INOTAI LÁSZLÓ
LÁZÁR LÁSZLÓ

IBM PC XT/AT rendszerprogramozás I.

Alapozás és a DOS

Novotrade Kiadó

INOTAI LÁSZLÓ
LÁZÁR LÁSZLÓ

IBM PC XT/AT rendszerprogramozás

I.

Alapozás és a DOS

Novotrade Kiadó

Írta : Inotai László
Lázár László

Lektorálta : Ila László

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

Tipográfia és fedélterv: Erdősi Zoltán, fotó: Ács László

Szedte az Anteus Kft.

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

ISBN 963 585 109 X (I., II., III. kötet)
ISBN 963 585 110 3 (I. kötet)

TARTALOMJEGYZÉK

Mottó: Mondottam, ember:
küzdj, és bízva bízzál!

(Madách Imre: Az ember tragédiája)

ELŐSZÓ

Ezt a könyvet elsősorban azoknak ajánljuk, akik már némi jártasságra tettek szert a számítástechnika valamely területén, és most egy kicsit jobban meg akarják ismerni a számítógépek működésének belső mechanizmusát. Ez a mechanizmus a számítógépek „lelkivilága", amely a felhasználók elől általában elrejtőzik, és úgy viselkedik, mintha tulajdonképpen nem is lenne. Maga a mechanizmus természetesen szolgál, mint egy gép – elvégre ez a feladata!

A rejtett titkait viszont csak azoknak tárja fel, akik vele egy nyelven beszélnek. Azoknak viszont, akik készek arra, hogy ezt a nyelvet megtanulják, minden kívánságát készséggel teljesíti.

Érdemes tanulni…

BEVEZETŐ

A könyv az IBM PC gépcsalád rendszerprogramozását ismerteti. Rövid áttekintést ad a számítógépcsalád hardverfelépítéséről, az ezzel kapcsolatos legfontosabb tudnivalókról, majd részletesen foglalkozik a számítógép működését alapvetően meghatározó, és a rendszerprogramozók számára nélkülözhetetlen DOS és BIOS rendszerprogramokkal.

Részletesen tárgyalja a számítógép és a legfontosabb külső egységek (billentyűzet, képernyő, hajlékony- és merevlemezes meghajtó, nyomtató, soros interfész stb.) közötti adatforgalom szervezését és lebonyolítását.

Úgy gondoljuk, hogy a BIOS és a DOS megszakítások egységes rendszerbe foglalása hasznos segítséget jelent a rendszerprogramozók számára (itt alapvetően az eredeti Intel és MS-DOS dokumentációkra támaszkodtunk).

A legfontosabb megszakításokhoz illusztrációként mintaprogramokat készítettünk. Reméljük, hogy a példák különböző programozási nyelveken megírt változatai közül minden olvasó ki tudja választani az ízlésének megfelelőt (a többi pedig talán további ösztönzést is adhat).

Feltételezzük, hogy az olvasó valamilyen számítógépes alapismerettel rendelkezik, és az assembly, BASIC, PASCAL vagy a C nyelvek valamelyikén már írt valamilyen – saját használatra alkalmas – programot. Ezért nem foglalkozunk az olyan alapvető ismeretekkel, mint egy számítógépes rendszer összeállítása, a rendszer indítása, az alapvető DOS parancsok funkciója vagy a RAM és a ROM közötti különbség. Azt is feltételezzük, hogy az olvasó eligazodik a kettes és a tizenhatos számrendszerekben (segítséget azért adunk).

Azok viszont, akik a fenti alapismeretek birtokában vannak, reményeink szerint haszonnal forgathatják ezt a könyvet.

Ehhez kívánunk eredményes tanulást, és sok sikert.

Budapest, 1990. január

Inotai László
Lázár László

I. rész

Alapozás

1. A PC „AGYA"

Bizonyára sokan elgondolkodtak már azok, akik – akár laikusként, akár felhasználóként – valamilyen kapcsolatba kerültek egy számítógéppel azon, hogy milyen bámulatosan „okos" is ez a gép. Játszi könnyedséggel számol ki bonyolult képleteket, méghozzá gyorsan és pontosan, ha kell, szöveget szerkeszt vagy táblázatot kezel, esetleg gyönyörű, színes ábrát varázsol a képernyőre, és mennyi „esze" van, hogy mindenre olyan jól emlékezik! Mindez – és még sok más – igaz is, meg nem is. Igaz, hiszen ezeket a gyakorlatban tapasztaljuk, másrészt viszont – és ezt már jóval kevesebben tudják – a számítógép bizonyos értelemben rendkívül primitív eszköz. Az értelmi képességei csak arra terjednek ki, hogy két, egymástól eltérő dolgot vagy állapotot meg tudjon különböztetni: pl. az alacsonyat és a magasat, vagy történetesen a 0-t és az 1-et. Igazából az ember az okos, aki ezeket a 0-kat és 1-eket olyan sorrendben, kombinációban és mennyiségben képes a számítógép értésére adni, hogy az ezeket a 0-kat és 1-eket – ugyancsak az ember által meghatározott módon feldolgozva – a kívánt, és valóban bámulatraméltó eredményt szolgáltassa.

Röviden vegyük szemügyre a számítógépnek azt az alkotó elemét, amely ezeknek a 0-knak és 1-eknek a kezelésében a legfontosabb szerepet játssza, és amely minden számítógép nélkülözhetetlen része. Ez az alkotó elem a mikroprocesszor, vagy más néven a központi feldolgozó egység. A szakirodalom ezt általában csak CPU-ként említi (az eredeti, angol Central Processing Unit rövidítéseként); a rövidség és az egyértelműség miatt a továbbiakban mi is ezt használjuk. A számítógépen belül a mikroprocesszor az egyetlen olyan alkatrész, amely bizonyos matematikai és logikai műveletek elvégzésére képes: csak ez az egység tud két számot összeadni vagy összeszorozni, két számot összehasonlítani, feltételeket vizsgálni és ezektől függően döntéseket hozni stb., így ez tekinthető a számítógép „agyának".

Az IBM a PC számítógépcsaládjának különböző típusaiba az Intel Corporation amerikai mikroelektronikai vállalat által tervezett és gyártott mikroprocesszorokat építi be. Maguk az Intel mikroprocesszorok is családot alkotnak. E család tagjai a 16 bites, harmadik generációs mikroprocesszoroktól felfelé, a típusjelölésük szerint:

– 8086-os (megjelent 1978-ban),
– 8088-as (megjelent 1979-ben), erre épül az IBM PC és a PC/XT gép,

- 80186/88-as (megjelent 1982-ben),
- 80286-os (megjelent 1984-ben), erre épül az IBM PC/AT,
- 80386-os (megjelent 1986-ban).

A felsorolt processzorok egymással alulról felfelé kompatibilisek, ami azt jelenti, hogy a sorban következő processzor az őt megelőző processzor valamennyi utasítását képes megérteni, és végrehajtani. Fordítva ez nem igaz: egy 8086-os processzor olyan programot, ill. utasítást, ami csak a 80286-os vagy a 80386-os utasításkészletében szerepel, nem tud végrehajtani.

A 8086-os és a 8088-as processzor szoftveroldalról egymással teljesen kompatibilis, utasításkészletük is teljesen azonos. A kettő közötti különbség csak az adatbusz szélességében van: míg a 8088-as adatbusza csak 8 bites, a 8086-osé már 16 bites. Belülről nézve azonban mindkét processzor 16 bit széles csatornán keresztül bonyolítja le az adatforgalmat. Mindkét processzor címbusza 20 bit szélességű, így a címezhető tárterület nagysága 2^{20} bájt, azaz 1 Mbájt.

A 80286-os mikroprocesszor a CPU-k újabb generációját jelentette. A megnövekedett feladatokhoz kibővítették a processzor belső regisztereit, bővült az utasításkészlet, a 24 címvezetékkel és az ún. virtuális tárkezeléses módszerrel 1 Gbájt címezhető tárterületet 16 Mbájt nagyságú fizikai tárra tud leképezni. Ez a virtuális tárkezelés, valamint a védelmi rendszer bevezetése jelenti az előző generációhoz képest a legjelentősebb újítást: a 80286-os processzorra épülő gépek – megfelelő operációs rendszer segítségével – lehetővé tesznek többfelhasználós, többfeladatos (multiuser, multitasking) alkalmazásokat.

A processzorcsalád legújabb tagja a 80386-os processzor, melynek külső adatbusza 32 bites, a címezhető fizikai tárterület pedig 4 Gbájt. Ez a processzor már az IBM új, PS/2 (Personal System) számítógépcsalád nagyobb teljesítményű tagjainak az alapját képezi.

Az egyértelműség kedvéért a továbbiakban mikroprocesszoron mindig az IBM PC/XT-ben lévő 8088-as processzort értjük.

1.1. A 8088-as regiszterei

A következőkben megvizsgáljuk a processzor regisztereit, és megismerkedünk fő feladataikkal. A regiszter lényegében nem más, mint egy olyan (esetünkben 8 vagy 16 bites) tárhely, amely nem a külső munkatárban, hanem magában a processzorban helyezkedik el. Ez azt jelenti, hogy a processzor a regiszterekben lévő adatokat lényegesen gyorsabban tudja elérni, mint a külső (munka)tárban lévőket, mivel a regiszterekből való olvasáskor nem kell külön gépi ciklusokkal a tárhoz fordulnia.

A processzor mindegyik regisztere 16 bites, azaz mindegyik regiszter 2 bájtnyi információt tárolhat. A regiszter tartalmának akkor a legkisebb az értéke, ha a benne lévő – 0-tól 15-ig számozott – mindegyik bit értéke 0. Ez a legkisebb érték természetesen

mind a bináris, mind a decimális számrendszerben 0. A regiszter akkor veszi fel a legnagyobb értéket, ha a 16 bit mindegyikének értéke 1. Ez a szám a bináris számrendszerben az 1111111111111111, hexadecimálisan FFFF(h), decimális számrendszerben pedig 65535. Eszerint egy 16 bites regiszter a 0...65535 közötti bármelyik decimális értéket felveheti, ami 65536 különböző állapot ábrázolását teszi lehetővé (64 kbájt).

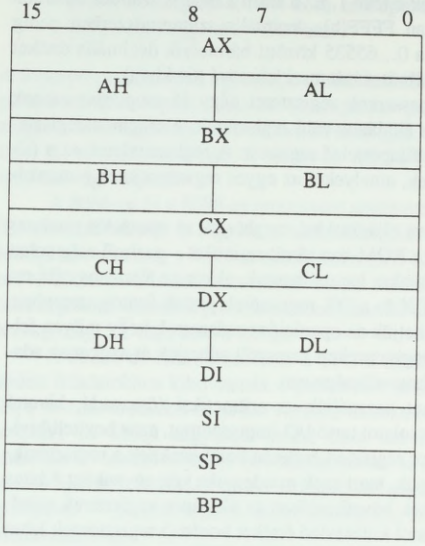
Amint az 1. ábrán látjuk, a processzorok regiszterei négy fő csoportba vannak osztva. Egy-egy csoportot képeznek az általános célú regiszterek, a szegmensregiszterek, valamint az utasításszámláló és az állapotjelző regiszter. A regisztereknek ez a felosztása megfelel azoknak a feladatoknak, amelyeket az egyes regisztereknek a munkájuk során el kell végezniük.

Amikor a későbbiekben bemutatott eljárásokkal meghívjuk az operációs rendszer (DOS) különböző megszakításait vagy a ROM-ban tárolt rutinokat – ezekről még sokat olvashatunk –, hogy azokat a saját céljainkra használhassuk, akkor az általános célú regiszterek, mindenekelőtt az AX, BX, CX és a DX regiszterek jutnak fontos szerephez. Ezeken a regisztereken keresztül közölhetjük az operációs rendszerrel, hogy milyen feladatokat kell elvégeznie, ill. ezeken a regisztereken keresztül adhatjuk át azokat az adatokat, amelyekre a feladat végrehajtásához szüksége van.

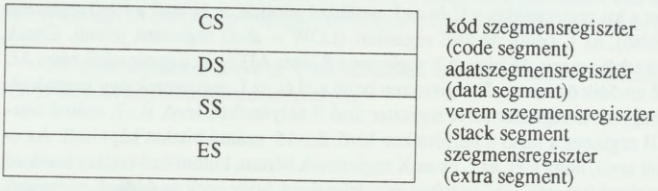
Ugyancsak ezeket a regisztereket használják az aritmetikai (összeadó, kivonó stb.), valamint a PC többi részével kapcsolatot tartó I/O (input/output, azaz beviteli/kiviteli) utasítások is. A 8088-as processzor regiszterkészletén belül ezeknek a regisztereknek azért van megkülönböztetett szerepük, mert ezek mindegyike két, egyenként 8 bites (1 bájtos) regiszterre osztható fel. Ennek következtében az általános regiszterek mindegyikét tulajdonképpen három, egymástól különböző értéket hordozó regiszternek lehet tekinteni: egyrészt egyetlen, „nagy", 16 bites regiszternek, másrészt pedig két, egyenként 8 bites, „kis" vagy fél regiszternek.

Ezeket a kis regisztereket a H és az L betűkkel jelöljük. A H betű a felső regisztert (HIGH = felső), az L pedig az alsó regisztert (LOW = alsó) regisztert jelenti. Ennek megfelelően tehát a nagy, 16 bites AX regiszter a 8 bites AH és az ugyancsak 8 bites AL regiszterből tevődik össze. Az X regiszteren belül a H és az L regiszterek úgy vannak elrendezve, hogy az L regiszter az X regiszter alsó 8 helyértékén levő, 0...7. számú biteket, míg a H regiszter a felső 8 helyértéken lévő, 8...15. számú biteket képviseli. Az előbb szó volt arról, hogy a H, az L és az X regiszterek három, különböző értéket hordozó regisztereknek tekinthetők, ami azonban nem jelenti azt, hogy ezek az értékek egymástól függetlenek. Nyilvánvaló, hogy akár az L, akár a H regiszterben lévő érték megváltoztatásával az X regiszter értéke is megváltozik. Fordítva ez azzal a megszorítással igaz, hogy az X regiszter értékének megváltoztatásakor attól függően változik a H vagy az L, vagy mindkettő értéke, hogy az X regiszter mely bitjének vagy bitjeinek az értéke változott. Például, ha az X regiszterben csak a 4. bit értéke változik (ez az L regiszternek szintén a 4. bitje), akkor a H értéke változatlan marad. Ha viszont az X regiszterben nemcsak a 4. bit, hanem mondjuk a 10. bit értéke is változik (ez a H regiszter 2. bitjének felel meg), akkor egyúttal az L és a H értéke is változik.

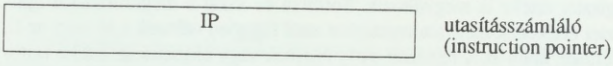
ÁLTALÁNOS CÉLÚ REGISZTEREK



SZEGMENSREGISZTEREK



UTASÍTÁSSZÁMLÁLÓ



ÁLLAPOTJELZŐ (FLAG) REGISZTER

—	—	—	—	O	D	I	T	S	Z	—	A	—	P	—	C
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

1. ábra: A 8088-as mikroprocesszor regiszterkészlete

Az L, H és az X regiszterek értékei közötti összefüggések matematikailag is megfogalmazhatók. Az L és a H regiszterek értékeinek ismeretében úgy számítható ki az X regiszter értéke, hogy a H regiszter értékét megszorozzuk 256-tal (ezzel tulajdonképpen az X regiszteren belül a „helyére" tesszük), majd hozzáadjuk az L regiszter értékét (ez eleve a „helyén" van). Képletben:

$$X = H * 256 + L$$

A számítás fordított irányban is elvégezhető: X értékének ismeretében kiszámítható az L és a H értéke. Ez a számítás egy kicsit bonyolultabb: a H értékét úgy kapjuk meg, hogy az X értékét elosztjuk 256-tal, és a hányados egész (integer) részét vesszük. Az L úgy számítható, hogy a H-ra előbb kapott értéket megszorozzuk 256-tal, és ezt kivonjuk az X-ből. A különbség az L regiszter értéke. Képletben:

$$H = \text{int}\ (\ X / 256\ )$$
$$L = X - H * 256$$

Ezeket az átszámításokat a későbbiekben még sokszor használjuk, sőt nemcsak két 8 bites értékkel, hanem esetenként négy, egyenként 8 bites, vagy két, egyenként 16 bites (azaz 2 bájtos) értékekkel kapcsolatban is. Az átszámítások ugyanígy végezhetők el ez utóbbiaknál is, csak a megfelelő helyértékek változnak.

Térjünk vissza a processzor regisztereihez. Az, hogy az általános célú regiszterek közül az AX, BX, CX és a DX regiszterek alsó és felső regiszterekre oszthatók, lehetővé teszi, hogy szükség esetén csak az egyikbe írjunk vagy olvassunk értéket, ill. a tár és csak az egyik (fél)regiszter között mozgassunk adatokat. Ennek egyrészt az az előnye, hogy a teljes regiszter másik fele nem változik meg, másrészt pedig az, hogy a processzornak ilyenkor csak egyetlen, ún. gépi ciklussal kell a tárhoz fordulnia, ami a feldolgozási sebességet jelentősen megnöveli (egyébként ez az a legkisebb információmennyiség, amit a processzor egy ciklusban kezelni tud, ezért mondják azt, hogy ez a processzor „bájt szervezésű").

1.2. Címszámítás szegmens- és ofszetcímekkel

Ahhoz, hogy a következő fontos csoportba sorolt regisztereknek, a szegmensregisztereknek a feladatát és jelentőségét megérthessük, röviden fel kell idéznünk a 8088-as processzor keletkezésének történetét.

A fejlesztéskor az volt a cél, hogy olyan mikroprocesszort fejlesszenek ki, amelynek a kor- és versenytárs 8 bites processzorokhoz (6502-es, Z80-as stb.) képest kedvezőbb az utasításkészlete, és amellyel az akkori, mágikus 64 kbájtos tárkapacitáson is túl lehet lépni. Ez utóbbi célnak különösen nagy volt a jelentősége, mert a processzorok egyre növekvő teljesítménye egyre bonyolultabb alkalmazásokat tett lehetővé, amelyek

viszont egyre nagyobb és nagyobb tárméretet igényeltek. A 8088-as processzor tervezői azt a nagyratörő tervet tűzték maguk elé, hogy az új processzor címtartományát – vagyis azoknak a tárhelyeknek a számát, amelyhez a processzor a címvezetékein hozzá tud férni – a kortárs processzorokéhoz képest nemcsak megkétszerezik, hanem 1 Mbájtra bővítve, a 16-szorosára növelik.

Ehhez tudni kell azt, hogy azoknak a tárhelyeknek vagy más szóval rekeszeknek a száma, amelyekhez a processzor hozzá tud férni, egy speciális, címregiszternek nevezett regiszter szélességétől függ. Mivel mindegyik tárhelynek saját azonosító címe van, amelyen keresztül az illető tárhelyhez lehet fordulni, és az egyes tárhelyek 0-tól kezdve folyamatosan vannak sorszámozva (ezeket a sorszámokat nevezzük címeknek), a címregiszterben ábrázolható legnagyobb szám határozza meg azt, hogy a processzor hány különböző tárhelyhez tud fordulni, azaz hány tárhelyet képes megcímezni.

A 8088-as elődeinél a címregiszter 16 bit széles volt. Egy ilyen regiszter 0 és 65535 ($0 - 2^{16}$) közötti értéket vehet fel, ezen keresztül tehát összesen 65536 tárhely címezhető meg (0. sorszámú tárhely is van!). Ez összesen 64 kbájt tárkapacitást jelent. Ebből már világosan látszik, hogy a 64 kbájt túllépéséhez a címregiszternek 16 bitnél szélesebbnek kell lennie. Egészen pontosan 20 bites címregiszterre van szükség ahhoz, hogy a processzor 1 Mbájtnyi tárterületet meg tudjon címezni. Joggal felmerülhet a kérdés, hogy mi ebben a rendkívüli, hiszen kézenfekvő a megoldás: a processzorba be kell építeni egy 20 bites címregisztert, és kész. Figyelembe kell azonban azt venni, hogy a technika akkori szintjén egy ilyen bonyolultságú processzort még nem tudtak megvalósítani. A 16 bitből csakis egy trükk segítségével lehetett 20 bitet előállítani: ez a trükk az ún. szegmentálás.

A 8088-asban már nincs meg a régi címregiszter, hanem az egyes tárhelyek megcímzéséhez a processzor a címet két, közönséges, 16 bites regiszterrel állítja elő úgy, hogy a két regiszterben lévő értékeket egy 20 biten ábrázolható értékké, azaz egy 20 bites címmé kapcsolja össze. Így lesz a 16 bites regiszterből (igaz, hogy ebből kettő kell) egy 20 bites cím.

A címszámításhoz a processzor az egyik értéket minden esetben a 4 szegmensregiszter egyikéből veszi elő. A másik érték a processzor valamelyik másik regiszterében, vagy a tár valamelyik rekeszében lehet. A két, 16 bites értéket a processzor azonban nem a szokásos, egyszerű módon adja össze (hiszen így maximum csak egy 17 bites érték jöhet létre, ha túlcsordulás van), hanem az összeadás előtt az elsőként vett értéket, tehát a szegmensregiszter tartalmát először 4 bittel balra tolja. Ez decimális számrendszerben az érték 16-tal való megszorzását jelenti. Hexadecimális számrendszerben ez a szorzás úgy ábrázolható, hogy a négy hexadecimális számjegyhez jobbról hozzáírunk egy 0-t, míg bináris számrendszerben a 16 bináris számjegyhez négy darab 0 értékű bitet. A szegmenscím képzését a különböző számrendszerekben a 2. ábra szemlélteti. (Itt feltételeztük, hogy a regiszter a benne ábrázolható legnagyobb értéket tartalmazza, de természetesen bármilyen, ennél kisebb értéket is tartalmazhat.)

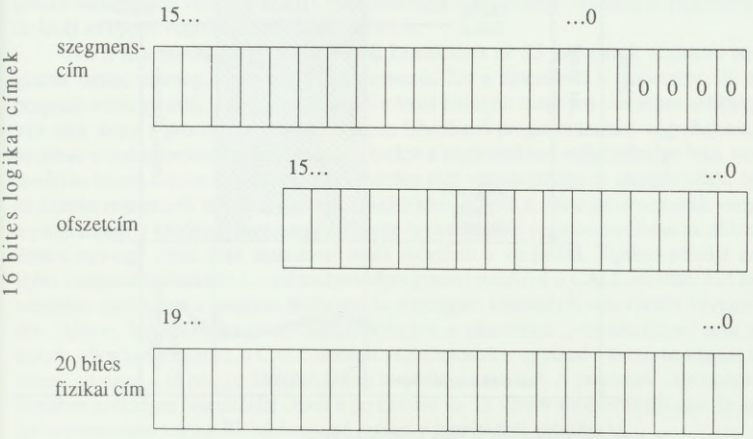
számrendszer	szegmensregiszter tartalma	szegmenscím (tartalom * 16)
decimális	65535	1048560
hexadecimális	FFFF	FFFF0
bináris	1111111111111111	11111111111111110000

2. ábra: Szegmenscím számítása

A processzor az így kiszámított szegmenscímhez hozzáadja a másik 16 bites regiszterből vagy rekeszből elővett értéket, és készen van a 20 bites tárcím.

Ezeket a címszámításhoz használt címeket szegmens- és ofszetcímeknek nevezik. Szegmenscímnek azt a címet nevezik, amelyiket a szegmensregiszterek valamelyike állít elő, és amely a tárterület valamely részének (az ún. szegmensnek) a kezdetét határozza meg. Az ofszetcím az a cím, amit a címszámításkor a szegmenscímhez hozzá kell adni. Az ofszetcím tehát annak a tárhelynek a fizikai címét adja meg, amelyet úgy kapunk meg, hogy a szegmens kezdőcíméhez hozzáadjuk az ofszetcímet. Mivel az ofszetcím 16 bitnél nem lehet szélesebb, egy szegmens mérete maximum 65535 bájt, azaz 64 kbájt lehet.

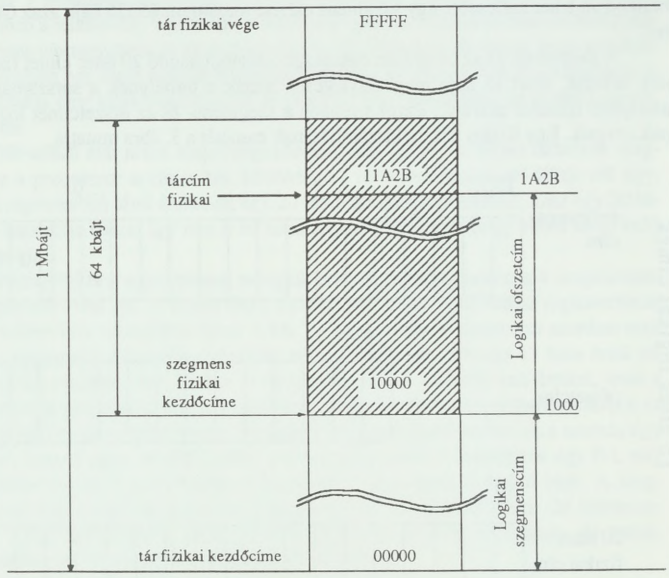
A szegmens- és az ofszetcím összekapcsolásából adódó 20 bites címet fizikai címnek nevezik, mert ez adja meg ténylegesen annak a tárhelynek a sorszámát (címét), amelyhez fordulni akarunk. Ezzel szemben a szegmens- és az ofszetcímet logikai címnek nevezik. Egy fizikai tárcím kiszámításának menetét a 3. ábra mutatja.



3. ábra: Egy fizikai tárcím kiszámítása szegmens- és ofszetcímből

Az előzőekből már nyilvánvaló lehet, hogy egy szegmens tárcíme nem kezdődhet a több mint 1 millió tárcím bármelyikén. Mivel egy szegmens kezdőcíme úgy jön létre, hogy a szegmensregiszter tartalmát 16-tal megszorozzuk, egy szegmens kezdőcíme csak egy 16-tal osztható szám lehet. Például a 34-es sorszámú tárcímen nem kezdődhet szegmens.

A szegmens- és az ofszetcímek ily módon történő összekapcsolásából kialakult egy jelölésmód, amellyel megadható egy tárhely fizikai címe. Először megadjuk a tárhely szegmenscímét, majd ezt követően, az előzőtől egy kettősponttal elválasztva az ofszetcímét, mindkettőt hexadecimális számrendszerben. Megállapodás szerint ennél a jelölésmódnál az értékeket mindig hexadecimális számrendszerben adjuk meg, ezért a számrendszer jelölésére egyébként használatos h jelet itt külön nem kell megadni. E megállapodás szerint ha annak a változónak a címét akarjuk megadni, amelynek a szegmenscíme 1000(h) és az ofszetcíme 1A2B(h), akkor ennek a jelölésére az 1000:1A2B jelölést használjuk. Az így jelölt tárcímnek a táron belüli, fizikai helyét a 4. ábrán szemléltetjük.



4. ábra: Egy szegmentált cím táron belüli fizikai helye

Megjegyezzük, hogy ugyanazt a fizikai tárcímet nem csak egyetlen szegmens-cím:ofszetcím kombinációval lehet megadni (pl. az 1000:1A2B és az 1100:0A2B egyaránt az 11A2B fizikai tárcímet jelenti).

Amint említettük, a 8088-asnak négy szegmensregisztere van, amelyek – a processzor más regisztereivel együttműködve – fontos szerepet játszanak a gépi kódú programok végrehajtásában. Az a tény, hogy nemcsak egy, hanem négy szegmensregiszter létezik, a programok szerkezetére vezethető vissza. Először is a program egy sor utasításból áll, amelyeket együttesen kódnak nevezünk. Emellett vannak azok az adatok és változók, amelyeket a program a futása során felhasznál, ill. feldolgoz. Ahhoz, hogy egy program jól áttekinthető felépítésű és rugalmasan változtatható legyen, továbbá, hogy egy program fejlesztésén egyidejűleg többen is dolgozhassanak, szükséges a kódnak és az adatoknak a tárban való szétválasztása. Ez úgy valósítható meg, hogy mindkettőhöz hozzá van rendelve egy-egy szegmens. Ebből az is következik, hogy mindegyik szegmenshez kell egy-egy szegmensregiszter. Nézzük meg közelebbről az egyes szegmensregiszterek feladatát.

A CS jelű (code segment) kód szegmensregiszter az ofszetcímként használt IP (instruction pointer) utasításmutató vagy más néven utasításszámláló (program counter) regiszterrel együtt azt a tárhelyet határozza meg, ahonnan a CPU-nak a soron következő gépi utasítást be kell olvasnia. Miután a processzor a beolvasott utasítást végrehajtotta, az IP regiszter tartalma automatikusan megnövekszik úgy, hogy most a következő utasításra mutasson. A processzor utasításról utasításra haladva, így hajtja végre a programot.

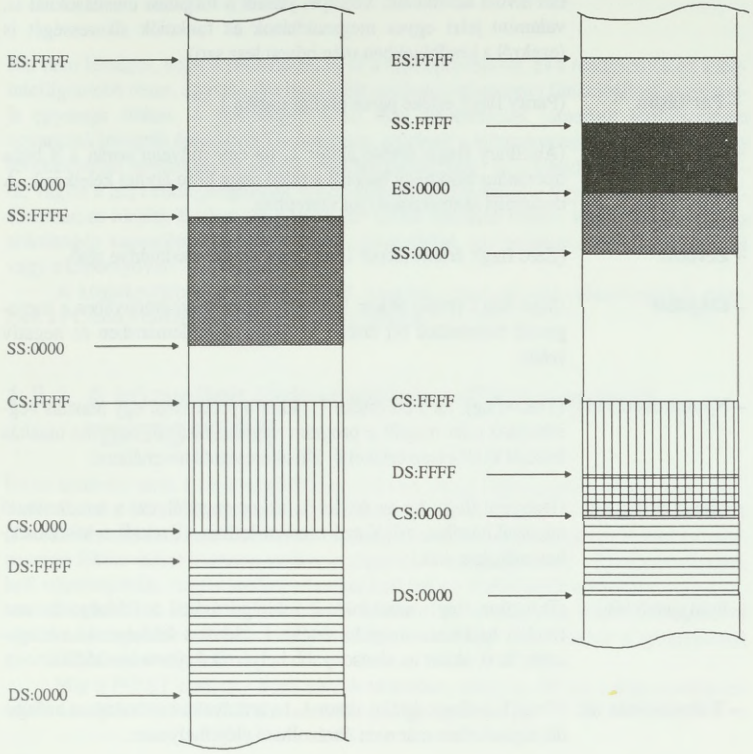
A CS regisztertől eltérően a DS (data segment) adat szegmensregisztert akkor használja a processzor, amikor a program adataihoz kell hozzáférnie, tehát amikor a tárból adatokat kell olvasnia, vagy a tárba adatokat kell írnia. Azt a címet, amelyet a DS regiszter tartalmához hozzá kell adni, tartalmazhatja a processzor valamelyik regisztere, de lehet az éppen végrehajtandó utasítás része is.

Azt a tárterületet, amelynek a tárbeli kezdőcímét az SS jelű (stack segment) regiszter határozza meg, veremnek (stack) nevezik. Ezt a tárterületet a processzor, ill. a program mintegy saját, belső memóriájaként lehet felfogni: bizonyos esetekben szükség van arra, hogy a processzor, mielőtt a soron következő programutasítást végrehajtaná, azoknak a regisztereknek a tartalmát, amelyekre a későbbiekben még szüksége lesz, valamilyen biztos helyen megőrizze. Ezt követően már végrehajthatja az utasítás(oka)t, és az érintett regiszterek tartalma ezeknek megfelelően változhat. Ha a processzornak vagy a programnak a későbbiekben ismét szüksége lesz a korábbi regisztertartalmakra, akkor ezeket egy-egy rövid gépi utasítással ismét előveheti a veremből. Tipikus példája az ilyen veremműveleteknek a – valamilyen alprogramot meghívó – CALL utasítás. Ezt az utasítást végrehajtva a program feldolgozása a program valamelyik más részén folytatódik. Ahhoz, hogy az alprogram befejeződésekor a processzor „visszataláljon" arra a helyre, ahonnan elágazott, a CALL utasítás végrehajtásakor egyúttal a verembe írja azt a címet, ahonnan a fő program végrehajtását majd folytatnia kell. A veremhez való hozzáféréshez szükséges hozzáférési címet a processzor az SS verem szegmensregiszter és az SP veremmutató vagy a BP bázismutató regiszter tartalmából számítja ki.

Vizsgáljuk meg végül az ES jelű (extra segment) járulékos szegmensregisztert, amely csak néhány, az adatok átmásolásához szükséges gépi kódú utasítással össefüg-

gésben jut szerephez. Ha egy program az adatok átmásolásához csak olyan forrás- és célcímet használ, amelyek egy szegmensen (pl. a DS szegmensregiszter által kijelölt szegmensen) belül vannak, akkor ennek a szegmensregiszternek nincs szerepe. Ha viszont a programnak az adatokat a teljes megcímezhető tárban kell tudni mozgatnia, akkor ehhez a processzornak a DS és az ES regiszterekkel külön forrás- és külön célszegmenst kell kijelölnie. Az adatok átmásolását végző gépi utasítás végrehajtásakor a processzor az ES regiszter tartalmához adja hozzá a DI (destination index) cél indexregiszter tartalmát.

Miután röviden megismerkedtünk az egyes szegmensek feladatával, vizsgáljuk meg, hogy az egyes szegmensek egymáshoz képest hogyan helyezkednek, ill. helyezhetők el a tárban. Arról már volt szó, hogy a szegmensek tárbeli helyét az egyes szegmensregiszterek értéke határozza meg, és mivel a szegmensen belüli ofszetcím 16 bites, a szegmensek mérete maximum 64 kbájt lehet. Miután ezek a megállapítások mind a négy szegmensre egyaránt vonatkoznak, következik, hogy az egyes szegmensek a tárban akár négy különböző helyen is lehetnek, de egymást akár részben vagy egészben át is lapolhatják. Utóbbi esetben mind a négy szegmensnek azonos a kezdő és a végcíme. Mivel az adatok és a kód (program) tárolásához sok esetben nincs szükség a szegmens teljes, 64 kbájtos tartományára, különösen fontos, gyakorlati jelentősége van annak, hogy a szegmensek egymást átlapolhatják. Így például az adatok (DS regiszter) közvetlenül a gépi kódú utasítások (CS regiszter) után helyezhetők el a tárban, és ezáltal tárhely takarítható meg. Az 5. ábra egymást átlapoló és egymást nem átlapoló szegmensek táron belüli elhelyezkedését szemlélteti.



5. ábra Egymást nem átlapoló (a) és egymást átlapoló (b) szegmensek

Végezetül vizsgáljuk meg az állapotjelző regiszter (széles körben használatos angol nevén flag regiszter) funkcióit. Ez a regiszter alapvetően különbözik az összes többi regisztertől abban az értelemben, hogy míg valamennyi, más regiszter valamilyen értéket, címet, kódot stb. tartalmaz, az állapotjelző regiszter egyes bitjeinek önálló funkciói vannak. Ezek a bitek alapvetően a processzornak azt az állapotát jelzik, amelyet az az egyes gépi kódú utasítások végrehajtása után felvesz. Az egyes bitek jelentése (a nevek után zárójelben megadjuk a szakirodalomban széles körben használt angol megnevezést is):

– **Átvitelbit** (Carry flag): az értéke akkor 1, ha egy 8 vagy 16 bites művelet során átvitel keletkezik. Szerepet játszik a forgatási utasításoknál is, valamint jelzi egyes megszakítások és funkciók sikerességét is (ezekről a későbbiekben még bőven lesz szó).

– **Paritásbit** (Parity flag): értéke páros paritás esetén 1.

– **Segédátvitel bit** (Auxiliary flag): értéke akkor 1, ha egy művelet során a 8 bites operandus alsó négy bitjéről a felső négy bitre átvitel keletkezik. A decimális aritmetikánál jut szerephez.

– **Zérobit** (Zero flag): értéke akkor 1, ha egy művelet eredménye zéró.

– **Előjelbit** (Sign flag): értéke akkor 1, ha egy művelet eredményében a legnagyobb helyértékű bit értéke 1 (kettes komplemensben ez negatív szám).

– **Nyomkövetés bit** (Trace flag): ha a bit értéke 1, akkor a processzor egy utasítás végrehajtása után megáll a program végrehajtásával, hogy az utasítás hatását ki lehessen értékelni. Hibakeresésnél használatos.

– **Megszakítás bit** (Interrupt flag): ha az értéke 1, akkor engedélyezi a maszkolható megszakításokat, ellenkező esetben letiltja. (Ezekről a későbbiekben még lesz szó.)

– **Irányjelző bit** (Direction flag): adatblokkok feldolgozásánál a feldolgozás sorrendjét határozza meg: ha értéke 1, akkor a feldolgozás a magasabb, ha 0, akkor az alacsonyabb helyértékű címen kezdődik.

– **Túlcsordulás bit** (Overflow flag): értéke akkor 1, ha a művelet eredménye a befogadó regiszterben már nem ábrázolható előjelhelyesen.

A processzor regiszterkészletét bemutató, 1. ábrából látható, hogy a processzor a regiszterben lévő 16 bit egy részét használja csak fel. A használaton kívüli bitek olvasáskor 0-t adnak, egy esetleges állítási kísérletet figyelmen kívül hagynak.

1.3. A CPU „munkatársai"

Bár nem kétséges, hogy a számítógép agya a mikroprocesszor, és a rendszernek ez a legintelligensebb része, egy ennyire bonyolult rendszer valamennyi funkcióját mégsem tudja egymaga ellátni. A számítógép ezért más alkatrészeket (amelyek természetesen ugyancsak integrált áramkörök) is tartalmaz, amelyek a mikroprocesszor válláról számos tevékenységet levesznek, és ezáltal lehetővé teszik, hogy az a tulajdonképpeni feladatára, vagyis a gépi kódú programok végrehajtására koncentrálja a figyelmét (és nem utolsósorban az idejét). Ezek a „munkatársak" teszik lehetővé többek között azt is, hogy a számítógép kapcsolatot tartson a külső egységekkel, így például a lemezmeghajtóval vagy a képernyővel.

A következőkben megismerkedünk azokkal a legfontosabb alkatrészekkel, amelyek a PC-ben a mikroprocesszor munkáját segítik.

1.3.1. A közvetlen tárhozzáférést (DMA) vezérlő egység (8237)

Ez az alkatrész arról a képességéről kapta a nevét, hogy a processzort megkerülve, közvetlenül tud a RAM-ba adatokat írni, ill. onnan adatokat olvasni. A röviden csak DMA (Direct Memory Access = közvetlen tárhozzáférés) elemnek nevezett egységnek ez a képessége főként akkor hasznos, amikor a gépnek – viszonylag lassú – lemezműveleteket kell végrehajtania, vagyis amikor adatokat kell írnia a RAM-jából a lemezre vagy a lemezről valamilyen adatot kell a tárba beolvasnia. Ez az alkatrész jelentős feladatokat vesz le a processzor válláról, és így lehetővé teszi, hogy a processzor a programokat gyorsabban hajtsa végre.

Míg a PC/XT csak egy ilyen elemet tartalmaz, addig az AT-be a teljesítmény további növelése céljából két DMA-vezérlő egységet építettek be.

1.3.2. Megszakításvezérlő (8259)

A megszakításjelek (interrupt) arra szolgálnak, hogy a számítógép különböző egységei jelezhessék a CPU-nak azt a szándékukat, hogy a processzort valamilyen tevékenység elvégzésére kérik. Mivel egyidejűleg több egység is fordulhat ilyen kéréssel a CPU-hoz, ezek a megszakítási kérések először a megszakításvezérlő egységre kerülnek, amely azokat a CPU-hoz továbbítja. A megszakításvezérlőnek az a feladata, hogy az egyes megszakítási kéréseket a fontosságuk szerint rangsorolja (prioritást rendeljen hozzájuk), és elsőként mindig a legfontosabb, azaz a legmagasabb prioritású megszakítási kérést továbbítsa a CPU-hoz. A PC/XT-ben lévő megszakításvezérlő egyidejűleg 8 megszakítási kérést tud kezelni. Az AT munkájához ennyi azonban nem elegendő, ezért – a DMA ve-

zérlőhöz hasonlóan – ebben a rendszerben két megszakításvezérlő van egymással összekapcsolva, amelyek együttesen 15 megszakítási kérést képesek fogadni.

1.3.3. Periféria interfész (8255)

A nevének megfelelően a periféria interfész a CPU és a különböző perifériák, így pl. a billentyűzet vagy a hangszóró között létesít kapcsolatot. Ez az egység a közvetítő szerepét tölti be, amelyhez a CPU akkor fordul, amikor valamelyik egységgel valamit közölni akar. A CPU és a perifériák közötti kommunikáció minden esetben ezen a periféria interfészen keresztül bonyolódik le, közvetlen kapcsolat nem lehetséges.

1.3.4. Ütemjel-generátor (8284)

Ha egy számítógép agyának a mikroprocesszort tekintjük, akkor ennek analógiájára az ütemjel-generátort a rendszer „szívének" kell neveznünk. Ez a szív, amely tulajdonképpen egy kvarckristály, másodpercenként több milliót ver (pontosan 14.3228 MHz frekvenciával), és ez ütemezi a mikroprocesszort és a rendszer többi egységét. Mivel azonban egyik egység sem lenne képes ezzel az ütemmel lépést tartani, ezt a frekvenciát az áramkör, harmadára osztva is kiadja.

1.3.5. Órajel-generátor

Ez az – angol nevén timernek nevezett – egység számlálóként és órajel-generátorként is használható. Az egységnek az a feladata, hogy a kimenetein meghatározott, állandó időközönként ismétlődő impulzusokat, ún. órajeleket állítson elő. A különböző kimeneteken megjelenő impulzusok frekvenciája nincs előre meghatározva, hanem ezeket a bemenetére érkező jelek és az áramkör programozása határozza meg. Ez azt jelenti, hogy a kimenetek bármelyikének saját, rá jellemző frekvenciája van. Ezeknek a kimeneteknek mindegyike valamelyik perifériával van összekötve. Így például az egyik kimenet a hangszóróhoz vezet, míg egy másik a megszakításvezérlőhöz, amely minden egyes impulzus hatására kiváltja a 8-as megszakítást. Ennek pedig az a feladata, hogy a PC-ben az idő múlását számlálja.

1.3.6. Képernyővezérlő

Az eddig ismertetett alkatrészektől eltérően a képernyővezérlő nem a PC fő áramköri lemezén (az alaplapon), hanem az ún. videokártyán helyezkedik el, amely a bővítők egyikéhez van csatlakoztatva. Ezek a kártyák képességüket tekintve ugyan jelentősen eltérhetnek egymástól (monokróm vagy színes kép), többnyire azonban a 6845-ös típusú

képernyővezérlőre épülnek. Ennek az egységnek az a feladata, hogy a PC RAM-jában lévő képernyőtartalmat a géphez csatlakoztatott monitoron vagy tv-készüléken megjelenítse. E feladat végrehajtásához egy sor regisztert tartalmaz, amelyek a képernyőn megjelenő kép felépítését vezérlik, és amelyeket az operációs rendszeren keresztül lehet programozni.

1.3.7. Hajlékonylemez-meghajtó vezérlő

Ez az egység is egy bővítő kártyán helyezkedik el. Az egység – amelyet az operációs rendszer kezel – közvetlenül vezérli a lemezmeghajtókat. A lemezmeghajtó író/olvasó fejét a lemez megfelelő pontjára pozícionálja, és a lemezről adatokat olvas be, ill. a lemezre adatokat ír.

1.3.8. Aritmetikai társprocesszorok (8087, 80287, 80387)

Mivel a 8086-os, 80286-os és a 80386-os processzorok egyike sem képes lebegőpontos vagy valós tizedespontos számok feldolgozására, a PC fő áramköri lemezébe beépítettek egy foglalatot, amelyik egy aritmetikai társprocesszor befogadására alkalmas.

A PC/XT-ben ez a társprocesszor a 8087-es, az AT-ben a 80287-es, míg a legújabb, 80386-os processzorra épülő számítógépben a 80387-es típusjelet kapta. A lebegőpontos számok (mint pl. a 3.1415...) ugyan szoftver segítségével is feldolgozhatók, az aritmetikai társprocesszor azonban ezeket a számokat sokkal pontosabban és (akár 100-szor) gyorsabban dolgozza fel.

Ezek az aritmetikai társprocesszorok nemcsak az alapműveletek (összeadás, kivonás, szorzás és osztás), elvégzésére, hanem a trigonometrikus függvények (színusz, koszínusz stb.) értékeinek kiszámítására is képesek. Egy szám négyzetgyökének kiszámítása sem jelent problémát.

1.4. A CPU és a tár

Míg a számítógépes rendszer eddig ismertetett elemei a rendszer intelligens alkatrészeinek számítanak, a tár a rendszer passzív része, amely a megfelelő utasítások hatására adatokat tárol, ill. a tárolt adatokat kívánságra szolgáltatja. Mindegyik tárhely (rekesz) 1 bájtnyi (8 bit) információt tartalmaz, és az egyes tárhelyek 0-tól kezdődően sorban számozva vannak.

A tárral való kapcsolattartáshoz a számítógépes rendszer egyes elemei (a mikroprocesszort is beleértve) két, busznak vagy sínnek nevezett, számos vezetékből álló vezetékrendszert használnak (címbusz és adatbusz).

Amikor a processzor egy tárhelyhez akar fordulni, akkor először az illető tárhely sorszámát, azaz a tárhely címét a címbuszra helyezi. A címbusz meghatározott számú vezetékből áll, amelyek mindegyike egy-egy címbitet képvisel, így az értékük 0 vagy 1 lehet. A vezetékek együttvéve egy kettes számrendszerben ábrázolható értéket alkotnak, amelyek a tárnak azt a sorszámát adják meg, amelyik tárhelyhez a processzor fordulni akar. Mivel természetesen csak olyan tárhelyhez lehet hozzáférni, amelynek a címe a címbusz vezetékein ábrázolható, a megcímezhető tárhelyek számát ezeknek a vezetékeknek a száma határolja be.

A PC/XT-ben a címbusz szélessége 20 bit, így maximum 2^{20}, azaz több mint 1 millió (pontosan 1 Mbájt) tárhelyet tud megcímezni. A PC/AT címbuszának szélessége 24 bit, így ez a gép több mint 16 millió tárhelyet tud megcímezni (16 Mbájt).

Miután a processzor már tudja, hogy a tárnak melyik címéhez kell fordulnia, következhet a adatátvitelt kérő elem és a kiválasztott tárhely közötti adatmozgatás. A processzor ehhez az adatbuszt használja. Az ebben a buszban lévő vezetékek száma határozza meg azt, hogy a processzor egy ciklusban egyidejűleg hány bitet képes a tárból, ill. a tárba átvinni.

A PC/XT-ben lévő 8088-as processzor adatbusza 8 bites, vagyis ez a gép egyidejűleg 8 bit, tehát 1 bájt átvitelére képes. Mivel azonban a 8088-asnak 16 bites regiszterei is vannak, előfordulhat, hogy a tárból egy 16 bites értéket kell elolvasnia, ill. oda beírnia. Az adatbuszban lévő vezetékek száma ehhez azonban kevés. Ezért a processzor azt csinálja, hogy a 16 bites számot két, egyenként 8-8 bites számra (közismert elnevezés szerint egy felső és egy alsó bájtra) osztja fel, és ezeket egymás után továbbítja.

Az AT (valamint a 8086-osra és a 80286-osra épülő) számítógépes rendszerek ezt a problémát nem ismerik, mert ezekben az adatbusz szélessége 16 bit, és így egyidejűleg 16 bit széles adatokat tudnak a tár és a számítógép többi egysége között mozgatni. Többek között ez is a magyarázata annak, hogy miért gyorsabb az AT kisebb testvéreinél.

Az Intel 80xx-as processzorcsalád mindegyik tagja azonos módon tárolja az adatokat a tárban. A processzor a 0...7 biteket, vagyis az alsó (LOW) bájtot az alacsonyabb tárcímen, míg a 8...15 biteket, vagyis a felső (HIGH) bájtot az ezt követő, magasabb tárcímen tárolja. Ha például a processzornak az 1A2B(h) számot az 1000:0500 címen kell tárolnia, akkor az 1000:0500 címre a szám alsó bájtját (2B(h)-t), az 1000:0501 címre pedig a felső bájtját (1A(h)-t) helyezi el.

