, majd miután elérte a szektor végét (17. szektor), az ugyanazon cilindersorszámhoz tartozó következő lemezoldal (fej) első szektorán folytatódik. Miután így sorra végighaladt az adott cilinderhez tartozó szektorokon és lemezoldalakon, rátér a következő cilinderre, ahol a műveletet az első (0. sorszámú) lemezoldal 1. sorszámú szektorán folytatja, és így tovább.

A 3. sorszámú, írási funkcióra értelemszerűen ugyanaz vonatkozik, mint a 2. sorszámú, olvasási funkcióra.

A 4. sorszámú funkcióval egy vagy több szektor ellenőrizhető (verifikálható). A funkció nem a merevlemezre felírt és a tárban lévő adatokat hasonlítja össze (ezért puffert nem is kell kijelölni), hanem a szektor adatbájtjaiból az ECC-algoritmus szerint (lásd előbb) 4 bájtot képez, és ezeket hasonlítja össze a szektor(ok) formálásával vagy a szektorra írással egyidejűleg lemezre írt ECC-bájtok értékével. Az ellenőrizendő szektorok számát itt is az AL regiszterben kell megadni.

Az 5. sorszámú funkcióval a merevlemez egy teljes sávja formálható. A sáv kijelöléséhez természetesen meg kell adni a cilinder és a lemezoldal számát, továbbá az ES:BX regiszterpárban egy puffer címét. A puffer hosszát 512 bájtra kell meghatározni attól függetlenül, hogy a funkció ebből csak az első 34 bájtot használja. Ez a 34 bájt a sávon létrehozandó 17 szektor mindegyikére vonatkozóan 2–2 bájtot tartalmaz. Ezekből az első bájt azt adja meg, hogy az illető szektort jó vagy hibás szektorként akarjuk-e formálni (00(h) = jó, 80(h) = hibás szektor), míg a második azt írja elő, hogy a szektorhoz a sávon belül melyik (logikai) számot rendeljük. A funkció az így előkészített puffer első két bájtját a formálandó sáv első fizikai szektorához, a puffer második két bájtját a sáv második szektorához rendeli, és így tovább. Mivel a szektorokat nyilván jó szektorokként akarjuk formálni, a puffer páros sorszámú bájtjaiba (0., 2., 4., ..., 32.) 00-t írunk. Logikusnak tűnik, hogy páratlan sorszámú bájtokba írt (logikai) szektorszámok egyezzenek meg a hozzájuk tartozó fizikai szektorok sorszámával, vagyis az értékük rendre 1, 2, 3,...,17 legyen. A gyakorlatban azonban nem így történik a számozás. Ennek az az oka, hogy a merevlemezes meghajtó 3600 fordulat/percre megnövelt forgási sebessége olyan mértékben megnövelte az adatátviteli sebességet, hogy ezzel a meghajtót vezérlő áramkörök már nem tudnak lépést tartani.

Tegyük fel, hogy egy funkcióval két, logikailag egymást követő szektort akarunk beolvasni, és a szektorok logikai és fizikai sorszáma megegyezik. A funkció meghívásakor az író/olvasófej rááll a kért lemezoldal kért sávjára, és várja, hogy a sáv első szektora „befordúljon". Amikor a szektor befordul, a fej beolvassa, de még mielőtt be tudná fejezni az adatok továbbítását, már megérkezett a második szektor. Ezt most nem tudja beolvasni, mert még ez előzővel van elfoglalva. Meg kell várnia tehát, hogy a lemez körbefordúljon, és csak most tudja a második szektort is beolvasni. Tiszta időpocsékolás. Ha tudjuk azt, hogy a lemezmeghajtó vezérlője egy szektor beolvasását követően pl. öt „szektornyi" idő alatt képes a szektor adatainak továbbítására, akkor a logikailag második szektort a logikailag előző szektorhoz képest fizikailag öt szektornyi távolsággal eltoljuk. Ebben az esetben a második szektor éppen akkor érkezik a fej alá, amikor a vezérlő az előzőleg beolvasott szektor adatainak továbbításával végzett, és készen áll a következő szektor beolvasására. Ezt a szektorok fizikai és logikai számozása közötti eltolást angolul interleave-nek nevezik. Az eltolás mértékére az interleave-tényező elnevezést használjuk.

Az előző okfejtésből kitűnik, hogy minél gyorsabb az adatátvitel, annál kevesebb szektort kell „átugrani". Míg az XT merevlemezes meghajtói még 1 : 6 interleave-tényezővel dolgoznak, ez az arány az AT-knél már 1 : 3, illetve 1 : 2 mértékre csökkent. A 10. ábrán bemutatjuk egy sáv fizikai és logikai számozású szektorainak kapcsolatát 1 : 6 és 1 : 2 interleave-tényező mellett, valamint a formáláskor átadandó 512 bájtos puffer első 34 bájtjának tartalmát (interleave-tényező 1 : 2). Egy sáv 17 szektort tartalmaz.

Szektorok számozása egy sávon belül, különböző interleave-tényezők mellett

fizikai		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
logikai	1 : 6	1	4	7	10	13	16	2	5	8	11	14	17	3	6	9	12	15
	1 : 2	1	10	2	11	3	12	4	13	5	14	6	15	7	16	8	17	9

Formáláskor átadandó puffer első 34 bájtjának tartalma (interleave-tényező 1 : 2)

```
db    00h, 01h, 00h, 0Ah, 00h, 02h, 00h, 08h, 00h, 03h, 00h, 0Ch
db    00h, 04h, 00h, 0Dh, 00h, 05h, 00h, 0Eh, 00h, 06h, 00h, 0Fh
db    00h, 07h, 00h, 10h, 00h, 08h, 00h, 11h, 00h, 09h
```

10. ábra: Szektorok számozása és a puffer tartalma

Térjünk vissza a formálás funkcióhoz. A funkció a formálás során a pufferben lévő kétbájtos szektorazonosítók első bájtjait attól függően írja felül, hogy az éppen formált szektort megfelelőnek tartja-e adattárolásra. Ha a szektor megfelelő, akkor a bájt értéke marad 00, ellenkező esetben pedig 80(h)-t ír bele. (Lektor megjegyzése: Az IBM PC/XT gépeknél lehetőség van a merevlemez formálására pufferben átadott azonosítók nélkül is. Ebben az esetben az AL regiszterben kell megadni az interleave-tényező értékét.)

A 8. sorszámú funkcióval a lemezmeghajtó legfontosabb paraméterei kérdezhetők le. A funkció előkészítéséhez csak a meghajtó azonosító számát kell beírni a DL regiszterbe. A megszakítási rutin visszatérésekor a következő információkat szolgáltatja: a DL regiszterben a rendszerhez (fizikailag is !) csatlakoztatott merevlemezes meghajtók számát tartalmazza (azaz a funkció a lemez esetleges partícionálását nem veszi figyelembe). Ennek megfelelően a visszaadott szám 0 (vagyis nincs merevlemez a rendszerben), 1 vagy 2 lehet. A DH regiszterben a lemezoldalak (fejek) száma, míg a CH regiszter 8 bitjéből és a CL regiszter felső 2 bitjéből képzett 10 biten a cilinderek száma található. A lemezoldalak és a cilinderek számozása is 0-tól kezdődik. Végül a CH regiszter alsó 6 bitje (0 – 5. bitek) az egy sávon elhelyezhető maximális szektorszámot adja meg (a szektorok sorszámozása 1-től kezdődik!).

Megjegyezzük, hogy a 8-as funkcióval a PC/AT gépek nagykapacitású hajlékonylemezes meghajtóinak a paraméterei is lekérdezhetők. Ekkor a DL regiszterbe értelemszerűen ennek a meghajtónak az azonosítóját (0 vagy 1) kell beírni. A funkció visszatérésekor az előbb leírt paramétereken túl a BL regiszterben még egy kódszámot kapunk vissza, és az ES:DI regiszterpár az illető lemezmeghajtó paramétertáblájának címét tartalmazza. A BL-ben visszakapott kód jelentése:

```
01(h)  :   360 kbájtos, 40 sávos, 5.25"-os lemezegység
02(h)  :   1.2 Mbájtos, 80 sávos, 5.25"-os lemezegység
03(h)  :   720 kbájtos, 80 sávos, 3.5"-os lemezegység
04(h)  :   1.44 Mbájtos, 80 sávos, 3.5"-os lemezegység
```

A 9. sorszámú funkció lehetővé teszi, hogy az általánosan használt merevlemezes meghajtóktól eltérő paraméterű lemezmeghajtókat is a rendszerbe illesszünk. A BIOS a merevlemezes meghajtókra vonatkozóan is tartalmaz egy sor paramétertáblát a ROM-ban. A BIOS betöltésekor a rendszer konfigurációjának megfelelően ezek egyikét (vagy két merevlemezes meghajtó esetén kettőt) bemásolja a RAM-ba, és a megszakítási vektortáblában lévő 41(h) vektort (ill. két meghajtó esetén a 46(h) vektort is) úgy állítja be, hogy a vektorok ezekre a RAM-ba másolt paramétertáblákra mutassanak. Ha olyan meghajtót kell a rendszerbe illesztenünk, amelynek paraméterei eltérnek a BIOS-ban megadott paraméterektől, akkor annyi a teendőnk, hogy az illető meghajtó számára vala-

hol a RAM-ban létrehozunk egy adatterületet, ebbe beírjuk a szükséges paramétereket, és a 41(h), ill. a 46(h) vektorokat az új paramétertábla címére állítjuk.

Itt is érvényes a hajlékonylemezes meghajtóknál tett megjegyzésünk. Eszerint ha a számítógép már eleve beépítve tartalmazza a merevlemezes meghajtót, akkor abból indulhatunk ki, hogy a gép gyártója a meghajtó és a BIOS között már biztosítja az optimális összhangot, ezért a paramétertáblák paramétereinek megváltoztatása nem ajánlott.

A funkció meghívásához az AH regiszterbe a funkció számát, a DL regiszterbe pedig a meghajtó azonosítóját (első meghajtó = 80(h), második meghajtó 81(h)) kell beírni. A 41(h) vektornak az első, a 46(h) vektornak pedig a (ha van) második meghajtó paramétertáblájára kell mutatnia.

Tekintettel arra, hogy számos, különböző típusú merevlemezes meghajtó van forgalomban, a paraméterek értékéről egységes áttekintés nem adható (és, mint mondtuk, nincs is rá szükség).

A 10. és 11. sorszámú funkciókkal a 2. és 3. sorszámú funkciókhoz hasonlóan a lemez egy sávja olvasható (10), ill. írható (11). A különbség az, hogy a 10-es és a 11-es funkciókkal egy szektor 512 adatbájtján túl még a minden egyes szektor végén, az adatbájtok után álló 4 ECC-bájt is olvasható, ill. írható. A funkciókat ezért „hosszú" (long) vagy bővített olvasásnak ill. írásnak nevezik. Mivel ezeknél a funkcióknál az elvégzendő művelet nem 512, hanem 516 bájtos, „bővített" szektor(ok)ra vonatkozik, egyrészt a szektor(oka)t befogadó puffer méretét is ennek megfelelően meg kell növelni, másrészt – mivel a DMA most is csak 64 kbájtot tud egyszerre mozgatni – az egyetlen funkcióhívással beolvasható vagy kiírható szektorok száma 128 helyett csak 127 lehet. A paraméterek átadásában és a regiszterkiosztásban nincs különbség.

A 16. sorszámú funkcióval az vizsgálható, hogy a meghajtó üzemkész állapotban van-e, vagyis hogy az utolsó művelet befejeződött-e. A meghajtó azonosítóját most is a DL regiszterben kell megadni. Ha a meghajtó üzemkész, akkor visszatéréskor az átvitelbit értéke 0, és az AH tartalma is 00. Ellenkező esetben az AH-ban a hibakód áll.

A 17. sorszámú funkció újraállítja (angol szóhasználattal „rekalibrálja") a meghajtó író/olvasó fejeit. Az újraállítás azt jelenti, hogy a fejek a 0. cilinderre állnak be. Hibás végrehajtás esetén a hiba kódját most is az AH regiszterben kapjuk vissza.

A 20. sorszámú funkció meghívásakor a meghajtót vezérlő áramkörökön lefut egy belső öntesztelő rutin, amely ugyancsak az átvitelbit értékével jelzi a művelet végrehajtásának sikerességét.

A 21. sorszámú funkcióval – amely csak PC/AT gépeken használható – a lemezmeghajtó típusa kérdezhető le. A meghajtó azonosító számát (80(h) vagy 81(h)) most is a DL regiszterben kell megadni. Ha a hivatkozott meghajtó nem létező meghajtó, akkor a funkció az AH regiszterbe a 00 értéket írja vissza. Ha az AH tartalma 03, akkor ez azt jelenti, hogy a kérdezett meghajtó merevlemezes meghajtó, és a CX:DX regiszterpárban lévő 32 bites érték a merevlemezen elhelyezkedő 512 bájtos szektorok számát adja meg.

Ezzel befejeztük azoknak a funkcióknak az ismertetését, amellyekkel a merevlemezes meghajtó BIOS-ból vezérelhető. Ehhez a fejezethez nem adunk példaprogramokat, mert – mint erről már volt szó – egy ilyen programban elég egy véletlen „mellényúlás", és a lemezen máris jóvátehetetlen károkat okozhatunk. Hajlékonylemezek esetén a helyzet nem ennyire súlyos, hiszen azokról – mielőtt esetleg még elrontanánk – könnyűszerrel

készíthető másolat, így a kísérletezgetés nem jár igazi veszéllyel. A merevlemezekről viszont nem lehet ilyen könnyen másolatot készíteni (hacsak nincs streamerünk), ezért a merevlemezes meghajtók vezérlését lehetőség szerint bízzuk a DOS-ra, vagy valamely közismerten bevált segédprogramra

1.6. A soros interfész vezérlése

A számítógépek alkalmazásának egyik legérdekesebb és legtöbbet ígérő felhasználási területe a számítógépes távadatforgalmazás. Távadatforgalmazás alatt itt egészen általánosan azt az eljárást értjük, amely a számítógépeknek azt a képességét használja ki, hogy ezek a gépek – egymással alkalmas módon összekapcsolva – nemcsak a közvetlenül hozzájuk kapcsolt billentyűzetről vagy a lemezmeghajtóikban lévő lemezekről, hanem a tőlük – elméletileg – bármilyen távolságra lévő gépekről is képesek adatokat fogadni, ill. azokra adatokat küldeni. (Csak a rend kedvéért: adatok természetesen egy távoli nyomtatóra is küldhetők.)

Ahhoz, hogy két (vagy több) számítógép között ilyen kapcsolat létrejöjjön – nagyon leegyszerűsítve – alapvetően három dolog szükséges. Ezek közül az egyik az, hogy az egymással kapcsolatban álló számítógépeknek legyen olyan „érzékszervük", amellyel képesek venni a másik gép üzenetét, illetve amellyel üzenetet tudnak küldeni. A másik dolog az, hogy – mivel a köztük lévő távolságot elméletben nem korlátozzuk – vagy legyen igen érzékeny a „fülük" és erőteljes a „hangjuk", vagy valamilyen más módot kell találni az üzenetváltáshoz. A megértés és megértetés harmadik feltétele a közös nyelv használata. (Igazából ezt a három feltételt közösen megvalósító hardver és szoftver együttest kellene interfésznek – eredeti, angol írásmódja interface – nevezni, de a számítástechnikai köznyelv sokszor ezek egyes elemeit is így hívja.)

A számítógépek közötti kapcsolat létrehozásához ezt a három feltételt a következőképpen valósítják meg:

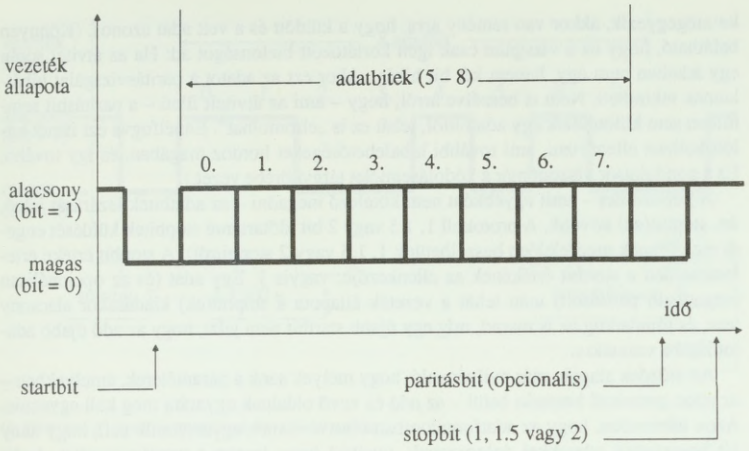
- A számítógépbe hardver egységként beépíthető az ún. soros interfész. Soros aszinkron kommunikációs adapternek is nevezik, angol neve asynchronous communication adapter.) Magát az interfészt egy bonyolult nyomtatott áramköri kártya, az ún. RS232 kártya valósítja meg, amelyet a PC alaplemezén lévő bővítő csatlakozók valamelyikébe kell helyezni. A számítógépek közötti kapcsolatban ez a – már szabvánnyá vált – RS232 kártya jelenti a gépek „érzékszervét". A kártya alapvető feladata, hogy a számítógép által előállított adatokat – megfelelő szoftvertámogatással – a számítógép kimenetére küldje, illetve a bemenetére érkező adatokat alkalmas formában a gép rendelkezésére bocsássa.
- A számítógépek között az adatok a gépeket közvetlenül összekötő kábelen vagy telefonvonalakon keresztül haladhatnak. A kábeles kapcsolat a külső elektromos zajok és a jelveszteség miatt csak egymástól néhány tíz méter távolságban lévő gépek között hozható létre. Ennél nagyobb távolságok áthidalásához a hétköznapi telefon-összeköttetéseket is megvalósító postai telefonvonalakat kell igénybe ven-

ni. Mivel azonban ezeket a vonalakat az ember számára hallható hangok továbbítására alakították ki, gondoskodni kell arról, hogy a számítógép jeleiből is hallható hangok legyenek. Ezt az átalakítást végzik el az ún. akusztikus csatolók vagy modemek. Ezek a készülékek a bemenetükön a számítógép (pontosabban az RS232 kártya) által kibocsátott, a küldendő adatokat reprezentáló bitsorozatot fogadják, ezeket hallható hangokká (de nem beszéddé!) alakítják át, majd ezeket a hangokat „belemondják" a kimenetükhöz alkalmas módon illesztett közönséges telefonkagyló mikrofonjába. Ha most az előzőleg telefonon hívott fél telefonkagylója is egy ilyen modemhez, ezen keresztül pedig egy számítógéphez csatlakozik, akkor – elméletileg – létrejött a két gép közötti kapcsolat: A hívott fél modemje „hallja" a telefonkagyló hangszórójából érkező hangokat, amelyek ugyancsak egy RS232 kártyán keresztül a számítógép számára érthetővé és feldolgozhatóvá válnak.

– Harmadik feltételként említettük, hogy ahhoz, hogy az adatforgalmazásban résztvevő felek az elküldött, illetve vett adatokat, vezérlőjeleket azonosan értelmezzék, vagyis hogy megértsék egymást, valamiféle közös nyelvet kell használniuk. Ezt a közös nyelvet az adatforgalmazásban protokollnak nevezik, amely lényegében nemzetközileg elfogadott szabvány-, ill. ajánlásgyűjtemény. Ez a gyűjtemény az adatforgalmazásban szerepet játszó áramkörök működését, a kapcsolatot létrehozó vezetékek jelszintjét, időzítési előírásokat, a csatlakozók kialakítását stb. írja elő. (A PC-k közötti adatforgalomban – jelenleg – használandó szabványgyűjtemény jele az EIA amerikai szabvány szerint RS232C, a (nyugat-)európai CCITT szabvány szerint V.24.)

Ennyi bevezető után nézzük meg, hogy a soros interfész által kiküldésre előkészített adat hogyan jelenik meg a vezetéken. Adatokon természetesen most is valamilyen – megállapodás szerinti számú bitekből álló – bitsorozatot értünk. Mivel az interfész soros üzemű, az egyes bitek időben egymás után kerülnek a vezetékre. (Az adatforgalmazás egyébként ún. full-duplex módon folyik, vagyis mind az adás, mind a vétel külön vezetékeken történik, tehát az adatok mindkét irányban időben egyszerre forgalmazhatók.)

A protokoll szerint a vezetéknek csak két állapota lehet: az egyik állapotot alacsonynak (angolul low), a másikat magasnak (angolul high) nevezzük. A két különböző állapotot természetesen két különböző feszültségszint valósítja meg, amelyek megállapodás szerint a bit két lehetséges értékét reprezentálják: a –3 és –12 V közötti feszültségszint az alacsony (bitérték = 1), a +3 és a +12 V közötti feszültségszint a magas állapotnak (bitérték = 0) felel meg. A 11. ábra idődiagramján bemutatjuk, hogy a protokoll szerint időben hogyan jelennek meg egy adat bitjei a vezetéken.



11. ábra: A soros átviteli protokoll idődiagramja

Az ábrából látható, hogy a megfigyelési „időszelet" kezdetén a vezeték állapota alacsony. Ez azt jelzi, hogy a vezetéken nem folyik adattovábbítás. A vevő oldal számára ennek az állapotnak a megváltozása jelzi, hogy az adó adatokat akar küldeni. Ez az állapot egy „bitnyi" ideig tart (hogy ez mennyi, arra még visszatérünk). Ezt nevezzük startbitnek, amely időtartam alatt a vevő oldal szinkronba állhat az adó oldallal, és felkészülhet az adatbitek fogadására. A startbitet követik az adatbitek, amelyek száma a protokoll szerint ugyan 5 – 8 lehet, de a BIOS csak 7, ill. 8 bit hosszúságú adatok kezelését támogatja. Az adatbitek továbbítása során a vezeték állapota természetesen a kiküldött bitek értékétől függően magas vagy alacsony. Az idődiagramból az is látható, hogy az időben elsőként kiküldött bit az adat legkisebb, az utolsóként kiküldött bit pedig a legnagyobb helyértékű bitje.

Az adatbiteket időben egy ún. paritásbit követheti, amely segítségével a kiküldött adat vételének helyessége – meglehetősen szűk kereteken belül – vizsgálható. A vizsgálat páros vagy páratlan paritásvizsgálatra vonatkozhat. A vizsgálat lényege, hogy a küldő fél megvizsgálja, hogy az éppen elküldött adatban hány 1 értékű bit volt. Ha a vizsgálat – előzetes megegyezés szerint – páros paritásra vonatkozik, akkor a paritásbit értékét 1-re vagy 0-ra állítja, hogy az adatbitek a paritásbittel kiegészítve páros számú 1 értékű biteket tartalmazzanak. Ha tehát a kiküldött adatban pl. 3 darab 1 értékű bit volt, akkor a paritásbit értékét 1-re állítja, így ezek együttesen 4, azaz páros számú 1 értékű bitet fognak tartalmazni. Ezzel szemben ha az adatbit 1 értékű bitjeinek a száma páros volt, akkor a paritásbit értéke 0 lesz. A vevő oldal az adat vételekor ugyancsak megszámolja a vett adatban lévő 1 értékű biteket, és a saját számítása alapján kapott értéket egészíti ki egy 1 vagy 0 értékű paritásbittel. Ha a küldött és a saját számítás szerinti paritásbit érté-

ke megegyezik, akkor van remény arra, hogy a küldött és a vett adat azonos. (Könnyen belátható, hogy ez a vizsgálat csak igen korlátozott biztonságot ad: Ha az átvitel során egy adatban nem egy, hanem két bit hibás, akkor ezt az adatot a paritásvizsgálat hibátlannak tekintheti. Nem is beszélve arról, hogy – ami az átvitelt illeti – a paritásbit semmiben sem különbözik egy adatbittől, tehát ez is „elromolhat". Ennélfogva ezt ismét külön kellene ellenőrizni, ami további hibalehetőségeket hordoz magában, és így tovább. Ez a gondolatsor viszont már a kódoláselmélet tárgykörébe vezet.)

A paritásbitet – amit egyébként nem kötelező megadni – az adatbitek lezárását jelző, ún. stopbit(ek) követik. A protokoll 1, 1.5 vagy 2 bit időtartamú stopbitek küldését engedi meg (ennek megfelelően beszélhetünk 1, 1.5 vagy 2 stopbitről). A stopbit értéke értelemszerűen a startbit értékének az ellenkezője, vagyis 1. Egy adat (és az opcionálisan megadható paritásbit) után tehát a vezeték állapota a stopbit(ek) kiadásakor alacsony lesz, és mindaddig az is marad, míg egy újabb startbit nem jelzi, hogy az adó újabb adatot küld a vezetékre.

Az előzőek alapján már nyilvánvaló, hogy melyek azok a paraméterek, amelyekben – az adott protokoll keretein belül – az adó és vevő oldalnak egyaránt meg kell egyeznie. Azon túlmenően, hogy az adatforgalomban résztvevőknek egyeztetniük kell, hogy hány bit hosszúságú adatokkal dolgozzanak, továbbá hogy legyen-e paritásvizsgálat, és ha igen, akkor ez páros vagy páratlan legyen, alapvető fontosságú, hogy megállapodjanak az adatátvitel sebességében.

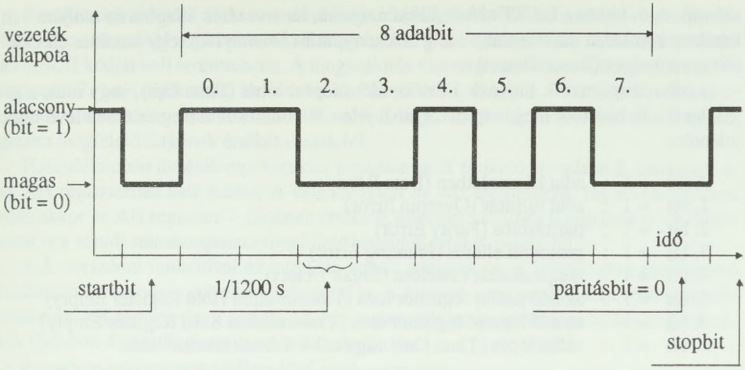
A digitális adatátvitelben az adatátvitel sebességének egysége a baud (bd), amely a másodpercenként továbbított bitek számát jelenti. Bár az RS232 elég széles határokat állapít meg az adatátviteli sebességekre, a ROM-BIOS maximum 9600 bit/s sebességet támogat.

A vezetékre küldendő adatbitek száma a továbbítandó adatok jellegétől függ. Ha csak a normál (0 – 127) ASCII-kódú adatok átvitelére van szükség, akkor ezekhez elegendő 7 adatbit, míg a 128-tól kezdődő, kibővített ASCII-kódú karakterek továbbításához mind a 8 adatbitet igénybe kell venni.

Az adatátvitel biztonságát növeli, ha az átvitel során paritásvizsgálatot végzünk. Annak, hogy ez páros vagy páratlan paritásra vonatkozik, nincs semmi jelentősége.

Ugyancsak egyeztetni kell a stopbitek hosszúságát, hogy a következő adatbitet indító startbit ebbe ne „lógjon" bele.

A 12. ábrán bemutatott idődiagram hasonlít a 11. ábrán bemutatott diagramhoz, amelyen azt mutattuk be, hogy a soros interfész által kiküldésre előkészített adat a soros adatátviteli protokoll szerint általában hogyan jelenhet meg a küldő vezetéken. A 12. ábrán a protokollt konkretizáltuk: az adatátvitel sebességét 1200 baudban határoztuk meg, páros paritásvizsgálatot választottunk, a stopbitek időtartamát 1 bitre korlátoztuk, és egy adatot 8 bittel adtunk meg. Lássuk, hogy ilyen protokoll mellett miként alakul az idődiagram, ha az „S" karaktert akarjuk kiküldeni. Az S karakter ASCII-kódja a decimális 83, ez bináris alakban 01010011(b). Ebben a bitsorozatban 4 bit értéke 1, ezért a paritásbit értéke most 0 lesz (lásd a 12. ábrát).



12. ábra: Az „S" karakter átvitele adott protokoll mellett

Ahhoz, hogy a BIOS által a soros interfész vezérléséhez rendelkezésünkre bocsátott funkciókat megfelelőképpen használni tudjuk, röviden meg kell ismerkednünk az RS232 kártya fő áramköri egységét alkotó processzor legfontosabb regisztereivel. A processzornak a szakirodalomban UART a neve (ami a Universal Asynchronous Receiver/Transmitter – univerzális aszinkron vevő/adó – kifejezésből képzett betűszó).

Amikor egy karaktert küldünk az adatvezetékre, akkor ez először az UART processzor Transmission Hold Registerébe kerül, és mindaddig itt marad, míg a processzor az előzőleg beérkezett karaktert fel nem dolgozta. Ezután a karakter átkerül a Transmission Shift Registerbe, ahonnan a processzor a karakter bitjeit sorban a vezetékre küldi. Ennek során a beállított konfigurációnak megfelelően a bitfolyamba illeszti a paritás- és a stopbiteket. Mint majd látni fogunk, a BIOS-funkciók a megszakításból való visszatéréskor a soros interfész ún. vonali státuszregiszterét (Line Status Register), amely az adatokat forgalmazó vezeték állapotáról ad felvilágosítást, a (CPU processzorának) AH regiszterébe másolják át. Ennek a vonali státuszregiszternek az 5. és 6. bitje jelzi, hogy az előbb említett két regiszter (Transmission Hold Register és Transmission Shift Register) üres-e.

Amikor az adatvezetéken egy karakter érkezik, akkor ez először az UART ún. Receiver Shift Registerébe kerül, ahonnan áttöltődik a processzor Receiver Buffer Registerébe. Itt az UART a bitsorozatból kiszűri a paritás- és stopbiteket. Ha a regiszterben még volt egy – korábban vett – karakter, akkor ezt az újonnan érkező karakter felülírja, és az interfész vonali státuszregiszterében az 1. bit magasra állításával hibát jelez (túlfutási – overrun – hiba). A felülírt karakter elveszik.

A státuszregiszter 0. bitjének (Data Ready) 1 értéke jelzi, hogy egy vett karakter átkerült a Receiver Buffer Registerbe. A bit a karakter kiolvasásakor automatikusan törlődik. Ha az UART a vett karakter feldolgozása során paritáshibát észlel, akkor a státuszregiszter 2. bitjét magasra állítja. A státuszregiszter 3. bitje azt jelzi, hogy a küldő fél a karakter küldése során betartotta-e a megállapodás szerinti protokollt (paritásbit és stopbitek

száma). A 4. bitet az UART akkor állítja magasra, ha a vezeték állapota az utoljára vett karakter elküldése óta hosszabb ideig alacsony, mint amennyi idő egy karakter elküldéséhez szükséges (Break Detect).

A státuszregiszter 7. bitjének 1 értéke időtúllépési hibát (Time Out), vagy más, a regiszter 0 – 6. bitjeiből megállapítható hibát jelez. A vonali státuszregiszterben lévő bitek jelentése:

0. bit = 1 : adat készenlétben (Data Ready)
1. bit = 1 : adat túlfutás (Overrun Error)
2. bit = 1 : paritáshiba (Parity Error)
3. bit = 1 : protokoll eltérés (Framing Error)
4. bit = 1 : megszakadás észlelése (Break Detect)
5. bit = 1 : az adó puffer regiszter üres (Transmission Hold Register Empty)
6. bit = 1 : az adó léptető regiszter üres (Transmission Shift Register Empty)
7. bit = 1 : időtúllépés (Time Out) vagy a 0 – 6. bitek szerinti hiba

Ezek után lássuk a soros interfész vezérlésére szolgáló BIOS funkciókat. Ezekből mindössze 4 van, amelyek mindegyike a 20 – 14(h) megszakításokon keresztül hívható meg.

A 0. sorszámú funkcióval a megállapodás szerinti protokoll állítható be, ami egyet jelent az RS232 kártya inicializálásával és konfigurálásával. Ekkor állítható be az adatátviteli sebesség értéke, az adatbitek száma (erre a BIOS csak 7, ill. 8 bitet tesz lehetővé), a paritásvizsgálat és a stopbitek száma. Ezeket a paramétereket az AL regiszterben kell átadni a következők szerint:

0 – 1. bit: adathossz
10 = 7 bit
11 = 8 bit
2. bit: stopbitek száma
0 = 1 stop bit
1 = 2 stop bit
3 – 4. bit: paritásvizsgálat
00 = nincs paritásvizsgálat
01 = páratlan paritás
11 = páros paritás
5 – 7. bit: átviteli sebesség
000 = 110 baud
001 = 150 baud
010 = 300 baud
011 = 600 baud
100 = 1200 baud
101 = 2400 baud
110 = 4800 baud
111 = 9600 baud

Az inicializálást követően a megszakítás az AH regiszterbe az interfész vonali státuszregiszterét másolja be.

Karakter kiküldésére az 1. sorszámú funkció szolgál. A funkció számát – mint általában – most is az AH regiszterben kell átadni. Az AL regiszternek a kiküldendő karakter ASCII kódját kell tartalmaznia. A megszakítás visszatérésekor az eredményről az AH regiszter 7. bitje tájékoztat: ha a művelet sikeres volt, akkor a bit értéke 0, ellenkező esetben a karaktert nem sikerült elküldeni, és a regiszter 0 – 6. bitjei a vonali státuszregiszter megfelelő bitjeinek értékét veszik fel.

Hasonlóképpen történik egy karakter fogadása is. A funkció sorszáma 2, amit most is az AH regiszterben kell átadni. A vett karaktert az AL tartalmazza. Ha a vétel sikeres volt, akkor az AH regiszter 7. bitjének értéke 0, ellenkező esetben a regiszter 0 – 6. bitjei most is a vonali státuszregiszter megfelelő bitjeinek értékét veszik fel.

A 3. sorszámú funkcióval az interfész vonali státusza, és az interfészhez esetlegesen csatlakoztatott modem státusza kérdezhető le. A funkció a megszakításból való visszatéréskor az AH regiszterbe a vonali, az AL regiszterbe pedig a modem státuszregiszterének (Modem Status Register) tartalmát másolja be.

A modem státuszregiszterében lévő bitek jelentése:

0. bit: CTS változás (Delta Clear To Send, DCTS)
1. bit: DSR változás (Delta Data Set Ready, DDSR)
2. bit: RI változás (Trailing Edge Ring Indicator)
3. bit: RLSD változás (Delta Receive Line Signal Detect)
4. bit: modem adásra kész (Clear To Send, CTS)
5. bit: modem rákapcsolva (Data Set Ready, DSR)
6. bit: telefon cseng (Ring Indicator, RI)
7. bit: kapcsolat a vevő oldali modemmel létrejött (Receive Line Signal Detect, RLSD)

Az fenti felsorolásból látszik, hogy a regiszter 4 – 7. bitjei lényegében ugyanazt a jelentést hordozzák, mint a 0 – 3. bitek. A különbség az, hogy míg a 4 – 7. bitek a modem mindenkori állapotáról adnak felvilágosítást, a 0 – 3. bitek – erre utal a változásjelző megjegyzés (angol megfelelőjük a Delta) – a modem állapotában az utolsó lekérdezés óta esetlegesen bekövetkezett változást tükrözik vissza. Eszerint tehát ha pl. a 2. bit értéke 1, akkor ez azt tükrözi, hogy a 6. bit értéke a státusz utolsó lekérdezése óta megváltozott. A gyakorlatban ez azt jelenti, hogy a 2. bit lekérdezésével az állapítható meg, hogy a telefon – a 6. bit értékétől függően – éppen megkezdte vagy abbahagyta a csengetést.

1.7. A (korábbi) kazettamegszakítás BIOS funkciói az AT-n

A BIOS 15(h) számú, egykoron „kazettamegszakítás"-nak nevezett megszakítása a PC-k eddigi fejlődése során meglehetősen változatos szerepet kapott. Amint erről korábban már volt szó, a kezdeti időkben, amikor az első PC-k megjelentek a piacon, az IBM – talán óvatosságból, talán piaci meggondolásokból – az első gépeket úgy tervezte meg,

hogy azok háttértárként kazettás magnóval is működhessenek. Ebből a célból építette be a ROM-BIOS-ba a 15(h) számú megszakítást, amellyel a kazettás egységet lehetett vezérelni. E megszakításon keresztül összesen 4 funkciót lehetett meghívni, amelyek a magnó motorjának be-, ill. kikapcsolására, valamint a szalag írására és olvasására szolgáltak.

Mint tudjuk, ezek a kazettás magnók a gyakorlatban nem terjedtek el, és a PC-k lényegében kezdettől fogva kizárólag hajlékony- és merevlemezes egységeket használtak és használnak elsődleges háttértárként. A kazettamegszakítás rutinjai azonban az XT-k megjelenéséig változatlanul a ROM-BIOS részét képezték. Az XT-k piacra kerülésekor már mindenki számára egyértelművé vált, hogy a géphez senki sem fog kazettás egységet használni, így ezekből a rutinokból is csak annyi maradt, hogy meghívásukkor magasra állították az átvitelbitet és egy hibakóddal jelezték, hogy a hívott funkció nem létezik.

Az AT megjelenésekor a kazettamegszakítás – a korábbi 4 „üres" funkciót megtartva – ismét szerephez jutott, természetesen teljesen más feladatokkal és kibővített funkciókészlettel. A következőkben ezeket az új funkciókat vizsgáljuk meg.

A 131 – 83(h) és a 134 – 86(h) funkciók időméréssel kapcsolatos feladatokhoz használhatók. Mint bizonyára ismert, az AT gépek alapkiépítettségükben tartalmaznak egy RTC/CMOS-nak nevezett integrált áramkört (az RTC a Real-Time Clock = valós idejű óra, a CMOS a Complementary Metal Oxide Semiconductor = komplementer fém-oxid félvezető rövidítése), amelynek egy, a gép operatív tárától független, saját 64 bájtos RAM tára van. Ez az áramkör a működtető feszültséget egy gombakkumulátorról kapja, ami lehetővé teszi, hogy a RAM tartalma a gép kikapcsolásakor vagy áramszünet esetén sem veszik el. Maga a RAM egyrészt a számítógép konfigurációjáról tartalmaz információkat, másrészt a valós idejű órát működtető áramkörök itt tárolják a „pontos" időt. (A BIOS a gép bekapcsolásakor innen is olvassa ki, ezért nincs szükség arra, hogy – miként az XT-nél – minden rendszerindításkor külön megadjuk.)

Az említett, 83(h) és 86(h) funkciók számára is az ebben a CMOS RAM-ban tárolt idő képezi az időmérés alapját. A 83(h) funkció a meghívását követően a programozó által megadott idő eltelte után egy – ugyancsak a programozó által megadott – jelzőbájt (ún. szemafor) 7. bitjét magasra állítja. A funkció számát az AH regiszterbe, a jelzőbájt szegmens- és ofszetcímét az ES:BX regiszterpárba, azt az időintervallumot pedig, aminek az eltelését mérjük, a CX és DX regiszterekbe kell beírni. A CX és a DX regiszter most egy 32 bites számot tartalmaz, amely az időintervallumot adja meg mikroszekundumokban (milliomod másodpercekben). A CX regiszter a 32 bites szám felső 16 bitjét, a DX pedig az alsó 16 bitjét tartalmazza az alábbiak szerint:

31 16	15 0
int (időintervallum/65536)	időintervallum mod 65536
CX	DX

Megjegyezzük, hogy bár a funkció valóban milliomod másodperceket mér, a mérés egysége 976 milliomod másodperc, azaz a funkció csak azt tudja figyelni, hogy az adott időintervallumon belül hányszor 976 milliomod másodperc telt el. Ezért a funkció meg-

hívása és a reagálás között annyiszor 976 milliomod másodperc telik el, ahányszor elkezdődik egy 976 milliomod másodperces számlálási ciklus a megadott intervallumon belül.

A felhasználói program feladata, hogy a funkció meghívása előtt a jelzőbájt (szemafor) 7. bitjét törölje. Ha a funkciót úgy hívjuk meg, hogy a bit értéke 1, akkor a funkció hibajelzéssel (átvitelbit = 1) tér vissza. A funkció a meghívását követően nem várja meg a megadott időtartam elteltét, hanem azonnal visszatér a meghívó programhoz, amely rögtön folytathatja a megszakítás meghívásakor abbahagyott program végrehajtását. Íly módon a funkció jól használható arra, hogy a processzor rendszeres időközönként akkor is figyelni tudjon valamilyen külső eseményre, ha az ideje nagy részét éppen egy bonyolult feladat végrehajtása foglalja le.

A bit beállítását ugyancsak ezzel a funkcióval érvényteleníthetjük úgy, hogy a funkció meghívásakor az AL regiszterbe 1-et írunk.

A másik, 86(h) számú, ugyancsak időmérésre szolgáló funkció lényegében az előző ellentétének tekinthető: nem állít be semmiféle jelzőbitet, és a meghívását követően nem tér azonnal vissza, hanem a megadott időtartamra felfüggeszti a meghívó program végrehajtását. Ennek megfelelően a funkció meghívásakor csak a funkció számát és a várakozási időt kell megadni. Ez utóbbit most is a CX:DX regiszterpárnak kell tartalmaznia, a 83(h) funkciónál ismertetett módon. A 86(h) funkció csak akkor hívható meg, ha egy esetlegesen előzőleg meghívott 83(h) funkcióval megadott időintervallum már letelt.

A 15(h) megszakítás másik két funkciója a tárkezeléssel kapcsolatos. Mint ismeretes, az AT bevezetésével lehetővé vált, hogy a számítógép az ún. hagyományos, 640 kbájtos operatív tárnál nagyobb RAM tárat tartalmazzon. Ezt az teszi lehetővé, hogy az AT a korábbi típusoktól eltérően már nem a 8086/88-as, hanem a család újabb, 80286-os típusjelű mikroprocesszorára épül. Mielőtt ismertetnénk ezt a két funkciót, röviden foglaljuk össze azokat a – témánk szempontjából – leglényegesebb újdonságokat, amelyeket a 80286-os processzor megjelenése magával hozott.

A 80286-os processzorban a címvezetékek száma 20-ról 24-re bővült, így a (fizikailag) címezhető tártartomány 1 Mbájtról 16 Mbájtra nőtt. Egy másik lényeges változás, hogy a processzor két, egymástól alapvetően különböző üzemmódban működtethető. Ezek egyike, az ún. valós üzemmód (real mode), amely teljes mértékben megegyezik a PC-kben és XT-kben lévő processzor üzemmódjával, vagyis ebben az üzemmódban az AT pontosan úgy programozható, mint egy PC vagy egy XT (legalábbis ami a processzort illeti; a DOS és BIOS rendszerprogramok eltéréseit természetesen figyelembe kell venni). A 80286-os másik, „új" üzemmódját virtuális, védett címzési módnak nevezik (virtual protected mode): ebben az üzemmódban a processzor 1 Gigabájtos virtuális tárterületet képes megcímezni úgy, hogy ezt a processzor tárkezelő egysége – erre alkalmas operációs rendszer segítségével – a számítógépben fizikailag is létező 16 Mbájtos tartományra képezi le. A védett címzési mód másik alapvető újdonsága, hogy a processzoron időben (legalábbis látszólag) egyszerre több felhasználó különböző programja futhat anélkül, hogy ezek egymást akadályoznák, vagy hogy az egyik felhasználó egy másik felhasználó adataihoz jogtalanul hozzáférne (ezt nevezik angolul multiuser/multitasking felhasználásnak).

Sajnos a DOS, amely az IBM PC/AT-nek is az operációs rendszere, a 80286-os mikroprocesszornak csak a valós címzési módban történő programozását segíti. Ez azt jelenti, hogy a DOS sem a virtuális címzési mód használatát (és így az 1 Mbájt feletti tár kezelését), sem a többfelhasználós, többfeladatos felhasználást nem teszi lehetővé. BIOS szintről is mindössze két olyan funkció létezik, amely a 16 Mbájtra kibővült fizikai tár kezelésével kapcsolatos. Ez az a két funkció, amelyet az imént említettünk, és amelyek a 15(h) megszakítás meghívásával érhetők el. Nézzük meg most ezeket.

A 136 – 88(h) számú funkcióval azt állapíthatjuk meg, hogy a számítógépben mekkora az az 1 Mbájt feletti – HI-RAM-nak vagy Extended Memory-nak is nevezett – RAM terület, amely csak akkor címezhető meg, ha a 80286-os processzor védett üzemmódban dolgozik. A megszakítás meghívásához csak a funkció számát kell az AH regiszterbe írni. Visszatéréskor az AX regiszter tartalmazza a bővített, vagyis az 1 Mbájt feletti tár méretét kbájtokban. Mint már mondtuk, a DOS ezekhez a tárcímekhez nem tud hozzáférni, ezért ezeken a bővített tárterületeken programok nem is helyezhetők el. Ennek ellenére ezek a területek – mint rögtön látni fogjuk – a DOS alatt programozók számára is érdekesek lehetnek.

A 135 – 87(h) számú funkció lehetővé teszi, hogy a 80286-os mikroprocesszor által fizikailag megcímezhető teljes RAM területen belül (640 kbájt hagyományos tár és a video RAM az 1 Mbájtos határ alatt, valamint 15 Mbájt az 1 Mbájtos határ fölött) meghatározott hosszúságú, egybefüggő blokkokat egy meghatározott kezdőcímű RAM területről egy másik, meghatározott kezdőcímű RAM területre átmásoljunk, vagyis meghatározott hosszúságú blokkokat a rendelkezésre álló tárterületen belül mozgathassunk. A funkció használatához a szokásos paraméterátadásokon kívül még további, előzetes intézkedések is szükségesek. Így például meg kell különböztetnünk azokat az eseteket, amikor a blokkot az 1 Mbájt alatti vagy az 1 Mbájt feletti területen belül, ill. az 1 Mbájt alatti területről az 1 Mbájt feletti területre vagy ellenkező irányba akarjuk mozgatni, továbbá a bővített tárcímek definiálásához 24 bites címeket is létre kell hozni. Ezeket az előkészítő lépéseket a programpéldák megjegyzései tartalmazzák.

A funkció végrehajtásához az AH regiszterben a funkció számát, a CX regiszterben az áthelyezendő blokk szavakban számított méretét, az ES:SI regiszterpárban pedig az ún. globális leíró tábla (Global Descriptor Table, GDT) kezdőcímét kell megadni. Ezen a ponton ismét egy rövid kitérőt kell tennünk, hogy a szegmensek védett címzésmódbeli definiálását és a globális leíró tábla e funkció által várt felépítését – legalábbis olyan mélységig, hogy a funkciót használni tudjuk – megismerjük.

Mint tudjuk, a 8086/88-as processzorok (és valós üzemmódban a 80286-os processzorok is) a szegmenscímeket egy-egy 16 bites szegmensregiszterben tartalmazzák. A szegmentált címszámításból következően a szegmensek a címezhető tár csak 16-tal osztható címein kezdődhetnek, és maximum 64 kbájt hosszúságúak lehetnek. Ezzel szemben a 80286-os virtuális, védett címzési módjában egy szegmens hossza 1 bájttól 64 kbájtig terjedhet, és a tár bármelyik címén kezdődhet. Ebben a címzési módban a szegmensregiszterbe betöltött érték nem egy szegmens címét, hanem egy ún. szelektort tartalmaz, amely egy, a tárban elhelyezkedő, és a szegmensre vonatkozó legfontosabb információkat leíró adatmezőre mutat. Ennek a 8 bájt hosszú adatmezőnek (amelyet a szegmens deszkriptorának is neveznek) a felépítését a 13. ábrán mutatjuk be.

15 . . .	. . . 8 7 . . .	. . . 0	bájtok 0 – 7
szegmens hossza			1 – 0
szegmens kezdőcímének 15 . . . 0 bitjei			3 – 2
hozzáférési jog bájt	a kezdőcím 23 . . . 16 bitjei		5 – 4
fenntartott, kompatibilitási okok miatt értéke 0			7 – 6

13. ábra: A szegmensleíró adatmező felépítése

Amint az ábrából látszik, a 8 bájtos szegmensleíró adatmező az első két bájton a szegmens hosszát, az ezt követő 3 bájton pedig a szegmens 24 bites kezdőcímét tartalmazza. Az 5-ös ofszetcímen az ún. hozzáférési jog bájt áll, amely a védett címzési módban a tárvédelemmel kapcsolatos információkat tartalmazza (a szegmens privilégiumszintje, a szegmens írhatósága stb.). Az adatmező utolsó két bájtjának kompatibilitási okok miatt 0-nak kell lennie.

Térjünk vissza a globális leíró táblához. A 80286-os védett címzési módjában a programokhoz tartozik – több más adatszerkezet mellett – egy globális leíró tábla, amely – többek között – azoknak a szegmenseknek a szegmensleíró adatmezőjét tartalmazza, amelyeket a páhuzamosan futó programok mindegyike használhat. Az a globális leíró tábla, amelyet az általunk vizsgált blokkmozgató funkció vár, szintén szegmensleíró adatmezőkből áll. A táblának 6 darab ilyen, egyenként 8–8 bájtos adatmezőt kell tartalmaznia. A tábla felépítését a 14. ábrán mutatjuk be.

	ofszetcím
fenntartott (8 db 0 értékű bájt)	00(h) – 07(h)
magát a leíró táblát leíró adatmező	08(h) – 0F(h)
a mozgatandó blokk forrásszegmensét leíró adatmező	10(h) – 17(h)
a mozgatandó blokk célszegmensét leíró adatmező	18(h) – 1F(h)
a védett módban futó rendszerprogram kódszegmensét leíró adatmező	20(h) – 27(h)
a BIOS által védett módban létrehozott veremszegmenst leíró adatmező	28(h) – 2F(h)

14. ábra: A globális leíró tábla (GDT) felépítése

A funkció meghívásakor a fenti táblába csak a mozgatandó blokk forrás- és célszegmensének kezdőcímét, hosszát és a hozzáférési jog bájtját kell beírnunk, az összes többi bájt értékének 0-nak kell lennie. Nézzük meg, hogy eszerint mit kell tartalmaznia az ES:SI által megcímzett puffernek:

00(h) – 0F(h) : 16 darab 0 értékű bájt. Az első 8 bájt marad 0, a második 8 bájtnak a funkció végrehajtásakor a BIOS ad értéket.

10(h) – 11(h) : Forrásszegmens hossza bájtokban (minimum a CX-ben megadott szavak kétszerese – 1, maximum 64 kbájt).

12(h) – 14(h) : Forrásszegmens kezdőcíme 24 biten. A cím itt nem szegmentált (szegmens:ofszet), hanem egy, a 000000(h) – FFFFFF(h) tartományban lévő fizikai cím. A legkisebb helyértékű bit a legalacsonyabb, a legnagyobb helyértékű bit a legmagasabb címet jelenti.

15(h): Hozzáférési jog bájt (az értéke 92(h) legyen, mert ez jelenti azt, hogy a leírt szegmens a legnagyobb privilégiumszintű adatszegmens, amely a tárban van, a megadott kezdőcímtől felfelé nő és írható.

16(h) – 17(h) : Fenntartott terület, 2 darab 0 értékű bájt.

18(h) – 19(h) : Célszegmens hossza bájtokban (lásd a 10(h) – 11(h) címeket).

1A(h) – 1C(h) : Célszegmens kezdőcíme (lásd a 12(h) – 14(h) címeket).

1D(h): Hozzáférési jog bájt (az értéke 92(h), lásd a 15(h) címet).

1E(h) – 2F(h) : Fenntartott terület. A funkció előkészítésekor 0-ákkal kell feltölteni. Végrehajtáskor a BIOS módosítja.

A megszakításból való visszatéréskor most is az átvitelbit 0 értéke jelzi a művelet sikerességét. Ha az átvitelbit értéke 1, akkor a művelet sikertelen volt, és az AH egy hibakódot tartalmaz a következők szerint:

AH = 1 : RAM paritáshiba
AH = 2 : GDT a funkció meghívásakor nem volt rendben
AH = 3 : védett üzemmódot nem sikerült megfelelően inicializálni

A blokkmozgató funkcióval kapcsolatban meg kell jegyezni, hogy a funkció végrehajtásának idejére valamennyi megszakítás le van tiltva, vagyis a processzor erre az időre semmiféle megszakításkérésnek sem tud eleget tenni (még a rendszer belső órája is „megáll").

Megemlítjük még az ugyancsak a virtuális, védett címzésmóddal kapcsolatos 89(h) sorszámú funkciót: ezzel lehet a processzort a valós címzésmódból védett címzésmódba átkapcsolni. Mivel azonban – mint említettük – a DOS és a BIOS rendszerprogramok ebben az üzemmódban nem használhatók, ezzel a funkcióval nem foglalkozunk.

Ugyancsak a 15(h) megszakításon keresztül hívható meg az a funkció, amellyel az AT-hez kapcsolható két botkormány státusza kérdezhető le. A funkció, amelynek a sorszáma 132 – 84(h), két alfunkciót foglal magában. Közös bennük, hogy mindkét alfunkció visszatéréskor magasra állítja az átvitelbitet, ha a gépben nincs játékadapter, amelyhez a botkormány(oka)t csatlakoztatni lehetne.

Az első alfunkció használatához a DX regiszterbe 0-t kell írni. Visszatéréskor az AL regiszter a 4 – 7. bitjein a botkormányok tűzgombjainak állapotát tartalmazza.

A második alfunkcióval a botkormány(ok) állása kérdezhető le. Ehhez a DX regiszterbe 1-et kell írni. Visszatéréskor az AX és BX regiszterek az első botkormány, a CX és a DX regiszterek pedig a második botkormány X és Y irányú állását tartalmazzák.

Végezetül megemlítjük még a 133 – 85(h) sorszámú funkciót, amely a SysReq (System Request, rendszer igénylés) nevű billentyű állapotát kérdezi le. Ezt a funkciót hívja a BIOS billentyűzetmegszakító rutinja a SysReq billentyű lenyomásakor és felengedésekor. Maga a rutin a jelenlegi BIOS verziókban nem csinál semmi egyebet, minthogy egy IRET-tel azonnal visszatér. Egy felhasználói program a 15(h) megszakítási vektor átírásával és az AH regiszter figyelésével saját céljaira felhasználhatja ezt az

„üres" funkciót. Magának a billentyűnek az állapotát az AL regiszter tartalmazza: a 00 érték a billentyű lenyomott, a 01 a felengedett állapotot jelenti.

1.7.1. Programpéldák

A kazettamegszakítás által nyújtott szolgáltatások közül a mindennapi életben az 1 Mbájtos határ feletti tárterület kezelését segítő funkciók a leghasznosabbak. Ezek segítségével az egyébként a DOS által elérhetetlen szabad területek hozzáférhetővé válnak (ilyen területek csak az AT-nél létezhetnek). A példaprogramok a 87(h) funkció használatát szemléltetik, amelynek segítségével adatok mozgathatók az 1 Mbájtos határon keresztül.

Maguk a programok a képernyő RAM tartalmát másolják át közvetlenül az 1 Mbájtos határ fölé. Ez azért nagyon szemléletes, mert az átmásolást követően a video RAM-ot a programok azonnal felülírják, tehát vizuálisan is nyomon követhető a bekövetkező „adatvesztés". Az 1 Mbájt fölé tárolt adatok egy tetszőleges billentyű megnyomása után visszamásolódnak eredeti helyükre, a video RAM-ba. A példából is jól látszik, hogy az 1 Mbájt feletti tárterület jól használható pl. a képernyőterület veremszerű mentésére. A felhasználói programokban gyakran okoznak betelési problémákat a képernyőmentések, ezzel a módszerrel viszont sok (50–100) képernyőmentés is végezhető a program által egyébként kihasznál(hat)atlan területre.

