IBM PC/XT

felhasználóknak és programozóknak

3

Pethő Ádám

A ROM BIOS és ami mögötte van

PROFESSZIONÁLIS SZEMÉLYI SZÁMÍTÓGÉPEK

Ádám Pethő

IBM PC/XT

FOR USERS
AND
PROGRAMMERS

3.

ROM BIOS

and the Hardware Elements

Computing Applications and Service Co. ■ Budapest, 1988

Pethő Ádám

IBM PC/XT

FELHASZNÁLÓKNAK
ÉS
PROGRAMOZÓKNAK

3.

A ROM BIOS

és ami mögötte van

Számítástechnika-alkalmazási Vállalat ■ Budapest, 1988

Lektorálta: Radnóty László

Supervised by László Radnóty

ISBN 963 553 133 8

ISBN 963 553 123 0 (1. kötet)
ISBN 963 553 131 1 (2. kötet)
ISBN 963 553 129 X (összkiadás)

Felelős kiadó: Havass Miklós, a Számalk vezérigazgatója
Felelős szerkesztő: Lukács Erzsébet
Műszaki szerkesztő: G. Müller Zsuzsa
Fedélterv: Molnár Zoltán
Megjelent: 29,25 A/5) iv terjedelemben
Széchenyi Nyomda, Győr 88. K–1216
Felelős nyomdavezető: Nagy Iván igazgató

Tartalomjegyzék

Előszó

Az IBM PC/XT felhasználásáról, programozásáról szóló könyvsorozat harmadik kötete a gép és a rajta futó operációs rendszer olyan kérdéseit tárgyalja, melyek az első két kötet olvastán joggal merülnek fel Olvasóinkban. Milyen hardware- és software-eszközökkel valósítják meg az MS-DOS operációs rendszert? Milyen áramkörök, és vezérlésükhöz milyen rutinok állanak a programozó rendelkezésére? A kötet megírásakor e kérdések tárgyalását tűztem ki célul.

Előre kell bocsátanom, hogy csakis a programozó szempontjai vezettek az anyag összegyűjtése és csoportosítása során. Amíg csak lehetséges, programozói gyakorlatomban is kerülöm a hardware közvetlen programozását. Ez bizony nem szerencsés felfogás egy mikrogépes programozó számára, mivel e kis gépek operációs rendszere sokszor nem ad lehetőséget a közvetlen programozás elkerülésére. Fel kell azonban hívnom az Olvasó figyelmét, hogy tanuljon meg pontos különbséget tenni a sarlatánság és a programozás, a programozgatás és a profi szintű munka között. Sajnos, a mikrogépek világában egyre szélesebb tere nyílik az amatőröknek (az amatőr szót itt rossz értelemben használva). Sokak előtt nyílt meg a számítástechnika, és ebben a rendkívül bonyolult és igényes szakmában egyre több a félművelt ember. Kis túlzással azt mondhatjuk, hogy aki már képes lemezt formázni, vagy (hogy ilyen messzire mégse menjünk) ismer olyan trükköket, hogy miképpen kell a képet kimásolni az IBM PC/XT monitoráról, az már azt hiszi, hogy ért a számítógépekhez. Elterjedtek olyan nézetek is, hogy néhány nap, esetleg néhány hét alatt már meg lehet tanulni "számítógépezni".

Ebben a szemléletben bizony főbűnösök az olyan programozási nyelvek, mint az (egyébként zseniális) Basic, az olyan gépek, mint az (egyébként nagyszerű) Commodore-64, mert a régebbi, sok tanulást, nagy lexikai, elméleti felkészültséget igénylő gépek és rendszerek helyett gyakorlatilag tudás nélkül "magukhoz engedik" a felhasználót. Ennek természetesen nagyon sok előnye van, de ma is tudomásul kell vennünk azt, hogy a számítógépek világa, felhasználásuk és programozásuk egész embert kíván, és sokéves komoly felkészülést. Egy igazi szakember soha nem hagyja abba a tanulást, és csak újabb és újabb problémákat lát maga előtt, melyek kezelését, megoldásait meg kell tanulnia.

Pedig a komoly munka pedig nem trükkök elsajátításából és alkalmazásából, nem a nehézségek megkerüléséből áll! Alapos ismeretek szükségesek hozzá, és nagyon-nagyon precízen átgondolt tervezés. Ebben természetesen szerepet játszanak a trükkök is; ezek nélkül bizony szegényesebb lenne a programozó tárháza. Nem

nélkülözhető azonban a mély elméleti és gyakorlati tudás, a kifejezett számítógépes és programozói gyakorlat, az egyre szebb, tisztább munkára való törekvés.

Az IBM PC/XT és legelterjedtebb programozási nyelvei sokkal bonyolultabbak, mint a fent "megrótt" gép, illetve nyelv. Sajnos itt is nagy mértékben elterjedt a félműveltség. Ezt a gépet is sokan akarják úgy programozni, hogy "...és akkor a 427 címre kiírsz egy 659-et, meg kiadsz egy STI utasítást..." Azt remélem, hogy könyveim (a bennük leírt ismeretanyag mellett) ahhoz is hozzásegítik az Olvasókat, hogy amennyire lehet, szépen, eredményesen programozzanak, és kerüljék az előbb elrettentő példaként citált gyakorlatot. Jóllehet éppen ebben a kötetben adom meg az ilyesmihez szükséges ismereteket, bízom abban, hogy a teljesebb áttekintés (melyet könyveim mellett a felsorolt szakirodalom és még rengeteg egyéb könyv adhat) nem annyira a "csirkefogás", mint inkább a korrekt munka felé irányítja az Olvasót. Gondoljuk meg, hogy micsoda nagyszerű programokat írtak erre a gépre a különböző fordítóprogramoktól kezdve a szakértői rendszerekig vagy a wordprocesszorokig. Ha valamennyire is versenyképesek akarunk lenni, akkor tudomásul kell vennünk, hogy ezek a programok nem "varázslás", nem trükkök eredményei. A kötetben leírt ismeretek szükségesek mindezek megvalósításához, de csak ujjgyakorlat szintjén állnak; az igazi munka ezek felett áll.

Kérek tehát minden komoly szándékú embert, hogy e könyvemet és a benne leírt dolgokat magasabbrendű céljai megvalósításában eszköznek tekintse és ne abban élje ki magát, hogy beletúr a rendszer lelkivilágába. Ha lehet, kerülje a hardware elemek közvetlen programozását (különös tekintettel a disk- és diskette vezérlőkre). Ha ez mégis szükséges (pl. a soros vonal esetében), akkor nagyon gondosan mérlegelje a rá váró feladatot.

(i) Irodalomjegyzék

A könyv megírása során természetesen leggyakrabban az

(1) IBM Software
Hardware Technical Refence Manual

c. könyv különféle verzióit forgattam. Ez a könyv függelékként tartalmazza a ROM BIOS listáját, így a leghasznosabb ismereteket nyújtja a programozónak. Több verziója ismeretes, mert mind a PC-hez, mind az XT-hez, mind az AT-hez külön megírták; de van eltérés az egyes gépek különböző generációi között is.

E kötet megírása során segített leginkább

(2) Peter Norton:
Programmer's Guide to the IBM PC
(Microsoft Press, 1985)

c. könyve, amely rengeteg elvi ismeret mellett tartalmazza az összes ROM BIOS funkció részletes leírását. Sajnos, ez a könyv egyáltalán nem mutat túl a ROM BIOS funkciók által megvalósított szolgáltatásokon, bár a gép hardware elemei általában többet tudnak, mint amennyit a rendszer megvalósít, kihasznál. Az ilyen lehetőségek legalább vázlatos megismerésében hasznos

(3) David J. Bradley:
Assembly Langauge Programming of the IBM PC
(Prentice Hall, 1984)

c. kitűnő munkája. Ebből tanulhatunk meg leginkább muzsikálni, soros vonalat kezelni, lebegőpontos számokkal számolni, és még sok-sok apró szépséget. Ugyanakkor (bár nem ismerteti teljes alapossággal a hardware közvetlen programozási lehetőségeit) sokat javít olvasóinak szemléletén, programozási stílusán.

(ii) Kinek szánjuk ezt a könyvet?

Olyan "mindenre elszánt" programozóknak ajánljuk, akik a programozásról szóló elméleti és gyakorlati könyvek tételeit, a lefektetett elveket tagadva lemondanak a (többé-kevésbé) gépfüggetlen programozásról. Szigorúan csak az IBM PC/XT-re (esetleg AT-re) akarnak olyan programokat írni, amelyek kihasználják e számítógépek speciális adottságainak jó részét.

Ugyancsak haszonnal forgathatják azok, akik mélyebbre akarnak látni a gép működésébe, az operációs rendszerbe, illetve az alatta meghúzódó finomságokba. Ilyen szempontból ez a kötet hasznos lehet minden programozó, program- és rendszerszervező, sőt felhasználó számára. Már a második kötetben hangoztattam azt a nézetemet, hogy a szervezőknek nagyon alaposan kell ismerniük a rendszer, a driverek és a hardware **lehetőségeit** (és nem azok pontos felhasználását). Ez azért lényeges, mert munkájuk során nekik kell a leginkább tisztában lenniük azzal, mikor milyen terheket raknak a programozók vállára; mikor lépik át a rendszer korlátait (ami semmiképpen nem hasznos, de néha szükséges). Ha átlépik e határokat, és a ROM BIOS, illetve a hardware közvetlen elérésére szorítják a programozókat, akkor pontosan fel kell mérniük, hogy mit nyernek, és mit vesztenek ez-

zel. Különösen élesen vetődik ez fel akkor, ha még a ROM BIOS drivereinek korlátait is át kell lépni, és a hardware közvetlen programozását végezni. Ez általában olyan mértékig rontja a program hordozhatóságát, ami a klasszikus programozási elvek szerint már nem engedhető meg.

(iii) A könyv szerkezetéről

Az egyes fejezetek természetesen az egyes hardware-elemeket tárgyalják. A fejezetek bevezető részében általában a gép vizsgált elemének általános leírását olvashatjuk. Ezután következik az adott hardware és közvetlen programozásának ismertetése, az ROM BIOS ide tartozó részének általános áttekintése, és a rutinhívások részletes leírása. Több fejezet végén példaprogramokat találunk, amelyek illusztrálják a korábban leírt dolgokat. A példaprogramok során egyáltalán nem mutatjuk be a nyilvánvaló lehetőségeket, inkább a nehezebb (és néha hasznosabb) megoldásokra összpontosítunk. A legtöbb fejezethez kifejlesztettem egy-egy hasznos include-file-t, amelyben az adott témakörben felmerülő konstansokat és makrókat definiáltam. Ezeket azonban helyszűke miatt általában nem tudom közölni. Felhívom a figyelmet, hogy ezek a file-ok és a példaprogramok lemezen is megkaphatók a Kiadónál.

A fejezetek ilyen tagolása azonban nem mindig volt megvalósítható; egyes elemek vezérlése nem megy ROM BIOS rutinból, míg másoké (pl. a nyomtató) egyáltalán nem szokásos közvetlen programozással. Van, ahol elenyésző helyet igényelnek csak a ROM BIOS lehetőségei a közvetlen programozási lehetőségek vizsgálata mellett (pl. az aszinkron vonal).

A könyv utolsó két fejezete olyan ismereteket tárgyal, amelyek nem illeszthetők be közvetlenül egyik alapvető hardware elem ismertetésébe sem. Ne feledjük azonban, hogy itt is lényeges és fontos ismereteket találhatunk (pl. az interrupt vezérlő ismertetése), amelyek nélkül számos feladatot egyszerűen nem oldhatunk meg.

(iv) Tematikai megjegyzések

Nagyon nehéz feladat megvonni azokat a határokat, ameddig egy ilyen könyv az ismeretek taglalásában elmehet, vagy ameddig el kell mennie. Különösen nehéz feladat ez éppen ebben a témakörben.

A fő szempont a könyv címében is megfogalmazott cél: a gép bemutatása felhasználók és programozók számára. Igyekeztem a rendelkezésemre álló irodalomból mindazon ismereteket összefoglalni, amelyek programozói szempontból érdekesek. Ezek közé tartozik természetesen minden MS-DOS és ROM BIOS hívás, tehát az operációs rendszer szabályos "kapui". Vannak azonban olyan pontok, ahol a jogos programozói és felhasználó igények messze túlmutatnak a rendszer kínálta szabályos lehetőségeken. Ezek a területek elsősorban az aszinkron vonali kommunikáció vezérlése, ahol (mint ez az erről szóló fejezetből kitűnik) a "szabályos" utak nem teszik lehetővé az értelmes felhasználást, vagy olyan pontok, ahol a rendszer nem is kínál semmilyen lehetőséget a kívánt cél elérésére (pl. a hangszóró használata). Sokan gyűlölik a daloló-muzsikáló programokat, és szempontjaikat feltétlenül figyelembe kell venni. De számos olyan program van, amelyben (pl. hibajelzésre) hasznos igénybe venni a hangszórót. Amíg a játékprogramoknak komoly nemzetközi piaca van, a fantáziadús és ügyeskezű programozók helyesen teszik, ha igyekeznek ennek lehetőségeit kihasználni; a játékok megírásához pedig fontos a változatos hangeffektusok kihasználása.

Gyakorlati szempontból (és éppen a profi programozás területén) óriási jelentősége van a floppy-diskek sajátkezű vezérlésének, mert ez ad kézenfekvő lehetőséget programjaink másolás elleni védelmére. Ez az a pont azonban, ahol megfelelő tapasztalatok hiányában kénytelen vagyok visszavonulni. Ismertetek ugyan néhány triviális lehetőséget, melyek azonban régóta közismertek a védelmeket építő és leromboló szakemberek körében; ezen a téren csak egészen laikusoknak tudok újat mondani.

Az egyéb (háttérben levő) elektronikai eszközök, áramkörök felhasználásával csak szűk terjedelemben tudok foglalkozni, mert ezek már túlmutatnak a hétköznapi programozó és felhasználó igényein és lehetőségein.

A könyv írása során vezérelvként követtem, hogy semmit nem írok le "szájhagyomány" útján szerzett ismeretekből. Amire csak lehetőségem nyílott, azt többször is kipróbáltam. Sajnos, erre nem minden esetben volt módom; ezekben az esetekben valamilyen hivatalos helyről származó leírást követek. E téren pedig még a nagyszerű és igen precíz Peter Norton könyveit sem tekinthetem hivatalosnak, csak a Microsoft Corporation által kiadott "Hardware Technical Reference Manual" megfelelő verzióit, és az integrált áramköröket gyártó cégek (Intel, Motorola és International Semiconductor) katalógusait.

Természetes, hogy a kipróbálás sem adhat százszázalékos biztonságot, mert apró eltérések még két eredeti gép között is lehetnek, nem beszélve a hasonmás gépekről.

(v) Terminológiai megjegyzések

A könyvben a numerikus konstansok általában hexadecimálisak annak ellenére, hogy a szakirodalom (sokszor elég következetlenül) váltogatja a számrendszereket. Például az Microsoft könyveiben hexadecimálisak az összes címek (természetesen); hexadecimálisan adja meg az interruptok sorszámát és az MS-DOS funkciók kódját, de az MS-DOS hibakódokat decimálisan (?). A ROM BIOS interruptok sorszáma ugyan hexadecimális, de a funkciók sorszáma decimális. E téren igyekeztem következetesebben eljárni, és mindent tizenhatos számrendszerben megadni; jóérzésű programozó ezeket az értékeket amúgyis konstansként definiálja és ha lehet, az INCLUDE utasítással építi be a programjába...

A hardware, mint tudjuk, eléggé hardware-függő. Az itt leírt ismeretek jó része igaz az IBM PC-re, egy másik része az IBM XT-re, megint más része pedig az IBM AT-re. Ami PC-re igaz, az nagy vonalakban elmondható a többi típusról is. Ezért csak azokat a bekezdéseket jelöljük meg külön, amelyek specifikusan csak az egyik típusra igazak. A

Csak IBM PC! vagy **Csak IBM XT!** vagy **Csak IBM AT!**

jelzés mutatja azokat a részeket, amelyek csak az adott géptípusra igazak.

Az első két kötettel szemben tehát ez a harmadik sokkal kevésbé általános. Ha az IBM AT-t az MS-DOS operációs rendszerrel használjuk, akkor az első két kötetben foglaltakat gyakorlatilag változatlan formában használhatjuk fel; az itteni ismeretek közül "egy az egyben" csakis a ROM BIOS funkciók használhatók, melyek hívási szinten felülről kompatibilisek az IBM PC és XT megfelelő szolgáltatásaival (jóllehet az AT ROM BIOS-a valamivel többre képes).

Ugyanakkor le kell szögezni azt is, hogy amit itt leírtam, egyáltalán **nem köteles** futni az ún. IBM-kompatibilis gépeken. Könnyen előfordulhat ugyanis, hogy ezek a másolatok nemhogy hardware, de még ROM BIOS szinten sem kompatibilisek az eredetivel. Hiszen az eredeti és a hasonmás gépek között a különbség éppen abban áll, hogy egyes hardware elemeket módosítottak bennük. Az eltérő áramköri elem vagy többet, vagy kevesebbet, vagy éppen egészen mást tud, mint az eredetileg alkalmazott elem. Esetleg programozása is teljesen eltérő; ennek következményeit természetesen az operációs rendszer "magján", a ROM BIOS programjaiban is láthatjuk. Lehet, hogy gépünk ROM BIOS-a egészen más, ezért nem minden funkció működik, és nem biztos, hogy pontosan ugyanúgy.

Lássunk egy nyilvánvaló példát! Van lehetőség arra, hogy a képernyőn levő képet fel vagy le mozgassuk egy vagy több sorral. Ha az egész ernyőt akarjuk törölni, akkor (hiszi a jámbor programozó) éppen huszonöt sorral kell mozgatni az ernyőt. A ROM BIOS leírása szerint azonban ez esetben azt kell megadni, hogy nulla sorral akarunk léptetni. Az érdekes különbség a következő: az eredeti IBM PC-n, vagy egyes PROPER 16 gépeken huszonöt soros rollozás megadása esetén az első törlés kifogástalan, a második esetén már belép néhány színes sor az ernyő alján, a harmadik kísérlet még több színes sort eredményez (tehát a képernyőterületre memóriapiszok lépett be). Ugyanakkor a Magyarországon elterjedt CONTROL-gépeken kifogástalanul működik ez a művelet akkor is, ha az előírt Ø helyett a téves huszonötöt adjuk meg. Tehát az IBM ROM BIOS egyik apró "hibáját" kijavították (a hiba azért van idézőjelben, mert a huszonöt megadása, bár logikus, mégis specifikálatlan paraméter).

Hardware-közelben folytatván vizsgálódásainkat, számos olyan kifejezéssel találkozunk, amelyeknek magyarra való átültetése enyhén szólva konfúz.

Minden perifériális egység, amelyet a gép kezel, egy sajátos felépítésű elektronikus elem, a kontroller segítségével kapcsolódik a számítógéphez. Az operációs rendszernek minden egyes kontrollerhez van egy olyan programja, mely közvetlenül, "hardware-szinten" kommunikál az adott perifériakontrollerrel. Ezt a programot drivernek hívják. Ha ezekhez még hozzávesszük magát a perifériális egységet (melyet angolul gyakran neveznek "drive"-nak), akkor e szavak magyarra való átültetésekor kitör a teljes pánik. A drive-ot sokszor fordítják meghajtónak; de akkor mi a driver? A kontrollert természetesen nevezhetjük vezérlőnek, de voltaképpen a driver is vezérlő. Ismerkedjünk meg most vázlatosan ezeknek az elemeknek a szerepével, és találjunk nekik értelmes neveket.

A perifériális egység (a lemezegység, a monitor, a klaviatúra, a nyomtató stb.) az a berendezés, amely fizikailag végrehajtja a kívánt input-output műveleteket. A perfiériák egy sokeres kábel segítségével kapcsolódnak a kontrollerekhez. A kontroller "kétarcú" elektronikus berendezés. Egyik arcát a perifériá felé fordítja, és képes vele a kábel különböző csatlakozásain a perfiéria igényeinek megfelelően kommunikálni, azaz a kábel egyes vonalain a megkívánt pillanatban a szükséges jelszinteket kialakítani, illetve a periféfia által beadott jeleket fogadni és értelmezni. Befelé, a processzor irányába "számítógépes" arcát mutatja: hogy a processzorral (pontosabban a processzor által végrehajtott programmal) kommunikálni tudjon, olyan elemei vannak, amelyek megfelelnek egy számítógép belső

felépítésének. Magyarul a kontrollernek van néhány regisztere, melyeket a program írhat és olvashat, ezáltal vezérelve a perifériakontrollert, vagy fogadva a kontroller által a perifériáról olvasott adatokat.

Tehát az adatátvitel, adatrögzítés eddig megismert két fontos eleme elektronikus jellegű berendezés. A harmadik (és az erre épülő negyedik) szint már nem hardware, hanem software, azaz nem elektronika, hanem program. A driver az a program, amely a kontroller közvetlen vezérlését végzi. Bizonyos perifériális egységek kezelésére egy egyszerű driver tökéletesen elegendő is lehet; ha a periféria "egyszerű" (például nyomtató, monitor vagy klaviatúra), akkor sok mindent nem is kell tudnia; esetleg elég lehet, ha fogad vagy bead egy-egy karaktert.

Vannak azonban sokkal bonyolultabb perifériák, elsősorban a lemezek. A központi tár és a lemezek közötti adatmozgás során nem elég azt megmondani, hogy mit kell kiírni a lemezre, hanem azt is, hogy hová, a lemez mely pontjára. A file-orientált perifériák kezelése a fizikai driver szintjén túl igényel még egy fontos lépcsőt. Ha programunkból a lemezre adatokat akarunk írni, akkor szinte sohasem az a célunk, hogy a **lemez** egy részét módosítsuk, hanem az, hogy a lemezen lévő **file**-ok valamelyikébe írjuk be az adatainkat. Tehát a kényelmes felhasználáshoz kell egy olyan programcsomag, amely a file-ok írására vonatkozó kéréseinket lefordítja a driver nyelvére, vagyis amely a lemez adminisztrációs területeinek (az MS-DOS-ban a directory-k és a FAT) felhasználásával megadja, hogy amikor mi egy file valamely pontjára akarunk írni, akkor az a fizikai lemez mely pontjának a felülírásával történik meg. Ezt a programcsomagot angol nyelven "File Service Routines" néven illetik.

Mindezek alapján a továbbiakban a kissé következetlen magyar kifejezések helyett hívjuk a fenti három (négy) fogalmat eredeti nevén: a perifériális egységet monitornak, klaviatúrának, drive-nak (vagy lemezegységnek); a kontrollert kontrollernek, a drivert drivernek. Az ismeretek tárgyalásának ezen a szintjén nemigen hivatkozunk a File Service Routines-ra, de ha igen, akkor nevezzük őt így.

(vi) Mentegetőzés

Ide kívánkozik a könyvek, programok elejéről sohasem hiányzó néhány megjegyzés azokról a garanciákról, amelyeket a szerző az általa elkövetett munkáért vállal. Négykötetes munkám során éppen itt lett volna a legfontosabb, hogy eredeti gépen próbálhassak ki minden egyes lehetőséget. A sors azonban úgy hozta,

hogy ezúttal nem áll rendelkezésemre eredeti gép. Nem volt alkalmam ilyenen kipróbálni sem a ROM BIOS felhasználásának, sem a hardware közvetlen programozásának lehetőségeit. Számos esetben lett volna szükségem két gépre, ráadásul úgy, hogy az egyiket "tönkretehessem", elpusztíthassam a Winchester tartalmát stb. Sajnos, erre sem volt lehetőség. Ezért nem próbálhattam ki például a lemez formázását; nem ronthatom el azt a Winchestert, amelyen dolgozom.

Ezek alapján kénytelen vagyok kijelenteni, hogy a könyvben szereplő ismeretek egyikéért sem tudok felelősséget vállalni. Kipróbálni nagyon kevés dolgot tudtam; amit pedig ilyen vagy olyan eredménnyel kipróbálhattam, azok esetében ki vagyok téve annak, hogy a hasonmás gép eltérő hardware (és ebből adódóan eltérő software) felépítése miatt nem pontosan úgy viselkedik, ahogy azt a hivatalos leírások alapján várhatnánk.

I. Az IBM PC/XT hardware elemeinek áttekintése

A gép belsejébe vetett első pillantásra látható, hogy dobozának alján (az alapkártyán) sok áramkör van. A rengeteg áramkör között itt kapott helyet a processzor, valamint az alapmemória (ez eredeti IBM PC/XT-n legalább 16, legfeljebb 64 Kb); általában úgynevezett statikus RAM.

> Megjegyzés: az újabb gépeken sokszor mind a 640Kb memória az alapkártyán van...

Szintén az alapkártyán található az IBM PC/XT ROM területe. Ez a memória tartalmazza az alapvető drivereket, valamint egy beégetett Basic interpretert (mely a hasonmás gépeken általában hiányzik). Ezzel a kiépítéssel (kiegészítve persze egy képernyővezérlő áramkörrel) azonban csak ez a Basic használható, ez is inkább csak elméletben (egy közepes méretű program már teljesen betöltheti). A gyakorlatban a gépet legalább 256 Kb RAM területtel szállítják. Korszerű programok általában legalább 512 Kb hosszúságú memóriában érzik csak igazán jól magukat. A fontos elemek közül az alapkártyán van még a klaviatúravezérlő áramkör.

Számos további elem szükséges a gép működtetéséhez; ezek az alapkártyára merőlegesen, függőlegesen elhelyezkedő bővítőkártyákon vannak. Ezek további memóriát, valamint perifériavezérlő áramköröket tartalmaznak. A bővítőkártyák egy része nélkülözhetetlen. Ilyen a memóriabővítés (legfeljebb 640 Kb-ig), amely nélkül a gép gyakorlatilag használhatatlan, valamint a képernyővezérlő áramkör, amely azért nincs az alapkártyán, hogy tetszésünk szerint választhassunk színes vagy monokromatikus monitort. "Nélkülözhető" elemek a nyomtatóvezérlő, a soros vonali csatoló, és számos egyéb szabványos vagy utólag létrehozott elem.

Tekintsük most át azokat a legfontosabb áramköri elemeket, amelyek a programozó számára érdekesek lehetnek! A processzor, mint a legfontosabb programozható egység, az első kötet tárgya volt. Az alapkártyán helyezkedik el egy sereg egyéb elem, melyek igen fontosak a gép működése szempontjából, de nem feltétlenül kell személyesen is ismernünk őket. Nélkülözhetetlen például a 8284 típusjelű "Clock Generator", a gép órajelét adó áramkör; a 8288-as "Bus Controller", amely a gép belső buszának vezérlésére hivatott. Ezekkel azonban sohasem lesz dolgunk: ennek kezelése nem éppen programozói vagy felhasználói feladat.

Már magán az alapkártyán is több olyan fontos beépített elemet találunk, amely programozható, azaz amely bizonyos önálló

tevékenység elvégzésére képes, és felhasználói szempontból is nagy jelentőségű.

Első helyen emlékezzünk meg az Intel 8087 típusjelű lebegőpontos (szerencsésebb nevén aritmetikai) koprocesszorról, amely sajátos formátumú egyszeres és kétszeres pontosságú lebegőpontos számokkal tud igen sokféle műveletet elvégezni. A koprocesszor ára elég borsos, de valószínűleg megéri. Használata lehetséges mind assembly, mind egyéb programozási nyelven írott programokban.

A 8255 típusjelű programozható perifériális interface áramkör (PPI, Programmable Peripheral Interface) legfontosabb feladata a klaviatúráról érkező karakterek fogadása. Nagyon hasznos még a 8253 időzítő-számláló (Timer/Counter), a 8259 típusjelű interrupt vezérlő (Interrupt Controller) és a 8237 "Direct Memory Access Controller", a gyors adattovábbítást a processzor igénybevétele nélkül végző input/output áramkör.

A gép további fontos részei már a bővítőkártyákon kaptak helyet. A "kiszorult" elemek egy része azért került ide, hogy cserélhető legyen, mások pedig azért, mert nincs rájuk feltétlenül szükség, és nem akarták őket rákényszeríteni a vevőkre.

A hivatalos bővítési lehetőségek a következők: fekete-fehér vagy színes monitorvezérlő, floppy-disk vezérlő, Winchester vezérlő, aszinkron kommunikációs adapter, párhuzamos nyomtató vezérlő, valamint játékvezérlő kártya (melyhez a közkedvelt botkormány csatlakoztatható). Igen fontos lehet még a Streamer, a mágnesszalagegység mikrogépes változata, szintén külön bővítőkártyával.

Sok olyan eszköz van, amely a gép hivatalos bővítési lehetőségein túl, célhardware-ek bekapcsolására szolgál. Napjainkban egyre elterjedtebb az egér. Használható a soros vonalon át is, de létezik olyan egér, amely külön kártyával működik (egy ilyennel ismerkedhetünk meg a könyvben). Ismeretes még egy halom egyéb eszköz, például programvédelmet biztosító, vagy éppen védett program másolását lehetővé tevő kártya, többféle analóg vezérlő és a többi. Ezek tárgyalása azonban meghaladja könyvünk kereteit.

Ismerkedjünk meg a bővítési lehetőségekkel és elnevezésükkel részletesebben is! Legfontosabbak talán a képernyővezérlő kártyák, amelyeket a Motorola cég 6845 jelű CRT (Cathod Ray Tube Controller) áramkörére építettek. A legelterjedtebb kártyák: a monokróm képernyő adapter, amely tartalmaz egy párhuzamos nyomtatóvezérlőt is, a színes adapter (melyet CGA [Color Graphics Adapter] vagy RGB [Red-Green-Blue] adapternek is szokás nevezni), az EGA (Enhanced Graphics Adapter) és a Hercules kártya (a monokróm grafikus adapter).

Ha nem a monokróm adaptert használjuk, akkor külön kártya szükséges a nyomtató kezeléséhez, a PPA (Paralell Printer Adapter).

Igen fontos a floppy-disk kontrollert tartalmazó kártya. Ennek legfontosabb eleme a Nippon Electric Company által kifejlesztett és világszerte szabványként tisztelt floppy-vezérlő áramkörrel kompatibilis vezérlőáramkör, melyet egyszerűen NEC controller-nek szoktak nevezni. Az IBM PC/XT-n használt kártya elvben két beépített és két külső (tehát a gép dobozán kívül elhelyezett) floppy-egység vezérlésére alkalmas.

A Winchester kezelésére külön kártya szolgál. Ennek felépítéséről szinte semmit nem írhatunk, hiszen minden gyártó cég más-más felépítésű kártyát állít elő. Ezen a kártyán a lemez fizikai kezeléséhez szükséges áramkörökön kívül egy nagyobb méretű ROM területet is találunk; ez a memória tartalmazza azokat a programokat, amelyek a Winchester belépésével válnak szükségessé: a Winchester drivere és egy új rendszerbetöltő program, amely nem csak az A: drive-ról, hanem a Winchesterről is képes betölteni az operációs rendszert.

A soros vonali kártya alapját az International Semiconductor cég 8250 típusjelű aszinkron csatoló (ACE, Asynchronous Communication Element) áramköre adja.

Az IBM XT gépekhez akad még egy-két kiegészítő elem. Ilyen a BSC és az SDLC adapter, amelyek igen hasznosak lehetnek, mert két elterjedt adatátviteli (hálózati) szabvány szerint képesek más gépekkel való kommunikációra: az elsősorban IBM nagygépeken alkalmazott Binary Synchronous Communication, valamint a Synchronous Data Link Communication elnevezésű szabványban ismertetett adatkapcsolatot valósítják meg. Ezekkel (főként terjedelmi okokból) nem foglalkozunk.

A fent felsorolt elemeket rendre megadott I/O-portokra kapcsolták (mindegyikhez több port tartozik). Az elemek közvetlen programozását tehát a processzor IN és OUT utasításai segítségével oldhatjuk meg. Vannak esetek, amikor valóban ilyen "szigorú" eszközhöz kell folyamodnunk (ilyen eset például a hanggenerálás). Az esetek nagyobb részében azonban a BIOS rutinjai elvégzik a kényes munkát, nekünk csak a megfelelő paramétereket kell megadni, és meghívni a megfelelő rutint.

Az alábbiakban először rövid áttekintést adunk az elemekről és szerepükről, a további fejezetekben pedig részletezzük a szükséges tudnivalókat. Felhívjuk a figyelmet arra, hogy az itt felsorolt I/O portok címei, amelyeken át az adott áramkör programozható, nem teljesen pontosak. A hivatalos leírásokban tudniilik nem a valóban használt két-három portcímet adják meg, hanem egy (általában tizenhat címes) tartományt, amelyben a va-

lóban használt I/O portok vannak. Ezért a tartomány a legtöbbször szélesebb a leírásokban, mint a valóságban. szükséges lenne. Egy kártya portjait a címvonalak segítségével címezzük meg, és a gyártók az alsó néhány bit felett rendelkeznek (míg a magasabb bitek szabják meg, hogy a portok címterében hová is kerül a kártya). Nos, a belső címvonalak közül nem szükséges mindegyiket bekötni. A nemlétező portokra való kiírás egyébként nem okoz bajt.

I.1. Az IBM PC/XT programozható elemei

Ebben a fejezetben kissé részletesebb és rendszeresebb ismerkedésre nyílik lehetőségünk a fent név szerint megismert fontos elemekkel. Az egyes elemek neve után funkciójukat, a rendszerben betöltött szerepüket olvashatjuk, a hozzájuk rendelt portcím tartománnyal együtt. A részletes ismertetés a megfelelő fejezetben található. Az alábbiakban elsősorban áramkörökre hivatkozunk, de a programozási gyakorlatban egy perifériacsatoló és az azt vezérlő áramkör eléggé egybemosódik. Lesz tehát olyan logikai elem, amelyre a lényegét adó áramkör nevével hivatkozunk, és lesz olyan, ahol a vezérlőkártya nevét írjuk (nem lévén benne jellemző, nagy integráltságú áramkör).

- 8259 ITC - az interrupt vezérlő 8 csatornája a kiszolgálást kérő hardware-eszközök és a processzor közötti "tolmács" szerepét játssza. Az interruptot kérő eszköz az interrupt vezérlőhöz fordul, amely értesíti a processzort, és továbbítja az interrupt szint számát. Alapkiépítésben egy interrupt vezérlő van a gépben, amelynek nyolc csatornája közül hetet használnak ki. Az interrupt vezérlő portjai: 20H - 21H. Leírását lásd: "Egyéb tudnivalók a hardware-ről" fejezet.
- 8253 T/C - háromcsatornás számláló-időzítő, melynek 0. csatornája szolgál a gép software-órájaként, 1. csatornája a dinamikus RAM frissítését vezérli, a 2. csatorna pedig a hangszóróhoz van kötve. A Timer/Counter portjai: 40H - 43H. Leírását lásd: "Egyéb tudnivalók a hardware-ről" fejezet.
- 8255 PPI - háromcsatornás párhuzamos átvitelre képes I/O vezérlő. Az IBM PC/XT tervezői a PPI csatornáit (vagy ezek bitjeit) jelölték ki a klaviatúráról érkező karakterek fogadására, a hangszóró, az esetleges kazettás magnó motorja (csak IBM PC), és belső tevékenységek vezérlésére. A PPI portjai: 60H - 63H. Leírását lásd: "Egyéb tudnivalók a hardware-ről" fejezet.

- 8237 DMA - a "Direct Memory Access" áramkör a lemezek és a memória közötti gyors adatátvitelre szolgál. Jellemző rá, hogy amint működésbe lép, azonnal leállítja a processzort, és egy menetben több byte-ot olvashat be a memóriába, vagy írhat ki onnan. A DMA portjai: 00H-0FH, valamint 80H-83H. Leírását lásd: "Egyéb tudnivalók a hardware-ről" fejezet.
- INS 8250 ACE - az International Semiconductor cég által kifejlesztett aszinkron vonali vezérlő áramkör, mely a gép soros ki- és bemenetét vezérli. Ez a "kapu" (RS232-nek is szokták nevezni) lehetővé teszi perifériális egységek (pl. egy második terminál vagy egy soros vonalon meghajtható nyomtató) csatlakoztatását. Másik fontos felhasználási területe gép-gép kapcsolat megteremtése: különböző gépeken futó programok közötti kommunikáció biztosítása. Az aszinkron adapter külön kártyán van. Az ACE portjai: 3F8H - 3FFH.
  Csak IBM XT !
  Van lehetőség egy másik kommunikációs adapter beépítésére is. Ekkor a másodlagos adapter portjai:2F8H - 2FFH.
  Leírását lásd az "Aszinkron vonali adapter" fejezetben.
- Motorola 6845 - képernyővezérlő áramkör, amely a memóriában kijelölt képernyőterület tartalmát megjeleníti gépünk monitorán (a Motorola cég gyártmánya). Ez az integrált áramkör mind a színes, mind a fekete-fehér adapter alapja. Külön kártyán helyezkedik el. A monokróm képernyővezérlő bővítőn egy párhuzamos nyomtatóvezérlőt is találunk; színes adapter használata esetén a párhuzamos nyomtató kezeléséhez nyomtatóvezérlő kártyára van szükségünk. A 6845 áramkörnek tizennyolc belső regisztere van, melyeket indexezéssel, jóval kevesebb I/O port használatával érhetünk el. A vezérlőhöz rendelt portok címe monokróm adapter esetén 3B0H - 3BFH (ebből három a nyomtatóvezérlő áramkör programozására szolgál), ha színes adaptert használunk, a portok címe 3D0H - 3DFH. Leírása: "A képernyő" fejezetben.
- Parallel Printer Adapter kártya - a párhuzamos nyomtató vezérlője (amely nem egy ilyen "okos" áramkörre épül, csak egynéhány logikai kaput tartalmazó kártya) elsősorban arra szolgál, hogy egy nyomtatót vezérelhessünk vele egy párhuzamos porton át. Az, hogy a port párhuzamos, csak annyit jelent, hogy nem egy vezetéken, bitenként továbbítja a kiküldendő karaktert, hanem annyi vezetéke van, ahány bit szélességű a kiküldendő karakter. A párhuzamos adapter külön kártyán van. A közvetlen vezérlésére szolgáló portok: 378H - 37FH. Ha a párhuzamos nyomtató vezérlő a fekete-fehér adapteren van, akkor a programozására használható portok sorszáma: 3BCH - 3BEH. Leírását lásd: "A nyomtató" fejezetben.

- Diskette vezérlő - alapja egy külön kártyán elhelyezett NEC µPD765 kompatibilis áramkör, amely egyidejűleg négy floppy-disk egység vezérlésére képes. Lehetővé teszi a diskette különféle paramétereinek eléggé szabad, rugalmas meghatározását. A diskette közvetlen kezelésére szolgáló portok címe 3F0H - 3F7H. Leírása: "Lemezkezelés" fejezet.

Csak IBM XT !

- Winchester vezérlő - külön kártyán helyezkedik el. Elvben csak IBM XT-n van Winchester vezérlő, de az IBM PC is bővíthető vele. A Winchester kezelésére szolgáló portok sorszáma szabvány szerint 320H - 32FH. Mivel sok típusa ismeretes, leírása meghaladná könyvünk kereteit. Az elérésére szolgáló funkciókat a "Lemezkezelés" fejezetben olvashatjuk.

Csak IBM XT !

- BSC (8251) - az opcionális Binary Synchronous Communication (szinkron kommunikációs adapter) az IBM XT-hez külön vásárolható bővítőkártya. Alapja az Intel 8251A típusjelű USART (Universal Synchronous and Asynchronous Receiver and Transmitter, univerzális szinkron és aszinkron adó-vevő) áramkör. A kártyán elhelyeztek még egy 8253 típusjelű időzítő-számláló áramkört, és egy 8255 jelű perifériális interface-t. Elvben csak IBM XT-hez illeszthető BSC adapter, de az IBM PC is bővíthető vele. Részletes leírására nem került sor. A BSC adapterhez kapcsolt I/O portok (az adapter számára fenntartott) 3A0H - 3A9H. A másodlagos BSC adapter portjai: 380H - 389H.

Csak IBM XT !

- SDLC adapter - az opcionális Synchronous Data Link Control (szinkron adatátviteli vezérlő) az IBM XT-hez külön vásárolható bővítőkártya. Az adapter alapja egy 8273 típusjelő SDLC-vezérlő áramkör. A kártyán elhelyeztek egy kisegítő 8253 jelű időzítő-számláló áramkört, valamint egy 8255 jelű perifériális interface-t is. Elvben csak IBM XT-hez illeszthető SDLC adapter, de az IBM PC is bővíthető vele. Részletes leírására nem került sor. Az SDLC adapterhez kapcsolt I/O portok sorszáma 380H - 38FH. Mivel ezek a portok átfedik a BSC adapter címtartományát, egy gépben egyszerre csak az egyik lehet.

I.2. A ROM BIOS

A BIOS (Basic I/O System) rutinjai a memória felső 8 Kb-ján, ROM területen helyezkednek el. Egyebek mellett a legalapvetőbb fizikai I/O tevékenység elvégzésére szolgálnak. Ezeket a rutinokat kellő felkészültséggel nyugodtan használhatjuk. Olyan lehetőségekhez jutunk általuk, melyek gyorsabbá, elegánsabb meg-

jelenésűvé teszik programjainkat, és segítségükkel olyasmit is meg tudunk oldani, melyeket az MS-DOS funkciói egyáltalán nem, vagy csak nehezen tesznek lehetővé.

Igen fontos alapelv mind a hardware elemek direkt programozása, mind pedig a ROM BIOS funkciók használata esetén, hogy ezzel programunk általánossága, hordozhatósága lényegesen csökken. Az IBM PC/XT egyes másolatai, jóllehet "kompatibilitást" igérnek, ezt nem tudják egészen megvalósítani. Egyes hardware elemek eltérőek lehetnek az eredeti gép megfelelő elemeitől. Ez persze részint konstrukciós, részint programozási problémákat vet fel. Egy kissé eltérő hardware elemet kissé eltérő módon kell felprogramozni és használni. Következésképpen megváltozik a ROM BIOS néhány rutinja. Ha a rutinok felhasználói szintje valóban kompatibilis, akkor igen jól sikerült másolattal állunk szemben.

A gyakorlat azonban az, hogy a ROM BIOS rutinok egy része jelentéktelen mértékben ugyan, de különbözik az eredetitől. Az eredeti gépekbe épített ROM BIOS általában nemcsak azért tér el a hasonmás gép BIOS-ától, hogy ne vetődhessen fel a szerzői jogvédelem és a szabadalmi védelem kérdése. Az eredeti rutinok (lévén a hardware elemek némileg különbözők), nem lennének képesek megfelelően vezérelni gépünket. Sok eseben bizony bele kell törődnünk abba, hogy a gépünk nem száz százalékig kompatibilis az eredetivel, és nem feltétlenül számíthatunk arra, hogy minden program le fog futni rajta.

Mi akkor hát a kompatibilitás? Erre a kérdésre (filozófiai alapállásunktól függően) adhatunk egy igen szigorú, és több engedékenyebb választ.

A szigorú válasz az, hogy kompatibilis az a gép, amely minden áramkörében, minden kapcsolásában pontosan megegyezik az eredetivel (beleértve a ROM áramköröket is). Ilyen gép nagyon kevés van. Az ennyire kompatibilis gépek persze minden olyan program futtatására képesek, amelyek az eredetin futnak. Mindazon gépek, amelyek nem ütik meg a kompatibilitásnak ezt a fokát, valamilyen mértékben eltérnek az eredetitől, és ezért inkompatibilisnek kell őket mondanunk. A kompatibilitás igen/nem kérdés, nem mondhatjuk tehát, hogy "valamennyire" vagy "némileg" fennáll.

Egy engedékenyebb válasz azt köti ki, hogy a programok egy bizonyos halmaza hibátlanul fusson a gépen. Ezt a hamazt véleményem szerint úgy határozhatnánk meg, hogy a gépen legyen futásra képes az MS-DOS operációs rendszer valamennyi változata, és minden olyan program, amely nem nyúl az MS-DOS szintje alá. Ezen a nem túl szigorú rostán azonban számos "kompatibilis" gép átesik - tudunk olyan berendezésekről, amelyek nem tudják be-

tölteni például az eredeti MS-DOS 3.00 verzióját. Ez az inkompatibilitásnak már elég goromba foka.

Igen jónak mondhatjuk a komptibilitást akkor, ha a ROM BIOS felhasználói szintje valóban azonos az eredetivel. Ekkor minden olyan program hibátlanul fut, amely nem programozza közvetlenül a hardware-t.

> Megjegyzés: az utóbbi időben egy nagyon fontos szempont merült fel a programozás különböző szintjeivel kapcsolatban. Az Intel 80386 jelű processzorára épülő gépek (PS-2/80 és társai) felülről kompatibilisak az eredeti IBM PC/XT-vel; mindazok a programok futtathatók rajtuk, amelyek nem használnak direkt memóriaelérést (például nem írnak nullát a DS-be az interrupt vektorok címzése céljából), és nem programozzák a hardware-t direkt módon. Ha semmilyen más gépre nem vagyunk tekintettel, a PS-2 akkor is olyan berendezés, amelyre idővel mindenképpen át akarjuk vinni a programunkat. Ha viszont így programozunk, akkor a gyakorlatban sok erősen kompatibilis géppel találkozunk, és programjaink jó eséllyel lesznek hordozhatók (legalábbis az IBM PC-k és hasonmásaik között).

I.3. A ROM BIOS funkciók hívása

A ROM BIOS funkciói a megfelelő interrupt aktivizálásával hívhatók. A szabályok a DOS-funkciók hívásához hasonlatosak. A BIOS funkciók éppúgy egy funkciókódot várnak az AH regiszterben, a többi általános regiszter pedig esetleges egyéb paraméterek átadására szolgál. A kívánt paraméterek betöltése után az

```
INT     n
```

utasítással aktivizáljuk a megfelelő interruptot, amelynek hatására lefut a meghívott BIOS funkció. Visszatéréskor a fő válaszinformáció általában az AL regiszterben található, és (a meghívott funkció függvényében) további regiszterek is tartalmazhatnak visszatérési értékeket.

A BIOS funkciók helyreállítják a szegmensregiszterek, az SP és az IP értékét, nem módosítják az indexregisztereket sem. Az általános regiszterek tartalmára azonban nem számíthatunk - a fontos értékeket a funkció hívása előtt mentenünk kell.

I.4. A hardware elemek közvetlen elérése

Mint ismeretes, az IBM PC/XT programozó által elérhető hardware elemeit I/O portok közvetlen írásával és olvasásával vezérelhetjük. Erre a célra rendelkezésünkre áll a ki- és beviteli utasítások két-két változata: a direkt és az indirekt címzésű utasítások. Mindkét változat képes lehet 8 vagy 16 bites ki- és bevitelre.

Egy direkt címzésű I/O utasítás egyik operandusa mindig az AL vagy AX regiszter (aszerint, hogy 8 vagy 16 bites műveletet kívánunk-e végezni), a másik operandus pedig a megcímezni kívánt I/O port sorszáma:

```
IN      AL, 50H         ;8 bites direkt input
OUT     51H, AL         ;8 bites direkt output

IN      AX, 58H         ;16 bites direkt input
OUT     55H, AX         ;16 bites direkt output
```

Felhívjuk az Olvasó figyelmét arra, hogy a direkt módon megcímezhető portok a 00H - FFH portok, tehát amelyek sorszáma 8 biten elfér!

Az indirekt címzésű utasítások ettől mindössze annyiban térnek el, hogy a port sorszámát nem az utasításba írjuk, hanem előre be kell állítani a DX regiszterben (és a port sorszáma tizenhat bites lehet, így az elérhető portok száma 2^{16}!):

```
MOV     DX, 340H
IN      AL, DX          ;Indirekt input, 8 bit
INC     DX
IN      AX, DX          ;Indirekt input, 16 bit

MOV     DX, 360H
OUT     DX, AL          ;Indirekt output, 8 bit
DEC     DX
OUT     DX, AX          ;Indirekt output,16 bit
```

Megjegyzés: a fenti példákban használt portok nem tartoznak semmilyen hardware-eszközhöz, be nem kötött, de egyébként ötletszerűen kiválasztott portokat használtunk.

Mint "Az IBM PC/XT programozható elemei" c. fejezetből kitűnik, a programozható elemek általában több szomszédos porton át címezhetők közvetlenül. Jó volna valamiféle indexelt címzést használni. Lehetséges ez? Tekintsük a következő makrókat és

konstansdeklarációkat (a példa most konkrét, az aszinkron vonal közvetlen programozását illusztrálja):

```
ASN_BASE        EQU     3F8H            ;Alapport
ASN_THR         EQU       0H            ;Transmit. Holding Reg.
ASN_RBR         EQU       0H            ;Receive Buffer Reg.
ASN_IEN         EQU       1H            ;Interrupt Enable Reg.
ASN_DLRL        EQU       0H            ;Divisor Latch, Low
ASN_DLRH        EQU       1H            ;Divisor Latch, High
ASN_INT         EQU       2H            ;IT Identification Reg.
ASN_LCR         EQU       3H            ;Line Control Register
ASN_MCR         EQU       4H            ;Modem Control Register
ASN_LSR         EQU       5H            ;Line Status Register
ASN_MSR         EQU       6H            ;Modem Status Register
```

Ezután a portok elérését elég szemléletessé tehetjük:

```
        MOV     DX, ASN_BASE + ASN_MSR  ;Modem státusz beolv.
        IN      AL, DX                  ; az AL-be
```

Akinek ez az eljárás még mindig nem elég szép, az az adatkiírást és beolvasást egy efféle makróval végezheti:

```
MACRO   OUTPUT  DATA, BASE, OFFS
                MOV     AL, DATA        ;;Adat AL-be
        IFNB    <BASE>                  ;;
            IFNB        <OFFS>          ;;
                MOV     DX, BASE + OFFS ;;Portcím DX-be
            ELSE                        ;;
                MOV     DX, BASE        ;;Alapport kell
            ENDIF                       ;;
        ENDIF                           ;;
                OUT     DX, AL          ;;Adatkiírás
        ENDM                            ;
                                        ;
MACRO   INPUT   BASE, OFFS              ;
        IFNB    <BASE>                  ;;
            IFNB        <OFFS>          ;;
                MOV     DX, BASE + OFFS ;;Portcím DX-be
            ELSE                        ;;
                MOV     DX, BASE        ;;Alapport kell
            ENDIF                       ;;
        ENDIF                           ;;
                IN      AL, DX          ;;Adat beolvasás
        ENDM                            ;
```

Ezek a makrók teljesen tiszta, világos kódolást tesznek lehetővé és elég kényelmesek is: lehetővé teszik a BASE és OFFSET értékek elhagyását, ha a DX regiszterben valamely korábbi művelet eredményeképpen benne van a kívánt cím. Ha jól definiált konstansokat használunk, akkor első ránézésre látjuk, melyik áramkör melyik regiszterét akarjuk éppen elérni. A BASE és az OFFSET természetesen csak konstansok lehetnek; ez persze nem nagy baj, hiszen nagyon ritkán dolgozunk indirekt portcímekkel (mondhatni sohasem). Mivel pedig az összeadás fordítási időben megy végbe, a program futása semmivel sem lesz lassúbb, mint hogyha mágikus konstansként töltenénk be a portcímet a DX regiszterbe.

I.5. Az IT-MOD.INC file

Itt adjuk meg egyetlen include-file-unk szövegét teljes egészében. Ez a ROM BIOS összes interrupt sorszámának konstansként való deklarálása után két egyszerű makrót tartalmaz. Hasznosak lesznek, mert biztonságosan oldják meg egy interrupt vektor eredeti tartalmának mentését és helyreállítását. Használatukra a későbbi fejezetek példaprogramjai számos példával szolgálnak.

```
IT-MOD          EQU     1       ;Jelezzük IT-MOD beemelését!
;-------------------------------;
; BIOS interruptok - az elnevezési logika:
;       BiosSoftware_...
;       BiosHardware_... és a kezelt eszköz vagy a funkció neve
;       BiosTable_...
;-----------------------------------------;
        NMI_INT         EQU     02H     ;Non maskable IT
        BS_HARDCPY      EQU     05H     ;Screen hard copy
                                        ;
        BH_TIMTICK      EQU     08H     ;Timer tick
        BH_KBD          EQU     09H     ;Keyboard
        BH_RSSEC        EQU     0BH     ;Sec. RS232
        BH_RSPRIM       EQU     0CH     ;Prim RS232
        BH_FDISK        EQU     0DH     ;Floppy disk
        BH_HDISK        EQU     0EH     ;Hard disk
        BH_PRINTER      EQU     0FH     ;Printer
                                        ;
        BS_VIDEO        EQU     10H     ;Video comm.
        BS_EQCHK        EQU     11H     ;Equipment check
        BS_MEMSIZ       EQU     12H     ;Memory size
        BS_FDISK        EQU     13H     ;Floppy comm.
```

```
BS_RS232        EQU     14H     ;RS232 comm.
BS_DEVDRV       EQU     15H     ;Gen. interface
BS_KBD          EQU     16H     ;Keyboard comm.
BS_PRINTER      EQU     17H     ;Printer comm.
BS_BASIC        EQU     18H     ;Call BASIC
BS_BOOT         EQU     19H     ;Bootstrap loader
BS_TMCNT        EQU     1AH     ;Timer control
BS_CTBRK        EQU     1BH     ;CTRL-BREAK user
BS_TMUSR        EQU     1CH     ;Timer user IT
BT_VIDEO        EQU     1DH     ;Video par tab.
BT_FDISK        EQU     1EH     ;Floppy par tab.
BT_GRTAB        EQU     1FH     ;Graphic char gen.
                                ;
XT_DISKSAV      EQU     40H     ;Eredeti 13H IT vektor
XT_WPARTAB      EQU     41H     ;Winchester par. tábla
AT_W2PARTAB     EQU     46H     ;2. Winch. par. tábla
AT_ALARM        EQU     4AH     ;Alarm Csak IBM AT !
                                ;
;-----------------------------------------------;
; S_ITV         az IT_NUM interrupt vektor eredeti tartalmának
;               mentése és új értékkel való felülírása
;-----------------------------------------------------------;
                                                ;
S_ITV           MACRO   IT_NUM, OWN_ADDR, SAVE_ADDR
                PUSH    AX                      ;;Felhasznált
                PUSH    BX                      ;; regiszterek
                PUSH    DX                      ;;
                PUSH    DS                      ;;
                PUSH    ES                      ;; mentése
                MOV     AL, IT_NUM              ;;IT vektor
                DOSCLL  DOS_GETITVECT           ;; beolvasása,
                WRPNT   ES, BX, SAVE_ADDR       ;; mentése
                LDS     DX, DWORD PTR OWN_ADDR  ;;Az új érték;
                MOV     AL, IT_NUM              ;;
                CLI                             ;;;IT letiltás
                DOSCLL  DOS_SETITVECT           ;;;IT vektor
                STI                             ;;IT enged.
                POP     ES                      ;;
                POP     DS                      ;;Felhasznált
                POP     DX                      ;; regiszterek
                POP     BX                      ;; helyreállí-
                POP     AX                      ;; tása
        ENDM                                    ;
                                                ;
```

```
;-------------------------------------------------------------;
; R_ITV         az IT_NUM interrupt vektor helyreállítása     ;
;               a SAVE_ADDR által adott pozícióról            ;
;-------------------------------------------------------------;
                                                              ;
R_ITV           MACRO   IT_NUM, SAVE_ADDR                     ;
                                                              ;
                PUSH    AX                                    ;;Felhasznált
                PUSH    DX                                    ;; regiszterek
                PUSH    DS                                    ;; mentése
                                                              ;;
                LDS     DX, DWORD PTR SAVE_ADDR               ;;Cím felolv.
                MOV     AL, IT_NUM                            ;;
                CLI                                           ;;;IT letiltás
                DOSCLL  DOS_SETITVECT                         ;;;Vektor tölt.
                STI                                           ;;IT enged.
                                                              ;;
                POP     DS                                    ;;
                POP     DX                                    ;;
                POP     AX                                    ;;
                                                              ;
                ENDM                                          ;
```

Mielőtt továbblépnénk, olvassuk el figyelmesen a két makrót, mert a későbbiekben sokszor fogjuk használni őket. Az interrupt vektorok módosítása nagyon fontos és veszedelmes lépés. Pont elég nehéz jól működő interrupt-rutint írni; éppen ezért lényeges, hogy legalább a vektor pontos mentésében és módosításában megbízhassunk. Azt se felejtsük el, hogy a vektor felülírása előtt mindenképpen menteni kell annak eredeti tartalmát, és a program kilépése előtt gondosan vissza kell állítani (éppen ezt teszik a makrók). Figyeljünk arra is, hogy ha interrupt vektort módosító programunk nem a szokásos úton lép ki (hanem például kilőjük), akkor a helyreállítás már nem történik meg és ez később zavarhatja a gép működését.

A fenti file egyébként csupán alapul szolgál a további fejezetekben foglalt információk programra (pontosabban konstans-, makró- és struktúradeklarációk nyelvére) fordításához. Ezt (és a összes többi) file-t az Olvasó saját kezűleg is könnyűszerrel létrehozhatja.

A fő szempontok az alábbiak lehetnek: egy-egy include-file tartalmazzon minden olyan ismeretet konstans-, makró-, struktúra- és rekorddefiníciók formájában, amely egy fontos témakörrel kapcsolatban egyáltalán felmerülhet. Ha viszont elég alaposan írjuk meg a file-t, elkerülhetetlenek bizonyos átfedések (pél-

dául a képernyővezérlő funkciókhoz használt file okszerűen tartalmazza a video-interrupt sorszámát, amit pedig itt is megadtunk). Gondosan ügyeljünk arra, hogy több átfedő include-file beemelése ne okozzon többszörös konstans- és makródefiníciót. Erre szolgál a file elején megadott, a file nevével megegyező szimbólum. Ha egy ilyen file-t beemelünk, akkor ez a szimbólum rögtön definiálttá válik; a további file-okban pedig egy-egy feltételes blokkak védhetjük meg a közös definíciókat:

```
        IFND    IT-MOD
BS_VIDEO        EQU     10H
        ENDIF
```

Egy-egy ilyen szerkezet sok kellemetlenségtől óvhat meg bennünket. A programozás során a más file-okra, tehát más programrészletekre vonatkozó indirekt hivatkozások nagyon megzavarhatják a fordítóprogramok (és ennek következtében a programozók) lelkivilágát. Az assembly programozásban ez nem különösebben nagy baj, mert a szimbólumok többszörös deklarációja tilos (bár a makrókkal nem az a helyzet). Ha például a BS_VIDEO értéket nem védjük le, akkor az assembler kétszer találkozik az erre vonatkozó definícióval, és ez fatális hiba. De ha egy makró definíciója két include file-ban is szerepel, akkor a fordító nem jelez hibát. Mi történik, ha a két azonos nevű (és persze azonos célú) makró szövege nem egyezik meg, és ráadásul a másodszor beemelt változat hibás? A programunk rosszul fog működni; esetleg az elsőként beemelt include file-ban keressük a hibát, ahol pedig a makró szövege jó. Ez már kezd boszorkánysággnak tűnni, pedig csak a "minden dolognak pontosan egy forrása legyen" elvet sértettük meg.

Még élesebb a helyzet a magasszintű nyelvekkel, mert a konstansok többszöri definiálása nem mindig okoz hibát és emiatt nem biztos, hogy egyáltalán észrevesszük a bajt. Különösen fontosak az include file-ok a C nyelvben. Itt nagyon furcsa dolgokat okoz egy teljesen hibátlan include file kétszeri beemelése. Az álmoskönyv szerint ilyenkor számos figyelmeztetés érkezik, amelyeket az ember nem tud mire vélni. Egy idő múlva megunja és átsiklik a dolog felett, hiszen a programja nagyjában-egészében életképes. Valószínűleg nem is lesz vele semmi gond (sohasem tapasztaltam, hogy egy file többszöri beemelése a lefordított kódra valami hatással lenne). Azonban az emiatt bekövetkező figyelmeztetések elfedhetik a valódi és mély programhibákra utalókat; a sok felesleges között könnyen átsiklunk a lényegesen.

II. Az Intel 8087 koprocesszor

Az Intel 8088 processzor nem támogatja a lebegőpontos aritmetikai műveleteket. Ha ilyen számításokat akarunk végezni, akkor magasszintű nyelven kell programoznunk (ezekben megvalósították a lebegőpontos számok kezelését), vagy kénytelenek vagyunk magunk megírni valamilyen lebegőpontos rutincsomagot. A lebegőpontos számok (meglepetésünkre) elég könnyen kezelhetők; bizonyos algoritmikus nehézségek leküzdése után viszonylag egyszerűen végezhetünk szorzást, osztást stb. Számolnunk kell azzal, hogy meg kell majd írnunk egy sor olyan szubrutint, amely például nyolc byte hosszúságú számokat oszt el nyolc byte-os számokkal stb. (A bináris osztás meglepően egyszerű, könnyebb végrehajtani, mint tízes számrendszerben.) Az elkészült rutincsomag hátránya lesz azonban, hogy az aritmetikai műveletek elvégzése nem lesz nagyon gyors; gondoljuk meg, hogy pl. egy összeadáshoz az egyik szám mantisszáját valószínűleg párszor jobbra vagy balra kell rotálni, hogy a két szám karakterisztikája azonos legyen; a művelet csak így végezhető el.

Az Intel cég éppen a fenti nehézségek leküzdésére fejlesztette ki az Intel 8087 lebegőpontos koprocesszort. Ez, mint neve is mutatja, "ko"-processzor, olyan processzor, amely a gép főprocesszorával "párhuzamosan" működik.

A koprocesszor beépítése egyszerű. Ha felnyitjuk a gépet, akkor az alapkártyán, a processzor mellett (amit legegyszerűbben a nagyságáról ismerhetünk fel) a processzorral azonos méretű üres helyet találunk; ide kell betenni a koprocesszort az irányhelyességre ügyelve. A tok egyik végén egy kis bevágást találunk. Hasonló mélyedés van az áramköri aljzat egyik végén is; a tok "sliccének" az aljzat bevágásával azonos oldalon kell lennie. A koprocesszor ezután (még egy-két mikrokapcsoló átállításával) munkára kész.

Működésének alapelvei: állandóan "lesi" a processzor buszát, és amikor a processzor egy ESC utasítást ad ki (amely - mint az első kötetben olvasható - egy hat bites utasítást kényszerít ki a buszra, majd egy tizenhat bites címet), "elkapja" ezt a hatbites utasítást, értelmezi, majd pedig, ha kell, a cím segítségével elérhet valamilyen memóriapozíciót a gép memóriájában. Tekintsük át most vázlatosan az Intel 8087 utasításkészletét, hogy azután megvizsgálhassuk a használható adattípusokat, majd ezek ismeretében a koprocesszor utasításkészletét, immár teljes precizitással.

A koprocesszor belsejében egy nyolcelemű, veremszerűen kezelt belső memória van, amelynek minden eleme igen nagy pontossággal ábrázolhat lebegőpontos számokat. Egy elem tíz byte-nyi

hosszúságú, tehát nyolcvan bites. A koprocesszor képes arra, hogy a gép memóriájából tetszőleges 8086-os adatcímzési móddal kiolvasson bármilyen legális formátumú adatot, és azt saját belső számábrázolása szerint konvertálva a stack tetejére helyezze; ugyanezt visszafelé is meg tudja tenni: egy, a stackben levő elemet a programozó által óhajtott konverzió elvégzése után ki tudja írni a memóriába. Munkájának érdemi része a lebegőpontos műveletek elvégzése. Ennek feltétele, hogy az operandus vagy (kétoperandusú művelet esetén legalább az egyik operandus) a stack tetején legyen. Tehát a koprocesszor segítségével úgy végezhetünk aritmetikai műveleteket, hogy először beolvassuk a kívánt értékeket a koprocesszor belső memóriájába, a stackre, elvégezzük a szükséges műveleteket, az eredményt pedig kiírjuk a gép memóriájába.

II.1. Az INTEL 8087 által ismert adattípusok

A koprocesszor egyik nagy erénye, hogy nemcsak lebegőpontos, de egész, sőt pakolt decimális adattípusokat is ismer; ha tehát egészek között bonyolult műveleteket akarunk igen gyorsan végezni, a koprocesszor felhasználásával ez is megoldható, nem kell fáradságos munkával először egészből lebegőpontossá, majd vissza alakítanunk az adatokat csak azért, hogy a koprocesszor számára emészthetők legyenek.

Egész típusok

a) Szavas egész - tizenhatbites egész szám, melynek első bitje az előjelbit, így tizenöt értékes bitje van. Kettes komplemensű számábrázolás. A szavas egész sémája:

Szavas egészek számára a DW operátorral foglalhatunk helyet.

b) Rövid egész - harminckét bites egész szám, melynek első bitje az előjelbit - így harmincegy értékes bitje van. Kettes komplemensű számábrázolás. A rövid egész sémája:

Rövid egészek számára a DD operátorral foglalhatunk helyet. A "rövid" elnevezés a koprocesszor irodalmában e változóra használt "short integer"-ből származik; szokványos programozói felfogásunk szerint azonban nem éppen rövid, hiszen ez például a C nyelv "long integer" típusa!

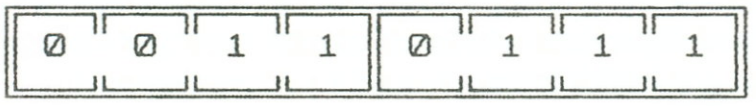
c) Hosszú egész - hatvannégy bites egész szám, melynek első bitje az előjelbit - így hatvanhárom értékes bitje van. Kettes komplemensű számábrázolás. A hosszú egész sémája:

Ilyen változók számára a DQ operátor segítségével foglalhatunk helyet. Ez a legutóbbi típus már igen súlyos gondokat okoz; az érték meghatározásához alig-alig van egyéb eszközünk, mint: kézzel kiszámítani! A PC-n elérhető magasszintű nyelvek adattípusai nem képesek ilyen nagy számok ábrázolására. Assembler "programot" írni az érték meghatározására triviális:

```
TWOP63  DQ      7FFFFFFFFFFFFFFF
```

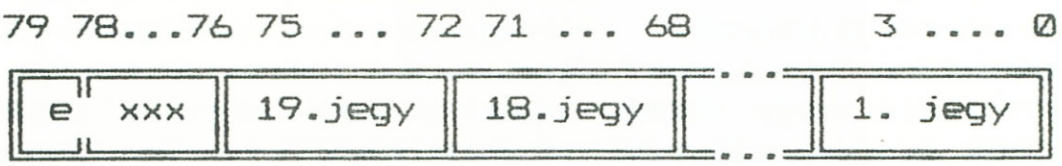
azonban a bináris érték decimálissá alakítása több mint barátságtalan feladat. Magam úgy jutottam hozzá, hogy C nyelven írtam egy kis programot, amely kétszeres pontosságú lebegőpontos változóban határozta meg a kívánt értéket. De ez a kettőnek ötvenötödik hatványától kezdve pontatlan, mert a számok "aprajáát", a tízeseket-százasokat már a C "double" típusa is elnyeli. Itt tehát kézi korrekciót alkalmaztam.

d) BCD egész, pakolt decimális szám - tíz byte-on ábrázolt tizenkilenc jegyű decimális szám.
A pakolt decimális ábrázolás a következő: minden számjegy számára négy bitet tartunk fenn, amelyben egy 0 és 9 közé eső értéket tárolunk, bináris alakban. Egy byte-on tehát két, tíz byte-on húsz számjegyet helyezhetünk el. Lássunk egy példát egy kétjegyű pakolt decimális számra! Legyen a szám értéke 37. Ekkor a byte alsó négy bitjén kap helyet a hetes, a felső négy biten pedig a hármas bináris érték:



A koprocesszor tizenkilenc jegyű pakolt decimális számokat kezel. Ezek tíz byte-ot foglalnak le úgy, hogy a legmagasabb helyiértékű (és című) byte felső négy bitjét nem számjegy-

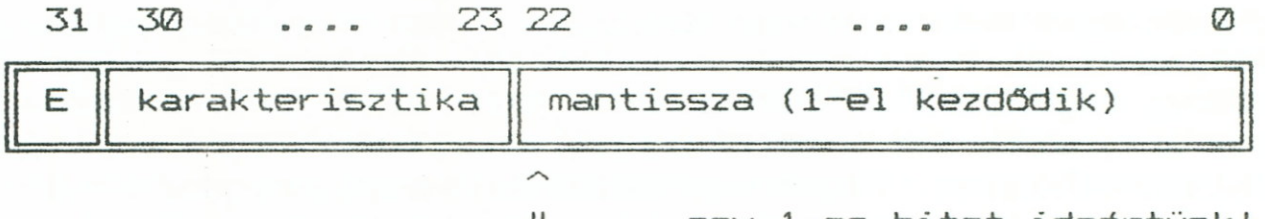
ként kezeljük, hanem a legfelső bit az előjelbit, a másik hármat nem vesszük figyelembe:



Ilyen változók számára a DT operátor segítségével foglalhatunk helyet.

Lebegőpontos típusok

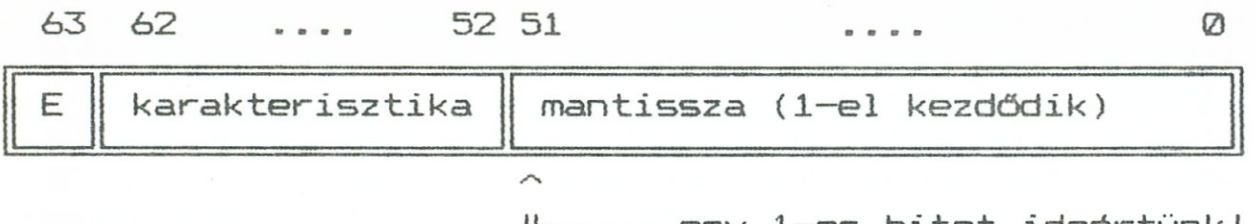
e) Rövid valós szám - harminckét biten ábrázolt lebegőpontos szám.
A számot külön karakterisztikára és mantisszára bontva ábrázoljuk. A karakterisztika hossza nyolc bit, melyből a legmagasabb a karakterisztika sajátosan értelmezett előjelbitje. A mantissza hossza fizikailag huszonhárom bit. Az egész szám előjelét a legmagasabb helyiértékű bit adja meg.



Minthogy a lebegőpontos számok ezen ábrázolása mindig normalizált számokkal dolgozik, az első bitje mindig 1; ezt ki sem kell írni, a koprocesszor automatikusan odagondolja. A mantissza hossza valójában tehát huszonnégy bit.
Fontos tudnunk, hogy ez milyen gyakorlati pontosságot jelent. A mantissza hat-hét decimális jegyet ér, míg a karakterisztika nyolc bitje a tíz mintegy harmincnyolcadik hatványáig engedi nőni a számot (pontosan azért nem adható meg ez az érték, mert a szám nagysága a mantisszától is függ); a legnagyobb szám körülbelül egy egész héttized szorozva a tíz harmincnyolcadik hatványával. Az ábrázolható legkisebb pozitív valós szám (tessék?! igen!... itt ilyen is van!) a tíz mínusz harmincnyolcadik hatványa körül van.
A MASM számára lebegőpontos változókat a DD operátorral definiálhatunk. Amennyiben tizedespontot teszünk a kezdeti értékbe, akkor a szám lebegőpontos konverzió után kerül majd be a tárba. Példákat később olvashatunk.

f) Hosszú valós szám - hatvannégy bites lebegőpontos szám. A számot külön karakterisztikára és mantisszára bontva ábrázoljuk. A karakterisztika hossza tizenegy bit, ebből a legmagasabb a karakterisztika sajátosan értelmezett előjelbit-

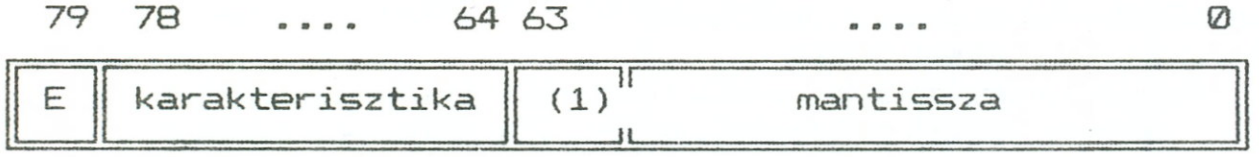
je. A mantissza fizikai hossza ötvenkét bit. Az egész szám előjelét a legmagasabb helyiértékű bit adja meg.



Minthogy a lebegőpontos számok ezen ábrázolása mindig normalizált számokkal dolgozik, az első bitje mindig 1; ezt ki sem kell írni, a koprocesszor automatikusan odagondolja. A mantissza hossza most tehát ötvenhárom bit. Mindez körülbelül tizenöt-tizenhat számjegyet ér; a szám nagyságrendje pedig a tíz háromszázhetedik hatványáig emelkedhet. Kis számok körében a pontosság igen nagy, hiszen a legkisebb ábrázolható pozitív érték a tíznek valahol a mínusz háromszázhetedik hatványa körül van.

A MASM számára hosszú lebegőpontos változókat a DQ operátorral definiálhatunk. Ha tizedespontot teszünk a kezdeti értékbe, akkor a szám lebegőpontos konverzió után kerül majd be a tárba. Példákat később olvashatunk.

g) Emelt pontosságú valós szám - nyolcvan bites lebegőpontos szám. A számot külön karakterisztikára és mantisszára bontva ábrázoljuk. A karakterisztika hossza tizenöt bit, melyből a legmagasabb a karakterisztika sajátosan értelmezett előjelbitje. A mantissza hossza hatvannégy bit. Az egész szám előjelét a legmagasabb helyiértékű bit adja meg.



Az emelt pontosságú lebegőpontos számok nem mindig normalizáltak, ezért a szám nem kezdődik kötelezően egy egyes értékű bittel. Tehát a mantissza hossza itt valóban hatvannégy bit.

Az emelt pontosságú számok 64 bites, nem normalizált mantisszája körülbelül tizenkilenc decimális számjegyet ér. A karakterisztika tizenöt bit, ebből következően a legnagyobb ábrázolható szám nagyságrendje tíz a négyezerkilencszázharminckettediken. Mivel a számot nem mindig normalizált alakban ábrázolják (a legmagasabb értékes jegyet nem mindig tolják fel a mantissza legfelső pontjára), a legkisebb ábrázol-

ható szám jóval kisebb, mint várnánk; a mínusz négyezerkilencszázharminckettőhöz durván még mínusz tizenkilencet adhatunk. Ez a számábrázolási mód nagyon érzékeny arra, hogy egy szám különbözik-e a nullától, vagy sem. A MASM számára emelt pontosságú lebegőpontos változókat a DT operátorral definiálhatunk. Ha tizedespontot teszünk a kezdeti értékbe, akkor a szám lebegőpontos konverzió után kerül majd a tárba.

Az alábbiakban a fent használt kifejezések magyarázatát olvashatjuk. Először ismerkedjünk meg a kettes komplementumú számábrázolással! Ezt a számábrázolási formát használjuk az egész típusú számok esetén.

A számítástechnikában a kettes komplemens terjedt el az előjeles egész számok ábrázolására. Ennek előnye, hogy az aritmetikai műveletek során az esetek többségében nem igényel korrekciót, és nincs algoritmikus különbség az előjel nélküli és előjelesen ábrázolt számokkal végzendő addíciós jellegű műveletek elvégzésében. A kettes kompementumú ábrázolás hátránya, hogy ember számára csaknem emészthetetlen; a komplementálás elvégzése nélkül becsülni is alig lehet egy negatív szám értékét.

A módszer lényege: a rendelkezésünkre álló bitekből egyről lemondunk, az hordozza az előjelet. A nulla értéket jelentse a csupa nulla jegyből álló bináris szám, innen a pozitív számokat növeléssel érjük el. A negatív számokat pedig egyszerűen ismételt csökkentéssel kaphatjuk meg. Lássuk ezt egy nyolc bites szám esetén:

dec. szám	bináris megfelelője	
0	0000 0000	
1	0000 0001	0-ból egy növeléssel
2	0000 0010	1-ből egy növeléssel
.		
122	0111 1010	többszörös növeléssel
127	0111 1111	többszörös növeléssel
-1	1111 1111	0-ból egy csökkentéssel
-2	1111 1110	-1-ből újabb csökkentéssel
.		
-122	1000 0110	több csökkentéssel
-128	1000 0000	legkisebb negatív

Kettes komplemensű számábrázolás

Most nem térünk ki a kettes komplemensű számábrázolás aritmetikai sajátosságaira. Erről kicsit bővebben olvashattunk az 1. kötetben, a flagek viselkedéséről szóló fejezetben. Itt csak annyit figyeljünk meg, hogy akárhány bitről legyen is szó, az előjelet mindig a legmagasabb helyiértékű jegy mutatja (a szám persze nem ettől negatív, hanem attól, hogy programból negatív számként kezeljük!); ha ez a jegy 0, akkor a szám pozitív, ha a jegy 1, akkor negatív. Végezetül jegyezzük meg a legegyszerűbb konverziós algoritmust. Ez pedig a következő:

> Fordítsuk meg a szám valamennyi bitjét (írjunk a nullák helyére egyet és viszont), majd az eredményt eggyel növeljük meg. A túlcsordulást hagyjuk figyelmen kívül.

Programunk kétféleképpen végezheti el a konverziót. Az egyik út a NEG utasítás használata, amelynek során a processzor hajtja végre a fenti műveleteket. A másik lehetőség pedig a NOT (mely az egyes komplemenst adja vissza) és az INC használata. A két út mellékhatásaiban tér el egymástól; lesznek esetek, mikor ugyanazon szám kettes komplemensének kiszámítása után más értékeket hagynak a flagekben. Egyébként hosszabb egészek konverziója csak az utóbbi módon lehet: minden szót NOT-olunk, majd lentről haladva "végigcsorgatunk" egy egyest az egész számon:

```
VERYLONG        DQ      ?                       ;Nagyon hosszú egész

        NOT     WORD PTR VERYLONG               ;Minden
        NOT     WORD PTR VERYLONG + 2           ; bitet
        NOT     WORD PTR VERYLONG + 4           ; gondosan
        NOT     WORD PTR VERYLONG + 6           ; megforgatunk
                                                ;
        ADD     WORD PTR VERYLONG,  1           ;Megnöveljük
        ADC     WORD PTR VERYLONG,  0           ; az egész
        ADC     WORD PTR VERYLONG,  0           ; számot
        ADC     WORD PTR VERYLONG,  0           ; eggyel
```

Az algoritmus helyességének igazolását az Olvasóra bízzuk. Aki pedig járatlan a kettes komplemensű számábrázolásban, ne sajnálja az időt és a fáradságot arra, hogy jónéhány példát végigszámol kézzel: összeadásokat, szorzásokat stb. Nem helyes, ha elhisszük azt, hogy minden jól működik; győződjünk meg róla. A nyereség az lesz, hogy át tudjuk tekinteni az összes aritmetikai problémát.

A következő kérdés a pakolt BCD számok ábrázolása. Mint látni fogjuk, igen kényelmesen, jól használhatjuk a BCD számokat

programból is, nemcsak a koprocesszor segítségével. Azonban az előjeles BCD számokkal végzendő műveletek bizonyos problémákat vetnek fel, melyek áttekintése során áldva gondolunk majd a kettes komplemensű ábrázolás kiagyalójára, akit az előző bekezdésben még a pokolnak ajánlottunk.

Mint leírtuk, a BCD szám tíz byte-on helyezkedik el, tizenkilenc jegyű, és a legmagasabb helyiértékű pozíción találjuk az előjelbitet. Lássunk egy ilyen BCD számot úgy, ahogyan azt az assembler előállítja! Egy példaprogram részletét (a fordítási listát a jobb olvashatóság kedvéért kissé kozmetikázva) másoltuk be a szövegbe:

```
0000  56 34 12 00 00 00 00 00 00 00     BCD1    DT       123456
000A  56 34 12 00 00 00 00 00 00 80     BCD2    DT      -123456
```

Ez az adatszegmensrészlet két DT operátort tartalmaz, kezdeti értékük egymás ellentettje. Látható, hogy a számjegyek négy-négy bitet foglalnak le, és a legalacsonyabb helyiértékű szám kerül a tíz byte-nyi hosszúságú terület legalacsonyabb helyiértékű négy bitjére. A szám előjelét a legmagasabb helyiértékű négy bit, annak is legmagasabb helyiértékű bitje tartalmazza (a legutolsó 00, illetve 80).

Mint az első kötetben is olvasható, az addíciós jellegű műveletek (összeadás és növelés, valamint kivonás és csökkentés) igen egyszerűen végezhetők el a BCD számok körében. Hiába támogatja azonban a gépi kód a BCD formátumú szorzást és osztást, ezek elvégzése assembly nyelven több mint kellemetlen. Gondoljuk csak meg, hogy az elemi matematika egyik legbonyolultabb algoritmusa az osztás (nem kisebb személyiség, mint a pedagógusnak is kiváló Németh László jelentette ki, hogy aki az osztás műveletét megérti, az már bármit képes megérteni). Maga az osztás binárisan sokkal egyszerűbb, mint a decimális számrendszerben, mert nem kell "eltalálni" a hányadost (hiszen vagy megvan az osztandó megfelelő néhány jegyében az osztó, vagy nincs), azonban a "lehozom a következő jegyet" lépés assembly megvalósítása bizony egy cseppet sem egyszerű!

A bináris osztás megvalósítása azonban sokkal könnyebb, mint a BCD osztás, hiszen a BCD ábrázolásban bizony algoritmikusan nem bináris, hanem decimális számrendszerben írjuk le a számokat. A BCD ábrázolás elsősorban a Cobol nyelv "könnyű" megvalósítása kedvéért kapott helyet a processzor és a koprocesszor adatábrázolási módjai között. Ha azonban nemcsak egész, hanem lebegőpontos jellegű ábrázolást is megengedünk a BCD filozófiájára alapozva, pontosabb számábrázolást kapunk: ez az ábrázolás nem kerekít, és a "valahány-egész-háromtized" egész pontosan

megadható. A magam részéről a szorzást és az osztást inkább a Cobol fordítóprogramra bízom. Akiben megvan az ehhez szükséges elszántság, ám merüljön bele az algoritmusba.

Most lássuk az érdekesebb kérdést, a lebegőpontos számok ábrázolását. Az olyan alapfogalmak ismertetésétől, mint a "mantissza" és a "karakterisztika", eltekintek. Azonban egy dolog mindenkinek szemet szúrhat a lebegőpontos számok rövid ismertetésénél: a karakterisztika "sajátosan felfogott előjelbitje". Mit is jelent ez? A 0.0000 számot csupa 0 bittel ábrázoljuk, ez tiszta. Lássuk azonban az 1.0000 szám ábrázolását (cimkéje FL1)! Mint a rövid leírásban olvasható, a mantissza első bitje kötelezően egy. Ezt "kettedes törtként" így ábrázolhatjuk:

$$1.00000 \times 2^{0}$$

Pontosan ezt látjuk az FL1 értékénél (felülről lefelé olvasva: 3F 80 00 00). A mantissza értéke csupa 0, mivel a kezdő egyest ki sem írjuk; a karakterisztika 7FH (mely a legfelső byte 3FH értékéből és a következő byte-ot kezdő 1 bitből alakul ki), de miért? A karakterisztika értékét 7FH-val megnövelve kell megadni azért, hogy ne kelljen a karakterisztika kezelésében rendkívül bonyolult kettes komplemensű számábrázolással kínlódni. Innen már világosan látható, hogyan alakul ki a 2.00000 szám:

$$1.00000 \times 2^{1}$$

ahol a mantissza csupa-csupa nulla (hiszen a ki nem írt egyessel kezdődik); a karakterisztika pedig 80H (ti. 7FH + 1). Pontosan ez látható az FL2 szám lefordított alakjában (felülről lefelé olvasva a byte-okat: 40 00 00 00). A nagyobb kettő-hatványok pontosan ugyanígy keletkeznek, a karakterisztika folytonos növelésével, míg a negatív kettő hatványok egyszerűen úgy keletkeznek, hogy a számok legfelső bitjeit 1-be kell állítani (pl. a -2.00000 érték byte-onként felülről lefelé olvasva: C0 00 00 00, azaz: 1 [előjelbit], 40 [karakterisztika] és 23 db 0 értékű bit mint mantissza). Innen már az is látható, hogy a 2.00000 helyes "olvasata": 0 [előjelbit], 40 [karakterisztika] és 23 db 0 értékű bit [mantissza].

> Megjegyzés: az alábbi lista a Microsoft MASM 4.00 verziójával készült, a "/R" opció alkalmazásával. Felhívjuk a figyelmet, hogy a MASM támogatja a lebegőpontos műveletek emulált végrehajtását. Erről a későbbiekben olvashatunk még.

```
0000  00 00 00 00          FL0      DD      0.0000
0004  00 00 80 3F          FL1      DD      1.0000
0008  00 00 00 40          FL2      DD      2.0000
000C  00 00 80 40          FL4      DD      4.0000
0010  00 00 00 41          FL8      DD      8.0000

0010  00 00 80 BF          FLM1     DD     -1.0000
0018  00 00 00 C0          FLM2     DD     -2.0000
001C  00 00 80 C0          FLM4     DD     -4.0000
0020  00 00 00 C1          FLM8     DD     -8.0000
```

Következő példaként ismét az 1.00000-ból induljunk ki, de ezúttal ne szorozzunk, hanem osszunk kettővel. Ekkor sem történik semmilyen érdemi változás a számok mantisszájában; az továbbra is 23 db 0 értékű bit marad. A karakterisztika pedig az ismételt osztások folyamán elkezd szépen csökkenni:

```
0024  00 00 00 3F          FL05     DD      0.5000
0028  00 00 80 3E          FL025    DD      0.2500
002C  00 00 00 3E          FL0125   DD      0.1250

0030  00 00 00 BF          FLM05    DD     -0.5000
0034  00 00 80 BE          FLM025   DD     -0.2500
0038  00 00 00 BE          FLM0125  DD     -0.1250
```

Mint látható, az FL05 számból elmaradt a második byte 80 értéke, azaz a karakterisztika 7FH-ról 7EH-ra csökkent, és így tovább. A negatív változatok esetén szintén csak az egész szám legmagasabb helyiértékű bitje állt be 1-be, jelezvén a negatív előjelet.

A következő lépésben nem tiszta kettőhatványokkal kísérletezünk, hadd lássunk valamit a mantisszában is! A számértékek: a 3.00000 kettőhatványokkal való szorzatai.

```
003C  00 00 C0 3F          FL15     DD       1.5000
0040  00 00 40 40          FL3      DD       3.0000
0044  00 00 C0 40          FL6      DD       6.0000
0048  00 00 40 41          FL12     DD      12.0000
004C  00 00 C0 41          FL24     DD      24.0000
```

Az alapos ránézés megmutatja, hogy a 3.00000 csak egy bitben különbözik a 2.00000 értékétől. A második byte-ban áll a 6. helyiértékű bit (ettől lesz a byte értéke 40H). Ez pedig nem más, mint a mantissza legmagasabb helyiértékű bitje. Ehhez hozzátéve a mantissza odaértendő legmagasabb bitjét, kialakul az "1100.."

értékű mantissza, melybe a két egyes közé értendő a kettedespont. A karakterisztika értéke persze 1 (megjelenése: 7FH + 1).

```
0050  00 00 C0 BF                FL15    DD      -1.5000
0054  00 00 40 C0                FLM3    DD      -3.0000
0058  00 00 C0 C0                FLM6    DD      -6.0000
005C  00 00 40 C1                FLM12   DD     -12.0000
0060  00 00 C0 C1                FLM24   DD     -24.0000
```

A fenti számoknak megfelelő negatív értékeket ugyanúgy kaphatjuk meg; erre kár is szót vesztegetni. Az **egyharmad** közelítő értéke (ez végtelen kettedes tört, legfeljebb csonkított formában ábrázolható) a következő: 0 [előjelbit], 7D [karakterisztika], majd 23 biten át a 010101... sorozat. A karakterisztika 7D értéke -2-t "ér"; persze, hiszen a szám kisebb, mint egyketted!

```
0064  AA AA AA 3E                FL03    DD      0.3333333
```

Vizsgáljuk meg búcsúzóul a kétszeres pontosságú és az emelt pontosságú számábrázolást is. Itt csak az a cél, hogy bemutassuk az itt alkalmazandó értelmezéseket. Lássuk először a kétszeres pontossággal ábrázolt példákat (a fordítási listát a jobb olvashatóság kedvéért egy kissé kozmetikáztuk)!

```
0068  00 00 00 00 00 00 F0 3F    DFL1    DQ      1.0000
0070  00 00 00 00 00 00 00 40    DFL2    DQ      2.0000
0078  00 00 00 00 00 00 10 40    DFL4    DQ      4.0000
0080  00 00 00 00 00 00 E0 3F    DFL05   DQ      0.5000
0088  00 00 00 00 00 00 D0 3F    DFL025  DQ      0.2500
0090  00 00 00 00 00 00 C0 3F    DFL0125 DQ      0.1250

0098  00 00 00 00 00 00 F0 BF    DFLM1   DQ     -1.0000
00A0  00 00 00 00 00 00 00 C0    DFLM2   DQ     -2.0000

00A8  00 00 00 00 00 00 E0 BF    DFLM05  DQ     -0.5000
00B0  00 00 00 00 00 00 D0 BF    DFLM025 DQ     -0.2500
```

A kétszeres pontosságú számok után következzenek az azonos értékű emelt pontosságú számok (a fordítási lista ismét "feljavított"):

```
00B8  00 00 00 00 00 00 00 80 FF 3F    TFL1    DT      1.0000
00C2  00 00 00 00 00 00 00 80 00 40    TFL2    DT      2.0000
00CC  00 00 00 00 00 00 00 80 01 40    TFL4    DT      4.0000
00E4  00 00 00 00 00 00 00 80 02 40    TFL8    DT      8.0000
```

```
00FE  00 00 00 00 00 00 00 80 FE 3F    TFL05      DT      0.5000
0108  00 00 00 00 00 00 00 80 FD 3F    TFL025     DT      0.2500
0112  00 00 00 00 00 00 00 80 FF BF    TFLM1      DT     -1.0000
011C  00 00 00 00 00 00 00 80 00 C0    TFLM2      DT     -2.0000
0124  00 00 00 00 00 00 00 80 FE BF    TFLM05     DT     -0.5000
012E  00 00 00 00 00 00 00 80 FD BF    TFLM025    DT     -0.2500
```

Kétszeres pontosságú szám karakterisztikája 12 bit, a kiegészítő érték 3FFH. Olvassuk csak el az 1.00000 kétszeres pontosságú alakját! 0 [előjelbit], 3FFH [karakterisztika, ennek előjelbitje 0], és 52 db 0 értékű bit (melyből 53 bit úgy lesz, hogy oda számítjuk a vezető egyest is). Emelt pontosságú számábrázolás esetén a karakterisztika 15 bites; a felülről számított harmadik byte legfelső bitje ezúttal a mantissza része.