Az eddigiekben egészen általánosan tárról beszéltünk, és két lényeges részletet még nem említettünk. Az egyik az, hogy a processzor számára teljesen mindegy, hogy az éppen kiválasztott tárcím mögött egy RAM vagy egy ROM tárbeli cím található, a másik pedig az, hogy a processzor természetesen csak fizikailag is létező tárcímekhez fordulhat. Önmagában az a tény, hogy a processzor 1 Mbájtnyi tárat tud megcímezni, még nem jelenti azt, hogy a rendszer a valóságban is tartalmaz ennyi ROM vagy RAM tárelemet.

A tárral kapcsolatban beszélni kell a tárfelosztásról is. Bár a processzor a tárat nem osztja fel előre RAM vagy ROM részekre, az IBM PC-k széles körű elterjedtsége követ-

keztében mégis kialakult egyfajta tárfelosztás, amelyet a (legtöbb) kompatibilis számítógép is követ.

Ennek megfelelően a mikroprocesszor 1 Mbájtos címtára 16, egyenként 64 kbájtos tartományra, szegmensekre van felosztva (6. ábra).

szegmens	tárcím	rendeltetés
15	F000:0000 – F000:FFFF	BIOS ROM
14	E000:0000 – E000:FFFF	bővítő ROM kártyák helye
13	D000:0000 – D000:FFFF	
12	C000:0000 – C000:FFFF	további BIOS ROM
11	B000:0000 – B000:FFFF	video RAM
10	A000:0000 – A000:FFFF	további video RAM
9	9000:0000 – 9000:FFFF	RAM 640 kbájtig
8	8000:0000 – 8000:FFFF	RAM 576 kbájtig
7	7000:0000 – 7000:FFFF	RAM 512 kbájtig
6	6000:0000 – 6000:FFFF	RAM 448 kbájtig
5	5000:0000 – 5000:FFFF	RAM 384 kbájtig
4	4000:0000 – 4000:FFFF	RAM 320 kbájtig
3	3000:0000 – 3000:FFFF	RAM 256 kbájtig
2	2000:0000 – 2000:FFFF	RAM 192 kbájtig
1	1000:0000 – 1000:FFFF	RAM 128 kbájtig
0	0000:0000 – 0000:FFFF	RAM 64 kbájtig

6. ábra: A mikroprocesszor által megcímezhető 1 Mbájtnyi tár felosztása

A tár alsó 10 szegmense képezi a számítógép belső RAM tárát, amelynek együttes nagysága így maximum 640 kbájt lehet. Ennek a tárnak a mérete a különböző PC-gyártóknál eltérő lehet, de minimum 64 kbájtnak kell lennie, és a 0. szegmensen kell elhe-

lyezkednie. Ha ehhez még további RAM elemek is be vannak építve, akkor ezeknek az alsó címeken lévő tárakhoz hézagmentesen kell illeszkedniük, mert az egyes RAM szegmensek között nem lehet „üres" tárcím. A 0. társzegmensnek kitüntetett szerepe van: az operációs rendszer ennek az alsó részén helyez el a működéséhez szükséges számos fontos adatot és rutint.

A RAM tár után az A jelű társzegmens következik, ahol egy EGA nevű (Enhanced Graphics Adapter) grafikus kártya tárcímei találhatók. Ez a kártya különböző grafikus üzemmódokban képernyőtárként szolgál.

A B társzegmens a monokróm és a színes videokártya részére van fenntartva. Ezen a tárrészen a két kártya megosztozik: a a monokróm kártyához a tárrész alsó 32 kbájtja, a színes videokártyához pedig a felső 32 kbájtja van hozzárendelve. A kártyák azonban csak annyi tárcímet vesznek igénybe, amennyire a képernyőn lévő információ tárolásához ténylegesen szükségük van. Ez monokróm képernyő esetén mindössze 4 kbájt, színes képernyő esetén – tekintve, hogy ez további grafikus lehetőségekkel is rendelkezik – 16 kbájt.

Az e fölötti szegmensek már teljes egészükben ROM tárak. Az első, C-n kezdődő ROM szegmens néhány számítógépnél olyan BIOS rutinokat tartalmaz, amelyek nem részei az eredeti BIOS magnak. Így pl. az XT-nél itt helyezkednek el a merevlemezes meghajtó kezelését támogató rutinok. Mivel ez a tárterület még nincs teljesen kihasználva, ide még további hardvertámogató rutinok kerülhetnek.

A D és az E szegmensek külső, bővítő ROM kártyák számára vannak fenntartva. Mivel ilyeneket a PC-knél általában nem használnak, ez a terület gyakorlatilag kihasználatlan.

Végül az F szegmens tartalmazza a tulajdonképpeni BIOS rutinokat, a számítógép bekapcsolásakor lefutó öntesztelő rutinokat, a rendszert indító rutint, valamint a néhány típusban meglévő ROM BASIC-et.

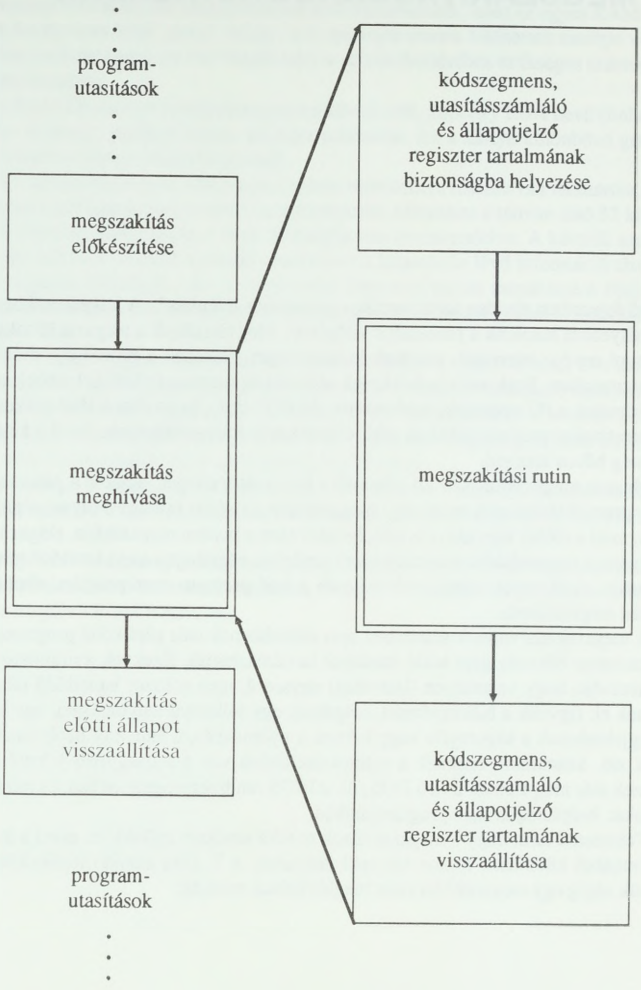
2. A MEGSZAKÍTÁSOKRÓL ÁLTALÁBAN

Az előző fejezetben röviden áttekintettük a processzor felépítését. A következőkben egy kicsit mélyebbre hatolunk a processzor világában. Megvizsgáljuk a megszakításokat (ismert angol nevén: interrupt), amelyek kulcsszerepet játszanak a processzor működési mechanizmusában. Ezek azért is érdekesek számunkra, mert segítségükkel többek között hozzáférhetünk a PC operációs rendszeréhez (a DOS-hoz), és az ebben lévő funkciókat, rendszerrutinokat programjainkban saját céljainkra is felhasználhatjuk. Erről a későbbiekben még bőven lesz szó.

Maga a megszakítás szó jól jellemzi a folyamat lényegét: amikor a processzor a futó program feldolgozása során egy megszakítási utasítást (ez ugyanolyan gépi kódú utasítás, mint a többi) kap, akkor a feldolgozást ezen a ponton megszakítja, elágazik arra a címre, ami a megszakítási utasításhoz van rendelve, végrehajtja az itt kezdődő megszakítási rutint, majd ennek végeztével visszatér a futó program azon pontjára, ahol a feldolgozást megszakította.

A megszakítási rutinok semmiben sem különböznek más gépi kódú programoktól, és a processzor bármely gépi kódú utasítását tartalmazhatják. Ezeknek a rutinoknak közös jellemzője, hogy valamilyen (látszólag) egyszerű, igen sokszor ismétlődő feladatokat látnak el: figyelik a billentyűzetet, reagálnak egy billentyű lenyomására, egy karaktert megjelenítenek a képernyőn vagy kiírnak a nyomtatóra, a lemezről beolvasnak egy blokkot stb. Számunkra ezeknek a rutinoknak abban van a legnagyobb jelentőségük, hogy ezek már meg vannak írva a DOS, ill. a BIOS rendszerprogramokban, és mint kész modulokat, beépíthetjük saját programjainkba.

Tekintettel arra, hogy a megszakítások mind a rendszer működése, mind a témánk szempontjából különösen fontos szerepet játszanak, a 7. ábra alapján lépésről-lépésre kövessük végig egy megszakítási rutin meghívásának menetét:



7. ábra: Egy megszakítási rutin hívása programból

Amint már említettük, amikor a processzor egy program végrehajtása során egy megszakítási utasításhoz érkezik, a futását azon a címen folytatja, amelyet a megszakítási utasítás, számára kijelöl. Ahhoz viszont, hogy a megszakítási rutin a tőle megkívánt feladatot lássa el (pl. egy meghatározott karaktert írjon a nyomtatóra), a programnak bizonyos előkészületeket kell tennie (pl. meg kell adnia, hogy milyen karakter jelenjen meg). Azt is figyelembe kell venni, hogy a megszakítási rutin végrehajtása során a processzor bizonyos regisztereinek tartalmát megváltoztathatja. Ahhoz, hogy a megszakításból való visszatéréskor a processzor ugyanazon regisztertartalmakkal folytathassa a munkáját, mint a megszakítás előtt, ezeket a regisztertartalmakat biztonságba kell helyezni. Ez is a program(ozó) feladata*. Ha mindezek az előkészületek megtörténtek, akkor már valóban sor kerülhet a megszakítási rutin meghívására. Mivel azonban aligha valószínű, hogy ez a rutin ugyanabban a szegmensben helyezkedik el, mint az éppen megszakított program, a kódszegmens és az utasításszámláló regiszterek tartalmát is meg kell változtatni. Ezt viszont már a processzor végzi: a regisztertartalmakat a verembe menti, végrehajtja a megszakítási rutint, majd ismét visszaállítja a regiszterek eredeti tartalmát, és visszatér a futó programhoz. (A teljesség kedvéért megemlítjük, hogy a processzor az állapotjelző regiszter tartalmát is tárolja, ill. visszaállítja. A megszakítási rutin által esetlegesen megváltoztatott regiszterek tartalmát most ismét a program(ozónak) kell helyreállítania.)

2.1. A megszakítási vektortábla

Az előzőekben csak általánosan egy megszakításról vagy egy megszakítási rutinról beszéltünk, és nem említettük azt, hogy a 8088-as nem csak egy, hanem összesen 256 megszakítást képes kezelni. Az egyes megszakítások 0-tól 255-ig sorszámmal vannak ellátva.

Mindegyik megszakítás egy-egy megszakítási rutint jelent, amelyet a processzor az illető megszakítás meghívásakor hajt végre. A megszakítások és a megszakítási rutinok közötti összekötőkapocs az ún. megszakítási vektortábla. Ez a tábla a 256 megszakítás mindegyikéhez tartalmaz egy bejegyzést, amely – mint egy vektor – a PC tárában lévő megszakítási rutin (szegmentált) kezdőcímére mutat.

Amikor meghívunk egy megszakítást, akkor a processzor ebből a vektortáblából automatikusan kiveszi a megszakításhoz tartozó megszakítási rutin kezdőcímét, és megkezdi a rutin végrehajtását.

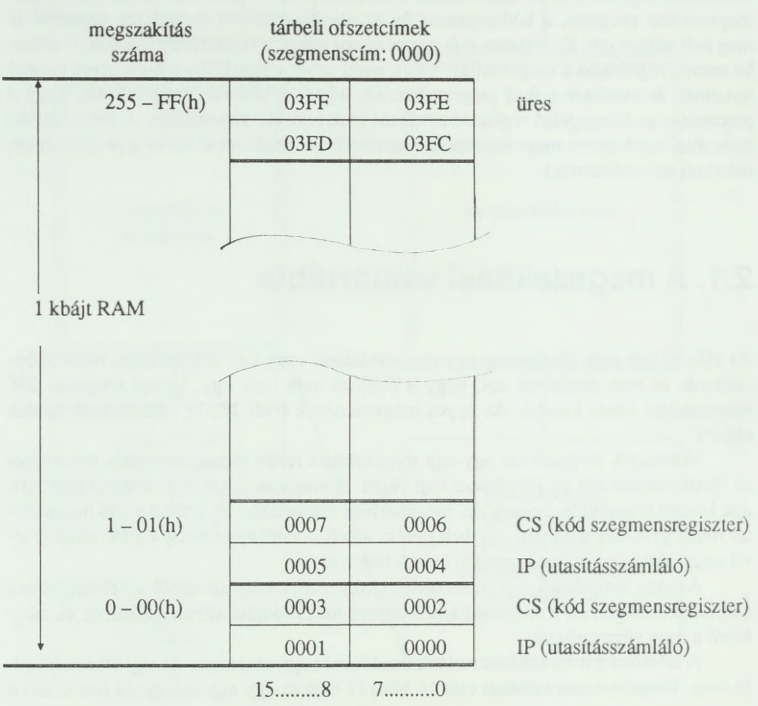
A táblában a megszakítási rutinok kezdőcímét egy szegmens- és egy ofszetcím adja meg. Természetesen mindkét cím 16 bites (2 bájtos), így egy bejegyzés (rutincím) 4

---

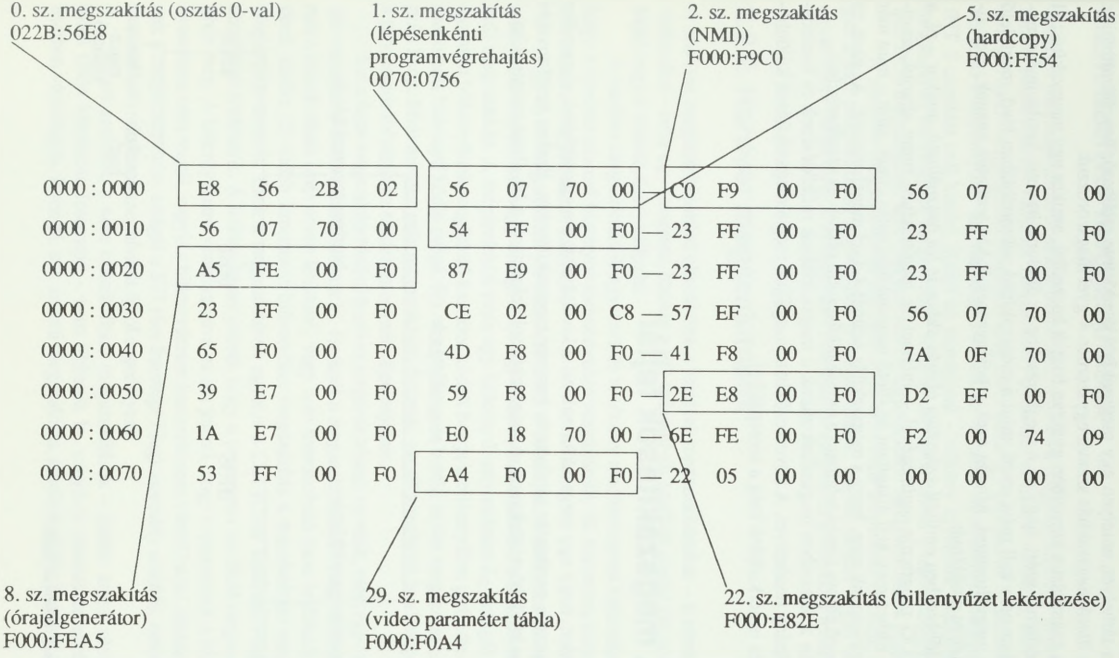
* [A processzor külső megszakításai esetén (amelyek a CPU szempontjából aszinkron érkeznek) a megszakításban használt regiszterek tartalmát a megszakítási rutin őrzi meg. – Lektor megjegyzése.]

bájton adható meg. Ezen belül az ofszetcím az alsó, a szegmenscím pedig a felső két bájton helyezkedik el. Az előzőekből következik, hogy a vektortábla teljes mérete 256 * 4 bájt = 1024 bájt (1 kbájt).

A megszakítási vektortábla a tár legalsó részén, a 0000:0000 és a 0000:03FF közötti címeken helyezkedik el. Az egyes rutinokhoz tartozó sorszámok megfelelnek a táblabeli bejegyzések számának: így a 0. számú megszakítási rutin a tábla 0(h) – 3(h) tárhelyein, az 1. számú rutin a 4(h) – 7(h) tárhelyein stb. található. Az utolsó, 255. számú megszakítás a 3FC(h) – 3FF(h) tárhelyeken található, és ez zárja le a megszakítási vektortáblát. A 8. ábrán bemutatjuk a megszakítási vektortábla felépítését, a 9. ábrán pedig a tárbeli elhelyezkedését. A teljes vektortáblát sorszámokkal, címekkel és a funkcióik leírásával együtt a III. kötet tartalmazza.



8. ábra: A megszakítási vektortábla felépítése



9. ábra: A megszakítási vektortábla tárbeli elhelyezkedése (részlet) és néhány rutin kezdőcíme

Azt a tárhelyet, amelyen egy megszakítás kezdőcíme található, úgy számíthatjuk ki, hogy az illető megszakítás számát egyszerűen 4-gyel megszorozzuk.

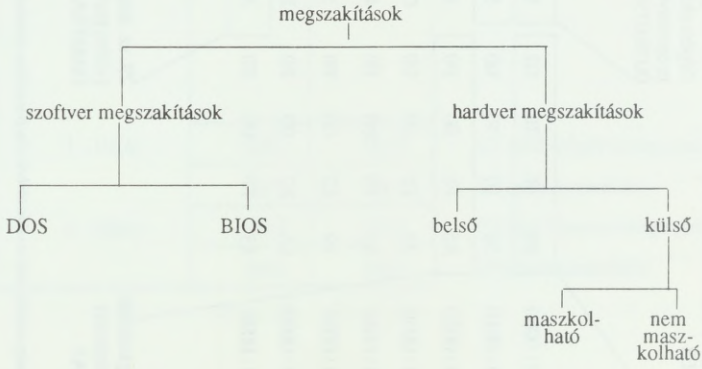
Ezt a számítást a könyvben gyakran fogjuk használni, amikor egy megszakítást el akarunk „téríteni" azért, hogy az a meghívásakor az általunk írt rutin kezdőcímére mutasson. Ehhez nem kell mást tenni, mint a megszakítási vektortáblában lévő, megfelelő bejegyzést megváltoztatni. Miután ezek a bejegyzések RAM területen vannak, ezek átírása nem jelent problémát.

Az előbbiekben említett számítási eljárás akkor is jól használható, amikor – pl. a DOS DEBUG programja segítségével – valamelyik megszakítási rutint akarjuk tanulmányozni. Ehhez meg kell állapítani az illető megszakítási rutin címét, amit – mint már tudjuk – úgy kapunk meg, hogy a megszakítás számát 4-gyel megszorozzuk. Az ezen, és a rákövetkező három címen található a megszakítási rutin ofszet- és szegmenscíme.

Ebben a fejezetben megismerkedtünk a megszakítások működésének és hasznosságának néhány részletével. A következőkben vizsgáljuk meg a megszakítások különböző fajtáit, és ismerkedjünk meg a szerepükkel.

2.2. A megszakítások fajtái

Attól függően, hogy egy megszakításnak mi a kiváltó oka, a megszakítások alapvetően két fő csoportba oszthatók: szoftver és haruver megszakításokra. Ezeken belül további csoportokat különböztetünk meg. A megszakítások felosztását a 10. ábrán mutatjuk be.



10. ábra: A megszakítások fajtái

2.2.1. Szoftver megszakítások

Szoftver megszakításokról akkor beszélünk, amikor a megszakítást egy gépi kódú programon belüli utasítás váltja ki. Az assembly nyelvben ez az INT utasítás, amellyel együtt a meghívandó megszakítás számát is meg lehet adni. Így például az 5. számú megszakítás (ami a képernyőről ún. hardcopyt készít az aktuális nyomtatóra) meghívásához az INT 5 utasítást kell kiadni. Ezzel az utasítással – néhány kivételtől eltekintve – a 256 megszakítás mindegyike meghívható. E megszakítások közül a legfontosabb a 21. számú, DOS megszakítás, amelyen keresztül a legfontosabb rendszerrutinok hívhatók meg.

A szoftver megszakítások természetesen nem csak assembler szintről, hanem magas szintű programozási nyelvekből is meghívhatók, hiszen ezeket végső soron mindegyik nyelv fordítója az INT assembly utasításnak megfelelő gépi kódra fordítja le.

2.2.2. Hardver megszakítások

A szoftver megszakításokkal ellentétben a hardver megszakításokat – a nevükből is következően – nem egy program, hanem a PC valamelyik hardver eleme (pl. a lemezmeghajtó vagy a billentyűzet) váltja ki.

A hardver megszakítások egyszerű és hatékony mechanizmust biztosítanak ahhoz, hogy a hardver elemei bizonyos eseményekre reagáljanak, ill. bizonyos események bekövetkezését figyelemmel kísérjék.

Egy ilyen megszakítás jól szemléltethető a billentyűzet példáján: mindannyiszor, ahányszor lenyomunk vagy elengedünk egy billentyűt, kiváltódik a billentyűzet (9. számú) megszakítás. A megszakítási rutin egy billentyű lenyomására úgy reagál, hogy a lenyomott billentyűnek megfelelő karakter kódját beírja a billentyűzet pufferbe az utoljára lenyomott billentyű kódja mögé. Ha ez a puffer már betelt, akkor megszólal az ismert sípoló hang. Mint minden más megszakítási rutinnál, ennél a rutinnál is a befejeződésekor a processzor folytatja az eredeti program végrehajtását.

Ezt a fajta megszakítást külső hardver megszakításnak nevezik, mert egy külső egység (nevezetesen a billentyűzet) váltotta ki. Ezeknél a megszakításoknál még különbséget tesznek a maszkolható (más néven letiltható) vagy nem maszkolható (nem letiltható) megszakítások között. Ez előbb ismertetett billentyűzet megszakítás a maszkolható megszakítások csoportjába tartozik. Ezek a megszakítások ugyanis a CLI (Clear Interrupt Flag) assembly utasítással letilthatók. A letiltás itt azt jelenti, hogy a processzor az illető megszakítást nem hívja meg. A billentyűzetre vonatkoztatva ez azt jelenti, hogy a processzor a billentyűk lenyomására nem reagál, vagyis a billentyűzeten keresztül nem vihető be jel. Ennek az állapotnak a megszüntetésére szolgál a STI (Set Interrupt Flag) assembly utasítás, amely a letiltott maszkolható utasítások meghívását újból engedélyezi.

Ezzel szemben a nem maszkolható megszakítások – ismert angol rövidítésük az NMI (Non Maskable Interrupt) – nem tilthatók le, vagyis a processzor ezeket minden esetben végrehajtja. Ilyen pl. a 2. számú megszakítás, amelyik akkor kerül végrehajtásra,

ha a PC a RAM tárban hibát észlel. Ekkor a képernyőre kiíródik a hibának megfelelő üzenet, amely jelzi, hogy valamelyik RAM elemet ki kell cserélni.

Hátra van még a belső hardver megszakítások ismertetése. Ezeket nem valamelyik külső egység, hanem a számítógép fő áramköri lapján lévő processzorok valamelyike váltja ki. Erre példa a 0. számú megszakítás, amelyet a CPU-ban osztáskor keletkező állapot válthat ki. Ha ugyanis osztásnál a hányados nem fér el a rendelkezésre álló helyen, ez a megszakítás kerül végrehajtásra (nullával való osztás).

3. MEGSZAKÍTÁSOK MEGHÍVÁSA MAGAS SZINTŰ PROGRAMOZÁSI NYELVEKEN

Az assembly nyelven programozók számára egy megszakítás meghívása nem jelent különösebb problémát. A dolguk mindössze annyi, hogy a megfelelő regiszterekbe beírják a megszakítás által várt értékeket, majd ezt követően kiadják az INT megszakításszám utasítást.

Mivel a magas szintű nyelveken ez a lehetőség nem áll rendelkezésre, a programozónak a kívánt megszakítás meghívásához a használt nyelvtől függően különböző műveleteket, procedúrákat vagy alprogramokat kell igénybe vennie.

A Turbo Pascal-lal, a Borland Turbo C-vel és a Microsoft C-vel szemben a BASIC még ezt a lehetőséget sem biztosítja. Ha ennek ellenére nem akarunk lemondani azokról az előnyökről, amelyeket a megszakítások meghívásának lehetősége jelent, akkor a BASIC-ben arra kényszerülünk, hogy a Pascal-ban és a C-ben erre a célra definiált funkciókat egy rövid, assembler szintű rutin bekapcsolásával valósítsuk meg. A következő fejezetben bemutatjuk az ezt a feladatot megvalósító rutint, és elmagyarázzuk a működését.

A következő három fejezet mindegyike a megszakítások meghívásával foglalkozik, mégpedig azon a három magas szintű nyelven, amelyet a könyv későbbi példaprogramjaiban is használunk. Mindegyik fejezetben részletesen bemutatjuk, hogyan kell az egyes nyelveken a megszakításokat meghívni, és hogy ennek során mire kell ügyelni. Azért, hogy az elméleti fejtegetéseket a gyakorlatban is hasznosítani lehessen, mindhárom fejezet végén egy rövid példaprogram található.

Úgy gondoljuk, hogy a különböző nyelveken megírt példaprogramok egymással való összevetése is hasznos lehet. Bár a programok csak a megszakítások meghívását példázzák, mégis kerek, futtatható programok.

3.1. Megszakítások meghívása BASIC nyelven

Az IBM PC-ken legelterjedtebben használt két BASIC interpreter a BASICA (az IBM szoftvere) és a GW-BASIC (a Microsoft szoftvere). Mi a könyvünkben az utóbbival foglalkozunk, mert ez gyakorlatilag valamennyi, IBM PC-vel kompatibilis gépen is futtatható.

A fejezet elején már említettük, hogy a GW-BASIC-ben sajnos nincsen olyan funkció, amellyel egy megszakítást közvetlenül meg lehetne hívni. Van viszont egy olyan utasítás, amellyel gépi kódú programok hívhatók meg, és hajthatók végre. Ez a CALL utasítás, amellyel azon túlmenően, hogy egy gépi kódú alprogram hívható meg, a meghívott alprogramnak még paraméterek is átadhatók. A meghívandó gépi kódnak azon a 64 kbájtos területen belül kell lennie, amelyet a GW-BASIC a programsorok és a változók tárolására is használ. Ezért gépi kódú alprogramok meghívásakor az interpreter tudomására kell hozni, hogy a BASIC programsorok és változók tárolásához már nem használhatja a teljes, 64 kbájtos területet, nehogy ezek esetleg felülírják az ugyanitt elhelyezkedő gépi kódú programot. Ha ezt nem biztosítanánk, akkor egy ilyen, esetleges felülírás után a a gépi rutin meghívásakor a programunk egészen bizonyosan „elszállna". Azt, hogy az interpreter nem használhatja saját céljaira a teljes területet, már a DOS-ból való meghívásakor közölhetjük vele. Ha a meghívásakor a GW-BASIC név után megadjuk a /m: paramétert, majd utána annak a tárhelynek a címét, amelyen a gépi kód kezdődik, és amelyet az interpreternek nem szabad megzavarnia, akkor biztosak lehetünk abban, hogy a tárban a BASIC és a gépi kód nem keveredik össze. Mivel a példaprogramokban a gépi kódú rutinunkat a használható tárterület legfelső részére, egészen pontosan a 60000-es tárcímtől kezdődően fogjuk elhelyezni, az interpretert a DOS-ból így kell meghívni:

```
C>gwbasic/m:60000
```

Ezzel biztosítottuk, hogy az interpreter a 60000-es tárcím feletti területet nem fogja használni, így ide nyugodtan beírhatjuk a gépi kódú programunkat. Ezt úgy tehetjük meg, hogy a gépi kódú programot mint alprogramot beépítjük a BASIC nyelvű programunkba, majd ezt a program minden egyes indításakor végrehajtjuk. A következő programlista ezt az alprogramot, ill. a meghívását tartalmazza.

```
9000 '
9010 '  A MEGSZAKÍTASHÍVÓ RUTIN INICIALIZÁLÁSA
9020 '     BE: -
9030 '     KI: RCIM a Rutin CÍMe
9040 '
9050 '
9060 RCIM = 60000!                'A rutin kezdőcíme a BASIC szegmensben.
9070 DEF SEG                      'A BASIC szegmens beállítása.
9080 RESTORE 9140
9090 FOR I% = 0 TO 160                              'A rutin betöltése a
9100     READ X% : POKE RCIM + I%, X%               'tárba.
9110 NEXT
9120 RETURN
9130 '
9140 DATA    85, 139, 236,  30,   6, 139, 118,  30, 139,   4, 232, 140
9150 DATA     0, 139, 118,  12, 139,  60, 139, 118,   8, 139,   4,  61
9160 DATA   255, 255, 117,   2, 140, 216, 142, 192, 139, 118,  28, 138
9170 DATA    36, 139, 118,  26, 138,   4, 139, 118,  24, 138,  60, 139
9180 DATA   118,  22, 138,  28, 139, 118,  20, 138,  44, 139, 118,  18
9190 DATA   138,  12, 139, 118,  16, 138,  52, 139, 118,  14, 138,  20
9200 DATA   139, 118,  10, 139,  52,  85, 205,   0,  93,  86, 156, 139
9210 DATA   118,  12, 137,  60, 139, 118,  28, 136,  36, 139, 118,  26
9220 DATA   136,   4, 139, 118,  24, 136,  60, 139, 118,  22, 136,  28
9230 DATA   139, 118,  20, 136,  44, 139, 118,  18, 136,  12, 139, 118
9240 DATA    16, 136,  52, 139, 118,  14, 136,  20, 139, 118,   8, 140
9250 DATA   192, 137,   4,  88, 139, 118,   6, 137,   4,  88, 139, 118
9260 DATA    10, 137,   4,   7,  31,  93, 202,  26,   0,  91,  46, 136
9270 DATA    71,  66, 233, 108, 255
```

A megszakítás meghívását végző assembler rutint a DATA sorok tartalmazzák, ahonnan a BASIC program az egyes adatokat egy FOR – READ – POKE – NEXT ciklussal kiolvassa, és a 60000-es tárcímtől kezdődően folyamatosan a tárba írja. Megjegyezzük, hogy a 60000-es kezdőcím önkényesen megválasztott cím (a választást csak az indokolja, hogy így egyrészt a gépi kódú rutinunk még bőven elfér a tárban, másrészt viszont a BASIC program számára is meglehetősen nagy, egybefüggő terület áll rendelkezésre). A rutin más címtől kezdődően is elhelyezhető: ehhez csak a 9060-as sorban levő értéket kell megváltoztatni. Ekkor természetesen arról sem szabad megfeledkezni, hogy a GW-BASIC-et az operációs rendszer szintjéről ennek megfelelő paraméterrel kell meghívni.

Ahhoz, hogy az assembler programunkat egy megszakítás meghívására használhassuk, a programot először természetesen le kell futtatni. Ezért a BASIC program elejére el kell helyezni egy GOSUB utasítást, amely a 2. példaprogramban az 1230-as sorban áll:

```
1230 GOSUB 9000
```

A program ennek az utasításnak a végrehajtásával inicializálja a gépi kódú alprogramot, amely ezek után az 1230-as és a 9000-es sorszámok közé elhelyezett fő program bármely helyéről meghívható. A meghívást végző programsor:

```
CALL RCIM (MSZ%, AH%, AL%, BH%, BL%, CH%, CL%, DH%, DL%, DI%, SI%, ES%, FL%)
```

A zárójelek között azok a változók vannak felsorolva, amelyeket paraméterekként az assembler programnak átadunk. Valamennyi változóra vonatkozik, hogy ezeket valóban egész számú változókként, és nem állandókként kell átadni (ellenkező esetben a BASIC interpreter nem találná meg), továbbá az, hogy az egyes változók fenti nevei csak jelképesek, és természetesen más nevük is lehet.

Az első változó, amelynek a példánkban MSZ% a neve, a meghívandó megszakítás számát tartalmazza. Ennél a változónál különös gondossággal kell eljárni, mert ha ennek a változónak véletlenül hibás értéket adunk, vagy ha előzőleg nem adnánk neki értéket, akkor ez kiszámíthatatlan következményekhez vezethet. Az ez után következő változók tartalma a jelölésüknek megfelelő processzorregiszterekbe másolódik át. Azokba a regiszterekbe, amelyekre a megszakítás meghívásához nincs szükség, tetszőleg értékű, egész típusú érték kerülhet, így ezeknél előzetes értékadásra nincs is szükség (ezek az ún. dummy-változók, amelyeket a továbbiakban üres változóknak nevezünk). Az összes többi regiszterrel ellentétben az ES regiszternek átadott érték nem kerül automatikusan az ES regiszterbe. Ha ugyanis az ES regiszternek a –1 értéket adjuk át, akkor az ES regiszterbe a DS regiszter tartalma másolódik át.

A megszakítás meghívása után az átadott változókba azok az értékek kerülnek, amelyekkel a megszakítás a befejeződése után visszatér, és a megfelelő regiszterekbe visszatölt. Ez egyúttal azt is jelenti, hogy az érintett változóknak a megszakítás meghívása előtti és utáni értéke nem feltétlenül azonos. Ezért bizonyos körülmények között szükség lehet arra, hogy egy és ugyanazon megszakítás közvetlenül egymásra következő meghívása között, a megszakítás ismételt meghívása előtt a változóknak újból kezdeti értéket kell adni. Erről a későbbiekben még lesz szó.

Amint a megszakítási alprogramunkat meghívó BASIC programsorunkból látszik, az alprogram a processzor általános regiszterei (AX, BX, CX, DX) körében bájtos regiszterekkel dolgozik (AH, AL, BH, BL stb.). Vannak azonban olyan megszakítások is, amelyeknél a kívánt információt nem a 8, hanem a teljes, 16 bites regiszter tartalmazza. Ezekben az esetekben a BASIC programban a két fél regisztert így kapcsolhatjuk össze egy teljes regiszterré:

```
30 AX% = AH% * 256 + AL%
```

Ennek fordítottját, azaz egy teljes regiszter két bájtos regiszterre bontását ezekkel a programsorokkal végezhetjük el:

```
40 AH% = INT(AX% / 256)
50 AL% = AX% AND 255
```

Számos olyan megszakítás van, amelynél a végrehajtás sikerességét az állapotjelző (flag) regiszterben lévő átvitelbit (Carry flag) értéke mutatja. Ezért a BASIC nyelvű programokban is szükség lehet arra, hogy a megszakítás végrehajtását követően ennek – ill. esetenként valamelyik másik jelzőbitnek – az értékét megvizsgáljuk. Mivel a megszakítás befejeződésekor az állapotjelző regiszter tartalma az FL% nevű változóba íródik

be, ennek „bitekre szedésével" az egyes bitek értéke megállapítható. A következő programsorok az átvitelbit és a zéróbit értékét vizsgálják:

```
60 IF FL% AND 1 = 0 THEN PRINT „ÁTVITELBIT TÖRÖLVE" ELSE
   PRINT „ÁTVITELBIT BEKAPCSOLVA"
70 IF FL% AND 64 = 0 THEN PRINT „ZÉRÓBIT TÖRÖLVE" ELSE
   PRINT „ZÉRÓBIT BEKAPCSOLVA"
```

Vannak olyan megszakítások, amelyek meghívásához egy karakterlánc tárbeli kezdőcímét (szegmens- és ofszetcímet) kell átadni. A BASIC a karakterláncokat természetesen sztringváltozókként tárolja, amelyeket változónevek azonosítanak. A karakterlánc megadásakor nem szabad megfeledkezni a BIOS, ill. a DOS által a karakterlánc lezárásaként várt, és ezért a karakterlánc végére beírandó $ vagy CHR$(0) karakterről. A GW-BASIC-ben a sztringváltozók címének megállapítására a VARPTR funkció szolgál. Az ennek segítségével meghatározott – kétbájtos – cím a karakterlánc kezdetének ofszetcímére mutat. Tekintettel azonban arra, hogy a BASIC-ben a változók mindegyikének azonos a szegmenscíme, a megszakítás meghívásához elegendő csak az ofszetcím átadása.

Az ofszetcím alsó és felső (LO és HI) bájtját a következő két programsorral állapíthatjuk meg:

```
80 LO = PEEK(VARPTR(STR_NEV)+1)     'ofszetcím alsó bájtja
90 HI = PEEK(VARPTR(STR_NEV)+2)     'ofszetcím felső bájtja
```

Egy karakterlánc címének megállapítása kapcsán felhívjuk a figyelmet a BASIC interpreter egyik sajátosságára, az ún. szemétgyűjtő (garbage collection) rutinra. Ennek a rutinnak az a feladata, hogy a már nem használt változókat időről-időre törölje, vagyis mintegy „kisöpörje" a tárból, hogy ezzel tárhelyet szabadítson fel. Ez a rendcsinálás azonban egyúttal azzal is jár, hogy a még használatban lévő változók tárbeli címei megváltoznak. Ezért azokban az esetekben, amikor egy megszakítás meghívásához egy karakterlánc címét kell átadni, akkor ennek a címét közvetlenül a megszakítás meghívása előtt kell megállapítani. Ha ugyanis ezt a címet valahol már a program elején megállapítanánk, akkor előfordulhat, hogy a „szemétgyűjtő" rutin miatt a cím a megszakítás meghívásáig akár többször is megváltozik.

Következzen most egy kis példaprogram, amely az előzőekben bemutatott gépi kódú alprogramot arra használja, hogy egy DOS funkció segítségével szöveget írjon a képernyőre.

```
1000 '
1005 '  ┌──────────────────────────────────────────────────────┐
1010 '  │                    M S Z D E M O . B A S             │
1020 '  │                                                      │
1030 '  │ Funkciója : megszakításhívás szemléltetése karakterlánc │
1040 '  │             megjelenítésével.                        │
1045 '  │                                                      │
1050 '  └──────────────────────────────────────────────────────┘
1060 '
1070 DEFINT Q                                   'Q "dummy" egész érték.
1080 KEY OFF : CLS
1090 PRINT : PRINT
1100 PRINT "FIGYELMEZTETÉS :" : PRINT
1110 PRINT "A 60000-es címtől kezdődően betöltött megszakításhívó ru-"
1120 PRINT "tin felulírását elkerülendő a GWBASIC-et GWBASIC /M:60000"
1130 PRINT "parancssal kell indítania. Ha nem így hívta a GWBASIC-et,"
1140 PRINT "akkor üsse le az <s> billentyűt, egyébként egy tetszöle-"
1150 PRINT "ges másik billentyűt !"
1160 '
1170 Z$ = ""
1180 WHILE Z$ = ""                              'Helytelen indítás
1190     Z$ = INKEY$                            'esetén a program
1200 WEND                                       'leállítása.
1210 IF Z$ = "s" OR Z$ = "S" THEN CLS:END
1220 '
1230 GOSUB 9000                                 'Megszakításhívó rutin
1240                                            'betoltése.
1250 CLS
1260 '
2000 '
2010 '  ┌──────────────────────────────────┐
2020 '  │ KAR.LÁNC MEGJELENÍTÉS MEGSZAKÍTÁSSAL │
2060 '  └──────────────────────────────────┘
2070 '
2100 PRINT : PRINT
2200 TXT$ = "******** A megjelenítés a 21(h) megszakítás "
2205 TXT$ = TXT$ + "9-es funkciójával történt ********$"
2207 '
2210 MSZ% = &H21                                '21(h) megszakítás.
2220 FUN% = &H9                                 'Karlánc meggjel. funkció.
2230 OFFL% = PEEK(VARPTR(TXT$)+1)               'Ofszetcím LOW  bájt.
2240 OFFH% = PEEK(VARPTR(TXT$)+2)               'Ofszetcím HIGH bájt.
2242 '
2250 CALL RCIM(MSZ%,FUN%,Q,Q,Q,Q,Q,OFFH%,OFFL%,Q,Q,Q,Q)
2252 '
2255 PRINT : PRINT
2258 Z$ = ""
2260 WHILE Z$ = ""
2270     Z$ = INKEY$                            'Kiszállás előtt egy
2280 WEND                                       'billentyű megnyomá-
2282 CLS                                        'sára várakozás.
2284 END
2288 '
9000 '
9010 '  ┌──────────────────────────────────────┐
9020 '  │ A MEGSZAKíTASHíVó RUTIN INICIALIZALASA │
9030 '  │     BE: -                            │
9040 '  │     KI: RCIM a Rutin CIMe            │
9050 '  └──────────────────────────────────────┘
9060 RCIM = 60000!                  'A rutin kezdőcíme a BASIC szegmensben.
9070 DEF SEG                        'A BASIC szegmens beállítása.
9080 RESTORE 9140
9090 FOR I% = 0 TO 160                          'A rutin betoltése a
9100     READ X% : POKE RCIM + I%, X%           'tárba.
9110 NEXT
```

```
9120 RETURN
9130 '
9140 DATA     85,  139,  236,   30,    6,  139,  118,   30,  139,    4,  232,  140
9150 DATA      0,  139,  118,   12,  139,   60,  139,  118,    8,  139,    4,   61
9160 DATA    255,  255,  117,    2,  140,  216,  142,  192,  139,  118,   28,  138
9170 DATA     36,  139,  118,   26,  138,    4,  139,  118,   24,  138,   60,  139,
9180 DATA    118,   22,  138,   28,  139,  118,   20,  138,   44,  139,  118,   18
9190 DATA    138,   12,  139,  118,   16,  138,   52,  139,  118,   14,  138,   20
9200 DATA    139,  118,   10,  139,   52,   85,  205,    0,   93,   86,  156,  139
9210 DATA    118,   12,  137,   60,  139,  118,   28,  136,   36,  139,  118,   26
9220 DATA    136,    4,  139,  118,   24,  136,   60,  139,  118,   22,  136,   28
9230 DATA    139,  118,   20,  136,   44,  139,  118,   18,  136,   12,  139,  118
9240 DATA     16,  136,   52,  139,  118,   14,  136,   20,  139,  118,    8,  140
9250 DATA    192,  137,    4,   88,  139,  118,    6,  137,    4,   88,  139,  118
9260 DATA     10,  137,    4,    7,   31,   93,  202,   26,    0,   91,   46,  136
9270 DATA     71,   66,  233,  108,  255
```

A program a következő részekből áll: az 1230-es sorban meghívjuk azt az alprogramot, amelyik a megszakítások meghívásához szolgáló assembler rutint inicializálja. Ezt követően a megszakítás meghívásához szükséges változóknak értékeket adunk. A TXT$-ba a kiírandó szöveg kerül. A szöveget egy $ karakter zárja le, mert a karakterlánc kiírását végző DOS funkció számára ez jelzi a karakterlánc végét (a funkció ezt a végjelet természetesen nem írja ki). Az MSZ% és a FUN% nevű változókba a meghívandó megszakítás és az illető funkció száma kerül. Az OFFL% és az OFFH% változók a TXT$ ofszetcímét tartalmazzák.

Magát a megszakítást a 2250-es sorban lévő CALL utasítás hívja meg. Az utasítás első paraméterként az MSZ%-ot adja át, amelyik a meghívandó megszakítás számát tartalmazza. Ez után következik a FUN%, amelynek tartalma a megszakítás meghívása előtt a processzor AH regiszterébe másolódik, és ezáltal közli a 21(h)-es DOS megszakítással a meghívandó funkció számát. Ezt néhány, Q-val jelölt, „üres" (dummy) változó követi, amelyek azoknak a regisztereknek felelnek meg – és adnak át értéket –, amelyeknek a meghívott funkció szempontjából nincs jelentőségük. Ebből következően ezeknek a változóknak az értékei közömbösek. Esetünkben ezek sorrendben az AL, BH, BL, CH és a CL regisztereket jelentik. Ezek után következnek az OFFH% és az OFFL% nevű változók, amelyek a karakterlánc ofszetcímét adják át a DX regiszternek. A további regiszterek tartalma szintén közömbös a funkció meghívása szempontjából, ezért ezeknek megint megfeleltethető a Q változó.

Amint már említettük, vannak olyan megszakítások, ill. funkciók, amelyek meghívásához egy karakterlánc tárbeli kezdőcímét (szegmens- és ofszetcímet) kell átadni. Ilyen a példánkban használt, karakterláncot kiíró funkció is. A GW-BASIC-ben azonban elegendő, ha a karakterláncnak csak az ofszetcímét adjuk át, ugyanis a szegmenscímet – amit a funkció a DS regiszterben vár – a BASIC interpreter automatikusan beállítja.

Azok számára, akiket érdekel a BASIC program DATA soraiban elhelyezett gépi kódú rutin, következzék a rutin assembly nyelvű listája.

```
;
;                    M S Z H I V A S . A S M
;
;  Funkciója: GWBASIC-ből teszi lehetővé megszakítások hívását.
;
;  Hívása : CALL CIM(MSZ,AH,AL,BH,BL,CH,CL,DH,DL,DI,SI,ES,FLAGS)
;
;           ahol minden egyes változónak egész típusúnak kell
;           lennie.
;
;  A GWBASIC-ből történő híváskor a változók a következő pozí-
;  ciókkal kerulnek a veremre :
;
;     MSZ = SP+30     AH = SP+28     CH = SP+20     DI    = SP+12
;                     AL = SP+26     CL = SP+18     SI    = SP+10
;                     BH = SP+24     DH = SP+16     ES    = SP+ 8
;                     BL = SP+22     DL = SP+14     FLAGS = SP+ 6
;
;
;— KÓD ----------------------------------------------------------

kod     segment

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

;— Megszakítás hívó rutin ---------------------------------------

basmsz  proc far

        push bp                 ; A GWBASIC szegmensregiszterek
        mov  bp,sp              ; mentése.
        push ds
        push es

        mov  si,[bp+30]         ; MSZ% --> AX.
        mov  ax,[si]
        call mszbeall           ; MSZ% tárolása.

cimi    label near

        mov  si,[bp+12]         ; DI% --> DI.
        mov  di,[si]
        mov  si,[bp+8]          ; ES% --> AX.
        mov  ax,[si]
        cmp  ax,-1

        jne  cimke              ; Ha ES = -1 lett átadva, akkor
        mov  ax,ds              ; DS = ES.

cimke:  mov  es,ax              ; ES% --> ES.
        mov  si,[bp+28]         ; AH% --> AH.
        mov  ah,[si]
        mov  si,[bp+26]         ; AL% --> AL.
        mov  al,[si]
        mov  si,[bp+24]         ; BH% --> BH.
        mov  bh,[si]
        mov  si,[bp+22]         ; BL% --> BL.
        mov  bl,[si]
        mov  si,[bp+20]         ; CH% --> CH.
        mov  ch,[si]
        mov  si,[bp+18]         ; CL% --> CL.
```

```
        mov  cl,[si]
        mov  si,[bp+16]          ; DH% --> DH.
        mov  dh,[si]
        mov  si,[bp+14]          ; DL% --> DL.
        mov  dl,[si]

        mov  si,[bp+10]          ; SI% --> SI.
        mov  si,[si]
        push bp                  ; Bázismutató a veremre.

cim2    label near

        int 0                    ; Megszakítás hívás.

        pop  bp                  ; Bázismutató visszaállítása.
        push si                  ; SI a veremre.
        pushf                    ; Flag-ek a veremre.

        mov  si,[bp+12]          ; DI% értékének beállítása.
        mov  [si],di
        mov  si,[bp+28]          ; AH% értékének beállítása.
        mov  [si],ah
        mov  si,[bp+26]          ; AL% értékének beállítása.
        mov  [si],al
        mov  si,[bp+24]          ; BH% értékének beállítása.
        mov  [si],bh
        mov  si,[bp+22]          ; BL% értékének beállítása.
        mov  [si],bl
        mov  si,[bp+20]          ; CH% értékének beállítása.
        mov  [si],ch
        mov  si,[bp+18]          ; CL% értékének beállítása.
        mov  [si],cl
        mov  si,[bp+16]          ; DH% értékének beállítása.
        mov  [si],dh
        mov  si,[bp+14]          ; DL% értékének beállítása.
        mov  [si],dl
        mov  si,[bp+8]           ; ES% értékének beállítása.
        mov  ax,es
        mov  [si],ax
        pop  ax                  ; FLAGS% értékének beállítása.
        mov  si,[bp+6]
        mov  [si],ax
        pop  ax                  ; SI% értékének beállítása.
        mov  si,[bp+10]
        mov  [si],ax

        pop  es                  ; A GWBASIC szegmensregiszterek
        pop  ds                  ; visszaállítása.
        pop  bp

        ret  26

basmsz  endp

;-------------------------------------------------------------------------

mszbeall proc near               ; A megszakítás sorszámának mentése.

         pop  bx
         mov  cs:[bx+cim2-cim1+1],al
         jmp  cim1
```

```
mszbeall endp

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

kod     ends                    ; A (kód)szegmens vége.
        end                     ; Az assembler forrásprogram vége.
```

Azok számára, akik esetleg még nem tudnák, hogyan kell egy assembly nyelvű programot egy GW-BASIC programba beépíteni és meghívni, néhány magyarázatot fűzünk a programlistához.