A programok központi eljárása a globális leíró táblát építi fel, a 87(h) funkció számára átadásra kerül a leíró tábla címe, valamint a forrás- és célterület címe. Az 1 Mbájt feletti tartomány kezeléséhez természetesen a szokványos szegmentált címeket 24 bites fizikai címekké kell alakítani. A felsorolt teendőket mindegyik magasszintű nyelvben gépi kódú programrészletek végzik. Az egyes nyelvekben eltér a gépi kódú rész beágyazásának módszere. A BASIC program „DATA" adatsorokat és a „POKE" parancsot használja, a Turbo Pascal változat az „INLINE" paranccsal szúrja be a forráskód megfelelő helyére a megfelelelő eljárást. A C nyelvű példaprogramban csak az összeszerkesztés (LINK) során integrálódnak a gépi kódú eljárások. Megjegyzendő, hogy a gépi kódú programok jóformán csak a verem kezelésében, a paraméterátadás mechanizmusában térnek el egymástól.

```
1000 '
1005 '  ┌──────────────────────────────────────────────────────────┐
1010 '  │                      E G Y M E G A . B A S               │
1020 '  │                                                          │
1030 '  │ Funkciója : az 1 MB-os határ feletti tárterület (AT) kezelé-
1040 '  │             sének szemléltetése a video RAM "mozgatásának"
1050 '  │             bemutatásával.                               │
1055 '  │                                                          │
1060 '  └──────────────────────────────────────────────────────────┘
1070 '
1080 DEFINT Q                                        'Q "dummy" egész érték.
1090 KEY OFF : CLS
1100 PRINT : PRINT
1110 PRINT "FIGYELMEZTETÉS :" : PRINT
1120 PRINT "A 60000-es címtől kezdődően betoltott megszakításhívó ru-"
1130 PRINT "tin és tárterület mozgató rutin felulírását elkerulendő a"
1140 PRINT "GWBASIC-et GWBASIC /M:60000 paranccsal kell indítania. Ha"
1150 PRINT "nem így  hívta a GWBASIC-et, akkor usse le az <s> billen-"
1160 PRINT "tyűt, egyébként egy  tetszőleges másik billentyűt !"
1170 '
1180 Z$ = ""
1190 WHILE Z$ = ""                                   'Helytelen indítás
1200    Z$ = INKEY$                                  'esetén a program
1210 WEND                                            'leállítása.
1220 IF Z$ = "s" OR Z$ = "S" THEN CLS:END
1230 '
1260 CLS
1265 '
2000 '
2010 '  ┌──────────────────────────────────┐
2020 '  │        P R O G R A M T Ö R Z S    │
2030 '  └──────────────────────────────────┘
2040 PRINT : PRINT
2050 PRINT "Az 1  Mbájt feletti tárterület  elérését, ill.  kezelését"
2060 PRINT "szemlélteti a  tesztprogram. A jelenlegi képernyőtartalom"
2070 PRINT "(a video RAM  tartalma) elmentődik az 1 Mbájtos tártarto-"
2080 PRINT "mány legelejére, majd felulírődik. A teszt folytatása so-"
2090 PRINT "rán  visszamentődik  az eredeti video RAM tartalom a kép-"
2100 PRINT "ernyőre."
2105 PRINT
2110 PRINT "A teszt egy billentyű leutésére indul ..."
2120 '
2130 Z$ = ""                                         'Várakozás egy billentyű
2140 WHILE Z$ = ""                                   'leütésére.
2150    Z$ = INKEY$
2160 WEND
2170 '                                               'Ha a F000:FFFE bájt tar-
2180 DEF SEG = &HF000                                'talma nem FC(h), a teszt
2190 IF PEEK(&HFFFE) = &HFC THEN 2240                'nem futtatható (nem AT).
2200 '
2210 PRINT:PRINT "A tesztprogram csak AT-n futtatható !":PRINT
2220 END
2230 '
2240 DEF SEG                                     'A BASIC szegmens beállítása.
2250 GOSUB 9000                                  'Megszakításhívó és mozgató
2260 GOSUB 9500                                  'rutinok betoltése.
2270 '
2280 GOSUB 3000                                  'Videokártya meghatározása.
2290 IF VIDEO% = 7 THEN VIDSZEG% = &HB000 ELSE VIDSZEG% = &HB800
2300 '
2310 FORO% = 0 : FORS% = VIDSZEG%     'Forrás : a video RAM.
2320 CELO% = 0 : CELS% = 0            'Cél    : 1 MB felett 0000:0000.
2330 IRANY% = 1                       '1 Mb alattról 1 MB felé.
```

```
2340 NSZO%  = 2000                      '2000 szót kell átmásolni.
2350 GOSUB 4000                         'Átmásolás.
2360 '
2370 CLS                                                    'Video RAM
2380 FOR I% = 1 TO 880 : PRINT CHR$(1); : NEXT              'felulirása.
2390 FOR I% = 1 TO  80 : PRINT     " "; : NEXT
2400 PRINT "Eredeti képernyö elmentve 1 MB folé - ";
2410 PRINT "üsson le egy billentyüt visszatoltéséhez..";
2420 FOR I% = 1 TO  80 : PRINT     " "; : NEXT
2430 FOR I% = 1 TO 800 : PRINT CHR$(2); : NEXT
2440 '
2450 Z$ = ""                                    'Várakozás egy billentyü
2460 WHILE Z$ = ""                              'leütésére.
2470    Z$ = INKEY$
2480 WEND
2490 '
2500 FORO%  = 0 : FORS% = 0             'Forrás : 1 MB folott 0000:0000.
2510 CELO%  = 0 : CELS% = VIDSZEG%      'Cél    : a video RAM.
2520 IRANY% = 2                         '1 Mb felettről 1 MB alá.
2530 GOSUB 4000                         'Átmásolás.
2540 '
2550 LOCATE 24,1
2560 END
2570 '
3000 '
3010 '  A VIDEOKARTYA TIPUSANAK MEGHATAROZASA
3020 '     BE: -
3025 '     KI: monokróm videckártya esetén 7,
3030 '         színes kártyánál ettöl eltérö.
3040 '
3050 '
3060 FUN% = 15                          'Video uzemmód beolv. funkció.
3070 MSZ% = &H10                        '10(h) megszakítás.
3080 '
3090 CALL RCIM(MSZ%,FUN%,VIDEO%,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q)
3100 RETURN
3110 '
4000 '
4010 '  TARTERULET ELMOZGATASA (MASOLASA)
4020 '  BE : FORS%  = Forrásterület szegmenscím
4030 '       FORO%  = Forrásterület ofszetcím
4040 '       CELS%  = Célterület szegmenscím
4050 '       CELO%  = Célterület ofszetcím
4060 '       NSZO%  = Átmásolandó szavak száma
4070 '       IRANY% = Eltolási iránypараméter :
4080 '                0 : 1 MB alól  1 MB alá
4090 '                1 : 1 MB alól  1 MB folé
4100 '                2 : 1 MB folul 1 MB alá
4110 '                3 : 1 MB folul 1 MB folé
4120 '
4130 '
4140 CALL MOZGAT(FORS%,FORO%,CELS%,CELO%,NSZO%,IRANY%)
4150 RETURN
4160 '
9000 '
9010 '  A MEGSZAKíTASHíVó RUTIN INICIALIZALASA
9020 '     BE: -
9030 '     KI: RCIM a Rutin CIMe
9040 '
9050 '
9060 RCIM = 60000!                      'A rutin kezdöcíme a BASIC szegmensben.
9070 DEF SEG                            'A BASIC szegmens beállitása.
```

```
9080 RESTORE 9140
9090 FOR I% = 0 TO 160                          'A rutin betoltése a
9100     READ X% : POKE RCIM + I%, X%           'tárba.
9110 NEXT
9120 RETURN
9130 '
9140 DATA    85, 139, 236,  30,   6, 139, 118,  30, 139,   4, 232, 140
9150 DATA     0, 139, 118,  12, 139,  60, 139, 118,   8, 139,   4,  61
9160 DATA   255, 255, 117,   2, 140, 216, 142, 192, 139, 118,  28, 138
9170 DATA    36, 139, 118,  26, 138,   4, 139, 118,  24, 138,  60, 139
9180 DATA   118,  22, 138,  28, 139, 118,  20, 138,  44, 139, 118,  18
9190 DATA   138,  12, 139, 118,  16, 138,  52, 139, 118,  14, 138,  20
9200 DATA   139, 118,  10, 139,  52,  85, 205,   0,  93,  86, 156, 139
9210 DATA   118,  12, 137,  60, 139, 118,  28, 136,  36, 139, 118,  26
9220 DATA   136,   4, 139, 118,  24, 136,  60, 139, 118,  22, 136,  28
9230 DATA   139, 118,  20, 136,  44, 139, 118,  18, 136,  12, 139, 118
9240 DATA    16, 136,  52, 139, 118,  14, 136,  20, 139, 118,   8, 140
9250 DATA   192, 137,   4,  88, 139, 118,   6, 137,   4,  88, 139, 118
9260 DATA    10, 137,   4,   7,  31,  93, 202,  26,   0,  91,  46, 136
9270 DATA    71,  66, 233, 108, 255
9280 '
9500 '
9510 '  A TARTERULETMOZGATÓ RUTIN INICIALIZALASA
9520 '     BE: -
9530 '     KI: MOZGAT a rutin címe
9540 '
9550 '
9560 DEF SEG                            'A BASIC szegmens beállítása.
9570 MOZGAT = 61000!                    'A rutin kezdőcíme a BASIC szegmensben.
9580 RESTORE 9640
9590 FOR I% = 0 TO 140                          'A rutin betoltése a
9600     READ X% : POKE MOZGAT + I%, X%         'tárba.
9610 NEXT
9620 RETURN
9630 '
9640 DATA &HE8, &H73, &H00, &H00, &H00, &H00, &H00, &H00, &H00, &H00, &H00, &H00
9650 DATA &H00, &H00, &H00, &H00, &H00, &H00, &H00, &H00, &HFF, &HFF, &H00, &H00, &H10
9660 DATA &H92, &H00, &H00, &HFF, &HFF, &H00, &H00, &H00, &H92, &H00, &H00, &H00
9670 DATA &H00, &H00, &H00, &H00, &H00, &H00, &H00, &H00, &H00, &H00, &H00, &H00
9680 DATA &H00, &H00, &H00, &H55, &H8B, &HEC, &H8B, &H7E, &H06, &H8A, &H2D, &H8B
9690 DATA &H7E, &H0C, &H8B, &H05, &H8B, &H7E, &H0A, &H8B, &H1D, &HF6, &HC5, &H01
9700 DATA &HE8, &H2E, &H00, &H88, &H54, &H1C, &H89, &H44, &H1A, &H8B, &H7E, &H10
9710 DATA &H8B, &H05, &H8B, &H7E, &H0E, &H8B, &H1D, &HF6, &HC5, &H02, &HE8, &H18
9720 DATA &H00, &H88, &H54, &H14, &H89, &H44, &H12, &HB4, &H87, &H8B, &H7E, &H08
9730 DATA &H8B, &H0D, &HCD, &H15, &H8B, &HE5, &H5D, &HCA, &H0C, &H00, &H5E, &HEB
9740 DATA &HBA, &H8A, &HD4, &HB1, &H04, &HD2, &HEA, &H75, &H03, &H80, &HCA, &H10
9750 DATA &HD3, &HE0, &H03, &HC3, &H73, &H02, &HFE, &HC2, &HC3
```

```
;
;   E G Y M E G A B . A S M
;
;   Funkciója : az 1 MB-os határ feletti tárterület (AT) kezelé-
;               sére szolgáló gépi kódú MOZGAT rutin definiálása
;               a BASIC nyelvű mintaprogram számára.
;
;
;-- KÓD ---------------------------------------------------------------
;
kod     segment

        assume cs:kod, ds:kod, es:kod, ss:kod

;-- MOZGAT : Adatmozgatás 1MB alatti és fölötti tárteruletek között --
;
;-- Hívás GW BASIC-ből : CALL MOZGAT(forrásszegmens, forrásofszet,
;                                    célszegmens,    célofszet,
;                                    szavak száma,   irányparaméter)
;
;           Az irányparaméter lehetséges értékei :
;
;           0 :  forrás 1MB alatt,  cél 1MB alatt
;           1 :  forrás 1MB alatt,  cél 1MB felett
;           2 :  forrás 1MB felett, cél 1MB alatt
;           3 :  forrás 1MB felett, cél 1MB felett

mozgat  proc far

        call glticim            ; A GLT címe.

;-- A globális leíró tábla adatai -------------------------------------

glt     equ this word

        dw 8 dup (0)            ; * Fenntartott terulet 16 bájt.
        dw 0FFFFh               ; Forrásszegmens hossza 64 KB.
forl    dw (?)                  ; Fizikai forráscím alsó 16 bitje.
forh    db (?)                  ; Fizikai forráscím felső 8 bitje.
        db 92h                  ; Hozzáférési jog bájt.
        dw 0h                   ; * Fenntartott terulet 2 bájt.
        dw 0FFFFh               ; Célszegmens hossza 64 KB.
cell    dw (?)                  ; Fizikai célcím alsó 16 bitje.
celh    db (?)                  ; Fizikai célcím felső 8 bitje.
        db 92h                  ; Hozzáférési jog bájt.
        dw 9 dup (0)            ; * Fenntartott terulet 18 bájt.

;-- A MOVE-rutin kodja ------------------------------------------------

mozg:   push bp                 ; BP -> verem.
        mov  bp,sp              ; SP -> BP.

        mov  di,[bp+6]          ; Mozgatás irányparamétere -> CH.
        mov  ch,[di]            ;
        mov  di,[bp+12]         ; Célterulet szegmenscíme -> AX.
        mov  ax,[di]            ;
        mov  di,[bp+10]         ; Célterulet ofszetcíme -> BX.
        mov  bx,[di]            ;
        test ch,1               ; Cél 1MB felett -> zérobit = 1.
        call fizcim             ; 24 bites cím kialakítása, majd
        mov  [si+celh-glt],dl   ;    felső 8 bit -> celh és
        mov  [si+cell-glt],ax   ;    alsó 16 bit -> cell.
```

```
            mov   di,[bp+16]          ; Forrásterület szegmenscíme -> AX.
            mov   ax,[di]             ;
            mov   di,[bp+14]          ; Forrásterület ofszetcíme -> BX.
            mov   bx,[di]             ;
            test  ch,2                ; Forrás 1MB felett -> zérobit = 1.
            call  fizcim              ; 24 bites cím kialakítása, majd
            mov   [si+forh-glt],dl    ;    felsõ 8 bit -> forh és
            mov   [si+forl-glt],ax    ;    alsó 16 bit -> forl.
            mov   ah,087h             ; Áteltolás funkciókódja.
            mov   di,[bp+8]           ; Szavak száma -> CX.
            mov   cx,[di]             ;
            int   15h                 ; 15(h) megszakítás.

            mov   sp,bp               ; SP visszaállítása.
            pop   bp                  ; Verem -> BP.
            ret   12                  ; A verembeli változók már nem kellenek

MOZGAT      endp

;— GLTCIM : A GLT ofszetcímét állítja be SI-ben ————————————————————
;— BE     : -
;— KI     : SI a GLT ofszetcíme
;— REG    : SI megváltozik

GLTCIM      proc near

            pop   si                  ; A GLT címe van a verem tetején.
            jmp   short mozg          ; Ugrás a másolást végzõ rutinra.

GLTCIM      endp

;— FIZCIM : 24 bites fizikai cím képzése ———————————————————————————
;— BE     : AX:BX = átalakítandó szegmentált cím,
;           ha a zérobit értéke 1, a cím 1MB felett található
;— KI     : DL = a cím felsõ 8 bitje
;           BX = a cím alsó 16 bitje
;— REG    : AX, BX, DL, CL és FLAGS értéke megváltozik

FIZCIM      proc near

            mov   dl,ah               ; A szegmenscím (AX) felsõ félbájtja
            mov   cl,4                ; DL alsó félbájtjába kerül.
            shr   dl,cl               ;

            jne   cim1                ; Ha nem 1MB alatti tartomány,
            or    dl,010h             ; DL 4. bitje is bebillen (10h).

cim1:       shl   ax,cl               ; Szegmenscím * 16 + ofszetcím
            add   ax,bx               ; képzése.
            jnc   cim2                ;

            inc   dl                  ; Ha volt átvitel, DL 1-gyel nõ.

cim2:       ret                       ; Vissza a hívóhoz.

FIZCIM      endp

;— VÉGE ———————————————————————————————————————————————————————————

kod         ends
            end
```

```
/*  ┌──────────────────────────────────────────────────────────────┐  */
/*  │                                                              │  */
/*  │                        E G Y M E G A . C                     │  */
/*  │                                                              │  */
/*  │  Funkciója : az 1 MB-os határ feletti tárterület (AT) kezelé-│  */
/*  │              sének szemléltetése a video RAM "mozgatásának"  │  */
/*  │              bemutatásával.                                  │  */
/*  │                                                              │  */
/*  │   FONTOS   : a memóriakezelő "PEEKB" a PEEKPOKE.ASM forrásál-│  */
/*  │              lományban, a mozgatást végző "MOZGAT" pedig az  │  */
/*  │              EGYMEGAC.ASM forrásállományban helyezkedik el,  │  */
/*  │              ezért a LINK parancs helyes formája a következő:│  */
/*  │                                                              │  */
/*  │              LINK EGYMEGA EGYMEGAC PEEKPOKE                  │  */
/*  │                                                              │  */
/*  └──────────────────────────────────────────────────────────────┘  */

extern short int PEEKB();                   /* Mindkét assembler rutin */
extern void MOZGAT();                       /* beszerkesztendő.        */

#include <dos.h>
#include <io.h>
#include <conio.h>

#define  NM    0x07                         /* Normál megjelenítés.    */

#define byte unsigned char

/*  ┌──────────────────────────────────────────────────┐  */
/*  │ AKTOLDAL : AZ AKTUALIS KÉPERNYŐOLDAL MEG-        │  */
/*  │            HATÁROZÁSA                            │  */
/*  └──────────────────────────────────────────────────┘  */

byte AKTOLDAL()

{
union REGS regiszter;                       /* Regiszter union deklaráció. */

regiszter.h.ah = 0x0F;                      /* Video olvasás funkció.  */
int86(0x10, &regiszter, &regiszter);        /* 10(h) megszakítás.      */
return(regiszter.h.bh);                     /* Aktuális képernyőoldal. */
}

/*  ┌──────────────────────────────────────────────────┐  */
/*  │ KURPOZ : A KURZOR POZÍCIONÁLÁSA A KIVÁ-          │  */
/*  │          LASZTOTT KÉPERNYŐOLDALON                │  */
/*  └──────────────────────────────────────────────────┘  */

void KURPOZ(oldal, oszlop, sor)
int oldal;
int oszlop;
int sor;

{
union REGS regiszter;                       /* Regiszter union deklaráció. */

regiszter.h.ah = 2;                         /* Kurzor beállítása az    */
regiszter.h.bh = oldal;                     /* aktuális oldalon az     */
regiszter.h.dh = sor;                       /* átadott sor- és osz-    */
regiszter.h.dl = oszlop;                    /* lopozícióra.            */
int86(0x10, &regiszter, &regiszter);        /* 10(h) megszakítás.      */
}
```

```
/*
/*  GORGETES_FEL : MEGADOTT KÉPERNYŐTARTOMÁNY      */
/*          FELFELÉ GÖRGETÉSE, ill. TÖRLÉSE        */
/*                                                 */

void GORGETES_FEL(gsor, szin, bfo, bfs, jao, jas)

int gsor;           /* (G)örgetendő (sor)ok száma.                    */
int szin;           /* A törölt terület (szín)e.                      */
int bfs ;           /* Képernyőtartomány (b)al  (f)első (s)ora.       */
int bfo ;           /*                   (b)al  (f)első (o)szlopa.    */
int jas ;           /*                   (b)al  (a)lsó  (s)ora.       */
int jao ;           /*                   (j)obb (a)lsó  (o)szlopa.    */

{
union REGS regiszter;                   /* Regiszter union deklaráció. */

regiszter.h.ah = 0x06;                  /* Görgetés funkció.           */
regiszter.h.al = gsor;                  /* Görgetendő sorok száma.     */
regiszter.h.bh = szin;                  /* Az üres sorok színe.        */
regiszter.h.ch = bfs ;                  /* Ablak-                      */
regiszter.h.cl = bfo ;                  /*        koor-                */
regiszter.h.dh = jas ;                  /*              dinátáK.       */
regiszter.h.dl = jao ;                  /*                             */

int86(0x10, &regiszter, &regiszter);          /* 10(h) megszakítás.    */
}

/*
/*  CLRSCR : KÉPERNYŐ TÖRLÉSE                      */
/*                                                 */

void CLRSCR()

{
GORGETES_FEL(0, NM, 0, 0, 79, 24);      /* Törlés görgetéssel.         */
KURPOZ(AKTOLDAL(), 0, 0);               /* Kurzor "home" pozícióba.    */
}

/*
/*  VIDSZEG : VIDEO RAM SZEGMENSCÍMÉNEK            */
/*            MEGALLAPÍTÁSA                        */
/*                                                 */

unsigned int VIDSZEG()

{
union REGS regiszter;                   /* Regiszter union deklaráció. */

regiszter.h.ah = 0x0F;                        /* Video olvasás funkció.*/
int86(0x10, &regiszter, &regiszter);          /* 10(h) megszakítás.    */

return((regiszter.h.al == 7) ? 0xB000 : 0xB800);  /* Ha AL 7, monó,    */
}                                                 /* másként színes    */

/*
/*                  F Ő P R O G R A M              */
/*                                                 */

void main()
```

```
{
int i;
printf("");
printf("Az 1 Mbájt feletti tárterület elérését ill. kezelését");
printf("szemlélteti a tesztprogram. A jelenlegi képernyőtartalom");
printf("(a video RAM tartalma) elmentődik az 1 Mbájtos tártarto-");
printf("mány legelejére, majd felulíródik. A teszt folytatása so-");
printf("rán visszamentődik az eredeti video RAM tartalom a kép-");
printf("ernyőre.");
printf("A teszt egy billentyű leütésére indul ...");
getch();
if (PEEKB(0xF000, 0xFFFE) != 0xFC)
   printf("A tesztprogram csak AT-n futtatható !");
else
   {
   MOZGAT(VIDSZEG(), 0x0000, 0x0000, 0x0000, 0x2000, 1);

   CLRSCR();                                        /* Video RAM    */
   KURPOZ(AKTOLDAL(), 0, 0);                        /* felulírása.  */
   for (i=0;i<880;i++)
      printf("%c",1);
   KURPOZ(AKTOLDAL(), 0,14);
   for (i=1;i<880;i++)
      printf("%c",2);
   KURPOZ(AKTOLDAL(), 0,12);
   printf("Eredeti képernyő elmentve 1 MB folé - ");
   printf("üsson le egy billentyűt visszatoltéséhez..");
   getch();

   MOZGAT(0x0000, 0x0000, VIDSZEG(), 0x0000, 0x2000, 2);
   KURPOZ(AKTOLDAL(), 0,24);
   }
}
```

```
;
;
;                          E G Y M E G A C . A S M
;
;   Funkciója : az 1 MB-os határ feletti tárterület (AT) kezelé-
;               sére szolgáló MOZGAT eljárás definiálása az MSC
;               mintaprogram számára.
;
;
;
;— DEFINíCIóK ——————————————————————————————————————————————

IGROUP  group _TEXT                     ; Programszegmens.
DGROUP  group CONST, _BSS, _DATA        ; Adatszegmens.

assume  CS:IGROUP, DS:DGROUP, ES:DGROUP, SS:DGROUP

        public _MOZGAT                  ; A MOZGAT eljárás kívülről
                                        ; is hívható.

CONST   segment word public 'CONST'     ; Csak olvasható konstansok.
CONST   ends

_BSS    segment word public 'BSS'       ; Nem inicializált statikus
_BSS    ends                            ; változók szegmense.

_DATA   segment word public 'DATA'      ; Inicializált globális és
                                        ; statikus változók szegmense.
;— A globális leíró tábla adatai ——————————————————————————————

GLT     equ this word

        dw 8 dup (0)            ; * Fenntartott terület 16 bájt.
        dw 0FFFFh               ; Forrásszegmens hossza 64 KB.
forl    dw (?)                  ; Fizikai forráscím alsó 16 bitje.
forh    db (?)                  ; Fizikai forráscím felső 8 bitje.
        db 92h                  ; Hozzáférési jog bájt.
        dw 0h                   ; * Fenntartott terület 2 bájt.
        dw 0FFFFh               ; Célszegmens hossza 64 KB.
cell    dw (?)                  ; Fizikai célcím alsó 16 bitje.
celh    db (?)                  ; Fizikai célcím felső 8 bitje.
        db 92h                  ; Hozzáférési jog bájt.
        dw 9 dup (0)            ; * Fenntartott terület 18 bájt.

_DATA   ends

;— KÓD ————————————————————————————————————————————————————

_TEXT   segment byte public 'CODE'      ; Program (kód) szegmens.

;— MOZGAT : Adatmozgatás 1MB alatti és fölötti tárterületek között —
;
;— Hívás MSC-ből : MOZGAT(forásszegmens, forrásofszet,
;                         célszegmens,    célofszet,
;                         szavak száma,   irányparaméter)
;
;           Az irányparaméter lehetséges értékei :
;
;           0 :  forrás 1MB alatt,  cél 1MB alatt
;           1 :  forrás 1MB alatt,  cél 1MB felett
;           2 :  forrás 1MB felett, cél 1MB alatt
;           3 :  forrás 1MB felett, cél 1MB felett
```

```
_MOZGAT  proc near

         push bp                    ; BP -> verem.
         mov  bp,sp                 ; SP -> BP.
         push si                    ; SI -> verem.

         mov  ch,[bp+14]            ; Mozgatás irányparamétere -> CH.
         mov  ax,[bp+8]             ; Célterület szegmenscíme  -> AX.
         mov  bx,[bp+10]            ; Célterület ofszetcíme    -> BX.
         test ch,1                  ; Cél 1MB felett -> zérobit = 1.
         call fizcim                ; 24 bites cím kialakítása, majd
         mov  celh,dl               ;    felső 8 bit -> celh és
         mov  cell,ax               ;    alsó 16 bit -> cell.
         mov  ax,[bp+4]             ; Forrásterület szegmenscíme -> AX.
         mov  bx,[bp+6]             ; Forrásterület ofszetcíme -> BX.
         test ch,2                  ; Forrás 1MB felett -> zérobit = 1.
         call fizcim                ; 24 bites cím kialakítása, majd
         mov  forh,dl               ;    felső 8 bit -> forh és
         mov  forl,ax               ;    alsó 16 bit -> forl.
         mov  ah,087h               ; Áthelyezés funkciókódja.
         push ds                    ;
         pop  es                    ; DS -> ES.
         mov  cx,[bp+12]            ; Szavak száma -> CX.

         mov  si,offset DGROUP:GLT      ; Glob. leíró tábla ofszet-> SI.

         int  15h                   ; 15(h) megszakítás.

         pop  si                    ; Verem -> SI.
         mov  sp,bp                 ; SP visszaállítása.
         pop  bp                    ; Verem -> BP.
         ret                        ; Visszatérés az MSC programba.

_MOZGAT  endp

;— FIZCIM : 24 bites fizikai cím képzése ————————————————————————
;— BE     : AX:BX = átalakítandó szegmentált cím,
;           ha a zérobit értéke 1, a cím 1MB felett található
;— KI     : DL = a cím felső 8 bitje
;           BX = a cím alsó 16 bitje
;— REG    : AX, BX, DL, CL és FLAGS értéke megváltozik

FIZCIM   proc near

         mov  dl,ah                 ; A szegmenscím (AX) felső félbájtja
         mov  cl,4                  ; DL alsó félbájtjába kerül.
         shr  dl,cl                 ;

         jne  cim1                  ; Ha nem 1MB alatti tartomány,
         or   dl,010h               ; DL 4. bitje is bebillen (10h).

cim1:    shl  ax,cl                 ; Szegmenscím * 16 + ofszetcím
         add  ax,bx                 ; képzése.
         jnc  cim2                  ;

         inc  dl                    ; Ha volt átvitel, DL 1-gyel nő.

cim2:    ret                        ; Vissza a hívóhoz.

FIZCIM   endp

;— VÉGE ————————————————————————————————————————————————————————

_TEXT    ends
         end
```

```
{
{  ┌──────────────────────────────────────────────────────────┐
{  │                                                          │
{  │                      E G Y M E G A . P A S               │
{  │                                                          │
{  │  Funkciója : az 1 MB-os határ feletti tárterület (AT) kezelé-
{  │              sének szemléltetése a video RAM "mozgatásának"
{  │              bemutatásával.                              │
{  │                                                          │
{  └──────────────────────────────────────────────────────────┘
{

program EGYMEGA;

type dosreg = record
              ax, bx, cx, dx, bp, di, si, ds, es, flags : integer;
              end;

var z : char;                { Csak a várakozási állapotokhoz szükséges. }
    i : integer;

{
{  ┌──────────────────────────────────────────┐
{  │ VIDSZEG : VIDEO RAM SZEGMENSCÍMÉNEK      │
{  │           MEGALLAPíTASA                  │
{  └──────────────────────────────────────────┘
{

function VIDSZEG : integer;

var regiszter : dosreg;                  { Regiszter rekord deklaráció. }

begin
   regiszter.ax := $0F shl 8;                    { Video olvasás funkció. }
   intr($10, regiszter);                         { 10(h) megszakítás.     }

   if lo(regiszter.ax) = 7 then VIDSZEG := $B000     { Ha AL 7, monó, }
                           else VIDSZEG := $B800;    { másként színes.}
end;

{
{  ┌──────────────────────────────────────────────────┐
{  │ MOZGAT : INLINE KÓDÚ ELJARAS                    │
{  │                                                  │
{  │ TARTERULET ELMOZGATASA (MASOLASA)                │
{  │ BE : FORS  = Forrásterület szegmenscím           │
{  │      FORO  = Forrásterület ofszetcím             │
{  │      CELS  = Célterület szegmenscím              │
{  │      CELO  = Célterület ofszetcím                │
{  │      NSZO  = Átmásolandó szavak száma            │
{  │      IRANY = Eltolási iránypараméter :           │
{  │              0 : 1 MB alól  1 MB alá             │
{  │              1 : 1 MB alól  1 MB felé            │
{  │              2 : 1 MB fölül 1 MB alá             │
{  │              3 : 1 MB fölül 1 MB felé            │
{  └──────────────────────────────────────────────────┘
{

procedure MOZGAT(fors, foro, cels, celo, nszo, irany : integer);
```

```
begin
   inline(
      $55/$8B/$EC/$8B/$7E/$12/$8B/$76/$10/$8B/$46/$0E/$8E/$C0/$8B/$5E/
      $0C/$8B/$46/$0A/$8B/$4E/$08/$8A/$E9/$E8/$5E/$00/$00/$00/$00/$00/
      $00/$00/$00/$00/$00/$00/$00/$00/$00/$00/$00/$00/$FF/$FF/$00/$00/
      $00/$92/$00/$00/$FF/$FF/$00/$00/$00/$92/$00/$00/$00/$00/$00/$00/
      $00/$00/$00/$00/$00/$00/$00/$00/$00/$00/$00/$00/$50/$8C/$C0/$F6/
      $C5/$01/$E8/$28/$00/$2E/$88/$56/$1C/$2E/$89/$46/$1A/$8B/$C7/$8B/
      $DE/$F6/$C5/$02/$E8/$16/$00/$2E/$88/$56/$14/$2E/$89/$46/$12/$B4/
      $87/$0E/$07/$59/$8B/$F5/$CD/$15/$EB/$17/$5D/$EB/$CF/$8A/$D4/$B1/
      $04/$D2/$EA/$75/$03/$80/$CA/$10/$D3/$E0/$03/$C3/$73/$02/$FE/$C2/
      $C3/$5D
      );
end;

{ ┌──────────────────────────────────────────┐ }
{ │              F Ő P R O G R A M           │ }
{ └──────────────────────────────────────────┘ }

begin
   writeln;
   writeln('Az 1  Mbájt feletti tárterület elérését, ill. kezelését');
   writeln('szemlélteti a tesztprogram. A jelenlegi képernyőtartalom');
   writeln('(a videoRAM tartalma) elmentődik az 1 Mbájtos tártarto-');
   writeln('mány legelejére, majd  felülíródik. A teszt  folytatása');
   writeln('során  visszamentődik  az eredeti  video RAM tartalom a');
   writeln('képernyőre.');
   writeln;
   writeln('A teszt egy billentyű leütésére indul ...');
   writeln;
   read(Kbd, z);
   if mem[$F000:$FFFE] <> $FC
      then writeln('A tesztprogram csak AT-n futtatható !')
   else
      begin
         MOZGAT(VIDSZEG, $0000, $0000, $0000, $2000, $1);

         clrscr;
         gotoxy(1, 1);                                  { Video RAM   }
         for i := 1 to 880 do write(chr(1));            { felülírása. }
         gotoxy(1,15);
         for i := 1 to 879 do write(chr(2));
         gotoxy(1,13);
         write('Eredeti képernyő elmentve 1 MB fölé - ');
         write('üssön le egy billentyűt visszatöltéséhez..');
         read(Kbd, z);

         MOZGAT($0000, $0000, VIDSZEG, $0000, $2000, $2);
         gotoxy(1,25);
      end;
end.
```

```
;    ┌──────────────────────────────────────────────────────────────┐   ;
;    │                                                              │   ;
;    │                    E G Y M E G A P . A S M                   │   ;
;    │                                                              │   ;
;    │  Funkciója : az 1 MB-os határ feletti tárterület (AT) kezelé-│   ;
;    │              sének szemléltetése a video RAM "mozgatásának"  │   ;
;    │              bemutatásával.                                  │   ;
;    │                                                              │   ;
;    └──────────────────────────────────────────────────────────────┘   ;

;— KÓD ———————————————————————————————————————————————————————————————————

kod     segment para 'CODE'        ; A kódszegmens definiálása.

        org 100h                   ; A program a PSP után álló első
                                   ; címen kezdődik.

        assume cs:kod, ds:kod, es:kod, ss:kod

;— PROGRAM ———————————————————————————————————————————————————————————————

PROG    proc near

        push bp                    ; BP -> verem.
        mov  bp, sp                ; SP -> BP.
        mov  di, [bp+18]           ; Forrás szegmenscíme a veremről.
        mov  si, [bp+16]           ; Forrás ofszetcíme a veremről.
        mov  ax, [bp+14]           ; Cél szegmenscíme a veremről.
        mov  es, ax
        mov  bx, [bp+12]           ; Cél ofszetcíme a veremről.
        mov  ax, [bp+10]           ; Átviendő szavak száma a veremről.
        mov  cx, [bp+ 8]           ; Az átvitel iránya a veremről.
        mov  ch, cl
        call gltcim                ; A GLT címe.

;— A globális leíró tábla adatai ——————————————————————————————————————

GLT     equ this word

        dw 8 dup (0)               ; * Fenntartott terület 16 bájt.
        dw 0FFFFh                  ; Forrásszegmens hossza 64 KB.
forl    dw (?)                     ; Fizikai forráscím alsó 16 bit.
forh    db (?)                     ; Fizikai forráscím felső 8 bit.
        db 92h                     ; Hozzáférési jog bájt.
        dw 0h                      ; * Fenntartott terület 2 bájt.
        dw 0FFFFh                  ; Célszegmens hossza 64 KB.
cell    dw (?)                     ; Fizikai célcím alsó 16 bitje.
celh    db (?)                     ; Fizikai célcím felső 8 bitje.
        db 92h                     ; Hozzáférési jog bájt.
        dw 9 dup (0)               ; * Fenntartott terület 18 bájt.

;— MOZGAT : Adatmozgatás 1MB alatti és fölötti tárterületek között —
;— BE     : DI:SI = A forrásterület szegmens-/ofszetcíme
;           ES:BX = A célterület szegmens-/ofszetcíme
;           AX    = A mozgatandó szavak száma (max. 8000H)
;           CH    = A mozgatást a legalsó két bit értéke definiálja
;
;                            0. bit 0               0. bit 1
;
;             1. bit 0    forrás 1MB alatt       forrás 1MB alatt
;                         cél 1MB alatt          cél 1MB felett
;
```

```
;                       1. bit 1    forrás 1MB felett     forrás 1MB felett
;                                   cél 1MB alatt         cél 1MB felett
;
;— KI    : Ha az átvitelbit 1 : hiba, melyet AH tartalma azonosít

;— REG   : AX, BX, DL, CL, SI, ES és FLAGS értéke megváltozik

mozgat:  push ax                    ; Szavak száma -> verem.
         mov  ax,es                 ; Célterület szegmenscíme -> AX.
         test ch,1                  ; Cél 1MB felett -> zérobit = 1.
         call fizcim                ; 24 bites cím kialakítása, majd
         mov  cs:[bp+28],dl         ;    felső 8 bit -> celh és
         mov  cs:[bp+26],ax         ;    alsó 16 bit -> cell.
         mov  ax,di                 ; Forrásterület szegmenscíme -> AX.
         mov  bx,si                 ; Forrásterület ofszetcíme   -> BX.
         test ch,2                  ; Forrás 1MB felett -> zérobit = 1.
         call fizcim                ; 24 bites cím kialakítása, majd
         mov  cs:[bp+20],dl         ;    felső 8 bit -> forh és
         mov  cs:[bp+18],ax         ;    alsó 16 bit -> forl.
         mov  ah,087h               ; Táreltolás funkciókódja.
         push cs
         pop  es                    ; DS -> ES.
         pop  cx                    ; Szavak száma a veremből -> CX.
         mov  si,bp                 ; Glob. leíró tábla ofszet -> SI.
         int  15h                   ; 15(h) megszakítás.
         jmp short vege

PROG     endp

;— GLTCIM : A GLT ofszetcímét állítja be BP-ben ————————————————
;— BE     : -
;— KI     : BP a GLT ofszetcíme
;— REG    : BP megváltozik

GLTCIM   proc near

         pop  bp                    ; A GLT címe van a verem tetején.
         jmp  short mozgat          ; Ugrás a másolást végző rutinra.

GLTCIM   endp

;— FIZCIM : 24 bites fizikai cím képzése ———————————————————————
;— BE     : AX:BX = átalakítandó szegmentált cím,
;             ha a zérobit értéke 1, a cím 1MB felett található
;— KI     : DL = a cím felső 8 bitje
;           BX = a cím alsó 16 bitje
;— REG    : AX, BX, DL, CL és FLAGS értéke megváltozik

FIZCIM   proc near

         mov  dl,ah                 ; A szegmenscím (AX) felső félbájtja
         mov  cl,4                  ; DL alsó félbájtjába kerül.
         shr  dl,cl                 ;

         jne  cimi                  ; Ha nem 1MB alatti tartomány,
         or   dl,010h               ; DL 4. bitje is bebillen (10h).

cimi:    shl  ax,cl                 ; Szegmenscím * 16 + ofszetcím
         add  ax,bx                 ; képzése.
         jnc  cim2                  ;

         inc  dl                    ; Ha volt átvitel, DL 1-gyel nő.
```

```
cim2:   ret                     ; Vissza a hivóhoz.

FIZCIM  endp

vege    label near              ; A forrásprogram vége.
        pop bp                  ; Verem -> BP.

;— VÉGE ————————————————————————————————————————————————————————————

kod     ends
        end  PROG
```

```
;
;   ┌──────────────────────────────────────────────────────────┐
;   │                                                          │
;   │                      E G Y M E G A . A S M               │
;   │                                                          │
;   │  Funkciója : az 1 MB-os határ feletti tárterület (AT) kezelé-
;   │              sének szemléltetése a video RAM "mozgatásának"
;   │              bemutatásával.                              │
;   │                                                          │
;   └──────────────────────────────────────────────────────────┘
;

;— BIOS ————————————————————————————————————————————————————

bios    segment at 0F000h       ; Az F000:FFFE bájt alapján
                                ; határozható meg a géptípus.
        org 0FFFEh

tipus   equ this byte

bios    ends

;— KÓD —————————————————————————————————————————————————————

kod     segment para 'CODE'     ; A kódszegmens definiálása.

        org 100h                ; A program a PSP után álló első
                                ; címen kezdődik.

        assume cs:kod, ds:kod, es:bios, ss:kod

;— PROGRAM ——————————————————————————————————————————————————

TESZT   proc near

        mov  dx,offset indit    ; Az indítási üzenet címe.
        mov  ah,9               ; Karakterlánc kiírás funkciókódja.
        int  21h                ; 21(h) megszakítás.

        xor  ah,ah              ; Várakozás billentyű megnyomására.
        int  16h                ; 16(h) megszakítás.

        mov  ax,0F000h          ; BIOS szegmenscíme -> ES.
        mov  es,ax              ;
        cmp  es:tipus,0FCh      ; Ha a F000:FFFE bájt tartalma 0FCh,
        je   tovabb             ; a gép AT, ha nem, akkor nem ren-
                                ; delkezhet 1 MB feletti tárterület-
        mov  dx,offset nem_at   ; tel, ekkor hibaüzenettel kiszállás.
        jmp  short nemat

        ;— Eredeti video RAM tartalom eltolása 1 MB fölé ———————————

tovabb: call vidszeg
        mov  di,ax              ; Video RAM  szegmenscím     -> DI.
        xor  si,si              ;            ofszetcím   = 0 -> SI.
        xor  bx,bx              ; Célterület ofszetcím   = 0 -> BX.
        mov  es,bx              ;            szegmenscím = 0 -> ES.
        mov  ch,1               ; A cél 1 MB fölött lesz, a forrás
                                ; pedig 1 MB alatt van -> CH.
        mov  ax,2000            ; 2000 szó másolandó -> AX.

        call mozgat             ; Video RAM másolása.
        jc   hiba               ; Ha hiba történt : vége.
```

```
        ;— A video RAM feltoltése (eredeti tartalom torlése) ————————

        call vidszeg
        mov  es,ax              ; Video RAM szegmenscím -> ES.
        xor  di,di              ; Video RAM ofszetcím : 0 -> DI.
        mov  cx,880             ; 11 sor (880 szó) inicializálandó
        mov  ax,0701h           ; a 01-es ASCII karakterrel (a kép-
        rep  stosw              ; ernyő felső 11 sora).

        mov  di,2240            ; Video RAM ofszetcím : 2240 -> DI.
        mov  cx,880             ; 11 sor (880 szó) inicializálandó
        mov  ax,0702h           ; a 02-es ASCII karakterrel (a kép-
        rep  stosw              ; ernyő alsó 11 sora).

        ;— Uzenet megjelenítés - billentyűleütésre várakozás ————————

        mov  ah,2               ; Kurzor pozicionálás funkció -> AH.
        xor  bh,bh              ;       Képernyőoldal  -> BH.
        mov  dh,11              ;       Képernyősor    -> DH.
        xor  dl,dl              ;       Képernyőoszlop -> DL.
        int  10h                ; 10(h) megszakítás.

        mov  dx,offset uzenet   ; Az üzenet címe.
        mov  ah,9               ; Karakterlánc kiírás funkciókódja.
        int  21h                ; 21(h) megszakítás.

        mov  ah,2               ; Kurzor pozicionálás funkció -> AH.
        xor  bh,bh              ;       Képernyőoldal  -> BH.
        mov  dh,24              ;       Képernyősor    -> DH.
        xor  dl,dl              ;       Képernyőoszlop -> DL.
        int  10h                ; 10(h) megszakítás.

        xor  ah,ah              ; Várakozás billentyű megnyomására.
        int  16h                ; 16(h) megszakítás.

        ;— Eredeti video RAM tartalom visszamásolása 1 MB felul ——————

        xor  di,di              ; Forrás szegmenscím   : 0 -> DI.
        xor  si,si              ;        ofszetcím     : 0 -> SI.
        xor  bx,bx              ; Célterület ofszetcím : 0 -> BX.
        mov  ch,2               ; A cél 1 MB alatt lesz, a forrás
                                ; pedig 1 MB felett van -> CH.
        mov  ax,2000            ; 2000 szó másolandó -> AX.

        call mozgat             ; Tárterület mozgatása.
        jc   hiba               ; Ha hiba történt : vége.

        mov  ax,4C00h           ; Befejezés 0-ás hibakóddal.
        int  21h                ; 21(h) megszakítás.

hiba:   mov  dx,offset hibauz   ; A hibauzenet címe.
nemat:  mov  ah,9               ; Karakterlánc kiírás funkciókódja.
        int  21h                ; 21(h) megszakítás.

        mov  ax,4C01h           ; Befejezés 1-es hibakóddal.
        int  21h                ; 21(h) megszakítás.

TESZT   endp

;— VIDSZEG : A video RAM szegmenscímének meghatározása ——————————
;— BE      : -
;— KI      : AX = a video RAM szegmenscíme
;— REG     : AX, BH és FLAGS értéke megváltozik
```

```
VIDSZEG  proc near

         mov  ah,0FH              ; Video üzemmód beolv. funkció,
         int  10h                 ; 10(h) BIOS megszakítás.
         cmp  al,7                ; Ha a visszakapott AL érték 7,
         je   mono                ; akkor monokróm, ellenkező esetben
                                  ; színes videokártya.

         mov  ax,0B800h           ; A színes video RAM szegmenscíme
         ret                      ; B8000(h).

mono:    mov  ax,0B000h           ; A monokróm video RAM szegmenscíme
         ret                      ; B000(h).

VIDSZEG  endp

;— MOZGAT : Adatmozgatás 1MB alatti és folotti tárterületek között —
;— BE     : DI:SI = A forrásterület szegmens-/ofszetcíme
;           ES:BX = A célterület szegmens-/ofszetcíme
;           AX    = A mozgatandó szavak száma (max. 8000H)
;           CH    = A mozgatást a legalsó két bit értéke definiálja
;
;                                  0. bit 0          0. bit 1
;
;                1. bit 0    forrás 1MB alatt   forrás 1MB alatt
;                            cél 1MB alatt      cél 1MB felett
;
;                1. bit 1    forrás 1MB felett  forrás 1MB felett
;                            cél 1MB alatt      cél 1MB felett
;
;— KI   : Ha az átvitelbit 1 : hiba, melyet AH tartalma azonosít
;— REG  : AX, BX, DL, CL, SI, ES és FLAGS értéke megváltozik

MOZGAT   proc near

         push ax                  ; Szavak száma -> verem.
         mov  ax,es               ; Célterület szegmenscíme -> AX.
         test ch,1                ; Cél 1MB felett -> zéróbit = 1.
         call fizcim              ; 24 bites cím kialakítása, majd
         mov  celh,dl             ;    felső 8 bit -> celh és
         mov  cell,ax             ;    alsó 16 bit -> cell.
         mov  ax,di               ; Forrásterület szegmenscíme -> AX.
         mov  bx,si               ; Forrásterület ofszetcíme -> BX.
         test ch,2                ; Forrás 1MB felett -> zéróbit = 1.
         call fizcim              ; 24 bites cím kialakítása, majd
         mov  forh,dl             ;    felső 8 bit -> forh és
         mov  forl,ax             ;    alsó 16 bit -> forl.
         mov  ah,087h             ; Táreltolás funkciókódja.
         push ds
         pop  es                  ; DS -> ES.
         pop  cx                  ; Szavak száma a veremből -> CX.
         mov  si,offset GLT       ; Glob. leíró tábla ofszet -> SI.
         int  15h                 ; 15(h) megszakítás.
         ret

;— A globális leíró tábla adatai ————————————————————————————

GLT      equ this word

         dw 8 dup (0)             ; * Fenntartott terület 16 bájt.
         dw 0FFFFh                ; Forrásszegmens hossza 64 KB.
forl     dw (?)                   ; Fizikai forráscím alsó 16 bitje.
```

```
forh    db (?)                  ; Fizikai forráscím felső 8 bitje.
        db 92h                  ; Hozzáférési jog bájt.
        dw 0h                   ; * Fenntartott terület 2 bájt.
        dw 0FFFFh               ; Célszegmens hossza 64 KB.
cel1    dw (?)                  ; Fizikai célcím alsó 16 bitje.
celh    db (?)                  ; Fizikai célcím felső 8 bitje.
        db 92h                  ; Hozzáférési jog bájt.
        dw 9 dup (0)            ; * Fenntartott terület 18 bájt.

MOZGAT  endp

;— FIZCIM : 24 bites fizikai cím képzése ————————————————————————
;— BE     : AX:BX = átalakítandó szegmentált cím,
;           ha a zérőbit értéke 1, a cím 1MB felett található
;— KI     : DL = a cím felső 8 bitje
;           BX = a cím alsó 16 bitje
;— REG    : AX, BX, DL, CL és FLAGS értéke megváltozik

FIZCIM  proc near

        mov  dl,ah              ; A szegmenscím (AX) felső félbájtja
        mov  cl,4               ; DL alsó félbájtjába kerül.
        shr  dl,cl              ;

        jne  cim1               ; Ha nem 1MB alatti tartomány,
        or   dl,010h            ; DL 4. bitje is bebillen (10h).

cim1:   shl  ax,cl              ; Szegmenscím * 16 + ofszetcím
        add  ax,bx              ; képzése.
        jnc  cim2               ;

        inc  dl                 ; Ha volt átvitel, DL 1-gyel nő.

cim2:   ret                     ; Vissza a hívóhoz.

FIZCIM  endp

;— ADATOK ———————————————————————————————————————————————————————

indit   db 13,10
        db "Az 1 Mbájt feletti tárterület elérését, ill. ",13,10
        db "Kezelését szemlélteti a  tesztprogram. A je-",13,10
        db "lenlegi  képernyőtartalom (a video RAM  tar-",13,10
        db "talma) elmentődik  az 1 Mbájtos tártartomány",13,10
        db "legelejére, majd felülíródik. A teszt  foly-",13,10
        db "tatása során visszamentődik az eredeti video",13,10
        db "RAM tartalom a képernyőre."                  ,13,10,13,10
        db "A teszt egy billentyű leütésére indul ..."   ,13,10,"$"

uzenet  db 80 dup (32)
        db "Eredeti képernyő elmentve 1 MB felé - "
        db "usson le egy billentyűt visszatoltéséhez.."
        db 80 dup (32)
        db  "$"

hibauz  db "Hiba az 1Mbájt feletti adatok kezelésében !",13,10,"$"

nem_at  db "A tesztprogram csak AT-n futtatható !",13,10,"$"

;— VÉGE ——————————————————————————————————————————————————————————

kod     ends
        end  TESZT
```

1.8. A billentyűzet kezelése

A számítógép és kezelője közötti kapcsolatban a képernyő mellett a billentyűzet játssza a legfontosabb szerepet. Alapértelmezés szerint ez az az elsődleges beviteli (standard input) eszköz, amelyen keresztül kívánságainkat (programutasításokat) a géppel közöljük, és az adatokat a program rendelkezésére bocsátjuk.

A billentyűzet önmagában is egy intelligens, aktív áramköri egység, amelynek saját, beépített mikroprocesszora van. (az XT-ben pl. Intel 8048, az AT-ben pl. Intel 8042). A billentyűzet működése nagy vonalakban így foglalható össze: Az XT billentyűzet mikroprocesszora folyamatosan figyeli a billentyűzet állapotát, és azonnal érzékeli, ha a kezelő valamelyik billentyűt lenyomja, ill. felengedi. Amikor egy ilyen eseményt észlel, az ennek megfelelő kódot a kimeneti pufferébe írja, és ha teheti, átküldi az XT alapkártyájának. Az átküldés kivált egy hardver megszakítást (a 9. számú billentyűzet megszakítást) jelezve, hogy igényt tart a központi egység figyelmére. Amikor a rendszer a megszakítást lekezeli, az átküldött kódot a számítógép alapkártyáján lévő, programozható perifériakezelő áramkör (PPI) egyik – 60(h) című – portján éri el. (A billentyű lenyomásakor magát a kódot, felengedésekor a kód 80(h)-val megnövelt értékét.) A számítógép megfelelő ROM BIOS rutinjai a porton lévő kódot elolvassák, értelmezik, majd ugyanezen a porton keresztül törlik a küldött kódot.

A következőkben ismerkedjünk meg a billentyűkkel és a hozzájuk tartozó billentyűkódokkal.

1.8.1. Billentyűk és billentyűkódok

Amikor karakterkódokról beszélünk, akkor a legtöbb esetben ezeken az illető karakterekhez rendelt, szabványban rögzített ASCII-kódokat értjük. Más a helyzet azonban akkor, ha a billentyűzet egyes billentyűihez tartozó kódokról van szó. A billentyűzetben lévő mikroprocesszor ugyanis az egyes billentyűket nem aszerint azonosítja, hogy az illető billentyűhöz milyen karakter tartozik (hiszen számos olyan billentyű van – mint pl. a kurzorbillentyűk –, amelyekhez semmilyen karakter sem tartozik), hanem másfajta kódolási rendszert használ. Ez a rendszer – amely alapvetően az egyes billentyűcsoportoknak a billentyűzeten belüli elhelyezkedéséhez igazodik – minden egyes billentyűhöz egy-egy, az illető billentyű sorszámának tekinthető billentyűkódot rendel. Ezeket a billentyűkódokat – amelyeknek semmi közük az ASCII kódokhoz –, az angol nyelvű szakirodalom a billentyűk scan-kódjának nevezi. Mivel az XT és az AT gépek billentyűzetén a billentyűk elhelyezkedése nem teljesen azonos, a kétféle billentyűzet billentyűkódjai is különbözők. Az XT 83 billentyűjét az 1 – 83 billentyűkódok, az AT 84 billentyűjét az 1 – 108 billentyűkódok azonosítják (az utóbbi esetében ezek nem folyamatos sorszámok – lásd a III. kötetben).

A billentyűzet hardver megszakításának rutinjai egy billentyű lenyomásakor ezt a billentyűkódot beírják egy pufferbe, ahonnan a ROM BIOS rutinok ezt ki tudják olvasni.

Mivel minden egyes billentyűhöz csak egy, meghatározott billentyűkód tartozik, ez a billentyűkód önmagában csak azt képes jelezni, hogy egy, meghatározott billentyű le lett nyomva. Ugyanakkor tudjuk, hogy számos billentyűkombináció létezik, amelyek különböző karakterek beírását (mint pl. nagybetűk) teszik lehetővé, ill. amelyekkel különböző vezérlési feladatok végezhetők el (képernyő törlése, kurzor bal felső sarokba állítása, lapozás stb.). Ezek értelmezéséhez a BIOS funkcióknak további információkra van szükségük. Ezekre az információkra, valamint a billentyűzet pufferre rövidesen visszatérünk. Előbb azonban lássuk az egyes billentyűk szerepét.

A billentyűzeten lévő billentyűk a következő négy fő csoportba oszthatók:

- Karakterbillentyűk: Lenyomásukkor egy ASCII-kódú karakter képződik.
- Funkcióbillentyűk: A bal oldalon lévő F1 – F10 jelű billentyűk és a jobb oldali számbillentyűk.
- Váltóbillentyűk: Alt, Ctrl, bal és jobb váltóbillentyű (Shift billentyűk).
- Kapcsolóbillentyűk: CapsLock, Ins, NumLock, ScrollLock billentyűk.

Ebből a négy csoportból a programjaink számára értelmezhető kódokat önmagukban – néhány kivételtől eltekintve – csak a karakter- és a funkcióbillentyűk hoznak létre. A kapcsoló- és a váltóbillentyűkkel egyrészt a billentyűzet üzemmódját (nagybetűk, számjegyek bevitele), illetve a beszúrásos írásmódot (insert) választhatjuk meg, másrészt elérhető, hogy az előző két csoportba tartozó billentyűk egy részével kombinálva lényegesen több kód legyen értelmezhető anélkül, hogy a billentyűk számát növelni kellene. Van továbbá néhány speciális kapcsoló- és váltóbillentyű kombináció is, amelyekre később még kitérünk. Lássuk először a karakterbillentyűk csoportját.

Ha a karakterbillentyűk valamelyikét lenyomjuk, akkor a billentyűzet megszakítási rutinja ennek a billentyűnek az ASCII- és billentyűkódját 2 bájton beírja a billentyűzet pufferbe (az alsó bájtra az ASCII-kódot, a felső bájtra a billentyűkódot). Mivel nincs annyi billentyű, mint ahány ASCII-kód, ahhoz, hogy mind a 256 ASCII-kódot bevihessük, a karakterbillentyűket billentyűkombinációkban is kell használni. Az egyik leggyakrabban használt billentyűkombináció a bal vagy jobb oldali váltó (Shift) billentyű és egy karakterbillentyű egyidejű lenyomása. A karakterbillentyűk önmagukban vagy a váltóbillentyűkkel kombinált lenyomásával a 32 – 127 közötti ASCII-kódok írhatók be. Ismeretes az a billentyűkombináció, amellyel – a 0 (nulla) ASCII-kód kivételével – valamennyi ASCII-kódú karakter beírható: az Alt váltóbillentyűt lenyomva tartva a jobb oldali számbillentyűzeten számjegyenként beírjuk a karakter (tizes számrendszerbeli) ASCII-kódját, majd felengedjük az Alt billentyűt. (A 0 ASCII-kód azért nem írható be, mert – mint majd látni fogjuk – ezt a rendszer a bővített billentyűkódok jelzésére tartja fenn.) Az 1 – 32 közötti ASCII-kódokat a Ctrl váltóbillentyű és egy karakterbillentyű kombinációjával is be lehet írni. Nézzük meg most a funkcióbillentyűk csoportját.

Amint az előbb láttuk, a karakterbillentyűk önmagukban és az említett kombinációkban lefedik az ASCII-kódtáblázat 1 – 255 közötti kódjait. Ahhoz tehát, hogy egy funkcióbillentyű lenyomásakor a billentyűzet pufferbe kerülő kétbájtos bejegyzés első bájtját a puffert olvasó BIOS rutinok ne ASCII-kódként értelmezzék, ebbe a bájtba egy megkülönböztető értéknek kell kerülnie. Ez az érték éppen a 0, amely jelzi, hogy a lenyomott billentyű nem karakter-, hanem funkcióbillentyű volt (ezért nem írható be ASCII karakterként a 0 kód). A

bejegyzés felső bájtjára most is a funkcióbillentyű billentyűkódja kerül. A funkcióbillentyűk kódkészlete is jelentősen bővíthető a váltóbillentyűk révén. Így például a billentyűzeten jelölt, alapértelmezés szerinti 10 darab, F1 – F10 jelű funkcióbillentyű jelentése a Shift-, Ctrl- és Alt-kombinációkkal további 30-cal növelhető. Ezekhez a billentyűkombinációkhoz természetesen speciális, bővített billentyűkódok tartoznak. A 15. ábrán felsoroljuk az összes, önmagukban és kombinációban használható billentyűkombinációt és az ezekhez tartozó kibővített billentyűkódokat. A kombinációk nem használják ki az összes lehetőséget, de még így is bőséges választékot kínálnak. Az ábrában nem szereplő kombinációk nem hoznak létre billentyűkódot, ezért ezek a BIOS funkciókkal nem is kérdezhetők le.

Billentyűkód	Billentyűkombináció
15	Shift + Tab
16 – 25	Alt + Q, W, E, R, T, Y, U, I, O, P
30 – 38	Alt + A, S, D, F, G, H, J, K, L,
44 – 50	Alt + Z, X, C, V, B, N, M
59 – 68	F1 – F10
71	Home (kiinduló pozíció)
72	Cursor Up (kurzor felfelé nyíl)
73	Page Up (lapozás felfelé)
75	Cursor Left (kurzor balra nyíl)
77	Cursor Right (kurzor jobbra nyíl)
79	End (vég)
80	Cursor Down (kurzor lefelé nyíl)
81	Page Down (lapozás lefelé)
82	Ins (inzert, karakterbeszúrás)
83	Del (delete, karaktertörlés)
84 – 93	Shift + F1 – F10
94 – 103	Ctrl + F1 – F10
104 – 113	Alt + F1 – F10
114	Ctrl + PrtSc
115	Ctrl + Cursor Left
116	Ctrl + Cursor Right
117	Ctrl + End
118	Ctrl + Page Down
119	Ctrl + Home
120 – 131	Alt + 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, –, =
132	Ctrl + Page Up

15. ábra: Billentyűkombinációk és billentyűkódok

A kapcsoló- és váltóbillentyűk legfontosabb feladatairól az előzőekben már volt szó. Nézzük meg most az ezekkel létrehozható speciális kombinációkat. Négy olyan kombináció van, amelyet a BIOS a többi billentyűkombinációtól eltérően kezel. A billentyűzetet figyelő BIOS rutinok folyamatosan ellenőrzik, hogy a beérkező kódok között előfor-

dul-e a négy speciális kombináció kódjának valamelyike. Egy ilyen kódot ugyanis a BIOS parancsnak tekint, és a parancsot azonnal végre is hajtja. A négy kombináció:

Ctrl + NumLock: Felfüggeszti a program futását, és várakozási ciklusba lép. A program futása egy billentyű lenyomásakor folytatódik. Sem a Ctrl-NumLock, sem a lenyomott billentyű kódja nem kerül be a billentyűzetpufferbe.

Ctrl + Break: Meghívja az 1B(h) megszakítási vektor által címzett rutint, amely BIOS szinten csak egy IRET utasításból áll. Ezt a megszakítási rutint azonban a DOS úgy módosítja, hogy beállítson egy jelzőbitet, amelyet a DOS karakterkezelő funkciói vizsgálnak. Ha észlelik ezt a bitet, akkor megszakítják a program futását, és visszaadják a vezérlést a DOS-nak. Ha azt akarjuk, hogy egy programot a Ctrl + Break kombinációval ne lehessen „lelőni", akkor az 1B(h) megszakítási rutint ennek megfelelően módosítani kell. (Azokon a billentyűzeteken, amelyeken nincs Break billentyű, ezt a kombinációt a Ctrl + ScrollLock lenyomása jelenti.)

Shift + PrtSc: Az 5(h) megszakítási vektor által címzett rutin meghívásával kinyomtatja a képernyő aktuális tartalmát (hardcopy).

Ctrl + Alt + Del: Újraindítja a teljes rendszert (úgynevezett melegindítás). Ha a billentyűkombináció kódjának érkezésekor a megszakítások fogadása le van tiltva, vagy ha a megszakítási vektortábla 9(h) címén lévő billentyűzetmegszakítási vektor tartalma módosult, akkor a Ctrl + Alt + Del kombináció hatástalanná válhat, és a rendszer csak a gép ki- és bekapcsolásával indítható újra.

1.8.2. A billentyűzet puffere

Az eddigiekben már többször szóba került a BIOS billentyűzet puffere anélkül, hogy bármi közelebbit mondtunk volna róla. Vizsgáljuk meg most ezt egy kicsit részletesebben.

A billentyűzet puffere a RAM-ban lévő, 00400(h) abszolút címen kezdődő BIOS munkaterületen (amivel egy későbbi fejezetben még külön foglalkozunk), a 0040:001E – 003D(h) címeken helyezkedik el, és 32 bájt hosszúságú. Egy billentyű lenyomásakor a billentyűzet megszakítási rutinja a billentyű kódját 2 bájtos bejegyzésként beírja a billentyűzet pufferébe (az alsó bájtra – amennyiben a lenyomott billentyű karakterbillentyű volt – a megfelelő ASCII-kódot, ellenkező esetben 0-t, a felső bájtra pedig a billentyűkódot). Ebből következően a puffer összesen 16 billentyű kódjának tárolására képes. A puffer gyűrűs kialakítású, amelybe folyamatosan írhatók és olvashatók az adatok. Magát a puffert két kétbájtos mutató kezeli: az egyik mutató a 0040:1A – 1B(h), a másik a 0040:1C – 1D(h) címeken áll. Az 1A – 1B(h) ofszetcímeken lévő mutató, amelyet a puffer eleje mutatónak nevezünk, a puffer azon címére mutat, ahonnan a BIOS rutinoknak a

kódot ki kell olvasniuk, míg az 1C – 1D(h) puffer vége mutató arra a címre, ahová a billentyűzetről érkező kódot be kell írniuk.