Mielőtt befejeznénk a számábrázolás rejtelmeivel való ismerkedést, még gondoljuk át az eltolt karakterisztika egyik legfontosabb előnyét. Ha össze akarunk hasonlítani két számot, akkor felülről lefelé, bitről bitre hajthatjuk végre a műveletet; az első különböző érték már eldönti, melyikük a nagyobb. Ha kettes komplemens segítségével ábrázolnánk a számokat, akkor az összehasonlítás sokkalta bonyolultabb lenne.

A koprocesszor belső ábrázolásában egyébként találkozhatunk néhány érdekes esettel is. A koprocesszor (bár magát a 0.0 számot csupa 0 bittel ábrázolja) ismeri a negatív 0 értéket is; belső ábrázolásában van lehetőség a végtelen érték, sőt az előjeles végtelen értékek ábrázolására. (Végtelen egyébként nullával való osztás vagy osztási túlcsordulás esetén keletkezhet.)

A karakterisztika fizikai 0 értéke (ami logikailag a legkisebb exponensnek felelne meg, rövid valós esetén pl. -128-nak, -80H-nak) egy speciális, nem definiált értéket jelent.

II.2. Műveleti hibák (kivételek)

A lebegőpontos műveletek végrehajtása során számos hiba léphet fel kezdve a triviális alogritmikus hibákon, a számábrázolás korlátaiból eredő többé-kevésbé mély hibákig. Ezeket összefoglaló néven, az eredeti angol kifejezés (exception) buta fordításával kivételeknek fogjuk nevezni. Az alábbiakban megismerkedünk e rendellenességek típusaival, valamint kezelésük elvi lehetőségeivel.

Hiba bekövetkezése esetén a koprocesszor kétféleképpen működhet. Ha a programozó engedélyezi, interrupttal értesíti a programunkat a hibáról (az IBM PC-n a koprocesszor az NMI-re, a processzor le nem tiltható interruptjára van kötve). Amennyiben nem engedélyezzük az interruptot, akkor a koprocesszor saját

hatáskörében oldja meg a kérdést, és az alábbiakban részletezett hibák esetén az ugyancsak itt olvasható módon reagál. A koprocesszor tervezői a felsorolt hat osztályba sorolták a hibákat:

(1) Invalid operation, érvénytelen művelet.
Ilyenek: a koprocesszor belső stackjének túl- és alulcsordulása, és meghatározhatatlan eredmény keletkezése.
A stack túlcsordulása azt jelenti, hogy a nyolcelemű stack már megtelt, és még egy elemet akarunk betölteni. Alulcsordulás akkor következik be, ha olyan elemhez akarunk visszanyúlni, amely a stackben már nem létezik (például egy elemet töltöttünk csak a stackbe, de két stackbeli elemet akarunk összeadni). Ezek általában (nem is kis) algoritmushibák; a koprocesszor nem hajtja végre az utasítást. Meghatározhatatlan eredmény jellegzetesen akkor (nem) keletkezik, ha 0.0-t osztunk el 0.0-val; erre a koprocesszor nincs felkészülve. Hasonló "eredmény" keletkezik akkor, ha végtelenből akarunk végtelent kivonni stb. Ezek (bár algoritmikusan elkerülhetők) mégsem olyan durva hibák, mint a stack túl- vagy alulcsordulása. Szintén ilyen "eredmény" keletkezik akkor, ha egy koprocesszor függvényt nem megengedett paraméterekkel akarunk végrehajtani.
Meghatározhatatlan eredmény bekövetkezése esetén a koprocesszor egy erre a célra fenntartott értéket (csupa 0 bit) ír be a karakterisztikába.

(2) Overflow, túlcsordulás
A keletkezett eredmény meghaladja az ábrázolható legnagyobb számot. A koprocesszor végtelen értéket ír az eredmény helyére, és folytatja a munkát.

(3) Zero Divisor, nullával való osztás
Az elvégzendő osztás osztója nulla, míg az osztandó nem nulla vagy nem végtelen érték. A koprocesszor végtelen értéket ír az eredmény helyére, és folytatja a munkát.

(4) Underflow, alulcsordulás
A keletkezett eredmény abszolútértékben kisebb, mint az ábrázolható legkisebb érték.
Az eredmény nulla lesz; a koprocesszor folytatja a munkát.

(5) Denormalized operand, nem normalizált operandus
Ez a kivétel akkor következik be, ha az operandusok valamelyike nem normalizált, vagy az eredmény ábrázolhatatlan ebben az alakban (például olyan kicsi, hogy normalizálása lehetetlen).
A koprocesszor folytatja a munkát (a 0-tól valószínűleg különböző eredmény elvész, 0 lesz).

(6) Inexact Result, pontatlan eredmény
A művelet eredménye a kényszerűen vagy előírtan bekövetkezett kerekítés miatt pontatlan. Ez a hiba osztás után várható; ha elosztjuk pl. a 2.0-t 3.0-val, az eredmény csak végtelen kettedes törtben ábrázolható! A koprocesszor elvégzi a kerekítést, és folytatja a munkát.

A fenti feltételek nagyjából az itteni sorrendben követik egymást a hiba súlyosságát illetően. Ha például stack-túlcsordulás következett be, akkor a program egyszerűen rossz; kár is folytatni vele a munkát. Túl- vagy alulcsordulás, nullával (vagy végtelennel) való osztás is leginkább algoritmikus hiba eredménye (különös tekintettel arra, hogy a koprocesszor belső számábrázolása nagyon pontos, és az ábrázolható számtartomány is igen széles). Nem normalizált eredmény könnyen bekövetkezhet; ehhez csak elég kicsi, de nullától mégis különböző számokkal kell dolgoznunk. Mindenesetre, ez is utalhat algoritmikus hibára (ti. a "koordinátarendszer", a választott mértékegység helytelen voltára). Végül a kerekítési hiba a legtöbb esetben teljesen lényegtelen, hiszen bekövetkezése elkerülhetetlen szinte minden osztás után.

A koprocesszor tehát többé-kevésbé kezeli a hibákat saját hatáskörében is. Nagyon kell vigyáznunk azonban arra, hogy lehetőség szerint figyeljünk a hibákra, és bekövetkezésük esetén próbáljuk meg elhárítani a következményeket, különben szerencsétlen adatok esetén a program teljesen hamis eredményeket is produkálhat. Mint már említettük, két út áll előttünk. Az egyik lehetőség az, hogy minden egyes művelet után megvizsgáljuk a koprocesszor státuszát, és ha valami rendellenes dolgot tapasztalunk, akkor megpróbáljuk értelmezni, a következményeket elhárítani (pl. alulcsordulás esetén számos esetben már nem érdemes folytatni a munkát, hiszen olyan részeredménnyel számolnánk, amely nem egyszerűen pontatlan, hanem nullától különböző érték helyett 0 - a további szorzások már értelmetlenek).

A másik lehetőség az, hogy engedélyezzük a hibajelző interruptot. Ez esetben arra kell tekintettel lennünk, hogy csak azokat a kivételeket kezeljük interrupttal, amelyekre reálisan számítani lehet. Az alábbiakban látni fogjuk, hogy ha interruptot akarunk, akkor egészében véve engedélyezni kell az interrupt aktivizálását, de külön-külön osztályonként is megszabhatjuk, hogy ezt vagy azt a kivételt jelezze-e a koprocesszor.

A hibavizsgálatokról szólva úgy tűnik, hogy nem fontos a stack alul- vagy túlcsordulására felkészülni, mert ez kifejezetten programozói hiba. Középiskolai matematikai tanulmányainkból emlékezhetünk arra, hogy jó elkerülni a nullával való

osztást és minden egyéb olyan műveletet, amely lebegőpontos túlcsordulást eredményez. Ezeket a veszélyeket a legtöbb algoritmusban elég jól meg lehet előzni - pl. tudhatjuk azt, hogy a szereplő értékek természetüknél fogva nem lehetnek bizonyos értéknél nagyobbak. Az alulcsordulás valamelyest szintén elkerülhető a jól fogalmazott algoritmussal. Egy alulcsorduló részeredménnyel fenyegető műveletsort egyszer-egyszer át lehet úgy rendezni, hogy a kritikus lépés kimaradjon belőle.

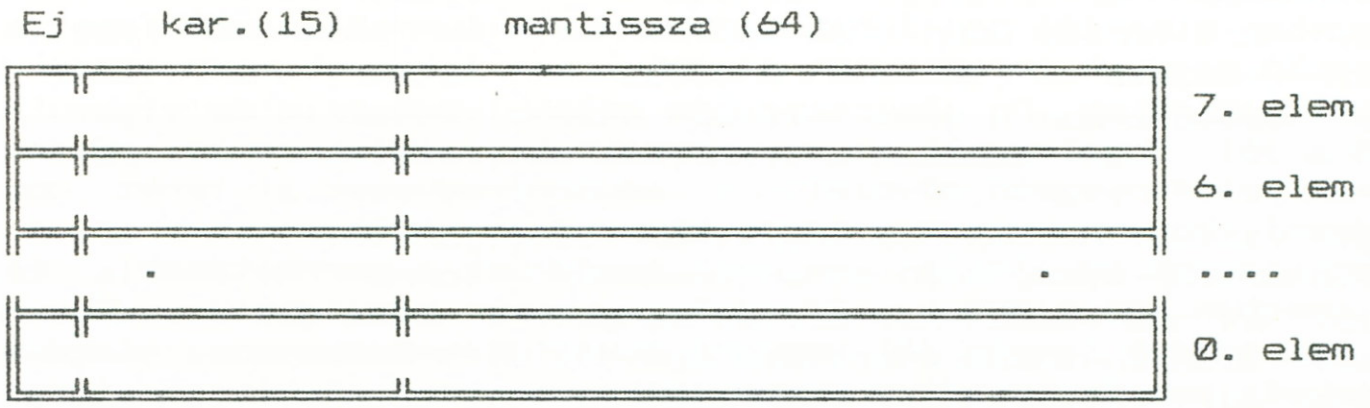
Ugyanakkor egyáltalán nem kell törődnünk a kerekítéssel. Ez sok esetben kényszerűen előfordul; és akkor mit tehetünk? Papíron sem tudjuk igazán jól kezelni a végtelen szakaszos tizedes törteket; nem is beszélve a nem szakaszosakról, tehát az irracionális számokról. Gyakorlati szempontból édesmindegy, hogy egy szám huszadik-huszonötödik jegye elvész-e vagy sem; nem az hordozza a döntő fontosságú információt. Ebben a kérdésben (mielőtt elkezdenénk programot kódolni) alapos megfontolásokra van szükség; tanulmányozni kell a feladat során esetlegesen előforduló értékeket (beszámítva a program felhasználójának esetleges tévedéseit is), a kívánt számábrázolási pontosságot, a rendelkezésre álló futásidőt és memóriát. Mint a számábrázolási módok ismertetésénél láttuk, a rövid valós számok pontossága (hat-hét jegy) nem biztos, hogy minden gyakorlati feladatra jó. A hosszú valósok pontossága (eltekintve űrkutatási és katonai alkalmazásoktól) csaknem biztosan elégséges, de kétszeres a helyigényük.

A feladat gondos tanulmányozása felesleges munkától kímélhet meg bennünket. Ha pl. kiderül, hogy az eléggé jó megoldás húsz megabyte-nyi nagyon gyors elérésű tárat igényel, akkor bízvást kijelenthetjük: "...azért nem harangozunk, mert pro primo nincs harang...", egy percet sem érdemes vesztegetnünk a feladat megoldására IBM PC vagy akár AT nagyságrendű géppel.

II.3. A koprocesszor belső regiszterei

Felhasználói szempontból legfontosabb a belső stack. Minden művelet címzettje a stack tetején levő elem. Ezért a stack elemeit a következőképpen szokás jelölni: st(0), st(1) stb., ahol st(0) a stack teteje, st(1) az alatta levő elem és így tovább. Kényelmetlen dolog, hogy ha assemblerben programozunk, fejben kell tartanunk: mikor, melyik érték a stack hányadik eleme; ha pedig új elemet helyezünk el, akkor azt is tudnunk kell, hogy minden elem lép egyet: a korábban betöltött elemek indexe megnövekszik; hasonló földindulást okoz egy elem törlése.

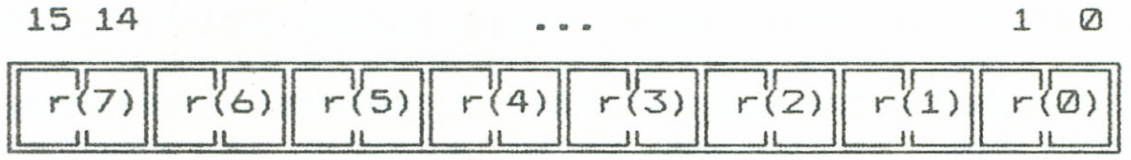
II.3.1. Az Intel 8087 stackje



A koprocesszor stackje

A stack vázlatos bemutatásán az "Ej" természetesen az előjelet, a "kar" a karakterisztikát jelenti. A legelső (az első adattöltő utasítás címzettjeként szereplő) elem van alul.

II.3.2. Stackleíró regiszter (Tag Word)



A bitpárok alulról felfelé alkotják a 0., 1. stb. stack elem leíró regisztereit. Egy bitpár az alábbi jelentést hordozhatja:

A stackleíró regiszter bitpárjainak értékei	jelentése
00	A megfelelő elem érvényes adatot tartalmaz
01	A megfelelő elem nullát tartalmaz
10	A megfelelő elem különleges értéket tartalmaz
11	A megfelelő elem üres

A stackleíró regiszter értelmezése

Itt a "különleges érték" szorul magyarázatra: azt jelenti, hogy egy művelet eredményeképpen a stack elemben végtelen érték szerepel, vagy valamilyen okból érvénytelen az eredmény.

II.3.3. Vezérlőregiszter (Control Word)

A vezérlőregiszter 16 bites. A programozó szabadon állíthatja be tartalmát, és így vezérelheti a koprocesszor működésének olyan finomságait, mint a kerekítés során alkalmazandó módszert stb. Ezek részletes leírása a vezérlőregiszter bitjeinek ismertetése alatt olvasható.

Mint látható, két részre osztottuk a tizenhat bites regisztert, mert míg a felső nyolc bit a koprocesszor működési stratégiáját szabja meg, az alsó nyolc bit a hibák okozta interruptok vezérélésére szolgál.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	x	I	R.C.		P.C.		M	x	PM	UM	OM	ZM	DM	IM

ahol "x" nem használt bitet jelent. A további bitek és a hozzájuk tartozó magyarázatok:

- I, Infinity Control a végtelen számok kezelési módja. Ha nullával osztunk, akkor ír a koprocesszor a hányados helyére végtelen értéket. Ezt jelzi a stackleíró szóban a megfelelő bitpár (10) értéke. A számkör lezárásának matematikai értelemben kétféle módja van: projektív és affin lezárás. Durván szólva az a különbség, hogy az affin lezárás kétféle (pozitív és negatív) végtelent ismer, míg a projektív lezárás "összeejti" ezt a két értéket, mondván, hogy a végtelen egy és oszthatatlan. Egyik módszer sem helyezhető a másik fölé. A bit jelentése:
  0 - projektív (alapértelmezés),
  1 - affin lezárással kívánunk dolgozni.
- R.C., Rounding Control a kerekítés vezérlése.
  00 - kerekítés a legközelebbi ábrázolható értékre
  01 - lefelé kerekítés (a mínusz végtelen felé)
  10 - felfelé kerekítés (a plusz végtelen felé)
  11 - csonkítás - nulla felé kerekítés
  Figyeljük meg, hogy a csonkítás a C nyelv szabványos lebegőpontos-egész konverziós eljárása! Tehát a koprocesszor erősen támogatja a C megvalósítását.
- P.C., Precision Control az alkamazandó számítási pontosság vezérlése. Bizonyos esetekben nem kívánatos az, hogy a kelletkezett eredményt a belső számábrázolás precizitásával kezeljük, bár az mindig ilyen. Ha már régebben megírt eljárásokkal dolgozunk, melyek valahogy kezelik a szokásos 4-byte-os ábrázolásból adódó kerekítési hibákat, akkor a nagyobb pontosság könnyen okozhat valami galibát.

A Precision Control bitpár értékei:
00 - 24 bit - megfelel a rövid valós pontosságának
01 - fenntartott
10 - 53 bit - megfelel a hosszú valós pontosságának
11 - 64 bit - emelt pontosság (alapértelmezés)

- M, Mask - a koprocesszor interruptjának engedélyezése vagy tiltása.
A koprocesszor a lebegőpontos műveletek közben bekövetkezett hibák esetén interruptot ad a processzornak. A kiváltott interrupt az IBM PC-n az NMI, az Intel 8088 le nem tiltható interruptja.
A Mask bit értékei:
0 - kérjük az interrupt beadását
1 - letiltjuk az interrupt beadását.

A további bitekkel azt szabhatjuk meg, hogy a kivételek (hibák) közül melyek váltsanak ki valóban interruptot. Ez hasznos lehet akkor, ha egyáltalán nem vagyunk kíváncsiak az adott kivételre, vagy a státusz olvasásával programból akarjuk kezelni a problémát. Az alább ismertetett bitek 0 értékükkel engedélyezik és 1 értékükkel tiltják a megfelelő interrupt beadását.

- PM, Precision Mask - kerekítést jelző interrupt;
- UM, Underflow Mask - túlcsordulást jelző interrupt;
- OM, Overflow Mask - alulcsordulást jelző interrupt;
- ZM, Zedo Divide Mask - nullával osztást jelző interrupt;
- DM, Denormalized Operand Mask - nem normalizált operandust vagy eredményt jelző interrupt;
- IM, Invalid Operation Mask - érvénytelen művelet-interrupt.

II.3.4. Státuszregiszter (Status Word)

A státuszregiszter tizenhat bites. Tartalma az utoljára végrehajtott lebegőpontos művelet eredménye szerint áll be. Felhasználása nélkülözhetetlen információkhoz juttatja a programozót. Fontos ismeret, hogy (mint látni fogjuk) felső nyolc bitjéből két státuszbit pontosan megfelel az Intel 8088 státuszregiszterében a Zero és a Carry bitnek. Minthogy pedig az SAHF utasítás segítségével bármilyen értéket betölthetünk a STATUS alsó nyolc bitjére, egy koprocesszorművelet után beolvashatjuk és közönséges feltételes vezérlésátadási utasításokkal felhasználhatjuk ezeket a biteket.

15	14	13 12 11	10	9	8	7	6	5	4	3	2	1	0
B	C3	Stack p.	C2	C1	C0	IR	x	PE	UE	OE	ZE	DE	IE

Itt is két részre osztottuk a tizenhat bites regisztert. Az alsó nyolc bit azt mutatja, hogy okozna-e interruptot a koprocesszor által utoljára végrehajtott művelet, és ha igen, akkor melyik osztályba tartozó kivétel váltotta volna ki. A felső nyolc bit mutatja a koprocesszor aritmetikai állapotát. Kivételesen kezdjük a bitekkel való ismerkedést alulról, az egyszerűbb esetekkel!

- IR, Interrupt Request - ha értéke 1, azt jelzi, hogy a koprocesszor interruptot adna. Az itt részletezendő hat egyéb bit valamelyike ekkor szintén egyes értéket kap; ez mutatja, hogy a bekövetkezett hiba melyik osztályba tartozik.
- IE, Invalid Operation Error - érvénytelen művelet;
- DE, Denormalized Operand Error - a hibát nem normalizált operandus vagy nem nomralizálható eredmény okozta;
- ZE, Zerodivide Error - a hibát nullával való osztás okozta;
- OE, Overflow Error - a hibát túlcsordulás okozta;
- UE, Underflow Error - a hibát alulcsordulás okozta;
- PE, Precision Error - az eredményt kerekíteni kellett.
- C0 (0. feltételi bit) - az eredmény természetére utal;
- C1 (1. feltételi bit) - az eredmény természetére utal;
- C2 (2. feltételi bit) - az eredmény természetére utal;
- C3 (3. feltételi bit) - az eredmény természetére utal (a C0, C1, C2 és C3 biteket illetően lásd az alábbi táblázatokat);
- Stack p. - ez a három bit a stack tetejét mutatja. A 000 érték azt jelzi, hogy a stack üres, az első betöltendő elem a nulladik stack elembe kerül; az 111 érték azt jelenti, hogy a stack tele van.
- B, Busy - azt mutatja, hogy a koprocesszor dolgozik-e vagy sem. Ha értéke 1, akkor éppen dolgozik, tehát újabb utasítást nem szabad kiadnunk a koprocesszornak. Ez a bit lehetővé teszi a programunk és a koprocesszor software-úton megvalósított szinkronizálását.

A C0, C1, C2 és C3 bitek jelentését táblázattal szemléltetjük. Mint látni fogjuk, a feltételbitek értelmezése nem egyszerű. A gyakorlatban csak egy-két bitet kell vizsgálnunk, amely az e fejezet kezdetén tett megjegyzéssel viszonylag könnyű: kihasználhatjuk, hogy a STATUS Zero és Carry flagjeinek helyén azonos értelmű feltételi bitek vannak.

C3 Zero	C2	C1	C0 Carry	Előjel	A kombináció jelentése
0	0	0	0	+	érvényes, nem normalizált
0	0	0	1	+	nem szám (kitevő 0)
0	0	1	0	−	érvényes, nem normalizált
0	0	1	1	−	nem szám (kitevő 0)
0	1	0	0	+	érvényes normalizált pozitív
0	1	0	1	+	pozitív végtelen
0	1	1	0	−	érvényes normalizált
0	1	1	1	−	negatív végtelen
1	0	0	0	+	nulla (pozitív)
1	0	0	1	üres	———
1	0	1	0	−	nulla (negatív)
1	0	1	1	üres	———
1	1	0	0	+	érvénytelen, nem normalizált
1	1	0	1	üres	———
1	1	1	0	−	érvénytelen, nem normalizált
1	1	1	1	üres	———

A feltételi bitek jelentése FXAM utasítás után

A feltételi bitek értelmezése igen bonyodalmas, komoly elmélyülést igénylő feladat. Kevés olyan feltétel van, melyhez egyértelműen kapcsolható egy-egy bit. Az alábbi bekezdésekben ehhez kísérlünk meg támpontot adni.

A C3 bit értéke akkor 0, ha a művelet eredménye (akár normalizált, akár nem) nullától különbözik. Ha a bit értéke 1, akkor az eredmény nulla vagy érvénytelen, vagy pedig üres a vizsgált stack elem. Elmondhatjuk, hogy a C3 bit nagyjából megfelel a STATUS Zero bitjének. A C2 bit értelmezése a C3 bit értékétől függ. Amennyiben a C3 bit értéke 0, akkor a C2 bit 1 értéke jelzi a normalizált, a 0 pedig a nem normalizált eredményt. Ha a C3 bit értéke 1 (tehát az eredmény zérus, vagy a stack elem üres), akkor éppen fordítva: a C2 egyes értéke érvénytelen számot, a C2 nullás értéke a nullát jelenti.

A C1 bit, mint a táblázatból kitűnik, a szám előjelére utal. Ha az eredmény negatív, a C1 bit értéke 1, különben 0. A C0 bit arra utal, hogy az eredmény érvényes vagy érvénytelen. Ha e bit értéke 0, akkor nincs nagy baj; ha 1, az eredmény mindenképpen érvénytelen (nem szám; végtelen vagy egyéb különleges érték).

Ha azonban nem közönséges lebegőpontos műveleteket hajtunk végre, hanem összehasonlító utasításokat, akkor a bitek értelmezése is megváltozik:

C3	C2	C1	C0	A kombináció jelentése
0	0	x	0	ST(0) > "op"
0	0	x	1	ST(0) < "op"
1	0	x	0	ST(0) = "op"
1	1	x	1	ST(0) és "op" nem összehasonlítható

A feltételi bitek összehasonlítás után

Ha tehát C3 értéke 0, akkor az összehasonlítás eredményét a C0 bitből olvashatjuk ki. Ha C3 értéke 1, akkor a C2 mondja meg azt, hogy egyenlőek-e a számok, vagy pedig ST(0) nem összehasonlítható (pl. üres vagy végtelen).

A parciális maradékképző művelet után (lásd FPREM) ismét más értelme van a feltételi biteknek. Ez esetben C0, C1 és C3 (alulról felfelé ebben a sorrendben!) a maradékos osztás során keletkező hányados alsó egy, két vagy három bitjét tartalmazzák. C2 értéke 0 egy értelmes maradékképzés után, 1 értéke pedig hibát jelez. Ezután a bitek értelmezése a következő (a parciális maradékképzésről bővebben a megfelelő utasítás ismertetése során olvashatunk):

Az osztó és osztandó aránya	C3	C1	C0
osztó > osztandó/2	x	x	0. bit
osztó > osztandó/4	x	1. bit	0. bit
osztó <= osztandó/4	2. bit	1. bit	0. bit

A feltételi bitek parciális maradékképzés után

Az "x" azt jelenti, hogy az adott esetben a megfelelő bitek megőrzik eredeti értéküket. Például ha az osztó nagyobb az osztandónál, akkor parciális maradéka megegyezik az osztandóval, a hányados pedig 0. Ekkor C3 és C1 megőrzi eredeti értékét, míg C0 nulla lesz, jelezve, hogy a hányados nulla. Ha pedig az osztó kisebb az osztandó felénél, de nagyobb annak negyedrészénél, akkor a hányados lehet kettő vagy három; ezt találjuk a C1, C0 bitpárban, míg C3 megőrzi eredeti értékét.

II.3.5. Utasításregiszter (Instruction Pointer)

Az utasításregiszter a koprocesszor által utoljára végrehajtott lebegőpontos utasítás fizikai címét és operációs kódját tartalmazza:

15 12	11 10	0
A fizikai cím bitjei (15 - 0)		
(19 - 16)	0	Az utasításkód (10 - 0 bitek)

Ez a regiszter akkor lehet segítségünkre, ha interrupt-rutint írunk, amely a koprocesszor-műveletek során előforduló hibákat hivatott kezelni. Ilyen esetben nagyon fontos, hogy ismerjük a hibát kiváltó utasítás címét (tehát fizikai helyét a gép fő memóriájában) és operációs kódját. Gondoljuk át, hogy a program nem köteles megvárni a koprocesszort; míg az dolgozik, végezheti saját bokros teendőit. Csak akkor kell "ránéznünk" a koprocesszorra, ha szükségünk van az eredményre, vagy újabb műveletet akarunk vele végrehajtatni. Ekkor pedig (minthogy programunk már régen túllépett a hibát kiváltó utasításon) semmilyen más módon nem tudhatjuk meg, mi is volt a koprocesszornak küldött utolsó utasítás. Az utasításkódnak fenntartott hely hosszabb a valódi helyigénynél; a kód "jobbra igazítva" szerepel itt.

II.3.6. Adatregiszter (Data Pointer)

Az adatregiszter az utoljára végrehajtott lebegőpontos utasítás által felhasznált külső adat fizikai címét tartalmazza:

15 12	11 0
Az adat fizikai címének bitjei (15 - 0)	
(19 - 16)	0

Ez a regiszter, az előzőhöz hasonlóan, akkor lehet segítségünkre, ha interrupt rutint írunk a koprocesszor-műveletek során előforduló hibák kezelésére. Ilyenkor ismernünk kell a hibát kiváltó utasítás által kezelt (és számára külső) adat memóriacímét.

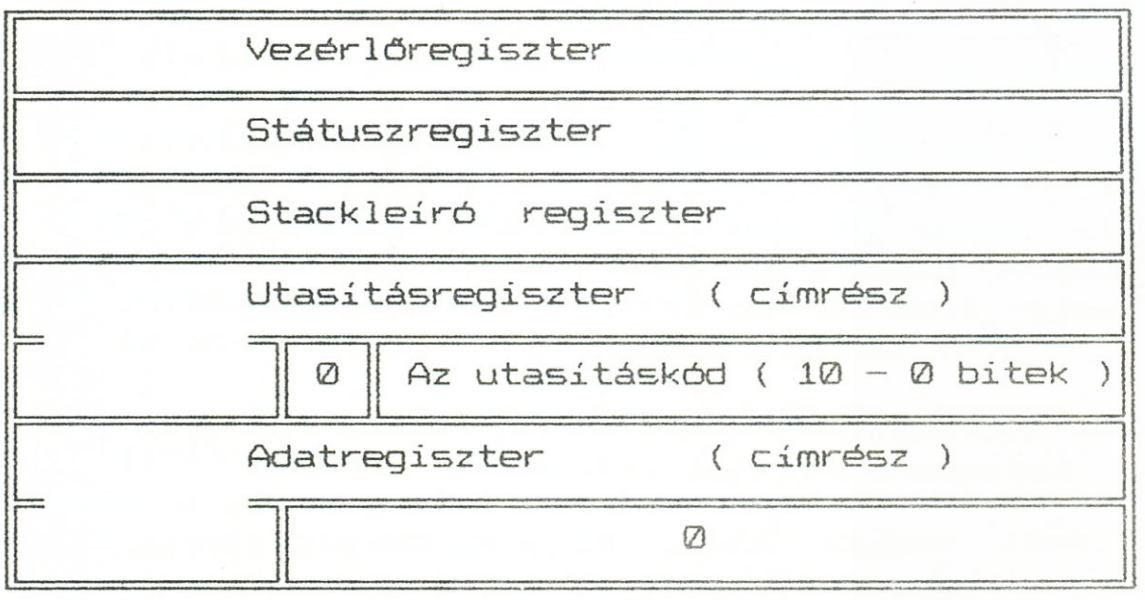
II.4. A koprocesszor utasításkészlete

A koprocesszor programozását az assembler ESC utasításával végezzük, amely az adatbuszra egy hatbites operációs kódot, majd (ha az adott utasításhoz erre szükség van) a címvonalakra egy memóriacímet küld ki. A koprocesszor "elkapja" a neki szóló utasítást, és elkezdi végrehajtását. Közben a processzor folytathatja a program futtatását; kétféleképpen kerülhet ismét szinkronba a koprocesszorral: vagy megnézi a koprocesszor státuszát (ami egyébként elég időigényes művelet), vagy pedig kiad egy WAIT utasítást. A két processzor munkája így software- és hardware-úton is szinkronizálható.

A MASM 1.0 verziója makrók segítségével oldja meg a lebegőpontos utasítások fordítását. Ehhez egy file-t kell includeolni. A makrók WAIT utasítással kezdődnek, melyeket a megfelelően definiált ESC követ. Az assembler újabb verziói már alapszóként ismerik a lebegőpontos utasításokat; ezeket szintén ESC utasításokra fordítják le.

A stack tartalmára az ST(0), ST(1) stb. szimbólumokkal hivatkozhatunk. Az ST(0) a stack tetejét, tehát az utolsó elemet jelöli. Az ST(1) az eggyel alatta levőt és így tovább. Tehát a stack tetejéhez viszonyított relatív címzéssel hivatkozhatunk a stackben korábban elhelyezett adatokra.

Meg kell ismerkednünk a koprocesszorkörnyezet, és a koprocesszorállapot fogalmával és a kezelésükre szolgáló struktúrákkal. A koprocesszor-környezet (angolban environment) a koprocesszor összes belső regiszterét jelenti, kivéve magát a stacket. Hossza tizennégy byte, felépítése az alábbi:



A koprocesszor környezete

Ebből kézenfekvő az alábbi struktúradeklaráció:

```
CP_ENV          STRUC
        CP_CONT         DW      0       ;Vezérlőregiszter
        CP_STATE        DW      0       ;Státuszregiszter
        CP_TAG          DW      0       ;Leíró regiszter
        CP_IPOFFS       DW      0       ;I.P. offset rész
        CP_IPEXT        DW      0       ;I.P. kiterj & op. kód
        CP_DATA         DD      0       ;Adatregiszter
CP_ENV          ENDS

CP_INSTR        RECORD  CP_IPEX:4,CP_ZERO:1,CP_OPCOD:11
```

Struktúrába rekordot sajnálatos módon nem építhetünk. Persze, ha meggondoljuk, hogy egy rekorddeklarációnak voltaképpen nincs is más értelme, mint hogy kényelmesen és világosan definiálhatjuk a rekord mezőinek kezeléséhez szükséges konstansokat (hiszen a MASM egyébként semmivel nem segíti a rekordok használatát), akkor átlátjuk, hogy ez nem is olyan nagy hiányosság.

A hasonló szerkezetű utasításregiszter és adatregiszter leírása azért eltérő, mert az adatregisztert legkényelmesebb valóban duplaszavasnak felfogni, míg az utasításregiszter felső szava két különböző részre oszlik.

A koprocesszor állapota a belső regiszterek teljes készlete. Az első tizennégy byte pontosan megegyezik a környezettel, ezt követi a stack elemeinek sorozata. Ez utóbbi felfogható úgy is, mint egy nyolcelemű, a stackelemet leíró struktúrából szervezett tömb. Lássunk először példát a stack egy elemét leíró struktúrára, majd pedig hozzunk létre a teljes koprocesszorállapot befogadására szolgáló struktúrát.

```
CP_STEL         STRUC
        CS_M0           DW      0       ;Mantissza  0-15 bitjei
        CS_M1           DW      0       ;Mantissza 16-31 bitjei
        CS_M2           DW      0       ;Mantissza 32-47 bitjei
        CS_M3           DW      0       ;Mantissza 48-63 bitjei
        CS_EXP          DW      0       ;Előjel,karakterisztika
CP_STEL         ENDS

CP_EXPW         RECORD  CP_SIGN:1, CP_EXPT:15
```

Az utóbbi definíció egy rekordot hoz létre, amely a CS_EXP nevű tag kezelését segíti azzal, hogy különválaszthatjuk az előjelet! A továbblépés során ismét az előbb már említett akadályba ütközünk. A MASM nem engedi meg, hogy struktúrában fel-

használjunk egy már megadott struktúradefiníciót. Emiatt leszünk kénytelenek (mint egy ízben már a második kötetben - lásd a kiterjesztett FCB definícióját) struktúra helyett makró segítségével definiálni egy koprocesszorállapotot. Lássuk!

```
CP_STATE        MACRO   NAME
NAME    LABEL   BYTE
        CP_ENV          <>                  ;Környezeti rész
        CP_STEL         8   DUP ( <> )      ;Koprocesszor stack
        ENDM
```

A "NAME" szimbólumot azért deklaráltuk BYTE típusúként, mert leggyakrabban így kell rá hivatkoznunk.

A rutinos Olvasó már sejtheti, hogy azért fordítottunk ekkora gondot a fenti adatstruktúrák deklarálására, mert közvetlenül elérhetjük a koprocesszor belső regisztereit. A vezérlőutasítások között fogunk majd olyanokat találni, amelyek felülírják a koprocesszor írható regisztereit, vagy kimásolják a regiszterek egy részét vagy a teljes regiszterkészletet a gép memóriájába.

Az alábbiakban használni fogjuk az "addr"-t valamely memóriacím jelölésére. Ezalatt tetszőleges olyan címkifejezést értünk, amely az egyéb assembly utasításokban legális. Használhatunk bármilyen címzési módot és minden olyan operátort, amely a címzésben szerepelhet (pl. a PTR operátort), és az alapértelmezett szegmensregisztert átdefiniálhatjuk (például ES:cím). Az "op" (operandus) jelölés jelenthet egy stack elemet vagy a négy fontos típushoz tartozó memóriaváltozót.

> Megjegyzés: a négy fontos típus: szavas és rövid egész, rövid és kétszeres pontosságú lebegőpontos változó. A koprocesszor tervezői úgy találták, hogy a kimaradt három típushoz (hosszú, nyolcbyte-os egész, emelt pontosságú [10 byte-os] lebegőpontos, valamint pakolt decimális) tartozó változókat nem szükséges olyan gyakran használni. A ritkább esetekben pedig nyugodtan be lehet tölteni ezeket a stackre, és nem kell valamilyen memóriacímzéssel, direkt módon elérni.

Végül egy-két szót a mnemonikokról! Figyeljük meg, hogy valamennyi "F" betűvel kezdődik, jelezve a művelet lebegőpontos jellegét.

II.4.1. Adatmozgatási utasítások

Ezek az utasítások több fontos alcsoportba sorolhatók: betöltő és kiíró utasítások, belső adatmozgatások, valamint konstanstöltő utasítások.

Betöltő (LOAD) utasítások

FILD addr — Az "addr" címen található egész változót tölti a stackre. A változó típusát (szavas, rövid vagy hosszú) a definíció határozza meg (ti. hogy DW, DD vagy DQ operátort használtunk a változó definíciójára). A szükséges konverzió a betöltés során megtörténik.

FLD addr — Az "addr" címen található értéket, mint rövid vagy hosszú lebegőpontos változót a stack tetejére tölti. A változó típusát (rövid, kétszeres vagy emelt pontosságú) a definíció határozza meg (ti. hogy DD, DQ vagy DT operátort használtunk a változó definíciójára). A szükséges konverzió a betöltés során megtörténik.

FBLD addr — Az "addr" címen található pakolt decimális változót a stack tetejére tölti. A változót előzőleg a DT operátorral szokás definiálni. A szükséges konverzió a betöltés során megtörténik.

Kiíró (STORE) utasítások

FIST addr — Az ST(Ø) értékét az "addr" címre írja ki egész számként. A kiírt érték típusa csak szavas vagy rövid (négy byte hosszú) egész lehet; ezt az "addr" definíciója határozza meg (ti. hogy DW vagy DD operátort használtunk a változó definíciójára). A stack teteje és ott az ST(Ø) értéke érintetlen marad. A kiírás során a szükséges konverzió végbemegy.

FISTP addr — Az ST(Ø) értékét az "addr" címre írja ki egész számként; az ST(Ø)-t leveszi a stackről (azaz eggyel csökkenti a stack tetejét mutató érté-

ket). A kiírt szám típusa tetszőleges egész lehet; hogy melyik (szavas, rövid vagy hosszú), azt az "addr" definíciója határozza meg (hogy DW, DD vagy DQ operátort használtunk a változó definíciójára). A kiírás során a szükséges konverzió végbemegy.

FST addr — Az ST(0) értékét az "addr" címre írja ki rövid vagy kétszeres pontosságú lebegőpontos számként. A kiírt érték típusát az "addr" definíciója határozza meg (ti. hogy DD vagy DQ operátort használtunk a változó definíciójára). A kiírás után a stack teteje és ott az ST(0) értéke változatlan marad. A kiírás során a szükséges konverzió végbemegy.

FSTP addr — Az ST(0) értékét az "addr" címre írja ki lebegőpontos számként; az ST(0)-t leveszi a stackről (eggyel csökkenti a stack tetejét mutató értéket). A kiírt érték típusát (rövid, kétszeres vagy emelt pontosságú) az "addr" definíciója határozza meg (hogy DD, DQ vagy DT operátort használtunk a változó definíciójára). A kiírás során a szükséges konverzió végbemegy.

FBSTP addr — Az ST(0) értékét az "addr" címre írja ki pakolt decimális számként; az ST(0)-t leveszi a stackről (eggyel csökkenti a stack tetejét mutató értéket). A változó definíciójára DT operátort szokás használni. A kiírás során a szükséges konverzió végbemegy.

Ezek az utasítások a gép programunk által kezelt memóriája, és a koprocesszor belső stackje közötti adatforgalmat biztosítják. Alaposan gondoljuk át a fent olvasottakat, és figyeljük meg, hogy míg tölteni bármilyen típusú értéket tudunk (amelyet a koprocesszor egyáltalán ismer), a kiírás már némileg korlátozott. Ha a kiírt értéket végleg el is vesszük a stackről, akkor a hét adattípus bármelyikére végezhetünk kiírást. Ha azonban a kiírandó értéket meg akarjuk őrizni a stackben további műveletek céljára, akkor csak a négy fontos típusra tudunk kiírni és konvertálni (szavas és rövid egész, rövid és kétszeres pontosságú lebegőpontos). Mint látni fogjuk, a kétoperandusú aritmetikai műveletek is el tudják érni a memóriát (onnan olvasva az egyik operandust), azonban csak a négy alaptípust!

Belső adatmozgatás

FLD	ST(i)	Az ST(i) értéket ismét ráteszi a stackre. Ezzel létrejön egy új elem, amelynek értéke megegyezik az ST(i) értékkel. A művelet után az eredeti ST(i)-t már ST(i+1)-ként lehet elérni!
FST	ST(i)	Az ST(Ø) értéket átmásolja a stack tetejétől számított "i"-edik elemre, melynek értéke elvész. Ez megfelel egy MOV utasításnak.
FSTP	ST(i)	Az ST(Ø) értéket átmásolja a stack tetejétől számított "i"-edik elemre; az ST(Ø) értéket leveszi a stackről (eggyel csökkenti a stack tetejét címző értéket). Minden index csökken.
FXCH	ST(i)	Az ST(Ø) és ST(i) elem értékét megcseréli.

Konstansok betöltése

FLDZ	Zérus értéket tölt a stack tetejére.
FLD1	A stack tetejére az 1.Ø értéket tölti.
FLDPI	A stack tetejére "pi" (3,14...) értékét tölti.
FLDL2T	A stack tetejére a tíz kettő alapú logaritmusát helyezi el.
FLDL2E	A stack tetejére az "e" (2,71...) kettő alapú logaritmusát helyezi el.
FLDLG2	A stack tetejére a kettő tíz alapú logaritmusát helyezi el.
FLDLN2	A stack tetejére a kettő "e" alapú (természetes) logaritmusát helyezi el.

A "beavatottak" számára világos, hogy ezek az utasítások a trigonometrikus és egyéb aritmetikai számítások elvégzéséhez szükséges legfontosabb értékeket töltik a koprocesszor stackjére anélkül, hogy nekünk magunknak kellene nagy fáradsággal területet foglalni és (az itteninél sokkal kisebb pontossággal!) inicializálni.

II.4.2. Aritmetikai és összehasonlítási utasítások

Az aritmetikai utasítások többnyire kétoperandusúak. Az egyik operandus mindig a stack tetején levő érték (az ST(Ø) szimbólummal jelölt elem). Ezt felülírva a legtöbb esetben ide kerül az eredmény is.

Minden alapműveletet korlátlanul végezhetünk, és mind a négy alapműveletet háromféle, eltérő stratégiával hajthatjuk végre. Az első (amikor csak az operációs kódot kell kiírni) a legegyszerűbb eset: a részt vevő operandusok ST(Ø) és ST(1). A második esetben ki kell írni egy operandust, amely lehet egy memóriaváltozó vagy a stack egy eleme [okszerűen nem az ST(1)]. A harmadik eset igen érdekes és ravasz trükkökre ad alkalmat. Két operandust írunk ki; az első a stack egy eleme [nem ST(Ø)], a második pedig ST(Ø). Ekkor a céloperandus az első helyen megadott elem, míg ST(Ø) a művelet után törlődik a stackről (a koprocesszor stack pointere csökken, és a leíró regiszter jelzi, hogy az adott elem szabad). A harmadik esetben az operációs kódot "megfejeljük" egy "P" betűvel is, hogy egyértelműbb legyen a dolog.

Aritmetikai utasítások

FADD — ST(1) értékét összeadja ST(Ø)-val; az eredmény az ST(Ø) helyén jelenik meg, felülírja annak eredeti értékét.

FADD op — "op" (memóriában levő lebegőpontos változó) értékét összeadja ST(Ø)-val; az eredmény ST(Ø) helyén jelenik meg, felülírja annak eredeti értékét.

FIADD op — "op" (szavas vagy rövid egész típusú memóriaváltozó) értékét összeadja ST(Ø)-val; az eredmény ST(Ø) helyén jelenik meg, felülírja annak eredeti értékét.

FADDP ST(i), ST(Ø) — ST(i) értékét összeadja ST(Ø)-val; az eredmény ST(i) helyén jelenik meg, felülírja annak eredeti értékét. ST(Ø) törlődik a stackről.

FSUB — ST(1) értékét kivonja ST(Ø)-ból; az eredmény az ST(Ø) helyén jelenik meg, felülírja annak eredeti értékét.

FSUB op "op" (memóriában levő lebegőpontos változó) értékét kivonja ST(Ø)-ból; az eredmény ST(Ø) helyén jelenik meg, felülírja annak eredeti értékét.

FISUB op "op" (szavas vagy rövid egész típusú memóriaváltozó) értékét kivonja ST(Ø)-ból; az eredmény ST(Ø) helyén jelenik meg, felülírja annak eredeti értékét.

FSUBP ST(i), ST(Ø)
ST(Ø) értékét kivonja ST(i)-ből; az eredmény az ST(i) helyén jelenik meg, felülírja annak eredeti értékét. ST(Ø) törlődik a stackről.

FSUBR ST(i) ST(Ø) értékét kivonja ST(i)-ből; az eredmény az ST(i) helyén jelenik meg, felülírja annak eredeti értékét. Ez az utasítás fordítottja az

FSUB ST(i)

utasításnak. Használata akkor kényelmes, ha az operandusok nem a kívánatos sorrendben állnak rendelkezésünkre.

FMUL ST(1) értékét összeszorozza ST(Ø)-val; az eredmény ST(Ø) helyén jelenik meg, felülírja annak eredeti értékét.

FMUL op "op" (memóriában levő lebegőpontos változó) értékét összeszorozza ST(Ø)-val; az eredmény az ST(Ø) helyén jelenik meg, felülírja annak eredeti értékét.

FIMUL op "op" (szavas vagy rövid egész típusú memóriaváltozó) értékét összeszorozza ST(Ø)-val; az eredmény ST(Ø) helyén jelenik meg, felülírja annak eredeti értékét.

FMULP ST(i), ST(Ø)
ST(i) értékét összeszorozza ST(Ø)-val; az eredmény ST(i) helyén jelenik meg, felülírja annak eredeti értékét. ST(Ø) törlődik a stackről.

FDIV — ST(0) értékét elosztja ST(1)-el; az eredmény az ST(0) helyén jelenik meg, felülírja annak eredeti értékét.

FDIV op — ST(0) értékét elosztja "op"-al (memóriában levő lebegőpontos változó); az eredmény ST(0) helyén jelenik meg, felülírja annak eredeti értékét.

FIDIV op — ST(0) értékét elosztja "op"-al (szavas vagy rövid egész típusú memóriaváltozó); az eredmény ST(0) helyén jelenik meg, felülírja annak eredeti értékét.

FDIVP ST(i), ST(0)
ST(i) értékét elosztja ST(0)-val; az eredmény az ST(i) helyén jelenik meg, felülírja annak eredeti értékét. ST(0) törlődik a stackről.

FDIVR ST(i) — ST(i) értékét elosztja ST(0)-val; az eredmény az ST(i) helyén jelenik meg, felülírja annak eredeti értékét. Ez az utasítás fordítottja az

```
FDIV    ST(i)
```

utasításnak. Használata akkor kényelmes, ha az operandusok nem a kívánatos sorrendben állanak rendelkezésünkre.

Számértékek összehasonlítása

FCOM — ST(0) és ST(1) értékét összehasonlítja; az eredményt a C3, C2 és C0 bitek vizsgálatából tudhatjuk meg.

FCOM op — ST(0) és "op" (memóriában levő lebegőpontos változó) értékét összehasonlítja; az eredményt a C3, C2 és C0 bitek vizsgálatából tudhatjuk meg.

FICOM op — ST(0) és "op" (memóriában levő szavas egész) értékét összehasonlítja; az eredményt a C3, C2 és C0 bitek vizsgálatából tudhatjuk meg.

FCOMP — ST(Ø) és ST(1) értékét összehasonlítja; ST(Ø) értékét elveszi a stackről. Az eredményt a C3, C2 és CØ bitek vizsgálatából tudhatjuk meg.

FICOMP op — ST(Ø) és "op" (memóriában levő szavas egész) értékét összehasonlítja; ST(Ø) értékét elveszi a stackről. Az eredményt a C3, C2 és CØ bitek vizsgálatából tudhatjuk meg.

FCOMPP — ST(Ø) és ST(1) értékét összehasonlítja, s mindkét elemet elveszi a stackről. Az eredményt a C3, C2 és CØ bitek vizsgálatából tudhatjuk meg.

Megjegyzés: az FCOMP és FCOMPP utasítások adják a leginkább kézenfekvő lehetőséget arra, hogy a stack legfelső (vagy legfelső két) elemét töröljük, méghozzá a koprocesszor stack pointerének visszahúzásával.

FTST — Megvizsgálja ST(Ø) értékét (azaz Ø.Ø-val hasonlítja össze). Az eredményt a C3, C2 és CØ bitek vizsgálatából tudhatjuk meg.

FXAM — Nem összehasonlítást, hanem vizsgálatot végez. ST(Ø) értéke alapján beállítja a feltételi biteket.

A stack tetején minden szokásos művelet után természetesen normalizált pozitív vagy negatív szám van. Az FXAM utasítást nem kell minden lebegőpontos művelet után kiadni, viszont mindig meg kell vizsgálnunk azt, hogy következett-e be valamilyen hiba, vagy sem (meg kell vizsgálnunk a koprocesszor státuszának megfelelő bitjeit). Csak akkor szükséges mélyebb vizsgálatba bonyolódnunk, ha a hibafeltételből nem világos a hiba. Ekkor kiadunk egy FXAM utasítást, és a hiba előfordulásának helyéből (tudván, milyen művelet váltotta ki) és a rendelkezésre álló információból kell felderíteni az okot, és eldönteni, hogy futhat-e tovább a program, vagy felhasználói beavatkozást kell kérni, esetleg vészes gyorsasággal ki kell lépnünk.

II.4.3. Lebegőpontos függvények

A koprocesszor nemcsak az alapműveletek elvégzésével, hanem számos matematikai függvény értékének kiszámításával is segíti munkánkat. Lássuk őket!

FSQRT — Négyzetgyök - ST(0) négyzetgyökét írja ST(0)-ba. Amennyiben ST(0) negatív volt, az eredmény érvénytelen lesz (lásd kivételek). Az utasítás előtt ST(0)-nak természetesen pozitívnak kell lennie.

FSCALE — Kettőhatvány - ST(0)-ba írja a kettő ST(1)-edik hatványát, ST(0)-val megszorozva. Ez képletben:

$$ST(0) = ST(0)*2^{ST(1)}$$

Az utasítás előtt ST(0) abszolútértékének a kettő tizenötödik hatványánál kisebbnek, ST(1)-nek pedig egésznek kell lennie. Ez az utasítás tehát ST(0)-t a kettő egész hatványaival szorozza meg.

FPREM — Parciális maradékképzés - ST(0)-at elosztja ST(1)-el, és ST(0) helyére a parciális maradékot írja. Megkeresi a legnagyobb olyan egész számot, amely kisebb, mint ST(0)/ST(1). Ezután "visszaszorozza" ST(1)-et ezzel az egész számmal, és a szorzatot kivonja ST(0)-ból.

Legyen kiindulási érték az ST(0)-ban 53.459, az ST(1)-ben pedig 17.32! Ekkor az FPREM függvény kiértékelése után a hányados 3.08554. Tehát hárommal szorzunk; a keletkező szorzat 51.96, ezt az eredeti értékből kivonva 1.499-et kapunk. Ez a parciális maradék.

FRNDINT — Egészre kerekítés - ST(0) helyére ST(0) egészre kerekített értékét írja. A kerekítés stratégiáját a koprocesszor vezérlőszavában szabhatjuk meg.

FXTRACT — Külön karakterisztikára és mantisszára bontja ST(0) értékét. Egy új elemet hoz létre a stacken. A művelet után ST(1) a szám mantisszáját, míg (az új) ST(0) a szám karakterisztikáját jelenti.

FABS — Abszolútérték-képzés - ST(0) helyére az eredeti szám abszolút értékét írja.

FCHS — Előjelváltás - ST(0) helyére az eredeti érték ellentettjét írja.

FPTAN — Parciális tangens – a művelet új elemet ír a stackre. Az új ST(Ø) egy "X", az ST(1) pedig egy "Y" érték; Y/X megegyezik az ST(Ø) eredeti értékének tangensével. A művelet végrehajtásához ST(Ø)-ba a kívánt szög ívmértékét írjuk; ennek Ø és a "pi" negyedrésze közé kell esnie. (Ennek teljesítésére jó az FPREM utasítás).

Megjegyzés: mint középiskolai matematikai tanulmányainkból tudjuk, a tangens ismeretében könnyűszerrel kiszámíthatjuk egy adott érték sinusát és cosinusát:

$$SIN^2(X) = \frac{TG^2(X)}{1 + TG^2(X)}$$

és

$$COS^2(X) = \frac{1}{1 + TG^2(X)}$$

Ebből gyökvonással nem nehéz a kívánt értékeket megkapni. Az arcsin(), arccos(), valamint az arctg() függvények között is hasonló összefüggések állnak fenn.

FPATAN — Parciális arcus tangens – ST(1)/ST(Ø) arcus tangensét tölti az ST(Ø) helyre (egy elem kikerül a stackből, így a művelet után a kurrens ST(Ø) a régi ST(1) lesz). Az utasítás előtt ST(Ø)-nak pozitívnak, ST(1)-nek pedig ST(Ø)-nál nagyobb véges értéknek kell lennie.

F2XM1 — Kettőhatvány számítása – ST(Ø) helyére a kettő ST(Ø)-adik hatványánál eggyel kevesebbet ír:

$$ST(Ø) = 2^{ST(Ø)} - 1$$

Ez a művelet csak akkor hajtható végre, ha az ST(Ø) értéke Ø.Ø és Ø.5 közé esett.

E függvény segítségével, felhasználva a konstansbetöltő utasításokat, könnyen kiszámíthatjuk a tíz és az "e" (a természetes logaritmus alapszáma) ST(Ø)-adik hatványát:

$$10^x = 2^{x * \log 10}$$
$$e^x = 2^{x * \log e}$$
$$y^x = 2^{x * \log y}$$

ahol a "log" persze a kettő alapú logaritmust jelenti. Tehát bármilyen érték kettő, tíz vagy "e" alapú logaritmusa ismeretében kiszámíthatjuk magát az értéket.

FYL2X — Logaritmusszámítás - ST(Ø) helyére ST(Ø) értékének kettő alapú logaritmusának és ST(1) értékének szorzatát írja:

$$ST(Ø) = ST(1) * LOG_2(ST(Ø))$$

Az utasítás végrehajtása után a régi ST(Ø) eltűnik a stackről, és a régi ST(1) helyére kerül az eredmény (most már természetesen ez lesz az ST(Ø) érték). A bemenő paraméterek közül ST(Ø) csak pozitív véges, míg ST(1) tetszőleges előjelű, de véges érték lehet.

Ez különösen alkalmas hatványok kiszámítására; ha ST(Ø) értékének ST(1)-edik hatványára vagyunk kíváncsiak, akkor egyszerűen kiadjuk az FYL2X utasítást, amely ST(Ø) logaritmusának és ST(1) értékének szorzatát helyezi el a stack tetején. Ezt már csak vissza kell számolni az F2XM1 függvénnyel, és megkaptuk a kívánt hatványt.

FYL2XP1 — Csaknem pontosan megegyezik FYL2X-el, csak a művelet előtt egyet ad ST(Ø) értékéhez. Ennek akkor van jelentősége, ha az ST(Ø) értéke közel van a nullához, vagyis a logaritmus értéke nagy negatív érték lenne. Ilyen esetben ez a művelet pontosabb számításokat tesz lehetővé.

$$ST(Ø) = ST(1) * LOG_2(ST(Ø) + 1)$$

Az utasítás végrehajtása után a régi ST(Ø) eltűnik a stackről, és a régi ST(1) helyére kerül az eredmény (természetesen most már ez lesz az ST(Ø) érték). A bemenő paraméterek közül ST(Ø) csak "kis" pozitív véges, míg ST(1) tetszőleges előjelű, de véges érték lehet. Mit jelent az, hogy ST(Ø) "kis" érték? Azt, hogy ST(Ø) abszolútértékben kisebb, mint kb. Ø.3; egészen pontosan:

$$Ø < ABS(ST(Ø)) < 1 - \frac{\sqrt{2}}{2}$$

II.4.4. Vezérlőutasítások

A vezérlőutasítások a koprocesszor tevékenységének szabályozását végzik. Általában nincs aritmetikai szerepük, bár közülük egyesek komolyan befolyásolják a koprocesszor aritmetikai működését is, tekintve, hogy vannak utasítások, melyek mentik, illetve betöltik a koprocesszor státuszát, azaz minden regiszterét. Ebbe beleértendő a teljes stack is, így ezek az utasítások gigantikus betöltő- és kiíróutasításoknak is tekinthetők.

FINIT		Előkészítés - a koprocesszor alapállapotba hozása; "software reset". A FINIT utasítás után a koprocesszor belső regiszterei mind alapállapotba kerülnek, a stack üres.
FENI		Interrupt engedélyezése - ahhoz, hogy a koprocesszor interruptot adjon valamilyen hiba bekövetkezése esetén, a vezérlőregiszter interrupt vezérlő bitjeinek beállítása mellett szükséges az is, hogy külön engedélyezzük az interruptot.
FDISI		Interrupt tiltása - ez az utasítás a vezérlőregiszter megfelelő bitjeinek beállításától függetlenül tiltja az interruptok beadását; újabb interruptokhoz ki kell adnunk a FENI utasítást.
FLDCW	addr	Vezérlőregiszter betöltése a memóriából - az "addr" címen levő szavas változó tartalmának bemásolása a koprocesszor vezérlőszavára.
FSTCW	addr	Vezérlőregiszter kiírása a memóriába - a koprocesszor vezérlőregiszterének kiírása az "addr" címen levő szavas változóba.
FSTSW	addr	Státuszregiszter kiírása - a koprocesszor státuszregiszterének kiírása a "addr" címen levő szavas változóba.
FCLEX		A kivételeket jelző bitek törlése - az utasítás a státuszregiszter hibajelző bitjeinek állásától függetlenül törli a biteket.
FSTENV	addr	A környezet kiírása - a koprocesszor belső regisztereinek kiírása az "addr" címen levő, tizennégy byte hosszúságú területre.

FLDENV addr — A környezet betöltése – az "addr" címen levő, tizennégy byte hosszúságú terület bemásolása a koprocesszor belső regisztereibe.

FSAVE addr — A státusz kiírása – a koprocesszor státuszának (a belső regiszteknek és a stack tartalmának) kiírása az "addr" címen levő, kilencvennégy byte hosszúságú területre.

FRSTOR addr — A státusz betöltése – az "addr" címen levő kilencvennégy byte hosszúságú terület bemásolása a koprocesszorba (a belső regiszterek, valamint a stack felülírása).

FINCSTP — A stack pointer növelése – az utasítás hatására eggyel megnövekszik a koprocesszor stack pointere; az így ST(0)-vá előlépett új elem persze változatlan marad (ezt a leíró szó megfelelő bitjei jelzik).

FDECSTP — A stack pointer csökkentése – az utasítás hatására eggyel csökken a koprocesszor stack pointere; a stack elemek tartalma változatlan marad (ezt a leíró szó megfelelő bitjei jelzik).

FFREE ST(i) — A stack "i"-edik eleme törlődik. A művelet nem érinti a stack pointert.

Megjegyzés: ha a stack legfelső elemét törölni akarjuk, akkor a következőképpen járhatunk el:

a) kiadunk egy FFREE utasítást az ST(0) elemre, majd visszahúzzuk eggyel a stack pointert:

```
FFREE   ST(0)
FDECSTP
```

b) lényegesen egyszerűbb, ha összehasonlítjuk az ST(0) elemet önmagával, egyúttal "pop"-olva az elemet:

```
FCOMP   ST(0)
```

FNOP — Nincs tevékenység

FWAIT — Az Intel 8086 WAIT utasításával egyenértékű; megvárja az aktuális koprocesszor-művelet befejezését.

II.5. Az assembler és a lebegőpontos aritmetika

Az assembly programozásban a lebegőpontos aritmetika használata nem barátságos dolog. Ha nincs koprocesszorunk, a feladat elég bonyodalmas; sajnos, olyan rutinkönyvtárra van szükségünk, amely nem terem minden bokorban.

II.5.1. Lebegőpontos aritmetika koprocesszorral

A koprocesszor használatához mindössze annyit kell tennünk, hogy a MASM assemblert (feltéve, hogy legalább a 3.00 verziót használjuk) a /R opcióval kell meghívni:

```
C>masm /R source,,,;
```

Ekkor a fordító szabályosan beépíti a lebegőpontos utasításokat, és a koprocesszor (ha működik) végrehajtja.

II.5.2. Lebegőpontos aritmetika emulátorral

Azt tapasztaltam, hogy a C fordítók is "befordítják" a koprocesszor utasításokat az elkészített OBJ file-ba, és ezek a linkelés során alakulnak (előttem teljesen ismeretlen úton) interrupt utasításokká. Ha debuggerrel nyomonkövetünk egy lefordított, leszerkesztett C nyelvű programot, akkor találkozhatunk a 38H, 39H stb. interruptok hívásával. Ebből szűrhetjük le azt a következtetést, hogy az EM.LIB ezekre a vektorokra tölti le az emuláció során meghívandó rutinok címeit.

Egy ideig hiába próbálkoztam azzal, hogy a Microsoft C fordítóprogramjához tartozó emulátor könyvtárat felhasználjam assembler programokhoz. Az első kísérletek a linkelés pillanatában elakadtak; hiába adtam meg a C nyelvű programokhoz szükséges LIB file-ok nevét, maradt néhány definiálatlan szimbólumom. Azt is megpróbáltam, hogy a C fordítóval létrehozott assembler file-t lefordítsam a MASM megfelelő verziójával. A fordítás sikeres volt, de a LINK továbbra is talált néhány definiálatlan hivatkozást. Végül is nagyon egyszerű lépés vezetett célra:

```
/* Az alábbi C program meghív egy assembly programot, melynek
 * belépési pontja kötelezően a _ENTRY rutin (típusa NEAR)
 */
extern  void    entry(void);    /* meghívandó assembly prog. */
double  x;                      /* csak hogy legyen EM.LIB   */
```

```
main()
{
        entry();
        exit( 0 );
} /* main */
```

Maga az assembly program, legalábbis annak kerete, az alábbi lehet (a mentések, előkészítő lépések célját illetően lásd "A hangszóró kezelése" fejezet példaprogramjait, valamint a 4. kötetet):

```
DATA    SEGMENT PARA    PUBLIC  'DATA'
        ...     ;A szüséges adatterületek definiciója
DATA    ENDS

_TEXT   SEGMENT BYTE    PUBLIC  'CODE'

_ENTRY  PROC    NEAR

                PUSH    BP
;               MOV     BP, SP
                PUSH    SI              ;A C által (elvben)
                PUSH    DI              ; használt regiszterek
                                        ;
                PUSH    DS              ;
                MOV     AX, DATA        ;
                MOV     DS, AX          ;Saját adatszegmens
        ASSUME  DS:DATA ;---------------;
                ...     ;A kívánt lebegőpontos műveletek
                POP     DS              ;DS helyreállítása
                POP     DI              ;Indexregiszterek
                POP     SI              ; helyreállítása
                POP     BP
                RET
_ENTRY  ENDP
_TEXT   ENDS
                END
```

A fenti program sikeres fordításához azt kell tudni, hogy a MASM-ot /E opcióval kell elindítani:

```
        C>masm /E source,,,;
```

Ellenkező esetben a fordító nem ismeri a lebegőpontos utasításokat, és a lebegőpontos konstansokat a koprocesszor (és az

emulátor) által megkívánt forma helyett az elterjedt IEEE szabvány szerint fordítja. A magasszintű programozási nyelvek és az assembler kapcsolatáról egyébként bővebben a negyedik, utolsó kötetben olvashatunk.

Ezzel a trükkel ugyan sokkal, de sokkal hosszabb lesz az elkészült EXE file (hiszen rengeteg olyan "apróságot" szerkesztünk a programunkhoz, mint a C programok teljes be- és kiléptető modulja, valamint egy sor olyan könyvtári modul, amely felesleges vagy legalábbis annak látszik), viszont könnyedén valósíthatjuk meg assemblerben a legvadabb lebegőpontos műveleteket is. Elképzelhető egyébként, hogy saját kezünkbe véve az emulátor vezérlését, lényegesen gyorsabb és hatékonyabb programokat írhatunk, mint C nyelvben, hiszen amíg ott csak a nyelv által megvalósított aritmetikai műveletekhez férhetünk, assembler szinten nincs akadálya a teljes függvénykészlet és a vezérlőutasítások éles kihasználásának.

III. A klaviatúra

III.1. A klaviatúra működése

Az IBM PC igen "intelligens" klaviatúrával rendelkezik (ezt mi sem mutatja jobban, mint hogy belső vezérlésére külön mikroprocesszort használnak). A BIOS a következőképpen kezeli a klaviatúrát:

a klaviatúra 83 billentyűből áll, melyen minden billentyűhöz egy kód tartozik (scan-code). Ezt a továbbiakban billentyűkódnak nevezzük. A billentyűkód egy szám, amely a billentyű sorszámának tekinthető. Bármely billentyű lenyomása és felengedése hardware-interruptot vált ki, melynek végrehajtása során a billentyűkód egy ciklikusan kezelt bufferbe íródik be, és ha a program a klaviatúráról olvas, akkor tulajdonképpen ezt a buffert olvassa, és csak akkor vár fizikai inputra, ha a buffer üres.

A klaviatúra hardware interrupt rutinja a bufferbe a billentyűkóddal együtt beírja a billentyűhöz rendelt ASCII kódot is. Ennek kialakításában még egy tényező játszik szerepet, a shift billentyűk állása. Az "A"-val jelölt billentyű lenyomása eredményezheti az "a" (kódja 97), az "A" (kódja 65) vagy az 1 kódú CTRL-A karaktert is attól függően, hogy magában, vagy a SHIFT-tel, vagy a CTRL-el együtt nyomtuk meg, sőt, hogy a CAPS LOCK billentyűt lenyomtuk-e korábban, vagy sem. A rutin egy belső változóban állandóan nyilvántartja e billentyűk állását, és ennek alapján "keveri ki" az adott billentyűkódhoz rendelt ASCII kódot. Ezt a változót a továbbiakban shiftstátusznak hívjuk.

A klaviatúra tehát egyrészt elszakítja a program által kért beolvasást a valódi, fizikai inputtól, így ez időben megelőzheti a programigényt, "előre" lehet gépelni néhány (legf. 15) karaktert, másrészt elválasztja a billentyű fizikai helyét a hozzárendelt kódtól, így viszonylag könnyen rendelhetünk bármely billentyűhöz bármilyen jelentést, mivel mindig lekérdezhetjük a shiftstátusz tartalmát.

Ugyanakkor a klaviatúra-drivernek van egy olyan elvi hibája, amely nagyon ritkán okoz valódi zűrzavart, de elvi hiba az elvi hiba. Ez pedig a következő: mivel a shiftstátuszt tartalmazó változó mindig a pillanatnyi állapotot jelzi, a klaviatúra 15 leütés "nyomának" befogadására képes buffere pedig viszonylag régen leütött billentyű jelentését is tartalmazhatja, az általunk kiolvasott, már régen érkezett billentyűkódnak és a pillanatnyi shiftstátusznak nem sok köze van egymáshoz. Előfordulhat, hogy egy adott shift-állapot mellett leütött billentyű jelentésének kiolvasásakor a shiftstátusz már megváltozott. Az

eredeti státuszt pedig már nem kaphatjuk meg, tehát ha magunk akarjuk a billentyű jelentését a kódból és a shiftstátuszból kikeverni, akkor eltévedhetünk. Ezen a hibán csak a klaviatúra driverének újraírása segíthet.

III.2. A klaviatúrakezelő rutinok elvi működése

A klaviatúra kezelése két részre választva, egy hardware- és egy software-interrupt segítségével történik. A klaviatúrát fizikailag a 8255 PPI vezérli. Ez az áramkör interruptot generál a billentyű lenyomásakor és felengedésekor is. Mikor lenyomjuk a billentyűt, a PPI első portjáról a billentyű kódját lehet beolvasni, míg a billentyű felengedésekor a billentyűkód 80H-val (128-al) megnövelt értékét. Ha a felhasználó huzamosabb ideig tart lenyomva egy billentyűt, ez ismét interruptot okoz, mindig a lenyomáskor keletkező normál billentyűkódot adva be.

Ha egyszerre több billentyűt tartunk lenyomva, akkor mindig az utoljára lenyomott billentyű okoz ismételt interruptot. Ha ezt felengedjük, akkor már a többi, még lenyomva tartott billentyű sem generál újabb interruptot. "Normál" billentyű (lásd a következő bekezdést) interruptsorozatát megszakítja bármely shift billentyű lenyomása; ha shift mellett nyomunk normál billentyűt, akkor a shift addig van érvényben, amíg fel nem engedjük. Ha eközben a normál billentyűt folyamatosan nyomva tartjuk, akkor jelentése megváltozik (pl. "D"-ből "d" lesz).

A billentyűket szerepük szerint két csoportra oszthatjuk. Az egyik csoport a közönséges billentyűk csoportja (az összes karakterbillentyű, valamint a funkcióbillentyűk és a numerikus billentyűzet elemei). A másik csoport a shift billentyűk csoportja. Bár minden billentyű pontosan ugyanolyan hardware interruptot vált ki, természetes, hogy a rutin egészen másként kezeli ezt a két csoportot. Tekintsük először a shift billentyűket. Ezek a következők: SHIFT LEFT (bal shift), SHIFT RIGHT (jobb shift), CONTROL, ALTMODE, valamint a CAPS LOCK, a SCROLL LOCK, a NUM LOCK és az INSERT. A "valamint" szó e csoport két fontos alcsoportját választja el egymástól: a "shift"-eket (eltolásokat) és a "toggle"-ket (kapcsolókat). A shiftek csak addig vannak érvényben, ameddig lenyomva tartjuk őket. Az eközben generált ismétlési interruptokat a kezelő rutin "lenyeli". Ha valamelyik shiftbillentyű leütésére utaló billentyűkód érkezik, akkor a rutin automatikusan egyre állítja a shiftstátusz megfelelő bitjét, és ezt mindaddig fenntartja, amíg a billentyű felengedését jelző interrupt meg nem érkezik.

A normál billentyűk körében is két csoport van. Az egyiket nevezzük ASCII-csoportnak, a másikat funkciócsoportnak. Az első csoport elemei azok a billentyűk, amelyek valamilyen ASCII-kódkészlethez tartozó karaktert adnak be. Ezek a billentyűk nagyjából a klaviatúra központi, legnagyobb terjedelmű részén találhatók. A funkcióbillentyűk csoportjába az F1, F2, ... F10 billentyűk, valamint a jobb oldali numerikus billentyűzet elemei tartoznak. Ha ezek bármelyikét lenyomjuk, akkor az interrupt rutin beírja ezek ASCII- és billentyűkódját ciklikus bufferébe. Az ASCII-billentyűk esetén ez világos: a szavas bufferbejegyzés alsó byte-ja az ASCII-kód, a felső a billentyűkód. A funkcióbillentyűknek nincs ASCII-kódja. Ezért e csoport tagjainak ASCII-kódjaként 0-t helyez el a bufferben, míg a billentyűkód megmarad. Az is külön érdeklődésre tarthat számot, hogy a SHIFT-el, ALTMODE-al vagy CONTROL-al kombinált funkcióbillentyűket más (83-nál magasabb) billentyűkóddal helyezi el a bufferben (részletes ismertetését lásd a függelékben).

> Megjegyzés: a klaviatúrakezelő hardware interrupt rutin érdekes és tanulságos olvasmány; egyfelől, mert ismerete hasznos, ha meg akarjuk érteni, hogy a klaviatúra miért úgy működik, ahogy; másfelől, mert sok elrettentő programozási példát láthatunk benne.
> Ami rögtön szemet szúr: a rutinok belső belépési pontjait a "K1", "K2" stb. címkékkel látták el. A sorszámozás hatvanháromig tart. Ezek az igazán szellemes, és a megcímzett objektum funkciójára oly kifejezően utaló nevek igencsak "megkönnyítik" a program olvasását. Rengeteg az oda-vissza ugrálás a rutinban, annál kevesebb a belső szubrutinhívás. Igen sok "mágikus" számot olvashatunk, például:
>
> ```
> CMP     AL, 59
> ```
>
> ahol is az olvasó némi rendszerismerettel elég hamar rájön, hogy ez az 59 nem más, mint a funkcióbillentyűk csoportjának legelső eleme, de... A szempont valószínűleg az volt, hogy a Microsoft úgy adja ki kezéből az információt, hogy mégse adja ki.

A klaviatúra hardware interruptjának sorszáma 09H. Ha valami ravaszabb trükköt akarunk végezni (például a SideKick "forró billentyűjéhez" hasonlóan speciálisan kezelni egy-egy billentyűt vagy kombinációt), át kell vennünk ezt az interruptot. A fejezet végén egy egyszerű klaviatúrakezelő program szerepel, melyet mintául használva el lehet indulni. Mindenesetre ügyelni kell a gyorsaságra!

III.3. Klaviatúrakezelési funkciók

Interrupt-sorszám: 16H
Javasolt interrupt-név: BS_KBD

A klaviatúra kommunikációs interruptjának hívásához javasolt makró (alfunkció nincs):

```
KEYBCL  MACRO   CODE                    ;
                MOV     AH, CODE        ;;
                INT     BS_KBD          ;
        ENDM                            ;
```

III.3.1. Karakter beolvasása

Funkciókód: AH = 00H INT 16H
Javasolt konstansnév: KBD_CHRIN
Be:
Semmi egyéb
Válasz:
AL = ASCII-kód,
AH = billentyűkód

E funkció szinkron olvasást végez. A vezérlést csak akkor kapjuk vissza, ha a karakter beolvasása befejeződött. Ha a bufferben volt már legalább egy karakter, a rutin azt adja fel. Ha a buffer üres, akkor addig vár, míg nem jön egy karakter.

III.3.2. Buffer lekérdezése

Funkciókód: AH = 01H INT 16H
Javasolt konstansnév: KBD_CHRASK
Be:
Semmi egyéb
Válasz:
ZF = 1 ha a buffer üres
ZF = 0 ha a buffer nem üres
AL = ASCII-kód
AH = billentyűkód

Ha a buffer üres, akkor csak a Zero flagben kapunk választ. Ha nem, akkor nem csak ennek jelzését kapjuk a Zero flagben, de magát a karaktert is (AX-ben); a bejegyzés azonban bent marad a bufferben, kiolvasásához meg kell hívnunk a 01 funkciót.

III.3.3. Shiftstátusz lekérdezése

Funkciókód: AH = 02H INT 16H
Javasolt konstansnév: KBD_SHFTST

Be:

Semmi egyéb

Válasz:

AL = shiftstátusz (a megfelelő bit 1 akkor, ha a shift aktív):

7	6	5	4	3	2	1	0
Insert state	Caps-lock	Num-lock	Scroll lock	Alt-mode	Control	Norm. left shift	Norm. right shift

A shifteket bemutató táblázatból jól látható, hogy a klaviatúra nyolc shift jellegű billentyűje a következő:

- Normál Shift - a nagy- és kisbetűk, számok és írásjelek közti váltásra szolgál. Ennek két változata van: a jobb oldali és bal oldali shift állását külön le lehet kérdezni és más jelentőséget tulajdonítani neki;
- Control - hatására a rendszer a billentyűkhöz rendelt vezérlőkódokat adja fel (pl. a Ctrl-A hatására az '1' ASCII-kód keletkezik);
- Altmode - ezt a billentyűt a rendszer nem használja ki igazán. A Controlhoz hasonló funkciókra használhatjuk (a BIOS a CTRL-ALT-DELETE kombinációt értelmezi újraindítási parancsgyanánt; a folyamat teljes leírását lásd a fejezet végén);
- Scroll Lock - a Control-al együtt használható; a BIOS a Ctrl és a Scroll-Lock kombinációra a Ctrl-C-t (a 3 ASCII-kódot) adja be, amely a futó program abortálására szolgál. A folyamat teljes leírását lásd a fejezet végén;
- Num Lock - a numerikus billentűzet kezelése során használatos a cursorvezérlő funkciók (nyilak) és a számjegyek közti váltásra. Másik fontos funkciója a Ctrl-Num Lock kombináció, mely a futó program felfüggesztésére szolgál. A folyamat teljes leírását lásd e fejezet végén;
- Caps Lock - a bal és jobb oldali shiftek rögzítését jelenti. Ha a Caps Lock billentyűt lenyomjuk, akkor nagybetűk jönnek kisbetűk helyett, de a számok maradnak; ha a Caps Lock aktív

állapotában Shift-et is nyomunk, akkor minden megfordul: az alapértelmezett nagybetűk helyett kisbetűket kapunk, a számok helyett a megfelelő írásjeleket.
- Insert - főként szövegszerkesztő funkciók megvalósítására használják: aktív állapotában a legtöbb szövegszerkesztő beszúrást, inaktív állapotában pedig felülírást hajt végre.

A klaviatúra bufferének törlése (ami egy program belépésekor fontos lehet) csakis a 0CH sorszámú, több alfunkciós MS-DOS hívással lehetséges; használjuk e funkcióval a 06H alfunkciót (lásd 2. kötet)! Eléggé meglepő, hogy a ROM BIOS erre nem ad lehetőséget (pedig inkább az ő dolga volna, mint az MS-DOS-é).

III.4. Speciális klaviatúrafunkciók

Ezen a helyen három kardinális fontosságú klaviatúrafunkcióval foglalkozunk: a CTRL-ALT-DEL, a CTRL-BREAK és a CTRL-NUM kombinációval. Mind a három elég lényeges ahhoz, hogy külön alfejezetet szenteljünk nekik.

III.4.1. A CTRL-ALT-DEL kombináció

A CTRL-ALT-DEL kombináció az operációs rendszer újratöltésére, ún. melegindításra szolgál. Minden olyan esetben segítségünkre lehet, mikor a klaviatúra hardware-interruptja be tud jutni (azaz nincs letiltva az interruptok fogadása, nincs kikapcsolva a 8259 interrupt-vezérlő, és a hardware interrupt rutin nem sérült meg).

Ha a klaviatúra hardware-interrupt rutinja mindhárom billentyű lenyomását érzékeli, akkor minden külön értesítés helyett (egy globális változóba egy rögzített értéket írván) elugrik a ROM BIOS RESET nevű belépési pontjára.

> Megjegyzés: nem, kedves Olvasóm, nem. Már látom felcsillanó szemeit, de... ön tisztességes úton ugyanezt nem teheti meg, mert a ROM BIOS belső belépési pontjáról van szó. A listából tudhatja, hogy ez hol van, de ki nem használhatja, hacsak nem akar saját egyetlen és változhatatlan gépéhez ragadni. A rendszer megalkotói fenntartják a jogot, hogy a ROM BIOS belsejét minden értesítés nélkül megváltoztassák. Itt két megoldás van:
> a) használjuk ki, hogy a 19H interrupt vektor a melegindítás végző rutinra mutat. Ha progra-

munkban felismerjük az azonnali kilépés sürgős szükségét, meghívhatjuk ezt az interruptot. Ez tisztességes és hordozható megoldás (csak IBM AT-ra nem mindig vihető át...)

b) tudjuk, hogy "hardware reset" esetén a processzor az FFFFØH abszolút címre adja a vezérlést. Itt áll elő az a különös helyzet, hogy gépünket könnyebb programból újraindítani, mint az (egyébként hiányzó) frontpanelről, mivel az IBM PC/XT fájdalmasan nélkülözi a RESET gombot. Ha úgy érezzük, hogy semmi más nem segít, akkor ide is átadhatjuk a vezérlést...

III.4.2. A CTRL-BREAK kombináció

Valószínűleg ez a leggyakrabban használt kombináció, mivel ez lövi ki a "szerencsétlenül járt" programokat. Amennyiben a klaviatúra hardware-interrupt rutinja még él, akkor van esélyünk a sikerre. Mi történik ekkor?

(1) az interrupt-rutin alapállapotba hozza a ciklikus buffert, és elhelyez benne egy (Ø,Ø) kódpárt;

(2) meghívja az 1BH interrupt vektor által címzett felhasználói rutint (alapértelmezésben ez egy ártatlan kis IRET utasításra mutat);

(3) visszaadja a vezérlést a hívónak. Ha a hívó program még valamelyes életjelenségeket mutat, és meghív egy olyan MS-DOS funkciót, amely teszteli a CTRL-BREAK feltételt, akkor az MS-DOS aktivizálja a 23H interruptot, amely teljesíti kötelességét. Ha emögött az eredeti MS-DOS kód van, akkor az abortálni fogja a programot. Ha viszont a felhasználó "magához szólította" a 23H interruptot, akkor az fog történni, amit ő akar.
Mely MS-DOS funkciók tesztelik a CTRL-BREAK bekövetkezését? A legtöbb standard file-kezelő funkció (kivéve a Ø6H és Ø7H funkciókat), valamint az összes többi akkor, ha az MS-DOS ún. BREAK kapcsolója be van állítva.

III.4.3. A CTRL-NUM kombináció

Programozási szempontból talán ez a legérdekesebb kombináció, mivel a megszakított megszakítási rutin szabadítja ki a processzort a felfüggesztett programvégrehajtás állapotából.

Ha a hardware interrupt rutin a CTRL-NUM kombinációt érzékeli, akkor

(1) beállít egy flaget egy belső változójában, amely jelzi majd a várakozási állapotot;
(2) színes monitor használata esetén saját kebelében található utasításokkal leállítja a képernyővezérlőt, hogy ne fusson a kép;
(3) felszabadítja az interrupt rendszert, és addig vár a rutin belsejében egy (kívülről vezérelt) pár utasításos cikluson, míg "valaki" vissza nem állítja a flaget.

Ki lehet ez a királyfi, aki felébreszti álmából a programot? Nem lehet senki, hiszen a vezérlés végig a CTRL-NUM-ot érzékelő klaviatúra interrupt rutinnál van. Nem lehet senki más, mint... maga a klaviatúra interrupt rutin, második kiadásban. Ha bármelyik billentyűt leütjük a CTRL-NUM után, egy újabb interrupt éleződik. Mivel az interruptrendszer teljes egészében fel van szabadítva, a rutinnak egy másik példánya kezd futni; megnézi, áll-e a HOLD (a várakozási) flag. Ha áll (most pedig bizony áll), akkor megnézi, nem a NUM billentyűt nyomtuk-e meg. Ha igen, akkor semmit nem tesz, visszatér; ha nem, ha bármi egyebet nyomtunk le, akkor törli a HOLD flaget, és visszatér megszakított önnön magához. Itt újra megvizsgálja, hogy áll-e a HOLD flag. Mivel előzőleg a másik példány törölte, a flag 0, és így az első interrupt rutin (végre) visszatér ahhoz a programhoz, amit ő megszakított.

III.4.4. Az ALT billentyű és a numerikus billentyűzet

Az IBM PC/XT klaviatúrakezelő programja egyéb igen hasznos szolgáltatásokat is nyújt. Amennyiben az ALT billentyűt folyamatosan lenyomva tartjuk, és a numerikus billentyűzeten (a klaviatúra jobb oldalán) begépelünk egy (többjegyű) számot, azt a hardware interrupt rutin decimális számsorozatként konvertálja, és azt az ASCII-kódot helyezi el a klaviatúrabufferben, amelynek kódját beírtuk. Ha például az ALT folyamatos lenyomva tartásával leütjük a '6', majd az '5' billentyűket (azaz a 65 számot gépeljük be), a következő olvasás az 'A' karaktert fogja beadni. Ilyen módon minden egyes számról eldönthetjük, hogy milyen karakternek a kódja - és mi jelenik meg a képernyőn akkor, ha azt a karakterkódot írnánk ki.

III.4.5. A CTRL-PRTSC kombináció

A CTRL-PRTSC kombináció a képernyő tartalmának kinyomtatására szolgál. Ha a klaviatúrakezelő hardware interrupt rutin ezt a kombinációt észleli, a következők történnek:

(1) a klaviatúrarutin felszabadítja az interruptrendszert, és aktivizálja az 5. interruptot.

(2) az 5. interrupt-rutin lekérdezi a képernyő üzemmódját, lekérdezi és elmenti a cursor pillanatnyi pozícióját, majd a képernyő-pozíciók sorról sorra való kiolvasásával "megszerzi" a képernyőtartalmat karakterről karakterre, és (minden sor előtt egy kocsivissza-soremelést nyomtatva) minden karaktert kiküld a nyomtatóra.
Az 5. interrupt rutin a képernyő tartalmának kiolvasására a video-interruptot, míg a karakterek nyomtatására a nyomtató kommunikációs interruptját használja.

A hard copy egyik komoly problémája a sokféle "time-out", azaz azok a várakozási idők, amelyek megszabják, mennyit kell várnia a nyomtató interrupt rutinjának abban az esetben, ha a nyomtató nincs bekapcsolva.

Ha egyáltalán nincs nyomtatóvezérlő áramkör a gépben, akkor ezt a nyomtató interrupt rutin azonnal érzékeli, és minden különösebb késleltetés nélkül kilép. Azt azonban nem érzékelheti, ha a nyomtató nincs bekapcsolva. Amennyiben tehát van nyomtatóvezérlő áramkör a gépben (azaz a berendezés bekapcsoláskor történt ellenőrzése észlelte az áramkört), de a nyomtató ki van kapcsolva, akkor a ROM BIOS nyomtatókezelő rutinja kiküldi a karaktert a megfelelő portra, majd megvárja a nyomtatás befejezését. CX-et nullázza, majd értékét egyesével csökkentve addig olvasgatja a nyomtatóvezérlő státuszát, ameddig CX el nem fogy. Ráadásul az egész procedúrát tízszer ismétli meg. Ez pedig minden egyes képernyő-pozíció esetén végbemegy, tehát ha a hard copy kiadása előtt elmulasztottuk bekapcsolni a nyomtatót, akkor jó sokáig (percekig) várhatunk, mire gépünk ismét hajlandó velünk foglalkozni.

III.5. A klaviatúra interruptja - példaprogram

Az alábbiakban megadott példaprogram önmagában nem igazán használható, de egyfelől jól szemlélteti azokat a teendőket, amelyeket el kell végeznünk, ha saját kezelésünkbe kell vennünk a klaviatúra hardware-interruptját, másfelől példát ad a képernyő direkt írására - tisztességesen. Még egy nagy erénye van: bemutatja és bizonyítja, hogy minden billentyű egyenlő, és csak a driver tesz közöttük különbséget.

A program annyit tesz, hogy az ernyő törlése után kirajzolja a billentyűzet sematikus vázlatát az ernyőre, elkéri a klaviatúra hardware-interruptját, és ha megnyomunk egy billentyűt, akkor a képernyő megfelelő pozícióin megfordítja az attributumot, hogy az eredeti inverze legyen. Egy billentyű képe akkor változik vissza az eredetire, ha elengedjük. Látni fogjuk, hogy egyszerre több billentyűt is lenyomva tarthatunk, és hogy minden egyes billentyűre ugyanaz történik.