A program először a verembe tárolja a bázismutatót, mert a következő utasítás megváltoztatja a tartalmát, az eredetire viszont a GW-BASIC-be való visszatéréshez ismét szükségünk lesz. A bázismutatóba most betöltődik a veremmutató értéke, hogy a veremben lévő adatokhoz hozzá lehessen férni. Erre azért van szükség, mert a GW-BASIC a CALL utasításban felsorolt változókat a veremben adja át a gépi kódú rutinnak. A következő lépésben a DS és az ES regiszter tartalma kerül a verembe, mert a rutin végrehajtásakor ezek `k a tartalma is megváltozhat, a GW-BASIC-hez való visszatéréshez viszont ismét csak az eredeti értékükre van szükségünk.

Most már megkezdheti a rutin az átadott változók beolvasását, és a tartalmuknak a processzor megfelelő regisztereibe való bemásolását. Fontos tudni azt, hogy a veremben nem az egyes változók tartalma, hanem a DS regiszterben lévő szegmenscímhez viszonyított, relatív címe van. Ezért a veremből először mindig az illető változó címét kell kiolvasni, majd az ezen a címen lévő értéket kell a megfelelő regiszterbe beírni. Az assembly rutin fejrészében feltüntettük, hogy az egyes változók címei a veremben mely címeken találhatók.

Elsőként a meghívandó megszakítás számát kell megállapítani. Ezt az értéket a veremben lévő, többi változótól eltérően kell kezelni, mert ennek az átadása nem regiszterben történik. A megszakítás száma ugyanis az INT utasítás részét alkotja, és ezért magába a programkódba, az INT utasítás CD(h) kódját tartalmazó tárcím után következő címre kell beíródnia. Ennek a címnek a megállapítása azonban némi problémát okoz: azt ugyanis nem írjuk elő, hogy a BASIC program a rutint az általa használt tárterület melyik részében helyezze el. Ebből következik, hogy az INT utasításkód tárbeli címe sem állandó, hanem a rutin mindenkori kezdőcímétől függ. A rutin viszont maga sem ismeri a kezdőcímét, következésképp az INT utasítás címét sem ismerheti. Ebből a látszólagos csapdából azonban viszonylag egyszerűen kijuthatunk: felhasználjuk azt a tényt, hogy a processzornak ismernie kell a rutin kezdőcímét, hiszen enélkül nem tudná végrehajtani. Ha tehát a rutinból meghívunk egy alprogramot, akkor a processzor automatikusan a verembe menti azt a címet, ahonnan a főprogramot folytatnia kell, így ezt a címet most már a rutinunk is könnyűszerrel megismerheti. Ezzel rendelkezésére áll egy fix pont a további címszámításhoz. Ezt a gondolatmenetet a program úgy valósítja meg, hogy a meghívandó megszakítás számát betölti az AX regiszterbe, majd meghívja az MSZBEALL alprogramot. Az alprogramra való ugrás előtt a processzor a verembe tölti azt a címet, amelytől kezdődően az alprogram befejeződése után a főprogram végrehajtását folytatnia kell. Programunkban ezt a címet a CIM1 címkével jelöltük meg. Az alprogram tehát ezt a címet veszi elő a veremből. Ennek a címnek az ismerete már elegendő ahhoz,

hogy az INT utasítás tárbeli címét követő címet kiszámíthassuk: tekintettel arra, hogy a CIM1 címkével jelölt tárcím és az INT utasítás tárbeli címe – ezt a CIM2 címkével jelöltük – közötti távolság mindig állandó, könnyen kiszámítható az a tárcím, amelyre a megszakítás számát be kell írni. A megszakítás számának beírásával befejeződik az alprogram és folytatódhat a főprogram végrehajtása (a CIM1 címkétől).

A rutin további része többé-kevésbé ismétlődő utasítássorozatokból áll, amelyek arra valók, hogy a különböző BASIC változók címét a veremből kiolvassák, és az ezeken a címeken talált értékeket a processzor megfelelő regisztereibe betöltsék. Csupán az ES regiszter tartalmát kell külön vizsgálni: ha a regiszterben –1 van, akkor e helyett a DS regiszter tartalma másolódik át az ES regiszterbe.

Miután valamennyi regiszterbe betöltődött a szükséges érték, a rutin meghívja a kívánt megszakítást, majd a regiszterek új tartalmát visszamásolja a verembe.

Befejezésül előveszi az indulásakor a verembe mentett regisztertartalmakat, és visszaadja a vezérlést a GW-BASIC-nek.

3.2. Megszakítások hívása Turbo Pascal nyelven

A PC-ken használt programozási nyelvek közül az egyik legszélesebb körben elterjedt nyelv a Turbo Pascal. Számos előnyös tulajdonsága közül itt most csak azt emeljük ki, amelyikkel a megszakítások meghívását támogatja.

A Turbo Pascalban egy megszakítás meghívására az INTR eljárás (procedure) szolgál. Az eljárás meghívásának alakja:

```
Intr (megszakítás_szám, regiszter);
```

A megszakítás_szám paraméternek egész (integer) típusú konstansnak kell lennie, és – amint a neve is mutatja –, a meghívandó megszakítás számát adja meg. Mivel ez a paraméter a 0 – 255 közötti bármelyik értéket felveheti, segítségével bármelyik megszakítás meghívható.

Az INTR eljárás egyik különleges alakja az MSDOS nevű eljárás, amellyel speciálisan a DOS funkciók hívhatók meg. Az eljárás meghívásának alakja:

```
MSDOS (regiszter);
```

Mivel ez az eljárás speciálisan a 21(h) számú DOS megszakítást hívja meg, magának a megszakításnak a számát nem is kell paraméterként átadni. Ezzel az eljárással az operációs rendszer szinte valamennyi funkciója meghívható.

A regiszter paraméter mindkét eljárásnál rekord típusú adatszerkezet, amely az átadandó regisztertartalmakat tárolja. Ezek az értékek a megszakítás meghívása előtt átmásolódnak a processzor megfelelő regisztereibe.

Ahhoz, hogy a regiszter paramétert rekord típusú adatszerkezetként létrehozzuk, előbb természetesen definiálni kell a megfelelő rekordot. Ezt a TYPE utasítással tehetjük meg:

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

Miután definiáltuk a regtip nevű rekord típust, a VAR utasítással létrehozhatjuk a regiszter paramétert:

```
var regiszter : regtip;
```

A regiszter nevű paraméter most a következő adatelemeket tartalmazza:

- regiszter.ax,
- regiszter.bx,
- regiszter.cx,

stb.

Ha most egy megszakítás meghívásához az AX regiszterbe például a 128-as értéket kell betölteni, akkor ez egy egyszerű értékadó utasítással elvégezhető:

```
regiszter.ax := 128;
```

Természetesen ugyanez vonatkozik az összes többi regiszterre is.

Ezen a módon viszont közvetlenül nem adhatók értékek a fél regisztereknek (AH, AL, BH, BL stb.), bár ez sokszor hasznos lenne. Ezen úgy segíthetünk, hogy nem teljes, 16 bites regisztereket definiálunk, hanem a 8 bites, fél regisztereket közönséges integer vagy bájt típusú változókként definiáljuk, majd ezeket egy egész regiszterré kapcsoljuk össze. Az AX regiszter esetén ez így valósítható meg:

```
var al,
    ah : integer;
regiszter.ax := ah shl 8 + al;
```

Ezzel az értékadó utasítással az AX regiszterbe beírjuk az AH regiszterben lévő érték 256-szorosának – a 8 helyértékkel történő balra léptetés 256-tal való szorzásnak felel meg – és az AL regiszter értékének az összegét. Ha egy programban gyakran kell a fél regisztereket egész regiszterekké összekapcsolni, akkor érdemes erre egy eljárást (függvényt) definiálni, amely ezt az összekapcsolást elvégzi:

```
function egeszregiszter (LO, HI : integer) : integer;
```

```
begin
    egeszregiszter := HI shl 8 + LO ;
end;
```

Az előbbiekben leírt összekapcsolás helyett most ez írható:

```
regiszter.ax : = egeszregiszter (al, ah);
```

Egy megszakítás meghívása előtt természetesen csak azokhoz a regiszterekhez kell értéket rendelni, amelyeket a meghívandó megszakítás fel is használ. A többi regiszter tartalma a megszakítási rutin végrehajtása szempontjából közömbös. A rutinból való visszatéréskor a rutin által esetlegesen szolgáltatott visszatérési értékek a regiszter változó egyes elemeiből olvashatók ki.

Ha például a megszakítás a visszatérésekor értéket ad vissza az AX regiszterben, akkor ehhez az értékhez a regiszter.ax változón keresztül lehet hozzáférni.

Ha egy egész regiszter fél regisztereinek értékét akarjuk olvasni, akkor ehhez az egész regisztert fel kell bontatni a felső és az alsó fél regisztereire:

```
ah := regiszter.ax shr 8 ;
al := regiszter.ax and 255 ;
```

A Turbo Pascal erre a célra már eleve beépített függvényeket tartalmaz. A hívásuk:

```
ah := Hi (regiszter.ax) ;
al := Lo (regiszter.ax) ;
```

Számos olyan megszakítás van, amelynél a végrehajtás sikerességét az állapotjelző (flag) regiszterben lévő átvitelbit (Carry flag) értéke mutatja. Ezért a programjainkban gyakran lehet szükség arra, hogy a megszakítás végrehajtását követően ennek – ill. esetenként valamelyik másik jelzőbitnek – az értékét megvizsgáljuk.

Egy regiszterben az egyes bitek értékét a logikai AND (ÉS) művelettel vizsgálhatjuk meg. Attól függően, hogy az illető bit értéke 1 vagy 0, a művelet eredménye logikai igaz (TRUE) vagy hamis (FALSE). A bitek a következő Pascal utasításokkal vizsgálhatók:

```
regiszter.flags and 1     átvitelbit (carry flag)
                          vizsgálata
regiszter.flags and 64    zéróbit (zero flag)
                          vizsgálata
regiszter.flags and 128   előjelbit (sign flag)
                          vizsgálata
```

Gyakran van szükség arra, hogy a megszakítás részére egy változó – többnyire egy karakterláncot tartalmazó puffer – kezdőcímét kell átadni. A Turbo Pascal erre az esetre is tartalmaz beépített függvényeket: az Ofs és a Seg függvények az argumentumként megadott változó ofszet-, ill. szegmenscímét szolgáltatják:

ofs (változónév)
seg (változónév)

A karakterláncokat tartalmazó pufferekkel kapcsolatban meg kell jegyezni, hogy a Turbo Pascal a karakterláncokat a DOS-tól és a BIOS-tól eltérő formátumban tárolja. Míg a DOS és a BIOS számára egy 00(h) bájt (a 0 ASCII-kód) vagy a $ karakter (a 36-os ASCII-kód) jelzi a karakterlánc végét (a karakterlánc kezdetét természetesen a megfelelő regiszterek tartalmazzák), addig a Turbo Pascal a puffer első bájtján a karakterlánc bájtokban számított hosszát tárolja, és ebből állapítja meg a karakterlánc végét. Ahhoz tehát, hogy a Turbo Pascalbeli karakterláncokat a DOS és a BIOS felhasználhassa, ezeket a megfelelő formátumra át kell alakítani. Az átalakítást a következő utasítások egyikével kell elvégezni:

```
karakterlánc := karakterlánc + #0;
karakterlánc := karakterlánc + #36;
```

(Arra, hogy mikor melyiket kell használni, az egyes megszakítások ismertetésénél külön kitérünk.) Az eltérő tárolási formátumból következik az is, hogy a megszakítás részére nem azt az ofszetcímet kell átadni, amelyiket az ofs függvény szolgáltat (ezen a címen ugyanis a karakterlánc hosszát tartalmazó bájt áll), hanem az ennél eggyel nagyobb címet.

Ennyi bevezető után lássuk a példaprogramot. Ez a program – csakúgy mint az előző, BASIC nyelvű program – a 21(h) DOS megszakítás 9-es funkciója segítségével egy adott karakterláncot ír a képernyőre.

```
{ ┌──────────────────────────────────────────────────────────┐ }
{ │                                                          │ }
{ │                      M S Z D E M O . P A S               │ }
{ │                                                          │ }
{ │  Funkciója : megszakításhívás szemléltetése karakterlánc │ }
{ │              megjelenítésével.                           │ }
{ │                                                          │ }
{ └──────────────────────────────────────────────────────────┘ }

program MSZDEMO;

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

var regiszter : dosreg;                    { Regiszter rekord deklaráció. }
    txt       : string[128];

{ ┌──────────────────────────────────┐ }
{ │ KAR.LÁNC MEGJELENÍTÉS MEGSZAKÍTÁSSAL │ }
{ └──────────────────────────────────┘ }

begin

   writeln;
   writeln;
   txt := '******** A megjelenítés a 21(h) megszakítás ' +
          '9-es funkciójával történt ********' + #36;

   regiszter.ax := $0900;                  { Kar.lánc megjel. funkció. }
   regiszter.dx := ofs(txt)+4;             { Kar.lánc ofszetcím.       }
   regiszter.ds := seg(txt);               { Kar.lánc szegmenscím.     }
   intr($21, regiszter);                   { Megszakításhívás.         }
   writeln;
   writeln;

end.
```

A program elején elvégezzük a szükséges típus és változó definiálásokat, majd a főprogramban a TXT nevű változóhoz hozzárendeljük a kiírandó szöveget. A karakterlánc utolsó karaktere a $ jel, amelyet a meghívandó DOS-funkció a karakterlánc lezárásaként vár (de ezt már nem írja ki).

A következő sor a meghívandó DOS-funkció számát (9) az AH regiszterbe tölti. Mivel a Turbo Pascal közvetlenül nem teszi lehetővé, hogy külön-külön forduljunk valamelyik fél regiszterhez (bár mint az előzőekben láttuk, programból ez is megtehető), a teljes AX regisztert használjuk. Az AL regiszterrészbe most 0-t töltünk, de ez az érték bármi más is lehetne, mivel ennek a meghívandó funkció szempontjából nincs jelentősége. A következő két sorban még a DS:DX regiszterpárnak átadjuk a TXT szegmens- és ofszetcímét, majd az MSDOS vagy INTR utasítással meghívjuk a 21(h) megszakítást.

3.3. Megszakítások hívása C nyelven

Az 1970-es évek közepén kifejlesztett C nyelv rövid idő alatt igen nagy karriert futott be. Elegendő talán csak annyit megemlíteni, hogy (jórészt) ezen a nyelven írták meg a ma már ipari szabványnak tekinthető UNIX, vagy az IBM új személyi számítógéprendszerein futó OS/2 operációs rendszert. A C nyelv népszerűségét elsősorban a viszonylag kicsi, de igen jól megválasztott utasításkészletének köszönheti. Ez tette lehetővé az olyan fordítóprogramok (Microsoft C, Turbo C) megírását, amelyek a C nyelv „hordozhatóságát", vagy más kifejezéssel a hardverfüggetlenségét biztosítják. Azt is mondják, hogy ez az a magasszintű nyelv, amely a legközelebb áll a hardverhez. A témánk szempontjából ez a legértékesebb tulajdonsága: a megszakítások meghívására mind a Microsoft C, mind a Borland Turbo C fordító (compiler) standard könyvtára egy sor, könnyen kezelhető funkciót bocsát a rendelkezésünkre.

A különböző megszakítások meghívásával kapcsolatban a következő funkciók tartanak számot az érdeklődésünkre:

int86
int86x
intdos
intdosx
segread

E funkciók és a hozzájuk tartozó adatstruktúrák a C fordítóprogram részét képező DOS.H nevű állományban vannak deklarálva. Ezért azokba a programokba, amelyek e funkciók valamelyikét meghívják, ezt az állományt az #include preprocesszor utasítással be kell szerkeszteni.

Ebben az állományban a regiszterértékek átadására három struktúra van definiálva: a WORDREGS, a BYTEREGS és a SEGREGS.

A WORDREGS az AX, BX, CX, DX, SI, DI egész regisztereket, valamint az állapotjelző regisztert (flags), a BYTEREGS az AH, AL, BH, BL, CH, CL, DH és DL fél regisztereket, a SEGREGS pedig a DS, CS, SS és ES szegmensregisztereket tartalmazza.

A BYTEREGS és a WORDREGS adatstruktúrák a REGS nevű adatstruktúrává vannak összekapcsolva (union REGS), ami a programozó számára lehetővé teszi, hogy szükség szerint fél vagy egész regiszterekkel dolgozzon.

Azt követően, hogy egy REGS típusú változót – nevezzük ezt az egyszerűség kedvéért REGISZTER-nek – a

union REGS regiszter;

utasítással létrehoztuk, az egyes regiszterekhez a következőképpen férhetünk hozzá:

AX: regiszter.x.ax
BX: regiszter.x.bx

AH: regiszter.h.ah
AL: regiszter.h.al
BH: regiszter.h.bh
stb.

Az átvitelbit (carry flag) értékét a regiszter.x.cflag nevű változó tartalmazza. Ha a változó értéke 0, akkor az átvitelbit értéke 0 (törölve van), ellenkező esetben a bit értéke 1 (be van kapcsolva).

A szegmensregisztereket reprezentáló változót – nevezzük ezt az egyszerűség kedvéért SZEGREGISZTER-nek – a következőképpen lehet létrehozni:

```
struct SREGS szegregiszter;
```

Ennek a változónak az egyes adatelemei, mint pl. a szegregiszter.ds, a szegregiszter.es stb. a processzor megfelelő szegmensregisztereinek tartalmát hordozzák.

A most elmondottakat a jobb megértés céljából majd egy példaprogramon elmagyarázzuk. Előbb azonban lássuk még az egyes funkciókat.

Az „int" szócskával kezdődő funkciók mindegyike megszakítások meghívására szolgál, míg a SEGREAD funkció a szegmensregiszterek tartalmát olvassa.

A megszakításokat meghívó funkciók az általános (tehát a nem szegmens) regiszterek körében szigorúan megkülönböztetik a beviteli (input) és a kiviteli (output) regisztereket. Ennek akkor van jelentősége, amikor a megszakítás az őt meghívó programnak ugyanazokban a regiszterekben ad vissza információkat, amelyekben a meghívásához szükséges információkat is kapta.

Mivel az egyes funkciók a meghívásukkor a regiszterekben nem magát a változót, hanem a változó címét kapják meg, a beviteli és a kiviteli regiszterek egyetlen, közös változóba is összefoghatók. Ilyenkor elegendő mind a beviteli, mind a kiviteli regiszter által hivatkozott változó címeként az egyetlen, közös változó címének a megadása (a példaprogramunkban mi is ezt az eljárást használjuk).

Az egyes megszakítások meghívása előtt a beviteli (vagyis a bemenő) változók tartalma bemásolódik a processzor megfelelő regisztereibe, majd a megszakítási rutin befejeződésekor ezek tartalma a kiviteli (vagyis a kimenő) változókba kerül. Valamennyi megszakítási funkció a visszatérésekor még az AX regiszter tartalmát is szolgáltatja. Lássuk most az egyes funkciókat és meghívásukat.

Az int86 funkció meghívása:

```
int86 (intszám, inregiszter, outregiszter);
```

Az intszám egy változó vagy egy állandó, amely a megszakítás számát adja meg, míg az inregiszter és az outregiszter kettő (vagy egy), REGS típusú változó címe. Már a változók nevéből is látszik, hogy az inregiszter a megszakítás meghívása előtti, az outregiszter pedig az megszakítás befejeződése utáni regiszterértékeket tartalmazza.

Ettől az int86x funkció annyiban tér el, hogy ennél harmadik argumentumként még egy SREGS típusú változó címét is át kell adni. Ennek a változónak a tartalma a

megszakítás meghívása előtt átmásolódik a szegmensregiszterekbe, a megszakítás befejeződésekor azonban már nem töltődik vissza. Ennek a funkciónak a meghívása:

int86x(intszám, inregiszter, outregiszter, szegregiszter);

Az intdos és az intdosx funkciók annyiban különböznek az előző kettőtől, hogy ezeknél a megszakítás számát nem kell átadni. Amint a nevükben levő dos mutatja, ezek a 21(h) megszakítást hívják meg, amelyen keresztül egyébként szinte valamennyi DOS funkció meghívható. Ezért az intdos funkció részére csak a processzorregisztereket reprezentáló beviteli és kiviteli változók címét kell átadni:

intdos (inregiszter, outregiszter);

Az intdosx funkciónál – csakúgy, mint az int86x-nél – még meg kell adni a szegmensregisztereket tartalmazó változó címét. A funkció hívása:

intdosx(inregiszter, outregiszter, szegregiszter);

Miután már tudjuk, hogyan kell egy megszakítást C nyelven meghívni, és ennek során hogyan változnak az egyes regiszterek, még azt kell megnéznünk, hogyan férhetünk hozzá egy változó címéhez. Mivel a C nyelvben sűrűn előfordul, hogy egy változó címét meg kell állapítani, a nyelv erre a célra egy egyszerű eljárást bocsát a rendelkezésünkre. A nyelvben definiálták a & címoperátort, amellyel tetszőleges változó ofszetcíme lekérdezhető.

A szegmenscím megállapításához a már említett SEGREAD funkciót használjuk. Ennek a funkciónak átadásra kerül egy SREGS típusú változó címe (aminek a megállapítása természetesen szintén a & címoperátor segítségével történik), majd a változóba bemásolódik a szegmensregiszterek tartalma. Ha a funkció részére például annak a szegregiszter változónak a címét adjuk át, amelyet előzőleg a

struct SREGS szegregiszter;

utasítással definiáltunk, akkor a SEGREAD funkció meghívása után a szegregiszter.ds a keresett változó szegmenscímét tartalmazza.

Míg tehát a C nyelv számos funkcióval támogatja a megszakítások meghívását, a Microsoft C fordítóprogram könyvtára a Borland Turbo C könyvtárával ellentétben sajnos nem tartalmaz olyan műveletet, amely segítségével egy tárhely tartalmát kiolvashatnánk. Mivel egy ilyen művelet számos programban igen hasznos lehet, a következőkben bemutatunk egy assembly programot, amely két, PEEKB-nek és POKEB-nek nevezett műveletet végez el, és amely beszerkeszthető a Microsoft C fordítóval és szerkesztővel készített programokba. A PEEKB művelet eredménye egy tárhely tartalma (1 bájt), míg a POKEB művelettel egy adott tárhelyre egy értéket (1 bájtot) lehet beírni.

Annak, aki Borland Turbo C-vel dolgozik, erre a rutinra nincs szüksége, mert – mint említettük –, ennek a standard könyvtára már tartalmazza a PEEK, PEEKB, POKE és POKEB műveleteket. Azok viszont, akik Microsoft C-vel dolgoznak, először egy szö-

vegszerkesztővel írják egy állományba az itt következő assembly listát, tárolják PEEK-POKE.ASM néven, hívják meg a Microsoft Macro-assemblerét, majd a

masm peekpoke;

utasítással fordítsák le.

```
;
;   ╔═══════════════════════════════════════════════════════╗
;   ║                                                       ║
;   ║                  P E E K P O K E . A S M              ║
;   ║                                                       ║
;   ║  Funkciója : az MSC mintaprogramok számára a PEEKB és POKEB  ║
;   ║              memóriakezelő eljárások definiálása.     ║
;   ║                                                       ║
;   ╚═══════════════════════════════════════════════════════╝
;

;— DEFINíCIÓK ————————————————————————————————————————————

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

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

        public _PEEKB                   ; PEEKB és POKEB kívülről
        public _POKEB                   ; is hívható eljárások.

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

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

_DATA   segment word public 'DATA'      ; Inicializált globális és
_DATA   ends                            ; statikus változók szegmense.

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

;— PEEKB : A memória egy bájtjának kiolvasása ———————————————
;— Hívás MSC-ből : int(eredmény) = PEEKB(int szegmens, int ofszet)

_PEEKB  proc near

        push bp                 ; BP -> verem.
        mov  bp,sp              ; SP -> BP.
        push ds                 ; DS -> verem.
        mov  ax,[bp]+4          ; Első param. : szegmenscím átvétele
        mov  ds,ax              ;               és beállítása.
        mov  bx,[bp]+6          ; Második param. : ofszetcím.
        mov  al,[bx]            ; A cella kiolvasása.
        xor  ah,ah              ; Megszakítás HI-bájt 0.
        jmp  short vege         ; Ugrás a VEGE címkére.

_PEEKB  endp

;— POKEB : Közvetlen írás a memória egy bájtjába ————————————
;— Hívás MSC-ből : POKEB(int szegmens, int ofszet, short int érték)

_POKEB  proc near

        push bp                 ; BP -> verem.
        mov  bp,sp              ; SP -> BP.
        push ds                 ; DS -> verem.
        mov  ax,[bp]+4          ; Első param. : szegmenscím átvétele
        mov  ds,ax              ;               és beállítása.
        mov  bx,[bp]+6          ; Második param. : ofszetcím.
        mov  al,[bp]+8          ; Harmadik param. : a beírandó érték.
        mov  [bx],al            ; Az érték beírása a cellába.
```

```
vege:   pop  ds                 ; Verem -> DS.
        mov  sp,bp              ; Veremtármutató visszaállítása.
        pop  bp                 ; verem -> BP.
        ret                     ; VÉGE.

_POKEB  endp

;-- VÉGE ------------------------------------------------------------

_TEXT   ends                    ; Programszegmens vége.
        end                     ; Assembler forráskód vége.
```

A fejezethez tartozó példaprogramban a most bemutatott műveletet arra használjuk fel, hogy segítségével kiolvassuk a PC típusazonosító bájtját, majd a típus nevét egy DOS funkció meghívásával kiírjuk a képernyőre.

```
/*
/*                                                              */
/*                          M S Z D E M O . C                   */
/*                                                              */
/*   Funkciója : megszakítás hívás szemléltetése a számítógép   */
/*               típusának megjelenítésén keresztül.            */
/*                                                              */
/*                                                              */

#include <dos.h>                        /* Megszakítás híváshoz      */
                                        /* header állomány.          */
extern short int PEEKB();               /* PEEKB hozzászerkesztendő. */

/*
/*  SZ.GÉP TÍPUS MEGJELENÍTÉS MEGSZAKÍTÁSSAL                    */
/*                                                              */

void main()

{
static char AT[] = "A számítógép AT$";
static char XT[] = "A számítógép XT$";
static char PC[] = "A számítógép PC$";

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

regiszter.h.ah = 9;                     /* Kar.lánc megjelenítés funkció. */
switch (PEEKB(0xF000, 0xFFFE))          /* A géptípus kiolvasása.       */
   {
   case 0xFE : regiszter.x.dx = (int) XT;  /* A típusnak megfelelő      */
               break;                      /* szöveg címének betöl-     */
   case 0xFC : regiszter.x.dx = (int) AT;  /* tése DX-be.               */
               break;
   case 0xFF :
   default   : regiszter.x.dx = (int) PC;
   }
intdos(&regiszter, &regiszter);         /* 21(h) megszakítás hívása.    */
}
```

A főprogramban először három CHAR vektort definiálunk, amelyek az egyes PC-típusokra vonatkozó szöveget tartalmazzák. Mindegyik szöveg elején és végén két „\n" jel van, aminek hatására a szöveg elé és mögé két üres sor íródik.

A főprogram első utasításával a karakterlánc képernyőre írását végző DOS funkció számát (9) betöltjük az AH regiszterbe, ahol azt a 21(h) megszakítás a meghívásakor várja. Ezután az általunk definiált PEEKB művelettel kiolvassuk az F000:FFFE tárcímen lévő bájt értékét. A kapott értéktől függően a megfelelő szöveg ofszetcímét beírjuk a DX regiszterbe, ahol azt a hívandó funkció várja.

Az ofszetcím mellett még a szöveg szegmenscímét is át kell adni a DS regiszterben. Mivel azonban ezt a fordítóprogram automatikusan beállítja, nekünk ezzel nem kell törődnünk. Utolsó utasításként meghívjuk az INTDOS funkciót, amely a részéről meghívja a 21(h) megszakítást az általunk megadott regisztertartalmakkal.

A programot először természetesen le kell fordítani a Microsoft C fordítóval, majd a következő lépésben el kell végezni a szerkesztést. Fontos, hogy a programunkba beszerkesszük az előbb assemblált PEEKPOKE programot, hogy az így előállított program a PEEKB és POKEB műveleteket is tartalmazza, és ezeket a C programunk használhassa.

A fejezet lezárásaként még egy rövid megjegyzés a Borland Turbo C fordítóprogramhoz: a könyv több olyan példaprogramot tartalmaz, amelyekbe assembly nyelvű programokat kell beszerkeszteni. Mivel a Turbo C a Microsoft C-től eltérően a funkciónevek írásánál nagyon kényes a nagybetűs, ill. kisbetűs írásmódra, ebből problémák adódhatnak. Ennek elkerülése érdekében a programok készítésekor a Turbo C parancssorából meg kell hívni az OPTIONS, majd ezen belül a LINKER menüt. Az ekkor megnyíló ablak legalsó sora tartalmazza a „Case-sensitive link" opciót. Ha erre (a RETURN billentyű lenyomásával) az „Off" esetet választjuk ki, akkor elkerülhetjük a nagybetűs/kisbetűs írásmódból adódó nehézségeket.

4. MEGSZAKÍTÁSOK HÍVÁSA ASSEMBLY NYELVEN

A magas szintű nyelveken programozókkal szemben az assembly nyelv használóinak nem kell bonyolult funkciókhoz vagy eljárásokhoz fordulniuk ahhoz, hogy egy megszakítást meghívjanak. Ehhez elegendő, ha a megszakítás által várt paramétereket betöltik a megfelelő regiszterekbe, majd az INT utasítással meghívják az illető megszakítást.

Bizonyos megszakításokat, ill. a megszakítások mögött meghúzódó rendszerrutinokat sok programban igen gyakran kell meghívni. Ilyen például a 21(h) DOS megszakítás 9-es funkciója, amely egy szöveget ír a képernyőre. A meghívásához mindössze annyi a teendő, hogy a funkció számát az AH regiszterbe, a szöveg szegmens- és ofszetcímét pedig a DS:DX regiszterpárba töltjük (a szegmenscím külön betöltésére általában nincs is szükség). Assembly nyelven ez így írható meg:

```
mov     ah,9                ;a 9-es funkciószám betöltése
mov     dx, offset szoveg   ;a szöveg ofszetcímének betöltése
int     21h                 ;21(h) DOS megszakítás hívása
```

A programozástechnikában általánosan használt gyakorlat, hogy a sokszor ismétlődő programrészeket csak egyszer írják meg alprogramként, amelyet aztán a főprogram tetszőleges helyről, bármikor meghívhat. Ennek nyilvánvaló előnye a rövidebb programkód és a kisebb tárigény. A fenti példa, amellyel csak egy megszakítás meghívását akartuk szemléltetni, a maga mindössze három sorával – bármilyen gyakran is kell ezeket végrehajtani – nem „érdemli ki" egy alprogram „rangját". A kiírandó szöveg ofszetcímét úgyis minden alkalommal át kellene adni az alprogramnak, továbbá ezt meg is kellene hívni, úgyhogy igazából nem sokat nyernénk. Ami miatt ezt mégis szóba hoztuk, annak tulajdonképpen az a célja, hogy ennek kapcsán röviden megismerkedjünk az assemblerek makró hívási lehetőségeivel.

4.1. Makrók hívása assemblerben

A makrók olyan, egy vagy több assembly utasításból álló utasítássorozatok, amelyeket a programból a programozó által megválasztott néven lehet hívni. Céljuk és feladatuk lényegében azonos, mint az eljárásoké (szubrutinoké) – ismétlődő programrészek többszöri végrehajtása – , de a definiálásuk és végrehajtásuk ezektől jelentősen eltér. Abban megegyeznek, hogy a programban mindegyiket csak egyszer kell megírni. Lényegesen különbözik viszont a végrehajtásuk: amikor a program a végrehajtása során egy eljárásdefinícióhoz ér, akkor a futását az eljárás kezdőcímén folytatja, majd az eljárás befejeződésekor visszatér az ugrás előtti címre. Makrók esetén nincs vezérlésátadás: amikor az assembler fordító a lefordítandó programban egy makró nevével találkozik, akkor az első menetben a makrót bejegyzi egy táblázatba, majd a második menetben a makróban lévő utasításokat (a makró törzsét) az átadott paraméterekkel bemásolja a program forrásszövegébe (ezt nevezik a makró kifejtésének), és befordítja a tárgykódba. Ebből következik, hogy a makrók a tárgykód hosszát nem csökkentik, viszont az alkalmazásukkal programozói munka takarítható meg, a paraméterezhetőségük pedig lehetővé teszi rugalmas használatukat.

A makrók definiálását és meghívását a Microsoft makroassembler (MASM) fordítóprogramján keresztül mutatjuk be. Mivel a példabeli makrónk egy karakterláncot ír a képernyőre, legyen a neve kiir:

```
kiir macro karlanc      ;a makró feje névvel és
                        ;paraméterrel
mov ah,9                ;9-es funkciószám betöltése
mov dx,offset karlanc   ;karlánc ofszetcímének
                        ;betöltése
int 21h                 ;21(h) DOS megszakítás
                        ;meghívása
endm                    ;a makró lezárása
```

A vizsgálataink szempontjából az első programsor a legfontosabb. Ez a sor tartalmazza a makró nevét, ami alapján az assembler a makrót azonosítja. Ezt követi – előírásszerűen – a macro utasításszó, amely közli az assemblerrel, hogy az előző név egy makrót definiál. Ezt az utasításszót egy vagy több paraméter követi, amelyek jelentésére mindjárt kitérünk.

A makró definíciós sora után következnek azok a programsorok, amelyek a makró meghívásakor a tárgykódba befordítódnak. A makró törzsét kötelezően az endm utasításszó zárja le.

Ha egy programon belül a makró segítségével egy karakterláncot akarunk kiíratni, akkor a programba a

```
kiir szoveg
```

sort kell beírni, ahol a szoveg annak a változónak a neve, amelyik a kiírandó karakterláncot tartalmazza. A makró definíciós sorában megadott karlanc paraméter a makró

meghívásakor átveszi a szoveg változóval megadott karakterláncot. A makró kifejtése az alábbi programsorokat eredményezi:

```
mov ah,9
mov dx,offset szoveg
int 21h
```

Ezek után lássunk most egy rövid programot, amely az imént ismertetett makrót használja:

```
;
;
;                    M A K D E M O . A S M
;
;   Funkciója : makróhívás szemléltetése. A makró egy megszakítás
;               segítségével egy karakterláncot jelenít meg.
;
;
;
;— MAKRÓ ————————————————————————————————————————————————————————

kiir    macro karlanc                   ; A makró feje névvel és paraméter-
                                        ; rel.

        mov ah,9                        ; Karakterláncot megjelenítő
        mov dx,offset karlanc           ; funkció és megszakítás hívása.
        int 21h

        endm                            ; A makró lezárása.

;— KÓDSZEGMENS ——————————————————————————————————————————————————

kod     segment para 'CODE'             ; A kódszegmens definiálása.

        assume cs:kod, ds:adat, ss:verem

                                        ; Szegmensregiszterek hozzárendelése
                                        ; a szegmensekhez.

prog    proc far                        ; Itt kezdődik a tulajdonképpeni
                                        ; főprogram. A program indításakor
                                        ; ide kerül a vezérlés.

        mov  ax,adat                    ; Adatszegmens cím --> DS.
        mov  ds,ax

        kiir szoveg                     ; A definiált makró meghívása.

        mov  ax,4C00h                   ; 21(h) megszakítás, a program a 0
        int  21h                        ; hibakóddal ér véget.

prog    endp                            ; A prog eljárás vége.

kod     ends                            ; A kódszegmens vége.

;— KÓDSZEGMENS VÉGE ————————————————————————————————————————————

;— ADATSZEGMENS ————————————————————————————————————————————————

adat    segment para 'DATA'             ; Az adatszegmens definiálása.

CR      equ 13                          ; Kocsivissza ASCII kódja.
LF      equ 10                          ; Soremelés ASCII kódja.
KEND    equ "$"                         ; Karakterlánc vége jelzés.

szoveg  db CR,LF,"Igy alkalmazható egy makró !",CR,LF,KEND

adat    ends                            ; Az adatszegmens vége.

;— ADATSZEGMENS VÉGE ———————————————————————————————————————————

;— VEREMSZEGMENS ————————————————————————————————————————————————
```

```
verem    segment para stack        ; A veremszegmens definiálása.
         dw 64 dup (?)             ; A verem hossza 64 szó.
verem    ends                      ; A veremszegmens vége.
;— VEREMSZEGMENS VÉGE ————————————————————————————————
;— VÉGE ———————————————————————————————————————————————
end      prog                      ; Az assembler forrásprogram vége.
                                   ; A program végrehajtása a  prog
                                   ; eljárással kezdődik.
```

Miután egy szövegszerkesztő segítésével a programot gépbe írtuk és – pl. MAKDEMO néven, ASM kiterjesztéssel – lemezen tároltuk, a MASM makroassemblerrel lefordíthatjuk, szerkeszthetjük, majd az így kapott futtatható programot végrehajthatjuk.

A lista legtöbb sora assemblernek szóló definíció és deklaráció, amelyeknek a tulajdonképpeni programhoz nincs közvetlen közük. Először definiáljuk a makrót, kiír néven. Ezután a kódszegmenst definiáljuk, amelybe a végrehajtandó programutasítások kerülnek. A tényleges program mindössze öt sorból áll. Ezek közül az első kettő csak a program inicializálását végzi: az adatszegmens kezdőcímét betölti a DS szegmensregiszterbe, hogy a szegmensben lévő adatokhoz hozzá lehessen férni. Ezután hívjuk a makrónkat, és átadjuk részére a szöveg nevű paramétert. A programot lezáró két utasítás – ugyancsak egy DOS funkció hívásával – befejezi a programot.

Ezután az adatszegmens definiálása következik. Ebben először néhány konstanst definiálunk, amelyek a kiírandó karakterlánc olvashatóságát könnyítik meg, majd szöveg néven elhelyezzük a kiírandó karakterláncot. A karakterlánc elé és mögé beszúrunk egy-egy kocsi vissza (CR = Carriage Return) és soremelés (LF = Line Feed) vezérlőkaraktert, hogy a szöveg új sorba íródjon, és mind előtte, mind utána egy üres sor álljon. A karakterláncot egy $ jel zárja le, mert a meghívandó DOS funkció erről ismeri fel a karakterlánc végét.

Utolsóként a veremszegmenst definiáljuk, ahová a program végrehajtása közben a verem kerül. Mivel a programunk nem mondható túlságosan nagynak, veremterületként 64 szó bőven elegendő.

Végezetül ismét egy, a fordítóprogramnak szóló utasítás zárja le a forrásprogramot, és meghatározza a program végrehajtásának kezdetét.

Az itt bemutatott makróhoz hasonlóan számos más, sűrűn ismétlődő feladathoz írhatók kisebb-nagyobb makrók, amelyeket a gyakorlati munkában igen jól fel lehet használni. Egy idő után ezeket a makrókat az egyéni szükségleteknek megfelelően önálló állományba, ún. makró könyvtárba össze is lehet fogni. Ha a programunkban használni akarunk egy makrót ebből az állományból, akkor az állományt az INCLUDE utasítással beszerkeszthetjük a programba. Azzal, hogy ez az állomány esetleg olyan makrókat is tartalmaz, amelyeket a programunk nem használ fel, nem pocsékolunk tárhelyet. Az állományból csak azok a makrók épülnek be a programkódba, amelyeket a program a futása során meghív.

II. rész

A DOS

5. A RENDSZERPROGRAMOZÁSRÓL ÁLTALÁBAN

Úgy gondoljuk, joggal feltételezzük, hogy a számítógéppel valamilyen – akár felhasználói, akár hobbi programozói minőségben – kapcsolatba került emberek többsége találkozott már a rendszerprogramozás, ill. a rendszerprogramok kifejezéssel. Közülük sokan bizonyára közelebbi ismeretségbe is kerültek velük, míg mások szemében ezeket a fogalmakat valószínűleg továbbra is valamilyen misztikus köd borítja. Az utóbbiak azt is gondolhatják, hogy a számítógépes programozásnak ez már az az „ingoványos" területe, ahová csak a „bennfentesek" merészkedhetnek be, és a kívülállók számára nem marad más, mint továbbra is „gyalogosan" programozni, és eközben titkolt irigységgel a háttérből figyelni a „varázslókat".

Nos, a dologban van valami: lehet programozni, sőt egészen jó programokat is készíteni anélkül, hogy a programozónak akár csak sejtelme lenne a rendszerprogramok mibenlétéről, vagy egyáltalán a létezésükről. Másrészről viszont hatékony, gyors és rugalmas programok igazából csak a hardver, ill. az operációs rendszer által a programozó rendelkezésére bocsátott szolgáltatások lehetséges legteljesebb körű igénybevételével írhatók. Ehhez már nélkülözhetetlen a rendszerprogramokkal kötendő szoros ismeretség.

Nagyon reméljük, hogy azok számára, akik ezt a könyvet áttanulmányozzák, ez az ismerkedés nem lesz túlságosan nehéz, és a befektetett munka meghozza a gyümölcsét.

Azoknak, akik a rendszerprogramozásról eddig még csak hallottak, de magát a komplett számítógépes rendszert – amely alapvetően a számítógépet mint hardvert, és a rajta futó különböző pogramokat mint szoftvert foglalja magába – esetleg nem tudják egy egységes rendszerként áttekinteni, felvázolunk egy analógiát. Annak ellenére, hogy minden hasonlat sántít (ez alól a miénk sem kivétel) reméljük, hogy az alábbi gondolattársítás segíti az összefüggések megértését.

Képzeljünk magunk elé egy szállodaépületet, teljes berendezésével és összes tárgyi felszerelésével együtt. Fontos hangsúlyozni, hogy csak tárgyakról van szó: a szállóban egy lélek sincs – se személyzet, se vendég. Egy számítógépes rendszerben ilyen szállodának tekinthető maga a számítógép, azaz a hardver. Legyen ez a gondolatmenetünk kiindulópontja, és keressük meg a további hasonlóságokat: a szállodaépítők felépítették a szállodát, a számítógép lekerült a szerelőszalagról, vagyis elkészült valami kerek egész. Most már csak egy baj van: így, ahogy van, sem a szálló, sem a gép nem használható: nincs, aki kinyitná a szálloda ajtaját, és nincs aki (ami) „szólásra" bírná a számítógépet. Enélkül pedig nincs vendég, és nincs programfeldolgozás.

Fordítsuk meg a gondolatmenetünk irányát, és induljunk el a vendég, ill. a programfeldolgozás oldaláról. Ahhoz, hogy egy vendég egy szállóban éjszakai nyugodalmat találjon – erős leegyszerűsítéssel – természetesen kell egy szálló, amelyben fekhely van, és egy személyzet, amely megmutatja, hogy az a fekhely hol található. Ugyanez „számítógépül": ahhoz, hogy egy számítógépes programot végre lehessen hajtani, természetesen kell egy számítógép, és kell egy olyan program, amely a végrehajtandó programnak a gépen való eligazodását segíti.

Úgy gondoljuk, az előbbi gondolattársításból már világos a rendszerprogram(ok)nak egy számítógépes rendszeren belül elfoglalt helye és szerepe: amilyen feladatot a szállodai személyzet a szálló (mint épület, tárgy) és a vendég (mint a szálló használója) közötti kapcsolatban ellát, olyan szerepet játszanak a rendszerprogramok a számítógép (mint hardver) és a végrehajtandó programok közötti kapcsolatban.

Ha rendszerprogramokról beszélünk, akkor ezeket általában mindig többesszámban említjük. Ennek az az oka, hogy a számítógépeken egyidejűleg két, egymástól eléggé jól megkülönböztethető rendszerprogram dolgozik: a DOS és a BIOS. Tekintettel arra, hogy az egész könyv alapvető célja ezeknek a rendszerprogramoknak az ismertetése, itt most csak annyiban térünk ki rájuk, amennyiben a szerepük és feladatuk megértését az előző – szállodai – gondolattársítás elősegítheti.

A szállodai személyzetet gondolatban – és meglehetősen önkényesen – osszuk két csoportba. Az első csoportba tartozzanak a személyzet azon tagjai, akik alapvetően a vendégekkel, míg a másodikba azok, akik elsősorban a külvilággal állnak kapcsolatban. Az egyszerűség kedvéért az első csoportot nevezzük portaszolgálatnak, a másodikat pedig háttérszolgálatnak. Ha gondolatban elvégeztük ezt a felosztást, akkor – merész leegyszerűsítéssel – a portaszolgálatnak a DOS-t, a háttérszolgálatnak pedig a BIOS-t feleltethetjük meg. Nézzük meg, mit csinál a portaszolgálat (azaz a DOS), ha egy vendég (végrehajtandó program) érkezik. Először is megnézi, hogy a szállodában van-e üres szoba, ha van, hány napra és hol, ha pedig nincs, akkor mikor lesz. Ha talál szobát, átadja a kulcsot, és a vendéget bejegyzi a nyilvántartásába. Ez alapján bármikor meg tudja mondani, hogy éppen mely szobák foglaltak, azokban kik laknak, meddig maradnak stb. Általában elmondható, hogy – kisebb szállodát feltételezve – a vendég minden kérésével a portaszolgálathoz fordul, amely azokat a lehetőségekhez képest teljesíti is. A portaszolgálattal szoros összhangban, de ettől mégis eltérő feladatokat lát el a háttérszolgálat (azaz a BIOS). Ezzel a szolgálattal a vendég közvetlenül szinte sohasem találkozik, de a kényelméhez nélkülözhetetlen: ez a szolgálat tartja és szervezi a szálloda és a külvilág közötti kapcsolatokat. Ha például a vendég telefonon virágot rendel, akkor – még ha a telefont a portaszolgálat is veszi fel – ez az a szolgálat, amely elmegy a virágboltba, megveszi és elhozza a virágot. A példák sorát hosszan lehetne folytatni (asztalrendelés, színházjegy- vagy taxirendelés stb.). A hangsúly azon van, hogy ez az a szolgálat (vagyis a BIOS rendszerprogram), amelyik – többnyire a portaszolgálaton (DOS) keresztül – a vendég (a program) igénye szerinti külső kapcsolatokat (billentyűzet, képernyő, lemezmeghajtó, nyomtató stb.) szervezi.

A fenti okoskodással kapcsolatban felmerülhet a kérdés: ugyan mit érdekli a vendéget, hogy mi és hogyan is történik valami egy szállodában? Másképpen fogalmazva: miért jó az, ha ismerjük a rendszerprogramozás rejtelmeit? A válaszunkban ismét hasz-

náljuk a hasonlatunkat: azért jó, hogy olyan programokat tudjunk írni, amelyek úgy tudnak „viselkedni", mint egy profi világutazó: a világ bármely szállodájában otthon érzi magát (pénzről ne essék szó), a portással azonnal szót ért (van közös nyelvük), és képes arra, hogy a nyújtott szolgáltatásokat a legteljesebb mértékben és természetességgel a saját kényelmére használja.