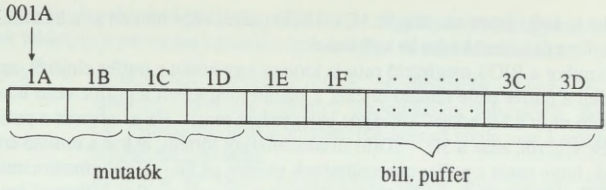
Amikor a BIOS megfelelő rutinja kiolvas egy kódot a puffer elejéről, az olvasást követően a puffer eleje mutató értékét 2 bájttal megnöveli a puffer vége irányában, így a mutató most a következő kétbájtos bejegyzésre mutat. Ha az olvasás a puffer címterületének végéről, azaz a 3C – 3D(h) ofszetcímekről történt, akkor a mutató értéke úgy változik, hogy ismét a puffer címterületének elejére, az 1E – 1F(h) címekre mutasson.

Ehhez hasonlóan viselkedik a puffer vége mutató is. Egy billentyű lenyomásakor a kétbájtos kódot a BIOS rutinok azokra a címekre írják be, amelyekre a puffer vége mutató mutat, majd a mutató értékét 2 bájttal megnövelik, szintén a puffer vége irányában.

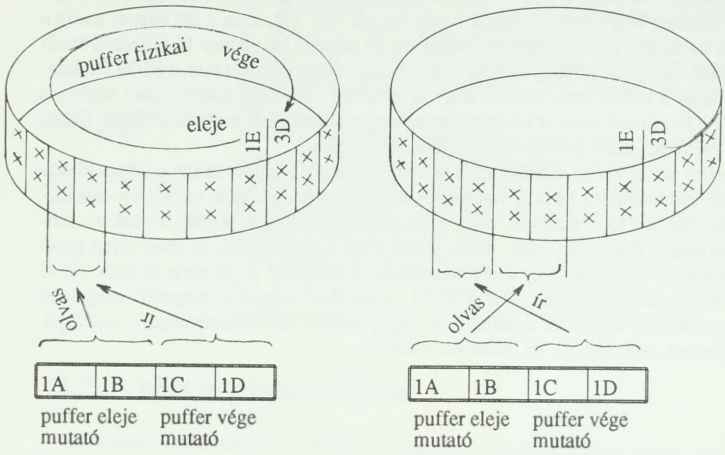
Normál esetben, amikor egy kód beérkezését azonnal követi egy olvasási művelet, a két mutató azonos címre mutat, jelezve, hogy a puffer üres.

(Megjegyezzük, hogy a puffer ilyenkor egyáltalán nem üres, hanem éppen ellenkezőleg, az előzőleg lenyomott billentyűk kódjaival tele van. Sőt, azon a két bájton, amire a mutatók mutatnak, is van bejegyzés, mégpedig az utoljára lenyomott 16 billentyű közül az elsőként lenyomott billentyű kódja. A két mutató azonos értékét csak a BIOS értelmezi úgy, hogy a puffer üres, tudván, hogy az előzőleg belekerült kódokat már kiolvasta onnan. BIOS szintről nincs is lehetőség arra, hogy a billentyűzet pufferét töröljük. Ehhez a DOS 21(h) megszakítás 0C(h) funkcióját kell használni.)

A puffer imént vázolt működéséből következik, hogy a puffer akkor is képes kódok (karakterek) fogadására, ha még van(nak) benne kiolvasandó kódok, vagyis a beírás időben megelőzheti a kiolvasást (ún. előre tartó, angolul type-ahead puffer). Ilyenkor a puffer vége mutató értéke nagyobb, mint a puffer eleje mutató értéke, és most olyan gyorsan kell olvasni a puffert, hogy a vége mutató ne „körözhesse le" az eleje mutatót. Ha ez mégis megtörténne, akkor a puffer a BIOS „számítása" szerint is megtelik, megszólal egy figyelmeztető sípszó, és a puffer nem fogad további billentyűkódokat. A billentyűzet pufferének működését a 16. ábrán szemléltetjük.



A billentyűzet pufferének tárbeli elhelyezkedése



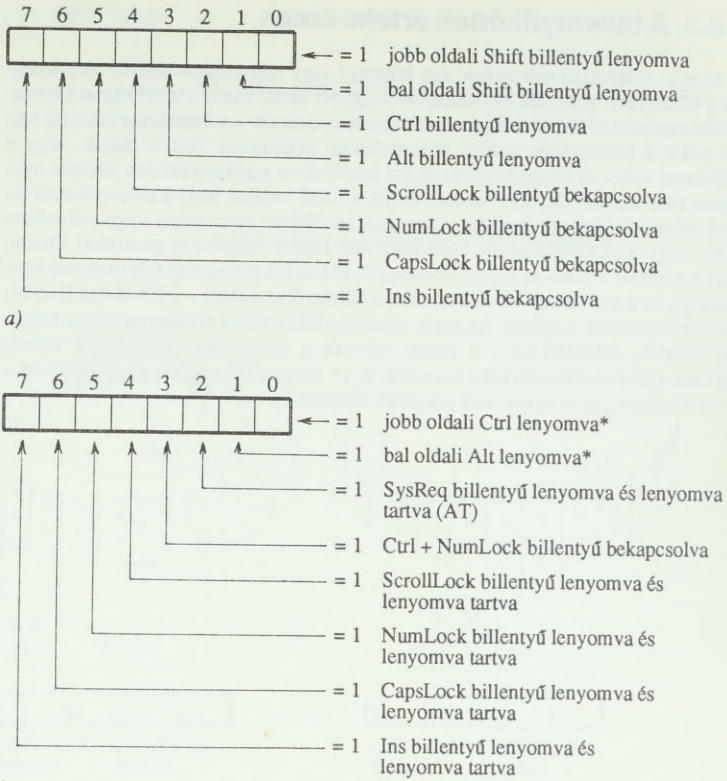
a) a billentyűzet puffere „üres" (nincs kiolvasandó kód)

b) a billentyűzet pufferében van kiolvasandó kód (az írás megelőzte az olvasást)

16. ábra: A billentyűzet pufferének működése

1.8.3. A billentyűkódok értelmezése

Mint az előzőekből már tudjuk, egy billentyű vagy billentyűkombináció lenyomásakor a billentyűzet pufferébe egy kétbájtos bejegyzés kerül, amely az első bájton a billentyűhöz tartozó ASCII-kódot vagy – funkcióbillentyű esetén – a 0 értéket, a második bájton pedig a billentyűhöz tartozó billentyűkódot tartalmazza. Azt is tudjuk, hogy a különböző váltó- és kapcsolóbillentyűkkel kombinálva a billentyűkódok jelentős mértékben kibővíthetők. De vajon honnan tudják a BIOS rutinjai, hogy a billentyűzetről érkező billentyűkódot egy karakter- vagy funkcióbillentyű önmagában vagy valamilyen váltó- vagy kapcsolóbillentyűs kombinációban történt lenyomása generálta? Onnan, hogy a rendszer a váltó- és kapcsolóbillentyűk állását két változóban folyamatosan nyilvántartja. Ez a két változó – csakúgy, mint a billentyűzet puffere – a RAM-ban lévő BIOS munkaterületen található. Az egyik változót a billentyűzet elsődleges státusz bájtjának (tárcíme 0040:0017(h)), a másik változót a billentyűzet másodlagos státusz bájtjának (tárcíme 0040:0018(h)) nevezzük. A 17. ábrán a billentyűzet e két státusz bájtjának felépítését, és az egyes bitek jelentését mutatjuk be.



17. ábra: A billentyűzet státusz bájtjainak felépítése és az egyes bitek jelentése
a) A billentyűzet elsődleges státusz bájtja; b) A billentyűzet másodlagos státusz bájtja.
** Csak a 101/102 gombos billentyűzetnél*

Nézzük meg, hogyan értelmezhetők e státusz bájtok bitjeinek jelentései.

Lássuk először a 17(h) ofszetcímen lévő státusz bájtot. A 0 – 3. bitek jelentéséhez nem kell különösebb megjegyzést fűzni. Talán csak arra érdemes felhívni a figyelmet, hogy különböző bitérték tartozik a bal és a jobb oldali Shift billentyű lenyomásához. A 4 – 7. bitek a négy kapcsolóbillentyű állását tartják nyilván. Mivel ugyanaz a kapcsolóbillentyű szolgál egy állapot be- vagy kikapcsolására, a két állapot között a billentyű lenyomásával választhatunk. A kapcsolóbillentyűk minden egyes lenyomása az előző állapot megváltozását jelenti. A bit 1 értéke a bekapcsolt, a 0 értéke a kikapcsolt állapotot jelzi.

Nézzük most a 18(h) ofszetcímen lévő másodlagos státusz bájtot. Ebben tulajdonképpen csak a 4 – 7. bitek jelentése tarthat számot az érdeklődésünkre. Megfigyelhető, hogy ezek ugyanúgy, mint a 17(h) ofszetcímű bájt azonos bitjei, a négy kapcsolóbillentyűre vonatkoznak. Logikus az a következtetés, hogy e két bájt ezen bitjei között valamilyen összefüggés van. A másodlagos státusz bájt e bitjei – és ez az eltérés a 17(h) bájt megfelelő bitjeihez képest – azonban nem azt jelzik, hogy egy billentyű lenyomásakor az illető kapcsolóbillentyű be vagy ki volt-e kapcsolva, hanem azt, hogy a billentyű lenyomására az illető kapcsolóbillentyű lenyomott vagy felengedett helyzetében került-e sor. Ha a billentyű lenyomására úgy került sor, hogy a kapcsolóbillentyű lenyomott helyzetben volt és lenyomva maradt, akkor a 18(h) ofszetcímű bájt megfelelő bitjének az értéke 1 lesz, függetlenül attól, hogy a kapcsolóbillentyűnek ez a lenyomása be- vagy kikapcsolást jelentett-e. A kapcsolóbillentyűk be- vagy kikapcsolt állapotát – amint erről már volt szó –, a 17(h) ofszetcímű bájt megfelelő bitjei figyelik.

Ezek azok az információk, amelyek birtokában a BIOS billentyűzet rutinjai képesek arra, hogy a billentyűzetről beérkező kódokat a felhasználói program számára értelmezhető módon „tálalják" a billentyűzet pufferében. A BIOS a 16(h) megszakításon keresztül három olyan funkciót bocsát a rendelkezésünkre, amelyekkel a billentyűzet puffere és a státusz bájtja olvasható. A következőkben ezekről lesz szó.

1.8.4. A billentyűzetet kezelő BIOS funkciók

A 0. sorszámú funkció a billentyűzet pufferét olvassa. A megszakítás meghívásához csak az AL regiszterbe kell a 0 értéket beírni. Ha a megszakítás meghívásakor a pufferben volt kiolvasásra váró kód, akkor ezt a funkció „kiolvassa" (pontosabban – mint láttuk – a puffer kezdetét címző mutató értékét 2-vel megnöveli), és a kétbájtos kódot az AH, ill. az AL regiszterbe tölti. Ha az AL értéke 0-tól különböző, akkor ez azt jelzi, hogy a lenyomott billentyű karakterbillentyű volt, és az AL regiszterben a lenyomott billentyű ASCII-kódja, az AH regiszterben pedig a billentyűkód áll. Ha viszont az AL regiszterben a 0 értéket kapjuk vissza, akkor – mint erről már részletesen volt szó – ez azt jelzi, hogy a lenyomott billentyű funkcióbillentyű vagy valamilyen billentyűkombináció volt, és az AH regiszter most az ennek megfelelő bővített billentyűkódot tartalmazza (a lehetséges billentyűkombinációkat és a hozzájuk tartozó billentyűkódokat a 15. ábrában soroltuk fel).

Ha a megszakítás meghívásakor nem volt kiolvasásra váró kód a billentyűzet pufferben, akkor a megszakítás mindaddig vár, míg a kezelő le nem nyom egy billentyűt, és

csak ezt követően tér vissza a meghívó programhoz. Nyilvánvalóan ez a legegyszerűbb módja annak, hogy a programunk egy billentyű lenyomására várakozzon, és a lenyomott billentyűre megfelelően reagáljon.

Az 1. sorszámú funkcióval is a billentyűzet puffere kérdezhető le. A 0. számú funkcióhoz képest az egyik eltérés az, hogy az 1. számú a megszakítás meghívását követően nem várakozik egy billentyű esetleges lenyomására, hanem – függetlenül attól, hogy van-e a pufferben kiolvasásra váró kód vagy nincs – azonnal visszatér a meghívó programhoz. Ha nem talált kiolvasandó kódot, akkor a zéróbitet (Zero flag) magasra állítja, ha talált, akkor 0 értékű zéróbittel tér vissza. Ekkor a rendelkezésre álló kódot az előbb megismert módon beírja az AH, ill. az AL regiszterekbe, de – és a 0. funkcióhoz képest ez a másik eltérés – a kódot nem olvassa ki a pufferből, azaz a mutatókat nem állítja át. Ebből következik, hogy az 1. számú funkció ismételt meghívása az AH, ill. AL regiszterekbe ugyanazokat az információkat írja be, mint a megelőző meghívás.

A 2. sorszámú funkcióval a 0040:0017(h) címen lévő elsődleges státusz bájt kérdezhető le. A megszakítás visszatérésekor az AL regiszterben szolgáltatja a státusz bájtot (felépítését lásd a 17.a) ábrán). A másodlagos státusz bájt BIOS szintről nem kérdezhető le.

1.8.5. Programpéldák

A billentyűzetet kezelő BIOS funkciókat szemléltető példaprogramjaink két csoportba sorolhatók. Az első csoportba az eddigiekhez hasonlóan BASIC, C és Turbo Pascal nyelvű programok tartoznak, amelyek mindegyike a billentyűzet elsődleges státuszbájtjának értelmezését és kezelését szemlélteti. A második csoport valójában nem is csoport, hanem mindössze egy assembly program, amely a billentyűzetkezelő szolgáltatás megszakítás vektorcímét állítja rá egy külső megszakításkezelő eljárásra.

A billentyűzet elsődleges státuszbájtjának magasszintű nyelvekből történő kezelése, lekérdezése a felhasználói programok készítői számára is hasznos lehetőségeket rejt magában. Programpéldáink terjedelme igen szűkreszabott, a billentyűzetkezelő funkciók közül csak a státuszbájt lekérdezésére alkalmas 2-es funkciót használják, ezt hívják végtelen ciklusban a program CTRL–BREAK megnyomásával történő félbeszakításáig. Minden egyes inicializálást egy azonnali kijelzés követ, a státuszbájtnak az előző fejezetben ismertetett felépítése alapján folyamatosan nyomon követhető a négy váltó- és négy kapcsolóbillentyű aktuális állapota (az aktív állapotot a váltóbillentyűk esetén a lenyomva tartás, a kapcsolóbillentyűk esetén az egyszeri megnyomás idézi elő). Nem véletlen, hogy ilyen egyszerű példákat mutatunk be, a billentyűzet itt bemutatott mélységű kezelése alapul szolgálhat saját, sokoldalú billentyűzetkezelő program kifejlesztéséhez, amennyiben kiegészítjük a billentyűzetpuffer folyamatos figyelésével (1(h) funkció) és a BIOS segítségével történő beolvasással (0(h) funkció). Saját szövegszerkesztő, menürendszer stb. íly módon való elkészítése igen hatékony megoldásokat eredményezhet, azonban ennek további járulékos kellékei is vannak (kurzorpozíció, szín mentése a státuszbájt tartalmának megjelenítése előtt majd visszaállítása a megjelenítés után stb.).

A kapcsolóbillentyűknek a programokban bemutatott módon történő figyelése kiemelt jelentőségű azokon a géptípusokon (XT), ahol nem jelzik fénydiódák ezen billentyűk aktuális állapotát.

Az assembly példaprogram a billentyűzetkezelés szemléltetése mellett elvi jelentőséggel is bír. A program egy tárban maradó (rezidens) megszakítás meghajtó. Ez a megnevezés jelzi, hogy a programot betöltése után egyéb programok, ill. azok adatterületei nem írják felül. A program funkciója, mint arra előző megnevezése is utal, egy eddig a DOS vagy a BIOS által végrehajtandó alapértelmezés szerinti megszakítás rutin kiváltása. Általánosan kijelenthetjük, hogy ez a program egyszerűbb esetekben mintául szolgálhat saját rezidens megszakítás rutinok írásához.

A program feladata, hogy egy meghatározott billentyű megnyomásakor megjelenítse a képernyőn az aktuális időt. Nincs alapértelmezett billentyű, azt a parancssorban kell megadni. Ezzel kapcsolatban az egyetlen megszorítás, hogy csak bővített billentyűkódokat kezelhetünk le ily módon (alapértelmezés a megadott billentyűkód előtt a „0" kód vizsgálata, ily módon pl. ha 15 lett paraméterként átadva, a rutin a SHIFT-TAB megnyomásakor fejti ki hatását). A képernyőkoordináták megadása a program indításakor opcionális, azok hiánya esetén az idő a jobb felső sarokban jelenik meg. A képernyőn megjelenő óra mindaddig nem tűnik el, amíg egy újabb billentyűt le nem nyomunk, ezt követően a képernyő óra alatti területének tartalma visszaállítódik.

A program hívásakor a paraméterezés teljes elhagyása a program tárbeli eltávolításának igényét jelzi. Ha a program még nem lett installálva, akkor ez természetesen nem lehetséges. Ezzel teljesen analóg jellegű hiba, amikor kétszer egymásután paraméterezve hívjuk meg a programot, hiszen azt kétszer egymásután nem lehet installálni. A program az azonosító kód (két bájt) alapján tudja eldönteni, hogy installálva lett-e már vagy sem. Az új megszakítás rutin installálásakor tárolni kell a megszakítás vektor megfelelő pozícióján levő címet (esetünkben ez a 16(h) megszakítás vektorbeli címe). A cím a 21(h) DOS megszakítás 35(h) funkciójával kapható meg, a kiszállás (inaktiválás) során pedig a 25(h) funkcióval állítható be újra eredeti értékére.

Az új megszakítás rutin installálása után minden egyes 16(h) megszakítás híváskor megkapja a vezérlést. Ekkor az első lépés annak a megállapítása, hogy a 0(h) funkcióval lett-e meghíva a megszakítás. Ha nem, akkor természetesen azonnal továbbkerül a vezérlés az eredeti megszakítás rutinra, ilyen esetben semmilyen járulékos tevékenységet nem végez programunk. Ha a 0(h) funkció lett híva, úgy be kell olvasni egy karaktert a billentyűzetről, majd meg kell állapítani, hogy az megegyezik-e az installáláskor definiált bővített billentyűkóddal. Ha igen, akkor következik a program érdemi részének végrehajtása: az óra számára kijelölt képernyőterület tárolása, az óra megjelenítése, ismételt karakterbeolvasás, majd a beolvasott billentyűtől függően ismét megjelenítés vagy a képernyő helyreállítása és a vezérlés visszaadása.

```
1000 '
1005 '  ┌──────────────────────────────────────────────────────┐
1010 '  │                  B L T Y U Z E T . B A S             │
1020 '  │                                                      │
1030 '  │ Funkciója : a váltó- és kapcsolóbillentyűk állapotának fo-
1040 '  │             lyamatos figyelése és kijelzése.         │
1050 '  │                                                      │
1060 '  └──────────────────────────────────────────────────────┘
1070 '
1080 DEFINT Q                                        'Q "dummy" egész érték.
1090 KEY OFF : CLS
1100 PRINT : PRINT
1110 PRINT "FIGYELMEZTETÉS :" : PRINT
1120 PRINT "A 60000-es címtől kezdődően betoltott megszakításhívó ru-"
1130 PRINT "tin felülírását elkerulendő a GWBASIC-et GWBASIC /M:60000"
1140 PRINT "parancsal kell indítania. Ha nem igy hívta a GWBASIC-et,"
1150 PRINT "akkor usse le az <s> billentyűt, egyébként egy tetszöle-"
1160 PRINT "ges másik billentyűt !"
1170 '
1180 Z$ = ""
1190 WHILE Z$ = ""                                   'Helytelen indítás
1200    Z$ = INKEY$                                  'esetén a program
1210 WEND                                            'leállítása.
1220 IF Z$ = "s" OR Z$ = "S" THEN CLS:END
1230 '
1240 GOSUB 9000                                      'Megszakításhívó rutin
1250                                                 'betöltése.
1260 CLS
1270 '
2000 '
2010 '  ┌──────────────────────────────────────┐
2020 '  │          P R O G R A M T Ö R Z S      │
2030 '  └──────────────────────────────────────┘
2040 PRINT :PRINT:PRINT
2050 PRINT "A billentyűzet váltó- és kapcsolóbillentyűinek mindenkori"
2060 PRINT "állását szemléltető program a <CTRL-C> kombináció megnyo-"
2070 PRINT "másával állítható le..."
2080 PRINT :PRINT
2090 PRINT "1. Jobb oldali Shift billentyű lenyomva"
2100 PRINT "2. Bal oldali Shift billentyű lenyomva"
2110 PRINT "3. Ctrl billentyű lenyomva"
2120 PRINT "4. Alt billentyű lenyomva"
2130 PRINT "5. ScrollLock billentyű bekapcsolva"
2140 PRINT "6. NumLock billentyű bekapcsolva"
2150 PRINT "7. CapsLock billentyű bekapcsolva"
2160 PRINT "8. Ins billentyű bekapcsolva"
2170 PRINT :PRINT
2180 PRINT "   1     2     3     4     5     6     7     8   "
2190 PRINT "                                                  "
2200 PRINT "                                                  "
2210 PRINT "                                                  "
2220 '
2230 VAN$  = "***"
2240 NINCS$ = "   "
2250 '
2260 GOSUB 3000                                      ' Inicializálás.
2270 GOSUB 4000                                      ' Kiírás.
2280 GOTO 2260
2290 '
2300 CLS:END
2310 '
```

```
3000 '
3010 '  A BILLENTYOZET ELSÖDLEGES STATUSZBAJTJA-
3015 '  NAK INICIALIZALASA
3020 '      KI : BSTAT% az elsödleges státuszbájt
3030 '
3040 '
3050 FUN%=2                                  ' Billentyüzet státusz funkció.
3060 MSZ%=&H16                               ' 16(h) megszakítás.
3070 '
3080 CALL RCIM(MSZ%,FUN%,BSTAT%,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q)
3090 RETURN
3100 '
4000 '
4010 '  A BILLENTYOZET ELSÖDLEGES STATUSZBAJTJA-
4020 '  NAK ELEMZÉSE ÉS KIÍRÁSA
4030 '      BE : BSTAT% az elsödleges státuszbájt
4040 '      KI : -
4050 '
4060 '
4070 I% = 1
4080 J% = 0
4090 '
4100 LOCATE 21, 3+J%*6
4110 IF ((BSTAT% AND I%) = I%) THEN KAR$ = VAN$ ELSE KAR$ = NINCS$
4120 PRINT KAR$
4130 J% = J%+1                                ' Az egyes bitek
4140 I% = I%*2                                ' jelentését a
4150 IF I% < 256 THEN GOTO 4100               ' program futása
4160 RETURN                                   ' során kiírja.
4170 '
9000 '
9010 '  A MEGSZAKÍTAS Hívó RUTIN INICIALIZALASA
9020 '      BE: -
9030 '      KI: RCIM a Rutin CIMe
9040 '
9050 '
9060 RCIM = 60000!                  'A rutin kezdöcíme a BASIC szegmensben.
9070 DEF SEG                        'A BASIC szegmens beállítása.
9080 RESTORE 9140
9090 FOR I% = 0 TO 160                              'A rutin betoltése a
9100     READ X% : POKE RCIM + I%, X%               'tárba.
9110 NEXT
9120 RETURN
9130
9140 DATA    85, 139, 236,  30,   6, 139, 118,  30, 139,   4, 232, 140
9150 DATA     0, 139, 118,  12, 139,  60, 139, 118,   8, 139,   4,  61
9160 DATA   255, 255, 117,   2, 140, 216, 142, 192, 139, 118,  28, 138
9170 DATA    36, 139, 118,  26, 138,   4, 139, 118,  24, 138,  60, 139
9180 DATA   118,  22, 138,  28, 139, 118,  20, 138,  44, 139, 118,  18
9190 DATA   138,  12, 139, 118,  16, 138,  52, 139, 118,  14, 138,  20
9200 DATA   139, 118,  10, 139,  52,  85, 205,   0,  93,  86, 156, 139
9210 DATA   118,  12, 137,  60, 139, 118,  28, 136,  36, 139, 118,  26
9220 DATA   136,   4, 139, 118,  24, 136,  60, 139, 118,  22, 136,  28
9230 DATA   139, 118,  20, 136,  44, 139, 118,  18, 136,  12, 139, 118
9240 DATA    16, 136,  52, 139, 118,  14, 136,  20, 139, 118,   8, 140
9250 DATA   192, 137,   4,  88, 139, 118,   6, 137,   4,  88, 139, 118
9260 DATA    10, 137,   4,   7,  31,  93, 202,  26,   0,  91,  46, 136
9270 DATA    71,  66, 233, 108, 255
```

```
/*
/*  ┌──────────────────────────────────────────────────────────┐
/*  │                                                          │
/*  │                      B L T Y U Z E T . C                 │
/*  │                                                          │
/*  │  Funkciója : a váltó- és kapcsolóbillentyűk állapotának fo-  │
/*  │              lyamatos figyelése és kijelzése.            │
/*  │                                                          │
/*  └──────────────────────────────────────────────────────────┘

#include <dos.h>
#include <io.h>

#define  FALSE    0
#define  TRUE     1

#define  NM    0x07                          /* Normál megjelenítés. */

#define byte unsigned char

/*
/*  ┌──────────────────────────────────────────────────────────┐
/*  │  AKTOLDAL : AZ AKTUALIS KÉPERNYŐOLDAL MEG-               │
/*  │             HATAROZASA                                   │
/*  └──────────────────────────────────────────────────────────┘

byte AKTOLDAL()

{
union REGS regiszter;                     /* Regiszter union deklaráció. */

regiszter.h.ah = 0x0F;                        /* Videó olvasás funkció.  */
int86(0x10, &regiszter, &regiszter);          /* 10(h) megszakítás.      */
return(regiszter.h.bh);                       /* Aktuális képernyőoldal. */
}

/*
/*  ┌──────────────────────────────────────────────────────────┐
/*  │  KURPOZ : A KURZOR POZíCIONALASA AZ AKT.                 │
/*  │           KÉPERNYŐOLDALON                                │
/*  └──────────────────────────────────────────────────────────┘

void KURPOZ(oszlop, sor)
int oszlop;
int sor;

{
union REGS regiszter;                     /* Regiszter union deklaráció. */

regiszter.h.ah = 2;                           /* Kurzor beállítása az    */
regiszter.h.bh = AKTOLDAL();                  /* aktuális oldalon az     */
regiszter.h.dh = sor;                         /* átadott sor- és osz-    */
regiszter.h.dl = oszlop;                      /* loppozícióra.           */
int86(0x10, &regiszter, &regiszter);          /* 10(h) megszakítás.      */
}

/*
/*  ┌──────────────────────────────────────────────────────────┐
/*  │  GORGETES : MEGADOTT KÉPERNYŐTARTOMÁNY                   │
/*  │             FELFELE GÖRGETÉSE, TÖRLÉSE                   │
/*  └──────────────────────────────────────────────────────────┘

void GORGETES(gsor, szin, bfo, bfs, jao, jas)

int gsor;              /* (G)örgetendő (sor)ok száma.                  */
int szin;              /* A törölt terület (szín)e.                    */
int bfs ;              /* Képernyőtartomány (b)al  (f)első (s)ora.     */
int bfo ;              /*                   (b)al  (f)első (o)szlopa.  */
```

```
int jas ;              /*                  (b)al  (a)lsó  (s)ora.     */
int jao ;              /*                  (j)obb (a)lsó  (o)szlopa.  */
{
union REGS regiszter;                  /* Regiszter union deklaráció. */

regiszter.h.ah = 0x06;                 /* Görgetés funkció.           */
regiszter.h.al = gsor;                 /* Görgetendő sorok száma.     */
regiszter.h.bh = szin;                 /* Az üres sorok színe.        */
regiszter.h.ch = bfs ;                 /* Ablak-                      */
regiszter.h.cl = bfo ;                 /*        koor-                */
regiszter.h.dh = jas ;                 /*             dináták.        */
regiszter.h.dl = jao ;                 /*                             */

int86(0x10, &regiszter, &regiszter);          /* 10(h) megszakítás.   */
}

/*
/*  CLRSCR : KÉPERNYŐ TÖRLÉSE                                          */
/*                                                                     */

void CLRSCR()

{
GORGETES(0, NM, 0, 0, 79, 24);         /* Törlés görgetéssel.         */
KURPOZ(0, 0);                          /* Kurzor "home" pozícióba.    */
}

/*
/*  INIC : A BILLENTYŰZET ELSŐDLEGES STÁTUSZ-                          */
/*         BAJTJÁNAK INICIALIZÁLÁSA                                    */
/*

INIC()

{
union REGS regiszter;                  /* Regiszter union deklaráció. */

regiszter.h.ah = 2;                      /* Billentyűzet státusz fun. */
int86(0x16, &regiszter, &regiszter);     /* 16(h) megszakítás.        */
return regiszter.h.al;
}

/*
/*  STAT : A BILLENTYŰZET ELSŐDLEGES STÁTUSZ-                          */
/*         BAJTJÁNAK KIÍRÁSA AZ EGYES BITEK                            */
/*         JELENTÉSÉT FIGYELEMBE VÉVE.                                 */
/*

void STAT(bitystat)
int bitystat;

{
char van = '*';
char nincs = ' ';
char kar;
int  i, j;
for (i=1, j=0; i<256; i=i*2, j=j+1)                  /* Az egyes bitek */
   {                                                 /* jelentését a   */
   KURPOZ(2+j*6, 21);                                /* program futása */
   kar = bitystat & i ? van : nincs;                 /* során kiírja.  */
   printf("%c%c%c", kar, kar, kar);
   }
}
```

```
/*
/*                    FŐPROGRAM                    */
/*                                                 */
                                                   */

void main()

{
CLRSCR();
printf("");
printf("A billentyűzet váltó- és kapcsolóbillentyűinek mindenkori");
printf("állását szemléltető program a <CTRL-C> kombináció megnyo-");
printf("másával állítható le...");
printf("1. Jobb oldali Shift billentyű lenyomva");
printf("2. Bal oldali Shift billentyű lenyomva");
printf("3. Ctrl billentyű lenyomva");
printf("4. Alt billentyű lenyomva");
printf("5. ScrollLock billentyű bekapcsolva");
printf("6. NumLock billentyű bekapcsolva");
printf("7. CapsLock billentyű bekapcsolva");
printf("8. Ins billentyű bekapcsolva");
printf("    1     2     3     4     5     6     7     8   ");
printf("                                                   ");
printf("                                                   ");
printf("                                                   ");

while (TRUE)
   STAT(INIC());
return;
}
```

```
{ ┌──────────────────────────────────────────────────────────┐ }
{ │                                                          │ }
{ │                     B L T Y U Z E T . P A S              │ }
{ │                                                          │ }
{ │  Funkciója : a váltó- és kapcsolóbillentyűk állapotának fo- │ }
{ │              lyamatos figyelése és kijelzése.            │ }
{ │                                                          │ }
{ └──────────────────────────────────────────────────────────┘ }

program BLTYUZET;

type dosreg     = record
                  ax, bx, cx, dx, bp, di, si, ds, es, flags : integer;
                  end;

{ ┌──────────────────────────────────────────────────┐ }
{ │ INIC : A BILLENTYŰZET ELSŐDLEGES STATUSZ-        │ }
{ │        BAJTJÁNAK INICIALIZÁLÁSA                  │ }
{ └──────────────────────────────────────────────────┘ }

function INIC:integer;

var regiszter : dosreg;                  { Regiszter rekord deklaráció.  }

begin
   regiszter.ax := $200;                 { Billentyűzet státusz funkció. }
   intr($16, regiszter);                 { 16(H) megszakítás.            }
   INIC := lo(regiszter.ax);
end;

{ ┌──────────────────────────────────────────────────┐ }
{ │ STAT : A BILLENTYŰZET ELSŐDLEGES STATUSZ-        │ }
{ │        BAJTJÁNAK KIÍRÁSA AZ EGYES BITEK          │ }
{ │        JELENTÉSÉT FIGYELEMBE VÉVE.               │ }
{ └──────────────────────────────────────────────────┘ }

procedure STAT(bitystat : integer);

var   i, j             : integer;
      van, nincs, kar : char;
begin
van   := '*';
nincs := ' ';
j     := 0;
i     := 1;
while i < 256 do                                  { Az egyes bitek }
   begin                                          { jelentését a   }
   gotoxy(3+j*6, 22);                             { program futása }
   if (bitystat and i) <> 0 then kar := van       { során kiírja.  }
                            else kar := nincs;
   write(kar, kar, kar);
   i := i*2;
   j := j+1;
   end
end;
```

```
{ ┌──────────────────────────────────────┐ }
{ │          F Ő P R O G R A M           │ }
{ └──────────────────────────────────────┘ }

begin
clrscr;
writeln(#13#10#13#10);
writeln('A billentyűzet váltó- és kapcsolóbillentyűinek mindenkori');
writeln('állását szemléltető program a <CTRL-C> kombináció megnyo-');
writeln('másával állítható le...');
writeln(#13#10);
writeln('1. Jobb oldali Shift billenytű lenyomva');
writeln('2. Bal oldali Shift billentyű lenyomva');
writeln('3. Ctrl billentyű lenyomva');
writeln('4. Alt billentyű lenyomva');
writeln('5. ScrollLock billentyű bekapcsolva');
writeln('6. NumLock billentyű bekapcsolva');
writeln('7. CapsLock billentyű bekapcsolva');
writeln('8. Ins billentyű bekapcsolva');
writeln(#13#10#13#10);
writeln('     1        2        3        4        5        6        7        8    ');
writeln(' ┌────────┬────────┬────────┬────────┬────────┬────────┬────────┬────────┐');
writeln(' │        │        │        │        │        │        │        │        │');
writeln(' └────────┴────────┴────────┴────────┴────────┴────────┴────────┴────────┘');

while TRUE do STAT(INIC);

end.
```

```
;
;  ┌──────────────────────────────────────────────────────┐
;  │                                                      │
;  │                      O R A 1 . A S M                 │
;  │                                                      │
;  │  Funkciója: óraidő megjelenítése a megadott kiterjesztett
;  │             billentyű megnyomásakor.                 │
;  │                                                      │
;  │   Hívása  : ORA1 [bkod] /S[sor] /O[oszlop]           │
;  │                                                      │
;  └──────────────────────────────────────────────────────┘
;
;— KONSTANSOK ──────────────────────────────────────────────────

        SZIN = 70h                  ; Az óra színe.

;— KÓD ─────────────────────────────────────────────────────────

kod     segment para 'CODE'         ; A kódszegmens definiálása.

        org 100h                    ; A program a PSP után álló első
                                    ; címen kezdődik.

        assume cs:kod, ds:kod, es:kod, ss:kod

                                    ; Csak egy, közös szegmens van.

start:  jmp  inic                   ; Az inicializáló rutin hívása.

;— ADATOK ──────────────────────────────────────────────────────

origmsz  equ this dword             ; Az eredeti 16(h) megszakítás címe :
origmszo dw (?)                     ;      ofszetcím,
origmszs dw (?)                     ;      szegmenscím.

bovbill  db (1)                     ; Bővített billentyűkód, amely a
billkod  db (?)                     ; programot aktiválja.

orapoz   equ this word
oszlop   db 75                      ; Az időkijelzés sora és oszlopa.
sor      db 0

tarolo   dw 5 dup (?)               ; Az óra alatti karakterek mentése.

;— A memóriában maradó új megszakítás rutin ────────────────────

oramsz  proc far

        jmp  short cim1

        db "ZZ"                     ; A program neve.

cim1:   or   ah,ah                  ; Ha nem karakter olvasás funkció,
        je   cim2                   ; akkor ugrás az eredeti megsza-
        jmp  cim5                   ; kításra.

cim2:   pushf                       ; A régi megszakítás meghívása, az
        call cs:[origmsz]           ; egy billentyűkóddal tér vissza.

        cmp  ax,cs:word ptr bovbill ; Ha ez a program megadott
                                    ; billentyűje,
```

```
            je   kiiras                ; akkor ki kell jelezni az órát.
            jmp  vege                  ;

            ;-- A definiált billentyű megnyomása ----------------------

kiiras:     pushf                      ; Regiszterek elmentése a veremre.
            push ax
            push bx
            push cx
            push dx
            push di
            push si
            push es
            push ds

            cld
            mov  ah,15                 ; Képernyőoldal lekérdezése funkció.
            int  10h                   ; 10(h) megszakítás.
            mov  ah,3                  ; Kurzorpozíció lekérdezése funkció.
            int  10h                   ; 10(h) megszakítás.
            push dx                    ; Kurzorpozíció mentése a veremre.

            push cs                    ; CS --> DS.
            pop  ds                    ;
            mov  dx,orapoz             ; Az óra képernyőpozíció beállítása.
            mov  ah,2                  ; Kurzorpozícionálás funkció.
            int  10h                   ; 10(h) megszakítás.
            push cs                    ; CS --> ES.
            pop  es                    ;
            mov  cx,5                  ; 5 karakter lesz beolvasva.
            mov  di,offset tarolo      ; Karakterpuffer címe --> DI.
cim3:       mov  ah,8                  ; Karakter olvasás funkció.
            int  10h                   ; 10(h) megszakítás.
            stosw                      ; A karakter eltárolása.
            inc  di                    ; Átlépés a következő oszlopra.
            mov  ah,2                  ; Kurzorpozícionálás funkció.
            int  10h                   ; 10(h) megszakítás.
            loop cim3                  ; Vissza a következő kar. olvasására.
            mov  dx,orapoz             ; Az óra képernyőpozíció beállítása.
            mov  ah,2                  ; Kurzorpozícionálás funkció.
            int  10h                   ; 10(h) megszakítás.
            mov  ah,2CH                ; óra lekérdezés funkció.
            int  21h                   ; 21(h) megszakítás.
            mov  bl,SZIN               ; Az óra színe --> BL.
            push cx                    ; Idő --> verem.
            mov  al,ch                 ; órák ASCII konverziója majd meg-
            call binasc                ; jelenítése.
            mov  al,":"                ; Kettőspont megjelenítése.
            call prn                   ;
            pop  ax                    ; Verem --> idő.
            call binasc                ; Percek ASCII konverziója majd meg-
                                       ; jelenítése.
            mov  dx,orapoz             ; Az óra képernyőpozíció beállítása.
            mov  ah,2                  ; Kurzorpozícionálás funkció.
            int  10h                   ; 10(h) megszakítás.
            xor  ah,ah                 ; Várakozás egy billentyűre.
            pushf
            call cs:[origmsz]          ; Az eredeti megszakítás felhívása.
```

```
          mov  cx,1                ; Mindig egy karakter jelenítendő meg.
          mov  di,5                ; 5 karakter lesz megjelenítve.
          mov  si,offset tarolo    ; Karakterpuffer címe --> SI.
          mov  dx,orapoz           ; Az óra képernyőpozíció beállítása.
cim4:     lodsw                    ; Karakter és szín a pufferből.
          mov  bl,9                ; Karakter kiírás funkció.
          xchg bl,ah               ; AH <--> BL.
          int  10h                 ; 10(h) megszakítás.
          inc  dl                  ; Következő oszlop.
          mov  ah,2                ; Kurzorpozícionálás funkció.
          int  10h                 ; 10(h) megszakítás.
          dec  di                  ; Ha van még karakter, ugrás a
          jne  cim4                ; ciklus elejére.
          pop  dx                  ; Eredeti kurzorpozíció visszahívása.
          mov  ah,2                ; Kurzorpozícionálás funkció.
          int  10h                 ; 10(h) megszakítás.

          pop  ds                  ; A regiszterek visszatoltése a
          pop  es                  ; veremről.
          pop  si
          pop  di
          pop  dx
          pop  cx
          pop  bx
          pop  ax
          popf
          xor   ah,ah
          jmp   cim2

cim5:     pushf                    ; Megszakítás szimulálás.
          call cs:[origmsz]        ;
vege:     ret  2                   ; Stack paraméter "eldobása".

oramsz    endp

;— BINASC : Binárisról ASCII kódra konvertálás ————————————————
;— BE : AL = az átalakítandó szám
;— KI : -
;— Regiszterek : CX, AX, DL és FLAGS változnak

binasc    proc near

          mov  cl,10               ; Tízes számrendszer előírása.
          xor  ah,ah               ;
          div  cl                  ; AX / CL.
          or   ax,3030h            ; Az eredményt ASCII-vá alakítása.
          push ax                  ; Eredmény --> verem.
          call prn                 ; Karakter kiírás, kurzor léptetés.
          pop  ax                  ; Verem --> eredmény.
          mov  al,ah               ; A második karakter --> AL.
          call prn                 ; Karakter kiírás, kurzor léptetés.
          ret                      ;

binasc    endp

;— PRN : Karakter megjelenítés, kurzor léptetés ———————————————
;— BE : BH = megjelenítés képernyőoldala
;—      AL = megjelenítendő karakter
;—      BL = karakter attribútum
;— KI : -
;— Regiszterek : CX, AH, DL és FLAGS változnak
```

```
prn     proc near

        mov  cx,1                   ; A karakter egyszer jelenik meg.
        mov  ah,9                   ; Karakter kiírás funkció.
        int  10h                    ; 10(h) megszakítás.
        mov  ah,3                   ; Kurzorpozíció lekérdezése funkció.
        int  10h                    ; 10(h) megszakítás.
        inc  dl                     ; Az oszlopszám növelése.
        mov  ah,2                   ; Kurzorpozícionálás funkció.
        int  10h                    ; 10(h) megszakítás.
        ret                         ;

prn     endp

vegcim  equ this byte               ; Installálás esetén innen kezdve
                                    ; válik a tár felulírhatóvá.
;— ADATOK ——————————————————————————————————————————————

CR      equ 13                      ; Kocsivissza ASCII kódja.
LF      equ 10                      ; Soremelés ASCII kódja.
KEND    equ "$"                     ; Karakterlánc vége jelzés.

txt_1   db CR,LF,"Hibás paraméterezés.",CR,LF,KEND
txt_2   db CR,LF,"Az óramegjelenítés megszakítás installálva.",CR,LF
        db "Inaktiválása egy újbóli (paraméter nélküli)",CR,LF
        db "felhívással történik.",CR,LF,KEND
txt_3   db CR,LF,"Az óramegjelenítés megszakítás inaktiválva."
        db CR,LF,KEND
txt_4   db CR,LF,"Az óramegjelenítés megszakítás egyszer már",CR,LF
        db "installálva lett.",CR,LF,KEND
txt_5   db CR,LF,"Az óramegjelenítés megszakítás még nem lett",CR,LF
        db "installálva.",CR,LF,KEND

parsor  dw 63 dup (?)               ; Parancssor paraméterek címe.

;— PROGRAM — a DOS FELULÍRHATJA ———————————————————————

inaktiv label near                  ; Az ORA1 inaktiválása.

        mov  ax,3516h               ; Megszakítás cím lekérdezés funkció.
        int  21h                    ; 21(h) DOS megszakítás.
                                    ; A megsz. rutin címe ES:BX-be kerül.

        cmp  word ptr es:[bx+2],"ZZ"  ; Annak eldöntése, hogy az
                                      ; ORA1 megsz. rutin aktív-e.

        jne  inakt                  ; Nincs installálva, nem inkativáható.
                                    ;
        mov  dx,es:origmszo         ; A régi megszakítás visszaállítása,
        mov  ax,es:origmszs         ; a címe DS:DX lesz.
        mov  ds,ax                  ;
        mov  ax,2516h               ; Megszakítás vektor beáll. funkció.
        int  21h                    ; DOS megszakítás hívása.
                                    ;
        mov  ah,49h                 ; Memória felszabadítás funkció.
        int  21h                    ; 21(h) DOS megszakítás.
                                    ;
        push cs                     ; CS  -->  DS.
        pop  ds

torol:  mov  dx,offset txt_3        ; Inaktiválási üzenet,
        xor  al,al                  ; program végére ugrás.
        jmp  irasveg
```

```
inakt:  mov  dx,offset txt_5    ; Hibaüzenet, nincs az óra
        jmp  short inakth       ; installálva.

;— Inicializálás ——————————————————————————————————

inic    proc near

        cld
        mov  di,offset parsor   ; Parancssor paraméterek címe.
        call param              ; Ugrás a param. kiértékelésre.
        or   dl,dl              ; Ha nincs paraméter, inaktiválni
        je   inaktiv            ; kell a programot.

        ;— Paraméter kiértékelés ——————————————————————

        mov  bx,offset parsor   ; Parancssor paraméterek címe.
ertekel: mov  si,[bx]           ; Paraméter cím beolvasása.
        lodsw                   ;
        and  ah,11011111b       ; A paraméter első két karaktere
        cmp  ax,"S/"            ; (kisbetű nagybetűre konvertálva)
        je   sorolv             ; alapján a sor-, ill. oszlopparaméter
        cmp  ax,"O/"            ; azonosítása.
        je   oszolv             ;

        ;— Az első paraméter a billentyűkód ——————————————

        cmp  bovbill,0          ; Ha lett már billentyűkód definiálva
        je   parhiba            ; paraméterhiba.

        push bx                 ; BX --> verem.
        push dx                 ; DX --> verem.
        sub  si,2               ; SI-t a szám elejére állítani.
        call ascbin             ; ASCII - bináris átalakítás.
        pop  dx                 ; Verem --> DX.
        pop  bx                 ; Verem --> BX.

        jc   parhiba            ; Ha az átadott  paraméter nem szám,
        or   ah,ah              ; ill. nagyobb 255-nél, ill. a DH-ban
        jne  parhiba            ; megengedettnél, akkor paraméter
                                ; átadási hiba.
        mov  billkod,al         ; Ha elfogadható az érték, eltárolás.
        mov  bovbill,0          ; 0 jelzi, hogy van beolvasott kód.

kovpar: add  bx,2               ; Következő parancssor param. címe.
        dec  dl                 ; Ha az utolsó paraméter, a program
        jne  ertekel            ; installálása, ha nem, akkor az újabb
        jmp  short install      ; paraméter kiértékelése.

sorolv: mov  di,offset sor      ; Sor változó ofszetcíme.
        mov  dh,24              ; Sor maximális értéke 24.
        jmp  karert             ; Paraméter kiértékelés.

oszolv: mov  di,offset oszlop   ; Oszlop változó ofszetcíme.
        mov  dh,75              ; Oszlop maximális értéke 75.

karert: push bx                 ; BX --> verem.
        push dx                 ; DX --> verem.
        call ascbin             ; Bináris átalakítás.
        pop  dx                 ; Verem --> DX.
        pop  bx                 ; Verem --> BX.

        jc   parhiba            ; Ha az átadott  paraméter nem szám,
        or   ah,ah              ; ill. nagyobb 255-nél, ill. a DH-ban
```

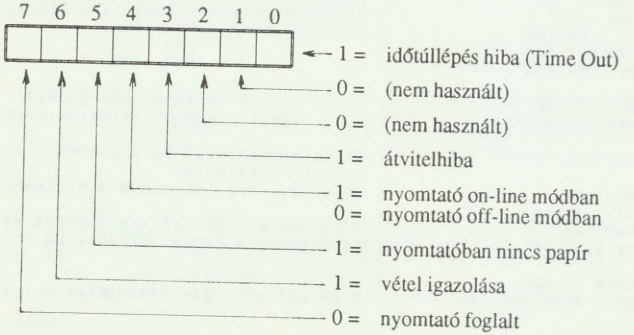
```
            jne   parhiba               ; megengedettnél, akkor paraméter
            cmp   al,dh                 ; átadási hiba.
            ja    parhiba               ;
            mov   [di],al               ; Ha helyes érték : eltárolás.
            jmp   short kovpar          ;

hiba4:      mov   dx,offset txt_4       ; Hibaüzenet : a program már
            jmp   short inakth          ; installálva lett.

parhiba:    mov   dx,offset txt_1       ; Hibaüzenet : érvénytelen
inakth:     mov   al,1                  ; paraméter(ek).
            jmp   irasveg               ;

install:    cmp   bovbill,0             ; Ha nincs billentyű definíció :hiba.
            jne   parhiba               ;
            mov   ax,3516h              ; Megszakítás cím lekérdezés funkció.
            int   21h                   ; 21(h) DOS megszakítás.
                                        ; A megsz. rutin címe ES:BX-be kerül.

            cmp   word ptr es:[bx+2],"ZZ"  ; Annak eldöntése, hogy az
                                           ; ORA1 megsz. rutin aktív-e
            je    hiba4                 ; Igen : hibaüzenet megjelenítés.

            mov   origmszs,es           ; Az eredeti megszakítás eltárolása.
            mov   origmszo,bx

            mov   dx,offset oramsz      ; Az új megszakítás rutin címe
                                        ; DS:DX lesz.
            mov   ax,2516h              ; Megszakítás vektor beáll. funkció.
            int   21h                   ; DOS megszakítás hívása.
            mov   dx,offset txt_2       ; Üzenet kiírása.
            mov   ah,9                  ; Karakterlánc kiírás funkció.
            int   21h                   ; 21(h) DOS megszakítás.

            mov   dx,offset vegcim      ; A programhoz szükséges paragrafusok
            mov   cl,4                  ; (16 bájt) számának kiszámítása.
            shr   dx,cl                 ;
            inc   dx
            mov   ax,3100h              ; Vissza a DOS-hoz 0-ás
            int   21h                   ; hibakóddal (rezidenssé tétel).

irasveg:    mov   ah,9                  ; Karakterlánc kiírás funkció.
            int   21h                   ; DOS megszakítás hívása.
            mov   ah,4Ch                ; Program befejezés funkció.
            int   21h                   ; 21(h) DOS megszakítás.

inic        endp

;— ASCBIN : ASCII (szám) karakter(ek) binárisra alakítása ——————————
;— BE     : DS:SI az ASCII karakterlánc címe (0 zárja le)
;— KI     : AX a bináris érték (max 16 bit, a túlcsordulást
;            az átvitelbit = 1 jelzi)
;— Regiszterek : AX, BX, CX, SI és FLAGS megváltoznak

ascbin   proc near

         xor   bh,bh                ; A felső bájt 0 (nulla).
         mov   cx,10                ; Tízes számrendszer beállítása.
         xor   ax,ax                ; AX = 0.
ascii1:  mov   bl,[si]              ; A következő számjegy, ha ez a 0
         or    bl,bl                ; karakter, akkor kész.
         je    ascii2               ;
```

```
        cmp   bl,"0"              ; Ha nem számjegy ("0"-nál kisebb
        jb    ascii4              ; vagy "9"-nél nagyobb), akkor hiba.
        cmp   bl,"9"              ;
        ja    ascii3              ;
        mul   cx                  ; Az eddigi eredmény * 10.
        jc    ascii4              ; Ha nagyobb, mint 65535 : hiba.
        and   bl,1111b            ; A számjegy binárisra alakítása és
        add   ax,bx               ; az eddigi eredményhez adása.
        inc   si                  ;
        jmp   short ascii1        ; Ugrás a következő számjegyre.

ascii2: clc                       ; Az átalakítás sikeres volt.
        ret                       ;

ascii3: stc                       ; A konvertálás során hiba lépett
ascii4: ret                       ; fel.

ascbin  endp

;— PARAM : A parancssor paramétereinek megállapítása ————————————————
;— BE : DS:0000 = a PSP címe
;— KI : DL = a beolvasott praméterek száma
;— Regiszterek : AX, CX, DX, SI és FLAGS változnak

param proc near

        cld
        xor   dx,dx               ; A beolvasott paraméterek száma.
        mov   si,80h              ; A parancssorbeli karakterek
                                  ; számának a címe a PSP-ben.
        mov   cl,byte ptr [si]    ; A karakterek számának beolvasása,
        or    cl,cl               ; ha nem lett paraméter átadva,
        je    param3              ; akkor vége.

        inc   si                  ; SI a parancssor elejére mutat.
        xor   ch,ch               ; CX a karakterek számát tartalmazza.
param1: lodsb                     ; Karakter AL-be töltése, ha nem üres
        cmp   al," "              ; karakter (betűköz vagy tabulátor),
        je    param4              ; akkor olvasás folytatása.
        cmp   al,9                ;
        je    param4              ;

        or    dh,dh               ; Ha az utolsó karakter nem szóköz
        jne   param2              ; volt, ugrás a következő karakterre.

        inc   dl                  ; DL az azonosított paraméterek
                                  ; számát tartalmazza.
        not   dh                  ; DH jelzi, hogy nem üres a karakter.
        mov   ax,si               ;
        dec   ax                  ; A paraméter címének kiszámítása és
        stosw                     ; bejegyzése a param. táblázatba.

param2: loop  param1              ;
        mov   byte ptr [si],0     ; A paraméter végét 0 karakter jelzi.

param3: ret

param4: or    dh,dh               ; Ha az utolsó kar. üres kar. volt,
        je    param2              ; a következő kar. feldolgozása.

        ;— következő paraméter ——————————————————————————————————————
```

```
                xor   dh,dh                    ; Mivel a karakter üres volt, 0
                mov   byte ptr [si-1],0       ; karakterrel a paraméter végének
                jmp   short param2            ; jelzése, ugrás a köv. param.-re.
param endp

;— VÉGE ————————————————————————————————————————————————————————————

kod     ends
        end   start
```

1.9. A nyomtatót kezelő BIOS funkciók

A PC-hez csatlakoztatott nyomtatóval/nyomtatókkal való kapcsolattartáshoz a BIOS három funkciót bocsát a rendelkezésünkre. Mindhárom funkció a 23 – 17(h) megszakításon keresztül hívható meg. A funkciók BIOS szintről történő meghívásával egyértelműen kiválasztható, hogy a PC-hez a DOS szinten maximum három csatlakoztatható nyomtató közül melyik az, amelyikhez fordulni akarunk. (Amint az I. kötet DOS-szal foglalkozó részében erről már volt szó, DOS szintről ez az egyértelmű kijelölés csak egy külön handle megnyitásával lehetséges. Felhívjuk a figyelmet arra, hogy míg a DOS-ban a lehetséges három nyomtató logikai neve LPT1, LPT2 és LPT3, addig a BIOS ezeket a 0, 1 és 2 sorszámokkal azonosítja.) Mindhárom BIOS funkció meghívásakor ezen sorszámok valamelyikét kell a DX regiszterben átadni. A megszakítás visszatérésekor mindhárom funkció az AH regiszterben szolgáltatja a nyomtató státusz bájtját. A státusz bájt felépítését és az egyes bitek jelentését a 18. ábra mutatja.



18. ábra: A nyomtató státusz bájtja

Az időtúllépés hiba akkor lép fel, ha – azt követően, hogy a BIOS egy meghatározott időn keresztül adatokat próbált a nyomtatóra küldeni – ez az adatokat nem fogadja, vagy foglaltat jelez (7. bit értéke 0). A hiba oka valószínűleg az, hogy a megszakítás meghívásakor a nyomtató nem volt bekapcsolva. Azt, hogy a BIOS mennyi idő eltelte után jelezzen időtúllépési hibát, a BIOS a RAM-ban (0040:0078 – 7B(h) címeken) lévő bájtokból állapítja meg. A BIOS a rendszer indításakor ezekre a tárcímekre beírja a gépbe behelyezhető 3 nyomtató adapterre vonatkozó időtúllépési értékeket (a negyedik bájt nem használt, fenntartott tárhely). Alapértelmezés szerint mind a négy bájt tartalma 20 – 14(h). Ezt az értéket 4-gyel, majd 65536-tal megszorozva megkapjuk a hozzáférési kísérletek számát (ami 5 milliónál több kísérletet jelent). Figyelembe véve, hogy egy-egy kísérletet megvalósító lekérdezési ciklus csak néhány gépi kódú utasításból áll, az ismétlések első pillanatban talán igen nagynak tűnő száma a valóságban csak néhány másodpercet jelent.

A nyomtató különböző állapotai különböző módon állíthatják be a státusz bájt egyes bitjeit. Így például ha a nyomtató nyomtatásra kész, és on-line üzemmódban van, akkor a 7. és a 4. bit értéke 1. Ha most a nyomtató off-line módba kerül (pl. lapdobás miatt), akkor ennek nemcsak az a következménye, hogy a 7. és a 4. bit törlődik, hanem az is, hogy a 3. bit értéke 1 lesz, ami átviteli hibát jelent. Ez a hiba a felhasználói programból lekérdezhető.

Térjünk vissza a BIOS funkciókhoz. Ezek közül a 0. sorszámú funkcióval egy karaktert küldhetünk a nyomtatóra. A kinyomtatandó karakter ASCII-kódját az AL regiszterben kell átadni. A megszakítás visszatérésekor az AH regiszter a státusz bájtot tartalmazza. Megjegyezzük, hogy azoknál a nyomtatóknál, amelyeknek bemeneti pufferük van, a kiküldött karakter először ebbe a pufferbe kerül, és ténylegesen csak akkor íródik a papírra, ha a puffer megtelik, vagy ha a kiküldött karakterláncot egy kocsi vissza és soremelés (CR, LF) karakter zár le.

Az 1. számú funkció a nyomtató inicializálását végzi. Ezt a szolgáltatást a nyomtatóra való első adatküldés előtt célszerű igénybe venni. A megszakítás visszatérésekor most is a státusz bájt kerül az AH regiszterbe.

A 2. számú funkció feladata mindössze annyi, hogy a státusz bájtot beírja az AH regiszterbe (anélkül, hogy a nyomtatót inicializálná).

1.9.1. Programpéldák

A nyomtatót kezelő 17(h) BIOS megszakítás kezelésére 4 példaprogramot mutatunk be. A magasszintű nyelveken írt programok felépítésüket és szerepüket tekintve igen hasonlóak, mindegyik a BIOS által biztosított funkciókat is felhasználva végez tesztnyomtatást a nyomtatás sikerességének/sikertelenségének állandó figyelésével. Az assembler példa a megszakítás vektor nyomtatáshoz tartozó rutinját egy saját rutinra cseréli le.

Az elsőként említett példacsoport programjainak felépítése rendkívül egyszerű. Első (megegyező) lépésük a 0. nyomtató inicializálása a 17(h) megszakítás 2-es funkciója segítségével. Sikertelen inicializálás esetén mindaddig újabb és újabb inicializálási kísérletek történnek, amíg a nyomtató sikeresen nem inicializálódik vagy nem nyomjuk meg az

ESC billentyűt. A sikeres inicializálást követően körülbelül féloldalnyi szöveg kinyomtatására történik kísérlet (ugyanazt a karakterláncot nyomtatják ki 100-szor egymás után a programok). A nyomtatás a karakterlánc éppen nyomtatandó karakterének azonosítása után a 17(h) megszakítás 0-ás funkciójával történik úgy, hogy közben a programok minden egyes karakter nyomtatása után ellenőrzik, sikeres volt-e a művelet. Ha nem, mindaddig nem következik be adatvesztés a nyomtatón, amíg az újabb és újabb nyomtatási kísérletekről az <ESC> billentyű megnyomásával le nem mondunk. A nyomtatásellenőrzés bemutatott módja jól beágyazható felhasználói programokba is, ahol állandó problémát szokott jelenteni, ha pl. nincs nyomtató csatlakoztatva a számítógéphez vagy elfogy a papír nyomtatás közben.