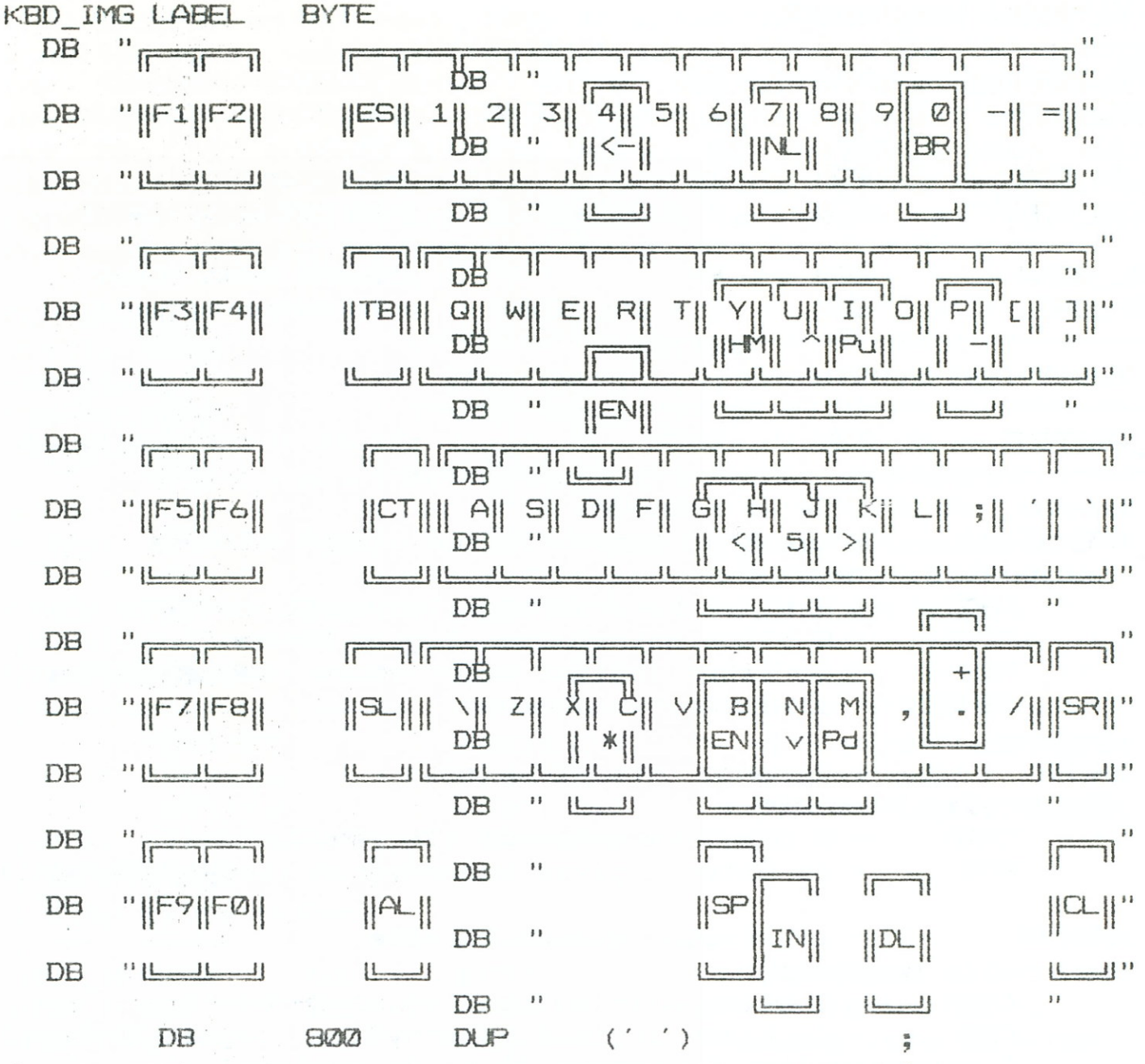
```
COMMENT *
A file neve:                        kbsc.asm
Fordítás:                   MASM kbsc[,,,];
Szerkesztés:                LINK kbsc[,,/M,];
Futtatás:                   kbsc
        A program a CTRL-ALT-DELETE kombinációra lép ki.
*
;_TEST            equ     1       ;Beállítandó teszteléshez
                                    ;-------;
STACK_SIZE      EQU     100H            ;A stack hossza
;-------------------------------------------;
; Inklúziók
;-------------------------------------------;
        INCLUDE         DOSCLL.INC      ;IT-vektor kezeléshez
        INCLUDE         IT-MOD.INC      ;Makrók miatt
        INCLUDE         VIDEO.INC       ;Direkt képernyőkezés
;-------------------------------------------;
; Konstansok
; Display konstansok
;-------------------------------------------;
ATTR_NPRESS     EQU     017H    ;Alap-attributum
ATTR_PRESS      EQU     074H    ;Nyomott billentyű attr.
ROW_SIZE        EQU     050H    ;Sorhosszúság karakterekben
                                ;Képernyősorok kezdőpozíciói
IRP     X, <0,1,2,3,4,5,6,7,8,9,10,11,12,13>
        ROW_&X  EQU     X * ROW_SIZE
ENDM                            ;           (rövid sorok miatt
IRP     X, <14,15,16,17,18,19,20,21,22,23,24>
        ROW_&X  EQU     X * ROW_SIZE
ENDM                            ;            kettéválasztva)
;-------------------------------------;
; Keyboard konstansok
;-------------------------------------------;
KB_DATA         EQU     60H             ;Adatport
KB_CTL          EQU     61H             ;Vezérlőport
        KB_RESET        EQU     80H     ;Reset parancs
```

```
; Kilépési konstansok
;-----------------------------------------;
CTRL_SCAN       EQU     29              ;CTRL scan kódja
CTRL_OFFS       EQU     2 * CTRL_SCAN   ; és tábla-offsetje
                                        ;
ALTM_SCAN       EQU     56              ;ALT scan kódja
ALTM_OFFS       EQU     2 * ALTM_SCAN   ; és tábla-offsetje
                                        ;
DEL_SCAN        EQU     83              ;DELETE scan kódja
DEL_OFFS        EQU     2 * DEL_SCAN    ; és tábla-offsetje
;-----------------------------------------;
; Tesztelési konstansok
;-----------------------------------------;
K_TESTINT       EQU     0FFH            ;Teszt-interrupt száma
;-----------------------------------------;
; Makrók, struktúrák és rekordok
; WRCHAT        - karakter és attributum direkt kiírása
;  Opcionális bemenetek:
;       CHAR    - 8 bites karakterkód (legyen CL-ben!)
;       ATTR    - 8 bites attributum (legyen CH-ban!)
;       POS     - 16 bites logikai képernyőcím (legyen BC-ben!)
;       DSPSEG  - a képernyő szegmense
;
; WRCHR         - karakter direkt kiírása
;  Opcionális bemenetek:
;       CHAR    - 8 bites karakterkód (legyen CL-ben!)
;       POS     - 16 bites logikai képernyőcím (legyen BX-ben!)
;
; WRATTR        - attributum direkt kiírása
; Opcionális bemenetek:
;       ATTR    - 8 bites attributum (legyen CH-ban!)
;       POS     - 16 bites logikai képernyőcím (legyen BX-ben!)
;       E makrókat lásd "A képernyő" c. fejezetben!
;
; Billentyűleíró rekord

KBD_PRESS    RECORD  PRESS:1, REPEAT:1, CHANGE:1, RES:1, POS:12

;-----------------------------------------;
; Adatterületek

DATA    SEGMENT PARA    PUBLIC  'DATA'
; KBD_IMG       - a klaviatúra képe karakterekkel;
;                 a sorokat helyszűke miatt törtem el
;-----------------------------------------;
```



```
KBD_IMG LABEL   BYTE
        DB      800     DUP     (' ')                   ;
;-------------------------------------------------------;
; KBD_PRESS_TAB - lenyomva tartott billentyűk adminisztrációja
;       128 darab KBD_PRESS rekordból álló táblázat
;       Tartalma: a címmel adott scan-kódú karakter helye
;                 a képernyőn, a sematikus ábrában
;-------------------------------------------------------;
                                                        ;
KBD_PRESS_TAB   LABEL   WORD                            ;
        DW      0FFFFH                                  ; 0 scan-kód
                                                        ;
        IRP     X, <12,15,18,21,24,27,30,33,36,39,42,45,48,54>
                DW      X + ROW_1                       ;  1 - 14
        ENDM                                            ;
```

```
IRP     X, <12,16,19,22,25,28,31,34,37,40,43,46,49>
        DW      X + ROW_4           ; 15 - 27
ENDM                                ;
        DW      55 + ROW_5          ; !! 28 !!
IRP     X, <13,17,20,23,26,29,32,35,38,41,44,47,50>
        DW      X + ROW_7           ; 29 - 41
ENDM                                ;
IRP     X, <12,16,19,22,25,28,31,34,37,40,43,46,50,55>
        DW      X + ROW_10          ; 42 - 55
ENDM                                ;
IRP     X, <13,31,50>               ;
        DW      X + ROW_13          ; 56 - 58
ENDM                                ;
IRP     X, < 1, 4>                  ;
        DW      X + ROW_1           ; 59 - 60
ENDM                                ;
IRP     X, < 1, 4>                  ;
        DW      X + ROW_4           ; 61 - 62
ENDM                                ;
IRP     X, < 1, 4>                  ;
        DW      X + ROW_7           ; 63 - 64
ENDM                                ;
IRP     X, < 1, 4>                  ;
        DW      X + ROW_10          ; 65 - 66
ENDM                                ;
IRP     X, < 1, 4>                  ;
        DW      X + ROW_13          ; 67 - 68
ENDM                                ;
IRP     X, <63,71>                  ;
        DW      X + ROW_1           ; 69 - 70
ENDM                                ;
IRP     X, <62,65,68,74>            ;
        DW      X + ROW_4           ; 71 - 74
ENDM                                ;
IRP     X, <62,65,68>               ;
        DW      X + ROW_7           ; 75 - 77
ENDM                                ;
        DW      74 + ROW_9          ; !! 78 !!
IRP     X, <62,65,68>               ;
        DW      X + ROW_10          ; 79 - 81
ENDM                                ;
IRP     X, <65,71>                  ;
        DW      X + ROW_13          ; 82 - 83
ENDM                                ;
DW      44      DUP     ( -1 )      ;"maradék" scan-kódok
```

```
   SIZ_KBD_PRESS_TAB   EQU   ( $ - KBD_PRESS_TAB ) / 2
                     ;-------------------------;
                     ; Mentőterületek
                     ;-------------------------;
KBD_INT_SAVE     DD       0                    ;Eredeti kbd interrupt
KBD_INT_OWN      DD       KBD_INTR             ;Saját kbd interrupt
DATA     ENDS                                  ;
                                               ;
;---------------------------------------------;
; Stack
;---------------------------------------------;
STACK    SEGMENT PARA     STACK    'STACK'
         DW       STACK_SIZE       DUP      ( ? )
STACK    ENDS
;---------------------------------------------;
; Kódterület
;---------------------------------------------;
                                               ;
CODE     SEGMENT PARA     PUBLIC   'CODE'      ;
         ASSUME  CS:CODE                       ;
                                               ;
START:                                         ;
                 MOV      AX, SEG DATA         ;
                 MOV      DS, AX               ;DS -> DATA szegmens
         ASSUME  DS:DATA                       ;
                                               ;
                 CALL     SCKB_INI             ;Képernyő előkészítése
                                               ;
         ifdef   _TEST                         ;Ha tesztelünk, akkor
                 S_ITV    K_TESTINT, KBD_INT_OWN, KBD_INT_SAVE
                          mov      al, 16h ; szimuláljuk a klavia-
                          int      k_testint
                          mov      al, 16h ; túra interruptját
                          int      k_testint
                          mov      al, 17h ;
                          int      k_testint
                 CALL     KBD_SHOW             ;
                          mov      al, 96h ;
                          int      k_testint
         else
                 S_ITV    BH_KBD, KBD_INT_OWN, KBD_INT_SAVE
         endif                                 ;
                 CALL     KBD_SHOW             ;Várakozás és a nyomott
                                               ; billentyűk jelzése
                                               ;
```

```
        ifdef   _TEST                           ;-------;
                R_ITV   K_TESTINT, KBD_INT_SAVE ;IT vissza
        else                                    ;
                R_ITV   BH_KBD, KBD_INT_SAVE    ;
        endif                                   ;-------;
                CALL    SCR_ERASE               ;Képernyőtörlés
                EXIT    0                       ;Kilépés
                                                ;
;-----------------------------------------------;
; SCR_ERASE     - ernyőtörlés direkt kiírással
;-----------------------------------------------;
SCR_ERASE       PROC    NEAR                    ;
                                                ;
                PUSH    ES                      ;
                PUSH    AX                      ;
                PUSH    CX                      ;
                PUSH    DI                      ;
                                                ;
                MOV     AX, SEG DISPLAY         ;
                MOV     ES, AX                  ;Képernyőter szegmens,
        ASSUME  ES:DISPLAY                      ;-------;
                MOV     DI, OFFSET DSP_ST       ;offset, ka-
                MOV     AX, 0720H               ;rakter és attr
                MOV     CX, DSP_SIZE_WORD       ;képernyőhossz
                CALL    SCR_DV                  ;-------;Video kikapcs.
        REP     STOSW                           ;Direkt kiírás
                CALL    SCR_EV                  ;Video visszakapcsolás
                                                ;
                POP     DI                      ;
                POP     CX                      ;
                POP     AX                      ;
                POP     ES                      ;
        ASSUME  ES:NOTHING                      ;
                                                ;
                RET                             ;
SCR_ERASE       ENDP                            ;
;-----------------------------------------------;
; SCKB_INI      - képernyő előkészítése a bemutatásra
;-----------------------------------------------;
SCKB_INI        PROC    NEAR                    ;
                PUSH    ES                      ;
                PUSH    AX                      ;
                PUSH    CX                      ;
                PUSH    SI                      ;
                PUSH    DI                      ;
```

```
                MOV     AX, SEG DISPLAY ;
                MOV     ES, AX          ;Képernyő szegmense
        ASSUME  ES:DISPLAY              ;
                                        ;
                MOV     DI, OFFSET DSP_ST       ; és offsetje
                                        ;
                MOV     AH, ATTR_NPRESS ;Alapattributum
                MOV     CX, DSP_SIZE_WORD       ;Képernyőhossz
                MOV     SI, OFFSET  KBD_IMG     ;Kép offsetje
                CALL    SCR_DV          ;Video kikapcsolása
SCKB_INI_LOOP:                          ;
                LODSB                   ;Következő karakter be
                STOSW                   ;Ki attributummal
                LOOP    SCKB_INI_LOOP   ;
                CALL    SCR_EV          ;Video vissza
                                        ;
                POP     DI              ;
                POP     SI              ;
                POP     CX              ;
                POP     AX              ;
                POP     ES              ;
        ASSUME  ES:NOTHING              ;
                                        ;
                RET                     ;
SCKB_INI        ENDP                    ;
;---------------------------------------;
; SCR_WRPOS     - képernyő-pozíció módosítása
;       Input:
;               CX      - kiírandó karakter (attr:code)
;               BX      - képernyő-pozíció
;       Returns:
;               semmi
;       minden regiszter mentve
;---------------------------------------;
SCR_WRPOS       PROC    NEAR            ;
                                        ;
                PUSH    ES              ;
                PUSH    BX              ;
                                        ;
                WRCHAT  ,,,DISPLAY      ;Kiírás makróval
                                        ;
                POP     BX              ;
                POP     ES              ;
                RET                     ;
SCR_WRPOS       ENDP                    ;
```

```
; SCR_CHATR      - egy karakter attributumának módosítása
;       Input:
;                CH       - kiírandó attributum
;                BX       - képernyőpozíció
;       Output: nincs
;       Minden regiszter mentve
;-----------------------------------------;
SCR_CHATR       PROC    NEAR            ;
                                        ;
                PUSH    ES              ;
                PUSH    BX              ;
                                        ;
                WRATTR  ,,DISPLAY       ;
                                        ;
                POP     BX              ;
                POP     ES              ;
                                        ;
                RET                     ;
SCR_CHATR       ENDP                    ;
;-----------------------------------------;
; SCR_DV         - video kikapcsolása a megfelelő módparanccsal
;
; SCR_EV         - video bekapcsolása a megfelelő módparanccsal
;       A rutinok kódját "A képernyő" c. fejezetben találjuk
;-----------------------------------------;
; KBD_SHOW       - nyilvántartja a változásokat a képernyőn
;       Végigolvassa a táblázatot, és azokat a billentyűképeket
;       inverz videoban jeleníti meg, amelyeket nyomva tartanak
;-----------------------------------------;
                                        ;
KBD_SHOW        PROC    NEAR            ;
                                        ;
KBD_SHOW_MAIN:                          ;
                MOV     SI, 0           ;
                MOV     CX, SIZ_KBD_PRESS_TAB
KBD_SHOW_LOOP:                          ;
                PUSH    CX              ;-------;
                MOV     AX, KBD_PRESS_TAB[ SI ] ;
                                        ;-------;
                CMP     AX, 0FFFFH      ;Csupa egyes?
                JE      KBD_SHOW_NOKEY  ; ha igen, nem érv.
                TEST    AX, MASK CHANGE ;Le van nyomva?
                JZ      KBD_SHOW_NOCH   ;Nem, folytatjuk
                MOV     BX, AX          ;Igen, megjelenítés
                AND     BX, MASK POS    ;Pozíció előállítása
```

```
                TEST    AX, MASK PRESS  ;Ismétlési flag áll?
                JNZ     KBD_SHOW_SET    ;
KBD_SHOW_RESET:                         ;Felengedés volt
                MOV     CH, ATTR_NPRESS ;
                JMP     KBD_SHOW_FRESH  ;Frissítjük a képet
KBD_SHOW_SET:                           ;Lenyomás volt
                MOV     CH, ATTR_PRESS  ;
KBD_SHOW_FRESH:                         ;
                CALL    SCR_CHATR       ;Attributumot váltunk
                INC     BX              ; két pozíció-
                CALL    SCR_CHATR       ; ban is
KBD_SHOW_NOKEY:                         ;
KBD_SHOW_NOCH:                          ;
                ADD     SI, 2           ;Következő pozíció,
                POP     CX              ;
                LOOP    KBD_SHOW_LOOP   ; végig a táblán
                                        ;
                CALL    KBD_EXIT_TEST   ;Kilépés volt?
                JNC     KBD_SHOW_MAIN   ;Ha nem, elölről
                                        ;
                RET                     ;
                                        ;
KBD_SHOW        ENDP                    ;
;---------------------------------------;
; KBD_EXIT_TEST - megvizsgálja, hogy a CTRL-ALT-DELETE
;                 kombinációt lenyomták-e
;       Input:
;                 Nincs
;       Return:
;                 Carry   - 1 ha lenyomták,
;                           0 ha nem
;-----------------------------------------------;
                                                ;
KBD_EXIT_TEST   PROC    NEAR                    ;
                                                ;
                MOV     SI, CTRL_OFFS           ;CTRL van-e?
                TEST    KBD_PRESS_TAB[ SI ], MASK PRESS
                JZ      KBD_EXIT_TEST_NOEX      ;Nem, kész
                                                ;CTRL már van
                MOV     SI, ALTM_OFFS           ;ALT van-e?
                TEST    KBD_PRESS_TAB[ SI ], MASK PRESS
                JZ      KBD_EXIT_TEST_NOEX      ;Nem, kész
                                                ;CTRL-ALT...
                MOV     SI, DEL_OFFS            ;DEL van-e?
                TEST    KBD_PRESS_TAB[ SI ], MASK PRESS
```

```
                JZ      KBD_EXIT_TEST_NOEX      ;Nem, kész
KBD_EXIT_TEST_EXIT:                             ;CTRL-ALT-DEL,
                STC                             ; Carry-be 1
                JMP     KBD_EXIT_TEST_QUIT      ;
KBD_EXIT_TEST_NOEX:                             ;
                CLC                             ;Carry-be 0
                                                ;
KBD_EXIT_TEST_QUIT:                             ;
                                                ;
                RET                             ;
KBD_EXIT_TEST   ENDP                            ;
                                                ;
;-----------------------------------------------;
; KBD_INTR      - keyboard hardware interrupt rutin
;-----------------------------------------------;
                                                ;
KBD_INTR    PROC    FAR                         ;;;
                                                ;;;
                PUSH    AX                      ;;;A regiszte-
                PUSH    BX                      ;;;
                PUSH    CX                      ;;; rek
                PUSH    DI                      ;;;
                PUSH    DS                      ;;; mentése
                PUSH    ES                      ;;;
                                                ;;;
        IFDEF   _TEST                           ;;;Ha TEST, AL
                PUSH    AX                      ;;; a kód!
        ENDIF                           ;End of TEST condition
                MOV     AX, DATA                ;;;DATA szeg-
                MOV     DS, AX                  ;;; mens beáll.
        ASSUME  DS:DATA                         ;;;
                                                ;;;
        IFDEF   _TEST                           ;;;Ha TEST,
                POP     AX                      ;;; kód vissza
        ELSE                                    ;;;
                IN      AL, KB_DATA             ;;;Scan kód be
                PUSH    AX                      ;;;Mentjük
                IN      AL, KB_CTL              ;;;Vez.port be
                MOV     AH, AL                  ;;;Eredeti ment
                OR      AL, KB_RESET            ;;;Reset keyb.
                OUT     KB_CTL, AL              ;;;Parancs ki
                MOV     AL, AH                  ;;;Eredeti is-
                OUT     KB_CTL, AL              ;;; mét ki
                POP     AX                      ;;;Scan kód
        ENDIF                           ;End of TEST condition
```

```
                MOV     BL, AL                          ;;;
                SHL     BX, 1                           ;;;Index lesz!
                TEST    BH, Ø1H                         ;;;Felengedés?
                JNZ     KBD_INTR_DEPRESS                ;;;Igen
;-----------------------------------------------------;
; Billentyűlenyomáskor ez fut le
;-----------------------------------------------------;
KBD_INTR_PRESS:                                         ;;;
                XOR     BH, BH                          ;;;
                MOV     AX, KBD_PRESS_TAB[ BX ]         ;;;
                TEST    AX, MASK REPEAT                 ;;;Ismételt?
                JNZ     KBD_INT_PRESS_NOCH              ;;;Igen
                OR      AX, MASK CHANGE                 ;;;
KBD_INT_PRESS_NOCH:                                     ;;;
                OR      AX, MASK REPEAT                 ;;;Ismétlés be
                OR      AX, MASK PRESS                  ;;;Lenyomás be
                                                        ;;;
                MOV     KBD_PRESS_TAB[ BX ], AX         ;;;Tábla kész,
                JMP     KBD_INTR_EXIT                   ;;; vége
;-----------------------------------------------------;
; Ez a rész fut, ha elengedtek egy billentyűt
;-----------------------------------------------------;
KBD_INTR_DEPRESS:                                       ;;;
                XOR     BH, BH                          ;;;
                MOV     AX, KBD_PRESS_TAB[ BX ]         ;;;Elem AX-be
                AND     AX, NOT ( MASK PRESS OR MASK REPEAT )
                OR      AX, MASK CHANGE                 ;;;Bitkutyulás
                MOV     KBD_PRESS_TAB[ BX ], AX         ;;;
KBD_INTR_EXIT:                                          ;;;
        IFNDEF  _TEST                                   ;;;
                MOV     AL, ITC_EOI                     ;;;Valódi IT,
                OUT     Ø20H, AL                        ;;;8259 parancs
        ENDIF                                   ;End of TEST condition
                POP     ES                              ;;;
                POP     DS                              ;;;A felhasz-
                POP     DI                              ;;; nált re-
                POP     CX                              ;;; giszterek
                POP     BX                              ;;; helyre-
                POP     AX                              ;;; állítása
                IRET                                    ;;;
                                                        ;
KBD_INTR        ENDP                                    ;

                                                        ;
CODE    ENDS                                            ;
                                                        ;
```

```
;-------------------------------------------------;
; Display szegmens; rekordok és konstansok
;-------------------------------------------------;

SC_POS  RECORD  CCOD:8, FGCOL:4, BGCOL:3, BLINK:1

DSP_NUM_ROWS    EQU     25
DSP_NUM_COLS    EQU     80

DISPLAY SEGMENT AT      0B800H
DSP_ST          LABEL   WORD                ;Display kezdete
        ORG     DSP_NUM_ROWS * DSP_NUM_COLS * ( TYPE SC_POS )
DSP_END         LABEL   WORD                ;Display vége

    DSP_SIZE_BYTE    EQU   DSP_END - DSP_ST
    DSP_SIZE_WORD    EQU  (DSP_END - DSP_ST)/(TYPE SC_POS)
DISPLAY ENDS
                END     START
```

Számos figyelemreméltó apróság van ebben a programban. A hiányzó makrókat tessék a képernyőről szóló fejezetből kiolvasni, hely hiányában nem adtuk meg őket kétszer.

A program logikája a következő: a hardware interrupt rutin (neve: KBD_INTR) fogadja a billentyűkódokat a klaviatúráról, és egy 128 szavas táblázatot módosít aszerint, hogy melyik billentyűkód érkezett meg éppen. A táblázat egyes indexű eleme az egyes billentyűkódhoz van rendelve, a huszonötös indexű a huszonötös billentyűkódhoz és így tovább. Egy ilyen elem egy rekord. Ez a billentyű képének a képernyő-pozícióját tartalmazza, valamint három adminisztrációs bitet: a PRESS bitet, mely jelzi, hogy a billentyű le van nyomva, a REPEAT bitet, amely a folyamatos nyomvatartást jelzi, és a CHANGE bitet, amely azt mutatja, hogy volt-e változás a billentyű kezelésében. Ezeket a hardware interrupt rutin megfelelően karbantartja; ez a bemutatás alapja. A megszakítható szint (a fő rutin neve: KBD_SHOW) folytonosan olvassa a táblázatot, és ha valamelyik elem státusza megváltozott, akkor módosítja a képernyő megfelelő pontjának attributumát. Ezeken kívül alig van lényeges elem a programban. Figyelni kell az interrupt-vektorok precíz mentésére és helyreállítására, valamint (hardware-interrupt esetén) a 8259 programozására, az interrupt-eljárás végének jelzésére.

Az egész program legnagyobb értéke talán az az ötlet, amelynek segítségével követhetővé tettük a követhetetlent. Semmiféle debuggerrel nem tudjuk figyelni ezt a programot, hiszen ezek a debuggerek használják az eredeti klaviatúrát. Ha mi ezt a vek-

tort módosítjuk, akkor a debugger nem kap többé semmilyen parancsot. Az ötlet (a sok feltételes blokkban) a következő: a Ø9H interrupt helyett egy másikat használunk, esetünkben az FFH-et. Ezt persze nem tudjuk hardware interruptként élezni, ezért a főprogram egy párszor software úton meghívja; így a programot már nyomon tudjuk követni, és a helyzet csak akkor "fokozódik", mikor valóban bekacsoljuk a hardware interruptot. De ekkor már egy kipróbált, remélhetően jó programot aktivizálunk a hardware interrupttal!

IV. A képernyő

Már alapkiépítésben is kétféle képernyővezérlő kártyát vásárolhatunk gépünkhöz: a színes és a fekete-fehér (monokróm) vezérlőt. A színes adapter (CGA, Color Graphics Adapter [színes grafikus] vagy RGB, Red-Green-Blue [vörös-zöld-kék adapter]) a szokásos üzemmódokon kívül grafikus megjelenítésre is alkalmas. A monokróm adapter csak 80x25 karakteres egyszínű megjelenítésre alkalmas, a kártya azonban még egy párhuzamos nyomtatóillesztőt tartalmaz. Ez tehát nemcsak olcsóbb, de bizonyos szempontból összetettebb funkciójú kártya is. Egyébként mindkét szabványos adapter ugyanazt a képernyővezérlő áramkört használja: a Motorola 6845-öt (e fejezetben, ahol szó esik róla, csak a számával fogjuk említeni).

Maga a 6845 sokkal többet tud annál, mint amit a szabványos képernyővezérlők megvalósítanak. Ezt használták ki a fekete-fehér grafikus adapter, a Hercules kártya tervezői. A Hercules ugyanezt az áramkört használja, de grafikában sokkal többet tud, mint a színes grafikus adapter. A népszerű EGA (Enhanced Graphics Adapter) más áramkörre épül, amelynek belső kezelése egyébként hasonlít valamelyest a 6845-höz; grafikus képességei lényegesen jobbak.

A képernyőkezelésről szólva első lépésként a képernyőterület fogalmával kell megismerkednünk. A 6845 közvetlenül a memória egy területéről viszi ki az adatokat a képernyőre. E terület egy-egy szava egy-egy képernyő-pozíciónak felel meg.

Még egy fontos fogalommal kell megismerkednünk, mielőtt elmerülnénk a képernyővezérlés gyönyörűségeiben: a karaktergenerátorral. Ez olyan memóriaterület, amelyben a képernyőre kiírandó karakterek képét tárolják valamilyen bittérkép formájában. Míg az alapvezérlők karaktergenerátora ROM, azaz egyszer s mindenkorra beégetett karakterképeket tartalmaz, az EGA és a Hercules kártyán RAM területen helyezkedik el; ez azt jelenti, hogy programból megváltoztathatjuk a karakterek rajzát. A szokásos CGA kártyával ez csak grafikus módban lehetséges, ami azonban lényegesen lassítja a szöveg megjelenítését.

Első lépésként ismerkedjünk meg az alapvető képernyőüzemmódokkal és azok lehetőségeivel! Monokróm adapter csak egy üzemmódban működik, a szokásos alfanumerikus megjelenítésre képes. A szabványos CGA kártya hétféle, az EGA adapter még ennél is több üzemmódot ismer. Az alábbiakban a CGA kártya lehetőségeire koncentrálunk.

Két alapvetően különböző módon használhatjuk a képernyőt: alfanumerikus és grafikus módon. Ezek változatai adják a számos lehetőséget.

IV.1. Alfanumerikus ernyő

Az alfanumerikus ernyő az áramkör felprogramozásától függően 25 (EGA kártyával esetleg 43) sorban, soronként 40 vagy 80 karakter megjelenítésére képes. Minden egyes pozíciónak egy memóriaszó felel meg a képernyőterületen. Az első 80 szó az első sor, a második 80 szó a második sor tartalmát határozza meg, és így tovább;

- a soron belül az első szó felel meg az első karakternek, a második a másodiknak és így tovább;
- a szó alsó byte-ja a karakterkód, az ide beírt 8 bites értéket mint a kiterjesztett ASCII-készlet elemét, a videovezérlő azonnal megjeleníti a képernyőn, míg a felső byte az ún. attributumbyte, mely a karakter színét stb. határozza meg:

7	6	5	4	3	2	1	0
Villogtatás	Háttér láthatósága			Magas fényerő	Előtér láthatósága		

A monokróm képernyő attributumbyte-ja

Az előtér és a háttér "láthatósága" azt jelenti, hogy minél nagyobb (azaz a maximális heteshez közelebbi érték) szerepel az adott helyen, annál több látszik az előtérből (a karakterből) vagy a háttérből. Ennek megfelelően például ha a háttér értéke 0, az előtéré pedig 7, akkor fekete háttéren világos karaktereket látunk, míg ha a háttér értéke 7, az előtéré 0, akkor az adott karakter inverz video módban jelenik meg.

7	6	5	4	3	2	1	0
Villogtatás	(R) (piros)	(G) (zöld)	(B) (kék) Háttérszín (8 féle)	Magas fényerő	(R) (piros)	(G) (zöld)	(B) (kék) Előtérszín (8 féle)

A színes képernyő attributumbyte-ja

Külön szabhatjuk meg az előtér (a karakter) és a háttér színét, mindkettőt nyolc alapszínből választva. A magas fényerő az előtér színére vonatkozik, amely így 16 féle lehet. A villogtatást hardware végzi úgy, hogy az előtér színét adott időközönként váltogatja a specifikált előtér- és háttérszínek között, ami a szemlélőben a villogás érzetét kelti.

Inten-zitás	2.bit piros	1.bit zöld	Ø.bit kék	hexa érték	Szín
Ø	Ø	Ø	Ø	Ø	Fehér
Ø	Ø	Ø	1	1	Sötétkék
Ø	Ø	1	Ø	2	Sötétzöld
Ø	Ø	1	1	3	Türkiz
Ø	1	Ø	Ø	4	Vörös
Ø	1	Ø	1	5	Sötétlila
Ø	1	1	Ø	6	Barna
Ø	1	1	1	7	Világosszürke
Az intenzitási bit az előtéren nagy intenzitást, a háttérben viszont villogtatást jelent!					
1	Ø	Ø	Ø	8	Sötétszürke
1	Ø	Ø	1	9	Világoskék
1	Ø	1	Ø	A	Világoszöld
1	Ø	1	1	B	Világos türkiz
1	1	Ø	Ø	C	Piros
1	1	Ø	1	D	Világos lila
1	1	1	Ø	E	Sárga
1	1	1	1	F	Fehér

A CGA kártya színei alfanumerikus módban

A színes képernyő átprogramozható úgy is, hogy 40x25 karakter megjelenítésére legyen alkalmas. Ekkor a megfeleltetés pontosan ugyanilyen, csak a sorok hossza lesz 40 szó a 80 helyett.

IV.2. Grafikus képernyő

A színes adapterrel kezelt képernyő grafikus módban is használható. A színes IBM videovezérlők háromféle grafikus üzemmódot ismernek: 320x200 pontos színes, 320x200, illetve 640x200 pontos egyszínű grafikus módot. A képernyőt a gép ez esetben is a képernyőterület felhasználásával kezeli. Ezúttal monokróm grafika esetén egy bit, színes grafika esetén pedig két bit felel meg egy raszterpontnak (amiből azonnal látható, hogy a "színes" grafika négy színt ismer). Grafikus ernyők esetén a memória-képernyő hozzárendelés valamivel bonyolultabb, mint al-

fanumerikus módban. A képernyő 200 sorból áll, melyek 0-tól 199-ig számozottak. A memória alsó fele a páros, a felső fele pedig a páratlan sorokhoz van rendelve. Egy byte a felbontástól függően 4 vagy 8 pont jelentését szabja meg, a byte-ok "vízszintesek" (azaz a bitekhez rendelt pontok egy sorban helyezkednek el) és sorban követik egymást - egy sor első byte-ja az első 4 (8) pontot írja le, a második a második 4 (8) pontot, és így tovább. A hozzárendelés idáig teljesen logikus; itt következik egy kis "meglepetés": az alacsonyabb sorszámú pontokhoz tartoznak a magasab helyiértékű bitek. Legyen pl. a következő byte a képernyőterület legelső byte-ja, amely a grafikus ernyő első (0. számú) sorának első 4 pontjához van rendelve. Ekkor:

7	6	5	4	3	2	1	0
első (0.) pont		második (1.) pont		harmadik (2.) pont		negyedik (3.) pont	

Ugyanez nagy felbontású (640x200 pontos) grafikus képernyőn:

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

Itt tehát a hozzárendelés némileg eltér a várttól, a bitek fordított sorrendben felelnek meg a pontoknak.

A 320x200 pontos színes grafikus ernyő színei nem választhatók olyan szabadon, mint az alfanumerikus képernyőn. Két úgynevezett paletta áll rendelkezésünkre, amelyek három színt tartalmaznak, a negyedik az alapszín. Az alapszín a grafikus ernyőn lehetséges tizenhat közül szabadon választható, a 00 bináris érték a megfelelő pontot az alapszínre színezi. A paletták első színét a 01, a másodikat az 10, a harmadikat pedig az 11 bináris számokkal kapjuk meg. A palettaszínek a következők:

0. paletta: piros (1) - zöld (2) - sárga (3);
1. paletta: türkiz (1) - lila (2) - fehér (3).

Ezek a színek persze fekete háttérszín esetén egyeznek meg a fent elmondottakkal. Amennyiben más színű hátteret adunk meg, az előtér színei is változni fognak.

IV.3. A képernyő lapozása

Mekkora a grafikus képernyőterület? A közepes felbontású (320x200 pontos) színes ernyőn egy pont két bitet, így egy sor 80 byte-ot, a 200 sor pedig 16000 byte-ot igényel, azaz valamivel kevesebb mint 16 Kb-ot. Mekkora egy alfanumerikus képernyőterület memóriaigénye? 80x25x2 byte, azaz 4000 byte, kicsit kevesebb mint 4 Kb. Sőt, ha 40x25 karakteres ernyőt veszünk, akkor ennek fele, nem egészen 2 Kb. Ez ad ötletet ahhoz, hogy az amúgy is rendelkezésre álló 16 Kb-nyi grafikus képernyőterület több alfanumerikus képernyőterületet is tartalmazhatna. Valóban, 80x25 karakteres normál képernyőből négy, 40x25 karakteresből pedig 8 különálló képernyőterületet használhatunk. Ezek a képernyő "lapjai". Mindegyiken teljesen eltérő alfanumerikus tartalmat definiálhatunk, és rendkívül gyorsan válthatunk az ernyők között.

A későbbiekben ismertetendő képernyőkezelési funkciók egy része az aktív lapra vonatkozik (a képernyő rollozásai); mások viszont a háttérben levő lapon is elvégezhetők (a kiírások, a cursor pozicionálása stb).

IV.4. A Motorola 6845 programozása

A monitor közvetlen programozásához először a rendelkezésünkre álló képernyő és a szükséges vezérlőelemek olyan adataival kell tisztában lennünk, mint a képfrekvencia nagysága (vagyis a teljes kép frissítésének gyakorisága), valamint a sorfrekvencia (a sor frissítésének lehetséges gyakorisága). A nem megfelelő felprogramozás igen nagy bajokat okozhat. Ha tehát az itt megadottaktól eltérő paramétereket akarunk használni, akkor alaposan kell ismernünk a képernyővezérlő áramkör (esetünkben a 6845), a kártya és a felhasznált monitor képességeit. Ha például túl gyorsan akarjuk visszarántani a sorok végén a képet rajzoló elektronsugarat, komoly károsodást okozhatunk a monitorban (bár a jobb monitorokat ez ellen védik).

Az alábbiak megértéséhez mindenekelőtt néhány alapfogalommal kell megismerkednünk. Az első ilyen az elemi pont. Ezek vízszintes sora alkotja a scan-sort, az elemi TV-sort. Egy logikai sor a monitortól függő számú scan-sor, amelyben a karaktereket kirajzoljuk (vagyis a karakterek magassága elemi pontokban). Ez fekete-fehér adapter esetén 14, színes adapter esetén 8. A karakterszélesség a karakter elemi pontokban mért szélessége; értéke 9, illetve 8 pont.

A kép kirajzolása vázlatosan úgy történik, hogy a monitor elektronágyújából "kilőtt" elektronsugarat vízszintes és függőleges eltérítéssel úgy irányítják, hogy sorról sorra végigpásztázza a képernyő minden pontját, és ahol kell (az intenzitás változtatásával), "gyújtson világot", azaz hagyjon fénylő nyomot a képernyőn. A sor végén az elektronsugarat villámgyorsan visszarántják a következő sor elejére. Ez a sorvisszafutás, a sorszinkron ideje. Amikor pedig az elektronsugár eléri a képernyő jobb alsó sarkát, akkor az eltérítéssel visszavezérlik a jobb felső sarokba. Ez a képvisszafutás vagy függőleges szinkron. Otthoni TV-készülékeink akkor futnak vízszintesen vagy függőlegesen, ha ezen paraméterek értékét nem jól szabályozzuk be. Ugyanezt persze a monitorral is megeshet.

A különféle képernyőkezelési rendszerek még egy igen érdekes sajátossággal rendelkeznek, ez pedig az "interlace". A képet frissíthetjük sorról sorra folyamatosan, vagy úgy, hogy a sugár először minden páros sort jár be és frissít, azután pedig minden páratlan sort. Ha tehát egy monitort percenként 25 alkalommal frissítünk, ezt megtehetjük a teljes képet frissítve, vagy pedig az ötvenszeri frissítés sebességével minden második soron át haladva. A TV-készülékek általában az interlace módot használják, hiszen ez jobb a mozgó képek megjelenítéséhez. A számítógépek monitorai esetében mindkét eljárás elterjedt.

IV.4.1. A Motorola 6845 regiszterei

Most pedig térjünk rá a konkrét ismeretekre! A Motorola 6845 tizennyolc belső regiszterrel rendelkezik. Ezeket a regisztereket azonban a beépítés során nem képezték le a processzor I/O-címterébe, azaz nem kötötték mindegyiket egy I/O porthoz. Ehelyett igen ravasz eljárást alkalmaztak: az áramkörhöz rendelt portok közül kettő szolgál szinte az összes 6845 regiszter elérésére oly módon, hogy egyikük a címregiszter, másikuk az adatregiszter. A címregiszterre ki kell adnunk annak az áramköri regiszternek a címét, amelyet írni vagy olvasni akarunk, s ezután az adatregisztert használjuk az adat kiküldésére vagy beolvasására. Kiírás esetén ugyanis a 6845 felkészül arra, hogy az adatregiszterbe írt adatot a címregiszterben megjelölt saját regiszterébe helyezze el, míg beolvasás esetén a címregiszterben megjelölt regiszter tartalmát teszi láthatóvá az adatregiszteren keresztül.

A címregiszter portcíme 3D4H, az adatregiszteré pedig 3D5H. A 6845 belső regisztereinek ismertetésekor általában nem adunk részletes bittérképet, mivel legtöbbjük numerikus jellegű.

- R0, Horizontal Total - csak írható, bitjeinek száma 8; arra az időre utaló paraméter, ami egy scan-sor frissítésének kezdetétől a következő sor frissítésének kezdetéig eltelik. Ebbe beleértendő a megjelenített részek frissítése és a sorszinkron (a képet rajzoló elektronsugár visszatérítésének) ideje. Ezt az időt az egy karakterszélesség frissítéséhez szükséges egységekben mérik, és a kívánt értéknél eggyel kevesebbet kell kiírni.
  Szokásos értéke fekete-fehér adapter esetén 61H, színes adapter esetén 38H (40 karakter széles ernyőn és grafikus módban), 71H (80 karakter széles ernyőn).
- R1, Horizontal Displayed - csak írható, bitjeinek száma 8. Ez a regiszter adja meg az egy sorban látható karakterek számát. Az R0 és R1 regiszter értékei egymástól függetlenül nem állíthatók.
  Szokásos értékei 50H fekete-fehér adapter esetén, 28H vagy 50H színes adapterrel 40 karakter széles ernyőn vagy grafikus módban, illetve 80 karakter széles ernyőn.
- R2, Horizontal Synchron Position - csak írható, bitjeinek száma 8. A szinkronjel kezdetére utal a sor frissítésének kezdetéhez képest ugyanabban az egységben mérve, mint az R0 paraméter.
  Szokásos értéke fekete-fehér adapter esetén 52H, színes adapter esetén 2DH (40 karakter széles ernyőn és grafikus módban), 5AH 80 karakter széles ernyőn.
- R3, Horizontal Synchron Width - csak írható, bitjeinek száma 4. A szinkronjel hossza, ugyanabban az időtartamban mérve, mint az R0 paraméter.
  Szokásos értéke fekete-fehér adapter esetén 0FH, színes adapter esetén 0AH bármelyik üzemmódban.

A fenti paraméterek között az alábbi, igen fontos összefüggés állapítható meg:

sorfrekvencia * (R0+1) * karakterszélesség = pontfrekvencia,

amely a kártyára épített órajel-generátor frekvenciájától függ. Ennek várható értéke egyébként 16 és 22 megahertz között (EGA esetén esetleg 35 megahertz is) lehet.

A paraméterek alaposabb megértéséhez gondoljuk át a sor megjelenítésének menetét! A sor frissítése a látható karakterek rajzolásával kezdődik, majd a sor befejezésekor a monitornak egy szinkronjelet kell adni. A monitor ennek hatására futtatja vissza a sugárnyalábot a következő sor elejére. Az R0 tehát a teljes sor megjelenítésének idejét, az R2 a szinkronjel kezde-

tét adja meg a sor frissítési idején belül, az R3 a szinkronjel (vagyis a visszafutási idő) hosszát. A regiszterekbe írandó értékek nagyságukat tekintve a következőképpen viszonyulnak egymáshoz: R2 nagyobb, mint R1, RØ nagyobb, mint R1 és mint R2, végül RØ nagyobb, mint R2 és R3 összege.

- R4, Vertical Total - csak írható, bitjeinek száma 7. Ez a regiszter az R5-el együtt adja meg a teljes képernyő frissítéséhez szükséges időt. Az R4 és R5 meghatározásához két fontos adat szükséges: az egyik az, hogy hány scan-sor megjelenítési ideje egyenlő a teljes képernyő frissítéséhez szükséges idővel (tehát hogy hány scan-sor van a képernyőn, beleértve a látható sorokat, valamint a függőleges visszatérítési idő alatt "kitehető", nem létező scan-sorokat). A másik fontos adat pedig az, hogy egy karakter hány scan-sor magas (az R9 regiszter tartalma plusz egy).
  Ha az első számot elosztjuk a másodikkal, akkor a hányados értéke kerül (eggyel csökkentve) az R4, míg a maradék az R5 regiszterbe.
  R4 szokásos értékei fekete-fehér adapter esetén 19H, színes adapter esetén bármelyik alfanumerikus módban 1FH, míg grafikus módokban 7FH.
- R5, Vertical Total Adjust - csak írható, bitjeinek száma 5. Ez a regiszter az R4-el együtt adja a teljes képernyő frissítéséhez szükséges időt; a részleteket lásd R4 leírásánál.
  Szokásos értéke 6H fekete-fehér és színes adapter esetén egyaránt.
- R6, Vertical Displayed - csak írható, bitjeinek száma 7. A látható logikai sorok számát adhatjuk meg itt.
  Szokásos értéke mind fekete-fehér, mind színes adapter esetén alfanumerikus módban 19H, grafikus módban pedig 64H.
- R7, Vertical Synchron Position - bitjeinek száma 7. A képernyő frissítése során a függőleges visszafutás kezdetét adja meg, logikai sorokban mérve. Ez lényegében ugyanaz a paraméter a függőleges szinkronizálás számára, mint az R2-ben megadandó szinkronpozíció a vízszintes frissítés megadása során.
  Szokásos értéke fekete-fehér adapter esetén 19H, színes adapter esetén alfanumerikus módban 1CH, grafikus módban 70H.
- R8, Interlace Mode - csak írható, bitjeinek száma 2. Ez a regiszter határozza meg, hogy a monitor folytonos frissítést alkalmazzon-e, vagy páros és páratlan sorokat váltva hajtsa-e végre a frissítést. Ha a Ø. bit értéke Ø, akkor a frissítés folytonos lesz. Ha ez az érték 1, akkor a frissítés váltva történik. Kétféle interlace-módot ismer a 6845. Az első esetben a karakterek magassága elemi sorokban számítva a várt érték kétszerese lesz; a scan-sort úgy tekinthetjük, hogy nem

egy, hanem két elemi sorból áll. Tehát (nyolcsoros karaktert véve számításba) a karakter nyolc páros és nyolc páratlan sorból fog állni. A másik interlace-mód esetén a scan-sor egy elemi sornak számít, így a karakter magassága a várt érték. Ezzel az üzemmóddal kétszer annyi karaktert zsúfolhatunk a képenyőre, mint az előbbivel. IBM adapterek és a szokásos monitorok esetében ez egyébként nem ajánlatos.
Szokásos értéke 2H, ami folytonos frissítést jelent.

- R9, Maximum Scan Line Address - csak írható, bitjeinek száma 5. Ez a regiszter mutatja meg a logikai sor magasságát elemi pontokban: értéke a logikai sor magassága mínusz egy, tehát a színes ernyő hetes értéke azt jelenti, hogy a karakter nyolc elemi sor magasságú. Ne feledjük a regiszter nevében szereplő "address" szót: a legnagyobb scan-sor címét mutatja; a scan-sorok számozása pedig nullától kezdődik.
Szokásos értéke fekete-fehér adapter esetén 0DH, színes adapter esetén alfanumerikus módban 7H, grafikus módban pedig 1H.

Ez a hat regiszter, mint látható, a képernyő teljes frissítésének, valamint függőleges tagolásának paramétereit határozza meg. Közöttük az alábbi összefüggés áll fenn:

[(R4+1) * (R9+1) + R5] * képfrekvencia = sorfrekvencia

- R10, Cursor Start - csak írható, bitjeinek száma 7;
- R11, Cursor End - csak írható, bitjeinek száma 5;

Ez a két regiszter a képernyőn látható cursor alakját szabja meg. Az R10 alsó öt bitje adja meg, hogy a karakterkép melyik elemi pozícióján kezdődjön, az R11 pedig azt, hogy melyik elemi pozícióján végződjön a cursor. Ha a kezdőérték magasabb, mint a végérték, akkor a cursor "lyukas" lesz. Ha a kezdő- és/vagy a végérték magasabb, mint az elemi sorok száma, akkor a cursor (bár továbbra is létezik és mozgatható) egyáltalán nem látszik. Az R10 6. bitje azt szabja meg, hogy a cursor villogjon-e vagy sem, míg az 5. bit a cursor villogásának gyorsaságát vezérli (megjegyezzük, hogy a PC-n ezek a bitek mást jelentenek: az 5. bit beállítása láthatatlanná teszi a cursort, a 6. pedig nincs bekötve, ezért a PC monitorának cursora mindig villog).

- R12, Start Address High - csak írható, bitjeinek száma 6;
- R13, Start Address Low - csak írható, bitjeinek száma 8;

Ez a két regiszter együtt arra szolgál, hogy segítségükkel kijelölhessük a képernyőterület helyét a memóriában. Mivel a két

regiszter együtt is csak tizennégy bites, nincs valami széles választékunk. Az IBM PC-n a kártya tervezésekor dőlt el, hogy hol (ti. melyik hatvannégy kilobyte-ban) lesz a képernyő-memória és e két regiszterrel csak ezen belül, s itt is csak korlátozott mértékben mozoghatunk. A regiszterek szokásos kezdőértéke 0, az érték átírásával lehet egyfelől képernyőlapot váltani, másfelől pedig (megfelelő ügyeskedéssel) egy hardware-rollozást megvalósítani. Fontos még, hogy a címzés egysége nem byte, hanem szó; ez összhangban van azzal, hogy egy pozíciót nem egy byte, hanem egy szó határoz meg. Tehát (szerencsére) nem lehet a képernyőterületet byte-határra pozicionálni, így nem fenyeget az a veszély, hogy felcserélődik a karakterkód és az attributum szerepe.

Ha a megjelenítést egy sorral lejjebb szeretnénk folytatni, akkor a sor karakterekben kifejezett hosszúságát kell hozzáadnunk az R12:R13 regiszterekbe korábban kiírt értékhez, és újra kiírni. Ha pedig csak eggyel növeljük meg az értéket, akkor igen érdekes dolog történik: az egész ernyő lép egyet hátra. A bal felső sarokból eltűnik a karakter, az első sor balra lép, a második sor első karaktere az első sor végére kerül és így tovább. Tehát "kígyózik" a képernyő tartalma, ha a kiírt értéket egyesével növeljük.

- R14, Cursor High - írható-olvasható, bitjeinek száma 6;
- R15, Cursor Low - írható-olvasható, bitjeinek száma 8;

Ez a két regiszter annak a memóriapozíciónak a címét tartalmazza, ahol a cursor áll. Ez a pozíció az IBM PC/XT számára relatív, de a 6845 számára abszolút érték: a cursor pozícióját mindig a képernyőterület kezdetéhez viszonyítva adja meg, függetlenül attól, hogy azon belül hol kezdődik éppen a megjelenítés. A nulladik képernyőlap bal felső sarka tehát a képernyőterület első pozíciója (ekkor az R14 és R15 regiszterek 0-t tartalmaznak). Az első lap bal felső sarkához az 1000H pozíció tartozik és így tovább. Ezért mondjuk azt, hogy ez az érték a 6845 számára abszolút. Csakhogy ami a 6845 számára a memória kezdete, az az IBM PC/XT számára a B000:0000 vagy B800:0000 pozíció. A processzor számára tehát az R4:R5 regiszterpár értéke relatív a képernyő-memória kezdetéhez viszonyítva. Figyeljük meg azt is, hogy a cursorpozíció kiolvasásával megtudhatjuk, melyik képernyőlapon vagyunk (bár erre van egyszerűbb eljárás is). A 6845 a cursor helyét nem sor-oszlop koordinátákban tárolja. Ez a regiszterpár írható és olvasható, vagyis bármikor lekérdezhetjük a cursor pillanatnyi pozícióját. Megjegyzendő még, hogy a címzés egysége nem byte, hanem szó, összhangban azzal, hogy egy

pozíciót nem egy byte, hanem egy szó határoz meg. Tehát nem lehetséges a cursort két karakter "közé" (ti. valamely karakter attributumára) pozicionálni.

- R16, Light Pen High - csak olvasható, bitjeinek száma 6; az IBM PC-n csak színes adapter esetén használható.
- R17, Light Pen Low - csak olvasható, bitjeinek száma 8; az IBM PC-n csak színes adapter esetén használható.

A két utolsó regiszter tartalmazza a fényceruza által éppen célzott pozíció memóriacímét a képernyőterületen belül. Ez pontosan ugyanúgy abszolút érték, mint a cursor pozíciója.

IV.4.2. A képernyővezérlő kártyák portjai

a) A színes adapter portjai

A színes adapter a legfontosabb index- és adatportokon kívül még három porton át programozható: a Mode Control Register, a Color Select Register és a Status Register. A Mode Control Register I/O címe 3D8H, tehát ugyanazt a helyet foglalja el a porttömbben, mint a fekete-fehér adapter megfelelő regisztere, bitkiosztása a következő (hasonlít, csak a közös bitek elnevezése [ki tudja, miért?] eltérő):

7	6	5	4	3	2	1	0
Nem haszn.	Nem haszn.	Blinking Enable	High Resol. Mode	Video Enable	Black/White Select	Graphics Select	80x25 Alpha-Num.

A bitek:
- 80x25 Alphanumeric Mode:
  Választ a 40x25 vagy 80x25 alfanumerikus módok között.
- Graphics Select
  Egyes értéke grafikus módba lépteti a 6845-öt.
- Black & White Select
  Egyes értéke fekete-fehér módba lépteti a 6845-öt. Ez olyan esetben hasznos, ha színes adapterrel fekete-fehér képernyőt kezelünk, és valamely program színesen ír ki.
- Video Enable
  Egyes értéke engedélyezi a képernyőterület megjelenítését a képernyőn. Ha nullázzuk, a képernyő kialszik.

- High Resolution Mode
  Egyes értéke a nagyfelbontású grafikus üzemmódot szabja meg.
- Blinking Enable
  Egyes értéke esetén az attibutum-byte legmagasabb helyiértékű bitje a megfelelő karakter villogtatását jelenti, míg a nulla érték esetén ez a bit a háttérszín megadásában játszik szerepet.

A Mode-Select regiszter használatához (lévén csak írható) tudnunk kell az utoljára kiadott (és így érvényben lévő) módparancsot. Ha ugyanis a parancs nem felel meg a belső regiszterekben megadott felprogramozásnak, akkor a képernyőn sok mindent látunk, csak azt nem, amit kellene. Ha saját magunk programozzuk a monitort, akkor persze ismerhetjük a kiadott módparancsot. Ha azonban nincs más célunk, mint időnként letiltani a megjelenítést (például mert egy egész képernyőterületet akarunk bemásolni), akkor pontosan azt a módparancsot kell kiadnunk, mint előzőleg, csak a Video Enable bitet törölve; a megjelenítés bekapcsolásához pedig meg kell ismételnünk az előzőleg kiadott módparancsot, persze beállítva a Video Enable bitet. Honnan ismerhetjük "törvényesen" a kiadott módparancsot? Egyrészt ki lehet olvasni a ROM BIOS programok RAM területéről (nem legális eljárás), másrészt pedig (miután a megfelelő ROM BIOS funkcióval lekérdeztük a képernyő üzemmódját) a megfelelő parancsot kiolvashatjuk a video paraméter táblázatból (lásd "Egyéb ROM BIOS interruptok" fejezet). Most foglaljuk össze a szokásos képernyőmódokhoz tartozó módparancsokat!

Elnevezés	mód	parancs
40x25 fekete-fehér alfanumerikus mód	0	2CH
80x25 fekete-fehér alfanumerikus mód	1	28H
40x25 színes alfanumerikus mód	2	2DH
80x25 színes alfanumerikus mód	3	29H
320x200 színes grafikus mód	4	2AH
320x200 egyszínű grafikus mód	5	2EH
640x200 egyszínű grafikus mód	6	1EH
Fekete-fehér adapterhez tartozó érték	7	29H

A módparancsok összefoglalása

A Color Select Register I/O címe 3D9H, a képernyőn megjelenítendő színeket és az attributum-byte értelmezését szabja meg.

7	6	5	4	3	2	1	0
Nem használt	Nem használt	Choose Color Pal.	Alt. Graph. ColSet	High Intensity	Red (piros)	Green (zöld)	Blue (kék)

- Blue, Green, Red, High Intensity
  Ezek a bitek alfanumerikus módban a keret színét határozzák meg, közepes felbontású színes grafikus módban a háttér, míg nagyfelbontású grafikus módban az előtér (a látható pontok) színét (tizenhat lehetséges érték közül választhatunk).
- Alternate Intensified Graphics Color Set
  Ez a bit a grafikus üzemmódokban egy más intenzitású színkészlet kiválasztását eredményezi.
- Choose Color Palette
  Színes grafikus üzemmódban ez a bit választ a két lehetséges színpaletta közül.

Lássuk végül a státuszport bitkiosztását, amelynek ismerete valóban nagyon fontos a képernyő közvetlen programozásában (címe 3DAH színes, és 3BAH monokróm adapter esetén).

7	6	5	4	3	2	1	0
Nem használt	Nem használt	Nem használt	Nem használt	Vertical Retr.	Light Pen Switch	Light Pen Trigg.	Display Enable

- Display Enable
  Mikor e bit értéke 1, akkor büntetlenül felülírhatjuk a képernyőterület tartalmát (a vízszintes visszatérítést jelzi).
- Light Pen Triggered
  E bit akkor kap egyes értéket, mikor a kártyához kapcsolt fénycerura megérintette a képernyőt, azaz mikor a fényceruzáról érkező jelet az adapter észleli.
- Light Pen Switch Made
  Ez a bit jelzi, hogy van-e fényceruza egyáltalán. Ha a bit értéke Ø, akkor van, ellenkező esetben nincs.
- Vertical Retrace
  Mikor e bit értéke 1, akkor felülírhatjuk a képernyőterület tartalmát (a függőleges visszatérítést jelzi).

Fontos észrevétel, hogy (bár az Enable Display és a Vertical Retrace bit magyarázata megegyezik) van különbség a két bit között. A képernyőterület direkt írásakor azt tapasztaltam, hogy ha a Vertical Retrace bitet használtam, akkor a kiírás rettentően lassú volt ugyan, de valóban nem havazott a képernyő. Ha a Display Enable bitet használtam, akkor (igen kis mértékű) havazás volt ugyan, de a kiírás igen gyors lett.

b) A fekete-fehér adapter portjai

A monokróm vezérlő I/O portjainak tartománya a 3B0H címtől a 3BFH-ig terjed. Ezek közül csak hetet használ ki a rendszer. A 3B4H port a 6845 indexregisztere (itt címezhetjük meg a kívánt 6845 regisztert), míg a 3B5H az adatregiszter; a címzés végrehajtása után ide kell kiírni, vagy innen beolvasni a kívánt adatot. Az áramkör vezérlőportja a 3B8H, és végül a 3BAH port a státuszport (ezek leírását lásd alább).

A további három érvényes port a kártyára épített nyomtatóvezérlő áramkör elérésére szolgál: a 3BCH a nyomtató adatportja, a 3BDH a státuszport, és a 3BEH a vezérlőport (bővebb ismertetésüket lásd "A nyomtató" fejezetben). A többi portot nem használjuk. Lássuk a még ismeretlen két port bitkiosztását (előbb a vezérlő-, majd a státuszport ismertetésére kerül sor);

7	6	5	4	3	2	1	0
Nem haszn.	Nem haszn.	Blinking Enable	Nem haszn.	Video Enable	Nem haszn.	Nem haszn.	High Resol. Mode

Monokróm adapter vezérlőportja (portcíme 3D8H)

- High Resolution Mode
  Egyes értéke a nagyfelbontású módot szabja meg (az IBM PC/XT fekete-fehér adapterén mindig 1)
- Video Enable
  Egyes értéke engedélyezi a képernyőterület megjelenítését a képernyőn. Ha nullázzuk, a képernyő kialszik.
- Blinking Enable
  Egyes értéke esetén az attibutum-byte legmagasabb helyiértékű bitje a megfelelő karakter villogtatását jelenti, a nulla érték esetén ez a bit a háttérszín megadásában játszik szerepet.

7	6	5	4	3	2	1	0
Nem használt	Nem használt	Nem használt	Nem használt	Black & wh video	Fenntartott	Fenntartott	Horizontal drive

Monokróm adapter státuszportja (portcíme: 3BAH)

A bitek:

- Horizontal Drive:
  Ez a bit akkor kap egyes értéket, amikor a monitor éppen a függőleges visszatérítést végzi. Ilyenkor szabad módosítani a képernyő-memória tartalmát.
  Megjegyezzük, hogy fekete-fehér adapter esetén e bit vizsgálatának nincs jelentősége, mivel ez az adapter nem okoz "havazást" a képernyőn, bármikor büntetlenül írhatjuk a képernyőterület tartalmát.
- Black & White Video Mode:
  Ez a bit akkor kap egyes értéket, ha a színes megjelenítést letiltottuk. Fekete-fehér adapter esetén mindig egyes értékű.

Látható, hogy e portok (akár monokróm, akár színes adapter esetén) igen hasznosak a mindenre elszánt varázslók számára. Okos dolog például, hogy a "Video Enable" bit segítségével kioltjuk a képernyőn a képet. Ezt a lehetőséget használják ki a különböző képernyőkímélő programok (BLANK, SCRN sbt.). Ezek figyelik a képernyő és a klaviatúra használatát, valamint a bekövetkezett események között eltelt időt. Ha megszabott ideig nem érkezik egyetlen klaviatúra hardware interrupt sem, és a képernyő kommunikációs interruptját sem aktivizálja senki (tehát nincs se beolvasás, se kiírás), akkor kioltják a képet; ha pedig akár a képernyő kommunikációs interruptját, akár a klaviatúra hardware interrupját aktivizálja valaki, akkor a bit beállításával újra bekapcsolják a képernyőt.

Egy másik igen hasznos dolog, hogy a képernyő-attributum legfelső bitjét nem a villogtatás vezérlésére, hanem a háttérszín normál vagy nagy intenzitásának kezelésére használjuk fel. Ehhez a "Blinking Enable" bitet kell módosítanunk. Ha átállítjuk, nem kapunk ugyan villogó karaktereket a képernyőn, viszont a háttérszínt is tizenhat érték közül választhatjuk ki.

IV.5. Képernyőkezelési funkciók

Interrupt-sorszám: 10H
Javasolt interrupt-név: BS_VIDEO
A képernyő kommunikációs interruptjának aktivizálására szolgáló makró:

```
VIDCLL  MACRO   CODE
        IFNB    <SUBCODE>                       ;;Lehet, hogy
                MOV     AL, SUBCODE             ;; nincs!
        ENDIF                                   ;;
                MOV     AH, CODE                ;;Funkciókód
                INT     BS_VIDEO                ;;Video-hívás
        ENDM
```

Mielőtt elmélyednénk a képernyőkezelési funkciókban, nézzük meg, hogy a szolgáltatásokon kívül mit kapunk e funkcióktól, vagyis mely regiszterek értékét kapjuk vissza, és melyeket nem. A hivatalos irodalom azt állítja (szó szerint idézem):

CS, SS, DS, ES, BX, CX, DX preserved during call
All others destroyed

azaz a szegmensregiszterek és BX, CX, DX értéke változatlan marad, a többi megváltozik. Ez nem lehet egészen igaz, mert a ROM BIOS lista elárulja: a funkciók IRET utasítással lépnek ki, tehát az SP is az eredeti értékét kapja vissza (van irodalom, mely szerint az SP is megsérül - ez lenne még csak szép...). A többi regisztert (STATUS, indexregiszterek és különösen a BP) nekünk kell menteni.

IV.5.1. Képernyőüzemmód kiválasztása

Funkciókód: AH = 00H INT 10H
Javasolt konstansnév: VID_STMODE
Be:
AL = a kívánt üzemmód kódja

0	40x25 egyszínű
1	40x25 színes
2	80x25 egyszínű
3	80x25 színes
4	320x200 színes grafikus
5	320x200 egyszínű grafikus
6	640x200 egyszínű grafikus

Válasz:
nincs.

Ez a funkció szolgál a képernyő kívánt üzemmódjának megadására. Itt csak a normál színes képernyővezérlő adapter (CGA) üzemmódjait soroltuk fel; ha EGA kártyánk van, akkor rendelkezésünkre áll még néhány nagyon hasznos lehetőség.

Megjegyzés: a 4. és 5. mód között a kézikönyvekben olvasható különbségek helyett több adapter esetén az az eltérés, hogy a 4. módban választhatunk a paletták között, az 5. mód pedig mindig az 1. számú palettát használja.

IV.5.2. Cursortípus beállítása

Funkciókód: AH = 01H INT 10H
Javasolt konstansnév: VID_STCURT
Be:
CH, 0-4 bitek a cursor kezdősora
CL, 0-4 bitek a cursor végsora
Válasz:
nincs

A cursor mindig villog, és csak a nagyságát szabályozhatjuk. A karakterpozíció nyolc (monokróm adapteren 14) elemi pontsorból álló téglalap, amely a kezdősorától végsoráig látható. A sorokat színes adapteren értelmesen 0 és 7, monokróm adatper esetén pedig 0 és 13 között állíthatjuk be, ahol 0 a legmagasabb sor.

Ha a kezdősor alacsonyabb, mint a végsor, a cursor "lyukas" lesz. Ha a kezdősor nagyobb mint 7, akkor a cursor egyáltalán nem látható. Ha a végsor nagyobb, és a kezdősor kisebb mint 7, akkor a cursor egy 8 elemi sor magas téglalap.

Az egyes képernyőlapok pozíció szempontjából külön, független cursorral rendelkeznek, a háttérben levő lapok cursorát is tetszőlegesen pozicionálhatjuk. A cursortípus megadása csak az aktív lapra vonatkozik, mert a több, látszólag független cursort a ROM BIOS kezeli; magának a 6845-nek egy cursora van.

IV.5.3. Cursor pozicionálása

Funkciókód: AH = 02H INT 10H
Javasolt konstansnév: VID_STCURP
Be:
DH a kívánt sor száma (y koordináta, 0-24)
DL a kívánt oszlop száma (x koordináta 0-79/39)
BH a kívánt lap száma (0-3 vagy 0-7 lehet)
Válasz:
nincs

A képernyő e koordináta-rendszerének kiindulópontja a bal felső sarok, az "y tengely" függőleges; tehát a (0,0) pont a bal felső sarok, a (0,24) pedig a bal alsó. Grafikus módban BH kötelezően 0, hiszen nincs több képernyőlapunk.

Ez a funkció óvatosan kezelendő, mert nem ellenőrzi a cursor pozícióját. Ha egy 80 karakteres ernyőn a cursort a 10. sor 100. pozíciójába tesszük, akkor a 11. sor 20. karakterére kerül. Ha ennél is gorombábban címezzük el, akkor egy másik lapra is mutathat; a képernyőn egy félig-meddig értelmes helyen jelenik meg, de a kiírások a valójában címzett lapra kerülnek ki.

IV.5.4. Cursor lekérdezése

Funkciókód: AH = 03H INT 10H
Javasolt konstansnév: VID_GTCUR
Be:

BH	a kívánt lap száma (0-3 vagy 0-7 lehet)

Válasz:

DH	a cursor sorának száma
DL	a cursor oszlopának száma
CH	a cursor felső széle
CL	a cursor alsó széle

Ez a funkció minden szükségeset elmond az adott lap cursoráról; DX-t és CX-t mentve bármikor helyreállítható a cursor egy adott állapota.

IV.5.5. Fényceruza-pozíció beolvasása

Funkciókód: AH = 04H INT 10H
Javasolt konstansnév: VID_GTLGTPEN
Be:

semmi egyéb

Válasz:

AH	0, ha nincs beolvasható pozíció; 1, ha van
DH, DL	a címzett sor és oszlop pozíciója (karakteres)
CH	a sor száma (grafikus, 0-199)
BX	az oszlop száma (grafikus, 0-319/639)

AH jelzi, hogy be van-e kapcsolva a fényceruza, és ha igen, használatban van-e a lekérdezés pillanatában. Ha AH 1, akkor a többi regiszterben korrekt értékek találhatók, ha nem, akkor a regiszterek értelmetlenek. A BX:CH pár (alfanumerikusan kezelt ernyő esetében is) azt adja meg, hogy melyik grafikus alappontot (pixel, dot) célozta meg a felhasználó a fényceruzával, a DH:DL pár azt, hogy ez melyik karakterpozícióra esik.

IV.5.6. Az aktív lap kiválasztása

Funkciókód: AH = 05H INT 10H
Javasolt konstansnév: VID_STPAGE
Be:
AL a kívánt lap száma
Válasz:
nincs
A képernyőlapok csak alfanumerikus üzemmódokban használhatók; megengedett értékek: 0-7 a 0 és az 1 módokban, 0-3 a 2 és a 3 módokban (lásd a 00 funkciót).

IV.5.7. Ablak felfelé léptetése

Funkciókód: AH = 06H INT 10H
Javasolt konstansnév: VID_SCRUP

Az ablak az alfanumerikus ernyő egy téglalapja, melyet bal felső és jobb alsó sarkával lehet megadni. A léptetés azt jelenti, hogy a BIOS felfelé (vagy lefelé) mozgatja az ablak tartalmát a kívánt számú sorral. A kicsorgó sorok eltűnnek, és a másik oldalon üres sorok lépnek be helyettük.
Be:
AL a léptetések száma
CH, CL bal felső sarok sora és oszlopa
DH, DL jobb alsó sarok sora és oszlopa
BH az új sorok attributuma
Válasz:
nincs
Mint látható, ez a legegyszerűbb módja annak, hogy az egész képernyőt töröljük vagy átszínezzük. Ha az egész ablakot törölni akarjuk, az AL-ben 0-t kell megadni. Ha ugyanis az ablak sorainak számát adjuk meg (pl. az egész ernyő esetében 25-öt), akkor számos ROM BIOS változat (köztük az eredeti (!)) esetén az első két törlés sikeres, a továbbiak azonban legnagyobb meglepetésünkre színezett sorokat stb. hoznak be alulról.

IV.5.8. Ablak lefelé léptetése

Funkciókód: AH = 07H INT 10H
Javasolt konstansnév: VID_SCRDN
Be:
mint a felfelé léptetésnél, de AH = 07H
Válasz:
nincs

IV.5.9. Karakter és attributum kiolvasása

```
Funkciókód:              AH = 08H                    INT     10H
Javasolt konstansnév:    VID_GTCHATR
Be:
        BH      a kívánt lap száma
Válasz:
        AL      a kiolvasott karakter kódja
        AH      a kiolvasott karakter attributuma
```

Az adott lap cursora által címzett karaktert és annak attribu-tumát olvashatjuk ki e funkció segítségével, beleértve termé-szetesen bármelyik háttérben levő lapot is.
A funkció grafikus ernyőn is működik (!). Ilyenkor voltaképpen alakelemzést kell végeznie. Lényeges a dologban az, hogy nem csak a beégetett alsó 128 kódot ismeri fel, hanem a felső 128 kódot is, melyet a felhasználó definiált!

IV.5.10. Karakter és attributum kiírása

```
Funkciókód:              AH = 09H                    INT     10H
Javasolt konstansnév:    VID_WRCHATR
Be:
        AL      a karakter kódja
        BH      a kívánt lap száma
        BL      a kívánt attributum
        CX      az ismétlések száma
Válasz:
        nincs
```

Ez a funkció CX-szer ismételve írja ki az AL-ben adott karak-tert. A karakter attributuma is megadandó. Figyeljük meg, hogy bármelyik képernyőlapra írhatunk, nem csak az aktívra! Ez lehe-tővé teszi azt, hogy a háttérben levő lapon előkészítsünk vala-milyen szöveget, hogy aztán egyetlen lapváltással, látszólag rendkívül gyorsan jelenítsük meg az ernyőn.
A karakter(ek) kiírása a kívánt lapon, a cursor által címzett pozíción kezdődik. A cursor a kiírás során eredeti helyén ma-rad. Ha a CX értéke 1, akkor csak egy karakter kerül ki, egyéb-ként a karakter kiírása annyiszor ismétlődik, amennyit CX-ben megadtunk (ha pl. 0, akkor 65536-szor, ami nem feltétlenül ki-vánatos). Alfanumerikus módban a sor végének elérése után a ki-írás automatikusan folytatódik a következő sorban is.
Grafikus módokban a videovezérlő a karaktereket pontról pontra rajzolja. Ugyanezen funkcióval tehát ki tudunk írni valamilyen

szöveget grafikus módban is. A karakterek rajza a memóriában helyezkedik el, összesen 256 különböző karakter írható ki. Az első 128 a szokásos ASCII-készlet (ROM területen), a többi felhasználó által definiálható grafikus karakterkészlet; a 128 karakter rajzát leíró terület címét az 1FH interrupt vektorban kell megadni.
Egy-egy rajz 8x8 raszterpontból áll, 8 szomszédos byte-ot foglal le. A rajzot alkotó byte-ok "vízszintesek", a bal felső sarokpont az első byte legmagasabb helyiértékű bitje. Lássunk egy példát! A "." fekete (alapszínű) elemi pontot jelent, a "#" pedig az attributumnak megfelelő színűt (ez a kis rajzocska egy futó ember sziluettjét hivatott ábrázolni):

Rajz	Bináris érték	Hexadecimális érték
. . . # # . . .	00011000	18
. . . #	00010000	10
. . # # # . . #	00111001	39
. # . # . # # .	01010110	56
# . . #	10010000	90
. . # #	00110000	30
. # . . # . . .	01001000	48
# . . #	10010000	90

A kívánt ábrákat egy folytonos memóriaterületre kell letenni, melynek hossza 128x8 (1024) byte; ennek címét kell offset-szegmens formában elhelyezni az 1FH interrupt vektorban. Ha grafikus módban végzett többszörös kiírás közben érjük el a sor végét, akkor a művelet nem folytatódik a következő sorban.

IV.5.11. Karakter kiírása

Funkciókód: AH = 0AH INT 10H
Javasolt konstansnév: VID_WRCHR
Be:

AL	a kiírandó karakterek kódja
CX	a kiírandó karakterek száma
BH	a kívánt lap száma

Válasz:

nincs

Ez a funkció abban különbözik az előzőtől, hogy nem állíthatjuk be az attributumot, a kiírt karakterek az eredeti attributum szerint jelennek meg.

IV.5.12. Színpaletta vagy háttérszín beállítása

Funkciókód: AH = 0BH INT 10H
Javasolt konstansnév: VID_PALETTE
Be:
BH alparancs:
0 - háttérszínválasztás
1 - palettaválasztás
BL színkód vagy palettakód
Válasz:
nincs
a) BH=0 - a BL alfanumerikus módban a keret, grafikus módban a háttér színét választja ki a lehetséges 16 szín közül.
b) BH=1 - a 04 grafikus módban a BL regiszterben szabhatjuk meg a palettakódot (0 vagy 1). Az 5. üzemmódban ez a parancs hatástalan.

IV.5.13. Raszterpont kiírása

Funkciókód: AH = 0CH INT 10H
Javasolt konstansnév: VID_WRDOT
Be:
AL a kívánt szín (0-3 vagy 80-83)
DL a rasztersor száma (y koordináta, 0-199)
CX a raszteroszlop száma (x koordináta, 0-319/639)
Válasz:
nincs
A koordinátarendszer kezdőpontja itt is a bal felső sarok, koordinátái (0,0), az y tengely függőleges.
Ha az AL regiszter 7. bitjét 0-ra állítjuk (AL = 0-3), akkor a kívánt színben jelenik meg a pont. Ha azonban a 7. bit is áll (AL = 80-83), akkor az általunk megadott szín és az eredeti között egy kizáró vagy műveletet hajt végre a rutin, ezzel garantálva azt, hogy valami változni fog az adott ponton. Ezzel segíti a ROM BIOS a gumiszál-technika megvalósítását.

IV.5.14. Raszterpont visszaolvasása

Funkciókód: AH = 0DH INT 10H
Javasolt konstansnév: VID_GTDOT
Be:
DL a rasztersor száma (y koordináta, 0-219)
CX a raszteroszlop száma (x koordináta, 0-319/639)
Válasz:
AL a kért pont értéke (0 - 3)

E funkció jelentősége abban áll, hogy egyes pontokat le tudunk kérdezni vele. Valószínű, hogy ha valamilyen célból egy teljes grafikus képet le kell olvasnunk (például bitről bitre file-ba akarjuk menteni), akkor nem ezt a fáradságos módszert fogjuk választani, hanem egyszerűen egy ismételt MOVS utasítással kimásoljuk az egész grafikus ernyő tartalmát.

IV.5.15. Karakterkiírás teletype módban

Funkciókód: AH = 0EH INT 10H
Javasolt konstansnév: VID_WRTTY
Be:

AL	a karakter kódja
BH	a kívánt lap száma (alfanumerikus módban)
BL	a kívánt szín kódja (grafikus módban)

Válasz:
nincs

Ez a funkció szolgál a rutinszerű karakterkiírásra. Alfanumerikus módban az attributum "öröklődik" az előző karakterről a következőre, a cursor minden kiírásnál lép egyet jobbra (sor végéről a következő sor elejére lépteti a cursort, az utolsó sor végén egy sorral felfelé léptetve az ernyőt).
Grafikus módban a cursor hasonlóképpen lép; ilyenkor meg kell adni a karakter színét.
Ne felejtsük el azonban, hogy ez a funkció csak nekünk egyszerű. A BIOS driver szépen meghívogatja önmagát, pl. a 2. funkcióval mozdítja előre a cursort. Tehát ha nagy kiírási sebességre van szükségünk, és a szükséges adatok (cursorpozíció stb.) rendelkezésünkre állnak, akkor ne ezt a funkciót használjuk!

IV.5.16. Pillanatnyi üzemmód lekérdezése

Funkciókód: AH = 0FH INT 10H
Javasolt konstansnév: VID_GTMODE
Be:
semmi egyéb
Válasz:

AL	pillanatnyi üzemmód
AH	karakterek száma soronként (40 vagy 80)
BH	az aktív lap sorszáma (0 - 3 vagy 0 - 7)

Mielőtt bármilyen "trükkös" képernyőkezelésbe kezdenénk, első kötelességünk legyen lekérdezni a pillanatnyilag aktív képer-

nyőmódot és a cursor állapotát. Programunk a végleges kilépés előtt persze majd helyreállítja legalább a képernyőmódot és a cursortípust; a pozíció helyreállításának (hiszen a módváltás az ernyő törlésével jár és így kihúztuk a cursor alól az általa éppen címzett karaktert) nem mindig van értelme. Emiatt nehéz a képernyő eredeti attributumának helyreállítása is. A 08H funkcióval lekérdezhetjük ugyan egy karakter attributumát is, de ez nem feltétlenül egyezik az uralkodó attributummal. Ha a képernyő eredeti tartalma valóban fontos, akkor teljes egészében kimenthetjük és a kívánt időben helyreállíthatjuk. Ez kissé memóriaigényes, de ha valami kell, akkor kell (lásd a példákat).

Csak IBM AT !

IV.5.17. String kiírása

Funkciókód: AH = 13H INT 10H
Javasolt konstansnév: VID_WRSTR

Ez az AT funkció kellemesebbé teheti az életünket, mert a leggyakoribb feladatot, a stringek kiírását egy ROM BIOS hívással oldja meg, ráadásul sokkal gyorsabb is, mintha karakterenként küldenénk ki a stringet a képernyőre, hiszen nem nekünk kell a cursort előremozgatni; némi munkát és sok időt takarítunk meg. A funkció használata viszont AT-specifikussá teszi a programunkat. Négy alfunkció van, melyek segítségével többféleképpen írhatjuk ki a stringet. A közös bemenetek a következők:

Be:

AL	alfunkció
BH	kívánt képernyőlap
(BL	kívánt attributum)
CX	a string hossza
DH, DL	a kiírás kezdetének koordinátái
ES:BX	a string címe a memóriában

Alfunkciók és a javasolt alfunkciókódok:

- 00H: DOS WRSTR STUN - az attributum minden karakterre BL; a cursor a kezdőponton marad (a DH:DL ponton).
- 01H: DOS WRSTR ENDUN - az attributum minden karakterre BL; a cursor a stringet követő első helyre kerül.
- 02H: DOS WRSTR STIN - a string minden karakterét egy, a képernyőpozícióval megegyező szerkezetű szó adja (karakter-attributum pár); a cursor a kezdőponton marad (a DH:DL ponton).
- 03H: DOS WRSTR ENDIN - a string minden karakterét egy, a képernyőpozícióval megegyező szerkezetű szó adja (karakter-attributum pár); a cursor a stringet követő első helyre kerül.

Az alfunkció neve abból ered, hogy a kiírt string elejére vagy végére kerül a cursor (STart vagy END), illetve azonos vagy különböző attributumokat kapnak-e az egyes karakterek (UNiform vagy INdividual).

IV.6. Néhány példaprogram

A képernyőkezeléssel kapcsolatban néhány "törvényes" lehetőség magyarázata mellett egy-két olyan, a ROM BIOS funkciók közvetlen használatán túlmutató ötletet mutatunk be, amelyek hasznosak látványosabb, gyorsabban futó programok megírása során.

IV.6.1. Alapvető funkciók

Ezek a példák a legfontosabb képernyőkezelési funkcióknak nem a legegyszerűbb, de biztonságosnak mondható megvalósítását mutatják be a cursor pozicionálásától a különféle törlési funkciókig, egy-két segédrutinnal: SCR CURPOS (C kompatibilis), a leggyakrabban használt eljárások egyike; az SCR ERASE, amely az ernyő törlésének legegyszerűbb módját mutatja be a felfelé rollozás segítségével; az SCR ERENDL, amely a cursor pillanatnyi pozíciójától kezdve a sor végéig törli a képernyő tartalmát, és az SCR ERFLD, amely egy sorban egy "mezőt", adott hosszúságú területet töröl. A segédrutinok a képernyő paramétereinek, a cursor állapotának lekérdezését végzik: SCR GETPAR, SCR GETCUR és ennek párjaként az SCR RESCUR, amely az előző rutin által elmentett cursorállapotot állítja helyre.

A technikai megvalósításban felhasználtunk egy struktúrát, amely a képernyő pillanatnyi állapotát írja le; ezt persze fel kell tölteni a megfelelő értékekkel. Ezt végzi el a SCR GETPAR rutin. E struktúra párja a cursorleíró strukúra, amelyet az SCR GETCUR rutin tölt fel a megfelelő értékekkel. Ha viszont ilyen szépen, a rutincsomag belügyeként kezelve akarjuk az ernyőt vezérelni, akkor teljes szolgáltatást kell nyújtanunk, ügyelve arra, hogy a struktúrák mindig a tényleges állapotot mutassák; ehhez olyan rutinok szükségesek, amelyek a struktúrákat állandóan frissítik stb. E rutincsomag megvalósítása lényegesen túllépné könyvünk kereteit; egy teljesnek mondható megoldás a negyedik kötetben kapott helyet. Itt csak azért említjük egyáltalán a problémát, hogy felhívjuk rá a figyelmet.

Erre azért van szükség, mert bármelyik törlést végző szubrutint vesszük, tudnunk kell, milyen hosszú a képernyő egy sora (mert meg kell adnunk a jobb alsó sarkot, vagy a sor végéig

hátralevő karakterek számát). Majdnem minden művelethez szükségünk van az aktív lap sorszámára is. Megtehetnénk persze, hogy mindig lekérdezzük a paramétereket, de ez (mivel a BIOS videofunkciói nem a gyorsaságukról híresek) kissé lassú lenne.

a) A paraméterek lekérdezése: SCR GETPAR, SCR GETCUR

Ez a két rutin két struktúrát tölt fel a megfelelő értékekkel; a valóságban a struktúrák sokkal többen vannak, hiszen az egyik (amely a képernyőt írja le) azon adatokat fogja össze, amelyek függetlenek a képernyő-lapoktól, viszont mivel minden egyes képernyő-lapnak külön cursora van, minden lehetséges lap számára külön cursorleíró struktúrára van szükség. Az ezekben nyilvántartott adatok fontosak lehetnek, mint mentett (és a futás végén helyreállítandó) adatok; a cursor pillanatnyi pozíciójának nyilvántartása is lényeges lehet. Ekkor nem egy, hanem két struktúratömbre volna szükségünk, az egyik a mentett, a másik a pillanatnyi állapotot rögzítené. Mi most csak az egyiket oldjuk meg, azt is csak a kurrens lapra.

```
;-----------------------------------------------;
; Screen leíró struktúra definíciója
; SCRDESC       - SCR_GETPAR tölti fel, a többi SCR_ használja
;-----------------------------------------------;
SCRDESC         STRUC                         ;
        CMOD            DB      Ø             ;;Kurrens video mód
        VIDMOD          DB      Ø             ;;Kurrens módparancs
        CPAG            DB      Ø             ;;Kurrens aktív lap
        PAG_NUM         DB      Ø             ;;Lapok száma
        SIZ_X_A         DB      Ø             ;;Szélesség (alfanum.)
        SIZ_Y_A         DB      Ø             ;;Magasság (alfanum.)
        SIZ_X_G         DW      Ø             ;;Szélesség (grafikus)
        SIZ_Y_G         DW      Ø             ;;Magasság (grafikus)
        SIZ_PAGE        DW      Ø             ;;Laphosszúság byte-ban
SCRDESC         ENDS                          ;
;-----------------------------------------------;
; Cursor leíró struktúra definíciója
; CURDESC       - SCR_GETCUR tölti, SCR_RESCUR használja
;-----------------------------------------------;
CURDESC STRUC                                 ;
        POS_X           DB      Ø             ;;X pozíció
        POS_Y           DB      Ø             ;;Y pozíció
        TYP_END         DB      Ø             ;;Végsor
        TYP_ST          DB      Ø             ;;Kezdősor
CURDESC ENDS                                  ;
```

```
;-------------------------------------------------;
; Módleíró tömb, melyből a kurrens video-mód kiolvasható
;-------------------------------------------------;
MODDESC         MACRO                           ;
                DB      2CH, 28H, 2DH, 29H, 2AH, 2EH, 1EH, 29H
;Megfelelő üzemmódok:    0,   1,   2,   3,   4,   5,   6,   7
        ENDM                                    ;
                                                ;
CODE    SEGMENT BYTE    PUBLIC  'CODE'
                                                ;A struktúrák belügyek!
SCRD    SCRDESC         <>      ;Képernyőleíró struktúra
CURD    CURDESC  8 DUP  <>      ;Cursorleíró struktúratömb
MODD    LABEL   BYTE            ;Módtáblázat
        MODDESC                 ;Makróhívás VIDEO.INC-ből
;-------------------------------------;
; SCR_GETPAR    - az aktuális video paraméterek lekérdezése
;       Input, válasz:  nincs
; A rögzített értékeket a további rutinok felhasználják!
;-------------------------------------------------;
SCR_GETPAR      PROC    NEAR                    ;
                PUSH    AX                      ;A felhasznált
                PUSH    BX                      ; regiszterek
                PUSH    CX                      ;
                PUSH    DX                      ; mentése
                                                ;
                VIDCLL  VID_GTMODE              ;BIOS hívás
                                                ;-------;
                MOV     CS: SCRD.SIZ_Y_A, SCR_SIZ_Y_A   ;Fix
                MOV     CS: SCRD.SIZ_Y_G, SCR_SIZ_Y_G   ; érték
                MOV     CS: SCRD.CMOD, AL       ;Mód kitöltése
                MOV     CS: SCRD.CPAG, BH       ;Kurrens lap
                MOV     CS: SCRD.SIZ_X_A, AH    ;Karakterek sz.
                XOR     AH, AH                  ;AX: mód
                MOV     BX, AX                  ;
                MOV     AH, CS: MODD[ BX ]      ;Mód kiolvasás
                MOV     CS: SCRD.VIDMOD, AH     ; és kitöltés
                                                ;
                CMP     AL, VID_MOD_BWAD        ;Monokróm ad.?
                JE      SCR_GETPAR_MONO         ;Igen
                                                ;
                CMP     AL, VID_MOD_CLMR        ;Grafikus mód?
                JB      SCR_GETPAR_NOGRAPH      ;-------;Nem
                MOV     CS: SCRD.PAG_NUM, VID_PGNUM_GR  ;
                MOV     CS: SCRD.SIZ_PAGE, SCR_BUFSIZ_G ;
                JMP     SCR_GETPAR_STGRSIZ              ;
```

```
SCR_GETPAR_NOGRAPH:                                     ;Nem graf. mód
                    CMP     AL, VID_MOD_BW80            ;Nagy felb.?
                    JB      SCR_GETPAR_MEDIUM           ;--------;
                    MOV     CS: SCRD.PAG_NUM, VID_PGNUM_HIGH;
                    MOV     CS: SCRD.SIZ_PAGE, SCR_BUFSIZ_AH;
                    JMP     SCR_GETPAR_STGRSIZ                ;
                                                              ;
SCR_GETPAR_MEDIUM:                                            ;
                    MOV     CS: SCRD.PAG_NUM, VID_PGNUM_MED   ;
                    MOV     CS: SCRD.SIZ_PAGE, SCR_BUFSIZ_AM;
                                                        ;--------;
SCR_GETPAR_STGRSIZ:                                     ;
                    MOV     AL, AH                      ;Kiszámítjuk a
                    MOV     BH, 8                       ; grafikus sor-
                    MUL     BH                          ; szélességet
                    MOV     CS: SCRD.SIZ_X_G, AX        ;Grafikus szél.
                    JMP     SCR_GETPAR_QUIT             ;Alfanum kész
SCR_GETPAR_MONO:                                        ;Mono adapter
                    MOV     CS: SCRD.PAG_NUM, VID_PGNUM_MONO
                    XOR     AX, AX                      ;Egy lap, és
                    MOV     CS: SCRD.SIZ_X_G, AX        ; nincs grafi-
                    MOV     CS: SCRD.SIZ_Y_G, AX        ; kus mód
SCR_GETPAR_QUIT:                                        ;
                    POP     DX                          ;
                    POP     CX                          ;A regiszterek
                    POP     BX                          ; helyre-
                    POP     AX                          ; állítása
                                                        ;
                    RET                                 ;
SCR_GETPAR          ENDP                                ;
;-------------------------------------------------------;
; SCR_GETCUR    - egy lap cursorának lekérdezése
;       Input:
;                   ES:DI   - a cursorleíró struktúra címe
;       Válasz:
;                   semmi
;-------------------------------------------------------;
SCR_GETCUR          PROC    NEAR                        ;
                    PUSH    AX                          ;Használt
                    PUSH    BX                          ; regiszterek
                    PUSH    CX                          ; mentése
                    PUSH    DX                          ;
                    VIDCLL  VID_GTCUR                   ;--------;
                    MOV     WORD PTR ES:[ DI ].POS_X,    DX   ;
                    MOV     WORD PTR ES:[ DI ].TYP_END, CX    ;
```

```
                POP     DX                              ;
                POP     CX                              ;A regiszterek
                POP     BX                              ; helyre-
                POP     AX                              ; állítása
                                                        ;
                RET                                     ;
SCR_GETCUR      ENDP                                    ;
;-------------------------------------------------------;
; SCR_RESCUR    - a kurrens lap cursorának helyreállítása
;       Input:
;               ES:DI   - a cursorleíró struktúra címe,
;                         melybe előzetesen elmentettük
;                         a kurrens lap cursorát
;       Válasz: semmi
;-------------------------------------------------------;
                                                        ;
SCR_RESCUR      PROC    NEAR                            ;
                                                        ;
                PUSH    AX                              ;Használt
                PUSH    BX                              ; regiszterek
                PUSH    CX                              ; mentése
                PUSH    DX                              ;
                MOV     BH, CS: SCRD.CPAG               ;--------;
                MOV     DX, WORD PTR ES:[ DI ].POS_X    ;Pozí-
                VIDCLL  VID_STCURP                      ; ció
                MOV     CX, WORD PTR ES:[ DI ].TYP_END  ;Cursor
                VIDCLL  VID_STCURT                      ; típus
                                                        ;--------;
                POP     DX                              ;
                POP     CX                              ;A regiszterek
                POP     BX                              ; helyre-
                POP     AX                              ; állítása
                RET                                     ;
                                                        ;
SCR_RESCUR      ENDP                                    ;
                                                        ;
CODE    ENDS                                            ;
;-------------------------------------------------------;
```

A fenti két rutin "mentségére" pár fontos dolgot el kell mondani. Először is, miért a rutinok szegmensében kapott helyet az SCRD struktúra? Ennek két oka van: az egyik, hogy így inkább el van rejtve ez a terület, mint ha (valamelyik) felhasználói adatszegmensben lenne; a másik ok a "valamelyik" szóban rejlik: nem tudhatjuk, hogy egy-egy képernyőkezelő rutin hívásakor me-

lyik adatszegmens érhető el éppen; az a kódszegmens viszont, melyben a rutinok vannak, egészen biztos, hogy elérhető. A cursor lekérdezése itt csak a teljesség kedvéért szerepel; a szép megoldás persze C kompatibilis rutin megírása lenne, de (mint a negyedik kötetben látjuk majd) cím átvétele C függvénytől elég problematikus lehet. Tudnunk kell, hogy jön-e vele a szegmens, vagy nem, stb. Kézenfekvő volna persze cím helyett lapszámot, s ezáltal struktúratömb-indexet átvenni, de ez nem túl szerencsés megoldás, a struktúratömb bedrótozását jelenti. Ilyen cursorleíró struktúrára pedig sok helyen lehet szükségünk, nem jó, ha a címe be van égetve a szubrutinba.

b) A cursor pozicionálása

A cursor pozicionálása rendkívül gyakori művelet. Természetesen célszerű eljárást írni rá, de milyen legyen ez: nyílt eljárás (magyarul: makró), vagy zárt (szubrutin)? Mindkettőre adunk példát. Az eljárás legalább két paramétert vesz át (a kívánt koordinátákat). A paraméterek átadása regiszterben a legkényelmesebb, de ha bebűvöljük a két értéket egy-egy regiszterbe, akkor már miért ne hívjuk meg a video-interruptot? Ez az egyik érv, ami a makró mellett szól, a másik a makró nagyobb gyorsasága. A szubrutin egyáltalán nem célszerű ez esetben. Ha valamiért mégis érdemes ilyen szubrutint írni, akkor azért, hogy például C nyelvből könnyen tudjuk a cursort mozgatni. Erre persze meg lehet kérdezni: miért kell assembly rutint írni a C nyelvhez, mikor a C-ből közvetlenül is meg lehet hívni a video-interruptot? Azért, mert ennek során tucatnyi felesleges utasítás hajtódik végre (az összes regisztert feltöltik a "kívánt" értékekkel), bár nekünk csak négy félregiszter kell. A közvetlen BIOS hívás helyett ezért célirányos egy külön assembly rutint megírni. Lássuk először a makrót!

```
CURPOS  MACRO   X, Y
                                                        ;
        IFNB    <X>                                     ;;
                MOV     DL, X                           ;;Sorpozíció
        ENDIF                                           ;;
        IFNB    <Y>                                     ;;
                MOV     DH, Y                           ;;Oszloppozíció
        ENDIF                                           ;;
                MOV     BH, CS: SCRD.CPAG               ;;Kurrens lap
                VIDCLL  VID_STCURP                      ;;BIOS hívás
                                                        ;
        ENDM                                            ;
```

A makró a kurrens lap számát a korábban leírt (és a program elején, vagy képernyőmód-váltás során meghívott) SCR_GETPAR rutin által feltöltött struktúrából veszi. Egyetlen apróság, hogy feltételes blokkban van a DH és DL feltöltése; ha a hívó már beállította a regisztereket, akkor elhagyhatja a paramétereket.