6. A DOS-RÓL ÁLTALÁBAN

A DOS az angol Disk Operating System kifejezés rövidítése, amelyet magyarul lemezes operációs rendszernek lehet mondani. A lemezes jelző itt azt jelenti, hogy az operációs rendszert mágneslemez tárolja (eltérően pl. a Commodore 64-estől, ahol az operációs rendszer magába a gépbe, ROM-ba van beépítve), és ezt a rendszer minden egyes indításakor ismételten be kell tölteni a gép belső tárába.

Amint a bevezetőben említettük, feltételezzük, hogy az olvasó ismeri az alapvető DOS parancsokat és használatukat. Itt olyanokra gondoltunk, mint pl. a DIR, COPY vagy a FORMAT. Míg azonban a felhasználók többségének elegendő, ha e parancsok működéséből csak annyit értenek, amennyi belőlük „szabad szemmel" látható (vagyis ha beírják a DIR parancsot, akkor a képernyőn megjelenik az adott lemez tartalomjegyzéke), addig mi ebben a könyvben ezeket a parancsokat mikroszkóp alá helyezzük, és ízekre szedjük (megtudjuk például, hogy a DIR parancs honnan „tudja meg", milyen méretű egy állomány). Lemerülünk a DOS felszíne alá, és megvizsgáljuk, milyen folyamatok játszódnak le az operációs rendszer alsóbb régióiban. Megismerkedünk egy sor olyan rutinnal (ezeket funkcióknak is nevezzük), amelyeket a DOS a munkája során használ, ugyanakkor kellő ismeretek birtokában ezeket mi is hasznosíthatjuk. Felfedező útunk során azt a célt állítottuk a középpontba, hogy olyan rutinokat ismerjünk meg, amelyeket a programozói munkánkban közvetlenül felhasználhatunk. Ezáltal sok felesleges munkától kímélhetjük meg magunkat, programjaink pedig rövidebbek és hatékonyabbak lesznek. Megismerkedünk továbbá azokkal az adatstruktúrákkal, amelyek a DOS különböző funkciói között az összekötő kapcsot képezik, és amelyek lehetővé teszik pl. a hajlékony- vagy a merevlemezes meghajtók kezelését.

Előbb azonban röviden tekintsük át a DOS keletkezésének és fejlődésének történetét, amelynek során több, a későbbiekben felmerülő kérdésre is választ kaphatunk.

6.1. A DOS keletkezésének és fejlődésének története

A kezdet 1980-ra nyúlik vissza, tehát arra, az időre, amikor a piacot a mikroszámítógépek területén az Intel 8080-as és a Zilog Z80-as mikroprocesszorára épült 8 bites gépek, és ezek operációs rendszere, a Digital Research által fejlesztett CP/M-80-as uralták. Számos alkalmazói program is létezett már, amelyek e rendszer alatt futottak. Két évvel korábban, 1978-ban jelent meg az Intel első, 16 bites mikroprocesszora (8086), amelyre épülve rövidesen megjelentek az első generációs, 16 bites mikroszámítógépek is.

1980 közepén Tim Paterson, a Seattle Computer Products programozója megírta a 86-DOS névre keresztelt operációs rendszert, amely a jelenlegi MS-DOS, ill. PC-DOS ősének tekinthető.

Mivel abban az időben már számos olyan felhasználói szoftver volt a piacon, amelyek a CP/M alatt futottak, az új operációs rendszer kifejlesztésénél alapvető szempont volt, hogy a meglévő programok lehetőség szerint változtatás nélkül, vagy csak kismértékű átalakítással futhassanak az új, 16 bites környezetben is. Ezért Tim Paterson az új operációs rendszerbe javított formában átvette a CP/M számos funkcióját és adatstruktúráját (állományvezérlő blokk, program szegmens prefix, végrehajtható állományok).

Időközben az IBM is elkészítette a 16 bites számítógépcsaládjának első típusait, és felkérte az akkori legnagyobb mikroszámítógépes szoftverházat, a Microsoft Corp. céget, hogy új gépcsaládja számára alkalmas operációs rendszert keressen. Mivel a Microsoftnak ilyen célra nem volt saját operációs rendszere, megvásárolta a Seattle Computer Products-tól a Paterson által kifejlesztett operációs rendszert, azt jelentős mértékben megváltoztatta, és elnevezte MS-DOS-nak. Amikor 1981 őszén megjelent a piacon az első IBM PC, az IBM a gép elsődleges operációs rendszereként már az MS-DOS-t ajánlotta (akkor még PC-DOS 1.0 volt a neve).

Ezen első verzió megjelenése óta számos lényeges változtatást hajtottak végre a DOS-ban. Mivel azonban a DOS-programozók számára éppen ezek a változtatások lehetnek érdekesek, a DOS fejlődése során végrehajtott minden fontosabb újításnak külön fejezetet szentelünk, bemutatva a korábbi verziókhoz képesti jelentősebb eltéréseket. Így nyomon követhetjük a fejlődés legfontosabb állomásait, és első megközelítésben képet alkothatunk az operációs rendszer bonyolultságáról.

Az 1.0 verzió

A Microsoft számára az 1.0 verzió lényegében még kompromisszum volt, mivel ez a már meglévő programok hordozhatósága céljából erősen támaszkodott a CP/M-80-ra. Ez pl. abban is kifejeződött, hogy egy állomány azonosítása (8 karakter az állomány nevére, 3 karakter a kiterjesztésre) ugyanúgy történt, mint a CP/M-80-nál. De a lemezmeghajtók jelölése vagy a teljes belső struktúra tekintetében is igen sok párhuzamosságot mutatott a sikeres, 8 bites rendszerrel. Mindezek ellenére a DOS 1.0 verzió már igen sok újdonságot is tartalmazott a CP/M-80-hoz képest.

Mivel a hardver vonatkozásában abban az időben még igen sok újdonsággal és továbbfejlesztéssel lehetett számolni (nagyobb és olcsóbb RAM-tárak, gyorsabb lemezmeghajtók), a Microsoft úgy határozott, hogy a DOS-t nagymértékben függetleníti a hardvertől. Ezért először a lemezkezelésnél megszüntette a fizikai és a logikai adathosszak egymáshoz való kapcsolását.

A CP/M-80 alatt minden lemez 128 bájtos egységekre volt felosztva, és ezekhez az egységekhez mindig csak egy „darabban" lehetett hozzáférni. A rendszer nem tette lehetővé, hogy a lemezen csak egyes, meghatározott bájtokat érjünk el, ami felesleges programozási munkát és időt jelentett. Ezt a hátrányt a DOS megszüntette azzal, hogy a logikai és a fizikai adathosszakat egymástól szétválasztotta. A rendszerbe továbbá néhány olyan funkciót is beépítettek, amelyek lehetővé tették, hogy egy állomány ne mindig csak egy, hanem egyidejűleg több rekordját is lemezre lehessen írni, ill. onnan beolvasni. Az is növelte a hardverfüggetlenséget, hogy a rendszer a beviteli és a kiviteli egységeket állományokként kezelte. Ebből a célból ezek az eszközök saját azonosító nevet kaptak:

CON (billentyűzet és képernyő)
PRN (nyomtató)
AUX (soros interfész)

Ha tehát egy DOS-rutinon keresztül egy állományhoz való hozzáféréskor nem egy állománynevet adtak meg, hanem a fenti három eszköznév valamelyikét, akkor ez azt jelentette, hogy a művelet nem az állományt tartalmazó lemezmeghajtóra, hanem a megadott eszközre vonatkozott. Így tehát az adatmozgatást a billentyűzetről, ill. a képernyőről át lehetett irányítani egy állományba vagy egy másik eszközre.

Korábban a DOS csak olyan programállományokat támogatott, amelyeket a számítógép fő tárában csak egy előre meghatározott helyre lehetett betölteni, és ott végrehajtani. Mivel ez a CP/M-es időkből származó örökség már nem bizonyult megfelelőnek, bevezettek egy új programállomány típust. Megkülönböztetésül a korábbi, COM kiterjesztésű állományoktól ezeket az új típusú állományokat az EXE (executable) kiterjesztéssel jelölték. Ezeket az állományokat a számítógép tárának (majdnem) bármelyik részére be lehet tölteni.

A parancsprocesszor (commandprocessor) – ez az operációs rendszernek az a része, amely a felhasználó parancsait értelmezi és a végrehajtásukat vezérli – vonatkozásában két újítással növelték a rendszer rugalmasságát. Egyrészt a parancsprocesszort COMMAND.COM néven, végrehajtható programként egy különálló állományba helyezték el, ami a felhasználó számára lehetővé tette, hogy saját, pl. menüvezérelt parancsprocesszort fejlesszen ki, és ezt a rendszerbe beillessze.

A másik újítás az volt, hogy a parancsprocesszort felbontották egy rezidens és egy tranziens részre. Ezt még az a körülmény indokolta, hogy az akkori PC-k RAM-tára meglehetősen szerény kapacitású volt, és ezért a parancsprocesszor rezidens részének, azaz annak a részének, amelyet állandóan az operatív tárban kellett tartani, célszerűen kicsinek kellett lennie. Ezért az operációs rendszer néhány olyan utasítását, mint pl. a FORMAT-ot végrehajtó programkódot mágneslemezen tárolták (tranziens rész), és csak szükség esetén hívták be a RAM-ba.

Lényeges újítást, és egyidejűleg a CP/M-80-tól való további eltávolodás irányába tett lépést jelentett a FAT (file allocation table), magyarul az állományelhelyezési tábla bevezetése. (Más magyar nyelvű szakirodalom az állománykijelölési vagy állományhozzárendelési tábla kifejezést is használja. Mi a továbbiakban az egyszerűség és az egyértelműség kedvéért megtartjuk az eredeti, angol nyelvű terminológiát, és a táblát FAT-nak nevezzük.) Ez a tábla bejegyzéseket tartalmaz, amelyek mindegyike egy, a lemezen lévő, 512 bájt méretű adatterületnek (szektornak) felel meg. E bejegyzések alapján megállapítható, hogy az illető adatterület már hozzá van-e rendelve egy (meghatározott) állományhoz vagy sem. Különleges jelentősége van annak az összefüggésnek is, amely a FAT és a lemezen tárolt állományok katalógusba (directory) való bejegyzése között fennáll. A katalógusbejegyzés az állomány neve és néhány más információ mellett megadja annak a FAT-bejegyzésnek a számát is, amelyik az állomány első szektorának felel meg. Ez a FAT-bejegyzés, csakúgy mint a többi, egy másik FAT-bejegyzésre mutat, amely az illető állomány következő szektorát adja meg.

Végül megemlítünk még két újítást, amelyek a felhasználó munkáját könnyítik meg: az egyik az ún. batch vagy kötegelt feldolgozás, a másik az a lehetőség, hogy egy állomány módosításának az időpontja tárolható a lemezen.

A kötegelt feldolgozást az IBM kívánságára vezették be. Ez az eljárás lehetővé teszi, hogy a felhasználó több DOS parancsot egyetlen, ún. batch-állományba (.BAT) fogjon össze. Ha ezt az állományt meghívjuk, akkor a DOS az állományban lévő parancsokat elolvassa, és sorban végrehajtja úgy, mintha azokat a billentyűzetről írtuk volna be.

A felhasználó számára gyakran hasznos, ha tudja, mikor lett egy állomány utoljára módosítva. Ezért a DOS egy állomány módosításakor az aktuális dátumot és időpontot beírja az állomány katalógusbejegyzésébe, és a katalógus lekérdezésekor ezt a dátumot és időpontot írja ki a képernyőre.

1982 júniusában megjelent az MS-DOS 1.25 jelű verziója (az IBM terminológiája szerint a PC-DOS 1.1), amely a hajlékonylemeznek már nemcsak az egyik, hanem mindkét oldalát használta adattárolásra. Így egyetlen hajlékonylemezen már 320 kbájtnyi adatot lehetett tárolni. Az új verzió egyúttal tovább csökkentette a hardvertől való függőséget.

A 2.0 verzió

1983 márciusában az IBM bejelentette az újabb, PC/XT névre keresztelt személyi számítógépét (az XT az eXtended Technology, azaz bővített technológia jelzőből képzett betűösszetétel), amely az elődeihez képest már nemcsak hajlékonylemezes meghajtót, hanem egy merevlemezes (ún. winchester típusú) meghajtót is tartalmazott. A merevlemez (hard disk vagy fixed disk) akkoriban szinte elképzelhetetlenül nagy, 10 Mbájtos kapacitása teljesen új távlatokat nyitott a mikroszámítógépek felhasználásában, ugyanakkor komoly kihívást jelentett az operációs rendszerek fejlesztői számára. Többek között erre a kihívásra válaszolva fejlesztette ki a Microsoft az MS-DOS 2.0 verzióját (az IBM terminológiájában PC-DOS 2.0), amely alapvetően az IBM PC/XT gépek standard operációs rendszerévé vált.

Az elmúlt évek távlatából visszatekintve ez a verzió lényegében egy szinte teljesen új operációs rendszert képvisel (csak azért szinte, mert a tervezők gondosan ügyeltek ar-

ra, hogy a korábbi, 1.0 verzióval továbbra is fenntartsák a kompatibilitást.) A 2.0 verzió számos, lényeges újdonságot vezetett be. Az új, nagy kapacitású merevlemezek bevezetésével kapcsolatban az egyik legnagyobb problémát a lemezeken elhelyezhető állományok megfelelő szervezése jelentette. A korábbi operációs rendszerek ugyanis minden egyes adathordozón csak egyetlen katalógust kezeltek, ami a kis tárkapacitás miatt elegendő is volt. Merőben más helyzet állt elő viszont a merevlemez igen nagy kapacitása következtében: egy ilyen lemezen elhelyezhető több száz vagy ezer különböző állomány egyetlen katalógusba való bezsúfolása gyakorlatilag lehetetlenné tette volna az állományok megfelelő nyilvántartását és kezelését. A probléma megoldására a Microsoft számára két alternatíva kínálkozott: vagy a CP/M-80, vagy a UNIX (ez utóbbi az AT&T Bell Laboratories fejlesztése) operációs rendszerben használt eljárást követi.

A CP/M a merevlemezt úgy tekintette, mint több, egyedi hajlékonylemezből álló egységet. A rendszer ennek megfelelően a merevlemez tárkapacitását több, kisebb egységre osztotta fel, és ezeket mint egy-egy hajlékonylemezt, külön-külön kezelte. Ebből következően mindegyik ilyen tároló egységnek csak egyetlen katalógusa volt.

Ezzel szemben a UNIX az állományokat hierarchikus rendszerben helyezte el: mindegyik adathordozónak van egy ún. fő- vagy gyökérkatalógusa (root directory), amelyek állományokat is, és további katalógusokat is tartalmazhatnak. Egy ilyen katalógus a főkatalógushoz képest alkatalógust (subdirectory) képez. Természetesen az alkatalógus további alkatalógusokat is tartalmazhat, vagyis ilyen formán az egyes katalógusok egy fa struktúrát alkotnak, amelyben a fa gyökere a főkatalógusnak, a fa csúcsai pedig az alkatalógusoknak feleltethetők meg.

Ismeretes, hogy a Microsoft az állományok hierarchikus szervezése mellett döntött, és ezzel a lépésével még jobban eltávolodott a CP/M irányvonalától. Ennek a hierarchikus rendszernek a bevezetése miatt néhány fontos változtatást kellett végrehajtani az állománykezelés addigi módszerében is.

Addig az ideig egy állományhoz való hozzáférés egy FCB-nek (file control block) nevezett, állományvezérlő adatszerkezeten keresztül bonyolódott le. (A továbbiakban ezt az adatszerkezetet az egyszerűség és az egyértelműség kedvéért az eredeti terminológiának megfelelően FCB-nek fogjuk nevezni). Ez az FCB, amelyet a CP/M-80-nal való kompatibilitás miatt hagytak meg a rendszerben, fontos információkat tartalmazott az állomány nagyságáról, a lemezen elfoglalt helyéről, továbbá tartalmazta az állomány nevét is. Emiatt viszont egy másik katalógusban lévő állományhoz nem lehetett hozzáférni (a CP/M amúgy is csak egy katalógust ismert).

Ezért a DOS fejlesztői úgy határoztak, hogy átszervezik az állományok elérésének módját. A DOS 2.0 verziójától kezdődően az állományokhoz való hozzáférés kizárólag egy ún. handle-n (állománykezelő azonosítón) keresztül történt. A handle egy numerikus érték, amelyet a DOS egy állomány megnyitásakor hozzárendel az állományhoz. A továbbiakban ezen az állományon erre a handle-re hivatkozva lehet műveleteket végezni. A handle bevezetése nem jelentette az FCB-k megszűnését, csupán csak a programozó ezekkel már nem került kapcsolatba.

Egy másik, fontos újítás az installálható perifériavezérlő programok bevezetése volt (ezeknek a vezérlő programoknak a DOS terminológiában device driver a nevük; mi a továbbiakban a rövidség kedvéért a magyar szakirodalomban is használt eszköz-

meghajtó elnevezést használjuk). Ezek az eszközmeghajtók lehetővé teszik, hogy a programozók bizonyos perifériákat, így pl. egy különleges merevlemezes egységet, egeret vagy streamert egyszerű módon installálhassanak a DOS-ba. A képernyő-megjelenítés vonatkozásában a 2.0 verziótól vezették be az ANSI.SYS eszközmeghajtót, amely a programozó számára lehetővé teszi, hogy DOS-funkciókon keresztül pozícionálja a kurzort, vagy megválassza a színeket.

Ezzel a verzióval tették meg az első számottevő lépést az ún. többfeladatos feldolgozás (multitasking) irányába azzal, hogy bevezették az ún. háttérfeldolgozást (background processing). Ennek az a lényege, hogy a processzor (látszólag) egyidőben egyszerre két vagy több programot hajt végre, és a különböző feldolgozások közötti átkapcsolást az operációs rendszer végzi úgy, hogy mindez a felhasználó számára láthatatlan marad. Ezzel a módszerrel dolgozik pl. a PRINT.COM nevű DOS program, amely úgy tud kinyomtatni egy állományt, hogy eközben a számítógépen egy másik felhasználói program is futhat.

A hajlékonylemezek kapacitásának bővítése céljából bevezették azt az opciót, amellyel a lemezeken a sávokat 8 helyett 9 szektorra is fel lehetett osztani. Ezáltal az egyoldalon formált lemezek kapacitása a korábbi 160 kbájtról 180-ra, a kétoldalon formált lemezeké pedig 320 kbájtról 360 kbájtra nőtt.

További újdonságot jelentett, hogy a felhasználó szabadon konfigurálhatta a rendszert, vagy lehetőség nyílt arra, hogy az ún. környezeti blokkon (environment block) keresztül a programok egymás között információkat adhassanak át.

A 2.0-nak alverziói is készültek, amelyek elsősorban a távol-keleti országok igényeihez alkalmazkodtak (japán és koreai karakterkészlet stb.).

A 3.0 verzió

A 2.0 verzióhoz hasonlóan a 3.0 verzió is egy új, még nagyobb teljesítményű számítógéphez, az IBM PC/AT-hez kapcsolódóan jelent meg. (Az AT az Advanced Technology, ami magyarra a továbbfejlesztett technológia névvel fordítható jelzőből képzett betűösszetétel.) Az MS-DOS 3.0 (PC-DOS 3.0) verzió, amelyet 1984 augusztusában mutattak be, a teljes operációs rendszer vonatkozásában számos újdonságot hozott. Ez a verzió az Intel 80286-os mikroprocesszorára épülő AT 20 Mbájtra megnövelt merevelemeze mellett az új, 1.2 Mbájt kapacitású hajlékonylemezes meghajtóját is támogatja.

A 2.0 verzióhoz képest igen sok változtatást hajtottak végre a DOS belső rutinjaiban. Ezek meghatározott műveletek végrehajtásának sebességét növelik, de a programozó számára általában láthatatlanok maradnak. További újdonságot jelentett az osztott állománykezelés és az állományok rekordszintű védelme.

A 3.0 alverziói tovább bővítették az operációs rendszer szolgáltatásait. Az 1984 novemberében kibocsátott 3.1 verzió már lehetővé tette a gépek hálózatba kapcsolását, az 1986-ban megjelent 3.2 verzió pedig a 3.5 inch átmérőjű hajlékonylemezek kezelését. 1987-ben, az IBM új gépcsaládjával, a PS/2-vel egyidőben jelent meg a 3.3 verzió, amely az állománykezeléssel és az országra jellemző karakterkészletekkel kapcsolatban hozott újdonságokat.

6.2. A DOS belső felépítése

A DOS a hardver és a felhasználói programok közötti kapcsolatrendszerben három, egymástól jól elkülöníthető szintre osztható. A hardverhez legközelebb álló szinten van a DOS-BIOS (Basic Input/Output System), középen foglal helyet az ún. DOS-mag (DOS kernel), míg a felhasználói programmal a parancsértelmező processzor (command processor vagy shell) áll közvetlen kapcsolatban. A három különböző szintet képviselő rendszerprogram külön-külön állományban helyezkedik el.

A DOS-BIOS-t az IBMBIO.COM nevű állomány tartalmazza, amely ún. rejtett (hidden) és SYS attribútumú állomány, ezért a lemez katalógusának kilistázásakor a képernyőre nem íródik ki. Maga az állomány, amely a rendszer inicializálásakor kerül a számítógép RAM-tárába, azokat a programokat (eszközmeghajtókat) tartalmazza, amelyek a következő hardverelemeket vezérlik:

CON (billentyűzet és képernyő),
PRN (nyomtató),
AUX (soros interfész),
CLOCK (dátum és idő),
lemezmeghajtó(k), amelyek azonosítója az A, B és C.

Amikor a DOS ezek közül az egységek közül valamelyikkel kapcsolatba akar lépni, akkor az ebben a modulban lévő vezérlő programokhoz fordul, amelyek meghívják a ROM-ban elhelyezett BIOS rutinokat. Ez tehát a DOS-nak az a része, amely közvetlenül kapcsolódik a gépbe beépített ROM-BIOS-hoz, és ezért ez a leginkább hardverfüggő modul.

Az eszközmeghajtókat aszerint is megkülönböztetik, hogy rezidensek vagy installálhatók. Rezidens eszközmeghajtóknak nevezik azokat a vezérlő programokat, amelyeket a ROM-BIOS tartalmaz (és természetesen nem változtathatók meg), ezzel szemben installálhatók (és az igények szerint alakíthatók) azok az eszközmeghajtók, amelyek a DEVICE paranccsal specifikálva a rendszer inicializálásakor a CONFIG.SYS állományon keresztül a rendszer részévé válnak (pl. a virtuális lemezegység).

A DOS-magot (kernel) az IBMDOS.COM állomány tartalmazza, amely ugyancsak rejtett és SYS attribútumú, és ugyancsak a rendszer inicializálásakor töltődik a RAM-tárba. Többek között állományokon történő műveletek végzésére, a RAM-tár kezelésére valamint karakterek be- és kivitelére alkalmas rutinokat tartalmaz.

Ezek a rutinok, amelyeket általánosságban rendszerrutinoknak vagy rendszerfunkcióknak neveznek, a hardvertől függetlenül vannak kialakítva (a hardverhez illesztést az előző, BIOS modul intézi), ezért lényegében bármely PC-n futtathatók. A felhasználói programok ezekhez a funkciókhoz ugyanúgy férhetnek hozzá, mint a ROM-BIOS rutinjaihoz: mindegyik funkció egy szoftvermegszakítással érhető el. A funkció számának és a paramétereknek az átadása a processzor regisztereiben történik.

Az előző két modultól eltérően a parancsértelmező processzort egy látható állomány tartalmazza, amelynek a COMMAND.COM nevet adták. Mivel ez az a modul,

amely közvetlen kapcsolatot tart a felhasználóval (pl. ez írja a képernyőre a jólismert A> vagy a C> rendszerpromptot) és fogadja az operációs rendszernek szóló parancsokat, sokan – tévesen – ezzel azonosítják magát a teljes operációs rendszert is. A valóságban azonban a COMMAND.COM csak egy speciális értelmező program, amely a DOS alatt fut. Maga ez a parancsértelmező processzor (amelyet shellnek is neveznek) három részre osztható fel: egy rezidens részre, egy inicializáló rutinra és egy tranziens részre.

A parancsértelmő processzornak a rezidens része – amint a neve is mutatja –, állandóan a belső RAM-tárban van, és különböző, megszakításkezelő rutinokat tartalmaz, amelyek lehetővé teszik, hogy a rendszer bizonyos eseményekre azonnal reagálhasson. Ilyen esemény lehet pl., ha a gép kezelője lenyomja a Control-C vagy a Control-Break billentyűt, vagy ha a külső egységekkel folytatott adatforgalmazásban bizonyos hibák fordulnak elő. Ekkor a képernyőre írja a bizonyára jól ismert üzenetet:

Abort, Retry, Ignore?
(megszakítani, újra próbálni, figyelmen kívül hagyni?)

Ez a rezidens rész tartalmazza továbbá azt a rutint, amely szükség esetén ismét betölti a COMMAND.COM tranziens részét.

A rezidens rész a tár alsó részébe, közvetlenül a DOS-mag és a hozzá tartozó pufferek és táblák fölé töltődik be. A rendszer a különböző felhasználói programok betöltésekor ügyel arra, hogy a tárnak ez a része ne íródjon felül.

Az inicializáló rész a rendszer indításakor (bootstrap), tehát a gép bekapcsolásakor vagy újraindításakor töltődik be, és feladata a DOS inicializálása. A parancsértelmező processzornak erre a részére a következő fejezetben részletesen kitérünk. Itt csak annyit említünk, hogy ez a rész hajtja végre az AUTOEXEC.BAT állományt (amennyiben van ilyen), majd törlődik a tárból.

A COMMAND.COM tranziens része a tár legfelső szabad részére töltődik be, amelyet egyébként a felhasználói programok felülírhatnak. Ennek a tranziens résznek az a feladata, hogy kiírja a rendszerpromptot, értelmezze a felhasználói parancsokat, és a hozzájuk tartozó programkódot végrehajtsa. Mivel azonban ez a parancsértelmező rész és ezek a programok általában elég jelentős nagyságú tárhelyet foglalnak le, ugyanakkor csak ritkán van rájuk szükség, az ésszerű tárgazdálkodást szem előtt tartva a rendszerfejlesztők azt a megoldást választották, hogy a felhasználói parancsokat három csoportba osztották: belső parancsokra, külső parancsokra és batch-állományokra.

A belső – leggyakrabban használt – parancsokat maga a COMMAND.COM állomány tartalmazza (ezért belső parancsok). Az e parancsokhoz tartozó programok tulajdonképpen félrezidens programoknak tekinthetők, ugyanis mindaddig, amíg a rendszer indítását követően egy felhasználói program ezt a tárrészt felül nem írja, az ezekhez a parancsokhoz tartozó rutinok rezidens módon a tárban vannak. Amikor egy felhasználói program befejeződik, és a vezérlés visszakerül a DOS-hoz, akkor a COMMAND.COM rezidens része egy ellenőrző összeg segítségével megvizsgálja, hogy változott-e a tranziens rész tartalma. Ha változott, akkor lemezről ismételten betölti a teljes tranziens részt (ezért is jó, ha a számítógépes rendszerünk mindig tartalmaz olyan lemezt, amelyen a

rendszerprogramok megtalálhatók). Ha nem talál ilyet, akkor felszólítást küld, hogy helyezzünk be egy rendszerlemezt. Belső parancs pl. a COPY, a DIR vagy a RENAME.

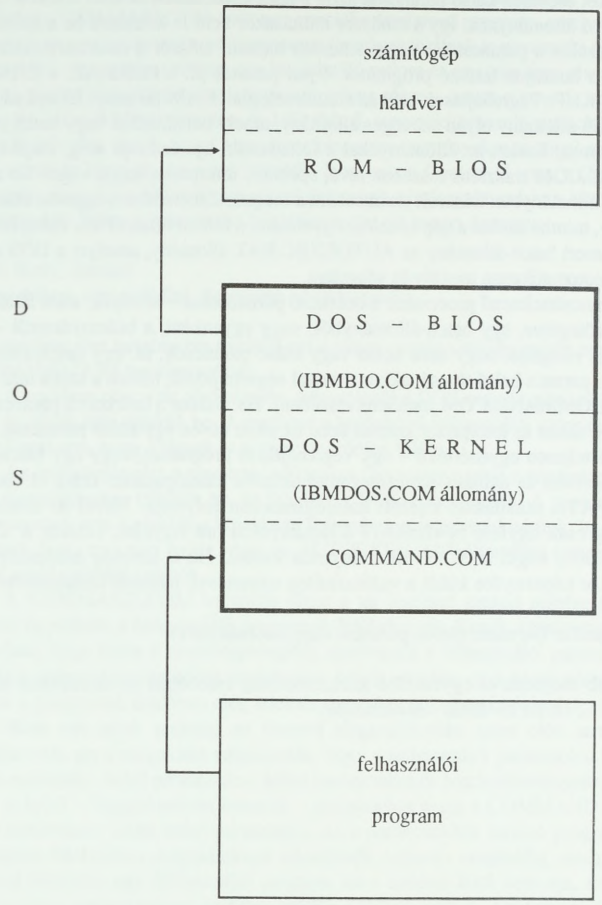
Ezekkel szemben külső parancsok azok a parancsok, amelyek nem részei a COMMAND.COM állománynak, így a rendszer indításakor nem is töltődnek be a tárba. Ahhoz, hogy ezeket a parancsokat végre lehessen hajtani, először a rendszerlemezről be kell tölteni a hozzájuk tartozó programot. Ilyen parancs pl. a FORMAT, a CHKDSK vagy a BACKUP. Végrehajtásuk után az általuk lefoglalt RAM-tár ismét felszabadul.

A batch-állomány olyan szöveges állomány, amely belső, külső vagy batch parancsokat tartalmaz. Ezeket az állományokat a felhasználó írja és hívja meg, majd egy, a COMMAND.COM tranziens részében lévő, speciális interpreter hajtja végre. Ez az interpreter az állományban felsorolt parancsokat a megadott sorrendben, egymás után végrehajtja úgy, mintha ezeket a gép kezelője egyenként, a billentyűzetről írta volna be. Egy ilyen, közismert batch-állomány az AUTOEXEC.BAT állomány, amelyet a DOS az indítása után automatikusan meghív és végrehajt.

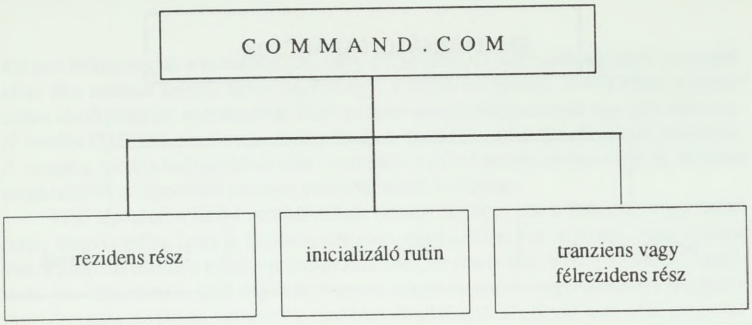
A parancsértelmező processzor a beérkező parancsokat – amelyek, mint láttuk, érkezhetnek kötegelve, egy batch-állományból, vagy egyenként, a billentyűzetről – először aszerint vizsgálja, hogy ezek belső vagy külső parancsok, ill. egy batch-állomány hívása. Ha a parancs belső típusú, akkor azonnal végrehajtódik, hiszen a kódja már a tárban van (a COMMAND.COM tranziens részében). Ha viszont a beérkezett parancs nem belső típusú, akkor az interpreter keresni kezd az adott néven egy külső parancsot, vagy – ami ezzel teljesen egyenértékű – egy végrehajtható programot, vagy egy batch-állományt. A keresést az aktuális lemezmeghajtó aktuális katalógusában kezdi el, majd a legutolsó PATH utasítással kijelölt katalógusokban folytatja. Mivel az értelmező egyidejűleg csak egyféle bővítményű állományokat tud figyelni, először a .COM, utána az .EXE, végül a .BAT állományokat keresi. Ha a keresés eredménytelen marad, akkor képernyőre küldi a valószínűleg ugyancsak ismerős hibaüzenetet:

Bad command or file name (hibás parancs- vagy állománynév)

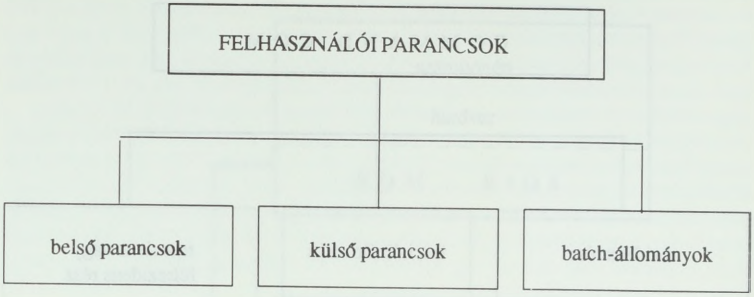
A jobb megértés és egyszerűbb áttekinthetőség érdekében az előzőekben elmondottakat a 11., 12. és 13. ábrán szemléltetjük.



11. ábra: A DOS kapcsolódásai és felépítése



12. ábra: A COMMAND.COM részei



13. ábra: A felhasználói parancsok felosztása

6.3. A DOS betöltése

Amikor bekapcsoljuk a számítógépet, vagy a Ctrl-Alt-Del billentyűk egyidejű lenyomásával újra indítjuk (reset), akkor elindul egy, a ROM-BIOS-ban, az FFFF0(h) abszolút címen tárolt program végrehajtása. Ezen a címen tulajdonképpen csak egy gépi kódú ugró utasítás (JMP) áll, amely a vezérlést átadja a hardver saját, belső ellenőrző rutinjának. A vizsgálat sikeres befejeződése után a vezérlést a ROM betöltő rutinja veszi át, és ezzel megkezdődik az operációs rendszer betöltődésének folyamata.

Első lépésben a ROM betöltő rutinja (ismert angol nevén a bootstrap vagy bootrutin) megvizsgálja, hogy a hajlékonylemezes meghajtóban van-e lemez. Amennyiben van, a rutin ezt tekinti a rendszert indító lemeznek, és innen kísérli meg a rendszer betöltését. Ha viszont nem talál hajlékonylemezt, akkor megnézi, hogy tartozik-e a géphez merevlemezes meghajtó, amelyről a rendszert elindíthatná. Ha ilyent sem talál, akkor hibajelzést küld, ill. felszólítja a felhasználót, hogy helyezzen egy rendszerlemezt valamelyik meghajtóba (esetleg a BASIC jelentkezik be).

Ha a ROM betöltő rutinja talált lemezt valamelyik meghajtóban, beolvassa a tárba a lemez első szektorát és átadja neki a vezérlést. Ezzel a ROM-rutin elvégezte a dolgát, a rendszer további betöltése már magának a DOS-nak a feladata.

A lemez első szektora, az ún. betöltő (bootstrap) szektor – többek között – tartalmaz egy betöltő (boot) rutint, amely megvizsgálja, hogy az illető lemez valóban rendszerlemez-e. Ehhez beolvassa a főkatalógus első szektorát és megnézi, hogy az első két állomány az IBMBIO.COM és az IBMDOS.COM állomány-e (ilyen sorrendben). Ha nem találja ezeket az állományokat ezen a helyen, akkor a lemez nem betölthető rendszerlemez, és megjelenik a képernyőn a lemezcserére figyelmeztető üzenet. Ha viszont a betöltő rutin megtalálta a rendszerállományokat, akkor ezeket beolvassa a tárba.

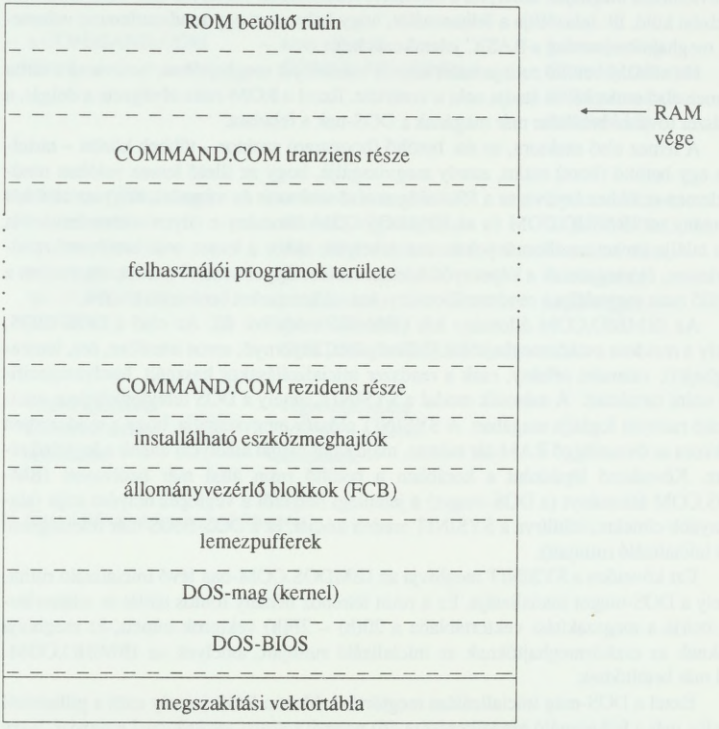
Az IBMBIO.COM állomány két különálló modulból áll. Az első a DOS-BIOS, amely a rezidens eszközmeghajtókat (billentyűzet, képernyő, soros interfész, óra, lemezmeghajtó), valamint néhány, csak a rendszer inicializálásakor használt, hardverspecifikus rutint tartalmaz. A második modul a SYSINIT, amely a DOS tulajdonképpeni inicializáló rutinjait foglalja magában. A SYSINIT először megvizsgálja, hogy a rendszerben mekkora az összefüggő RAM-tár mérete, majd saját magát áthelyezi ennek a legfelső részére. Következő lépésként a korábban a betöltő rutin által már beolvasott IBMDOS.COM állományt (a DOS-magot) a jelenlegi helyéről a végleges helyére tolja (alacsonyabb címekre, felülírva a SYSINIT eredeti kódját, és a DOS-BIOS már feleslegessé vált inicializáló rutinjait).

Ezt követően a SYSINIT meghívja az IBMDOS.COM-ban lévő inicializáló rutint, amely a DOS-magot inicializálja. Ez a rutin létrehoz néhány fontos táblát és adatterületet, beírja a megszakítási vektortáblába a 20(h) – 2F(h) vektorok címeit, és meghívja azoknak az eszközmeghajtóknak az inicializáló rutinjait, amelyek az IBMBIO.COM-mal már betöltődtek.

Ezzel a DOS-mag inicializálása megtörtént. Most a DOS (amely ettől a pillanattól kezdve már a felhasználó rendelkezésére áll) normál rutinjai segítségével megnézi, hogy van-e specifikálva CONFIG.SYS nevű állomány, amely segítségével a felhasználó az

igényeinek megfelelően konfigurálhatja a rendszert. Ha talál ilyen állományt, akkor betölti, és a benne lévő adatoknak megfelelően elvégzi a rendszer konfigurálását. Létrehozza a kívánt számú lemezpuffert és az egyes állományvezérlő blokkokat (FCB), beállítja az egyidejűleg nyitva tartható állományok számát, megjegyzi a parancsértelmező processzor (shell) nevét. Betölti a megadott installálható eszközmeghajtókat, amelyeket az init moduljaik meghívásával inicializál és a többi meghajtót tartalmazó listába beszerkeszt. Ezt követően lezárja az összes megnyitott handle-t, majd újra megnyitja az alapértelmezés (default) szerinti beviteli/kiviteli eszközökhöz tartozókat.

Végül betöltődik a parancsértelmező processzor (alapértelmezés szerint ez a COMMAND.COM, de a CONFIG.SYS állományban ehelyett más is kijelölhető), amely átveszi az összes további folyamat vezérlését. Ezzel befejeződik az operációs rendszer betöltődése. Megjelenik a képernyőn a rendszer promptja jelezve, hogy várja a felhasználó parancsait. A 14. ábrán bemutatjuk a DOS betöltése után kialakult tárfelosztást.



14. ábra: Tárfelosztás a DOS betöltése után

6.4. COM és EXE programok

Az IBM PC-ken futó programok alapvetően COM vagy EXE típusúak (a COM és az EXE a programnév kiterjesztései). Ebben a fejezetben e két programtípus felépítésével és működésével foglalkozunk.

A két programtípus közötti különbség már a méretükben is megmutatkozik. Míg egy EXE program a DOS által rendelkezésre bocsátott teljes RAM-tárat is elfoglalhatja (sőt ennél nagyobb is lehet, csak akkor egyszerre nem tölthető be), addig egy COM program mérete a 64 kbájtot nem haladhatja meg.

COM programok esetén a programkódot, az adatokat és a veremtárat is egyetlen összefüggő, 64 kbájtos szegmensben kell elhelyezni. Ebből következik, hogy valamennyi szegmensregiszternek a program indulásakor, és a végrehajtás közben is azonos és változatlan az értéke, és ez a 64 kbájtos szegmens tárbeli kezdőcímére mutat. (Csak az ES regiszter tartalma változtatható, mivel ennek a program végrehajtása szempontjából nincs jelentősége.) Az EXE állományoknál a szegmensek elrendezése nem ennyire szigorú. A kód, az adatok és a veremtár különböző szegmensekben helyezkednek el, sőt a méretüktől függően több szegmensre is kiterjedhetnek. Ezért aztán egy EXE program végrehajtása közben az egyes szegmensregiszterekben lévő értékek változhatnak.

Egy másik különbség az állományok tárolási módjában van: a DOS a COM állományokat pontosan úgy tárolja a hajlékony- vagy a merevlemezen, ahogyan azok betöltődésükkor a számítógép tárában elhelyezkednek. Ezzel szemben az EXE programokat az operációs rendszer speciális formátumú állományokban tárolja, amelyek – többek között – a betöltésükre vonatkozó információkat is tartalmazzák.

Mindkét típusú programállományt a DOS EXEC funkciójával lehet betölteni és elindítani. Ez a funkció a felhasználó rendelkezésére áll, de ezt használja a parancsértelmező processzor is a külső parancsok (pontosabban fogalmazva a parancsokhoz tartozó COM vagy EXE programok) végrehajtására. Mielőtt az EXEC funkció a meghívandó programot betöltené a tárba, más DOS-funkciókon keresztül lefoglalja a szükséges RAM területet. A lefoglalt terület elejére elhelyez egy speciális, program szegmens előtagnak nevezett adatszerkezetet (ismert angol neve program segment prefix, általánosan használt rövidítéssel PSP). A PSP végétől kezdődően betölti a végrehajtandó programot, betölti a kezdőértékeket a szegmensregiszterekbe és a verembe, majd elindítja a programot. A program befejeződése után a PSP és a program által elfoglalt területet felszabadítja (hacsak ezt a program nem tiltja), és visszaadja a vezérlést a COMMAND.COM-nak.

6.4.1. Program szegmens előtag (PSP)

Ebben az alfejezetben röviden megismerkedünk a COM és az EXE programok végrehajtásához is használt, program szegmens előtagnak (PSP, program segment prefix) nevezett adatszerkezettel. A PSP egy 256 bájt hosszú adatszerkezet, amely mind a DOS, mind a végrehajtandó program számára fontos információkat tartalmaz. Az „ősé" a leg-

első DOS-verziókban is megtalálható volt, ezért számos párhuzamosság fedezhető fel közte és a CP/M által használt, hasonló célú adatstruktúra között. A PSP-t a DOS EXEC funkciója hozza létre a tárban, és közvetlenül a végrehajtandó program kódja elé helyezi el. A következőkben vizsgáljuk meg a PSP szerkezetét (15. ábra).

00(h)	program befejezése megszakítási rutin hívása INT 20(h)	2 bájt
02(h)	program által lefoglalt terület végének szegmenscíme	1 szó
04(h)	fenntartott	1 bájt
05(h)	DOS funkciókat hívó megszakítás hívása INT 21(h)	5 bájt
0A(h)	program vége megszakítási rutin címe INT 22(h)	2 szó
0E(h)	Ctrl-C (Break) megszakítási rutin címe INT 23(h)	2 szó
12(h)	kritikus hibát kezelő megszakítási rutin címe INT 24(h)	2 szó
16(h)	fenntartott	22 bájt
2C(h)	környezeti blokk szegmenscíme	1 szó
2E(h)	fenntartott	46 bájt
5C(h)	első FCB	16 bájt
6C(h)	második FCB	16 bájt
80(h)	parancssorban lévő karakterek száma	1 bájt
81(h)	parancssor és egyben az alapértelmezés szerinti DTA	128 bájt

15. ábra: A PSP (program szegmens előtag) szerkezete

Az ábra bal oldalán látható (az egyes adattételekhez tartozó) kezdőcímek a PSP 00(h) kezdőcímétől számított ofszetcímeket jelentik. Mivel a PSP tárbeli tényleges címe több különböző körülménytől függ (DOS-verzió mérete, más programok a tárban), a tényleges helye előre nem adható meg.

A 00(h) ofszetcímen egy program befejezésére szolgáló DOS megszakítás hívása áll (INT 20(h)). Ehhez a megszakításhoz tartozó rutin felszabadítja a program által lefoglalt tárterületet, és visszaadja a vezérlést a COMMAND.COM-nak, ill. a programot meghívó programnak. A 02(h) ofszetcímen álló szó (2 bájt) a program által lefoglalt terület utáni első szabad szegmenscímet adja meg. A program ez alapján tudja megállapítani, hogy a feladata elvégzéséhez kell-e (más DOS funkció hívásával) további RAM területet igényelnie, vagy éppenséggel más folyamatok számára átengedhet felesleges, lefoglalt helyet.

A 05(h) ofszetcímen egy 5 bájtból álló gépi kódú utasítás (CALL) áll, amely a 21(h) DOS megszakítást hívja meg. Ezen a megszakításon keresztül hívhatók meg a DOS olyan, alapvető funkciói, mint az állományokon való műveletek végzése, karakterek bevitele, kivitele és számos más egyéb. E funkciók ilyen módon történő meghívásának azonban ma már nincs gyakorlati jelentősége.

A PSP 0A(h), 0E(h) és 12(h) ofszetcímei egy-egy megszakításvektort tartalmaznak, amelyek rendre a program vége (INT 22(h)), a Control-C vagy a Break (INT 23(h)), valamint a kritikus hibát kezelő (INT 24(h)) megszakítási rutinok kezdőcímeinek felelnek meg. Ha a program a futása közben e vektorok valamelyikét a megszakítási vektortáblában átírná, akkor a DOS a program befejeződését követően a megváltoztatott vektor eredeti tartalmát a PSP-ből elolvassa, és visszaírja a vektortáblába.

A 2C(h) ofszetcímen az ún. környezeti blokk (angol nevén environment block) szegmenscíme áll. A környezeti blokk ASCII-karakterláncok sorozatából áll, amelyek mindegyikét egy-egy végjel (a 0 ASCII-kód) zár le (ezeket ASCIIZ-karakterláncoknak is nevezik). Az ilyen módon egymástól elválasztott karakterláncok olyan információkat tartalmaznak, mint például a COMMAND.COM által használt kereső út (path) vagy magának a COMMAND.COM-nak a lemezen lévő helye. A környezeti blokk teszi lehetővé, hogy egy program az általa meghívott programnak bizonyos információkat adhasson át (erről a későbbiekben még lesz szó).

A CP/M-mel való kompatibilitás fenntartása érdekében a PSP 5C(h) és 6C(h) ofszetcímén egy-egy FCB (file control block) található. A DOS a program meghívásakor ebbe a két blokkba bemásolja a programnév után álló első két paramétert feltételezve, hogy ezek állománynevek. (Azt, hogy ezek valóban állománynevek-e, a felhasználói programnak kell eldöntenie.) Ha a paraméterek valóban állománynevek, akkor is csak a meghajtó azonosítója és az állományok neve íródik be az FCB-kbe, mert ezek az adatszerkezetek a hierarchikus álománykezelést nem támogatják. Tekintettel arra, hogy a programok fejlesztésénél a CP/M-mel való kompatibilitás már semmilyen szerepet sem játszik, a handle-funkciók pedig (amint majd látni fogjuk) számos előnyt kínálnak, ennek a két FCB-nek a mai programok többségénél már nincs jelentősége.

A program meghívásakor a programnév után megadott összes paraméter (ezek összességét nevezik angolul program tailnek) a 81(h) ofszetcímtől kezdődően bemásolódik a PSP-be. Azok a paraméterek azonban, amelyek a bevitel vagy a kivitel átirányítására vonatkoznak, figyelmen kívül maradnak, mert ez a művelet az operációs rendszer dolga, és a program számára láthatatlan marad. A 81(h) és a PSP vége (ofszetcíme FF(h)) közötti területre – amit most paraméterterületnek is nevezhetünk – bemásolt ka-

rakterek száma a PSP 80(h) ofszetcímére íródik be. Ebbe a számba a paramétersort lezáró kocsi vissza (CR) karakter nem számít bele.