Az esetleges hibaüzenetek a nyomtató státuszbájtjának elemzése alapján kerülnek megjelenítésre. Ezzel kapcsolatban két apró megjegyzést kell tennünk. A nyomtató státuszbájtjának kezelése hardverfüggő, különböző eltérő BIOS-szal rendelkező AT-k esetén pl. bizony előfordult, hogy ugyanaz a tesztprogram ugyanolyan körülmények között eltérő hibaüzenetet eredményezett (a hibás esetek felismerésében nem volt közöttük eltérés). A másik megjegyzés ahhoz az esethez kapcsolódik, amikor a géphez nem csatlakozik nyomtató. Ilyenkor a hiba felismerése szintén mindig rendben megtörténik, de a megjelenő hibaüzenetek minden tendencia nélkül váltogatják egymást.

Az assembly példa kapcsán visszautalnánk a DOS szűrők ismertetésénél bemutatott példaprogramjainkra. A magyar ékezetes karakterek nyomtatásának ott ismertetett folyamatától teljesen eltérő módon oldja meg ezt a problémát az itt bemutatandó assembly program, amely egy tárrezidens megszakítás rutinra irányítja át a BIOS nyomtató kezelő megszakítását (végsősoron ez a program is szűrő funkciót valósít meg, akárcsak a korábban bemutatottak).

A megszakítás rutin csak akkor végez installálása után érdemleges tevékenységet, ha a 17(h) megszakításon belül a 0-ás funkció végrehajtását kéri a rendszer (ez a karakternyomtatás funkció), egyéb esetekben a vezérlés azonnal tovább kerül az eredeti BIOS megszakítás rutinra. A karakter nyomtatási funkció kérése esetén a program elkezdi végigjárni a kétbájtos egységekből álló konverziós kódtáblázatot. A kétbájtos egységek (szavak) második bájtja a konvertálandó karakter, az első bájt pedig a konvertálás eredménye. Amennyiben az éppen nyomtatandó karakter szerepel az átalakítandó karakterek között, a program elvégzi a konverziót (helyettesítést) és kilép a ciklusból. Ha a karaktert nem kell lecserélni, akkor a kódtáblázat végét elérve ér véget a ciklus, ekkor nem történik konverzió. Mindkét esetben a vezérlés visszakerül az eredeti BIOS megszakításra, amely ezek után a korábban megismert módon elvégzi a karakter kinyomtatását.

Fontos része még a programnak, hogy hívásakor megvizsgálja, melyik megszakítás rutin van éppen érvényben. Amennyiben az eredeti, úgy installálódik az új, amennyiben az új, úgy visszaállítódik az eredeti. Egyszerűségén kívül további előnye ennek a megoldásnak, hogy a korábban ismertetettekkel szemben a DOS „PRINT" parancsa „alatt" működik és ezzel hatása használati értékét tekintve egy rendszerszintű új nyomtatási paranccsal egyenértékű.

```
1000 '
1005 '  ╔═══════════════════════════════════════════════════════════╗
1010 '  ║                      N Y O M T A T . B A S                ║
1020 '  ║                                                           ║
1030 '  ║ Funkciója : egy tesztsor 100 alkalommal történő kinyomtatása ║
1040 '  ║             a 0. nyomtatón a nyomtató állapotának állandó ║
1050 '  ║             figyelésével.                                 ║
1060 '  ╚═══════════════════════════════════════════════════════════╝
1070 '
1080 '
1090 DEFINT Q                                   'Q "dummy" egész érték.
1100 KEY OFF : CLS
1110 PRINT : PRINT
1120 PRINT "FIGYELMEZTETÉS :" : PRINT
1130 PRINT "A 60000-es címtől kezdődően betöltött megszakításhívó ru-"
1140 PRINT "tin felülírását elkerülendő a GWBASIC-et GWBASIC /M:60000"
1150 PRINT "paranccsal kell indítania. Ha nem így hívta a GWBASIC-et,"
1160 PRINT "akkor üsse le az <s> billentyűt, egyébként egy tetszőle-"
1170 PRINT "ges másik billentyűt !"
1180 '
1190 Z$ = ""
1200 WHILE Z$ = ""                              'Helytelen indítás
1210     Z$ = INKEY$                            'esetén a program
1220 WEND                                       'leállítása.
1230 IF Z$ = "s" OR Z$ = "S" THEN CLS:END
1240 '
1250 GOSUB 9000                                 'Megszakításhívó rutin
1260 CLS                                        'betöltése.
1270 '
2000 '
2010 '  ┌─────────────────────────────────────┐
2020 '  │          P R O G R A M T Ö R Z S     │
2030 '  └─────────────────────────────────────┘
2040 PRINT
2050 PRINT "Párhuzamos nyomtatón nyomtatási teszt indul ..."
2060 PRINT
2070 NNYOM% = 0
2080 TEXT$  = "A tesztsor százszor nyomtatódik ki !"
2090 HOSSZ% = LEN(TEXT$)
2100 '
2110 GOSUB 3000                                 ' Inicializálási kísérlet,
2120 GOSUB 6000                                 ' ha nem sikeres, üzenet.
2130 IF STATUSZ% = 0 THEN GOTO 2210             ' Ha jó : ugrás a nyomta-
2140 Z$ = ""                                    ' tási ciklusra.
2150 WHILE Z$ = ""
2160     Z$ = INKEY$                            ' Hiba esetén várakozás
2170 WEND                                       ' billentyű lenyomására.
2180 IF Z$ = CHR$(27) THEN GOTO 2300
2190 GOTO 2110                                  ' Újabb próbálkozás.
2200 '
2210 FOR I% = 1 TO 100
2220     GOSUB 4000                             ' Egy sor nyomtatása.
2230     IF STATUSZ% <> 0 THEN I% = 100
2240 NEXT
2250 PRINT : PRINT
2260 IF STATUSZ% = 0 THEN PRINT "A nyomtatás sikeres volt !" : PRINT
2270 IF STATUSZ% > 0 THEN PRINT "A nyomtatás félbeszakítva !" : PRINT
2280 GOTO 2310
2290 '
2300 PRINT : PRINT "A nyomtatás nem volt lehetséges !" : PRINT
2310 END
2320 '
```

```
3000 '
3010 '  A NYOMTATÓ INICIALIZÁLÁSA
3020 '       BE : NNYOM% a nyomtató sorszáma
3030 '       KI : STATUSZ% a művelet hibakódja
3040 '
3050 '
3060 Q = 0
3070 FUN%=2                                    ' Inicializálás funkció.
3080 MSZ%=&H17                                 ' 17(h) megszakítás.
3090 '
3100 CALL RCIM(MSZ%, FUN%, Q, Q, Q, Q, Q, Q, NNYOM%, Q, Q, Q, Q)
3110 '
3120 STATUSZ% = FUN% AND &H29                  ' A visszaadott érték alap-
3130 RETURN                                    ' ján hibakód generálása.
3140 '
4000 '
4010 '  EGY KARAKTERLÁNC KARAKTERENKÉNTI KINYOM-
4020 '  TATÁSA NYOMTATÓ STATUSZ FIGYELÉSSEL
4030 '       BE : TEXT$ a karakterlánc
4040 '       KI : STATUSZ% a művelet hibakódja
4050 '
4060 '
4070 FOR J% = 1 TO HOSSZ%
4080     KAR$ = MID$(TEXT$, J%, 1)             ' Egy karakter elküldése
4090     GOSUB 5000                            ' a nyomtatóra.
4100     GOSUB 6000                            ' Hibaüzenet (ha kell).
4110     IF STATUSZ% = 0 THEN GOTO 4170        ' Ha sikerült : tovább a
4120     Z$ = ""                               ' következő karakterre.
4130     WHILE Z$ = ""
4140        Z$ = INKEY$                        ' Hiba esetén várakozás
4150     WEND                                  ' billentyű lenyomására.
4160     IF Z$ = CHR$(27) THEN J% = HOSSZ%
4170 NEXT J%
4180 RETURN
4190 '
5000 '
5010 '  EGY KARAKTER KINYOMTATÁSA
5020 '        NNYOM% a nyomtató sorszáma
5030 '    KI : STATUSZ% a művelet hibakódja
5040 '
5050 '
5060 Q = 0
5070 KAR% = ASC(KAR$)
5080 FUN%=0                                    ' Kar. nyomtatás funkció.
5090 MSZ%=&H17                                 ' 17(h) megszakítás.
5100 '
5110 CALL RCIM(MSZ%, FUN%, KAR%, Q, Q, Q, Q, Q, NNYOM%, Q, Q, Q, Q)
5120 '
5130 STATUSZ% = FUN% AND &H29                  ' A visszaadott érték alap-
5140 RETURN                                    ' ján hibakód generálása.
5150 '
6000 '
6010 '  HIBAKIIR : A NYOMTATÓ STATUSZ ALAPJÁN
6020 '             HIBAÜZENET MEGJELENÍTÉSE
6030 '        BE : STATUSZ% a nyomtató STATUSZa
6040 '
6050 '
6060 IF STATUSZ% = 0 THEN RETURN
6070 IF (STATUSZ% AND 1) = 0 THEN GOTO 6100
6080    PRINT("Time-out hiba a nyomtatón - ESC kiszállás, ");
6090    PRINT("más billentyűvel újra próbálkozás !") : RETURN
6100 IF (STATUSZ% AND 8) = 0 THEN GOTO 6130
6110    PRINT("I/O hiba a nyomtatón - ESC kiszállás, ");
```

```
6120     PRINT("más billentyűvel újra próbálkozás !") : RETURN
6130 IF (STATUSZ% AND 32) = 0 THEN GOTO 6160
6140     PRINT("Nincs papír a nyomtatóban - ESC kiszállás, ");
6150     PRINT("más billentyűvel újra próbálkozás !") : RETURN
6160 PRINT("Azonosíthatatlan nyomtatási hiba - ESC kiszállás, ");
6170 PRINT("más billentyűvel újra próbálkozás !") : RETURN
6180 RETURN
6190 '
9000 '
9010 '  A MEGSZAKÍTÁS Hívó RUTIN INICIALIZÁLÁSA
9020 '     BE: -
9030 '     KI: RCIM a Rutin CÍMe
9040 '
9050 '
9060 RCIM = 60000!                    'A rutin kezdőcíme a BASIC szegmensben.
9070 DEF SEG                          'A BASIC szegmens beállítása.
9080 RESTORE 9140
9090 FOR I% = 0 TO 160                                    'A rutin betöltése a
9100     READ X% : POKE RCIM + I%, X%                     'tárba.
9110 NEXT
9120 RETURN
9130 '
9140 DATA     85, 139, 236,  30,   6, 139, 118,  30, 139,   4, 232, 140
9150 DATA      0, 139, 118,  12, 139,  60, 139, 118,   8, 139,   4,  61
9160 DATA    255, 255, 117,   2, 140, 216, 142, 192, 139, 118,  28, 138
9170 DATA     36, 139, 118,  26, 138,   4, 139, 118,  24, 138,  60, 139
9180 DATA    118,  22, 138,  28, 139, 118,  20, 138,  44, 139, 118,  18
9190 DATA    138,  12, 139, 118,  16, 138,  52, 139, 118,  14, 138,  20
9200 DATA    139, 118,  10, 139,  52,  85, 205,   0,  93,  86, 156, 139
9210 DATA    118,  12, 137,  60, 139, 118,  28, 136,  36, 139, 118,  26
9220 DATA    136,   4, 139, 118,  24, 136,  60, 139, 118,  22, 136,  28
9230 DATA    139, 118,  20, 136,  44, 139, 118,  18, 136,  12, 139, 118
9240 DATA     16, 136,  52, 139, 118,  14, 136,  20, 139, 118,   8, 140
9250 DATA    192, 137,   4,  88, 139, 118,   6, 137,   4,  88, 139, 118
9260 DATA     10, 137,   4,   7,  31,  93, 202,  26,   0,  91,  46, 136
9270 DATA     71,  66, 233, 108, 255
```

```
/*
/*
/*                          N Y O M T A T . C
/*
/*   Funkciója : egy tesztsor 100 alkalommal történő kinyomtatása
/*               a 0. nyomtatón a nyomtató állapotának állandó
/*               figyelésével.
/*
/*
/*
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/

#include <dos.h>
#include <io.h>

#define FALSE  0
#define TRUE   1

#define byte  unsigned char

char nyomtatas = TRUE;

/*
/*   HIBAKIIR : A NYOMTATÓ STATUSZ ALAPJÁN
/*              HIBAÜZENET MEGJELENÍTÉSE
/*
*/
*/
*/
*/

void HIBAKIIR(statusz)
byte statusz;

{
if (!statusz) return;
if      (statusz &  1) printf ("Time-out hiba a nyomtatón");
else if (statusz &  8) printf ("I/O hiba a nyomtatón");
else if (statusz & 32) printf ("Nincs papír a nyomtatóban");
else                   printf ("Azonosíthatatlan nyomtatási hiba");
printf (" - ESC kiszállás, más billentyűvel újra próbálkozás !");
return;
}

/*
/*   KARNYOM : EGY KARAKTER KINYOMTATASA
/*
*/
*/
*/

byte KARNYOM(Karakter)
char Karakter;

{
union REGS regiszter;                    /* Regiszter union deklaráció. */
byte statusz;

regiszter.h.ah = 0;                      /* Nyomtatás funkció -> AH.    */
regiszter.h.al = Karakter;               /* Karakter kód      -> AL.    */
regiszter.x.dx = 0;                      /* Nyomtató sorszám  -> DX.    */
int86(0x17, &regiszter, &regiszter);     /* 17(h) megszakítás.          */
statusz = (regiszter.h.ah & 0x29);       /* A vizsgálandó 3 bit által   */
                                         /* képzett maszk 1+8+32=0x29   */
HIBAKIIR (statusz);                      /* A visszaadott érték alap-   */
                                         /* ján hibaüzenet kiadása.     */
return !statusz;
}
```

```
/*
/*   NYOM : EGY KARAKTERLÁNC KARAKTERENKÉNTI
/*          KINYOMTATÁSA NYOMTATÓ STATUSZ
/*          FIGYELÉSSEL
/*
/*

void NYOM(Karlanc)
char *Karlanc;

{
while (nyomtatas && *Karlanc)          /* A karakterlánc karaktere- */
   {                                   /* inek nyomtatása.          */
   if (KARNYOM(*Karlanc))              /* Ha hiba lép fel, ESC-től  */
      *Karlanc++;                      /* eltérő billentyű megnyo-  */
   else if (getch () == 27)            /* mására a program újra és  */
      nyomtatas = FALSE;               /* újra megkísérli az adott  */
   }                                   /* karaktert kinyomtatni.    */
}                                      /* Egyetlen kar. sem vész el */

/*
/*   NY_INIC : A 0. NYOMTATÓ INICIALIZÁLÁSA
/*

byte NY_INIC()

{
union REGS regiszter;                  /* Regiszter union deklaráció. */
byte statusz;

regiszter.h.ah = 1;                    /* Inicializálás fun.  -> AH.  */
regiszter.x.dx = 0;                    /* Nyomtató sorszám    -> DX.  */
int86(0x17, &regiszter, &regiszter);   /* 17(h) megszakítás.          */
statusz = (regiszter.h.ah & 0x29);
HIBAKIIR (statusz);                    /* A visszaadott érték alap-   */
                                       /* ján hibaüzenet kiadása.     */
return !statusz;
}

/*
/*   NY_UZEMKESZ : A NYOMTATÓ ÜZEMKÉPESSÉGÉ-
/*                 NEK MEGÁLLAPÍTÁSA, INICI-
/*                 ALIZÁLÁS
/*

NY_UZEMKESZ()

{
while (TRUE)
   {
   if (NY_INIC())
      return TRUE;
   else if (getch () == 27)
      return FALSE;
   }
}
```

```
/*
/*                    F Ő P R O G R A M                    */
/*                                                         */
                                                           */

void main()
{
int i;

printf("Párhuzamos nyomtatón nyomtatási teszt indul ...");
if (NY_UZEMKESZ())
   {
   for (i=0; nyomtatas && i<100; i++)
      NYOM ("A tesztsor százszor nyomtatódik ki !");
   if (nyomtatas)
      printf ("A nyomtatás sikeres volt !");
   else
      printf ("A nyomtatás félbeszakítva !");
   }
else
   printf ("A nyomtatás nem volt lehetséges !");
}
```

```
{ ┌──────────────────────────────────────────────────────────────┐ }
{ │                                                              │ }
{ │                        N Y O M T A T . P A S                 │ }
{ │                                                              │ }
{ │  Funkciója : egy tesztsor 100 alkalommal történő kinyomtatása │ }
{ │              a 0. nyomtatón, az esetleges nyomtatóhiba kijel- │ }
{ │              zése.                                           │ }
{ │                                                              │ }
{ └──────────────────────────────────────────────────────────────┘ }

program NYOMTAT;

type dosreg     = record
                  ax, bx, cx, dx, bp, di, si, ds, es, flags : integer;
                  end;

     nyomsor    = string[255];

var statusz     : byte;
    nyomtatas   : boolean;
    z           : char;          { A várakozási állapotokhoz szükséges. }
    i           : integer;

{ ┌──────────────────────────────────────────┐ }
{ │ HIBAKIIR : A NYOMTATÓ STATUSZ ALAPJÁN    │ }
{ │            HIBAÜZENET MEGJELENÍTÉSE      │ }
{ └──────────────────────────────────────────┘ }

procedure HIBAKIIR(statusz : byte);

label vege;

begin
if (statusz and 1) <> 0 then
   begin
   write('Time-out hiba a nyomtatón - ESC kiszállás, ');
   writeln('más billentyűvel újra próbálkozás !');
   goto vege
   end;
if (statusz and 8) <> 0 then
   begin
   write('I/O hiba a nyomtatón - ESC kiszállás, ');
   writeln('más billentyűvel újra próbálkozás !');
   goto vege
   end;
if (statusz and 32) <> 0 then
   begin
   write('Nincs papír a nyomtatóban - ESC kiszállás, ');
   writeln('más billentyűvel újra próbálkozás !');
   goto vege
   end;
if (statusz <> 0) then
   begin
   write('Azonosíthatatlan nyomtatási hiba - ESC kiszállás, ');
   writeln('más billentyűvel újra próbálkozás !');
   goto vege
   end;
vege:
end;
```

```
{
{  KARNYOM : EGY KARAKTER KINYOMTATASA                    }
{                                                         }
{                                                         }

function KARNYOM(karakter : char) : boolean;

var regiszter : dosreg;                    { Regiszter rekord deklaráció. }
    statusz   : byte;

begin

regiszter.ax := ord(karakter);             { Nyomtatás funkció -> AH.     }
                                           { Karakter kód      -> AL.     }
regiszter.dx := 0;                         { Nyomtató sorszám  -> DX.     }
intr($17, regiszter);                      { 17(h) megszakítás.           }
statusz := hi(regiszter.ax) and $29;       { A vizsgálandó 3 bit által    }
                                           { képzett maszk 1+8+32=0x29    }
HIBAKIIR(statusz);                         { A visszaadott érték alap-    }
                                           { ján hibaüzenet kiadása.      }
KARNYOM := statusz = 0;
end;
```

```
{
{  NYOM : EGY KARAKTERLANC KARAKTERENKÉNTI                }
{         KINYOMTATASA NYOMTATÓ STATUSZ                   }
{         FIGYELÉSSEL                                     }
{                                                         }

procedure NYOM(karlanc : nyomsor);

var i : integer;

begin
i := 1;
while (nyomtatas and (i<=length(karlanc))) do
begin
   if (KARNYOM(karlanc[i])) then           { A karakterlánc karaktere-    }
      i := succ(i)                         { inek nyomtatása.             }
   else                                    { Ha hiba lép fel, ESC-től     }
      begin                                { eltérő billentyű megnyo-     }
         read(kbd, z);                     { mására a program újra és     }
         if z = chr(27) then               { újra megkísérli az adott     }
            nyomtatas := FALSE;            { karaktert kinyomtatni.       }
      end                                  { Egyetlen kar. sem vész el    }
end
end;
```

```
{
{  NY_INIC : A 0. NYOMTATÓ INICIALIZALASA                 }
{                                                         }

function NY_INIC : boolean;

var regiszter : dosreg;                    { Regiszter rekord deklaráció. }
    statusz   : byte;

begin
regiszter.ax := $0100;                     { Inicializálás fun. -> AH.    }
regiszter.dx := 0;                         { Nyomtató sorszám   -> DX.    }
```

```
intr($17, regiszter);                    { 17(h) megszakítás.          }
statusz := hi(regiszter.ax) and $29;
HIBAKIIR(statusz);                       { A visszaadott érték alap-   }
                                         { ján hibaüzenet kiadása.     }
NY_INIC := statusz = 0;
end;

{                                                                      }
{  NY_UZEMKESZ : A NYOMTATÓ UZEMKÉPESSÉGÉ-                             }
{                NEK MEGALLAPITASA, INICI-                             }
{                ALIZALAS                                              }
{                                                                      }

function NY_UZEMKESZ :boolean;

label start;

begin
start:
if NY_INIC then
     NY_UZEMKESZ := TRUE
else begin
     read(kbd, z);
     if z = chr(27) then NY_UZEMKESZ := FALSE
                    else goto start
     end;
end;

{                                                                      }
{  F Ő P R O G R A M : NYOMTATAS                                        }
{                                                                      }

begin

clrscr;
writeln('Párhuzamos nyomtatón nyomtatási teszt indul ...');
writeln;
if NY_UZEMKESZ then
   begin
   nyomtatas := TRUE;
   for i := 1 to 100 do
      begin
         NYOM('A tesztsor százszor nyomtatódik ki !');
         if not(nyomtatas) then
            i := 100
      end;
   if nyomtatas then writeln('A nyomtatás sikeres volt !')
                else writeln('A nyomtatás félbeszakítva !');
   end
else
   begin
      writeln;
      write ('A nyomtatás nem volt lehetséges !')
   end;
writeln;
end.
```

```
;
;
;                          K A R K O N V . A S M
;
;   Funkciója: RX - 80 nyomtatón a magyar karakterkészletnek meg-
;              felelõ nyomtatási kép kialakítása az eredeti nyom-
;              tató megszakítás átdefiniálásával.
;
;
;— KÓD ------------------------------------------------------------------

kod         segment para 'CODE'         ; A kódszegmens definiálása.

            org 100h                    ; A program a PSP után álló elsõ
                                        ; címen kezdõdik.

            assume cs:kod, ds:kod, es:kod, ss:kod

                                        ; Csak egy, közös szegmens van.

start:      jmp inicial                 ; Ugrás a programhoz.

;— ADATOK ---------------------------------------------------------------

origmsz  equ this dword                 ; Az eredeti megszakítás címe :
origmszo dw (?)                         ;      ofszetcím,
origmszs dw (?)                         ;      szegmenscím.

                                        ; Új                    Eredeti
                                        ; karakterek.           karakterek.
                                        ;
kodtab      db  235, 160                ; "ò"       <———        "á"
            db  236, 130                ; "œ"       <———        "é"
            db  237, 161                ; "ø"       <———        "í"
            db  239, 148                ; "Ô"       <———        "ö"
            db  242, 147                ; "ł"       <———        "õ"
            db  238, 162                ; "É"       <———        "ó"
            db  246, 129                ; "ü"       <———        "u"
            db  247, 150                ; "ž"       <———        "û"
            db  243, 163                ; "ś"       <———        "ú"
            db  224, 143                ; "α"       <———        "Á"
            db  225, 144                ; "ß"       <———        "É"
            db  226, 140                ; "Γ"       <———        "í"
            db  228, 153                ; "Σ"       <———        "ö"
            db  229, 167                ; "σ"       <———        "õ"
            db  227, 149                ; "π"       <———        "ó"
            db  232, 154                ; "Φ"       <———        "Ü"
            db  233, 152                ; "Θ"       <———        "Û"
            db  230, 151                ; "µ"       <———        "ú"
                                        ;
            db  0                       ; A kódtáblázat vége.

;— Az új reziđens nyomtató megszakítás ---------------------------------

ekezet      proc far

            jmp  short cim_1

            db "XX"                     ; A program neve.

cim_1:      or   ah, ah                 ; Ha nem karakternyomtatás funkció
            jne  cim_5                  ; (AH = 0), az eredeti megszakításra
                                        ; kell ugrani.
```

```
        pushf                   ; A regiszterek mentése.
        push bx
        push si
        push ds

        push cs                 ; CS --> DS.
        pop  ds

;-- Módosítandó karakter ? ------------------------------------------

        cld
        mov  si,offset kodtab   ; A kódtáblázat címe.
        mov  bl,al              ; Karakter --> BL.
cim_2:  lodsw                   ; A régi (AH) és az új (AL) kód
                                ; betoltése.
        or   al,al              ; Táblázat vége ?
        je   cim_3              ; Igen : nem módosítandó karakter.
        cmp  ah,bl              ; A táblázatban van a karakter ?
        jne  cim_2              ; Nem  : tovább vizsgálni.
        jmp  short cim_4        ; Igen : módosítani.

cim_3:  mov  al,bl              ; Karakter --> AL.
cim_4:  mov  ah,0               ; A funkciószám vissza AH-ba.
        pop  ds                 ; A regiszterek visszatoltése.
        pop  si
        pop  bx
        popf

cim_5:  jmp  cs:[origmsz]       ; Ugrás a régi megszakításra.

ekezet  endp

memvege equ this byte           ; A kód eddig terjedő része rezidens.

;-- ADATOK ----------------------------------------------------------

CR      equ 13                  ; Kocsivissza ASCII kódja.
LF      equ 10                  ; Sorvemelés ASCII kódja.
KEND    equ "$"                 ; Karakterlánc vége jelzés.

txt_1   db CR,LF,"Az új nyomtató megszakítás installálva.",CR,LF
        db "Inaktiválása egy újbóli felhívással történik.",CR,LF,KEND
txt_2   db 13,10,"Az új nyomtató megszakítás inaktiválva.",CR,LF,KEND

;-- FŐPROGRAM -- a DOS FELULIRHATJA ---------------------------------

inicial label near

        mov  ax,3517h           ; Megszakítás cím lekérdezés funkció.
        int  21h                ; 21(h) DOS megszakítás.
                                ; A megsz. rutin címe ES:BX-be kerul.

        cmp  word ptr es:[bx+2],"XX"  ; Annak eldöntése, hogy a
                                      ; KARKONV megsz. rutin aktív-e.

        jne  install            ; Nem : installálás.

        ;-- Inaktiválás -------------------------------------------

        mov  dx,es:origmszo     ; A régi megszakítás visszaállítása,
        mov  ax,es:origmszs     ; a címe DS:DX lesz.
        mov  ds,ax
        mov  ax,2517h           ; Megszakítás vektor beáll. funkció.
        int  21h                ; DOS megszakítás hívása.
```

```
        mov   ah,49h                ; Memória felszabadítás funkció.
        int   21h                   ; 21(h) DOS megszakítás.
        push  cs                    ; CS --> DS.
        pop   ds

        mov   dx,offset txt_2       ; Üzenet kiírása.
        mov   ah,9                  ; Karakterlánc kiírás funkció.
        int   21h                   ; 21(h) DOS megszakítás.

        mov   ax,4C00h              ; Vissza a DOS-hoz 0-ás
        int   21h                   ; hibakóddal.

        ;— Installálás ————————————————————————————————————————

install label near

        mov   ax,3517h              ; Megszakítás cím lekérdezés funkció.
        int   21h                   ; 21(h) DOS megszakítás.
                                    ; A megsz. rutin címe ES:BX-be kerül.
        mov   origmszs,es           ; A megszakítás címének eltárolása.
        mov   origmszo,bx

        mov   dx,offset ekezet      ; Az új megszakítás rutin címe
                                    ; DS:DX lesz.
        mov   ax,2517h              ; Megszakítás vektor beáll. funkció.
        int   21h                   ; DOS megszakítás hívása.

        mov   dx,offset txt_1       ; Üzenet kiírása.
        mov   ah,9                  ; Karakterlánc kiírás funkció.
        int   21h                   ; DOS megszakítás hívása.

        mov   dx,offset memvege     ; A paragrafusok számának kiszámítása
        mov   cl,4                  ; amelyeket a program igénybe vesz.
        shr   dx,cl
        inc   dx
        mov   ax,3100h              ; Vissza a DOS-hoz 0-ás
        int   21h                   ; hibakóddal (rezidenssé tétel).

;— VÉGE ————————————————————————————————————————————————————

Kod     ends
        end   start
```

1.10. Az időt és a dátumot kezelő BIOS funkciók

Az időt és a dátumot kezelő BIOS funkciók az 1A(h) BIOS megszakításon keresztül hívhatók meg. Míg a PC-n és az XT-n csak két funkció áll rendelkezésünkre, az AT-n ezek száma 8-ra bővült. A szolgáltatásoknak ezt a szélesebb körét az AT-vel bevezetett úgynevezett valós idejű óra (RTC, real-time clock) teszi lehetővé, amely képes arra, hogy egy, a számítógép központi RAM tárától független, telepről táplált RAM-ban – többek között – a beállított dátumot és időt „megőrizze". Mivel ezt a RAM tárat kezelő áramkört a behelyezett telep folyamatosan táplálja, az általa megvalósított óra akkor is „ketyeg", amikor a számítógépet kikapcsoljuk. Az idő mérését egy megszakítás rutin

végzi a következők szerint. A számítógépben lévő kvarc oszcillátor a rendszer belső órajelét előállító áramkörön keresztül 1.193180 MHz frekvenciájú jelet küld a 8253 jelű IC (Timer/Counter) bemenetére. Ez utóbbi a beérkező impulzusokat számlálja, és amikor ezek száma eléri a 65536-ot, akkor kiváltja a 8-as számú, órajeladónak (angolul timer interruptnak vagy timer ticknek) nevezett hardver megszakítást, majd ismét 0-ról kezdi a számlálást. A fenti adatok alapján egy ilyen megszakítás jó közelítéssel másodpercenként 18.2-szer aktiválódik, amelyet a 8259 jelű megszakításvezérlő a CPU-hoz továbbít. Mivel a megszakítás kiváltásának ez az ismétlődési gyakorisága független a rendszer órafrekvenciájától, jól használható az idő mérésére. A 8-as megszakítás a visszatérésekor még meghívja a megszakítási vektortábla 1C(h) vektorával címzett ROM BIOS megszakítási rutint, amely így másodpercenként 18.2-szer aktiválódik (periodikus megszakítás). Az idő méréséhez már csak az kell, hogy ez a rutin – amely a BIOS betöltésekor mindössze egy IRET utasítást tartalmaz – alkalmas módon kezeljen egy számlálót. Az AT-ben ettől annyiban tér el a dátum és az idő kezelése, hogy bekapcsoláskor az adatokat az RTC-ből veszi, és a DOS-nak egy ebből számított 4 bájtos számot ad át a BIOS változó területen. Az innen való továbbszámlálás már a 8-as megszakítással történik.

Az idő alapvető kezelésére két funkció áll rendelkezésünkre: a 0. számú funkcióval kiolvasható, az 1. számúval pedig beállítható (írható) az aktuális idő. Ennek megfelelően az idő olvasásához az AH regiszterben a 0 értéket kell átadni. Az 1A(h) megszakítás visszatérésekor az aktuális időt – kódolt alakban – a CX és a DX regiszterekbe írja vissza. Ez a két regiszter együttesen egy 32 bites számlálót képez, amelyből a CX a felső, DX az alsó 16 bitet ábrázolja. A számláló értékét a BIOS a Timer/Counter által kiváltott 8-as megszakítás minden egyes észlelésekor – azaz másodpercenként 18.2 alkalommal – 1-gyel megnöveli. Ennek a 32 bites számlálónak az értékét úgy számíthatjuk át másodpercekre, hogy a felső 16 biten (a CX regiszterben) lévő értéket megszorozzuk 65536-tal, ehhez hozzáadjuk az alsó 16 biten (a DX regiszterben) lévő értéket, majd az így kapott összeget elosztjuk 18.2-del. A számítás eredménye a kiolvasott idő másodpercekben, ami most már egyszerűen visszavezethető percekre és órákra. Az így kiszámított időt természetesen másként kell értelmezni a PC/XT és az AT esetén. Míg ugyanis a BIOS a PC/XT gépeknél a CX – DX számláló értékét a rendszer betöltésekor 0 értékre állítja, és az idő múlását ettől a pillanattól kezdve számolja, addig az AT-nél a számlálóba (a szükséges átszámításokat elvégezve) a valós idejű órában tárolt időt tölti be, és innen kezdi (vagyis folytatja) a számlálást. Ebből következik, hogy az AT minden bekapcsolásakor a valós időt, míg a PC/XT – hacsak be nem állítjuk a megfelelő funkcióval -, a bekapcsolás óta eltelt időt tartalmazza.

A funkció a megszakítás visszatérésekor a fent ismertetett számlálón túl még az AL regiszterben is visszaad egy értéket. Ha ez az érték 0, akkor ez azt jelenti, hogy az utolsó rendszerindítás, időbeállítás vagy kiolvasás óta nem telt el 24 óra, azaz nem haladtuk meg az éjfélt. Ha az AL értéke 0-tól különböző, akkor az utolsó rendszerindítás óta már legalább egyszer elmúlt éjfél. (Arra azonban nem lehet következtetni, hogy hányszor.)

A számláló értékének órákra, percekre és másodpercekre való átszámítását megtakaríthatjuk a DOS 21(h) megszakítás 2C(h) funkciójának meghívásával. Ez a funkció ugyanis – miután meghívta a BIOS 1A(h) megszakítás 0. számú funkcióját – éppen ezt az átszámítást végzi el.

Az idő alapvető kezelésére szolgáló másik funkció az 1. számú funkció, amellyel az aktuális időt tartalmazó számlálót egy meghatározott értékre lehet beállítani. A számlálót ennél a funkciónál is a CX és a DX regiszter képezi a lekérdező funkciónál megismert módon. A beállítandó óra, perc és másodperc átszámítását itt is megtakaríthatjuk, ha a BIOS megszakítás helyett a DOS 21(h) megszakítás 2D(h) funkcióját vesszük igénybe az idő beállításához. Ez a DOS funkció ugyanis éppen ezt az átszámítást végzi el, majd meghívja a BIOS funkciót, és az átadott értéknek megfelelően beállítja a számlálót. A BIOS funkció meghívásakor az éjfél átlépését mutató jelzőbit törlődik.

A következőkben ismertetésre kerülő 6 funkció csak az AT-n hívható. A PC/XT-n való meghívásuk semmiféle következménnyel sem jár (még az átvitelbitet sem állítják át). Ezért azoknak a programoknak, amelyek e funkciók valamelyikét használni akarják, először meg kell győződniük arról, hogy AT-n futnak, és csak ezt követően vehetik igénybe ezeket a szolgáltatásokat.

Mind a hat funkcióra vonatkozik, hogy az időt és a dátumot BCD (binárisan kódolt decimális) alakban kezeli. Ennek az a lényege, hogy egy bájt egy kétjegyű decimális számnak felel meg, ahol a bájt felső fél bájtja (nibble) a kétjegyű szám magasabb helyértékű számjegyét, az alsó fél bájtja pedig az alacsonyabb helyértékű számjegyét ábrázolja. Ugyancsak közös mind a hat funkcióban, hogy a megszakításból való visszatéréskor az átvitelbit 1 értéke azt jelenti, hogy a művelet végrehajtása sikertelen volt (mert pl. a valós idejű órát tápláló telep lemerült).

A 2. számú funkcióval a valós idejű órában tárolt időpontot olvashatjuk ki. A funkció végrehajtását követően a CH regiszter az órák, a CL regiszter a percek, a DH regiszter pedig a másodpercek számát tartalmazza. Érdekes jelentése van a DL regiszternek. Ha a regiszter tartalma 0, akkor a valós idejű óra – mindaddig, míg a telep le nem merül, vagy az órát át nem állítjuk – „normál" módon számolja az idő múlását. Ha viszont a regiszter értéke 1, akkor ez azt jelenti, hogy a valós idejű óra azt is „tudja", hogy – a nemzetközi gyakorlatnak megfelelően – tavasszal, a nyári időszámítás megkezdésekor az órákat 1 órával előre, ősszel pedig a „normál" időszámításra való visszatéréskor 1 órával vissza kell állítani, és ezeket az átállításokat automatikusan elvégzi. Az IBM PC AT Műszaki Kézikönyve szerint az átállítások időpontja: április utolsó vasárnapja, amikor hajnali 1 óra 59 perc 59 másodperc után 3 óra 0 perc 0 másodperc következik, valamint október utolsó vasárnapja, amikor 1 óra 59 perc 59 másodperc után 1 óra 0 perc 0 másodperc következik.

Hasonló a regiszterek kiosztása a 3. számú funkciónál is, amely az aktuális idő beállítását végzi. Az órák számát a CH, a percek számát a CL és a másodpercek számát a DL regiszterbe kell beírni (szintén BCD alakban). A DL regiszterben most is megadhatjuk, hogy figyelembe vegye-e az óra a nyári időszámításra való áttérést (DL = 1) vagy ne.

A 4. és 5. számú funkciók a valós idejű órában tárolt dátumot kezelik. A dátum olvasására a 4. számú funkció alkalmas. A funkció végrehajtását követően a CH regiszter az évszám első két számjegyét (vagyis az évszázadot, ami 19 vagy 20 lehet), a CL regiszter az évszám második két számjegyét (pl. 89-et), a DH a hónap, a DL pedig a hónap napjának számát tartalmazza. Bár a valós idejű óra a hét napjának számát is tartalmazza (1 = vasárnap, 2 = hétfő stb.), ez a szám ezzel a funkcióval nem kérdezhető le (de – mint majd látni fogjuk – egyéb szoftver úton hozzá lehet férni). Ugyanilyen regiszterkiosztással állítható be a dátum, azzal a különbséggel, hogy ehhez az AH regiszterben az 5-ös értéket kell átadni.

A 6. számú funkcióval egy riasztási – vagy finomabban fogalmazva ébresztési – időpont állítható be. Mivel a funkció a riasztásnak csak órára, percre és másodpercre történő beállítását teszi lehetővé, a riasztás csak az aktuális napon érvényes. Ha a valós idejű óra eléri a riasztási időpontot, akkor meghív egy BIOS rutint, amely a maga részéről meghívja a 4A(h) megszakítást. A rendszer indításakor ez a megszakítási rutin mindössze egy IRET utasítást tartalmaz, vagyis alapesetben a riasztási idő elérésekor a rutin azonnal visszatér a meghívójához anélkül, hogy bármi egyebet tenne. Egy felhasználói program ezt a megszakítást felhasználhatja arra, hogy a saját rutinjára mutasson, amely az aktiválásakor pl. megszólaltja a gép hangszóróját. A funkció meghívásakor a riasztási időpont óráját, percét és másodpercét rendre a CH, CL és a DH regiszterekbe kell beírni.

Egyidejűleg mindig csak egy riasztási időpont lehet érvényben. Ez azt jelenti, hogy ha a 6. funkcióval akkor akarunk egy riasztási időt beállítani, amikor egy előzőleg beállított időpont még érvényben van, akkor a funkciót nem sikerül végrehajtani, és az átvitelbit értéke 1 lesz. Az érvényben volt riasztás továbbra is érvényben marad. Ha azt akarjuk, hogy ez megváltozzon vagy érvényét vessze, akkor ehhez a 7. számú funkciót kell igénybe venni. Ehhez csak a funkció számát kell az AH regiszterben átadni, majd természetesen meg kell hívni az 1A(h) megszakítást. Ezzel a beállított riasztási időpontot töröltük, és most már beállítható egy új időpont.

1.11. A BIOS változói

Az előző alfejezetekben már többször szóba kerültek a BIOS változók. Tudjuk, hogy a BIOS-t alkotó programok a számítógépnek a csak olvasható, és a felhasználói program által nem megváltoztatható ROM tárában helyezkednek el. Ahhoz azonban, hogy ezek a programok megfelelően működhessenek, nyilvánvalóan rendelkezniük kell olyan tárterülettel is, ahol a futásuk során ideiglenesen különböző értéket tárolhatnak, ezek változását figyelhetik, összehasonlításokat végezhetnek, bizonyos paramétereket a környezet igényeihez illeszthetnek stb. Röviden: szükségük van olyan tárterületre, ahol az általuk használt értékek futás közben változhatnak. Mivel ilyen tárterület csak a RAM-ban van, egyenesen következik, hogy BIOS-nak a RAM egy részét is igénybe kell vennie a saját céljaira. A BIOS ehhez a RAM 00400 – 004FF(h) abszolút tárcímek közötti, 256 bájtos területet foglalja le. Ezt a területet a BIOS munkaterületének nevezik. (Ezen kívül a BIOS még a megszakítási vektortábla egy részét is használja, de ezt ebben az összefüggésben nem tekintjük a munkaterület részének.) A BIOS munkaterületén tárolt változók többségéhez BIOS funkciókkal és természetesen közvetlenül is hozzá lehet férni, sőt – miután RAM-ban vannak – akár meg is változtathatók. A közvetlen hozzáférés azonban – még ha esetleg valamelyest sebességnövekedést is eredményez – semmiképpen sem ajánlott. A BIOS különböző verziói ugyanis egymástól eltérően használhatják a munkaterület különböző részeit, így az a program, amely egy konkrét tárcímen lévő értéket használ fel, nem biztos, hogy azt is találja ott, amire számít. Ezért azok a programok,

amelyeknek használniuk kell a BIOS szintet, ezekhez a változókhoz mindig csak a megfelelő BIOS funkciókon keresztül forduljanak.

A következőkben megnézzük, hogy milyen változók találhatók a BIOS munkaterületén. Az egyes változók tárcímei – a szakirodalomban általánosan használt gyakorlatnak megfelelően – a munkaterület 0040(h) szegmenscímével viszonyított mindenkori ofszetcímeket jelentenek. Eszerint a 10(h) ofszetcímű változó szegmentált címe 0040:0010(h).

00(h) – 07(h)

A BIOS betöltésekor egy BIOS rutin feltérképezi a számítógép konfigurációját. Ennek során – többek között – azt is számba veszi, hogy hány soros interfész kártya (RS232) található a gépben. A rutin a kártyák alapportjainak a címét erre a 4 szó hosszúságú területre írja be. Mindegyik portcím kétbájtos cím, amely a tárban a szokásos ábrázolásmód szerint helyezkedik el: a kisebb tárcím a port alsó, a nagyobb tárcím a felső bájtját tartalmazza. A 00(h) – 07(h) területen összesen 4 interfész kártya alapportjának címe tárolható, azaz a gépbe a BIOS részéről maximum 4 soros interfész csatlakoztatható.

08(h) – 0F(h)

A soros interfész kártyákhoz hasonlóan a csatlakoztatható (maximum) 4 nyomtatóadapter alapportjainak címei.

10(h) – 11(h)

A BIOS betöltésekor egy BIOS rutin feltérképezi a számítógép konfigurációját, és az ezzel kapcsolatos információkat egy 16 bites szóban tárolja. Ezt a szót írja be a 10(h) – 11(h) tárcímekre, ahonnan azt a BIOS 11(h) megszakítás meghívásával kiolvashatjuk.

12(h)

Belső tesztelési célokra fenntartott terület.

13(h) – 14(h)

A RAM tár mérete kbájtokban (nem tartalmazza a video RAM területét és az AT opcionális, 1 Mbájt feletti tárbővítését). A változó értéke a BIOS 12(h) megszakításával kérdezhető le.

15(h) – 16(h)

Belső tesztelési célokra fenntartott terület.

17(h)

A billentyűzet elsődleges státusz bájtja (felépítését lásd a billentyűzetről szóló 1.8.3. pont 17.a) ábráján). A változó értéke a BIOS 16(h) billentyűzetmegszakítás 2. számú funkciójával kérdezhető le.

18(h)

A billentyűzet másodlagos státusz bájtja (felépítését lásd a billentyűzetről szóló 1.8.3. pont 17.b) ábráján). A változó értéke BIOS megszakítással nem kérdezhető le.

19(h)

Billentyűzetkezelési célokra fenntartott terület.

1A(h) – 1B(h)

Mutató, amely a billentyűzet pufferén belül a „puffer eleje" címre mutat. A billentyűzet pufferét olvasó BIOS rutin erről a címről olvassa a billentyű kódját. (Lásd a billentyűzet pufferéről szóló 1.8.2. pontot és a 16. ábrát.)

1C(h) – 1D(h)

Mutató, amely a billentyűzet pufferén belül a „puffer vége" címre mutat. A billentyűzet pufferébe író BIOS rutin erre a címre írja be a billentyű kódját. (Lásd a billentyűzet pufferéről szóló 1.8.2. pontot és a 16. ábrát.)

1E(h) – 3D(h)

A billentyűzet pufferének területe. A puffer felépítésére és működésére vonatkozóan lásd az 1.8.2. pontot és a 16. ábrát.

3E(h)

Az itt tárolt bájt alsó 4 bitje a PC-hez csatlakoztatható (maximum) 4 hajlékonylemezes meghajtóra vonatkozik (0. bit = A: jelű meghajtó stb.), és azt jelzi, hogy az illető meghajtó igényel-e előzetes kalibrálást. Erre többnyire akkor van szükség, amikor a meghajtóra kiadott írási, olvasási vagy keresési művelet sikertelen volt (mert pl. nem volt lemez a meghajtóban). Ebben az esetben az illető meghajtóra vonatkozó bit értéke 0.

3F(h)

Az itt tárolt bájt alsó 4 bitje is a PC-hez csatlakoztatható (maximum) 4 hajlékonylemezes meghajtóra vonatkozik (0. bit = A: jelű meghajtó stb.), és az illető meghajtó motorjának állapotát jelzi (a bit 1 értéke forgó motort jelez).

40(h)

Az ezen a tárcímen lévő bájt egy numerikus értéket tartalmaz, amely azt adja meg, hogy az utolsó lemezműveletet követően mennyi idő elteltével kell a lemezmeghajtó motorját kikapcsolni. Mivel a BIOS egyidőben mindig csak egy lemezmeghajtóhoz tud fordulni, az itt tárolt érték mindig arra a meghajtóra vonatkozik, amelyen az utolsó műveletet végezte. A BIOS közvetlenül az utolsó művelet befejeződése után a 25(h) (decimális 37) értéket írja a bájtba, majd az értéket a 8-as időzítő (timer) megszakítás minden egyes aktíválásakor, azaz másodpercenként kb. 18.2-szer 1-gyel csökkenti. Amikor az érték eléri a 0-t, kikapcsolja a meghajtó motorját. Ez az időtartam megközelítőleg 2 másodperc.

41(h)

Az itt tárolt bájt az utolsó lemezművelet státuszát adja meg. Ha a bájt értéke 0, akkor az utolsó művelet sikeres volt, ellenkező esetben a bájt a hibakódot tartalmazza. A bájt értéke a BIOS 13(h) megszakításával lekérdezhető.

42(h) – 48(h)

Az itt tárolt 7 bájt a hajlékonylemezes meghajtó NEC (Nippon Electric Company) típusjelű vezérlő áramkörének státuszát tartalmazza.

49(h)

Erre a tárcímre a BIOS az aktuális video üzemmód kódját írja be. Az itt lévő kód megegyezik annak a video üzemmódnak a kódjával, amelyet a BIOS 10(h) megszakításának 0. számú funkciójával beállíthatunk. A bájt értéke monokróm képernyőkártya esetén 7.

4A(h) – 4B(h)

Ezek a tárcímek az egy képernyősorba írható oszlopok számát tartalmazzák.

4C(h) – 4D(h)

Ezen a tárcímen kezdődő szóba a BIOS az aktuális video üzemmódban egy képernyőoldal ábrázolásához szükséges bájtok számát írja be. Ez a 80 * 25 felbontású, monokróm, (szöveges) üzemmódban 4000 bájt.

4E(h) – 4F(h)

A BIOS az itt tárolt szóba az aktuális képernyőoldalnak a video RAM kezdőcímével viszonyított címét írja be. Színes-grafikus képernyőkártya esetén a video RAM kezdőcíme B800:0000(h), ennek megfelelően az első képernyőoldal relatív kezdőcíme 0000(h). 80 * 25 felbontású (szöveges) üzemmódban egy képernyőoldal hossza 4 kbájt, ezért a második képernyőoldal kezdőcíme B800:1000. Ha ez lenne az aktuális képernyőoldal, akkor a fenti tárcímre az 1000(h) érték kerül.

50(h) – 5F(h)

A BIOS maximum 8 képernyőoldalt tud kezelni (0 – 7. sorszámokkal). Mindegyik képernyőoldalnak saját kurzora van, amelyek aktuális pozícióját ezen a 16 bájtos területen tárolja. Egy-egy kurzorpozíciót két bájton ábrázol. Az első bájt az illető kurzor oszlopának számát (ez a felbontástól függően 0 – 39 vagy 0 – 79 lehet), a második bájt a kurzor sorának számát (0 – 24) tartalmazza. A 16 bájtos területen belül az első két bájt a 0. képernyőoldal kurzorának, a második két bájt az 1. képernyőoldal kurzorának felel meg, és így tovább.

60(h) – 61(h)

Ezen a két tárcímen a villogó kurzor alakját meghatározó két adat, nevezetesen a kurzornak a karaktermátrixon belüli kezdő- és végsora áll. A karaktermátrix monokróm videokártyánál 0 – 13 sorból, színes-grafikus videokártyánál 0 – 7 sorból áll, ezért a kur-

zor kezdő- és végsorának ezen értékeken belül kell lennie. A 60(h) tárcím a kezdősor sorszámát, a 61(h) tárcím a végsor sorszámát tartalmazza. A BIOS ezeket az értékeket alap-értelmezésben a 11 – 12. sorra (monokróm kártya) ill. a 6 – 7. sorra (színes kártya) állítja be. Csak a tárcímek átírásával a kurzor alakja még nem változik meg; ezeket az értékeket a BIOS megfelelő rutinján keresztül még át kell adni a képernyővezérlő adapternek.

62(h)

A tárcím az aktuális képernyőoldal számát tartalmazza.

63(h) – 64(h)

Ezen a két címen a számítógépben lévő videokártya alapportjának címe áll. Ha a gépben egyidejűleg több videokártya van, akkor itt az éppen aktív kártya portcíme található.

65(h)

Az aktuális video üzemmódot a mindenkori videokártya vezérlő, ill. üzemmódválasztó regiszterének tartalma határozza meg. Ezen a tárcímen ennek a regiszternek a tartalma áll.

66(h)

A színes-grafikus kártya a 320 * 200 képpont felbontású grafikus üzemmódban 4 szín megjelenítésére képes. A 4 színből 3 szín két paletta egyikéből választható ki. Ezen a tárcímen az aktuális paletta száma áll (0 vagy 1).

67(h) – 6B(h)

A BIOS belső használatára fenntartott terület.

6C(h) – 6F(h)

Ezt a négybájtos adatterületet a BIOS (és a DOS is) egy 32 bites számlálónak tekinti, amelynek értékét a 8-as számú órajeladó hardver megszakítás (timer interrupt) minden egyes aktiválásakor, azaz másodpercenként 18.2-szer 1-gyel megnövel. Ezáltal a számláló, mintegy a rendszer szoftver órájaként lehetővé teszi az idő mérését. A számláló értéke a BIOS 1A(h) megszakításával olvasható és beállítható. Amikor a számláló értéke eléri a 24 órának megfelelő értéket, tartalma törlődik, és az idő mérése 0-ról indul.

70(h)

Az itt lévő bájt értéke azt jelzi, hogy az időszámláló értéke elérte a 24 órának megfelelő értéket, vagyis a gép bekapcsolása vagy az időpont utolsó beállítása óta elmúlt már éjfél. A bájt éjfél előtti értéke 0, azt követően mindig 1. Az időpontnak a BIOS 1A(h) megszakításával történő beállításakor a bájt értéke 0 lesz.

71(h)

Az itt lévő bájt 7. bitjének 1 értéke a Ctrl – C vagy a Ctrl – Break billentyűkombináció lenyomását jelzi.

72(h) – 73(h)

A rendszer az indításakor egy reset parancsot küld a billentyűzet vezérlőnek. A reset (újraállítás) idejére ezekre a tárcímekre az 1234(h) érték íródik be.

74(h) – 77(h)

Az itt lévő 4 bájt a merevlemezes meghajtók vezérlőjének szóló információkat tartalmazza.

78(h) – 7B(h)

A BIOS a rendszer indításakor ezekre a tárcímekre beírja a gépbe behelyezhető 3 párhuzamos nyomtató adapterre megengedett időtúllépési (Time-out) értékeket (a negyedik bájt nem használt, fenntartott tárhely). Alapértelmezés szerint mind a négy bájt értéke 14(h).

7C(h) – 7F(h)

A BIOS a rendszer indításakor ezekre a tárcímekre beírja a gépbe behelyezhető 2 soros interfészre megengedett időtúllépési (Time-out) értékeket (a harmadik és a negyedik bájt nem használt, fenntartott tárhely). Alapértelmezés szerint mind a négy bájt értéke 01(h).

80(h) – 81(h)

Az itt tárolt szó a billentyűzet pufferének kezdőcímét tartalmazza a 0040(h) szegmenscímhez viszonyított ofszetcímként. Mivel a billentyűzet puffere normál esetben a 0040:001E(h) címen kezdődik, a tárolt szó 001E(h).

82(h) – 83(h)

Az itt tárolt szó a billentyűzet pufferének végcímét tartalmazza a 0040(h) szegmenscímhez viszonyított ofszetcímként. Mivel a billentyűzet puffere normál esetben a 0040:003E(h) címen végződik, a tárolt szó 003E(h).

A most következő tárcímeket csak az AT-k használják.

8B(h) – 95(h)

Ezeken a tárcímeken lévő bájtok a hajlékony- vagy a merevlemezes meghajtókkal kapcsolatos műveletekre vonatkozó, nem dokumentált változókat tartalmazzák.

96(h)

A billentyűzet járulékos státusz bájtja.

97(h)

Ezen a tárcímen is a billentyűzetre vonatkozó bájt található, amely a csak az AT billentyűzetén meglévő LED diódák (fénydiódák) állapotát jelzi.

98(h) – A0(h)

A valós idejű órában (RTC) tárolt várakozási változókat tartalmazó terület.

A1(h) – A7(h)

Fenntartott terület.

A8(h) – AB(h)

Módosított video paramétertábla mutató (EGA).

AC(h) – FF(h)

Fenntartott terület.

Külön meg kell említeni a 0050:0000(h) tárcímen lévő bájtot, amely bár már nem a BIOS, hanem a DOS által használt területen található, de amelyet mind a BIOS, mind a DOS egyaránt használ (a PC-ben, XT-ben és az AT-ben is). Ezt a bájtot az 5. számú hardcopy (Print Screen) megszakítás figyeli abból a célból, hogy egy hardcopy végrehajtása közben ne kelljen egy esetlegesen beérkező újabb hardcopy megszakításnak eleget tennie. Ezért a hardcopy rutin először megvizsgálja a 0050:0000(h) címen lévő státusz bájt értékét. Ha a megszakítás meghívásakor a bájt értéke 0, akkor ez azt jelenti, hogy éppen nem folyik nyomtatás, tehát a megszakításkérésnek eleget tehet. Egyúttal a bájt értékét 1-re állítja jelezve, hogy a nyomtató éppen foglalt, és egy most érkező, hardcopyt kérő megszakítás ne zavarja meg a folyamatban lévő műveletet. A rutin a nyomtatás befejeződésekor a bájt értékét ismét 0-ra állítja jelezve, hogy kész a további hardcopy-megszakítások fogadására. Ha a rutin a nyomtatás során a nyomtatóval való kapcsolattartásban valamilyen hibát észlel, akkor a bájtba az FF(h) értéket írja jelezve, hogy újabb nyomtatás nem hajtható végre.

2. A HARDVER KÖZVETLEN PROGRAMOZÁSA

Az eddigiekben részletesen megismerkedtünk az IBM PC/XT/AT számítógépek DOS és BIOS néven ismert rendszerprogramjaival, ezeken belül pedig azokkal a rendszerrutinokkal, amelyek használata jelentős mértékben leegyszerűsíti e gépek programozását, és növeli a programok hatékonyságát. Úgy gondoljuk, hogy ezzel a bevezetőben leírt, és a rendszerprogramozásról általánosságban megfogalmazott célkitűzéseknek eleget tettünk.

Ezt a fejezetet azok figyelmébe ajánljuk, akik adott esetben nem elégednek meg a rendszerrutinok szolgáltatásaival, hanem azt is látni szeretnék, ami ezeken túl van: közvetlenül a hardvert szeretnék programozni. Ezzel kapcsolatban ismételten felhívjuk a figyelmet arra, hogy egy programnak olyan mértékben csökken a „hordozhatósága", azaz a konkrét hardvertől való függetlensége, amilyen mértékben eltávolodik a DOS-tól: ha a program DOS rutinok mellett BIOS rutinokat is meghív, akkor a kifogástalan futására már csak ún. BIOS-kompatibilis gépeken számíthatunk (a tapasztalatok szerint a forgalomban lévő utánzatok általában BIOS-kompatibilisek). A hardver közvetlen programozása pedig természetesen még tovább növeli a program hardverfüggőségét. Ennek ellenére számos esetben szükség lehet arra, hogy – minden jótanácsot és intelmet félretéve – vegyük magunknak a bátorságot, és a DOS és BIOS megkerülésével közvetlenül a hardverhez forduljunk, mert például a rendszerrutinok nem teszik lehetővé a hangszóró megszólaltatását, vagy a megfelelően gyors képváltást. A lelkiismeretünket ilyenkor nyugtassuk meg azzal, hogy a közismert profi programok többsége is él ezzel a lehetőséggel. További érvként hozhatjuk fel, hogy a PC-k számos hardver eleme ma már szabványosnak tekinthető, és ha ezek jellemzőit a programjainkban figyelembe vesszük, akkor túlzottan nagy kockázatot nem kell vállalnunk. Mindenesetre továbbra is követendő az az alapszabály, hogy a programjainkban – ahol csak lehet – a dokumentált, és szabványosnak deklarált rendszerrutinokat használjuk.

2.1. Közvetlen kapcsolat a perifériákkal (portok)

Mint ismeretes, a számítógép mint hardver alapvetően a CPU-nak nevezett központi feldolgozó egységből (mikroprocesszorból), a központi egység munkáját segítő számos elektronikus elemből (perifériából) és a gépen részben kívül elhelyezkedő, különböző külső egységekből áll. Azt is tudjuk, hogy a számítógép egyes fizikai alkotó elemei között végbemenő valamennyi művelet – néhány kivételtől eltekintve – a mikroprocesszoron keresztül bonyolódik le. Ez azt is jelenti, hogy a számítógépes rendszerben lévő bármelyik hardver elemnek vagy külső egységnek – például a megszakításvezérlő áramkörnek, a képernyő adapternek vagy a billentyűzetnek stb. – minden olyan esetben, amikor valamilyen műveletet kell elvégeznie, vagy vezérlőjelet kell adnia/fogadnia, szükségszerűen kapcsolatba kell kerülnie a központi egységgel. Azokat az illeszkedési felületeket vagy helyeket (puffereket, regisztereket), amelyeken keresztül a perifériák, ill. külső egységek és a központi egység közötti adatforgalom lebonyolódik, az illető perifériához, ill. külső egységhez tartozó portoknak (kapuknak) nevezik. A port tulajdonképpen olyan eszköznek is tekinthető, amelynek saját azonosító címe van (portcím). A processzor a portokkal a számítógép cím- és adatbuszán keresztül kommunikál. Amikor a processzor egy adatot vagy valamilyen információt akar valamelyik portra kiküldeni (vagy beolvasni), akkor először a 8288-as jelű buszvezérlő IC-n keresztül kiad egy vezérlőjelet, amellyel a rendszerbuszra csatlakozó valamennyi hardver elemmel azt tudatja, hogy az a cím, amelyet most a címbuszra helyez, nem tárcímként értelmezendő, hanem a megszólítandó periféria portcíme (a portcím „látszatra" semmiben sem különbözik egy „normál" tárcímtől, de a valóságban természetesen semmi közük sincs egymáshoz). Ezt követően a processzor a címbusznak a 0 – 15. biteket reprezentáló alsó 16 címvezetékére helyezi a kiválasztott port címét, az adatbuszon pedig kiküldi, ill. bekéri a szükséges adatot. E műveletek eredményeként a megszólított periféria, ill. egység – pontosabban fogalmazva az ezeket vezérlő áramkör, adapter – kiválasztott portján most rendelkezésre áll a kiküldött vagy bekért jel, amelyre az illető periféria vagy a processzor megfelelőképpen reagálhat.