A függvény annyival nyújt többet, mint a makró, hogy kívánságra meg is vizsgálja: a kapott paraméterek legálisak-e vagy sem (belövés alatt álló program gyakran küldi el a cursort egészen érdekes helyekre...).

```
;-------------------------------------------------------;
; _SCR_CURPOS    - cursorpozicionálás; a paramétereket stacken
;                  veszi át; feltételes blokkban ellenőrzés!
;       Input:
;                0. paraméter     - kért X koordináta
;                1. paraméter     - kért Y koordináta
;       Válasz:
;                AX               - 0, ha sikeres
;                                   nem 0, ha nem sikeres
;-------------------------------------------------------;
        _PUBLIC _SCR_CURPOS                             ;Globális!
                                                        ;
PAR_DIST        EQU     4                               ;Rövid modell
X               EQU     0                               ;Kért X és Y
Y               EQU     2                               ; koord. helye
                                                        ;
_SCR_CURPOS     PROC    NEAR                            ;
                                                        ;
                PUSH    BP                              ;Stack címzés
                MOV     BP, SP                          ; BP-vel !!
                PUSH    BX                              ;Mindent
                PUSH    DX                              ; mentünk
                                                        ;
                MOV     DX, X[ BP + PAR_DIST ]          ;Kívánt sor
                MOV     AX, Y[ BP + PAR_DIST ]          ;Kívánt oszlop
                                ;---------------;
                                ;Ha kértek ellenőrzést
                                ;---------------;
        IFDEF   _TEST                                   ;
                TEST    DH, DH                          ;Felső byte 0?
                JNE     SCR_CURPOS_UNS                  ;
                CMP     DL, CS: SCRD.SIZ_X_A            ;Belül van?
                JAE     SCR_CURPOS_UNS                  ;
                TEST    AH, AH                          ;Felső byte 0?
                JNE     SCR_CURPOS_UNS                  ;
```

```
                CMP     AL, CS: SCRD.SIZ_Y_A    ;Belül van?
                JAE     SCR_CURPOS_UNS          ;
        ENDIF                                   ;
                MOV     DH, AL                  ;
                MOV     BH, CS: SCRD.CPAG       ;Kurrens lap!
                VIDCLL  VID_STCURP              ;BIOS hívás
                XOR     AX, AX                  ;Sikeres volt
        IFDEF   _TEST                           ;
                JMP     SCR_CURPOS_QUIT         ;
SCR_CURPOS_UNS:                                 ;
                MOV     AX, -1                  ;Sikertelen !
SCR_CURPOS_QUIT:                                ;
        ENDIF                                   ;
                POP     DX                      ;Regiszterek
                POP     BX                      ;helyreállítása
                POP     BP                      ;BP vissza !
                RET                             ;
_SCR_CURPOS     ENDP                            ;
```

E rutin teljes megértéséhez ismerni kell a C nyelv rutinhívási (pontosabban paraméterátadási) stratégiáját. Erről teljes alapossággal a negyedik kötetben esik szó, és valamivel többet találhatunk a hangszóró kezeléséről szóló fejezet egyik példaprogramjában. Számunkra e pillanatban inkább a feltételes blokk használata lehet tanulságos.

Egy fontos adat: százezer cursorpozicionálással kimértem, hogy a közvetlen ROM BIOS hívás, vagy az assembly rutin segítségével végrehajtott pozicionálás a gyorsabb, és meglepő eredményeket kaptam. Ha a C-ben olyan függvényt használtam, amely a paraméterek ellenőrzésével pozicionálta a cursort (az "int86()" könyvtári függvénnyel), akkor a százezer ismétlés kb. 15 másodpercet vett igénybe. Ugyanez a fenti assembly rutin segítségével kb. 9 másodperc; a különbség nem elhanyagolható. Ha elhagytam az ellenőrzéseket (ami belőtt programok esetében már elmaradhat), akkor az eredmény drámaibb volt: az "int86()" használatával kb. 14 másodpercre volt szükség a százezer pozicionáláshoz, míg az assembly rutin hívásával mindössze kb. 4-re. Itt tehát döbbenetes a különbség. Ez egyértelműen az assembly rutin mellett szól, még akkor is, ha megírása kicsit fáradságosabb.

c) Törlési funkciók

Három szubrutint adunk meg, melyek közül kettő triviálisan "C-kompatibilis" (egyáltalán nem vesznek át paramétereket). Megfigyelésre az az egyetlen apróság érdemes, hogy használhat-

juk a munka csökkentésére a szintezést. Az SCR_ERASE az egész ernyőt törli; csak azért érdemes szubrutinként megírni, hogy ne kelljen mindig legépelni a sok értékadást; ez az eljárás lehetne makró is, hiszen ez nem kerülne semmibe. Az SCR_ERENDL rutin a cursor pillanatnyi pozíciójától a sor végéig törli a képernyőt; erre az SCR_ERFLD rutint használja. Meghatározza a sor hosszát (a hátralevő pozíciók számát), és ezt adja paraméterként az egyébként kívülről is hívható rutinnak.

```
;-------------------------------------------------------;
; SCR_ERASE     - a kurrens lap teljes képernyőjének törlése
;       Input, válasz:
;                       nincs
;       Minden regiszter mentve
;-------------------------------------------------------;
SCR_ERASE       PROC    NEAR                            ;
                PUSH    AX                              ;A használt
                PUSH    BX                              ; regiszterek
                PUSH    CX                              ; mentése
                PUSH    DX                              ;
                                                        ;
                MOV     CX, 0000H                       ;Bal felső és
                MOV     DX, WORD PTR SCRD.SIZ_X_A
                MOV     DL, SCRD.SIZ_X_A                : jobb alsó
                DEC     DL                              ; sarok meg-
                DEC     DH                              ; határozása
                                                        ;
                MOV     AL, ' '                         ;Blank-oljuk őt
                MOV     BH, 07H                         ;Fehér feketén
                VIDCLL  VID_SCRUP                       ;BIOS hívás
                                                        ;
                POP     DX                              ;A regiszterek
                POP     CX                              ; helyre-
                POP     BX                              ; állítása
                POP     AX                              ;
                RET                                     ;
SCR_ERASE       ENDP                                    ;
;-------------------------------------------------------;
; SCR_ERENDL    - ernyőtörlés a sor végéig a kurrens lapon
;       Input:
;               semmi
;       Válasz:
;               AX      - a sor végének hossza
;       Minden regiszter mentve
;-------------------------------------------------------;
```

```
SCR_ERENDL      PROC    NEAR                    ;
                PUSH    BX                      ;
                PUSH    CX                      ;
                PUSH    DX                      ;
                                                ;
                MOV     BH, CS: SCRD.CPAG       ;Kurrens lap
                VIDCLL  VID_GTCUR               ;Cursor lekérd.
                MOV     CL, SCRD.SIZ_X_A        ;Sorhossz ki-
                SUB     CL, DL                  ; számítása
                XOR     CH, CH                  ;
                PUSH    CX                      ;Válasz mentése
                CALL    SCR_ERFLD               ;Mezőtörlés
                POP     AX                      ;Válasz AX-be
                                                ;
                POP     DX                      ;A regiszterek
                POP     CX                      ; helyre-
                POP     BX                      ; állítása
                RET                             ;
SCR_ERENDL      ENDP                            ;
;-----------------------------------------------;
; SCR_ERFLD     - field (mező) törlése az adott sorban
;       Input:
;               CX      - a mező hossza pozíciókban
;       Válasz:
;               semmi
;       Nincs sorvégellenőrzés.
;       Minden regiszter mentve.
;-----------------------------------------------;
SCR_ERFLD       PROC    NEAR                    ;
                PUSH    AX                      ;A használt
                PUSH    BX                      ; regiszterek
                PUSH    CX                      ; mentése
                                                ;
                MOV     BH, CS: SCRD.CPAG       ;Kurrens lap
                MOV     AL, ' '                 ;Blank-oljuk
                MOV     BL, 07H                 ;Fehér feketén
                VIDCLL  VID_WRCHATR             ;BIOS hívás
                                                ;
                POP     CX                      ;A regiszterek
                POP     BX                      ; helyre-
                POP     AX                      ; állítása
                                                ;
                RET                             ;
SCR_ERFLD       ENDP                            ;
```

IV.6.2. Képernyőtartalom váltása

Tegyük fel, hogy rendelkezésünkre áll egy olyan file, amely pontosan képernyőnyi méretekben tartalmaz szöveges információt. Ezt az információt tehát laponként érhetjük el a file direkt olvasásával. Feladatunk az, hogy a lehető leggyorsabban jelenítsük meg a kívánt szöveglap tartalmát a képernyőn úgy, hogy az eredeti tartalom ne változzon meg.

A feladat megoldásához ismernünk kell lennünk az adapter típusát, és (színes adapter esetén) a pillanatnyi üzemmódot. Ha meghívjuk a korábban ismertetett SCR_GETPAR rutint, akkor az üzemmód rendelkezésünkre áll. Ennek használatával dolgozhatunk tovább. Az egyszerűség kedvéért tegyük fel, hogy vagy monokróm adapterünk van, vagy 80x25-ös színes vagy fekete-fehér módban vagyunk. Ekkor egy szöveglap (attributum nélkül) éppen kétezer byte hosszúságú. Az üzemmód ismerete azért szükséges, hogy tudjuk: rendelkezésünkre áll-e háttérben levő képernyőlap, vagy sem. Ha igen (tehát színes adapterrel dolgozunk), akkor a feladat annyi, hogy valamelyik (háttérben levő) lapra szépen kivisszük a szöveget valamilyen uralkodó attributum szerint, majd lapot váltunk; várunk a szöveget "elengedő" parancsra (például az ESC karakterre), és visszaváltunk az eredeti (elhagyott) képernyőre (ezt a módszert később illusztrálom).

Fekete-fehér adapter esetén a feladat annyival bonyolultabb, hogy nincs segítségünkre háttérben levő lap, viszont nem kell annyira vigyáznunk a havazás elkerülésére. Ilyen esetben egy belső adatterületre elmentjük a képernyő teljes tartalmát, valamint a cursor címét és alakját. Ezután az előzőleg felolvasott szöveget karakterről karakterre kivisszük a képernyőre, majd a lapot elengedő parancs vétele után visszamásoljuk az eredeti tartalmat, és helyreállítjuk a cursort. Az eredeti tartalom mentése és visszahozása a gyakorlatban célszerűen egy ismételt MOVS utasítással mehet végbe.

Példáinkban csupán két ravasz szubrutint (meg egy egyszerű makrót) mutatunk be, melyek segítségével igen szépen és gyorsan lehet teljes lapnyi információt adni a képernyőre. Lássuk őket!

```
;-----------------------------------------------;
; SCR_SAVE      - elmenti a képernyő területét
;       Input:
;               ES:DI   - mentési buffer címe
;       Válasz:
;               semmi
;       Minden regiszter mentve
;-----------------------------------------------;
```

```
SCR_SAVE        PROC    NEAR                            ;
                PUSH    BX                              ;A felhasznált
                PUSH    CX                              ;
                PUSH    DX                              ; regiszterek
                PUSH    SI                              ;
                PUSH    DS                              ; mentése
                PUSH    ES                              ;
                                                        ;
                MOV     CX, CS: SCRD.SIZ_PAGE           ;Laphossz CX-be
                CMP     CS: SCRD.CMOD, VID_MOD_BWAD     ;Mono?
                JE      SCR_SAVE_MONO                   ;-------;
SCR_SAVE_COLOR:                                         ;Színes
                CALL    SCR_GTBUFADDR                   ;Ernyőcím SI-be
                MOV     BX, COLOR_VIDEO_SEG             ;Ernyő szegmens
                MOV     DS, BX                          ;DS:SI megvan
                CALL    SCR_DV                          ;Video kikapcs.
        REP     MOVSB                                   ;Másolás
                CALL    SCR_EV                          ;Video vissza
                JMP     SCR_SAVE_QUIT                   ;Készen van
SCR_SAVE_MONO:                                          ;
                XOR     SI, SI                          ;Ernyő offset 0
                MOV     BX, MONO_VIDEO_SEG              ;Ernyő szegmens
                MOV     DS, BX                          ;DS:SI megvan
        REP     MOVSB                                   ;Másolás
SCR_SAVE_QUIT:                                          ;
                POP     ES                              ;
                POP     DS                              ;A regiszterek
                POP     SI                              ;
                POP     DX                              ; helyre-
                POP     CX                              ;
                POP     BX                              ; állítása
                                                        ;
                RET                                     ;
SCR_SAVE        ENDP                                    ;
                                                        ;
;-------------------------------------------------------;
; SCR_RETRIEVE  - helyreállítja a képernyő területét
;       Input:
;               ES:DI   - mentési buffer címe
;       Válasz:
;               semmi
;       A rutin hívása előtt meg kell hívni az SCR_GETPAR-t!
;       Minden regiszter mentve
;-------------------------------------------------------;
```

```
SCR_RETRIEVE    PROC    NEAR                            ;
                PUSH    BX                              ;A fel-
                PUSH    CX                              ; használt
                PUSH    DX                              ;
                PUSH    SI                              ; regiszterek
                PUSH    DS                              ;
                PUSH    ES                              ; mentése
                MOV     CX, CS: SCRD.SIZ_PAGE           ;Laphossz CX-be
                CMP     CS: SCRD.CMOD, VID_MOD_BWAD     ;
                JE      SCR_RETR_MONO                   ;--------;Mono?
SCR_RETR_COLOR:                                         ;Színes
                CALL    SCR_GTBUFADDR                   ;Ernyőcím SI-be
                MOV     BX, COLOR_VIDEO_SEG             ;Ernyő szegmens
                PUSH    ES                              ;
                MOV     ES, BX                          ;ES-DS csere!
                POP     DS                              ;
                XCHG    SI, DI                          ;SI-DI csere
                CALL    SCR_DV                          ;Video kikapcs.
        REP     MOVSB                                   ;Másolás
                CALL    SCR_EV                          ;Video vissza
                JMP     SCR_RETR_QUIT                   ;Készen van
SCR_RETR_MONO:                                          ;
                XOR     SI, SI                          ;Ernyő offset 0
                MOV     BX, MONO_VIDEO_SEG              ;Ernyő szegmens
                PUSH    ES                              ;
                MOV     ES, BX                          ;ES-DS csere
                POP     DS                              ;
                XCHG    SI, DI                          ;SI-DI csere
        REP     MOVSB                                   ;Másolás
SCR_RETR_QUIT:                                          ;
                POP     ES                              ;
                POP     DS                              ;A regiszterek
                POP     SI                              ;
                POP     DX                              ; helyre-
                POP     CX                              ;
                POP     BX                              ; állítása
                RET                                     ;
SCR_RETRIEVE    ENDP                                    ;
;-------------------------------------------------------;
; SCR_GTBUFADDR         - kurrens lap címének lekérdezése
;       Input:
;                 semmi   (SCRD.CPAG, kurrens lap)
;       Válasz:
;                 SI      - a kurrens lap offsetje
;-------------------------------------------------------;
```

```
SCR_GTBUFADDR   PROC    NEAR                    ;
                                                ;
                PUSH    CX                      ;
                MOV     CL, CS: SCRD.CPAG       ;Kurrens lap
                XOR     CH, CH                  ;
                MOV     SI, CX                  ;
                MOV     CL, 13                  ;Shiftek száma
                SHL     SI, CL                  ;Lapcím SI-be
                POP     CX                      ;
                RET                             ;
                                                ;
SCR_GTBUFADDR   ENDP                            ;
```

Magukban a rutinokban nincs sok ravaszság; meghatározzák a képernyőlap offsetjét (már ha ez egyáltalán szükséges), és előkészítenek és elvégeznek egy másolást a tár egy megadott pontja és a képernyőterület között (a megjelenítést tiltó és engedélyező rutint lásd alább). A rutinokat azonban ravaszul használhatjuk! Nem vagyunk kötelesek ugyanazt a címet adni a visszamásoláshoz. Tehát kimentjük a képernyőt, és az előkészített, egy lapnyi szöveget egy mozdulattal bemásolhatjuk a képernyőre. Ha már nem kell, egy hívással helyreállíthatjuk a képernyőt, így:

```
                LES     DI, SAVE_BUFF           ;Mentőbuffer
                CALL    SCR_SAVE                ;Mentés
                LES     DI, TEXT_BUFF           ;Szövegbuffer
                CALL    SCR_RETRIEVE            ;Szöveg ki
                ...                             ;Várakozás
                LES     DI, SAVE_BUFF           ;Nem történt
                CALL    SCR_RETRIEVE            ; semmi, ...
```

Nincs akadálya annak, hogy így több lapnyi szöveget is váltogassunk a képernyőn.

Még egy apróság: színes ernyőn a rutin az SCRD struktúrából tudta meg, hogy melyik a kurrens lap. A lapváltás felhasználásával mentés nélkül is kivihetünk egész lapnyi információt, csak előbb ismerkedjünk meg az alábbi makróval:

```
PAGESEL MACRO   PAGE                            ;
                PUSH    AX                      ;;Jó lehet még
                MOV     AL, PAGE                ;;Ezt kérték
                MOV     CS: SCRD.CPAG, AL       ;;Admin.
                VIDCLL  VID_STPAGE              ;;Lapváltás
                POP     AL                      ;;
        ENDM                                    ;
```

E makró segítségével úgy váltunk képernyőlapot, hogy egyúttal nyilván is tartjuk a változást az SCRD struktúra megfelelő elemében. Ezzel pedig már az SCR_RETRIEVE rutin tudtára is adtuk, hogy melyik lapot kell írnia. A program által használt kurrens lapot tehát nem szükséges menteni, hanem egyszerűen egy eddig nem használt, háttérben levő lapra írhatunk.

Az Olvasó talán elnézi, hogy nem adtuk meg annak a programrésznek a kódját, amely a file-ból valamilyen bufferbe olvassa a kért szöveget; ennek megoldása kedvéért érdemes a második kötetet fellapozni.

IV.6.3. A megjelenítés ki- és bekapcsolása

Elég sok esetben lehet arra szükség, hogy valamilyen okból ki- és bekapcsoljuk a képernyőn való megjelenítést. Ez nem túl nehéz feladat akkor, ha ismerjük a pillanatnyilag érvényben levő módparancsot. Ha ilyen "alattomos" dolgokra szánjuk magunkat, akkor gondos tervezéssel kell eljárnunk. Legcélszerűbb egy teljes képernyőkezelő modult alkotnunk, amelyben teljes rendszert valósítunk meg. Az első lépés mindig az, hogy lekérdezzük a monitor típusát és állapotát, és ezt a modul belső változóiban elmentjük. Ezekre a változókra számíthatunk aztán a későbbiekben. Ezúttal tegyük fel, hogy programunk valahol nyilvántartja a kurrens video módot (ez az SCR_GETPAR által feltöltött SCRD struktúra egyik eleme; neve VIDMOD). Ekkor igen egyszerű lesz a dolgunk:

```
; SCR_DV        - video kikapcsolása
;-----------------------------;----------------;
VIDEO_BASE      EQU     3D0H    ;3B0 is lehet   ;Alapportcím
VIDEO_CONT      EQU     08H     ;---------------;Vezérlőport
VP_CONT_ENVID   EQU     08H                     ;Video be/ki
                                                ;
SCR_DV  PROC    NEAR                            ;
                PUSH    AX                      ;Mentés
                PUSH    DX                      ;
                MOV     AL, CS: SRCD.VIDMOD     ;
                AND     AL, NOT VP_CONT_ENVID   ;Video kikapcs.
                MOV     DX, VIDEO_BASE + VIDEO_CONT
                OUT     DX, AL                  ;
                POP     DX                      ;
                POP     AX                      ;
                RET                             ;
SCR_DV  ENDP                                    ;
```

```
;-------------------------------------------------------;
; SCR_EV        - video bekapcsolása                    ;
;-------------------------------------------------------;
SCR_EV  PROC    NEAR                                    ;
                PUSH    AX                              ;
                PUSH    DX                              ;
                MOV     AL, CS: SRCD.VIDMOD             ;
                OR      AL, VP_CONT_ENVID               ;Bit beáll.
                MOV     DX, VIDEO_BASE + VIDEO_CONT
                OUT     DX, AL                          ;
                POP     DX                              ;
                POP     AX                              ;
                RET                                     ;
SCR_EV  ENDP                                            ;
```

Ennél bonyolultabb a feladat, ha nem ismerjük az érvényben levő módparancsot, hanem a videomódból (melyet lekérdezhetünk) nekünk kell előállítani. A lekérdezés egyszerűen a 15H funkcióval (VID_GTMODE) lehetséges. A kapott funkciókóddal, felhasználva az "egyéb ROM BIOS interruptok" fejezetben leírt video paramétertáblát, melynek címe az 1DH interrupt vektorban van, egy egyszerű (?) bázisregiszteres indirekt címzéssel kaphatjuk meg a módparancsot (csak egy programrészletet közlünk):

```
                ...
                PUSH    BX                      ;
                PUSH    SI                      ;
                PUSH    DI                      ;
                PUSH    BP                      ;
                PUSH    ES                      ;Regisztermentés
                                                ;
                VIDCLL  VID_GTMODE              ;
                XOR     AH, AH                  ;AX - video mód
                PUSH    AX                      ;
                MOV     AL, BT_VIDPAR           ;Interrupt sorszám
                DOSCLL  DOS_GETITVECT           ;Interrupt lekérdezés
                                                ;ES:BX -> video tábla
                POP     SI                      ;Mód (index) SI-be
                MOV     AL, ES:MODE_COMMS.[ SI ][ BX ]
                POP     ES                      ;
                POP     BP                      ;
                POP     DI                      ;
                POP     SI                      ;
                POP     BX                      ;
                ...                             ;
```

A fenti kis programrészlet elmenti a munkaregisztereket (és még végez egy pár "luxusmentést", mivel a video-interrupt elront(hat)ja a DI és BP regisztereket is), lekérdezi a videomódot (AH-ba Ø-t tesz, mert AL értékét később indexként használja majd); ezután ES:BX-be bekéri az 1DH interrupt vektor tartalmát, azaz a videotábla címét. SI-be visszahozza az előzőleg elmentett videomódot, és a hosszú-hosszú indexelt címzésű utasítással kiveszi a módparancsot a videotáblából. Ezzel aztán azt csinálunk, amit akarunk; például fehasználhatjuk a fenti két szubrutinban is.

IV.6.4. Karakter és attribútum direkt kiírása

Egy karakter és a hozzá tartozó attribútum direkt kiírása nem túl nehéz feladat akkor, ha fekete-fehér adapterünk van. Ez ugyanis a megjelenítés bármely fázisában megengedi, hogy közvetlenül írjunk a képernyőterületre, a színes adapter esetén azonban meg kell várnunk azt a pillanatot, amikor a képernyő és a memória között éppen nincs direkt kapcsolat (a függőleges visszatérítés idején). E célból be kell olvasnunk az adapter státuszát, meg kell vizsgálnunk a Ø. bitet (Display Enable), és ismételt beolvasásokkal addig várnunk, míg ez a bit egyes értéket nem kap. Ez a trükk lényege, azonban az alaposabb átgondolás kicsit nagyobb óvatosságra késztet. Mi történik akkor, ha rutinunk éppen olyan pillanatban indul, mikor a képernyővisszafutás folyamatban van? Ez látszólag nagyon jó lenne, mert azonnal írhatnánk is ki a karaktert. Azonban a visszafutás ideje hihetetlenül rövid. Kimértem, hogy még egy IBM AT-vel is mindössze három-négy, legfeljebb hat karaktert lehet kivinni a visszatérítés ideje alatt (REP MOVSW utasítást használtam). Tehát egyáltalán nem biztos az, hogy ha a visszafutási idő vége felé érkezünk, akkor valóban el tudjuk végezni a kiírást, mielőtt befejeződne. Először tehát azt nézzük meg, hogy visszafutási fázisban van-e az ernyő. Ha igen, megvárjuk ennek a végét; ha nem, akkor rögtön a második várakozási ciklusba lépünk, és megvárjuk, míg a monitor ismét várakozási ciklusba lép. Ezt a feladatot egyébként legcélszerűbb egy makró segítségével megoldani. Magát a várakozást semmiképpen nem oldhatjuk meg szubrutin segítségével, mert mire a rutinból visszatérünk, már vége is a visssszatérítésnek!

Mielőtt megoldanánk a feladatot, gondolkodjunk el még egy kicsit. Megvártuk, hogy a képernyő frissítése során elkezdődjön a visszatérítési szakasz, s mikor ez végre bekövetkezik, írnánk ki lelkesen a karakterünket. De ebben a drámai pillanatban váratlanul becsap egy interrupt (például a Timer Ø8 sorszámú in-

terruptja, ami nem ritka eset: másodpercenként tizennyolcszor érkezik), és megakasztja programunk futását, tehát elrontja a játékunkat. Ezt meg kell akadályoznunk: a második várakozási ciklust bizony már az interrupt-rendszer letiltásával kell végrehajtanunk.

Megjegyzés: kezdetleges módszerrel ugyan, de megpróbáltam meghatározni a frissítés és a visszatérítés időarányait. Az eljárás az volt, hogy az interrupt-rendszer letiltása után megvártam egy ciklus kezdetét, majd úgy hajtottam végre egy teljes ciklust, hogy közben elsőre szavas, később duplaszavas számlálókban számoltam a kiváráshoz szükséges beolvasásokat (a duplaszóra azért volt szükség, hogy igazoljam: nem csordultak túl a szavas számlálók). Mivel különböző időtartamokat ugyanannyi utasítás végrehajtásának idejével növeltem, a kapott arány egy kicsit csalóka. Az eredmények: egyszavas számlálókat használva a visszatérítés-frissítés arány 8:18, míg duplaszavas számlálók használata esetén (melyek növelése lassúbb) 6:13. A valóságos arány tehát kisebb, mint a 8:18.

Az alábbiakban megadjuk a makrót, mely elvégzi ezt a feladatot. De nem viszi ki a karaktert, mivel többféle kiírást végezhetünk. Hasznos lehet a karakter-attribútum pár kiírása, de lehet, hogy csak a karaktert vagy csak az attribútumot akarjuk frissíteni. Ezért választjuk el a várakozási makrótól magát a kivitelt. De ez veszélyes lépés, mert a várakozás és a kiírás között nem szabad interruptot beengednünk; emiatt viszont a várakozási makró az Interrupt flagben nullát hagy. Ha a makró hívója nem végzi el azonnal a kiírást és utána nem engedélyezi az interruptokat, akkor gyakorlatilag megállította a rendszert. Ezért nem egy, hanem négy makrót írunk. Az első, melyet a felhasználó számára nem dokumentálunk és el se áruljuk, hogy létezik, WR_RETR-nek hívjuk; feladata a várakozás megvalósítása. A többiek egymás közeli rokonai: a WRCHAT nevű makró karaktert és attribútumot ír ki, a WRCHR nevű csak karaktert, míg a WRATTR csak attribútumot. A három utóbbi makró meghívja az elsőt, és ügyelnek arra, hogy felszabadítsák az interrupt-rendszert! Bemeneteik opcionálisak: egy nyolcbites karakterkód, egy nyolcbites attribútum és egy tizenhatbites cím. Ez a módosítani kívánt pozíció offsetje a képernyőszegmens kezdetéhez viszonyítva. Ha a 10 sor 34 pozíciójába akarunk kiírni, akkor ez az offset a

10 * sorhossz + 34

képlettel állítható elő. A tapasztalatok azt mutatták, hogy ez a definíció igen kényelmes; könnyen térhetünk át egy szomszédos pozícióra vagy sorra. Lássuk akkor a makrókat!

```
; WT_RETR        - várakozás a visszatérítés kezdetéig
;       Bemenet:
;                DSPSEG  - a képernyő szegmense
;-------------------------------------------------------;
        VIDEO_STATUS    =       0AH             ;A szükséges
        VP_STAT_ENDISP  =       01H             ; konstansok
WT_RETR         MACRO   DSPSEG          ;;------;
        LOCAL   WAIT_NO_RETR, WAIT_RETR ;;
                MOV     DX, SEG DSPSEG  ;;Képernyő szegmenscím
                MOV     ES, DX          ;; töltése ES-be
        ASSUME  ES:DISPLAY              ;;
                MOV     DX, VIDEO_BASE + VIDEO_STATUS
WAIT_NO_RETR:                           ;;Visszatérítés
                IN      AL, DX          ;; végéig várunk
                TEST    AL, VP_STAT_ENDISP
                JNZ     WAIT_NO_RETR    ;;
                CLI                     ;;; !!!
WAIT_RETR:                              ;;;Visszatérítés
                IN      AL, DX          ;;; kezdetéig várunk
                TEST    AL, VP_STAT_ENDISP
                JZ      WAIT_RETR       ;;;
                ENDM                    ;
;-------------------------------------------------------;
; WRCHAT        - karakter és attributum direkt kiírása
;   Opcionális bemenetek:
;       CHAR    - 8 bites karakterkód (legyen CL-ben!)
;       ATTR    - 8 bites attributum (legyen CH-ban!)
;       POS     - 16 bites logikai képernyőcím (legyen BX-ben!)
;       DSPSEG  - a képernyő szegmense
;-------------------------------------------------------;
WRCHAT  MACRO   CHAR, ATTR, POS,  DSPSEG
        IFNB    <CHAR>                  ;;Ha CHAR megadott, ...
                MOV     CL, CHAR        ;;
        ENDIF                           ;;
        IFNB    <ATTR>                  ;;Ha ATTR megadott, ...
                MOV     CH, ATTR        ;;
        ENDIF                           ;;
        IFNB    <POS>                   ;;Ha POS megadott, ...
                MOV     BX, POS         ;;
        ENDIF                           ;;
                SHL     BX, 1           ;;Logikai -> fizikai
                WT_RETR DSPSEG          ;;Várakozás
                MOV     ES:[ BX ], CX   ;;;Kiírás az ernyőre
                STI                     ;; !!!
        ENDM                            ;
```

```
; WRCHR          - karakter direkt kiírása
;   Opcionális bemenetek:
;       CHAR     - 8 bites karakterkód (legyen CL-ben!)
;       POS      - 16 bites logikai képernyőcím (legyen BX-ben!)
;-------------------------------------------------;
WRCHR   MACRO   CHAR,  POS,  DSPSEG     ;
        IFNB    <CHAR>                  ;;Ha CHAR megadott, ...
                MOV     CL, CHAR        ;;
        ENDIF                           ;;
        IFNB    <POS>                   ;;Ha POS megadott, ...
                MOV     BX, POS         ;;
        ENDIF                           ;;
                SHL     BX, 1           ;;Logikai -> fizikai
                WT_RETR DSPSEG          ;;Várakozás
                MOV     ES:[ BX ], CL   ;;;Kiírás az ernyőre
                STI                     ;; !!!
        ENDM                            ;
;-------------------------------------------------;
; WRATTR         - attributum direkt kiírása
;   Opcionális bemenetek:
;       ATTR     - 8 bites attributum (legyen CH-ban!)
;       POS      - 16 bites logikai képernyőcím (legyen BX-ben!)
;-------------------------------------------------;
WRATTR  MACRO   ATTR,  POS,  DSPSEG     ;
        IFNB    <ATTR>                  ;;Ha ATTR megadott, ...
                MOV     CH, ATTR        ;;
        ENDIF                           ;;
        IFNB    <POS>                   ;;Ha POS megadott, ...
                MOV     BX, POS         ;;
        ENDIF                           ;;
                SHL     BX, 1           ;;Logikai -> fizikai
                WT_RETR DSPSEG          ;;Várakozás
                MOV     ES:[ BX+1 ], CH ;;;Kiírás az ernyőre
                STI                     ;; !!!
        ENDM                            ;
```

Fontos apróság, hogy (mivel a WT_RETR makróban adtuk meg a szükséges konstansokat) nem használhattuk az EQU operátort, mivel az a makró többszöri meghívása esetén többszörösen definiálná a szimbólumokat. A tiszta megoldás persze a VIDEO.INC file beemelése, a konstansok definíciójának ott a helye.

Már csak arra hívjuk fel az Olvasó figyelmét, hogy a képernyőterület szegmenscímét a hívónak meg kell adnia. Ez lehet AT operátorral a megfelelő pozícióra elhelyezett szegmens neve, de lehet a képernyőterületre helyezett szegmensben levő címke is, mert a makróban a paraméter nevét megelőzi a SEG kulcsszó.

V. A nyomtatóadapter

V.1. A nyomtatóadapter működése és fizikai kezelése

Az IBM PC/XT egy párhuzamos printervezérlővel rendelkezik. Ennek neve PPA, "Paralell Printer Adapter". A párhuzamos szó azt jelenti, hogy az adatforgalom nyolc adatvonalon át folyik.

E fejezetben nem taglaljuk, hogy egy adott nyomtató, pl. az EPSON FX-100 milyen lehetőséggekkel rendelkezik, hiszen ez nem a gép vagy egy belső elem sajátossága. Célunk a nyomtató (vagy egyéb periéria) vezérlésére szolgáló PPA vázlatos ismertetése.

V.1.1. A PPA csatlakozója és vonalai

Első lépésként ismerkedjünk meg a PPA "külső szolgáltatásaival", azaz a külső csatlakozó felépítésével. Ez egy 25 pólusú "D" alakú csatlakozó, amelynek kiosztása a következő:

Sorszám		Vonal neve	KSorszám	
	1	<---------- Strobe ---------->	1	
	2	<---------- Data bit 0 ---------->	2	
	3	<---------- Data bit 1 ---------->	3	
	4	<---------- Data bit 2 ---------->	4	
	5	<---------- Data bit 3 ---------->	5	
Para-	6	<---------- Data bit 4 ---------->	6	
	7	<---------- Data bit 5 ---------->	7	
	8	<---------- Data bit 6 ---------->	8	
	9	<---------- Data bit 7 ---------->	9	
llel	10	<---------- -Acknowledge ----------	10	Prin-
	11	<---------- +Busy ----------	11	
	12	<---------- +Out of Paper ----------	12	
	13	---------- +Select ---------->	13	
Prin-	14	<---------- -Auto Feed ----------	14	
	15	<---------- -Error ----------	15	
	16	--------- -Initialize Printer --------->	16	
	17	<---------- -Select Input ----------	17	
ter	18	---------- Ground ----------	18	ter
	19	---------- Ground ----------	19	
	20	---------- Ground ----------	20	
	21	---------- Ground ----------	21	
Adapt.	22	---------- Ground ----------	22	
	23	---------- Ground ----------	23	
	24	---------- Ground ----------	24	
	25	---------- Ground ----------	25	

A fenti angol nyelven adott nevű vonalak szerepe röviden a következő (a nevek előtti mínuszjel arra utal, hogy az adott vonal komplementált):

- Strobe ..... Érvényes adat jelzése
  Az adapter (vagy beolvasás esetén a PPA által vezérelt külső eszköz) e vonalon kiadott jellel hozza a másik oldal tudomására azt, hogy az adatvonalakon érvényes adat van.
- Data Bit 0-7 ..... Párhuzamos adatbitek
- Acknowledge ..... A vétel nyugtázása
  Ezen a vonalon nyugtázza a külső eszköz a kiküldött adatok vételét (negált érték).
- Busy ..... Az eszköz foglalt
  Ezen a vonalon jelzi a külső eszköz, hogy nem tud karaktert fogadni, mert dolgozik.
- Out of Paper ..... Elfogyott a papír
  Ezen a vonalon jelzi a külső eszköz, hogy felhasználói beavatkozást igénylő ok miatt nem tudja folytatni a munkát. Ez nyomtató esetében jellegzetesen a papír kifogyása.
- Select ..... Az üzemkész állapot jelzése
  E vonalon jelzi a külső eszköz, hogy működésre kész. Nyomtató esetén ez a vonal az ON LINE/OFF LINE állapotot jelzi.
- Auto Feed ..... Automatikus soremelés be- és kikapcsolása
  Ez a vonal jellegzetesen a nyomtató vezérlésére szolgál. Bizonyos nyomtatók (így például az IBM Parallel Printere) képesek arra, hogy egykarakteres kocsivezérlő "sorozatokat" fogadjanak; ekkor a kocsivissza karakterhez automatikusan fűzik a soremelést. E funkciót vezérli az "Auto Feed" vonal (negált érték).
- Error ..... Hibajelző vonal
  Ezen a vonalon jelzi a külső eszköz, hogy működésében valami hiba lépett fel (negált érték).
- Initialize Printer ..... Utasítás a nyomtató előkészítésére
  Az adapter ezen a vonalon kiadott jellel hozza alapállapotba a külső eszközt (negált érték).
- Select Input ..... Az adapter fogadásra kész
  Az adapter ezen a vonalon jelzi a külső eszköznek, hogy kész a kívülről küldendő adat fogadására. Nyomtató vezérlése esetén e bitnek nincs jelentősége (negált érték).
- Ground ..... A nyomtató és a PPA közös földje

E vonalak közül számosat közvetlenül I/O portokra való kiírással vezérelhetünk, vagy a külső eszköz által ott létrehozott jeleket közvetlenül olvashatjuk. Ebből következik, hogy a PPA szintén portos vezérlésű, portjainak címe a hexa 37CH-tól 37EH-ig terjed. A második PPA portjai a hexa 378H-tól 37AH-ig terje-

dő címeken helyezkednek el. A monokróm képernyő-adapterbe épített printervezérlő portjai 3BCH-tól 3BEH-ig terjednek.

Egy printeradapternek két input-output és egy input portja van. A legalacsonyabb című port (37CH vagy 378H) adatkiküldésre vagy fogadásra szolgál, a legmagasabb címűn (37EH vagy 37AH) parancsokat adhatunk ki. Ennek az alsó 5 bitje értékes, és számunkra csak a 0. bit fontos, amelynek 1-be állítása kiküldi a printerre a megfelelő output portra előzőleg már kivitt karaktert.

Mint a vonalak tanulmányozásából kitűnik, a PPA nem csupán nyomtatók, hanem egyéb párhuzamos felületen illeszthető perifériák kezelésére is alkalmas. Azért használják elsősorban nyomtatóvezérlésre, mert a nyomtatók a gép közvetlen közelében helyezkednek el; a sérülékenyebb kábel, és a nagyobb távolság nem okoz nehézségeket. Hardware szinten egyébként egyszerűbb és olcsóbb egy párhuzamos, mint egy soros illesztés.

V.1.2. A printer adapter portjai

Az alábbiakban részletesen megismerkedünk a fent említett három I/O port használatával és a portok bitjeinek jelentésével. A portokat nullától kettőig számozzuk; a 0. sorszámú jelenti a legalacsonyabb című portot (378H, 37CH vagy 3BCH), és így tovább, emelkedő sorrendben.

- 0. port: adatport
  E port bitjei emelkedő sorrendben felelnek meg a csatlakozó 2-9. vonalainak, az adatvonalaknak. A port írásával közvetlenül vezéreljük az adapter adatbitjeit, beolvasással az adapterrel vezérelt külső eszköz által küldött adatot kapjuk meg.
- 1. port: státuszport
  Ezen a porton input utasítás végrehajtásával a legfontosabb státuszinformációkat kérdezhetjük le; a megadott bitek közvetlenül kapcsolódnak a csatlakozó megfelelő vonalaihoz:

7	6	5	4	3	2	1	0
11. v. Busy	10. v. Acknowlegde	12. v. Paper End	13. v. Select	15. v. Error	Nem használt	Nem használt	Nem használt

A bitek pontosan megfelelnek az adapter és az általa vezérelt periféria közötti felület megadott vonalainak, nem igényelnek külön magyarázatot. Ne feledjük, hogy az Acknowlegde és az Error bitek komplementáltak.

- 2. port: státusz- és vezérlőport
  Kiírás esetén a port bitjeinek jelentése a következő:

7	6	5	4	3	2	1	0
Nem használt	Nem használt	Nem használt	Interrupt Enable	17. v. Select Input	16. v. Init. print.	14. v. Auto Feed	1. v. Strobe

A második port alsó négy bitje közvetlenül a csatlakozó négy megadott vonalához van kötve; az ötödik bit az interrupt letiltását-engedélyezését vezérli. Ha kiírást végzünk, akkor vezérelhetjük az e vonalakra küldött jelet, a port olvasásával pedig lekérdezhetjük a másik oldal által küldött jeleket (ennek akkor van jelentősége, ha a PPA-t nem egy nyomtató, hanem valamilyen egyéb eszköz vezérlésére használjuk).
Beolvasáskor a bitek jelentése pontosan megegyezik. Ha az adapterrel nyomtatót vezérlünk, akkor a beolvasással pontosan azt fogjuk visszakapni, amit az utolsó kiírással kivittünk. Ha valamilyen egyéb külső eszközt vezérlünk, akkor a perfiéria (az interrupt-bit kivételével) módosíthatja a bitek jelentését! A bitek között csak az interrupt-bit szorul bővebb magyarázatra:
- Interrupt Request Enable
  A nyomtató interruptjának engedélyezése. Ha ezt a bitet egybe állítjuk, akkor az adapter interruptot fog adni minden esetben, ha a 10. vonalon bejön egy jel (azaz a printer nyugtázza a kiküldött karakter vételét).

V.2. Nyomtatókezelési funkciók

Interrupt-sorszám: 17H
Javasolt interrupt-név: BS_PRN
Az adapter software interruptjának hívására javasolt makró:

```
PRTCLL  MACRO   CODE, PRINTNUM          ;
        IFNB    <PRINTNUM>             ;;
                MOV     DX, PRINTNUM   ;;
        ELSE                           ;;
                XOR     DX, DX         ;;
        ENDIF                          ;;
                MOV     AH, CODE       ;;
                INT     BS_PRNCOM      ;;
        ENDM                           ;
```

Három funkció áll rendelkezésünkre: az egyik egy karakter kiküldésére, a másik a printer előkészítésére, a harmadik pedig a printer státuszának beolvasására szolgál.

V.2.1. Karakter kiíratása

Funkciókód: AH = 00H INT 17H
Javasolt konstansnév: PRN_CHROUT
Be:
AL a kiírandó karakter
DX a kívánt nyomtató sorszáma
Válasz:
AH a nyomtató státusza (lásd 02H funkció)

Megjegyezzük, hogy egyes nyomtatók bufferelik a kiküldött karaktereket, így ha egy-egy karaktert kiküldünk, nem biztos, hogy a nyomtató azonnal dolgozni kezd. A buffer mérete nyomtatótól, sőt az adott nyomtató felprogramozásától függően is változik. Egy EPSON FX-100 nyomtatón például akkor kezdődik el a nyomtatás, ha egy teljes sor bent van a nyomtató bufferében (azaz egy olyan karaktersorozat, amelynek a végén ott a kocsi-vissza-soremelés), vagy pedig 136 karakter megérkezése után.

V.2.2. A nyomtatóvezérlő előkészítése

Funkciókód: AH = 01H INT 17H
Javasolt konstansnév: PRN_INI
Be:
semmi egyéb

Válasz:
AH a printer státusza (lásd 02H funkció)

V.2.3. A nyomtatóvezérlő státuszának lekérdezése

Funkciókód: AH = 02H INT 17H
Javasolt konstansnév: PRN_GTSTATE

Be:
semmi egyéb

Válasz:
AH a printer státusza; a 2. port ismertetésénél felsorolt 5 bit, melyek közül kettő komplementált értékű, és a nem használt bitek töröltek.

A komplementált bitek a hatodik és a harmadik (Acknowledge és I/O error). A státusz bitjei a következők:

- Ø. bit: Time Out - a printer megadott időn belül nem válaszolt (valószínűleg ki van kapcsolva)
- 1. és 2. bit: nem használt
- 3. bit: I/O error - valamilyen átviteli hiba következett be.
- 4. bit: Selected - 1 értéke jelzi,hogy a printer üzemkész, ON LINE állapotban van.
- 5. bit: Out of Paper - 1 értéke jelzi, hogy kifogyott a papír
- 6. bit: Acknowlegde - az itt szereplő bit 1 értéke nyugtázza a kiadott karakter fogadását
- 7. bit: Busy - a nyomtató dolgozik, nem fogad karaktert.

Látható, hogy ez a státusz pontosan megfelel az adapter 1. portja bitkiosztásának. Az egyetlen eltérés: a porton nem használt Ø. bitet a ROM BIOS driver arra használja, hogy jelezze a Time Out eseményt, vagyis azt, hogy a nyomtató a megadott időn belül nem nyugtázta Acknowlegde jellel a kiküldött karaktert.

A printer "kézzel" vezérlése nem szokásos művelet. Sokkal jobban járunk, ha a kinyomtatandó szövegeket a standard printer file-ra küldjük ki (lásd 2. kötet), vagy pedig a ROM BIOS printerkezelő funkcióit használjuk.

V.3. A hard copy működése

A hard copy - a képernyő tartalmának kinyomtatása - a CTRL-PRTSC billentyűkombináció leütésére lép működésbe. E funkció részletes ismertetését lásd a "Klaviatúra" c. fejezetben.

VI. Az aszinkron vonali adapter

VI.1. Az aszinkron vonal működése

A PC/XT aszinkron vonali kommunikációra is képes a beépített (egyébként az International Semiconductor cégtől származó) 8250 típusjelű "Asynchronous Communication Element", az ACE segítségével. Ez az aszinkron vonal soros vonal, ami azt jelenti, hogy egy vezeték(pár)on folyik az adatforgalom: a biteket nem egy időben, hanem egymás után küldik ki a vonalra. Aszinkronnak pedig azért mondjuk, mert a berendezések csak egy-egy karakter átküldésének idejére veszik fel a bitszinkront.

> Megjegyzés: a szinkron kommunikáció során az egymással szembenálló berendezések külön órajel segítségével akár végtelen hosszú időre, a gyakorlatban több száz vagy több ezer karakter átküldésének idejére veszik fel a bitszinkront, és a megállapodott karakterhosszúság szerint vágják fel a bitfolyamot különálló karakterekre.

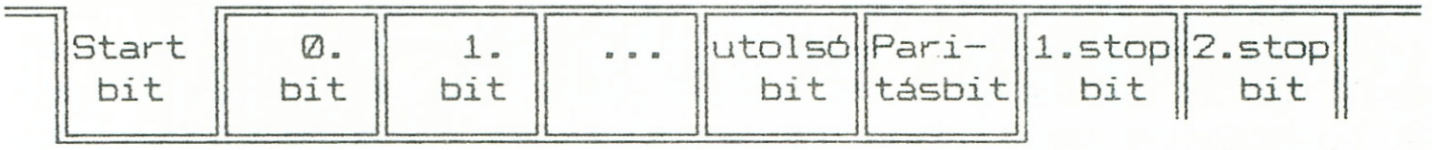
Algoritmikus szempontból (és elektronikusan is) valamivel egyszerűbb az aszinkron vonal kezelése.

Szóljunk egy-két szót az RS 232 elnevezéséről is! Olyan interface-ről (két szembenálló berendezés közötti felület) beszélünk majd, amely számos (általában huszonöt) áramkört tartalmaz. Ezen áramkörök leírását a CCITT V.24 számú ajánlása tartalmazza, ezért ezt az interface-t V.24-nek szokás nevezni. Gyakran hivatkoznak az ennek megfelelő amerikai szabványra, melynek jele RS 232. A két leírás gyakorlatilag azonos. Az IBM PC/XT-be épített aszinkron vonali adapter elnevezése természetesen nem RS 232; csak a "köznyelvben" ragadt rá ez az elnevezés. A továbbiakban egyszerűen adapternek fogjuk szólítani vizsgálatunk tárgyát.

Az aszinkron vonali kommunikáció menete durván a következő: a berendezések előkészítése (melyre most nem térünk ki) után a küldő berendezés startbitet küld ki a vonalra, amelyet a fogadó érzékel, felveszi a bitszinkront, és felkészül a további bitek fogadására. A küldő folyamatosan kiküldi a karakter bitjeit a vonalra, esetleg a megállapodás szerinti paritásbitet (később részletezzük) is elküldi, végül egy vagy két stopbittel zárja le a karakter elküldését. A jelszint a vonalon ezután meghatározatlan ideig 1 marad, és az újabb startbit jelenti az újabb karakter érkezését. Látható, hogy az aszinkron kommunikáció nem túlzottan gazdaságos, mert minden karakterhez 2-3 "felesleges" szinkronizáló bitet is át kell küldeni. Az aszinkron port ter-

mészetesen "full-dupplex", azaz egy időben mindkét oldal képes venni a másik oldalról küldött biteket, és folytatni az általa megkezdett adást.

Egy karakter képe a vonalon tehát a következő:



Ismerkedjünk meg az alapfogalmakkal! A legfontosabb talán a "baud rate", amely azt mondja meg, hány bit/sec sebességgel folyik az adatforgalom. Az RS 232 szerinti értékek igen széles sebességtartományba esnek: a szokásos alsó és felső határ 50, illetve 38400 bit másodpercenként. Megjegyezzük, hogy megfelelő programozással az adapter lényegesen meghaladhatja a szabványban előírt sebességhatárokat. A 8250 egyébként nem a gép órajelét használja a vonal kezelésére, hanem saját belső óráját. A vonalra küldött karakter extra bitjei miatt a 9600 bit/sec sebesség nem azt jelenti, hogy másodpercenként 1200 karakter megy át, hiszen át kell küldeni a start- és stopbiteket, valamint a paritást is!

> Megjegyzés: a ROM BIOS az RS 232-ben előírt sebességtartományból a 110, 300, 600, 1200, 2400, 4800 és 9600 bites másodpercenkénti sebességet valósítja meg.

A paritás az adatforgalom ellenőrzésére szolgál. Nagyjából a következőket jelenti (tegyük fel, hogy páros paritást ellenőrzünk): a küldő megszámolja az egyes értékű biteket az átküldött karakterben, és ha az egyesek száma páros, elküld még egy egyes értékű bitet, ha pedig páratlan számú egyest tartalmazott a karakter, akkor egy nulla bitet küld kiegészítésként. Ez a külön bit hordozza tehát azt az információt, hogy az átküldött karakterben páros vagy páratlan számú egyes bit van. A vevő szintén megszámolja az egyes értékű biteket, kialakítja saját elképzelését a paritásról, és ezt összehasonlítja az adó által átküldött paritásbittel. Ha csak egy bit sérül meg az átvitel során, akkor a hiba mindenképpen kiderül. Sajnos, ha két bit sérül meg, akkor nem vesszük észre a hibát. Az előbb elmondottakat nevezik páros keresztparitásnak.

Természetesen ismeretes a páratlan paritás fogalma is: ez abban különbözik a párostól, hogy nem páros számú egyes bitre küld egyes értéket paritásként, hanem páratlan számú esetén. Magától értetődik, hogy a vevő és az adó csak azonos karakter-

hosszúsággal és azonos paritásellenőrzési móddal tud együttműködni; ha megfordítjuk a vevő paritásellenőrzését, akkor az minden egyes jól átküldött karaktert paritáshibásnak érzékel.

Szokás még hosszparitásra is ellenőrizni: a sorban egymást követő karakterek "egymás alatti" (azonos helyiértékű) bitjei paritását is kiszámítják, és ezt, mint egy utolsó karktert, elküldik az üzenet végén. Utolsó lépésként pedig meghatározzák a keresztparitások hosszparitását, és a hosszparitások keresztparitását (ez már egyetlen bit). A két értéknek természetesen egyeznie kell. Ha nem túl sok a hiba az üzenetben, vagy nem túl szerencsétlenül helyezkednek el, akkor ez az eljárás kimutatja.

Az adapter minden lehetséges esetre felprogramozható: kívánság szerint el is maradhat a paritás átküldése, ha átküldjük, lehet páratlan és páros paritást ellenőrizni; megadhatjuk azt is, hogy paritásbitként rögzítetten Ø vagy 1 érték szerepeljen. Megjegyezzük, hogy ez utóbbit a ROM BIOS driver nem teszi lehetővé.

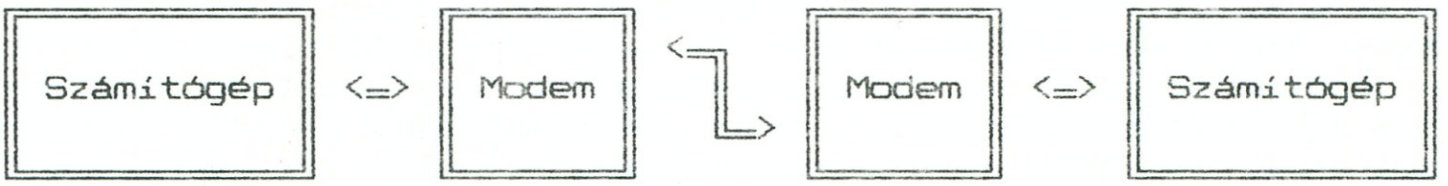
A tapasztalatok szerint egyébként a paritásellenőrzés nem hoz igazán jó eredményeket, mert a vonali hibák gyakran jelentkeznek "csomóban" (azaz számos egymáshoz közeli bit sérül meg), ilyen esetben pedig könnyen fordulhat elő az, hogy hiába vizsgálunk hossz- és keresztparitást, az nem mutatja ki a hibát. Korszerűbb és hatékonyabb módszereket kínálnak az összefoglaló néven CRC-nek (Cyclic Redundancy Check) nevezett eljárások, melyek bonyolultabb algoritmussal képeznek ki valamilyen ellenőrző bitsorozatot. Az egyik elterjedt módszer az, hogy az átküldendő üzenetet mint sokbites számot egy 16 bites konstanssal elosztják, és a maradékot is átküldik az üzenet végén. A fogadó is kiszámítja a maradékot (ez a CRC), és összeveti a kapottal. Ha a kettő nem egyezik, az üzenet hibás. Nagy előnye ennek a CRC-nek, hogy karakterenként is képezhető, így kiszámítása az üzenet karakterenkénti vétele közben is lehetséges, nem szükséges az egész üzenetet megvárni.

Tudnunk kell még, hogy az ACE képes 5, 6, 7 és 8 bites karakterek küldésére és vételére is. Megszabhatjuk a stopbitek számát is: küldhetünk/várhatunk egy vagy két stopbitet.

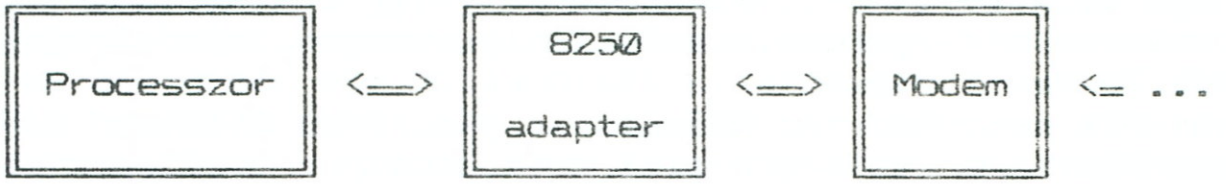
> Megjegyzés: a ROM BIOS csak 7 vagy 8 bit hosszúságú karakterek mozgatását teszi lehetővé. Igen fontos megjegyzés még, hogy az adapter által kezelt vonalon mindkét irányban ugyanazon szabályok szerint kell folynia az adatforgalomnak.

VI.2. Az aszinkron interface leírása

Ismerkedjünk meg az aszinkron kommunikáció alapfogalmaival! Maga az adatforgalom vagy két közvetlenül egymás közelében levő számítógép között folyik, vagy a fizikailag távolabb levő gépeket modemek közvetítésével telefonvonal köti össze egymással. Nem ritka eset, hogy a soros aszinkron vonalon egy perifériális eszköz, például egy terminál vagy egy nyomtató lóg. Megjegyezzük, hogy a népszerű egér kezelésének legegyszerűbb módja a soros vonal használata. A jellegzetes eset a számítógép-modem kapcsolat. Ez jól mutat rá a kommunikáció azon aspektusára, hogy a kommunikáció nem egyenrangú felek között zajlik; jólehet a modem is fordulhat különféle kérésekkel a számítógéphez, a gép mégis uralkodó helyzetben van.



A "<=>" jelzi a számítógép és a modem közötti interface-t, míg a megtört vonal a modemeket összekötő telefonvonalat. Számunkra ez utóbbi nem lényeges, egyedül a számítógép-modem kapcsolat vezérlése van a kezünkben. A számítógép azonban csak a modem szemszögéből nézve egy egység. Számunkra az aszinkron kommunikáció természetesen a processzor és az INS8250 adapter közötti interface-en át történik. Ezen az interface-en input és output utasításokkal kommunikál a program és az adapter:



Lássuk először az adapter és a modem, majd a processzor és az adapter közötti felületet! Az előbbi a vonalak, az utóbbi a 8250 belső regisztereinek megismerését jelenti. Elnézést kérek az angol nyelvben járatlan Olvasótól, de az egyes vezetékek, valamint a regiszterek nevét angolul adom meg, mégpedig egyszerűen azért, mert úgy sem fogunk e nevekkel magyarul találkozni. Az eredeti szakirodalom olvasása során nem hátrány, ha tudjuk azonosítani a vonalakat, a biteket. A táblázat alatt megadom minden elnevezés magyar változatát és rövid magyarázatát is.

Szabvány szerint 25 vezeték köti össze a gép soros aszinkron vonali adapterét a modemmel. A következő táblázatban láthatjuk az egyes vezetékek elnevezését és szerepét. A kommunikáló két oldal maga az aszinkron adapter és egy "külső eszköz", általában modem. Az egyes vonalakon nyíl jelzi az adatforgalom irányát, azt, hogy az adott vonalon ki kinek küldhet üzenetet.

Az alábbi (eléggé szövevényes) táblázaton szemléltetjük az adapter és a modem közötti felület áramköreit. Az adaptert és a külső eszközt jelentő téglalapokon belül levő számok az egyes vezetékek itteni számozását mutatják, míg a vonalakon baloldalt látható százas nagyságrendű értékek a V.24 szabvány szerinti áramköri számozást. A vonalak nevei az adapter leírásából származnak, és nem mindenütt felelnek meg az eredeti V.24-es neveknek. Az eredeti neveket (ahol szükséges) zárójelben adtuk meg.

Sorszám		Vezeték neve	KSorszám	
	1	101 ----- NC (Protective Ground) -----	1	
	2	103 -------- Transmitted Data -------->	2	
	3	104 <---------- Received Data ----------	3	
	4	105 ----- Request to Send, RTS ----->	4	
	5	105 <------ Clear to Send, CTS ------	5	
	6	107 <------ Data Set Ready, DSR ------	6	Kül-
	7	102 ---------- Signal Ground ----------	7	
Asyn.	8	109 <----- Rec.L.S.Det.(carrier) -----	8	
	9	--- Transmit Current Loop Return --->	9	
	10	---------- NC ----------	10	ső
	11	(126) --- Transmit Cur. Loop Data --->	11	
Comm.	12	---------- NC ----------	12	
	13	---------- NC ----------	13	
	14	---------- NC ----------	14	ké-
	15	(114) -------- NC (Trans Clock) --------	15	
Adap-	16	---------- NC ----------	16	
	17	(115) -------- NC (Rec. Clock) --------	17	
	18	<----- Receive Current Loop Data -----	18	szü-
	19	---------- NC ----------	19	
ter	20	108 ----Data Terminal Ready, DTR ----->	20	
	21	---------- NC ----------	21	
	22	125 <------- Ring Indicator, RI ------	22	lék
	23	---------- NC ----------	23	
	24	---------- NC ----------	24	
	25	<----- Receive Current Loop Return ----	25	

- NC, Not Connected ..... Nincs összeköttetés
- Transmitted Data ..... Adatküldési vonal
  Az adapter ezen a vonalon írja ki a kiküldendő adat bitjeit. A kivitel a startbit után a Ø. sorszámú, legalacsonyabb helyiértékű bittel kezdődik. Az adatküldéshez általában megkívánják az RTS, CTS, DSR és DTR áramkörök bekapcsolt állapotát. Ez a gép-modem kapcsolat felépítésekor megy végbe.
- Received Data ..... Adatfogadási vonal
  Az adapter ezen a vonalon kapja a modem vagy külső eszköz által küldött adat bitjeit. A beolvasás során a startbit után érkező első bit lesz a vett adat Ø., legalacsonyabb sorszámú bitje. A beolvasott adat a Receive Buffer Registerbe kerül, ahonnan a program azonnal kiolvashatja.
- Request to Send, RTS ..... Adáskérés a modemtől
  Az adapter ezen a vonalon jelzi a modemnek vagy külső eszköznek, hogy kész adatok fogadására. Az erre adandó válasz a CTS biten érkezik a külső eszköztől. Vezérlését a Modem Control Register hasonnevű bitjének állításával végezhetjük.
- Clear To Send, CTS ..... A modem adási készségének jelzése
  Az adapter ezen a vonalon át észleli, hogy a külső eszköz kész adatok küldésére. Ezt kell a modemnek felelnie az adaptertől kapott RTS jelre. A CTS vonal lekérdezését a Modem Status Register hasonnevű bitjének vizsgálatával végezhetjük.
- Data Set Ready, DSR ..... A külső eszköz üzemkész állapota
  Az adapter ezen a vonalon át észleli, hogy a külső eszköz üzemkész (a DTR jelzésre adandó válasz). A DSR vonal lekérdezését a Modem Status Register hasonnevű bitjének vizsgálatával végezhetjük.
- Signal Ground ..... Az interface földelése
  Ez a földelés minden kapcsolatteremtő vonal "másik" ága.
- Received Line Signal Detector .... Különleges esemény jelzése
  Eredeti elnevezés: carrier
  Az adapter ezen a vonalon át észleli, hogy a külső eszköz valamilyen üzemzavarról akarja értesíteni. A V.24 szabvány ezen áramköre egyszerűen a kapcsolat meglétét jelenti; ha jelszintje normális, akkor a modem üzemkész, ha nem, az komoly hibát jelez. Az RLSD lekérdezését a Modem Control Register 7. bitjének vizsgálatával végezhetjük.
- Transmit Current Loop Data ..... Áramhurkolt output kimenete
- Transmit Current Loop Return .... Áramhurkolt output kimenete
- Receive Current Loop Data ..... Áramhurkolt output bemenete
- Receive Current Loop Return ..... Áramhurkolt output bemenete
  E vonalak bizonyos IBM nyomtatók bekapcsolására és tesztelésre szolgálnak; távoli, modem nélküli átvitelt tesznek lehetővé.

- Data Terminal Ready, DTR ..... Az adapter üzemkészségét jelzi
  Az adapter ezen a vonalon jelzi a külső eszköznek, hogy üzemkész. A külső eszköz (modem) erre a DSR jellel válaszol.
- Ring Indicator, RI ..... A modem a másik oldali hívást jelzi
  A Ring Indicator jelet a modem arra használja, hogy a másik modemtől (és így a másik géptől) érkező hívásokat jelezze az aszinkron vonali interface-nek.

A kapcsolatfelvétel menete durván a következő: az adaptert vezérlő program kiküldi a DTR-t a modemnek, és megvárja a külső eszköz válaszát, a DSR-t. Ezután ki kell adnia egy RTS-et, és megvárnia a CTS választ. Ekkor van meg az összeköttetés.

Mindezek a jelek komplementáltak, azaz aktív állapotukat nem a logikai Ø, hanem a logikai 1 szint jelzi. Minthogy azonban a programozás során is komplementált jelekkel dolgozunk, éppen 1-be kell állítani például a DTR jelet ahhoz, hogy az adapter aktív állapotát jelezzük a külső eszköznek. Programozóknak kevésbé fontos, bár nem haszontalan ismeret, hogy a vonalak 1 szintjét a -3V-nál kisebb, a Ø szintet a +3V-nál nagyobb feszültség jelenti. A legnagyobb áthidalható távolság kb. 15-20 méter.

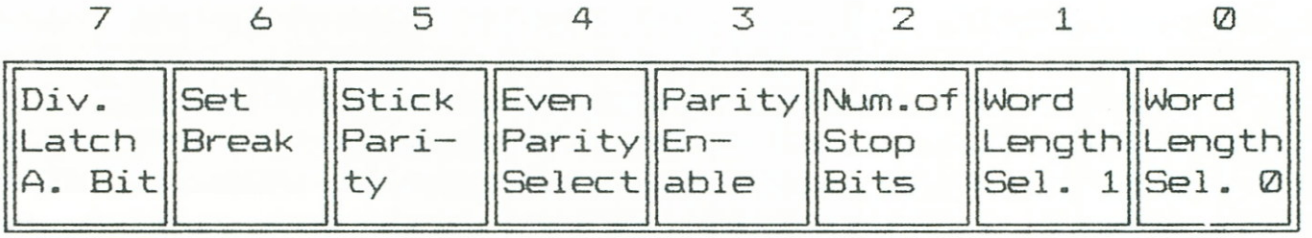
VI.3. Az adapter regiszterei

A következőkben ismerkedjünk meg az INS 8250 áramkör belső regisztereivel, szerepükkel és elérésük módjával. Mint a bevezetésben, valamint az "I/O portok" c. függelékben olvasható, az INS 8250 regiszterei I/O portok írásával és olvasásával érhetők el. Az áramkör regisztereihez a 3F8H és 3FEH közötti I/O portokat kötötték. E portok közül az első kettő több regiszter elérésére is használatos. Az alábbi felsorolásban (ahol szükséges, bitről bitre részletezve) megadjuk a programozó számára fontos regisztereket. Minden regiszternél szerepel az I/O portcím is, mely az elsődleges adapterre vonatkozik; értéke 3F8H-tól 3FEH-ig terjedhet. A másodlagos adapter regiszterei pontosan ugyanezek, csak a báziscím különböző: itt 2F8H. Az offsetek segítik a másodlagos adapter regisztereinek azonosítását.

VI.3.1. Az adapter előkészítése

(1) Line Control Register - írható-olvasható, címe 3FBH, illetve 2FBH, offsetje 3.

E regiszter segítségével határozhatjuk meg a vonal kezelésének módját. Bitkiosztása (a következő oldalon):



A Line Control Register bitkiosztása

- WLSØ és WLS1 - az alsó két bit szabja meg a kiküldendő karakterek hosszúságát bitekben az alábbi módon:
  ØØ 5 bit (ROM BIOS nem valósítja meg)
  Ø1 6 bit (ROM BIOS nem valósítja meg)
  1Ø 7 bit
  11 8 bit
- Number of Stop Bits - a kiküldendő, illetve várt stop bitek számát adja meg:
  Ø 1 stop bit minden karakter után
  1 2 stop bit (másfél, ha a karakter hossza 5 bit)
- Parity Enable - a paritásellenőrzést szabályozza:
  Ø nincs paritásellenőrzés
  1 ha a Stick Parity 1, akkor rögzített értékű paritásbitet küld, illetve vár;
  ha a Stick Parity Ø, akkor az "Even Parity Select" által meghatározott paritásellenőrzés
- Even Parity Select - a paritásellenőrzés módját szabja meg:
  Ø páratlan paritás, azaz akkor küld, illetve vár egyes értéket paritásként, ha a karakterben szereplő egyesek száma páratlan;
  1 páros paritás, azaz akkor küld, illetve vár egyes értéket paritásként, ha a karakterben szereplő egyesek száma páros.
- Stick Parity - előre megszabott paritásbit:
  Ø szabályos paritásellenőrzés, mint ahogy fent leírtuk a "Parity Enable" és az "Even Parity Select" biteknél;
  1 ha a "Parity Enable" bit 1, akkor az "Even Parity Select" által meghatározott, rögzített érték kiküldése, illetve tesztelése a paritásbit helyén:
  - ha az "Even Parity Select" értéke Ø, akkor a paritásbit mindig 1;
  - ha az "Even Parity Select" értéke 1, akkor a paritásbit mindig Ø.
- Set Break - ha itt 1-et adunk meg, az adapter mindig logikai Ø szintet kapcsol az interface soros kimenetére akkor, ha éppen nincs kiküldendő karakter. Terminálok soros vonalon át való meghajtása esetén használatos.

- Divisor Latch Address Bit - ez a bit választja ki a 3F8H és 3F9H című (Ø és 1 offsetű) regiszterek szerepét. Ha ez a bit (a továbbiakban DLAB) Ø értékű, akkor a Ø és 1 offsetű regiszterek kiküldő/beolvasó, illetve az interruptokat vezérlő regiszterek, ellenkező esetben a bitsebesség, a Baud Rate meghatározására szolgáló adatregiszterek.

(2) Modem Control Register - írható/olvasható, címe 3FCH, illetve 2FCH, offsetje 4.

A modemmel való kommunikáció vezérlésére szolgáló regiszter. Az ide kiírt parancsokkal módosíthatjuk a modem felé a vezérlőjellegű vonalakon (DTR, RTS, OUT1, OUT2) kiadandó jelek értékét.

7	6	5	4	3	2	1	Ø
Ø	Ø	Ø	LOOP; teszt-mód	OUT2	OUT1	Req. to Send	Data Term. Ready

A Modem Control Register bittiosztása

- A Data Terminal Ready és Request to Send bitek közvetlenül vezérlik az interface megfelelő vonalait. Az OUT1 és OUT2 bitek tesztelésre szolgáló felhasználó-definiálta átvitel megvalósítását teszik lehetővé; bővebben lásd alább.
- A LOOP bit 1-be állítása tesztmódba lépteti a 8250-et. Ekkor a következők történnek:
  - a Transmitter Shift Register outputja a Receive Buffer Register inputján érkezik vissza;
  - a négy státuszvonal (CTS, DSR, RLSD és RI) lekapcsolódik, látszólagos vezérlésüket közvetlenül programból végezhetjük a Modem Control Register alsó 4 bitjének módosításával (a DTR bittel a CTS, az RTS-el a DSR, az OUT1-el az RLSD, és az OUT2-vel az RI állását). Ha interrupt adására programoztuk fel az adaptert, akkor e bitek megváltoztatása pontosan úgy kiváltja az adapter interruptját, mintha valódi modem adta volna a megfelelő jeleket.

(3) Divisor Latch Register, Low & High - csak írhatók, címük 3F8H és 3F9H, illetve 2F8H és 2F9H (a Line Control Register DLA bitjének 1 állása mellett), offsetjük Ø, illetve 1.

A Baud Rate meghatározásakor egy osztót kell megadni. Ez szabja meg, hogy a 8250 belső órajelének (frekvenciája 1,8432 Mhz)

hányszorosa legyen a bitsebesség. A táblázat mutatja a különféle bitsbességekhez tartozó osztókat (ezeket kell kiírni a Divisor Latch Register-ekbe, előbb az alsó, majd a felső címre).

Baud Rate	Osztó (decimális)	Osztó (hexadec.)	Hiba százalékban
50	2304	0900	0.0
75	1536	0600	0.0
110	1047	0417	0.026
134	857	0359	0.058
150	768	0300	0.0
300	384	0180	0.0
600	192	00C0	0.0
1200	96	0060	0.0
1800	64	0040	0.0
2000	58	003A	0.069
2400	48	0030	0.0
3600	32	0020	0.0
4800	24	0018	0.0
7200	16	0010	0.0
9600	12	000C	0.0

Az RS232 sebesség-konstansai

Túl tudjuk lépni a 9600 bit másodpercenkénti sebességet, hiszen az ehhez tartozó osztó még elég nagy (a legnagyobb sebesség 52 Kbit/sec körül van); a ROM BIOS csak 9600-at tesz lehetővé.

VI.3.2. Az adapter hardware interruptjai

Soros vonal kezelése csak hardware-interrupt segítségével oldható meg hatékonyan. Csak így kerülhető el, hogy a vonalat vezérlő program állandóan az adapter státuszregisztereinek olvasásával legyen kénytelen foglalkozni, nem adva lehetőséget más programtevékenységek folytatására. Ha a soros vonal interruptos kezelésére vállalkozunk, a következő lépéseket kell megtennünk:

- az adaptert kívánságaink szerint felprogramozni úgy, hogy beadja a kívánt interruptokat (ez az Interrupt Enable Register segítségével lehetséges);
- a processzor megfelelő interrupt vektorában (az elsődleges adapter esetén a 0CH vektorban [IRQ4], a másodlagos adapter esetén pedig a 0BH vektorban [IRQ3]) beállítani a hardware-interruptokat kezelő rutin belépési címét;

- a 8259 típusjelű interrupt vezérlőt átprogramozni úgy, hogy "beengedje" az adapter interruptjait.

Az interruptnak (bár a processzor számára mindig azonos interrupt-szinten következik be) négy, különböző prioritású oka lehet. Az első és legmagasabb prioritású szinthez a fogadó oldal státuszában bekövetkező változások tartoznak. A második szintet karakter vétele okozza. Az interrupt harmadik szintjét a Transmitter Holding Register kiürülése (vagyis a megelőző karakter kiküldésének befejezése) váltja ki. Végül a legalacsonyabb szintű interrupt akkor következhet be, ha a modem státuszában állt be valami változás. Ezt és az interrupt okának "dekódolására", azonosítására szolgáló "Interrupt Identification Register" három bitjének értelmezését mutatja a táblázat:

Int. Ident. Reg. bitjei	Az interrupt prioritása	Az interrupt feltételt kiváltó és az azt megszüntető események és tevékenységek		
		IT típus	Kiváltó ok	Megszüntetése
001	--	--	nincs interrupt	--
110	0. legmagasabb	Fogadó oldali státuszban beállott változás	Overrun Error Framing Error Parity Error Break Interrupt	A Line Status Register kiolvasása
100	1.	Karakter vétele	A Receive Registerben karakter van	A Receive Buffer Register kiolvasása
010	2.	Karakter kiküldésének befejezése	A Transmitter Holding Register kiürült	Az Interrupt Id. Register kiolv, vagy új karakter kiírás
000	3.	Modem státuszában beállott változás	CTS, DSR, RING Indicator vált. vagy RSLD érkezett	A Modem Status Register kiolvasása

Az RS232 interrupt-regisztere

A fentiekből következik egyfelől az Interrupt Identification Register használata, bitjeinek értelmezése, másfelől pedig az a

tény, hogy "kvázi-interruptosan" is kezelhetjük a vonalat. Ez pedig a következőképpen lehetséges: időről időre (lehetőség szerint sűrűn) beolvassuk az Interrupt Identification Register tartalmát, és megvizsgáljuk a legalsó bitet. Ha értéke 1, nincs semmi olyan ok, amely interruptot váltana ki akkor, ha az interruptok kiváltására programoztuk volna fel az adaptert. Ha ez a bit 0, akkor az 1. és 2. bit elemzése megmutatja, hogy az interrupt milyen okból következne be.

Most pedig lássuk magukat a regisztereket:

(1) Interrupt Enable Register - csak írható, címe 3F9H, illetve 2F9h (a Line Control Register DLA bitjének 0 állása mellett), offsetje 1. Bitkiosztása a következő:

7	6	5	4	3	2	1	0
0	0	0	0	Enable Modem St. IT	Enable R.Line St. IT	Enable Tr.H. R. IT	Enable Rec.ch IT

- a 0. bit (Enable Reveive Character Interrupt) szabja meg, hogy az 1. prioritási szintű interrupt bejöhet-e vagy sem;
- az 1. bit (Enable Transmit Holding Register Empty Interrupt) a kiírás befejezésekor adandó interruptot,
- a 2. bit (Enable Receive Line Status Interrupt) a vételi oldal státuszának változásakor bekövetkező interruptot, végül
- a 3. bit (Enable Modem Status Interrupt) a modem státuszának változásakor beadandó interruptot vezérli (látható, hogy a bitek kiosztása nincs összhangban a prioritási szintekkel).

Felmerül egy jogos kérdés: miért kell elválasztani ezeket a szinteket? Miért nem úgy programozhatjuk az adaptert, hogy vagy adjon be minden interruptot, vagy ne adjon be egyet sem? A válasz az, hogy a soros vonallal nemcsak modemet (és annak segítségével "feleselni" is képes másik gépet) kezelhetünk, hanem olyan hardware-eszközöket (pl. rajzgépet vagy egeret), amely egyáltalán nem hasonlít egy modemhez. Valamilyen vonali zavar folytán persze itt is bekövetkezhet például egy látszólagos Modem Status Interrupt, de minek kínlódjunk ezzel az esettel, ha nem muszáj? Egyszerűen letiltjuk ezt a szintet, és biztonságban vagyunk; a kábelhiba nem fog állandó interrupttal gyötörni bennünket.

(2) Interrupt Identification Register - csak olvasható, címe 3FAH, illetve 2FAH, offsetje 2.

E regiszternek mindössze 3 alsó bitje használt; ezek jelentését lásd a fenti ábrán. Figyeljük meg, hogy (mivel az interrupt-állapotot a legalsó bit Ø értéke jelzi) a regiszter címindexként használható!

VI.3.3. Input-output az adapter segítségével

(1) Transmitter Holding Register - csak írható, címe 3F8H, illetve 2F8H (a Line Control Register DLA bitjének Ø állása mellett), offsetje Ø.

Az elküldendő karaktert ide kell beírnunk. Ha a Transmitter Shift Register kiürül, azaz az áramkör megkezdheti egy újabb karakter elküldését, a kiírandó karakter átkerül a Transmitter Holdig Registerből a Transmitter Shift Registerbe, és megkezdődik bitenkénti kiküldése a vonalra.

Ha a Transmitter Holding Register kiürül (azaz ide be lehet írni az újabb karaktert), az áramkör interruptot generál. A regiszter állapotát le lehet kérdezni a Line Status Register 5. bitjének tesztelésével.

(2) Transmitter Shift Register - belső regiszter, közvetlen írása vagy olvasása nem lehetséges.

Az elküldendő karakter a Transmitter Holding Registerből ebbe a regiszterbe kerül, ahonnan az áramkör bitenként kilépteti a vonalra (elsőként a Ø., legalacsonyabb helyiértékű bitet). Amíg ez a folyamat be nem fejeződik, az áramkör nem veszi át a Transmitter Holding Registerben tárolt következő karaktert ebbe a regiszterbe. Amikor azonban a karakter kiküldése befejeződött, az áramkör automatikusan átveszi a Transmitter Holding Registerben tárolt karaktert a Transmitter Shift Registerbe, és megkezdi annak kiírását.

Ha a Transmitter Shift Register kiürül (azaz megkezdődhetne az újabb karakter kiküldése, de a Transmitter Holding Register üres), az áramkör interruptot generál. Ebben az esetben az áramkör két elküldendő karaktert is képes fogadni gyors egymásutánban. A regiszter állapotát le lehet kérdezni a Line Status Register 6. bitjének tesztelésével.

(3) Receiver Buffer Register - csak olvasható; címe 3F8H, illetve 2F8H (a Line Control Register DLA bitjének Ø állása mellett), offsetje Ø.

Ha az áramkör a vonalról karaktert fogadott, ebbe a regiszterbe helyezi el (az elsőként vett bit a Ø., legalacsonyabb helyiértékű). A kezelő programnak innen kell kiolvasnia, mégpedig előbb, mint hogy az áramkör újabb karaktert fogad. Az újabb karakter érkezése "Overrun Error"-t, túlfutási hibát okoz. A fogadás csakúgy, mint a túlfutás, interruptot vált ki. A regiszter állapotát a Line Status Register Ø. bitje jelzi.

VI.3.4. Az adapter státuszának vizsgálata

(1) Modem Status Register - csak olvasható, címe 3FEH, illetve 2FEH, offsetje 6.

A modem státuszát leíró információk tárolására szolgáló regiszter. Kiolvasásával az adapter és a vele összekapcsolt külső eszköz közötti interface állapotát, az ott bekövetkezett eseményeket vizsgálhatjuk.

7	6	5	4	3	2	1	Ø
Rec. Line Signal	Ring Indicator	Data Set Ready	Clear to Send	Delta Line Signal	Trail. Edge Ring I	Delta Data Set R.	Delta Clear to S.
		(DSR)	(CTS)			(DDSR)	(DCTS)

A Modem Status Register bitkiosztása

A felső négy bitet vö. az interface megfelelő vonalaival; az alsó négy bit azt jelzi, hogy a megfelelő vonalon történt-e valami változás a Modem Status Register utolsó kiolvasása óta.

(2) Line Status Register - csak olvasható; címe 3FDH, illetve 2FDH, offsetje 5.

Ez a regiszter a vonal állapotát tartja nyilván.

7	6	5	4	3	2	1	Ø
Ø	Trans. Sh. R. Empty	Trans. H.Reg. Empty	Break Interrupt	Framing Error	Parity Error	Overrun Error	Data Ready

A Line Status Register bitkiosztása

A Line Status Register alsó öt bitje a vételi, a következő kettő pedig az adási oldal státuszára vonatkozó információt tartalmazza.

a) A vételi oldal bitjei:

- Data Ready - e bit 1, ha vett karakter található a Receiver Buffer Registerben. A bitet a karakter kiolvasása automatikusan törli. Egy karakter érkezése interruptot generál.
- Overrun Error - túlfutási hiba. Nem olvastuk ki időben az korábban érkezett karaktert, és az adapter újabbat fogadott a Receive Buffer Registerben. Ez egy (vagy több) karakter elvesztését jelenti. Az Overrun Error interruptot okoz.
- Parity Error - a beolvasott karakter paritáshibával érkezett. A Parity Error interruptot okoz.
- Framing Error - nem "ép" karaktert vett az adapter, nem sikerült fognia a megállapodás szerinti számú stopbitet. A Framing Error interruptot okoz.

  Megjegyezzük, hogy a vétel során mindig elég egy stop bit; a hibajelzés azt mutatja, hogy egyet sem sikerült venni.

- Break Interrupt - azt jelzi, hogy az utoljára fogadott karakter óta egy teljes karakter átküldéséhez szükséges időnél tovább volt nulla a vonal szintje. Ezt az adatátviteli eljárásokban speciális jelzésre, például bontáskérésre használják fel. A Break Interrupt (mint neve is mutatja) interruptot generál.

b) A fogadási oldal bitjei:

- Transmitter Holding Register Emtpy - az előzőleg kiírt karakter kiküldése befejeződött, így a Transmitter Holding Registerbe utoljára beírt karakter átkerült a Transmitter Shift Registerbe, és megkezdődött a kiküldése. Az adapter képes újabb kiküldendő adat fogadására. A Transmitter Holding Register kiürülése interruptot generál.
- Transmitter Shift Register Emtpy - nemcsak hogy a Transmitter Holding Register üres, de már a Shift Register is, tehát az adapter már két teljes karaktert küldött ki a vonalra azóta, hogy a program újabb kiküldendő karaktert adott át neki.

VI.4. Az aszinkron vonal kezelése

Az aszinkron vonal kezelésére négyféle kényelmes (?) út kínálkozik - a gyakorlatban valószínűleg az ötödiket kell majd választanunk! Az első út, hogy MS-DOS parancsszintről perifériális eszközként kezeljük a vonalat. Ha kiadjuk a

```
C>copy con com1
```

parancsot, akkor a konzolról begépelt karakterek az 1. kommunikációs eszközön, ti. az elsődleges aszinkron vonalon át szépen kisorjáznak. A fogadó gép pedig az ellenkező irányú paranccsal fogadhatja üzenetünket. A vonal paramétereinek egyező beállításáról a MODE nevű MS-DOS paranccsal kell gondoskodnunk.

Első pillantásra látszik, hogy ez az út (bár nagyon egyszerű) nem igazán járható, mert a fogadó gépet nem tudjuk értesíteni az üzenet érkezéséről; a fogadónak áhítatos lélekkel kell várnia az üzenetet adó gép kegyét. Arról ne is beszéljünk, hogy a vezeték hosszúsága is komoly szerepet játszik az ilyen átvitelben. Nagyobb távolságra gyakorlatilag csak valamilyen modem segítségével továbbíthatjuk a kódokat, mivel a vonal zaja, és a jel erősségének csökkenése lehetetlenné teszi az üzenet vételét, ha a vezeték hosszabb néhány tíz méternél. Ennél azonban sokkal nagyobb baj az, hogy a gyakorlatban csak 1200 bit másodpercenkénti sebességgel dolgozhatunk, és menet közben semmiféle lemezműveletet nem végezhetünk.

A második lehetőség: programból egyszerűen az erre szolgáló MS-DOS funkciók segítségével kezeljük a vonalat (lásd 2. kötet; a 03 és 04 funkciók, vagy az MS-DOS szintű file-kezelés megfelelő funkciói). Ez az út sem vezet sehová, akár az előző. Először is, a vonal felprogramozása nem lesz egyszerű. Másodszor, nem szabadultunk meg attól a nyűgtől, hogy a fogadó gépnek állandóan lesni kell az adó gép minden szavát, különben elveszt néhány tucat karaktert.

A harmadik a ROM BIOS felhasználása; de előre kell bocsátanunk: az aszinkron vonali driver alulmúlja a többi, eléggé minimális tudású ROM BIOS drivert is. Gyakorlati célokra egyszerűen használhatatlan. Most pedig végre mondjuk ki az alapvető okot: minden egyes karakter érkezését meg kell várnunk, az eddig felvázolt három út bármelyikét kövessük is. Azonban ez lenne a kisebbik baj. A nagyobb az, hogy nem elég idejekorán átadni a vezérlést a ROM BIOS drivernek (persze az MS-DOS funkciói is ezt használják). Mivel ez nem az idők végeztéig, hanem egy bizonyos ideig vár a karakter megérkezésére, lehet, hogy kissé korán érkezünk: mikorra a várt karakter megjön, a várakozási idő lejárt. Ekkor persze a kommunikációt vezérlő program egy kicsit meditál azon, hogy miért is kapta vissza a vezérlést anélkül, hogy megérkezett volna a karakter; ezalatt pedig, ha a másik oldal hirtelen elküld egy újabb karaktert, akkor ez máris felülírja az előbb hiába várt, kissé megkésett társát – és az üzenetet már csak sérülten vehetjük.

Az első három lehetőség tehát azért nem jöhet szóba, mert "szinkron" módon kezelik a vonalat, azaz ha olvasni akarunk egy karaktert a vonalról, addig kell várnunk, míg meg nem érkezik, holott egy ilyen vonal éppen "aszinkronban" kezelendő; tehát programunk végzi szokásos tevékenységét, és a vonalról kapott karakter(ek)re már csak akkor reagál, ha összeállt számára egy értelmes üzenet.

> Megjegyzés: mint látható, itt a "szinkron" és "aszinkron" kifejezéseket egészen más értelemben használjuk: nem bitszinkront értünk alatta, hanem egy programozástechnikai szinkront vagy szinkron nélküliséget: a kérdés az, hogy a program szinkronban működik az I/O tevékenységgel, vagy sem.

A negyedik út: saját kezünkbe vesszük a vonal teljes kezelését. Ennek fő nehézsége az, hogy hardware interruptot kell majd kezelnünk. Ha ezt a lehetőséget mérlegeljük, akkor vegyük figyelembe, hogy a hardware interrupt lekezelésénél alig van nehezebb programozási feladat, hiszen az algoritmikus elven működő programnak olyan folyamatot kell szinkronizálni, amely nincs a kezében, és sok esetben egyáltalán nem algoritmikus, hiszen a másik oldal azt csinál, amit akar, nem beszélve a kiszámíthatatlan vonali hibákról.

Végül az ötödik: sutba dobjuk az egész aszinkron vonalat, és beszerzünk valami lokális hálózati hardware-t és az azt működtető programokat. Ez az út persze drága, és nem is biztos, hogy nagyon jó eredményeket kapunk, de legalább nem bennünket fognak szidni - és az is valami.

VI.5. Vonalkezelési funkciók

Interrupt-sorszám: 14H
Javasolt interrupt-név: BS_RS232

Az aszinkron vonal kommunikációs interruptjának hívására javasolt makró:

```
ASNCLL  MACRO   CODE, NUMBER                ;
                MOV     AH, CODE            ;;
        IFNB    <NUMBER>                    ;;
                MOV     DX, NUMBER          ;;
        ELSE                                ;;
```

```
                MOV     DX, 0           ;;Elsődleges adapter
        ENDIF                           ;;
                INT     BS_RS232        ;;
        ENDM                            ;
```

A BIOS funkciói nem oldják meg kielégítően az aszinkron vonal kezelését. Mégis jól használhatjuk őket, hacsak nem akarjuk elérni, hogy gépünk a "háttérben" kezelje a vonalat, s mi valamilyen másik tevékenységet is végezhessünk. Kis ügyességgel vezérelhetünk valamilyen, az aszinkron vonalon elérhető perifériális eszközt, például egy rajzgépet vagy egy második nyomtatót.

VI.5.1. A vonal előkészítése

Funkciókód: AH = 00H INT 14H
Javasolt konstansnév: ASN_INI
Be:

AL paraméter, melynek bitjei a következők:
0-1 karakterhossz kódja;
00, 01 nem használt (nem adja tovább)
10 - 7 bites karakter, 11 - 8 bites karakter
2 stopbit kód
0 egy stopbit
1 két stopbit,
azaz az adás során a karakter végén legalább egy vagy legalább két egyes értékű bitet küld az adó.
3-4 paritásellenőrzés módja
00,10 nincs ellenőrzés
01 - páratlan paritás, 11 - páros paritás
Azaz: ha a 3. bit 0, nincs paritásellenőrzés, ha 1, akkor van. Ha van paritásellenőrzés, az lehet "páros" vagy "páratlan", azaz páros vagy páratlan számú 1 bit esetén küldünk, és várunk 1-est mint paritásbitet. Ha a 4. bit 0, páratlan, ha 1, akkor páros paritásellenőrzés lesz.
5-7 bitsebesség

000 - 110 bit/sec,	100 - 1200 bit/sec
001 - 150 bit/sec,	101 - 2400 bit/sec
010 - 300 bit/sec,	110 - 4800 bit/sec
011 - 600 bit/sec,	111 - 9600 bit/sec

DX a kiválasztott vonal sorszáma

Válasz:

Semmi.

VI.5.2. Karakter elküldése

Funkciókód: AH = 01H INT 14H
Javasolt konstansnév: ASN_SDCHAR
Be:

AL az elküldendő karakter
DX a kiválasztott vonal sorszáma (0 - 1).

Válasz:

AH hibakód. Ha AH nulla, minden rendben; ha a 7. bitje 1, akkor a további bitek informálnak a hiba természetéről (lásd 03H funkció).

Megjegyzés: hiba esetén hívjuk meg az ASN_GTSTATE funkciót, mert az AH 7. bitje maga is egy hibajelző bit!

VI.5.3. Karakter beolvasása

Funkciókód: AH = 02H INT 14H
Javasolt konstansnév: ASN_RCCHAR
Be:

DX a kiválasztott vonal sorszáma (0 - 1).

A BIOS addig vár, míg karakter nem érkezik az aszinkron vonalon, vagy egyéb esemény nem zárja le a műveletet (pl. time out, a várakozási idő lejárása).

Válasz:

AL a vett karakter kódja
AH státusz, mint a 01 funkciónál.

VI.5.4. A vonal státuszának lekérdezése

Funkciókód: AH = 03H INT 14H
Javasolt konstansnév: ASN_GTSTATE
Be:

DX A használni kívánt adapter sorszáma (0 - 1).