Ezt a paraméterterületet azonban a DOS a lemezen lévő állomány és a tárban lévő program közötti adatátviteli célokra is használja. (Ennek a pufferként szolgáló területnek az angol neve Disk Transfer Area, ami magyarul lemezátviteli területet jelent, de funkcióját tekintve célszerűbb az adatátviteli terület megnevezés. A továbbiakban ez utóbbit, ill. az elterjedt DTA rövidítést használjuk.)

Amint majd látni fogjuk, ez a – mindössze 128 bájtos – adatátviteli terület az esetek nagy részében igen kicsinek bizonyul, ráadásul meglehetősen kényelmetlen helyen is van (közvetlenül utána már a programkód következik). Ezért az FCB-vel dolgozó programok ezt az adatterületet – egy DOS funkció hívásával – általában a tár valamely más részére helyezik át. Mindaddig azonban, amig ezt az áthelyezést egy program nem végzi el, a DOS a PSP-nek ezt a részét használja adatátviteli célokra.

6.4.2. COM programok

A COM állományokat a hajlékony- és a merevlemezek pontosan olyan formában tárolják, amilyen formában azok a számítógép RAM tárában elhelyezkednek. Betöltésükhöz nincs szükség különleges információkra, ezért az .EXE programokhoz képest valamelyest gyorsabban betölthetők és végrehajthatók.

Miután a DOS EXEC funkciója a COM programot közvetlenül a PSP után betöltötte, a vezérlést a PSP utolsó tárhelyét követő tárcímre, tehát a 100(h) ofszetcímre adja át, és ezzel megkezdődik a program végrehajtása. Emiatt ezen a címen mindig egy végrehajtható utasításnak kell állnia (ez persze lehet egy ugró utasítás is, ami a vezérlést aztán valahová a program belsejébe továbbíthatja).

Amint már említettük, egy COM program maximum 64 kbájt hosszúságú lehet, amiből még le kell számítani a PSP 256 bájtját, és az EXEC által eleve a verembe helyezett szót (két, zéró értékű bájt). E korlátozás ellenére a DOS a program számára mégis a rendelkezésre álló teljes RAM területet lefoglalja (ez is a CP/M-es időkből fennmaradt örökség: az akkori tárak mérete nem haladta meg a 64 kbájtot). A teljes tárterület lefoglalása miatt a COM programnak így nem marad lehetősége arra, hogy futása közben egy másik programot meghívjon. Ezen a problémán úgy lehet segíteni, hogy a COM program egy DOS funkció segítségével az általa nem használt RAM területet felszabadítja.

Amikor a vezérlés átkerül a COM programhoz, valamennyi szegmensregiszter a PSP kezdőcímére, a 00(h) ofszetcímre mutat. Ezért a COM program mindig a 100(h) ofszetcímen kezdődik. A veremmutató értéke FFFE(h) lesz, így a program által elfoglalt 64 kbájtos szegmens felső végére kerül. A verem a COM programon belüli minden alprogram meghívásakor 2 bájttal nő az alacsonyabb címek, vagyis a programkód vége irányában. A programozónak ezért gondoskodnia kell arról – és ezt csak ő teheti meg – hogy a verem ne növekedjen meg olyan mértékben, hogy a programkódot esetleg felülírhassa. Ez ugyanis szinte bizonyosan a program „fejre állását" okozná.

Egy COM program befejezésére, és a vezérlésnek a DOS vagy a meghívó program részére történő visszaadására több lehetőség van. Ha a program a DOS 1.0 verziója alatt

fut, akkor a 21(h) megszakítás 00(h) funkciójának, vagy a 20(h) megszakításnak a meghívásával fejezhető be. Egy másik, az első pillantásra talán meglepőnek tűnő lehetőség az, hogy a programot egy NEAR RETURN (RET) assembler utasítással fejezzük be. Ennek az utasításnak a végrehajtásakor a program futása azon a címen folytatódik, amelyik a verem tetején van. Mivel az EXEC funkció a COM program részére történt vezérlésátadás előtt két 0 értékű bájtot helyezett oda, a program végrehajtása a CS:0000 tárcímen folytatódik. Ez viszont a PSP kezdőcíme, ahol a 20(h) megszakítás hívása található, ami befejezi a programot.

Minden más verzió alatt a DOS 21(h) megszakítás 4C(h) funkcióját kell előnyben részesíteni a program befejezéséhez. Ennek segítségével az éppen befejezett program numerikus értékként egy paramétert (az ún. visszatérési kódot) is átadhat a meghívójának, amely egy DOS funkcióval lekérdezhető. E kód alapján a meghívó program megállapíthatja, hogy a meghívott program rendben végrehajtódott-e. Megállapodás szerint a 0 érték átadása azt jelenti, hogy a program sikeresen elvégezte a feladatát, míg minden más érték hibát jelent.

Azoknak, akik valamilyen magasszintű nyelven programoznak, a COM állományok létrehozásával nem kell törődniük, mert az ezzel kapcsolatos teendőket elvégzi helyettük a nyelv fordítóprogramja, vagy az az interpreter, amely a program feldolgozását végzi. Ezzel szemben az assembly nyelven programozóknak egy COM program kifejlesztésénél néhány dolgot figyelembe kell venniük. A most következő néhány bekezdést ez utóbbiak figyelmébe ajánljuk.

A fejezet elején már említettük, hogy a COM programokat a hajlékony- és a merevlemezek pontosan olyan formában tárolják, amilyen formában ezek a számítógép tárában elhelyezkednek. Abból a tényből, hogy ezek a betöltésükre vonatkozóan semmilyen információt sem tartalmaznak, következik, hogy a DOS ezeket a programokat a tár bármely (16-tal osztható) címtől kezdődő részére betöltheti. Ez viszont azt jelenti, hogy a COM programok nem tartalmazhatnak explicit szegmenscímeket (vagyis a program a szegmensregisztereknek nem adhat értéket). A betöltés helyét (és így a szegmensregiszterek értékét) a tár mindenkori foglaltságától függően a DOS határozza meg, és a program csak az így kijelölt szegmensen belüli ofszetcímekkel dolgozhat.

Ez az oka annak, hogy egy COM program mérete nem haladhatja meg a 64 kbájtot, és emiatt a programozó a programból nem hívhat meg FAR típusú eljárásokat. Mivel a COM programoknak – beleértve az adatokat és a veremtárat is – egyetlen szegmensben kell elhelyezkedniük, nem tartalmazhatnak olyan utasításokat, amelyek a program végrehajtásának folytatását valamely más szegmensregiszter-tartalommal kijelölt helyen írnák elő. Ezek helyett csak NEAR utasítások használhatók.

A szegmensregiszterek értékadásának ez a korlátozása csak magának a COM programnak a szegmensregisztereire vonatkozik (hiszen ezek értékei a program minden egyes meghívásakor mások lehetnek). Olyan, az operációs rendszer által meghatározott, konstans értékek, mint például a video-RAM szegmenscíme természetesen a COM programon belül is betölthetők a megfelelő szegmensregiszterekbe.

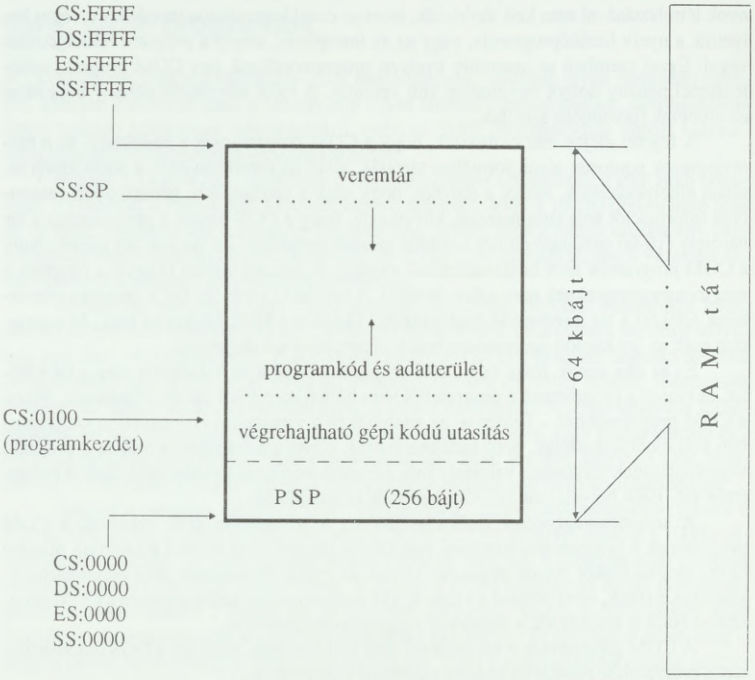
A COM programnak a veremtárral nem kell törődnie, mert ezt a DOS automatikusan a programhoz rendelt 64 kbájtos szegmens végére helyezi.

A COM program meghívása előtt a DOS a rendelkezésre álló összes tárterületet lefoglalja a program számára. Ez a legtöbb esetben nem jelent problémát, mert a prog-

ram a végrehajtását követően azonnal el is tűnik a tárból, és az általa lefoglalt terület ismét felszabadul. Ha viszont egy COM program rezidens, azaz a végrehajtását követően is a tárban marad, akkor ez már problémát jelent, mert (a DOS szemszögéből nézve) a tárban már nincs további üres hely, ahová más programokat tölthetne be.

Ugyanez a probléma merül fel akkor is, amikor egy nem rezidens COM program futása során a DOS EXEC funkciójával egy másik programot szeretne betölteni és végrehajtani. Ez sem lehetséges, hiszen a DOS nyilvántartása szerint a tárban nincs üres hely – ami volt, azt maga a DOS foglalta le nagyvonalúan, az első program betöltésekor.

Természetesen mindkét problémán lehet segíteni: az erre való DOS funkciók hívásával a valójában nem használt tárterületek felszabadíthatók, és ezeken más programok is elhelyezhetők. Erre két lehetőség is kínálkozik. Mielőtt azonban ezeket megvizsgálnánk, a szemléletesség kedvéért nézzük meg a 16. ábrát, amely egy COM program tárbeli felépítését mutatja be.



16. ábra: COM program tárbeli felépítése

Az ábra ahhoz is hozzásegít, hogy a COM program által lefoglalt tárterület felszabadításának kétféle lehetőségét könnyebben megértsük. Az egyik lehetőség az, hogy csak a 64 kbájtos COM szegmensen kívüli területet tesszük szabaddá, a másik pedig az, hogy a teljes, használaton kívüli területet felszabadítjuk, beleértve a COM szegmensen belüli, kihasználatlan területet is. A második megoldással természetesen több tárhelyet szabadítunk fel, ez azonban azzal a veszéllyel jár, hogy a COM program veremtára a programhoz tartozó szegmensen kívülre kerül, és így más programok könnyen felülírhatják. Ezért ebben az esetben a veremtárat előbb át kell helyezni a kód- és az adatterület végére, és csak ezt követően szabad a tárat a verem végétől kezdődően felszabadítani. A verem méretét ekkor természetesen valamilyen, önkényesen megválasztott méretre rögzíteni kell. Az esetek többségében ehhez 512 bájt elegendőnek bizonyul.

A következő programpélda mintaként szolgálhat valamennyi, általunk kifejlesztett COM programhoz. A program indítása után egy kis rutin a veremtárat a programszegmens végére helyezi, és a fennmaradó teljes tárterületet felszabadítja. Ez a kis rutin akkor is hasznos lehet, ha a programunknak más programot nem kell meghívnia, és rezidensnek sem kell lennie, ezért célszerű valamennyi COM programba beilleszteni. Ha ugyanis a jövőbeli DOS verziók majd lehetővé teszik több program egyidejű futását, akkor ezek végrehajtásához amúgy is szükség lesz szabad tárhelyre. Ez pedig csak úgy biztosítható, ha a használaton kívüli tárterületeket felszabadítjuk.

```
;
;                          C O M V A Z . A S M
;
;   Funkciója: egy COM program betöltésekor feleslegesen lefog-
;              lalt tárterület felszabadítása.
;
;
;--- KÓD ----------------------------------------------------------

kod     segment para 'CODE'     ; A kódszegmens definiálása.

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

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

                                ; Csak egy, közös szegmens van.

start:  jmp freemem             ; Tárterületet felszabadító rutin
                                ; hívása.
;--- ADATOK -------------------------------------------------------

                                ; Itt helyezhetők el az adatok, puf-
                                ; ferek, változók és egyéb munkaterü-
                                ; letek. Most egy karakterláncot he-
                                ; lyeztünk ide, amit a program kód-
                                ; részében kiíratunk, hogy végrehaj-
                                ; tásakor lássunk is valamit.

CR      equ 13                  ; Kocsivissza ASCII kódja.
LF      equ 10                  ; Soremelés ASCII kódja.
KEND    equ "$"                 ; Karakterlánc vége jelzés.

karlanc db CR,LF,"Ez a program felszabadítja a program betoltésekor"
        db CR,LF,"felesleges en lefoglalt tárterületet.",CR,LF,KEND

;------------------------------------------------------------------

prog    proc near               ; A karakterláncot megjelenítő
                                ; program.

        mov  ah,9               ; Karakterláncot megjelenítő
        mov  dx,offset karlanc  ; funkció és megszakítás
        int  21h                ; hívása.

        mov  ax,4c00h           ; 21(h) megszakítás, a program a 0
        int  21h                ; hibakóddal ér véget.

prog    endp

;--- FREEMEM : A felesleges tárterület felszabadítása -------------

freemem: mov  bx,offset veg     ; Program végét jelző tárcím -> BX.
         mov  cl,4              ; Mivel paragrafusokban kell
         shr  bx,cl             ; számolni, osztás 16-tal.
         inc  bx                ; A biztonság kedvéért kerekítés
                                ; paragrafushatárra.
         mov  ah,4ah            ; Tárterületet változtató funkció.
         mov  sp,offset veg     ; SP átállítása a szegmens végéről
                                ; a program végére.
```

```
        int  21h
        jmp  prog                       ; Ugrás a karakterláncot megjelenítő
                                        ; programrészre.

freememveg label near                   ; Tárterületet felszabadító rutin
                                        ; végcíme.

;— VEREM ———————————————————————————————————————————————

        dw (256-((freememveg-freemem) shr 1)) dup (?)

                                        ; A verem hossza választásunk szerint
                                        ; 256 szó. Mivel a program futtatása
                                        ; után a freemem rutinra már nincs
                                        ; szükség, az általa elfoglalt terü-
                                        ; let felülírható, így ez is a verem
                                        ; részét képezheti.

;———————————————————————————————————————————————————————

veg     equ this byte                   ; Ezen a tárcímen van a program vége.

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

kod     ends                            ; A (kód)szegmens vége.
        end  start                      ; Az assembler forrásprogram vége.
                                        ; A program végrehajtása a start
                                        ; címkén kezdődik.
```

Ahhoz, hogy egy COM program futtatható legyen, előbb természetesen le kell fordítani egy assembler fordító programmal (MASM), majd a LINK programmal el kell végezni a program szerkesztését. A szerkesztés során képernyőre íródik a

Warning: No stack segment
(Figyelem: nincs veremszegmens)

üzenet, amivel azonban most nem kell törődnünk. Ha más hibaüzenetet nem kapunk, akkor a LINK programszerkesztő egy EXE állományt hoz létre. Mivel azonban mi nem EXE, hanem COM állományt akarunk készíteni, utolsó lépésként meg kell hívni az EXE2BIN nevű programot. Ez a program egy EXE állományt COM állománnyá alakít át. Ezeket a lépéseket egy TESZT.ASM nevű, assembly forrásprogramra az alábbi sorok mutatják:

masm teszt;
link teszt;
exe2bin teszt.exe teszt.com

Ha a hívott programok ezeket a lépéseket rendben végrehajtották, akkor a TESZT.COM állományt a DOS szintjéről a TESZT beírásával hívhatjuk. (A TESZT.EXE állomány ekkor természetesen már törölhető.)

6.4.3. EXE programok

Az EXE programoknak a COM programokkal szemben az az előnyük, hogy a kód, az adatok és a verem maximális hossza nincs 64 kbájtra korlátozva. Ezzel szemben viszont az EXE állományok a COM állományokhoz képest bonyolultabbak, ami abban is megmutatkozik, hogy egy EXE állomány a tulajdonképpeni programon kívül még egy sor más információt is tartalmaz. A jövőbeli DOS verziókra tekintettel azonban ezekből a szükséges többlet-információkból előny is származhat, mert ezek segítségével az EXE programok könnyebben illeszthetők a várható újdonságokhoz, így pl. a többfeladatos rendszerekhez (multitasking).

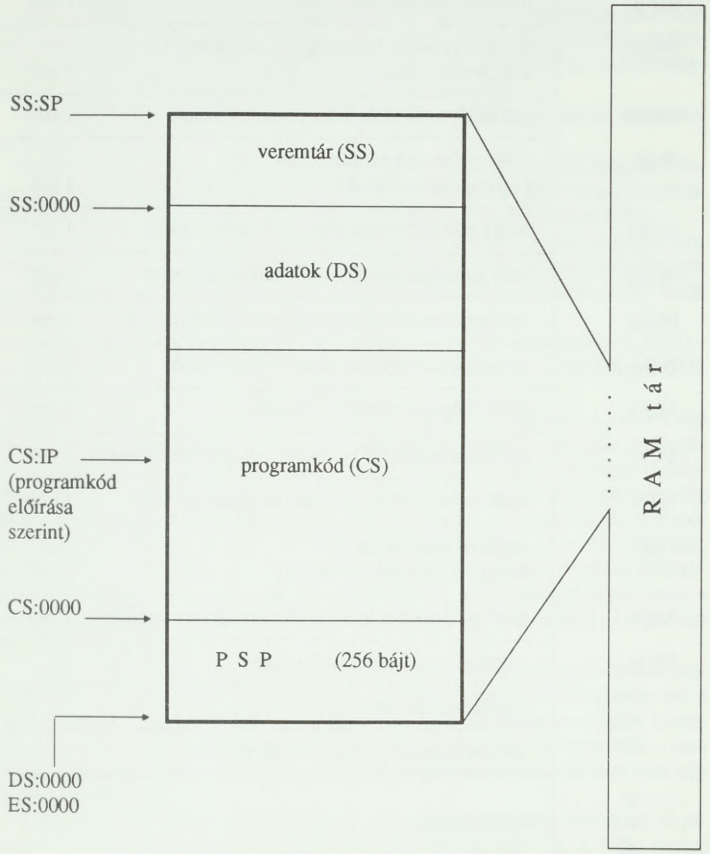
Az EXE programokban a kód, az adatok és a verem külön szegmensekben helyezkednek el. Az így létrejövő kód-, adat- és veremszegmensek egymástól függetlenek, és a programon belül tetszőleges a sorrendjük. A sorrendnek elsősorban akkor van jelentősége, ha az assembly nyelvű programunkat magasszintű programnyelven megírt könyvtári állományokkal akarjuk összeszerkeszteni. A magasszintű nyelvi fordítóprogramok ugyanis megállapodás szerint az assembly forrásprogramban elsőként a kódszegmens(eke)t, ezután az adatszegmens(eke)t, legvégül pedig a veremszegmenst várják. Ennek a sorrendnek a betartása egyéb okokból is célszerű (l. később a tárterületet felszabadító programpéldát).

Az EXE programok a COM programokhoz hasonlóan nem előre megadott, hanem tetszőleges, 16-tal osztható tárcímtől kezdődően töltődnek a tárba. Mivel azonban egy EXE program általában több szegmensből áll, elkerülhetetlen a FAR típusú utasítás használata, ha egyik szegmensből egy másik szegmensben lévő alprogramot kell meghívnunk. Ilyen alprogram meghívásához természetesen az ofszetcím mellett az alprogram szegmenscímét is meg kell adni. Ha viszont a program a meghívásakor nem előre meghatározott helyre töltődik be, akkor nyilvánvalóan a szegmenscímek sem lehetnek rögzített értékek, hanem esetről-esetre változnak.

COM programok esetén ilyen probléma nincs, hiszen az egész program egyetlen szegmensben helyezkedik el (éppen ezért FAR utasítást nem is tartalmazhat). A szegmenscímeket elegendő csak egyszer, a program betöltésekor rögzíteni, mert ezek a program futása során nem változnak. Ezzel szemben az EXE programoknál a szegmensregiszterek tartalmát minden egyes betöltéskor újra meg kell határozni (ezt nevezik címáthelyezésének vagy relokálásnak). Az assembler fordítóprogram részét képező LINK programszerkesztő program mindegyik EXE program elejére elhelyez egy adatszerkezetet – nevezzük ezt az EXE állomány fejlécének (angol neve header) –, amely – többek között – valamennyi szegmenshivatkozás címét, azaz mindazon tárhely címét tartalmazza, amelyekre a program végrehajtása során egy szegmens címe kerül.

Amikor tehát az EXEC funkció betölt egy EXE programot, akkor már ismeri azokat a címeket, amelyekre az EXE program különböző szegmensei betöltődnek, így ezekre a címekre beírhatja a megfelelő értékeket. Emiatt ugyan valamivel több idő telik el a program betöltése és indítása között mint a COM állományok esetén, és az EXE állományok mérete is megnövekszik a fejléccel, azonban ezek a hátrányok eltörpülnek ahhoz az előnyhöz képest, hogy így 64 kbájtnál nagyobb programokat is létre lehet hozni.

A következő ábrák egy EXE program tárbeli felépítését (17. ábra), valamint az EXE állomány fejlécének szerkezetét mutatják be (18. ábra).



17. ábra: EXE program tárbeli felépítése (a szegmensek ettől eltérő sorrendben is elhelyezkedhetnek)

00(h)	EXE állomány azonosítója 5A4D(h)	2 bájt
02(h)	állomány hossza modulo 512	1 szó
04(h)	állomány hossza 512 bájtos egységekben, a fejlécet is beleértve	1 szó
06(h)	címáthelyezési tábla bejegyzéseinek száma	1 szó
08(h)	fejléc mérete paragrafusokban (1 paragrafus = 16 bájt)	1 szó
0A(h)	még szükséges paragrafusok minimális száma	1 szó
0C(h)	még szükséges paragrafusok maximális száma	1 szó
0E(h)	veremszegmens relatív kezdete paragrafusokban	1 szó
10(h)	veremmutató tartalma program indulásakor	1 szó
12(h)	EXE állomány ellenőrző összege	1 szó
14(h)	utasításszámláló (IP) tartalma a program indulásakor	1 szó
16(h)	kódszegmens relatív kezdete paragrafusokban	1 szó
18(h)	címáthelyezési tábla első bejegyzésének relatív címe	1 szó
1A(h)	overlay szintek száma (rezidens program esetén 0)	1 bájt
1B(h)	változó hosszúságú foglalt terület	
	címáthelyezési tábla	
	változó hosszúságú foglalt terület	
program	kódszegmens adatszegmens veremszegmens	

18. ábra: EXE állomány fejlécének szerkezete

Ahhoz, hogy egy EXE program betöltésének folyamatát, majd a betöltését követő tárbeli elhelyezkedését pontosan értsük, vizsgáljuk meg részletesen az állomány lemezen tárolt képét, valamint a fejléc és a program közötti kapcsolatot.

(Bár az állomány és a program kifejezést általában azonos értelemben használják – így mi is –, itt mégis célszerűnek látszik a megkülönböztetésük. Állomány alatt most azt értjük, ami az adott néven a lemezen található, vagyis az állomány fejlécét és az állomány programrészét együttvéve. Ettől annyiban tér el a program, hogy ez az állománynak csak az a része, ami a betöltésekor ténylegesen a tárba kerül.)

A következőkben a fejléc és a program közötti kapcsolatot egy konkrét példán keresztül vizsgáljuk meg.

Példaként az I. rész 4.1. alfejezetében bemutatott MAKDEMO.EXE állományt választottuk, amelynek lemezen tárolt képét a 19. ábra szemlélteti. (A dumpok szokásos ábrázolásától eltérően itt az egyes bájtokhoz tartozó ASCII-karaktereket nem tüntettük fel.)

Amint látható, az állomány alapvetően két részre oszlik: fejlécre és programra. Mivel sem a fejléc, sem a program hossza nem előre megadott hosszúságú, mindkét rész relatív értékeket tartalmaz, amelyek a saját kezdetükhöz képesti eltolási címek. Ezért mind a fejléc, mind a program 00(h) címtől kezdődik.

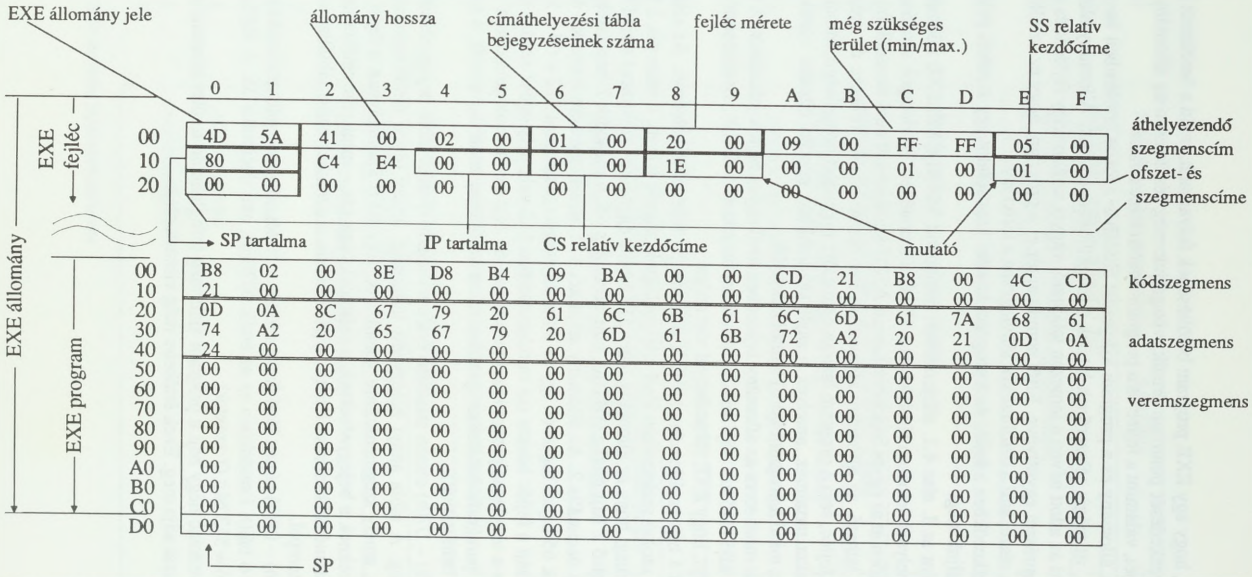
Vegyük most sorra az állomány fejlécében lévő egyes bájtok jelentését. A 00(h) – 01(h) címen álló első két bájt az EXE állományok jele (5A4D(h)). Betöltéskor a DOS erről ismeri fel, hogy EXE állománnyal van dolga.

A következő 4 bájt az állomány hosszát adja meg, a fejlécet is beleértve. Az első 2 bájt az állomány utolsó szektorában lévő értékes bájtok számát jelenti, a második 2 bájt pedig azt adja meg, hogy az állomány hány 512 bájtos egységre (szektorra) terjed ki. Esetünkben az első 2 bájt értéke 0041(h), ami decimális 65, a második 2 bájt értéke pedig 0002(h), ami decimális 2. Az állomány tehát két 512 bájtos egységre terjed ki, de a másodikból csak 65 bájtot foglal el. Így az állomány tényleges hossza 512 + 65 = 577 bájt, amiből 512 bájt a fejléc hossza (ez minden esetben 512 valamely egész számú többszöröse), vagyis a tényleges programkód mindössze 65 bájt (ez is csak „majdnem" igaz: a szegmensek paragrafushatárokra igazítása miatt valójában ennél kevesebb; a számítást később még bemutatjuk).

A 06(h) – 07(h) címen található érték a címáthelyezési tábla bejegyzéseinek számát adja meg. A tábla annyi bejegyzést tartalmaz, ahány olyan címhivatkozás van a programban, amely szegmenscímre vonatkozik. Egy EXE állományban a fejléc mérete valójában ezeknek a bejegyzéseknek a számától függ (de – mint már említettük – csak 512 egész számú többszöröse lehet). A példánkban a címáthelyezési táblában csak egy bejegyzés szerepel.

A 08(h) – 09(h) címen lévő érték a fejléc méretét adja meg paragrafusokban (1 paragrafus = 16 bájt). Esetünkben ez az érték 0020(h), ami decimális 32. A fejléc hossza tehát 32 x 16 = 512 bájt (1 szektor).

A következő négy bájt a program által még lefoglalandó terület minimális és maximális hosszát adja meg. Ezek értékeire még visszatérünk.



19. ábra: A MAKDEMO.EXE állomány lemezen tárolt képe

A 0E(h) – 0F(h) címek a veremszegmensnek a program kezdetétől számított relatív kezdőcímét tartalmazzák paragrafusokban. Példánkban ez az érték 0005(h), ami decimális 5 paragrafus, azaz 80 bájt (50(h)). Valóban, amint az ábrán látható, a veremszegmens a program kezdetétől számított 50(h) címen kezdődik.

A következő két bájt értéke a veremmutatónak a program indulásakor felvett tartalmát adja meg. Mivel a példaprogramunkban a verem részére 64 szó (128 bájt) hosszú területet írtunk elő, a program indulásakor pedig a veremmutató a verem tetejére mutat, a veremmutató értéke 128, vagyis 0080(h) a veremszegmensben.

A következő két bájt egy ellenőrző összeget (checksum) tartalmaz, amit a LINK számít ki az állomány hosszából. Ezzel nekünk nem kell törődnünk.

A 14(h) – 15(h) címen lévő bájtok az utasításszámlálónak (IP) a program indulásakor felvett értékét adják meg. Ezt az értéket az assembler forrásprogram END direktívája állítja be, és mint ofszetcím a kódszegmens szegmenscímével együtt a program belépési pontját (az első végrehajtandó utasítás címét) határozza meg. A fejléc e két bájtjának értéke 0000(h), jelezve, hogy a példaprogramunk végrehajtása a kódszegmens első utasításával kezdődik.

A 16(h) – 17(h) címen álló bájtok a kódszegmensnek (CS) a program kezdetétől számított relatív kezdőcímét tartalmazzák paragrafusokban. Példánkban ez az érték ugyancsak 0000(h). Valóban, amint az ábrán látható, a kódszegmens a program 0000(h) ofszetcímén kezdődik. (Ez várható is volt, hiszen a forrásprogramban ez az elsőként definiált szegmens.)

A következő, 18(h) – 19(h) címen álló bájtok egy mutatót képeznek, amely a fejlécen belül a címáthelyezési tábla kezdőcímére (vagyis a tábla első bejegyzésére) mutat. A példánkban ennek a mutatónak az értéke 001E(h), jelezve, hogy a címáthelyezési tábla a fejléc kezdetétől számított 001E(h) címen kezdődik.

Az 1A(h) címen lévő bájt értéke az overlay szintek számát határozza meg (rezidens program esetén 0).

A fejléc ezt követő területe egészen a címáthelyezési tábla kezdetéig, majd azt követően is változó hosszúságú, és fenntartott adatterület. A címáthelyezési tábla mindegyik bejegyzése négy bájtból, azaz két szóból áll: az első szó az áthelyezendő szegmenscím ofszetcímét, a második a szegmenscímét tartalmazza (azáltal, hogy az áthelyezendő szegmenscím szegmenscíme is változtatható, egy program mérete több szegmensre is kiterjedhet). Példánkban az áthelyezendő szegmenscím szegmenscíme 0000(h), ami megegyezik a fejléc 16(h) – 17(h) címén lévő kódszegmens kezdőcímével (vagyis maradunk ugyanazon kódszegmensen belül), az ofszetcíme pedig 0001(h) (vagyis az áthelyezendő szegmenscím – amint az ábrán is látható – a program 00(h) ofszetcímen kezdődő kódszegmens 01(h) címén kezdődik). Az ezen és a következő címen lévő 02(h) és 00(h) értékű bájt a program betöltésekor kapja meg a megfelelő szegmenscím értékét. Nézzük meg, hogyan is történik ez!

A két, 02(h) és 00(h) értékű bájt a program lemezen tárolt képében egyelőre csak egy 0002(h) értékű logikai szegmenscímet képez, amely a szegmentált címképzésnek megfelelően 00020(h) fizikai címnek felel meg. Amikor a programot a tárba töltjük, az – a tár foglaltságától és a betöltés módjától függően – valamely, előre nem meghatározott fizikai címtől kezdődően töltődik be. Hogy megint csak konkrétek legyünk: a mi ese-

tünkben a DEBUG.COM a példaprogramot a 37640(h) fizikai címtől kezdődően töltötte be (más esetben ez természetesen más cím is lehet). Most már kiszámítható az áthelyezendő logikai szegmenscímnek megfelelő fizikai szegmenscím is:

a betöltés fizikai kezdőcíme:		37640(h)
a PSP hossza:	+	100(h)
az áthelyezendő szegmenscímnek megfelelő fizikai ofszetérték:	+	20(h)
az áthelyezendő szegmenscím fizikai címe:		37760(h)

Az így kiszámított fizikai cím ismét csak a szegmentált címképzésnek megfelelően természetesen a 3776(h) szegmenscímet jelenti. Tehát a program betöltésekor a lemezen tárolt 02(h) és 00(h) bájtok értéke 76(h), ill. 37(h) lesz. Azt már csak megjegyezzük, hogy az ebből képzett 3776(h) cím a példaprogramunk adatszegmensének kezdőcíme. (Hangsúlyozzuk: a fenti fizikai címek természetesen mások is lehetnek!)

Miután az EXE programon belüli szegmenshivatkozások megkapták a tényleges címeiket, az EXEC funkció a DS és ES szegmensregisztereket a PSP kezdetére állítja. Ezen keresztül az EXE program is egyszerűen hozzáférhet a környezeti blokk címéhez vagy a parancssorban átadott paraméterekhez. E kezdőértékek beállítása után megkezdődhet a program végrehajtása.

Azért, hogy a jövőbeli DOS verziókkal biztosítsuk a kompatibilitást, az EXE program befejezéséhez a DOS 21(h) megszakítás 4C(h) funkciójának meghívását kell előnyben részesíteni.

Korábbi ígéretünknek megfelelően térjünk most vissza az EXE programok által elfoglalandó tárterületekre. Természetesen egy EXE program betöltéséhez is megfelelő területet kell biztosítani a tárban. Ez a COM programokhoz képest némileg eltérő módon történik. Az EXE program egyes szegmenseinek méretét, és így a programnak a tárban lefoglalandó együttes nagyságát a DOS EXEC betöltő funkciója az EXE állományból olvashatja ki, a szükséges területet pedig egy másik funkció meghívásával igényelheti. A tulajdonképpeni programhosszon túlmenően azonban az EXE program még további tárterületet is igényelhet (pl. a kódszegmensen belüli munkaterületek vagy a veremtár számára, netán a programból egy újabb program meghívásához). Az utolsóként említett eset kivételével a még lefoglalandó terület mérete a program fejlécében lévő két mező tartalmától függ. Ez a két mező az EXE állomány fejlécének 0A(h) – 0D(h) ofszetcímén található. A 0A(h) és a 0B(h) címek a program által még lefoglalandó minimális tárterület nagyságát, a 0C(h) és a 0D(h) címek pedig a maximális hosszát adják meg paragrafusokban (ezeket a területeket az angol nyelvű szakirodalom MIN ALLOC, ill. MAX ALLOC névvel jelöli).

Az EXEC betöltő funkció először a maximális paragrafusoknak megfelelő területet igyekszik lefoglalni. Ez alapértelmezés szerint FFFF(h) paragrafusnak megfelelő hosszat, azaz 1 Mbájtnyi területet jelent, ami meglehetősen illuzorikus. Ha ekkora terület lefoglalása nem lehetséges, akkor beéri a tár rendelkezésre álló részével, ami azonban

nem lehet kisebb a minimális paragrafusokkal megadott területnél. A két mező nagyságát egyébként nem a LINK programszerkesztő, hanem az assembler fordítóprogam (compiler) határozza meg.

Mivel a maximálisan lefoglalandó terület értékével nincs mit kezdenünk, nézzük meg, mit takar a betöltendő program által még igényelt terület? Visszatérve a példaprogramunk lemezen tárolt képét bemutató ábrára, láthatjuk, hogy a fejléc 0A(h) és 0B(h) ofszetcímén a 09(h) és a 00(h) érték áll. Ez azt jelenti, hogy a betöltendő program a saját hosszán túl még további 9 paragrafusnak, azaz 9 * 16 = 144 bájtnak megfelelő területet kíván lefoglalni. Vajon miért pont ennyit? A példaprogramunk és az ábra alapján számoljunk utána!

Először vegyük figyelembe azt, hogy a példaprogram szegmenskijelöléseit a PARA(grafus) paraméterrel paragrafushatárokra illesztettük, azaz mindegyik szegmens paragrafushatáron kezdődik. Az ábrára nézve megszámolható, hogy a program kódrésze igazából csak 17 bájtból áll, és a paragrafushatárra való illesztés miatt nem kevesebb, mint 15 bájtot „elpocsékoltunk". Hasonló a helyzet az adatszegmenssel is: ez 33 „értékes" bájtot tartalmaz, az illesztés miatt azonban 15 bájt itt is kárbavész.

Hát akkor most valójában milyen hosszú is a programunk? A kódrész 17 + 15 bájtját biztosan be kell számítani, hiszen az „elpocsékolt" 15 bájttal nincs mit kezdeni, hiszen ezt még az adatszegmens követi, és az ebben lévő információ csak a lemezről olvasható be. Ez az információ – mint láttuk – 33 bájt hosszú. Ami viszont a lemezen ez után következik, annak a tárban futó program számára már nincs jelentősége, hiszen itt már csak vagy kihasználatlan 00(h) bájtok, vagy valamilyen más program által hátrahagyott „hulladék" állhat. A program a veremre vonatkozó információit sem innen kapja. Röviden: a program tényleges hossza a kódrész 17 + 15 = 32 bájtjának és az adatrész igazából kihasznált 33 bájtjának az összege, vagyis 65 bájt. A DOS azonban ehhez még hozzászámítja az EXE programok előtt álló 512 bájt hosszú fejlécet is (végül is a fejléc is helyet foglal el a lemezen), ezért a katalógusbejegyzések az ezzel megnövelt hosszat tekintik a program hosszának. A példaprogramunk hossza ezek szerint 512 + 65 = 577 bájt.

E rövid, de nélkülözhetetlen kitérő után térjünk vissza a lefoglalandó tár méretére. A számítás most már nagyon egyszerű lesz. Nézzük meg, betöltés után hogyan helyezkedik el a programunk a tárban:

kódszegmens	értékes bájtok száma	17	
	paragrafushatárra való kerekítés miatti „kitöltő" bájtok száma	15	
	Összesen:	17 + 15	= 32
adatszegmens	értékes bájtok száma	33	
	paragrafushatárra való kerekítés miatti „kitöltő" bájtok száma	15	
	Összesen:	33 + 15	= 48
verem-szegmens	programban definiált bájtok száma:	128	
	Összesen:		128
tárbeli program által igényelt összes terület bájtokban:			208
lemezen tárolt program hossza bájtokban:			65
még lefoglalandó terület bájtokban:		208 - 65	= 143
paragrafushatárokra kerekítve:			144
még lefoglalandó terület paragrafusokban:		144 : 16	= 9 paragrafus

Úgy gondoljuk, hogy az előbbi számításhoz nem kell további magyarázat.

Az előzőek alapján – egyszerű esetet feltételezve – a DOS ki tudja számítani, hogy a betöltendő program számára mekkora tárterületet kell lefoglalnia. Ennél bonyolultabb a dolog akkor, amikor egy EXE program a futása során további programot vagy programokat akar meghívni. Ahhoz, hogy ennek sikerében reménykedhessen, az általa feleslegesen lefoglalt tárterületet „illik" felszabadítania.

Az itt következő példaprogram ismét egy olyan rutint mutat be, amely a meghívó program által igénybe vett tárterületet a minimumra csökkenti. A program egy kód-, adat- és veremszegmenst tartalmaz. Amennyiben egy felhasználói programban használt szegmensek nem haladják meg külön-külön a 64 kbájtot, akkor ez a példaprogram minden ilyen EXE program vázaként is használható.

Miután megírtunk egy assembler forráskódú EXE programot, futtatás előtt természetesen egy assembler fordítóval le kell fordítani (mi a Microsoft MASM makro-assemblerét használtuk). Ezt követően a lefordított programot a LINK programmal össze kell szerkeszteni. Ha a programban nincs hiba, akkor a szerkesztést követően létrejön a futtatható EXE állomány. Ezek a lépések egy MINTA.ASM nevű assembler forráskódú programra így írhatók le:

```
masm minta;
link minta;
```

Ha a rendszer ezeket a lépéseket rendben végrehajtotta, akkor a MINTA.EXE programot a DOS-ból a MINTA név beírásával közvetlenül indíthatjuk.

```
;
;                          E X E V A Z . A S M
;
;  Funkciója: egy EXE program betoltésekor feleslegesen lefog-
;             lalt tárterület felszabadítása.
;
;
;— KÓD ————————————————————————————————————————————————————————

Kod     segment para 'CODE'     ; A kódszegmens definiálása.

        assume cs:kod, ds:adat, ss:verem

                                ; Szegmensregiszterek hozzárendelése
                                ; a szegmensekhez.

prog    proc far                ; Itt kezdődik a tulajdonképpeni
                                ; főprogram. A program indításakor
                                ; ide kerül a vezérlés.

        mov  ax,adat            ; Adatszegmens cím --> DS.
        mov  ds,ax

        call freemem            ; Tárterületet felszabadító rutin
                                ; hívása.

;————————————————————————————————————————————————————————————————

                                ; Itt helyezhető el a programkód. Most
                                ; egy karakterláncot megjelenítő
                                ; kódot helyeztünk ide, hogy a prog-
                                ; ram végrehajtásakor lássunk is
                                ; valamit. A karakterlánc az adat-
                                ; szegmensben van.

;————————————————————————————————————————————————————————————————

kiir    proc near               ; A karakterláncot megjelenítő
                                ; eljárás.

        mov ah,9                ; Karakterláncot megjelenítő
        mov dx,offset karlanc   ; funkció és megszakítás hívása.
        int 21h

kiir    endp                    ; A kiir eljárás vége.

        mov  ax,4C00h           ; 21(h) megszakítás, a program a 0
        int  21h                ; hibakóddal ér véget.

prog    endp                    ; A prog eljárás vége.

;— FREEMEM : A felesleges tárterület felszabadítása ———————————————
;— BE      : ES = a PSP címe
;— KI      : —
;
;   Mivel megállapodás szerint a veremszegmens az EXE állomány utol-
;   só szegmense, és a DOS az állomány betoltésekor az ES regisztert
;   az állomány elé elhelyezett PSP elejére, SS:SP-t pedig az állomány
;   végére állítja, ezekből az adatokból az állomány hossza kiszámít-
;   ható.
;
;— REG     : AX, BX, CL és FLAGS értéke megváltozik.
```

```
freemem  proc near
         mov bx,ss                  ; A két szegmens egymásból való kivo-
         mov ax,es                  ; nása megadja azoknak a paragrafu-
         sub bx,ax                  ; soknak a számát, amelyet az állo-
                                    ; mány a PSP kezdőcímétől a veremtár
                                    ; kezdetéig lefoglal.

         mov ax,sp                  ; Mivel a veremmutató a szegmens vé-
         mov cl,4                   ; gére mutat, a tartalma megadja a
         shr ax,cl                  ; verem hosszát. Ezt az előbb ki-
         add bx,ax                  ; számított paragrafusokhoz hozzá
                                    ; kell adni.

         inc bx                     ; A biztonság kedvéért kerekítés
                                    ; paragrafushatárra.

         mov ah,4ah                 ; Tárterületet változtató funkció
         int 21h                    ; és megszakítás hívása.

         ret                        ; Visszatérés a meghívó programhoz.

freemem  endp                       ; A freemem eljárás vége.

kod      ends                       ; A kódszegmens vége.

;— KÓDSZEGMENS VÉGE ——————————————————————————————————————

;— ADATSZEGMENS ——————————————————————————————————————

adat     segment para 'DATA'        ; Az adatszegmens definiálása.

                                    ; Itt helyezhetők el az adatok, puf-
                                    ; ferek, változók és egyéb munkaterü-
                                    ; letek. Most egy karakterláncot he-
                                    ; lyeztünk ide, amit a program kódré-
                                    ; szében kiíratunk, hogy végrehajtá-
                                    ; sakor lássunk is valamit.

CR       equ 13                     ; Kocsivissza ASCII kódja.
LF       equ 10                     ; Soremelés ASCII kódja.
KEND     equ "$"                    ; Karakterlánc vége jelzés.

karlanc  db CR,LF,"Ez a program felszabadítja a program betoltésekor"
         db CR,LF,"feleslegesen lefoglalt tárterületet.",CR,LF,KEND

adat     ends                       ; Az adatszegmens vége.

;— ADATSZEGMENS VÉGE ——————————————————————————————————————

;— VEREM ——————————————————————————————————————————————————

verem    segment para stack         ; A veremszegmens definiálása.

         dw 256 dup (?)             ; A verem hossza 256 szó.

verem    ends                       ; A veremszegmens vége.

;— VEREMSZEGMENS VÉGE —————————————————————————————————————

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

end      prog                       ; Az assembler forrásprogram vége.
                                    ; A program végrehajtása a prog
                                    ; eljárással kezdődik.
```

6.5. Karakterek be- és kivitele DOS-szal

A leggyakrabban és legegyszerűbben használható DOS funkciók közé tartoznak a karakterek be- és kivitelére szolgáló funkciók. A következőkben ezeket vizsgáljuk meg.

A karakterek be- és kivitelére szolgáló DOS funkciók segítségével a billentyűzethez, a képernyőhöz, a nyomtatóhoz és a soros interfészhez lehet fordulni. Maguk a funkciók két csoportra oszthatók: egyik részük a CP/M, másik részük a UNIX operációs rendszer ekvivalens funkcióival kompatibilis. Az első csoportba tartozókat hagyományos funkcióknak nevezzük. E csoport azonban a CP/M korábbi funkcióin kívül még további funkciókat is tartalmaz, amelyek egyszerűbbé teszik a CP/M programok DOS-környezetbe való átültetését.

A UNIX-kompatibilis funkciókat „handle"-funkcióknak nevezzük, mert a meghívásukkor a funkció részére egy numerikus értéket kell átadni (ezt a numerikus értéket nevezi a DOS-terminológia handle-nek). A DOS ezzel a handle-lel egy meghatározott egységet (perifériát) azonosít, és az adatkivitelre, ill. -bevitelre vonatkozó utasítást ezen az egységen hajtja végre.

Ha egy kész programot az említett két operációs rendszer valamelyikéből DOS-környezetbe kell átültetnünk, akkor a funkcióknak azt a csoportját kell igénybe vennünk, amelyik a forrás operációs rendszerrel kompatibilis. Amikor viszont új programot fejlesztünk, akkor a két csoport bármelyikét választhatjuk. Figyelembe véve azonban azt az egyértelmű tendenciát, amely a DOS-nak a CP/M-től való eltávolodása, és a UNIX felé történő közeledése irányába mutat, az új programok fejlesztésénél a handle-funkciókat kell előnyben részesíteni.

6.5.1. Handle-funkciók

Ebben a fejezetben azt vizsgáljuk, hogy miként lehet a handle-funkciókkal az előbb említett egységeken karaktereket be-, ill. kivinni. Ezzel összefüggésben azonban nem hagyható szó nélkül, hogy a handle-funkciókkal állományok is kezelhetők, hiszen az állományoknak és a perifériáknak ez a közös bánásmódja tekinthető a handle-műveletek egyik alapvető újszerűségének: a DOS ugyanolyan állományoknak tekinti a billentyűzetet, a képernyőt, a nyomtatót és a soros interfészt, mint a „rendes" állományokat, amelyeknek nevük van, és amelyeket a hozzájuk rendelt handle-n keresztül megnyithat, műveleteket végezhet rajtuk, és lezárhat. Ennek megfelelően a DOS ezeknek az egységeknek nevet ad, amelyekhez handle-ket rendel. Így amikor a DOS egy művelet végzésekor megkap egy handle-t, ebből tudja, hogy egy állományhoz vagy valamelyik egységhez kell-e fordulnia.