Mivel a processzor a portok címzéséhez a címbusz alsó 16 vezetékét használja, következik, hogy az így megadható 16 bites portcímekkel elvileg összesen 65536 port címezhető meg a 0000(h) – FFFF(h) címtartományban. A valóságban azonban ennél jóval kevesebb a használt portok száma. Egyébként ha nemlétező portcímre küldünk ki valamilyen adatot, akkor semmi sem történik. A PC gépcsalád tagjaiban és az utánzatokban lévő hardver elemek portcímeivel kapcsolatban megjegyezzük, hogy a dokumentációkban általában nemcsak a valóban használt néhány portcímet, hanem egy – általában 16, egymásra következő címet magában foglaló – címtartományt adnak meg, amely a ténylegesen használt címeket is tartalmazza. (A vizsgálódásainkhoz szükséges portcímeket a megfelelő helyen természetesen megadjuk. Az IBM PC/XT/AT gépek által használt portcímeket a III. kötet tartalmazza.)

Assembly nyelven a portokkal való kommunikációt a mikroprocesszor IN és OUT utasításai teszik lehetővé. Az IN utasítás egy adott portcímről beolvas, az OUT utasítás pedig egy adott portcímre kiír egy 8 vagy 16 bites adatot.

Az utasítások használatánál a következő megszorítást kell figyelembe venni: Az utasításokkal közvetlen módon csak a maximum 8 bites portcímek olvashatók, ill. írhatók, vagyis azok, amelyek a 00(h) – FF(h) címtartományon belül vannak. A processzor az adatokat az AL, ill. az AX regiszterbe olvassa be (IN) vagy onnan írja ki (OUT) attól függően, hogy az adat 8 vagy 16 bites. Ezzel szemben, ha a portcím 8 bitnél szélesebb (a 100(h) – FFFF(h) tartományban van), akkor csak a regiszteren keresztüli, közvetett címzési mód használható, és a közvetítő regiszter a DX regiszter. Az adatokat most is az AL vagy az AX regiszter adja át, ill. fogadja az adat szélességétől függően.

Az előbbiek alapján a portokat olvasó/író assembler utasítások helyes szintaxisa:
a) ha a portcím 00(h) – FF(h) között van, akkor közvetlen címzéssel:

```
IN      AL, portcím;      8 bites adat beolvasása
OUT     portcím,AL;       8 bites adat kiírása

IN      AX,portcím;       16 bites adat beolvasása
OUT     portcím,AX;       16 bites adat kiírása
```

b) ha a portcím 100(h) – FFFF(h) között van, akkor közvetett címzéssel, a DX regiszteren keresztül:

```
MOV     DX, portcím;      közvetett beolvasás,
IN      AL, DX;           8 bites adat

MOV     DX, portcím;      közvetett kiírás,
OUT     AL, DX;           8 bites adat

MOV     DX, portcím;      közvetett beolvasás,
IN      AX, DX;           16 bites adat

MOV     DX, portcím;      közvetett kiírás,
OUT     AX, DX;           16 bites adat
```

A következő fejezetekben megismerkedünk néhány olyan hardver elem közvetlen programozásával, amelyeket a forgalomban lévő legtöbb professzionális felhasználói program is a rendszerprogramok megkerülésével, azaz közvetlenül programoz. E hardver elemek közé tartoznak mindenek előtt a PC-kben használatos különböző képernyőkártyák (más szóval videokártyák), valamint a hangszóró.

2.2. A képernyőkártyák közvetlen programozása

Ebben a fejezetben bemutatjuk a PC-kben legelterjedtebben használt képernyőkártyák felépítését és programozását. Vizsgálódásaink az IBM monokróm képernyőkártyára, a Hercules grafikus képernyőkártyára és az IBM színes-grafikus képernyőkártyára terjednek ki. (E képernyőkártyákkal a könyv BIOS-ról szóló részének 1.3. alfejezetében már foglalkoztunk, így az ezekkel kapcsolatos jónéhány fogalom már is-

mert lehet.) Megjegyezzük, hogy az említett képernyőkártyákon túl még számos különböző, főleg a színes-grafikus megjelenítést illetően ezeknél lényegesen jobb videokártyák is forgalomban vannak – mint pl. az EGA (Enhanced Graphics Adapter) vagy a VGA (Video Graphics Array) –, de ezeknek, valamint a használatukhoz szükséges színes monitoroknak az ára ma még gátolja a széles körű elterjedésüket.

Ahhoz, hogy a képernyőkártyák közvetlen programozásának technikáját megértsük, elegendő a video megjelenítés néhány alapvető fogalmának ismerete. A képernyőn (monitoron) a kép képpontok sokaságaként jelenik meg. Az egyes képpontokat a katódsugárcső (CRT, Cathode Ray Tube) által kibocsátott elektronsugár, ill. színes megjelenítés esetén a három alapszínnek megfelelő három elektronsugár teszi láthatóvá vagy láthatatlanná a kibocsátott elektronsugár intenzitásától függően. A kibocsátott elektronsugár a képernyő egyes pontjait sorra végigpásztázza, majd amikor eléri a sor végét, rátér a következő sor elejére. Amikor ilyen módon eléri a képernyő utolsó sorának utolsó képpontját, visszatér az első sor első képpontjára. Mindez olyan sebességgel történik, hogy a sor-, ill. a képváltások az emberi szem számára észrevehetetlenek maradnak, így az a képpontokból felépülő képet egyetlen, összefüggő és egész képként érzékeli. (Megjegyezzük, hogy ezzel a magyarázattal csak az eljárás megértését kívánjuk elősegíteni; maga a folyamat a valóságban ennél sokkal összetettebb, és nem is pont így megy végbe.) Már csak annyit kell tudnunk, hogy a katódsugárcső a vezérlő jeleit – és ezek azok, amelyek végső soron meghatározzák, hogy egy adott képpont látszódjon vagy ne – a monitor áramkörein keresztül a számítógépben lévő képernyőkártyáról, ezen belül is a képernyőadapterről (videovezérlőről) kapja. Ennyi háttérismeret birtokában most már valóban lássuk a képernyőkártyák programozását.

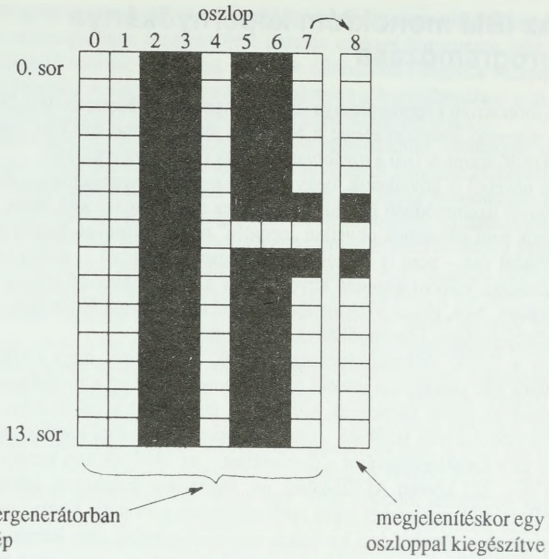
Mint említettük, a képernyőn megjelenő képpontok vezérlését a képernyőkártya végzi. Ahhoz, hogy a képernyőkártya a képet minden pillanatban a kívánt módon elő tudja állítani, természetesen tudnia kell, hogy melyik képpontot és milyen színben kell láttatnia. Ehhez szüksége van egy olyan RAM tárterületre, amelyből a megfelelő adatokat folyamatosan kiolvashatja. (Az adatokat ide természetesen a programjaink írják majd be.) Ezt a RAM területet video RAM-nak nevezik, és az ezt fizikailag megvalósító integrált áramkörök magán a képernyőkártyán helyezkednek el. (A video-RAM a könyv BIOS-ról szóló részéből már ismerős fogalom.) A továbbiakban még bőven lesz szó róla. Magának a képernyőnek a felépítéséhez azonban még nem elegendő, ha a video RAM-ban a videokártya rendelkezésére állnak a megjelenítendő adatok. Ezen túlmenően a megfelelő portokon keresztül még számos adatot kell közölnünk a megjelenítést végző, és ugyancsak a képernyőkártyán elhelyezett videovezérlő áramkörrel is. Mivel ezek az adatok az egyes képernyőkártyáknál többé-kevésbé eltérnek egymástól, az ezzel kapcsolatos tudnivalókat a célbavett képernyőkártyáknál külön-külön ismertetjük.

2.2.1. Az IBM monokróm képernyőkártya programozása

Az IBM monokróm képernyőkártya (ismert angol nevén Monochrome Display Adapter, MDA) alapvető áramköri eleme a Motorola által gyártott 6845-ös jelű, intelligens képernyővezérlő áramkör (ezt a továbbiakban csak 6845-ként említjük).

A kártya nevéből is következik, hogy csak monokróm megjelenítésre és csak szöveges (karakteres) üzemmódban használható. Ehhez rögtön hozzá kell tenni, hogy ezek a megszorítások nem jelentenek okvetlen „negatív" tulajdonságokat, hiszen számos olyan ügyviteli feladat van – nem is beszélve a szövegszerkesztésről –, amely nem igényli a színek használatát, viszont alapvető követelmény a megjelenítendő szöveg, számok stb. jó olvashatósága. Nos, ebben a tekintetben a monokróm kártya felbontása jócskán meghaladja a színes-grafikus képernyőkártya képességeit.

A kártya a kiváló felbontóképességét annak köszönheti, hogy a karaktereket egy 9 * 14 pontból álló pontrácson jeleníti meg. A pontrácsnak ez a formátuma azért is különleges, mert az egyes karakterek bitmintáját tartalmazó karaktergenerátor – amely ugyancsak a videokártyán található – csak 8 képpont szélességű karaktereket képes előállítani. Így ez a karaktergenerátor – önmagában – az IBM bővített karakterkészletében szereplő, 176 – 223 közötti ASCII-kódú, ún. félgrafikus karakterek között található 9 képpont szélességű karaktereket teljes szélességükben nem tudja kirajzolni, következésképpen a vizszintes vonalak és a kereteket meghúzó vonalak nem lennének folyamatosak. Ezért a videokártyán egy további áramkört helyeztek el, amely ezt a problémát úgy oldja meg, hogy a fent említett karakterek 8. képpontját a pontrács mindegyik sorában átmásolja a 9. képpontba. A karakterkép ilyen módon történő „kibővítését" a 19. ábra mutatja.



*19. ábra: A monokróm képernyőkártya karakterábrázolása a 9 * 14 képpontból álló pontrácson*

Eszerint az ábrázolásmód szerint a karaktergenerátor egy karakter egyetlen sorát 8 biten (azaz 1 bájton), a teljes karaktert pedig 14 soron tárolja. Egy karakter tárolásához tehát 14 bájtra, a teljes karakterkészlet tárolására pedig közel 4000 bájtra van szükség (a karaktergenerátor részére a valóságban ennél jóval nagyobb, 8 kbájtnyi ROM áll a rendelkezésére).

A képernyőkártya video-RAM területe a B0000(h) fizikai tárcímen kezdődik, és 4 kbájt területet foglal el. Mivel a képernyőkártya csak szöveges üzemmódban használható, amelyben a felbontás 25 sor * 80 oszlop, egyidejűleg összesen 2000 karakter jeleníthető meg, amelyek tárolásához csak 4000 bájtra van szükség (ne feledkezzünk meg, hogy egy karaktert 2 bájt tárol: ebből az első a karakter ASCII-kódját, a második pedig az attribútumát tartalmazza). A rendelkezésre álló video-RAM legfelső 96 bájtja így kihasználatlan marad (szabadon felhasználható).

A képernyő első karakterpozíciója a bal felső sarokban van. Az itt megjelenítendő karakternek megfelelő két bájtot a video-RAM a legalsó két címén, a B000:0000 – 0001(h) tárcímeken tárolja. A következő karakterpozíciót ugyanazen sor következő oszlopa jelenti, az ehhez tartozó két bájtot pedig a video-RAM következő két tárcíme tartalmazza. Ennek megfelelően a képernyő első sorában megjelenő karaktereket és az attribútum bájtokat – egymást váltogatva – a video-RAM első 160 címe tárolja. Az ezt követő címek – anélkül, hogy a sorok között bármiféle elválasztó jel lenne – folyamatosan veszik fel a második képernyősor első, második stb. karaktereinek a bájtjait mindaddig, míg a

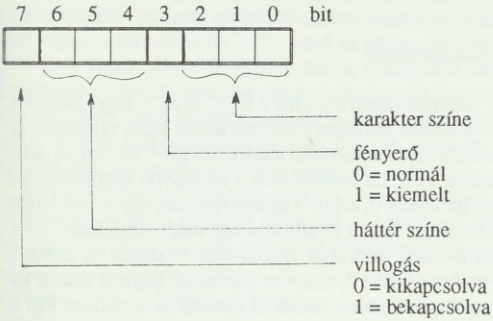
képernyő utolsó, 25. sorának utolsó, 80. oszlopát el nem érik. Az utolsó karakterpozíción lévő karaktert és az attribútum bájtját a B000:0F9E – 0F9F tárcímek tartalmazzák.

Mivel a képernyősorok és a tárcímek ilyen szépen, folyamatosan követik egymást, könnyen megadható egy képlet, amellyel a képernyő bármely adott sorához és oszlopához tartozó karakterpozíció tárcíme kiszámítható:

tárcím = sor * 160 + oszlop * 2

A tárcím itt természetesen a video-RAM kezdőcímétől számított ofszetcímként értendő, míg a sorok és az oszlopok számozása 0-tól kezdődik. Az így kiszámított tárcímhez 1-et hozzáadva megkapjuk az illető karakterpozícióhoz tartozó attribútum bájt tárcímét.

Az attribútum bájt felépítésével, az egyes bitek jelentésével és ezek ésszerű kombinációival a könyv BIOS-ról szóló részének 1.3.1. pontjában már részletesen foglalkoztunk (lásd a 3. és 4. ábrát is), ezért itt most – ismétlésként – csak a bájt felépítését mutatjuk be:



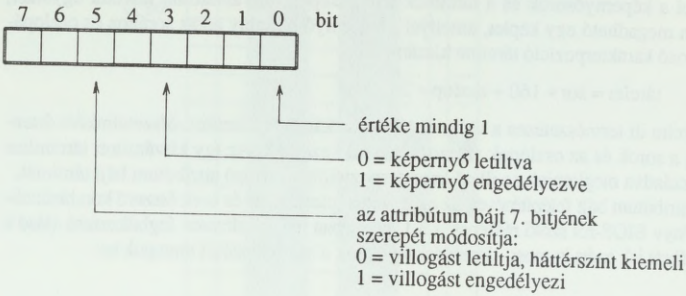
Ezzel kapcsolatban ide kívánkozik még az említett fejezetben is megtett megjegyzésünk:

Az attribútum bájt mind a karakter, mind a háttér színére 3–3 bitet foglal le, jóllehet tudjuk, hogy monokróm képernyőkártya esetén szín nem is választható meg (maximum normál és az inverz, ill. adott esetben az aláhúzott megjelenítés közül választhatunk). Ennek az a magyarázata, hogy karakteres üzemmódban a színes megjelenítés is ugyanezt az attribútum bájtot használja a színek kijelzésére, ahol a most feleslegesnek tűnő bitek majd önálló szerephez jutnak.

A video-RAM-ot illetően a képernyőkártya közvetlen programozásához ennyi ismeret már elegendő. Lássuk most a portokat és a monitor közvetlen vezérlését végző 6845-ös különböző regisztereit.

Az IBM monokróm képernyőkártya portcímei a 3B0(h) – 3BF(h) I/O címtartományba esnek. Ezek közül a rendszer csak hét portcímet használ ki, ebből is csak négyet a képernyő vezérléséhez (a többi három porton keresztül az ugyanezen a kártyán található nyomtatóvezérlő áramkör programozható). Vegyük sorra a képernyővezérlő portokat.

A 3B8(h) portcímen található a képernyőkártya vezérlőregisztere. Felépítését a 20. ábra mutatja.

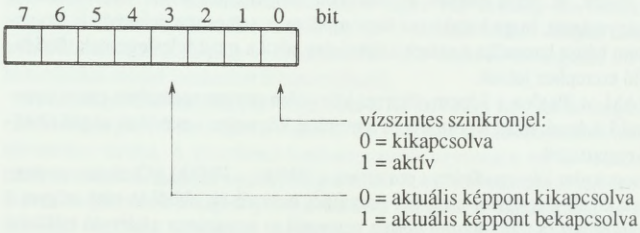


20. ábra: A vezérlőregiszter felépítése (portcíme 3B8(h))

Amint az ábrából látható, a vezérlőregiszternek csak a 0., 3. és az 5. bitjei játszanak szerepet. A 0. bittel – elvileg – a kétféle szöveges felbontás között lehet választani, de mivel monokróm kártya esetén csak egyféle felbontás létezik (25 sor * 80 oszlop), a bit értékének mindig 1-nek kell lennie. A 3. bit beállításával a képernyőn megjelenítendő kép be-, ill. kikapcsolható. Ha a bit értéke 0, akkor a képernyőn csak egy fekete felület látható, és a villogó kurzor is eltűnik. A bitet 1-re állítva a kép ismét megjelenik. Az 5. bit az attribútum bájt 7. bitjének szerepét módosítja, amennyiben ez utóbbi bit értéke 1: ha ekkor a vezérlőregiszter 5. bitjének értéke 0, akkor az attribútum bájthoz tartozó karakter villogását letiltja, és kiemeli a háttérszínt. Ebben az esetben az attribútum bájt 7. bitje tulajdonképpen a háttér színére vonatkozó intenzitásbitnek tekinthető.

A vezérlőregiszter csak írható regiszter. A rendszer indításakor az alapértelmezés szerinti értéke 29(h), vagyis a szerephez jutó mindhárom bit értéke 1. (Ez a 29(h) érték kerül a 65(h) ofszetcímű BIOS változóba.)

A 3BA(h) portcímen a képernyőkártya státuszregisztere érhető el. Felépítését a 21. ábra mutatja.



21. ábra: A státuszregiszter felépítése (portcíme 3BA(h))

A rendszer a státuszregiszternek csak a 0. és a 3. bitjét használja. A 0. bit azt adja meg, hogy képet megrajzoló elektronsugarat vízszintes irányban eltérítő vezérlőjel éppen egy vízszintes (sor)szinkron jelet tartalmaz-e. Ezt a jelet a 6845-ös minden alkalommal kiküldi a képernyőhöz, amikor az elektronsugár egy teljes szélességű pontsort kirajzolt. Ez a szinkronjel jelzi, hogy az elektronsugár elérte a képernyő végét, és vissza kell ugrania a következő sor elejére. Ebben az időpillanatban a bit értéke 1. A 3. bit értékét az elektronsugár által éppen megjelenített (vagy nem megjelenített) képpont határozza meg: ha a képpontot be kell kapcsolni, akkor a bit értéke 1, ellenkező esetben 0.

A státuszregiszter a vezérlőregisztertől eltérően csak olvasható regiszter.

E két regiszteren kívül a képernyőkártya még két, speciális rendeltetésű regisztert tartalmaz, amelyek ugyancsak portcímeken keresztül érhetők el. Az egyik neve indexregiszter (portcíme 3B4(h)), a másiké adatregiszter (portcíme 3B5(h)). E két regiszter együttesen arra használható, hogy rajtuk keresztül hozzá lehessen férni a 6845-ös összesen 18 darab belső regiszteréhez. Ha e regiszterek valamelyikébe értéket akarunk írni (a regiszterek többsége nem olvasható), akkor először az indexregiszterbe be kell írni a kívánt regiszter sorszámát (0 – 17), majd az adatregiszterbe az átadandó adatot. Ekkor a 6845-ös a kapott adatot átveszi, és beírja a megadott sorszámú belső regiszterébe. A programozói gyakorlatban csak ritkán fordulhat elő, hogy ezekbe a regiszterekbe az eredeti értékektől eltérő értékeket kellene írnunk, továbbá nem is ajánlott, mert a regiszterek a monitor működésével kapcsolatos olyan, alapvető paramétereket tartalmaznak, amelyek megváltoztatása a monitor károsodását okozhatja.

A következőkben felsoroljuk ezeknek a regisztereknek a jelentését, és megadjuk a szokásos értéküket (a részletes jelentésük, a különböző mértékegységek, számítási módok stb. ismertetése túlmutat a tárgyalni kívánt témakörön):

Sorszám	A regiszter jelentése	Érték
0	Egy képernyősor (scan-sor) megrajzolásához szükséges idő karakteregységben	61(h)
1	Egy képernyősorban megjelenített karakterek száma	50(h)
2	Vízszintes szinkronjel kezdetének ideje	52(h)
3	Vízszintes szinkronjel szélességének időtartama	0F(h)
4	Függőleges képernyősorok számát jellemző érték (5. regiszterrel együtt)	19(h)
5	Függőleges képernyősorok számát jellemző érték (4. regiszterrel együtt)	06(h)
6	Látható szövegsorok száma	19(h)
7	Függőleges szinkronjel kezdetének ideje	19(h)
8	Képernyősorok letapogatásának módja	02(h)
9	Szövegsor magassága képernyősoronként számítva mínusz 1	0D(h)
10	A villogó kurzor kezdő képernyősora	0B(h)
11	A villogó kurzor befejező képernyősora	0C(h)
12	A képernyőoldal kezdőcímének felső bájtja	00(h)
13	A képernyőoldal kezdőcímének alsó bájtja	00(h)
14	A villogó kurzor pozíciójának felső bájtja	00(h)
15	A villogó kurzor pozíciójának alsó bájtja	00(h)
16	Fenntartott	00(h)
17	Fenntartott	00(h)

Ebből a 18 regiszterből a programozók érdeklődésére elsősorban a kurzor alakját meghatározó 10 – 11. regiszterek, valamint a kurzor pozícióját meghatározó 14 – 15. regiszterek tarthatnak számot.

A kurzor alakját a kurzort mint „karaktert" létrehozó képernyősoroknak (a pásztázó elektronsugárnak) a szövegsoron belüli kezdő és befejező sora határozza meg. Mivel a monokróm képernyőkártya a karaktereket egy 9 * 14 pontból álló pontrácson (mátrixon) jeleníti meg, a kurzor kezdő és végsora a 0 – 13. közötti tartományban lehet. Ha a kezdősor nagyobb mint a végsor, akkor a kurzor „hézagos" lesz. Ha a kezdő és a végsor bármelyikére 13-nál nagyobb értéket adunk meg, akkor a kurzor láthatatlanná válik. Alapértelmezés szerint a kurzor két sor szélességű, és a 11 – 12. sorokat foglalja el (a legfelső sor a 0. sor). A teljes magasságú kurzor kezdősora a 0., végsora a 13. sor. A kurzor alakjának beállításához a kezdősor értékét a 6845-ös 10. regiszterébe, a végsor értékét a 11. regiszterébe kell beírni. A regiszterek nem olvashatók.

A 14 – 15. regiszterek a villogó kurzor pozícióját jelölik ki, de nem a már ismert, sor- és oszlopkoordinátákkal megadott módon. A 6845-ös ugyanis a kurzor pozícióját aszerint tartja nyilván, hogy ez a video-RAM kezdőcímétől számított hányadik tárcímen van. Így a 0. képernyőoldal (ami monokróm kártya esetén egyúttal az egyetlen lehetséges képernyőoldal) bal felső sarka a 0. kurzorpozíciót jelenti, és ekkor a regiszterek tartalma 0. Arról azonban nem szabad megfeledkezni, hogy ezt a kurzorpozíciót csak a 6845-ös tekinti a 0. pozíciónak, a CPU számára ez a video-RAM kezdőcímét, esetünkben a B000:0000(h) tárcímet jelenti. Ha a kurzort a 6845-ös e regiszterein keresztül akarjuk pozícionálni, akkor azt is figyelembe kell venni, hogy a 6845-ös számára egy kurzorpozíció két szomszédos bájtot jelent, annak megfelelően, hogy egy képernyőpozícióhoz egy karakter- és egy attribútum bájt tartozik. Következésképpen a képernyő jobb alsó sarkában lévő kurzorpozíciót a 6845-ös az 1999-edik pozícióként értelmezi. Mivel a 6845-ös regiszterei 8 bitesek (vagy rövidebbek), ezt a számot egy regiszterben nem lehet elhelyezni. Ezért a kurzorpozíció számának alsó 8 bitjét a 15. regiszterbe, a maradék biteket pedig a 14. regiszterbe kell beírni (ill. innen olvashatók ki). Ha a regiszterekbe 1999-nél nagyobb értéket írunk, akkor a kurzor eltűnik a képernyőről.

A következő fejezetben bemutatunk egy assembly nyelvű programpéldát, amely a monokróm képernyőkártya közvetlen programozásának lehetőségeit szemlélteti.

2.2.2. Programpélda

```
;
;   I B M M O N O . A S M
;
;   Funkciója : az IBM monokróm kártya néhány tulajdonságának, a
;               közvetlen programozási lehetőségek szemléltetése.
;
;-- KÓD ---------------------------------------------------------------

kod     segment para 'CODE'     ; A kódszegmens definiálása.

        org 100h                ; A program a PSP után álló első
                                ; címen kezdődik.

        assume cs:kod, ds:kod, es:kod, ss:kod

                                ; Csak egy, közös szegmens van.

;-- Demonstrációs program ----------------------------------------------

start:  mov  ah,9               ; Karakterláncot megjelenítő
        mov  dx,offset karlanc  ; funkció és megszakítás
        int  21h                ; hívása.
                                ;
        xor  ah,ah              ; Várakozás billentyű megnyomására.
        int  16h                ; 16(h) megszakítás.
        cmp  al,"s"             ;
        je   nemmono            ;
        cmp  al,"S"             ; Az <s> vagy <S> billentyű leütése
        jne  mono               ; esetén a program futása végetér.

nemmono: jmp  vege

mono:   mov  di,3000            ; A kurzor elrejtése.
        call kurpoz             ;
        call clst               ; Képernyő törlése.
        mov  di,0               ; Kezdés a bal felső sarokban.
        mov  si,offset speckar  ;
        mov  cx,2000            ; A képernyő 2000 pozíciójának
        mov  al,07h             ; feltöltése fekete háttéren fehér
cim_1:  call kiiro              ; karakterekkel ciklusban.
        loop cim_1              ;

        ;-- Képernyőablakok kialakítása ------------------------------

                                ; Főablak :
        mov  bx,1201h           ;     bal felső sarok koord -> BL,BH.
        mov  dx,2B03h           ;     méretek (sor,oszlop). -> DL,DH.
        mov  ah,07h             ; Fekete háttéren fehér karakterek.
        call torol              ; Az ablak törlése.
        mov  bx,1502h           ; A főcím képernyőkoordinátái -> BX.
        call cimszam            ; Ofszetcím számítás.
        mov  si,offset txt0     ; A karakterlánc ofszetcíme -> SI.
        mov  ah,70h             ; Fehér háttéren fekete karakterek.
        call kiiro              ; Karakterlánc kiírás.
```

```
                                ;
                                ; 6. oszlop (a jobb oldali az első)
                                ; Címablak :
mov  bx,0405h                   ;      bal felső sarok koord -> BL,BH.
mov  dx,0703h                   ;      méretek (sor,oszlop). -> DL,DH.
mov  ah,07h                     ; Fekete háttéren fehér karakterek.
call torol                      ; Az ablak törlése.
mov  bx,0506h                   ; A főcím képernyőkoordinátái -> BX.
call cimszam                    ; Ofszetcím számítás.
mov  si,offset txt1             ; A karakterlánc ofszetcíme -> SI.
mov  ah,70h                     ; Fehér háttéren fekete karakterek.
call kiiro                      ; Karakterlánc kiírás.
                                ; Munkaablak :
mov  bx,0409h                   ;      bal felső sarok koord -> BL,BH.
mov  dx,070Eh                   ;      méretek (sor,oszlop). -> DL,DH.
mov  ah,07h                     ; Fekete háttéren fehér karakterek.
call torol                      ; Az ablak törlése.
                                ;
                                ; 5. oszlop
                                ; Címablak :
mov  bx,1105h                   ;      bal felső sarok koord -> BL,BH.
mov  dx,0703h                   ;      méretek (sor,oszlop). -> DL,DH.
mov  ah,07h                     ; Fekete háttéren fehér karakterek.
call torol                      ; Az ablak törlése.
mov  bx,1206h                   ; A főcím képernyőkoordinátái -> BX.
call cimszam                    ; Ofszetcím számítás.
mov  si,offset txt2             ; A karakterlánc ofszetcíme -> SI.
mov  ah,70h                     ; Fehér háttéren fekete karakterek.
call kiiro                      ; Karakterlánc kiírás.
                                ; Munkaablak :
mov  bx,1109h                   ;      bal felső sarok koord -> BL,BH.
mov  dx,070Eh                   ;      méretek (sor,oszlop). -> DL,DH.
mov  ah,07h                     ; Fekete háttéren fehér karakterek.
call torol                      ; Az ablak törlése.
                                ;
                                ; 4. oszlop
                                ; Címablak :
mov  bx,1E05h                   ;      bal felső sarok koord -> BL,BH.
mov  dx,0703h                   ;      méretek (sor,oszlop). -> DL,DH.
mov  ah,07h                     ; Fekete háttéren fehér karakterek.
call torol                      ; Az ablak törlése.
mov  bx,1F06h                   ; A főcím képernyőkoordinátái -> BX.
call cimszam                    ; Ofszetcím számítás.
mov  si,offset txt3             ; A karakterlánc ofszetcíme -> SI.
mov  ah,70h                     ; Fehér háttéren fekete karakterek.
call kiiro                      ; Karakterlánc kiírás.
                                ; Munkaablak :
mov  bx,1E09h                   ;      bal felső sarok koord -> BL,BH.
mov  dx,070Eh                   ;      méretek (sor,oszlop). -> DL,DH.
mov  ah,07h                     ; Fekete háttéren fehér karakterek.
call torol                      ; Az ablak törlése.
                                ;
                                ; 3. oszlop
                                ; Címablak :
mov  bx,2B05h                   ;      bal felső sarok koord -> BL,BH.
mov  dx,0703h                   ;      méretek (sor,oszlop). -> DL,DH.
mov  ah,07h                     ; Fekete háttéren fehér karakterek.
call torol                      ; Az ablak törlése.
mov  bx,2C06h                   ; A főcím képernyőkoordinátái -> BX.
call cimszam                    ; Ofszetcím számítás.
mov  si,offset txt4             ; A karakterlánc ofszetcíme -> SI.
mov  ah,70h                     ; Fehér háttéren fekete karakterek.
call kiiro                      ; Karakterlánc kiírás.
```

```
                                ; Munkaablak :
        mov  bx,2B09h           ;     bal felső sarok koord -> BL,BH.
        mov  dx,070Eh           ;     méretek (sor,oszlop). -> DL,DH.
        mov  ah,07h             ; Fekete háttéren fehér karakterek.
        call torol              ; Az ablak törlése.
                                ;
                                ; 2.oszlop
                                ; Címablak :
        mov  bx,3805h           ;     bal felső sarok koord -> BL,BH.
        mov  dx,0703h           ;     méretek (sor,oszlop). -> DL,DH.
        mov  ah,07h             ; Fekete háttéren fehér karakterek.
        call torol              ; Az ablak törlése.
        mov  bx,3906h           ; A főcím képernyőkoordinátái -> BX.
        call cimszam            ; Ofszetcím számítás.
        mov  si,offset txt5     ; A karakterlánc ofszetcíme -> SI.
        mov  ah,70h             ; Fehér háttéren fekete karakterek.
        call kiiro              ; Karakterlánc kiírás.
                                ; Munkaablak :
        mov  bx,3809h           ;     bal felső sarok koord -> BL,BH.
        mov  dx,070Eh           ;     méretek (sor,oszlop). -> DL,DH.
        mov  ah,07h             ; Fekete háttéren fehér karakterek.
        call torol              ; Az ablak törlése.
                                ;
                                ; 1.oszlop
                                ; Címablak :
        mov  bx,4505h           ;     bal felső sarok koord -> BL,BH.
        mov  dx,0703h           ;     méretek (sor,oszlop). -> DL,DH.
        mov  ah,07h             ; Fekete háttéren fehér karakterek.
        call torol              ; Az ablak törlése.
        mov  bx,4606h           ; A főcím képernyőkoordinátái -> BX.
        call cimszam            ; Ofszetcím számítás.
        mov  si,offset txt6     ; A karakterlánc ofszetcíme -> SI.
        mov  ah,70h             ; Fehér háttéren fekete karakterek.
        call kiiro              ; Karakterlánc kiírás.
                                ; Munkaablak :
        mov  bx,4509h           ;     bal felső sarok koord -> BL,BH.
        mov  dx,070Eh           ;     méretek (sor,oszlop). -> DL,DH.
        mov  ah,07h             ; Fekete háttéren fehér karakterek.
        call torol              ; Az ablak törlése.

        ;— Demonstrációs szakasz ————————————————————————————————

                                ; 1. ablak :
cim_2:  mov  bx,450Ah           ; Bal  f. sarok koordinátái -> BL,BH.
        mov  dx,4B15h           ; Jobb a. sarok koordinátái -> DL,DH.
        mov  ah,07h             ; Fekete háttéren fehér karakterek.
        mov  cl,1               ; 1 sornyi görgetés szükséges.
        call gorgfel            ; Görgetés FEL.
                                ;
        mov  speckar,"•"        ; A szabaddá vált sorba írandó
        mov  ax,szaml           ; karakter meghatározása (felváltva
        and  ax,1b              ; "•" vagy "●").
        jnz  cim_3              ;
        mov  speckar,"●"        ;
cim_3:  mov  bx,4815h           ; A kiírás képernyőpozíciója -> BX.
        call cimszam            ; Ofszetcím számítás.
        mov  si,offset speckar  ; A karakterlánc ofszetcíme -> SI.
        mov  ah,0Fh             ; Fekete háttéren világos karakterek.
        call kiiro              ; Karakterlánc kiírás.
                                ;
                                ; 2. ablak :
```

```
        mov   ax,szaml          ;
        and   ax,1b             ; Csak minden 2. alkalommal kell az
        jz    cim_4             ; ablakon müveletet végezni.
        jmp   cikend            ;
cim_4:  mov   bx,380Ah          ; Bal  f. sarok koordinátái -> BL,BH.
        mov   dx,3E15h          ; Jobb a. sarok koordinátái -> DL,DH.
        mov   ah,07h            ; Fekete háttéren fehér karakterek.
        mov   cl,1              ; 1 sornyi görgetés szükséges.
        call  gorgle            ; Görgetés LE.
                                ;
        mov   speckar,"•"       ; A szabaddá vált sorba irandó
        mov   ax,szaml          ; karakter meghatározása (felváltva
        and   ax,10b            ; "•" vagy "●").
        jnz   cim_5             ;
        mov   speckar,"●"       ;
cim_5:  mov   bx,3B0Ah          ; A kiírás képernyőpozíciója -> BX.
        call  cimszam           ; Ofszetcím számítás.
        mov   si,offset speckar ; A karakterlánc ofszetcíme -> SI.
        mov   ah,0Fh            ; Fekete háttéren világos karakterek.
        call  kiiro             ; Karakterlánc kiírás.
                                ;
                                ; 3. ablak :
        mov   ax,szaml          ;
        and   ax,11b            ; Csak minden 4. alkalommal kell az
        jz    cim_6             ; ablakon müveletet végezni.
        jmp   cikend            ;
cim_6:  mov   bx,2B0Ah          ; Bal  f. sarok koordinátái -> BL,BH.
        mov   dx,3115h          ; Jobb a. sarok koordinátái -> DL,DH.
        mov   ah,07h            ; Fekete háttéren fehér karakterek.
        mov   cl,1              ; 1 sornyi görgetés szükséges.
        call  gorgfel           ; Görgetés FEL.
                                ;
        mov   speckar,"•"       ; A szabaddá vált sorba irandó
        mov   ax,szaml          ; karakter meghatározása (felváltva
        and   ax,100b           ; "•" vagy "●").
        jnz   cim_7             ;
        mov   speckar,"●"       ;
cim_7:  mov   bx,2E15h          ; A kiírás képernyőpozíciója -> BX.
        call  cimszam           ; Ofszetcím számítás.
        mov   si,offset speckar ; A karakterlánc ofszetcíme -> SI.
        mov   ah,0Fh            ; Fekete háttéren világos karakterek.
        call  kiiro             ; Karakterlánc kiírás.
                                ;
                                ; 4. ablak :
        mov   ax,szaml          ;
        and   ax,111b           ; Csak minden 8. alkalommal kell az
        jz    cim_8             ; ablakon müveletet végezni.
        jmp   cikend            ;
cim_8:  mov   bx,1E0Ah          ; Bal  f. sarok koordinátái -> BL,BH.
        mov   dx,2415h          ; Jobb a. sarok koordinátái -> DL,DH.
        mov   ah,07h            ; Fekete háttéren fehér karakterek.
        mov   cl,1              ; 1 sornyi görgetés szükséges.
        call  gorgle            ; Görgetés LE.
                                ;
        mov   speckar,"•"       ; A szabaddá vált sorba irandó
        mov   ax,szaml          ; karakter meghatározása (felváltva
        and   ax,1000b          ; "•" vagy "●").
        jnz   cim_9             ;
        mov   speckar,"●"       ;
cim_9:  mov   bx,210Ah          ; A kiírás képernyőpozíciója -> BX.
        call  cimszam           ; Ofszetcím számítás.
        mov   si,offset speckar ; A karakterlánc ofszetcíme -> SI.
```

```
                mov  ah,0Fh             ; Fekete háttéren világos karakterek.
                call kiiro              ; Karakterlánc kiírás.
                                        ;
                                        ; 5. ablak :
                mov  ax,szaml           ;
                and  ax,1111b           ; Csak minden 16. alkalommal kell az
                jz   cim_10             ; ablakon müveletet végezni.
                jmp  cikend             ;
cim_10:         mov  bx,110Ah           ; Bal  f. sarok koordinátái -> BL,BH.
                mov  dx,1715h           ; Jobb a. sarok koordinátái -> DL,DH.
                mov  ah,07h             ; Fekete háttéren fehér karakterek.
                mov  cl,1               ; 1 sornyi görgetés szükséges.
                call gorgfel            ; Görgetés FEL.
                                        ;
                mov  speckar,"•"        ; A szabaddá vált sorba irandó
                mov  ax,szaml           ; karakter meghatározása (felváltva
                and  ax,10000b          ; "•" vagy "●").
                jnz  cim_11             ;
                mov  speckar,"●"        ;
cim_11:         mov  bx,1415h           ; A kiírás képernyőpozíciója -> BX.
                call cimszam            ; Ofszetcím számítás.
                mov  si,offset speckar  ; A karakterlánc ofszetcíme -> SI.
                mov  ah,0Fh             ; Fekete háttéren világos karakterek.
                call kiiro              ; Karakterlánc kiírás.
                                        ;
                                        ; 6. ablak :
                mov  ax,szaml           ;
                and  ax,11111b          ; Csak minden 32. alkalommal kell az
                jnz  cikend             ; ablakon müveletet végezni.
                                        ;
                mov  bx,040Ah           ; Bal  f. sarok koordinátái -> BL,BH.
                mov  dx,0A15h           ; Jobb a. sarok koordinátái -> DL,DH.
                mov  ah,07h             ; Fekete háttéren fehér karakterek.
                mov  cl,1               ; 1 sornyi görgetés szükséges.
                call gorgle             ; Görgetés LE.
                                        ;
                mov  speckar,"•"        ; A szabaddá vált sorba irandó
                mov  ax,szaml           ; karakter meghatározása (felváltva
                and  ax,100000b         ; "•" vagy "●").
                jnz  cim_12             ;
                mov  speckar,"●"        ;
cim_12:         mov  bx,070Ah           ; A kiírás képernyőpozíciója -> BX.
                call cimszam            ; Ofszetcím számítás.
                mov  si,offset speckar  ; A karakterlánc ofszetcíme -> SI.
                mov  ah,0Fh             ; Fekete háttéren világos karakterek.
                call kiiro              ; Karakterlánc kiírás.

                ;— Demonstrációs szakasz vége ————————————————————————

cikend:         dec  szaml              ; A ciklust 1000x kell végrehajtani.
                je   vege               ;
                mov  cx,60000           ; Várakozási ciklus a sebesség
szunet:         loop szunet             ; beállításához.
                jmp  cim_2              ; Ugrás a demo ciklus elejére.
                                        ;
vege:           xor  ah,ah              ; Várakozás billentyű megnyomására.
                int  16h                ; 16(h) megszakítás.
                                        ;
                mov  di,0               ; A kurzor visszaállítása.
                call kurpoz             ;
                call clst               ; A képernyő törlése.
                mov  ax,4C00h           ; 21(h) megszakítás, a program a 0
                int  21h                ; hibakóddal ér véget.
```

```
;— ADATOK ————————————————————————————————————————————————————————

szaml     dw 1000
speckar   db "*",0

txt0      db "Görgetés a leggyorsabb ablakban 1000x", 0
txt1      db "1/32",0
txt2      db "1/16",0
txt3      db " 1/8",0
txt4      db " 1/4",0
txt5      db " 1/2",0
txt6      db "  1 ",0

CR        equ 13                    ; Kocsivissza ASCII kódja.
LF        equ 10                    ; Soremelés ASCII kódja.
KEND      equ "$"                   ; Karakterlánc vége jelzés.

Karlanc   db CR,LF,"A példaprogram csak IBM  mono képernyőkártyával"
          db CR,LF,"rendelkező gépeken futtatható. Amennyiben az on"
          db CR,LF,"gépében ettől eltérő képernyőkártya található,"
          db CR,LF,"állítsa  le a programot  az <s> billentyű meg-"
          db CR,LF,"nyomásával,  ellenkező esetben  egy tetszőleges"
          db CR,LF,"másik billentyű  megnyomásával a  program foly-"
          db CR,LF,"tatható ...",CR,LF,KEND

;————————————————————————————————————————————————————————————————
;— A demonstrációhoz felhasznált funkciók ——————————————————————————
;————————————————————————————————————————————————————————————————

;— KURPOZ : a kurzor pozíciójának beállítása ————————————————————
;— BE : DI = a kurzor ofszetcíme
;— KI : -
;— Regiszterek : AX, BX és DX megváltoznak

Kurpoz    proc near

          mov  bx,di                ; Kurzor ofszetcím -> BX.
          mov  al,14                ; A 14-es regiszterbe kerül
          mov  ah,bh                ; az ofszetcím felső bájtja.
          call regir                ; Regiszter beállítás.
          mov  al,15                ; A 15-ös regiszterbe kerul
          mov  ah,bl                ; az ofszetcím alsó bájtja.
          call regir                ; Regiszter beállítás.
                                    ;
          ret                       ; Vissza a hívóhoz.

Kurpoz    endp

;— REGIR : a vezérlő egyik regiszterébe megadott érték írása ———————
;— BE : AL = a regiszter sorszáma
;       AH = a beírandó érték
;— KI : -
;— Regiszterek : DX és AL megváltoznak

regir     proc near

          mov  dx,3B4h              ; A regiszter sorszámának kiírása
          out  dx,al                ; az indexregiszterbe (portcím 3B4h).
          mov  dx,3B5h              ;
          mov  al,ah                ; Az adat kiírása az adatregiszterbe
          out  dx,al                ; (portcím 3B5h).
                                    ;
          ret                       ; Vissza a hívóhoz.
```

```
regir   endp

;— REGOLV : a vezérlő egyik regiszterének kiolvasása ————————————
;— BE : AL = a regiszter sorszáma
;— KI : AL = a beolvasott érték
;— Regiszterek : DX és AL megváltoznak

regolv  proc near

        mov  dx,3B4h            ; A regiszter sorszámának kiírása
        out  dx,al              ; az indexregiszterbe (portcím 3B4h).
        mov  dx,3B5h            ; Az adat beolvasása az adat-
        in   al,dx              ; regiszterből (portcím 3B5h).
                                ;
        ret                     ; Vissza a hívóhoz.

regolv  endp

;— GORGFEL : egy ablak megadott sorral felfelé görgetése ————————
;— BE : BL = bal felső sarok sora
;       BH = bal felső sarok oszlopa
;       DL = jobb alsó sarok sora
;       DH = jobb alsó sarok oszlopa
;       CL = görgetendő sorok száma
;— KI : -
;— Regiszterek : FLAGS megváltozik

gorgfel proc near

        cld                     ; Karakterlánc műveletek beállítása.
        push ax                 ; A regiszterek elmentése a veremre.
        push bx                 ;
        push di                 ;
        push si                 ;
        push bx                 ;
        push cx                 ;
        push dx                 ;
        sub  dl,bl              ; Ablak sorai számának kiszámítása.
        inc  dl                 ;
        sub  dl,cl              ; A "megmaradó" sorok száma -> DL.
        sub  dh,bh              ; Ablak oszlopai számának kiszámítása.
        inc  dh                 ;
        call cimszam            ; Ofszetcím számítás (bal f. sarok).
        mov  si,di              ; Az ofszetcím elmentése -> SI.
        add  bl,cl              ; A "megmaradó" első sor -> BL.
        call cimszam            ; Ofszetcím számítás (új első sor).
        xchg si,di              ; SI és DI cseréje.
        push ds                 ; DS -> verem.
        push es                 ; ES -> verem.
        mov  ax,0B000h          ; Video RAM szegmenscím -> DS.
        mov  ds,ax              ;                       -> ES.
        mov  es,ax              ;
gorgi:  mov  ax,di              ; DI -> AX.
        mov  bx,si              ; SI -> BX.
        mov  cl,dh              ; Az oszlopok száma -> számláló (CL).
        rep  movsw              ; Egy sor mozgatása ciklusban (CX).
        mov  di,ax              ; AX -> DI.
        mov  si,bx              ; BX -> SI.
        add  di,160             ; A cél- és forráscímek megnövelése
        add  si,160             ; egy sorral.
        dec  dl                 ; HA még nem lett minden sor
        jne  gorgi              ; elmozgatva, ugrás a következőre.
```

```
        pop  es                 ;
        pop  ds                 ; Az eredeti regisztertartalmak
        pop  dx                 ; visszatoltése a veremről.
        pop  cx                 ;
        pop  bx                 ;
                                ;
        mov  bl,dl              ; Az eredeti sorkoordináták és a
        sub  bl,cl              ; görgetett sorok alapján az "üressé"
        inc  bl                 ; vált terület bal felső sora -> BL.
        sub  dl,bl              ; A törlendő sorok száma -> DL.
        inc  dl                 ;
        sub  dh,bh              ; A törlendő oszlopok száma -> DH.
        inc  dh                 ;
        mov  ah,07h             ; Fekete háttéren fehér karakterek.
        call torol              ; Az üres terület törlése.
                                ;
        pop  si                 ; Regiszterek visszatoltése a
        pop  di                 ; veremről.
        pop  bx                 ;
        pop  ax                 ;
        ret                     ; Vissza a hívóhoz.

gorgfel  endp

;-- GORGLE : egy ablak megadott sorral lefelé görgetése ----------
;-- BE : BL = bal felső sarok sora
;        BH = bal felső sarok oszlopa
;        DL = jobb alsó sarok sora
;        DH = jobb alsó sarok oszlopa
;        CL = görgetendő sorok száma
;-- KI : -
;-- Regiszterek : FLAGS megváltozik

gorgle   proc near

        cld                     ; Karakterlánc műveletek beállítása.
        push ax                 ; A regiszterek elmentése a veremre.
        push bx                 ;
        push di                 ;
        push si                 ;
        push bx                 ;
        push cx                 ;
        push dx                 ;
        sub  dh,bh              ; Ablak oszlopai számának kiszámítása.
        inc  dh                 ;
        mov  al,bl              ; Bal felső sor -> AL.
        mov  bl,dl              ; Jobb alsó sor -> BL.
        call cimszam            ; Ofszetcím számítás (bal f. sarok).
        mov  si,di              ; Az ofszetcím elmentése -> SI.
        sub  bl,cl              ; Görgetendő sorok számának levonása.
        call cimszam            ; Ofszetcím számítás.
        xchg si,di              ; SI és DI cseréje.
        sub  dl,al              ; Ablak sorai számának kiszámítása.
        inc  dl                 ;
        sub  dl,cl              ; A "megmaradó" sorok száma -> DL.
        push ds                 ; DS -> verem.
        push es                 ; ES -> verem.
        mov  ax,0B000h          ; Video RAM szegmenscím -> DS.
        mov  ds,ax              ;                       -> ES.
        mov  es,ax              ;
gorg2:  mov  ax,di              ; DI -> AX.
        mov  bx,si              ; SI -> BX.
```

```
        mov   cl,dh             ; Az oszlopok száma -> számláló (CL).
        rep   movsw             ; Egy sor mozgatása ciklusban (CX).
        mov   di,ax             ; AX -> DI.
        mov   si,bx             ; BX -> SI.
        sub   di,160            ; A cél- és forráscímek megnövelése
        sub   si,160            ; egy sorral.
        dec   dl                ; HA még nem lett minden sor
        jne   gorg2             ; elmozgatva, ugrás a következőre.
        pop   es                ;
        pop   ds                ; Az eredeti regisztertartalmak
        pop   dx                ; visszatöltése a veremről.
        pop   cx                ;
        pop   bx                ;
        mov   dl,bl             ; Az eredeti sorkoordináták és a
        add   dl,cl             ; görgetett sorok alapján az "üressé"
        dec   dl                ; vált terület jobb alsó sora -> DL.
        sub   dl,bl             ; A törlendő sorok száma -> DL.
        inc   dl                ;
        sub   dh,bh             ; A törlendő oszlopok száma -> DH.
        inc   dh                ;
        mov   ah,07h            ; Fekete háttéren fehér karakterek.
        call  torol             ; Az üres terület törlése.
                                ;
        pop   si                ; Regiszterek visszatöltése a
        pop   di                ; veremről.
        pop   bx                ;
        pop   ax                ;
        ret                     ; Vissza a hívóhoz.

gorgle  endp

;-- CLST : a teljes képernyő letörlése karakteres üzemmódban ----------
;-- BE  : -
;-- KI  : -
;-- Regiszterek : FLAGS megváltozik

clst    proc near

                                ; A törlendő ablak definiálása :
                                ;
        mov   ah,07h            ;     Színe fekete - fehér lesz.
        mov   bl,0              ;     Bal felső sor : 0.
        mov   bh,0              ;     Bal felső oszlop : 0.
        mov   dl,25             ;     összesen 25 sor.
        mov   dh,80             ;     összesen 80 oszlop.
                                ;
        call  torol             ; Az ablak törlése.
        ret                     ; Visszatérés a hívóhoz.

clst    endp

;-- TOROL : megadott képernyőablak feltöltése szókozokkel (törlés) ----
;-- BE : AH = szín / atribútum
;       BL = a bal felső sarok sora
;       BH = a bal felső sarok oszlopa
;       DL = a sorok száma összesen
;       DH = az oszlopok száma összesen
;-- KI : -
;-- Regiszterek : FLAGS megváltozik

torol   proc near
```

```
          cld                       ;
          push cx                   ; Regiszterek elmentése a veremre.
          push si                   ;
          push di                   ;
          push es                   ;
          call cimszam              ; Video RAM ofszetcím számítás.
          mov  cx,0B000h            ; Video RAM szegmenscím -> ES.
          mov  es,cx                ;
          mov  ch,0                 ;
          mov  al," "               ; Szóköz karakter -> AL.
toroli:   mov  si,di                ; DI -> SI mentés.
          mov  cl,dh                ; Oszlopok száma -> CL (számlálóba).
          rep  stosw                ; Szó kiírása (itt szóköz) ciklusban.
          mov  di,si                ; SI -> DI visszatoltés.
          add  di,160               ;
          dec  dl                   ; Ha nem ez volt az utolsó sor, akkor
          jne  toroli               ; ofszetcím számítás és újra kiírás.
                                    ;
          pop  es                   ; A regiszterek visszaállítása.
          pop  di                   ;
          pop  si                   ;
          pop  cx                   ;
          ret                       ; Vissza a hívóhoz.

torol     endp

;— KIIRO : egy karakterlánc kiírása a képernyőre ————————————————
;— BE : AH = szín / attribútum
;       DI = a képernyő kívánt helyének a címe
;       SI = a kiírandó karakterlánc címe
;— KI : DI a képernyőn a karakterlánc utáni első pozícióra mutat
;— Regiszterek : AL, DI és FLAGS megváltoznak

kiiro     proc near

          cld                       ;
          push si                   ; Regiszterek elmentése a veremre.
          push es                   ;
          push dx                   ;
          mov  dx,0B000h            ; Video RAM szegmenscím -> ES.
          mov  es,dx                ;
          lodsb                     ; Ha a karakterlánc első karaktere
          or   al,al                ; a '0' karakter, akkor vége.
          je   kiiro2               ;
kiiro1:   stosw                     ; Karakter és színattribútum
          lodsb                     ; ciklusban a képernyőre, mindaddig
          or   al,al                ; amíg a lezáró '0' sorra nem kerul.
          jne  kiiro1               ;
                                    ;
kiiro2:   pop  dx                   ; Regiszterek visszatoltése a
          pop  es                   ; veremről.
          pop  si                   ;
          ret                       ; Vissza a hívóhoz.

kiiro     endp

;— CIMSZAM : sor és oszlop értékből cím képzése ——————————————————
;— BE : BL = sor
;       BH = oszlop
;— KI : DI = a kiszámított cím
;— Regiszterek : DI és FLAGS megváltoznak
```

```
cimszam  proc near

         push ax                    ; Regiszterek elmentése a veremre.
         push bx                    ;
                                    ;
         shl  bx,1                  ; Sorok és oszlopok 2-vel szorzandók
                                    ; (minden második pozíción van kar.).
         mov  al,bh                 ; Oszlopkoordináta -> AL.
         mov  bh,0                  ; A sorkoordináta -> DI, az adat-
         mov  di,[sorcim+bx]        ; táblázat alapján.
         mov  ah,0                  ; Az oszlopkoordináta hozzáadódik
         add  di,ax                 ; DI értékéhez.
                                    ;
         pop  bx                    ; Regiszterek visszatöltése
         pop  ax                    ; a veremről.
         ret                        ; Vissza a hívóhoz.

cimszam  endp

;— ADATOK ——————————————————————————————————————————————————————

sorcim   dw    0, 160, 320, 480, 640      ; A 25 képernyősor oldalon
         dw  800, 960,1120,1280,1440      ; belüli ofszetcíme a
         dw 1600,1760,1920,2080,2240      ; video RAM-ban.
         dw 2400,2560,2720,2880,3040
         dw 3200,3360,3520,3680,3840

;— VÉGE ———————————————————————————————————————————————————————

kod      ends                       ; A (kód)szegmens vége.
         end  start                 ; Az assembler forrásprogram vége.
                                    ; A program végrehajtása a start
                                    ; címkén kezdődik.
```

2.3. A Hercules grafikus képernyőkártya programozása

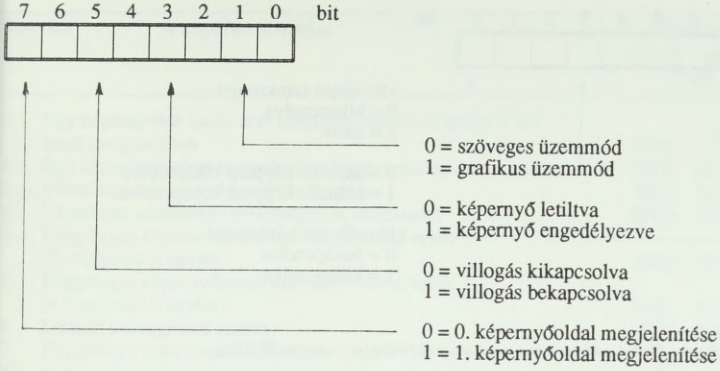
A Hercules grafikus képernyőkártya – amely egyébként ugyancsak monokróm megjelenítést tesz lehetővé – az IBM monokróm kártyával szemben azzal az óriási előnnyel rendelkezik, hogy azonosan jó szövegfelbontás mellett igen jó felbontású grafikák rajzolására is képes. Grafikus üzemmódban a felbontása 720 * 348 képpont. További előnye, hogy a 64 kbájtos video-RAM-jában két, egyenként 32–32 kbájtos képernyőoldalt képes tárolni és kezelni (szöveges üzemmódban ezekből csak 4–4 kbájtot használ fel). A video-RAM a B000:0000 – FFFF közötti tárterületen található, amelyből az első képernyőoldal a B000:0000 – 7FFF közötti, a második képernyőoldal a B800:0000 – FFFF közötti, 32–32 kilobájtos területet foglalja el.

Szöveges üzemmódban a Hercules grafikus képernyőkártya video-RAM-jának felépítése, a kurzor ábrázolása és a portok címei teljes mértékben megegyeznek az IBM monokróm kártyánál írtakkal. A grafikus ábrázolási képessége csak annyi többletet jelent, hogy az IBM kártya státusz- és vezérlőregiszterében néhány, ott kihasználatlan bit most szerephez jut.

A normál monokróm kártyához képest a Hercules kártya még egy konfigurációs regisztert is tartalmaz, amelynek portcíme 3BF(h). Ez csak írható regiszter. A regiszterben csak a 0. és az 1. bit játszik szerepet: a 0. bit azt adja meg, hogy van-e lehetőség a grafikus üzemmód bekapcsolására (az 1 értéke engedélyezi, a 0 letiltja), az 1. bit pedig azt mutatja meg, hogy használható-e a második képernyőoldal (ha az értéke 1, akkor használható).

A színes-grafikus képernyőkártyával foglalkozó fejezetben látni fogjuk, hogy az e kártya által használt video-RAM teljes egészében azon a tárterületen belül van, amit a Hercules kártya is használ grafikus üzemmódban. Ha tehát a számítógépben a Hercules kártyán kívül még egy színes-grafikus kártya is van, akkor a rendszerprogramoknak gondoskodniuk kell arról, hogy a két kártya emiatt ne kerüljön „konfliktusba" egymással. Ezért a számítógép bekapcsolásakor a konfigurációs regiszterben mind a 0., mind az 1. bit értéke 0-ra állítódik, úgyhogy induláskor sem a grafikus üzemmód, sem a második képernyőoldal használata nem engedélyezett. Ha egy program a Hercules kártya e képességeit ki akarja használni, akkor a konfigurációs regiszterben először át kell állítania ezt a két bitet.