Válasz:

AX a vonal és a modem státusza:

AH bitjei (a vonal státusza):
- 0. bit: vett adat van a fogadó regiszterben;
- 1. bit: Overrun - túlfutás, adatvesztés, nem olvastuk ki a vett adatot, és újabb érkezett;
- 2. bit: Parity Error - paritáshiba következett be;

- 3. bit: Framing Error - szabálytalan karakter (nem megfelelő stopbitek);
- 4. bit: Break-Detect jelzés - több, mint egy karakter átküldésének ideje óta nem érkezett karakter;
- 5. bit: Transmitter Holding Register Empty - újabb karaktert lehet átadni;
  Megjegyzés: e regiszterből kerül a karakter a shift-regiszterbe, ahonnan aztán bitenként kiíródik a vonalra.
- 6. bit: Transmitter Shift Register Empty - minden karakter kiment (tehát nemcsak hogy a várakozási regiszterbe írhatunk új adatot, de a shiftregiszter is üres; a várakozási regiszterbe írt adat kiküldése azonnal megkezdődhet);
- 7. bit: Time-Out Error - lejárt a várakozási idő.

AL bitjei (a modem státusza):
- 0. bit: Delta Clear-To-Send - a CTS értéke megváltozott;
- 1. bit: Delta Data-Set-Ready - a DSR értéke megváltozott;
- 2. bit: Trailing-edge Ring Indicator - a Ring Indicator áramkörön felment a jelszint;
- 3. bit: Delta Receive Line Detect - a fogadási vonalon változás következett be;
- 4. bit: Clear To Send értéke;
- 5. bit: Data Set Ready értéke;
- 6. bit: Ring Indicator értéke;
- 7. bit: Received Line Signal Detect értéke.

Megjegyezzük, hogy az AL regiszter megegyezik a Modem Status Register tartalmával.

VI.6. Aszinkron vonal kézi használata

Az alábbi rutinok közül három az aszinkron vonal előkészítését, karakter szinkron kiírását, és karakter szinkron beolvasását szemlélteti. A további kettő hardware interrupttal kapcsolatos. Az egyik előkészíti az aszinkron vonalat, hogy interrupttal jelezze egy karakter vételét, a másik pedig elvégzi az interrupt bekövetkezésekor szükséges tevékenységeket.

Lássuk először a programozástechnikailag szinkron és interrupt-mentes vonalkezelést!

Az INI_LINE rutin bemenetként egy, a baud rate-ra jellemző konstanst vár a BX regiszterben. Ez a konstans azt mutatja meg, mennyivel kell osztani az órajelet a kívánt baud rate eléréséhez. Figyeljük meg, hogy a konstans alsó fele a 3F8H, a felső

pedig a 3F9H portra kerül ki; a kiírást szavas I/O utasítással is el lehetne végezni. E kis rutin kilépése előtt a kommunikáció módját is megszabja a vezérlő portra való kiírással.

```
;
; Konstansok
;-----------------------------------------------;
ASN_BAS          EQU     03F8H            ;ACE bázisport
ASN_THR          EQU       00H            ;Küldő adatregiszter
ASN_RBR          EQU       01H            ;Fogadó adatregiszter
ASN_IEN          EQU       01H            ;Interrupt regiszter
ASN_INT          EQU       02H            ;IT azonosító reg.
ASN_LCR          EQU       03H            ;Vonalvezérlő regiszter
ASN_MCR          EQU       04H            ;Modemvezérlő regiszter
ASN_LSR          EQU       05H            ;Vonalstátusz regiszter
        ASN_LSR_CHRDY   EQU      01H      ;Karakter érkezett
        ASN_LSR_THREMP  EQU      20H      ;Karakter küldhető
ASN_MSR          EQU       06H            ;Modemstátusz regiszter
BS_RS232         EQU     0CH              ;Interrupt száma
;-----------------------------------------------;
; INI_LINE - aszinkron vonal előkészítése
;       Input:
;                BX      - osztó a baud rate-hez
;-----------------------------------------------;
INI_LINE         PROC    FAR              ;
                                          ;
                 MOV     DX, ASN_BAS + ASN_LCR
                 MOV     AL, 80H          ;Baud rate következik
                 OUT     DX, AL           ;
                 MOV     DX, ASN_BAS      ;A baud rate osztó-
                 MOV     BL, AL           ; jának kiírása
                 OUT     DX, AL           ;Osztó alsó fele ki
                 INC     DX               ;
                 MOV     AL, BH           ;
                 OUT     DX, AL           ;Osztó felső fele ki
                 MOV     DX, ASN_BAS + ASN_LCR
                 MOV     AL, 03H          ;Nincs paritás, 8 bit
                 OUT     DX, AL           ;a karakterhossz
                 RET                      ;
INI_LINE         ENDP                     ;
;-----------------------------------------------;
```

A következő rutin egy karakter kiküldését végzi el. A kiírandó karaktert az AL regiszterben kapja meg. A kiírás előtt természetesen meg kell nézni, hogy képes-e az ACE a kiküldendő

karakter fogadására, azaz nem folgalt-e a küldő regiszter. Ehhez beolvassuk a vonal státuszát, amelynek 5. bitje mutatja a küldő regiszter állapotát. Ha foglalt (azaz e bit 1), akkor várni kell, míg az ACE befejezi az előzőleg elküldött karakter postázását (emiatt foglalt a regiszter). Ha a regiszter szabad, akkor kiküldjük az adatot - ezzel be is fejeztük a rutint.

```
; SEND_CHAR - karakter kiküldése az aszinkron vonalra
;       Input:
;               AL      - kiküldendő karakter
;-----------------------------------------------;
SEND_CHAR       PROC    FAR                     ;
                PUSH    AX                      ;-------;
                MOV     DX, ASN_BAS + ASN_LSR   ;Vonalstátusz
SEND_WAIT:                                      ;-------;
                IN      AL, DX                  ;Státusz beolvasása
                TEST    AL, ASN_LSR_THEMP       ;Küldhetünk?
                JZ      SEND_WAIT               ;Nem, várunk
                MOV     DX, ASN_BAS             ;Igen, adatport
                POP     AX                      ;Karakter vissza
                OUT     DX, AL                  ;Karakter kiküldve
                RET                             ;
SEND_CHAR       ENDP                            ;
;-----------------------------------------------;
```

A harmadik rutin az előző párja. Csak abban különbözik, hogy más státuszbitet vizsgál, és beolvas kiírás helyett.

```
;-----------------------------------------------;
REC_CHAR        PROC    FAR                     ;-------;
                MOV     DX, ASN_BAS + ASN_LSR   ;Vonalstátusz
REC_WAIT:                                       ;-------;
                IN      AL, DX                  ;Státusz beolvasása
                TEST    AL, ASN_LSR_CHRDY       ;Van karakter?
                JZ      ASYN_RWT                ;Nincs, várunk rá
                MOV     DX, ASN_BAS             ;Adatport címe
                IN      AL, DX                  ;Karakter beolvasása
                RET                             ;
REC_CHAR        ENDP                            ;
;-----------------------------------------------;
```

A következő rutin az aszinkron vonal interruptos kezelését készíti elő. Egyben szemlélteti a 25H és 35H DOS funkciók használatát is. Első dolga, hogy lekérdezi az aszinkron vonal hardware interrupt vektorának tartalmát, s ezt elmenti. Ezután ki-

írja az interrupt vektorba a handler rutin címét, majd előkészíti az ACE-t és az interrupt vezérlőt az interrupt élezésére, és előkészíti a modemet is, majd visszatér.

```
; INI_LINE_INT - előkészíti az ACE-t interrupt adására, az
;                interrupt vezérlőt pedig annak fogadására
;       Input:   semmi
;-----------------------------------------;
INT_BAS          EQU     20H              ;ITC bázisportja
                                          ;
INI_LINE_INT     PROC    FAR              ;
                 PUSH    DS               ;
                 MOV     AX, CS           ;
                 MOV     DS, AX           ;DS előkészítése
        ASSUME   DS:CODE                  ;Offsetek CODE-ból
                 PUSH    DS               ;Biztos ami biztos
                 S_ITV   BS_RS232, REC_CHAR_PNT, INT_SAVE
                                          ;IT vektor ment/töltés
                 POP     DS               ;DS visszaállítás
                                          ;--------;
                 MOV     DX, ASN_BAS+ASN_IEN      ;Interrupt port
                 MOV     AL, 04H          ;--------;Fogadási in-
                 OUT     DX, AL           ;terrupt engedélyezése
                                          ;
                 IN      AL, INT_BAS+1    ;Interrupt maszk reg.
                 AND     AL, 0F7H         ;3. bit törlése
                 OUT     INT_BAS+1, AL    ;Fogadás engedélyezése
                                          ;--------;
                 MOV     DX, ASN_BAS+ASN_MCR      ;Modemvezerlő
                 MOV     AL, 08H          ; OUT2 bitjének
                 OUT     DX, AL           ; bekapcsolása
                                          ;
                 POP     DS               ;
                 RET                      ;
                                  ;--------;
INT_SAVE         DD      0       ;Interrupt vektor mentésére
REC_CHAR_PNT     DD      REC_CHAR_INT
INI_LINE_INT     ENDP
```

A következőkben egy végletesen egyszerű modell segítségével illusztráljuk az aszinkron vonal interruptos kezelését. A feladat a következő:

Két azonos gép (IBM PC/XT) áll kapcsolatban. A gépek (pontosabban a rajtuk futó programok) üzeneteket váltanak egymás között. A programok számos egyéb feladatuk mellett időnként "in-

formáljáK" a másik programot (azaz elküldenek egy üzenetet), és az érkező üzeneteket, mikor azok teljesen megérkeztek, feldolgozzák.

Az egyszerűség kedvéért feltesszük, hogy a vonal garantáltan hibamentes - minden elküldött karakter sértetlenül fog megérkezni a másik oldalra. Feltesszük továbbá, hogy semmiféle sebességi, szinkronizálási probléma nem merül fel, és a gépek (a programok) elegendően gyorsak az üzenetek továbbítására, vételére és rendes tevékenységük folytatására.

> Megjegyzés: a távadatfeldolgozás legfőbb örömeit ezek a problémák jelentik. Mi most a legegyszerűbb esetet tárgyaljuk, annak is csak interrupt szintű megoldásával.

Egy üzenet legfeljebb 64 byte hosszú lehet. Tetszőleges karaktereket tartalmazhat, kivéve a Ø kódú karaktert. Ez a karakter lesz az "üzenet vége" jel, amit a továbbiakban EOM-nak (End Of Message) fogunk nevezni.

A feladatot megvalósító programokat két részre bontjuk. A felhasználói szintnek kizárólag az interrupt szinttel kommunikáló részét tervezzük meg, arra nem is gondolunk, hogy milyen természetű üzenetek jönnek-mennek, és hogyan kell őket feldolgozni. Nem térünk ki a küldésre sem, mert ennek problematikája lényegesen egyszerűbb (valódi feladat esetén persze nem). Munkánk első lépéseként az egymással kommunikáló szinteket tervezzük meg, majd áttekintjük a kommunikáció rendjét. Ha a tervek nem jók, akkor módosítjuk őket, míg csak hibátlanoknak nem látszanak. Ekkor fogunk csak a kódoláshoz (és valódi feladat esetén ekkor jönnek majd az igazi meglepetések).

Először az interrupt-szintet tervezzük meg. Mielőtt felvázolnánk a karakter-beolvasó rutint, gondoljuk át az interrupt folyamatát. Az alaposabb leírást az "8259 Interrupt Controller" fejezetben olvashatjuk. Az interruptot kérő egység értesíti szándékáról az interrupt vezérlőt, amely továbbítja a kérést a processzornak, ha ez lehetséges (ha nincs éppen aktív magasabb prioritású interrupt-kérés érvényben). Az interrupt fogadása után a processzor automatikusan "disable interrupt" módba lép, azaz semmilyen további interruptot nem fogad. Szokásos ilyen esetben egy STI utasítással engedélyezni újabb interruptok fogadását a **processzornak**. Az interrupt vezérlő azonban (tudván, hogy interrupt-eljárás van folyamatban) csak az éppen kiszolgálás alatt levő interruptnál magasabb prioritású interruptokat enged be. Ahhoz, hogy az interrupt-rendszert teljesen felszabadítsuk, egy erre a célra szolgáló parancs segítségével az **interrupt vezérlő** tudomására kell hozni az eljárás végét.

```
EOI             EQU     20H             ;Interrupt állapot vége

                ...
                MOV     AL, EOI
                OUT     INT_BAS, AL     ;IT-vezérlő bekapcs.
                ...
```

Ha ezután valamilyen okból újra le kell tiltanunk az interruptok érkezését, akkor elég a CLI utasítást kiadnunk. Ilyenkor azonban vigyáznunk kell; míg az EOI kiküldése előtt a vezérlő várakoztatta és sorbaállította az esetleges interrupt-kéréseket, most (mivel úgy tudja, hogy vége az előző interrupt lekezelésének) újabb és újabb interrupt kérésekkel bombázza az interruptok fogadására képtelen processzort! Emiatt nem is szokásos eljárás az EOI kiküldése az interrupt eljárás teljes befejezése előtt; viszont mindig be szoktuk engedni a magasabb prioritású megszakításokat.

A karakter-beolvasó rutin egy 128 karakteres, kódba épített bufferbe olvassa be a kapott üzenetet. A buffert ciklikusan kezeli: ha az írás során elérte a buffer végét, akkor automatikusan az elején folytatja az írást. Ebből következik, hogy a felhasználói szintnek időről időre ki kell olvasnia a buffer tartalmát, ha nem akar adatot veszteni. Figyelheti a buffer telítettségét is, ez azonban nem célszerű. A kernel-szint inkább egy ún. indikátorváltozóban jelzi majd, ha egy teljes üzenet megérkezett, azaz megjött az EOM is.

> Megjegyzés: az indikátorváltozó egy szavas vagy byte-os egész, melynek értéke 0 vagy 1 lehet. Célja valamilyen esemény bekövetkezésének jelzése (indikálása). Míg az esemény nem következik be, a változó értéke 0, ha bekövetkezik, akkor az ezt regisztráló program 1-re állítja a változót. Ez tehát egy flag, azaz jelzőbit.

A kernel-rutin tehát minden érkező karaktert betesz a bufferbe, és ha a karakter EOM volt, akkor beállítja az indikátorváltozót is. A beírás menete a következő: a kernel-szint egy írási mutatót használ fel, mely mindig a soron következő byteot címzi. Ezt felhasználva valamilyen indirekt címzéssel kiírja a karaktert, majd megnöveli a mutatót. Ha a növelés után a mutató túlcímzi a buffert, akkor az interrupt rutin visszaállítja őt a buffer elejére.

A user-szint előkészíti a vonalat (lefuttat valamilyen rutint, amivel megszabja a paramétereket, meghívja a fenti, az interrupt fogadását előkészítő rutint), majd "napi munkája" so-

rán rendszeresen megnézi az indikátort, és ha ez 1, akkor kiolvassa a buffer tartalmát. Mivel a kiolvasás memóriában zajlik, sokkal gyorsabb, mint a vonali adatvétel. Remélhető tehát, hogy a buffer sohasem fog végzetesen betelni, bár a kernel-szint a kiolvasás alatt már megkapja az újabb üzenet első karaktereit.

A kiolvasás menete a következő: a user-szint egy másik mutatót, az olvasási mutatót használja fel, amely mindig az első ki nem olvasott byte címét tartalmazza (az első üzenet olvasása előtt persze a buffer elejére címez). A user-szint minden karakter kiolvasása után megnöveli az olvasási mutatót. A buffer telítettségétől függetlenül csak az első EOM-ig olvas. Az üzenet kiolvasása után az indikátort törli. Ezután a mutató persze a következő üzenet első karakterére fog mutatni - innen kezdjük a következő olvasási eljárást. A buffer végének elérése után a mutatót automatikusan visszaállítjuk az elejére. (Feltettük, hogy egy üzenet nem hosszabb, mint 64 karakter, ezért a 128 karakteres buffer biztosan elég. Ha ugyanis az üzenet végének megérkezése után elég gyorsan kiolvassuk a buffert, a kernel-szint nem fog semmit felülírni, hiszen bármely pillanatban elmondható, hogy legalább egy üzenetnyi hely van benne.)

Végezetül rendkívül gondosan mérlegelnünk kell, hogy a user-szint valamely pontja nem "interrupt-érzékeny"-e, azaz nem következik-e be valami jóvátehetetlen hiba akkor, ha azon a ponton érkezik egy karakter vagy esetleg egy EOM. Mivel a kiolvasás és beírás folyamatát a két mutatóval eléggé szétválasztottuk, egyetlen ilyen veszélyforrást látok. Ez pedig a következő: ha egy üzenet kiolvasása közben (mialatt tovább folyik a következő üzenet vétele) olyan pillanatban érkezne újabb EOM, amikor az indikátort még nem töröltük, akkor a kernel-szint alulról gondosan 1-el írja felül az 1-et, amit a visszatérés után a user-szint töröl - így nem szerez tudomást az újabb beérkezett üzenetről. Semmi baj; amikor megint jön egy EOM, akkor kiolvassa ezt a most ottfelejtett üzenetet, tehát látszólag csak késésben és nem lekésésben van. Igen ám, de ekkor jóval több karakter van a bufferben, és bekövetkezhet az, amitől rettegtünk; még nem olvastunk ki valamit, de a kernel-szint már felül is írta. Ez a késés miatt lehetséges: bár egy üzenet hossza feltevésünk szerint nem haladja meg a 64 byte-ot, most két és fél üzenet is lehet egyszerre a bufferben, ami már több, mint 128 byte. Hogyan kerülhetjük el ezt a csapdát?

A tervezés során hibát követtünk el, mivel nem azonos eseteket is összemostunk; az indikátorban egyformán 1-el jeleztük, ha egy vagy ha több EOM van a bufferben. Tehát változtassuk meg a kernel-szintet úgy, hogy ne 1-el töltse a kérdéses változót az EOM vételekor, hanem növelje meg eggyel. A user-szint pedig

olvasson EOM-ig, majd csökkentse a változót. Ekkor persze ez már nem is indikátor, hanem számláló. A továbbiakban mégis indikátornak nevezzük, mert elsősorban arra vagyunk kíváncsiak, hogy Ø-e vagy sem, és csak a második közelítésben használjuk fel a konkrét értéket.

Most már valóban nem interrupt-érzékeny a user-szint? Ha az indikátor csökkentését egy utasítással oldjuk meg, akkor nem, mivel hardware interrupt csak két utasítás között jöhet be. De ha a csökkentés és vizsgálat céljából beolvassuk a indikátort egy regiszterbe, akkor igen! Tegyük fel, hogy értéke 3 volt, melyet behoztunk az AX regiszterbe. Ekkor bevág egy interrupt, amely az újabb EOM érkezését jelzi. Az AX a stackre kerül, majd a kernel-szint megnöveli az indikátort 4-re. Ezután a kernel-szint helyreállítja a regisztereket, és kilép. A folytatódó user-szint az AX-ben 3-at talál, amit csökkent és visszaírja az indikátorba - most is bekövetkezett az üzenetvesztés. Ott vagyunk tehát, ahonnan elindultunk? Korántsem! Az előző koncepció szerint ugyanis az egész kiolvasó rutin volt interrupt-érzékeny, most pedig csak két-három utasítás. Ezeket egy

```
CLI                ;;;Interrupt tiltása
.
.
STI                ;  Interrupt engedélyezése
```

utasításpár segítségével "megvédhetjük". De nem lehetett volna ugyanezt tenni az előző esetben? Dehogynem; de csak az egész kiolvasó rutinnal. Ez az utasításpár azonban a közte levő kódot lényegében kernel-szintűvé teszi. A második koncepció két-három utasítása bármikor futhat kernel-szinten, az első terv több száz utasításvégrehajtásból álló rutinja azonban nem, mert az könnyen vezethet interrupt vesztéshez.

Lássuk most az interrupt rutin pontos tervét szinte utasításról utasításra. A rutin a használt regiszterek mentése után beolvassa a vonali státuszt, majd megvizsgálja, van-e valóban vett karakter a fogadó regiszterben. Ha nincs, akkor a példában egyszerűen átsiklunk a hiba fölött, ha pedig van, beolvassuk, betöltjük a bufferbe, összevetjük az EOM-al. Ha EOM, akkor megnöveljük az indikátorváltozót is. Dolgunk végeztével helyreállítjuk a használt regisztereket, és visszatérünk.

> Megjegyzés: itt is láthatjuk, mennyire nehéz interrupt rutint írni: ha nincs vett karakter, és az interrupt más okból következett be, akkor mit csináljunk, hogyan

értesíthetjük a user-szintet a hibáról? Erre és még sok váratlan eseményre valós körülmények között fel kell készülni!

A mutatókat két szóban célszerű elhelyezni. Az egyik szó tartalmazza a buffer kezdőcímét, a másik pedig egy offsetet, amely megmutatja, hol is tartunk a buffer írásában vagy olvasásában. Ennek igazi hasznát a rutin utáni egyik megjegyzésben látjuk majd.

Vegyük észre, hogy a növelés arra is jó, hogy a user-szint mindig tudja: van-e a bufferben üzenetvége jelzés, és ha igen, hány van. Ha a jelzettnél kevesebbet talált, akkor nyilván elveszett egy, mert a kernel-szint (körülszaladva a bufferen) közben felülírta - íme egy olyan hiba, amelyről senki nem tehet. Ha a user-szint nem tud elég gyorsan olvasni, akkor karaktervesztés (és ennek következtében esetleg üzenetvesztés) lép fel.

Az indikátor alapértelmezésben 0, és a gyakorlatban nem szabad 1-nél nagyobbra nőnie. A user-szint csak az indikátort figyeli. Ha az nem 0 (mert a kernel-szint megnövelte), akkor fürgén kiolvassa az üzenetet az utolsó általa olvasott karakter utáni byte-tól (melynek címét az olvasási mutató tartalmazza) az EOM-ig, majd csökkenti az indikátort. Mivel a user-szint memóriában dolgozik, sokkal gyorsabb, mint a vonal, így feltehetően nem érkezett be még három-négy karakternél több a kiolvasás idején - remélhető, hogy üzenetvége jel sem, a csökkentés után az indikátor értéke ismét 0.

Megjegyzés: ez sem egyszerű dolog: a user- és kernel-szint kommunikációja még egy ilyen rendkívül egyszerű esetben is mennyi problémát vet fel, pedig ez a példa a teljes megoldásnak csak nagyon kezdetleges vázlata. Ha a feladatot teljes precizitással, gondosan, "bombabiztosan" akarjuk megoldani, akkor több hetes vagy (kellő gyakorlat híján) több hónapos küszködés következik.

```
;
; REC_CHAR_INT - interrupt handler rutin karakter
;                fogadásához
;-------------------------------------;
ASN_B_SIZ       EQU     128     ;Buffer hossza
ASN_EOM         EQU     0       ;End-Of-Message, üzenet vége
                                ;-------------------------;
ASN_BUF         DB      ASN_B_SIZ       DUP     (?)     ;
                                ;-------------------------;
```

```
ASN_B_BAS       DW      ASN_BUF ;Kezdőcím
ASN_B_WOFS      DW      Ø       ;Irási offset
ASYN_B_ROFS     DW      Ø       ;Olvasási offset
ASYN_EOM_IND    DB      Ø       ;End Of Message indikátor
                                ;
REC_CHAR_INT    PROC    FAR     ;--------;
                PUSH    AX              ;;;
                PUSH    BX              ;;;Használt
                PUSH    DX              ;;; regiszterek
                PUSH    DI              ;;; mentése
                                        ;;;------;;;
                MOV     DX, ASN_BAS + ASN_LSR   ;;;Vonalstátusz
                IN      AL, DX          ;;;------;;; beolvasás
                TEST    AL, Ø1H         ;;;Fogadás volt?
                JZ      ASN_REC_INT_RET ;;;Nem, visszatér
                                        ;;;
                MOV     DX, ASN_BAS     ;;;
                IN      AL, DX          ;;;Karakter be
                                        ;;;
                MOV     BX, CS:ASN_B_BAS;;;Alapcím
                MOV     DI, CS:ASN_B_OFS;;;Szabad hely címe
                MOV     CS:[DI][BX], AL ;;;Bufferbe tölt
                                        ;;;
                INC     DI              ;;;
                CMP     DI, ASN_B_SIZ   ;;;Tele a buffer?   (*)
                JNE     REC_CHAR_INT_UPD;;;                 (*)
                XOR     DI, DI          ;;;Igen, elölről    (*)
REC_CHAR_INT_UPD:                       ;;;                 (*)
                MOV     CS:ASYN_B_OFS,DI;;;új számláló
                                        ;;;
                CMP     AL, ASN_EOM     ;;;Nem Ø-ra, EOM-ra
                                        ;;; vizsgálunk !!!
                JNE     REC_CHAR_INT_NEOM
                INC     ASN_EOM_IND     ;;;Számlálja őket !
REC_CHAR_INT_NEOM:                      ;;;
REC_CHAR_INT_RET:                       ;;;Visszatérés
                MOV     AL, EOI         ;;;Minden interrupt
                OUT     INT_BAS, AL     ;;; engedélyezve
                POP     DI              ;;;
                POP     DX              ;;;
                POP     BX              ;;;
                POP     AX              ;;;
                IRET                    ;;;
REC_CHAR_INT    ENDP                    ;
;---------------------------------------;
```

Térjünk ki még néhány fontos dologra, amelyet a fenti példával kapcsolatban érdemes átgondolni és megfigyelni.

a) Amikor azt vizsgáljuk, hogy a beolvasott karakter EOM-e vagy sem, akkor nem azt vizsgáltuk, AL nulla-e (ezt a

```
TEST    AL, AL
```

utasítás végezte volna el), hanem a CMP-t alkalmaztuk. Azért lényeges ez, mert nem használtuk ki az EOM különleges értékét, később bármivel helyettesíthetjük majd.

b) Figyeljük meg, hogy e rutin változói mind a kódszegmensben helyezkednek el. Nem töltöttük át CS értékét DS-be, ezzel is időt nyerünk az interrupt szinten. Ehelyett minden adatcímzés elé odaírtuk a CS: prefixumot. Ezt az eljárást mint igen ritkán, csak indokolt esetben követendő eljárást tessék értékelni. A kódszegmensben adatokat akkor célszerű elhelyezni, ha azokat "el akarjuk dugni" más programrészek szeme elől. Tegyük fel, hogy programunk több kódszegmenst tartalmaz. Ekkor persze más kódszegmensből igen bajos (bár nem lehetetlen) ezen adatok elérése!

c) Minden kommentárt három pontosvesszővel írtunk ki, ezzel is kiemelve azt, hogy ez a program egy kernel-szintű kód.

d) Ha kihasználjuk, hogy a buffer hossza 128 byte (vagy kettőnek bármilyen egyéb hatványa), akkor megspórolhatjuk a buffer végének elérésére vonatkozó vizsgálatot. Ekkor az offset Ø-tól 127-ig változhat. Ha minden növelés után maszkoljuk a 128 helyiértékű bitet, akkor az offset sohasem nőhet 128 fölé, és elérve azt, automatikusan visszaáll Ø-ra. Ekkor a "*"-al jelzett négy sor (három utasítás) helyett egyszerűen írhatjuk az

```
AND     DI, NOT ASYN_B_SIZ
```

utasítást, nincs feltételvizsgálat. Itt persze kihasználtuk azt is, hogy a buffert bázisregiszteres indirekt címzéssel címeztük (így az index, a bufferen belüli pointer nullától indult).

e) Minden interrupt rutin írása esetén mérlegelnünk kell, hogy a rutin mely pontjain engedhetjük meg újabb interrupt fogadását. Ez különösen fontos software interrupt rutinok esetén. A hardware interrupt rutinok általában jobb, ha nem engedik meg az újabb megszakítást. Csak a pillanatnyilag aktív interruptnál magasabb prioritású interrupt aktivizálódik. Esetünkben a klaviatúra és a timer jöhet szóba. Nos, mindkettő fontos, és döntenünk kell, beengedjük-e őket. Mivel a

mi interrupt rutinunk nagyon rövid, nyugodtan megengedhetjük magunknak azt a luxust, hogy nem engedjük be őket, várjanak csak meg bennünket. De ha szükségesnek látszik, akkor a rutin legelején kiadhatunk egy

CLI

utasítást. A rutin ekkor is kernel-szintű marad, mert csak magasabb prioritású interrupt szakíthatja meg, így a következő karakternek várni kell.

f) Valamilyen ravasz trükkel értesíthetjük a user-szintet az EOM megérkezéséről, hogy ne legyen anarchikus az indikátor vizsgálata és így az üzenet fogadásának érzékelése.
Ez történhet egyszerűen annak a rutinnak a meghívásával is, amely rutin kiolvasná a buffert, az üzenetet végleges helyére téve. Ez azonban nemcsak célszerűtlen, hanem egyenesen tilos; nem szabad ugyanis túl hosszú időt az interrupt szinten tölteni; nem beszélve arról, hogy ez a rutin valószínűleg szeretne MS-DOS funkciókat használni, ez pedig hardware interrupt szinten nem megengedett, hiszen nem tudhatjuk: felhasználói szintű vagy MS-DOS szintű programot szakított-e meg az interrupt.
Ha a kiolvasó rutin hívása előtt felszabadítjuk az interrupt rendszert, azaz engedélyezzük bármely interrupt (beleértve az aszinkron vonali interrupt) fogadását is, akkor ez a rutin lényegében megszakítható szinten fut. Ekkor azonban garantálni kell, hogy az interrupt rutint megszakító másodszori interrupt semmit nem fog összekeverni sem az adatstruktúrában, sem pedig a veremben.
Egy másik lehetőség egy elágazási (fork) eljárás alkalmazása. Ez úgy működik, hogy a vermen a visszatérési cím és a mentett regiszterek közé "belopja" a user-szintű rutin címét (3 szó, ahol az első a kívánt státusz, a másik kettő pedig a rutin szegmense és offsetje). Az interrupt rutin visszatérésekor tehát nem a hívóhoz, hanem egy másik rutinhoz tér vissza, és dolga végeztével ez fog majd visszatérni a valódi hívóhoz. Az ilyen elágazási eljárást rendkívüli gondossággal és precizitással kell megírnunk, hiszen a vermen operál, és a legkisebb hiba is végzetes következményekkel jár.

VII. A hangszóró kezelése

Az IBM PC/XT-ben van egy hangszóró, amely igen változatos effektusok létrehozására alkalmas (erről meggyőződhetünk a forgalomban levő számtalan játékprogram bármelyikét elindítva).

A hangszóró kezelésére ROM BIOS funkciók nincsenek, ezt magunknak, programból kell megoldani. Erre két lehetőség van. Az egyik: a hangszóró gyors ki-bekapcsolásával hozzuk létre a kívánt magasságú és hosszúságú hangot, a másik pedig az, hogy ezzel a feladattal a beépített 8253 típusjelű Timert bízzuk meg.

A hangszórót a többcélú 8255 típusjelű PPI-vel, annak is a középső (B) portjával vezérelhetjük (címe 61H). E port két alsó bitje van hozzárendelve a hangszóróhoz és a Timernek a hangszóróval összekötött 2. csatornájához (bővebben az "Egyéb tudnivalók a hardware-ről" fejezetben). A port legalsó (Ø sorszámú) bitje a Timer 2. csatornájának Gate (a csatorna működését engedélyező) bemenete; ennek 1-be állítása kapcsolja be az előzetesen felprogramozott Timert. Az 1 sorszámú bit a hangszóró kapcsolója. Ennek 1 értéke bekapcsolja a hangszórót, a Ø pedig ki.

A direkt vezérléshez a legalsó bitet törölni kell, hiszen nem használjuk a Timert. Ezután ha a második bitet 1-be állítjuk, akkor feszültség alá kerül a hangszóró, és pittyen egyet, ha pedig töröljük, akkor megszűnik a feszültség, és egy újabb pittyenést hallunk. Folyamatos, adott magasságú hang létrehozásához legegyszerűbb egy kettősciklust használni, amelyben a külső ciklus szabályozza a hang hosszát, a belső várakozási ciklus pedig a hang magasságát.

A másik lehetőség a Timer használata a hang magasságát megadó belső ciklus helyett. A háromcsatornás Timer harmadik csatornája van hozzákapcsolva a hangszóróhoz (ez persze becsapós dolog, mert a számozás nullával kezdődik, így ez a csatorna sorszám szerint a kettes). Ehhez fel kell programoznunk a Timert, és be kell kapcsolnunk a hangszórót. Mi is történik ebben az esetben? A Timer előkészítésekor azt adjuk meg, hogy az órajel alapfrekvenciájának hányadrésze legyen a kívánt frekvencia. A Timer ezután a megadott rezgésszámú jelet generálja a hangszóróhoz kapcsolt kimenetén, és ez a jel fogja ki- és bekapcsolni a hangszórót. A hangot ezután addig halljuk, míg a hangszórót ki nem kapcsoljuk, vagy a Timert le nem programozzuk.

VII.1. Hangmagassági táblázat

A hangok létrehozásaához egy olyan, a hang magasságára jellemző konstansra is szükségünk van, amelyet a Timer/Counter felprogramozásához használunk fel. Erre a következő egyszerű

képlet szolgál: a gép órájának alapfrekvenciáját el kell osztanunk a kívánt rezgésszámmal. Ez az érték 1 193 180 (a Timer legfeljebb a másodperc ennyied részét tudja lemérni). A képlet:

$$\text{számláló} = \frac{1\ 193\ 180}{\text{kívánt hangfrekvencia}}$$

Lássuk most a C-dúr skála alaphangjait, rezgésszámukat és a hozzájuk tartozó egész értékeket.

hang	rezgésszám	számláló (dec)	számláló (hexa)
c	261.63	4560	11D0H
d	293.66	4063	FDFH
e	329.63	3619	E23H
f	349.23	3416	D58H
g	392.00	3043	BE3H
a	440.00	2711	A97H
h	493.88	2415	96FH
c'	523.25	2280	8E8H

Hangmagasságok és a hozzájuk tartozó osztók

VII.2. Példaprogramok

A hangszóró kezelésére első közelítésben két példa olvasható, egy egyszerű direkt, és egy bonyolultabban megvalósított timeres kezelés (utóbbi úgy van megírva, hogy a Microsoft 4.00 verziójú C fordítójával kompatibilis, tehát ez a rutin C nyelvből is meghívható). Figyeljünk arra, hogy az első változat regiszterekben, a második pedig stacken kapja a paramétereket.

VII.2.1. A hangszóró direkt vezérlése

```
;       Input:
;               BX      - hosszra utaló érték
;               DX      - hangmagasság (számláló)
;       Output:
;               semmi
;-----------------------------------------;
PPI_B           EQU     61H             ;Portcím
                                        ;
```

```
DSND            PROC    NEAR            ;
                PUSH    AX              ;AX mentése
                IN      AL, PPI_B       ;Port beolvasás
                MOV     AH, AL          ;Eredeti mentés
                AND     AL, NOT 01H     ;Alsó bit törlés
                OUT     PPI_B, AL       ;Direkt vezérlés
DSND_EX:                        ;Külső ciklus
                MOV     CX, DX          ;Belső c. előkész.
                OR      AL, 02H         ;Hangszóró be
                OUT     PPI_B, AL       ;
DSND_IN:                        ;Belső ciklus
                NOP                     ;
                LOOP    DSND_IN         ;Míg CX nem 0
                OR      AL, NOT 02H     ;Hangszóró ki
                OUT     PPI_B, AL       ;
                DEC     BX              ;Hang hosszúság
                JNZ     DSND_EX         ;
                MOV     AL, AH          ;
                OUT     PPI_B, AH       ;Vezérlőport visszaáll.
                POP     AX              ;AX helyreállítása
                RET                     ;
DSND            ENDP                    ;
;-----------------------------------------;
```

VII.2.2. A hangszóró vezérlése timerrel

Természetes igény, hogy ne "direktben", hanem a Timer erre szolgáló csatornája segítségével vezéreljük a hangszórót. Gondoljuk csak el, mi történik akkor, ha olyan programot írunk, amely a fenti szubrutint használja a hang létrehozására, majd nagy mellénnyel átvisszük egy olyan gépre, melynek órajele eltérő frekvenciájú, és így az utasítások végrehajtása gyorsabb (vagy lassúbb). Ekkor a keletkező hang egészen más magasságú és időtartamú lesz, mint amit eredetileg elképzeltünk.

Ez ellen csak úgy védekezhetünk, hogy az IBM PC-típusú gépek közös Timerét használjuk mind a hang magasságának beállítására, mind időtartamának meghatározására. Ehhez tudnunk kell, hogy a Timer harmadik csatornája közvetlenül a hangszóróhoz van kötve; csak fel kell programoznunk őt, és be kell kapcsolni a hangszórót, máris szól a kívánt magasságú hang.

A hang hosszának vezérlésére ennél ravaszabb trükköt kell kitalálnunk. Az alapötlet itt is a Timer felhasználása lesz. A Timer első csatornája (sorszám szerint a 0.) az interrupt vezérlővel van összekötve. Ez másodpercenként kb. 18-szor generálja az ún. timer-interruptot. Ha pedig "magunkhoz szólítjuk"

ezt a hardware-interruptot (sorszáma 08H), vagy pedig a Timer felhasználói interruptját (sorszáma 1CH), akkor egyszerű lesz a hang hosszának vezérlése (felhívjuk a figyelmet arra, hogy mindig a második utat kell választanunk!). A felhasználói interrupt egy számlálót fog csökkenteni. Ha ez a számláló elfogy, akkor kell befejezni a hang kibocsátását, és elindítani a következőt, vagy leállítani az egész eljárást.

Első lépésként oldjuk meg a hangmagasság beállítását a Timer segítségével. A következő lépés legyen a probléma teljes megoldása, a Timer felhasználói interruptjának bekapcsolása.

```
;  _SOUND - adott magasságú hang generálása
;       C-kompatibilis rutin (Microsoft 4.00, modellfüggetlen)
;       Input paraméterek a stack-en:
;                 első paratméter          - a hang időtartama
;                 második paraméter        - a hang frekvenciája
;-------------------------------------;
;LMODEL    EQU      1                  ;Hosszú modellnél töröld ";"-t!
                                        ;
        IFDEF   LMODEL                  ;Paraméterblokk távolsága
PRDS            EQU      06H            ; hosszú és
        ELSE                            ;
PRDS            EQU      04H            ; rövid modell esetén
        ENDIF                           ;
VRDS            EQU     -06H            ;Változóblokk távolsága
                                        ;
SSIZ            EQU      00H            ;Paraméterek offsetje a
SFRQ            EQU      02H            ; paraméterblokkban
                                        ;
TMC_CHN2        EQU      42H            ;Timer 2. csatorna adatport
TMC_CONT        EQU      43H            ;Timer vezérlőport
                                        ;
        IFDEF   LMODEL                  ;Hosszú modell
_SOUND_TEXT     SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:_SOUND_TEXT, DS:NOTHING, ES:NOTHING
_SOUND          PROC    FAR             ;
        ELSE                            ;Rövid modell
_TEXT           SEGMENT BYTE    PUBLIC  'CODE'
        ASSUME  CS:_TEXT, DS:NOTHING, ES:NOTHING
_SOUND          PROC    NEAR            ;
        ENDIF                           ;IF LMODEL vége
                PUSH    BP              ;--------;BP mentése
                MOV     BP,  SP                 ;BP előkészítése
                PUSH    SI                      ;Indexregiszterek
                PUSH    DI                      ; mentése
```

```
                        ;-----------------;
                        ;Timer felprogramozása
                        ;-----------------;
                MOV     AL, 0B6H          ;Timer 2. csatorna
                OUT     TMC_CONT, AL      ; előkészítése
                MOV     AX,  SFRQ[ BP + PRDS ]  ;Hangma-   (*)
                OUT     TMC_CHN2, AL      ;gasság beáll.
                MOV     AL,  AH           ;
                OUT     TMC_CHN2, AL      ;
                        ;-----------------;
                        ;Hangszóró bekapcsolása
                        ;-----------------;
                IN      AL,  PPI_B        ;
                MOV     AH,  AL           ;Eredeti mentése
                OR      AL,  3            ;Hangszóró be,
                OUT     PPI_B, AL         ; Timerrel
                        ;-----------------;
                        ;Várakozás a megszabott ideig
                        ;-------------------------;
                MOV     CX,  SSIZ[ BP + PRDS ]  ;Hossz be  (*)
SOUND_WEX:                                ;Várakozás
                MOV     BL,  10           ; ezalatt valami
SOUND_WIN:                                ; mást is lehetne
                DEC     BL                ; csinálni
                JNZ     SOUND_WIN         ;
                LOOP    SOUND_WEX         ;Hang valódi hossza
                        ;-----------------;
                        ;Hangszóró kikapcsolása
                        ;-----------------;
                MOV     AL,  AH           ;Eredeti érték vissza-
                OUT     PPI_B, AL         ; írása a vezérlőportra
                POP     DI                ;Indexregiszterek
                POP     SI                ;helyreállítása
                                          ;
                POP     BP                ;BP (és SP) visszaáll.
                RET                       ;
_SOUND  ENDP                              ;
                                ;-------;
        IFDEF   LMODEL          ;Hosszú modell
_SOUND_TEXT     ENDS
        ELSE                    ;Rövid modell
_TEXT           ENDS
        ENDIF                   ;IF LMODEL vége
        END                     ;Nincs kezdőcím!
```

Szubrutinunk elmentette a BP értékét, és az SP tartalmát átmásolta a BP-be; ezáltal BP ugyanoda mutat, mint SP. Ezután következett a két indexregiszter mentése (ez most nem feltétlenül szükséges, mert a rutin nem használja őket).

> Megjegyzés: azért írtam ki mégis a mentéseket, mert ki akartam emelni e regiszterek mentésének és eredeti értéke visszaadásának fontosságát. Itt kapnak helyet a C nyelv "register" típusú változói; ha a rutinunkat meghívó C függvény definiált ilyeneket, és mi elrontjuk e regiszterek értékét, az bizony nagy bajt okoz!

Figyeljük meg, hogyan definiáljuk a szegmenst és az eljárás nevét! Először is, minden nevet az aláhúzás karakterrel kezdünk, mivel a Microsoft C fordító minden felhasználói név elé odateszi ezt a karaktert. A továbbiak megértéséhez címszavakban meg kell ismerkednünk a memóriamodellek fogalmával.

A Microsoft magasszintű fordítóprogramjainak megvalósításában "memóriamodelleket" definiál. A modellek azt szabják meg, hogy hány (azaz, hogy egy vagy több) szegmensbe fordítja-szerkeszti a fordítórendszer az általunk létrehozott programot. Az úgynevezett rövid modellben mind a program adat-, mind a kódterülete egy-egy szegmensbe kerül, így a program összes memóriaigénye nem lehet több, mint 128 kb. Ilyen esetben az adatcímzések és a rutinhívások mind közeliek lesznek, így sohasem kerül sor a szegmensregiszterek értékének módosítására. Az egyéb modellekben vagy a kód-, vagy az adatterület, vagy mindkettő átlépheti a 64 Kb-ot, azaz az egy szegmensnyi méretet. Ennek viszont az a konzekvenciája, hogy vagy a rutinhívásoknál, vagy az adatcímzések során, vagy pedig mindkét esetben sűrűn kell módosítani a szegmensregiszterek értékét. A szubrutinhívások (a C nyelvben: függvényhívások) általában távoliak lesznek, vagyis a hívás során két szót mentenek el a veremre. Ennek egyik következménye az, hogy nagyobb lesz a paraméterblokk távolsága a Stack Pointer által a belépéskor címzett szótól, a másik az, hogy közeli visszatérés helyett távolit kell végrehajtanunk.

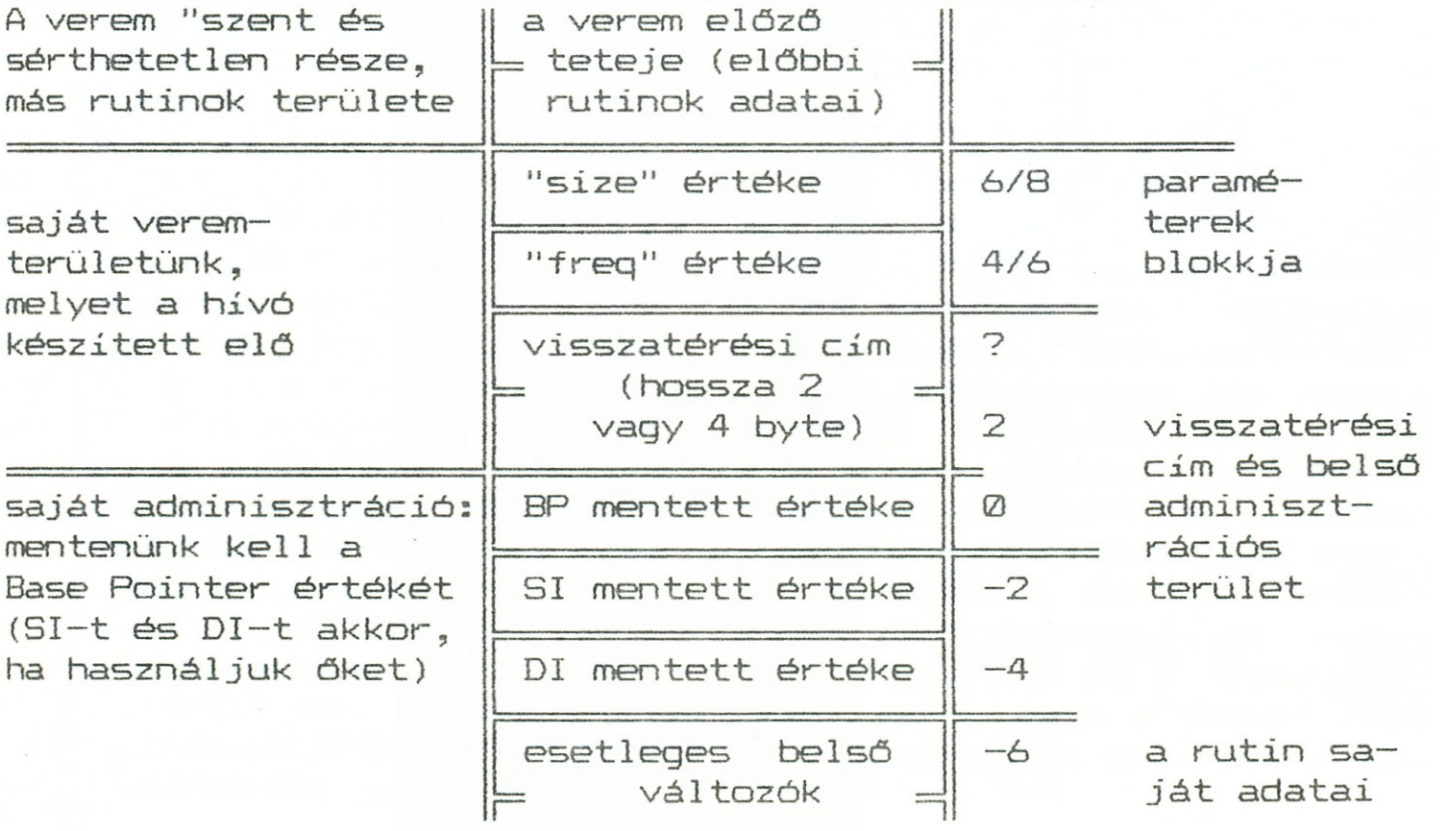
A feltételes fordítási blokkok megkönnyítik a modellek módosítását. Egyszerűen definiáljuk az LMODEL szimbólumot akkor, ha több kódszegmenst tartalmazó programhoz akarjuk szerkeszteni a rutint, újrafordítjuk, és minden rendben is van. A szegmenst is egy feltételes blokkban definiáljuk, mégpedig "_TEXT" néven rövid modell esetén (azért, hogy rutinunk egy szegmensbe kerüljön a C nyelv rutinjaival), vagy egyéb szegmensnéven (azért, hogy rutinunk **nehogy** egy szegmensbe kerüljön a C nyelv rutinjaival).

A rövid modell esetén "BYTE" igazítási típust adunk meg, hogy a "_TEXT" nevű, "CODE" osztályú szegmens kihagyás vagy (horribile dictu, volt már rá példa - megengedem, hogy LINK hiba miatt) **átfedés** nélkül illeszkedjen a C kódszegmensébe. Hosszú modell esetében az igazítás nyugodtan lehet "PARA", hiszen különálló szegmenst hozunk létre, amely úgyis csak paragrafushatáron kezdődhet. Ugyanebben a blokkban definiáltuk a szubrutin nevét és típusát is, a modellel megegyező módon "NEAR" vagy "FAR" típusra. Ezzel szabjuk meg a visszatérés módját.

Egy másik probléma a paraméterek átvétele. A legtöbb magasszintű programozási nyelv a vermen adja át a paramétereket a meghívott külső eljárásnak. Ismerni kell a nyelv protokollját: milyen hosszú memóriaterületet foglal egy adott típusú változó számára, és milyen sorrendben helyezi el a paramétereket. A C egy egész számára általában két byte-ot igényel, és a paraméterek elvermelése a híváskor megadott sorrend megfordításával történik, vagyis az utoljára megadott paraméter kerül le elsőként a verembe, azaz ez lesz majd legtávolabb a verem tetejétől. Ennek logikus oka van; ismertetésére a negyedik kötetben kerül sor. Lássuk most a vermet azután, hogy a C nyelvű program két egész paramétert elhelyezett, és meghívta szubrutinunkat!

A C nyelvű hívó sor: `sound( freq, size );`

A verem szerkezete (egy téglalap egy szót jelent):



Kézenfekvő ötlet, hogy a Base Pointert használjuk a vermen elhelyezett paraméterek elérésére. Ez a gondolat azonban nem csak nekünk természetes; éppen ezért szigorúan mentenünk kell a Base Pointert, hiszen a bennünket hívó C program függvényei is ezt használják. Hasonló okokból vagyunk kötelesek menteni az SI és DI regisztereket.

A paraméterek távolsága, mint a kódból és az ábrából is kitűnik, két összetevőből számítható ki: az adminisztrációs terület hosszúságából és a paramétereknek a blokkon belül elfoglalt helyzetéből. Ezt illusztrálja a konstansok definiálása és felhasználása is. A paraméterek felolvasása során élünk azzal a szabadsággal, amelyet az assembler biztosít. A példaprogramban (*)-al jelölt sorok olvassák fel a paramétereket. Ez a megfogalmazás szemlélteti a legjobban az egyes konstansok szerepét.

Megjegyezzük még, hogy (bár az adott rutinnak belső változói nincsenek) a belső (lokális) változókat ugyancsak a BP segítségével címezhetjük meg. Ilyen esetben az offsetek (mint az ábra is szemlélteti) negatív értékűek lesznek.

A fent megadott szubrutin felhasználására adunk egy egyszerű assembly példát: ez a kis program lejátssza a C-dúr skálát.

```
TITLE   C-dúr skála
        EXTRN   _SOUND:NEAR
; Konstansdefiníciók
;-----------------------------------------;
        .XLIST                          ;
                INCLUDE         DOSCLL.INC
        .LIST                           ;
                                        ;
SND_C           EQU     4560            ;Továbbiak hasonlók,
SND_D           EQU     4063            ; lásd a táblázatot
SND_C1          EQU     2280
        ...                             ;
SND_SIZE        EQU     1000            ;

DATA    SEGMENT PARA    PUBLIC  'CODE'  ;
SND_TAB         DW      SND_C, SND_D, SND_E, SND_F
                DW      SND_G, SND_A, SND_H, SND_C1
                DW      0               ;A tábla vége
DATA    ENDS

CODE    SEGMENT PARA    PUBLIC  'CODE'
START:                                  ;
                MOV     CX, SND_SIZE    ;A hívási ciklus
                MOV     DI, OFFSET SND_TAB      ;előkészítése
```

```
SOUND_LOOP:                                     ;
                PUSH    CX                      ;Hosszparaméter
                PUSH    WORD PTR [ DI ]         ;Hangmagasság
                ADD     DI, WORD                ;Cím módosítása
                CALL    _SOUND                  ;Rutinhívás
                ADD     SP, 2 * WORD            ;Stack Pointer vissza
                CMP     WORD PTR [ DI ], NULL
                JNE     SOUND_LOOP              ;
                DOSCLL  DOS_EXIT, NULL          ;
CODE    ENDS                                    ;
        END     START                           ;
```

Fontos, hogy ez a rutin NEAR típusúnak definiálja a "_SOUND" rutint; ebből azonban az is következik, hogy a "_SOUND"-ot tartalmazó file-ban az _LPROG változó elé be kell iktatnunk a pontosvesszőt.

VII.2.3. Hangszóró-driver megvalósítása

A továbbiakban ennél is jobban használható rutint írunk: célunk egy olyan hangszórókezelés megírása, mely az előkészítés után azonnal visszaadja a vezérlést a hívónak, és saját munkáját háttérben végzi. Ennek felhasználásával a hívó nyugodtan folytathatja munkáját, tehát a zajcsinálás nem igényel állandó programjelenlétet.

A feladat vázlata már fent olvasható; a hang magasságát a már megismert módon, a Timer 2. csatornájának felhasználásával szabályozzuk, a hang hosszát pedig a Timer 0. csatornájának figyelésével. Most nem is annyira a technikai, mint a szervezési problémákra fogunk koncentrálni. A programunk egy minimális driver lesz (csekély fáradsággal tovább lehet fejleszteni, hogy akár a CONFIG.SYS-ben lehessen driverként beépíteni a rendszerbe, akár az ANSI.SYS-t). Egy driver természetesen rezindens (most először fogunk találkozni azzal, hogy egy program bentmaradva lép ki a rendszerbe, vagyis a kód megőrzésével tér vissza az őt hívó programhoz). Maga a program látszólag semmit nem csinál; a valóságban azonban elvégez néhány előkészítő lépést. Először is, lefoglal egy szabad interrupt vektort (a példában az FFH-et használtuk); ez lesz a kommunikációs interrupt; ezenkívül még egy interrupt vektort szólít magához, az 1BH sorszámút, amely a CTRL-BREAK esetén meghívandó felhasználói rutin címét tartalmazza. Erre azért van szükség, hogy ha a felhasználó CTRL-BREAK-kel kilövi a programját, akkor maga a driver lője ki saját magát, azaz hallgasson el az abortált program által megszólaltatott zene is.

Több teendője nincs is; szépen előkészíti a rezidens kilépést. Figyeljük meg, hogy két kódszegmens van; az alacsonyabb címre kerülő CODE szegmens tartalmazza a fontos rutinok kódját; a garantáltan a legmagasabb területre töltődő MEMORY szegmens pedig az előkészítő részeket. Ez azért ilyen bonyodalmas, hogy a rezidens részt a lehető legkisebbre szorítsuk össze.

A program a kommunikációs interrupton át ugyanúgy kommunikál a hívóval, mint a ROM BIOS rutincsomagjai. Az AH regiszter tartalmazza a funkciókódot, és (egyébként egyetlen funkció esetén) más regiszterekben vannak a paraméterek. Driverünk három funkciót valósít meg:

- AH = 00H, hangsorozat átadása a drivernek
  Az ES:BX regiszterpárban kell átadni egy olyan struktúratömb kezdőcímét, amelynek egy eleme egy hangot ír le, hosszával (időtartamával) és magasságával (az előállításához szükséges számlálóértékkel). A tömb tetszőleges hosszúságú lehet, a program arról ismeri meg a végét, hogy az utolsó elem időt leíró elemének értéke 0.
  E funkció hívása után a driver haladéktalanul elkezdi a hangsorozat (dallam) megszólaltatását, és visszaadja a vezérlést a hívónak. Sorban egymás után veszi a hangokat, programozza Timer 2. csatornáját és a hangszórót, majd a Timer 0. csatornája segítségével megvárja a hang végét. Ha a dallamban szünetet akarunk beiktatni, akkor a hang magasságaként 0 értéket kell megadni.
  A hangsorozat végének elérésekor a program automatikusan befejezi működését, az újabb hívásig.
  Ha a driver e funkció hívásakor aktív (vagyis egy korábbi kérés kiszolgálásával van elfoglalva), akkor megszakítja a korábban kapott hangsorozat megszólaltatását, és azonnal hozzáfog az újhoz.

- AH = 01H, hagsorozat megszólaltatásának megszakítása
  Ez a hívás nem igényel egyéb paramétereket. Hatására azonnal abbamarad annak a hangsorozatnak a megszólaltatása, melyet a driver éppen kezel. Ha a driver nem aktív, akkor a funkció hatástalan (nem okoz bajt).

- AH = 02H, a driver státuszának lekérdezése
  Ez a funkció sem igényel paramétereket. Válaszértékként az AL-ben egyes értéket kapunk, ha a driver aktív, vagyis éppen egy hangsorozat megszólaltatása folyik; az AL-ben nullát kapunk vissza, ha a driver inaktív. Segítségével tudhatja meg a drivert használó program, hogy befejeződött-e a korábban elküldött hangsorozat megszólaltatása, vagy sem.

Látszólag elbonyolítottuk a feladatot, de be fogjuk látni, hogy egy mégoly egyszerű driver írása sem olyan nagyon egyszerű, ha komoly szolgáltatást akarunk nyújtani. A legnehezebb talán egy alapos diszkusszió felállítása: minden lehető és lehetetlen esetet figyelembe kell venni, és programunkat (ha lehet) a legrosszabb szituációkra is felkészíteni.

```
TITLE   Speaker driver
COMMENT *
A file neve:                sddrv.asm
Fordítás                    MASM 4.00, opciók nélkül
Szerkesztés:                LINK opciók nélkül
Futtatás:                   sddrv
        A program az újabb rendszertöltésig aktív marad.
*
;-------------------------------------;
; Makrók és struktúrák
; S_ITV         az IT_NUM interrupt vektor eredeti tartalmának
;               mentése és új értékkel való felülírása
; Lásd IT-MOD.INC (Bevezető fejezet)
; R_ITV         az IT_NUM interrupt vektor helyreállítása
;               a SAVE_ADDR által adott pozícióról
; Lásd IT-MOD.INC (Bevezető fejezet)
;
; A hang leírására szolgáló struktúra
;-------------------------------------;
SD_STR  STRUC                         ;
        SD_SIZ  DW      0             ;A hang hossza 1/18 sec-ben
        SD_FR   DW      0             ;Frekvencia-megadási érték
SD_STR  ENDS                          ;
;-------------------------------------;
; Konstansok  - interrupt-sorszám, funkciókódok, válaszértékek
;-------------------------------------;
SD_INT          EQU     0FFH          ;Kommunikációs interrupt
                        ;--------;
                        ; Funkciókódok
                        ;--------;
SND_START       EQU     00H           ;Hangsorozat megszólaltaltása
SND_CANCEL      EQU     01H           ;Megszólaltatás felfüggesztése
SND_ASTST       EQU     02H           ;Státusz lekérdezése
                        ;--------;
                        ; Válaszinformációk
                        ;--------;
        SD_BUSY EQU     1             ;A driver aktív
        SD_IDLE EQU     0             ;A driver nem aktív
```

```
;-------------------------------------;
; Adatterületek
;-------------------------------------------------;
DATA    SEGMENT PARA    PUBLIC  'DATA'  ;
TM_USER_SAVE    DD      Ø               ;Mentőterület (INT 1CH)
TM_USER_OWN     DD      SD_TIMER        ;Saját rutin (INT 1CH)
                                        ;"Hardware" interrupt
SD_SAVE         DD      Ø               ;Mentőterület (INT FFH)
SD_ENTRYP       DD      SD_ENTRY        ;Saját rutin (INT FFH)
                                        ;Kommunikációs belépés
SD_QSAVE        DD      Ø               ;Mentőterület (INT 1BH)
SD_QUITP        DD      SD_BREAK        ;Saját rutin (INT 1BH)
                                        ;CTRL-BREAK kezelés
TM_COUNTER      DW      Ø               ;Pillanatnyi számláló
TM_TABPNT       DD      Ø               ;Pointer a sorozatra
TM_INDICATOR    DW      Ø               ;Aktív/inaktív ind.
DATA    ENDS                            ;
;-------------------------------------------------;
; Stack terület                                   ;
STACK   SEGMENT PARA    STACK   'STACK' ;
        ;-----------------------------------------------------;
        ; !!! E programnak nincs saját stack-je !!!          ;
        ;-----------------------------------------------------;
STACK   ENDS                            ;
;-------------------------------------------------;
; Kódterület
;-----------------------------------------------------------;
                                                            ;
CODE    SEGMENT PARA    PUBLIC  'CODE'                      ;
        ASSUME  CS:CODE, DS:DATA, ES:NOTHING, SS:STACK
;-----------------------------------------------------------;
; SD_ENTRY      - kommunikácóis belépési pont
;       Funkciók:
;       AH      - Ø     hangsorozat megszólaltatása
;                 ES:BX címzi a hagsorozat leírását
;       AH      - 1     a megszólaltatás leállítása
;       AH      - 2     a driver státuszának lekérdezése
;                 AL - 1 ha aktív, Ø ha inaktív
;-----------------------------------------------------------;
SD_ENTRY        PROC    FAR                                 ;
                PUSH    DS                                  ;DS mentése
                PUSH    AX                                  ;Funkciókód el
                MOV     AX, DATA                            ;DATA szegmens
                MOV     DS, AX                              ;beállítása
        ASSUME  DS:DATA                                     ;
```

```
                POP     AX                              ;Funkciókód fel
                TEST    AH, AH                          ;SND_START?
                JNZ     SD_NSTART                       ;Nem, folytatás
                CALL    SD_START                        ;
                JMP     SD_ENTRY_QUIT                   ;
SD_NSTART:                                              ;
                DEC     AH                              ;SND_CANCEL?
                JNZ     SD_NABORT                       ;Nem, folytatás
                CALL    SD_CANCEL                       ;
                JMP     SD_ENTRY_QUIT                   ;
SD_NABORT:                                              ;
                DEC     AH                              ;SND_ASKST?
                JNZ     SD_NASKST                       ;Nem, folytatás
                CALL    SD_ASKST                        ;
SD_NASKST:                                              ;
SD_ENTRY_QUIT:                                          ;
                POP     DS                              ;
                IRET                                    ;
                                                        ;
SD_ENTRY        ENDP                                    ;
                                                        ;
;-------------------------------------------------------;
; SD_START      - hangsorozat elindítása
;       Input:
;               ES:BX   - a hangokat leíró struktúratömb címe
;       Válasz:
;               nincs
;-------------------------------------------------------;
                                                        ;
SD_START        PROC    NEAR                            ;
                                                        ;
                MOV     AX, TM_INDICATOR                ;
                TEST    AX, AX                          ;A driver aktív
                JE      SD_START_IDLE                   ;Igen, rajta!
                CALL    SD_CANCEL                       ;Előzőt megöli
SD_START_IDLE:                                          ;
                MOV     TM_INDICATOR, SD_BUSY           ;Aktív lesz!
                WRPNT   ES, BX, TM_TABPNT               ;Cím a helyére
                MOV     TM_COUNTER, 0                   ;Nincs hang
                                                        ;---------------;
                S_ITV   BS_TMUSR, TM_USER_OWN, TM_USER_SAVE   ;
                                                        ;---------------;
                RET                                     ;Kész
                                                        ;
SD_START        ENDP                                    ;
```

```
;-------------------------------------------------------;
; SD_CANCEL     - hangsorozat leállítása
;       Input, válasz:
;                nincs
;-------------------------------------------------------;
SD_CANCEL       PROC    NEAR                            ;
                PUSH    AX                              ;A driver
                CMP     TM_INDICATOR, SD_IDLE           ; aktív ?
                JE      SD_CANCEL_QUIT                  ;Nem, kész van
                IN      AL, SP_CONT                     ;Hangszóró és
                AND     AL, NOT (SP_TGATE OR SP_ENABLE)
                OUT     SP_CONT, AL                     ; timer kikapcs
                XOR     AX, AX                          ;
                WRPNT   AX, AX, TM_TABPNT               ;Paraméterek
                MOV     TM_INDICATOR, AX                ; törlése
                MOV     TM_COUNTER, AX                  ;
                R_ITV   BS_TMUSR, TM_USER_SAVE          ;IT vissza
SD_CANCEL_QUIT:                                         ;
                POP     AX                              ;
                RET                                     ;
SD_CANCEL       ENDP                                    ;
;-------------------------------------------------------;
; SD_ASKST      - a driver státuszának lekérdezése
;       Input:  nincs
;       Válasz:
;                AL      - Ø ha a driver inaktív, 1, ha aktív
;-------------------------------------------------------;
                                                        ;
SD_ASKST        PROC    NEAR                            ;
                MOV     AX, TM_INDICATOR                ;Ennyi!
                RET                                     ;
SD_ASKST        ENDP                                    ;
                                                        ;
;-------------------------------------------------------;
; SD_BREAK      - felhasználói CTRL-BREAK rutin az éppen
;                 aktív drivermőködés leállítására
;       Input, válasz:  nincs
;-------------------------------------------------------;
                                                        ;
SD_BREAK        PROC    FAR                             ;
                PUSH    AX                              ;
                SPKCLL  SND_CANCEL                      ;A leállítás a
                POP     AX                              ;driverrel megy
                IRET                                    ;
SD_BREAK        ENDP                                    ;
```

```
;-------------------------------------------------------------;
; SD_TIMER      - hardware interrupt rutin, melyet az
;                 INT 08H rutin aktivizál
;-------------------------------------------------------------;
SD_TIMER        PROC    FAR                     ;;;
                PUSH    AX                      ;;;Regiszterek
                PUSH    BX                      ;;; mentése
                PUSH    DS                      ;;;
                PUSH    ES                      ;;;
                                                ;;;
                MOV     AX, DATA                ;;;DATA szeg-
                MOV     DS, AX                  ;;; mens elő-
        ASSUME  DS:DATA                         ;;; készítés
                DEC     TM_COUNTER              ;;;Időmérés
                JL      SD_TIMER_CNTRES         ;;;Lejárt, új!
                JMP     SD_TIMER_QUIT           ;;;Oké, várunk!
SD_TIMER_CNTRES:                                ;;;
                LES     BX, TM_TABPNT           ;;;Következő
                MOV     AX, ES:SD_SIZ.[ BX ]    ;;; hang hossza
                TEST    AX, AX                  ;;;Vége?
                JE      SD_TIMER_FINE           ;;;Leállítani
                MOV     TM_COUNTER, AX          ;;;Számláló
                        ;---------------;
                        ; Timer előkészítése, és a
                        ; hangszóró bekapcsolása
                        ;---------------;
                MOV     AL,  0B6H               ;;;2.cs,4. mód
                OUT     TMC_CONT, AL            ;;;előkészítés
                MOV     AX,  ES:SD_FR.[ BX ]    ;;;Hangfrekv.
                CMP     AX, 0                   ;;;Szünet?
                JE      SD_TIMER_PAUSE          ;;;
                OUT     TMC_CHN2, AL            ;;;Timer fel-
                MOV     AL,  AH                 ;;;programozás
                OUT     TMC_CHN2, AL            ;;;
                IN      AL,  SP_CONT            ;;;Hangszóró
                OR      AL,  SP_TGATE OR SP_ENABLE      ;;; be,
                OUT     SP_CONT, AL             ;;; Timer-el
                JMP     SD_TIMER_CONT           ;;;
SD_TIMER_PAUSE:         ;---------------;;;Szünet lesz!
                        ; Hangszóró kikapcsolása
                        ;---------------;
                IN      AL,  SP_CONT            ;;;Hangszóró
                AND     AL,  NOT (SP_TGATE OR SP_ENABLE);;; ki,
                OUT     SP_CONT, AL             ;;; Timer ki
```

```
SD_TIMER_CONT:                          ;----------------;
                                        ; Pointer állítása (offset)
                                        ;----------------;
                ADD     BX, 2 * WORD                    ;;;-----;;;
                MOV     WORD PTR OFFS.TM_TABPNT, BX     ;;;
                JMP     SD_TIMER_QUIT                   ;;;-----;;;
SD_TIMER_FINE:                                          ;;;Vége a so-
                CALL    SD_CANCEL                       ;;;rozatnak
                                                        ;;;
SD_TIMER_QUIT:                                          ;;;
                POP     ES                              ;;;Regiszterek
                POP     DS                              ;;; helyre-
                POP     BX                              ;;; állítása
                POP     AX                              ;;;
                IRET                                    ;;;
SD_TIMER        ENDP                                    ;
CODE    ENDS                                            ;
;-------------------------------------------------------;
; START         - a speaker driver előkészítése
;               az FFH és az 1BH interrupt vektorok
;               beállítása, és rezidens kilépés
;-------------------------------------------------------;
MEMORY  SEGMENT PARA    PUBLIC  'MEMORY'                ;
        ASSUME  CS:MEMORY                               ;
START:                                                  ;
                MOV     AX, SEG DATA                    ;DATA szegmens
                MOV     DS, AX                          ; beállítása
        ASSUME  DS:DATA         ;----------------;
                                ;IT-vektorok előkészítése
                                ;CTRL-BREAK felhasználói IT
                                ;----------------;
                S_ITV   BS_CTBRK, SD_QUITP, SD_QSAVE
                                ;----------------;
                                ;Driver kommunikációs IT
                                ;----------------;
                S_ITV   SD_INT, SD_ENTRYP, SD_SAVE
                MOV     AX, ES                          ;Programhossz
                MOV     DX, SEG MEMORY                  ; kiszámítása
                SUB     DX, AX                          ;(MEMORY - PSP)
                DOSCLL  DOS_STAYRES, 0                  ;Rezidens ki-
MEMORY  ENDS                                            ; lépés
;-------------------------------------------------------;
                END     START
```

A következőkben lássunk egy hívási példát!

```
COMMENT *
A file neve:                    sd.asm
        Fordítás:               MASM 4.00 opciók nélkül
        Szerkesztés:            LINK opciók nélkül
        Futtatás:               sd
*
; Include fejezet
;-----------------------------------------;
        .XLIST                          ;
        INCLUDE         DOSCLL.INC      ;
        .LIST                           ;
;-----------------------------------------;
; Konstansok
;-------------------------------;
SD_INT          EQU     0FFH    ;Kommunikációs interrupt
                                ; Funkciókódok
SND_START       EQU     00H     ;Hangsorozat megszólaltaltása
SND_CANCEL      EQU     01H     ;Megszólaltatás felfüggesztése
SND_ASTST       EQU     02H     ;Státusz lekérdezése
;-------------------------------;
; Makródefiníciók
;-----------------------------------------;
SPKCLL          MACRO   CODE, PAR_ADDR  ;Speaker driver hívása
        IFNB    <PAR_ADDR>              ;;
                LES     BX, PAR_ADDR    ;;Opcionális paraméter
        ENDIF                           ;;
                MOV     AH, CODE        ;;Funkciókód
                INT     SD_INT          ;;Interrupt-hívás
        ENDM                            ;
;-----------------------------------------;
; Adatterületek
;-------------------------------------------------;
DATA    SEGMENT PARA    PUBLIC  'DATA'          ;
MUSIC_TEST      DD      MUSIC_STRING            ;
MUSIC_STRING    LABEL   WORD                    ;
   DW  TIM_SEC/2, SND_C,  TIM_SEC/4, SND_PAUSE  ;Ez egy C-dúr
   DW  TIM_SEC/2, SND_D,  TIM_SEC/4, SND_PAUSE  ; skála, a
   DW  TIM_SEC/2, SND_E,  TIM_SEC/4, SND_PAUSE  ; hangok kb.
   DW  TIM_SEC/2, SND_F,  TIM_SEC/4, SND_PAUSE  ; fél másod-
   DW  TIM_SEC/2, SND_G,  TIM_SEC/4, SND_PAUSE  ; percesek,
   DW  TIM_SEC/2, SND_A,  TIM_SEC/4, SND_PAUSE  ; közben negyed
   DW  TIM_SEC/2, SND_H,  TIM_SEC/4, SND_PAUSE  ; másodperc
   DW  TIM_SEC/2, SND_C1, TIM_SEC/4, SND_PAUSE  ; szünettel
   DW  0, 0                                     ;Sorozat vége
DATA            ENDS                            ;
```

```
;-------------------------------------------------------------;
; Stack terület                                               ;
STACK   SEGMENT PARA    STACK   'STACK'                       ;
        DW      100H    DUP     ( ? )                         ;
STACK   ENDS                                                  ;
;-------------------------------------------------------------;
; Kódterület
;-------------------------------------------------------------;
CODE    SEGMENT PARA    PUBLIC  'CODE'                        ;
        ASSUME  CS:CODE                                       ;
START:                                                        ;
                MOV     AX, SEG DATA                          ;DATA szegmens
                MOV     DS, AX                                ; beállítása
        ASSUME  DS:DATA                                       ;
                SPKCLL  SND_START, MUSIC_TEST                 ;A hangsor
SD_TEST_L:                                                    ; elindítása
                DOSCLL  DOS_STRDCH                            ;Közben csi-
                CMP     AL, 27                                ; nálunk
                JE      SD_TEST_Q                             ; valamit,
                MOV     DL, AL                                ; ami lát-
                DOSCLL  DOS_STWRCH                            ; szik
                JMP     SD_TEST_L                             ;(ciklusban)
SD_TEST_Q:                                                    ;
                SPKCLL  SND_CANCEL                            ;
                EXIT    0                                     ;Kilépés
;-------------------------------------------------------------;
CODE    ENDS
        END     START
```

A hívó program elindít egy hangsort, majd ciklusban olvassa a standard input file-t, és mindent visszaír a standard outputra. Ha valaki egy ESC karaktert ad be a standard inputról, akkor megszakítja a zenét, és kilép. Ezzel szemléltetjük, hogy a zenecsinálás valóban a háttérben történik.

VIII. Lemezkezelés

A lemezkezelés a ROM BIOS felhasználásának programozástechnikai szempontból nem bonyolult, de eléggé veszedelmes területe. A legtöbb perifériadriver közvetlen hívása teljesen megengedhető és törvényes dolog. Ezalól az MS-DOS operációs rendszerben csak a lemezdriver a kivétel: a driver felett (hiszen a lemez file-orientált periféria) még ott "lakik" a File Service Routines programcsomagja, amely file-szerűen kezeli a lemezen levő információt. Ha törvényesen akarjuk elérni az adatokat, akkor természetesen csak MS-DOS funkciókat hívhatunk. A ROM BIOS funkciók közvetlen hívása a File Service Routines megkerülését jelenti. Erre számos esetben lehet szükség; sose felejtsük el azonban azt, hogy nagy bajt a mágneses adathordozók rongálásával okozhatunk. Ha tehát közvetlen lemezkezelésre szánjuk magunkat, igen nagy körültekintéssel járjunk el!

Amit a fentiekben elsősorban a floppy-diskre gondolva írtunk, hatványozottan igaz a Winchesterre. Az IBM XT és AT gépek gyakorlati felhasználása során szinte minden lényeges információt a Winchesteren tárolunk (operációs rendszer, kisebb-nagyobb programcsomagok, saját fejlesztésű programjaink stb.). A munka során tehát állandóan szükségünk van a Winchester tartalmára. Bizonyos szempontból a merev lemez kevésbé értékes, mint a floppy, mert kultúrember természetesen minden lényeges dolgot diskette-re vagy (ha van neki ilyen) streamerre ment. Ezek szerint a Winchester tartalmára nem kell annyira vigyázni, mint a floppy-kéra, hiszen minden lényeges információ pótolható. Ez igaz, viszont a Winchesteren nem is maga az információ, hanem az információ struktúrája nagyon lényeges; a helyesen felépített directory-rendszer nagyban segítheti munkánkat. Egy ilyen struktúra teljes újjáépítése többórás procedúra is lehet.

Még nagyobb fontossága lehet azonban a Winchesteren kialakított partícióknak. Mint tudjuk, a Winchestereken több, egymástól függetlenül kezelhető, és különféle operációs rendszert tartalmazó partíciót hozhatunk létre. Gyakori eljárás például az, hogy két részre osztják a Winchestert. Az egyik MS-DOS-t, a másik például XENIX-et tartalmazhat (hogy a XENIX operációs rendszer egyes változatai történetesen két további partíciót alakítanak ki, itt most ne firtassuk).

Ha direkt módon kezeljük a Winchestert, akkor fennáll az a veszély, hogy elrontjuk a lemez olyan területeit, amelyhez semmi közünk; belerondíthatunk a másik partícióba, sőt, a Winchester "master boot record"-jába, a partíciótáblába is. Ez pedig valószínűleg jóvátehetetlen vagy alig javítható károkat fog

okozni - odavész a teljes XENIX partíció stb. Ezért semmiképpen nem ajánlatos a Winchester közvetlen kezelése. Az alább megadott információkat sokkal inkább tájékoztató jellegűnek, mint valódi programozási segítségnek szántuk.

Az intelmek után lássuk a konkrétumokat! Bevezetésképpen a lemez fizikai és logikai felépítéséről beszélünk egy keveset. A floppy-disk-et vesszük alapul, a merevlemez, a WINCHESTER felépítése ettől nem tér el döntő módon; a legfontosabb eltérésekre külön utalunk.

VIII.1. A lemezek fizikai felépítése

A lemez egy kétdimenziós felületű adathordozó, szemben az egydimenziós mágnesszalaggal. A lemezen az előkészítés során sávokat, angol szóval "track"-eket alakítanak ki. A sáv egy körgyűrű a lemez felületén. A sávokat szektorokra osztják. Különféle operációs rendszerek különböző módon formázzák meg az általuk használt diskette-et. Vannak olyanok, melyek a lemez belső sávjait kevesebb szektorra osztják, hogy az információ sűrűsége ne legyen olyan nagy.

A legtöbb esetben az egyszerűbb kezelés kedvéért minden sávon azonos számú és azonos hosszúságú szektorokat alakítanak ki. Ez persze azzal jár, hogy a szektorok számát és hosszát úgy kell megválasztani, hogy még a legbelső, tehát legsűrűbben írt szektorok adatbiztonsága is elegendően nagy legyen. Ebből azonban az következik, hogy a külső, ritkábban írott sávok kapacitását nem használjuk ki teljes egészében. Valószínű, hogy az ebből adódó veszteségek kisebbek, mint az a nyereség, amely az egyszerűbb, biztonságosabb (és talán gyorsabb) lemezkezelés eredménye.

Az IBM PC/XT floppyvezérlője lehetővé teszi azt, hogy a különböző sávokat különböző módon formázzuk meg, sőt azt is, hogy egy adott sávon különböző hosszúságú szektorokat hozzunk létre. Persze azt ne várjuk ezek után, hogy az MS-DOS operációs rendszer el tujda olvasni a lemezünket - de lehet, hogy éppen ez a cél! A programok másolás elleni védelmére az egyik módszer az lehet, hogy a lemez egy-két belső sávját másképpen formázzuk meg, s azután ezekre a sávokra fontos információkat írunk fel.

A floppy-disken 40 vagy 80 sávot alakíthatunk ki. A diskette szokásos formázása során minden sávon ugyanannyi, 9 szektort alakítunk ki, és minden szektor 512 byte rögzítésére alkalmas. A lemeznek két felülete van, így 2x40x9x512 byte befogadására képes; ez összesen 360 Kb-nyi információ befogadására elegendő.

Megjegyzés: az IBM AT-n használatos korszerűbb drive-ok (de csak elég jó minőségű, HD jelzéssel ellátott lemezeket használva) képesek ennél sokkal többet is befogadni: ha 80 sávosra formázzuk a diskette-et, akkor 720 Kb fér el; ha egy sávra 15 szektort helyezünk el, akkor a 80 sávosra formázott floppy 1200 Kb befogadására alkalmas.

A merev lemez ebből a szempontból a következőkben tér el a diskette-től: egy 10 Mb-os Winchester három egymás fölötti lemezből állhat, melyeknek két szélső (legalsó és legfelső) oldalát nem használják, így négy oldal marad. Minden oldalon ugyanannyi sávot alakítanak ki gyárilag. Az egymás fölötti sávokat együttesen "cylinder"-nek, "hengernek" nevezik (a továbbiakban a legelterjedtebb cilinder elnevezést használjuk, még diskette esetében is, ahol a cilinder két sávot tartalmaz). A 10 Mb-os Winchesteren 306 cilinder kialakítására van lehetőség; ha a lemez 21 Mb kapacitású, akkor a cilinderek száma általában 615. Az egyes cilinderek oldalanként 17 szektort tartalmaznak, a szektorok pedig 512 byte-ot képesek "megjegyezni".

A lemezen egyidejűleg legalább egy szektort tudunk írni vagy olvasni, ez a legkisebb címezhető egység. A szektor címzése kétféleképpen történhet, "logikailag" és "fizikailag". A fizikai címzés egy sajátos háromdimenziós koordinátarendszerben történik. Az első koordináta azt adja meg, melyik oldalt, a második, melyik sávot, s a harmadik, hogy melyik szektort címezzük meg. Az oldalak és sávok számozása nullától, a szektoroké egytől kezdődik. A szektorok logikai címe egy sorszám, amely floppy-diskette-en a következőképpen alakul ki: a 0. oldal 0. sávjának 1. szektora a lemez első szektora, ennek sorszáma 0. A következő szektorok a 0. oldalon helyezkednek el, a 9. szektorig. A logikailag 10. szektor az 1. oldal 0. sávjának 1 szektora; a számozás eszerint folytatódik. Merevlemezen értelemszerűen több oldal, azokon több sáv és több szektor van, de a logika ugyanaz. A továbbiakban a fizikai címzést BIOS-, a logikait pedig MS-DOS címzésnek, mert a BIOS, illetve az MS-DOS használja őket.

A szektorok belső felépítésének ismerete is igen érdekes és hasznos lehet. Az adatbyte-ok mellett minden szektor tartalmaz belső információkat is. Ezek: egy négy byte-os vezérlőblokk és egy CRC ("Cyclic Redundancy Check"). Ez utóbbi a szektor ellenőrzésére szolgál. A "verify", azaz ellenőrzési eljárás során ezt a CRC-t vizsgálják (hiszen az adatok helyességét, az adatok hiteles másolata hiányában, általában nem lehet ellenőrizni).

C(ylinder)	H(ead)	R(ecord)	N(umber)
cilinder száma (sáv száma)	fej száma (oldal)	Rekordszám (szektor)	Byte-ok száma szektoronként

A szektorazonosító felépítése

Amennyiben a lemezre felírt információkat megkeverjük, és a lemezt saját driverünk segítségével olvassuk, ezzel is sok borsot törhetünk azok orra alá, akik a programunkat lopás és nem vásárlás útján szándékoznak megszerezni.

Megjegyezzük még, hogy az "N" mező nem a valódi értéket, hanem annak egy kódját tartalmazza. A kódolás a következő:

Byte-ok száma	N
128 byte	0
256 byte	1
512 byte	2
1024 byte	3

vagyis a hossz $128 * 2^N$

A szektorhossz értelmezése

A Winchester szektorai a fenti információkon kívül még egy négybyte-os értéket is tartalmaznak, amelyet ECC-nek (Error Correct Code) nevez az angol szakirodalom. Ez egy olyan érték, amelynek segítségével bizonyos adathibák javíthatók. Ha az ECC felhasználására van szükség, akkor a Winchester-kezelő program az olvasás (ellenőrzés) után egy speciális hibakóddal (11H) tér vissza (lásd "A BIOS lemezkezelő funkciói" fejezetet); ebben az esetben a beolvasott adatokat bátran használhatjuk, de ne felejtsük el: a lemez nem egészen hibátlan.

VIII.2. A lemezek logikai felépítése

Az MS-DOS által formázott lemezek logikai struktúrája elvileg szintén független attól, hogy milyen típusú lemezzel van dolgunk. Minden adatot file-okba ("állományokba") osztva helyezünk el a lemezen. A file két fő részből áll. Az egyik rész tartalmazza a file-ba rögzített adatokat, a másik rész pedig a file azonosítását és elérését lehetővé tevő "directory-entry", egy bejegyzés a lemez directory-jában, tartalomjegyzékében.

A lemezterület logikailag a következő négy részre oszlik: a "Boot-record", a "File Allocation Table", a "Directory-terület" és az adatterület. Az utolsó három részletes leírását lásd a 2. kötetben ("A programozó és az MS-DOS").

VIII.3. A lemezek BOOT rekordja

A BOOT rekord floppy-disken a fizikailag első szektor (Winchesteren ott van előtte az ún. Master Boot Record, a partíciótábla). Ez a szektor tartalmazza azokat a legfontosabb információkat, amelyek a lemez kezeléséhez szükségesek. Ezen adatok mellett egy programot találhatunk a BOOT rekordon, amely (ha a lemez tartalmaz operációs rendszert) betölti a rendszert alkotó file-okat a megfelelő területre.

Mielőtt leírnánk a BOOT rekordról rendelkezésünkre álló információkat, felhívjuk az Olvasó figyelmét arra, hogy ezek az ismeretek BOOT rekordok direkt elolvasásából és (lehetőség szerint teljes) megfejtéséből származnak.

A BOOT rekordot öt fő részre oszthatjuk: az első három byte-on egy ugró utasítást találunk, amely a betöltés után átadja a vezérlést a valódi betöltő programra. Ezután nyolc byte fenntartott terület következik, ide kulcsszavakat ír a FORMAT program (pl: "IBM 3.2" [két szóközzel!]).

> Megjegyzés: a PCTOOLS 4.11 verzióval formázott diskette ezen a ponton a "Pc tools" szöveget tartalmazza...

A harmadik rész hordozza a lemez kezelése szempontjából legfontosabb információkat: innen olvashatjuk ki, hogy az adott lemez milyen kapacitású és a logikai egységek a formázáskor milyen méretet kaptak. A negyedik rész tartalmazza azt a programot, amely a ROM BIOS driverek közvetlen hívásával betölti az IBMBIO.COM és az IBMDOS.COM programokat. Végül a BOOT rekord utolsó három byte-ján ismét kulcsfontosságú információt találunk: értéke "00 55 AA". Ez a három byte látszólag teljesen semmitmondó. Közülük az első (ami floppy-disken mindig 0) a lényeges: ez az érték ugyanis Winchester esetén 80H - a lemezegység kódja! A betöltőprogram innen olvassa be a driver hívása előtt DL-be ezt a lényeges adatot.

Ismerkedjünk meg most részletesen ezekkel a dolgokkal (egy BOOT rekord a fejezet végén olvasható azzal az "alávaló" módszerrel együtt, ahogyan a tartalmához hozzá lehet jutni). Az első két rész tartalma számunkra e pillanatban szinte érdekte-

len. A legfontosabb harmadik rész, az első két résszel együtt (tapasztalati tények alapján) az alábbi struktúrában foglalható össze:

```
BOOTREC         STRUC

BJUMP       DB    3 DUP (0)  ;JMP a belépésre    offs Fl-d  Winch.
BRESV       DB    8 DUP (0)  ;Fenntartott ter.
SECTSIZ     DW    0  ;Szektorok hossza (byte)     11  0200H 0200H
CLUSTSIZ    DB    0  ;Clusterek hossza (sz)       13    02H   04H
BRECSIZ     DW    0  ;BOOT rekord hossza (sz)     14  0001H 0001H
FATNUMB     DB    0  ;FAT-ok száma (1 vagy 2)     16    02H   02H
RDIRSIZ     DW    0  ;ROOT directory hossza (bj)  17  0070H 0200H
SECTNUMB    DW    0  ;Szektorok száma             19  02D0H A307H
DISKCODE    DB    0  ;Disktípus azonosító         21    FDH   F8H
FATSIZ      DW    0  ;FAT hossza (szektor)        22  0002H 0029H
TRACKSIZ    DW    0  ;Sáv hossza (szektor)        24  0009H 0011H
SIDENUMB    DW    0  ;Oldalak száma               26  0002H 0004H
PARTBSIZ    DW    0  ;Partíciótábla hossza        28  0000H 0011H
DISKCD      DB    0  ; ezt bizony nem tudjuk      30    00H   80H
PAD         DB    ? DUP (0)  ;Kitöltő byte-ok
BPARTAB     DB   11 DUP (0)  ;Speciális disk-paraméterek helye

BOOTREC         ENDS
```

Ebből a struktúrából ki lehet olvasni a második rész legfontosabb elemeit. A megjegyzések végén olvashatjuk a struktúra megfelelő elemének offsetjét, valamint egy olyan értéket, amelyet egy szokásos módon formázott, kétoldalas, negyven sávos, sávonként kilenc szektoros floppy-disk BOOT rekordján találunk; emellett pedig egy Winchester BOOT rekordjának megfelelő értéket.

- BJUMP (3 byte) a betöltő programra adja a vezérlést; tartalma általában egy

```
JMP     ...
NOP
```

  utasítássorozaat, mert a JMP közeli és így két byte-os; ezt pedig a MASM mindig így fordítja le;
- BRESV (8 byte) a formázást végző program aláírása;
- SECTSIZ (szó) a szektorok byte-okban számított hosszát tartalmazza (ez is mutatja, hogy az MS-DOS csakis azonos szektorhosszúságú lemez kezelésére képes);
- CLUSTSIZ (byte) a lemez clustereinek hosszát tartalmazza, szektorokban számítva (a cluster, allokációs egység, a fileok számára lefoglalt lemezterület nagysága, lásd 2. kötet);

- BRECSIZ (szó) a BOOT rekord hossza szektorokban (értéke általában 1);
- FATNUMB (byte) a FAT-ok száma (az elérhető leírások szerint értéke egyetlen lemeztípus esetén lesz csak 1: az egyoldalas, sávonként 8 szektoros lemez esetén);
- RDIRSIZ (szó) a ROOT directory hossza bejegyzésekben (figyeljük meg, hogy floppy-n ez 70H, azaz 112, Winchesteren 200H, azaz 512);
- SECTNUMB (szó) elem a lemezen levő összes szektort tartalmazza. Ez ismét egy érdekes szám. Floppy-disken még csak stimmel a 2D0H érték (720), de mit kezdjünk a Winchester A307H értékével, amely decimálisra átszámítva 41735? A számítások alapján adódó érték (4 oldal, mindegyiken 615 sáv, melyeken 17 szektorunk van) 41820. Azt a Winchestert, amelynek BOOT rekordjáról a fenti adatokat "beszereztem", előkészítése idején (nem tudom, hogy miért) 615 helyett 614 sávosra formázták, és a Winchesterek 0 oldala 0 sávja a "Master BOOT record", magyarul a lemez partíciótáblája. Tehát így alakult ki a fenti érték: 4*614*17=41752 lenne várható, ami ennél 17-el több, éppen a partíciótábla hosszával;
- DISKCODE (byte) a lemez azonosítókódját tartalmazza (értéke például FFH, azaz -1 az egyoldalas, 8 szektoros lemez esetén, és F8H a Winchestereken);
- FATSIZ (szó) a File Allocation Table hossza szektorokban. Ebből az értékből, valamint a BOOT rekord hosszából (BRECSIZ) tudja meg a rendszer, hogy hol is kezdődik a ROOT directory. A ROOT directory bejegyzésekben számított hosszából kaphatjuk meg az első adatszektor logikai (MS-DOS számozás szerinti) helyét. Ez éppen tizenhatszorosa a szektorokban számított hossznak, mert egy szektor tizenhat directory bejegyzést tartalmazhat;
- TRACKSIZ (szó) a sávok szektorokban számított hosszát tartalmazza;
- SIDENUMB (szó) az odalak száma;
- PARTBSIZ (szó) a partíciótábla hosszúsága (floppy-n 0, Winchesteren 17 a szokásos értéke);
- DISKCD (byte) a lemezegység kódja akkor, mikor a lemezt betöltésre használjuk (floppy-n 0H, Winchesteren 80H);
- PAD: ez a terület tapasztalataim szerint mindig néhány 0 értékű byte-ot jelentett, és (minthogy a töltő program egyáltalán nem hivatkozik itteni címekre) valószínűleg pusztán helyfoglaló szerepe van, ezt mutatja a kérdőjeles hossz;
- BPARTAB (11 byte) speciális diskette-paraméterek helye. Szerkezete megegyezik az 1EH interrupt vektor által címzett diskette-paraméter tábláéval, amely a diskette kezeléséhez szük-

séges fizikai paramétereket szabja meg. Bármely értékét megadhatjuk. Betöltéskor a nem nulla értékek érvényben maradnak, a nulla értékek az alapértelmezett paraméterértékekkel íródnak felül. Kiolvasott értékei:

00 00 00 00 12 00 00 00 00 01 00

azaz mindenütt az alapértelmezés, kivéve a sávhosszt, melyet a betöltés idejére 12H-ra (18-ra) emel, valamint a fejcímzési időt, ami igen rövid (hiszen a file-ok közel vannak).