Az egységek azonosító nevei:

CON	billentyűzet és képernyő
AUX	soros interfész

PRN nyomtató
NUL nem létező egység, megszólításakor semmi sem történik

Az AUX, PRN és a NUL egységek esetén mind a kivitel, mind pedig a bevitel az illető perifériára vonatkozik (nyomtatóról természetesen nem lehet adatot bevinni). A CON esetén a kivitel a képernyőre, a bevitel a billentyűzetre vonatkozik.

Amikor a DOS átadja a vezérlést egy programnak, akkor előre meghatároz öt handle-t, amelyeken keresztül a program az egyes egységekhez hozzáférhet. Az öt handle öt egységhez van hozzárendelve a következők szerint:

0 standard beviteli egység (CON)
1 standard kiviteli egység (CON)
2 standard hibajelző egység (CON)
3 standard interfész (AUX)
4 standard nyomtató (PRN)

Nézzük meg egy példán, mit jelent egy ilyen hozzárendelés.
Ha egy program a felhasználótól adatokat vár, akkor a handle-funkció meghívásakor a 0-s számú handle-t adja meg, mert ezen keresztül lehet a standard beviteli egységhez fordulni. Alapértelmezés szerint ez a billentyűzet, tehát a program a felhasználó adatait a billentyűzetről kapja. Mivel azonban a standard bevitel forrását a felhasználó is átirányíthatja (pl. egy lemezes állományba), elképzelhető, hogy a program által várt adatok nem a billentyűzetről, hanem egy lemezes állományból származhatnak anélkül, hogy erről a program tudomást szerezne. Ezekről az átirányításokról, ill. letiltásukról a későbbiekben még lesz szó.

Mielőtt részletesen kitérnénk az egyes perifériákra, ismerkedjünk meg azokkal a funkciókkal, amelyek segítségével valamennyi egységhez egyformán lehet fordulni. Ahhoz, hogy valamelyik egységre adatokat küldjünk, a 21(h) megszakítás 40(h) funkciója használható. A funkció meghívásához az AH regiszterben meg kell adni a funkció számát (jelen esetben 40(h)-t), a BX regiszterben pedig az egységet azonosító handle-t. Ha például egy hibaüzenetet akarunk kiíratni, akkor handle-ként a 2-t kell megadni, mert ez jelöli a standard hibajelző egységet (ez egyébként nem irányítható át, úgyhogy ez mindig a képernyőt jelenti). A megszakítás meghívása előtt a CX regiszterbe be kell írni a megjelenítendő karakterek számát. Maguknak a karaktereknek ASCII-karakterlánc formájában egy pufferben kell lenniük. A puffer szegmenscímét a DS, ofszetcímét a DX regiszternek kell tartalmaznia.

A funkció meghívása után az átvitelbit (Carry flag) jelzi, hogy az adatok átvitele sikeres volt-e. Ha a jelzőbit értéke 0 (más szóval törölt vagy alacsony), akkor az átvitel sikerült, és az átvitt karakterek (bájtok) számát az AX regiszter tartalmazza. Amennyiben ez a szám 0 lenne, akkor ez azt jelenti, hogy megszólított eszközön (hajlékony- vagy merevlemez) már nem volt hely a karakterek fogadására. Ha viszont az átvitelbit értéke 1 (magas), akkor az adatok átvitele nem sikerült. Ebben az esetben az AX regiszterbe egy hibakód kerül. Az 5-ös kód azt jelenti, hogy a rendszer az egységhez való hozzáférést elutasította, a 6-os pedig azt, hogy a megadott handle érvénytelen volt, vagy – állományra vonatkozóan – az állomány nem volt megnyitva. (Ez utóbbi hiba csak állomá-

nyokkal kapcsolatban fordulhat elő, mert a standard egységekhez tartozó handle-ket külön nem kell megnyitni.)

A 40(h) funkcióhoz hasonlóan működik a 3F(h) funkció, amellyel egy egységről adatokat lehet beolvasni. A regiszterek szereposztása ugyanaz, mint a 40(h) funkciónál. Az AH regiszter a funkció számát, a BX regiszter a handle-t, a CX regiszter a beolvasandó karakterek számát, a DS:DX regiszterpár pedig annak a puffernek a címét tartalmazza, ahová az adatokat be kell olvasni.

A funkció meghívása után itt is az átvitelbit jelzi, ha a végrehajtás során hiba történt, és most is az AX regiszter tartalmazza a hiba kódját. A lehetséges 5-ös és 6-os hibakódoknak ugyanaz a jelentése, mint a 40(h) funkciónál. Az átvitelbit 0 értéke a művelet sikeres végrehajtását jelzi, és ekkor az AX regiszter a beolvasott, és a pufferben elhelyezett karakterek számát tartalmazza. Ha a regiszterben ez a szám 0, akkor ez azt jelenti, hogy az adatokat egy állományból kellett volna beolvasni, de az állományban már nem voltak adatok.

Már említettük, hogy az egyes egységekhez való hozzáférés során fennáll annak a veszélye, hogy a felhasználó az adatbevitelt vagy -kivitelt átirányíthatja úgy, hogy azok az adatok, amelyeket a program a billentyűzetről vár, nem onnan, hanem pl. egy állományból kerülnek a programba. Ez az átirányíthatóság úgy küszöbölhető ki, hogy megnyitunk egy új handle-t, amelyen keresztül biztosítható, hogy a kapcsolat csak azzal az egységgel jöhessen létre, amellyel az adatcserét le kell bonyolítani.

Az új handle a 21(h) megszakítás 3D(h) funkciójával nyitható meg. Az AH regiszterben a funkció számát, az AL regiszterben pedig a hozzáférési módot kell átadni. Ez utóbbi azt mondja meg, hogy az egységre adatokat akarunk-e küldeni, vagy hogy onnan adatokat akarunk-e fogadni. A 0 érték az adatok fogadását, vagyis az olvasást, az 1 érték az adatok küldését, vagyis az írást, míg a 2 mind az írást, mind az olvasást lehetővé teszi. A megszólított egység nevét egy puffer tárolja, amelynek szegmens-, ill. ofszetcímét a DS:DX regiszterpárnak kell tartalmaznia. Ahhoz, hogy a DOS a megszólítandó egység nevét helyesen ismerje fel, a nevet nagybetűkkel kell írni (pl. CON, PRN), amelyet a pufferben egy végjelnek (a 0 ASCII-kódnak) kell lezárnia.

Azt, hogy a handle megnyitása sikeres volt-e, a funkció meghívása után az átvitelbit értékéből lehet megállapítani. Ha a jelzőbit értéke 0, akkor a megnyitás sikerült, és az AX regiszter az új handle-t tartalmazza, amelyen keresztül az összes, ezután következő művelettel a kívánt egységhez lehet fordulni. Ha viszont a bit értéke 1, akkor a megnyitás nem sikerült, és az AX regiszterben a hibakód van.

Ha ezen a handle-n keresztül a továbbiakban már nem akarunk az egységhez fordulni (tehát legkésőbb a program végén), akkor a handle-t a 21(h) megszakítás 3E(h) funkciójával le kell zárni. Ha a megnyitott handle-ket nem zárnánk le, akkor a munka során előfordulhat, hogy egyidejűleg annyi lesz a megnyitott handle, hogy a rendszer újabbak megnyitását már nem engedélyezi. Ezért minden, általunk megnyitott handle-t le is kell zárni. Ehhez a funkció számát az AH regiszterbe, a lezárandó handle-t pedig a BX regiszterbe kell tölteni. Ha a handle lezárása sikerült, akkor az átvitelbit értéke zérus lesz, ellenkező esetben a művelet sikertelen volt (akár azért, mert a handle már le lett zárva, akár azért, mert a megadott handle nem is létezett).

Ezzel a funkcióval természetesen a 0 – 4. számú, előre definiált handle-k is lezárhatók. Ebben az esetben azonban nem árt a különleges óvatosság, mert pl. a 0-ás számú handle (standard beviteli egység) lezárásával a rendszer a billentyűzetről nem fogad adatokat, ami a rendszer működtetését gyakorlatilag lehetetlenné teszi. Az eddigiekben ismertetett funkciók valamennyi standard egységre egyformán vonatkoztak. Ezek után nézzük meg az egyes egységekre jellemző tulajdonságokat.

Billentyűzet

A billentyűzet természetesen csak adatbeviteli egységként értelmezhető, hiszen erre írni nem lehet. A billentyűzet olvasásánál a DOS kétféle üzemmódot különböztet meg: bináris (raw) és ASCII (cooked) üzemmódot. ASCII módban a DOS minden egyes, valamelyik egységnek kiküldött, vagy valamelyik egységről beérkezett karaktert megvizsgál arra vonatkozóan, hogy az illető karakter „közönséges" karakter vagy speciális vezérlőkarakter-e. Ha az illető karakter közönséges karakter, akkor egyszerűen továbbítja vagy fogadja, ha viszont vezérlőkarakter, akkor a neki megfelelő feladatot végrehajtja (pl. soremelés, lapdobás stb.), vagyis a karaktereket egyfajta módon megszűri. Ezzel szemben bináris módban a karaktereket nem vizsgálja, és az esetleges vezérlőkarakterekhez tartozó feladatokat nem is hajtja végre. Bár a DOS a karakter be- és kivitelére szolgáló egységeket alapértelmezés szerint ASCII módban vezérli, van lehetőség arra, hogy programból átkapcsoljunk bináris módba.(21(h) megszakítás, 44(h) funkció.) A következőkben ezt vizsgáljuk meg.

A billentyűzet példáján jól szemléltethető a két üzemmód közötti különbség. Tegyük fel, hogy a billentyűzetről 30 karaktert akarunk beolvasni ASCII módban. A karakterek beírásakor a DOS néhány vezérlőbillentyű, mint pl. a visszaléptető vagy törlőgomb (backspace) segítségével lehetővé teszi a beírás közbeni szerkesztést úgy, hogy a beírt karaktereket először egy belső, 128 bájt hosszú pufferben tárolja, és egyúttal a képernyőn is megjeleníti (ezt nevezik echo-nak). Beírás közben a DOS a Ctrl-C és a Ctrl-Break billentyűket is figyeli, és lenyomásukkor az adatátvitelt megszakítja. Ebben az üzemmódban a Ctrl-P és a Ctrl-S billentyűkombinációnak is különleges jelentősége van. A Ctrl-S lenyomásakor a program futása mindaddig leáll, míg egy billentyűt ismét le nem nyomunk. A Ctrl-P lenyomására a kivitel a képernyőről a nyomtatóra kerül. Ezt csak a Ctrl-P ismételt lenyomásával lehet megszüntetni. Ha lenyomjuk az ENTER/RETURN billentyűt, akkor az addig beírt karakterek (amennyiben a számuk nem haladja meg a 30-at) a DOS belső pufferéből átmásolódnak a program beviteli pufferébe. A szerkesztéshez használt vezérlőkarakterek figyelmen kívül maradnak.

Ezzel ellentétben bináris módban az összes beírt karakter – a vezérlőkaraktereket is beleértve – átkerül a beviteli funkciót meghívó program pufferébe. A DOS a RETURN billentyű lenyomását sem várja meg, hanem pontosan 30 karakter beírása után átadja a vezérlést a meghívó programnak. (A 30 karaktert viszont kivárja, még akkor is, ha történetesen már második karakterként egy RETURN érkezett volna.)

Képernyő

Karakterek képernyőre írásához általában a standard kiviteli egységhez rendelt 1-es számú handle-t használjuk. Mivel azonban ez az egység átirányítható, nem biztos, hogy az ezen a handle-n keresztül kiküldött adatok feltétlenül a képernyőre kerülnek. Ezzel szemben a standard hibajelző egység (ehhez a 2-es számú handle tartozik) nem irányítható át, ezért az ezen a handle-n keresztül átadott karakterek mindig a képernyőre íródnak. Minden olyan esetben tehát, amikor a karaktereknek a képernyőn, és csak ott kell megjelenniük, ajánlatos ennek a handle-nek a használata.

A DOS alapértelmezés szerint a képernyőt is ASCII módban kezeli, ami azt jelenti, hogy a kiírás előtt minden karaktert megvizsgál a Ctrl-C, ill. a Ctrl-Break lenyomására vonatkozóan. Éppen ez a vizsgálat néha annyira lelassíthatja a műveletet, hogy érdemes lehet bináris módra átkapcsolni. Az erre vonatkozó programot a fejezet végén mutatjuk be.

Nyomtató

A billentyűzettől és a képernyőtől eltérően a nyomtatóra történő kiírás (legalábbis felhasználói szinten) nem irányítható át. Ez alól kivételt képez a párhuzamos nyomtatóra történő kiírás átirányítása soros nyomtatóra. (Pontosabban fogalmazva a kivitel a párhuzamos portról valamelyik soros portra irányítható át, amelyhez soros nyomtató csatlakoztatható.)

Arra azonban van lehetőség, hogy a kiírandó karaktereket még a nyomtatóra kerülésük előtt egy állományban tároljuk, vagy egy hálózaton belül egy másik nyomtatóra küldjük. Ez azonban a DOS feladata, és a program számára észrevétlen marad.

A nyomtatóra íráshoz a 4-es számú handle áll rendelkezésünkre, amelyen keresztül az alapértelmezés szerinti nyomtatóhoz lehet fordulni. Ha viszont a PC-hez csatlakoztatható három nyomtató közül egy, meghatározott nyomtatót akarunk elérni, akkor ehhez egy handle-t kell megnyitni. Az 1-es számú nyomtató neve LPT1, a 2-es számúé LPT2, a 3-asé LPT3. A DOS alapértelmezés szerint az LPT1 nevű nyomtatót tekinti a PRN egységnek. Ha tehát a kivitelt a PRN-re irányítjuk, akkor ez az LPT1 nyomtatón jelenik meg. A handle megnyitásakor egységnévként az LPT1, LPT2 vagy az LPT3 nevet kell megadni. (Megjegyezzük, hogy a ROM BIOS, amely ugyancsak tartalmaz olyan funkciókat, amelyeken keresztül a nyomtatóhoz hozzá lehet férni, a nyomtatók azonosítására a 0, 1 és 2 számokat használja.)

Soros interfész

A soros interfészhez (amelyet aszinkron kommunikációs adapternek vagy soros portnak is neveznek) a DOS által előre megadott 3-as számú handle-n keresztül lehet hozzáférni. Ehhez a handle-hez van a standard interfész (AUX) hozzárendelve. A DOS az IBM PC/XT/AT gépeken két soros interfészt támogat, amelyek neve COM1 és COM2. (Csak megjegyzésként: a legújabb, PS/2 rendszereken hármat.) A soros interfészről kimenő vagy az arra érkező adatok nem irányíthatók át (még a párhuzamos portra sem).

Az előre megadott handle-n keresztül csak az első soros interfészhez (COM1) lehet hozzáférni. Ezért a COM2 interfészre való íráshoz vagy az onnan való olvasáshoz egy új handle-t kell megnyitni. Az íráshoz és az olvasáshoz az előző egységeknél már említett DOS funkciók használhatók.

A soros interfésszel kapcsolatban a DOS szakirodalom megemlíti, hogy néhány DOS verzióban van egy hiba: amikor a 21(h) megszakítás 3F(h) funkciójával a soros interfészt olvassuk, akkor a megszakítás visszatérésekor az AX regiszter nem az átadott karakterek számát tartalmazza, hanem ennél eggyel kevesebbet. Ilyen esetben a hiba úgy kerülhető meg, hogy a megszakítás meghívásakor a vártnál mindig nagyobb számú karaktert adunk meg, vagy úgy, hogy az interfészt a 21(h) megszakítás 44(h) funkciójával bináris (raw) módban kérdezzük le, még akkor is, ha a rendszer ebben a módban a különböző vezérlőkaraktereket, így például a Ctrl-C-t vagy az állomány vége jelet nem ellenőrzi.

6.5.2. Hagyományos funkciók

A karakterek be- és kivitelére szolgáló hagyományos funkciók a handle-funkcióktól teljesen eltérő elven alapulnak. Meghívásukkor nem kell handle-t átadni, mivel mindegyik funkció csak egy, meghatározott egységre vonatkozik. Ezért aztán olyan univerzális író/olvasó műveletek sincsenek, amelyekkel mindegyik egység egyformán elérhető lenne. A 2.0 verziótól felfelé a karakterek be- és kivitele a hagyományos funkciókon keresztül is átirányítható, de ez a program számára rejtve marad.

A következőkben bemutatjuk a különböző beviteli és kiviteli egységekkel való kapcsolattartásra szolgáló hagyományos funkciókat. Megismerkedünk meghívásuk módjával és sajátosságaikkal.

Billentyűzet

Az a hét funkció, amelyet a DOS a billentyűzettel való kapcsolattartáshoz rendelkezésünkre bocsát, többek között abban különbözik egymástól, hogy miként reagálnak a Ctrl-C vagy a CtrL-Break lenyomására, illetve hogy a bevitt karaktert kiírják a képernyőre vagy nem. Ha olyan billentyűt nyomunk le, amelyhez bővített billentyűkód tartozik (a bővített billentyűkódokról a későbbiekben még lesz szó), akkor a billentyűzetfunkciók először a 0 értéket adják vissza, jelezve a meghívónak, hogy bővített kód érkezik. A funkció következő meghívása már ténylegesen a bővített billentyűzetkódot szolgáltatja.

A funkciókat aszerint is meg lehet különböztetni, hogy egy vagy több karaktert olvasnak-e be. A 21(h) megszakítás 1, 6, 7 és 8-as funkcióival a billentyűzetről egyetlen karakter olvasható be. A funkció számát mindegyik esetben az AH regiszterben kell átadni. A meghívást követően a AL regiszterbe a beolvasott karakter kódja kerül. A 6-os funkciótól eltérően az 1-es, 7-es és 8-as funkciók akkor is várakoznak egy karakter bevitelére, ha a billentyűzetpuffer üres. Az 1-es funkció hívásakor a bevitt karakter megjele-

nik a képernyőn (ill. a standard kiviteli egységen), a 7-es és 8-as funkciók meghívásakor viszont nem (a karakternek ezt a megjelenítését echo-nak nevezik).

A 6-os funkció abban különbözik a többitől, hogy ezzel a karakterek be is olvashatók és ki is írhatók. Azt, hogy egy karaktert be kell olvasnia, úgy közöljük vele, hogy a funkció száma mellett a DL regiszterben 255-ös értéket adunk át. Ez a funkció nem vár karakter bevitelére, hanem a végrehajtása után a vezérlést azonnal visszaadja az őt meghívó programnak. Ha beolvasott egy karaktert (amit egyébként nem ír ki a képernyőre), akkor ezt úgy hozza a meghívó program tudomására, hogy visszatérésekor az állapotjelző regiszterben lévő zéróbitet törli. Ha viszont ennek a bitnek az értéke 1, akkor a funkciónak nem volt mit beolvasnia.

A 11-es funkcióval az állapítható meg, hogy van-e a billentyűzetpufferben – egy vagy több – karakter. A funkció meghívása után az AL regiszter tartalma mutatja meg, hogy vannak-e rendelkezésre álló karakterek a billentyűzetpufferben. Ha a regiszter tartalma 0, akkor nincsenek, ha 255, akkor a pufferben egy vagy több karakter beolvasásra vár.

Előfordulhat, hogy egy program a billentyűzetről egy meghatározott karaktert akar beolvasni, de a billentyűzetpufferben korábbról még maradtak karakterek. Ezért a beolvasó funkció meghívása előtt törölni kell a billentyűzetpuffert. Erre szolgál a 12-es funkció, amely azonban nem csak törli a puffert, hanem egyúttal azt a karakterbeolvasó funkciót is meghívja, amelyiknek a számát ennek a törlő funkciónak a meghívásakor az AL regiszterben átadjuk.

Az eddig bemutatott, csak egyetlen karakter bevitelére szolgáló funkciók mellett a DOS olyan funkciót is rendelkezésünkre bocsát, amellyel egyszerre több karaktert, vagyis egy karakterláncot lehet beolvasni. Ez a 10-es számú funkció. Meghívásához az AH regiszterben a funkció számát, valamint a DS:DX regiszterpárban a karakterláncot tartalmazó puffer kezdőcímét kell átadni. Ebbe a pufferbe nemcsak a beolvasott karakterek kerülnek, hanem a puffer első két bájtján (a puffer kezdetétől számított 0 – 1 ofszetcímen) saját magára vonatkozó információkat is tartalmaz. Az első bájt értékét a funkció meghívása előtt a programozónak, ill. a funkciót meghívó programnak kell megadnia; ez határozza meg a DOS számára, hogy a pufferben a 2-es ofszetcímtől kezdődően – a karakterláncot lezáró kocsi vissza (CR) karaktert is beleértve – hány karaktert helyezhet a pufferbe. A karakterek bevitele során valamennyi szerkesztő billentyű (törlés, Ctrl-C stb.) használható, a funkció a karakterek beolvasását csak az Enter/Return lenyomásakor fejezi be, illetve akkor, ha a puffer megtelt (további karaktert ezután már nem fogad, amit a sípoló hang kiadásával a felhasználó tudomására is hoz. Az Enter billentyű lenyomásakor a DOS a ténylegesen beolvasott karakterek számát (az Enter karaktert nem beleértve) beírja a puffer 1. ofszetcímén lévő bájtba (ez a puffer második bájtja). A puffer előbbiekben ismertetett szerkezetéből következik, hogy a puffer méretét a ténylegesen beviendő karakterek számánál eggyel hosszabbra kell megválasztani. A beviendő karakterek száma a bevitelt lezáró Enter nélkül értendő.

A következő táblázatban összefoglaljuk az eddig bemutatott funkciókat, és azt is feltüntetjük, hogy ezek közül melyek reagálnak a Ctrl-C vagy a Ctrl-Break billentyűk lenyomására, ill. melyek jelenítik meg a beolvasott karakter(eke)t (echo).

Funkció száma	Feladat	Ctrl-C vizsgálata	Echo
01(h)	Karakter bevitele	Igen	Igen
06(h)	Karakter közvetlen be-, ill. kivitele	Nem	Nem
07(h)	Karakter bevitele várakozással	Nem	Nem
08(h)	Karakter bevitele várakozás nélkül	Igen	Nem
0A(h)	Karakterlánc bevitele	Igen	Igen
0B(h)	Beviteli státusz (puffer) vizsgálata	Igen	Nem
0C(h)	Beviteli puffer törlése és beviteli funkció hívása	Változó	

Képernyő

A DOS-ban három funkció támogatja, hogy karaktereket vagy karakterláncokat írjunk a képernyőre (ill. az alapértelmezés szerinti kiviteli egységre). Ebből kettő karaktereket, egy pedig karakterláncot jelenít meg. Ez utóbbi a 9-es számú funkció, amely a karakterlánc kivitelére szolgáló BIOS funkcióktól eltérően lezáró jelként nem a végjelet (a 0 ASCII-kódot), hanem a $ jelet (36-os ASCII-kódot) várja a kiírandó karakterlánc végén. Természetesen az AH regiszterben most is át kell adni a funkció számát. A kiírandó karakterlánc címét a DS:DX regiszterpár tartalmazza: a DS-ben a szegmenscím, a DX-ben pedig az ofszetcím van.

A karakterláncba beírható vezérlőkarakterek ASCII-kódja és a karakterek jelentése:

7 bell, amely egy sípoló hangot szólaltat meg
8 backspace (visszaléptetés), törli a megelőző karaktert, és a kurzort egy hellyel balra mozgatja
10 „Line Feed" (LF, soremelés), a kurzort egy sorral lefelé mozgatja
13 „Carriage Return" (CR, kocsi vissza), a kuzort az aktuális sor elejére állítja

A 9-es funkció figyeli a Ctrl-C, ill. a Ctrl-Break billentyű lenyomását.

A 2-es funkció csak abban tér el a 9-estől, hogy meghívásával mindig csak egyetlen karakter írható a képernyőre. A kiírandó karakter kódját a DL regiszterben kell átadni.

A másik, ugyancsak egyetlen karakter megjelenítését végző funkció a 6-os, amelyet a karakterek bevitelével kapcsolatban már megismertünk. A megjelenítendő karaktert itt is a DL regiszterben kell átadni. Annyi megszorítás van, hogy a 255-ös ASCII-kód nem íratható ki, mert a 6-os funkció a DL regiszterben lévő 255-öt úgy értelmezi, hogy egy karaktert kell beolvasnia. Valamivel gyorsabban dolgozik, mint a társa, a 2-es funkció, mert az egyes karakterek kiírása közben nem ellenőrzi a Ctrl-C, ill. a Ctrl-Break billentyű lenyomását.

Nyomtató

A karakterek kivitelére vonatkozó hagyományos funkciók közül mindössze egy, az 5-ös számú funkció vonatkozik a nyomtatóra, amely a kiírandó szöveget karakterenként nyomtatja ki. Tekintettel arra, hogy ez a funkció is átirányítható, meghívásakor nem lehet pontosan tudni, hogy a kiírandó karakterek melyik nyomtatóra kerülnek. A rendszer ugyanis a karaktereket teljesen általánosan, az alapértelmezés szerinti (standard) nyomtatóra küldi, és ez a géphez csatlakoztatható nyomtatók bármelyike, de akár még a soros interfész is lehet.

Ha a standard nyomtató a funkció meghívásakor foglalt, akkor a funkció a jel kiírásával mindaddig vár, míg az fel nem szabadul, és csak ez után adja vissza a vezérlést a meghívó programnak. E várakozási idő alatt a funkció reagál a Ctrl-C vagy a Ctrl-Break lenyomására.

A funkció meghívásához a funkció számát az AH regiszterben, a kiírandó karakter kódját pedig a DL regiszterben kell átadni. Sajnos a funkció a visszatérésekor nem adja vissza a nyomtató státuszát, úgyhogy ez a DOS-ban nem kérdezhető le. Ezért a karakterek nyomtatóra küldéséhez a DOS funkció helyett célszerűbb egy BIOS funkciót használni. Ennek az az előnye, hogy segítségével nemcsak azt a nyomtatót lehet előre meghatározni, amelyre a karaktereket küldeni akarjuk, hanem ennek révén a nyomtatást követően a nyomtató státusza is lekérdezhető (pl. rendelkezésre áll-e a nyomtató, vagy van-e papír a készülékben). Ezekkel a funkciókkal a II. kötet BIOS-ról szóló, 1. fejezetben foglalkozunk.

Soros interfész

A soros interfésszel való kapcsolattartáshoz a DOS egy-egy író és olvasó funkciót bocsát rendelkezésünkre. Alapértelmezés szerint a továbbítandó adatok a standard soros interfészre (COM1) kerülnek, de mind a kivitel, mind a bevitel átirányítható a COM2-re. Mindkét funkció reagál a Ctrl-C, ill. a Ctrl-Break billentyűkre. Meghívásukat követően azonban ezek sem adják vissza a soros interfész státuszát, úgyhogy ez a DOS-ban nem vizsgálható, és az átvitel közben adódó hibák sem ismerhetők fel.

A soros interfészről egy karakter vételére a 3-as funkciót kell használni, amely a vett karaktert az AL regiszterben szolgáltatja. Sajnos a DOS a soros interfészen érkező adatokat nem puffereli, így előfordulhat, hogy gyorsabban érkeznek az adatok, mint ahogyan a funkció ezeket le tudja kérdezni. Emiatt adatok veszhetnek el.

A soros interfészre a 4-es funkcióval lehet adatokat küldeni. A funkció meghívásakor a kiküldendő karakter ASCII-kódját a DL regiszterben kell átadni. Ha küldéskor az interfész éppen foglalt, akkor a funkció addig várakozik, amíg az fel nem szabadul, és csak ezután küldi ki az adatokat.

E két funkció használata csak akkor célszerű, ha CP/M programokat kell DOS-környezetbe átültetnünk. A DOS alatt újonnan kifejlesztendő programoknál ebben az esetben is a handle-funkciókat kell előnyben részesíteni, már csak azért is, mert a DOS saját magán belül a hagyományos funkciókat amúgy is handle-funkciókra fordítja le.

A nyomtatófunkciókhoz hasonlóan itt is megjegyezzük, hogy a soros interfésszel kapcsolatos adatforgalomban a BIOS funkciók nagyobb kényelmet és rugalmasságot jelentenek.

6.5.3. Programpéldák

A billentyűzet olvasását és a képernyőre írást végző funkciók ismertetésekor már beszéltünk arról, hogy ezek az eszközök átkapcsolhatók bináris (raw) vagy ASCII (cooked) üzemmódba. Ezt az átkapcsolást végzik az itt következő programok, amelyek most is BASIC, PASCAL és C nyelven készültek. A programok az ún. IOCTL (I/O Control, azaz bemenet/kimenet vezérlő) funkciót használják, amelyen keresztül a DOS eszközmeghajtóihoz lehet hozzáférni. (Ezzel a funkcióval, ill. az alfunkcióival a későbbiekben még részletesen foglalkozunk.) Ezek azok a rutinok, amelyek a be- és kivitelt végző DOS funkciókat és a hardvert egymáshoz illesztik. A gépen futó programok ezen az IOCTL funkción keresztül közlik a billentyűzetet és a képernyőt, mint alapértelmezés szerinti beviteli és kiviteli eszközt vezérlő programmal, hogy bináris vagy ASCII üzemmódban kell-e működnie.

Azért, hogy az egyes vezérlőkarakterek különböző hatását a két üzemmódban szemléltessük, a példaprogramokban a CON eszközmeghajtót először bináris üzemmódba kapcsoljuk. Ezután erre a meghajtóra néhányszor kiküldünk egy vizsgáló szöveget. Az alapértelmezés szerinti ASCII üzemmódtól eltérően most a képernyőre íráskor a Ctrl-S billentyű lenyomása hatástalan, azaz a képernyő görgetése nem függeszthető fel. A következő lépésben a program átkapcsol ASCII üzemmódba. Az ekkor megjelenő szöveg felfelé gördülése a Ctrl-S billentyűvel már megállítható.

A két üzemmód között lényegében az eszközmeghajtó ún. eszközjellemző szavában lévő egyetlen bit értékének megváltoztatásával lehet átkapcsolni. (Az angol nyelvű DOS irodalomban ennek a szónak device information word a neve, és nem tévesztendő össze az eszköz attribútum szóval!)

A programunkban meghívott IOCTL funkció ezt az eszközjellemző szót olvassa be, majd a kívánt üzemmódnak megfelelően bebillenti, majd törli a megfelelő bitet.

```
1000 '
1005 '
1010 '                    B I N A S C I I . B A S
1020 '
1030 '  Funkciója : a BINARIS és ASCII üzemmód közötti eltérés
1040 '              szemléltetése.
1045 '
1050 '
1060 '
1070 DEFINT Q                                   'Q "dummy" egész érték.
1080 KEY OFF : CLS
1090 PRINT : PRINT
1100 PRINT "FIGYELMEZTETÉS :" : PRINT
1110 PRINT "A 60000-es címtől kezdődően betöltött megszakításhívó ru-"
1120 PRINT "tin felulírását elkerülendő a GWBASIC-et GWBASIC /M:60000"
1130 PRINT "paranccsal kell indítania. Ha nem így hívta a GWBASIC-et,"
1140 PRINT "akkor üsse le az <s> billentyűt, egyébként egy tetszõle-"
1150 PRINT "ges másik billentyűt !"
1160 '
1170 Z$ = ""
1180 WHILE Z$ = ""                              'Helytelen indítás
1190    Z$ = INKEY$                             'esetén a program
1200 WEND                                       'leállítása.
1210 IF Z$ = "s" OR Z$ = "S" THEN CLS:END
1220 '
1230 GOSUB 9000                                 'Megszakításhívó rutin
1240                                            'betöltése.
1250 CLS
1260 '
2000 '
2010 '          P R O G R A M T ö R Z S
2020 '
2030 '
2040 HNDX = 0                                   'A handle a konzolmeg-
2050 '                                          'hajtóra állítva.
2060 PRINT : PRINT
2070 PRINT "A Konzolmeghajtó a teszt első lépésében BINARIS üzem-"
2080 PRINT "módban dolgozik. Ekkor a vezérlõkarakterek (példánkban"
2090 PRINT "a képernyõ görgetését felfüggesztõ <CTRL-S> billentyû)"
2100 PRINT "hatástalanok."
2110 PRINT
2120 PRINT "Indítsa el a tesztet egy billentyû megnyomásával .."
2130 '
2140 Z$ = ""                                    'Várakozás egy billentyû
2150 WHILE Z$ = ""                              'leütésére.
2160    Z$ = INKEY$
2170 WEND
2180 '
2190 GOSUB 5000                                 'Konzolmeghajtó bináris
2200 MODUS$ = "b"                               'üzemmódba vezérlése.
2210 GOSUB 3000                                 'Teszt megjelenítése.
2220 '
2230 CLS
2240 PRINT : PRINT
2250 PRINT "A Konzolmeghajtó a teszt második lépésében ASCII üzem-"
2260 PRINT "módban dolgozik. Ekkor a vezérlõkarakterek (példánkban"
2270 PRINT "a képernyõ görgetését felfüggesztõ <CTRL-S> billentyû)"
2280 PRINT "mind a bemeneten (billentyûzet), mind a kimeneten (kép-"
2290 PRINT "ernyõ) megjelennek, vagyis hatásosak."
2300 PRINT
2310 PRINT "Indítsa el a tesztet egy billentyû megnyomásával .."
2320 '
2330 Z$ = ""                                    'Várakozás egy billentyû
2340 WHILE Z$ = ""                              'leütésére.
```

```
2350     Z$ = INKEY$
2360 WEND
2370 '
2380 GOSUB 4000                              'Konzolmeghajtó ASCII
2390 MODUS$ = "a"                            'üzemmódba vezérlése.
2400 GOSUB 3000                              'Teszt megjelenítése.
2410 '
2420 Z$ = ""
2430 WHILE Z$ = ""                           'Kiszállás előtt egy
2440     Z$ = INKEY$                         'billentyű megnyomá-
2450 WEND                                    'sára várakozás.
2460 CLS
2470 END
2480 '
3000 '
3010 '  TESZT KAR.LANC MEGJELENíTÉSE A STANDARD
3020 '  KIMENETI ESZKoZoN
3030 '
3040 '
3050 XXX$ = "●.●.●.●.●.●.●.●.●.●.●.●.●.●.●.●.●.●.●.●.●.●.●."
3060 IF MODUS$ <> "b" THEN GOTO 3080
3070 TEXT$ = "BINARIS üzemmódban a <Ctrl-S> leütésének nincs hatása!"
3080 IF MODUS$ <> "a" THEN GOTO 3100
3090 TEXT$ = "ASCII üzemmódban  a <Ctrl-S> szünetelteti a görgetést!"
3100 FOR I% = 0 TO 111
3110     J%      = I% MOD 28
3120     TXT$    = LEFT$(XXX$,40-J%)+LEFT$(TEXT$,J%*2)+LEFT$(XXX$,40-J%)
3130 '
3140     MSZ%    = &H21                      '21(h) megszakítás.
3150     FUN%    = &H40                      'Handle írás funkció.
3160     FUN1%   = &H0
3170     NL%     = 80                        'Megjelenítendő karakte-
3180     NH%     = 0                         'rek számának beállítása.
3190     OFFL%   = PEEK(VARPTR(TXT$)+1)      'Ofszetcím LOW  bájt.
3200     OFFH%   = PEEK(VARPTR(TXT$)+2)      'Ofszetcím HIGH bájt.
3210     HL%     = 1                         'Standard eszköz ki-
3220     HH%     = 0                         'jelölése.
3230     '
3240     CALL RCIM(MSZ%,FUN%,FUN1%,HH%,HL%,NH%,NL%,OFFH%,OFFL%,Q,Q,Q,Q)
3250 NEXT
3260 PRINT
3270 RETURN
3280 '
4000 '
4010 '  ASCII üZEMMÓD BIT BEBILLENTÉSE
4020 '
4030 '
4040 GOSUB 6000                              'Ha az eszközjellemző bájt
4050 ATTR% = ATTR% AND 223                   'hatodik pozícióján levő
4060 GOSUB 7000                              'bitje 0, az eszköz ASCII
4070 RETURN                                  'üzemmódban dolgozik.
4080 '
5000 '
5010 '  BINARIS üZEMMÓD BIT BEBILLENTÉSE
5020 '
5030 '
5040 GOSUB 6000                              'Ha az eszközjellemző bájt
5050 ATTR% = ATTR% OR 32                     'hatodik pozícióján levő
5060 GOSUB 7000                              'bitje 1, az eszköz BINA-
5070 RETURN                                  'RIS üzemmódban dolgozik.
5080 '
```

```
6000 '
6010 '  MEGHAJTó ESZKöZJELLEMZŐ BÁJT BEOLVASÁSA
6020 '     BE : HND% a kiválasztott handle
6030 '     KI : ATTR% az eszközjellemző bájt
6040 '  Csak a 0 - 7 bitek kerülnek beolvasásra
6050 '
6060 '
6070 MSZ%   = &H21                              '21(h) megszakítás.
6080 FUN%   = &H44                              'IOCTL funkció.
6090 FUN1%  = &H0                               'IOCTL attribútum be-
6100 '                                          'állítás alfunkció.
6110 HH%    = INT(HND%/256)                     'Handle HIGH bájt.
6120 HL%    = HND% AND 255                      'Handle LOW  bájt.
6130 '
6140 CALL RCIM(MSZ%, FUN%, FUN1%, HH%, HL%, Q, Q, Q, ATTR%, Q, Q, Q, Q)
6150 RETURN
6160 '
7000 '
7010 '  MEGHAJTó ESZKöZJELLEMZŐ BÁJT BEÁLLÍTÁSA
7020 '     BE : ATTR% a beállítani kívánt bájt
7030 '     KI : -
7040 '
7050 '
7060 MSZ%   = &H21                              '21(h) megszakítás.
7070 FUN%   = &H44                              'IOCTL funkció.
7080 FUN1%  = &H1                               'IOCTL jellemző be-
7090                                            'állítás alfunkció.
7100 HH%    = INT(HND%/256)                     'Handle HIGH bájt.
7110 HL%    = HND% AND 255                      'Handle LOW  bájt.
7120 ATHI%  = INT(ATTR%/256)                    'Attribútum HIGH bájt.
7130 ATLO%  = ATTR% AND 255                     'Attribútum LOW  bájt.
7140 '
7150 CALL RCIM(MSZ%, FUN%, FUN1%, HH%, HL%, Q, Q, ATHI%, ATLO%, Q, Q, Q, Q)
7160 RETURN
7170 '
9000 '
9010 '  A MEGSZAKÍTÁSHÍVÓ RUTIN INICIALIZÁLÁSA
9020 '     BE: -
9030 '     KI: RCIM a Rutin CÍMe
9040 '
9050 '
9060 RCIM = 60000!                  'A rutin kezdőcíme a BASIC szegmensben.
9070 DEF SEG                        'A BASIC szegmens beállítása.
9080 RESTORE 9140
9090 FOR I% = 0 TO 160                              'A rutin betöltése a
9100     READ X% : POKE RCIM + I%, X%               'tárba.
9110 NEXT
9120 RETURN
9130 '
9140 DATA    85, 139, 236,  30,   6, 139, 118,  30, 139,   4, 232, 140
9150 DATA     0, 139, 118,  12, 139,  60, 139, 118,   8, 139,   4,  61
9160 DATA   255, 255, 117,   2, 140, 216, 142, 192, 139, 118,  28, 138
9170 DATA    36, 139, 118,  26, 138,   4, 139, 118,  24, 138,  60, 139
9180 DATA   118,  22, 138,  28, 139, 118,  20, 138,  44, 139, 118,  18
9190 DATA   138,  12, 139, 118,  16, 138,  52, 139, 118,  14, 138,  20
9200 DATA   139, 118,  10, 139,  52,  85, 205,   0,  93,  86, 156, 139
9210 DATA   118,  12, 137,  60, 139, 118,  28, 136,  36, 139, 118,  26
9220 DATA   136,   4, 139, 118,  24, 136,  60, 139, 118,  22, 136,  28
9230 DATA   139, 118,  20, 136,  44, 139, 118,  18, 136,  12, 139, 118
9240 DATA    16, 136,  52, 139, 118,  14, 136,  20, 139, 118,   8, 140
9250 DATA   192, 137,   4,  88, 139, 118,   6, 137,   4,  88, 139, 118
9260 DATA    10, 137,   4,   7,  31,  93, 202,  26,   0,  91,  46, 136
9270 DATA    71,  66, 233, 108, 255
```

```
/*
/*                                                                    */
/*                          B I N A S C I I . C                       */
/*                                                                    */
/*   Funkciója : a BINÁRIS és az ASCII üzemmód közötti különbség      */
/*               szemléltetése.                                       */
/*                                                                    */
/*                                                                    */

#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>

#define STDBE 0  /* Standard bemenethez a 0. handle hozzárendelése.  */
#define STDKI 1  /* Standard kimenethez a 1. handle hozzárendelése.  */

/*                                                                    */
/*   UZEMMOD : MEGHAJTÓJELLEMZŐ BEOLVASÁSA                           */
/*      KI : A meghajtó jellemző bájt                                 */
/*                                                                    */

int UZEMMOD(handle)
int handle;

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

regiszter.x.ax = 0x4400;                 /* IOCTL mód olvasási alfunkció, */
regiszter.x.bx = handle;                 /* egylépésben kerül AX-be.      */
intdos(&regiszter, &regiszter);          /* 21(h) megszakítás.            */
return(regiszter.x.dx);                  /* Visszatérési érték az ü.mód.  */
}

/*                                                                    */
/*   BIN : KARAKTERKEZELŐ MEGHAJTÓ BINÁRIS                            */
/*         ÜZEMMÓDBA VEZÉRLÉSE                                        */
/*                                                                    */

int BIN(handle)
int handle;

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

regiszter.x.ax = 0x4401;             /* IOCTL üzemmód beállítás alfunkció. */
regiszter.x.bx = handle;
regiszter.x.dx = UZEMMOD(handle) & 255 | 32;
intdos(&regiszter, &regiszter);                  /* 21(h) megszakítás, a  */
                                                 /* BIN bit bebillentése. */
}

/*                                                                    */
/*   ASCII : KARAKTERKEZELŐ MEGHAJTÓ ASCII                            */
/*           ÜZEMMÓDBA VEZÉRLÉSE                                      */
/*                                                                    */

int ASCII(handle)
int handle;

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

regiszter.x.ax = 0x4401;             /* IOCTL üzemmód beállítás alfunkció. */
regiszter.x.bx = handle;
```

```
regiszter.x.dx = UZEMMOD(handle) & 223;               /* BIN bit nullázása. */
intdos(&regiszter, &regiszter);                        /* 21(h) megszakítás. */
}

/*┌──────────────────────────────────────────────────┐*/
/*│ TESZT : TESZT KAR.LANC CIKLIKUS KIÍRASA          │*/
/*│     BE : "umod" az aktuális üzemmód.             │*/
/*└──────────────────────────────────────────────────┘*/

void TESZT(umod)
char umod;
{
int i;
int j;
int k;

char *dumbegin;
char *txtbegin;

static char *dum = "*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.";
static char *txt =
  "                                                          ";

if (umod == 'b')
   strcpy(txt,
        "BINARIS üzemmódban a <CTRL-S> leütésének nincs hatása!");
if (umod == 'a')
   strcpy(txt,
        "ASCII üzemmódban a <CTRL-S>  szünetelteti a görgetést!");

dumbegin = dum;
txtbegin = txt;

for (i = 0; i < 4; i++)
 {
 for (j = 0; j < 28; j++)
  {
  for (k = 0; k < 40 - j ; k++, dum++)
   fputc(*dum, stdout);
  dum = dumbegin;
  for (k = 0; k < 2 * j  ; k++, txt++)
   fputc(*txt, stdout);
  txt = txtbegin;
  for (k = 0; k < 40 - j ; k++, dum++)
   fputc(*dum, stdout);
  dum = dumbegin;
  }
 }
printf("");
}

/*┌──────────────────────────────────────────────────┐*/
/*│ F Ö P R O G R A M : A TESZT FUTTATASA            │*/
/*└──────────────────────────────────────────────────┘*/

void main()

{
printf("");
printf("A konzolmeghajtó a teszt első lépésében BINARIS üzemmód-");
printf("ban dolgozik.Ekkor a vezérlőkarakterek (a példában a kép-");
printf("ernyő görgetését felfüggesztő <CTRL-S> billentyű) hatás-");
printf("talanok.");
```

```
printf("Indítsa el a tesztet egy billentyű megnyomásával ..");

getch();                              /* Várakozás egy billentyűre.      */
BIN(STDBE);                           /* Bináris üzemmódba váltás.       */
TESZT('b');                           /* Teszt kar. lánc megjelenítése.  */
while (kbhit())                       /* Billentyűzetpuffer kiürítése.   */
   getch();

printf("");
printf("A konzolmeghajtó a teszt második lépésében ASCII üzem-");
printf("módban dolgozik. Ekkor a vezérlőkarakterek (példánkban a");
printf("képernyő görgetését felfüggesztő <CTRL-S> billentyű) a");
printf("bemeneten (billentyűzet), és a kimeneten (képernyő) egy-");
printf("aránt megjelennek, vagyis hatásosak.");
printf("Indítsa el a tesztet egy billentyű megnyomásával ..");

getch();                              /* Várakozás egy billentyűre.      */
ASCII(STDBE);                         /* ASCII üzemmódba váltás.         */
TESZT('a');                           /* Teszt kar. lánc megjelenítése.  */

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

```
{ ┌──────────────────────────────────────────────────────┐ }
{ │                                                      │ }
{ │                  B I N A S C I I . P A S             │ }
{ │                                                      │ }
{ │  Funkciója : a BINARIS és az ASCII üzemmód közötti   │ }
{ │              szemléltetése.                          │ }
{ │                                                      │ }
{ └──────────────────────────────────────────────────────┘ }

program BINASCII;

const STDBE = 0;    { Standard bemenethez a 0. handle hozzárendelése. }
      STDKI = 1;    { Standard kimenethez a 1. handle hozzárendelése. }

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

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

{ ┌──────────────────────────────────────────────────┐ }
{ │ UZEMMOD : MEGHAJTÓ ESZKÖZJELLEMZŐ BEOL-         │ }
{ │ VASASA                                           │ }
{ │    KI : A meghajtó attribútuma.                  │ }
{ └──────────────────────────────────────────────────┘ }

function UZEMMOD(handle : integer) : integer;

var regiszter : dosreg;                  { Regiszter rekord deklaráció. }

begin

   regiszter.ax := $4400;           { IOCTL üzemmód beolvasási alfunk- }
   regiszter.bx := handle;          { ció, egy lépésben kerül AX-be.   }
   msdos(regiszter);                { 21(h) megszakítás.               }
   UZEMMOD := regiszter.dx          { A visszatérési érték beállítása. }

end;

{ ┌──────────────────────────────────────────────────┐ }
{ │ BIN : KARAKTERKEZELŐ MEGHAJTÓ BINARIS            │ }
{ │       ÜZEMMÓDBA VEZÉRLÉSE                        │ }
{ └──────────────────────────────────────────────────┘ }

procedure BIN(handle : integer);

var regiszter : dosreg;                  { Regiszter rekord deklaráció. }

begin

   regiszter.ax := $4401;          { IOCTL üzemmód beállítás alfunkció. }
   regiszter.bx := handle;
   regiszter.dx := UZEMMOD(handle) and 255 or 32;
   msdos(regiszter);                                { 21(h) megszakítás, a }
                                                    { BIN bit bebillentése. }

end;
```

```
{                                                        }
{  ASCII : KARAKTERKEZELŐ MEGHAJTÓ ASCII                 }
{          ÜZEMMÓDBA VEZÉRLÉSE                           }
{                                                        }

procedure ASCII(handle : integer);

var regiszter : dosreg;                    { Regiszter rekord deklaráció. }
begin
   regiszter.ax := $4401;                  { IOCTL üzemmód beállítás alfunkció. }
   regiszter.bx := handle;
   regiszter.dx := UZEMMOD(handle) and 223;  { BIN bit kimaszkolása. }
   msdos(regiszter);                         { 21(h) megszakítás.    }
end;