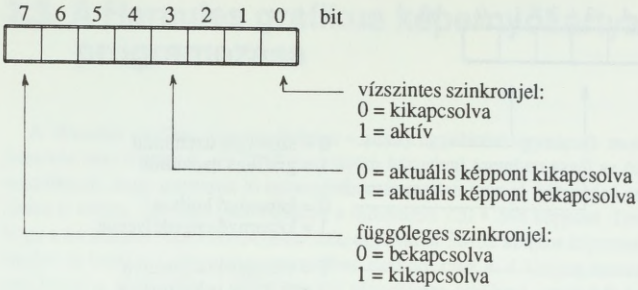
Az IBM monokróm kártyánál már megismert vezérlőregiszter felépítése itt némiképp változik (lásd a 22. ábrát).



22. ábra: A Hercules kártya vezérlőregiszterének felépítése (portcíme 3B8(h))

A vezérlőregiszter 1. bitjének értéke dönti el, hogy a kártya szöveges vagy grafikus üzemmódban dolgozzon. Az üzemmód beállításához azonban nem elegendő ennek az egy bitnek az átállítása, hanem a 6845-ös videoprocesszor belső regisztereibe is más értékeket kell írni. Erre az átprogramozási időre a képernyőt a 3. bit 0-ra állításával ki kell kapcsolni. A 7. bit értékével a megjeleníteni kívánt képernyőoldalt választhatjuk ki. Mivel a video-RAM írása/olvasása és a képi megjelenítése egymástól független műveletek, annak a képernyőoldalnak megfelelő RAM területet is írhatjuk, ill. olvashatjuk, amelyik éppen nem aktív. Így mód van arra, hogy amíg a képernyőn az egyik oldal látszik, addig a másik oldalon – láthatatlanul – egy új képet készítsünk el. Ez az eljárás igen gyors képváltást tesz lehetővé. A vezérlőregiszter csak írható regiszter.

A státuszregiszter felépítését a 23. ábra mutatja.



23. ábra: A Hercules kártya státuszregiszterének felépítése (portcíme 3BA(h))

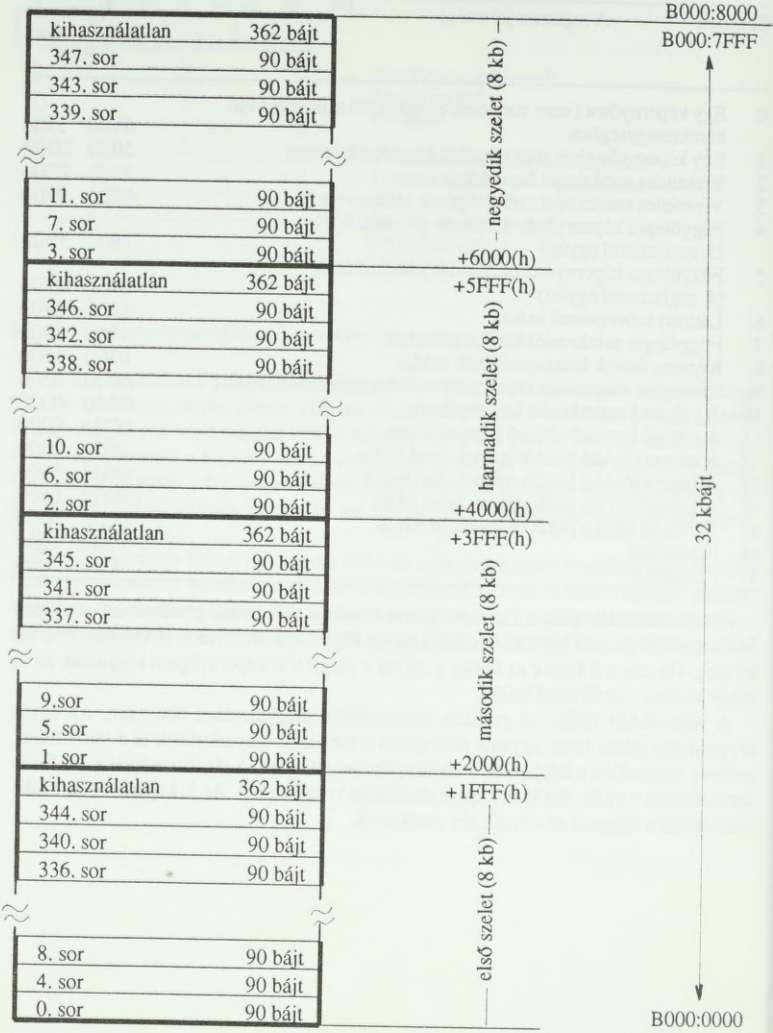
Az IBM monokróm kártya státuszregiszteréhez képest itt csak annyi a változás, hogy a 7. bit is szerepet kapott. Ennek a bitnek mindig akkor lesz 0 az értéke, amikor a 6845-ös egy függőleges szinkronjelet küld a képernyőre. Ezt a jelet a 6845-ös minden olyan alkalommal kiküldi a képernyőre, amikor az elektronsugár egy teljes képernyőt felépített. Ez a szinkronjel jelzi, hogy az elektronsugár elérte a képernyő legalsó sorának a végét, és vissza kell ugrania a legfelső sor elejére, hogy megkezdje a következő képernyőábra felépítését.

Mivel a Hercules kártya ugyanazt a 6845-ös videoprocesszort használja mint az IBM kártya, a processzor belső regiszterei és a regiszterek szerepe mindkét esetben azonos. Az index- és az adatregiszternek is azonos a portcíme. Az egyes regisztereknek a szöveges és a grafikus üzemmódban – célszerűen – a következő értékeket kell tartalmazniuk:

Sorszám	A regiszter jelentése	Üzemmód 80*25 érték	Üzemmód graf. érték
0	Egy képernyősor (scan-sor) megrajzolásához szükséges idő karakteregységben	61(h)	35(h)
1	Egy képernyősorban megjelenített karakterek száma	50(h)	2D(h)
2	Vízszintes szinkronjel kezdetének ideje	52(h)	2E(h)
3	Vízszintes szinkronjel szélességének időtartama	0F(h)	07(h)
4	Függőleges képernyősorok számát jellemző érték (5. regiszterrel együtt)	19(h)	5B(h)
5	Függőleges képernyősorok számát jellemző érték (4. regiszterrel együtt)	06(h)	02(h)
6	Látható szövegsorok száma	19(h)	57(h)
7	Függőleges szinkronjel kezdetének ideje	19(h)	57(h)
8	Képernyősorok letapogatásának módja	02(h)	02(h)
9	Szövegsor magassága képernyősoronként számítva mínusz 1	0D(h)	03(h)
10	A villogó kurzor kezdő képernyősora	0B(h)	00(h)
11	A villogó kurzor befejező képernyősora	0C(h)	00(h)
12	A képernyőoldal kezdőcímének felső bájtja	00(h)	00(h)
13	A képernyőoldal kezdőcímének alsó bájtja	00(h)	00(h)
14	A villogó kurzor pozíciójának felső bájtja	00(h)	00(h)
15	A villogó kurzor pozíciójának alsó bájtja	00(h)	00(h)
16	Fenntartott	00(h)	00(h)
17	Fenntartott	00(h)	00(h)

Amint már említettük, a Hercules grafikus kártya felbontása grafikus üzemmódban 348 sor * 720 pont. A képernyő minden egyes képpontjának a video-RAM-ban 1 bit felel meg. Ha ennek a bitnek az értéke 1, akkor a megfelelő képernyőpont kigyullad, ha 0, akkor kialszik (háttérszínű lesz).

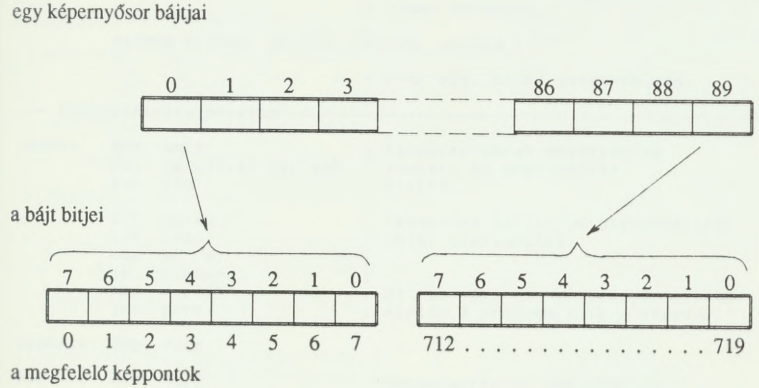
A video-RAM felépítése grafikus üzemmódban meglehetősen bonyolult. Az egyes képpontokat tároló bitek ugyanis nem olyan sorrendben helyezkednek el a tárcímeken, amilyen sorrendben a képpontok a képernyőn megjelennek. A tárolás módját a 24. ábrán szemléltetjük a video-RAM 0. képernyőoldalára vonatkozóan. Az 1. képernyőoldalra értelemszerűen ugyanez az elrendezés vonatkozik.



24. ábra: A képernyősorok tárolása a video-RAM-ban grafikus üzemmódban (0. képernyőoldal)

Az ábra némi magyarázatot igényel: a video-RAM 32 kbájtos képernyőoldala 4, egyenként 8–8 kbájtos szeletre van felosztva. A képernyő 0. sorát felépítő bitek az első, az 1. sorát felépítő bitek a második, a 2. sor bitjei a harmadik, a 3. soré pedig a negyedik szeletbe kerülnek. A következő sor ezután ismét az első, majd a második, harmadik stb. szeletbe kerül, és így tovább. Ha az így kialakuló elrendezést a video-RAM felől szemléljük, akkor azt látjuk, hogy az egyes 8 kbájtos szeletekben a képernyősorok sorszámai egymáshoz képest rendre 4-gyel el vannak tolva. Az első szelet a 0., 4., 8. stb. sorokat, a második az 1., 5., 9. stb. sorokat tárolja, és így tovább. Mint tudjuk, 1 sor 720 képpontból áll, azaz egy sor tárolásához 720 bitre, vagyis 90 bájtra van szükség. Ahhoz, hogy a 348 sor az egyes szeletek között egyenletesen oszoljon el, egy szeletnek 348/4 = 87 képernyősort kell tárolnia. Ez összesen 87 * 90 = 7830 bájtot vesz igénybe a szelet 8 kbájtjából, így minden szelet „tetején" 8192 – 7830 = 362 bájt kihasználatlan marad.

Most nézzük meg a grafikus képernyősorokat alkotó egyes képpontok (pixel) tárolásának módját. Láttuk, hogy egy-egy képernyősort 90 bájt tárol. Ezekben a 90 bájt hosszú sorokban a 0. bájt a sor első 8 képpontját, az 1. bájt a második 8 képpontját stb., végül a 89. bájt a sor utolsó 8 képpontját tartalmazza. A bájtok bitjeinek és a képpontok egymáshoz rendelését a 25. ábrán szemléltetjük.



25. ábra: Egy képernyősort alkotó bájtok bitjeinek és a megfelelő képpontok egymáshoz rendelése

Mivel az egyes képpontok a képernyőn – nem úgy, mint a video-RAM-ban – ilyen szépen sorban követik egymást, ezek folytonos sorszámokkal is azonosíthatók. A sorszámozás a 0. sor 0. képpontjától kezdődik, és a 347. sor 719. képpontjával fejeződik be (ez a képernyő jobb alsó sarkában lévő, 250559. képpont). A video-RAM tárolási módjából és a képpontok sorszámozásából levezethető egy képlet, amellyel kiszámítható annak a bájtnak a – képernyőoldal kezdőcímétől számított – relatív címe, amelyik egy meghatá-

rozott képpontra vonatkozó információt tartalmaz. A képpontot a képernyőn értelmezett X és Y koordinátáival kell megadni (X = 0 - 719, Y = 0 - 347). A bájt relatív címe:

cím = 8192 * (Y mod 4) + 90 * int(Y/4) + int(X/8)

A bájton belüli bit meghatározásához a következő képlet használható:

a bit száma = 7 - (X mod 8)

Ehhez a fejezethez is adunk egy programpéldát, amely bemutatja a Hercules grafikus kártya tulajdonságait.

2.3.1. Programpélda

```
;
;   ╔═══════════════════════════════════════════════════════════════╗
;   ║                                                               ;
;   ║                        H E R C U L E S . A S M                ;
;   ║                                                               ;
;   ║  Funkciója : az Hercules monokróm kártya néhány tulajdonságá- ;
;   ║              nak szemléltetése, beleértve a 720 x 348 képpon- ;
;   ║              tos grafikus és a szöveges üzemmódot.            ;
;   ╚═══════════════════════════════════════════════════════════════╝
;
;— MAKRÓ ---------------------------------------------------------------

setmode  macro ujertek              ; A képernyőkártya vezérlőregiszter
                                    ; beállítása.

         mov  dx,3B8h               ; A vezérlőregiszter portcíme.
         mov  al,ujertek            ; Új vezérlőbájt -> AL.
         out  dx,al                 ; Az új érték kiírása a regiszterbe.

         endm

;— KÓD -----------------------------------------------------------------

kod      segment para 'CODE'        ; A kódszegmens definiálása.

         org 100h                   ; A program a PSP után álló első
                                    ; címen kezdődik.

         assume cs:kod, ds:kod, es:kod, ss:kod

                                    ; Csak egy, közös szegmens van.

;— Demonstrációs program -----------------------------------------------

start:   mov  ah,9                  ; Karakterláncot megjelenítő
         mov  dx,offset karlanc     ; funkció és megszakítás
         int  21h                   ; hívása.
                                    ;
         xor  ah,ah                 ; Várakozás billentyű megnyomására.
         int  16h                   ; 16(h) megszakítás.
         cmp  al,"s"                ;
         je   nemherc               ;
         cmp  al,"S"                ; Az <s> vagy <S> billentyű leütése
         jne  herc                  ; esetén a program futása végetér.

nemherc: jmp  vege

herc:    mov  ah,9                  ; Karakterláncot megjelenítő
         mov  dx,offset inditas     ; funkció és megszakítás
         int  21h                   ; hívása.
         xor  ah,ah                 ; Várakozás billentyű megnyomására.
         int  16h                   ; 16(h) megszakítás.
         call clst                  ;
                                    ;
         mov  al,11b                ; Grafika lehetséges, két oldal van.
         call konf                  ; Konfigurálás.
         mov  bp,0                  ; 0. oldal -> BP.
         call grafika               ; Grafika bekapcsolása.
                                    ;
         mov  al,0                  ;
         call clsg                  ; A grafikus képernyő törlése.
                                    ;
         mov  bx,627                ; A rajz indítása az 627. oszlopon,
         mov  dx,299                ;                     299. soron.
```

```
                mov   ax, 3             ; Az első függőleges 3 pontból áll.
                mov   cx, 2             ; Az első vízszintes 2 pontból áll.
                                        ;
cim_1:          push  cx                ; Vonalhosszak elmentése a veremre.
                push  ax                ;
                                        ;
cim_2:          call  pontir            ; Jobbról balra vízszintes vonal
                dec   bx                ; húzása ciklikus képpontkiírással.
                loop  cim_2             ;
                mov   cx, 10000         ; Várakozási ciklus a sebesség
szun2:          loop  szun2             ; beállításához.
                pop   ax                ; Vonalhosszak behívása a veremről.
                pop   cx                ;
                add   cx, 2             ; A következő vízszintes hosszának
                                        ; beállítása.
                push  cx                ; Vonalhosszak elmentése a veremre.
                mov   cx, ax            ; A vonalhossz -> CX (számláló).
                push  ax                ;
                                        ;
cim_3:          call  pontir            ; Fentről lefele függőleges vonal
                inc   dx                ; húzása ciklikus képpontkiírással.
                loop  cim_3             ;
szun3:          loop  szun3             ; beállításához.
                mov   cx, 10000         ; Várakozási ciklus a sebesség
                pop   ax                ; Vonalhosszak behívása a veremről.
                pop   cx                ;
                add   ax, diffsor       ; A következő függőleges hosszának
                add   diffsor, 1        ; beállítása.
                push  cx                ; Vonalhosszak elmentése a veremre.
                push  ax                ;
                                        ;
cim_4:          call  pontir            ; Balról jobbra vízszintes vonal
                inc   bx                ; húzása ciklikus képpontkiírással.
                loop  cim_4             ;
                mov   cx, 10000         ; Várakozási ciklus a sebesség
szun4:          loop  szun4             ; beállításához.
                pop   ax                ; Vonalhosszak behívása a veremről.
                pop   cx                ;
                                        ;
                cmp   ax, 346           ; Ha az utolsó függőleges elérte a
                ja    cim_8             ; a max. hosszt, kész az ábra.
                                        ;
                add   cx, diffosz       ; A következő vízszintes hosszának
                add   diffosz, 2        ; beállítása.
                                        ;
                push  cx                ; Vonalhosszak elmentése a veremre.
                mov   cx, ax            ; A vonalhossz -> CX (számláló).
                push  ax                ;
                                        ;
cim_5:          call  pontir            ; Lentről felfele függőleges vonal
                dec   dx                ; húzása ciklikus képpontkiírással.
                loop  cim_5             ;
                mov   cx, 10000         ; Várakozási ciklus a sebesség
szun5:          loop  szun5             ; beállításához.
                pop   ax                ; Vonalhosszak behívása a veremről.
                pop   cx                ;
                add   ax, 2             ; A következő függőleges hosszának
                jmp   cim_1             ; beállítása, a ciklus újrakezdése.
                                        ;
cim_8:          xor   ah, ah            ; Várakozás billentyű megnyomására.
                int   16h               ; 16(h) megszakítás.
                                        ;
```

```
            call szoveg             ; A karakteres képernyõ bekapcsolása.
            mov  cx,0d00h           ; A teljes kurzor bekapcsolása.
            call kurdef             ;
            call clst               ; Képernyõ törlése.
            mov  bx,0               ; Kezdés a bal felsõ sarokban.
            call cimszam            ; Video RAM ofszetcím számítás.
            mov  si,offset txt1     ; Az 1. felirat ofszetcíme.
            mov  cx,200             ; A felirat 10 karakterbõl áll.
cim_6:      call kiiro              ; A felirat kiírása.
            loop cim_6              ;
                                    ;
            inc  bp                 ; Képernyõoldal váltás.
            call clst               ; Képernyõ törlése.
            mov  bx,0               ; Kezdés a bal felsõ sarokban.
            call cimszam            ; Video RAM ofszetcím számítás.
            mov  si,offset txt2     ; A 2. felirat ofszetcíme.
            mov  cx,200             ; A felirat 10 karakterbõl áll.
cim_7:      call kiiro              ; A felirat kiírása.
            loop cim_7              ;
                                    ;
cim_9:      setmode 10001000b       ; Az elsõ képernyõoldal bekapcsolása.
                                    ;
szunet1:    loop szunet1            ; Szunet.
                                    ;
            setmode 00001000b       ; A második képoldal bekapcsolása.
                                    ;
szunet2:    loop szunet2            ; Szunet.
                                    ;
            mov  ah,1               ; Billentyûtesztelés funkciókódja.
            int  16h                ; 16(h) megszakítás.
            je   cim_9              ; Ha nem volt billentyû : folytatni.
                                    ;
            xor  ah,ah              ; Várakozás billentyû megnyomására.
            int  16h                ; 16(h) megszakítás.
                                    ;
            mov  bp,0               ; Elsõ képernyõoldal -> BP.
            call clst               ; Képernyõ törlése.
            mov  cx,0D0Ch           ; A kurzor visszaállítása.
            call kurdef             ;
                                    ;
vege:       mov  ax,4C00h           ; 21(h) megszakítás, a program a 0
            int  21h                ; hibakóddal ér véget.

;— A demonstrációs program adatai ————————————————————————————

txt1        db " ◄◄ 1. oldal ►► ",0
txt2        db " ►► 2. oldal ◄◄ ",0

CR          equ 13                  ; Kocsivissza ASCII kódja.
LF          equ 10                  ; Soremelés ASCII kódja.
KEND        equ "$"                 ; Karakterlánc vége jelzés.

diffosz     dw 5
diffsor     dw 2

karlanc     db CR,LF,"A példaprogram csak Hercules  képernyõkártyával"
            db CR,LF,"rendelkezõ gépeken futtatható. Amennyiben az on"
            db CR,LF,"gépében ettõl eltérõ képernyõkártya  található,"
            db CR,LF,"állítsa  le a programot  az <s> billentyû  meg-"
            db CR,LF,"nyomásával,  ellenkezõ esetben  egy tetszõleges"
            db CR,LF,"másik billentyû  megnyomásával a  program foly-"
            db CR,LF,"tatható ... ",CR,LF,KEND
```

```
indítás  db CR,LF,"A program elsőként a grafikus képernyőkezelést,"
         db CR,LF,"majd a szöveges (karakteres) üzemmód egyes"
         db CR,LF,"elemeit szemlélteti. A várakozási állapotokból"
         db CR,LF,"egy tetszőleges billentyű megnyomásával léphet"
         db CR,LF,"tovább ....",CR,LF,KEND

;-------------------------------------------------------------------
;-- A demonstrációhoz felhasznált funkciók -----------------------
;-------------------------------------------------------------------
;-- KONF : a konfigurációs regiszter beállítása ------------------
;-- BE : AL = bit 0 : 0/1 csak karakteres/grafikus képernyő is
;--           bit 1 : 0/1 második képernyőoldal használható nem/igen
;-- KI : -
;-- Regiszterek : AX és DX megváltoznak

konf     proc near

         mov  dx,3BFh              ; A konfigurációs regiszter portcíme-
         out  dx,al                ; re (3BFh) az új érték kiküldése.
         ret                       ; Vissza a hívóhoz.

konf     endp

;-- SZOVEG : a karakteres képernyő bekapcsolása ------------------
;-- BE : -
;-- KI : -
;-- Regiszterek : AX és DX megváltoznak

szoveg   proc near

         mov  bl,00100000b         ; A vezérlőregiszter új tartalma
                                   ; (szöveg, 0. oldal, villogás
                                   ; engedélyezve).
         mov  si,offset szov       ; Az új regiszteradatok címe.
         call kkprog               ; A vezérlőkártya beállítása.
         ret                       ; Vissza a hívóhoz.

szoveg   endp

;-- GRAFIKA : a grafikus képernyő bekapcsolása -------------------
;-- BE : -
;-- KI : -
;-- Regiszterek : AX és DX megváltoznak

grafika  proc near

         mov  bl,00000010b         ; A vezérlőregiszter új tartalma
                                   ; (0. oldal, grafikus mód)
         mov  si,offset graf       ; Az új regiszteradatok címe.
         call kkprog               ; A vezérlőkártya beállítása.
         mov  bl,00001010b         ; Képernyő visszakapcsolása.
         setmode bl
         ret                       ; Vissza a hívóhoz.

grafika  endp

;-- KKPROG : a képernyőkártya programozása -----------------------
;-- BE : SI = a regiszterek táblázatos adatainak címe
;        BL = a vezérlőregiszter új tartalma
;-- KI : -
;-- Regiszterek : AX, SI, BH, DX és FLAGS változnak
```

```
kkprog  proc near

        setmode bl              ; Vezérlőregiszter beállítása (ha a
                                ; 3.bit = 0, a képernyő kikapcsolva).
                                ;
        mov  cx,12              ; Állítandó regiszterek száma -> CX.
        mov  bh,0               ;
cimi:   lodsb                   ; Ciklusban :
        mov  ah,al              ;     új érték   -> AH.
        mov  al,bh              ;     Reg. száma -> AL.
        call regir              ;     A beállítás elvégzése.
        inc  bh                 ;     Regiszterszámláló növelése.
        loop cimi               ;
                                ;
        or   bl,8               ; Vezérlőregiszter ismételt
        setmode bl              ; állítása (3.bit = 0, a képernyő
        ret                     ; bekapcsolása).

kkprog  endp

;— KURDEF : a kurzor kezdő- és végsorának beállítása ————————————
;— BE : CL = kurzor kezdősora
;—      CH = kurzor utolsó sora
;— KI : -
;— Regiszterek : AX és DX megváltoznak

kurdef  proc near

        mov  al,10              ; Kezdősor a 10-es regiszterbe kerül.
        mov  ah,cl              ; Kezdősor új értéke -> AH.
        call regir              ; A képernyővezérlő átállítása.
        mov  al,11              ; Végsor a 11-es regiszterbe kerül.
        mov  ah,ch              ; Végsor új értéke -> AH.
        call regir              ; A képernyővezérlő átállítása.

        ret                     ; Vissza a hívóhoz.

kurdef  endp

;— REGIR : a vezérlő egyik regiszterébe megadott érték írása ————
;— BE : AL = a regiszter sorszáma
;       AH = a beírandó érték
;— KI : -
;— Regiszterek : DX és AL megváltoznak

regir   proc near

        mov  dx,3B4h            ; A regiszter sorszámának kiírása
        out  dx,al              ; az indexregiszterbe (portcím 3B4h).
        mov  dx,3B5h            ;
        mov  al,ah              ; Az adat kiírása az adatregiszterbe
        out  dx,al              ; (portcím 3B5h).
                                ;
        ret                     ; Vissza a hívóhoz.

regir   endp

;— REGOLV : a vezérlő egyik regiszterének kiolvasása ————————————
;— BE : AL = a regiszter sorszáma
;— KI : AL = a beolvasott érték
;— Regiszterek : DX és AL megváltoznak

regolv  proc near
```

```
        mov   dx,3B4h                ; A regiszter sorszámának kiírása
        out   dx,al                  ; az indexregiszterbe (portcím 3B4h).
        mov   dx,3B5h                ; Az adat beolvasása az adat-
        in    al,dx                  ; regiszterből (portcím 3B5h).
                                     ;
        ret                          ; Vissza a hívóhoz.

regolv  endp

;— CLST : a teljes képernyő letörlése karakteres üzemmódban ————————
;— BE  : -
;— KI  : -
;— Regiszterek : FLAGS megváltozik

clst    proc near                    ; A torlendő ablak definiálása :
                                     ;
        mov   ah,07h                 ;     Színe fekete - fehér lesz.
        mov   bl,0                   ;     Bal felső sor : 0.
        mov   bh,0                   ;     Bal felső oszlopa :0.
        mov   dl,25                  ;     összesen 25 sor.
        mov   dh,80                  ;     összesen 80 oszlop.
                                     ;
        call  torol                  ; Az ablak törlése.
        ret                          ; Visszatérés a hívóhoz.

clst    endp

;— TOROL : megadott képernyőablak feltöltése szóközökkel (törlés) ——
;— BE : AH = szín / attribútum
;       BL = a bal felső sarok sora
;       BH = a bal felső sarok oszlopa
;       DL = a sorok száma összesen
;       DH = az oszlopok száma összesen
;— KI : -
;— Regiszterek : FLAGS megváltozik

torol   proc near

        cld                          ;
        push  cx                     ; Regiszterek elmentése a veremre.
        push  si                     ;
        push  di                     ;
        push  es                     ;
        call  cimszam                ; Video RAM ofszetcím számítás.
        mov   cx,0B000h              ; Video RAM szegmenscím -> ES.
        mov   es,cx                  ;
        mov   ch,0                   ;
        mov   al," "                 ; Szóköz karakter -> AL.
toroli: mov   si,di                  ; DI -> SI mentés.
        mov   cl,dh                  ; Oszlopok száma -> CL (számlálóba).
        rep   stosw                  ; Szó kiírása (itt szóköz) ciklusban.
        mov   di,si                  ; SI -> DI visszatöltés.
        add   di,160                 ;
        dec   dl                     ; Ha nem ez volt az utolsó sor, akkor
        jne   toroli                 ; ofszetcím számítás és újra kiírás.
                                     ;
        pop   es                     ; A regiszterek visszaállítása.
        pop   di                     ;
        pop   si                     ;
        pop   cx                     ;
        ret                          ; Vissza a hívóhoz.

torol   endp
```

```
;— KIIRO : egy karakterlánc kiírása a képernyőre ————————————————
;— BE : AH = szín / attribútum
;       DI = a képernyő kívánt helyének a címe
;       SI = a kiírandó karakterlánc címe
;— KI : DI a képernyőn a karakterlánc utáni első pozícióra mutat
;— Regiszterek : AL, DI és FLAGS megváltoznak

kiiro   proc near

        cld                     ;
        push si                 ; Regiszterek elmentése a veremre.
        push es                 ;
        push dx                 ;
        mov  dx,0B000h          ; Video RAM szegmenscím -> ES.
        mov  es,dx              ;
        lodsb                   ; Ha a karakterlánc első karaktere
        or   al,al              ; a '0' karakter, akkor vége.
        je   kiiro2             ;
kiiro1: stosw                   ; Karakter és színattribútum
        lodsb                   ; ciklusban a képernyőre, mindaddig
        or   al,al              ; amíg a lezáró '0' sorra nem kerül.
        jne  kiiro1             ;
                                ;
kiiro2: pop  dx                 ; Regiszterek visszatöltése a
        pop  es                 ; veremről.
        pop  si                 ;
        ret                     ; Vissza a hívóhoz.

kiiro   endp

;— CIMSZAM : sor és oszlop értékből cím képzése ——————————————————
;— BE : BL = sor
;       BH = oszlop
;— KI : DI = a kiszámított cím
;— Regiszterek : DI és FLAGS megváltoznak

cimszam proc near

        push ax                 ; Regiszterek elmentése a veremre.
        push bx                 ;
                                ;
        shl  bx,1               ; Sorok és oszlopok 2-vel szorzandók
                                ; (minden második pozíción van kar.).
        mov  al,bh              ; Oszlopkoordináta -> AL.
        mov  bh,0               ; A sorkoordináta -> DI, az adat-
        mov  di,[sorcim+bx]     ; táblázat alapján.
        mov  ah,0               ; Az oszlopkoordináta hozzáadódik
        add  di,ax              ; DI értékéhez.
                                ;
        or   bp,bp              ; Ha nem az első (0-ás) képernyő-
        je   tovabb             ; oldalon vagyunk, 32 kbájt (8000h)
        add  di,8000h           ; korrekció szükséges.
                                ;
tovabb: pop  bx                 ; Regiszterek visszatöltése
        pop  ax                 ; a veremről.
        ret                     ; Vissza a hívóhoz.

cimszam endp
```

```
;— CLSG : a grafikus képernyő törlése ————————————————————————
;— BE : AL = 00h : minden pont törlése
;            FFh : minden pont kiírása
;       BP = a képernyőoldal sorszáma
;— KI : -
;— Regiszterek   AH, BX, CX, DI és FLAGS megváltoznak

clsg    proc near

        push es                     ; ES -> verem.
        cbw                         ; AL kiterjesztése AH-ra.
                                    ;
        mov  di,0                   ; Video RAM ofszetcím -> DI.
        mov  bx,0B000h              ; Video RAM szegmenscím -> ES.
                                    ;
        or   bp,bp                  ; Ha az első oldalt kell törölni,
        je   clsg1                  ; (BP=0) akkor megfelel a szegmenscím.
                                    ;
        mov  bx,0B800h              ; A második oldal esetén új ofszet.
                                    ;
clsg1:  mov  es,bx                  ;
        mov  cx,4000h               ; 32 Kbájtot kell kiírni -> CX.
        rep  stosw                  ; Az oldal feltoltése ciklusban.
                                    ;
        pop  es                     ; Verem -> ES.
        ret                         ; Vissza a hívóhoz.

clsg    endp

;— PONTIR : egy pont kiírása a grafikus képernyőre ———————————————
;
;          A bájt relatív címének kiszámítása és a bájton
;          belül a bit behatárolása a fejezetben
;          ismertetett képleteknek megfelelően történik.
;
;— BE : BP = a képernyőoldal sorszáma (0 vagy 1)
;       BX = oszlop (0-719)
;       DX = sor    (0-347)
;— KI : -
;— Regiszterek : AX, DI és FLAGS megváltoznak

pontir  proc near

        push es                     ; Regiszterek elmentése a veremre.
        push bx                     ;
        push cx                     ;
        push dx                     ;
                                    ;
        mov  di,0                   ; Video RAM ofszetcím -> DI.
        mov  cx,0B000h              ; Video RAM szegmenscím -> CX.
                                    ;
        or   bp,bp                  ; Ha az első oldalt kell törölni,
        je   pontir1                ; (BP=0) akkor megfelel a szegmenscím.
                                    ;
        mov  cx,0B800h              ; A második oldal esetén új ofszet.
                                    ;
pontir1: mov es,cx                  ; Video RAM szegmenscím -> ES.
        mov  ax,dx                  ; Sorkoordináta -> AX.
        shr  ax,1                   ; INT(sorkoordináta/4) -> AX.
        shr  ax,1                   ;
        mov  cl,90                  ;
        mul  cl                     ; AX * 90 -> AX.
```

```
        and  dx,11b             ; Sorkoordináta mod 4 -> DX.
        mov  cl,3               ;
        ror  dx,cl              ; DX * 8192 -> DX (forgatással).
        mov  di,bx              ; Oszlopkoordináta -> DI.
        mov  cl,3               ; INT(oszlopkoordináta/8) -> DI.
        shr  di,cl              ;
        add  di,ax              ; DI-ben a relatív cím összeállítása
        add  di,dx              ; DI, BX és DX tartalmából.
                                ;
        mov  cl,7               ; A bit eltolása max. 7.
        and  bx,7               ;
        sub  cl,bl              ; 7 - oszlop mod 8 -> CL.
        mov  ah,1               ; A pont bitértékének kiszámítása.
        shl  ah,cl              ;
        mov  al,es:[di]         ; Egy bájt beolvasása a videoRAM-ból.
        or   al,ah              ; Az aktuális pont beállítása.
        mov  es:[di],al         ; A képernyőpont(ok) megjelenítése.
                                ;
        pop  dx                 ; A regiszterek visszatöltése
        pop  cx                 ; a veremről.
        pop  bx                 ;
        pop  es                 ;
        ret                     ; Vissza a hívóhoz.

pontir  endp

;-- ADATOK ------------------------------------------------------------

sorcim  dw    0, 160, 320, 480, 640     ; A 25 képernyősor oldalon
        dw  800, 960,1120,1280,1440     ; belüli ofszetcíme a
        dw 1600,1760,1920,2080,2240     ; video RAM-ban.
        dw 2400,2560,2720,2880,3040
        dw 3200,3360,3520,3680,3840

graf    db 35h, 2Dh, 2Eh, 07h, 5Bh, 02h ; A grafika bekapcsolásának
        db 57h, 57h, 02h, 03h, 00h, 00h ; regiszterértékei.

szov    db 61h, 50h, 52h, 0Fh, 19h, 06h ; A grafika kikapcsolásának
        db 19h, 19h, 02h, 0Dh, 0Bh, 0Ch ; regiszterértékei.

;-- VÉGE --------------------------------------------------------------

kod     ends                    ; A (kód)szegmens vége.
        end  start              ; Az assembler forrásprogram vége.
                                ; A program végrehajtása a start
                                ; címkén kezdődik.
```

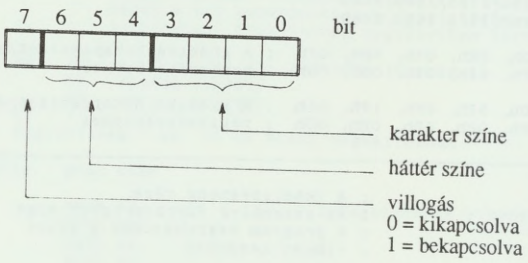
2.4. Az IBM színes-grafikus képernyőkártya programozása

Az IBM színes-grafikus képernyőkártyája kétféle szöveges, és háromféle grafikus felbontást tesz lehetővé. (Mint már volt róla szó, a PC-k az utóbbiból csak kettőt használnak ki.) Az előzőekben ismertetett kártyákhoz hasonlóan a színes-grafikus kártya is a Motorola 6845-ös videoprocesszorát tartalmazza. A kártya video-RAM-ja a B800:0000 – B800:BFFF közötti 16 kbájtos címterületen helyezkedik el.

Lássuk először a szöveges üzemmódokat. A két lehetséges üzemmód egyike a már ismert, 25 sor * 80 oszlop felbontású megjelenítés, míg a másik üzemmódban az oszlopok száma 40, és a karakterek kétszeres szélességűek. A karaktereket mindkét esetben egy 8 * 8 képpontból álló mátrix tartalmazza, aminek következtében lényegesen rosszabb felbontás érhető el, mint a monokróm kártyával. A video-RAM felépítése megegyezik a monokróm kártyáéval.

Az attribútum bájt felépítésével, az egyes bitek jelentésével és a különböző színeket létrehozó bitkombinációkkal a könyv BIOS-ról szóló részének 1.3.1. pontjában már részletesen foglalkoztunk (lásd az 5. és 6. ábrát is), ezért itt most – ismétlésként – csak a bájt felépítését és a színeknek megfelelő bitkombinációkat mutatjuk be.

Az attribútum bájt és bitjei:



Színek és bitkombinációk:

bitkombináció	érték	szín
0000	0	fekete
0001	1	kék
0010	2	zöld
0011	3	türkiz
0100	4	piros
0101	5	bíbor
0110	6	barna (esetleg sárga)
0111	7	világosszürke (esetleg fehér)
1000	8	sötétszürke (esetleg fekete)
1001	9	világoskék
1010	10	világoszöld
1011	11	világostürkiz
1100	12	világospiros
1101	13	világosbíbor
1110	14	sárga (esetleg világossárga)
1111	15	fehér (esetleg kiemelt fehér)

A 80 * 25 karakteres felbontásban a 16 kbájtos video-RAM-ban egyidejűleg 4 képernyőoldalnyi (4 * 4 kbájt), 40 * 25-ös felbontás esetén pedig 8 képernyőoldalnyi (8 * 2 kbájt) információ építhető fel. Az aktív képernyőoldal kiválasztására rövidesen visszatérünk.

Tekintsük át most a grafikus üzemmódokat. Mint erről már volt szó, a színes-grafikus videokártya a maga részéről háromféle felbontást tesz lehetővé: a 160 * 100 képpontos kis, a 320 * 200 képpontos közepes és a 640 * 200 képpontos nagy felbontást, de a kis felbontást a ROM-BIOS nem támogatja. A gyakorlatban ezért csak a közepes és a nagy felbontásnak van jelentősége. Közepes felbontásban a grafika megjelenítéséhez 4 szín választható, míg a nagy felbontás csak 2 szín (előtér, háttér) használatát teszi lehetővé.

A színek számát a videokártya meglehetősen szűkre szabott RAM tára korlátozza. Ha egy képpontot csak 1 biten ábrázolunk, akkor ezzel csak arra van lehetőségünk, hogy a képpontot be- vagy kikapcsoljuk – ez nem más, mint a monokróm megjelenítés. Ahhoz tehát, hogy a képpontnak valamiféle más színt is adhassunk, képpontonként legkevesebb 2 bittel kell számolnunk. Közepes felbontás esetén ez 320 * 200 * 2 = 128000 bit, azaz 16000 bájt kezelését jelenti. Ez a tárigény gyakorlatilag majdnem akkora, mint a videokártya rendelkezésére álló teljes RAM tár területe (16 kbájt). Ebből következik, hogy ebben a felbontásban egy képpontot maximum 2 biten ábrázolhatunk, és csak egy képernyőoldallal dolgozhatunk.

Mivel a képpontok ábrázolásához íly módon rendelkezésre álló 2–2 biten négy különböző érték ábrázolható, lehetővé válik, hogy egy képpont négy különböző szín valamelyikén jelenjen meg. A négy érték egyike – egy porton keresztül – a háttér színének megválasztását teszi lehetővé (a portra rövidesen kitérünk). A háttér színe 16 szín közül választható meg. Ha a képpontot a háttér színétől meg akarjuk különböztetni (vagyis láttatni akarjuk), akkor erre még három érték áll rendelkezésünkre. Azért, hogy ezt a három értéket – amelynek természetesen három szín feleltethető meg – a szűk tárkapacitás elle-

nére viszonylag rugalmasan lehessen felhasználni, az IBM két, ún. színpalettát definiált. A színpalettát is az előbb említett porton (ill. a megfelelő BIOS funkción) keresztül lehet kiválasztani. Mindegyik színpaletta három-három, előre definiált színt tartalmaz.

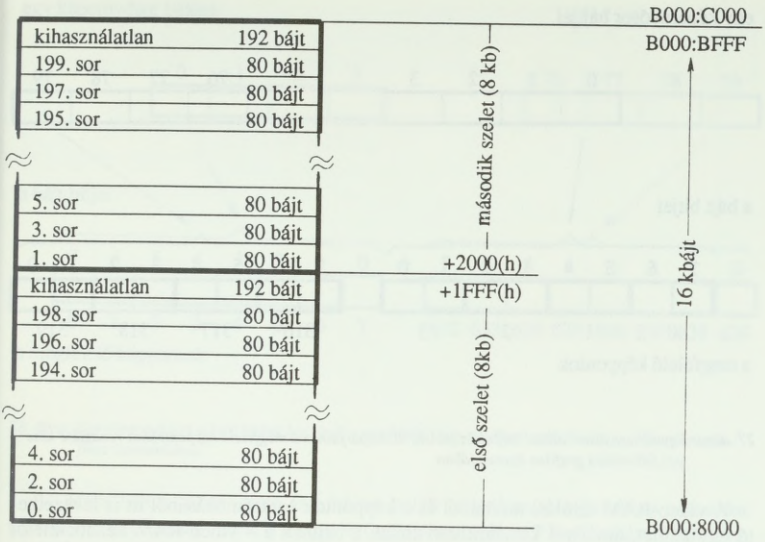
A színpalettákon lévő színek:

1. színpaletta	1. színe:	türkiz
	2. színe:	bíbor
	3. színe:	fehér
2. színpaletta	1. színe:	zöld
	2. színe:	piros
	3. szine:	sárga

A képpontot megjelenítő két bit kódolása:

00 = a 16 háttérszín egyike
01 = a kiválasztott színpaletta 1. színe
10 = a kiválasztott színpaletta 2. színe
11 = a kiválasztott színpaletta 3. színe

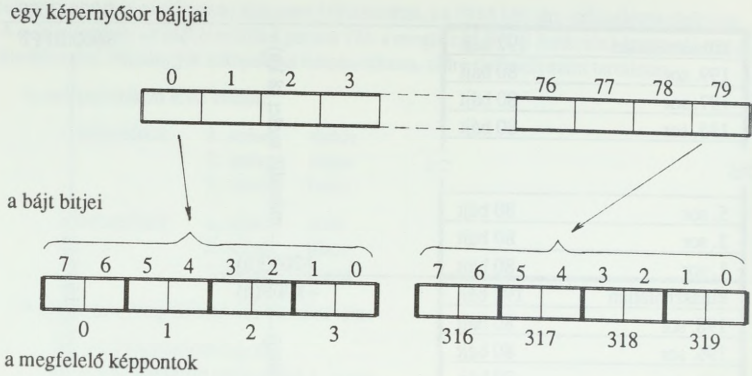
Grafikus üzemmódban a video-RAM hasonló elven épül fel, mint a Hercules grafikus kártya video-RAM-ja. Az IBM kártya azonban a video-RAM-t csak két, egyenként 8–8 kbájtos szeletre osztja fel, és a képernyősorokat felváltva tárolja hol az egyik, hol a másik szeletben. Így az egyik (alacsonyabb tárcímeken lévő) szelet csak a páros, míg a másik szelet csak a páratlan sorszámú képernyősorokat tartalmazza. A video-RAM felépítését és a képernyősorok tárolásának módját a 26. ábra szemlélteti.



26. ábra: A képernyősorok tárolása a video-RAM-ban grafikus üzemmódban (IBM CGA)

Az ábrához most is fűzünk rövid magyarázatot: mivel egy képernyősor 320 képpontból áll, és minden képponthoz 2 bit szükséges, a teljes képernyősor ábrázolásához 640 bitre, azaz 80 bájtra van szükség. Ahhoz, hogy a 200 sor a két szelet között egyenletesen oszoljon el, egy-egy szeletnek 100–100 képernyősort kell tárolnia. Ezek összesen 8000 bájtot vesznek igénybe a szeletek 8 kbájtjából, így a szeletek „tetején" 8192 – 8000 = 192 bájt kihasználatlan marad.

Vizsgáljuk meg most a képernyőpontokat ábrázoló bitpárok képernyősoron belüli tárolásának módját. Láttuk, hogy egy képernyősort 80 bájt tárol. A sorban a 0. bájt 8 bitje a sort alkotó első 4 (0–3.) képpontnak felel meg úgy, hogy a 7. és 6. bit a nulladik, az 5. és 4. bit az első, a 3. és 2. bit a második, az 1. és 0. bit a harmadik képpont színinformációit tartalmazza. A bájtok bitjeinek, ill. bitpárjainak a képpontokhoz rendelését a 27. ábra szemlélteti.



27. ábra: Egy képernyősort alkotó bájtok bitjeinek, ill. bitpárjainak a megfelelő képpontokhoz rendelése közepes felbontású grafikus üzemmódban

A video-RAM tárolási módjából és a képpontok sorszámozásából itt is levezethető egy képlet, amellyel kiszámítható annak a bájtnak a – video-RAM kezdőcímétől számított – relatív címe, amelyik egy meghatározott képpontra vonatkozó színinformációkat tartalmazza. A képpontot a képernyőn értelmezett X és Y koordinátáival kell megadni (X = 0 - 319, Y = 0 - 199).

A bájt relatív címe:

cím = 8192 * (Y mod 2) + 80 * int(Y/2) + int(X/4)

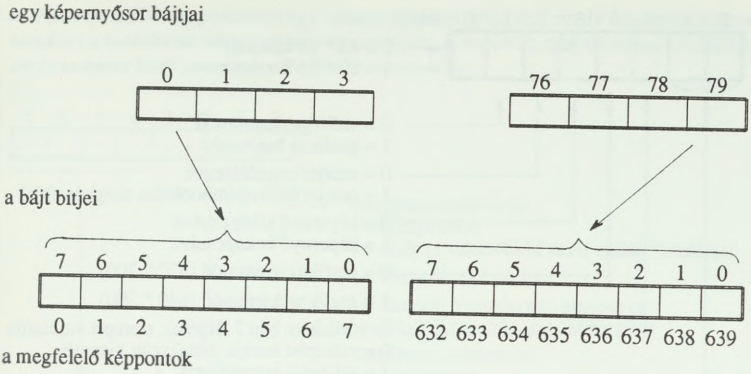
A bájton belüli bitpár meghatározásához a következő képlet használható:

a bit száma = 6 – 2 * (X mod 4)

Ha a bit így kiszámított száma 0, akkor a színinformációt a bájton belül a 0. és 1. bitek, ha pedig az eredmény pl. 6, akkor a 6. és 7. bitek tartalmazzák.

A 640 * 200 képpontos, nagy felbontású grafikus üzemmódban a rendszernek 128000 képpontot kell kezelnie, vagyis (majdnem) annyit, ahány bit a video-RAM-ban összesen a rendelkezésére áll. Emiatt egy képpont csak egy bittel ábrázolható, tehát a megjelenítés csak monokróm lehet.

A video-RAM felépítése ugyanolyan, mint a közepes felbontásnál. Egy képernyősort itt is 80 bájt tárol, csak az ebben lévő 640 bit most nem 320, hanem 640 képpontot jelenít meg. A bitek és képpontok egymáshoz rendelését a 28. ábra mutatja be.



28. ábra: Egy képernyősort alkotó bájtok bitjeinek a megfelelő képpontokhoz rendelése nagy felbontású grafikus üzemmódban

Az egyes bájtok relatív címe és a bájton belüli bit száma az előzőekhez hasonlóan számítható. A képpontot most is a képernyőn értelmezett X és Y koodinátáival kell megadni (X = 0 - 639, Y = 0 - 199).

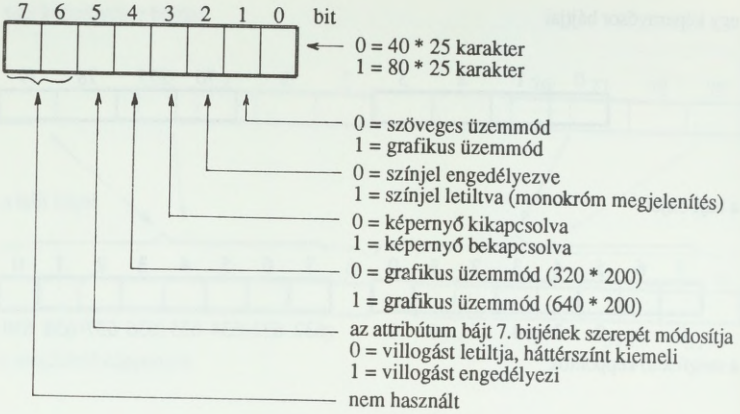
A bájt relatív címe:

cím = 8192 * (Y mod 2) + 80 * int(Y/2) + int(X/8)

A bit száma = 7 – (X mod 8)

A következőkben ismerkedjünk meg az IBM színes-grafikus videokártya portjaival és a kártya programozásával.

A kártya a 3D8(h) portcímen egy üzemmódválasztó regisztert tartalmaz, amely feladatát tekintve megfelel a monokróm kártya vezérlőregiszterének. A regiszter csak írható. Felépítését és az egyes bitek jelentését a 29. ábrán mutatjuk be.



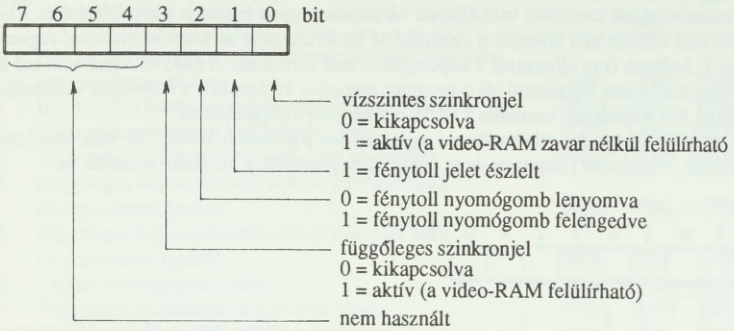
29. ábra: Az üzemmódválasztó regiszter felépítése és az egyes bitek jelentése (IBM CGA)

Csak néhány bithez kell megjegyzést fűzni: A 2. bit értéke csak azok számára érdekes, akik színes videokártyával és monokróm monitorral dolgoznak. Ha a bit értéke 1, akkor a 6845-ös elnyomja a színjelet, és monokróm képet jelenít meg. A 3. bit 0-ra állításával letiltjuk a kép előállítását (mintegy kikapcsoljuk a képernyőt). Erre a különböző megjelenítési módok közötti átváltáskor van szükség. Az 5. bitnek ugyanaz a jelentése, mint a monokróm kártyánál. A bit az attribútum bájt 7. bitjének szerepét módosítja, amennyiben ez utóbbi bit értéke 1: ha ekkor a vezérlőregiszter 5. bitjének értéke 0, akkor az attribútum bájthoz tartozó karakter villogását letiltja, és kiemeli a háttérszínt. Ebben az esetben az attribútum bájt 7. bitje tulajdonképpen a háttér színére vonatkozó intenzitásbitnek tekinthető.

Az ábrából látható, hogy a megjelenítés módját az üzemmódválasztó regiszter 0., 1., 2. és 4. bitje határozza meg. Az egyes kombinációk:

4. bit	2. bit	1. bit	0. bit	megjelenítés módja
0	1	0	0	40 * 25 szöveges mód, monokróm
0	0	0	0	40 * 25 szöveges mód, színes
0	1	0	1	80 * 25 szöveges mód, monokróm
0	0	0	1	80 * 25 szöveges mód, színes
0	1	1	0	320 * 200 grafikus mód, monokróm
0	0	1	0	320 * 200 grafikus mód, színes
1	1	1	0	640 * 200 grafikus mód, monokróm

A kártya a 3DA(h) portcímen egy státuszregisztert tartalmaz, amely feladatát tekintve megfelel a monokróm kártya státuszregiszterének. A regiszter csak olvasható. Felépítését és az egyes bitek jelentését a 30. ábrán mutatjuk be.



30. ábra: A státuszregiszter felépítése és az egyes bitek jelentése (portcíme 3DA(h))

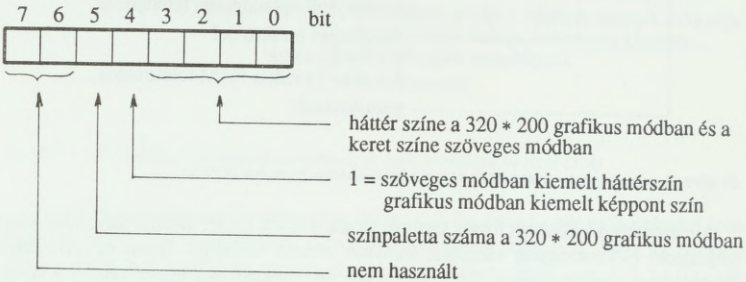
A regiszterben különösen fontos szerepet játszik a 0. bit. Ez azt mutatja meg, hogy képet megrajzoló elektronsugarat vízszintes irányban eltérítő vezérlőjel éppen egy vízszintes (sor)szinkron jelet tartalmaz-e. Ezt a jelet a 6845-ös minden alkalommal kiküldi a képernyőre, amikor az elektronsugár egy teljes szélességű pontsort kirajzolt. Ez a szinkronjel jelzi, hogy az elektronsugár elérte a képernyő végét, és vissza kell ugrania a következő sor elejére. Ebben az időpillanatban a bit értéke 1. A bit jelentőségét az adja, hogy az IBM színes-grafikus kártyánál a video-RAM-hoz nem lehet tetszőleges időben hozzáférni.

Ennek az az oka, hogy a 6845-ösnek folyamatosan olvasnia kell ezt a tárat ahhoz, hogy a képernyőn folyamatosan megjeleníthesse a tartalmát. Ha egy program éppen akkor akar a tárhoz hozzáférni, amikor a 6845-ös is, akkor ez az ütközés a képernyőn zavaró jelek formájában megjelenik (a képernyő „havazik"). Ez az ütközés úgy kerülhető el, ha a program mindig csak akkor férhet hozzá a video-RAM-hoz, amikor a 6845-ös „szünetet" tart. Ilyen szünet akkor van, amikor az elektronsugár elérte a sor végét, és a vízszintes (sor)szinkronjel hatására visszaugrik a következő sor elejére. Ennek a sorvisszafutásnak az időtartama ugyan nagyon rövid, de ahhoz elég, hogy közben néhány bájtot a video-RAM-ba írhassunk anélkül, hogy ez a megjelenített képen zavart okozna. Ezért az IBM színes-grafikus kártyájának programozásakor a video-RAM-hoz való hozzáférés előtt mindaddig be kell olvasni a státuszregiszter tartalmát, amig a 0. bit értéke nem lesz 1, és csak ekkor szabad a video-RAM-ba írni. A sorvisszafutási időhöz hasonlóan a képvisszafutási idő is kihasználható arra, hogy a megjelenített kép megzavarása nélkül férhessünk a video-RAM-hoz. Ennek a függőleges szinkronjelnek a kezdetét – vagyis amikor az elektronsugár elérte az utolsó sor utolsó képpontját, és visszaugrik az első sor első pontjához – a regiszter 3. bitjének 1-es értéke jelzi. A képvisszafutás ideje alatt ugyan-

csak hozzáférhetünk a video-RAM-hoz. Természetesen a várakozások mindkét esetben lelassítják a kép felépítését.

Ha a video-RAM-ot a lehető legrövidebb időközönként sokszor kell írni/olvasni (pl. a képernyő görgetésekor), akkor ez a módszer nem eléggé hatékony: az elektronsugár visszafutásának kezdetére való állandó várakozás nagyon lelassítja a kép felépítését. Ekkor más eljárást kell követni: a video-RAM felülírása előtt az üzemmódválasztó regiszter 3. bitjének 0-ra állításával a képernyőt ki kell kapcsolni. A 6845-ös most nem tud a video-RAM-hoz hozzáférni, és a program szabadon elvégezheti a szükséges változtatásokat. Ezt a módszert használja a BIOS is a képernyő görgetéséhez.

A videokártya ún. színkiválasztó regiszterének a portcíme 3D9(h). Ez csak írható regiszter. A regiszter felépítését és az egyes bitek jelentését a 31. ábrán mutatjuk be.



31. ábra: A színkiválasztó regiszter felépítése és az egyes bitek jelentése (portcíme 3D9(h))

A regiszter bitjeinek jelentése az ábrázolás módjától függ. A kétféle szöveges üzemmódban a 0–3. bitekkel a képernyő keretének színe választható ki a rendelkezésre álló 16 szín közül. A 320 * 200 képpontos, közepes felbontású grafikus üzemmódban ugyancsak ezek a bitek adják meg a háttér színét, vagyis mindazon bitek színét, amelyek – mint bitpárok – a 00 bináris bitkombinációval vannak kódolva (lásd ugyanezen fejezet megfelelő részét a 124. oldalon).

Az 5. bit a 320 * 200 képpontos grafikus üzemmódban használható két színpaletta egyikének kiválasztását teszi lehetővé. Ha a bit értékét 1-re állítjuk, akkor az első színpaletta színei (türkiz, bíbor, fehér), ha 0-ra állítjuk, akkor a második paletta színei (zöld, piros, sárga) közül válogathatunk.

Mivel az IBM színes-grafikus képernyőkártyáját is az előzőekben már megismert Motorola 6845-ös videoprocesszor vezérli, a processzor 18 belső regiszteréhez is az ismert módon lehet hozzáférni. A különbség mindössze annyi, hogy ennél a képernyőkártyánál a videoprocesszor regiszterei a 3D4(h) portcímű index-, és a 3D5(h) portcímű adatregiszteren keresztül érhetők el.

A következő felsorolás azokat az értékeket adja meg, amelyeket az egyes regisztereknek a különböző üzemmódokban célszerűen tartalmazniuk kell.

Sorszám	A regiszter jelentése	Üzemmód 40*25	80*25	graf. érték
0	Egy képernyősor (scan-sor) megrajzolásához szükséges idő karakteregységben	38(h)	71(h)	38(h)
1	Egy képernyősorban megjelenített karakterek száma	28(h)	50(h)	28(h)
2	Vízszintes szinkronjel kezdetének ideje	2D(h)	5A(h)	2D(h)
3	Vízszintes szinkronjel szélességének időtartama	0A(h)	0A(h)	0A(h)
4	Függőleges képernyősorok számát jellemző érték (5. regiszterrel együtt)	1F(h)	1F(h)	7F(h
5	Függőleges képernyősorok számát jellemző érték (4. regiszterrel együtt)	06(h)	06(h)	06(h)
6	Látható szövegsorok száma	19(h)	19(h)	64(h)
7	Függőleges szinkronjel kezdetének ideje	1C(h)	1C(h)	70(h)
8	Képernyősorok letapogatásának módja	02(h)	02(h)	02(h)
9	Szövegsor magassága képernyősoronként számítva mínusz 1	07(h)	07(h)	01(h)
10	A villogó kurzor kezdő képernyősora	06(h)	06(h)	06(h)
11	A villogó kurzor befejező képernyősora	07(h)	07(h)	07(h)
12	A képernyőoldal kezdőcímének felső bájtja	00(h)	00(h)	00(h)
13	A képernyőoldal kezdőcímének alsó bájtja	00(h)	00(h)	00(h)
14	A villogó kurzor pozíciójának felső bájtja	00(h)	00(h)	00(h)
15	A villogó kurzor pozíciójának alsó bájtja	00(h)	00(h)	00(h)
16	Fenntartott	00(h)	00(h)	00(h)
17	Fenntartott	00(h)	00(h)	00(h)

E regiszterek közül a programozók érdeklődésére ismét csak azok a regiszterek tarthatnak számot, amelyek segítségével a kurzornak a képernyőn belüli pozíciója és a kurzor alakja határozható meg. (A 8 * 8-as képpontrács miatt itt a kurzor kezdő és végsora a 0–7. közötti tartományban lehet.)