Figyelmesen tanulmányozva a struktúrát, rájöhetünk, miért nem képes az MS-DOS 40 Mb-os Winchesterek kezelésére: szabványos formázás szerint ugyanis több szektora lenne, mint amennyit a SECTNUMB befogadni képes; nagyobb szektorhosszat pedig azért nem használhatunk, mert az MS-DOS minden file-kezelő rutinja 512 byte-os szektorhosszra van felkészülve. Ezeket a hatalmas lemezeket ezért particionálva használják.

A fejezet lezárásaként álljon itt egy teljes BOOT rekord DEBUG által előállított képe és felbontása. Az eljárás egyébként a következő volt: a DEBUG programot behívtam úgy, hogy standard output file-ját átirányítottam egy BOOTREC.TXT nevű becsületes szövegfile-ba. Ezután (kissé vakon kellett dolgozni) a következő parancsokat gépeltem be:

```
-L DS:200 0 0 1      BOOT rekord beolvasása
-D 200:3FF           tartalmának hexa megjelenítése
-U 243:36D           A töltőprogram kifejtése vált kezdetétől
```

Mivel a standard outputot átirányítottam, a DEBUG összes kiírása a BOOTREC.TXT file-ba ment, ahonnan már szövegszerkesztővel tovább lehet menni. A program visszafejtése elég jól követhető ugyan, mégis azt hiszem, hogy abszolút érdektelen; annál hasznosabb lehet azonban egy trükk, amely segített az előbbi figura végrehajtásában: éspedig a DOSEDIT program használata. Nem kockázatmentes lépés egy lemez BOOT rekordját elolvasni; ha az ember véletlenül (minthogy vakon gépel) elüt valamit, esetleg bajt okozhat magának. A DOSEDIT azonban hajlandó megjegyezni azokat a parancsokat, amelyeket bármely olyan program számára gépeltünk be, amely a 0AH MS-DOS funkciót használja beolvasásra. A jó öreg DEBUG pedig ilyen program: az MS-DOS által neki adományozott standard input file-t használja beolvasásra, a standard output file-t kiírásra (ez utóbbin múlik egyébként az egész gazság); és a DOSEDIT kedve szerint kezeli az inputot. Azt tettem tehát, hogy először végrehajtottam a három parancsot

"látható" módban (a standard output átirányítása nélkül), majd kiléptem a programból. Ezután újraindítottam a DEBUG-ot, most már átirányítva; a lefelé nyíl segítségével elővettem a DOSEDIT bufferéből az ott előre elhelyezett parancsokat, és végrehajtattam őket; így nem kockáztattam meg, hogy elgépelek valamit (még rágondolni is rossz, mi történik, ha "L" helyett netán "W"-t ütünk...). Mielőtt elmerülnénk a BOOT rekord programjának "élvezetében", olvassuk el a ROM BIOS betöltőprogramjának mindössze néhány utasításból álló kódját!

```
                STI
                SUB     AX, AX          ;AX = 0
                MOV     DS, AX          ;
        ;
        ; 1EH interrupt vektor alapállapotba hozása
        ;       (ronda, de ... MS-DOS még nincs !!)
                MOV     WORD PTR DISK_POINTER, OFFSET DISK_BASE
                MOV     WORD PTR DISK_POINTER + 2, CS
        ;
        ; Rendszertöltés diskette-ről;
        ;       CX-ben a próbálkozások száma
                MOV     CX, 4
H1:
                PUSH    CX
                MOV     AH, 0           ;Diskette-driver
                INT     13H             ; előkészítése
                JC      H2              ;Hiba - újra próbáljuk
                MOV     AX, 0201H       ;Olvasunk 1 szektort...
                SUB     DX, DX          ;
                MOV     ES, DX          ; a 0000:7C00H címre...
                MOV     BX, 7C00H       ;
                MOV     CX, 0001H       ; a 0 sáv 1 szektortól
                INT     13H
H2:
                POP     CX              ;Számláló vissza
                JNC     H4              ;Sikeres - (oda)ugrunk
                LOOP    H1              ;Nem - próbáljuk újra
        ;
        ; Egyáltalán nem sikerült,
        ;       ugrás a ROM BASIC-re (ha van, akkor jó...)
H3:
                INT     18H
H4:
                JMP     0000:7C00H      ;Megvan - továbbtöltés
```

A fenti programot senki ne vegye készpénznek, mert szintaktikusan se jó; az abszolút címeket a gépikódból lehetett kiolvasni stb. A lényeget azonban jól érzékelteti. A diskette driver előkészítése után a ROM BIOS néhányszor megkísérli a BOOT rekord beolvasását a 0000:7C00H címre, melynek neve BOOT_LOCN, betöltési pozíció. Ha sikerül, akkor átadja ide a vezérlést; ha nem, akkor aktivizálja a ROM BASIC-et. Az IBM XT ROM BIOS-ának betöltőprogramja, mely Winchesterről is megkísérli a műveletet, alig tér el a fenti kódtól; némi előkészület után megkísérli előbb a 0 kódú lemezről (A:-ról), majd Winchesterről (C:-ről) a betöltést; a fenti szekvencia, csaknem változatlanul, kétszer szerepel. Most pedig lássuk a betöltést végző programot (a lemez, ahonnan ezt "lecsentem", az MS-DOS 3.30 verziójával formázott kétoldalas, kilenc sávos floppy-disk volt). Első lépésként az első három byte, majd a lényeg visszafejtése következik:

```
7C00 EB34          JMP  7C36              ;Ugrás a valódi
7C02 90            NOP                    ; betöltőprogramra
                   ...                    ;Struktúrák helye
7C36 FA            CLI                    ;Interrupt letiltás
7C37 33C0          XOR  AX,AX             ;
7C39 8ED0          MOV  SS,AX             ;Stack közvetlenül
7C3B BC007C        MOV  SP,7C00H          ; e program alatt
7C3E 16            PUSH SS                ;
7C3F 07            POP  ES                ;ES is 0000H-ra mutat
7C40 BB7800        MOV  BX,0078H          ;1EH interrupt vektor
7C43 36 C537       LDS  SI,SS:[BX]        ; betöltése DS:SI-be
7C46 1E            PUSH DS                ;Eredeti 1EH interrupt
7C47 56            PUSH SI                ; vektor mentése
7C48 16            PUSH SS                ;1EH interrupt vektor
7C49 53            PUSH BX                ; címének mentése
7C4A BF2B7C        MOV  DI,7C2BH          ;DI - spec. par. tab
7C4D B90B00        MOV  CX,000BH          ;Par. tábla hossza
7C50 FC            CLD                    ;
7C51 AC            LODSB                  ;Eredeti par. AL-be
7C52 26 803D00     CMP  BYTE PTR ES:[DI],00     ;Nincsenek spec
7C56 7403          JZ   7C5BH             ; paraméterek?
7C58 26 8A05       MOV  AL,ES:[DI]        ;De vannak, maradnak
7C5B AA            STOSB                  ;Part -> BOOT rekord
7C5C 8AC4          MOV  AL,AH             ;AL-be 0 !
7C5E E2F1          LOOP 7C51H             ;Ismétlés 11-szer
7C60 06            PUSH ES                ;
7C61 1F            POP  DS                ;DS -> 0000H
7C62 894702        MOV  [BX+02],AX        ;1EH v. szegmens:0000H!
7C65 C7072B7C      MOV  WORD PTR [BX],7C2BH
```

```
        ;Az eredeti diskette-paramétertábla helyett a
        ;BOOT rekordbeli és az átmásolt "összefésülését"
        ;fogja használni
7C69 FB             STI
7C6A CD13           INT   13H                 ;AX=0 - driver elők.
7C6C 7267           JB    7CD5H               ;Sikertelen
7C6E A0107C         MOV   AL, [7C10H]         ;FAT-ok száma AL-be,
7C71 98             CBW                       ; majd AX-be
7C72 F726167C       MUL   WORD PTR [7C16H];Szorozva a hosszával,
7C76 03061C7C       ADD   AX,[7C1CH]          ; + partícióhossz,
7C7A 03060E7C       ADD   AX,[7C0EH]          ; + BOOT rekord hossza
7C7E A33F7C         MOV   [7C3FH],AX          ;Elmenti programra!!
7C81 A3377C         MOV   [7C37H],AX          ;Elmenti programra!!
7C84 B82000         MOV   AX,0020H            ;Dir. bejegyzés hossza
7C87 F726117C       MUL   WORD PTR [7C11H]; szorozva számával
7C8B 8B1E0B7C       MOV   BX,[7C0BH]          ;Szektorhossz BX-ben
7C8F 03C3           ADD   AX,BX               ;AX:dir hossza+1 sz b
7C91 48             DEC   AX                  ;
7C92 F7F3           DIV   BX                  ;Directory hossza (sz)
7C94 0106377C       ADD   [7C37H],AX          ;Adatter. kezdőszektor
7C98 BB0005         MOV   BX,0500H            ;Beolvasási cím
7C9B A13F7C         MOV   AX,[7C3FH]          ;Directory kezdőszektor
7C9E E89F00         CALL  7D40H               ;Lebontja (old,sáv,sz)
7CA1 B80102         MOV   AX,0201H            ;Beolvasás következik
7CA4 E8B300         CALL  7D5AH               ; behozza dir-t
7CA7 7219           JB    7CC2H               ;Sikertelen, lemezcsere
7CA9 8BFB           MOV   DI,BX               ;Megnézi, IBMBIO.COM
7CAB B90B00         MOV   CX,000BH            ; ott van-e, ahol
7CAE BED67D         MOV   SI,7DD6H            ; kell neki!
7CB1 F3A6      REPZ CMPSB                     ;
7CB3 750D           JNZ   7CC2                ;Nem rendszerlemez!
7CB5 8D7F20         LEA   DI,[BX+20H]         ;Köv. dir.-bejegyzés
7CB8 BEE17D         MOV   SI,7DE1H            ;Megnézi, IBMDOS.COM
7CBB B90B00         MOV   CX,000BH            ; ott van-e, ahol
7CBE F3A6      REPZ CMPSB                     ; kell neki!
7CC0 7418           JZ    7CDAH               ;Igen, folytatja;
7CC2 BE777D         MOV   SI,7D77H            ;"Non-system" üzenet
7CC5 E86A00         CALL  7D32H               ; ki, és
7CC8 32E4           XOR   AH,AH               ; lemezcserére várunk
7CCA CD16           INT   16H                 ; egy leütéssel
7CCC 5E             POP   SI                  ;1EH vektor címe vissza
7CCD 1F             POP   DS                  ;
7CCE 8F04           POP   [SI]                ;1EH vektor tartalma
7CD0 8F4402         POP   [SI+02H]            ; vissza, és
7CD3 CD19           INT   19H                 ;
```

```
        ;Sikertelen a driver és a lemezegység
        ;előkészítése; "Disk boot failure"
        ;üzenet ki, majd újabb betöltési kísérlet
7CD5 BEC07D         MOV  SI,7DC0H
7CD8 EBEB           JMP  7CC5H
        ;Folytatódik a betöltés
7CDA A11C05         MOV  AX,[051CH]       ;DX:AX = IBMBIO.COM
7CDD 33D2           XOR  DX,DX            ; hossza
7CDF F7360B7C       DIV  WORD PTR [7C0BH];Hány szektor?
7CE3 FEC0           INC  AL               ;Eggyel több kell
7CE5 A23C7C         MOV  [7C3CH],AL       ;Szektorszám programra!
7CE8 A1377C         MOV  AX,[7C37H]       ;Adatterület kezdőcím
7CEB A33D7C         MOV  [7C3DH],AX       ;Kimentés programra!
7CEE BB0007         MOV  BX,0700H         ;Buffer IBMBIO-nak
        ;Ez a szekvencia tölti be az IBMBIO.COM
        ;file-t, mely kötelezően folytonos és az adatterület
        ;legelején helyezkedik el
7CF1 A1377C         MOV  AX,[7C37H]       ;Következő szektor fel
7CF4 E84900         CALL 7D40H            ;Lebontja (old,sáv,sz)
7CF7 A1187C         MOV  AX,[7C18H]       ;Sávhossz AX (AL)-be
7CFA 2A063B7C       SUB  AL,[7C3BH]       ;Szektorszám levonása
7CFE 40             INC  AX               ; és eggyel több
7CFF 38063C7C       CMP  [7C3CH],AL       ;elértük a végét?
7D03 7303           JNB  7D08H            ;Igen,
7D05 A03C7C         MOV  AL,[7C3CH]       ;Betöltjük megint
7D08 50             PUSH AX               ;
7D09 E84E00         CALL 7D5AH            ;Beolv. a köv szektort
7D0C 58             POP  AX
7D0D 72C6           JB   7CD5H            ;Sikertelen, "failure"
7D0F 28063C7C       SUB  [7C3CH],AL       ;Vége?
7D13 740C           JZ   7D21H            ;Igen, a többi IBMBIO!
7D15 0106377C       ADD  [7C37H],AX       ;
7D19 F7260B7C       MUL  WORD PTR [7C0BH];
7D1D 03D8           ADD  BX,AX            ;Buffercím+szektorhossz
7D1F EBD0           JMP  7CF1H            ;Olv. a köv. szektort
        ;IBMBIO bent van a tárban, innen már ő fut
        ;tovább!
7D21 8A2E157C       MOV  CH,[7C15H]       ;Disktípus CH-ba
7D25 8A16FD7D       MOV  DL,[7DFDH]       ;Drive kódja DL-be
7D29 8B1E3D7C       MOV  BX,[7C3DH]       ;Adatrész BX-be és
7D2D EA00007000     JMP  0070:0000H       ;Jöjjön IBMBIO.COM!
        ;Ez a rutin a DS:SI által címzett ASCIZ-stringet
        ;kiírja a képernyőre
7D32 AC             LODSB                 ;Következő karakter
7D33 0AC0           OR   AL,AL            ; nulla?
```

```
7D35 7422          JZ   7D59H          ;Igen, ugrás RET-re
7D37 B40E          MOV  AH,0EH         ;Write teletype
7D39 BB0700        MOV  BX,0007H       ;0 lap, feketén fehér
7D3C CD10          INT  10H            ;Video driver
7D3E EBF2          JMP  7D32H          ;Ciklus
      ;Ez a rutin a logikai szektorszámot (AX) lebontja
      ;(oldal,sáv,szektor) alakra;
      ;oldal: 7C2AH (b), sáv:7C39H (szó), szektor: 7C3BH (b)
7D40 33D2          XOR  DX,DX          ;DX:AX - szektorszám
7D42 F736187C      DIV  WORD PTR [7C18H]; osztva szektorhosszal
7D46 FEC2          INC  DL             ; maradék + 1 !
7D48 88163B7C      MOV  [7C3BH],DL     ;elmenti programra!
7D4C 33D2          XOR  DX,DX          ;DX:AX - "log cilinder"
7D4E F7361A7C      DIV  WORD PTR [7C1AH]; osztva oldalszámmal
7D52 88162A7C      MOV  [7C2AH],DL     ;maradék: oldal száma
7D56 A3397C        MOV  [7C39H],AX     ;sáv száma
7D59 C3            RET
      ;Ez a rutin beolvasást végez az ES:DX területre
      ;oldal: 7C2AH (b), sáv: 7C39H (szó), szektor: 7C3BH (b)
7D5A B402          MOV  AH,02H         ;Funkciókód
7D5C 8B16397C      MOV  DX,[7C39H]     ;Sáv szám DX-be
7D60 B106          MOV  CL,06H         ;9-10. két bit rotá-
7D62 D2E6          SHL  DH,CL          ; lása legfelülre és
7D64 0A363B7C      OR   DH,[7C3BH]     ; VAGYolása szektorral
7D68 8BCA          MOV  CX,DX          ;
7D6A 86E9          XCHG CH,CL          ;CH:sáv, CL:(sáv OR sz)
7D6C 8A16FD7D      MOV  DL,[7DFDH]     ;Drive kódja be
7D70 8A362A7C      MOV  DH,[7C2AH]     ;Oldal száma be
7D74 CD13          INT  13H
7D76 C3            RET

7D77    DB      CR,LF,'Non-System disk or disk error'CR,LF
7D98    DB      'Replace and strike any key when ready'CR,LF,0
7DC0    DB      CR,LF,'Disk boot failure',CR,LF,0
7DD6    DB      'IBMBIO  COM'
7DE1    DB      'IBMDOS  COM'
```

Ezt a programlistához hasonló valamit természetesen a DEBUG által visszafejtett kód kozmetikázásával hoztam létre. Innen is töröltem a szegmensértékeket, ahol csak tudtam, megjegyzéssel láttam el a visszafejtett kódot, végül pedig a program utolsó szakaszán levő hibaüzenetek és file-nevek szövegét alkalmassá tettem emberi fogyasztásra. Minden egyéb pontosan, bitről bitre úgy van, ahogyan azt a DEBUG a BOOT rekordból kiolvasta és a képernyőn megjelenítette.

VIII.4. A BIOS lemezkezelő funkciói

Interrupt-sorszám: 13H
Javasolt interrupt-név: BS_DISK

A diskkezelő driver kommunikációs interruptjának hívására javasolt makró:

```
DSKCLL  MACRO   CODE                    ;
                MOV     AH, CODE        ;;
                INT     BS_DISK         ;;
        ENDM                            ;
```

Eddigi szokásunktól eltérően ezúttal a "törvényes" utat tárgyaljuk előbb, s csak azután térünk ki a floppy-kontrollerek közvetlen programozására. Tesszük mindezt azért, mert előbb azokat az információkat szeretém megadni, amelyek általánosak, tehát a floppy-diskre éppenúgy igazak, mint Winchesterre. Ezek egyúttal azok az ismeretek, amelyek felhasználását (fejcsóválva bár, de) még megengedhetőnek tartom. A floppy-kontroller közvetlen programozása már komoly gyakorlattal és felkészültséggel rendelkező "varázslókat" kíván. A Winchester kontroller közvetlen programozásáról pedig e helyen szó se essék; ehhez csak az vegyen magának bátorságot, aki valóban csaknem mindent (de minden lényegeset) tud az operációs rendszerről, az alatta lévő hardware-ről és a Winchesterről.

A ROM BIOS eredeti floppy-drivere természetesen csak a floppyt tudja kezelni. A rendszer elindításakor azonban (ha van a gépben Winchesterkártya) a 13H interrupt vektor tartalmát úgy módosítják, hogy az nem az eredeti driverre, hanem a Winchester és a floppy vezérlésére egyaránt képes programcsomag belépési pontjára mutat, míg az eredeti driver címe a 40H interrupt vektorba kerül, és ott közvetlenül aktivizálható.

A Winchestert vezérlő programok a lemez vezérlőkártyáján található ROM területen vannak. Ez a terület szabvány szerint a C000H paragrafuscímre kerül a gép memóriájában. Ezért ROM BIOS külön fejezete a Winchester-driver, amely a közvetlen vezérlésen kívül egy másik "boot strap loader"-t is tartalmaz. A ROM BIOS előkészítő részei felülírják a 19 interrupt vektort, és a kezdeti rendszertöltést már ez a rutin fogja megkísérelni előbb a floppy-ról, majd a Winchesterről, s csak ezután ugrik végső kétségbeesésében a kútba, akarom mondani a ROM BASIC-be.

A floppy és a Winchester kezelése során természetesen figyelembe kell venni, hogy (bár logikai és fizikai felépítésük is hasonló) vannak bizonyos különbségek: a floppyn a sáv hossza 8,

9 vagy 15, a Winchestereken általában 17 szektor; több sáv van egy-oldalon (ez a Winchester típusától függ) és több oldal van. Egy szektor megcímzése a következő számhármassal történik:

(oldal,cilinder,szektor)

ahol a "cilinder" a sáv számát jelenti (floppy-disk-en 0-tól 39-ig vagy 79-ig, a legelterjedtebb 21 Megabyte-os Winchesteren 0-tól 614-ig terjedhet), az "oldal" a megcímezni kívánt oldal (író-olvasófej) száma (értéke 0 és 1, vagy 0 és 3 közé esik), a "szektor" a kívánt szektor sorszáma 1 (!) és 17 között.

A lemezkezelő driver képes egy meghívással több szektort is olvasni vagy írni. Floppy-diskek esetén szigorú megkötés azonban, hogy az egyszerre olvasott vagy írt szektoroknak azonos oldalon, egy sávon kell lenni. A floppy-disket, mint tudjuk, a DMA áramkör segítségével érjük el. A DMA sajátossága, hogy az általa megcímzett memóriaterület nem léphet át "kerek" 64 Kb-os határt, azaz a teljes megcímzett területnek a 00000H - 0FFFFH, az 10000H - 1FFFFH stb. határok közé kell esnie.

A Winchester driver szintén képes arra, hogy több sávot érjen el egy meghívásra; egyszerre legfeljebb százhuszonnyolc logikailag folytonos szektort olvashatunk vagy írhatunk.

A lemez kezelésére szolgáló kommunikációs interrupt használata a következő: mint minden korábbi esetben, az AH regiszter tartalmazza a funkciókódot. A DL regiszterben kell megadni a lemez kódját a következőképpen: ha az érték 80H-nál kisebb, akkor diskette-et, ha pedig nagyobb vagy egyenlő, akkor Winchestert címzünk. Tehát, ha a DL regiszterben 0-t vagy 1-et adunk meg, akkor az **A:** vagy a **B:** lemezt címezzük. Ha a DL-be 80H-t írunk, az az első Winchestert (a **C:** lemezt) jelenti, a 81H érték a második Winchestert és így tovább. A legtöbb lemezkezelő funkció egy vagy több szektorra vonatkozik, ezért meg kell adni a kezdőszektor címét, mégpedig a következőképpen:

- a kívánt oldal a DH regiszterben (nullától háromig);
- a sáv értékét a CH és CL regiszterekben kell megadni úgy, hogy a tízbites érték alsó nyolc bitje a CH regiszterbe kerül, míg a legfelső két bit a CL regiszter legfelső két bitjére (a 6. és 7. bitre);
- a művelet kezdőszektora pedig a CL regiszter alsó 6 bitjén foglal helyet (mivel a sávon belüli szektorszám egytől legfeljebb tizenhétig terjed, ez legfeljebb öt bitet vesz igénybe).
- végül az elérni kívánt szektorok számát az AL regiszterbe kell betölteni a hívás előtt.

Az IBM PC ROM BIOS-a a következő lehetőségeket nyújtja a diskkezelésben: a drive alapállapotba hozása, a paraméterek lekérdezése, szektor(ok) beolvasása, kiírása, ellenőrzése, és egy sáv leformázása. Winchester kezeléséhez új funkciókat is használhatunk, bár az első hat a leglényegesebb. A közös funkciókat (ahol lényeges különbségek vannak közöttük) két oszlopba fogjuk szedni, a bal oldali oszlop mindig az eredeti ROM BIOS rutin specifikációjára vonatkozik, a jobb oldali oszlop pedig a Winchestert is kezelő rutincsomag specifikációját ismerteti. A további (csak a Winchester kezelésére szolgáló) funkciók esetén már a megszokott "**Csak IBM XT !**" vagy "**Csak IBM AT !**" felirat utal majd arra, hogy az adott funkció melyik géptípuson is fut. Az új funkciók leírásánál arra is felhívjuk a figyelmet, hogy ezek közül egyesek nemcsak a Winchesterre, hanem a floppy-ra is alkalmazhatók. Egyébként néhány funkció nem is a megszokott lemezműveletekre szolgál, csak diagnosztikai szerepe van.

> Megjegyzés: a Magyarországon elterjedt gépekbe ritkán építenek be két Winchestert. Gyakoribb, hogy a memória egy részét RAM DISK-ként használják (különösen olyan IBM AT-ben, amely a 640 Kb-os maximális memóriakiépítésen túl rendelkezik memóriakiterjesztéssel is). Ez a rendszer számára (egy Winchester esetén) a D: lemez; azonban ROM BIOS szinten nem lehet a RAM DISK-et elérni, a lemezkezelő interrupt-hívás ilyen kísérletekre hibaüzenetet ad.

VIII.4.1. A diskette-rendszer előkészítése

Funkciókód: AH = 00H INT 13H
Javasolt konstansnév: DSK_INI

IBM PC ROM BIOS

Be:
Semmi egyéb
Válasz:
Semmi

IBM XT ROM BIOS

Be:
DL - drive
Válasz:
Carry 1 hiba után, 0 egyébként
AH a művelet státusza (bővebben lásd 01 funkció)

Ez a funkció alapállapotba hozza a diskette-kezelő programokat és magát a lemezegységet. Ez a táblázatok előkészítését jelenti a programok oldaláról, a lemezegység "újrakalibrálását", azaz a

fejeknek a Ø. sáv fölé mozgatását a drive-ok számára. E funkciót komoly lemezkezelési hibák után használhatjuk fel.

IBM PC ROM BIOS

A PC ROM BIOS e funkciójával kapcsolatban semmi további mondanivaló nincs.

IBM XT ROM BIOS

Az IBM XT ROM BIOS-a csak a floppy-drivert hozza alapállapotba, ha a DL 8ØH-nál kisebb értéket tartalmaz; 8ØH vagy afeletti érték esetén mind a floppy, mind a Winchester driver és drive előkészítése megtörténik.

VIII.4.2. Az utolsó művelet eredményének lekérdezése

Funkciókód: AH = ØlH INT 13H
Javasolt konstansnév: DSK_GTDST
Be:

Semmi.

Válasz:

Az XT ROM BIOS lista szerint AL-ben, Peter Norton sokat használt Programmer's Guide-ja szerint az AH-ban, tapasztalataim szerint (bár sajnos nem eredeti IBM AT gépem van, tehát se nem PC, se nem XT, se nem AT) mind az AL-ben, mind az AH-ban egy nyolcbites státuszértéket ad vissza (mindkettőben ugyanazt).

A régebbi IBM PC szakirodalom a választ bitenként adja meg:

7	6	5	4	3	2	1	Ø	Jelentés
1	.	.	.	.	.	.	.	Time Out - a drive nem válaszol
.	1	.	.	.	.	.	.	Bad Seek - nem találja a sávot
.	.	1	.	.	.	.	.	Vezérlőegység hibája
.	.	.	1	.	.	.	.	Bad CRC - CRC-hibás a szektor
.	.	.	.	1	.	.	.	DMA hiba - A DMA lekésett (túlfutás)
.	.	.	.	.	1	.	.	Bac Sector - nem találja a szektort
.	.	.	.	.	.	1	.	Sector ID error - azonosító hiba
.	.	.	.	.	.	.	1	Bad Command - illegális parancs
.	.	.	.	1	.	.	1	DMA Bound. Error - 64K-nál hosszabb
.	.	.	.	.	.	1	1	Write Protect - védett lemez írása

A lemezművelet floppy-BIOS által adott státusza

Most lássunk egy korszerűbb hibalistát, amely számos, a bitkombinációkból ki nem következtethető értéket is tartalmaz:

FFH	Sikertelen érzékelési művelet	(Winchester)
E0H	Státuszhiba	(Winchester)
CCH	Lemezírási hiba	(Winchester)
BBH	Közelebbről nem definiált hiba	(Winchester)
80H	A drive nem üzemkész	
40H	Sikertelen sávkeresés	
20H	Kontroller hiba	
11H	ECC-vel javított adat(ok)	(Winchester)
10H	CRC vagy ECC hiba lépett fel	
0FH	DMA-szint kívül a határokon	(Winchester)
0EH	Vezérlőadat-jelző érzékelése	(Winchester)
0DH	Érvénytelen szektorszám formázáskor	(Winchester)
0CH	A lemez típusa nem ismerhető fel	(floppy)
0BH	Rossz cilinder a lemezen	(Winchester)
0AH	Érvénytelen szektorjelző	(Winchester)
09H	DMA-művelet során 64 Kb határ átlépése	
08H	DMA hiba - adatvesztés túlfutás miatt	
07H	Lemezparaméterek lekérdezése sikertelen	(Winchester)
06H	Lemezcsere az utolsó művelet óta	(floppy)
05H	Reset művelet nem sikerült	(Winchester)
04H	Sávkeresési hiba	
03H	Kísérlet írásvédett lemez írására	
02H	Szektorazonosító hiba	
01H	Érvénytelen parancs	
00H	Sikeres művelet	

A lemezműveletek Winchester-BIOS által adott státusza

VIII.4.3. Szektor(ok) beolvasása

Funkciókód: AH = 02H INT 13H
Javasolt konstansnév: DSK_RDSECT
Be:

DL	drive sorszáma
DH	oldal száma
CH	a kért sáv száma
CL	a (kezdő) szektor száma
AL	beolvasandó szektorok száma
ES:BX	célbuffer címe (hatvannégy Kb-os határt nem léphet át!)

IBM PC ROM BIOS	IBM XT ROM BIOS
Drive sorszám: (0-3) a megcímezhető két belső és két külső lemezegység közül választ	Drive sorszám (0-3 / 80-81) 80-nál kisebb => floppy, 80 nagyobb => Winchester
Oldalszám: 0-1	Oldalszám: 0-3
Sáv száma: (0-39) Speciális esetekben megengedhető e határok átlépése; védett programok sűrűn megteszik	Sáv száma (0-39/79/615/...) A sáv számát tíz biten adjuk meg, felső két bit CL felső két bitjén
Szektorok száma: (1-8/9) Nem haladhatja meg egy sáv hosszát, azonos szektoron kell lenniük	Szektorok száma: (1-128) Winchesterről elhelyezkedésüktől függetlenül olvashatjuk

Válasz:

Carry	0, ha a művelet sikeres volt, 1, ha nem
AH	a driver státusza a művelet után (sikeres művelet után 00H)
AL	a beolvasott szektorok száma

Az ES:BX által címzett bufferben megtaláljuk a beolvasott szektorok tartalmát.
Megjegyezzük, hogy a szakirodalom szerint az AL regiszterben visszakapjuk a valójában beolvasott szektorok számát. Ezt azonban a tapasztalat nem igazolta; minden művelet után (függetlenül attól, hogy sikeres volt-e vagy sem) 00H volt az AL tartalma.

VIII.4.4. Szektor(ok) kiírása

Funkciókód: AH = 03H INT 13H
Javasolt konstansnév: DSK_WRSECT

Be:

Pontosan, mint az előző funkciónál

Válasz:

Carry	0, ha a művelet sikeres volt, 1, ha nem
AH	a driver státusza a művelet után (sikeres művelet után 00H)
AL	a kiírt szektorok száma

VIII.4.5. Szektor(ok) ellenőrzése

Funkciókód: AH = 04H INT 13H
Javasolt konstansnév: DSK_VFSECT
Be:
Pontosan, mint az előző funkciónál

Válasz:
Carry 0, ha a művelet sikeres volt,
1, ha nem
AH a driver státusza a művelet után (sikeres művelet után 00H)
Ez a funkció a szektor(ok) CRC-jét ellenőrzi, az adatokra vonatkozó összehasonlítási lépésre nem kerül sor.

VIII.4.6. Egy sáv formázása

Funkciókód: AH = 05H INT 13H
Javasolt konstansnév: DSK_FRMTRACK

IBM PC ROM BIOS

Be:
DL drive sorszáma (0-3)

DH oldal száma (0-1)
CH a formázandó sáv sorszáma (0-39)

AL az elhelyezendő szektorok száma (max 10)

ES:BX
Leírótábla címe (szektor-azonosítók táblázata)

A leírótábla szerkezete:
A tábla minden sora egy szektort ír le (a szektorok számát CL tartalmazza). Minden sor négy byte-ot tartalmaz. Ezek a byte-ok pontosan megfelelnek a lemez fizikai felépítésének ismertetése során leírt négy byte-nak, azaz az

IBM XT ROM BIOS

Be:
DL drive sorszáma (0-1 egy floppy-t, 80-81 Winchestert jelent)
DH oldal száma (0-3)
CH a formázandó sáv száma (0-39/79 floppy-n, 0-614 21 Mb-s Winchesteren)
AL interleave tényező, logikailag folytonos szektorok fizikai távolsága.
ES:BX
Leírótábla címe
Csak IBM AT: szektorsorszám-leíró tábla címe
A leírótábla szerkezete:
A Winchester leíró táblája 512 byte hosszú, melynek első 34 byte-ja hordozza a lényeges információt. A tábla e része byte-párokból áll, melyek száma 17 (szektoronként egy). Az első byte értéke 0 vagy 80H lehet asze-

első a szektorheaderekbe felveendő cilindersorszámot (C), a második az oldalszámot (H) a harmadik a szektorsorszámot (R), végül a negyedik a szektorhossz kódját tartalmazza. Egy példa ilyen táblára:

```
SEC_TAB    LABEL    BYTE
    DB    10, 0, 1, 2
    DB    10, 0, 2, 2
    DB    10, 0, 3, 2
    DB    10, 0, 4, 2
    DB    10, 0, 5, 2
    DB    10, 0, 6, 2
    DB    10, 0, 7, 2
    DB    10, 0, 8, 2
    DB    10, 0, 9, 2
```

Ez a tábla a 10. sáv 0. oldalát írja le úgy, hogy a sáv azonos hosszúságú szektorokat tartalmaz. Ha meg akarjuk keverni a lemezen levő információt, akkor össze kell keverni a fenti táblázatot.

rint, hogy a szektort jó vagy hibás szektorként akarjuk leformázni; a második byte írja elő a szektor logikai helyét a sávon belül. A szokásos hármas eltolást alkalmazva a következőképpen néz ki egy hibátlan sáv leírója:

```
SEC_TAB    LABEL    BYTE
 DB    00,01, 00,07, 00,13
 DB    00,02, 00,08, 00,14
 DB    00,03, 00,09, 00,15
 DB    00,04, 00,10, 00,16
 DB    00,05, 00,11, 00,17
 DB    00,06, 00,12
```

Egyébként azért szokás ilyen eltolást alkalmazni, hogy a hardware elősegíthesse a folytonos adatok gyors elérését. Mire a rendszer "felocsúdik" az előző műveletből, a lemez éppen befordul a következő sáv átviteléhez.

Válasz:

Carry – 0 ha sikeres,
– 1 ha nem; ekkor AH tartalmazza a hibakódot.

Ha a lemez formázása során eltérünk a szokott rendtől (pl. átrendezzük a sávokat, vagy a különböző sávokra különböző számú szektort helyezünk el, vagy egy sávon belül különböző hosszúságú szektorokat írunk), akkor az MS-DOS számára olvashatatlanná tesszük a lemezünket. Ez a programok másolás elleni védelmének legismertebb módszere (kiegészítve azzal, hogy egy-két sávval többet írunk fel a lemezre, mint a szokásos 40).

Meg kell azonban mondanunk, hogy a korszerű másolóprogramok (COPYWRIT, COPYIIPC stb.) az ilyen gyermeteg trükkökön könnyedén túllépnek, és egy igazi, hatékony programvédelemhez korábban kell felkelnünk. A leghatékonyabb védelem mellesleg talán az, ha olyan mellékes szolgáltatásokat nyújtunk, és olyan olcsón adjuk a programot, hogy ne érje meg lopni. Ez persze csak viszonylag kis programok és nagyon széles piac esetén működik. Számomra a legrokonszenvesebb az a módszer, melyet (többek között) a FREESOFT Corporation követ. Ez a társaság engedélyezi a

programok szabad terjesztését. Ha a lemásolt program sorszámát pár dollár társaságában elküldjük a társaságnak, akkor ők postafordultával visszaküldik az adott program legfrissebb verzióját. Ha valamivel több dollárt küldünk, akkor terjesztői jogkört kapunk. Valahányszor tőlünk származó programért kap pár dollárt a társaság, akkor nekünk is küld egy kis részesedést. Ha pedig nem küldünk egy vasat sem, az se baj; azt üzenik, hogy használjuk az adott verziót egészséggel. Ezzel elérik (ha elég széles a piac), hogy nem fizetnek rá, mert a kért összeg elég kicsi; feltehetően sokan küldik el a pénzt a legfrissebb verzió kedvéért.

Egyébként azt javaslom, hogy míg igen nagy programozási gyakorlatot nem szerzünk, ne "bravúroskodjunk" a lemezek közvetlen kezelésével! Erre a célra számos jól belőtt program áll rendelkezésünkre, mint maga az MS-DOS, a DEBUG, a NORTON UTILITIES, a PATHMINDER és sok egyéb. Ha "bányászni" akarunk a lemezeken, használjuk ezeket bizalommal.

A lemezkezelésben rendelkezésünkre álló további funkciók már csak az IBM XT és az IBM AT ROM BIOS-ában érhetők el. Ezek ismertetése során már nem két oszlopban adjuk meg a tudnivalókat, hanem a korábban megszokott formában. A funkció leírása előtt megadjuk, hogy XT vagy AT funkcióról van-e szó, míg az ezt ismertető sor jobb oldalán külön kiemeljük, ha az adott funkció csak floppy-diskre vagy csak Winchesterre alkalmazható.

Csak IBM XT ! **Winchester**

VIII.4.7. Sávformázás és rossz szektorok kijelölése

Funkciókód: AH = 06H INT 13H
Javasolt konstansnév: DSK_FRMTRSFLAG

Be:

AL	Interleave érték
CH	a formázandó sáv sorszáma, felső két bitje CL felső két bitjén (0-614)
DH	oldal száma (0-3)
DL	lemez száma (nagyobb-egyenlő, mint 80H !)

Válasz:

Carry	1 hiba esetén, 0 egyébként
AH	a művelet státusza

Ez a funkció automatikusan beállítja a hibás szektorokat jelző flageket, ha hibás szektort talál.

A lemezre korábban felírt összes információt törli!

Csak IBM XT ! **Winchester**

VIII.4.8. Lemezformázás adott sávtól kezdve

Funkciókód: AH = 07H INT 13H
Javasolt konstansnév: DSK_FRMDISK
Be:

AL	Interleave érték
CH	a formázandó sávok kezdő sorszáma, felső két bit CL felső két bitjén (0-614)
DL	lemez száma (nagyobb-egyenlő, mint 80H !)

Válasz:

Carry	1 hiba esetén, 0 egyébként
AH	a művelet státusza

A CL:CH-ban megadott sávtól kezdve formázza a teljes Winchestert. **A lemezre korábban felírt összes információt törli!**

Csak IBM XT ! **Winchester**
Csak IBM AT ! **Winchester, floppy**

VIII.4.9. Lemezparaméterek lekérdezése

Funkciókód: AH = 08H INT 13H
Javasolt konstansnév: DSK_GTDSKPAR
Be:

DL	lemezegység sorszáma

Válasz:

CH	cilinderek száma; felső két bit CL felső két bitjén;
CL	szektorok száma sávonként
DH	oldalak száma

Egyéb válaszértékek Winchester esetén:

Carry	1 hiba esetén, nulla egyébként
AH	a művelet státusza
DL	a Winchesterek száma

Egyéb válaszértékek floppy-disk esetén:

AX	mindig 0
BL	alsó négy biten a lekérdezett floppy-egység CMOS RAM-ban rögzített típusa:
	00 ismeretlen típus, vagy a CMOS sérült
	01 360 Kb-os (5,25 inch, 40 sávos egység)
	02 1,2 Mb-os (5,25 inch, 80 sávos egység)
	03 720 Kb-os (3,50 inch, 80 sávos egység)
BH	mindig 0
DL	floppy-egységek száma
ES:DI	a diskette paramétertábla címe

Ez a funkció a kiválasztott lemez legfontosabb paramétereit adja vissza. Winchesterre vonatkozóan kiadhatjuk minden olyan gépen, amelyen van egyáltalán Winchester; floppy-disk lekérdezésére csak olyan gépeken van lehetőség, melyekben nagykapacitású floppy van (gyakorlatilag tehát csak IBM AT-n).

> Megjegyzés: az IBM AT gépekben van egy olyan CMOS technológiával készült RAM terület, melyet egy akkumulátor vagy elem állandóan feszültség alatt tart. Ez sok lényeges információt őriz meg a gép két bekapcsolása között: a gép konfigurációját, és a valósidejű órát, mely (míg csak az áramforrás el nem romlik) állandóan jár. A fenti funkció IBM AT-n nem csak a lemezről, de a CMOS RAM-ról is ad információt.

Winchester

VIII.4.10. Winchester-leíró tábla előkészítése

Funkciókód: AH = 09H INT 13H
Javasolt konstansnév: DSK_INIDSKTAB

Be:
DL lemez száma (nagyobb-egyenlő, mint 80H)
Válasz:
Carry 1 hiba esetén, 0 egyébként
AH a művelet státusza

E funkció a 41H és a 46H interrupt vektorok által címzett leíró táblázatok ROM-ból való újra előkészítését idézi elő. Mint az az "Egyéb ROM BIOS interruptok" fejezetben olvasható, a 41H és 46H interrupt vektorok az első és a második Winchester leíró tábláját címzik.

Winchester

VIII.4.11. "Hosszú" olvasás

Funkciókód: AH = 0AH INT 13H
Javasolt konstansnév: DSK_RDLONG
Be:
Pontosan, mint 02H funkciónál.
Válasz:
Pontosan, mint 02H funkciónál.

Winchester

VIII.4.12. "Hosszú" kiírás

Funkciókód: AH = 0BH INT 13H
Javasolt konstansnév: DSK_WRLONG
Be:
Pontosan, mint 03H funkciónál.
Válasz:
Pontosan, mint 03H funkciónál.

Ezek a funkciók diagnosztikai célokat szolgálnak. Abban különböznek a 02-től, illetve a 03-tól, hogy nemcsak a címzett szektorok adatrészét, hanem azok ECC-jét is mozgatják. Ezért akkora buffert kell kijelölnünk, melyben elfér ez a szektoronként négy byte-nyi információ; kiírás esetén elő kell készítenünk a szektorok kívánt ECC-jét is.

Winchester

VIII.4.13. Sávkeresés

Funkciókód: AH = 0CH INT 13H
Javasolt konstansnév: DSK_SEEK
Be:

CH	keresendő sáv száma; felső két bit CL felső két bitjén
DH	oldal száma
DL	lemez száma (nagyobb-egyenlő, mint 80H)

Válasz:

Carry	1 hiba esetén, 0 egyébként
AH	a művelet státusza

E funkció a Winchester író/olvasófejének a kívánt sávra való parkolására, valamint teszelésként végrehajtott véletlen olvasásra és természetesen sávkeresésre szolgál. Megnéztem egy parkoló programot, mely a Winchestert illetően előbb egy 08H funkcióval lekérdezi a paramétereket, majd a 00H funkcióval alapállapotba hozza a Winchester-drivert, végül a 12H funkcióval végrehajt egy sávkeresést (a célzott sáv egy, a lemez rendszer- és adatterületen kívüli helye). A program lírai befejezése egyébként egy INT 22H utasítás, melyről köztudott, hogy tilos kiadni, mivel lepusztítja az MS-DOS-t. De a SHIPDISK programnak pont ez a célja: a parkolás után már semmilyen MS-DOS műveletet nem szabad végezni, hiszen az visszavinné a fejet a veszélyes helyekre.

Winchester

VIII.4.14. Winchester-rendszer előkészítése

Funkciókód: AH = 0DH INT 13H
Javasolt konstansnév: DSK_ALTRESET
Be:
DL lemez száma (nagyobb-egyenlő, mint 80H)
Válasz:
Carry 1 hiba esetén, 0 egyébként
AH a művelet státusza

Ez a funkció az "Alternate Disk Reset" címet viseli az angol eredetiben. Bővebben lásd a 00 funkciót; ez lényegét tekintve abban különbözik attól, hogy csak a Winchestert képes inicializálni, míg az a floppy-drivert is.

(A **0E** és **0F** funkciók (**Csak IBM XT**) belső diagnosztikai célokra vannak fenntartva.)

Winchester

VIII.4.15. A Winchester működéskészségének ellenőrzése

Funkciókód: AH = 10H INT 13H
Javasolt konstansnév: DSK_TSTDRRDY
Be:
DL lemez száma (nagyobb-egyenlő, mint 80H)
Válasz:
Carry 1 hiba esetén, 0 egyébként
AH a művelet státusza

A funkció segítségével lekérdezhető, hogy üzemkész-e a kiválasztott Winchester. Ha igen, akkor a Carry és az AH is 0 értéket kap.

Winchester

VIII.4.16. Winchester író/olvasófej előkészítése

Funkciókód: AH = 11H INT 13H
Javasolt konstansnév: DSK_WRECAL
Be:
DL lemez száma (nagyobb-egyenlő, mint 80H)
Válasz:
Carry 1 hiba esetén, 0 egyébként
AH a művelet státusza

A 00 és a 13 funkciók is meghívják ezt a funkciót, amely a Winchester író/olvasófejét "újrakalibrálja", azaz a 0. cilinderre viszi.

(A **12, 13** és **14** funkciók diagnosztikai célokra fenntartva.)

Csak IBM AT ! **Floppy**

VIII.4.17. Disktípus beolvasása

Funkciókód: AH = 15H INT 13H
Javasolt konstansnév: DSK_RDDASD
Be:
DL lemez száma (nagyobb-egyenlő, mint 80H)
Válasz:
Carry 1 hiba esetén, 0 egyébként
AH a művelet státusza
CX:DX sikeres művelet után ez a duplaszó tartalmazza a lemezen elérhető szektorok számát.

Ez a funkció elsősorban arra szolgál, hogy lekérdezhessük: érzékeli-e a floppy-egység a lemez cseréjét, vagy sem. (A lemezcsere tényének lekérdezésére a 16H funkció szolgál).

Csak IBM AT !

VIII.4.18. Lemezcsere lekérdezése

Funkciókód: AH = 16H INT 13H
Javasolt konstansnév: DSK_CHGLNST
Be:
DL lemez száma (kisebb, mint 80H)
Válasz:
Carry 1 hiba esetén, 0 egyébként
AH a művelet státusza; a lehetséges értékek:
00H nem volt lemezcsere az utolsó lemezművelet óta
01H érvénytelen lemezegység szám (rossz DL)
06H volt lemezcsere az utolsó lemezművelet óta
80H a kiválasztott lemezegység nem üzemkész

Csak IBM AT ! **Floppy**

VIII.4.19. Disktípus beállítása formázáshoz

Funkciókód: AH = 17H INT 13H
Javasolt konstansnév: DSK_STDASD
Be:
AL floppy-egység és floppy-disk meghatározása:
01H 320/360 Kb-os floppy egy 360 Kb-os lemezegységben

	02H	360 Kb-os floppy egy 1,2 Mb-os lemez-egységben
	03H	1,2 Mb-os floppy egy 1,2 Mb-os lemez-egységben
	04H	720 Kb-os floppy egy 720 Kb-os lemez-egységben
DL	lemez száma (kisebb, mint 80H)	

Válasz:

Carry	1 hiba esetén, 0 egyébként
AH	a művelet státusza

Csak IBM AT ! **Floppy**

VIII.4.20. Lemeztípus beállítása formázáshoz

Funkciókód: AH = 18H INT 13H
Javasolt konstansnév: DSK_STDSKTYP
Be:

CH	sávok száma; felső két bit CL felső két bitjén
CL	szektorok száma sávonként
DL	lemez száma (kisebb, mint 80H)

Válasz:

Carry	1 hiba esetén, 0 egyébként
AH	a művelet státusza
ES:DI	a kiválasztott lemez paramétertáblájának címe

Ez a funkció a formázás műveletéhez készít elő egy táblázatot. Azok a gépek, amelyek ROM BIOS-ában ez a funkció egyáltalán elérhető (modernebb, gyakorlatilag 1985 után megjelent AT-k) a 05H funkció előtt hívják meg ezt a funkciót.

VIII.5. A diskette fizikai kezelése

Ebben a fejezetben a diskette vezérlő direkt programozásáról olvashatunk. Meg kell ismerkednünk a portok sorszámával, jelentésével és (legalább vázlatosan) a kiadható parancsokkal, a portokról nyerhető információval.

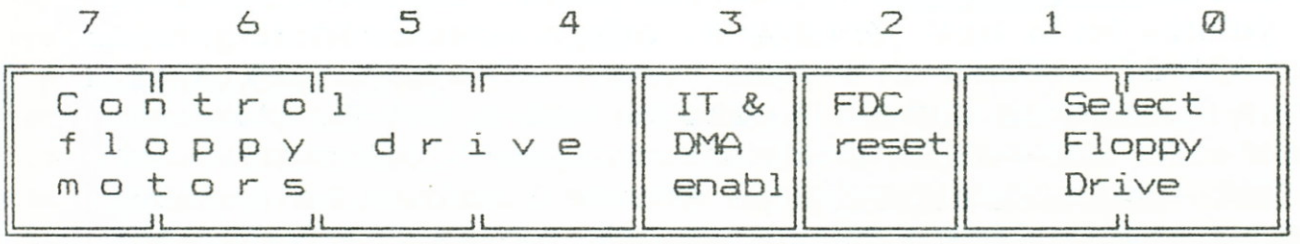
A diskette-vezérlő bővítőkártyáját tetszőleges helyre nyomhatjuk be a gépbe, bekapcsolva ezzel a gép belső buszába. Az IBM PC eredeti kártyája "befelé" és "kifelé" is rendelkezik csatlakozóval. Mind a belső, mind pedig a külső két-két floppy-egység kezelésére alkalmas, így az eredeti vezérlő két beépített és két külső diskette-egységet képes kezelni (egyidejűleg természetesen mindig csak egyet).

Először ismerkedjünk meg a vezérlő közvetlen programozásának, működésének elveivel. A kontrollert vezérlő program felprogramozza a 8237 típusjelű DMA egységet, majd a diskette-vezérlőt. A program ezután megvárja a kért művelet befejezését. Ez nagyjából úgy folyik le, hogy a diskette-vezérlő felkészül a művelet elvégzésére (megkeresi a sávot stb.), majd egy elektronikus jellel értesíti a DMA-t, amely átveszi a vezérlést a busz felett, és végrehajtja a kívánt átvitelt. Ennek befejezéséről a vezérlő egy hardware interrupttal értesíti a programot. Végül a kontrollert vezérlő program kiolvassa az eredményt, és működését attól függően folytatja, hogy a művelet sikeres volt-e.

A diskette-vezérlő felprogramozása nem egy-egy output-utasítással, hanem output-utasítások sorozatával történik. Tehát elmondhatjuk, hogy magas szinten programozhatjuk a floppy-disk vezérlőt, hiszen egy jól definiált parancssorozat kiadása után önállóan végez el igen összetett műveleteket. Ezeket a parancssorozatokat a továbbiakban egyszerűen programnak mondjuk majd.

VIII.5.1. A diskette adapter regiszterei

A vezérlőnek három fontos regisztere van, melyeket a szokásos módon, az I/O portok segítségével érhetünk el. Közülük egy (Digital Output Register) csak kiírásra, a második (Main Status Register) csak olvasásra szolgál, míg a harmadik (Data Register) írásra és olvasásra is használandó. A Data Register használata igen érdekes. A vezérlő felprogramozása során (a végrehajtandó művelet függvényében) megadott sorrendben egy vagy több adatot kell kiírni ebbe a regiszterbe (ez a parancssorozat a program). A művelet befejezése után az eredményt hasonlóan többszörös olvasással kaphatjuk meg. A floppy-vezérlő I/O portjai a 3F0H - 3FFH címűek. Ezek közül, a fentieknek megfelelően, csak hármat használnak. Címeik: 3F2H (Digital Output Register), 3F5H (Data Register) és 3F4H (Main Status Register).



A Digital Output Register felépítése

- Az alsó két bit választja ki a használandó lemezegységet:

00	- 0. egység (A:)	10	- 2. egység (C:)
01	- 1. egység (B:)	11	- 3. egység (D:)

- Az FDC reset bit Ø értéke "reset"-eli, azaz alapállapotba hozza a floppy-disk vezérlőt. Ha valamilyen műveletet akarunk végezni, akkor e bitnek egyes értéket kell kapnia.
- Az IT & DMA enable bit engedélyezi a művelet végét jelző interrupt beadását, valamint a DMA (Direct Memory Acces áramkör) használatát. E bit értéke okszerűen 1.
- A négy legmagasabb bit vezérli az egyes lemezegységek motorját. Amelyik bitet egybe állítjuk, annak motorja bekapcsol (a motor teljes felpörgését, persze, meg kell várni). A bitek
- 4. bit - Ø. egység (A:) motorja
- 5. bit - 1. egység (B:) motorja
- 6. bit - 2. egység (C:) motorja
- 7. bit - 3. egység (D:) motorja

A Main Status regiszter a vezérlő állapotára, tevékenységére jellemző legfontosabb adatokat tartalmazza. Bármikor elérhető, így mindig kaphatunk információt az éppen folyamatban levő lemezműveletről. A regiszter I/O címe: 3F4H.

7	6	5	4	3	2	1	Ø
Req. for Mast.	Data In/ Out	Non DMA Mode	FDC is Busy	Drive D: is Busy	Drive C: is Busy	Drive B: is Busy	Drive A: is Busy

A Main Status Register felépítése

- Az alsó négy bit az egyes lemezek (a floppy kontroller által vezérelt legfeljebb 4 egység) aktív/inaktív állapotát jelzi.
- A 4. bit (FDC is Busy) a floppy-vezérlő egység aktív vagy inaktív állapotát jelzi.
- Az 5. bit (Non DMA Mode) 1 értéke azt jelzi, hogy a vezérlő nem DMA-módban van.
- A 6. bit (Data In/Out) a vezérlő által végzett művelet irányát jelzi. Ha a bit értéke Ø, akkor a vezérlő a processzor memóriájából a lemez irányába végez adatmozgatást, míg az 1 érték a fordított irány. Tehát, ha ez a bit Ø, akkor a processzor (és így a futó program) szempontjából output folyik.
- A 7. bit (Request for Master) értéke 1 akkor, ha a floppy-vezérlő adatregisztere kész parancsok fogadására, vagy státuszinformációk beadására.

VIII.5.2. A diskette adapter programjai

A floppy-vezérlő közvetlen programozásához szükségünk van a programok ismeretére is. A floppy-vezérlőnek a Data Registeren át összesen tizenötféle programot adhatunk, melyek kiváltják a kívánt működést. Egy program a bevezető utasítás után (amely a műveletet szabja meg) a szükséges paraméterek kiadásából áll. A tizenhatodik program, amelynek bevezetője bármely illegális parancskód lehet, arra használható, hogy a vezérlőtől státuszinformációt nyerjünk válaszként.

A programok két jól elkülönülő csoportra oszthatók. Az egyik a szokásos lemezműveletek csoportja (írás, olvasás stb.). Erre a csoportra az jellemző, hogy a floppy-disk vezérlő a DMA áramkör segítségével valamilyen adatmozgatást végez a memória és a lemez között (vagy legalábbis a memória és a lemez egy területének összehasonlítását végzi el). A második csoport a vezérlőprogramok csoportja; ennek legjellemzőbb tagja a SEEK program, amely az író-olvasófej pozicionálását végzi.

Az első csoport maga is két alcsoportra oszlik; az egyik alcsoport nyolc programot tartalmaz; ezek szerkezete annyira azonos, hogy a programok utasításait egy helyen, a programok vázlatos ismertetése után adjuk majd meg. A másik alcsoport két programját külön tárgyaljuk.

Adatmozgató programok

1 Read Data
Adatok beolvasása; a floppy-disk vezérlő beolvassa a megadott szektor(ok) tartalmát a memóriába.

2 Read Deleted Data
Törölt adatok elolvasása; a floppyvezérlő a "Write Deleted Data" programmal kiírt adatokat olvas a lemezről. Lásd ott.

3 Write Data
Adatok kiírása; a floppy-disk vezérlő adatokat ír ki a lemez megadott szektoraiba

4 Write Deleted Data
Látszólag törölt adatok kiírása; adatok kiírása olyan szektorazonosítókkal, amelyeket a közönséges Read Data utasítás nem képes elolvásni (kitűnő lehetőség a programvédelemre). Megjegyezzük, hogy az MS-DOS ezzel a lehetőséggel nem él, minden, a rendszer funkciói segítségével kiírt adat normál adat lesz a lemezen. A "Deleted" (törölt) szó itt is félrevezető. Nem arról van szó, hogy ezzel törölnénk adatokat a lemezről, hiszen az MS-DOS ezt tisztán logikai úton teszi meg (lásd második kötet: a lemezek logikai felépítése).

5 Read a Track
Egy sáv beolvasása; a floppy-disk vezérlő egy teljes sávot olvas be a memóriába.

6 Scan Equal
Egyenlőségvisgálat; a floppy-disk vezérlő a memória egy területét és a lemez adott szektorait hasonlítja össze byte-ról byte-ra. Az összehasonlítás eredménye a program által adott válaszból derül ki.

7 Scan Low or Equal
Vizsgálat kisebb-egyenlőre; a floppy-disk vezérlő a memória egy területét és a lemez megadott szektorait hasonlítja össze byte-ról byte-ra. Az eredmény a program által adott válaszból derül ki.

8 Scan High or Equal
Vizsgálat nagyobb-egyenlőre; a floppy-disk vezérlő a memória egy területét és a lemez megadott szektorait hasonlítja össze byte-ról byte-ra. Az eredmény a program által adott válaszból derül ki.

A további programok megismerése előtt ismerkedjünk meg az első nyolc program belső felépítésével; ezek szinte szóról szóra azonosak, lényegében csak a bevezető parancs kódjában térnek el. Minden program kilenc output utasításból áll, a visszakapott információ beolvasásához hét input utasításra van szükség. A kiadandó kilenc utasításbyte a következő:

- parancskód
A parancskód öt vagy hat bit hosszúságú, és a byte alsó bitjeire kerül (a beolvasó műveletek parancskódja ötbites, a 0-4. biteken kap helyet; kiíró műveletek esetén az 5. bit mindig 0). Beolvasó műveletek esetén az 5. bit szabja meg, hogy a lemezegység átlépje-e a törölt adatokat jelző szektorazonosítót vagy hibajelzést adjon. Világos, hogy kiírás során ennek nincs jelentősége. E bitet a továbbiakban MT-vel (Mellőzd a Törlésjelet) fogjuk jelölni. A közös felső bitek jelentése: a 6. bit minden művelet esetén a lemez elérési módját adja meg: a bit 0 értéke közönséges frekvenciamodulációt jelent, míg a bit 1 értéke módosítottat. Az MS-DOS ez utóbbit használja; ennyit elég tudnunk róla. Ezt a bitet a továbbiakban IM-el, írási móddal fogjuk jelölni. Végül a 7. bit azt adja meg, hogy a művelet egy vagy két oldalra, azaz egy sávra vagy egy cilinderre, tehát a lemez azonos pontján, de az ellenkező oldalon elhelyezkedő részeire vonatkozik-e. A bit 0 értéke sávra, az 1 értéke pedig cilinderre vonatkozó műveletet jelent. A bitet S/C-vel (sáv vagy cilinder) jelöljük.

Program neve	1 v 2 oldal	Modu-láció	Tör. átlép	További parancsbitek				
Adatolv.	S/C	IM	MT	Ø	Ø	1	1	Ø
Tör.a.olv.	S/C	IM	MT	Ø	1	1	Ø	Ø
Adatírás	S/C	IM	Ø	Ø	Ø	1	Ø	1
Tör.adatírás	S/C	IM	Ø	Ø	1	Ø	Ø	1
Sáv beolv.	Ø	IM	MT	Ø	Ø	Ø	1	Ø
Egyenlővizsg	S/C	IM	MT	1	Ø	Ø	Ø	1
Kisebb-egy.	S/C	IM	MT	1	1	Ø	Ø	1
Nagyobb-egy.	S/C	IM	MT	1	1	1	Ø	1

A floppy-vezérlő adatmozgató utasításai

Ezek voltak az egyes programok bevezető byte-jai; a záró-jelben megadott szám a program neve. Most egy-egy bekezdésben ismertetjük a további nyolc byte jelentését.

- <u>lemez- és lemezoldal-azonosító</u>
  Itt csak az alsó három bit értékes, a további bitek tetszőlegesek lehetnek. A byte felépítése:

x	x	x	x	x	FEJ	D1	DØ

  ahol FEJ mondja meg, melyik oldalt kívánjuk használni, míg D1-DØ azonosítja a négy lehetséges lemezegység közül a címzettet.
- <u>logikai cilinder (sáv) azonosító</u>
  Szokásos jele C (Cylinder); lásd a lemez logikai felépítésénél. Ez a byte azt szabja meg, hogy a **lemezen** a szektorazonosítóban milyen sávazonosítónak kell lennie. Ne felejtsük el, hogy a lemezegység nem ezen érték alapján választja ki a sávot; azt a SEEK műveletek során szabjuk meg, hová kerüljön a fej. A lemezegység pedig ott olvas vagy ír, ahol a fej éppen van! Normális, szabályosan formázott lemez esetén, és sikeres fejpozicionálások során a fej vélt helyzete és a lemezen levő sávazonosítók megegyeznek. Ha azonban a driver "eltévedt" (nagynéha ez is előfordul), vagy a lemez nem szabályosan formázott, akkor a két érték nem azonos, és ilyenkor az MS-DOS driverei bajban vannak; megpróbálják a fejet a Ø. sávról ismét a kívánt helyre pozicionálni, de összekevert lemezen pár próbálkozás után feladják.

- logikai oldalazonosító
  Szokásos jele H (Head); lásd a lemez logikai felépítésénél. Ez a byte szabja meg, hogy a **lemezen** a szektorazonosítóban milyen oldalazonosítónak kell lennie. Lásd az előző pontot!
- logikai szektorazonosító
  Szokásos jele R (sectoR); lásd a lemez logikai felépítésénél. Ez a byte azt szabja meg, hogy a **lemezen** a szektorazonosítóban milyen szektorszámnak kell lennie. Lásd az előző pontot!
- szektorhosszazonosító
  Szokásos jele N (Number); lásd a lemez logikai felépítésénél. Ez a byte azt szabja meg, hogy a **lemezen** a szektorazonosítóban milyen szektorhossz-azonosítónak kell lennie. Lásd az előző pontot (a szektor hossza $128 * 2^N$)!
- utolsó szektor,
  vagyis a sávban levő szektorok száma. Ez az adat igen fontos a lemezkezelés szempontjából, hiszen az írás sűrűsége stb. ettől (is) függ.
- szektorok közötti távolság (Gap Length)
  Ez az adat szintén az írás sűrűségének meghatározásában játszik szerepet. A szabványos szektorköz a floppy-paraméterek leíró táblájából olvasható ki; lásd az "Egyéb ROM BIOS interruptok" fejezetet.
- Adathosszúság
  Ha N nulla, a szektorhossz 128 byte, akkor itt megadhatjuk az átviendő byte-ok számát.

Fontos tudnunk, hogy az összehasonlító műveletek esetében ez az utolsó byte mást jelent; ekkor értéke 1 vagy 2 lehet. Ha 1, akkor az összehasonlítás azonos lemezoldalon, folytonos szektorok vizsgálatával történik; ha a byte tartalma 2, akkor szektoronként váltott oldalakon levő adatok vizsgálata folyik.

Most ismerkedjünk meg a válaszként kapható hét byte-nyi információval! Az első három input egy-egy státuszértéket ad (Ø., 1. és 2. státusz, lásd alább), míg az ezt követő négy a műveletben szerepet játszó szektor **valódi**, tehát a lemezről olvasott szektor azonosítóját (a "C, H, R, N" byte-négyest).

VIII.5.3. A diskette adapter státuszbyte-jai

Ø. státusz bitjei:

1.-Ø. a kiválasztott egység kódja (Ø-3)
2. oldal száma
3. 1 értéke jelzi, hogy
   - a lemezegység nem üzemkész;
   - egyoldalas lemez 1. oldalát akartuk elérni

4. 1 értéke jelzi, hogy
   - a lemezegység hibát jelzett, vagy
   - a Ø. sávra pozicionálás után nem találja a Ø. sávot a lemezen
5. 1 értéke jelzi, hogy a kívánt fejpozicionálás bekövetkezett
7.-6. interrupt-kód bitek, jelentésük:
   ØØ - normális befejezés, sikeres művelet
   Ø1 - abnormális befejezés; a megkezdett művelet sikertelenül fejeződött be
   1Ø - érvénytelen parancs; a megadott művelet el sem kezdődött
   11 - sikertelen befejezés; a megkezdett művelet során a lemezegység meghibásodott

1. státusz bitjei:

Ø. 1 értéke jelzi, hogy a diskette-vezérlő nem találta meg a címzett szektor azonosítóját. Ez a bit együtt áll a 2. státusz Ø. bitjével.

1. 1 értéke azt jelzi, hogy egy írási jellegű művelet (írás, törölt adat írása vagy formázás) sikertelen volt, mert a lemez írásvédett.

2. 1 értéke jelzi, hogy egy olvasás, vagy törölt adat írása, vagy egy összehasonlítás sikertelen volt, mivel a lemezegység nem találta a keresett szektort.

3. nem használt, mindig Ø.

4. 1 értéke a "túlfutást" jelzi; a processzor (vagy a DMA) nem szolgálta ki a floppy-vezérlőt időben, azaz nem olvasta ki egy befejezett művelet eredményét.

5. adathiba; ha ez a bit 1 értékű, akkor a floppy-egység CRC-hibát észlelt, akár az azonosítóban, akár pedig az adatterületen.

6. nem használt, mindig Ø.

7. 1 értéke jelzi, hogy az adatelérési művelet nem létező szektorra vonatkozott: sorszáma magasabb, mint a sávon létező szektorok száma. Például nyolc szektoros sáv kilencedik szektorát akartuk elérni.

2. státusz bitjei

Ø. 1 értéke jelzi, hogy a diskette-vezérlő olvasás közben nem találta meg a szektorazonosítót. Ez a bit együtt áll az 1. státusz Ø. bitjével.

1. 1 értéke hibás sávot jelent: ha a lemezről olvasott sávazonosító (C) nem egyezik meg a parancsban megadottal (tehát a vezérlőprogram által vélt sávszámmal), és a felolvasott sávazonosító értéke FFH (vö. az 1. státusz 2. bitjével).

2. 1 értéke az összehasonlítás eredményét jelzi: a floppy-vezérlő nem talált a sávon olyan szektort, amely megfelelt volna a feltételnek.

3. 1 értéke jelzi, hogy az elvégzett összehasonlítás során a memória és a megcímzett sáv tartalma megegyezett.

4. 1 értéke jelzi, hogy a lemezről olvasott sávazonosító nem egyezik meg a parancsban megadottal (vö. a 2. státusz 1. bitjével, valamint az 1. státusz 2. bitjével).

5. 1 értéke adathibát, a megcímzett szektor adatrészének CRC-hibáját jelenti (vö. az 1. státusz 5. bitjével).

6. 1 értéke azt jelzi, hogy a floppy-vezérlő olvasás vagy összehasonlítás közben törölt adat cimkéjét érzékelte.

7. nem használt, értéke mindig Ø.

Most lássuk az első csoport még hátralevő két utasítását!

9 Read ID

Szektorazonosító beolvasása. A program hatására a floppy ott, ahol a fej éppen van, megkísérel beolvasni egy szektorazonosítót, azaz megkeresi a sávon a legközelebbi szektort. Ezzel megpróbálhatunk olvasni olyan lemezt is, melynek formázásáról, belső felépítéséről fogalmunk sincs. A program két output utasításból áll:

Program neve	1 v 2 oldal	Modulláció	Tör. átlép	További parancsbitek			
Azonosítóolv	Ø	IM	Ø	Ø	1	Ø	1	Ø

A második output itt is a lemez- és lemezoldal-azonosító:

x	x	x	x	x	FEJ	D1	DØ

ahol FEJ mondja meg, melyik oldalt kívánjuk használni, míg D1-DØ azonosítja a négy lemezegység közül a címzettet.
A válaszinformáció ugyanaz a hételemű byte-sorozat, mint az előző programok esetén.

10 Format a Track

Az első csoport utolsó programja a kurrens (már ti. a SEEK programmal megcélzott) sáv teljes formázását hajtja végre. Ez a program hat kiírást igényel. A bevezető parancs:

Program neve	1 v 2 oldal	Modu-láció	Tör. átlép	További parancsbitek			
Sávformázás	Ø	IM	Ø	Ø	1	1	Ø	Ø

A második output itt is a lemez- és lemezoldal-azonosító:

x	x	x	x	x	FEJ	D1	DØ

ahol FEJ mondja meg, melyik oldalt kívánjuk formázni, míg D1-DØ azonosítja a négy lemezegység közül a címzettet.
A további kiírandó byte-ok:

- a byte-ok száma szektoronként ("N" paraméter, lásd feljebb)
- szektorok száma a sávon (szokásos értéke lehet nyolc vagy kilenc)
- Szektorok közötti távolság (Gap Length). Szokásos értékét a diskette-leíró táblából olvashatjuk ki
- a formázott sáv adatterületeinek feltöltésére szolgáló byte

A válaszinformáció ugyanaz a hételemű byte-sorozat, mint az előző programok esetén.

Vezérlő programok

11 Recalibrate

Ez a program visszavezérli a fejet a lemez első (logikailag nulladik) sávjába. Előkészítés során, valamint olyankor hasznos, ha a lemezt kezelő program már teljesen eltévedt, és fogalma sincs arról, hogy a lemez mely szektoránál tart éppen (vö. a Seek programmal). A program két parancsból áll, és nem ad semmiféle választ. A bevezető parancs:

Program neve	Parancsbitek							
Fejállítás	Ø	Ø	Ø	Ø	Ø	1	1	1

A második parancs:

x	x	x	x	x	Ø	D1	DØ

azaz elmarad minden, a művelet paramétereit megszabó érték (ami persze természetes is, hiszen nincs semmiféle függő dolog.)

12 Sense Interrupt Status

Ez a program a lemezegység teljesen software úton való vezérlésében játszhat szerepet. Arra szolgál, hogy lekérdezzük: adna-e a diskette-vezérlő interruptot, azaz befejezte-e már a korábban elindított tevékenységet, avagy nem. A program egyetlen parancsból áll:

Program neve	Parancsbitek							
IT-st. beolv	Ø	Ø	Ø	Ø	1	Ø	Ø	Ø

Válaszként két byte-ot kapunk. Az első a Ø. státusz (lásd feljebb). A második byte tartalmazza annak a cilindernek a sorszámát, amelyen befejeződött az esetleges interruptot kiváltó program.

13 Specify

Ez a program a lemezműveletek előkészítésére szolgál. Számos olyan paramétert specifikálhatunk általa, amelynek szerepe van a további műveletek során. Három parancsból áll, válaszérték egyáltalán nincs. A bevezető parancs:

Program neve	Parancsbitek							
Par.megadás	Ø	Ø	Ø	Ø	Ø	Ø	1	1

A második kiírandó adat:

fejléptetés ideje	fej felemelési ideje

Az alsó négy bit (32 ezredmásodperces lépésekben) a fej felemelésének időigényét tartalmazza (értéke 1-től 15-ig terjedhet), a felső négy bit a fej léptetésének időigényét (2 ezredmásodperces egységben, értéke Ø-tól [ez a 2 ezredmásodperc] 15-ig terjedhet). A harmadik kiírandó adat:

fej betöltési ideje	DMA

Mint látható, az alsó bit szabja meg, hogy DMA áramkör segítségével, vagy közvetlen I/O utasításokkal visszük át az

adatokat a központi memória és a lemez között. A felső hét bit (4 ezredmásodperces lépésekben) a fej "betöltési" idejét adja meg, tehát azt az időtartamot, amelyre szükség van a fej leeresztéséhez, és a rezgések megszűnéséhez.

14 Sense Drive Status

Ez a program két parancs kiadása után egy byte-nyi válaszinformációt szolgáltat, amelyből a lemezegység (és nem a vezérlő) állapotát ismerhetjük meg. A bevezető parancs:

Program neve	Parancsbitek							
Dr.st. beolv	Ø	Ø	Ø	Ø	Ø	1	Ø	Ø

A második parancs a szokásos lemez- és oldalválasztás:

x	x	x	x	x	FEJ	D1	DØ

A válaszérték (melyet az eddig megismert Ø., 1. és 2. státusz folytatásaként 3. státusznak nevezünk):

1- Ø. megegyezik a lemez- és oldalválasztó parancs D1 és DØ bitjével, megadja, melyik lemezről van szó.

2. megegyezik a lemez- és oldalválasztó parancs FEJ bitjével, megmondja, hogy a kiválasztott lemez melyik oldaláról van szó.

3. 1 értéke jelzi, hogy a lemezegységben levő lemez kétoldalas.

4. 1 értéke azt jelzi, hogy a lemezegység író-olvasó feje a Ø. sávon áll.

5. 1 értéke a lemezegység üzemkészségét jelzi.

6. 1 értéke azt jelzi, hogy a lemezegységben levő lemez írásvédett.

7. 1 értéke a lemezegységtől érkezett hibajelzést mutatja.

15 Seek

Ez talán a leggyakrabban használt vezérlő program; ő végzi el minden művelet előtt a szükséges fejpozicionálást. Három outputból áll, válaszértéket nem ad. A bevezető parancs:

Program neve	Parancsbitek							
Sávkeresés	Ø	Ø	Ø	Ø	1	1	1	1

A második output a szokásos lemez- és oldalválasztás:

x	x	x	x	x	FEJ	D1	DØ

A harmadik kiírás azt szabja meg, melyik sávra szeretnénk pozicionálni a fejet; értelemszerűen Ø és 39 között lehet (a valóságban persze több is, hiszen nyugodtan túlírhatjuk a 39. sávot).

16 Invalid Operation

Ez az utolsó "program", amely egyetlen "parancsból" áll, igen hasznos lehet minden olyan esetben, mikor le akarjuk állítani a már elindított lemezműveletet. A bevezető parancs lehet bármi, amit nem soroltunk fel; válaszként a Ø. státuszt kapjuk vissza, melynek értéke 80H, azaz jelzi az érvénytelen műveletet. Erre alkalmas parancsok például:

Program neve	Parancsbitek							
Érvénytelen	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
Érvénytelen	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1

A Winchester közvetlen programozásáról semmit nem érdemes mondani egész egyszerűen azért, mert nagyon különböző adapterek vannak forgalomban és mindegyiknek a kezelése eltérő.

IX. Egyéb tudnivalók a hardware-ről

Könyvünk ezen fejezete azokba a "mélységekbe" enged egy kis bepillantást, ami az eddig elmondottak "alatt" van, és amely hasznos és érdekes lehet a programozó számára. Ne felejtsük el, hogy az eddig megismert elemek a gépet alkotó áramköröknek csupán a töredékét alkotják. Számos olyan áramkör van még, amely programozható, és a háttérben működve támogatja a felszínről jól látható áramkörök munkáját. Ennél természetesen még sokkal több olyan áramkör van, amely csak igen egyszerű funkciók végrehajtására képes; ezekkel (bár fontosságuk igen nagy) programozói szinten egyáltalán nem kell foglalkozni.

Az egyes elemek összehangolása és hibamentes együttműködésük biztosítása nagy feladat. Az IBM PC/XT világsikerét nagymértékben köszönheti annak, hogy konstruktőrei nagyon magas szinten és szerencsés kézzel oldották meg ezt a feladatot. Gyakorlatilag még nem tapasztaltam olyat, hogy egy IBM gép hibázott volna, hogy lászólag érthetetlenül kiakadt volna egy program, hogy file-ok másolása során bármiféle hiba lépett volna fel. Ha valaki akár egészen csekély számítógépes gyakorlattal rendelkezik, ezt az erényt igen nagyra fogja értékelni.

Mivel könyvsorozatunkat kifejezetten programozóknak szántuk, ebben a fejezetben is csak olyan elemekkel foglalkozunk, melyek a programozó számára hasznosak lehetnek, és egyáltalán nem törődünk a még mélyebben fekvő elemekkel.

IX.1. 8255 Programmable Peripheral Interface

A 8255 típusjelű áramkör az Intel cég régóta közismert és kedvelt perifériavezérlő eszköze, amely igen rugalmasan és sokoldalúan használható fel. Huszonnégy input-output vonalon át léphet kapcsolatba az általa vezérelt perifériákkal. A vonalakat sokféleképpen oszthatjuk csoportokra, két vagy három csatornát alakítva ki belőlük.

A leírás szerint az input-output vonalakat két csoportba sorolták: az A és a B csoport egyaránt tizenkét vonalat tartalmaz. Egyes esetekben azonban az A csoport "kölcsönvesz" egy vonalat a B csoporttól, így szerintem ennek a csoportosításnak sok értelme nincsen.

A processzor számára a huszonnégy vonal három nyolcbites porton át érhető el; ezeket A, B és C portnak hívják. Legtöbbször az A port, és a C port felső négy bitje tartozik az A csoportba, míg a B port, és a C port alsó négy bitje alkotja a B

csoportot. Tehát mindkét csoport két további részre oszlik, egy nyolcbites és egy négybites, párhuzamosan kezelt csatornára. A csatornák adatátviteli iránya külön-külön is meghatározható. Végeredményben tehát négy csatornát definiálhatunk a huszonnégy biten: az A és B port nyolc-nyolc bitje, valamint a csoportok között megosztott C port alsó, illetve felső négy bitje alkot egy-egy adatátviteli csatornát.

A C port egyes üzemmódokban egyszerű be- és kiviteli portnak is használható, ekkor a 8255 PPI valóban huszonnégy adatátviteli vonalon áll kapcsolatban a külvilággal, a hozzá kapcsolt perifériákkal. Ilyen esetben mindenféle státuszvizsgálat nélkül folyik az adatforgalom. A legtöbb esetben azonban úgy programozzák a PPI-t, hogy a két adatátviteli port mellett a megosztott C port az adatportok státuszaként szolgáljon.

A PPI háromféle üzemmódban használható, ezek közül a B csoport vonalaira az első kettő alkalmazható, az A csoportra mind a három. Ezek az üzemmódok: szokásos intput-output (Ø. mód), strobe input-output (1. mód) és kétirányú átvitel (2. mód). Az A és a B csoporthoz tartozó vonalak egymástól függetlenül programozhatók, egymástól függetlenül képesek adatátvitelre..

IX.1.1. A PPI üzemmódjai

a) Ø. mód, szabad input/output

Ebben a legegyszerűbb üzemmódban az adott csoport mindkét csatornája (az A csoport esetén az A, és a C port felső négy bitje, a B csoport esetén a B, és a C port alsó négy bitje) szolgálhat vagy beolvasásra, vagy kivitelre; semmiféle járulékos szolgáltatás (handshaking vagy interrupt) nincsen. Ha valamit kiküldünk a PPI-nek, akkor el kell higyjük: az valóban kiment, amit pedig a beolvasás során kapunk, arról el kell hinnünk: a periféria valóban azt akarta küldeni nekünk.
Megjegyezzük, hogy az IBM PC/XT alapkártyáján levő PPI mindkét vonalcsoportja ebben az üzemmódban dolgozik; lásd alább.

b) 1. mód, "kézfogásos" (handshaking) input/output

Ebben a módban csak a két fő portot használhatjuk adatátvitelre, míg a C port az A és B között megosztva státuszinformációkat szolgáltat; az alsó három bit a B, a felső öt bit pedig az A porthoz tartozik.
Ezzel kapcsolatban meg kell ismerkednünk a PPI interrupt-szol-

gáltatásaival. Mindkét csoportra nézve van egy belső bit, amelyet INTE-nek, INTerrupt Enable bitnek neveznek. Ezt a bitet egyébként a direkt bitállító műveletekkel módosíthatjuk. Ha értéke egy, akkor a PPI kezeli az interrupt-kimenetet (melynek aktuális értékét, pollozási céllal, a C port egy-egy bitjén át kérdezhetjük le), ha pedig nullára állítottuk, akkor a PPI nem kezeli az interrupt-kimenetet; sem pollozással, sem hardware-interrupt-figyeléssel nem tudjuk meg, hogy interruptra okot adó esemény bekövetkezett-e vagy sem.
Az A csoportra vonatkozó INTE beolvasás esetén a C port 4., kivitel esetén a C port 6., míg a B csopotra vonatkozó INTE a C port 2. bitjének direkt írásával állítható. A C port kiolvasásával nyerhető információk:

7	6	5	4	3	2	1	0
Output Buffer Full	Ac-knowl-edge	Input Buffer Full	Strobe Input	INT-A req-uest	Strobe Input/ Ackn.	IN/OUT Buffer Full	INT-B req-uest
<======= A csoport státusza =====>					<=== B státusza ==>		
<== output ==>		<== input ==>					

A PPI státuszának felépítése

Mint látható, az A csoport azért igényel öt bitet, mert szétválasztották a beolvasási és a kiviteli műveletek során használatos státuszbiteket. Ennek hasznát egyébként a 2. módban látjuk majd; ott ugyanis kétirányú adatátvitelt végezhetünk az A porton át. A bitek jelentése:

- A Strobe Input bemeneten leeső jel hatására kerül be a perifériáról bevitt adat a PPI tárolóregiszterébe, ahonnan a processzor a csatornához tartozó adatport olvasásával kapja meg az értékét; vagyis a periféria az adatküldés befejezését jelzi a jel leejtésével.
- Az Input Buffer Full kimeneti vonal jelzi a perifériának, hogy a PPI belső tárolóregisztere foglalt, tehát nem lehet újabb adatot továbbítani.
- Az Output Buffer Full kimenet aktív szintje jelzi a periférának, hogy a processzor kiírt egy adatot a PPI-nek, amelyet a PPI azonnal továbbíthat.
- Az Acknowledge bemeneten át jelzi a periféria, hogy átvette a PPI által továbbított adatot. E jel megérkezése után adhat a processzor újabb adatot a PPI-nek. Végül:
- Az Interrupt Request bit jelzi a processzornak, hogy a PPI megadott csatornáján valami fontos esemény következett be (a

fenti négy jel bármelyikének megváltozása). Ezt a kimenetet rá lehet kötni egy interruptot fogadni képes áramkör megfelelő bemenetére, ugyanakkor a C port kiolvasásával a processzor software-úton is megtudhatja, hogy van-e interruptra okot adó esemény, vagy nem (tehát a PPI-vezérelte perifériákat hardware interrupt segítségével is, pollozással is kezelhetjük).

c) 2. mód, kétirányú átvitel az A porton át

Ebben az üzemmódban csak az A csoport működhet (persze a B csoportot ettől függetlenül programozhatjuk 0. vagy 1. módba). Az A csoporthoz tartozó A port ekkor kétirányú adatátvitelre képes; a C port felső öt bitje (pontosan úgy, mint az 1. mód esetén) adja meg az A csatorna státuszát, az alsó három bit pedig a B csatornához tartozik. A bitek elnevezése és szerepe is pontosan megegyezik a fentiekkel. Egyetlen fontos információ még: az A csoport INTE bitjét a 2. módban a C port 6. bitjének direkt módosításával állíthatjuk.

IX.1.2. A PPI portjai és programozása

A PPI négy porton át érhető el. Az A, B és C portok emelkedő sorrendben követik egymást, és akár input, akár output utasításokkal elérhetők. A programozás szempontjából legfontosabb a legmagasabb című port, amely a PPI programozására szolgál. Az IBM PC/XT-n egyébként a PPI portjai a 60H - 63H portok, amelyek közül természetesen a 63H a vezérlőport. Lássuk most ennek a bitkiosztását, a parancsok jelentését!

Kétféle parancsot adhatunk ki a PPI-nek. Az egyik természetesen a minden áramkör esetén szokásos módparancs, mellyel felprogramozhatjuk a PPI-t. Itt azonban van egy másik parancs, a bittöltő és törlő parancs; segítségével módosíthatjuk a C port mögött levő bármely adatátviteli vonalon a jelszintet. Ehhez nem kell tudnunk, hogy milyen jelkombináció van a többi vonalon. Ha ezt a C portra kiadott OUT utasítással tennénk meg, az nemcsak a címzett vonalat érintené, hanem valamennyit!

7	6	5	4	3	2	1	0
módp. vagy bitp.	A csoport vezérlése: kívánt üzemmód		A port in/out	C f. in/out	B csoport vezérlése: kívánt mód		B port in/out	C alsó in/out

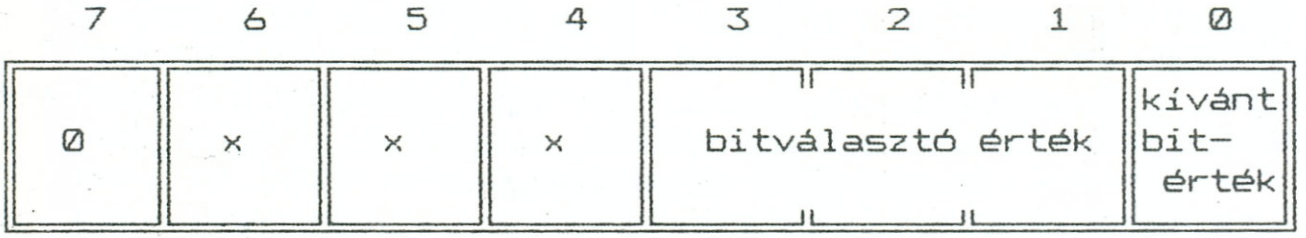
A PPI módparancsa

7. bit (parancsválasztás):
    0 a parancs bitállító művelet (a további bitek értelmezését lásd alább)
    1 a parancs módparancs (a további bitek jelentése ekkor a fenti)

A "B" csoport vezérlése (0-2)

    0. bit: C port alsó 4 (esetleg 3) bitjének irányítása
        0 - output, 1 - input
    1. bit: B port irányítása
        0 - output, 1 - input
    2. bit: módválasztás a B csoportra:
        0 - 0. mód, 1 - 1. mód

Az "A" csoport vezérlése (3-6)
    3. bit: C port felső 4 (esetleg 5) bitjének irányítása
        0 - output, 1 - input
    4. bit: A port irányítása
        0 - output, 1 - input
    5-6 bit: módválasztás az A csoportra
        00 - 0. mód 01 - 1. mód
        10 - 2. mód 11 - nem szokásos (2. mód)



A PPI bitállító parancsa

A bitek részletezése szinte szükségtelen. A 3.-2.-1. bit segítségével választjuk ki a megcélzott C port-beli vonal sorszámát (000 a C port 0. vonalát, 111 a C port 7. vonalát jelenti). A 0. bit szabja meg, hogy milyenre akarjuk állítani az adott vonalat; a 4., 5. és 6. bit érdektelen; a 7. bit 0 értéke mutatja meg a PPI-nek, hogy ez nem módparancs, hanem bitállítás.

IX.1.3. A PPI az IBM PC/XT alapkártyáján

Az alapkártyán elhelyezett PPI az IBM PC/XT gépeken (többek között) egy igen fontos feladatot, a klaviatúra vezérlését látja el. Egyéb feladatai: a hangszóró vezérlése, a Timer 2. csatornájának kapuzása, a memória paritásellenőrzésének vezérlése

és a kazettás magnó motorjának vezérlése. Azért kell hangsúlyozni, hogy az "alapkártyán elhelyezett" PPI-ről van szó, mert néhány bővítőkártya, így az e könyvben nem ismertett BSC adapter is tartalmaz PPI-t, amelyet természetesen másként kötöttek be, és egészen másként kell programozni.

Az alapkártyán levő PPI programozása roppant egyszerű, bár igen ravaszul van bekötve. Mindkét csoportot Ø. módra állítjuk, tehát a PPI számára nem létezik státuszinformáció; az A port beolvasásra, a B adatport kiírásra, a négy-négy bitre osztott C port mindkét része beolvasásra szolgál.

Az A port (6ØH) egyszerre két funkciót lát el. Ha "csak úgy" beolvassuk az A portot, akkor a kapott érték a gép alapkártyáján található mikrokapcsolók állását mutatja, azaz az alapkártyán levő memória hosszát. Ha klaviatúra-interrupt következett be, akkor az A portról beolvasott adat a billentyűkód (scan-kód). Addig őrzi a beolvasott billentyűkódot, míg csak a B port megfelelő bitjének módosításával nem jelezzük a klaviatúrának, hogy vettük az adatot. Tehát, míg a PPI-nek a B port adatport, a külső perifériák úgy vannak bekötve, hogy a B porthoz tartozó egy-egy vonal számukra vezérlővonalnak számít! A B port (61H) csak outputra szolgál és fontos vezérlési feladatokat lát el.

7	6	5	4	3	2	1	Ø
Keyb. clear/ M 6ØH	Keyb. clock cont.	Enable DRAM parity	Enable SRAM parity	Cas-ette m cont.	Multi-plex p. 62H	Speak-er cont.	Timer 2 Gate

A 61H port bitkiosztása

- Timer 2. Channel Gate - a Timer 2. csatornájának aktivizálására szolgál (lásd a "8253 Timer/Counter" fejezetet). 1 értéke aktivizálja a Timert, a Ø pedig leállitja.
- Speaker Control - a beépitett hangszóró be- és kikapcsolására szolgál (1 értéke be, Ø értéke kikapcsolja a hangszórót).
- Multiplex Port 62H - választ a kétfunkciós 62H port lehetőségei között (Ø értékénél a 62H című C port alsó négy bitje az 5. mikrokapcsoló lekérdezésére szolgál, 1 értéke az 1-4. kapcsolókra).
- Cassette Motor Control - a kazettás magnó motorjának be- és kikapcsolására ad utasitást. A jel negált, tehát a bekapcsolást a Ø, a kikapcsolást az 1 érték jelzi.
- Enable Static Ram Parity Check és
- Enable Dynamic Ram Parity Check - a statikus, illetve a dinamikus RAM paritásellenőrzését vezérlik. 1 értéküket be, a Ø pe-

dig kikapcsolja az ellenőrzést. Itt is említjük, hogy alapkiépítésben a paritáshiba NMI-t, Non Maskable Interruptot vált ki, amely egy HLT utasítással legyilkolja a rendszert.