{                                                        }
{  TESZT : TESZT KAR.LANC CIKLIKUS KIÍRASA               }
{     BE : "umod" az aktuális üzemmód.                   }
{                                                        }

procedure TESZT(umod : char);

var regiszter : dosreg;                   { Regiszter rekord deklaráció. }
    i         : integer;
    j         : integer;
    xxx       : string[40];               { "Dummy" kitöltő karakterek. }
    yyy       : string[40];               { "Dummy" kitöltő karakterek. }
    text      : string[54];
    textresz  : string[54];
    txt       : string[80];               { A képernyőn megjelenő kar. lánc. }

begin

   if umod = 'b' then
      text :='BINARIS üzemmódban a <CTRL-S> leütésének nincs hatása!';
   if umod = 'a' then
      text :='ASCII üzemmódban a <CTRL-S> szünetelteti a görgetést!';
   yyy   := '•.•.•.•.•.•.•.•.•.•.•.•.•.•.•.•.•.•.•.•.';
   for i := 0 to 111 do
      begin
         j               := (i mod 28);
         xxx             := copy(yyy,1,40-j);
         textresz        := copy(text,1,j*2);
         txt             := concat(xxx,textresz,xxx);
         regiszter.bx := STDKI;
         regiszter.cx := 80;                { Megjelenítendő kar. száma. }
         regiszter.ds := seg(txt);          { Kar. lánc szegmenscíme.    }
         regiszter.dx := ofs(txt)+1;        { Kar. lánc ofszetcíme.      }
         regiszter.ax := $4000;             { Handle írás funkció.       }
         msdos(regiszter);                  { 21(h) megszakítás.         }
      end;
  writeln;

end;
```

```
{
{   FŐPROGRAM: A TESZT FUTTATASA                    }
{                                                   }

begin

   clrscr;

   writeln('A  konzolmeghajtó a  teszt első lépésében BINÁRIS üzem-');
   writeln('módban dolgozik. Ekkor a vezérlőkarakterek  (példánkban');
   writeln('a képernyő  görgetését felfüggesztő <CTRL-S> billentyű)');
   writeln('hatástalanok.');
   writeln;
   writeln('Indítsa el a tesztet egy billentyű  megnyomásával ..');

   read(kbd, z);                        { Várakozás egy billentyűre.    }
   clrscr;
   BIN(STDBE);                          { Bináris üzemmódba váltás.     }
   TESZT('b');                          { Teszt kar. lánc megjelenítése. }
   clrscr;

   writeln('A  konzolmeghajtó a teszt második lépésében ASCII üzem-');
   writeln('módban dolgozik. Ekkor a  vezérlőkarakterek (példánkban');
   writeln('a képernyő  görgetését felfüggesztő <CTRL-S> billentyű)');
   writeln('mind a bemeneten (billentyűzet), mind a kimeneten (kép-');
   writeln('ernyő) megjelennek, vagyis hatásosak.');
   writeln;
   writeln('Indítsa el a tesztet egy billentyű  megnyomásával ..');

   read(kbd, z);                        { Várakozás egy billentyűre.    }
   clrscr;
   ASCII(STDBE);                        { ASCII üzemmódba váltás.       }
   TESZT('a');                          { Teszt kar. lánc megjelenítése. }
   clrscr;

end.
```

6.6. Állománykezelés a DOS-ban

A karakterek be- és kivitelére szolgáló funkciók mellett az állománykezelést végző funkciók azok a legalapvetőbb funkciók, amelyeket egy operációs rendszernek a felhasználó rendelkezésére kell bocsátania. Ezek közül a legfontosabbak:

- állományok létrehozása és törlése;
- állományok megnyitása és lezárása;
- írás egy állományba, ill. olvasás egy állományból.

A DOS-hoz hasonló, fejlett operációs rendszerek a fentieken kívül még egy sor további állománykezelő funkciót bocsátanak a felhasználó rendelkezésére. A DOS-ban az olyan funkciók mellett, amelyek valamilyen speciális információt szolgáltatnak egy állományról, például olyan is van, amellyel egy állomány nevét meg lehet változtatni. A DOS specialitása, hogy minden ilyen funkciót kétszer tartalmaz. Ennek a meglehetősen furcsa dolognak az áll a hátterében, hogy – mint már említettük – a DOS törekedett arra, hogy mind a CP/M, mind a UNIX operációs rendszerrel kompatibilis legyen. Ezért aztán

a DOS az állománykezelő funkcióknak mind a CP/M-mel, mind a UNIX-szal kompatibilis változatát tartalmazza.

A CP/M-mel kompatibilis funkciókat FCB-funkcióknak nevezzük, mert az alapjukat egy FCB-nek nevezett adatstruktúra képezi. Maga az FCB név az angol „file control block" (szó szerinti magyar fordításban: állományvezérlő blokk) kezdőbetűiből áll össze, és arra utal, hogy a DOS ezt az adatszerkezetet használja arra, hogy az állományon végzett műveletek során az állományra vonatkozó fontos információkat tárolja.

Jóllehet ennek az adatszerkezetnek a DOS a tulajdonképpeni „haszonélvezője", mégis a programozó dolga, hogy a programján belül erre a célra helyet biztosítson. Ennek fejében viszont lehetősége van arra, hogy ezen az FCB-n keresztül olyan funkciókat használhasson, mint a nyitás, zárás, írás, olvasás stb. Mivel a rendszer fejlesztői az FCB-funkciókat a CP/M-mel való kompatibilitás megtartása céljából vették be a rendszerbe, de a CP/M-ben az állományok nem hierarchikus rendszerbe szervezettek, a DOS 2.0 verziójában bevezetett új állományszervezési rendszert az FCB-funkciók nem támogatják. Ebből következik, hogy az FCB-funkciókon keresztül mindig csak az aktuális katalógusban lévő állományokhoz lehet hozzáférni.

A UNIX-kompatibilis handle-funkcióknál ilyen hátrány nincs. A handle – mint korábban már volt szó róla – egy numerikus értéket jelent, amelyen keresztül az illető állományhoz hozzá lehet férni. A handle-t a meghívó program az állomány megnyitásakor vagy létrehozásakor kapja meg. A DOS-nak természetesen a handle-lel támogatott állománykezelésnél is tárolnia kell bizonyos, az állományra vonatkozó információkat. Ezeket viszont az FCB-vel támogatott hozzáférésektől eltérően egy általa lefoglalt tárterületen lévő adatszerkezetben tárolja, és nem olyan területen, amelyet a meghívó programnak kell biztosítania.

E funkciók mindkét csoportját külön-külön megvizsgáljuk. Lássuk először ismét a handle-funkciókat.

6.6.1. Handle-funkciók

A programozó az FCB-funkciókhoz képest a handle-funkciókon keresztül sokkal kényelmesebben férhet hozzá az állományokhoz. Utóbbiak esetén ugyanis nem kell egy FCB-szerű adatszerkezetet a DOS rendelkezésére bocsátani. Az állományhoz való hozzáférés a UNIX operációs rendszer funkcióhoz hasonlóan először az állomány nevén keresztül történik. A rendszer az állomány nevét ASCII-karakterlánc formájában átadja az állományt megnyitó vagy létrehozó funkciónak, amelyet az állományhoz való első hozzáférés előtt mindig meg kell hívni. A név az állomány nevén kívül tartalmazhat még egységazonosítót, a katalóguson belüli teljes útvonalat (angol neve path), és az állománynév kiterjesztését. A karakterláncot a végjel (0 ASCII-kód) zárja le. Ha az állományt sikerült megnyitni vagy létrehozni, akkor a meghívott funkció egy handle-nek nevezett, 16 bites numerikus értékkel tér vissza, amelyet a meghívó programnak meg kell jegyeznie. Ettől kezdve az ehhez az állományhoz való összes további hozzáférés ezen a handle-n keresztül történik. Így például az olvasó funkció meghívásakor nem az állomány nevét, hanem a hozzá rendelt handle-t kell átadni. A handle átadása általában a

BX regiszterben történik, a funkció pedig minden esetben az átvitelbit értékével jelzi, hogy a művelet sikeresen végrehajtódott-e. Az átvitelbit a 0 értékkel a sikeres, az 1 értékkel a sikertelen végrehajtást jelzi; utóbbi esetben az AX regiszter a hiba jellegére utaló hibakódot tartalmazza.

Természetesen a DOS-nak az állományon végzett művelet során bizonyos, az állományra vonatkozó adatokat saját magán belül tárolnia kell. Az FCB-funkcióktól eltérően azonban ezeket az adatokat nem a mindenkori program által rendelkezésére bocsátott tárterületre, hanem a saját, belső tábláiba jegyzi be. Mivel azonban a rendelkezésre álló tárhely korlátozott, az egyidejűleg nyitva lévő állományok, és így az egyidejűleg használható handle-k száma is behatárolt. Az egyidejűleg megnyitható handle-k számát egy, a rendszert konfiguráló CONFIG.SYS állományba írt adat határozza meg. Ezt az adatot így lehet az állományba bejegyezni:

FILES = nn

ahol nn a rendelkezésre álló handle-k számát adja meg. A DOS 3.0 verziójánál ez az érték alapértelmezés szerint 8, maximális értéke pedig 255 lehet. A programozó ezt a konfiguráló állományban lévő mindenkori értéket egy szövegszerkesztő segítségével megváltoztathatja, és a saját igényeihez illesztheti. A változtatás hatása azonban csak a rendszer ismételt indítását követően jut érvényre, mert a DOS a CONFIG.SYS állományt csak az indításakor vizsgálja, és az ekkor talált adatoknak megfelelően hozza létre az illető táblákat, és konfigurálja a rendszert.

A DOS egy program számára egyidejűleg 20 handle megnyitását teszi lehetővé (feltételezve, hogy a rendszer belső tábláiban még elegendő hely van). Erre a korlátozásra azért van szükség, hogy egy program további programokat is meghívhasson, amelyeknek esetleg további handle-ket kell megnyitniuk. Ha ez a korlátozás nem lenne, akkor előfordulhat, hogy egy – többszörös áttételen keresztül meghívott – program számára már nem marad szabad handle. Azt sem szabad elfelejteni, hogy egy program meghívásakor a DOS a rendelkezésre álló 20 handle-ből 5-öt már előre, „önkényesen" hozzárendel az öt standard be-, ill. kiviteli egységhez, így a program lényegében csak 15-tel gazdálkodhat. (A DOS 3.3 verziójától felfelé már van mód az egy program rendelkezésére álló handle-k számának növelésére.)

A handle-funkciók az írás és olvasás elvégzésének módjában is különböznek az FCB-funkcióktól. Míg az FCB-funkciók az FCB-ben megadott rekordhosszal dolgoznak, a handle-funkciókkal minden esetben explicit módon közölni kell, hogy (egy vagy több rekordként) hány bájtot kell olvasniuk vagy írniuk. Ez lehetővé teszi, hogy változó adatszerkezetekkel dolgozzunk, és jelentősen leegyszerűsíti több, egymást követő rekord egyidejű elérését. A DOS az író és olvasó funkció mellett még egy funkciót bocsát rendelkezésünkre, amellyel az állományon belüli rekordmutató pozícionálható. Itt sem a rekordszámot, hanem az állomány valamely meghatározott helye (eleje, vége) és az elérni kívánt rekord közötti, bájtokban számított távolságot kell megadni. Ha ismerjük a rekord számát és az egy rekordban lévő bájtok számát, akkor ezt a távolságot igen egyszerűen ki tudjuk számítani:

távolság (bájtokban) = rekord száma * bájt/rekord

Ha egy állományt sorosan (szekvenciálisan) akarunk feldolgozni, akkor erre a funkcióra nincs szükség, mert a DOS egy állomány megnyitásakor vagy létrehozásakor a rekordmutatót az állomány első bájtjára állítja. Ezután minden egyes olvasási vagy írási műveletet követően a rekordmutatót a beolvasott vagy kiírt bájtok számával az állomány vége irányában megnöveli úgy, hogy a következő hozzáférés mindig az azt követő helyen kezdődik, ahol az előző befejeződött.

A következőkben röviden felsoroljuk az állományok kezelését végző legfontosabb handle-funkciókat:

Funkció száma	Feladat
3C(h)	Állomány létrehozása
3D(h)	Állomány megnyitása
3E(h)	Állomány lezárása
3F(h)	Írás állományba
40(h)	Olvasás állományból
42(h)	Rekordmutató mozgatása/állomány méretének meghatározása
43(h)	Állomány attribútumának írása/olvasása
56(h)	Állomány átnevezése
57(h)	Állomány módosítás dátumának/idejének írása/olvasása

E funkciók meghívásával kapcsolatos részletes tudnivalókat a III. kötet DOS megszakítások fejezete tartalmazza, ezért itt csak néhány, általános szabályt ismertetünk.

Azok a funkciók, amelyek argumentumként egy állomány nevét, ill. a név címét várják (pl. egy állományt létrehozó vagy megnyitó funkció), a név szegmenscímét a DS, ofszetcímét pedig a DX regiszterben várják. Ha ezek a funkciók a meghívásuk után egy handle-lel térnek vissza, akkor a handle-t az AX regiszter tartalmazza.

Azok a funkciók, amelyek argumentumként handle-t várnak, ezt a BX regiszterben kapják meg. Meghívásuk után az átvitelbittel jelzik, hogy a végrehajtás során volt-e hiba. Ha volt, akkor az átvitelbit értéke 1, és az AX regiszterben egy hibakód van, amely a hiba okáról tájékoztat.

A lemezműveletekkel kapcsolatos hibákról igen részletes információkat szolgáltat a DOS 21(h) megszakítás 59(h) funkciója, amit a 3.0 verziótól felfelé lehet igénybe venni.

6.6.2. FCB-funkciók

Az előző fejezetben már beszéltünk arról, hogy a DOS egy FCB-nek nevezett adatstruktúrához fordul, amikor egy állományon valamilyen műveletet kell elvégeznie. Ennek az adatstruktúrának azonban nem csak a DOS, hanem a felhasználó is használ veheti, mivel az ebben lévő, az állományra vonatkozó adatokat elolvashatja, és meg is változtathatja. Ezért mielőtt rátérnénk a funkciók ismertetésére, vizsgáljuk meg az FCB szerkezetét.

Az FCB (file control block) 37 bájtból álló adatszerkezet, amely különböző hosszúságú, önálló adatmezőkre oszlik, és a felhasználói program által használt tárterületen helyezkedik el. Az FCB szerkezetét a 20. ábra szemlélteti.

00(h)	meghajtóegység azonosítója	1 bájt
01(h)	állomány neve (8 karakter)	8 bájt
09(h)	állománynév kiterjesztése (3 kar.)	3 bájt
0C(h)	aktuális blokk száma	1 szó
0E(h)	rekord mérete	1 szó
10(h)	állomány mérete	2 szó
14(h)	módosítás dátuma	1 szó
16(h)	módosítás időpontja	1 szó
18(h)	fenntartott	8 bájt
20(h)	aktuális rekord száma	1 bájt
21(h)	relatív rekord száma	4 bájt

20. ábra: Az FCB szerkezete

Az FCB a 0. ofszetcímén annak a meghajtóegységnek az azonosítóját tartalmazza, amelyben az állományt létre kell hozni vagy meg kell nyitni. Ez az azonosító numerikus érték: a 0 az alapértelmezés szerinti meghajtót, az 1 az A, a 2 a B stb. jelű meghajtót jelenti.

Az állomány nevét ASCII-karakterláncként az 1. címen kezdődő mező tárolja. Mivel ez a mező 8 bájt hosszú, csak az állomány nevét tárolhatja, a katalóguson belüli útvonalkijelölést azonban nem. Ez az alapvető oka annak, hogy az FCB-funkciókon keresztül miért csak az aktuális katalógusban lévő állományokhoz lehet fordulni. Ha az állomány neve 8 bájtnál rövidebb, akkor az üresen maradó helyeket a DOS üres karakterekkel (space, 32-es ASCII-kód) tölti fel. Ugyanez vonatkozik az állománynevet követő mezőre. Ez az állománynév maximum 3 bájtos kiterjesztését tartalmazza.

A következő, 0C(h) ofszetcímen álló, szó hosszúságú adat a szekvenciális feldolgozás során az éppen feldolgozott blokk számát adja meg. Ez a szám az FCB 20(h) ofszetcímén álló aktuális rekordszámmal együtt az állomány szekvenciális írásánál/olvasásánál jut szerephez.

A 0E(h) címen lévő, szó hosszúságú adat az állomány rekordjának hosszát adja meg. A rekord hossza 1 és 65535 között lehet, és azoknak a karaktereknek a számát adja meg, amelyeket a DOS egyetlen, összefüggő adatcsoportként kezel, vagyis egyetlen művelettel ír ki vagy olvas be. A DOS a rekord méretét alapértelmezés szerint 128 bájtra állítja be. Amennyiben az alkalmazói program ettől eltérő rekordmérettel kíván dolgozni, akkor a méretet az állomány megnyitását vagy létrehozását követően kell a programnak az FCB-be beírnia.

A 10(h) címtől kezdődő 4 bájt az állomány hosszát tartalmazza olyan formában, mint a lemezkatalógusban: a legalsó címen a legkisebb helyértékű bájt van. A 4 bájtból a következő képlet segítségével számítható ki az állomány hossza:

10(h)_cím + 11(h)_cím * 256 + (12(h)_cím + 13(h)_cím * 256) * 65536

A 14(h) – 15(h) címeken az állomány létrehozásának, ill. utolsó módosításának dátumára valamint idejére vonatkozó információk találhatók. Az FCB ezeket az információkat 16 – 16 biten, kódolt formában tárolja (21. ábra).

Dátum:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	bit
év							hónap				nap					

bit	jelentése
0 – 4.	a hónap napja (1–31)
5 – 8.	hónap (1–12)
9 – 15.	év (1980-tól)

Idő:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	bit
óra					perc						másodperc					

bit	jelentése
0 – 4.	másodperc két másodperces lépésekben (0–29)
5 – 10.	perc (1–59)
11 – 15.	óra (0–23)

21. ábra: A dátumot és időt tartalmazó szó bitjeinek jelentése

E mezők után egy 8 bájtos adatterület következik, amely kizárólag a DOS rendelkezésére áll, és a programozónak nem szabad megváltoztatnia. Az adatterületen lévő egyes bájtok az egyes DOS verziókban is különbözők lehetnek.

A foglalt adatterület után az aktuális rekord számát tároló mező következik, amely a 0C(h) címen lévő aktuális blokkszámmal együtt az állomány rekordjainak soros feldolgozását teszi lehetővé.

Az FCB utolsó adatmezejében lévő relatív rekordszám a választás szerinti tárhozzáféréssel kapcsolatban jut szerephez (ezt a hozzáférést random vagy véletlen hozzáférésnek is nevezik, nem egészen helyesen). Ennél a hozzáférési módnál az állományon belüli rekordokat nem logikai sorrendjükben, hanem tetszőlegesen megválasztott sorrendben lehet elérni és feldolgozni (a kiválasztást és a sorrendet a program(ozó) határozza meg, és nem a véletlenen múlik). A relatív rekord számát egy 4 bájtos mező határozza meg. Ha a rekord hossza 64 bájt vagy ennél nagyobb, akkor a rendszer ebből csak az első három bájtot használja a rekord számának megadásához.

E „normál" FCB-k mellett a DOS az ún. bővített FCB-ket is támogatja. Ezek a normál FCB-ktől eltérően lehetővé teszik a speciális attribútumú állományokhoz, így például a rejtett vagy a rendszerállományokhoz való hozzáférést is. Lehetővé teszik továbbá a hajlékony- és a merevlemez nevéhez (az ún. kötetnévhez), valamint az alkatalógusok nevéhez való hozzáférést is (ami azonban nem jelenti azt, hogy az aktuális katalóguson kívüli katalógusban lévő állományokhoz hozzá lehetne férni).

Szerkezetüket tekintve csak annyiban térnek el a normál FCB-ktől, hogy 7 bájttal hosszabbak. Ez a 7 bájt, amellyel most az FCB 44 bájtra bővül, a normál FCB elején áll, aminek következtében a további mezők 7-7 bájttal nagyobb címekre tolódnak. A bővített FCB-k szerkezetét a 22. számú ábra mutatja.

00(h)	FF(h)	1 bájt
01(h)	fenntartott (00)	5 bájt
06(h)	állomány attribútuma	1 bájt
07(h)	meghajtóegység azonosítója	1 bájt
08(h)	állomány neve (8 karakter)	8 bájt
10(h)	állománynév kiterjesztése (3 kar.)	3 bájt
13(h)	aktuális blokk száma	1 szó
15(h)	rekord mérete	1 szó
17(h)	állomány mérete	2 szó
1B(h)	módosítás dátuma	1 szó
1D(h)	módosítás időpontja	1 szó
1F(h)	fenntartott	8 bájt
27(h)	aktuális rekord száma	1 bájt
28(h)	relatív rekord száma	4 bájt

22. ábra: Bővített FCB szerkezete

A bővített FCB 0. ofszetcímén mindig 255 áll: ez a bővített FCB azonosítója. Mivel egy normál FCB-nél ezen a tárhelyen mindig a meghajtóegység azonosítója van, ami soha nem lehet 255, a DOS az itt talált érték szerint tudja a normál és a bővített FCB-ket

megkülönböztetni. Ez az egyértelmű megkülönböztetés teszi alapvetően azt is lehetővé, hogy a DOS az FCB-vel támogatott funkciók meghívásakor mind a normál, mind a bővített FCB-ket képes legyen értelmezni.

Az azonosító kódot követő 5 bájtos területet a DOS saját céljaira foglalta le. Ezt a jelenlegi verziókban nem használja, ezért az itt lévő bájtok értéke 00, és ezeket nem is szabad megváltoztatni.

A bővítés utolsó bájtja a 6. ofszetcímen lévő, és csak itt megtalálható állományattribútum. Ennek az attribútumnak a szerepével a tömegtárak kezelését bemutató, 2.12 alfejezetben foglalkozunk részletesen. Miután a bővített FCB-k szerkezetével is megismerkedtünk, nézzük meg, hogyan lehet segítségükkel az állományokhoz hozzáférni.

Először is a programunknak a létrehozandó FCB részére tárterületről kell gondoskodnia, mert ezt a DOS nem teszi meg helyettünk. Ennek a tárterületnek a hossza normál FCB esetén 37 bájt, bővített FCB esetén 44 bájt. Ezt a területet közvetlenül, a programunk adatszegmensén belül is lefoglalhatjuk, de megtehetjük egy DOS funkció segítségével is (21(h) megszakítás, 48(h) funkció). A helyfoglalást követően az FCB valamennyi bájtját kinullázzuk (a bájtokba 0 értéket írunk). Ezután beírjuk az FCB-be annak az állománynak a nevét, amelyikhez hozzá akarunk férni. A név egyes karaktereit közvetlenül is beírhatjuk a megfelelő ofszetcímekre, de mégis az a célszerű, ha – amikor csak lehetséges – a DOS megfelelő szolgáltatásait vesszük igénybe. Ez nemcsak az áttekinthetőbb programozást segíti, hanem a program írójának munkáját is könnyíti azáltal, hogy nem kell minden alkalommal vizsgálnia, vajon bővített vagy normál FCB-vel van-e dolga. Az állománynévnek az FCB-be való beírására a 21(h) megszakítás 29(h) funkciója szolgál. A funkció részére az ES:DI regiszterpárban az FCB kezdőcímét, a DS:SI regiszterpárban pedig az állomány nevének kezdőcímét kell átadni. Az állománynévnek ASCII-karakterláncnak kell lennie, amelyet egy végjel (a 0 ASCII-kódja) zár le. Az állomány neve előtt megadható a meghajtóegység azonosítója, a név után pedig a kiterjesztés (mindkettő opcionális), és az állomány neve, ill. a kiterjesztés tartalmazhat ún. jokerkaraktereket (* vagy ?).

Természetesen az AH regiszterben át kell adni a funkció számát, az AL regiszter alsó bitjeinek pedig az állománynév beírásával kapcsolatos információkat kell tartalmaznia. Ezek jelentésére a funkció részletes ismertetésénél térünk ki (l. a III. kötet DOS megszakítások fejezetét). Egyelőre az is elég, ha a regiszterbe 0-t írunk.

Miután a funkció az állomány nevét beírta az FCB-be, az állomány (és ezzel együtt az FCB is) egy újabb DOS funkcióval megnyitható, ill. létrehozható. Ennek során a DOS egy sor, az állományra vonatkozó fontos információt ír az FCB-be. Ezek egy részét az állomány katalógusbejegyzéséből olvassa ki (állomány mérete, utolsó módosítás dátuma, ideje). Ettől a pillanattól kezdve megnyitott FCB-ről beszélünk.

A CP/M-mel való kompatibilitás érdekében a DOS az FCB megnyitásakor a rekordhosszat 128 bájtra állítja be. Ha ettől eltérő rekordhosszal akarunk dolgozni, akkor a rekordok bájtokban számított hosszát az FCB megnyitását követően be kell írnunk a megfelelő ofszetcímekre. Lényeges, hogy ezt a megnyitást követően végezzük el, mert ellenkező esetben a DOS az FCB megnyitásakor az általunk megadott értéket visszaírná az alapértelmezés szerinti 128 bájtra.

A DOS az FCB-orientált állományműveleteknél alapértelmezés szerint a rekordokat egy DTA-nak nevezett pufferbe helyezi el. (A DTA-ról, ami a Disk Transfer Area

angol kifejezés kezdőbetűit tartalmazza, a COM és az EXE programokkal foglalkozó fejezetben már volt szó.) Ezt az adatátviteli területet a DOS egy program meghívásakor a program elé elhelyezett PSP (program szegmens előtag) 80(h) ofszetcímétől kezdődően helyezi el. Mivel ezen a helyen csak 128 bájt terület áll rendelkezésre – a magasabb címeken már a program helyezkedik el – következik, hogy 128 bájtnál hosszabb rekordokkal csak akkor dolgozhatunk, ha ezt az adatterületet az alapértelmezés szerinti helyről a tár valamely más részére helyezzük át.

Ennél a műveletnél is érvényesek a korábban elmondottak: a tárterületet az adatszegmensben közvetlenül is lefoglalhatjuk, de rábízhatjuk ezt egy DOS funkcióra is. Soros hozzáférés esetén a tárterület nagyságának és a rekord hosszának meg kell egyeznie. Választás szerinti hozzáféréskor a tárterület nagysága attól függ, hogy a DOS megfelelő funkcióival maximum hány rekordot akarunk egyidejűleg beolvasni vagy kiírni (több rekord egyidejű írása/olvasása csak ennél a hozzáférési módnál lehetséges). A tárterület nagyságát a maximális rekordszám és a rekordhossz szorzata adja.

Ha most a programunk a DTA számára egy másik, az alapértelmezéstől eltérő címen kezdődő területet jelöl ki, akkor ennek a kezdőcímét természetesen a DOS tudomására kell hoznunk. Ezt a 21(h) megszakítás 1A(h) funkciójának meghívásával tehetjük meg: az áthelyezett DTA címét a DS (szegmenscím) és a DX (ofszetcím) regiszterekben adjuk át. A puffer méretét nem kell megadnunk, mert a DOS abból indul ki, hogy elegendően nagy ahhoz, hogy a mindenkor továbbítandó adatokat fogadhassa.

Ennyi előkészület után már nincs akadálya annak, hogy az állományon műveleteket végezzünk. Soros feldolgozás esetén az első elérhető rekord az állomány első rekordja. A következő hozzáférés a második, majd a harmadik, negyedik stb. rekordra irányul. A hozzáférések során a DOS az FCB-ben lévő információkat, így pl. az állomány hosszát vagy a rekordmutatót, amelyik az állományon belüli, következő rekord címét adja meg, mindig az aktuális helyzetnek megfelelően módosítja.

Ha viszont a rekordokhoz nem sorosan, hanem választás szerinti sorrendben akarunk hozzáférni, akkor a DOS-szal előzőleg közölni kell annak a rekordnak a számát, amelybe írni, vagy amelyből olvasni akarunk. Ehhez az illető rekord számát be kell írnunk az FCB utolsó mezejébe. A rekordmutató állományon belüli megfelelő pozicionálását aztán már maga az író/olvasó funkció végzi. Egyébként egy állomány feldolgozása közben tetszés szerint váltogathatjuk a soros vagy a választás szerinti hozzáférési módot, mert mindkettőhöz külön-külön funkciók léteznek, amelyek egymástól függetlenül hívhatók.

Magukra az FCB-funkciókra itt nem térünk ki, hanem csak felsoroljuk a legfontosabbakat. Az egyes funkciók részletes ismertetése a III. kötet DOS megszakítások fejezetében található.

Funkció száma	Feladat
0F(h)	Állomány megnyitása
10(h)	Állomány lezárása
13(h)	Állomány törlése
14(h)	Soros olvasás
15(h)	Soros írás
16(h)	Állomány létrehozása
17(h)	Állomány átnevezése
1A(h)	DTA-cím beállítása
21(h)	Választás szerinti olvasás (egy rekord)
22(h)	Választás szerinti írás (egy rekord)
23(h)	Állomány nagyságának lekérdezése
24(h)	Rekordszám beállítása választás szerinti hozzáféréshez
27(h)	Választás szerinti olvasás (egy vagy több rekord)
28(h)	Választás szerinti írás (egy vagy több rekord)
29(h)	Állomány nevének beírása FCB-be

Megemlítünk még néhány, ezekre a funkciókra vonatkozó általános szabályt. Természetesen az FCB-funkciók is lehetővé teszik, hogy egyidejűleg több állományhoz lehessen hozzáférni. Ehhez mindegyik állománynak saját FCB-vel kell rendelkeznie, amelyen keresztül az állomány azonosítható. Ahhoz, hogy a DOS-funkció tudja, hogy az állományok közül mikor és melyikhez kell hozzáférnie, a funkció részére át kell adni az illető állományhoz tartozó FCB címét. A szegmenscímet a DS, az ofszetcímet pedig a DX regiszterben kell elhelyezni. E funkciók többsége a végrehajtását követően egy értéket ír az AL regiszterbe, amivel jelzi, hogy a művelet végrehajtása sikeres volt vagy nem. A sikeres végrehajtást a 0 érték jelzi. Hiba esetén a törlő, nyitó és záró funkciók a 255-öt adják vissza, míg a rekordok írását és olvasását végző funkciók esetén a különböző hibaforrásokhoz és okokhoz saját hibakód tartozik. A hibaforrásokról és az esetleges teendőkről az 59(h) funkció ad részletes felvilágosítást. Ez a funkció azonban csak a DOS 3.0 verziótól felfelé létezik.

Handle-k vagy FCB-k?

Miután az előzőekben megismerkedtünk a DOS által rendelkezésünkre bocsátott két funkciócsoport lényegével, illik egyfajta, a választást megkönnyítendő összegzést adni az egyes csoportok előnyeiről, ill. hátrányairól. Azok számára, akik CP/M-ből vagy UNIX-ból akarnak DOS-környezetbe áttérni, a választás egyértelmű. Azok számára azonban, akik a DOS alatt új programot akarnak kifejleszteni, az itt következők segíthetnek a döntésben.

Handle: Előnyei közé tartozik elsősorban az, hogy a handle-funkciók támogatják a hierarchikus állományszervezést (vagyis segítségükkel tetszőleges katalógus tetszőleges alkatalógusában lévő állomány elérhető), a felhasználói programon belül nincs szükség külön adatterület (FCB) létrehozására, a programnak nem kell foglalkozni a DTA helyével. Használatuk leegyszerűsíti a rekordok választás szerinti elérését, egy állományon belül változó rekordhosszak is használhatók, továbbá hálózati környezetben támogatják az osztott állománykezelést, ill. az állományok rekordszintű védelmét. A handle-funkciókkal az alapértelmezés szerinti I/O eszközök állományként kezelhetők.
Hátrányként tekinthető, hogy a 2.0–3.2 közötti verziókban egy program egyidejűleg csak 20 handle-t tarthat nyitva (a standard I/O eszközökhöz tartozó handle-ket is beleértve). Ugyancsak apróbb hiányosság, hogy a kötetnév handle-funkcióval nem változtatható meg, és a speciális állománynak tekinthető katalógusok tartalmához sem lehet rajtuk keresztül hozzáférni (csak bővített FCB-n keresztül).

FCB: Előnyei ma már csak a korábbi, 1.0–2.0 verziók alatt készült programokban mutatkoznak meg, ahol az egyidejűleg nyitva tartható állományok száma nem volt korlátozva. További előnyként említhető, hogy az FCB-n keresztül könnyebben hozzá lehet férni egy már megnyitott állomány néhány jellemző adatához (méret, dátum, idő).
Hátrányként tulajdonképpen mindazok a jellemzők felsorolhatók, amelyek a handle-funkciók előnyeit jelentik (nem támogatja az állományok hierarchikus szervezését, külön adatterület létrehozását igényli stb.).

A fenti előnyök és hátrányok összevetéséből, valamint annak a korábban már említett tendenciának a figyelembevételéből, amely szerint a DOS egyre inkább a UNIX irányába halad, egyértelműen megállapítható, hogy mind a jelenlegi, mind a jövőbeli fejlesztések a handle-funkciókat részesítik előnyben. Ebből következik, hogy a DOS környezetben készülő programoknak – különös tekintettel a hálózati alkalmazásokra – a handle-funkciók használatára kell épülniük.

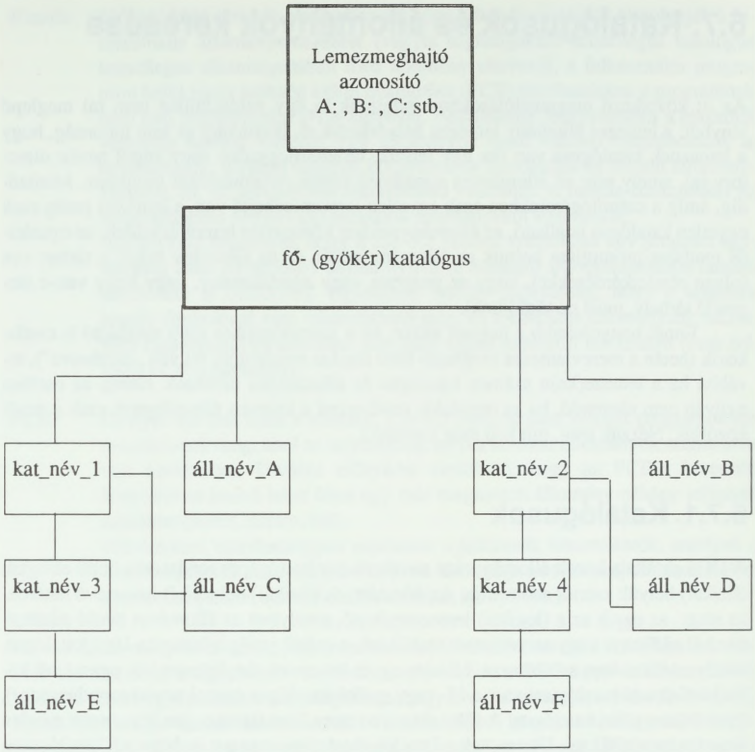
6.7. Katalógusok és állományok keresése

Az itt következő meggondolásokhoz induljunk ki egy valószínűleg nem túl meglepő tényből: a lemezes állomány lemezen helyezkedik el. Bizonyára az sem újdonság, hogy a lemeznek katalógusa van (ha úgy tetszik, tartalomjegyzéke vagy angol nevén directory-ja), amely erre az állományra vonatkozó fontos információkat tartalmaz. Mindaddig, amíg a számítógépünkben csak egyetlen lemezmeghajtó van, a lemezen pedig csak egyetlen katalógus található, az állományunkhoz könnyedén hozzáférhetünk: az operációs rendszer promptjára beírjuk az állomány nevét, és az állomány máris a tárban van (olyan részletkérdésekkel, hogy ez program vagy adatállomány, vagy hogy van-e elegendő tárhely, most ne törődjünk).

Ennél bonyolultabb a helyzet akkor, ha a számítógéphez több meghajtó is csatlakozik (netán a merevlemezes meghajtó több logikai meghajtóra fel van „darabolva"), továbbá ha a lemeze(ke)n számos katalógus és alkatalógus található. Ebben az esetben nyilván nem elegendő, ha az operációs rendszerrel a keresett állománynak csak a nevét közöljük. Nézzük meg, mit kell még tennünk?

6.7.1. Katalógusok

A DOS az általa kezelt állományokat nevük és a számítógépes rendszeren belül elfoglalt (fizikai) helyük szerint azonosítja. Az állományok (fizikai) helyét két összetevő határozza meg: az egyik az a (logikai) lemezmeghajtó, amelyben az állományt tároló adathordozó (hajlékony- vagy merevlemez) található, a másik pedig a lemezen lévő katalógus, amely az állományt tartalmazza. Minden egyes lemeznek ténylegesen két, egymástól különböző típusú katalógusa van: a fő- vagy gyökérkatalógus (angol nevén root directory), és az összes többi katalógus. A főkatalógus a lemez formálásakor jön létre, ezért minden lemezen megtalálható. Ugyancsak a formálás határozza meg azt is, hogy a főkatalógusba hány bejegyzés írható. Egy ilyen bejegyzés vonatkozhat állományra, vagy alkatalógusra, amely ismét tartalmazhat állományokat, ill. további alkatalógusokat. Az egyes alkatalógusok tetszőleges mélységig egymásba ágyazhatók, és tetszőleges számuk lehet. Így alakul ki a katalógusok és az állományok hierarchikus felépítése, ami egy fa struktúrával ábrázolható (23. ábra). Minden katalógusnak neve van, kivéve a főkatalógust, amelyet csak egy \ (fordított törtvonal, angol nevén backslash) karakter jelöl.



23. ábra: Katalógusok és állományok hierarchikus felépítése

Az eddig elmondottak és az ábra alapján már válaszolhatunk a fejezet elején feltett kérdésre. Ahhoz, hogy a DOS az általunk megadott állományt abban a sűrű erdőben, amelyet a különböző meghajtókban lévő katalógusok és állományok mint fák, ágak és levelek egy idő után alkotnak, megtalálja, az állomány nevén túlmenően meg kell adni azt a meghajtót, amelyben az állomány található, továbbá a fa gyökerétől kiindulva a katalógusoknak azt az útvonalát, amelynek a végén a keresett állomány van. A 23. ábrán lévő áll_név_E nevű állomány pl. a következőképpen jelölhető ki:

A:\kat_név_1\kat_név_3\áll_név_E (feltételezve, hogy a lemez az A: jelű meghajtóban van).

A DOS az így megadott útvonalat „megjegyzi", és ezt követően a specifikált meghajtót tekinti aktuális meghajtónak, ill. a specifikált útvonalat tekinti aktuális útvonalnak mindaddig, míg egy másik kijelölés ennek megváltoztatására nem utasítja. A megadott útvonalat a DOS PATH parancsával lehet kijelölni (a path magyar jelentése út, ösvény).

Mint említettük, a DOS a lemez formálásakor csak a főkatalógust hozza létre. A lemezen lévő további katalógusok, alkatalógusok, ill. állományok hierarchikus szerkezetének kialakításáról a felhasználónak kell gondoskodnia. A DOS erre felhasználói szinten számos lehetőséget biztosít: az MD (MAKE DIR) paranccsal új alkatalógus hozható létre, az RD (REMOVE DIR) paranccsal egy alkatalógus törölhető (de csak akkor, ha nem tartalmaz állományokat vagy alkatalógusokat), a CD (CHANGE DIR) paranccsal új aktuális katalógus jelölhető ki. Az aktuális meghajtó a meghajtó azonosítójelének (A, B, C stb.) és egy kettőspontnak a beírásával jelölhető ki (pl. A: vagy B:).

A DOS belsejében a fenti MD, RD és CD parancsoknak rendre a 21(h) megszakítás 39(h), 3A(h) és 3B(h) funkciói felelnek meg. Vizsgáljuk meg ezeket a funkciókat.

A regiszterek szerepe a funkció meghívásakor és visszatérésekor nagyon hasonló. Mindegyik esetben át kell adni a funkció számát (most is az AH regiszterben), valamint azt az útvonalat, amely a manipulálandó katalógushoz vezet. Az útvonal-kijelölés tartalmazhat lemezmeghajtó-azonosítót is, és a kijelölést – amelyet tulajdonképpen egy ASCII karakterlánc képez – egy végjelnek (0 ASCII-kód) kell lezárnia. Ha nem adunk meg meghajtóazonosítót, akkor a DOS a műveletet az aktuális meghajtóban lévő lemezen végzi el.

Az útvonal-kijelölés tárbeli címét a DS (szegmenscím) és a DX (ofszetcím) regiszterekben várják a funkciók. A végrehajtást követően mindhárom funkció törölt átvitelbittel jelzi, hogy a művelet végrehajtása sikeres volt. Hiba esetén az átvitelbit értéke 1, és a hiba kódját az AX regiszter tartalmazza.

A 39(h) funkcióval katalógus hozható létre. Az átadandó útvonal-kijelölés utolsó elemének a létrehozandó új katalógusnévnek kell lennie. Hiba következik be, ha az útvonal-kijelölésben specifikált katalógusok közül valamelyik nem létezik, vagy ha a megadott katalógus már létezik. Hibát okozhat a főkatalógusban egy alkatalógus létrehozása is, mivel az összes többi katalógussal szemben a főkatalóguson belül létrehozható alkatalógusok száma korlátozott, és ezért előfordulhat, hogy a főkatalógus már betelt.

Katalógus törlésére szolgál a 3A(h) funkció, amelynek a végrehajtását azonban a DOS különböző okokból elutasíthatja. Ilyen ok például, ha a törlésre kijelölt alkatalógusban még léteznek állományok, netán további alkatalógusok. A törlendő katalógus természetesen nem lehet az aktuális katalógus sem.

Ha az aktuális katalógust a 3B(h) funkcióval meg akarjuk változtatni, akkor ügyelni kell arra, hogy az útvonal-kijelölésben megadott katalógusok mindegyike létező katalógus legyen. Ennél a funkciónál más hibaok nincs.

Az aktuális lemezmeghajtó-kijelölés megváltoztatásához a 0E(h) funkciót kell meghívni. A funkciónak az AH regiszterben a funkció számát, a DL regiszterben pedig az új, aktuális meghajtó azonosítószámát kell átadni. Az A jelű meghajtó azonosítószáma 0, a B jelűé 1, a C jelűé 2 stb.

Mielőtt az aktuális katalógust a 3B(h) funkcióval megváltoztatnánk, szükség lehet arra, hogy emlékezetünkbe idézzük az éppen aktuális katalógust. A DOS a 47(h) funkciójával erre is egyszerű lehetőséget kínál. Mivel ez a funkció nemcsak az aktuális, hanem valamennyi meghajtóra vonatkozó aktuális katalógus útvonal-kijelölését képes szolgáltatni, meg kell adni annak a meghajtónak az azonosítószámát is, amelyikre a funkció hívása vonatkozik. Ha ez az aktuális egység, akkor a DL-ben 0-t kell átadni. Ellenkező esetben az A meghajtó azonosítója 1, a B meghajtóé 2 stb.

Az azonosítókódon kívül a funkció részére még át kell adni egy 64 bájt nagyságú puffer címét is. A programozó feladata, hogy ennek a puffernek a programon belül helyet foglaljon. A puffer szegmenscímét a DS, ofszetcímét az SI regiszterben kell átadni. A funkció visszatérésekor ebbe a pufferbe kerül a főkatalógustól kiinduló és az aktuális katalógusig vezető útvonal, amelyet egy végjel (a 0 ASCII-kódja) zár le. Az útvonal nem tartalmazza a meghajtó azonosítókódját, továbbá a \ karaktert sem. Ha tehát a funkció meghívásakor az aktuális katalógus éppen a főkatalógus, akkor a pufferbe csak az útvonalat lezáró végjel (00(h)) kerül. Ha a funkció részére ismeretlen meghajtóazonosító kódot adunk át, akkor a funkció visszatérésekor az átvitelbit értéke 1 lesz, és az AX regiszterbe a 0F(h) hibakód kerül.

Vizsgáljuk meg most azokat a funkciókat, amelyekkel az aktuális meghajtóban és az aktuális katalógusban lévő állományokhoz lehet hozzáférni. Amikor egy ilyen hozzáférést hajtunk végre, akkor tulajdonképpen az ismert keresési műveletet végezzük: elkészítünk egy, a keresett állományra jellemző mintát, és ezt a katalógusban lévő valamennyi állománnyal összehasonlítjuk. Ha találunk egyezést, akkor a keresett állományt megtaláltuk (legalábbis az első egyezőt). Az ilyen hozzáféréseket tehát állományok keresésének is felfoghatjuk. Ezeknél a műveleteknél is megtalálható a funkciók megkettőzése: az állományok FCB- és handle-funkciókkal is kereshetők. Az FCB-funkcióknak az a hátrányuk, hogy csak egy, meghatározott meghajtó aktuális katalógusában lévő állományok keresését teszik lehetővé, míg a handle-funkciókkal bármelyik meghajtó bármelyik katalógusában lévő állományok kereshetők. A handle megjelölés ezekre a funkciókra valójában nem illik, mert ezeket nem handle-n keresztül kell meghívni. A jelző inkább arra utal, hogy ezek a funkciók az alkatalógusoknak (és ezzel együtt a handle-funkcióknak) a DOS 2.0 verziójába történt bevezetésekor kerültek a DOS funkciókészletébe, míg az FCB-funkciók már az 1.0 verzióban is rendelkezésre álltak.

6.7.2. Állományok keresése FCB-funkciókkal

Állományok kereséséhez a DOS a keresés két, különböző típusát bocsátja a felhasználó rendelkezésére: az egyik típus az ún. „első katalógusbejegyzés keresése", a másik pedig a „további katalógusbejegyzések keresése". A keresés során először mindig az első bejegyzést (vagyis állományt) kell keresni. Az első itt természetesen nem a lemez katalógusában legelől álló bejegyzést, ill. állományt jelenti, hanem az első olyan állományt, amely a keresettnek megfelel (utalunk az előbb említett keresési mintára). Ha ez a keresés sikerrel jár, akkor következhet a további – természetesen ismét csak a mintának megfelelő – állományok keresése. (A keresésnek ugyanez a stratégiája a handle-funkcióknál is.)

Az előbbi, kétféle típusú keresésnek megfelelően a DOS két FCB-funkciót bocsát a felhasználó rendelkezésére: a 11(h) az első állomány keresésére, a 12(h) pedig a további állományok keresésére szolgál. A keresésnél fontos szerephez jutnak a 6.6.2. pontban már ismertetett FCB-k, mert ezeken keresztül bonyolódik le a meghívó program és a két funkció közötti adatcsere. Vegyük sorra most azokat a lépéseket, amelyeket egy állomány kereséséhez egy programon belül meg kell tenni.

Először is a programnak gondoskodnia kell arról, hogy két FCB számára megfelelő tárterület álljon rendelkezésre. Ehhez a program a saját adatterületén belül is lefoglalhat helyet, de a 48(h) funkció segítségével a DOS-tól is kérhet területet. A megfelelő terület alatt az értendő, hogy a programozó dönti el, hogy normál vagy bővített FCB-kel akar-e dolgozni. Az utóbbiaknak a normál FCB-khez képest az az előnyük, hogy rajtuk keresztül speciális attribútumú állományok (rejtett, csak olvasható, rendszerállományok stb.), továbbá kötetnevek és alkatalógusok is kereshetők. Mindkét funkció képes normál és bővített FCB-kel is dolgozni. A program által lefoglalt vagy a DOS által kiutalt területre kerülő két FCB közül az egyik a keresett állomány nevét tárolja – ez a kereső minta –, míg a másikba a DOS a talált állomány nevét (és egyéb jellemzőit) írja be. Azért, hogy a továbbiakban ezt a két FCB-t egymástól meg tudjuk különböztetni, legyen az egyik neve keres_FCB, a másiké pedig talált_FCB.

A korábbiakból már tudjuk, hogy a DOS az FCB-orientált állománykezelésnél igen fontos szerepet játszó adatátviteli területet (DTA) alapértelmezés szerint a tárban az állomány elé helyezett PSP-ben helyezi el. Igen lényeges tudni, hogy a DOS állománykereső funkciói is a DTA-t használják az adatcsere lebonyolítására.

A DTA azonban nemcsak e két funkció, hanem minden olyan funkció számára fontos adatterület, amely adatokat mozgat a lemez és a tár között. Arról is volt már szó, hogy ez az alapértelmezés szerinti 128 bájt általában túlságosan kicsi az adatcseréhez, ezért a programok többnyire új területre helyezik át a DTA-t.Ahhoz tehát, hogy az állománykereső funkciók számára a talált_FCB-t lehessen aktuális adatátviteli területnek kijelölni, először meg kell állapítani a mindenkori DTA címét, majd az itt talált DTA-t át kell helyezni a talált_FCB kezdőcímére.

A DTA címét a DOS 21(h) megszakítás 2F(h) funkciójával állapíthatjuk meg, majd az így kapott címet az 1A(h) funkcióval áthelyezzük a talált_FCB kezdőcímére. A keresés befejezése után a DTA-t ugyanezzel a funkcióval visszahelyezhetjük az eredeti helyére.

Miután a DTA-t a talált_FCB-re állítottuk, következő lépésként a keresendő állomány nevét beírjuk a keres_FCB-be. Azért, hogy a keresés ne csak egy, meghatározott állományra korlátozódjon, a funkció lehetővé teszi, hogy az állományok nevében jokerkarakterek is szerepeljenek. Az állomány nevét közvetlenül vagy a 29(h) funkcióval írhatjuk be az FCB-be. (Ha a közvetlen beírást választjuk – ami nem javasolt – , akkor az állomány nevét balra igazítva, a 0. ofszetcímtől kezdődően kell beírni, az üresen maradt helyet pedig szóköz karakterekkel (a 32-es ASCII-kód) kell feltölteni. Ugyanez vonatkozik az állomány kiterjesztésére is.) Ha a keresést valamennyi állományra ki akarjuk terjeszteni (ami úgye nem más, mind a DOS DIR parancsának megfelelő keresés), akkor az állomány nevét, ill. a kiterjesztést a „*.*" alakban kell megadnunk. Ha bővített FCB-vel dolgozunk, akkor a keres_FCB attribútummezejébe is beírható egy érték, amellyel meg-

határozott attribútumú állományok kereshetők (a különböző attribútumokról a későbbiekben még lesz szó).

Ezzel az előkészítő műveleteket elvégeztük, és megkezdődhet az aktuális katalóguson belüli első (megfelelő) állomány keresése. Ehhez meghívjuk a 11(h) funkciót úgy, hogy a funkció számát az AH regiszterbe, a keres_FCB szegmenscímét a DS, ofszetcímét pedig a DX regiszterbe írjuk. Ha a megadott néven sikerült egy állományt találni, akkor a funkció visszatérésekor az AL regiszterbe 0, ellenkező esetben 255 kerül. A talált állomány neve és attribútuma (amennyiben bővített FCB-kkel dolgozunk) most a talált_FCB-ből kiolvasható. A további állományok keresésére ezután már nem a 11(h), hanem a 12(h) funkciót kell meghívni. A regiszterek szerepe meghíváskor és visszatéréskor ugyanaz, mint a 11(h) esetén. Ha valamelyik meghívást követően az AL-ben 255-öt kapunk vissza, akkor ez azt jelenti, hogy a katalógusban a megadott mintának megfelelő további állomány már nincs, azaz a keresés befejeződött.

6.7.3. Állományok keresése handle-funkciókkal

A handle-funkciók az állományok keresését nemcsak kényelmesebbé, hanem egyszerűbbé is teszik. Itt is külön funkció szolgál az első és a további (megfelelő) állományok keresésére (a 4E(h) és a 4F(h) funkciók). A megtalált állományok adatait mindkét funkció a DTA-ba viszi. Ezért a DTA-t a funkciók meghívása előtt most is át kell helyezni egy, a mindenkori program által hozzáférhető pufferterületre, amelynek – mint látni fogjuk – minimum 43 bájt hosszúságúnak kell lennie. Csakúgy, mint az FCB-funkcióknál, itt is fennáll, hogy a DTA-t a keresés befejeződése után vissza kell helyezni az eredeti helyére.

A 4E(h) funkció meghívásához a funkció számát az AH regiszterben, a keresendő attribútumértéket a CX regiszterben, a keresendő állomány nevének tárbeli címét pedig a DS:DX regiszterpárban kell átadni. Az állománynévnek ASCII-karakterlánc formájában kell a tárban lennie, amelyet egy végjel (0 ASCII-kód) zár le. A karakterlánc a meghajtó azonosítókódja mellett teljes útvonal-kijelölést, és természetesen jokerkaraktereket is tartalmazhat, így tehát állományok csoportja is kereshető.

Ha nem adunk meg útvonalat, akkor a DOS az aktuális katalógusban keres. Ugyanez vonatkozik a meghajtóra is: ha nem specifikálunk meghajtót, akkor a DOS a katalógust az aktuális lemezmeghajtóban keresi. A funkció visszatérésekor az átvitelbit jelzi, hogy talált-e (megfelelő) állományt. Ha nem talált, akkor az átvitelbit értéke 1, és az AX regiszterben a hibakód van. A 2-es hibakód azt jelenti, hogy a megadott útvonal-kijelölés (vagy valamelyik eleme) nem létezik, a 18-as pedig azt, hogy nem talált olyan állományt, amely a megadott attribútumértéknek megfelel (a „közönséges" állományok attribútumértéke 0, vagyis a normál keresésnek a 0 attribútumérték felel meg). Ha viszont az átvitelbit értéke 0, akkor a DTA-ként szolgáló puffer első 43 bájtjába beíródtak a talált állományra vonatkozó adatok a következők szerint:

Bájt	Tartalom
0 – 20. :	foglalt
21. :	állomány attribútuma
22 – 23. :	állomány utolsó módosításának ideje
24 – 25. :	állomány utolsó módosításának dátuma
26 – 27. :	állomány méretének felső szava
28 – 29. :	állomány méretének alsó szava
30 – 42. :	állománynév és kiterjesztés ASCII-kódok karakterláncaként, egy végjellel (0 ASCII-kód) lezárva

Minden további keresés a 4F(h) funkción keresztül történik. Meghívásához a számának az AH regiszterben történő átadásán kívül más paramétert nem kell megadni. Itt is az átvitelbit jelzi, hogy a mindenkori katalógusban vannak-e még a keresési mintának megfelelő további állományok.

6.7.4. Programpéldák

A katalógusok felépítésével és kezelésével kapcsolatos elméleti ismeretek elmélyítését és szemléltetését az alábbi, Basic, Turbo Pascal és C nyelvű mintaprogramok segíthetik, amelyek az ismertetett handle funkciókat is felhasználva egy paraméterrel kijelölhető katalógus állományait jelenítik meg. A képernyőn egy ablakban jelennek meg az állományok nevei és főbb jellemzőik. A DOS DIR parancsnál megadható /P paraméter ezekben a programokban „beépített", egy teljes ablaknyi információ megjelenítése után egy tetszőleges billentyű megnyomásával kérheti a felhasználó a megjelenítés folytatását, így az állományok és adataik kényelmesen megtekinthetőek.

A megjeleníteni kívánt katalógus elérési útját paraméterként kell megadni. Ebben a paraméterben a DOS-ból ismert „joker" karakterek is felhasználhatók. Amennyiben nem adunk meg paramétert, alapértelmezés szerint az aktuális meghajtóegység aktuális katalógusának állományait jelenítik meg a programok.

A programok „lelke" a 21(h) DOS megszakítás 4E(h) és 4F(h) funkcióinak meghívása. Előbbi segítségével a kiválasztott katalógus első, az utóbbit felhasználva pedig a soronkövetkező bejegyzése olvasható be. Jól használhatók más programokban is a képernyőablakban görgetést végző rutinok, melyek a 21(h) megszakítás 10(h) funkcióját hívják meg. A mintaprogramok közül a Turbo Pascal változat kihasználja a nyelv adta lehetőségeket és a megjelenítés területét definiálja képernyőablakként, ezzel megoldja a görgetés problémáját.

A Turbo Pascal és a C nyelvű változatban egy katalógus bejegyzés összes adata magasabb szinten, egy egységként kezelhető (RECORD ill. STRUCT típusú változót definiálva a bejegyzés számára). Ezekben az esetekben a 21(h) megszakítás 1A(h) funkciójával kell a DTA címét a felvett változó címének megfelelően beállítani. A BASIC

változatban nem ennyire egyszerű a helyzet, hiszen mint ismert, a BASIC program futása során a tárban található változók rendre megváltoztathatják címüket. Ha tehát egy bejegyzés számára egy tömböt vennénk fel, annak kezdőcíme a futás során elcsúszhatna a DOS-ban tárolt DTA címhez képest. A megoldás a 21(h) megszakítás 2F(h) funkciójának segítségével a DTA szegmens- és ofszetcímének beolvasása. Ezt követően a bejegyzés megjelenítés során mindig ezen a címen keresztül érhetők el a kívánt adatok.