Ezeknek a regisztereknek a programozásával a 2.2.1. Az IBM monokróm képernyőkártya programozása című pontban már részletesen foglalkoztunk. Az ott ismertetett, 10–11. és 14–15. sorszámú 4 regiszterhez itt még hozzá kell venni a 12–13. sorszámú regisztereket. Ezeknek a regisztereknek a kártya 16 kbájtos video-RAM-jának kezdőcímétől (B800:0000) számított, minimálisan lehetséges 2–2 kbájtnyi képernyőoldala közül a mindenkor aktív képernyőoldal 16 bites kezdőcímét kell tartalmazniuk. A kezdőcím felső bájtját a 12. regiszter, alsó bájtját a 13. regiszter tárolja. Alapértelmezés szerint mindkét regiszter tartalma zérus, azaz az aktív képernyőoldal a B800:0000 címen kezdődik.

Az IBM színes-grafikus képernyőkártya programozását is egy példaprogram bemutatásával zárjuk le.

2.4.1. Programpélda

```
;
;  ┌──────────────────────────────────────────────────────────┐
;  │                    I B M C O L O R . A S M               │
;  │                                                          │
;  │ Funkciója : az IBM színes képernyőkártya néhány tulajdonsága-  │
;  │             nak szemléltetése. A 320 x 200 képpontos grafika   │
;  │             kezelése nem szerepel a programban, az formailag   │
;  │             teljesen megfelel az itt bemutatandó 640 x 200     │
;  │             grafika eszköztárának.                       │
;  └──────────────────────────────────────────────────────────┘
;
;— MAKRÓ ————————————————————————————————————————————————————

setmode  macro ujertek                  ; A képernyőkártya üzemmódválasztó
                                        ; regiszterének beállítása.

         mov   dx,3D8h                  ; Ü.mód választó regiszter portcíme.
         mov   al,ujertek               ; Új vezérlőbájt -> AL.
         out   dx,al                    ; Az új érték kiírása a regiszterbe.

         endm

;— KÓD ——————————————————————————————————————————————————————

kod      segment para 'CODE'            ; A kódszegmens definiálása.

         org 100h                       ; A program a PSP után álló első
                                        ; címen kezdődik.

         assume cs:kod, ds:kod, es:kod, ss:kod

                                        ; Csak egy, közös szegmens van.
;— Demonstrációs program ————————————————————————————————————

start:   mov   ah,9                     ; Karakterláncot megjelenítő
         mov   dx,offset karlanc        ; funkció és megszakítás
         int   21h                      ; hívása.
                                        ;
         xor   ah,ah                    ; Várakozás billentyű megnyomására.
         int   16h                      ; 16(h) megszakítás.
         cmp   al,"s"                   ;
         je    nemcol                   ;
         cmp   al,"S"                   ; Az <s> vagy <S> billentyű leütése
         jne   color                    ; esetén a program futása végetér.

nemcol:  jmp   vege

color:   mov   ah,9                     ; Karakterláncot megjelenítő
         mov   dx,offset inditas        ; funkció és megszakítás
         int   21h                      ; hívása.
         xor   ah,ah                    ; Várakozás billentyű megnyomására.
         int   16h                      ; 16(h) megszakítás.
                                        ;
         call  grafika                  ; 640 x 200 képpontos grafika
                                        ; bekapcsolása.
         mov   al,0                     ;
         call  clsg                     ; A grafikus képernyő törlése.
```

```
                        ;
        mov  bx,584     ; A rajz indítása az 584. oszlopon,
        mov  dx,160     ;                      150. soron.
        mov  ax,3       ; Az első függőleges 3 pontból áll.
        mov  cx,2       ; Az első vízszintes 2 pontból áll.
                        ;
cim_1:  push cx         ; Vonalhosszak elmentése a veremre.
        push ax         ;
        mov  al,1       ; A pont színének beállítása.
                        ;
cim_2:  call pontir     ; Jobbról balra vízszintes vonal
        dec  bx         ; húzása ciklikus képpontkiírással.
        loop cim_2      ;
        pop  ax         ; Vonalhosszak behívása a veremről.
        pop  cx         ;
        add  cx,2       ; A következő vízszintes hosszának
                        ; beállítása.
        push cx         ; Vonalhosszak elmentése a veremre.
        mov  cx,ax      ; A vonalhossz -> CX (számláló).
        push ax         ;
        mov  al,1       ; A pont színének beállítása.
                        ;
cim_3:  call pontir     ; Fentről lefele függőleges vonal
        inc  dx         ; húzása ciklikus képpontkiírással.
        loop cim_3      ;
        pop  ax         ; Vonalhosszak behívása a veremről.
        pop  cx         ;
        add  ax,diffsor ; A következő függőleges hosszának
        add  diffsor,1  ; beállítása.
        push cx         ; Vonalhosszak elmentése a veremre.
        push ax         ;
        mov  al,1       ; A pont színének beállítása.
                        ;
cim_4:  call pontir     ; Balról jobbra vízszintes vonal
        inc  bx         ; húzása ciklikus képpontkiírással.
        loop cim_4      ;
        pop  ax         ; Vonalhosszak behívása a veremről.
        pop  cx         ;
                        ;
        cmp  ax, 185    ; Ha az utolsó függőleges elérte a
        ja   cim_8      ; a max. hosszt, kész az ábra.
                        ;
        add  cx,diffosz ; A következő vízszintes hosszának
        add  diffosz,4  ; beállítása.
                        ;
        push cx         ; Vonalhosszak elmentése a veremre.
        mov  cx,ax      ; A vonalhossz -> CX (számláló).
        push ax         ;
        mov  al,1       ; A pont színének beállítása.
                        ;
cim_5:  call pontir     ; Lentről felfele függőleges vonal
        dec  dx         ; húzása ciklikus képpontkiírással.
        loop cim_5      ;
        pop  ax         ; Vonalhosszak behívása a veremről.
        pop  cx         ;
        add  ax,2       ; A következő függőleges hosszának
        jmp  cim_1      ; beállítása, a ciklus újrakezdése.
                        ;
cim_8:  xor  ah,ah      ; Várakozás billentyű megnyomására.
        int  16h        ; 16(h) megszakítás.
                        ;
        call szoveg     ; A karakteres képernyő be-
        mov  bp,0       ; kapcsolása, 0-dik oldal lesz aktív.
```

```
cim_6:  mov  al,"0"                ;
        or   ax,bp                 ; Az oldal száma karakteres alakban
        mov  txt1,al               ; a kiírandó szövegbe kerül.
        call szinez                ; A szín beállítása.
        call oldal                 ; A <BP> képernyőoldal aktiválása.
        call clst                  ; Képernyő törlése.
                                   ;
        mov  bx,0                  ; Kezdősor és -oszlop -> BX.
        call cimszam               ; Video RAM ofszetcím számítás.
        mov  cx,880                ; 880 karakter kerül kiírásra.
        mov  ah,0                  ; A 0-ás színkóddal indul a ciklus.
        mov  si,offset txt1        ; A szöveg címe -> SI.
        dec  ah                    ;
cim_7:  inc  ah                    ; A színérték megnövelése és a szöveg
        call kiiro                 ; (itt karakter) írása ciklusban
        loop cim_7                 ; a felső képernyőharmadba.
                                   ;
        mov  bl,12                 ; Kezdősor -> BL.
        mov  bh,30                 ; Kezdőoszlop -> BH.
        call cimszam               ; 880 karakter kerül kiírásra.
        mov  ah,19                 ;
        mov  si,offset txt1        ; Speciális színérték beállítása után
        call kiiro                 ; két részből álló üzenet megjele-
        mov  si,offset txt2        ; nítése a képewrnyő közepén.
        call kiiro                 ;
                                   ;
        mov  bl,14                 ; Kezdősor és -oszlop -> BX.
        mov  bh, 0                 ;
        call cimszam               ; Video RAM ofszetcím számítás.
        mov  cx,880                ; 880 karakter kerül kiírásra.
        mov  ah,0                  ; A 0-ás színkóddal indul a ciklus.
        mov  si,offset txt1        ; A szöveg címe -> SI.
        dec  ah                    ;
cim_9:  inc  ah                    ; A színérték megnövelése és a szöveg
        call kiiro                 ; (itt karakter) írása ciklusban
        loop cim_9                 ; az alsó képernyőharmadba.
                                   ;
        xor  ah,ah                 ; Várakozás billentyű megnyomására.
        int  16h                   ; 16(h) megszakítás.
        inc  bp                    ;
        cmp  bp,4                  ; Az oldalszám megnövelése után
        jne  cim_6                 ; ha szükséges, még egy tesztciklus.
                                   ;
        mov  bp,0                  ; A 0-dik oldalra kell vissza-
        call oldal                 ; kapcsolni a képernyő törlése
        call clst                  ; ill. a kiszállás előtt.
                                   ;
vege:   mov  ax,4C00h              ; 21(h) megszakítás, a program a 0
        int  21h                   ; hibakóddal ér véget.

;— A demonstrációs program adatai ————————————————————————————

txt1    db 1,0
txt2    db ". képernyőoldal ...",0

CR      equ 13                     ; Kocsivissza ASCII kódja.
LF      equ 10                     ; Soremelés ASCII kódja.
KEND    equ "$"                    ; Karakterlánc vége jelzés.

diffosz dw 3
diffsor dw 2
```

```
karlanc  db CR,LF,"A  példaprogram csak  színes  képernyőkártyával"
         db CR,LF,"rendelkező gépeken futtatható. Amennyiben az on"
         db CR,LF,"gépében ettől eltérő képernyőkártya található,"
         db CR,LF,"állítsa  le a programot  az <s> billentyű  meg-"
         db CR,LF,"nyomásával,  ellenkező esetben  egy tetszőleges"
         db CR,LF,"másik billentyű  megnyomásával a  program foly-"
         db CR,LF,"tatható ...",CR,LF,KEND

indítás  db CR,LF,"A program elsőként a grafikus képernyőkezelést,"
         db CR,LF,"majd az oldalanként történő szövegkezelés egyes"
         db CR,LF,"elemeit szemlélteti. A várakozási állapotokból"
         db CR,LF,"egy tetszőleges billentyű  megnyomásával léphet"
         db CR,LF,"tovább ....",CR,LF,KEND
;-------------------------------------------------------------------
;-- A demonstrációhoz felhasznált funkciók -----------------------
;-------------------------------------------------------------------

;-- SZOVEG : a karakteres képernyő bekapcsolása -------------------
;-- BE : -
;-- KI : -
;-- Regiszterek : AX, SI, BH, DX és FLAGS változnak

szoveg   proc near

         mov  bl,00100001b       ; Az üzemmódválasztó regiszter új
                                 ; tartalma (szöveg 25 x 80, villogás
                                 ; engedélyezve).
         mov  si,offset szov     ; Az új regiszteradatok címe.
         call kkprog             ; A vezérlőkártya beállítása.
         ret                     ; Vissza a hívóhoz.

szoveg   endp

;-- GRAFIKA : a 640 x 200 képpontos grafikus képernyő bekapcsolása --
;-- BE : -
;-- KI : -
;-- Regiszterek : AX, SI, BH, DX és FLAGS változnak

grafika  proc near

         mov  bl,00010010b       ; 640 x 200 képpontos grafika.
         mov  si,offset graf     ; Az új regiszteradatok címe.
         call kkprog             ; A vezérlőkártya beállítása.
         ret                     ; Vissza a hívóhoz.

grafika  endp

;-- KKPROG : a képernyőkártya programozása -----------------------
;-- BE : SI = a regiszterek táblázatos adatainak címe
;        BL = az üzemmódválasztó regiszter új tartalma
;-- KI : -
;-- Regiszterek : AX, SI, BH, DX és FLAGS változnak
```

```
kkprog   proc near

         setmode bl                ; Üzemmódválasztó regiszter
                                   ; beállítása (ha a 3.bit = 0, a
                                   ; képernyõ ki van kapcsolva).
                                   ;
         mov  cx,12                ; Állítandó regiszterek száma -> CX.
         mov  bh,0                 ;
cimi:    lodsb                     ; Ciklusban :
         mov  ah,al                ;     Új érték   -> AH.
         mov  al,bh                ;     Reg. száma -> AL.
         call regir                ;     A beállítás elvégzése.
         inc  bh                   ;     Regiszterszámláló növelése.
         loop cimi                 ;
                                   ;
         or   bl,8                 ; Üzemmódválasztó regiszter újbóli
         setmode bl                ; állítása (3.bit = 0, a képernyõ
         ret                       ; bekapcsolása).

kkprog   endp

;— SZINEZ : a keret és a háttér színének beállítása ————————————————
;— BE : AL = színérték
;— KI : -
;— Regiszterek : AX és DX megváltoznak

szinez   proc near

         mov  dx,3D9h              ; A színkiválasztó regiszter címe.
         out  dx,al                ; A színérték kiadása.
         ret                       ; Vissza a hívóhoz.

szinez   endp

;— OLDAL : a képernyõoldal beállítása ————————————————————————————
;— BE : BP = a beállítandó képernyõoldal száma
;— KI : -
;— Regiszterek : BX, AX, CX és DX változnak

oldal    proc near

         mov  bx,bp                ; Képernyõoldal -> BX.
         mov  cl,5                 ; 2048-cal szorozva kapható meg az
         ror  bx,cl                ; oldal kezdõcíme.
                                   ;
         mov  ah,bh                ; Oldalszám felsõ bájt -> AH.
         mov  al,12                ; Regiszter sorszám (12) -> AL.
         call regir                ; A regiszter beállítása.
                                   ;
         mov  ah,bl                ; Oldalszám alsó bájt -> AH.
         mov  al,13                ; Regiszter sorszám (13) -> AL.
         call regir                ; A regiszter beállítása.
                                   ;
         ret                       ; Vissza a hívóhoz.
```

```
oldal   endp

;— REGIR : a vezérlő egyik regiszterébe megadott érték írása ————————
;— BE : AL = a regiszter sorszáma
;       AH = a beírandó érték
;— KI : -
;— Regiszterek : DX és AL megváltoznak

regir   proc near

        mov  dx,3D4h            ; A regiszter sorszámának kiírása
        out  dx,al              ; az indexregiszterbe (portcím 3D4h).
        mov  dx,3D5h            ;
        mov  al,ah              ; Az adat kiírása az adatregiszterbe
        out  dx,al              ; (portcím 3D5h).
                                ;
        ret                     ; Vissza a hívóhoz.

regir   endp

;— REGOLV : a vezérlő egyik regiszterének kiolvasása ————————————————
;— BE : AL = a regiszter sorszáma
;— KI : AL = a beolvasott érték
;— Regiszterek : DX és AL megváltoznak

regolv  proc near

        mov  dx,3D4h            ; A regiszter sorszámának kiírása
        out  dx,al              ; az indexregiszterbe (portcím 3D4h).
        mov  dx,3D5h            ; Az adat beolvasása az adat-
        in   al,dx              ; regiszterből (portcím 3D5h).
                                ;
        ret                     ; Vissza a hívóhoz.

regolv  endp

;— CLST : a teljes képernyő letörlése karakteres üzemmódban ————————
;— BE  : -
;— KI  : -
;— Regiszterek : FLAGS megváltozik

clst    proc near

                                ; A törlendő ablak definiálása :
                                ;
        mov  ah,07h             ;     Színe fekete - fehér lesz.
        mov  bl,0               ;     Bal felső sor : 0.
        mov  bh,0               ;     Bal felső oszlopa :0.
        mov  dl,25              ;     összesen 25 sor.
        mov  dh,80              ;     összesen 80 oszlop.
                                ;
        call torol              ; Az ablak törlése.
        ret                     ; Visszatérés a hívóhoz.

clst    endp

;— TOROL : megadott képernyőablak feltoltése szóközökkel (törlés) ——
;— BE : AH = színattribútum
;       BL = a bal felső sarok sora
;       BH = a bal felső sarok oszlopa
;       DL = a sorok száma összesen
;       DH = az oszlopok száma összesen
```

```
;— KI : -
;— Regiszterek : FLAGS megváltozik

torol    proc near

         cld                          ;
         push cx                      ; Regiszterek elmentése a veremre.
         push si                      ;
         push di                      ;
         push es                      ;
         call cimszam                 ; Video RAM ofszetcím számítás.
         mov  cx,0B800h               ; Video RAM szegmenscím -> ES.
         mov  es,cx                   ;
         mov  ch,0                    ;
         mov  al," "                  ; Szóköz karakter -> AL.
toroli:  mov  si,di                   ; DI -> SI mentés.
         mov  cl,dh                   ; Oszlopok száma -> CL (számlálóba).
         rep  stosw                   ; Szó kiírása (itt szóköz) ciklusban.
         mov  di,si                   ; SI -> DI visszatoltés.
         add  di,160                  ;
         dec  dl                      ; Ha nem ez volt az utolsó sor, akkor
         jne  toroli                  ; ofszetcím számítás és újra kiírás.
                                      ;
         pop  es                      ; A regiszterek visszaállítása.
         pop  di                      ;
         pop  si                      ;
         pop  cx                      ;
         ret                          ; Vissza a hívóhoz.

torol    endp

;— KIIRO : egy karakterlánc kiírása a képernyőre ————————————————
;— BE : AH = színattribútum
;       DI = a képernyő kívánt helyének a címe
;       SI = a kiírandó karakterlánc címe
;— KI : DI a képernyőn a karakterlánc utáni első pozícióra mutat
;— Regiszterek : AL, DI és FLAGS megváltoznak

kiiro    proc near

         cld                          ;
         push si                      ; Regiszterek elmentése a veremre.
         push es                      ;
         push dx                      ;
         mov  dx,0B800h               ; Video RAM szegmenscím -> ES.
         mov  es,dx                   ;
         lodsb                        ; Ha a karakterlánc első karaktere
         or   al,al                   ; a '0' karakter, akkor vége.
         je   kiiro2                  ;
kiiro1:  stosw                        ; Karakter és színattribútum
         lodsb                        ; ciklusban a képernyőre, mindaddig
         or   al,al                   ; amíg a lezáró '0' sorra nem kerül.
         jne  kiiro1                  ;
                                      ;
kiiro2:  pop  dx                      ; Regiszterek visszatoltése a
         pop  es                      ; veremről.
         pop  si                      ;
         ret                          ; Vissza a hívóhoz.

kiiro    endp

;— CIMSZAM : sor és oszlop értékből cím képzése ——————————————————
;— BE : BL = sor
```

```
;         BH = oszlop
;         BP = az oldal sorszáma
;-- KI : DI = a kiszámított cím
;-- Regiszterek : DI és FLAGS megváltoznak

cimszam  proc near

         push ax                    ; Regiszterek elmentése a veremre.
         push bx                    ;
                                    ;
         shl  bx,1                  ; Sorok és oszlopok 2-vel szorzandók
                                    ; (minden második pozíción van kar.).
         mov  al,bh                 ; OszlopKoordináta -> AL.
         mov  bh,0                  ; A sorKoordináta -> DI, az adat-
         mov  di,[sorcim+bx]        ; táblázat alapján.
         mov  ah,0                  ; Az oszlopKoordináta hozzáadódik
         add  di,ax                 ; DI értékéhez.
         mov  bx,bp                 ;
         mov  cl,4                  ; Az oldal sorszáma * 4096 is hozzá-
         ror  bx,cl                 ; adódik a DI-ben számított
         add  di,bx                 ; ofszetcímhez.
                                    ;
         pop  bx                    ; Regiszterek visszatöltése
         pop  ax                    ; a veremről.
         ret                        ; Vissza a hívóhoz.

cimszam  endp

;-- CLSG : a grafikus képernyő törlése ------------------------------
;-- BE : AL = 00h : minden pont törlése
;             FFh : minden pont kirása
;-- KI : -
;-- Regiszterek   AH, BX, CX, DI és FLAGS megváltoznak

clsg     proc near

         push es                    ; ES -> verem.
         cbw                        ; AL kiterjesztése AH-ra
                                    ;
         mov  di,0                  ; Video RAM ofszetcím -> DI.
         mov  bx,0B800h             ; Video RAM szegmenscím -> ES.
         mov  es,bx                 ;
         mov  cx,2000h              ; 16 kbájtot kell kiírni -> CX.
         rep  stosw                 ; Az oldal feltöltése ciklusban
                                    ;
         pop  es                    ; Verem -> ES.
         ret                        ; Vissza a hívóhoz.

clsg     endp

;-- PONTIR : egy pont kiírása a 640*200 pontos grafikus képernyőre --
;
;            A bájt relatív címének kiszámítása és a bájton
;            belül a bit behatárolása a fejezetben
;            ismertetett képleteknek megfelelően történik.
;
;-- BE : BP = a képernyőoldal sorszáma (0 vagy 1)
;        AL = a pont színe (0 vagy 1)
;        BX = oszlop (0-639)
;        DX = sor    (0-199)
;-- KI : -
;-- Regiszterek : AX, DI és FLAGS megváltoznak
```

```
pontir  proc near

        push ax                 ; Regiszterek elmentése a veremre.
        push bx                 ;
        push cx                 ;
        mov  ah,bl              ; Oszlopkoordináta alsó bájt -> AH.
        and  ah,111b            ; Oszlop mod 8 -> AH.
        mov  cl,7               ;
        sub  cl,ah              ; 7 - oszlop mod 8 -> CL.
        mov  ah,1               ;
        shl  ax,cl              ; A kiszámított bitnek megfelelő
        not  ah                 ; (negált) maszk előállítása AH-ban.
        mov  cl,3               ;
        shr  bx,cl              ; INT(oszlopKoordináta/8) -> BX.

        ; Az oszlopérték már feldolgozva, a sorérték még nem.

pontir  endp

;— PONTKI : egy pontot tesz ki a grafikus képernyőre ————————————————
;— BE : BX = oszlopofszet előzőleg beállított értéke
;       DX = sor (0-199)
;       AH = maszkoló bájt, mindenhol 1, kivéve az aktuális bitnél
;       AL = az aktuális bit értéke
;— KI : -
;— Regiszterek : AX, DI és FLAGS megváltoznak

pontki  proc near

        push es                 ; Regiszterek elmentése a veremre.
        push dx                 ;
        push ax                 ;
        mov  di,0               ; Videó RAM ofszetcím -> DI.
        mov  cx,0B800h          ; Videó RAM szegmenscím -> ES.
        mov  es,cx              ;
        mov  ax,dx              ; Sorkoordináta -> AX.
        shr  ax,1               ; INT(sorkoordináta/2) -> AX.
        mov  cl,80              ;
        mul  cl                 ; AX * 80 -> AX.
        and  dx,1               ; Sorkoordináta mod 2 -> DX.
        mov  cl,3               ;
        ror  dx,cl              ; DX * 8192 -> DX (forgatással).
        mov  di,ax              ;
        add  di,dx              ; DI-ben a relatív cím összeállítása
        add  di,bx              ; DX, BX és AX tartalmából.
        pop  ax                 ;
        mov  bl,es:[di]         ; Egy bájt beolvasása a videoRAM-ból.
        and  bl,ah              ; Az aktuális bit (pont) törlése,
        or   bl,al              ; majd beállítása.
        mov  es:[di],bl         ; A képernyőpont(ok) megjelenítése.
                                ;
        pop  dx                 ; A regiszterek visszatoltése.
        pop  es                 ;
        pop  cx                 ;
        pop  bx                 ;
        pop  ax                 ;
        ret                     ; Vissza a hívóhoz.
```

```
pontki   endp

;— ADATOK ————————————————————————————————————————

sorcim   dw     0, 160, 320, 480, 640        ; A 25 képernyősor oldalon
         dw   800, 960,1120,1280,1440        ; belüli ofszetcíme a
         dw  1600,1760,1920,2080,2240        ; video RAM-ban.
         dw  2400,2560,2720,2880,3040
         dw  3200,3360,3520,3680,3840

graf     db 38h, 28h, 2Dh, 0Ah, 7Fh, 06h  ; A grafika bekapcsolásának
         db 64h, 70h, 02h, 01h, 06h, 07h  ; regiszterértékei.

szov     db 71h, 50h, 5Ah, 0Ah, 1Fh, 06h  ; A grafika kikapcsolásának
         db 19h, 1Ch, 02h, 07h, 06h, 07h  ; regiszterértékei.

;— VÉGE ——————————————————————————————————————————

kod      ends                             ; A (kód)szegmens vége.
         end  start                       ; Az assembler forrásprogram vége.
                                          ; A program végrehajtása a start
                                          ; címkén kezdődik.
```

2.5. A hangszóró programozása

Minden PC-ben megtalálható egy hangszóró, amelynek az a feladata, hogy bizonyos események bekövetkezésekor egy – beállítható frekvenciájú és időtartamú – hangot szólaltasson meg. A hangszórót is kezelő felhasználói programokban a hangot kiváltó események általában valamilyen, a program meg nem engedett használatával kapcsolatosak, vagy arra hívják fel a figyelmet, hogy az elvégzendő művelet a szokásosnál is nagyobb óvatosságot igényel (pl. egy nem létező opció kiválasztása, vagy egy lemez FAT-táblájának felülírása). Maga a ROM-BIOS is kiad hangjelzést pl. akkor, ha megtelt a billentyűzetpuffere.

A hangszórót sem DOS, sem BIOS szintről nem lehet megszólaltatni (ugyan a ROM-BIOS tartalmaz ilyen rutinokat, de ezekhez megszakításokon keresztül nem lehet hozzáférni). Így hangot csak a hardver közvetlen programozásával lehet kelteni. A következőkben erről lesz szó.

Magáról a hangképzésről itt csak annyit mondunk el, hogy a hangot a hangszóró membránjának periodikus mozgása hozza létre. A mozgás sebessége a hang frekvenciáját, vagyis a hang magasságát, a mozgás időtartama pedig a hang időtartamát határozza meg. Eszerint a számítógép hangszórójával úgy tudunk egy hangot előállítani, ha a membránját valamilyen frekvenciával valamennyi ideig mozgatjuk. Nézzük meg, hogy milyen lehetőségeink vannak erre.

A számítógép hangszórója közvetve kapcsolódik a 8255 jelű, programozható periféria illesztő (PPI) nevű IC-re. A hangszóró a periféria illesztő 61(h) című portján keresztül érhető el. Ha ezen a porton keresztül a hangszóró áramát megfelelő sebességgel be- és kikapcsoljuk, akkor a hangszóró a sebességtől függő frekvencián megszólal. Ez az el-

járás azonban igazából két dolog miatt sem jó megoldás. Az egyik probléma az, hogy az utasítások végrehajtási sebessége a számítógép órajelétől függ, és ezért a különböző órajelű gépeken a megszólaló hang frekvenciája, azaz a hangmagasság más és más lesz. A másik probléma az, hogy ha a hangszórót megszólaltató ciklus végrehajtása közben valamilyen megszakítás érkezik (pl. a billentyűzetről), akkor ez a ciklus végrehajtási időrendjét „felborítja", és a hang „fals" lesz. A hangkeltésre ezért más módszert kell választani. Mivel az előzőekből látszik, hogy a probléma a megfelelő időzítésekkel függ össze, a megoldást is ebben az irányban kell keresni.

Ismeretes, hogy minden PC-ben van egy programozható időzítő/számláló IC (ismert angol nevén Timer/Counter, röviden T/C), amely a gép hardver órajeléből egy belső, szoftver órajelet, angol nevén a timer tick jelet állítja elő (az XT-ben 8253-as jelű, az AT-ben 8254-as jelű T/C áramkör van). A T/C legfontosabb tulajdonsága az, hogy a gép órajelétől független, állandó frekvenciájú impulzussorozatokat képes előállítani. A T/C három csatornát tartalmaz, amelyek mindegyike egy-egy, egymástól függetlenül programozható számlálónak tekinthető. Az egyes számlálók számlálási sebessége (órajele) egymástól függetlenül határozható meg. Az előállított impulzussorozat frekvenciája a számláló programból meghatározható kezdőértékétől függ. A T/C számlálói a beállított érték (időtartam) elérésekor egy jelet adnak ki, és elölről kezdik a számlálást. Az áramkör azért használható a hangszóró vezérlésére, mert a 2. számú (harmadik) csatornája adja a hangszóróra jutó jelet.

Az idő mérésének alapjául a 8284-es jelű óragenerátor szolgál. Ez az áramkör állítja elő – többek között – a rendszer belső, 1.193180 MHz-es órajelét, amely közvetlenül kapcsolódik a T/C áramkör bemenetére. Ez számunkra azért fontos, mert így megadhatjuk, hogy a T/C a harmadik, a hangszóróhoz kapcsolt csatornáján a bemenetére érkező hányadik órajel után adjon ki egy impulzust. A bemenő, 1.193180 MHz-es impulzussorozat íly módon való leosztásával a T/C harmadik csatornáján beállíthatjuk a hangszóróra kerülő impulzussorozat frekvenciáját, vagyis a hang magasságát. A hang magasságának beprogramozásához mindössze annyit kell tennünk, hogy a T/C harmadik csatornájába be kell írnunk azt a számlálási értéket, amely megfelel a kívánt hang frekvenciájának.

A számlálási érték kiszámítása igen egyszerű:

$$\text{számlálási érték} = \frac{1\,193\,180}{\text{frekvencia}}$$

A hangok frekvenciatáblázatát a III. kötet tartalmazza.

A C-dúr skála hangjait, frekvenciájukat és a fenti képlettel számított számlálási értékeket a 32. ábrán mutatjuk be.

Hang	Frekvencia (Hz)	Számlálási érték	
		decimális	hexadecimális
c	261.63	4560	11D0(h)
cisz	277.18	4384	1120(h)
d	293.66	4063	FDF(h)
disz	311.13	3834	EFA(h)
e	329.63	3619	E23(h)
f	349.23	3416	D58(h)
fisz	369.99	3224	C98(h)
g	392.00	3043	BE3(h)
gisz	415.30	2873	B39(h)
a	440.00	2711	A97(h)
b	466.16	2559	9FF(h)
h	493.88	2415	96F(h)
c	523.25	2280	8E8(h)

32. ábra: A C-dúr skála jellemző értékei

A T/C programozása szintén portokon keresztül történik. A hangszóró vezérléshez a T/C négy portja közül a 43(h) címen lévő vezérlő portot és a 42(l_4) címen lévő számláló/időzítő portot kell beállítani (ez utóbbi felel meg a T/C harmadik, a hangszóróval összekapcsolt csatornájának).

A hangmagasság beállításához először a vezérlő portra ki kell írni azt az értéket, amely kiválasztja a programozni kívánt csatornát és az üzemmódot. Esetünkben ez az érték 182. Ezt követően kell a 42(h) című portra azt a számlálási értéket kiírni, amely a megszólaltatandó hang frekvenciájának felel meg. A T/C ezt az értéket egy 16 bites számként kezeli, ezért a számláló értékének 0 és 65535 között kell lennie. (Ebből következik, hogy a megszólaltatható hang frekvenciájának alsó határa 18.2 Hz – ami nem meglepő –, felső határa pedig 1.193180 MHz. Más dolog, hogy az emberi fül által hallható hangok tartománya a 20 Hz-20 kHz tartományon belül van.) Mivel a 42(h) port 8 bites, a 16 bites értéket csak két lépésben lehet kiírni: először a szó alsó, majd a felső bájtját kell átadni. Ezzel a lépéssel a hang magasságának beállítása megtörtént. Ahhoz azonban, hogy a hangszóró meg is szólaljon, a hangszórót be kell kapcsolni.

A hangszóró bekapcsolásáért a 8255-ös jelű, programozható periféria illesztő IC (angol nevén: Programmable Peripheral Interface, röviden PPI) felel. A hangszóró bekapcsolásához a PPI 61(h) portcímén lévő 8 bites regiszter 0. és 1. bitjét 1 értékre kell állítani. Mivel a PPI a regiszter többi bitjét más célokra használja, ügyelni kell arra, hogy valóban csak az alsó két bitet változtassuk meg. Ezért először be kell olvasni a port értékét, az alsó két bitet be kell kapcsolni, és az így kapott értéket vissza kell írni a portra.

Ekkor megszólal a hang, és mindaddig szól, míg a bekapcsolást végző két bitet nem töröljük.

A hang meghatározott hosszúságának beállítására ugyancsak a T/C időzítő képességét használhatjuk ki. Említettük, hogy a T/C első, 0. sorszámú csatornája szolgáltatja a rendszer belső, szoftver órajelét, amely jó közelítéssel másodpercenként 18.2 impulzussal „ketyeg". A korábbiakban arról is volt szó, hogy minden impulzus kiváltja a 8-as számú, timerként ismert hardver megszakítást. Ezt a megszakítást a BIOS – sok más mellett – arra is kihasználja, hogy meghívja az 1C(h) megszakítást, amely alapállapotában egy IRET-tel azonnal visszatér, lehetővé téve, hogy a felhasználói programok időmérési/számlálási feladatokra felhasználhassák.

A hang hosszúságának beállításához tehát azt kell tennünk, hogy az 1C(h) megszakítási vektor címét átállítjuk egy általunk írt rutinra, amelynek csak az a feladata, hogy a hang megkívánt hosszúságának megfelelő értéket a megszakítás minden egyes meghívásakor, azaz másodpercenként 18.2-szer 1-gyel csökkentse. Amikor a beállított érték – célszerűen – 0-ra csökken, a hangot ki kell kapcsolni, és az „eltérített" vektorcímet természetesen helyre kell állítani. Az itt leírtak megértését ugyancsak egy példaprogramon keresztül segítjük elő.

2.5.1. Programpéldák

A hangkeltést egy assembly és egy C nyelvű programon keresztül mutatjuk be. A megszakításvektor eltérítését, ill. a portok (43(h), 61(h)) korábbiakban ismertetett kezelését a C program esetén is egy hozzászerkesztett assembly rutin biztosítja, így tulajdonképpen magára a C programra csak a „zeneszerzés" magasabbszintű vezérlése hárul.

A hangszóró programozásával kapcsolatban ismertett elvek a példaprogramok magyarázatain keresztül jól követhetők. Kiegészítésképpen arra a programtechnikai fogásra szeretnénk felhívni a figyelmet, hogy az igen áttekinthetetlen frekvencia, ill. azzal egyenértékű számlálási értékek használata helyett célszerű a C programban látható módon tömbökbe foglalni a megszólaltatandó hangok paramétereit (tartam, magasság).

```
;
;   ╔═══════════════════════════════════════════════════════════╗
;   ║                      D A L L A M . A S M                  ║
;   ║                                                           ║
;   ║  Funkciója : a beépített hangszóró tesztelése egy rövid dallam ║
;   ║              megszólaltatásával.                          ║
;   ╚═══════════════════════════════════════════════════════════╝
;
;— KÓD ————————————————————————————————————————————————————————

Kod     segment para 'CODE'     ; A kódszegmens definiálása.

        org 100h                ; A program a PSP után álló első
                                ; címen kezdődik.

        assume cs:Kod, ds:Kod, es:Kod, ss:Kod

                                ; Csak egy, közös szegmens van.

;— Program ————————————————————————————————————————————————————

dallam  proc near

        ;— Induló üzenet kiírása ——————————————————————————————

        mov  dx,offset txt_1    ; Üzenet kiírása.
        mov  ah,9               ; Karakterlánc kiírás funkció.
        int  21h                ; 21(h) DOS megszakítás.

        ;— Dallam lejátszása ——————————————————————————————————

        mov  bl,0               ; Az első hangra pozícionálás.
Kovhang: call hangadas          ; Hang megszólaltatása.
        inc  bl                 ; Összesen 21 hangot kell lejátszani.
        cmp  bl,21              ; Összes hang lejátszva ?
        jne  Kovhang            ; Nem : tovább.

        ;— Befejező üzenet kiírása ———————————————————————————

        mov  dx,offset txt_2    ; Üzenet kiírása.
        mov  ah,9               ; Karakterlánc kiírás funkció.
        int  21h                ; 21(h) DOS megszakítás.

        mov  ax,4C00h           ; Vissza a DOS-hoz 0-ás
        int  21h                ; hibakóddal.

dallam  endp

;— Adatok ————————————————————————————————————————————————————

CR      equ 13                  ; Kocsivissza ASCII kódja.
LF      equ 10                  ; Soremelés ASCII kódja.
KEND    equ "$"                 ; Karakterlánc vége jelzés.

txt_1   db CR,LF,"Amennyiben a PC hangszórója működőképes, ",CR,LF
        db "ön most egy rövid dallamot hall.",CR,LF,KEND

txt_2   db CR,LF,"A tesztelés befejeződött.",CR,LF,KEND
```

```
;— HANGADAS : egy hang megszólaltatása ————————————————————————
;— BE : BL = a hang sorszáma
;—      DL = a hangadás időtartama 1/18 mp -ben
;— KI : -
;— Regiszterek : AX, CX, ES és FLAGS változnak

hangadas proc near

        push dx                     ; DX --> verem.
        push bx                     ; BX --> verem.

        ;— A periódikus megszakítás átirányítása a saját rutinra ——

        push dx                     ; DX --> verem.
        push bx                     ; BX --> verem.
        mov  ax,351Ch               ; Megszakítás cím lekérdezés funkció.
        int  21h                    ; 21(h) DOS megszakítás.
                                    ; A megsz. rutin címe ES:BX-be kerul.
        mov  origmsz,bx             ; Az eredeti msz. címének eltárolása.
        mov  origmsz+2,es

        mov  dx,offset ujmsz        ; Az új periodikus megszakítás címe.
        mov  ax,251Ch               ; Megszakítás vektor beáll. funkció.
        int  21h                    ; DOS megszakítás hívása.
        pop  bx                     ; Verem --> BX.
        pop  dx                     ; Verem --> DX.

        mov  al,182                 ; Hangmegszólaltatás előkészítése, az
        out  43h,al                 ; érték az időzítő regiszterbe kerul.

        mov  bh,0                   ; BH címezi a hangtáblázatot.
        mov  dl,[tartam+bx]         ; Időtartam eltárolása.
        shl  bx,1                   ; A hangtáblázat szavakat tartalmaz.
        mov  ax,[hangok+bx]         ; A kiválasztott hang --> AX.
        out  42h,al                 ; Az alsó és felső hangérték bájt
        mov  al,ah                  ; kikuldése az időzítő regiszterbe.
        out  42h,al
        in   al,61h                 ; A hangszóró vezérlőbájt beolvasása,
        or   al,11b                 ; a bekapcsoló alsó 2 bit beállítása.
        mov  hangveg,1              ; Nincs még vége.
        mov  szaml,dl               ; Időtartam eltárolása.
        out  61h,al                 ; A hangszóró bekapcsolása.

szol:   cmp  hangveg,0              ; Ha a hang még nem hangzott el,
        jne  szol                   ; tovább kell játszani.

        in   al,61h                 ; A hangszóró vezérlőbájt beolvasása,
        and  al,11111100b           ; a bekapcsoló alsó 2 bit törlése.
        out  61h,al                 ; A hangszóró kikapcsolása.

        ;— Az eredeti periódikus megszakítás visszaállítása ————————

        mov cx,ds                   ; CX = DS
        mov ax,251Ch                ; Megszakítás vektor beáll. funkció.

        lds dx,dword ptr origmsz    ; Az eredeti megszakítás címe.

        int  21h                    ; 21(h) DOS megszakítás.
        mov ds,cx                   ; CX --> DS.

        pop  bx                     ; Verem --> BX.
        pop  dx                     ; Verem --> DX.
        ret                         ; Vissza a hívóhoz.
```

```
hangadas endp

;— Az új periódikus megszakítás ————————————————————————

ujmsz   proc far                ; Másodpercenként 18-szor hívódik.

        dec  cs:szamlal         ; Időszámláló csökkentése.
        jne  vege               ; Ha a számláló eléri a 0-t, akkor
        mov  cs:hangveg,0       ; jelezni kell, hogy hang elhangzott.

vege:   jmp  dword ptr cs:[origmsz] ; Ugrás az eredeti megszakításra.

ujmsz   endp

;— Adatok ———————————————————————————————————————————————

origmsz dw (?),(?)              ; A régi periódikus megsz. címe.
szamlal db (?)                  ; Egy hang játszási ideje (1/18 mp).

hangveg db (?)                  ; Befejezést jelző változó.

hangok  dw 5423, 10, 5423, 10, 5423, 10
        dw 5423, 10, 4560, 10, 4831, 10
        dw 4831, 10, 5423, 10, 5423, 10
        dw 5746, 10, 5423

tartam  db 18, 2, 18, 2,  3, 2
        db 18, 2, 18, 2,  3, 2
        db 18, 2,  3, 2, 18, 2
        db  3, 2, 18

;— VÉGE ————————————————————————————————————————————————

kod     ends                    ; A (kód)szegmens vége.
        end  dallam             ; Az assembler forrásprogram vége.
                                ; A program végrehajtása a dallam
                                ; címkén kezdődik.
```

```
/*
/*
/*                                                                    */
/*                          D A L L A M . C                           */
/*                                                                    */
/*  Funkciója : a beépített hangszóró tesztelése egy rövid            */
/*              dallam megszólaltatásával.                            */
/*                                                                    */
/*  FONTOS    : a "hangadó" rutin a DRUTIN.ASM forrásállományban       */
/*              helyezkedik el, ezért a LINK parancs helyes           */
/*              formája a következő :                                 */
/*                                                                    */
/*                LINK DALLAM DRUTIN                                  */
/*                                                                    */
/*                                                                    */

extern void HANGADAS();          /* Az assembler rutin beszerkesztendő. */

/*
/*                        F Ő P R O G R A M                           */
/*                                                                    */

int  hangok[] = {  9,999,  9,999,  9,999,  9,999, 12,999, 11,999,
                  11,999,  9,999,  9,999,  8,999,  9,  0
                };

int  tartam[] = { 18,   2, 18,   2,  3,   2, 18,   2, 18,   2,  3,   2,
                  18,   2,  3,   2, 18,   2,  3,   2, 18
                };

void main()

{
int i;

printf("");
printf("A PC hangszórójának tesztelését egy tetszőleges billentyű");
printf("megnyomásával indíthatja el. Amennyiben a hangszóró működ-");
printf("őképes, egy rövid dallamot hall.");
printf("Indítsa el a tesztet ..");

getch();                                 /* Várakozás egy billentyűre. */

i = -1;
while (hangok[++i])
   HANGADAS(hangok[i], tartam[i]);

printf("A tesztelés befejeződött.");
}
```

```
;
;   ╔═══════════════════════════════════════════════════════════╗
;   ║                                                           ║
;   ║                       D R U T I N . A S M                 ║
;   ║                                                           ║
;   ║  Funkciója : az MSC mintaprogram számára a HANGADAS eljárás ║
;   ║              definiálása, mely a 2 - 5 oktávokba esõ hangok ║
;   ║              kiadására szolgál.                           ║
;   ╚═══════════════════════════════════════════════════════════╝
;

;— DEFINíCIÓK ——————————————————————————————————————————————

IGROUP  group _TEXT                     ; Programszegmens.
DGROUP  group CONST, _BSS, _DATA        ; Adatszegmens.

assume  CS:IGROUP, DS:DGROUP, ES:DGROUP, SS:DGROUP

        public _HANGADAS                ; HANGADAS kívülrõl is hívható.

CONST   segment word public 'CONST'     ; Csak olvasható konstansok.
CONST   ends

_BSS    segment word public 'BSS'       ; Nem inicializált statikus
_BSS    ends                            ; változók szegmense.

_DATA   segment word public 'DATA'      ; Inicializált globális és
                                        ; statikus változók szegmense.

origmsz dw (?),(?)                      ; Az eredeti periódikus
                                        ; megszakítás címe.
tartam  db (?)                          ; Idõegység (1/18 mp).

hangveg db (?)                          ; Befejezést jelzõ változó.

hangok  dw 18242, 17218, 16252          ; A 2. oktáv adatai.
        dw 15340, 14478, 13666
        dw 12898, 12174, 11492
        dw 10846, 10238,  9662
        dw  9121,  8609,  8126          ; A 3. oktáv adatai.
        dw  7670,  7239,  6833
        dw  6449,  6087,  5746
        dw  5423,  5119,  4831
        dw  4560,  4304,  4063          ; A 4. oktáv adatai.
        dw  3834,  3619,  3416
        dw  3224,  3043,  2873
        dw  2711,  2559,  2415
        dw  2280,  2152,  2031          ; Az 5. oktáv adatai.
        dw  1917,  1809,  1715
        dw  1612,  1521,  1436
        dw  1355,  1292,  1207

_DATA   ends

;— Program ————————————————————————————————————————————————

_TEXT   segment byte public 'CODE'      ; Program (kód) szegmens.

;— HANGADAS : Egy hang megszólaltatása ————————————————————————
;— Hívás MSC-bõl : HANGADAS(int hang, int tartam)
```

```
_HANGADAS    proc near

             push bp                    ; BP --> verem.
             mov  bp, sp                ; SP --> BP.

             ;— A periódikus megszakítás átirányítása a saját rutinra ——

             mov  word ptr cs:setds+1, ds   ; DS az új megszakításhoz.

             mov  ax, 351Ch             ; Megszakítás cím lekérdezés funkció.
             int  21h                   ; 21(h) DOS megszakítás.
             mov  origmsz, bx           ; Eredeti megszakítás címének
             mov  origmsz+2, es         ; eltárolása ES : BX.

             mov  word ptr cs:stjump+1, bx
             mov  word ptr cs:stjump+3, es

             mov  bx, ds                ; DS --> BX.
             push cs                    ; CS --> DS.
             pop  ds
             mov  dx, offset ujmsz      ; Az új periódikus megszakítás címe.
             mov  ax, 251Ch             ; Megszakítás vektor beáll. funkció.
             int  21h                   ; DOS megszakítás hívása.
             mov  ds, bx                ; BX --> DS.

             mov  al, 182               ; Hangmegszólaltatás előkészítése, az
             out  43h, al               ; érték az időzítő regiszterbe kerül.

             mov  hangveg, 1            ; Nincs még vége.
             mov  dl, [bp+6]            ; Hangadás időtartama a veremről
             mov  tartam, dl            ; a "tartam" változóba kerül.
             mov  bx, [bp+4]            ; A hang paramétere a veremről.
             mov  bh, 0                 ; BH címezi a hangtáblázatot,
             shl  bx, 1                 ; a táblázat szavakat tartalmaz.
             mov  ax, [hangok+bx]       ; A kiválasztott hang --> AX.
             out  42h, al               ; Az alsó és felső hangérték bájt
             mov  al, ah                ; kiküldése az időzítő regiszterbe.
             out  42h, al
             in   al, 61h               ; A hangszóró vezérlőbájt beolvasása,
             or   al, 11b               ; a bekapcsoló alsó 2 bit beállítása.
             out  61h, al               ; A hangszóró bekapcsolása.

szol:        cmp  hangveg, 0            ; Ha a hang még nem hangzott el,
             jne  szol                  ; tovább kell játszani.

             in   al, 61h               ; A hangszóró vezérlőbájt beolvasása,
             and  al, 11111100b         ; a bekapcsoló alsó 2 bit törlése.
             out  61h, al               ; A hangszóró kikapcsolása.

             ;— Az eredeti periódikus megszakítás visszaállítása ————————

             mov cx, ds                 ; CX = DS
             mov ax, 251Ch              ; Megszakítás vektor beáll. funkció.

             lds dx, dword ptr origmsz      ;Az eredeti megszakítás címe.

             int  21h                   ; 21(h) DOS megszakítás.
             mov ds, cx                 ; CX --> DS.

             mov  sp, bp                ; A veremmutató visszaállítása.
             pop  bp                    ; Verem --> BP.
             ret                        ; Vissza a hívóhoz.
```

```
_HANGADAS   endp
;— Az új periódikus megszakítás ————————————————————————————

ujmsz    proc far                  ; Másodpercenként 18-szor hívódik.

         push ax                   ; AX --> verem.
         push ds                   ; DS --> verem.
setds:   mov  ax,0000h             ; Ide kerül a C nyelv DS-e.
         mov  ds,ax
         dec  tartam               ; Időszámláló csökkentése.
         jne  vege                 ; Ha a számláló eléri a 0-t, akkor
         mov  hangveg,0            ; jelezni kell, hogy hang elhangzott.
vege:    pop  ds                   ; Verem --> DS.
         pop  ax                   ; Verem --> AX.

stjump:  db   0EAh,0,0,0,0         ; Ugrás az eredeti megszakításra.

ujmsz    endp

;— VÉGE ————————————————————————————————————————————————————

_TEXT    ends                      ; Programszegmens vége.
         end                       ; Assembler forráskód vége.
```

2.6. A valós idejű óra programozása és az RTC/CMOS RAM

A kazettamegszakítások ismertetésénél, valamint a dátumot és órát kezelő BIOS megszakításoknál (lásd 1.7. és 1.10. alfejezeteket) már szóltunk arról, hogy az AT gépek alapkiépítettségükben tartalmaznak egy Motorola gyártmányú, 146818-as típusjelű, RTC/CMOS-nak nevezett integrált áramkört (az RTC a Real-Time Clock = valós idejű óra, a CMOS a Complementary Metal Oxide Semiconductor = komplementer fém-oxid félvezető rövidítése), amelynek egy, a gép operatív tárától független, saját 64 bájtos RAM tára van. Ez az áramkör a működtető feszültségét külön telepről kapja, ami lehetővé teszi, hogy a RAM tartalma a gép kikapcsolásakor vagy áramszünet esetén se vesszen el. Maga a RAM egyrészt a számítógép konfigurációjáról tartalmaz információkat, másrészt itt tárolják a dátumot és a „pontos" időt. (Ezeket az adatokat a BIOS a gép bekapcsolásakor innen olvassa ki, ezért nincs szükség arra, hogy – miként az XT-nél – minden rendszerindításkor külön meg kelljen adni.)

Nézzük meg ennek a 64 bájtos RAM-nak a felépítését (33. ábra).

Cím	Jelentés
00(h) – 0D(h)	valós idejű óra információi
0E(h)	diagnosztika bájt
0F(h)	védett üzemmódból való visszatéréskor a rendszer által beállított bájt (shutdown status)
10(h)	hajlékonylemezes meghajtók típusát leíró bájt (A és B meghajtó)
11(h)	fenntartott
12(h)	merevlemezes meghajtók típusát leíró bájt (C és D meghajtó)
13(h)	fenntartott
14(h)	berendezés bájt
15(h)	alaplemezen lévő RAM kbájtokban számított kapacitásának alsó bájtja
16(h)	alaplemezen lévő RAM kbájtokban számított kapacitásának felső bájtja
17(h)	tárbővítő RAM kbájtokban számított kapacitásának alsó bájtja
18(h)	tárbővítő RAM kbájtokban számított kapacitásának felső bájtja
19(h) – 2D(h)	fenntartott
2E(h) – 2F(h)	a CMOS két bájtos vizsgálóösszege
30(h)	tárbővítő RAM kbájtokban számított kapacitásának alsó bájtja
31(h)	tárbővítő RAM kbájtokban számított kapacitásának felső bájtja
32(h)	évszázad első két számjegye BCD alakban
33(h)	a rendszer indításakor használt információk
34(h) – 3F(h)	fenntartott bájtok

33. ábra: Az RTC/CMOS RAM felépítése

Az ábrából látható, hogy a valós idejű óra a RAM 00(h) – 0D(h) közötti 14 bájtot foglalja el, míg a többi címeken jórészt a rendszer konfigurációjával kapcsolatos adatok állnak. (Megjegyezzük, hogy a 00(h) – 0F(h) és a 30(h) – 3F(h) címeken lévő bájtok nem részei a CMOS RAM ún. konfigurációs rekordjának, és a vizsgálóösszeg számítása sem terjed ki rájuk.)

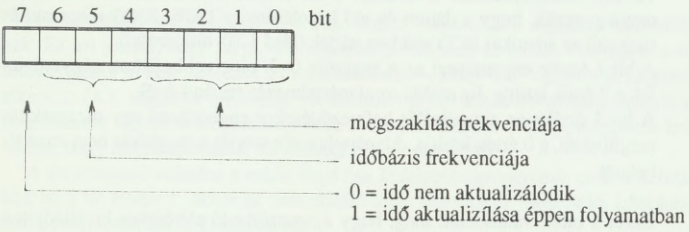
Nézzük először a valós idejű óra bájtjait (34. ábra).

Cím	Tartalom
00(h)	aktuális másodperc
01(h)	riasztás másodperce
02(h)	aktuális perc
03(h)	riasztás perce
04(h)	aktuális óra
05(h)	riasztás órája
06(h)	a hét napja
07(h)	a hónap napja
08(h)	hónap
09(h)	év
0A(h)	az óra A állapotregisztere
0B(h)	az óra B állapotregisztere
0C(h)	az óra C állapotregisztere
0D(h)	az óra D állapotregisztere

34. ábra: A valós idejű óra információi

Látható, hogy a másodperc, perc és az óra tárolására két-két cím áll rendelkezésre. Ezek közül az elsők a valós időpontot, míg az utánuk következők az esetleges riasztási időpontot tárolják. Megfelelő beállítással a riasztási időpont elérésekor kiváltódik egy megszakítás. Erre később még visszatérünk.

A „hét napja" bájt a héten belüli nap számát tartalmazza: az 1 a vasárnapot, a 2 a hétfőt stb. jelenti. Az év számát az 1980. évhez viszonyítva kell megadni. Eszerint az 1989. évet a 89 jelenti. Az óra programozásában fontos szerepet játszanak az A-D jelű állapotregiszterek. Az A regiszter felépítését a 35. ábrán szemléltetjük.



35. ábra: Az óra A állapotregisztere

A 0–3. és a 4–6. bitek értékét a rendszer inicializálásakor a ROM-BIOS állítja be, és ezek később nem is változnak meg. A megszakítás frekvenciájának 4 bitje a bináris 0110 értéket kapja. Ez az érték egy osztót jelent, ami a megszakítás frekvenciáját 1.024 kHz-re állítja be (minden 976.562 milliomod másodpercenként 1 megszakítás). Az időbázis

frekvenciájának 3 bitje a bináris 010 értéket kapja. Ez szintén egy osztó, amely az időbázis frekvenciáját 32.768 kHz-re választja meg.

Az óra programozásában csak a 7. bit jut szerephez. Ennek a bitnek az értéke jelzi, hogy a valósidejű óra aktualizálása éppen folyamatban van (a bit 1 értéke), vagy nem (a bit 0 értéke). Az órát egy programnak csak akkor szabad olvasnia, amikor éppen nem „ugrik a mutatója". Ellenkező esetben előfordulhat, hogy az órát éppen akkor olvassuk, amikor a másodpercmutató egy perc elteltét követően éppen 0-ra ugrik, de a percmutató még nem lépett előre. Az óra folyamatos kijelzése esetén ekkor előadódhat, hogy pl. az óra a 17.59.59 időről a 17.59.00 ugrik (a 18.00.00 helyett), és csak a következő másodpercben jelzi ki a már helyes, 18.00.01 időt.

Nézzük meg most a B állapotregisztert, egyes bitjeinek jelentését.

0. bit: Jelzi, hogy a valós idejű óra a „normál" vagy a nyári időszámítás szerint számlálja az idő múlását. A bit 1 értéke jelenti a nyári időszámítást: ekkor a gép az időt április utolsó vasárnapján hajnali 1 óra 59 perc 59 másodperc után 3 óra 0 perc 0 másodpercre előre állítja, október utolsó vasárnapján pedig 1 óra 59 perc 59 másodpercről 1 óra 0 perc 0 másodpercre visszaállítja. Ha a bit értéke 0, akkor ezek az átállítások elmaradnak, és az óra a „normál" – téli – időszámítás szerint méri az időt. A bit értéke alapértelmezésben 0.

1. bit: Az időmérés 12/24 órás módját jelzi. Ha a bit értéke 0, akkor déli 12 óra után 1 óra következik, ha 1, akkor 13 óra. A rendszer a bit értékét induláskor 1-re állítja.

2. bit: A dátummezőkben lévő idő ábrázolási módját mutatja. Ha a bit értéke 1, akkor a bájtok az adatokat a szokásos bináris módon tartalmazzák. Az 1989. évet ekkor a bináris 01011001 jelenti (csak a 89-et kell ábrázolni!). Ha a bit értéke 0, akkor az adatok ábrázolása BCD (binárisan kódolt decimális) alakban történik. Ekkor az 1989. évet a bináris 10001001 jelenti (8 = 1000b, 9 = 1001b). Bekapcsoláskor a rendszer a bit értékét 0-ra állítja. Emlékeztetőül megjegyezzük, hogy a dátum és idő kezelésére az 1A(h) BIOS megszakítás funkciói az adatokat BCD alakban várják (lásd 1.10. alfejezetet).

3. bit: A bit 1 értéke engedélyezi az A regiszter 0–3. bitjeivel beállított négyszögjelet, a 0 érték letiltja. Ez utóbbi az alapértelmezés szerinti érték.

4. bit: A bit 1 értéke az aktualizálás befejeződésekor engedélyezi egy megszakítás meghívását, a 0 érték letiltja. Alapértelmezés szerint a meghívás nem engedélyezett.

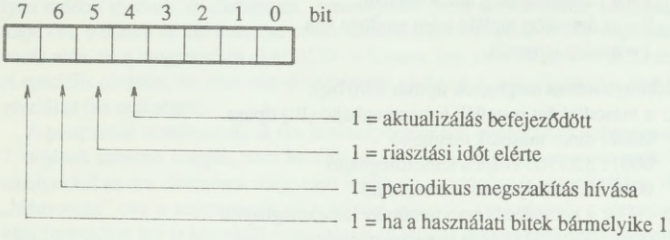
5. bit: Ezzel a bittel választható meg, hogy a riasztási idő elérésekor kiváltódjon-e egy megszakítás. A riasztási időt ugyancsak a valós idejű óra tartalmazza. A bit 1 értéke engedélyezi a megszakítást. A rendszer indításakor a bit értéke 0, tehát a megszakítás nem engedélyezett.

6. bit: A bit 1 értéke engedélyezi egy periodikus megszakítás kiváltását. A megszakítás gyakoriságát az A állapotregiszter 0–3. bitjeivel beállított frekvencia határozza meg. A rendszer indításakor a bit értéke 0, vagyis a megszakítás nem engedélyezett.

7. bit: A bit 1 értéke letiltja a folyamatban lévő aktualizálási ciklusokat, azaz megállítja az idő számlálását. Ekkor az óra mind a 14 bájtjába új érték írható. A bit 0-ra állításával az óra a beállított kezdőértékkel újra indítható. A rendszer az indításakor a bit értékét természetesen 0-ra állítja, hogy az órát ne állítsa meg.

A 4–6. bitek jelentésénél beszéltünk egy megszakításról anélkül, hogy ezt közelebbről meghatároztuk volna. Bár – mint láttuk – a megszakítás meghívásának három különböző oka lehet, mindegyik esetben ugyanarról a – 70(h) számú – megszakításról van szó. A megszakítás mögött egy BIOS rutin van, amelyet – többek között – a 15(h) BIOS megszakítás két időmérő funkciója (jelzőbit beállítása és várakozás) is használ.

Azt, hogy ezt a 70(h) megszakítást melyik ok váltja ki, a rutin az óra C jelű állapotregiszteréből tudja kiolvasni. Mindhárom oknak megfelel egy bit, amely azonosítja a kiváltó eseményt. Az állapotregiszter felépítését a 36. ábrán mutatjuk meg. A regiszter 7. bitje 1 értékű lesz, ha a 6–4. bitek bármelyike 1 értékű.