- Keyboard Clock Control - egyes értéke indítja el a klaviatúra belső időzítésére szolgáló órát.
- Keyboard Clear/Multiplex Port 60H - e bit fontosabb funkciója a klaviatúra hardware interruptja során annak jelzése, hogy az A portról (60H) elvettük a billentyűkódot. Ez úgy megy, hogy a 60H beolvasása után beolvassuk a 61H portot is, bebillentjük ezt a bitet, és kiírjuk az értéket; ezután visszaállítjuk a beolvasott értéket és ismét kiírjuk; erre példa a klaviatúra kezelésénél olvasható.
  Felhasználói szinten e bit az A port másodlagos funkciójának vezérlésére szolgál (a ROM BIOS nem használja ki). Akár a C port (62H), az A port is (címe 60H) kétfunkciós port. E bit pontosan ugyanazt jelenti a 60H port számára, mint a 2. bit a 62H portra nézve.

Végül a C port (címe 62H) szintén csak beolvasásra szolgál. Alsó négy bitje (akárcsak az A porté) az alapkártya mikrokapcsolóinak értékét mutatja; a 61H port megfelelő bitjével átállítható úgy, hogy a magasabb sorszámú mikrokapcsolók állását olvashassuk le az alsó négy bitben. A felső négy bit használata elég ködös; főként tesztelési célokat szolgál. A 4. bit a kazettás magnóról olvasott jel szintjével egyezik meg; az 5. bit a Timer kettes csatornájának kimenete, a 6. és 7. bitek pedig a memória paritáshibáit jelzik.

Igen érdekes dolgokra bukkantam, mikor nézegetni kezdtem a 62H portról beolvasható adatokat. Először is, nem tudtam felfedezni a Timer kettes csatornájával való "együttállást", pedig ez nem is volna rossz dolog. Másodszor azt tapasztaltam, hogy a legfelső bit "laffog", előttem ismeretlen okból hol 0, hol 1 az értéke. Lehet, hogy ez a Timer, de erre nézve pozitív bizonyítékot szerezni a nyomozás során nem tudtam. Végül a legérdekesebb: ha jókor nyúlunk le a 62H portra, akkor ott is nyakoncsíphetjük az egyébként interruptot kiváltó billentyűkódot (scan-kódot), ha valaki megnyom valamit a klaviatúrán. A próbaprogramom végtelen ciklusban olvasta a 62H portot és akkor lépett ki, ha a beolvasott érték 1 (azaz az ESC scan-kódja). A program lendületesen futott, míg csak le nem ütöttem az ESC-et; ekkor legnagyobb meglepetésre megnyugodott és kiszállt. Ehhez a merénylethez egyébként a szorgos próbálkozás adott ötletet; amíg csak egy DEBUG-ban írott programmal szimatolgattam a 62H portot, amely az 5. bitet figyelte, a program nem állt le (mert az 5. bit sehogy nem akart beállni). Egyszer unalmamban megü-

töttem a szóközt és a beolvasás után a szokásos érték helyett 39H-t találtam az A regiszterben (a program leállt, mert itt áll az 5. bit). Ez pedig decimálisan 57, azaz a szóköz scan-kódja. Nosza, jobban megnéztem, és valóban megtaláltam minden scan-kódot; e programok azóta az ESC-re lépnek ki (ez az egyetlen olyan billentyű, melynek scan-kódját bárki fejből, azonnal tudja: 1).

IX.2. 8237 Direct Memory Access

A 8237 típusjelű DMA egyike a leghasznosabb kiegészítő egységeknek. Mint neve is mutatja, arra szolgál, hogy a központi memória és külső perifériális egységek között gyors és a processzor beavatkozását nem igénylő adatátvitelt valósítson meg. Igen hasznos lehetősége még a memória-memória adatátvitel. Ha pedig a perifériális eszközök memóriába ágyazottak (ez az Intel cégnél egyébként nem elterjedt gyakorlat), akkor a DMA lehetőséget biztosít két periféria közötti közvetlen adatátvitelre is. Fontos megjegyzés még az, hogy a DMA maximális adatátviteli sebessége 1,6 Megabyte másodpercenként.

A DMA négycsatornás adatátviteli eszköz, tehát négy különböző adatátvitelt képes egymástól függetlenül egyidejűleg megvalósítani. A négy csatorna számozása nullától háromig terjed. Az IBM PC/XT-n a kettes csatorna a floppy-vezérlőhöz van kapcsolva, míg a nulladik igen érdekes és fontos műveletet, a dinamikus memória frissítését végzi.

A dinamikus RAM (a továbbiakban DRAM) olyan memória, melynek elérési ideje ugyan hosszabb, de az ára lényegesen alacsonyabb, mint a "közönséges", statikus memóriáké. Még egy fontos különbség van: a statikus RAM addig őrzi a beleírt tartalmat, míg ki nem kapcsoljuk a feszültséget, a DRAM csak a másodperc töredékéig. Ahhoz, hogy tovább is megőrizze tartalmát, ki kell olvasni. Minden kiolvasás növeli a memóriába írt adat élettartamát. A DRAM specifikációjában mindig megadják, hogy "hány bites" memóriáról van szó. Ez nem a kapacitását jelenti, hanem azt mondja meg, hogy a teljes címtérnek alulról számítva hány bitjét kell olvasni a frissítéshez. A magasabb sorszámú bitek ugyanis e folyamatban nem játszanak szerepet. A legelterjedtebbek a hét bites DRAM-ok, frissítésükhöz az alsó hét címbitet kell állandóan pörgetni. Ha kiolvassuk a 00000H címet, akkor ez frissíti a 00080H, a 00100H stb. címeket is. Ezt a feladatot látja el az IBM PC/XT-n a DMA 0. csatornája. A frissítések kezdetére egyébként a Timer/Counter megfelelően felprogramozott 1. csatornája

ad jelt. A folytonos frissítés lassítja a gép működését, hiszen a processzor WAIT állapotba kerül addig, míg a DMA a memórián operál.

Ha valakivel nagyon ki akarunk tolni, megtehetjük neki azt a szívességet, hogy időről időre lekapcsoljuk a DMA nulladik csatornáját, vagy a Timert programozzuk át lassúbb ütemadásra; ennek következtében a dinamikus RAM tartalma itt-ott meg fog változni. Ráadásul a változás eléggé sztochasztikus, így a trükk igen alkalmas a programvédelemre. Ha speciális lemezműveletekkel levédett programunkat mégis lemásolják, és programunk észreveszi a turpisságot, akkor ezúton "bosszút állhatunk". A felhasználó valószínűleg hardware-hibára fog gyanakodni, hiszen a memória tartalma másképpen nem szokott megváltozni. Meg kell mondani, hogy ez elég goromba dolog; csak nagyon indokolt esetben, szívünknek különösen "kedves" embereknek tegyük meg.

A kívánt művelet előtt fel kell programozni a DMA-t, amely vár a periféria felől érkező jelre (DREQ, Dma REQuest); ekkor elkéri a processzortól a busz vezérlésének jogát. Mikor az fogadja a kérést, a DMA átveszi a gép buszának vezérlését, és megkezdi az adatátvitelt. A művelet befejezése után visszaadja a busz vezérlését a processzornak. Tehát az egész adatátviteli műveletsorozat a processzor "háta mögött" megy végbe (példa erre a diskette vezérlése, lásd "Lemezkezelés" fejezet).

IX.2.1. A DMA állapotai és üzemmódjai

A DMA két alapvető állapotot ismer: a tétlen és az aktív állapotot. Az aktív állapot további alesetekre osztható, ezek száma négy. A DMA akkor van inaktív állapotban (eredeti szimbóluma SI, State Idle), ha nincs egyetlen kiszolgálásra várakozó DMA kérés sem. Az áramkör egyébként ilyenkor programozható.

Ha DMA kérés érkezik valamelyik csatornára, akkor a DMA aktív állapotba lép és a processzornak elküld egy Hold Requestet, egy várakozási kérést. Amíg a processzor ezt nem nyugtázza (azaz nem jelzi, hogy lemondott a gép buszának vezérléséről), a DMA még nem végez semmilyen műveletet; ez az állapot amolyan félálom a DMA számára (ennek szimbóluma SO, State On). Ha a nyugtázás megérkezett, akkor a DMA átveszi a busz vezérlését, és az aktív állapot valamely fázisába lép (ezeket S1, S2, S3 és S4 jelöli). Ha az átvitelhez túlzottan hosszú időre van szükség, amelyet a DMA belső időzítése már nem tud szinkronizálni, akkor a DMA várakozási állapotba lép az egyes aktív fázisok között (szimbóluma SW, State Wait). Az aktív állapotban négyféle üzemmód, további állapot lehetséges.

a) Single Transfer Mode, egyszerű (byte-onkénti) átvitel. Ebben a módban a kezelt periféria egybyte-os átvitelre van felprogramozva; a DMA elvégzi az átvitelt a memória és a periféria között, csökkenti az adott csatorna pillanatnyi számlálóját, és (a megkívánt iránytól függően) növeli vagy csökkenti a pillanatnyi címet. Ha a csatornát autoinicializálásra programoztuk fel, akkor a számláló elfogyása a paraméterek (pillanatnyi számláló és cím) újratöltését idézi elő.
b) Block Transfer Mode, blokkolt (sokbyte-os) átvitel. Ebben a módban az adatátvitel addig folytatódik, míg a számláló el nem fogy, vagy a kezelt periféria nem jelzi az átvitel befejezését. Ha a csatornát autoinicializálásra programoztuk fel, akkor a számláló elfogyása a paraméterek (pillanatnyi számláló és cím) újratöltését idézi elő.
c) Demand Transfer Mode, késleltetett átvitel. Az adatátvitel addig folytatódik, míg a periféria kapacitását ki nem merítettük, vagyis a perifériától származó EOP (End Of Process) jel szakítja meg az átvitelt. Ha eközben várakozási állapotba kell lépnie a DMA-nak, akkor az ideiglenes regiszterekből a pillanatnyi regiszterekbe kerül a tartalmuk; mikor a processzor visszakapja a busz vezérlését, kiolvashatja a pillanatnyi regisztereket, és ezáltal információt nyerhet az átvitel menetéről. Ha a csatornát autoinicializálásra programoztuk fel, akkor a művelet szabályos befejezése, az EOP megérkezése a paraméterek (pillanatnyi számláló és cím) újratöltését idézi elő.
d) Cascade Mode, lépcsőzött mód. A DMA felhasználható lépcsőzetesen is; egy-egy átviteli csatornájához újabb DMA áramköröket kapcsolhatunk. Ez esetben a messzebb levő áramkörök vagy további lépcsőzésre szolgálnak, vagy valódi adatátvitelt végeznek, a közelebb levők megfelelő csatornái pedig lépcsőzött módban működnek.

A fenti négy lehetőségből háromban folyik valódi adatátvitel. A három mód mindegyike háromféle adatátvitelt végezhet: a memória szempontjából írást, olvasást és ellenőrzést. Az írás azt jelenti, hogy a DMA a perifériáról olvasott adatokat a memóriába írja; olvasás esetén a memóriából olvasott adatokat továbbítja a periféria felé. Az ellenőrzés egy látszólagos átvitel, melynek során a DMA paritásellenőrzést végez.

A DMA ismer egy speciális adatátviteli lehetőséget, a memória-memória átvitelt. Ekkor a Ø. csatornát használjuk beolvasásra és az 1. csatornát kivitelre; az egyik természetesen olvasásra, a másik írásra van felprogramozva. A csatornák ideiglenes regiszterei a szokott módon működnek, és az ideiglenes

regiszterbe olvassák, illetve onnan írják ki a soron következő byte-ot. Az is érdekes, hogy a 0. csatornát úgy is programozhatjuk, hogy ideiglenes címregisztere ne változzon; ez lehetővé teszi azt, hogy egyetlen értékkel írjuk felül az egész célterületet. Megjegyezzük azonban, hogy ez a lehetőség (bármilyen kellemes volna is) nem áll nyitva az IBM PC programozója előtt, hiszen a DMA 0. csatornáját a dinamikus RAM frissítésére kell használni; átprogramozása végzetes következményekkel járna.

A DMA képes arra is, hogy a művelet befejezése után újra a kezdeti értékekre készítse elő az adott csatornát; ekkor már csak egy DMA-kérésre van szükség ahhoz, hogy a teljes művelet ismét végbemenjen. Ezt nevezzük autoinicializálásnak; ennek során a DMA a pillanatnyi cím- és számlálóregiszterek értékét az alap cím- és számlálóregiszterekből állítja helyre.

Igen fontosak a prioritási kérdések is. Mi lesz akkor, ha két, egymástól független adatátvitelt végző és egymástól függetlenül felprogramozott csatornára egyidőben érkezik DMA kérés? A DMA prioritás szerint előbb az egyiket, aztán a másikat fogadja.

Két lehetőség van, amelyek közül a DMA felprogramozásakor választhatunk: a rögzített és a rotált prioritás. A rögzített prioritás esetén mindig a 0. csatorna a legmagasabb, a 3. a legalacsonyabb prioritású. A rotált prioritás esetén (amelyet azonos rangú perifériák egyidejű kezelésére használhatunk) egy befejeződött DMA művelet automatikusan rotálja a prioritást; kezdetben a 0. csatornáé volt a legmagasabb prioritás; egy rotálás után az 1. csatorna a legmagasabb prioritású, őt a 2., a 3. és végül a 0. csatorna követi.

IX.2.2. A DMA regiszterei és programozása

A DMA vezérlésére számos belső regiszter szolgál, melyeket természetesen a már megszokott módon, input-output utasításokkal érhetünk el. A regiszterek, relatív címeik és portcímeik a következők (a DMA bázisportja az IBM PC-ben a 0 című port):

a) Base Address Registers, kezdőcím alapregiszterek. Négy ilyen regiszter van (csatornánként egy); mindegyik 16 bit hosszúságú. Programból közvetlenül nem érhetők el.
   Ezek a regiszterek tartalmazzák az adott csatorna által végzendő adatátvitel kezdőcímét; a processzor a felprogramozás során az azonos csatornához tartozó pillanatnyi címregiszterrel együtt írja. A kezdőcím-regisztereket a DMA az autoinicializáláshoz használja fel.
b) Base Word Count Registers, hossz alapregiszterek. Négy ilyen

regiszter van (csatornánként egy); mindegyik 16 bit hosszúságú. Programból közvetlenül nem érhetők el.
Ezek a regiszterek tartalmazzák az adott csatorna által végzendő adatátvitel hosszát byte-okban (nem szavakban, ahogyan az a regiszterek nevéből következne). E regisztereket a processzor a felprogramozás során az azonos csatornához tartozó pillanatnyi címregiszterrel együtt írja. A hosszregisztereket a DMA az autoinicializáláshoz használja fel.

c) Current Address Registers, pillanatnyi cím regiszterek. Négy ilyen regiszter van (csatornánként egy); mindegyik 16 bit hosszúságú.
Ezek a regiszterek írhatók/olvashatók; elérésük byte-onként lehetséges. Az első elérés az alacsonyabb, a második a magasabb helyiértékű nyolc bitre vonatkozik. Címük a bázishoz viszonyítva: 00H (0. csatorna), 02H (1. csatorna), 04H (2. csatorna) és 06H (3. csatorna); portcímeik azonosak.
Adatátvitel közben az éppen felhasznált memóriabyte címét tartalmazzák. Ha a processzor éppen nincs várakozási állapotban (mert a DMA nem blokkos, hanem byte-onkénti átvitelt végez) akkor ezek a regiszterek kiolvashatók; ilyenkor az ideiglenes regiszterek pillanatnyi állapotát tartalmazzák.

d) Current Word Count Registers, pillanatnyi számlálók. Négy ilyen regiszter van (csatornánként egy); mindegyik 16 bit hosszúságú.
Ezek a regiszterek írhatók/olvashatók; elérésük byte-onként lehetséges. Az első elérés az alacsonyabb, a második a magasabb helyiértékű nyolc bitre vonatkozik. Címük a bázishoz viszonyítva: 01H (0. csatorna), 03H (1. csatorna), 05H (2. csatorna) és 07H (3. csatorna); portcímeik azonosak.
E regiszterekt az átvitel során még átviendő byte-ok számát tartalmazzák. Ha a processzor éppen nincs várakozási állapotban (mert a DMA nem blokkos, hanem byte-onkénti átvitelt végez), akkor ezek a regiszterek kiolvashatók; ilyenkor az ideiglenes regiszterek pillanatnyi állapotát tartalmazzák.

e) Temporary Address Registers, cím munkaregiszterek. Négy cím munkaregiszter van (csatornánként egy); mindegyik 16 bit hosszúságú. E regiszterek programból nem érhetők el.
A cím munkaregiszterek adatátvitel közben az éppen felhasznált memóriabyte-ra mutatnak. Ezek a regiszterek az áramkör belső nyilvántartásának céljaira szolgálnak, ők tartalmazzák az áramkör által valóban használt értékeket.

f) Temporary Word Count Registers, munkaszámlálók. Négy ilyen van (csatornánként egy); mind 16 bit hosszúságú. Programból nem érhetők el.
Ezek tartalmazzák az adatátvitel során még átviendő byte-ok számát; az áramkör belső nyilvántartására szolgálnak.

g) Command Register, parancsregiszter. Egy ilyen regiszter van, hossza 8 bit. Csak kivitelre szolgál. Címe a bázishoz viszonyítva 08H, portcíme 08H. A parancsregiszterbe kell kiírni a DMA működését szabályozó alapvető parancsokat.

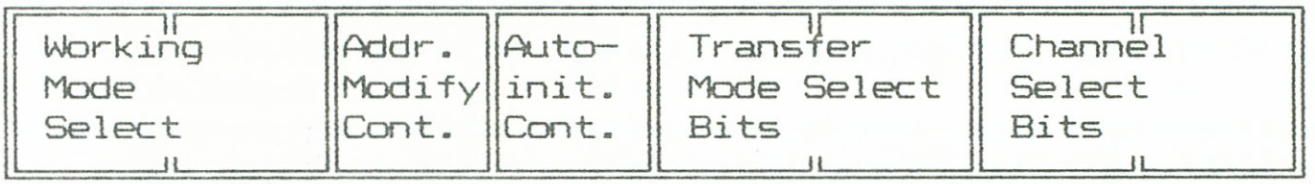
7	6	5	4	3	2	1	0
DACK Sense Cont.	DREQ Sense Cont.	Write Control	Priority Cont.	Timing Control	Controller Cont.	Chan.0 Add.H. Cont.	Mem.to Memory Cont.

A DMA parancsregisztere

A parancsregiszter bitjei:

- Memory-to-memory Control - ha értéke 1, akkor a 0. és 1. csatorna memóra-memória átvitelt végez, ha 0, akkor nem;
- Channel 0. Address Holding Control - ha a 0. bit értéke 1, akkor ez a bit szabályozza a 0. csatorna ideiglenes címregiszterének kezelését. Ha a 0. bit 0, akkor ez a bit érdektelen; ha a 0. bit 1 és e bit értéke 0, akkor a 0. csatorna címregisztere a szokásos módon változik a művelet során (memóriamásolás), ha pedig 1, akkor a cím változatlan marad (a célterület adott értékkel való végigírása);
- DMA Controller Control - a DMA vezérlő működésének engedélyezése/letiltása: az 1 bekapcsolja a DMA-t, a 0 ki;
- Timing Control - 0 értéke normál, 1 értéke pedig sűrített időzítést ír elő (ez utóbbi növeli a DMA teljesítményét);
- Priority Control - 0 értéke a rögzített, 1 értéke pedig a rotált prioritást választja ki;
- Write Control - 0 értéke a szokásos, 1 értéke pedig kiterjesztett írási módot választ;
- DREQ Sense Control - ez a bit szabja meg, hogy az áramkör milyen jelszintet tekintsen a DREQ vonalon (Dma REQuest) DMA kérésnek; 0 esetén a magas (felemelkedő), 1 esetén pedig az alacsony (leeső) érték számít DMA kérésnek;
- DACK Sense Control - ez a bit szabja meg, hogy az áramkör magas vagy alacsony jelszinttel nyugtázza a DMA kérést a DACK (Dma ACKnoledge) vonalon; 0 esetén a magas (felemelkedő), 1 érték esetén az alacsony (leeső) értékkel felel.

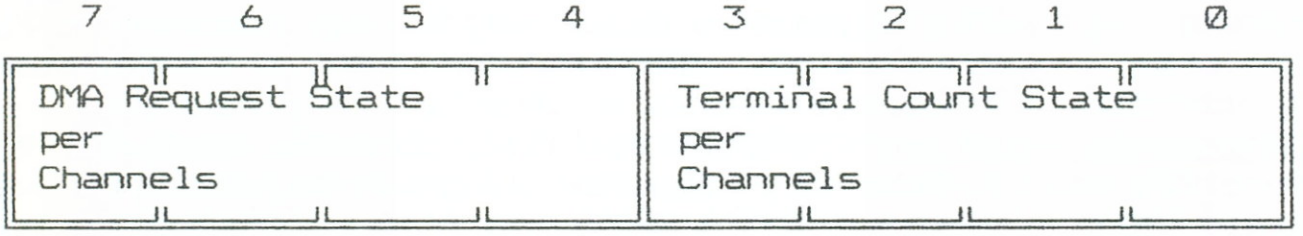
h) Mode Registers, módregiszterek. Négy módregiszter van (csatornánként egy), hosszuk 6 bit. Csak kivitelre szolgálnak. Egy port mögé van épitve valamennyi módregiszter; ha módparancsot akarunk küldeni, akkor a kiírt utasítás két bitje adja meg a választott módregisztert. Címe a bázishoz viszonyítva: 0BH, portcíme 0BH.



A DMA módregisztereinek bitkiosztása

- Channel Select Bits - ez a két bit mutatja meg, hogy melyik csatornát "célozzuk" a módparancs kiírására szolgáló egyetlen porton át. A megfeleltetés természetes:
  00 - 0. csatorna, 10 - 2. csatorna,
  01 - 1. csatorna, 11 - 3. csatorna;
- Transfer Mode Select Bits - ezek a bitek választanak az írási, olvasási és ellenőrzési módok között:
  00 - ellenőrzés
  01 - írás a perifériáról a memóriába
  10 - olvasás a memóriából a perifériára
  11 - nem legális
  E bitek érdektelenek, ha a 6-7. bit értéke 11, vagyis ha az adott csatornát lépcsőzött módban használjuk;
- Autoinitialization Control - 0 értéke letiltja, 1 értéke pedig előírja az autoinicializálást, azaz a pillanatnyi regisztereknek az alapregiszterekből való újraírását;
- Address Modify Control - ez a bit a csatorna pillanatnyi (és ideiglenes) cím regiszterének módosítási irányát szabja meg; 0 a cím növelését, 1 a csökkentését írja elő;
- Working Mode Select - ez a két bit választ a négy lehetséges üzemmód között:
  00 - Demand Mode,
  01 - Single Mode,
  10 - Block Mode,
  11 - Cascade Mode;

i) Status Register, állapotregiszter. Egy állapotregiszter van, hossza 8 bit. Csak olvasásra szolgál. Címe a bázishoz viszonyítva: 08H, portcíme 08H. A státuszregiszter információt ad a DMA összes csatornájának állapotáról.



A DMA státuszregisztere

- Terminal Count State per Channels - ezek a bitek mutatják, hogy a DMA megfelelő csatornájának pillanatnyi számlálója elérte-e már a nullát (illetve a Demand Mode esetén megérkezett-e az End Of Process jel). A megfeleltetés természetes (Ø.bit - Ø.csatorna, ..., 3.bit - 3.csatorna);
- DMA Request State per Channels - ezek a bitek mutatják, hogy a DMA megfelelő csatornájára érkezett-e akár hardware, akár program vezérelte DMA kérés. A megfeleltetés természetes (4.bit - Ø.csatorna, ..., 7.bit - 3.csatorna);

j) DMA Request Register, DMA-kérés regiszter. Egy ilyen regiszter van, hossza 3 bit. Címe a bázishoz viszonyítva Ø9H, portcíme Ø9H.

A DMA Request Register felépítése

- Channel Selection - ez a két bit választja ki a kívánt DMA csatornát. A megfeleltetés természetes:
  ØØ - Ø. csatorna, 10 - 2. csatorna
  Ø1 - 1. csatorna, 11 - 3. csatorna
- Set/Reset Request Bit - Ø értéke törli, 1 pedig egyre állítja (tölti) a kiválasztott csatornán a DMA kérést.

k) Mask Register, maszkregiszter. Egy ilyen regiszter van, hossza négy bit. Egy-egy bit egy csatornára vonatkozik. Minden DMA csatornához tartozik egy maszk, mely megszabja, hogy az adott csatorna fogadhatja-e a DMA kéréseket, vagy sem. A maszkregiszter ezt a négy bitet tartalmazza. A maszk 1 értéke tiltja, Ø értéke pedig engedélyezi a kérés fogadását. Elérése kétféleképpen történhet: egy speciális utasítással, melyben bitenként külön-külön írhatjuk elő a maszkok kívánt

értékét, vagy pedig egyszerre adjuk meg a négy maszkot. Címe a bázishoz viszonyítva az első esetben ØAH, a másodikban ØFH, portcíme azonos ezekkel az értékekkel. A parancsok:
- bitenkénti vezérlés (relatív cím ØAH, portcím azonos). Hárombites adat írandó ki, melynek alsó két bitje (a Ø. és az 1. bit) választja ki a csatornát, a következő tölti vagy törli a választott csatorna maszk bitjét (pontosan, mint a DMA-kérés regiszter esetén);
- közös vezérlés (relatív cím ØFH, portcím azonos). Az alsó négy bit természetes megfeleltetéssel vezérli a négy csatorna maszk bitjét, az 1 érték tölti a maszkot (tiltja a kérés fogadását), a Ø pedig törli (engedélyezi a kérés fogadását).

1) Temporary Register, ideiglenes regiszter. Egy ilyen regiszter van, hossza nyolc bit. Csak olvasásra szolgál. Címe a bázishoz viszonyítva: ØDH, portcíme ØDH.
E regiszter csak a memória-memória átvitel során játszik szerepet; az éppen átmásolt byte-ot tartalmazza. Egyszerű átvitel esetén a processzor is hozzájuthat az éppen másolt byte-hoz.

A DMA a fenti lehetőségeken kívül még két porton át érhető el. Az ide kiírt értékek érdektelenek; ha a program egyáltalán végez egy kiírást ide, akkor eléri a kívánt hatást.

m) Clear first/last Flip-flop, a tizenhatbites regiszterek elérésének előkészítése. Cím a bázisporthoz viszonyíta ØCH, portcím ugyanaz.
A ØCH portra kiírt akármilyen érték alapállapotba hozza azt a bitet, mely jelzi, hogy a tizenhatbites regiszterek (Current Address és Current Word Count) alacsony vagy magas helyiértékű byte-jainak elérése következik. Ha ezeket a regisztereket módosítani vagy olvasni akarjuk, akkor előbb ki kell küldeni egy Clear First/Last Flip-Flop utasítást, utána pedig két konzekutív I/O művelettel kell módosítani vagy kiolvasni a kívánt regisztert (előbb az alacsony, azután a magas helyiértékű byte-ot).

n) Master Reset, teljes előkészítés. Címe a bázisporthoz viszonyítva ØDH, a portcím ugyanaz. Ha a ØDH portra bármilyen értéket kiírunk, egyenértékű az áramkör hardware reset-jével, minden elemét alapállapotba hozza.

Az áramkörnek nem részei ugyan, de az IBM PC/XT-n fontos szerepet játszanak a lapregiszterek:

o) DMA Page Registers, lapregiszterek. Négy lapregiszter van (csatornánként egy); hosszuk 4 bit. Portcímük 80H - 83H. E regiszterek a DMA speciális bővítései, melyek célja, hogy a DMA képes legyen az 1 Mb nagyságú memória bármely pontjára/ról végezni az adatátvitelt, vagyis e regiszterek teszik alkalmassá a DMA-t az Intel 8086 processzorral való együttműködésre.
A regiszterek négy bitjére a teljes húszbites fizikai cím legmagasabb helyiértékű négy bitjét kell kiküldeni. E regiszterekről többet nem állt módomban megtudni; mindenesetre a ROM BIOS programjainak tanulmányozása azzal a furcsa eredménnyel járt, hogy ezek a regiszterek a természetes hozzárendelés fordítottjával kapcsolódnak a csatornákhoz. A leírásokból és a programokból is következik, hogy a diskette kezelésére a DMA 2. csatornáját használják (a felprogramozás során erre vonatkozó utasításokat ad ki a BIOS). A lapregiszterek közül pedig a 81H címen levőt használja fel, ami első látásra meglepő. Ugyanilyen meglepetést okoz, hogy a teljes rendszer előkészítése során a 83H portra küld ki a rendszer egy 0 értéket, pedig ekkor logikusan a 0. csatornához fordul. Ezek sugallták azt, hogy a lapregiszterek fordítva vannak bekötve: a 0. csatornához a 83H, az 1.-höz a 82H és így tovább.

IX.3. 8253 Timer/Counter

A 8253 típusjelű Timer/Counter (időzítő és számláló áramkör) egyike a gép legfontosabb elemeinek, bár jelentőségét első pillantásra nem tudjuk felmérni. Voltaképpen három, egymástól teljesen független számlálót tartalmaz. Ezeket a számlálókat tetszőlegesen programozhatjuk; a 8253 képes három különböző, egymástól független időtartam mérésére, és az idők lejártának jelzésére.

Az IBM PC/XT alapkártyájára épített mindhárom csatornát határozott célra használják.

> Megjegyzés: egyes bővítőkártyák (például az itt nem ismertetett SDLC adapter) saját Timer/Counter áramkört tartalmaznak.

A nulladik csatorna szolgál a gép órájának vezérlésére: másodpercenként körülbelül tizennyolcszor váltja ki a 08H sorszámú hardware-interruptot. Az interrupt rutin azután megnöveli az eltelt idő mérésére szolgáló négy byte hosszúságú számlálót. E kimenet értéke a 62H port (a PPI C portja) egy bitjéhez is hozzá van kötve: az 5. bit egyezik meg a 2. csatorna kimeneti jelével.

Az egyes csatorna a dinamikus memória frissítésében játszik szerepet: ennek kimenő jele van a DMA (lásd feljebb) nulladik csatornájához kapcsolva. Végül a második csatorna vezérli a hangszóró által kibocsátott hang frekvenciáját. Ennek kimenete közvetlenül a hangszóróhoz van kötve, így e csatorna felprogramozása és a hangszóró bekapcsolása után a hangszóró működése automatikussá válik mindaddig, amíg csak ki nem kapcsoljuk.

Ahhoz, hogy egy csatorna működési elvét megértsük, néhány alapfogalommal meg kell ismerkednünk. Minden csatornához tartozik egy számláló, egy tizenhat bites belső regiszter, melynek értékét a felprogramozás során adjuk meg. Minden csatornához tartozik egy kimenet, és egy bemeneti vezérlő kapu (Gate). Bizonyos üzemmódok függetlenek a kapu állapotától, másqk csak akkor működnek, ha a kapun kapott jel magas (egyes).

A Timer legfontosabb bemeneti jele a gép hardware órája, a "clock", amely a gép elemeinek belső szinkronizációjához szükséges ütemezést adja (egyébként a clock frekvenciájával szokták jellemezni egy számítógép működési sebességét). A Timer felprogramozása (és ha szükséges, a Gate bekapcsolása) után a Timer minden órajelre eggyel csökkenti a számlálót. Mikor a számláló elfogy (vagy valamilyen speciális értéket elér), akkor a Timer a kimeneti kapun át egy impulzust ad; ezzel jelzi az idő leteltét.

A számláló egyes üzemmódokban újratöltődik, és a számlálás elölről kezdődik, a számláló elfogyásakor pedig újabb kimeneti jel keletkezik. A Timer így nemcsak időtartamok mérésére, várakozás vezérlésére szolgál; felhasználhatjuk programból vezérelt ritmikus impulzusok adására is.

IX.3.1. A Timer/Counter üzemmódjai

A Timer/Counter csatornánként hatféle üzemmódot ismer; bármelyik csatornáját bármelyik üzemmódra felprogramozhatjuk, a többi csatornától függetlenül. Lássuk a lehetőségeket!

- **0. mód:** felfüggeszthető időmérés (Interrupt on Terminal Count). Ebben az üzemmódban a felprogramozás idején automatikusan lemegy a kimeneti szint. Az előkészítés befejeződése

után megkezdődik a számlálás, és mikor a számláló értéke nullára csökken, akkor a kimeneti jel felemelkedik és fent marad addig, amíg új számlálót nem adunk meg, vagy pedig újra fel nem programozzuk a csatornát. A kapu szintjének magasnak kell lennie.
A számlálást fel lehet függeszteni azzal, ha a kapu magas értékét a számlálás folyamata alatt levisszük. Mikor a bemenet értéke ismét magas lesz, a számlálás ott folytatódik, ahol előzőleg abbamaradt.
Ha működés közben módosítjuk a számlálót, akkor az időmérés megszakad; a számláló első byte-jának töltése felfüggeszti a számlálást, amely az új értékről indul újra, mikor a második byte-ot is kiírtuk.

Ez a mód megfelelő hardware-környezetben lehetőséget ad arra, hogy külön mérjük az interrupt vagy felhasználói szinten eltöltött időt; hogy a számlálás ne valósidejű legyen (ne folytatódjék állandóan), hanem csak akkor, mikor a felhasználói szinten futó program számára is "telik az idő" (vagyis amikor fut).

- 1. mód: újraindítható időmérés (Programmable One-Shot). A kimenet magas szinten lesz a felprogramozást követően mindaddig, amíg csak fel nem megy a szint a kapubemeneten. Ekkor indul el a számlálás; a kimenet szintje pedig leesik. Mikor a számláló nullára csökken, a kimenet szintje újra felmegy.
  Ha számlálás közben levesszük a kapubemenet szintjét, akkor a számlálás leáll, és a kapu újabb felemelése után elölről kezdődik. A kimenet szintje ezalatt mindvégig alacsony marad.

Megfelelő hardware-környezetben ez a mód alkalmas arra, hogy egy jel megváltozásától fogva megadott ideig várakozzunk. Ha ugyanis a szükséges időt megadjuk, és a figyelendő jel be van kötve a kapubemenetre, akkor a számlálás a jel megérkezésekor indul el. Ha a kivárás ideje alatt a jel leesik, akkor a számlálás megszakad, majd az újabb felemelkedéskor elölről indul; ha tehát a kimeneten megkapjuk a magas szintet, akkor a várt jel megérkezése óta pontosan a kívánt idő telt el.

- 2. mód: időközönként ismételt jel (Rate Generator). Ebben a módban a felprogramozás után az output jel magas lesz; a számláló elfogyásakor egy órajelnyi időre leesik; a számláló újra megkapja eredeti értékét, újraindul, a kimeneti jel ismét felmegy. Ezáltal ebben a módban a Timer megadott időközönként egy impulzust ad a kimeneten át. A számlálót menet közben módosíthatjuk; ez nincs hatással az éppen folyó cik-

lusra, csak annak befejezése után jut érvényre.
Ha a kapu szintje alacsony, akkor a kimenet szintje magas lesz, és a számlálás megszakad. A kapu szintjének felemelése után a számlálás a kezdeti értékről indul. Ez lehetővé teszi a jeladás hardware-szinkronizálását. A felprogramozás (a módparancs kiadása) után a kimenet magas, és számlálás csak a számláló értékének betöltése után kezdődik meg. Ez software-úton való szinkronizálást jelent.

Az 1. csatornát ebben a módban használják a DRAM frissítésének vezérlésére; a kimenet leeső szintje adja meg a jelet a DMA-nak a memóriafrissítés elkezdésére.

- 3. mód: ki-be jel generálása egyenletes időközönként (négyszögjelgenerátor, Square Wave Rate Generator). A Timer a felprogramozás után csökkenteni kezdi a számlálót, majd annak elfogyásakor ismét betölti az eredeti értéket, és elölről kezdi a számlálást. A kimeneten a számlálás időtartamának első felében magas, a második felében pedig alacsony jelet kapunk. A valóságban igen ravasz módon oldották meg ezt a kérdést: páros számláló esetén minden órajelre két csökkentés következik be, és a számláló kifogyása után újratöltődik az eredeti érték, a kimenet szintje megváltozik, és a folyamat elölről indul. Ha a számláló értéke páratlan, akkor a kimenet szintje fentől indul; a számláló elsőre eggyel, a továbbiakban kettővel csökken (így az érték felénél fél órajellel nagyobb időtartamot kapunk), a nulla elérésekor a kimenet szintje leesik, a számláló pedig az újratöltés után először hárommal, utána kettesével csökken. Ezzel a teljes időtartam felénél fél órajellel rövidebb értéket kapunk.
  Ha a kapu szintje alacsony, akkor a kimenet szintje magas lesz, és a számlálás megszakad. A kapu szintjének felemelése után a számlálás a kezdeti értékről indul. Ez lehetővé teszi a jeladás hardware-szinkronizálását.

A 3. mód a hanggenerálás során alkalmazott eljárás; a Timernek a hangszóróval összekötött 2. csatornája szabályos időközönként ad egyszer be-, egyszer pedig kikapcsolási impulzust.

- 4. mód: programvezérelt jeladás (Software Triggered Strobe). A módparancs kiadása után a kimenet szintje felmegy. A számláló beállítása indítja a számlálást, mialatt a kimenet végig magas. A számláló lefutása után a kimenet egy órajelnyi időre lemegy, majd ismét a magas szintre vált.

Ha időközben a kapu szintje lemegy, akkor a számlálás megszakad. Megjegyzem, hogy a rendelkezésemre álló Intel katalógus idődiagramjai és táblázatai nekem ellentmondónak tűnnek; a diagram szerint ebben az esetben a számlálás újraindul, míg a táblázat szerint a kapu szintjének ismételt felemelése nincs hatással rá. A kérdés mindenesetre érdekes.

- 5. mód: hardware-vezérelt jeladás (Hardware Triggered Strobe). A módparancs, valamint a számláló megadása után a kimenet magas, a Timer megvárja a kapu szintjének felemelkedését, és ekkor kezdi meg a számlálást. Ha a számláló elfogy, a kimenet egy órajelnyi időre leesik, majd ismét a magas szintre emelkedik. Ha a számlálás idején a kapu szintje leesik, akkor a számlálás folytatódik; ha a számláló lefutása előtt ismét felmegy a kapu szintje, akkor a számláló újratöltődik.

IX.3.2. A Timer/Counter portjai és programozása

Az áramkört négy porton át érhetjük el. Az alapporttól számított három első port szolgál a számlálók megadására vagy kiolvasására (méghozzá csatornánként egy port), míg a negyedik (melynek offsetje az alapporthoz viszonyítva 3) a különféle működésvezérlő parancsok kiadására; ezt hívjuk vezérlőportnak. Egy csatorna felprogramozása egy módparancs kiadásával kezdődik, amelyet természetesen a vezérlőportra küldünk. Ezzel választjuk ki az adott csatornát, megszabjuk a számláló megadásának módját és a kívánt üzemmódot. Ezután adhatjuk ki a számláló értékét a megfelelő porton keresztül. A számláló újratöltése általában megengedett és semmilyen komolyabb következménnyel nem jár. Ha a Timer egy csatornája már fel van programozva, akkor a számláló újratöltése a korábban megadott üzemmódra vonatkozik (bővebben lásd fent).

Az IBM PC/XT-n a Timer/Counter alapportjának címe 40H; így a vezérlőport a 43H címen található, a 0, 1 és 2. csatorna adatportjai a 40H, 41H és a 42H címűek.

7	6	5	4	3	2	1	0
Select Counter		Read/Load mode		Select Mode			Sel. BCD mode
SC1	SC0	RL1	RL0	M2	M1	M0	

A Timer/Counter vezérlőparancsa

- Select BCD mode - e bit szabja meg azt, hogy a címzett csatornához tartozó tizenhat bites számlálót hogy kezelje a Timer: négyjegyű BCD számként vagy valódi tizenhatbites értékként. Az előbbi esetben a számláló legnagyobb értéke decimális 9999, míg az utóbbiban 65535 ($2^{16}-1$). A bit 1 értéke választja a BCD módot, a nullás érték pedig bináris módot.
- Select Mode - e három bittel választjuk meg az adott csatorna üzemmódját. A bitek jelentése elég természetes:

M2	M1	M0	
0	0	0	0 üzemmód,
0	0	1	1 üzemmód,
(x)	1	0	2 üzemmód,
(x)	1	1	3 üzemmód,
1	0	0	4 üzemmód,
1	0	1	5 üzemmód,

  ahol az (x) azt jelenti, hogy az adott bit állása érdektelen. A Timer, mint látható, nem bináris értékre, hanem bitkombinációra vizsgál, ami ember számára elvontabb.
- Read/Load Mode - ez a két bit az adott csatornára kiküldendő számlálóértékek alakjának meghatározására szolgál. A kiadandó tizenhatbites értéket egy nyolcbites lukon át kell kiküldeni, ezért általában két lépésre van szükség. Van azonban lehetőség arra, hogy csak az egyik byte-ot érjük el (például az alacsonyabb helyiértékű byte időnkénti kiolvasását álvéletlen számok generálására használhatjuk fel). A kombinációk:

RL1	RL0	
0	0	Counter Latching Operation - e parancs hatására a Timer az SC0-SC1 bitekben megadott számláló pillanatnyi értékét (a számlálás folytatásával) kimenti egy belső regiszterbe; a későbbi kiolvasás ezt a befagyasztott értéket adja meg.
0	1	a számláló alacsonyabb helyiértékű byte-jának beállítása/kiolvasása
1	0	a számláló magasabb helyiértékű byte-jának beállítása/kiolvasása
1	1	előbb az alacsonyabb, azután a magasabb helyiértékű byte beállítása/kiolvasása

SC1	SC0	
0	0	0. csatorna kiválasztása
0	1	1. csatorna kiválasztása
1	0	2. csatorna kiválasztása
1	1	szabálytalan érték

A Timer programozása a következő logikai lépésekben végzendő: először a legmagasabb című porton át kiadunk egy módparan-

csot, mellyel megszabjuk a számláló kezelésének módját, a kívánt üzemmódot, megadjuk, hogy milyen rendben írjuk ki vagy olvassuk be a kiválasztott csatorna számlálóját (ezt a rendet aztán kínos pontossággal be kell tartanunk), és megadjuk, melyik csatornára vonatkozik mindez.

A következő lépés a kiválasztott csatorna felprogramozásának befejezése, azaz a kívánt számlálóérték megadása a megfelelő I/O porton át. Ha ugyanolyan üzemmód mellett más számlálóértéket akarunk megadni, akkor nem szükséges újabb módparancs kiadása; egyszerűen kiírjuk az új számláló értékét.

Mint fentebb említettük, hasznos lehet a számláló értékének kiolvasása is. Egy adott módparancs kiadása után bármikor kiolvashatjuk a számláló értékét; persze ekkor is be kell tartani a számláló elérésének rendjét. A kiolvasás során van még egy kis probléma: nem tanácsos futás közben kiolvasni a számlálót, különösen akkor, ha nemcsak az egyik byte-ot, hanem mindkettőt ki akarjuk olvasni. A számláló ugyanis továbbszalad közben; az alsó és a felső byte kiolvasása közben minden megváltozik, és teljesen hamis értéket kaphatunk. E nehézség leküzdésére szolgál a Counter Latching Operation, mely a számláló értékét befagyasztja, és a kiolvasás ezt az értéket szolgáltatja.

IX.4. 8259 Interrupt Controller

Nézzük meg vázlatosan, mi történik egy hardware-interrupt során attól a pillanattól kezdve, hogy valamelyik perifériális egység "fejében" megfogan az az ötlet, hogy kiszolgálást kérjen a processzortól, addig, amíg a kérését kielégítő rutin be nem fejeződik.

A hardware-interrupt generálására képes perifériális egységek nem közvetlenül a processzor INT lábához vannak kapcsolva, hanem egy áttételen, az interrupt vezérlőn át fordulhatnak csak a processzorhoz. Ez egy interface a processzor és a perifériakontrollerek között. Ez az áramkör több- (nyolc) csatornás, ami azt jelenti, hogy egyszerre több perifériális egységtől képes fogadni az interrupt kéréseket, s ezeket megfelelően sorrendbe állítva továbbítani a processzornak.

A lépések nagyjából a következők: valamelyik perifériakontroller érzékeli, hogy sürgős beavatkozást igénylő esemény következett be. Ekkor saját interrupt-vonalán át jelzést küld az interrupt vezérlő egyik csatornájára. Az megvizsgálja, hogy megengedett-e az interrupt továbbítása. Ha a processzor éppen egy másik, az éppen aktívnál magasabb vagy azzal megegyező pri-

oritású interrupt kiszolgálásával van elfoglalva, akkor az interrupt vezérlő nem adja fel a kérést; mint ahogy akkor sem, ha az adott csatorna le van tiltva (lásd alább). Ha az interrupt-kérés nem továbbítható, akkor az interrupt vezérlő sorbaállítja és váratja mindaddig, míg nem tudja továbbítani.

A továbbított és a processzor által fogadott interrupt-kérés kikényszeríti a meghatározott kezelő rutin lefuttatását (ha a processzor korábban kapott egy CLI utasítást, akkor az interrupt vezérlő hiába adja az interrupt-kérést, a processzor nem fogadja). Általában a rutin befejezése után engedhető meg újabb interrupt-kérés fogadása. A rutin befejezését valamilyen módon jelezni kell az interrupt vezérlőnek; ezután fogja az interrupt vezérlő továbbítani az egyéb (alacsonyabb prioritású) kéréseket.

Az alábbiakban részletesebben ismertetjük az interrupt vezérlő működését; úgy tűnik azonban, hogy ezen információk nagy része lényegtelen. Az IBM PC/XT programozása során semmi értelme nincs annak, hogy az interrupt vezérlő programozásával bravúroskodjunk. Mégis előfordulhat, hogy ennek a tudásnak hasznát vesszük, különösen akkor, ha újabb, hardware interrupttal vezérelendő elemeket akarunk a gépünkbe építeni.

IX.4.1. Az interrupt vezérlő belső regiszterei

A vezérlőáramkör belső felépítéséből számunkra csak néhány regiszternek az ismerete fontos, amelyekben a vezérlő nyilvántartja a fogadható, a beérkezett és a kiszolgálás alatt álló interrupt-kéréseket. E regiszterek az Interrupt Mask Register (a továbbiakban IMR), az Interrupt Request Register (IR) és Interrupt Service Register (IS).

A regiszterek bitjei megfelelnek a vezérlő nyolc csatornájának. Ezeket nullától hétig számozzuk; a nulladik bit felel meg a nulladik csatornának és így tovább.

Az IMR egyes bitjeinek nulla értéke azt jelzi, hogy az adott vonalon érkező interrupt-kérés fogadható; ha egy bit egyes, akkor a megfelelő csatornán érkező interrupt-kérést a vezérlő nem fogadhatja.

Ha egy fogadható interrupt-kérés érkezik valamelyik szinten, akkor a nyolcbites IR megfelelő bitje egyes értéket kap. Az interrupt vezérlő mindig a legmagasabb prioritásút (általában a legalacsonyabb sorszámút) továbbítja a processzornak. Ha az fogadja a kérést, akkor az IS regiszter megfelelő bitje áll egyre. A bitek az interrupt végén (mikor a processzor jelzi az interrupt rutin befejeződését) kapják vissza eredeti Ø értéküket.

IX.4.2. Interrupt-kérés továbbítása

A továbbítás folyamata: az interrupt vezérlő saját megfelelő kimenetén át küldött jellel "megcsiklandozza" a processzor INT lábát. Az egy pillanatra reflexszerűen felkapja csiklandozott talpát, és gyorsan befejezi azt a gépikódú utasítást, amelynek végrehajtása éppen folyamatban van. Ezután saját INTA (INTerrupt Acknowledge) kivezetésén át jelzi az interrupt vezérlőnek, hogy kész az interrupt fogadására. Az interrupt vezérlő erre valamilyen módon megadja a processzornak, hogy melyik csatornáján jött az interrupt-kérés, azonosítja a kiszolgálást igénylő perifériát. Az IBM PC/XT-ben ez az interrupt sorszámának megadását jelenti.

Az interrupt vezérlő nyolc vonala általában prioritási sorrendet is jelent. A nulladik vonalra kötött periféria kapja a legnagyobb, a hetedikre kötött pedig a legkisebb prioritást. A prioritási rend egyébként változhat (lásd alább). Ez azt jelenti, hogy amennyiben több vonalon van egyszerre interrupt-kérés, a vezérlő mindig a magasabb prioritásút továbbítja. A fentebb leírtakból következik, hogy a magasabb prioritású interrupt megszakíthatja az alacsonyabb prioritásút, de fordítva nem; tehát például az IBM PC/XT-n a Timer interruptja minden olyan interruptot megszakíthat, melynek során a processzor számára engedélyezik további interrupt fogadását (pl. klaviatúra hardware interrupt), a klaviatúra interruptja nem szakítja meg a Timerét, és így tovább (lásd a "ROM BIOS interruptok kiosztása" függeléket). Természetesen nem minden interrupt-rutinban és nem is minden pillanatban engedélyezhető a további megszakítások fogadása (erről bővebben az "Aszinkron vonal kezelése" fejezet utolsó példaprogramjánál).

IX.4.3. Az interrupt-eljárás vége

Az eljárás befejeződését valahogyan az interrupt vezérlő tudomására kell hozni. Ez kétféleképpen lehetséges: automatikusan vagy normál módon. Az automatikus lezárás azt jelenti, hogy az interrupt vezérlő az interrupt beadásával meg is tette kötelességét; miután a processzor fogadta az interruptot, ő máris kész egy újabb megszakítása beadására. A normál módban az eljárás befejeződését egy speciális utasítással kell jelezni az interrupt vezérlőnek. Az IBM PC/XT a normál üzemmódot használja. Az ehhez kiadandó parancsot a továbbiakban EOI-nek (End Of Interruptnak) nevezzük. Az EOI parancsban, az interrupt eljárás végét jelző bit mellett egyéb paramétereket is megszabhatunk.

Egyfelől előírhatunk automatikus vagy meghatározott prioritás-rotálást, másfelől pedig, a specifikus EOI parancs segítségével megadhatjuk, hogy a továbbiakban (az interrupt rendszer teljes felszabadításáig) melyik további interruptot akarjuk fogadni. Ez lehetőséget ad arra, hogy ne csak a pillanatnyinál magasabb, hanem alacsonyabb prioritású interrupt is érvényre juthasson.

IX.4.4. A felhasználható processzorok

Az INTEL 8259 típusjelű interrupt vezérlője kétféle üzemmódban működhet: MCS-80/85 és 8086/8088 módban. A legfontosabb különbség a két mód között a periféria azonosításában, vagyis az interrupt-szint megadásában van. Az előbbi módban a processzor három ízben küld jelet az interrupt vezérlőnek az INTA lábon keresztül. Az interrupt vezérlő az első jelre egy CALL utasítás gépikódját, a további kettőre pedig az interruptot kiszolgáló rutin belépési címének alsó és felső byte-ját küldi ki a processzor adatbuszára, vagyis úgy tesz, mintha a processzor egy CALL utasítást olvasott volna a program szövegében.

A számunkra fontosabb 8086/8088 módban a processzor csak két INTA jelet küld. Az első hatására az interrupt vezérlő befagyasztja a pillanatnyi állapotot, hogy a most beküldendő interruptnál csak magasabb prioritású interruptok éleződjenek. A második jelre egy nyolcbites adatot küld ki az adatbuszon át a processzornak. Ez választja ki a kívánt interrupt-szintet az Intel 8086 processzor 256 szintje közül. A nyolc bitből a felső ötöt az interrupt vezérlő felprogramozásakor mi adhatjuk meg, míg a három alsó bit azonosítja az interrupt vezérlő által kezelt nyolc vonal valamelyikét. Tehát az interrupt vezérlő bármely nyolcast ki tudja jelölni (a szintnyolcasoknak nyolccal osztható szinten kell kezdődni).

IX.4.5. Az interrupt-szintek kezelése

Az interrupt vezérlőt úgy is működtethetjük, hogy az általa kezelt szintek egyikét-másikát letiltjuk; ez növeli flexibilitását. Tehát a processzor interrupt-fogadási készségén (melyet az STI/CLI utasításpárral vezérelhetünk) letilthatjuk különkülön bármely hardware-interrupt fogadását. Ennek komoly jelentősége lehet minden olyan esetben, mikor feltétlenül biztosítani kell egy alacsony prioritású interrupt beérkezését, vagy ki kell védeni egy magas prioritású interruptot (pl. a klaviatúra hardware interruptját).

Az interrupt vezérlő az általa kezelt interrupt-szintek prioritását nem mereven, "bedrótozva" kezeli; programból felcserélhetjük, rotálhatjuk a prioritási sorrendet. Ennek az IBM PC/XT-hez hasonló általános célú számítógép esetében valószínűleg nincs jelentősége; annál hasznosabb lehet olyan esetben, ha valamilyen célhardware-ben alkalmazzuk ezt az áramkört. Könnyen előfordulhat, hogy lényegében véve azonos szerepű és jogkörű perifériák egy halmazát (például hálózati kapcsolatot biztosító adatvonalakat) kezelünk. Ekkor a vonalak nagyjából egyenrangúak, mégis kell valami prioritási sorrendnek lennie. A prioritás rotálásával elérhető, hogy mindig az utoljára kiszolgált vonal kapja a legalacsonyabb prioritást, ezzel utat engedve azoknak, amelyek már régen várakoznak.

A rotálásra kétféle eljárás van: az automatikus és a specifikus rotálás. Az automatikus rotálás után az éppen kiszolgált interrupt-szint kapja a legalacsonyabb prioritást, így esetleg kénytelen lesz megvárni a hét egyéb szint kiszolgálását ahhoz, hogy újra érvényre juthasson. Ha pl. a 3. szint kezelése fejeződött be, és automatikus rotálással kombinált EOI parancsot adunk ki, akkor a 3. szint kapja a legalacsonyabb, míg a 4. a legmagasabb prioritást (a sorrend: 4, 5, 6, 7, 0, 1, 2 és 3).

A specifikus rotálás esetén előírhatjuk, melyik szint legyen a legalacsonyabb prioritású. Ekkor pedig (legalábbis a következő parancs kiadásáig) rögzítjük a prioritási sorrendet. Tegyük fel, hogy a 6. szintet választottuk ki, mint legalacsonyabb prioritást; ekkor a sorrend a következő: 7, 0, 1, ..., 6. Mindenfajta rotálást a 2. vezérlőparanccsal hajthatunk végre.

IX.4.6. Több interrupt vezérlő egy rendszerben

Még egy fontos és hasznos sajátosságról kell megemlékeznünk, ez pedig az interrupt vezérlők lépcsőzetes felhasználásának lehetősége. Mivel egy vezérlő csak nyolc szinten tud interruptkéréseket fogadni, nagyobb rendszerek megvalósításában több interrupt vezérlőt kell alkalmaznunk. Ezeket azonban csak "sorba" kapcsolhatjuk. Egymás mellé kötésük lehetetlen, hiszen a processzornak csak egy interrupt-vonala van; másfelől ez teljesen tisztázatlan prioritási sorrendet eredményezne. A lépcsőzetes üzemmódban egy kijelölt interrupt vezérlő a főnök (master), a többiek beosztott (slave, szolga) szerepet töltenek be. A beosztott vezérlők természetesen nem a processzornak, hanem a náluk magasabb rangú vezérlőnek küldik kéréseiket, és annak közvetítésével kapnak engedélyt az azonosító adat beküldésére. Az IBM PC/XT-ben csak egy, az AT-ben két, lépcsőzetesen kötött interrupt vezérlő van.

A lépcsőzetesen felhasznált interrupt vezérlők kezelése kissé bonyolultabb; a kiadandó parancsokat nem csak az adott interruptot kezelő alárendelt vezérlőnek, hanem a főnöknek is el kell küldeni.

A lépcsőzetesen felhasznált interrupt vezérlők esetében van még egy kiegészítő lehetőség. Szokásos esetben, mint fent olvashattuk, csak magasabb prioritású interrupt-kérés szakíthatja meg az éppen aktív interrupt kiszolgálását. Lépcsőzött módban azonban egyáltalán nem egyértelmű a prioritás; minden egyes alárendelt vezérlő be van kötve a fő vezérlő egy csatornájára. A prioritás értelmezhető a fő- és alárendelt vezérlő szerint is. Ilyen esetben van különbség a két lehetőség: a normál (eredeti neve Fully Nested Mode) és a megengedő alárendelés (Special Fully Nested Mode) között.

> Megjegyzés: mint látható, a fordításnak semmi köze az eredeti elnevezéshez; azonban nekem úgy tűnt, hogy az eredetinek meg semmi köze a jelentéshez...

A Fully Nested Mode esetén az alárendelt vezérlő interruptját csak a fő interruptvezérlő magasabb csatornáján érkező kérés szakíthatja meg; a Special Fully Nested Mode alkalmazása esetén az alárendelt vezérlő interruptját mind a fő interrupt vezérlő magasabb csatornáján, mind pedig az adott slave magasabb prioritású csatornáján érkező interrupt-kérés megszakíthatja.

IX.4.7. Pszeudo-interruptok; pollozás

Bizonyos esetekben kívánatos lehet, hogy ne engedélyezzük interruptok fogadását, hanem mi magunk vizsgáljuk meg időről időre azt, hogy kért-e kiszolgálást valamely periféria. Ezt az eljárást nevezzük pollozásnak. A 3. vezérlőparancs ad lehetőséget erre a műveletre. Ha az ott szereplő POLL bitet egybe állítjuk, és így adjuk ki a 3. vezérlő parancsot, akkor az interrupt vezérlő nem továbbítja az interrupt-kéréseket, de mindent ugyanúgy nyilvántart, mintha elküldte volna őket. Nyilvántartja azt, hogy éppen melyiknek a kiszolgálása folyik (ezt az Interrupt Status regiszterből lehet megtudni). Ezt a regisztert egyébként az interrupt vezérlő az End Of Interrupt utasítások (2. vezérlőparancs) segítségével tartja karban.

IX.4.8. Az interrupt vezérlő előkészítése

Az interrupt vezérlő felhasználása két fázisra tagolható. Az első a felprogramozás, a második az üzemszerű felhasználás közben kiadott parancsok (ICW, Initialization Control Word, illetve OCW, Operation Control Word). A vezérlő előkészítése során négy parancsot kell kiküldeni az interrupt vezérlő két portjának egyikére. Az első parancsot az alapportra (az IBM PC/XT-n a 20H), míg a továbbiakat a másodikra (az IBM PC/XT-n a 21H):

1. előkészítő parancs, ICW1 (kiküldendő a 20H portra):

7	6	5	4	3	2	1	0
Az interrupt vektor címének felső három bitje (A7-A5)			1	Szint vagy él	CALL cím int.	Single vagy Casc.	4.elő-kész. szüks.

- 0. bit (ICW-4 needed) egyes értéke esetén szükséges a negyedik előkészítő parancs is, nulla értéke esetén nem; ebben az esetben az interrupt vezérlő úgy tekinti, hogy a negyedik előkészítő parancs "elhangzott", és minden bitje nulla volt (lásd a 4. parancs leírását);
- 1. bit (Single or Cascade Mode) 1 értéke normál, 0 értéke pedig lépcsőzött módot jelent;
- 2. bit (CALL address Interval) egyes értéke nyolcbites címintervallumot jelent, nulla értéke pedig négybitest (utóbbi egy ugrótáblában való szintkijelölésére alkalmas; csak MSC 80/85;
- 3. bit (Level or Edge Triggered Mode) egyes vagy nulla értéke az interrupt vezérlőt szint- vagy élérzékenységre állítja. Ettől függ, hogy a beérkező interrupt kérést a jel felfutó élére, vagy az aktív szint elérésére fogadja-e a vezérlő;
- 5-7. bit (Interrupt Vector Addresses A5-A7) adják MCS-80/85 módban az interrupt vektor címének 5-7. bitjeit; lásd ICW2.

2. előkészítő parancs - ICW2 (kiküldendő a 21H portra):

7	6	5	4	3	2	1	0
A15 vagy T7	A14 vagy T6	A13 vagy T5	A12 vagy T4	A11 vagy T3	A10	A9	A8

Mint talán látható, e bitek az interrupt-szint azonosításához szükséges adatokat adják meg. Az A15-A8 bitek az MSC-80/85 mód-

ban, míg a T7-T3 bitek 8086/8088 módban határozzák meg az interrupt vektor címét (utóbbi esetben tehát ezek a bitek szabják meg, melyik szintnyolcast használja az interrupt vezérlő saját nyolc interrupt-szintje azonosítására).

3. előkészítő parancs, ICW3 (kiküldendő a 21H portra):

7	6	5	4	3	2	1	0
S7 vagy 0	S6 vagy 0	S5 vagy 0	S4 vagy 0	S3 vagy 0	S2	S1 vagy	S0
					Slave azonosító		

A Master interrupt vezérlő számára ez a harmadik parancs azt adja meg, hogy melyik szintjére van Slave vezérlő bekötve. A megfelelő bit egyes értéke "beosztott" interrupt vezérlőt jelent, míg a nulla érték esetén az adott szinten (ha van valami egyáltalán) közönséges perifériális egység van.
Slave vezérlő számára az alsó három bit lényeges: itt adjuk meg a vezérlő azonosítóját mint bináris számot (a 000 érték a 0. Slave vezérlőt, míg az 111 a 7.-et jelenti). Ez azt adja meg, hogy az adott interrupt vezérlő a Master vezérlő melyik szintjére van bekötve.
Az IBM AT-n ez a szint 2 (a főnöki beosztású interrupt vezérlő 2. csatornájára van kötve az alárendelt vezérlő). Az alárendelt vezérlő portcímei egyébként: A0H és A1H.

4. előkészítő parancs, ICW4 (kiküldendő a 21H portra):

7	6	5	4	3	2	1	0
0	0	0	Nest. mód	Buff. mód	Mast. vagy Slave	EOI meg- adás	80/85 vagy 86/88

Láthatóan ez a legfontosabb, mégis elmaradhat bizonyos esetekben: ha az interrupt vezérlő egyedül van, és alapértelmezett (MCS-80/85) módban használjuk a vezérlőt. A bitek ismertetése:
- 0. bit (Microprocessor Mode) 1 értéke 8086/8088 módba, a 0 pedig MCS-80/85 módba kapcsolja az interrupt vezérlőt;
- 1. bit (End Of Interrupt Mode) 1 értéke automatikus interrupt lezárást jelent, a 0 pedig normál módot.
- 2. bit (Master or Slave Mode) 1 értéke (ha a 3. bit áll) Master, a 0 pedig Slave módba lépteti az interrupt vezérlőt.

- 3. bit (Buffered Mode) 1 értéke bufferelt módba lépteti a vezérlőt, míg a 0 nem bufferezett módba. Ekkor a 2. bit érdektelen. Master/Slave mód csak bufferezett módban képzelhető el (lásd alább);
- 4. bit (Special Fully Nested Mode) 1 értéke megengedett alárendelési módot választ ki.

A negyedik előkészítő parancsban foglalt információk külön részletezést igényelnek. Az esetek nagy részében olyan lehetőségekről van szó, amelyek érdekesek ugyan általános műveltségünk fejlesztése szempontjából, de az IBM PC/XT-ben nem játszanak nagy szerepet. Mégis írunk erről egyet-mást azért, mert utálatos dolog olyan eszközzel dolgozni, melynek lehetőségeiről csak halvány árnyalatokban tudunk valamit.

A bufferelt vagy nem bufferelt mód közötti különbségnek csak nagy rendszerekben van jelentősége, amelyekben több, lépcsőzetesen összekötött 8259-et használnak, és melyekben a busz használata bufferelt. Ha a bufferelt módot választunk a 8259 felprogramozására, akkor a Master szerepét játszó 8259 a megfelelő vonalon át egy jelet küld a buffer bekapcsolására. Az IBM PC-n ennek nincs jelentősége, de az AT-n igen.

IX.4.9. Az interrupt vezérlő üzemszerű használata

Most ismerkedjünk meg az üzem közben kiadandó vezérlési parancsokkal. Három ilyen parancs van: az interrupt maszk, amelynek segítségével szintenként tilthatjuk/engedélyezhetjük az interruptok fogadását; az interrupt végét jelző parancs, amely az interrupt-prioritások átrendezését is vezérelheti, és végül a maszk-mód vezérlésére szolgáló harmadik. A parancsok közül az elsőt az interrupt vezérlő második portjára kell kiadni (az IBM PC/XT-n 21H), a másik kettőt az alapporton (20H).

1. vezérlő parancs, OCW1 (kiküldendő a 21H portra):

7	6	5	4	3	2	1	0
M7	M6	M5	M4	M3	M2	M1	M0

A maszkbitek 1 értéke letiltja, míg a 0 engedélyezi a megfelelő interrupt fogadását. Ez a port olvasható, és ha előkészítés nélkül beolvassuk, akkor (alább részletezett több jelentése kö-

zül) magát a maszkszót kapjuk meg. Egy maszkparancs meggondolatlan kiadása működésképtelenné teheti az operációs rendszert.

2. vezérlő parancs, OCW2 (kiküldendő a 20H portra):

7	6	5	4	3	2	1	0
Rotate on EOI Mode	Spec. EOI Proc.	End Of Int.	0	0	aktivizálandó interrupt-szint megadása		

- 0-2. bit (Interrupt Request Level) szabja meg specifikus EOI parancs esetén, hogy a továbbiakban melyik interrupt-szint kapja a legalacsonyabb prioritást.
- 5. bit (End of Interrupt): ez a legfontosabb bit ebben a parancsban: egyes értéke jelzi a vezérlőnek, hogy befejeződött az interrupt-eljárás. Ebből adódik egyébként a 20H portra kiküldendő 20H parancs; az IBM PC-n nem használják ki azt a lehetőséget, hogy a prioritási rendet módosítani lehet.
- 6. bit (Specific EOI Process) beállítása esetén kap jelentőséget az alsó három bit. Ezek adják meg azt a szintet, amely a továbbiakban (a következő OWC2 kiadásáig) a legalacsonyabb prioritású lesz. Ha pl. a 4. szint lesz a legalacsonyabb prioritású, akkor az 5. lesz a legmagasabb, a 6. eggyel mögötte és így tovább.
- 7. bit (Rotate on EOI Mode) az automatikus rotálást kapcsolja be. Ha ezt a bitet beállítjuk, akkor a továbbiakban (a következő OCW2 kiadásáig) az a szint kapja a legalacsonyabb prioritást, amelyet éppen kiszolgáltunk. Ez akkor hasznos, ha azonos rangú eszközöket szolgálunk ki. Ha pl. a 2. szint kapja a legalacsonyabb prioritást, akkor a 3. lesz a legmagasabb, a 4. eggyel mögötte és így tovább.

3. vezérlő parancs, OCW3 (kiküldendő a 21H portra):

7	6	5	4	3	2	1	0
x	Spec. Mask M Enable	Spec. Mask Mode	0	1	Poll	Read Sec. Reg.	Read State Reg.

A harmadik vezérlő parancs a beolvasáskor több célt szolgáló második port (az IBM PC-n a 21H port) funkciói közül választ. Az alsó két bit mutatja meg, hogy a következő olvasással melyik, a második port mögé rejtett regisztert akarjuk kiolvasni.

Ha minden előkészítés nélkül olvassuk, akkor az interrupt maszk regisztert kapjuk meg. Az interrupt vezérlő még két számunkra fontos belső regiszterrel rendelkezik: az Interrupt Request regiszterrel, amelyben azt tartja nyilván, hogy mely vonal(ak)on van aktív interrupt-kérés, és az Interrupt State regiszterrel, amelyben az éppen teljesítés alatt álló interrupt-kérések vannak. Ha e regiszterek valamelyik bitje 1, az jelzi, hogy aktív kérés van érvényben, illetve a kérés hatására aktív interrupt-eljárás folyik.

- Read Secondary Status (1. bit) 1 értéke mutatja, hogy a második port soron következő olvasása nem az interrupt maszk regiszterre, hanem egy másodlagos státuszregiszterre vonatkozik. Ekkor a
- Read Status Register bit Ø értéke esetén az Interrupt Request regisztert, 1 értéke esetén pedig az Interrupt State regisztert akarjuk beolvasni. A
- Poll bit 1 értéke pollozási üzemmódba lépést jelent, a Ø értéke pedig az interrupt-módba lépést;

Az 5-6. bitek együtt vezérlik a speciális interrupt maszkolást:

- Enable Special Mask Mode Control bit Ø értéke esetén a Special Mask Mode Control bit értektelen, míg 1 értéke esetén vezérlő jelentése érvényes;
- Special Mask Mode Control Ø értéke letiltja, az egyes érték pedig bekapcsolja a különleges maszkolást. Ha kikapcsoljuk, akkor igaz az az alaptétel, hogy az éppen kiszolgálás alatt álló interrupt-kérést csak nála magasabb prioritású kérés szakíthatja meg. Ha viszont bekapcsoljuk, akkor a kiszolgálás alatt álló interrupt-kérést bármelyik, attól eltérő szinten érkező újabb interrupt-kérés megszakíthatja. Az IBM PC nem él ezzel a lehetőséggel.

IX.5. Az NMI szerepe és letiltása

Az NMI (Non Maskable Interrupt, le nem tiltható megszakítás) a gép normál működése közben a memória-paritáshibák kijelzésére szolgál. A bekapcsolási processzus idején az NMI le van tiltva. Bizony, erre is van lehetőség. Az NMI vezérlésére az AØH port szolgál. Ha erre a portra 8ØH-t küldünk ki, akkor az NMI él, bármikor bejöhet. Ha azonban a ØØH adatot küldjük ki erre a portra, akkor letiltottuk az NMI-t. A processzor persze a maga részéről fogadná (hiszen az NMI csak az ő számára "Non Maskable"); a külső hardware azonban nem élezi ezt az interruptot.

Mielőtt ettől a fejezettől búcsút vennénk, említsük meg még egyszer azt, hogy az Intel 8087 koprocesszor az NMI-t használja a kivételek, a működése során bekövetkezett hibák jelzésére.

IX.6. Játékadapter (botkormány)

Az IBM PC/XT elsősorban nem játékgépnek készült, de rengeteg (főként eredetileg Commodore 64-re írt) játékprogram van forgalomban. Ezek közül számos működhet botkormánnyal. A géphez a gyártó nem ad botkormányt, de kifejlesztett olyan adaptert, amelyhez (egyébként egyszerre két) ilyen csatlakoztatható.

Mielőtt megismerkednénk az adapterrel, szóljunk pár szót magáról a botkormányról! Ez, mint neve mutatja, egy marokba jól simuló "bot", amely egy pici lapos asztalkából emelkedik ki. Egy vagy két gomb szokott lenni rajta (a mutató- és a hüvelykujj alatt). A botkormányt előre-hátra, jobbra-balra mozgathatjuk, és a gombokat is nyomkodhatjuk. A gépen futó program az adapter közvetítésével észleli ezeket a mozgásokat. A botkormány voltaképpen két tolóellenállás, melyek közül az egyik az előre-hátra mozgatás hatására változik, a másik pedig, ha jobbra-balra mozgatjuk. A mozgásirányt az adapter a két ellenállási érték változásából ismeri fel, a gombok közönséges kapcsolók.

A játékadapter külön kártya; a külső oldalán levő tizenötpólusú D aljzathoz kapcsolhatjuk a botkormányokat.

Egyetlen porton át csatlakozik a processzorhoz, e port címe 201H. Ennek beolvasásával tudunk információt szerezni a botkormány helyzetéről. A beolvasott nyolcbites adat két részre van osztva; az alsó négy bit az ellenállásra, a felső négy pedig a gombok állására vonatkozik. A felső négy bit értelmezése világos; az ellenállásokat (tehát azt, hogy mennyire nyomják valamilyen irányban) időméréssel tudhatjuk meg.

7	6	5	4	3	2	1	0
2.bot-korm. 2.gomb	2.bot-korm. 1.gomb	1.bot-korm. 2.gomb	1.bot-korm. 1.gomb	2.bot-korm. Y koor	2.bot-korm. X koor	1.bot-korm. Y koor	1.bot-korm. X koor

A játékadapter státuszportjának bitkiosztása

- 0. és 1. bit: az 1. botkormány X, illetve Y koordinátája (bővebben lásd alább);
- 2. és 3. bit: a 2. botkormány X és Y koordinátája. Ezekből a bitekből a megfelelő botkormány állására következtethetünk: a

botkormány egy adott irányú ellenállása Ø és 1ØØ kiloohm között lineárisan változhat (a középső állásban értelemszerűen kb. 5Ø). Ha meg akarjuk tudni, hogy milyen az ellenállás a botkormányon, akkor a 2Ø1H portra ki kell írni egy akármilyen adatot. Ezután folyamatosan olvassuk a 2Ø1H portot addig, míg az ellenállásra jellemző bitek értéke 1; ez az időtartam arányos az ellenállással. A képlet a következő:

Időtartam [µsec] = 24,2 + Ø,Ø11 * R

ahol "R" az ellenállás ohmokban mért értéke. Az idők persze elég kicsik; voltaképpen abból következtethetünk ilyen rövid időtartamokra, hogy egy pár utasításos ciklus hányszor képes lefutni. Ne felejtsük el azonban, hogy igen nagy, több tízezer ohmos ellenállásokat kell így lemérnünk, tehát tapasztalati úton egyszerűen lehet meghatározni a szélsőértékeket.

> Megjegyzés: célszerű egyébként a botkormányadapter kezelésére olyan drivert írnunk, amely lehetővé teszi a botkormány kalibrálását: azt, hogy futás közben mérjük be a paramétereket, és ezeket valahogyan rögzítsük.

Az alapidő elég lehet a ciklus előkészítéséhez (kb. 1Ø-15 utasítás fut le ezalatt), maga a ciklus pedig már gyors. Egy-két kiloohm tévedés egyébként valószínűleg nem számít sokat.

- 4. és 5. bit: az 1. botkormány két gombjának állása. A Ø érték jelzi, hogy a gombok be vannak nyomva, az 1 pedig azt, hogy nincsenek.
- 6. és 7. bit: a 2. botkormány két gombjának állása. A Ø érték jelzi, hogy a gombok be vannak nyomva, az 1 pedig azt, hogy nincsenek.

IX.7. Az egér (mouse)

Számos korszerű program teszi lehetővé az egér használatát. Ez igen kényelmessé teszi a munkát, és sok gépeléstől kímél meg bennünket.

Az IBM PC/XT-hez többféle egér terjedt el. Olyan is akad közöttük, amely a soros vonalon át kezelhető egy megfelelő driverrel (e driverek neve általában MOUSE.SYS, és az ANSI.SYS-hez hasonlóan a CONFIG.SYS file-ban aktivizálhatók rendszertöltéskor). A sokféle közül mi nem egy ilyen változattal ismerkedünk

meg, hanem a leginkább szabványosnak tekinthető MICROSOFT egérrel. Ez külön kártyával kapcsolódik a géphez; software úton a 33H interrupt aktivizálásával érhetjük el.

Az egér egy pici (általában szürke, gömbölyű) műanyag tárgy, amely remekül simul a kezünkbe; tetején egy vagy két gomb van (a Microsoft egér "hátán" kettő); ha pedig felvesszük az asztalról, akkor sima hasába ágyazva egy kis gumigolyó látszik. Természetesen, mint minden jóravaló egérnek, hosszú farka is van, amely persze a kezelését végző kártyához csatlakozik. Ha az egeret letesszük egy sima asztallapra, és fel-le, jobbra-balra mozgatjuk, akkor gumigolyócskája révén érzékeli a mozgást, és annak jellemzőit továbbítja a kezelő kártyának. Ez egy "cursort" (amely lehet egy grafikus ábra, vagy a környezettől eltérő attributum) mozgat a képernyőn, pontosan követve az egér mozgását. Ha pedig olyan programmal dolgozunk, amely ismeri az egeret, és sok menüből való választás szerepel benne, akkor a cursort a kiválasztandó menüelemre visszük, majd megnyomjuk a(z egyik) gombot; a program úgy tekinti, hogy választottunk a menüről. Hasonlóképpen használhatjuk az egeret grafikus alkalmazásokban stb. Kis gyakorlattal tehát igen kényelmes lehet a használata; a botkormányt is helyettesítheti. Használatára két példa: az egér menüből való választásra szolgál a Microsoft Windows vagy a Microsoft CodeView programban, vagy grafikus segédeszközként a PcPaint nevű grafikus programban).

> Megjegyzés: létezik többféle optikai egér is, amelyeket nem egyszerű lapon, hanem egy megfelelő négyzethálóval ellátott felületen kell mozgatni; a fejlettebb változatok alkalmasak lehetnek képek raszterizálására is.

Az egér mellett természetesen változatlanul használható a klaviatúra is. Mindig választhatunk, melyik beviteli módot használjuk, ha egy funkciót billentyűzetről vagy egérrel is el lehet érni.

Az egér számára a képernyő egy virtuális képernyő, mely mindig elemi pontokból áll (eredeti nevén pixel). A virtuális ernyőn mozgatja a cursort. Az asztallapon való mozgás egysége a mickey (nem fordítom le, mert olyan aranyos: Mickey Mouse...). Ez a hüvelyknek (inch) kétszázad része. Ha egy mickey-vel mozdítjuk az egeret, akkor a cursor is egy pontnyit mozdul az ernyőn.

A Microsoft egér háromféle cursort tud kezelni: grafikust, software és hardware szöveges cursort. A grafikus cursor általában egy 16 pont magas, 8 pont széles ábra, melyet tetszésünk szerint adhatunk meg. A szöveges cursorok közül a software cur-

sor azt jelenti, hogy az éppen megcélzott karakter más attribútumú lesz a képernyőn, a hardware cursor pedig egy villogó téglalap az éppen címzett pozíción. A szöveges cursorok csak alfanumerikus képernyőmódokban használatosak, és mindig egy egész karakternyit mozognak. A grafikus cursor grafikus üzemmódokban pontról pontra vándorolhat; alfanumerikus módokban mindig egy karakterpozíció bal felső sarkára mutat; így koordinátái mindig oszthatók nyolccal, és egy valódi mozdulásához nyolc mickey-vel kell elmozdítani az asztalon az egeret.

IX.7.1. Az egér és a képernyő

A következő táblázat a különféle képernyőmódokat mutatja be az egér szemszögéből. A Microsoft egér természetesen ismeri a monokróm és a színes grafikus adaptert; az EGA és a HERCULES adaptereket, valamint a Magyarországon kevésbé elterjedt 3270 adaptert (IBM All Points Addressable Graphics Adapter).

Mód	Használható adapterek	Virtuális képernyő mérete	Cella-méret	Bitszám pontonként
0	CGA, EGA, 3270	640x200 pont	16x8	-
1	CGA, EGA, 3270	640x200 pont	16x8	-
2	CGA, EGA, 3270	640x200 pont	8x8	-
3	CGA, EGA, 3270	640x200 pont	8x8	-
4	CGA, EGA, 3270	640x200 pont	2x1	2
5	CGA, EGA, 3270	640x200 pont	2x1	2
6	CGA, EGA, 3270	640x200 pont	1x1	1
7	Mono, EGA, 3270	640x200 pont	8x8	-
E	EGA	640x200 pont	1x1	1
F	EGA	640x350 pont	1x1	1
10	EGA (128 Kb!)	640x350 pont	1x1	1
30	3270	720x350 pont	1x1	1
-	Hercules	720x348 pont	1x1	1

Látható, hogy az üzemmódok megfelelnek a kártya üzemmódjainak, és a cellaméret (tehát a minimális lépési távolság) is konzisztens. Alfanumerikus módokban (0-3 és 7) a képernyőn persze csak karakterek vannak. Ezek közül a 0 és 1 módban egy karakter szélessége 16 elemi pont, a cursor vízszintesen legalább ennyit mozdul, míg függőleges irányban minden grafikus módban 8 pontnyit. A grafikus módokban közepes felbontás esetén (4 és 5)

egy grafikus pontnak két képernyő-alappont felel meg, ezért a cursor legalább kettővel mozdul vízszintes irányban. A többi grafikus módban egy mickey elmozdulás valóban egy pontnyi cursormozgást idéz elő.

IX.7.2. Az egér cursora

A cursortípusok közül szabadon választhatunk, de az egér egyszerre csak egy cursort kezel. Vannak programok, amelyek képesek két független cursor kezelésére (pl. Microsoft Word). A grafikus cursor alakját előre definiált minták közül választhatjuk, vagy magunk szabhatjuk meg. Hasonlóképpen meghatározhatjuk a szöveges cursorok attributumát, illetve méretét. Munka közben válthatunk cursortípust is.

A grafikus cursor mérete függ az üzemmódtól (16x16 vagy 8x16 elemi pont lehet, utóbbi értelemszerűen a közepes felbontású 4 és 5 módban), tehát 256, illetve 128 elemi pontot foglal le. Minden ilyen elemi pontot két dolog határoz meg: a háttérmaszk és a cursormaszk. A cursor megjelenítése során a képernyő egy pontját "és"-elik a háttérmaszkkal, és "kizáró vagy"-olják a cursormaszkkal. Ha tehát a háttérmaszk 0, akkor a cursor adott pontja "eltakarja" a mögötte levő pontot. Ha a háttérmaszk 1, akkor a cursor adott pontja változatlanul hagyja az eredetit. A cursormaszk határozza meg a cursor alakját a téglalapon belül. Ha a cursormaszk egy ponton 0, akkor ott a háttér vagy eltűnik, vagy változatlan marad (a háttérmaszktól függően). Ha a cursormaszk 1, akkor az adott ponton látszani fog a cursor, vagy megfordítja a mögötte levő ernyőtartalmat. A közepes felbontású grafikus módokban persze nem egy, hanem két bit alkot egy-egy maszkot, tehát a cursor igen sokféleképpen jelenhet meg.

Még egy fontos kérdés van a grafikus cursorral kapcsolatban: ez a tekintélyes méretű téglalap valójában melyik pontra mutat? Azt, hogy melyik pont legyen a cursor "csúcspontja", magunk adhatjuk meg, tetszésünk szerint.

Szöveges software cursor használata során is meghatározhatjuk a háttér és a cursor attributumát. Ezek az értékek ugyanúgy játszanak szerepet a cursor által címzett pozíció attributumának meghatározásában, mint a grafikus cursor háttér- és cursormaszkja. Két szóban (16-16 biten) adhatjuk meg a software cursor maszkjait. A háttérmaszk azt szabja meg, hogy az attributum, illetve a karakter bitjeiből mit hagyjon élni a cursor (ez egy "és" művelet); a cursormaszk pedig azt, hogy ezek közül melyiket kell megfordítani (ez egy "kizáró vagy" művelet). Szokásos például úgy definiálni a software cursort, hogy invertálja az attributumot. Ekkor a háttér- és cursormaszk a következő:

```
Háttérmaszk:     0 111 0 111 1111111      (77FFH),
Cursormaszk:     0 111 0 111 0000000      (7700H)
```

vagyis a háttérmaszk a nagy intenzitás és a villogtatás kivételével mindent meghagy a karakter attributumából, míg a cursormaszk invertálja a háttér- és előtérszínt (így a kékből barna, a zöldből lila, és a türkizből piros lesz), viszont nem nyúl a karakter kódjához. Ha a cursor elhagyja az előbb címzett pozíciót, az természetesen visszakapja eredeti tartalmát és attributumát, bármi történt is vele.

A grafikus és a software szöveges cursor egyébként független a gép saját hardware cursorától. Van lehetőség arra is, hogy "egybeejtsük" ezeket; ez történik, ha az egér hardware szöveges cursorát használjuk. Ennek alakját pontosan azokkal a paraméterekkel lehet meghatározni, mint a ROM BIOS kezelte hardware-cursorét.

Természetesen arra is van lehetőség, hogy a cursort (bár mindig létezik) láthatatlanná tegyük. Ennek persze gyakorlati jelentősége nincsen; annál hasznosabb a feltételesen látható cursor: az egeret kezelő program megjeleníti a cursort, ha az egy bizonyos (téglalap alakú) tartományban van, és láthatatlanná teszi, ha onnan kiléptetjük.

IX.7.3. A Microsoft egér funkciói

```
Interrupt-sorszám:                33H
Javasolt interrupt-név:           BS_MOUSE
```

Az egeret kezelő funkciók hívására javasolt makró:

```
MSECLL          MACRO   CODE, PAR2, PAR3, PAR4
        IFNB    <PAR2>                          ;;
                MOV     BX, PAR2                ;;
        ENDIF                                   ;;
        IFNB    <PAR3>                          ;;
                MOV     CX, PAR3                ;;
        ENDIF                                   ;;
        IFNB    <PAR4>                          ;;
                MOV     DX, PAR4                ;;
        ENDIF                                   ;;
                MOV     AX, CODE                ;;!! Nem AH !!
                INT     BS_MOUSE                ;;
        ENDM                                    ;
```

Az egér funkcióhívásai tehát erősen elütnek a megszokott ROM BIOS hívásoktól, mivel igen nagy súlyt fektettek arra, hogy magasszintű nyelvekből közvetlenül és kényelmesen hívhatóak legyenek. A Pascal, Fortran és Basic programokból (a MOUSE függvény előzetes deklarálása után) egyszerűen meghívhatunk egy funkciót, négy egész paramétert adva át neki. Ezzel pedig nem fér össze az, hogy félregiszterekbe kelljen adatokat tölteni. A C programozási nyelvből való hívás egy kicsit problematikusabb, mivel a MOUSE függvény egyfelől nem érték, hanem cím szerint veszi át paramétereit, másfelől pedig visszatérési stratégiája Pascal-szerű (ezekről bővebben majd a negyedik kötetben). C-ből tehát így hívható egy MOUSE funkció:

```
extern  pascal  far     mouse( int *, int *, int *, int * );

int     p1, p2, p3, p4;

        p1 = funkciókód;
        p2 = par2; p3 = par3; p4 = par4;
        mouse( &p1, &p2, &p3, &p4 );
```

Egy Pascal deklarálási és hívási példa:

```
PROCEDURE       MOUSE                   {rövid paraméter-pointerekkel}
(VARS   M1, M2, M3, M4: INTEGER);
EXTRN;

        M1 := funkciókód
        M2 := par2
        MOUSE( M1, M2, M3, M4 )
```

Ha magasszintű nyelven akarjuk kezelni az egeret, akkor lényeges dolog: az első paraméter az AX-nek, a második a BX-nek, a harmadik a CX-nek, és a negyedik a DX-nek felel meg mind híváskor, mind visszatéréskor. Tehát bármely nyelvből hívhatjuk az egér funkcióit e megfeleltetés betartásával.

Ne felejtsük el, hogy a magasszintű nyelvek többsége stacken adja át a paramétereket, ezért (hiába nincs szükségünk például a második paraméterre) mindig, minden paramétert pontosan a megadott sorrendben kell átadni a függvénynek, különben eltéved a stacken, sőt végzetesen össze is keverheti azt.

IX.7.4. Státuszvizsgálat és előkészítés

```
Funkciókód:                          00H                    INT  33H
Javasolt funkciónév:                 MSE_INI
Be:
        AX      00H
Válasz:
        AX       0 ha az egér nem installált,
                -1 ha igen
        BX      az egér gombjainak száma (mindig kettő)
```

Ez a funkció alapállapotba hozza az egeret kezelő programokat. A cursort a képernyő közepére állítja, a grafikus cursor egy nyíl lesz, a szöveges software cursor invertált karakter stb.

IX.7.5. A cursor láthatóvá tétele

```
Funkciókód:                          01H                    INT  33H
Javasolt funkciónév:                 MSE_SHWCURS
Be:
        AX      01H
Válasz:
        nincs
```

IX.7.6. A cursor láthatatlanná tétele

```
Funkciókód:                          02H                    INT  33H
Javasolt funkciónév:                 MSE_HDCURS
Be:
        AX      02H
Válasz:
        nincs
```

IX.7.7. A cursor helyének és a gombok állásának lekérdezése

```
Funkciókód:                          03H                    INT  33H
Javasolt funkciónév:                 MSE_GTCPOSBT
Be:
        AX      03H
Válasz:
        BX      az egér gombjainak állása:
                0. bit értéke 1, ha a bal oldali, illetve
                1. bit értéke 1, ha a jobb gomb le van nyomva;
        CX      a cursor vízszintes pozíciója
        DX      a cursor függőleges pozíciója
```

IX.7.8. A cursor pozíciójának beállítása

Funkciókód: 04H INT 33H
Javasolt funkciónév: MSE_STCPOS
Be:
AX 04H
CX, DX a kívánt vízszintes, ill. függőleges koordináta
Válasz:
nincs
A koordinátáknak a pillanatnyi üzemmód szerinti virtuális képernyőn belül kell lenniük. Ha az ernyő nem nagyfelbontású, az egeret kezelő program a koordinátákat megfelelő értékekre kerekíti (pl. szöveges módban 8-al vagy 16-al oszthatóra).

IX.7.9. Egy gomb lenyomásainak lekérdezése

Funkciókód: 05H INT 33H
Javasolt funkciónév: MSE_GTBTPRS
Be:
AX 05H
BX 0 a bal, és 1 a jobb gomb lekérdezéséhez
Válasz:
AX a gomb állása:
0, ha nincs lenyomva, és 1, ha igen;
BX a gomb lenyomásainak száma az utolsó lekérdezés vagy az előkészítés óta;
CX, DX a cursor koordinátái a gomb utolsó lenyomásakor (vízszintes, illetve függőleges koordináták)

IX.7.10. Egy gomb felengedéseinek lekérdezése

Funkciókód: 06H INT 33H
Javasolt funkciónév: MSE_GTBTREL
Be:
AX 06H
BX 0 a bal, és
1 a jobb gomb lekérdezéséhez
Válasz:
AX a gomb állása:
0, ha nincs lenyomva, és 1, ha igen;
BX a gomb felengedéseinek száma az utolsó lekérdezés vagy az előkészítés óta;
CX, DX a cursor koordinátái a gomb utolsó felengedésekor (vízszintes, illetve függőleges koordináták)

IX.7.11. Vízszintes tartomány megadása

Funkciókód: 07H INT 33H
Javasolt funkciónév: MSE_STHRNG
Be:
AX 07H
CX, DX legkisebb, ill legnagyobb vízszintes koordináta
Válasz:
nincs

A további cursormozgatások az itt megadott koordináták között lehetségesek csak. Ha a cursor a funkció hívásakor a tartományon kívül volt, akkor a program a tartomány belsejébe viszi.

IX.7.12. Függőleges tartomány megadása

Funkciókód: 08H INT 33H
Javasolt funkciónév: MSE_STVRNG
Be:
AX 08H
CX legkisebb és
DX legnagyobb függőleges koordináta
Válasz:
nincs

A további cursormozgatások az itt megadott koordináták között lehetségesek csak. Ha a cursor a funkció hívásakor a tartományon kívül volt, akkor a program a tartomány belsejébe viszi.

IX.7.13. Grafikus cursor megadása

Funkciókód: 09H INT 33H
Javasolt funkciónév: MSE_STGRCURS
Be:
AX 09H
BX a cursor csúcsának függőleges koordinátája a cursor téglalapján belül
CX a cursor csúcsának vízszintes koordinátája a cursor téglalapján belül
DX pointer a háttér- és cursormaszkra
Válasz:
nincs

A háttér- és cursormaszkok két folytonos és szomszédos területen levő szavas tömböt alkotnak, melynek hossza 16 vagy (16x8-as cursor esetén) 8 elem.