36. ábra: Az óra C állapotregisztere

Amint látható, a C regiszter 4–6. bitje rendre megfelel a B regiszter megfelelő bitjének. Ez azt jelenti, hogy ha pl. a B regiszter 5. bitjének az értéke 1 – vagyis a riasztási idő elérésekor ki kell váltódnia a megszakításnak –, akkor a C regiszter 5. bitjének az értéke is 1. (A C regiszter bitjei csak olvashatók.) Ezért annak a rutinnak, amelyik a 70(h) megszakítást figyeli, a megszakítás észlelésekor elsőként a C állapotregisztert kell beolvasnia, hogy megállapíthassa, mi váltotta ki a megszakítást.

A programozó számára a valós idejű óra D állapotregiszterének csak a 7. bitje érdekes: ha a bit értéke 1, akkor az órát tápláló telep feszültsége megfelelő. Ellenkező esetben a telep lemerült, és ki kell cserélni.

Ezzel befejeztük a CMOS RAM valós idejű órával kapcsolatos bájtjainak ismertetését. Röviden nézzük még meg a 0E(h) címen lévő diagnosztika bájt, a 10(h) címen lévő, a hajlékonylemezes meghajtók típusát leíró és a 14(h) címen lévő berendezés bájt felépítését.

A diagnosztika bájt:

0 – 1. bit: fenntartott

2. bit: 0 = az idő érvényes
1 = az idő nem érvényes

3. bit: 0 = a merevlemezes lemezmeghajtó (C:) és a lemezvezérlő rendben
1 = a merevlemezes lemezmeghajtó (C:) és a lemezvezérlő nincs rendben

4. bit: 0 = a bekapcsolási teszt által megállapított tárméret megegyezik a CMOS RAM konfigurációs rekordjában a 15(h) – 18(h) címeken nyilvántartott tárérettel
1 = a két tárméret nem egyezik meg

5. bit: 0 = a bekapcsolási teszt a CMOS RAM 14(h) címén lévő berendezés bájt tartalmát érvényesnek találta
1 = a berendezés bájt tartalma nem érvényes

6. bit: 0 = a 2E(h) – 2F(h) címeken lévő vizsgálóösszeg rendben van
1 = a vizsgálóösszeg nincs rendben

7. bit: 0 = az áramkört tápláló telep rendben van
1 = a telep lemerült

A hajlékonylemezes meghajtók típusát leíró bájt:

0 – 3. bit: a másodikként installált lemezmeghajtó (B:) típusa
0000 : nincs második meghajtó
0001 : 320/360 kbájtos lemezmeghajtó
0010 : 1.2 Mbájtos lemezmeghajtó

4 – 7. bit: 0000 : a berendezés nem tartalmaz lemezmeghajtót
0001 : 320/360 kbájtos lemezmeghajtó
0010 : 1.2 Mbájtos lemezmeghajtó

A berendezés bájt:

0. bit : a berendezésben van(nak) hajlékonylemezes meghajtó(k)

1. bit : 0 = a berendezésben nincs matematikai társprocesszor
1 = a berendezésben van matematikai társprocesszor

2 – 3. bit: fenntartott

4 – 5. bit: elsődleges video megjelenítés
00 : fenntartott
01 : színes-grafikus kártya, 40 oszlop/sor
10 : színes-grafikus kártya, 80 oszlop/sor
11 : monokróm kártya

6 – 7. bit: a berendezésben lévő hajlékonylemezes meghajtók száma
00 : 1 lemezmeghajtó
01 : 2 lemezmeghajtó
10 : fenntartott
11 : fenntartott

Az RTC/CMOS a RAM tárának írásával/olvasásával programozható. Az áramkör a 70(h) és 71(h) portcímeken keresztül érhető el. Első lépésként a 70(h) portra ki kell küldeni a RAM tár címét. Az áramkör így értesül arról, hogy a program a RAM tárához

hozzá akar férni. A következő lépésben a 71(h) porton keresztül a megcímzett bájt írható (OUT) vagy olvasható (IN).

2.6.1. Programpéldák

A valós idejű óra kezelésének alapeseteit (az óra tartalmának beolvasása, ill. módosítása) mutatja be három egyszerű példaprogramunk. Teljesen analóg mindhárom program felépítése. Valamennyi tartalmaz egy-egy alapeljárást a valós idejű óra egy tárelemének kiolvasására, ill. beírására és egy-egy olyan olvasási, ill. írási eljárást, melyek az óra dátum- és időinformációkat tartalmazó tárhelyeinek elérésekor hívódnak. Ez az utóbbi eset azért kezelendő az alapesettől eltérően, mert a dátum, ill. az idő tárolása egyaránt történhet bináris vagy BCD alakban. Akár írásról, akár olvasásról legyen tehát szó, először be kell olvasni a B állapotregiszter tartalmát és az 1. bit alapján el kell dönteni, hogy milyen módon történik az adattárolás. Amennyiben BCD üzemmódban működik a valós idejű óra, a dátum és idő kezelése során adattípus konverziókat is kell végeznünk: beolvasás után, de a megjelenítés előtt BCD → bináris, írás előtt bináris → BCD átalakítást. A speciális olvasást, ill. írást végző eljárásaink elvégzik a vizsgálatot és a fent leírt konverziókat (ha szükséges).

A programok tartalmazzák az óra üzemképességének ellenőrzését a diagnosztikai bájt 7. bitjének tartalma alapján, nem kezelik viszont az A és B állapotregiszterek 7. bitjeit, amelyekkel az óra elérésének ideje alatt letiltható az időadatok aktualizálása. Az utóbbi „hiányosság" oka a terjengősség elkerülésére irányuló törekvésünk, a programok a kívánt funkciókat így is korrektül szemléltetik (a letiltás hiánya csak az óra „pontosságát" befolyásolja a korábbi fejezetben leírtaknak megfelelően).

A valós idejű óra demonstrációs célú átállítása opcionális, amennyiben átírja az idő- és dátumadatokat, úgy azokat a következő rendszerindítás után ismét be kell állítani a helyes értékekre.

```
1000 '
1005 '
1010 '                    V A L O R A . B A S
1020 '
1030 '  Funkciója : Valós idejű óra kezelési funkciók szemléltetése.
1040 '
1050 '
1060 '
1070 CPORT%  = &H70                          ' RTC címport.
1080 APORT%  = &H71                          ' RTC adatport.
1090 '
1100 MP%     =  0                            ' RTC-n belüli tárcímek.
1110 PERC%   =  2
1120 ORA%    =  4
1130 NAPNUM% =  6
1140 NAP%    =  7
1150 HO%     =  8
1160 EV%     =  9
1170 STATB%  = 11
1180 DIAG%   = 14
1190 '
1200 DEF FNADAT$(A%) = RIGHT$(STR$(A%), LEN(STR$(A%))-1)
1210 '
1220 CIM% = DIAG% : GOSUB 2000 : PRINT       ' Az elem ellenőrzése.
1230 IF (ADAT% AND 128) = 0 THEN 1270
1240 PRINT "Kimerult a valós idejű órában az elem ...."
1250 PRINT : GOTO 1810
1260 '
1270 CIM% = EV%      : GOSUB 3000 : E%  = ADAT%
1280 CIM% = HO%      : GOSUB 3000 : H%  = ADAT%
1290 CIM% = NAP%     : GOSUB 3000 : N%  = ADAT%
1300 CIM% = NAPNUM%  : GOSUB 3000 : NN% = ADAT%
1310 CIM% = ORA%     : GOSUB 3000 : O%  = ADAT%
1320 CIM% = PERC%    : GOSUB 3000 : P%  = ADAT%
1330 CIM% = MP%      : GOSUB 3000 : M%  = ADAT%
1340 '
1350 RESTORE 1820    : FOR I% = 1 TO NN% : READ NN$ : NEXT
1360 '
1370 PRINT "A valós idejű óra alapján : ";
1380 PRINT FNADAT$(E%);"-";FNADAT$(H%);"-";FNADAT$(N%);
1390 PRINT NN$;
1400 PRINT FNADAT$(O%);":";FNADAT$(P%);":";FNADAT$(M%)
1410 PRINT
1420 '
1430 CIM% = STATB%
1440 GOSUB 2000
1450 UM% = (ADAT% AND  2) * 6 + 12
1460 PRINT "Az óra";UM%;" órás üzemmódban működik." : PRINT
1470 PRINT "Ha nem kívánja az óra átállításának bemutatását, üsse"
1480 PRINT "le az <ESC>, ellenkező esetben egy másik billentyűt!"
1490 PRINT
1500 '
1510 Z$ = ""
1520 WHILE Z$ = ""                           ' <ESC> megnyomása esetén
1530     Z$ = INKEY$                         ' nem állítódik át az óra.
1540 WEND
1550 IF Z$ = CHR$(27) THEN GOTO 1810
1560 '
1570 CIM% = EV%      : ADAT% = 90 : GOSUB 5000
1580 CIM% = HO%      : ADAT% = 12 : GOSUB 5000
1590 CIM% = NAP%     : ADAT% = 31 : GOSUB 5000
1600 CIM% = NAPNUM%  : ADAT% =  2 : GOSUB 5000
1610 CIM% = ORA%     : ADAT% = 23 : GOSUB 5000
```

```
1620 CIM% = PERC%    : ADAT% = 59 : GOSUB 5000
1630 CIM% = MP%      : ADAT% =  0 : GOSUB 5000
1640 '
1650 CIM% = EV%      : GOSUB 3000 : E%  = ADAT%
1660 CIM% = HO%      : GOSUB 3000 : H%  = ADAT%
1670 CIM% = NAP%     : GOSUB 3000 : N%  = ADAT%
1680 CIM% = NAPNUM%  : GOSUB 3000 : NN% = ADAT%
1690 CIM% = ORA%     : GOSUB 3000 : O%  = ADAT%
1700 CIM% = PERC%    : GOSUB 3000 : P%  = ADAT%
1710 CIM% = MP%      : GOSUB 3000 : M%  = ADAT%
1720 '
1730 RESTORE 1820    : FOR I% = 1 TO NN% : READ NN$ : NEXT
1740 '
1750 PRINT "A valós idejű óra átállítva 1990 szilveszterére: ";
1760 PRINT FNADAT$(E%);"-";FNADAT$(H%);"-";FNADAT$(N%);
1770 PRINT NN$;
1780 PRINT FNADAT$(O%);":";FNADAT$(P%);":";FNADAT$(M%)
1790 PRINT
1800 '
1810 END
1820 DATA " Vasárnap ", " Hétfő ", " Kedd ", " Szerda "
1830 DATA " Csütörtök ", " Péntek ", " Szombat "
1840 '
2000 '
2010 ' AZ RTC RAM 1 BAJTJANAK BEOLVASASA
2020 '    BE : CIM% az RTC-n belüli cím
2030 '    KI : ADAT% a beolvasott adat
2040 '
2050 '
2060 OUT CPORT%,CIM%                    ' Az RTC cím a megfelelő portra.
2070 ADAT% = INP(APORT%)                ' A portról az adat beolvasása.
2080 RETURN
2090 '
3000 '
3010 ' AZ RTC RAM 1 BAJTJANAK BEOLVASASA, HA
3020 ' SZUKSÉGES, BINÁRIS ALAKRA HOZASA
3030 '    BE : CIM% az RTC-n belüli cím
3040 '    KI : ADAT% a beolvasott adat
3050 '
3060 '
3070 GOSUB 2000
3080 BCD% = ADAT%
3090 CIM% = STATB%
3100 GOSUB 2000
3110 IF (ADAT% AND 2) = 0 THEN GOTO 3130        ' BCD módus esetén
3120 BCD% = (BCD% AND 15) + INT(BCD% / 16) * 10 ' konverzió.
3130 ADAT% = BCD%
3140 RETURN
3150 '
4000 '
4010 ' AZ RTC RAM TERÜLETRE 1 BÁJT ÍRÁSA
4020 '    BE : CIM% az RTC-n belüli cím
4030 '         ADAT% a kiírandó adat
4040 '
4050 '
4060 OUT CPORT%,CIM%                    ' Az RTC cím a megfelelő portra.
4070 OUT APORT%,ADAT%                   ' Az adat kiírása a portra.
4080 RETURN
4090 '
```

```
5000 '
5010 '  AZ RTC RAM TERULETRE 1 BAJT IRASA, HA
5020 '  SZUKSEGES, ELOTTE BCD ALAKRA KONVERTALAS
5030 '     BE : CIM% az RTC-n belüli cím
5040 '          ADAT% a kiírandó adat
5050 '
5060 '
5070 BCD% = ADAT%
5080 TEMP% = CIM%
5090 CIM% = STATB%
5100 GOSUB 2000
5110 IF (ADAT% AND 2) = 0 THEN GOTO 5130                ' BCD módus esetén
5120 BCD% = INT(BCD% / 10) * 16 + BCD% MOD 10           ' konverzió BCD-re.
5130 ADAT% = BCD%
5140 CIM% = TEMP%
5150 GOSUB 4000
5160 RETURN
```

```
/*  ┌──────────────────────────────────────────────────────┐  */
/*  │                                                      │  */
/*  │                    V A L O R A . C                   │  */
/*  │                                                      │  */
/*  │  Funkciója : Valós idejű óra kezelési funkciók szemléltetése.  │  */
/*  │                                                      │  */
/*  └──────────────────────────────────────────────────────┘  */

#include <dos.h>
#include <conio.h>

#define byte unsigned char

#define RTCcPORT  0x70                          /* RTC címport.   */
#define RTCaPORT  0x71                          /* RTC adatport.  */

#define MP        0                     /* RTC-n belüli tárcímek. */
#define PERC      2
#define ORA       4
#define NAPNUM    6
#define NAP       7
#define HO        8
#define EV        9
#define STATB    11
#define DIAG     14

byte  temp;

/*  ┌──────────────────────────────────────┐  */
/*  │ OLV : AZ RTC RAM 1 BAJTJANAK BEOLVASASA │  */
/*  └──────────────────────────────────────┘  */

byte OLV(cim)
byte cim;                                   /* Az RTC-n belüli cím.  */

{
outp(RTCcPORT, cim);                /* Az RTC cím a megfelelő portra.  */
return(inp(RTCaPORT));              /* A portról az adat beolvasása.   */
}

/*  ┌──────────────────────────────────────┐  */
/*  │ OLVK : AZ RTC RAM 1 BAJTJANAK BEOLVASASA, │  */
/*  │        HA SZUKSEGES, BINARIS ALAKRA HOZAS │  */
/*  └──────────────────────────────────────┘  */

byte OLVK(cim)
byte cim;                                   /* Az RTC-n belüli cím.  */

{
temp = OLV(cim);
if (!(OLV(STATB) & 2))                          /* BCD módus esetén  */
   return(temp);                                /* konverzió.        */
else return((temp >> 4) * 10 + (temp & 15));
}

/*  ┌──────────────────────────────────┐  */
/*  │ IR : AZ RTC RAM TERULETRE 1 BAJT ÍRASA │  */
/*  └──────────────────────────────────┘  */

void IR(cim, adat)
byte cim;                                   /* Az RTC-n belüli cím.  */
byte adat;                                  /* A beírandó adat.      */
```

```
{
outp(RTCcPORT, cim);                    /* Az RTC cím a megfelelő portra. */
outp(RTCaPORT, adat);                   /* Az adat kiírása a portra.      */
}

/*
/*  IRK : AZ RTC RAM TERULETRE 1 BAJT IRASA                              */
/*        HA SZUKSEGES, BCD ALAKRA KONVERZIO                             */
/*                                                                       */

void IRK(cim, adat)
byte cim;
byte adat;                                  /* Az RTC-n belüli cím.      */
                                            /* A beírandó adat.          */
{
if ((OLV(STATB) & 2))                       /* BCD módus esetén          */
   adat = (adat / 10) * 16 + adat % 10;     /* konverzió BCD-re.         */
IR(cim, adat);
}

/*
/*                          F Ő P R O G R A M                            */
/*                                                                       */

void main()

{
static char *napneve[] = {
                          "Vasárnap", "Hétfő", "Kedd", "Szerda",
                          "Csütörtök", "Péntek", "Szombat"
                          };

if (!(OLV(DIAG) & 128))                     /* Az elem ellenőrzése.      */
   {
   printf("A valós idejű óra alapján : ");
   printf("%d-%d-%d", OLVK(EV), OLVK(HO), OLVK(NAP));
   printf(" %s ", napneve[OLVK(NAPNUM)-1]);
   printf("%d:%d:%d",OLVK(ORA), OLVK(PERC), OLVK(MP));
   printf("Az óra %d órás üzemmódban működik.",
          (OLV(STATB) & 2)*6+12);
   printf("Ha nem kívánja az óra átállításának bemutatását, üsse");
   printf("le az <ESC>, ellenkező esetben egy másik billentyűt!");
   if (getch() != 27)
      {
      IRK(EV, 90); IRK(HO, 12); IRK(NAP, 31); IRK(NAPNUM, 2);
      IRK(ORA, 23); IRK(PERC, 59); IRK(MP, 0);
      printf("A valós idejű óra átállítva 1990 szilveszterére: ");
      printf("%d-%d-%d", OLVK(EV), OLVK(HO), OLVK(NAP));
      printf(" %s ", napneve[OLVK(NAPNUM)-1]);
      printf("%d:%d:%d",OLVK(ORA), OLVK(PERC), OLVK(MP));
      }
   }
else printf("Kimerült a valós idejű órában az elem ....");
}
```

```
{ ┌──────────────────────────────────────────────────────────┐ }
{ │                                                          │ }
{ │                      V A L O R A . P A S                 │ }
{ │                                                          │ }
{ │  Funkciója : Valós idejű óra kezelési funkciók szemléltetése.  │ }
{ │                                                          │ }
{ └──────────────────────────────────────────────────────────┘ }

program VALORA;

const RTCcPORT  = $70;                          { RTC címregiszter.   }
      RTCaPORT  = $71;                          { RTC adatregiszter.  }

      MP        = 0;                        { RTC-n belüli tárcímek.  }
      PERC      = 2;
      ORA       = 4;
      NAPNUM    = 6;
      NAP       = 7;
      HO        = 8;
      EV        = 9;
      STATB     = 11;
      DIAG      = 14;

var z : char;              { Csak a várakozási állapotokhoz szükséges. }

{ ┌──────────────────────────────────────┐ }
{ │ OLV : AZ RTC RAM 1 BAJTJÁNAK BEOLVASÁSA │ }
{ └──────────────────────────────────────┘ }

function OLV(cim : integer) : integer;        { Az RTC-n belüli cím.  }

begin
   if (cim < 0) or (cim > 63)           { Az átadott cím ellenőrzése. }
      then OLV := -1
      else
      begin
         port[RTCcPORT] := cim;      { Az RTC cím a megfelelő portra. }
         OLV := port[RTCaPORT]       { A portról az adat beolvasása.  }
      end
end;

{ ┌──────────────────────────────────────┐ }
{ │ OLVK : AZ RTC RAM 1 BAJTJÁNAK BEOLVASÁSA, │ }
{ │        HA SZÜKSÉGES, BINÁRIS ALAKRA HOZÁSA │ }
{ └──────────────────────────────────────┘ }

function OLVK(cim : integer) : integer;       { Az RTC-n belüli cím.  }

var temp : integer;

begin
   if (OLV(STATB) and 2 = 0)                        { BCD módus esetén }
      then OLVK := OLV(cim)                         { Konverzió.       }
      else
      begin
         temp := OLV(cim);
         OLVK := (temp shr 4) * 10 + temp and 15
      end
end;
```

```
{
{  IR : AZ RTC RAM TERULETRE 1 BAJT IRASA                    }
{                                                            }
{                                                            }

procedure IR(cim  : integer;                  { Az RTC-n beluli cim. }
             adat : byte);                    { A beirando adat.     }
begin
   port[RTCcPORT] := cim;              { Az RTC cim a megfelelo portra. }
   port[RTCaPORT] := adat              { Az adat kiirasa a portra.      }
end;

{
{  IRK : AZ RTC RAM TERULETRE 1 BAJT IRASA                   }
{        HA SZUKSEGES, BCD ALAKRA KONVERZIO                  }
{                                                            }

procedure IRK(cim  : integer;                 { Az RTC-n beluli cim. }
              adat : byte);                   { A beirando adat.     }
begin
   if (OLV(STATB) and 2 <> 0) then                { BCD mod us esetén  }
      adat := (adat div 10) * 16 + adat mod 10;   { konverzio BCD-re. }
   IR(cim, adat)
end;

{
{                  F Ő P R O G R A M                         }
{                                                            }

begin
   writeln(#13#10);
   if OLV(DIAG) and 128 = 0 then                  { Az elem ellenorzese. }
   begin
      write('A valós idejü óra alapján : ');
      write(OLVK(EV), '-', OLVK(HO), '-', OLVK(NAP));
      case OLVK(NAPNUM) of
         1 : write(' Vasárnap ');
         2 : write(' Hétfő ');
         3 : write(' Kedd ');
         4 : write(' Szerda ');
         5 : write(' Csutortok ');
         6 : write(' Péntek ');
         7 : write(' Szombat ')
      end;
      write(OLVK(ORA), ':', OLVK(PERC), ':', OLVK(MP), #13#10#13#10);
      write('Az óra ',(OLV(STATB) and 2)*6+12);
      write(' órás üzemmódban mükodik.');
      write(#13#10#13#10);
      writeln('Ha nem kívánja az óra átállitásának bemutatását,usse');
      writeln('le az <ESC>, ellenkezo esetben egy másik billentyüt!');
      read(kbd, z);
      if z <> chr(27) then
         begin
            IRK(EV, 90); IRK(HO, 12); IRK(NAP, 31); IRK(NAPNUM, 2);
            IRK(ORA, 23); IRK(PERC, 59); IRK(MP, 0);
            write(#13#10,'Az óra átállitva 1990 szilveszterére: ');
            write(OLVK(EV), '-', OLVK(HO), '-', OLVK(NAP));
            case OLVK(NAPNUM) of
               1 : write(' Vasárnap ');
               2 : write(' Hétfő ');
               3 : write(' Kedd ');
               4 : write(' Szerda ');
               5 : write(' Csutortok ');
```

```
            6 : write(' Péntek ');
            7 : write(' Szombat ')
          end;
          write(OLVK(ORA), ':', OLVK(PERC), ':', OLVK(MP));
          write(#13#10#13#10)
        end
  end
  else
  begin
        write('Kimerült a valós idejű órában az elem ....');
        write(#13#10#13#10)
    end
  end.
```

2.7. Hardver megszakítások

A számítógépes rendszer belső működésében, a rendszert alkotó belső és külső egységek bonyolult áramkörei és az általuk vezérelt különböző folyamatok közötti összhang megteremtésében igen fontos szerepet játszanak a hardver megszakítások. Ezek azok a megszakítások, amelyeket nem valamely felhasználói program, hanem maga a CPU, azaz a számítógép központi végrehajtó egysége, vagy a rendszerben lévő valamelyik hardver elem vált ki. E megszakítások többsége mélyen a „felszín" alatti szinten, a felhasználó szeme elől elrejtve fut, akkor is, amikor a képernyőn a kurzor villogásán kívül látszólag semmi sem történik (egyébként a kurzor villogtatását is hardver megszakítás váltja ki). A hardver azonban természetesen ilyenkor sem „alszik" – elegendő csak arra gondolnunk, hogy egy billentyű lenyomására azonnal reagál, pl. megjeleníti a neki megfelelő karaktert a képernyőn. A megfelelő karakter kiválasztása és megjelenítése – szintén hardver megszakításokkal támogatva – ugyan már a BIOS vagy a DOS dolga, de azt, hogy valaki lenyomta a billentyűt, elsőként – mint látni fogjuk, a 8259-es megszakításvezérlő IC-n keresztül – a hardver érzékeli, természetesen ugyancsak egy hardver megszakításon keresztül. A példák vég nélkül sorolhatók.

A következőkben azokat a hardver megszakításokat foglaljuk össze, amelyeket egy felhasználói program – még ha ezekre ritkán is van szüksége – a megszakítási vektortáblán keresztül a saját céljaira is felhasználhat.

INT 00(h) – Osztási túlcsordulás

Ezt a megszakítást maga a mikroprocesszor hívja meg minden olyan DIV vagy az IDIV assembly utasítás végrehajtását követően, amikor a kapott eredmény – az osztási művelet hányadosa – nem fér el a célregiszterben. Természetesen ugyanezt a megszakítást váltja ki a 0-val való osztás is. A rendszer indításakor a DOS ezt a megszakítási vektort általában egy saját rutin címére írja át, amely a Division by Zero (Osztás nullával) hibaüzenetet írja ki, majd a program futása a soron következő utasítás végrehajtásával folytatódik.

INT 01(h) – Lépésenkénti programvégrehajtás (Trap)

Ezt a megszakítást ugyancsak a mikroprocesszor hívja meg abban az esetben, ha az állapotjelző (flag) regiszterében a nyomkövetés (TRAP) jelzőbit értéke 1. Ekkor a processzor minden egyes gépi kódú utasítás végrehajtását követően meghívja ezt a megszakítást. Ennek az az értelme, hogy ilyen módon egy programtesztelő program nyomon követheti a tesztelendő – gépi kódú – program utasításainak a hatását, lépésről-lépésre kiírathatja a regiszterek tartalmát stb.

A lépésenkénti programvégrehajtás alól maga a tesztelő program kivételt képez, hiszen ha a processzor ezt is ebben az üzemmódban hajtaná végre, akkor végtelen ciklusba kerülne. Amikor a processzor a lépésenkénti programvégrehajtás során áttér az INT 01(h) megszakítási rutinra, akkor először a verembe menti az állapotjelző regiszterét (benne a TF = 1), majd törli a TF bitet, és normál módon végrehajtja a megszakítási rutin utasításait. A rutin befejezését jelző IRET hatására előveszi a veremből az elmentett állapotjelző regiszterét. Ekkor a TF bit értéke ismét 1 lesz, azaz a program utasításait ismét lépésenként hajtja végre.

Mivel a normál felhasználói programok ezt a megszakítást általában nem használják, a DOS ezt a megszakítási vektort a rendszer indításakor egy IRET utasításra állítja.

INT 02(h) – NMI (Non-Maskable Interrupt)

Ezt a nem maszkolható megszakítást a processzor NMI bemenetére érkező jel váltja ki. Ilyen megszakítás kérő jelet általában akkor küld a rendszer, ha a RAM elemekben valamilyen hibát észlelt. Mivel egy ilyen hiba erősen korlátozza a rendszer működését, sőt a rendszer összeomlását is okozhatja, ez a megszakítás nem tiltható le, azaz nem maszkolható. A rendszer valamennyi megszakítási kérése közül ez a legnagyobb prioritású, és ezért a lehető leggyorsabban végrehajtódik. A megszakítás többnyire egy BIOS-rutint hív meg, amely közli a felhasználóval a hiba jellegét és a meghibásodott tárelem számát, majd leállítja a rendszer működését.

Ilyen, nem maszkolható megszakítást válthat ki az egyes PC-kben meglévő aritmetikai társprocesszor is, ha a működésében valamilyen hibát észlel. Bár ezt a megszakítást azért nevezik nem maszkolható megszakításnak, mert az összes többi hardver megszakítással ellentétben nem tiltható le, a PC-k esetében ez nem igaz. Mind a PC/XT, mind a PC/AT esetében van egy olyan port, amelynek a 7. bitje meghatározza, hogy az NMI-megszakítás végrehajtható-e. A PC és a XT gépeknél ez a port az A0(h), az AT-nél a 70(h). Ha ezekre a portokra a 0 értéket írjuk, akkor az NMI végrehajtására nem kerül sor, ha 80(h)-t, akkor a megszakítás végrehajtását engedélyezzük. Az AT 70(h) portcímének kettős kihasználását a valós idejű óra programozásánál vegyük figyelembe.

INT 03(h) – Töréspont (Breakpoint)

Ezt a megszakítást is programtesztelő programok használják. Ennek a megszakításnak az a különlegessége, hogy míg valamennyi más megszakítás meghívásához 2 bájtos gépi kódú utasításra van szükség (az első bájt értéke CD(h), a második a hívott megszakítás száma), a 3-as megszakítás hívásához csak egy 1 bájtos gépi kódú utasítást (CC(h)) kell kiadni. Ez a megszakítás különösen jól használható olyan programok tesztelésére, amelyeknél a program futását egy előre meghatározott ponton meg kell állítani, hogy a

bekövetkezett változásokat ki lehessen értékelni. Egy tesztelő programon (ilyen pl. a DOS DEBUG nevű programja) belül ez úgy valósítható meg, hogy a vizsgálandó programnak azokra a pontjaira, ahol a vizsgálatot el kell végezni, egy INT 3 utasítást (vagy más programnyelvekben a 3-as megszakítás meghívását elvégző utasításokat) kell beírni. Amikor egy így „preparált" program végrehajtására sor kerül, a program futása az elhelyezett töréspontokon áttér a megszakítási rutinra, és a kapott változások, regisztertartalmak, jelzőbitek stb. állapota megvizsgálható. Ha a vizsgált program már rendben működik, akkor ezeket a töréspontokat természetesen el kell távolítani.

INT 04(h) – Túlcsordulás (Overflow)

Ezt a megszakítást egy feltételhez kötött utasítás hívhatja meg. Ez az utasítás az INTO (Interrupt on Overflow) assembly nyelvű utasítás, amely akkor hívja meg a 4-es megszakítást, ha a végrehajtásakor a jelzőbitek regiszterében a túlcsordulás bit értéke 1. Erre aritmetikai műveleteknél (pl. szorzásnál) kerülhet sor, amikor a szorzás eredménye a regiszterben már nem ábrázolható. A megszakítás a normál INT utasítással is meghívható, de a végrehajtásnak ekkor nem feltétele a túlcsordulás bit értéke. A DOS a megszakítási vektort egy IRET utasításra állítja.

INT 05(h) – Hardcopy (PrintScreen)

Bár ez a megszakítás a hardver megszakítások között kapott helyet, valójában a BIOS használja. Ezt a megszakítást akkor hívja meg a BIOS, ha a billentyűzeten egyidejűleg lenyomjuk a Shift és a PrtSc (Print Screen) jelű billentyűket. A megszakítás feladata, hogy a képernyő tartalmát a géphez csatlakoztatott nyomtatóra kinyomtassa. Mivel a nyomtatóra való írás a BIOS feladata, a BIOS ezt a vektort úgy állítja be, hogy az a meghívásakor a ROM-BIOS-ban lévő nyomtató rutinra mutasson. A megszakítás természetesen assembly vagy más, magas szintű nyelven írt programokból is meghívható, így egy program futása közben is nyomtatóra írható a képernyő tartalma.

INT 06(h) – 07(h) – fenntartott megszakítások

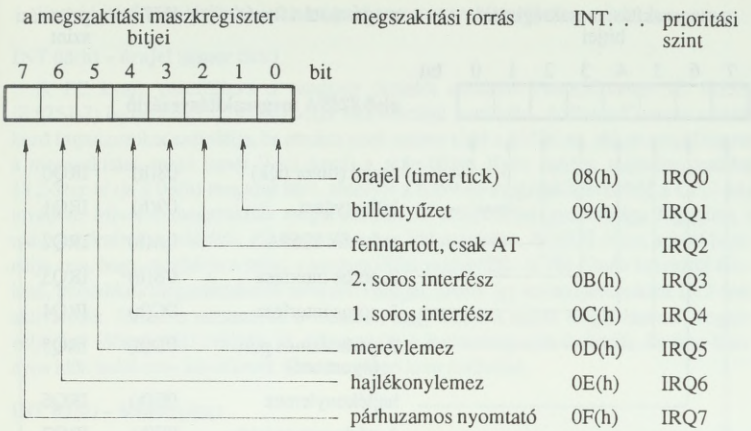
Mielőtt rátérnénk a további, 08(h) – 0F(h), ill. a csak az AT-nél meglévő 70(h) – 77(h) hardver megszakítások ismertetésére, kis kitérőt kell tennünk. Az eddig ismertetett megszakításokat az jellemezte, hogy azokat maga a mikroprocesszor váltotta ki. Ezzel szemben a 08(h) – 0F(h), ill. a 70(h) – 77(h) közötti megszakításokat valamely külső egység, periféria kezdeményezheti. Mivel a rendszerben számos olyan külső egység van, amely megszakítási kérelemmel fordulhat a processzorhoz, ráadásul ahhoz, hogy a rendszer egészének optimális működése biztosítható legyen, a processzornak ezeket a kérelmeket nem is az érkezési sorrendjükben kell kiszolgálnia, a rendszernek tartalmaznia kell olyan logikai áramkör(öke)t, amely ezeket a hardver megszakításokat érzékeli, „fontosságuk" (mint mondjuk: prioritásuk) szerint sorrendbe állítja, és így előkészítve továbbítja a processzorhoz. Ezt a feladatot a 8259-es jelű, röviden PIC-nek (Programable Interrupt Controller, programozható megszakításvezérlő) nevezett integrált áramkör látja el, amely egyidejűleg 8 megszakítási forrást tud kezelni. A külső egységek által bejelentett megszakítás kérések tehát először erre a 8259-esre kerülnek, amely – megfelelő előkészítés után – ezeket a mikroprocesszorhoz továbbítja. A mikroprocesszor a 8259-esről érkező megszakításkéréseket az INTR nevű bemenetén fogadja, és a kérésnek eleget is tesz abban az esetben, ha az állapotre-

giszterében lévő IF megszakításjelző bit értéke 1. A bit értéke az STI és a CLI assembly utasításokkal állítható be: az STI bekapcsolja az I bitet, vagyis engedélyezi, míg a CLI törli az IF bitet, azaz letiltja a megszakítás kiszolgálását.

A megszakításokat azonban nemcsak a mikroprocesszor tilthatja le, hanem a 8259-es megfelelő programozásával ezek már előzőleg is „kiszűrhetők", maszkolhatók. A megszakítások maszkolását a 8259-es 21(h) portcímen elérhető, ún. megszakítási maszkregisztere (Interrupt Mask Register) végzi. Ennek a 8 bites regiszternek mindegyik bitje hozzá van rendelve a 8259-es által kezelhető 8, megszakítást kiváltható egységhez: a 0. bit a 0-val, az 1. bit az 1-gyel azonosított stb. egységhez. Ezek a sorszámok egyúttal az egyes egységek által kiváltott megszakítások prioritását is jelzik: a 0-val azonosított egység megszakítása a legnagyobb, a 7-tel azonosított egység megszakítása a legkisebb prioritású. Ha a megszakítási maszkregiszter megfelelő bitjének értéke 0, akkor az ehhez tartozó egység megszakítási kérését a 8259-es továbbítja a processzorhoz, ellenkező esetben a megszakítást letiltja. Ha egyidejűleg több megszakítási kérés érkezik a 8259-re, akkor ez először a legmagasabb prioritásút továbbítja a processzorhoz, a többit pedig „elraktározza". Amikor a processzor a kért megszakítási rutint végrehajtotta, akkor a meghívó programnak a 8259-es 20(h) portcímen elérhető, ún. megszakítási parancsregiszterében (Interrupt Command Register) lévő 5. bitet (EOI, End of Interrupt) magasra kell állítania (vagyis a regiszterbe a 20(h) értéket kell beírnia). A meghívó program ezzel hozza a 8259-es tudomására, hogy a processzor a megszakítási kérését teljesítette, és következhet a prioritási sorrendben utána álló kérés továbbítása.

Mint említettük, a 8259-es megszakítási maszkregiszterének bitjei hozzá vannak rendelve a különböző megszakítási forrásokhoz (egységekhez), és ez a hozzárendelés egyúttal a prioritási sorrendet is meghatározza (legalábbis alapesetben). A megszakításvezérlő áramkörök számát, és ebből következően a kezelhető megszakítási források számát illetően a PC/XT és a PC/AT eltér egymástól: míg az XT az egyetlen 8259-es megszakításvezérlő áramkörével 8, addig az AT a két, 8259A jelű áramkörével 16 (helyesebben 15) megszakítási forrást tud kezelni. A prioritási sorrendet az IRQ0 – IRQ7, ill. az AT-nél az IRQ0 – IRQ15 sorszámokkal jelölik, ahol az IRQ az Interrupt Request (magyarul megszakításkérés) kifejezést jelenti.

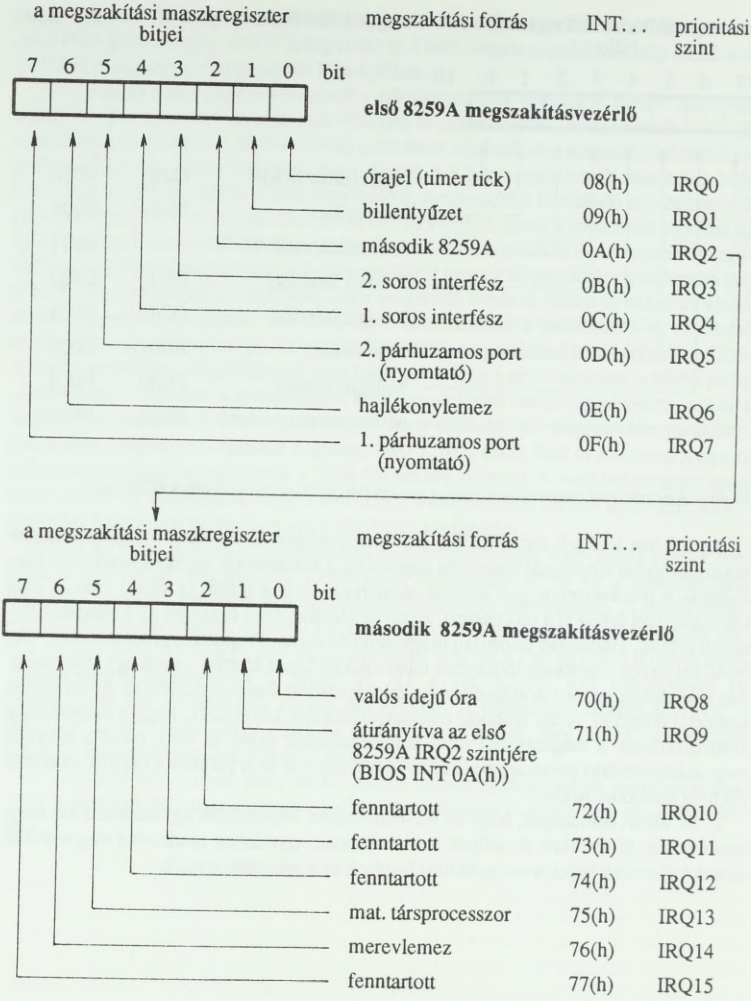
A 37. ábrán bemutatjuk, hogy az XT esetén milyen módon vannak egymáshoz rendelve a megszakítási maszkregiszter bitjei, a megszakítási források és a prioritási szintek.



37. ábra: A megszakítási maszkregiszter bitjei, a megszakítási források és a prioritások PC/XT-nél

Az AT-ben lévő két darab 8259A jelű megszakításvezérlő áramkör kaszkádolásnak nevezett módon egymással össze van kapcsolva: a két áramkör egyike közvetlenül kapcsolódik a processzorhoz (ezt nevezik masternek), míg a másik (a slave) csak az első közvetítésével érheti el a processzort. Ezzel az elrendezéssel összesen 15 különböző forrásból érkező, különböző prioritású megszakításkérés továbbítható a processzorhoz. (Az NMI-vel együtt a rendszer 16 hardver megszakítást képes kezelni – ezek egy része azonban nincs kihasználva.) A második megszakításvezérlő maszkregisztere az A1(h) porton keresztül érhető el. A két áramkör összekapcsolásából következik, hogy a megszakítást kérő eszköznek a megszakítási rutin befejeződésekor mind az első, mind a második megszakításvezérlő parancsregiszterébe be kell írnia az EOI parancsot (az első portcíme 20(h), a másodiké A0(h).

A 38. ábrán bemutatjuk, hogy az AT-ben hogyan kapcsolódik egymáshoz a két megszakításvezérlő áramkör, és milyen módon vannak egymáshoz rendelve a megszakítási maszkregiszterek bitjei, a megszakítási források és a prioritási szintek.



38. ábra: A megszakítási maszkregiszterek bitjei, a megszakítási források és a prioritások PC/AT-nél

Ennyi kitérő után térjünk vissza magukhoz a megszakításokhoz.

INT 08(h) – órajel (timer tick)

A PC kvarc oszcillátora a rendszer órajelét előállító Timer/Counter IC (8253, ill.8254-2) bemenetére 1.193180 MHz frekvenciájú jelet küld. A Timer/Counter a beérkező impulzusokat számlálja, és amikor ezek száma eléri a 65536-ot, akkor meghívja ezt a megszakítást, majd ismét 0-ról kezdi a számlálást. Ilyen módon másodpercenként 18.2-szer hívja a 08(h) megszakítást, amelyet a 8259-es megszakításvezérlő a CPU-hoz továbbít. Mivel e megszakítás meghívásának az ismétlődési gyakorisága független a rendszer órafrekvenciájától, jól használható az idő mérésére. A BIOS is ezt a jelet használja arra, hogy meghívja a saját, a megszakítási vektortábla 1C(h) címén keresztül elérhető, periodikus megszakításnak nevezett rutinját, amely így másodpercenként 18.2-szer aktiválódik. Ennek a rutinnak az a feladata, hogy minden egyes meghívásakor megnövelje egy időszámláló értékét, ill. kikapcsolja a lemezmeghajtó motorját, ha egy bizonyos időn belül nem következik be valamilyen lemezművelet.

INT 09(h) – billentyűzet

A billentyűzetben egy 8048-as (AT esetén pl. 8039-es) jelű mikroprocesszor található, amely a billentyűzetet figyeli, és a billentyűk lenyomását, ill. elengedését regisztrálja. Ha ilyen eseményt észlel, akkor ezt közölnie kell a CPU-val, hogy az a billentyű működtetésére megfelelően reagálhasson. Ezért a billentyűzet ilyen esetben a megszakításvezérlő áramkörön keresztül meghívja a 9-es megszakítást, amely elindít egy BIOS rutint. Ennek a rutinnak az a feladata, hogy a billentyűzetről a programozható perifériakezelő áramkör (PPI) portjára átküldött billenyűkódot elolvassa, értelmezze, és beírja a billentyűzet pufferbe.

INT 0A(h) – hardvertől függő megszakítás

PC/XT-nél nem használt megszakítás. PC/AT-nél a második megszakításvezérlőhöz rendelt csatorna.

INT 0B(h) – Második soros interfész hardver megszakítása

INT 0C(h) – Első soros interfész hardver megszakítása

INT 0D(h) – merevlemezes meghajtó vezérlő áramköre (XT)

A PC/XT-ben lévő merevlemezes meghajtó vezérlő áramköre által kiváltott hardver megszakítás. Ezt a megszakítást akkor váltja ki a meghajtó vezérlő áramköre, amikor egy írási/olvasási művelet befejeződött.

Az AT-ben a merevlemezes meghajtó vezérlője erre a célra a 76(h) megszakítást váltja ki. Az AT-ben a 0D(h) megszakítást a második párhuzamos port (nyomtató) használhatja.

INT 0E(h) – hajlékonylemezes meghajtó vezérlő áramköre

Ezt a megszakítást az XT/AT hajlékonylemezes meghajtójának vezérlő áramköre váltja ki minden olyan esetben, amikor igényt tart a processzor figyelmére. Az áramkör a megszakítás kiváltásakor egy BIOS rutint hív meg, amely egy jelzőbit beírásával rögzíti, hogy a megszakítás megtörtént.

INT 0F(h) – (első) párhuzamos nyomtató vezérlő áramköre

Ezt a megszakítást a párhuzamos nyomtató vezérlő áramköre váltja ki jelezve, hogy a processzor átküldheti a következő karaktert.

INT 70(h) – valós idejű óra (csak AT)

A valós idejű óra több okból is kiválthat megszakítást. Ilyen ok lehet a beállított riasztási idő elérése, az idő és a dátum aktualizálásának befejeződése, vagy valamely más, programból lekérdezett esemény bekövetkezése. Ezt a megszakítást általában BIOS rutinok használják, amelyek először azt vizsgálják, hogy mi váltotta ki a megszakítást, majd ennek ismeretében reagálnak a megszakításra.

INT 75(h) – matematikai társprocesszor által kiváltott megszakítás (csak AT)

INT 76(h) – Az AT merevlemezes meghajtójának vezérlő áramköre által kiváltott megszakítás

2.7.1. Programpélda

A hardver megszakításokkal kapcsolatban álljon itt befejezésül egy mintaprogram, mely sok szempontból rokonságot mutat a billentyűzetkezelésnél bemutatott ORA1.ASM programmal. Ez a program is egy olyan rezidens megszakítás rutin, melynek feladata az óraidő megjelenítése a képernyőn. A két program között azonban jelentős különbségek is vannak. A legfontosabb eltérés, hogy ez a program a 08(h) hardver órajel megszakítást „téríti el". Ebből következik, hogy installálása után felhívása nem egy akcióhoz (a korábbi példában egy billentyű megnyomása) kötődik, hanem periodikusan, másodpercenként kb. 18.2-szer hívódik meg. További különbség, hogy mivel ez a program paraméterek nélkül kerül hívásra, egyszerűbb az installálás/inaktiválás közötti választás, egyszerűen minden második meghívás installálja, ill. inaktiválja a programot.

Mint korábban szó volt már róla, a 08(h) megszakítás (lévén gépfüggetlen a hívás frekvenciája) alkalmas saját időmérő funkciók megvalósítására. Némi programozási többletmunkát okoz a fent említett másodpercenkénti 18.2 átlagos hívási érték, hiszen ha a program bármiféle korrekció nélkül 18 vagy 19 hívás alapján léptetné a másodperceket, a kijelzett óra erősen sietne, ill. késne. A dilemma feloldására a program a következő módszert alkalmazza.

Alapértelmezés szerint minden 18-dik híváskor ugorjon a rendszer órája egy másodpercet (mielőtt a részletes számításokra sort kerítenénk, szerepeljen itt a 18.2 kerekített érték némileg pontosabban : 18.20648 hívás másodpercenként). Ekkor az ötödik másodperc leteltét a 90. hívással azonosítja a rendszer, holott az ötödik másodperchez 5 * 18.2 = 91 hívás tartozna. Az óra ebben a konstrukcióban tehát kb. 1%-ot sietne. A megoldás minden 90. hívás után az óra visszaléptetése 1 hívásnyit (a következő másodpercet csak 19 hívás után fogja elkönyvelni). Ugyanezen az elven alkalmaztunk a programban még egy korrekciós tényezőt, melynek hatása hozzáadódik az előbb ismertetett korrekcióhoz : minden

31. 1 hívásnyi korrekció (5 * 31 = 155 másodperc) után van még egy hívásnyi korrekció, ami azt jelenti, hogy minden 156. másodperc csak 20 hívás után regisztrálódik.

Van még egy apróbb programozástechnikai fogás a példaprogramban. A konstansok között szerepel egy frissítési paraméter. Jóllehet az óra tartalma 18 (esetenként 19, ill. 20) hívás idejéig nem változik, alkalmazástól függően szükség lehet arra, hogy a meg nem változott órát is újra kijelezzük. Gyakori görgetés például szinte eltüntetné a képernyőről az órát, ha az csak minden aktualizálódáskor frissítődne fel. A frissítési paraméter értéke 1/18 másodpercben a két frissítés közötti időt adja meg. Példánkban így 1/3 másodperc adódik két frissítés között, de ez a paraméter a körülményektől függően megváltoztatható, ill. meg is kell változtatni.

Az óra megjelenítése a BIOS képernyőkezelő funkciókkal (kurzor pozícionálás, karakter kiírás stb.) történik.

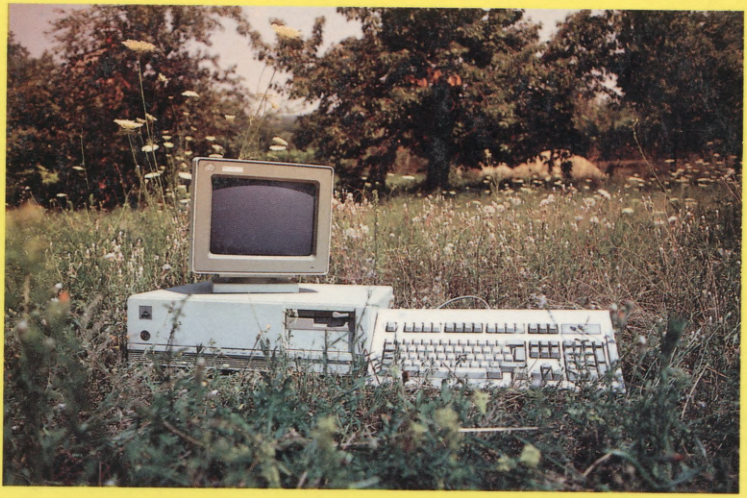
```
;
;
;                              O R A 2 . A S M
;
;   Funkciója: a képernyõ sarkában futó óra megjelenítése a
;              periódikus megszakítás átdefiniálásával.
;
;
;
;-- KONSTANSOK ------------------------------------------------------------

OSZ     = 64                    ; Az óra képernyõpozíciói.
SOR     = 0
ORAFRI  = 6                     ; A frissítés számlálója.
SZIN    = 70h                   ; Az óra színe.

;-- KÓD -------------------------------------------------------------------

kod     segment para 'CODE'     ; A kódszegmens definiálása.

        org 100h                ; A program a PSP után álló elsõ
                                ; címen kezdõdik.

        assume cs:kod, ds:kod, es:kod, ss:kod

                                ; Csak egy, közös szegmens van.

start:  jmp  inic               ; Az inicializáló rutin hívása.

;-- ADATOK ----------------------------------------------------------------

origmsz  equ this dword         ; Az eredeti megszakítás címe :
origmszo dw (?)                 ;     ofszetcím,
origmszs dw (?)                 ;     szegmenscím.

ido      equ this byte          ; Aktuális idõ az alábbiak szerint :
         db " ** "
ora1     db (?)                 ; órák elsõ jegye (ASCII).
ora2     db (?)                 ; órák második jegye (ASCII).
         db ":"
perc1    db (?)                 ; Percek elsõ jegye (ASCII).
perc2    db (?)                 ; Percek második jegye (ASCII).
         db ":"
mperc1   db (?)                 ; Másodpercek elsõ jegye (ASCII).
mperc2   db (?)                 ; Másodpercek második jegye (ASCII).
         db " ** "

nid      db 18                  ; Az idõzítés visszaszámlálója.
friss    db ORAFRI              ; Az órafrissítés számlálója.
korr1    db 5                   ; Korrekciós számlálók.
korr2    db 31                  ;

;-- Az új rezidens periódikus megszakítás ---------------------------------

oramsz   proc far

         jmp  short ujmsz

         db "YY"                ; A program neve.

ujmsz:   push ax                ; A regiszterek elmentése.
         push bx
         push cx
         push dx
```

```
        push di
        push si
        push es
        push ds

        push cs
        pop  ds                 ; CS --> DS.

        dec  friss              ; Az órafrissítés számlálójának
        jne  nemki              ; csökkentése, ha eléri a 0-át,

        mov  friss,ORAFRI       ; újra kell inicializálni.

nemki:  dec ni8                 ; Ha 18 ciklus lefutott, eltelt egy
        je  masodp              ; másodperc --> kijelzés következik.
        jmp vissza              ; Ha nem : újabb ciklus.

masodp: mov ni8,18              ; Ciklusszámláló újrainicializálása.
        dec korri               ;
        jne allitas             ; Kétszeres számláláskorrekció
        mov korri,5             ; alkalmazása :
        inc ni8                 ;
        dec korr2               ; - minden 5. másodperc és
        jne allitas             ; - minden 155. másodperc után
        mov korr2,31            ;
        inc ni8                 ; "szökömásodperc" beiktatása.

allitas: inc mperc2             ; A mperc második jegyének növelése.
        cmp mperc2,":"          ; = 10 (a "0" karakter után ":" jön)?
        jne kiiras              ; Nem : ugrás kiírásra, állítás vége.
        mov mperc2,"0"          ; Igen : a helyiérték nullázása.
        inc mperc1              ; A mperc első jegyének növelése.
        cmp mperc1,"6"          ; = 6 ?
        jne kiiras              ; Nem : ugrás kiírásra,állítás vége.
        mov mperc1,"0"          ; Igen : a helyiérték nullázása.
        inc perc2               ; A percek második jegyének növelése.
        cmp perc2,":"           ; = 10 ?
        jne kiiras              ; Nem : ugrás kiírásra,állítás vége.
        mov perc2,"0"           ; Igen : a helyiérték nullázása.
        inc perc1               ; A percek első jegyének növelése.
        cmp perc1,"6"           ; = 6 ?
        jne kiiras              ; Nem : ugrás kiírásra,állítás vége.
        mov perc1,"0"           ; Igen : a helyiérték nullázása.
        inc ora2                ; Az órák második jegyének növelése.
        cmp ora2,":"            ; = 10 ?
        jne napval              ; Nem : ugrás NAPVÁLTÁS figyelésre.
        mov ora2,"0"            ; Igen : a helyiérték nullázása.
        inc ora1                ; Az órák első jegyének növelése.
        jmp short kiiras

napval: cmp ora2,"4"            ; Az órák második jegye = 4 ?
        jne kiiras              ; Nem : ugrás kiírásra,állítás vége.
        cmp ora1,"2"            ; Az órák első jegye = 2 ?
        jne kiiras              ; Nem : ugrás kiírásra,állítás vége.
        mov ora1,"0"            ; Igen : az órák mindkét jegyét le
        mov ora2,"0"            ;        kell nullázni.

kiiras: mov  ah,15              ; Képernyőoldal lekérdezése funkció.
        int  10h                ; 10(h) megszakítás.
        mov  ah,3               ; Kurzorpozíció lekérdezése funkció.
        int  10h                ; 10(h) megszakítás.
        push dx                 ; Kurzorpozíció mentése a verembe.
```

```
                mov   si,offset ido          ; Az idő karakterlánc címe.
                mov   cx,1                   ; A karakterek egyszer jelennek meg.
                mov   dx,SOR shl 8 + OSZ     ; Az óra képernyőpozíció beállítása.
                mov   bl,SZIN                ; Az óra színe --> BL.
                mov   di,16                  ; 16 karakter kerül megjelenítésre.
kiirasi:        mov   ah,2                   ; Kurzorpozícionálás funkció.
                int   10h                    ; 10(h) megszakítás.
                mov   dh,SOR                 ; A sor újrapozícionálása.
                inc   dl                     ; Az oszlopszám növelése.
                mov   ah,9                   ; Karakter kiírás funkció.
                lodsb                        ; A karakter beolvasása AL-be.
                int   10h                    ; 10(h) megszakítás.
                dec   di                     ; Minden karakter kiírva ?
                jne kiirasi                  ; Ha nem : vissza a következőre.

                pop   dx                     ; Az eredeti kurzorpozíció --> DX.
                mov   ah,2                   ; Kurzorpozícionálás funkció.
                int   10h                    ; 10(h) megszakítás.

vissza:         pop   ds                     ; A regiszterek visszatöltése
                pop   es                     ; a veremről.
                pop   si
                pop   di
                pop   dx
                pop   cx
                pop   bx
                pop   ax
                jmp   cs:[origmsz]           ; Ugrás az eredeti megszakításra.

oramsz          endp

vegcim          equ this byte                ; Az utolsó a tárban maradó bájt.

;— ADATOK ——————————————————————————————————————————————————————————

CR              equ 13                       ; Kocsivissza ASCII kódja.
LF              equ 10                       ; Soremelés ASCII kódja.
KEND            equ "$"                      ; Karakterlánc vége jelzés.

txt_1           db CR,LF,"Az új futóóra megszakítás installálva.",CR,LF
                db "Inaktiválása egy újbóli felhívással történik.",CR,LF,KEND
txt_2           db CR,LF,"Az új futóóra megszakítás inaktiválva.",CR,LF,KEND

;— FŐPROGRAM — a DOS FELULÍRHATJA ——————————————————————————————————

inic            proc near

                mov   ax,351Ch               ; Megszakítás cím lekérdezés funkció.
                int   21h                    ; 21(h) DOS megszakítás.
                                             ; A megsz. rutin címe ES:BX-be kerül.

                cmp   word ptr es:[bx+2],"YY"  ; Annak eldöntése, hogy az
                                               ; ORA2 megsz. rutin aktív-e.

                jne   install                ; Nem : installálás.

                ;— Inaktiválás ——————————————————————————————————————

                mov   dx,es:origmszo         ; A régi megszakítás visszaállítása,
                mov   ax,es:origmszs         ; a címe DS:DX lesz.
                mov   ds,ax                  ;
```

```
        mov  ax,251Ch           ; Megszakítás vektor beáll. funkció.
        int  21h                ; DOS megszakítás hívása.

        mov  ah,49h             ; Memória felszabadítás funkció.
        int  21h                ; 21(h) DOS megszakítás.

        push cs                 ; CS  -->  DS.
        pop  ds

        mov  dx,offset txt_2    ; Üzenet kiírása.
        mov  ah,9               ; Karakterlánc kiírás funkció.
        int  21h                ; 21(h) DOS megszakítás.

        mov  ax,4C00h           ; Vissza a DOS-hoz 0-ás
        int  21h                ; hibakóddal.

        ;— Installálás ——————————————————————————————————————

install: mov  origmszs,es       ; Az eredeti megszakítás eltárolása.
        mov  origmszo,bx

        mov ah,02Ch             ; Az időlekérdezés funkciókódja.
        int 21h                 ; 21(h) megszakítás.
        push cx                 ; CX --> verem.
        mov al,ch               ; órák száma --> AL, a hozzá tartozó
        mov di,offset ora1      ; tárterület kijelolése.
        call binasc             ; ASCII átalakítás.
        pop cx                  ; Verem --> CX.
        mov al,cl               ; Percek száma --> AL, a hozzá tartozó
        mov di,offset perc1     ; tárterület kijelolése.
        call binasc             ; ASCII átalakítás.
        mov al,dh               ; Mpercek száma --> AL,a hozzá tartozó
        mov di,offset mperc1    ; tárterület kijelolése.
        call binasc             ; ASCII átalakítás.

        mov  dx,offset oramsz   ; Az új megszakítás rutin címe
                                ; DS:DX lesz.
        mov  ax,251Ch           ; Megszakítás vektor beáll. funkció.
        int  21h                ; DOS megszakítás hívása.

        mov  dx,offset txt_1    ; Üzenet kiírása.
        mov  ah,9               ; Karakterlánc kiírás funkció.
        int  21h                ; DOS megszakítás hívása.

        mov  dx,offset vegcim   ; A programhoz szükséges paragrafusok
        mov  cl,4               ; (16 bájt) számának kiszámítása.
        shr  dx,cl              ;
        inc  dx
        mov  ax,3100h           ; Vissza a DOS-hoz 0-ás
        int  21h                ; hibakóddal (rezidenssé tétel).

inic    endp

;— BINASC : Binárisról ASCII kódra konvertálás ——————————————
;— BE : AL = az átalakítandó szám
;       DI = az eredmény tárolására szolgáló hely címe.
;— KI : -
;— Regiszterek : CX, AX, DL és FLAGS változnak

binasc  proc near

        mov cl,10               ; Tízes számrendszer előírása.
        mov ah,0                ;
```

```
        div cl                  ; AX / CL.
        or  ax,3030h            ; Az eredményt ASCII-vá alakítása.
        mov [di],ax             ; Az eredmény eltárolása.
        ret                     ; Vissza a hívóhoz.

binasc  endp

;— VÉGE ————————————————————————————————————————————————————————

kod     ends
        end  start
```



NOVOTRADE

IBM PC XT/AT rendszerprogramozás II.