Például egy balra felfelé mutató nyíl mint cursor:

```
GR_CUR  LABEL   WORD
                        ;----------------;
                        ; Háttérmaszk csupa egyes, mert ...
                        ;----------------;
  DW    16  DUP   ( 1111111111111111B)  ;Átlátszó cursor!
                                        ;
                        ;----------------;
                        ; Cursormaszk:
                        ;----------------;
  DW    0000000000000000                ;A cursor csúcsának
  DW    0010000000000000                ; koordinátái:
  DW    0011100000000000                ; (2, 1) !
  DW    0011111000000000                ;
  DW    0011111110000000                ;
  DW    0011011000000000                ;
  DW    0000001100000000                ;
  DW    0000000110000000                ;
  DW    0000000011000000                ;
  DW    0000000001100000                ;
  DW    0000000000000000                ;
  DW    0000000000000000                ;
  DW    0000000000000000                ;
  DW    0000000000000000                ;
  DW    0000000000000000                ;
  DW    0000000000000000                ;
```

IX.7.14. Szöveges cursor megadása

Funkciókód: 0AH INT 33H
Javasolt funkciónév: MSE_STTXTCURS

Be:

AX	0AH
BX	cursorválasztás: 0 – software szöveges cursor; 1 – hardware szöveges cursor
CX	háttérmaszk értéke software, illetve a kezdősor hardware szöveges cursor esetén
DX	cursormaszk értéke software, illetve a végsor hardware szöveges cursor esetén

Válasz:

nincs

IX.7.15. Az egér mozgásának lekérdezése

Funkciókód: ØBH INT 33H
Javasolt funkciónév: MSE_GTMOTION
Be:
AX ØBH
Válasz:
CX az utolsó lekérdezés óta bekövetkezett vízszintes mozgás (mickey-ben); a pozitív érték jobbra, a negatív balra mozgást jelent
DX az utolsó lekérdezés óta bekövetkezett függőleges mozgás (mickey-ben); a pozitív érték felfelé, a negatív lefelé végzett mozgást jelent

Az egér jobbra-balra vagy fel-le mozgatása esetén a mozgások előjelesen összegződnek; így tehát előfordulhat, hogy összevissza rángatjuk az egeret az asztalon, és mégis nullának adódik a CX és DX értéke.

IX.7.16. Felhasználói rutin címe és maszkja

Funkciókód: ØCH INT 33H
Javasolt funkciónév: MSE_STUSRRUT
Be:
AX ØCH
CX hívási feltételi maszk (a bitek egyes értéke kéri, nullás értéke pedig tiltja a rutin meghívását az adott esemény bekövetkezésekor):
Ø. bit az egér elmozdult
1. bit bal oldali gombot lenyomták
2. bit bal oldali gombot felengedték
3. bit jobb oldali gombot lenyomták
4. bit jobb oldali gombot felengedték
DX a szubrutin címe (offset)
Válasz:
nincs

A felhasználói szubrutin egy hardware interrupt rutinhoz hasonló, amelyet az egeret kezelő software akkor aktivizál, ha az egérrel történik valami (elmozdul, megnyomják vagy felengedik valamelyik gombját). A CX regiszterben adjuk meg, hogy milyen esetben kérjük a rutin meghívását.
A rutin meghívásakor a következőket találjuk a regiszterekben:
AX feltételi maszk (bitjei, mint paraméteré); az a bit kap egyes értéket, amely a hívást kiváltó oknak felel meg;
BX a gombok állapota
CX, DX a cursor vízszintes, ill. függőleges koordinátája

IX.7.17. Fényceruza-emuláció bekapcsolása

Funkciókód: ØDH INT 33H
Javasolt funkciónév: MSE_LPENON
Be:
AX ØDH
Válasz:
nincs

A fényceruza-emuláció úgy működik, hogy mindkét gomb lenyomása jelzi az "érintést". A fényceruza-emuláció használata esetén egyébként például Basic-ből a PEN függvény segítségével lehet lekérdezni a "fényceruza" pillanatnyi pozícióját.

IX.7.18. Fényceruza-emuláció kikapcsolása

Funkciókód: ØEH INT 33H
Javasolt funkciónév: MSE_LPENOFF
Be:
AX ØEH
Válasz:
nincs

IX.7.19. Az egér és a cursor mozgási aránya

Funkciókód: ØFH INT 33H
Javasolt funkciónév: MSE_MCKPPX
Be:
AX ØFH
CX a vízszintes mozgások aránya (mickey/pixel)
DX a függőleges mozgások aránya (mickey/pixel)

Válasz:
nincs

Az arányok alapértelmezésben: vízszintesen egy mickey egy pixel mozgást ír elő, vagyis egy az egyhez; függőlegesen két mickey jelent egy pixeles mozgást. Ehhez az asztalon egy kb. 8x5 cm-es területre van szükség (3,2 x 2 hüvelyk).
Az aránnyal egyébként azt kell megadnunk, hogy 8 pixel (alap-pont) elmozduláshoz hány mickey-t kell mozognunk az egérrel. Tehát az alapértelmezett értékek megadásához a CX-be nyolcat, a DX-be tizenhatot kell tölteni.

IX.7.20. Szűkebb ablak megadása a cursornak

Funkciókód: 10H INT 33H
Javasolt funkciónév: MSE_WINDOW
Be:

AX	10H
CX	bal felső sarok vízszintes és
DX	függőleges koordinátája
SI	jobb alsó sarok vízszintes és
DI	függőleges koordinátája

Válasz:

nincs

A funkció segítségével a cursor mozgatását a képernyő egy adott tartományára korlátozhatjuk (például egy grafikus program esetén a grafikus mezőre).
A funkció egyébként láthatatlanná teszi a cursort, melyet a 01H funkció meghívásával tehetünk ismét láthatóvá.
Ezt a funkciót kényelmesen csak assemblerből hívhatjuk. Fontos tudnunk, hogy akár C-ből, akár Pascalból is könnyen meghívható az "int86()" függvény segítségével.

IX.7.21. Sebességküszöb beállítása

Funkciókód: 13H INT 33H
Javasolt funkciónév: MSE_SPDTHSLD
Be:

AX	13H
DX	a beállítandó sebességküszöb értéke (egysége: mickey/másodperc)

Válasz:

nincs

A sebességküszöb az egér kényelmes szolgáltatása, amely egymástól nagy távolságra elhelyezkedő objektumok megközelítésében játszik szerepet. Az alapértelmezett küszöbérték 64 mickey/sec. Ha az egér ennél gyorsabban mozog, akkor a vezérlőprogram megkétszerezi a cursor mozgási sebességét.
E funkció egyetlen paramétere a kívánt küszöbérték. Ha ezt magasabbra emeljük, mint a lehetséges érték (például 20000-re), akkor elérhetjük, hogy a cursor sosem fog kétszeres gyorsasággal mozogni.

X. Egyéb ROM BIOS interruptok

A 256 interrupt szint közül 16 tartozik a ROM BIOS és a felhasználó közötti felülethez. Ezek sorszáma a 10H-tól a 1FH-ig (16-tól 31-ig) terjed. Ezek között vannak nagy driverek és "apróságok" is. A driverek hívásait az előző fejezetek ismertették: 10H (Képernyőkezelés), 13H (Lemezkezelés), 14H (Az aszinkron vonal kezelése), 16H (A klaviatúra) és 17H (A nyomtató).

A szokás szerint ideszámított 16 interrupt vektoron kívül természetesen itt kell megemlékeznünk a 05H interruptról, amely a képernyőtartalom nyomtatását végzi. Ha pedig az IBM AT interruptjait vizsgáljuk, akkor igen hasznos szolgáltatásokat nyújt még a 4AH és a 70H interrupt is. Amennyire a terjedelem megengedi, ezekről is ejtünk itt néhány szót.

Könyvünk témája elsősorban az IBM PC/XT ismertetése; ha pedig a második típust komolyan gondoljuk, akkor az "egyéb" interruptok közé tartozna az XT gépek ROM BIOS csomagjának egy külön része, a Winchestert kezelő rutinok. Az XT BIOS a 40H interrupt vektorra helyezi el az eredeti floppy-kezelő rutin címét (mely pontosan megegyezik az IBM PC megfelelő rutinjaival), és a Winchestert és a floppy-t is kezelő rutincsomag belépési címét a 13H vektorba teszi. Ezzel a bővebb szolgáltatást nyújtó csomaggal foglalkoztunk a "Lemezkezelés" fejezetben. A 41H interrupt vektor a Winchester paramétereit tartalmazó táblára mutat. Néhány szót szólunk az NMI-ről is, amely a 2. interrupt vektor által címzett pontra adja a vezérlést.

X.1. NMI - Non Maskable Interrupt

Az NMI az interrupt-rendszer minden elemétől teljesen függetlenül bármelyik pillanatban bekövetkezhet, mivel (mint neve is mutatja) a processzor nem képes ellene védekezni, az interrupt vezérlő pedig nem is tud róla, hiszen nem az ő közreműködésével éleződik. Két funkciója lehet. Alapértelmezésben memória-paritáshiba esetén következik be. Az alapértelmezett rutin megvizsgálja, hogy valóban paritáshiba következett-e be, és ha igen, akkor egy hibaüzenet után a HLT utasítás kiadásával lemészárolja a gépet. Mielőtt azonban kiadná ezt az erélyes utasítást, még a CLI segítségével letiltja a közönséges hardware interruptok fogadását. Ezzel persze véglegessé teszi a gép leállítását; a HALT állapotból köztudottan hardware interrupt, vagy hardware reset billenti ki a processzort. Az NMI másik fontos alkalmazási területe az Intel 8087 koprocesszor kivételeinek jelzése. A koprocesszor használata esetén persze el kell ven-

nünk az NMI vektorát, saját kezűleg meggyőződni arról, hogy a koprocesszor váltotta-e ki az interruptot, vagy mégis paritáshiba. Az előbbi esetben értelemszerűen kezeljük a lebegőpontos hibát, az utóbbi esetben legokosabb a vezérlést az eredeti rutinra adni; ölje meg a processzort az.

X.2. Képernyőnyomtatás (hard copy)

Interrupt-sorszám: 05H
Javasolt interrupt-név: BS_HARDCPY

A magasabbfokú programozásban talán ez az interrupt (sorszáma 05H) használható a legjobban; mégpedig nem is eredeti céljára, hanem "hot key"-ként, forró billentyűként.

> Megjegyzés: saját programjainkban nem is célszerű a képernyőtartalom kinyomtatását engedélyezni, hiszen a felhasználó túl könnyen jut hozzá így a helpekben megadott szövegek tartalmához. Ha valaki ellopja a programunkat, akkor legalább szenvedjen meg a beleírt magyarázó szövegek kinyomtatásáért!
> Emellett szól az is, hogy a hard copy nagyon szerencsétlenül kezeli a nyomtatót; ha nincs bekapcsolva, akkor karakterenként kivárja a megfelelő időt, és hosszú percekre állítja le a program működését.

A klaviatúra hardware-interrupt rutinja megvizsgálja, hogy a beérkezett "karakter" nem egy CTRL-PRTSC kombináció-e (azaz a most leütött billentyű nem a PRTSC-e, és ha igen, akkor a CTRL billentyűt lenyomva tartják-e). Ha igen, akkor egyszerűen kiadja az

```
INT     05
```

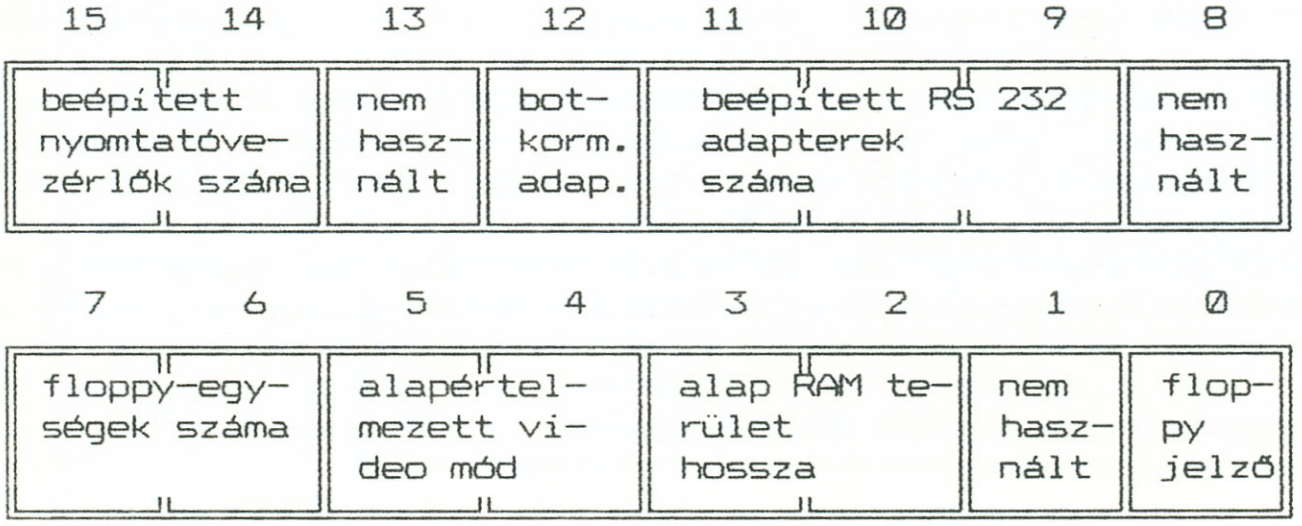
utasítást, azaz meghívja a "hard copy" rutint. Fontos tudnunk, hogy a software-interrupt aktivizálása előtt felszabadítja az interrupt-rendszert, így ez a rutin voltaképpen megszakítható szinten fut.

Ne felejtsük el, hogy a képernyőtartalom kinyomtatását végző eredeti rutin nem foglalkozik komolyan a képernyő üzemmódjával. Ha történetesen grafikus módban vagyunk, akkor szép szorgalmasan kiolvassa a képernyő tartalmát, mintha karakterek lennének rajta, és kinyomtatja az összeset; így aztán igen "mutatós" ábrákhoz juthatunk.

A leghasznosabb alkalmazások azonban, mint említettük, nem az 5. interrupt vektor által felhasznált rutin "ügyes" sajátosságait használják ki. Memóriarezidens programok klaviatúráról való aktivizálására szinte ez az egyetlen fájdalommentes mód; minden egyéb út a klaviatúra-interrupt rutin teljes átírását igényli. Ha valamilyen, rezidensen tárban levő programot akarunk behívni a klaviatúráról, akkor egyszerűen annak belépési pontját töltjük az 5. interrupt vektorba. Ha most leütjük a CTRL-PRTSC kombinációt, akkor a program elindul; a rutint lezáró IRET utasítás pedig a vezérlést az éppen aktív program azon utasítására adja vissza, amelynek végrehajtását a klaviatúra hardware-interruptja megakadályozta.

X.3. A gép elemeinek lekérdezése

Interrupt-sorszám: 11H
Javasolt interrupt-név: BS_EQCHK
Be:
semmi
Válasz:
AX a berendezés elemei. A bitek kiosztását az alábbi ábra szemlélteti:



A 11H interrupt által szolgáltatott leíró szó

- 14-15 bit: nyomtatóvezérlők száma - 0-tól 3-ig terjedhet;
- 12 bit: joystick (botkormány) adapter - 0: nincs, 1: van;
- 9-11 bit: RS 232 adapterek száma - 0-tól 7-ig terjedhet;
- 6-7 bit: floppy-egységek száma - 1 - a 0 egy adaptert, az 1 két adaptert jelent és így tovább; érvényes, ha a 0. bit értéke 1, azaz azt jelzi, hogy egyáltalán van floppy a gépben;

- 4-5 bit: alapértelmezett video-mód - bekapcsoláskor ez lép érvénybe; értelmezése:
  00 - nem használt 10 - 80x25, színes adapter
  01 - 40x25, színes adapter 11 - 80x25, monokróm adapter;
- 2-3 a RAM terület az alapkártyán - értelmezése:
  00 - 16 kilobyte, 10 - 48 kilobyte,
  01 - 32 kilobyte, 11 - 64 kilobyte;
  Az újabb gépeken mind a 640 Kb memória az alapkártyán kaphat helyet, így ez az információ érdektelen lehet.
- 1 bit: floppy-jelző bit - 1 jelzi, hogy van floppy a gépben.

X.4. Memóriahossz lekérdezése

Interrupt-sorszám: 12H
Javasolt interrupt-név: BS_MEMSIZ
Be:
semmi
Válasz:
AX az elérhető RAM hossza 1 Kb-okban számítva.

X.5. Kazettás magnó kezelése (csak IBM PC)

Mint több szakkönyvből ismeretes, az IBM PC gyermekkorának egyik bájos tévedése volt az a (csak tervezett) konfiguráció, amelyben a PC-kompatibilis gép háttértárolója egy kazettás magnó lett volna. Arról nincs tudomásom, hogy valaha használt-e bárki olyan gépet, amelyen ne lett volna legalább egy floppy. Az ilyen konfiguráció szinte használhatatlannak tűnik; nem érdektelen azonban, hogy van lehetőség egy kazettás magnó közvetlen csatlakoztatására. Az interface leírását elhagyjuk, mivel a gyakorlatban senki sem használ kazettás magnót a gépéhez.

Interrupt-sorszám: 15H
Javasolt interrupt-név: BS_DEVDRV

X.5.1. A magnó motorjának bekapcsolása

Funkciókód: AH = 00 INT 15H
Javasolt funkciónév: DEV_MOTORON
Be:
semmi egyéb
Válasz:
nincs

Ez a funkció felemeli a magnócsatlakozó megfelelő kivezetésén a feszültséget, de semmiféle visszajelzést nem ad arra nézve, hogy mikorra válik üzemképessé a magnó (azaz nekünk magunknak kell a megfelelő, tapasztalati úton meghatározott késleltetést beépíteni a programba).

X.5.2. A magnó motorjának kikapcsolása

Funkciókód: AH = 01H INT 15H
Javasolt funkciónév: DEV_MOTOROFF
Be:
semmi egyéb
Válasz:
nincs

Ez a funkció a magnócsatlakozó megfelelő lábán leveszi a feszültséget, arra utasítva ezzel a magnót, hogy állítsa le a motort. Semmiféle visszajelzést nem ad arra nézve, hogy mikorra áll le teljesen a magnó motorja.

X.5.3. Adatblokkok beolvasása

Funkciókód: AH = 02H INT 15H
Javasolt funkciónév: DEV_READBLK
Be:
CX a beolvasandó byte-ok száma
ES:BX a célterület kezdőcíme a memóriában
Válasz:
Carry jelzi, hogy a művelet sikeres volt-e;
0 - sikeres művelet;
1 - hibajelzés. Ekkor AH-ban egy hibakód van:

01H	Cyclical Redundancy Check hiba
02H	Elveszett a bitszinkron
04H	Nem található adat a szalagon

Sikeres művelet esetén:
DX a beolvasott byte-ok száma
ES:BX az utoljára belovasott karakter mögé mutat

Ez a funkció egy vagy több blokkot olvas be az ES:BX regiszterek által meghatározott területre. A byte-ok számát 64 Kb-ig tetszőlegesen adhatjuk meg függetlenül attól, hogy a szám többszöröse-e a kettőszázötvenhatnak (a szokásos rekordhossznak), vagy sem.

X.5.4. Adatblokkok kiírása

```
Funkciókód:                AH = 03H                    INT     15H
Javasolt funkciónév:       DEV_WRTBLK
Be:
        CX      a kiírandó byte-ok száma
        ES:BX   a kiírandó terület kezdőcíme a memóriában

Válasz:
        Carry   jelzi, hogy a művelet sikeres volt-e;
                0 - sikeres művelet;
                1 - hibajelzés; AH a 80H hibakódot  tartalmazza
                    (érvénytelen művelet).
        Sikeres művelet esetén:
        DX      a kiírt byte-ok száma
        ES:BX   az utolsóként kivitt karakter mögé mutat
```

Ez a funkció egy vagy több blokkot ír ki az ES:BX regiszterek által meghatározott területről. A byte-ok számát 64 Kb-ig tetszőlegesen adhatjuk meg attól függetlenül, hogy a szám többszöröse-e a kettőszázötvenhatnak (a szokásos rekordhossznak), vagy sem. Ha nem egész számú rekordot akarunk kiírni, a funkciót megvalósító rutin nulla karakterekkel tölti fel az utolsó megkezdett, de nem teljesen végigírt rekordot.

A magnót kezelő rutinok, mint látjuk, nem támogatnak bennünket szinte semmivel; a teljes file-kezelést nekünk kell megírnunk. Ez azonban nem tűnik nagy hiányosságnak, hacsak valaki nem akar esetleg más gépen (jelesül Commodore-64-en) megvalósított, Basic nyelvű programokat átvinni az IBM PC-re. Ez a művelet azonban akkor sem célszerű, ha netán sikerül: a PC ugyanis a Commodore grafikai lehetőségeit még csak részben sem képes megvalósítani, tehát minden "fontos" programot (azaz a játékprogramokat) szinte elölről meg kellene írni. Ha pedig erre szánjuk magunkat, akkor használjunk egy rendes programozási nyelvet, mert az minden szempontból messzebbre vezet.

X.6. A 15. interrupt az IBM AT-n

Az IBM AT gépek ROM BIOS-rendszerében sokkal alaposabban átgondolták ezt az interruptot. Az AT ROM BIOS számára ez nem egy szükséges rossz, hanem egy általános "device driver" belépési pontja. Természetesen felülről kompatibilis az eredeti ROM BIOS megfelelő funkcióival; itt most az eltérésekről adunk egy rövid

listát. A "DEV_"-el kezdődő nevű funkciók a multi-tasking rendszerek megvalósítását támogatják: elősegítik a rendszer adminisztrációs tevékenységét, lehetővé téve annak nyilvántartását, hogy melyik eszközt melyik procesz használja.

Az egyéb karaktersorozattal kezdődő nevű funkciók az IBM AT néhány speciális hardware-lehetőségét teszik hozzáférhetővé (például a memóriakiterjesztést).

Csak IBM AT !

X.6.1. Perifériális eszköz megnyitása

Funkciókód: AH = 80H INT 15H
Javasolt funkciónév: DEV_OPEN
Be:

BX - device azonosító
CX - eljárás (procesz) azonosító

Az adott procesz számára megnyitja (elérhetővé teszi) a kívánt eszközt. Az MS-DOS és a ROM BIOS jelenlegi megvalósításában nincs multi-tasking; e funkció egy IRET utasítást tartalmaz.

X.6.2. Perifériális eszköz lezárása

Funkciókód: AH = 81H INT 15H
Javasolt funkciónév: DEV_CLOSE
Be:

BX - device azonosító
CX - eljárás (procesz) azonosító

Az adott procesz számára lezárja (elérhetetlenné teszi) az eszközt. Az MS-DOS és a ROM BIOS jelenlegi megvalósításában nincs multi-tasking; e funkció egy IRET utasítást tartalmaz.

X.6.3. Eszköz használatának befejezése

Funkciókód: AH = 82H INT 15H
Javasolt funkciónév: DEV_TRMPROC
Be:

BX - device azonosító

E funkció az adott eszköz elérését lehetetlenné teszi, abortálván az eszköz driverét. Az MS-DOS és a ROM BIOS jelenlegi megvalósításában nincsen multi-tasking; e funkció egy üres IRET utasítást tartalmaz.

X.6.4. Eseményre várakozás

Funkciókód: AH = 83H INT 15H
Javasolt funkciónév: DEV_WTEVENT
Be:
AL - alfunkciókód
ES:BX - pointer a hívó memóriaterületére
CX:DX - várakozási idő milliomod másodpercben

E funkció arra szolgál, hogy milliomod másodpercekben megadott ideig várakozhassunk arra, bekövetkezik-e az adott eszközön valamilyen esemény (input vagy output művelet befejeződése). Bővebben lásd alább, az EXT_WAIT funkciót. Az MS-DOS és a ROM BIOS jelenlegi megvalósításában nincs multi-tasking; e funkció a valósidőt mérő óra ellenőrzése után visszatér.

X.6.5. A joystick (botkormány) kezelése

Funkciókód: AH = 84H INT 15H
Javasolt funkciónév: EXT_JSTK_SUPP
Ez a funkció két alfunkcióval rendelkezik:
a) a kapcsolók állapotának lekérdezése
Be:
DX - 0H
Válasz:
AL - 7-4. bitek tartalmazzák a kapcsolók állapotát
b) a joystick különböző ellenállási értékeinek lekérdezése
Be:
DX - 1H
Válasz:
AX - A(x) ellenállási érték
BX - A(y) ellenállási érték
CX - B(x) ellenállási érték
DX - B(y) ellenállási érték

A botkormányról bővebben lásd az "Egyéb tudnivalók a hardware-ről" fejezet megfelelő pontját.

X.6.6. System Request kezelése

Funkciókód: AH = 85H INT 15H
Javasolt funkciónév: EXT_SYSREQ
Be:
AL - 0 lenyomás
AL - 1 felengedés

A 15H interrupt e funkcióját aktivizálja a klaviatúra 09H sorszámú hardware interruptja akkor, ha az AT System Request billentyűjét lenyomják, illetve felengedik. Rendszerszinten későbbi felhasználásra fenntartva. Az alfunkció pillanatnyilag üres IRET-el ér véget. A 15H interrupt átirányításával felhasználhatjuk arra, hogy a billentyűzetről a SYSTEM REQUEST lenyomásával valamilyen funkció végrehajtását kikényszeríthessük.

X.6.7. Várakozás

```
Funkciókód:              AH = 86H                    INT     15H
Javasolt funkciónév:     EXT_WAIT
Be:
        CX:DX    - várakozás ideje milliomod másodpercekben
```

A funkció milliomod másodperces egységekben meghatározott ideig várakozik, azután adja vissza a vezérlést, ezzel a timer-interruptnál sokkal finomabb időzítést tesz lehetővé.

A várakozási idő egysége egyébként 976 millomod másodperc; a CX:DX-ben megadott duplaszó gyakorlatilag csak ennek többszörösét képes előállítani. Ezt úgy kell értenünk, hogy a CX:DX értéket valóban milliomod másodperces értéknek tekinti a rendszer, de ha a várakozási érték nem haladja meg a 976-ot, akkor legalább 976 milliomod másodpercet vár. Ha nagyobb, mint 976, de nem éri el az 1952-t, akkor 1952 milliomod másodpercig vár, és így tovább. Ebből adódik, hogy egy tíz másodperces várakozáshoz a CX:DX regiszterpárban a 989680H értéket kell megadni (aki nem hiszi, számoljon utána).

X.6.8. Blokk mozgatása

```
Funkciókód:              AH = 87H                    INT     15H
Javasolt funkciónév:     EXT_MOVBLK
Be:
        CX       - mozgatandó byte-ok száma (nem több, mint 32K)
        ES:SI    - a leíró táblázat kezdőcíme
```

Ez a funkció kínálja az egyetlen legális utat a memóriakiterjesztés kezeléséhez. Az IBM AT nem Intel 8088, hanem Intel 80286 processzorra épült, amely kétféle üzemmódot ismer: a Real Mode-t, a valós módot (ekkor azonos a 8086 processzorral, csak gyorsabb) és a virtuális módot, amelyben a processzor kifejtheti igazi képességeit: ekkor 8 Megabyte központi tárat tud címezni. Az MS-DOS alatt csak a 8086-nak megfelelő legfeljebb 1 Megabyte-nyi memória címezhető.

A nagyobb tárak MS-DOS alatti címzéséhez e funkciót használjuk. Belépéskor ES:SI egy 6 elemű, elemenként 8 byte-os hosszúságú blokkot címez, amely leírja a blokkmozgatás paramétereit:

Szegmens felső határa
Szegmens kezdőcíme, összesen 24 biten: alsó 8 bit; középső 8 bit; felső 8 bit
Elérési jogok kódja
mindig 0 értékű

A hat elemű (nyolc byte hosszú) leírótábla:

0. elem	8 darab nulla byte
1. elem	Magát a tömböt leíró tábla; a felhasználó által nullára előkészítve; a ROM BIOS módosítja (8 db 0 értékű byte)
2. elem	Kiindulási blokk leíró táblája; a felhasználó által előkészítve (8 byte)
3. elem	Célblokk leíró táblája; a felhasználó által előkészítve (8 byte)
4. elem	A Protected Program kódszegmensének leíró táblája; a felhasználó által nullára előkészítve; a ROM BIOS módosítja (8 db 0 értékű byte)
5. elem	A Protected Program stack-szegmensének leíró táblája; a felhasználó által nullára előkészítve; a ROM BIOS módosítja (8 db 0 értékű byte)

A fentiek alapos megértéséhez természetesen tisztában kell lennünk az Intel 80286 processzor tárkezelésének és memóriavédelmének minden ravaszságával. Mivel könyvünknek nem tárgya, röviden tudunk csak megemlékezni erről. A 80286 lapokra osztja a memóriát. A felosztást, valamint a memórialapok és a processzor által (egyidejűleg) futtatott programok egymáshoz rendelését a rendszerben Protected Mode-ban futtatott program (ami megfelel más rendszerek supervisorának) végzi. Minden laphoz tartozik egy ilyen leíró blokk, amely megadja a lap kezdőcímét, a lap hosszát és az elérési jogot (csak olvasás, csak írás, csak végrehajtás stb.). A leíró blokkokat csak a Protected Program módosíthatja, így a tárban egyidejűleg elhelyezkedő programok egymástól teljesen védve vannak.

Nem túlzottan gyakorlott programozók számára csak azt tanácsoljuk: ne végezzenek semmiféle közvetlen manipulációt a memóriakiterjesztésen. MS-DOS alatt e területet célszerűen a RAM DISK céljaira használhatjuk fel; ekkor pedig bízvást használhatjuk a VDISK drivert a /E kapcsolóval (mely szintén e funkciót használja).

X.6.9. Memóriakiterjesztés lekérdezése

Funkciókód: AH = 88H INT 15H
Javasolt funkciónév: EXT_EXTMEMSIZ
Be:
semmi egyéb
Válasz:
AX - az 1 Mb feletti memóriakiterjesztés hossza kilobyte-okban

X.6.10. Átváltás virtuális módba

Funkciókód: AH = 89H INT 15H
Javasolt funkciónév: EXT_SWVIRTM
Be:
AL - az elsődleges hardware-interrupt szint kezdőindexe (0-7. hardware interrupt-kérések kezdőindexe az interrupt vektorok között)
BL - a másodlagos hardware-interrupt szint kezdőindexe (8-15. hardware interrupt-kérések kezdőindexe az interrupt vektorok között)
ES:SI - a nyolcelemű táblaleíró blokkokból álló tömb címe.

A táblaleíró szerkezetét lásd az előző funkciónál. A nyolc táblaleíró blokknak az alábbiakat kell tartalmaznia:

0. elem	nulla
1. elem	Magát a tömböt leíró tábla; a felhasználó által előkészítve
2. elem	Interrupt-leíró tábla; az interrupt vektorokat tartalmazó lap leírása a felhasználó által előkészítve
3. elem	Az adatszegmens leíró táblája, a felhasználó által előkészítve
4. elem	Az extra szegmens leíró táblája, a felhasználó által előkészítve;
5. elem	A stack-szegmens leíró táblája, a felhasználó által előkészítve;
6. elem	A kódszegmens leíró táblája, a felhasználó által előkészítve;
7. elem	Az ideiglenes ROM BIOS kódszegmensének leíró táblája; a felhasználó által előkészítve;

A programozásban nem eléggé gyakorlottak (például, aki nem tudja, hogy a fentiek mit jelentenek) ne használják ez a funkciót.

X.6.11. Várakozás valamely eszközre

Funkciókód: AH = 90H INT 15H
Javasolt funkciónév: DEV_DEVWAIT
Be:

AL – az eszköz típusának kódja (mely arra utal, hogy az adott eszköz szekvencális hívásokkal újra felhasználható, újra beléptethető [többszörösen használható], vagy van rá valamilyen time-out, várakozási idő.)

A példa kedvéért néhány eszköz típuskódja:

00H – Winchester
01H – Floppy-disk
02H – klaviatúra
80H – hálózat
FEH – nyomtató

Egy processz így közölheti a rendszerrel, hogy a megadott típusú eszközre várakozik. A vezérlést csak akkor kapja vissza, ha az adott eszköz rendelkezésre áll (ti. befejezte eddigi tevékenységét). Az MS-DOS és a ROM BIOS jelenlegi megvalósításában nincsen multi-tasking; e funkció azonnal visszatér.

X.6.12. Eszköz felszabadítása

Funkciókód: AH = 91H INT 15H
Javasolt funkciónév: DEV_DEVFREE
Be:
AL - az eszköz típusának kódja (l. előző funkció)

Egy processz így közölheti a rendszerrel, hogy a megadott típusú eszköz felhasználását befejezte, és felszabadítja további felhasználásra.

Az MS-DOS és a ROM BIOS jelenlegi megvalósításában nincsen multi-tasking; e funkció azonnal visszatér.

X.6.13. Konfiguráció lekérdezése

Funkciókód: AH = C0H INT 15H
Javasolt funkciónév: EXT_EQCHK
Be:
Semmi egyéb
Válasz:
Carry - 1 ha valami hiba volt; 0 ha sikeres. Ekkor:
ES:SI egy (kilencbyte-os) táblázat címét tartalmazza:

0. byte	táblahossz (8)
1. byte	rendszermodell sorszáma
2. byte	rendszermodell alsorszáma
3. byte	BIOS verziószáma
4. byte	rendszerinformáció
többi	fenntartva; a legfontosabb byte-ok:

1. byte:
0FFH - IBM PC ROM BIOS rendszer
0FEH - IBM XT ROM BIOS rendszer
0FCH - IBM AT ROM BIOS rendszer
2. byte:
0-2 bit - fenntartva PS rendszerek számára
3 bit - a rendszer képes külső eseményre várakozni
4 bit - keyboard interrupt rutin a 09H által címzett
5 bit - valósidejű óra beépítve
6 bit - másodlagos interrupt vezérlő beépítve
7 bit - Winchester a 3. DMA csatornát használja

X.7. ROM BASIC elindítása

Csak IBM PC és XT !
Interrupt-sorszám: 18H
Javaslot interrupt-név: BS_BASICSTART
Be:
semmi
Válasz:
semmi
Elindítja az F6000H-as abszolút címen kezdődő ROM Basic interpretert. Ebből a

SYSTEM

Basic paranccsal léphetünk ki.

X.8. Rendszerindítás

Csak IBM PC és XT !
Interrupt-sorszám: 19H
Javasolt interrupt-név: BS_BOOTSTRAP

Ez az interrupt az operációs rendszer újraindítására szolgál. Az utasítás kiadása után a rendszer melegindítást hajt végre, azaz nem kerül sor a memória, a berendezés teljes ellenőrzésére, csak a rendszer betöltésére. Ha ennél keményebb eszközökre van szükség, használjuk ki azt az ismeretet, hogy a vezérlés a processzor bekapcsolása után az FFFF0H címre, azaz az FFFFH paragrafuson kezdődő szegmens 0 címére adódik. Ha valóban mindent elölről akarunk kezdeni, akkor adjuk át ide a vezérlést.

X.9. Az óra kezelése

Interrupt-sorszám: 1AH
Javasolt interrupt-név: BS_TMCNT

Az MS-DOS egy négy byte hosszúságú számlálót használ a gép bekapcsolása óta eltelt idő mérésére. Ezt a számlálót a timer hardware-interruptját kezelő rutin növeli meg másodpercenként körülbelül tizennyolcszor. A számláló kezdeti értékének beállítását a TIME belső parancs végzi.

Mint afféle négybyte-os számláló, a timer összesen körülbelül hét és fél évig lenne képes másodpercenként tizennyolcszor növekedni. A valóságban az MS-DOS mindig megvizsgálja, hogy nem éjfél van-e éppen; éjfélkor viszont nullát ír a számlálóba, és beállít egy változót, mely jelzi, hogy legalább egy nap eltelt. Ez a változó azonban valóban csak azt jelzi, hogy legalább egyszer volt éjfél a gép bekapcsolása óta. Semmilyen módon nem tudhatjuk meg azonban, hogy hányszor volt. Ez, úgy tűnik, nem a legszerencsésebb megoldás; kívülről tekintve bármelyik indikátor felhasználható számlálóként is; lehet, hogy van valami oka annak, hogy mégsem így használják. Most pedig lássuk, mit nyújt az időszámláló kezelését végző software interrupt!

X.9.1. Az időszámláló lekérdezése

Funkciókód: AH = 00H INT 1AH
Javasolt funkciónév: CLK_GETTCNT
Be:

Semmi egyéb

Válasz:

CX:DX az időszámláló értéke (CX a duplaszó felső, DX pedig az alsó tizenhat bitje)

AL nulla, ha az utolsó rendszerindítás, időbeállítás vagy az idő kiolvasása óta nem telt el huszonnégy óra, azaz nem értük még el az éjfélt; nullától különböző, ha az utolsó rendszerindítás óta legalább egyszer elértük az éjfélt.

A funkció által szolgáltatott duplaszót elég egyszerű az általunk kedvelt óra-perc-másodperc alakra hozni. Az eljárás a következő: első lépésként el kell osztani a négybyte-os értéket a timer-impulzusok másodpercenkénti számával (ez a lehető legnagyobb pontossággal 65543, vagy 10007H, mert az időinterrupt nem pontosan 18,2-szer következik be másodpercenként); a hányados az órák száma, a maradékkal tovább számolunk. Ezt el kell osztani a percenként bekövetkező timer-interruptok számával (kerekítve 1092, 444H). A hányados adja a perceket, a maradékkal tovább számolunk. Ezt most már 18-al kell osztani, hogy a másodperceket kapjuk meg; a századmásodperceket a maradékból már triviálisan határozhatjuk meg.

A gyakorlatban ennek a négybyte-os számlálónak az értékére sosincs szükségünk. Két okból kérhetjük egyáltalán ezt a szolgáltatást: az egyik az, hogy valóban az időre vagyunk kíváncsiak; a másik egy álvéletlen szám generálása. Ha IBM AT gépünk van, akkor az óra lekérdezése sokkal egyszerűbb a 02H funkcióval; véletlenszerű számok generálására azonban elég jól használható ez a funkció; az eredményül kapott duplaszó alsó byteja elég gyorsan változik ahhoz, hogy ha nem egy állandó hosszúságú ciklusban nyúlunk le az értékért, akkor a kapott érték véletlenszerűnek tekinthető.

X.9.2. Az időszámláló beállítása

Funkciókód: AH = 01H INT 1AH
Javasolt funkciónév: CLK_SETTCNT
Be:
CX:DX az időszámláló kívánt értéke (CX a duplaszó felső, DX az pedig alsó tizenhat bitje)

Csak IBM AT !

X.9.3. A valós idő lekérdezése BCD-alakban

Funkciókód: AH = 02H INT 1AH
Javasolt funkciónév: CLK_GETTIME
Be:
Semmi egyéb
Válasz:
Carry – 0, ha az óra működik,
1, ha nem
Ha az óra működik, akkor:
CH – az óra két BCD-számjeggyel
CL – a perc két BCD-számjeggyel
DH – a másodperc két BCD-számjeggyel
DL – 0, ha túlléptük az éjfélt,
1 különben

Ez a funkció az IBM AT valósidejű óráját olvassa, és elvégzi azt a konverziót, amely a 00H funkció használata esetén ránk várna. Ha tehát kényelmesebb szolgáltatásra vágyunk és nem pont a négybyte-os számlálóra van szükségünk, akkor jobb, ha ezt a funkciót hívjuk; a válaszul kapott BCD értékek képernyőn való megjelenítése sokkal egyszerűbb.

Csak IBM AT !
X.9.4. A valós idő megadása BCD-alakban

Funkciókód: AH = 03H INT 1AH
Javasolt funkciónév: CLK_SETTIME
Be:
Minden paraméter pontosan, mint a 02H funkciónál:
CH, CL - az óra, illetve a perc BCD-alakban
DH - a másodperc BCD-alakban
DL - 1, ha szimulálni akarjuk az éjfél átlépését,
0, ha nem
Válasz: nincs.

E funkció az IBM AT valósidejű órájának beállítását végzi az általunk megadott paraméterek alapján. A leírás, miként a szakirodalom, azt sugallja, hogy a rendszer órája (system clock) és a valósidejű óra (real time clock) között van valami különbség. Valóban van is: a valósidejű óra az IBM AT gépek úgynevezett CMOS RAM területén van nyilvántartva; mikor bekapcsoljuk a gépet, akkor a ROM BIOS innen tölti fel a rendszeridőt. Az MS-DOS felhasználói szintű TIME parancsa, és az óra kezelésére szolgáló MS-DOS és ROM BIOS parancsai a rendszeridőt kezelik, és nem nyúlnak a valós időhöz. Ha elrontjuk a rendszeridőt, akkor a gép kikapcsolása és ismételt bekapcsolása után ismét a valósidejű óra szerinti időt kapjuk meg. Ezzel kapcsolatban konkrét vizsgálatokat csak az utóbbi időkben, egy AT utánzaton folytattam. Tapasztalataim szerint a 00H, 01H, 02H és 03H funkciók mindkét paramétert módosítják; tehát minden, az időt érintő parancs hat a valósidejű órára és dátumszámlálóra is; semmi sem őrzi meg a felhasználói beavatkozástól sem a rendszeridőt, sem pedig a valósidejű órát. Nem szükséges a SETUP program futtatása, ha állítani akarjuk a gép belső óráját (ez naponta néhány másodpercet késik, egy-két havonta az időt korrigálni kell).

Csak IBM AT !
X.9.5. A dátum lekérdezése BCD alakban

Funkciókód: AH = 04H INT 1AH
Javasolt funkciónév: CLK_GETDATE
Be:
Semmi egyéb.
Válasz:
Carry - 1, ha az óra nem működik,
0, ha igen;
CH, CL - az évszázad, illetve az év két BCD-jeggyel
DH, DL - a hónap, illetve a nap két BCD-jeggyel

Csak IBM AT !
X.9.6. A dátum beállítása BCD alakban

Funkciókód: AH = 05H INT 1AH
Javasolt funkciónév: CLK_SETDATE
Be:
Pontosan, mint a 04H funkció válaszértékei:
CH, CL - az évszázad és az év két BCD-jeggyel
DH, DL - a hónap és a nap két BCD-jeggyel
Válasz:
Nincs válaszérték

Csak IBM AT !
X.9.7. Jeladás kérése adott időpontra

Funkciókód: AH = 06H INT 1AH
Javasolt funkciónév: CLK_SETALARM
Be:
CH, CL - a kívánt időpont órája és perce, két BCD-jegy
CH - a kívánt időpont másodperce, két BCD-jegy
Válasz:
Carry - 1, ha az óra nem működik, 0, ha igen

A jeladás úgy működik, hogy a ROM BIOS állandóan összeveti a rendszeridőt az 1AH interrupt e funkciója segítségével megadott időponttal, és ha a gép az adott időben be van kapcsolva (és persze a rendszer működik), akkor aktivizálja a 4AH sorszámú interruptot. A felhasználó ide tetszőleges rutint fűzhet be, tehát tetszőleges módon fogadhatja a rendszertől érkezett jelzést. Fontos még, hogy egyidejűleg csak egy riasztás lehet érvényben. A jeladás naponta ismétlődik akkor, ha a gép be van kapcsolva; a következő (07H kódú) funkcióval lehet leállítani.

Csak IBM AT !
X.9.8. A riasztás kikapcsolása

Funkciókód: AH = 07H INT 1AH
Javasolt funkciónév: CLK_CANCALARM
Be:
Semmmi egyéb.
Válasz:
Nincs válaszérték.

E funkció a 06H segítségével bekapcsolt jeladáskérés letiltására szolgál. Amíg le nem tiltjuk (vagy át nem programozzuk), a jeladás minden nap ugyanazon pillanatában bekövetkezik.

X.10. CTRL-BREAK felhasználói interrupt

Interrupt-sorszám: 1BH
Javasolt interrupt-név: BS_CTBREAK

Amikor lenyomjuk a CTRL-BREAK kombinációt, akkor a billentyűzetet kezelő hardware interrupt rutin (elvégezvén egyéb teendőit, lásd "A klaviatúra" fejezet) az interrupt rutinból való kilépés előtt még kiadja az

```
INT     1BH
```

utasítást, aktivizálja az 1BH interrupt vektor által címzett rutint. Ezt a vektort egyébként bekapcsoláskor a ROM BIOS úgy készíti elő, hogy egy IRET utasításra mutat. A felhasználói program megteheti, hogy átirányítja ezt ez interruptot saját rutinjára, és segítségével elvégez valamilyen, ekkor szükséges teendőt. Ez akár az is lehet, hogy kiirtja a klaviatúra ciklikus bufferéből a CTRL-BREAK-re utaló (0,0) értékű byte-párt, törölve ezzel a CTRL-BREAK feltételt.

Amennyiben ezt a lehetőséget ki akarjuk használni, akkor legyünk óvatosak, mivel a rutinunk még interrupt szinten fut, hiszen a klaviatúra hardware interrupt rutinja az interrupt vezérlő újraindítása előtt aktivizálja az 1BH interruptot. Interrupt szinten pedig mindig a lehető legkevesebb időt szabad csak eltöltenünk, mivel ezzel a további interruptok fogadását megakadályozzuk, és időzítési problémákat okozhatunk.

X.11. Timer felhasználói interrupt

Interrupt-sorszám: 1CH
Javasolt interrupt-név: BS_TMUSER

Rengeteg olyan alkalmazás van, amelyhez szükségünk lenne a Timer áramkör szolgáltatásainak felhasználására. Sajnálatos, hogy ez igen nehezen valósítható meg, mivel a Timer mindhárom csatornája "oda van drótozva" valamilyen (mégpedig nélkülözhetetlen) hardware-elemhez (Timer Tick, azaz az idő-interrupt, a DMA 0. csatornája, a dinamikus memória frissítése kedvéért, és végül a hangszóró).

Egy korlátozott időzítő-számláló funkció megvalósítását segíti ez az interrupt, mely mögött alapértelmezésben egy egyszerű IRET utasítás van. Ezt a "rutint" a Timer hardware interrupt

rutinja hívja meg minden egyes belépésekor utolsó teendőjeként. Ha az időszámláló kezeléséhez hasonló funkciót akarunk megvalósítani, akkor az 1CH interruptot "szólítsuk magunkhoz" (természetesen a 25H MS-DOS funkcióval).

Ezen interrupt jellegzetes felhasználása például olyan "muzsikáló" program megvalósítása, amely "háttérben" dolgozik (lásd "A hangszóró kezelése" c. fejezetet).

A felhasználói interrupt meg nem szakítható szinten dolgozik, ezért igen óvatosnak kell lennünk; ügyeljünk arra, hogy a lehető legkevesebb időt használjuk el, mert a rutinunk miatt késleltetett egyéb interruptok összegabalyodhatnak!

X.12. Video paraméterek

Interrupt-sorszám: 1DH
Javasolt interrupt-név: BT_VIDEOTB

A video paraméterek táblázata a Motorola 6845 képernyővezérlő áramkör felprogramozásához szükséges konstansokat tartalmazza. A képernyővezérlő program a képernyő kezdeti előkészítése, vagy üzemmódváltás esetén az innen kiolvasott értékeket írja be a 6845 áramkör megfelelő regisztereibe.

Ez a táblázat a ROM BIOS területén helyezkedik el. Tulajdonképpen luxus, hogy a ROM BIOS nem közvetlen, hanem indirekt címzéssel éri el ezt a táblázatot, hiszen saját beépített üzemmódjai úgyis azt tételezik fel, hogy a 6845 az abban foglaltak szerint van felprogramozva. Ezen a ponton olyan rugalmas a ROM BIOS megvalósítása, amennyire sok más helyen is annak kellene lennie. Ha ugyanis azt akarjuk, hogy a 6845 programozásához más értékeket használjon a képernyővezérlő program, akkor másik táblázat címét tölthetjük be az 1DH interrupt vektorba.

> Megjegyzés: ez a rugalmasság sokkal, de sokkal fontosabb lenne például a billentyűzet kezelésében. Milyen hasznos lenne, ha egy interrupt vektor tartalmazná a scan-kódokat definiáló táblázat címét, melyet a felhasználó tetszése szerint adhatna meg, és nem kellene viszonylag jelentéktelen, de fontos változtatások kedvéért teljesen újraírnia a klaviatúra hardware interrupt rutinját!

Végezetül, csak a teljesség kedvéért iktassuk ide a táblázat szerkezetét! Ez a tábla több részből áll. Az első, amely négyszer tizenhat byte hosszúságú, a 6845 felprogramozásához hasz-

nálandó konstansokat tartalmazza. Ebben a részben az első tizenhat byte a 40x25, a második a 80x25 alfanumerikus üzemmódhoz, a harmadik a grafikus módhoz szükséges konstansok sorozatát tartalmazza a regiszterek sorrendjében, végül a negyedik a fekete-fehér adapter felprogramozási paramétereinek sorozata.

A táblázat második része négy szóból áll, és az egyes üzemmódokhoz tartozó képernyőterületek hosszát tartalmazza byte-okban számítva. Az első szó tehát a 40x25 karakteres alfanumerikus, a második a 80x25 karakteres alfanumerikus üzemmód képernyőterületének hosszát jelenti byte-okban számolva (és kicsit kerekítve, hiszen az érték 800H, illetve 1000H, míg a valódi igény csak 720H, illetve F40H); a harmadik és negyedik szó pedig a közepes és a nagyfelbontású grafikus üzemmódokban használatos képernyőterület hossza.

A harmadik rész nyolc byte-nyi hosszúságú; tartalma pedig az egyes üzemmódokhoz tartozó sorhosszúság, karakterben számolva. Tehát ez a tábla másképpen van kódolva, mint az előzőek. Itt az egyes üzemmódok szolgálnak indexül (a 7 üzemmód a fekete-fehér adaptert jelenti).

Végül a tábla negyedik része ugyanilyen kódolással azt a módparancsot tartalmazza, amit a megfelelő felprogramozás után ki kell adni az adapternek. Számunkra ez a tábla a legfontosabb akkor, ha egyszer-egyszer le akarjuk kapcsolni a képet a monitorról, hiszen tudhatjuk az érvényben levő üzemmódot (lekérdezhető), s ezután az üzemmód alapján innen kiválasztott értékben kell törölni a VIDEO ENABLE bitet, és az így kapott értéket kiadni az adapter vezérlőportjára.

Saját tábla használata esetén egy ugyanilyen szerkezetű, de a kívánt értékeket tartalmazó táblázatot kell csinálnunk. Lássuk a táblázat értékeit úgy, ahogyan a rendelkezésemre álló IBM AT memóriájából kiolvastam (vö. "A Motorola 6845 regiszterei" fejezet)!

```
F000:F0A4   38 28 20 0A 1F 06 19 1C   02 07 06 07 00 00 00 00
F000:F0B4   71 50 5A 0A 1F 06 19 1C   02 07 06 07 00 00 00 00
F000:F0C4   38 28 20 0A 7F 06 64 70   02 01 06 07 00 00 00 00
F000:F0D4   61 50 52 0F 19 06 19 19   02 0D 0B 0C 00 00 00 00
F000:F0E4   00 08 00 10 00 40 00 40   28 28 50 50 28 28 50 50
F000:F0F4   2C 28 2D 29 2A 2E 1E 29   ...
```

Tehát: az F0A4H offsettől az F0B3H-ig olvashatók azok a felprogramozási konstansok, amelyek 40x25 karakteres alfanumerikus üzemmódban használatosak; az F0E4H-F0E5H szó tartalmazza ugyanezen üzemmód képernyőterületének hosszát byte-okban. Ezen módok kódja 0 vagy 1; a tábla harmadik és negyedik részében, tehát a

0. és 1. helyen, pontosan az F0ECH és F0EDH, illetve az F0F4H és F0F5H offseteken szerepelnek az ide tartozó sorhosszak (28H és 28H), illetve a módparancsok értékei (2CH és 28H; csak ez utóbbi különbözteti meg a fekete-fehér és a színes módot).

A táblázat többi részének értelmezése hasonló: a táblázat első két része tehát "összeejti" a fekete-fehér, illetve a színes üzemmódokat, míg a másik két rész minden üzemmódra külön értékeket tartalmaz. Az üzemmódok belső reprezentációjában a hetes kód a fekete-fehér adapter használatát jelzi; ahol szerepel egyáltalán, ott mindig az utolsó helyen van az erre az üzemmódra utaló érték vagy értéksorozat.

Most lássunk egy olyan struktúrát, amely segít a paraméterek értelmezésében:

```
VID_PARTAB      STRUC

REGS_4025       DB      16  DUP ( ? )   ;40x25 regiszterek
REGS_8025       DB      16  DUP ( ? )   ;80x25 regiszterek
REGS_GRAPH      DB      16  DUP ( ? )   ;Grafikus regiszterek
REGS_MONO       DB      16  DUP ( ? )   ;Mono adapter regiszt.
REGEN_SIZ_4025  DW       0              ;40x25 képernyőterület
REGEN_SIZ_8025  DW       0              ;80x25 képernyőterület
REGEN_SIZ_GMR   DW       0              ;320x200 képernyőter.
REGEN_SIZ_GHR   DW       0              ;640x200 képernyőter.
LINE_SIZES      DB       8  DUP ( ? )   ;Sorhosszak módonként
MODE_COMMS      DB       8  DUP ( ? )   ;Módparancsok módonként

VID_PARTAB      ENDS
```

X.13. Diskette paraméterek

Interrupt-sorszám: 1EH
Javasolt interrupt-név: BT_DISKETTETB

A diskette paramétertáblázat pontosan olyan szerepet játszik a lemezegységeket vezérlő áramkör kezelésében, mint a video paramétertábla a 6845 programozásában. Ez a tábla azonban mindössze 11 byte hosszú, és alig valamivel több paramétert tartalmaz (egyes byte-okba két paramétert is írtak). A paraméterek főként időzítési adatokat adnak meg.

Akik közelebbről ismerik az IBM PC-t, tudják, hogy az eredeti paramétertáblázat ROM területen van. Ez a tábla azonban túl óvatos értékeket tartalmazott. A gyakorlat igazolta, hogy a drive sokkal "szigorúbb" (az egyes műveletekre nézve sokkal rö-

videbb időket engedélyező) értékekkel is jól működik. Ezért az újabb ROM BIOS változatok (érintetlenül hagyva a beégetett értékeket) hatékonyabb táblázatot hoznak létre RAM területen, és annak címét töltik az 1EH vektorba; így rövidülnek a várakozási idők, tehát gyorsulni fog a rendszer.

Az alábbiakban megadjuk a diskette paramétertáblázat elemeinek sorrendjét és rövid leírását. (E tábla felhasználásáról bővebben olvashatunk a "Lemezkezelés" c. fejezetben.)

- 0 byte: Step Rate Time és Head Unload Time: a 0-3 bitek tartalmazzák a sávról sávra lépés idejét (SRT), a 4-7. bitek pedig a fej felemeléséhez szükséges időt;
- 1 byte: Head Load Time és DMA Mode: a legalsó bit 0. értéke DMA segítségével végzett műveleteket ír elő. A felső hét bit a fej beállítási idejét adja meg. E byte-ok a floppy-kontrollernek kiadandó "SPECIFY" parancs adatbyte-jai (lásd "A diskette fizikai kezelése");
- 2 byte: Wait Time until Motor off: azt az időt szabja meg, amennyi elteltével a motort ki kell kapcsolni az utolsó lemezművelet után. Ezt az időt a timer (08 interrupt) szerint mérik, egysége kb. 1/18 másodperc. Szokásos értéke 37 (25H), ami durván két másodperc;
- 3 byte: Sector Length Code: ugyanaz a paraméter, amelyet a szektorazonosítókban olvashatunk "N" elnevezéssel, lásd a "Lemezkezelés" fejezetet;
- 4 byte: Last Sector Number: a sáv hossza szektorokban (értéke általában 9);
- 5 byte: Gap Length between Sectors for Read/Write: ez az érték a szektorok közötti távolságra utal. Ebből tudja a drive, hogy mennyit kell várnia két szektor között, mielőtt új szektorazonosítót helyezhet el (formázáskor érdekes paraméter);
- 6 byte Maximum Data Length if Sectorlength not Specified: ez a fontos paraméter adja meg a szektorhosszt olyan esetekre, mikor az adott szektor hossza a 4. byte-ban nincs megadva, azaz N értéke 0;
- 7 byte: Gap Length between Sectors for Format: a formázás során alkalmazandó szektortávolság, amely természetesen nagyobb, mint az írás/olvasási műveletekhez megadott távolság;
- 8 byte: Data Value for Format: az a nyolcbites érték, mellyel a formázás során a kontroller felülírja a szektorok adatbyte-jait. Szokásos értéke F8H, CP/M esetén E5H;
- 9 byte Head-Settle Time:, fejmegnyugvási idő: ez az érték adja meg azt, hogy mennyit kell várni a fej pozicionálása után a mozgás okozta rezgések csillapodására;
- 10 byte Motor Start up Time: a lemezegység motorjának felpörgetéséhez szükséges időt adja meg.

X.14. Grafikus karaktergenerátor

Interrupt-sorszám: 1FH
Javasolt interrupt-név: BT_GRCHARG

Már az előző két esetben is láttuk, hogy az interrupt vektorok nem mindig szubrutinok címét tartalmazzák. Ez a vektor is egy táblázatra mutat, amely (értelmes felhasználás esetén) egy százhuszonnyolc elemű, elemenként nyolc byte hosszúságú karaktergenerátor. Ennek "i"-edik eleme a "128+i" kódú karakter rajza. Az itt megrajzolt karaktert tudjuk kitenni a képernyőre bármilyen grafikus üzemmódban, ha a kiíró BIOS funkciók valamelyikének ezt a kódot adjuk át.

Magát a karaktert egy nyolcbyte-os tömb írja le. A karakter egy 8x8 elemi pontból álló négyzet (fizikailag inkább keskenyebb ldalán álló téglalap, hiszen az elemi pontok vízszintes irányban közelebb esnek egymáshoz, mint függőlegesen). A négyzet sorai megfelelnek a byte-oknak (azaz a byte-ok fekszenek); a bitek kiosztása fordított, vagyis balról jobbra úgy haladunk, hogy a magasabb helyiértékű bitektől megyünk az alacsonyabbak felé. Amelyik bit értéke egy, ott a grafikus ernyőn világító pontot fogunk látni, ahol pedig nulla, ott az alapszínt kapjuk meg. Erre példát "A képernyő" fejezetben láthatunk.

X.15. Riasztás a valósidejű óra segítségével

Csak IBM AT !
Interrupt-sorszám: 4AH
Javasolt interrupt-név: RTC_ALARM

Ezt az interrupt vektort megfelelő előkészítés után az IBM AT valósidejű óráját kezelő rendszer aktivizálja akkor, ha a valósidejű óra elérte a beprogramozott értéket (1AH interrupt, 06H és 07H funkció). Az interrupt vektor alapértelmezésben egy IRET utasításra mutat; ennek helyére fűzhet be a felhasználó valamilyen, az óra lejárásakor meghívandó rutint.

Egy ilyen szolgáltatás a következő lehet: a program belép, indítási paraméterként veszi át a kívánt időpontot; előkészíti a 4AH interrupt vektort, amely mögé egy látványos, de veszélytelen rutint dug, majd előkészíti a jeladó programot, végül rezidens módon kilép. Az interrupt bekövetkezése után leállítja a jeladó rendszert. Nagyon jó volna, ha végleg ki tudna lépni, de ez bajos. A 4A interrupt ugyanis hardware-úton aktivizálódik, és nem tudjuk, hogy éppen felhasználói vagy MS-DOS szinten

vagyunk. Mint ismeretes, MS-DOS szintről MS-DOS funkciót nem szabad hívni. Eldönthető ugyan, hogy milyen szinten vagyunk (az DS-DOS funkciók közül a nem dokumentált 34H sorszámú visszaad egy pointert, amely egy byte-ra mutat és annak értéke megmondja), de hardware-interrupt szinten semmit sem tehetünk akkor, ha történetesen MS-DOS szintet szakítottunk meg; vissza kellene adni a vezérlést a megszakított programnak, hogy a funkció lefuthasson, de akkor hogyan kapjuk meg ismét a vezérlést? Maradjunk annyiban, hogy programunk nem lép ki. Ha jót akarunk magunknak, akkor mindig a rendszerindítás után hívjuk meg... Ez egyébként olyan szolgáltatás, amely mindnyájunk számára hasznos lehet; ha valami dolgunk van, akkor nem kell állandóan az órát figyelnünk, hanem egész egyszerűen meghívjuk ezt a programot, és nyugodtan dolgozunk a gépen; a program majd szól, ha be kell fejeznünk a munkát.

```
COMMENT *
A file neve:                              alarm.asm
Fordítás:                          MASM opciók nélkül
Szerkesztés:                       LINK opciók nélkül
Futtatás:                          SPDRV   (hangszóró-driver)
                                   ALARM óra:perc:másodperc
        ahol "óra", "perc", "másodperc" a kívánt riasztási idő
        órája, perce, másodperce kétjegyű decimális számokkal *

STACK_SIZE      EQU     100H
;
; Include fejezet
;
        .XLIST
        INCLUDE         DOSCLL.INC
        .LIST
;
; Makró fejezet
;-----------------------------------------------;
; S_ITV  az IT_NUM interrupt vektor eredeti tartalmának
;               mentése és új értékkel való felülírása
;       Lásd a IT-MOD.INC file-t!
;-----------------------------------------------;
; R_ITV  az IT_NUM interrupt vektor helyreállítása
;               a SAVE_ADDR által adott pozícióról
;       Lásd a IT-MOD.INC file-t!
;-----------------------------------------------;
```

```
;--------------------------------------------------------;
; Adatterület
;--------------------------------------------------------;
                                                         ;
DATA    SEGMENT PARA    PUBLIC  'DATA'                  ;
                                                         ;
INT4A_SAVE      DD      0                                ;Mentőterület,
INT4A_OWN       DD      INT4A_ENTRY                      ; saját belépé-
                                                         ; si pont (4AH)
TIME_HOUR       DB      0                                ;Munkaváltozók
TIME_MIN        DB      0                                ;
TIME_SEC        DB      0                                ;
                        ;-----------------------------;
                        ; A riasztási jel hangjainak leírása;
                        ;-----------------------------;
ALARM   LABEL   WORD                                     ;
        DW      TIM_SEC/9,      SND_C4                   ; összesen
        DW      TIM_SEC/9,      SND_E4                   ; háromszor
        DW      TIM_SEC/9,      SND_G4                   ; ismételve
        DW      TIM_SEC,        SND_C5                   ;
        DW      2*TIM_SEC,      SND_PAUSE                ;
                                                         ;
        DW      TIM_SEC/9,      SND_C4                   ;
        DW      TIM_SEC/9,      SND_E4                   ;
        DW      TIM_SEC/9,      SND_G4                   ;
        DW      TIM_SEC,        SND_C5                   ;
        DW      2*TIM_SEC,      SND_PAUSE                ;
                                                         ;
        DW      TIM_SEC/9,      SND_C4                   ;
        DW      TIM_SEC/9,      SND_E4                   ;
        DW      TIM_SEC/9,      SND_G4                   ;
        DW      TIM_SEC,        SND_C5                   ;
        DW      0,              0                        ;Sorozat vége
                                ;---------------;
                                ; Hibaüzenet szövege
                                ;---------------;
        MSGDEF  MSG_UNS, 'Clock is not active/alarm is set'
                                                         ;
DATA    ENDS                                             ;
;--------------------------------------------------------;
; Stack terület
;--------------------------------------------------------;
STACK   SEGMENT PARA    STACK   'STACK'
        DW      STACK_SIZE      DUP     ( ? )
STACK   ENDS                                             ;
```

```
;-------------------------------------------------------;
; Kódterület
;-------------------------------------------------------;
                                                        ;
CODE    SEGMENT PARA    PUBLIC  'CODE'                  ;
        ASSUME  CS:CODE                                 ;
                                                        ;
START:                                                  ;Belépési pont
                MOV     AX, SEG DATA                    ;DATA szegmens
                MOV     DS, AX                          ; bekészítése
        ASSUME  DS:DATA                                 ;
                                ;---------------;
                                ;IT-vektor előkészítése:
                                ;Timer alarm IT
                                ;---------------;
                S_ITV   AT_ALARM, INT4A_OWN, INT4A_SAVE
                                                        ;
                MOV     CH, ES:[PSP_CMDLINSIZ]          ;Paramétersor
                                                        ; hossza
                MOV     SI, PSP_CMDLIN                  ; és címe
CNV_LOOP:                                               ;
                CALL    C_A_BCD                         ;Az órák,
                MOV     TIME_HOUR, BL                   ;
                JZ      CNV_END                         ; (vége)
                JC      CNV_END                         ; (hibás)
                CALL    C_A_BCD                         ;A percek,
                MOV     TIME_MIN, BL                    ;
                JZ      CNV_END                         ; (vége)
                JC      CNV_END                         ; (hibás)
                CALL    C_A_BCD                         ;A másodpercek
                                                        ; konverziója
CNV_END:                                                ;
                MOV     DH, BL                          ; és
                MOV     CH, TIME_HOUR                   ; bekészítése
                MOV     CL, TIME_MIN                    ; CH, CL, DH-ba
                TIMCLL  TIM_SETALARM                    ;Alarm indítása
                JC      UNSUCC                          ;Nem sikeres
                MOV     DX, SEG MEMORY                  ;Sikeres,
                MOV     AX, ES                          ; rezidens
                SUB     DX, AX                          ; kilépés
                DOSCLL  DOS_STAYRES, 0                  ;
UNSUCC:                                                 ;Nem sikeres,
                R_ITV   AT_ALARM, INT4A_SAVE            ;IT vektor
                PRTSTR  MSG_UNS                         ; vissza,üzenet
                EXIT    1                               ; és kilépés
```

```
;-------------------------------------------------------;
; C_A_BCD       - ASCII karakterek konverziója BCD számmá
;       Input:
;               ES:SI   a karaktersorozat címe
;               CH      a karakterek száma
;       Output:
;               BX      a konvertált karakter
;               SI, CH  módosítva
;               Zero    0, ha van még;
;                       1, ha nincs már több karakter
;               Carry   0, ha szabályos volt (nn: alakú)
;                       1, ha szabálytalan
;-------------------------------------------------------;
C_A_BCD  PROC   NEAR                                    ;
                PUSH    AX                              ;
                XOR     BX, BX                          ;BX előkész.
                MOV     CL, 4                           ;Rotálási szám
C_LOOP:                                                 ;
                DEC     CH                              ;Van még kar.?
                JS      C_ENDLINE                       ;Nincs több
                MOV     AL, ES:[ SI ]                   ;Következőt
                INC     SI                              ; felolvassuk
                CMP     AL, ' '                         ;
                JE      C_LOOP                          ;Szóköz, tovább
                CMP     AL, ':'                         ;
                JE      C_ENDNUM                        ;Vége a számnak
                SUB     AL, '0'                         ;Konverzió
                JB      C_ERROR                         ;Nem számjegy
                CMP     AL, 9                           ;Számjegy volt?
                JA      C_ERROR                         ;Nem számjegy
                ROL     BX, CL                          ;4 x rotáljuk
                ADD     BL, AL                          ;
                JMP     C_LOOP                          ;Folytatjuk
C_ERROR:                                                ;
                STC                                     ;Nem számjegy!
                JMP     C_QUIT                          ;Carry=1
C_ENDLINE:                                              ;
                XOR     CL, CL                          ;Zero=1,Carry=0
                JMP     C_QUIT                          ;
C_ENDNUM:                                               ;Zero flag áll!
                TEST    AL, AL                          ;Zero=0,Carry=0
C_QUIT:                                                 ;
                POP     AX                              ;
                RET                                     ;
C_A_BCD  ENDP                                           ;
```

```
;-------------------------------------------------------;
; INT4A_ENTRY   - a riasztáskor aktivizálandó rutin kódja
;       Hardware-interrupt szerűen megírt rutin, csak a
;       8259 programozása marad ki (ROM BIOS végzi)
;-------------------------------------------------------;
INT4A_ENTRY     PROC    FAR                             ;;;
                                                        ;;;
                PUSH    AX                              ;;;Csak ennyi
                PUSH    BX                              ;;; regisztert
                PUSH    ES                              ;;; használunk!
                                                        ;;;
                MOV     AX, DATA                        ;;;DATA szeg-
                MOV     ES, AX                          ;;; mens az
        ASSUME  ES:DATA                                 ;;; ES-be
                                                        ;;;
                MOV     BX, OFFSET ALARM                ;;;Dallam címe
                SPKCLL  SND_START                       ;;;Dallam ki!
                TIMCLL  TIM_RESETALARM                  ;;;Alarm tilt.
                R_ITV   AT_ALARM, INT4A_SAVE            ;;;IT v. vissza
                                                        ;;;
                POP     ES                              ;;;
                POP     BX                              ;;;
                POP     AX                              ;;;
                                                        ;;;
                IRET                                    ;;;
                                                        ;
INT4A_ENTRY     ENDP                                    ;
CODE    ENDS                                            ;
;-------------------------------------------------------;
; Memory szegmens a rezidens kilépéshez
;-------------------------------------------------------;
                                                        ;
MEMORY  SEGMENT PARA    PUBLIC  'MEMORY'                ;
MEMORY  ENDS                                            ;
                                                        ;
;-------------------------------------------------------;
;
                END     START
```

Az SPKCLL makrót egyébként "A hangszóró kezelése" fejezetben láttuk; ez a program ugyanis zajongani fog, ha megkapja a riasztást. Futtatása előtt el kell indítani az ugyanott ismertetett hangszóró-driver programot.

Azt hiszem, hogy ez a program igazán nem túl bonyolult. Szeretném azonban felhívni a figyelmet egy olyan apróságra, amely

abszolút tisztességtelen, de az adott esetben nyugodtan megtehettük. Ez pedig az, hogy kihasználtuk: az óra/perc/másodperc ideiglenes tárolására szolgáló változók kezdeti értéke Ø. Ott használtuk ezt ki, hogy hibás vagy túl rövid paramétersor esetén nem léptünk rá a C_A_BCD rutin újabb meghívásaira, hanem egyből a Timer kommunikációs interrupt hívására ugrottunk. Ez a memóriából veszi fel a kívánt (már ti. a konverziós rutin által oda letett) értékeket. Ha pedig a rutint nem hívjuk meg, akkor a változók előkészítetlenek! Ez a lépés igazán nagyon csúnya és (főként ön)veszélyes. Az adott esetben azért használható, mert garantált: a változók tartalmát senki nem fogja módosítani, ha pedig az egész program újra fut, akkor a változók is újra értéket kapnak.

X.16. Winchester paramétertáblázatok

A Winchesterrel is felszerelt gépeken a rendszretöltés során az első Winchester paramétereinek táblázatát a 41H, az esetleges másodikét pedig a 46H interrupt vektor címzi meg. E táblázatok belső felépítésüket tekintve megegyeznek a floppy paraméter-táblázataival. Tartalmukról (minthogy számtalan típus van forgalomban, melyek paraméterei eltérőek) bővebben nem írhatunk.

A függelék: A ROM BIOS memóriaterületei

Ez a függelék nem a globális memóriáról ad vázlatos térképet, hanem a ROM BIOS RAM-területéről. Senkit nem bátorítok arra, hogy beleolvasson vagy egyenesen beleírjon a fizikai 400H címen (0040H:0000H) kezdődő és 100H hosszúságú ROM-BIOS memóriaterületre. Egyes esetekben azonban ez annyira csábító lehetőség, hogy nemigen tudunk ellenállni. Ilyen esetekben azonban csak két feltétel valamelyikének teljesülése adhat kellő biztonságot. Az egyik az, hogy garantálnunk kell: a ROM BIOS eredeti rutinja nem fog működésbe lépni (tehát nem fogunk ketten dolgozni ugyanazon memóriaterület felett). A másik eset, hogy teljes egészében, minden részletre kiterjedően megismerjük a ROM BIOS rutin által megvalósított algoritmust, és ennek ismeretében nyúlunk le a közvetlen memóriaterületre, pontosan ismerve tettünk következményeit. Az első út egyszerűbb: a hardware- és kommunikációs interruptot egyaránt saját kezelésünkbe vesszük. Ezzel biztosítjuk, hogy a ROM BIOS nem kaphatja meg a vezérlést. A másik (tudván tudva a hardware-interrupt kezelésének minden nehézségét) azért látszik bonyolultnak, mert a ROM BIOS rutinok nem igazán szépen megvalósított rutinok, rengeteg bennük a JMP utasítás; a kód át meg átfonja saját magát stb.

Az alább megadott memóriaterület ismerete tehát nem azért szükséges elsősorban, hogy közvetlenül felhasználjuk tartalmát. Ennél fontosabb szempont az, hogy a RAM terület ismerete megkönnyíti a rutinok olyan fokú megértését, amelyre szükségünk lehet.

Jómagam nem ismerem minden részletre kiterjedően még azoknak a rutinoknak a működését sem, amelyek helyett saját programozói gyakorlatomban más rutinokat építettem be a rendszerbe. Nem ismerem ezeket egész egyszerűen azért, mert nem volt rá szükségem. A megszerzett információk birtokában a feladatot meg tudtam oldani. Minthogy az eredeti rutinok által követett algoritmust mindenestül elvetettem (hiszen éppen azért volt szükséges másik rutin megírása), nem kellett minden részletében megismernem az elég bonyodalmasan lekódolt rutinokat. Felhívom azonban a figyelmet arra, hogy minden ilyen esetben saját adatterületet használtam, nem pedig a ROM BIOS eredeti (és a rutincserével használaton kívül helyezett) RAM területét.

Az alábbi táblázatban a bal oldalon szereplő értékek a 0040H paragrafuscímen kezdődő szegmens offsetjei. Mellettük a másik oszlopban a hozzájuk tartozó magyarázat olvasható.

		aszinkron vonalakat leíró terület
0000H - 0007H	(4 szó)	a COM1 - COM4 aszinkron vonali csatolók (INS 8250 ACE) alapportcímei
		nyomtatókat leíró terület
0008H - 000FH	(4 szó)	az LPT1 - LPT4 párhuzamos nyomtatóvezérlők alapportjai
		konfigurációs terület
0010H - 0011H	(1 szó)	az INT 11H által visszaadott konfigurációleíró szó; részletesen lásd az interrupt leírásánál.
0012H	(1 byte)	fenntartott
0013H - 0014H	(1 szó)	a RAM terület hossza kilobyte-okban
0015H - 0016H	(1 szó)	fenntartott
		billentyűzetleíró terület
0017H	(1 byte)	shiftstátusz; bitjei: 0. jobb shift lenyomva 1. bal shift lenyomva 2. CTRL lenyomva 3. ALT lenyomva 4. SCROLL LOCK aktív 5. NUM LOCK aktív 6. CAPS LOCK aktív 7. INSERT aktív
0018H	(1 byte)	másodlagos shiftstátusz (shiftbillentyűk folyamatos nyomásának nyilvántartására)
0019H	(1 byte)	alternatív klaviatúraleíró byte
001AH - 001BH	(1 szó)	pointer a klaviatúrabufferre - ez a szó mutatja a legkorábban beírt bejegyzést (a kiolvasó rutin e pointer felhasználásával olvas egy bejegyzést a bufferből)
001CH - 001DH	(1 szó)	pointer a klaviatúrabufferre - ez a szó mutatja a legrégebben beírt bejegyzést (a hardware-interrupt rutin e pointer segítségével ír be egy új bejegyzést a bufferbe)
001EH - 003DH	(16 szó)	klaviatúrabuffer; bár fizikai hossza tizenhat szó, sajátos kezelése miatt mégis csak tizenöt billentyűleütés nyomának befogadására alkalmas
		diskette-leíró terület
003EH	(1 byte)	alsó négy bitje jelzi, hogy a megfelelő lemezegységek igényelnek-e előkészítést (0.bit-A:, 1.bit-B:, stb.)

003FH	(1 byte)	alsó négy bitje a megfelelő lemezegységek motorjának állapotát jelzi (egyes érték utal a bekapcsolt motorra)
0040H	(1 byte)	időmérő számláló a motor leállításához
0041H	(1 byte)	státuszbyte, az utolsó lemezművelet eredményére utal (INT 13H-val közvetlenül lekérdezhető)
0042H - 0048H	(7 byte)	státuszbyte-ok a diskette-vezérlő számára
		Képernyővezérlő adatterülete
0049H	(1 byte)	Pillanatnyi képernyő-üzemmód (monokróm adapter esetén értéke 7)
004AH - 004BH	(1 szó)	oszlopok száma a képernyőn
004CH - 004DH	(1 szó)	képernyőterület hossza; 1000H színes és 400H monokróm adapter esetén
004EH - 004FH	(1 szó)	aktív lap kezdete a képernyőterületen belül (0. lap esetén 0000H)
0050H - 005FH	(8 szó)	az egyes képernyőlapok cursorainak pozíciója (összesen 8 lap számára)
0060H - 0061H	(1 szó)	a cursor alakja (kezdő- és végsora az elemi pozíción belül)
0062H	(1 byte)	az aktív lap sorszáma
0063H - 0064H	(1 szó)	az adapter bázisportja
0065H	(1 bte)	az éppen érvényben levő módparancs
0066H	(1 byte)	az érvényben levő színpaletta
0067H - 006BH	(5 byte)	fenntartott terület
		Csak IBM PC !
		Kazettás magnódriver adatterülete
		Timer adatterület
006CH - 006FH	(2 szó)	a gép software-órája, a 08. interrupt által kezelt négybyte-os számláló
0070H	(1 byte)	az éjfél elmúltát jelző indikátor
0071H	(1 byte)	a BREAK billentyű felengedését jelző indikátor
0072H - 0073H	(1 szó)	rendszertöltés módját megszabó változó Ennek szokásos értéke 1234H, ami a rendszer újratöltését jelenti, de a bekapcsoláskor végzett ellenőrzések nélkül. Egyéb értékek is megadhatók, ezek azonban csak egyes ROM BIOS verziókban élnek.

		Winchester adatterület
0074H - 0077H	(4 byte)	Belső felépítése nem dokumentált
		Nyomtató adatterület
0078H - 007BH	(4 byte)	A beépíthető négy nyomtató (LPT1-től LPT4-ig) Time Out változói
		Aszinkron adapter adatterület
007CH - 007FH	(4 byte)	A beépíthető négy aszinkron vonali adapter (COM1-től COM4-ig) Time Out változói
		Klaviatúra kiegészítő adatterület
0080H - 0081H	(1 szó)	A klaviatúra bufferének kezdőcíme
0082H - 0083H	(1 szó)	A klaviatúra bufferének végcíme

A ROM BIOS egyetlen egy byte-ot használ az MS-DOS 50H paragrafuscímen kezdődő szegmensében: ez a "STATUS_BYTE" nevű változó. Ne feledjük, hogy ez rendkívül fontos közös pont az alacsonyabb szintű ROM BIOS és az azt felülről használó MS-DOS között; mindkettejüknek tisztában kell lenniük azzal, hogy a nyomtató foglalt vagy szabad.

		Hard Copy státuszbyte
0050:0000	(1 byte)	A hard copy rutin aktivitását jelző indikátor; értéke 0, ha a hard copy rutin nem aktív, 1, ha igen. Ha a hard copy sikertelen a nyomtató hibája miatt, akkor ebben a byte-ban FFH-t találunk.

A továbbiakban ismertett területek csak újabb ROM BIOS verziókban használtak. Azért választottuk el őket a szokásos adatterülettől, hogy a "STATUS_FLAG" változót a szabványos egyéb változókkal együtt adjuk meg. Tehát az alább offsetjükkel felsorolt változók szegmenscíme ismét 0040H.

		Másodlagos video adatterület
0084H - 008AH	(7 byte)	Belső felépítése nem dokumentált.
		Floppy- és Winchester adatterület
008BH - 0095H	(11 byte)	Belső felépítése nem dokumentált (**Csak IBM AT !**)
		Klaviatúra adatterület
0096H	(1 byte)	Klaviatúra státuszflag (**Csak IBM AT !**)
0097H	(1 byte)	Klaviatúra világító diódáinak státusza (**Csak IBM AT !**)

		Várakozási változók (**Csak IBM AT !**)
		A 15H interrupt várakozási rutinjának változói
0098H - 009BH	(2 szó)	Hosszú pointer (alacsonyabb címen az offset-, magasabb címen a szegmens-érték) a várakozási flagre
009CH - 009FH	(2 szó)	a várakozási idő ezredmásodpercekben (alacsonyabb címen az alacsony, magasabb címen a magas helyiérték)
00A0H	(1 byte)	Várakozási indikátor; bitjeinek jelentése: 7 várakozási idő letelt 6-1 fenntartott 0 várakozási folyamat aktív
		Egyéb adatterületek
00A1H - 00A7H	(7 byte)	Fenntartott terület
00A8H - 00ABH	(2 szó)	Módosított video paramétertábla címe
00ACH - 00EFH	(68 byte)	Fenntartott terület
00F0H - 00FFH	(16 byte)	Felhasználói programok számára fenntartott kommunikációs terület

Befejezésül, kissé túllépve a ROM BIOS határait, megadjuk a 0050H címen kezdődő MS-DOS adatterület néhány olyan változóját, amely hasznos lehet a programozó számára (s amelyhez hozzá lehetett férni...). Tehát az alábbiakban offsetjükkel megadott változók szegmenscíme 0050H.

0000H	(1 byte)	Hard copy indikátor (lásd feljebb)
0004H	(1 byte)	Egydrive-os üzemmódot jelző byte
0010H - 0021H	(18 byte)	BASIC adatterület
0022H - 002FH	(14 byte)	Lemezformázáshoz használt MS-DOS változók
0030H - 0033H	(2 szó)	A MODE.COM program által használt változók
0034H - 00FFH		Az MS-DOS használatára fenntartva

B függelék: A ROM-BIOS interruptok kiosztása

Processzor-interruptok

00	Osztási túlcsordulás, zerodivide	
01	Lépésenkénti megszakítás, single-step	
02	NMI, nem maszkolható interrupt	(lásd 287. oldal)
03	Töréspont, breakpoint	
04	Túlcsordulási rutin hívása, overflow	

Alap BIOS interruptok

05	Képernyőnyomtatás (hard copy)	(lásd 288. oldal)
06, 07	Nem használt interruptok	

Hardware interruptok

08	Timer hardware interruptja ----------	(8259 IRQ0)
09	A klaviatúra hardware interruptja ----------	(8259 IRQ1)
0A	Nem használt ----------	(8259 IRQ2)
0B	Második aszinkron vonal hardware IT. ----------	(8259 IRQ3)
0C	Első aszinkron vonal hardware IT. ----------	(8259 IRQ4)
0D	Merevlemez interruptja ----------	(8259 IRQ5)
0E	A diskette hardware interruptja ----------	(8259 IRQ6)
0F	A nyomtató hardware interruptja ----------	(8259 IRQ7)

BIOS kommunikációs interruptok

10	Képernyő-driver hívása	(lásd 110. oldal)
11	Gép elemeinek lekérdezése	(lásd 289. oldal)
12	Memóriahossz ellenőrzése	(lásd 290. oldal)
13	Floppy-disk driver hívása	(lásd 210. oldal)
14	Aszinkron vonali driver hívása	(lásd 161. oldal)
15	Kazetta-driver hívása	(lásd 290. oldal)
16	Klaviatúra-driver hívása	(lásd 76. oldal)
17	Nyomtató-driver hívása	(lásd 142. oldal)
18	ROM BASIC belépési pont	(lásd 300. oldal)
19	Rendszerindítás, bootstrap	(lásd 300. oldal)
1A	A belső óra kezelése	(lásd 301. oldal)
1B	CTRL-BREAK felhasználói rutin	(lásd 305. oldal)
1C	Timer felhasználói rutin	(lásd 305. oldal)

BIOS paramétertáblázatok címei

1D	Video paramétertábla	(lásd 306. oldal)
1E	Diskette paramétertábla	(lásd 308. oldal)
1F	Grafikus karaktergenerátor	(lásd 310. oldal)

MS-DOS interruptok

20 - 3F Az MS-DOS számára fenntartva (lásd 2. kötet: A programozó és az MS-DOS)
60 - 67 Felhasználói interruptok (az Overlay Linker overlay ágak betöltésére használja fel)

XT és AT interruptok

40 **Csak IBM XT és AT !** Az eredeti 13H interrupt vektor, tehát az eredeti floppy-kezelő rutincsomag címe
41 **Csak IBM XT és AT !** Winchester paramétertábla címe
42 EGA kártya használata esetén az eredeti video paramétertábla címe (az eredeti 1EH vektor tartalma)
43 EGA kártya használata esetén az inicializálási paramétertábla címe
44 EGA kártya használata esetén az grafikus karaktergenerátor címe (szerepe az 1FH vektoréval egyezik meg)
45 fenntartott interrupt
46 **Csak IBM AT !** A második Winchester paramétertáblázatának címe
47 - 49 fenntartott interruptok
4A **Csak IBM AT !** A riasztáskor aktivizálandó felhasználói rutin címe
4A - 59 fenntartott interruptok

IBM AT hardware interruptok

70 A valósidejű óra hardware interrupt ------ (8259 IRQ8)
71 LAN adapter hardware interrupt ---------- (8259 IRQ9)
72 Fenntartott ---------- (8259 IRQ10)
73 Fenntartott ---------- (8259 IRQ11)
74 Fenntartott ---------- (8259 IRQ12)
75 80287 aritmetikai koprocesszor ---------- (8259 IRQ13)
76 Második Winchester hardware interrupt ---- (8259 IRQ14)
77 Fenntartott ---------- (8259 IRQ15)

Hálózati interrupt

86 A hálózati funkciókat ellátó BIOS funkciók kommunikációs interruptja

C függelék: Az I/O portok kiosztása

Az alábbi táblázat tartalmazza az I/O portok kiosztását. A lista a lehetőségekhez mérten teljes; a rendelkezésre álló, elég szegényes irodalomból ennyit sikerült kihámozni.

A táblázat első fele a direkt címzéssel is elérhető portokat sorolja fel, a második rész a 255-nél (FFH) magasabb című portokat tartalmazza.

Az IBM XT portjai némileg különböznek a PC portoktól; van néhány olyan port, amely a PC-ben még egyáltalán nem, vagy pedig fenntartott portként szerepel. Ezeket (*)-al jelöljük.

Hexa sorszámok	Hozzárendelt perifériavezérlő	
00 - 0F	8237 DMA vezérlő	
20 - 21	8259 interrupt vezérlő	
40 - 43	8253 Timer/Counter	
60 - 63	8255 párhuzamos progr. interface	
80 - 83	DMA vezérlő lapregiszterei	
A0	NMI maszkregiszter	
A1 - AF	Fenntartott I/O portok	
C0 - CF	Fenttartott I/O portok	
E0 - EF	Fenttartott I/O portok	
200 - 20F	Joystick-(botkormány-) adapter	
210 - 217	Memóriakiterjesztési egység portjai	(*)
220 - 24F	Fenntartott I/O portok	(*)
278 - 27F	Fenntartott I/O portok	
2F0 - 2F7	Fenntartott I/O portok	(*)
2F8 - 2FF	Második aszinkron vonali adapter	(*)
300 - 31F	Prototípuskártya	(*)
320 - 32F	Winchester vezérlő portjai	(*)
378 - 37F	Különálló párhuzamos nyomtatóvezérlő	
380 - 38F	SDLC kommunikációs adapter	(*)
3A0 - 3AF	Fenntartott I/O portok	(*)
3B0 - 3BF	Fekete-fehér monitor- és párhuzamos nyomtatóvezérlő	
3C0 - 3CF	Fenntartott I/O portok	(*)
3D0 - 3DF	Színes monitorvezérlő	
3E0 - 3E7	Fenntartott I/O portok	(*)
3F0 - 3F7	Diskette-vezérlő	
3F8 - 3FF	Első aszinkron vonali adapter	
400 - FFFF	Nem használható	

D függelék: A klaviatúra kódjai

Az alábbi táblázat a PC klaviatúrát szemléleteti a billentyűkódok feltüntetésével. A kódtáblázatban megadtuk a shift billentyűkkel kombinált funkcióbillentyűk kódjait.

A klaviatúra központi része

ESC 01	1 02	2 03	3 04	4 05	5 06	6 07	7 08	8 09	9 10	0 11	- 12	= 13	<--- 14
TAB 15	Q 16	W 17	E 18	R 19	T 20	Y 21	U 22	I 23	O 24	P 25	[26	] 27	EN-TER 28
CTRL 29	A 30	S 31	D 32	F 33	G 34	H 35	J 36	K 37	L 38	; 39	' 40	` 41	
SH-L 42	\ 43	Z 44	X 45	C 46	V 47	B 48	N 49	M 50	, 51	. 52	/ 53	SH-R 54	* 55
ALTMODE 56	S P A C E B A R 57											CAPS LOCK 58	

ahol "<---" a visszatörlés, TAB a tabulátor, CTRL a kontrol-, SH-L és SH-R a bal, illetve jobb oldali SHIFT billentyű.

Funkcióbillentyűk

F1 59	F2 60
F3 61	F4 62
F5 63	F6 64
F7 65	F7 66
F9 67	F10 68

Numerikus billentyűzet

NUM LOCK 69		SCR.LOCK 70	
HOME 71	^ 72	PGUP 73	- 74
<- 75	5 76	-> 77	+
END 79	v 80	PGDN 81	78
INS 82		DEL 83	

ahol F1,..., F10 a funkcióbillentyűk, SCR.LOCK a SCROLL LOCK.

Megjegyezzük még, hogy a a klaviatúra hardware-interrupt rutinja a SHIFT, CONTROL vagy ALT billentyűkkel együtt lenyomott funkcióbillentyűk hatására más billentyűkódokat ír be a klaviatúra-bufferbe. Ez igen hatékonnyá teheti a funkcióbillentyűk felhasználását. Az eltérő billentyűkódok listája:

SHIFT F1 84
SHIFT F2 85
SHIFT F3 86
SHIFT F4 87
SHIFT F5 88
SHIFT F6 89
SHIFT F7 90
SHIFT F8 91
SHIFT F9 92
SHIFT F10 93

CTRL F1 94
CTRL F2 95
CTRL F3 96
CTRL F4 97
CTRL F5 98
CTRL F6 99
CTRL F7 100
CTRL F8 101
CTRL F9 102
CTRL F10 103

ALT F1 104
ALT F2 105
ALT F3 106
ALT F4 107
ALT F5 108
ALT F6 109
ALT F7 110
ALT F8 111
ALT F9 112
ALT F10 113

CTRL PRTSC 114
CTRL LEFT 115
CTRL RIGHT 116
CTRL END 117
CTRL PG DN 118
CTRL PG UP ..... (!!) ... 132
CTRL HOME 119

E függelék: A ROM BIOS funkciók összefoglalása

Az alábbi lista a felhasználó által aktivizálható ROM-BIOS interrupt rutinok összes funkcióját tartalmazza, az interruptok és a funkciókódok emelkedő sorrendjében. A feltüntetett adatok:

i.szám f.kód angol elnevezés
magyar elnevezés ..... a leírás oldalszáma

240,– Ft