```
1000 '
1005 '
1010 '                    K A T A L O G . B A S
1020 '
1030 '  Funkciója : a kiválasztott (al)katalógus állományainak meg-
1040 '              jelenítése a képernyőn.
1045 '
1050 '
1060 '
1070 DEFINT Q                                   'Q "dummy" egész érték.
1080 KEY OFF : CLS
1090 PRINT : PRINT
1100 PRINT "FIGYELMEZTETÉS :" : PRINT
1110 PRINT "A 60000-es címtől kezdődően betöltött megszakításhívó ru-"
1120 PRINT "tin felülírását elkerülendő a GWBASIC-et GWBASIC /M:60000"
1130 PRINT "paranccsal kell indítania. Ha nem így hívta a GWBASIC-et,"
1140 PRINT "akkor üsse le az <s> billentyűt, egyébként egy tetszőle-"
1150 PRINT "ges másik billentyűt !"
1160 '
1170 Z$ = ""
1180 WHILE Z$ = ""                              'Helytelen indítás
1190    Z$ = INKEY$                             'esetén a program
1200 WEND                                       'leállítása.
1210 IF Z$ = "s" OR Z$ = "S" THEN CLS:END
1220 '
1230 GOSUB 9000                                 'Megszakításhívó rutin
1240                                            'betöltése.
1250 CLS
1260 '
2000 '
2010 '          P R O G R A M T Ö R Z S
2020 '
2030 '
2040 PRINT : PRINT
2050 PRINT "Adja meg a kívánt katalógushoz tartozó útvonalat ! A"
2060 PRINT "megadás során a DOS-ból ismert általános célú helyettesí-"
2070 PRINT "tő karakterek is használhatók (* és ?). Ha nem ad meg út-"
2080 PRINT "vonalat, az aktuális katalógus tartalmát jeleníti meg a"
2090 PRINT "program."
2100 PRINT : PRINT
2110 '
2120 INPUT "Útvonal : " , UT$                   'Útvonal beolvasása,
2130 IF UT$ = "" THEN UT$ = "*.*"               'ha üres, *.*-ra állítás.
2140 NSORZ = 18                                 'Sorok max. száma 18.
2150 GOSUB 3000                                 'Beolvasás, megjelenítés.
2160 '
2170 Z$ = ""
2180 WHILE Z$ = ""
2190    Z$ = INKEY$                             'Kiszállás előtt egy
2200 WEND                                       'billentyű megnyomá-
2210 CLS                                        'sára várakozás.
2220 END
2230 '
3000 '
3010 '  KATALÓGUS BEOLVASÁSA ÉS MEGJELENÍTÉSE
3020 '     BE : UT$ az útvonal.
3030 '          NSORZ a sorok száma az ablakban.
3040 '     KI : -
3050 '
3060 '
3070 DIM HONAP$[11]                             'Hónapnevek beolvasása
3080 RESTORE 3780                               'a HONAP$ tömbbe.
3090 FOR IZ = 0 TO 11
3100    READ HONAP$[IZ]
```

```
3110 NEXT
3120 MSZX = &H21                                    '21(h) megszakítás.
3130 FUNX = &H2F                                    'DTA cím funkció.
3140 '
3150 CALL RCIM(MSZX,FUNX,Q,OFFHX,OFFLX,Q,Q,Q,Q,Q,Q,DTASEGX,Q)
3160 '
3170 DTAOFSX = OFFLX + OFFHX * 256
3180 CLS
3190 ESORX = INT((20 - NSORX) / 2) + 1              'Első sor a képernyőn.
3200 LOCATE ESORX,13
3210 PRINT "Katalógus tartalomjegyzék : " + UT$
3220 XXX$ = SPACE$(12)
3230 PRINT  XXX$;
3240 PRINT "┌──────────────┬──────┬──────────┬──────┬─────┐"
3250 PRINT  XXX$;
3260 PRINT "│ Állománynév  │ Méret │   Dátum  │ Időpont │RHSVD│"
3270 PRINT  XXX$;
3280 PRINT "├──────────────┼──────┼──────────┼──────┼─────┤"
3290 FOR IX = 1 TO NSORX
3300 PRINT  XXX$;
3310 PRINT "│              │      │          │      │     │"
3320 NEXT
3330 PRINT  XXX$;
3340 PRINT "└──────────────┴──────┴──────────┴──────┴─────┘";
3350 '
3360 NX      =  -1                          'össz. bejegyzésszám növelése.
3370 NTOTALX =   0                          'Megjelenített bejegyzésszám
3380 ATTRX = 255                            'Tetsz. attr. beállítása.
3390 GOSUB 4000                             'Első bejegyzés keresése,
3400 IF NOT(TALALTX) THEN 3700              'ha nincs egy sem : VÉGE.
3410 NTOTALX = NTOTALX + 1                  'összes bejegyzésszám növelése.
3420 NX = NX + 1                            'Megjelent bejegyzésszám növelése.
3430 '
3440 IF NX <> NSORX THEN 3590               'Ha betelt az ablak,
3450 LOCATE ESORX+NSORX+5, 13               'az ablak alatti sorban
3460 COLOR 0,7                              'inverz üzenet megjelenítés.
3470 '
3480 PRINT "         Üssön le egy tetszőleges billentyűt ..          ";
3490 '
3500 Z$ = ""                                'Várakozás egy billentyű
3510 WHILE Z$ = ""                          'leütésére.
3520    Z$ = INKEY$
3530 WEND
3540 '
3550 LOCATE ESORX+NSORX+5,13                'Az ablak alatti
3560 COLOR 7,0                              'sor törlése normál
3570 PRINT SPC(56);                         'színnel.
3580 NX     = -1                            'Számláló "nullázása".
3590 GSORX =  1 : SZINX = 7
3600 BFSX  = ESORX + 3      : BFOX = 13     'Sor- és oszlopkoordi-
3610 JASX  = ESORX+NSORX + 2: JAOX = 66     'náták beállítása.
3620 GOSUB 7000                             'Felfelé görgetés.
3630 LOCATE ESORX+NSORX+3, 14               'Kiírás utolsó ablaksorba.
3640 '
3650 PRINT "                  │      │          │      │     ";
3660 '
3670 GOSUB 6000                             'Bejegyzés megjelenítése.
3680 GOSUB 5000                             'Köv. bejegyzés beolvasása.
3690 IF TALALTX THEN 3410                   'Ha volt még, akkor tovább,
3700                                        'ha nem volt több, az ablak
3710 LOCATE ESORX+NSORX+5, 21               'alatt az összegző üzenet
3720                                        'megjelenítése.
3730 '
3740 PRINT "összesen";NTOTALX;"állomány a(z a)katalógusban !";
```

```
3750 COLOR 7,0
3760 RETURN
3770 '
3780 DATA "JAN","FEB","MAR","APR","MAJ","JUN"
3790 DATA "JUL","AUG","SEP","OKT","NOV","DEC"
3800 '
4000 '
4010 '  ELSŐ BEJEGYZÉS MEGKERESÉSE
4020 '     BE : UT$ az UTvonal.
4030 '          ATTRX az állomány ATTRibútum
4040 '     KI : TALALTX -1 ha van, 0 ha nincs
4050 '               bejegyzés.
4060 '
4070 '
4080 UT$    = UT$ + CHR$(0)                    'Kar.lánc vége jelzés.
4090 MSZX   = &H21                             '21(h) megszakítás.
4100 FUNX   = &H4E                             'Első bejegyzés funkció.
4110 ATTLX  = ATTRX AND 255                    'Attribútum LOW bájt.
4120 ATTHX  = INT(ATTRX / 256)                 'Attribútum HIGH bájt.
4130 OFFLX  = PEEK(VARPTR(UT$)+1)              'Ofszetcím LOW bájt.
4140 OFFHX  = PEEK(VARPTR(UT$)+2)              'Ofszetcím HIGH bájt.
4150 '
4160 CALL RCIM(MSZX,FUNX,Q,Q,Q,ATTHX,ATTLX,OFFHX,OFFLX,Q,Q,Q,FLAGSX)
4170 '
4180 TALALTX = ((FLAGSX AND 1) = 0)            'Átvitel bit vizsgálata.
4190 RETURN
4200 '
5000 '
5010 '  KÖVETKEZŐ BEJEGYZÉS MEGKERESÉSE
5020 '     BE : UT$ az UTvonal.
5030 '          ATTRX az állomány ATTRibútum
5040 '     KI : TALALTX -1 ha van, 0 ha nincs
5050 '               több bejegyzés.
5060 '
5070 '
5080 MSZX = &H21                               '21(h) megszakítás.
5090 FUNX = &H4F                               'Következő bejegyzés funkció.
5100 '
5110 CALL RCIM(MSZX,FUNX,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,FLAGSX)
5120 '
5130 TALALTX = ((FLAGSX AND 1) = 0)            'Átvitel bit vizsgálata.
5140 RETURN
5150 '
6000 '
6010 '  EGY BEJEGYZÉS MEGJELENÍTÉSE A KÉPERNYŐN
6020 '     BE : ESORX az ablak Első SORa.
6030 '          NSORX a sorok száma az ablakban.
6040 '          DTAOFSX a DTA OFSzetcíme.
6050 '          HONAP$ a HÓNAPok rövidítése.
6060 '     KI : -
6070 '
6080 '
6090 DEF FNX(X) = PEEK(DTAOFSX + X)
6100 DEF SEG  = DTASEGX                        'DTA szegmenscím beállítása.
6110 '
6120 NEV$ = ""
6130 IX   = 30                                 'Állománynév ofszet DTA-ban.
6140 WHILE FNX(IX) <> 0                        'A karakterlánc végét jelző
6150    NEV$ = NEV$ + CHR$(FNX(IX))            'CHR$(0)-ig az állománynév
6160    IX = IX + 1                            'konkatenálása.
6170 WEND
6180 '
6190 MERET  = FNX(26) + 256*(FNX(27) + 256*(FNX(28) + 256*FNX(29)))
6200 DATUM  = FNX(24) + FNX(25)*256
```

```
6210 ORAIDO = FNX(22) + FNX(23)*256
6220 '
6230 LOCATE ESORX+NSORX+3, 15
6240 PRINT NEV$;                                        '-NÉV kiírása.
6250 '
6260 LOCATE ESORX+NSORX+3, 32
6270 PRINT USING "######"; MERET;                       '-MÉRET kiírása.
6280 '
6290 LOCATE ESORX+NSORX+3, 40
6300 PRINT USING "####";INT(DATUM / 512) + 1980;
6310 PRINT " " + HONAP$[(INT(DATUM / 32) AND 15) - 1];
6320 PRINT USING "###";DATUM AND 31;
6330 PRINT ".";                                         '-DATUM kiírása.
6340 '
6350 LOCATE ESORX+NSORX+3, 54
6360 PRINT USING "##";INT(ORAIDO / 2048);
6370 PRINT " : ";                                       '-IDŐPONT kiírása.
6380 PRINT USING "##";INT(ORAIDO / 32) AND 63;
6390 LOCATE ESORX+NSORX+3, 63
6400 '                                                  '-ATTR.-ok kiírása.
6410 '
6420 FOR IX = 0 TO 4
6430     IF (FNX(21) AND (2^IX)) <> 0 THEN PRINT "X"; ELSE PRINT " ";
6440 NEXT IX
6450 '
6460 DEF SEG : RETURN
6470 '
7000 '
7010 ' ┌──────────────────────────────────────────┐
7020 ' │ KÉPERNYŐ FELFELE GÖRGETÉSE               │
7030 ' │    BE : GSORX a Görgetendő SOROK száma   │
7040 ' │         BFOX a Bal Felső Oszlop          │
7050 ' │         BFSX a Bal Felső Sor             │
7060 ' │         JAOX a Jobb Alsó Oszlop          │
7070 ' │         JASX a Jobb Alsó Sor             │
7080 ' │         SZINX a törölt terület SZINe     │
7090 ' │    KI : -                                │
7100 ' │ Ha GSORX 0, akkor a terület törlődik.    │
7110 ' └──────────────────────────────────────────┘
7120 MSZX=&H10                                          '16(h) megszakítás.
7130 FUNX=6                                             'Felfelé görgetés funkció.
7140 '
7150 CALL RCIM(MSZX, FUNX, GSORX, SZINX, 0, BFSX, BFOX, JASX, JAOX, 0, 0, 0, 0)
7160 RETURN
7170 '
9000 '
9010 ' ┌──────────────────────────────────────────┐
9020 ' │ A MEGSZAKÍTÁSHÍVÓ RUTIN INICIALIZÁLÁSA   │
9030 ' │    BE: -                                 │
9040 ' │    KI: RCIM a Rutin CIMe                 │
9050 ' └──────────────────────────────────────────┘
9060 RCIM = 60000!                  'A rutin kezdőcíme a BASIC szegmensben.
9070 DEF SEG                        'A BASIC szegmens beállítása.
9080 RESTORE 9140
9090 FOR IX = 0 TO 160                                  'A rutin betöltése a
9100     READ XX : POKE RCIM + IX, XX                   'tárba.
9110 NEXT
9120 RETURN
9130 '
9140 DATA    85, 139, 236,  30,   6, 139, 118,  30, 139,   4, 232, 140
9150 DATA     0, 139, 118,  12, 139,  60, 139, 118,   8, 139,   4,  61
9160 DATA   255, 255, 117,   2, 140, 216, 142, 192, 139, 118,  28, 138
9170 DATA    36, 139, 118,  26, 138,   4, 139, 118,  24, 138,  60, 139
9180 DATA   118,  22, 138,  28, 139, 118,  20, 138,  44, 139, 118,  18
```

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

```
/*  ┌─────────────────────────────────────────────────────────────┐  */
/*  │                                                             │  */
/*  │                     K A T A L O G . C                       │  */
/*  │                                                             │  */
/*  │ Funkciója : a kiválasztott (al)katalógus állományainak meg- │  */
/*  │             jelenítése a képernyőn.                         │  */
/*  │                                                             │  */
/*  └─────────────────────────────────────────────────────────────┘  */

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

#define FALSE    0
#define TRUE     1
#define NSOR     18                         /* Sorok max. száma 18.  */
#define NM       0x07                       /* Normál megjelenítés.  */
#define IM       0x70                       /* Inverz megjelenítés.  */

#define byte unsigned char
#define ESOR (20-NSOR >> 1)                 /* Az ablak (e)lső (sor)a. */

struct bejegyzes {
                  char           dummy[21] ;
                  byte           attr      ;
                  unsigned int   idopont   ;
                  unsigned int   datum     ;
                  unsigned long  meret     ;
                  char           allnev[13] ;
                  };

/*  ┌──────────────────────────────────────────────┐  */
/*  │ AKTOLDAL : AZ AKTUALIS KÉPERNYŐOLDAL          │  */
/*  │            MEGHATAROZASA                      │  */
/*  └──────────────────────────────────────────────┘  */

byte AKTOLDAL()

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

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

/*  ┌──────────────────────────────────────────────┐  */
/*  │ GORGETES : MEGADOTT KÉPERNYŐTARTOMANY         │  */
/*  │            FELFELE GÖRGETÉSE v. TÖRLÉSE       │  */
/*  └──────────────────────────────────────────────┘  */

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

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

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

```
regiszter.h.ah = 0x06;                  /* Görgetés funkció.              */
regiszter.h.al = gsor;                  /* Görgetendő sorok száma.        */
regiszter.h.bh = szin;                  /* Az üres sorok színe.           */
regiszter.h.ch = bfs ;                  /* Ablak-                         */
regiszter.h.cl = bfo ;                  /*        koor-                   */
regiszter.h.dh = jas ;                  /*              diná-             */
regiszter.h.dl = jao ;                  /*                    ták.        */
int86(0x10, &regiszter, &regiszter);          /* 10(h) megszakítás.       */
}

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

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

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

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

/*  ┌──────────────────────────────────────────────┐                      */
/*  │ KUROLV : AZ AKT. KÉPERNYŐOLDALI KURZOR       │                      */
/*  │          POZíCIÓJÁNAK BEOLVASÁSA             │                      */
/*  └──────────────────────────────────────────────┘                      */

void KUROLV(oszlop, sor)
int *oszlop;                            /* Az eredmény átadása mutatók    */
int *sor;                               /* segítségével történik.         */

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

regiszter.h.ah = 3;                           /* Olvasás funkció az ak-   */
regiszter.h.bh = AKTOLDAL();                  /* tuális oldalra kiadva.   */
int86(0x10, &regiszter, &regiszter);          /* 10(h) megszakítás.       */
*oszlop = regiszter.h.dl;                     /* Az eredmény a DH - DL    */
*sor    = regiszter.h.dh;                     /* regiszterpárba kerül.    */
}

/*  ┌──────────────────────────────────────────────┐                      */
/*  │ KARIR : EGY KARAKTER KIÍRÁSA ADOTT           │                      */
/*  │         ATTRIBÚTUMMAL                        │                      */
/*  └──────────────────────────────────────────────┘                      */

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

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

regiszter.h.ah = 9;                     /* Karakter megjelenítés funkció. */
regiszter.h.al = kar;
regiszter.h.bh = AKTOLDAL();            /* Az aktuális képernyőoldal.     */
regiszter.h.bl = szin;
```

```
regiszter.x.cx = 1;                    /* Egyszer történik megjelenítés.  */

int86(0x10, &regiszter, &regiszter);          /* 10(h) megszakítás.       */
}
/*
/*  IR : KARAKTERLANC KIÍRÁSA ADOTT KÉPER-                                */
/*       NYŐPOZÍCIÓTÓL KEZDVE                                             */
/*  A kar. lánc végét egy '\0' karakter jelzi                             */
/*
void IR(oszlop, sor, txt, szín)
int  oszlop;
int  sor;                              /* A megjelenítés koordinátái, a   */
char *txt;                             /* a kiírandó kar.lánc és a hozzá  */
int  szín;                             /* tartozó attribútum, ill. szín.  */

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

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

/*
/*  CLRSCR : KÉPERNYŐ TÖRLÉSE                                             */
/*
void CLRSCR()

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

/*
/*  KERETEZES : LISTA KERETÉNEK KIRAJZOLÁSA                               */
/*
void KERETEZES()

{
byte i;

CLRSCR();
IR(14,ESOR,"Katalógus tartalomjegyzék : ",NM);
IR(14,ESOR+1,
   "╔═══════════════════╤═══════╤══════════╤═══════╤═══════╗",NM);
IR(14,ESOR+2,
   "║  Állománynév      | Méret |  Dátum   | Időpont |RHSVD|",NM);
IR(14,ESOR+3,
   "╟───────────────────┼───────┼──────────┼───────┼───────╢",NM);
for (i = ESOR+4; i < ESOR+4+NSOR; i++)
IR(14,i,
   "║                   |       |          |       |       ║",NM);
IR(14,ESOR+NSOR+4,
   "╚═══════════════════╧═══════╧══════════╧═══════╧═══════╝",NM);
}
```

```
/*
/*  KIIR : EGY BEJEGYZÉS KIÍRÁSA A KÉPERNYŐN                    */
/*
/*                                                              */

void KIIR(aktsor, sor)
struct bejegyzes *aktsor;
byte sor;                                       /* A megjelenítés sora. */

{
byte i;
static char *honap[] = {
                        "JAN", "FEB", "MAR", "APR", "MAJ", "JUN",
                        "JUL", "AUG", "SZE", "OKT", "NOV", "DEC"
                        };

KURPOZ(16, sor);
   for (i=0; (*aktsor).allnev[i] && i<15 ;
   printf("%c",
        (*aktsor).allnev[i++]));                        /* NÉV kiírás.      */

KURPOZ(33, sor);
   printf("%6lu",
      (*aktsor).meret);                                 /* MÉRET kiírás.    */

KURPOZ(41, sor);
   printf("%4d %3s %2d. ",                              /* DATUM kiírás.    */
         ((*aktsor).datum >> 9) + 1980,
    honap[(((*aktsor).datum >> 5 & 15) - 1],
         (*aktsor).datum & 31);

KURPOZ(55, sor);
   printf("%2d : %2d",                                  /* IDŐPONT kiírás.  */
      (*aktsor).idopont >> 11,
         ((*aktsor).idopont >> 5 & 63);

KURPOZ(64, sor);
   for (i = 1; i <= 16; i <<= 1)
     if ((*aktsor).attr & i) printf("X");
     else printf(" ");                                  /* ATTRIB. kiírás.  */
}

/*
/*  VANELSO : ELSŐ BEJEGYZÉS MEGKERESÉSE                        */
/*     KI : TRUE ha van, FALSE ha nincs be-                     */
/*             jegyzés.                                         */
/*                                                              */

byte VANELSO(utvonal, attr)
char *utvonal;                                  /* Katalógus maszk. */
unsigned int attr;                              /* Választott attr. */

{
union REGS regiszter;           /* Regiszter union deklaráció.      */
struct SREGS szegreg;           /* Szegmensreg. struktúra dekl.     */

segread(&szegreg);              /* Szegmensregiszterek beolvasása.  */

regiszter.x.cx = attr;
regiszter.x.dx = (unsigned int) utvonal;
regiszter.h.ah = 0x4E;                          /* Köv. bejegyzés funk. */
intdosx(&regiszter, &regiszter, &szegreg);      /* 21(h) megszakítás.   */
```

```
return(!regiszter.x.cflag);                    /* Átvitelbit vizsgála- */
}                                              /* ta, ha 0, talált.    */

/*                                                                      */
/*  VANMEG : KÖVETKEZŐ BEJEGYZÉS MEGKERESÉSE                            */
/*     KI : TRUE ha van, FALSE ha nincs több                            */
/*          bejegyzés.                                                  */
/*                                                                      */

byte VANMEG()
{
union REGS regiszter;                          /* Regiszter union deklaráció. */

regiszter.h.ah = 0x4F;                         /* Köv. bejegyzés funkció.  */
intdos(&regiszter, &regiszter);                /* 21(h) megszakítás.       */
return(!regiszter.x.cflag);                    /* Átvitelbit vizsgálata,   */
}                                              /* ha 0, talált.            */

/*                                                                      */
/*  DTABEALL : DTA CÍMÉNEK BEÁLLÍTÁSA                                   */
/*        BE : A beállítani kívánt ofszetcím                            */
/*                                                                      */

void DTABEALL(ofszet)
unsigned int ofszet;                           /* A beállítani kívánt ofszetcím. */

{
union REGS regiszter;                          /* Regiszter union deklaráció.    */
struct SREGS szegreg;                          /* Szegmensreg. struktúra dekl.   */

segread(&szegreg);                             /* Szegmensregiszterek beolvasása. */

regiszter.h.ah = 0x1A;                                 /* DTA cím funkció.  */
regiszter.x.dx = ofszet;                               /* Új DTA ofszetcím. */
intdosx(&regiszter, &regiszter, &szegreg);             /* 21(h) megszakítás. */
}

/*                                                                      */
/*  KAT : ÁLLOMÁNYONKÉNTI MEGJELENÍTÉSI                                 */
/*        CIKLUS                                                        */
/*                                                                      */

void KAT(utvonal, attr)
char *utvonal;                         /* A megjelenítendő állományok eléré- */
int attr;                              /* si útvonala (mutató) és attribútuma. */

{
int ntotal,                                    /* Összes bejegyzések száma.      */
    n;                                         /* Az ablak bejegyzéseinek száma. */
struct bejegyzes aktsor;                       /* Egy teljes bejegyzés kar. lánc. */

DTABEALL(&aktsor);                             /* DTA ofszetcím beállítása.      */
KERETEZES();

n = ntotal = 0;

if (VANELSO(utvonal, attr))
   {
   do
```

```
    {
    GORGETES(1, KM, 15, ESOR+4, 68, ESOR+3+NSOR);
    IR(15, ESOR+3+NSOR,
     "                |          |          |          |          ",KM);
    KIIR(&aktsor, ESOR+NSOR+3);
    if (++n == NSOR )                          /* Betelt az ablak ? */
     {
     n = 0;
     IR(14, ESOR+5+NSOR,
     "          Üssön le egy tetszőleges billentyűt ..          ",IM);
     getch();
     IR(14, ESOR+5+NSOR,
     "                                                          ",KM);
     }
    ++ntotal;
    }
while (VANMEG());
    }
KURPOZ(22, ESOR+5+NSOR);
printf("összesen %i állomány a(z al)katalógusban !",ntotal);
}

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

void main(npar, param)
int npar;                       /* A paraméterek egy "npar" elemű paramé- */
char *param[];                  /* ter vektorban kerülnek átadásra.       */
{
switch (npar)
    {
    case 1 : KAT("*.*", "0");           /* Nincs paraméter --> dir *.*    */
             break;
    case 2 : KAT(param[1], "0");        /* Egy paraméteres normál hívás.  */
             break;
    default : printf("Érvénytelen paraméterezés ..");
    }
getch();                                /* Várakozás egy billentyűre.     */
}
```

```
{ ┌──────────────────────────────────────────────────────────┐ }
{ │                                                          │ }
{ │                      K A T A L O G . P A S               │ }
{ │                                                          │ }
{ │  Funkciója : a kiválasztott (al)katalógus állományainak  │ }
{ │              megjelenítése a képernyőn.                  │ }
{ │                                                          │ }
{ └──────────────────────────────────────────────────────────┘ }

program KATALOG;

const NSOR = 18;                                  { Sorok max. száma 18. }

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

     bejegyzes = record
                  dummy   : array [1..21] of char;
                  attr    : byte;
                  idopont : integer;
                  datum   : integer;
                  meretl  : integer;
                  mereth  : integer;
                  allnev  : array [1..13] of char
                  end;

     utvonal    = string[65];

var aktsor  : bejegyzes;          { Egy katalógus sort tartalmaz.       }
    ut      : utvonal;            { A vizsgálandó állományok maszkja.   }

{ ┌──────────────────────────────────────────────────────────┐ }
{ │ VANELSO : ELSŐ BEJEGYZÉS MEGKERESÉSE                    │ }
{ │    KI : "true" ha van, "false" ha nincs                 │ }
{ │            bejegyzés.                                   │ }
{ │ A bejegyzés az "aktsor" rekordba kerül.                 │ }
{ └──────────────────────────────────────────────────────────┘ }

function VANELSO (allomany : utvonal;
                  attr     : integer) : boolean;

var regiszter : dosreg;                   { Regiszter rekord deklaráció. }

begin
   allomany     := allomany + #0;
   regiszter.ax := $4E shl 8;                  { Első bejegyzés funkció. }
   regiszter.cx := attr;                       { Állományok atribútuma.  }
   regiszter.ds := seg(allomany);              { Állomány szegmenscím.   }
   regiszter.dx := succ(ofs(allomany));        { Állomány ofszetcím.     }
   msdos(regiszter);                           { 21(h) megszakítás       }
   if (regiszter.flags and 1) = 0              { Átvitel bit vizsgálata. }
      then VANELSO := true                     {    ha   0 : talált,     }
      else VANELSO := false;                   {    ha nem : nem talált. }
end;

{ ┌──────────────────────────────────────────────────────────┐ }
{ │ VANMEG : KÖVETKEZŐ BEJEGYZÉS MEGKERESÉSE                │ }
{ │    KI : "true" ha van, "false" ha nincs                 │ }
{ │            több bejegyzés.                              │ }
{ └──────────────────────────────────────────────────────────┘ }

function VANMEG : boolean;
```

```
var regiszter : dosreg;                    { Regiszter rekord deklaráció. }

begin
   regiszter.ax := $4F shl 8;              { Következő bejegyzés funkció. }
   msdos(regiszter);                       { 21(h) megszakítás.           }
   if (regiszter.flags and 1) = 0          { Átvitel bit vizsgálata.      }
      then VANMEG := true                  {    ha   0 : talált,          }
      else VANMEG := false;                {    ha nem : nem talált.      }
end;

{ ┌──────────────────────────────────────────────┐ }
{ │ KIIR : EGY BEJEGYZÉS KIÍRÁSA A KÉPERNYŐN    │ }
{ │    BE : A bejegyzést tartalmazó              │ }
{ │         "aktsor" rekord.                     │ }
{ │    KI : -                                    │ }
{ └──────────────────────────────────────────────┘ }

procedure KIIR;

var i         : integer;
    hossz1,                                { A méret megállapításához.  }
    hossz2    : real;                      { felhasznált segédváltozók. }

begin

   writeln;
   gotoxy(1, NSOR);
   i := 1;                                 { Az állománynév megjelení-  }
   while (aktsor.allnev[i]<>#0) do         { tése karakterenként.       }
   begin
      write(aktsor.allnev[i]);                          { NÉV kiírás.   }
      i := succ(i)
   end;

   gotoxy(17, NSOR);

   hossz1 := aktsor.mereth;                             { Az állomány   }
   if hossz1 < 0 then hossz1 := hossz1 + 65536.0 ;      { méretének     }
   hossz2 := aktsor.meretl;                             { kiszámítása.  }
   if hossz2 < 0 then hossz2 := hossz2 + 65536.0 ;
   write('|', hossz1 * 65536.0 + hossz2:6:0);           { MÉRET kiírás. }

   gotoxy(25, NSOR);

   write('|',aktsor.datum shr 9 + 1980:4);              { DÁTUM kiírás. }
   case (aktsor.datum shr 5 and 15) of
      1 : write (' JAN');
      2 : write (' FEB');
      3 : write (' MAR');
      4 : write (' APR');
      5 : write (' MAJ');
      6 : write (' JUN');
      7 : write (' JUL');
      8 : write (' AUG');
      9 : write (' SZE');
     10 : write (' OKT');
     11 : write (' NOV');
     12 : write (' DEC')
   end;
   write(aktsor.datum and 31:3,'.');

   gotoxy(38, NSOR);
```

```
   write('|', aktsor.idopont shr 11:3, ' :');        { IDŐPONT kiírás. }
   write(aktsor.idopont shr 5 and 63:3);

   gotoxy(48, NSOR);                                 { Állomány attribútumok }
                                                     { egyenkénti kiírása.   }
   write('|');
   if (aktsor.attr and  1)<>0 then write('X')
                              else write(' ');
   if (aktsor.attr and  2)<>0 then write('X')
                              else write(' ');
   if (aktsor.attr and  4)<>0 then write('X')
                              else write(' ');
   if (aktsor.attr and  8)<>0 then write('X')
                              else write(' ');
   if (aktsor.attr and 16)<>0 then write('X')          { Katalógus ? }
                              else write(' ');
   write('|');

end;

{ ┌──────────────────────────────────────────────┐ }
{ │ DTABEALL : DTA CÍMÉNEK BEÁLLÍTÁSA            │ }
{ │    BE : A beállítani kívánt szegmens- és     │ }
{ │         ofszetcím.                           │ }
{ └──────────────────────────────────────────────┘ }

procedure DTABEALL(szegmens,                 { Beállítandó DTA szegmenscím. }
                   ofszet : integer);        { Beállítandó DTA ofszetcím.   }

var regiszter : dosreg;                      { Regiszter rekord deklaráció. }

begin
   regiszter.ax := $1A shl 8;                        { DTA cím funkció.     }
   regiszter.ds := szegmens;                         { Új DTA szegmenscím.  }
   regiszter.dx := ofszet;                           { Új DTA ofszetcím.    }
   msdos(regiszter);                                 { 21(h) megszakítás.   }
end;

{ ┌──────────────────────────────────────────────┐ }
{ │ KERETEZES : LISTA KERETÉNEK KIRAJZOLÁSA      │ }
{ └──────────────────────────────────────────────┘ }

procedure KERETEZES;

var i : integer;

begin
   clrscr;
   window(13,(18-NSOR) shr 1+1,68,(18-NSOR) shr 1 +6+NSOR);
   gotoxy(1,1);

   write('Katalógus tartalomjegyzék : ');
   writeln(ut);
   write('┌──────────────┬──────┬──────────┬──────────┬─────┐');
   write('│ Állománynév  │ Méret│  Dátum   │ Időpont  │RHSVD│');
   write('├──────────────┼──────┼──────────┼──────────┼─────┤');
   for i := 1 to NSOR do
   write('│              │      │          │          │     │');
   write('└──────────────┴──────┴──────────┴──────────┴─────┘');

   window(15,(18-NSOR) shr 1+5,69,(18-NSOR) shr 1 +4+NSOR);
   gotoxy(1,1);
end;
```

```
{
{  FŐPROGRAM : MEGJELENÍTŐ CIKLUS                                  }
{                                                                  }

var ntotal,                          { összesen megjelenített bejegyzés.     }
    n       : integer;               { Az ablakban megjelenített bejegyzés.  }
    z       : char;                  { Billentyűbeolvasás segédváltozója.    }

begin

   DTABEALL(Seg(aktsor), Ofs(aktsor));                  { DTA beállítása.  }
   clrscr;
   writeln('Adja meg a kívánt katalógushoz  tartozó  útvonalat !  A');
   writeln('megadás során a DOS-ból ismert általános célu helyette-');
   writeln('sítő  karakterek is használhatók (* és ?).  Ha nem  ad');
   writeln('meg útvonalat, az aktuális katalógus tartalmát jeleníti');
   writeln('meg a program.');
   writeln;
   writeln;
   write('Útvonal : ');
   readln(ut);                                  { Az útvonal beolvasása.   }
   if ut = '' then ut := '*.*';                 { Ha nincs megadva, akkor  }
                                                { az alapértelmezés *.* .  }
   KERETEZES;
   n       := -1;                               { Számlálók "nullázása" .  }
   ntotal  :=  0;
   if VANELSO(ut, 255) then                     { Az attribútum értéke     }
   repeat                                       { példánkban tetszőleges.  }
      ntotal := succ(ntotal);
      n := succ(n);                             { Számlálók növelése.      }
      if n = NSOR then
      begin
         window(13,(18-NSOR) shr 1+6+NSOR,69,(18-NSOR) shr 1+7+NSOR);

         gotoxy(1, 1);                  { Betelt ablak esetén              }
         textbackground(7);             { inverz üzenet megjelenítés.      }
         textcolor(0);
         write('          Üssön le egy tetszőleges' +
               ' billentyűt ..          ');
         repeat until keypressed;
         read(kbd, z);                          { Várakozás egy billentyű  }
         gotoxy(1, 1);                          { leütésére, majd az üze-  }
         textbackground(0);                     { netsor törlése.          }
         textcolor(15);
         write('                                    ' +
               '                                    ');
         window(15,(18-NSOR) shr 1+5,69,(18-NSOR) shr 1 +4+NSOR);
         gotoxy(1, NSOR);
         n := 0;                            { Képernyő számláló törlése.  }
      end;
      KIIR;                                 { Bejegyzés megjelenítése.    }
   until not(VANMEG);
   window(13,(18-NSOR) shr 1 +6+NSOR,69,(18-NSOR) shr 1 +7+NSOR);
   gotoxy(1, 1);
   textbackground(7);                                   { Inverz alapon az }
   textcolor(0);                                        { összegző üzenet  }
   gotoxy(1, 1);                                        { megjelenítése.   }
   write('       összesen');
   write(ntotal:4);
   write(' állomány a(z al)katalógusban !       ');
   gotoxy(1, 1);
   window(1, 1, 80, 25);       { Eredeti képernyőméret visszaállítása.  }

end.
```

6.8. Az EXEC-funkció

A könyv eddigi részében, főként a parancsértelmező processzorral kapcsolatban már többször szóba került az EXEC-funkció. Ebben az alfejezetben most ezzel foglalkozunk. Részletesen megvizsgáljuk működését és a meghívását.

Az EXEC-funkció is egyike azoknak a DOS-funkcióknak, amelyeket a 21(h) megszakításon keresztül lehet meghívni (a száma 4B(h)). Egészen általánosan fogalmazva egy szülőnek (parent) nevezett program a funkció segítségével meghív egy gyermeknek (child) nevezett programot. A rendszer a meghívott programot egy adathordozóról (rendszerint hajlékony- vagy merevlemezről) betölti a számítógép operatív tárába és végrehajtja. Végrehajtását követően – hacsak nem rezidens, azaz tárban maradó – a program törlődik a tárból, és az általa lefoglalt terület felszabadul. A gyermekprogram futása során ezen a funkción keresztül újabb programot hívhat meg, és ebben a kapcsolatban most ő lesz az újonnan meghívott program számára a szülő. Ily módon programok egész láncolata jöhet létre, amelyek együttes méretét csak a rendelkezésre álló tár területe szabja meg.

A DOS parancsértelmező processzora, a COMMAND.COM is az EXEC-funkciót használja arra, hogy a felhasználó által kiadott DOS parancsokhoz tartozó programokat meghívja és végrehajtsa. Számos ismert adatbázis-kezelő vagy szövegszerkesztő program is használja ezt a funkciót, lehetővé téve, hogy a felhasználó menet közben – anélkül, hogy a programból kilépne – DOS parancsokat hajtson végre (pl. kilistázza a lemez katalógusát vagy formáljon egy lemezt).

A szülőprogram különböző információkat adhat át a gyermekprogramnak. A paraméterek a parancssorban, az ún. környezeti blokkban (environment block) vagy a PSP-ben (program szegmens előtag) adhatók át. A paraméterátadást az teszi lehetővé, hogy – mint minden más végrehajtható program előtt – a gyermekprogram előtt is van a tárban egy PSP, amelybe a szülőprogram beírhatja üzeneteit. A paraméterek átadásának e lehetőségeire a későbbiekben még visszatérünk.

Miután a vezérlést átvette a gyermekprogram, mindazon állományokhoz és egységekhez hozzáférhet, amelyeket a szülőprogram, ill. a szülőprogramok valamelyike handle-funkcióval korábban már megnyitott. Így például a gyermekprogram olyan állományból is olvashat, ill. olyan állományba is írhat, amelynek a nevét nem is ismeri, hanem csak a hozzárendelt handle-t. Ez persze csak akkor lehetséges, ha a szülőprogram az illető handle számát az előbb említett valamelyik módon előzőleg már átadta (a DOS által a standard egységeknek előre kiosztott őt, állandóan nyitva lévő handle természetesen minden program számára hozzáférhető). Ha a gyermekprogram az ilyen, más program által megnyitott állományon valamilyen műveletet végez, akkor természetesen ugyanúgy változik a rekordmutató, mintha a műveletet a szülőprogram végezte volna. Amikor a gyermekprogram befejezi a futását, és a vezérlés visszakerül az őt meghívó szülőprogramhoz, akkor a rekordmutató nem áll vissza a korábbi értékére, vagyis a szülőprogram „látja", hogy a gyermekprogram „belenyúlt-e" valamelyik átadott állományba.

Amikor a gyermekprogram visszaadja a vezérlést a szülőprogramnak, egyúttal egy numerikus értéket is átadhat, amivel tájékoztatást adhat a befejeződés körülményeiről. A

4C(h) funkció meghívásával fejezhető be egy program úgy, hogy az a befejeződésekor egy kódot is átadjon a meghívó szülőprogramnak.

A szülő- és gyermekprogram közötti kommunikációnak természetesen csak akkor van értelme, ha az átadott kódot mindkét program azonosan értelmezi. A szülőprogram – miután a vezérlést visszakapta – az átadott kódot a 21(h) megszakítás 4D(h) funkciójával kérdezheti le. A funkció meghívásakor csak a funkció számát kell az AH regiszterbe írni. A funkció a visszatérésekor az AL regiszterben szolgáltatja a gyermekprogram által átadott kódot (a program ún. visszatérési kódját).

Az AH regiszterbe kerülő kód azt mutatja meg, hogy milyen körülmények között fejeződött be a gyermekprogram végrehajtása. A 0 érték a normál befejeződést jelenti, az 1 azt, hogy a gyermekprogram végrehajtása a Ctrl-C vagy a Ctrl-Break billentyű lenyomása miatt félbeszakadt. Ha a program futása közben valamelyik háttértárra irányuló hozzáféréskor olyan hiba lépett fel, ami a program végrehajtását megszakította, akkor az AH regiszterbe a 2-es kód kerül. Végül a 3-as azt jelenti, hogy a gyermekprogram a 21(h) megszakítás 31(h) funkciójának vagy a 27(h) megszakítás meghívásával fejeződött be, és a tárban rezidens marad.

Amint már említettük, az EXEC-funkció csak akkor tudja a gyermekprogramot betölteni, ha ehhez elegendő RAM-tár áll a rendelkezésére. A DOS az EXE programok tárigényét a fejlécben tárolt információk alapján viszonylag jól fel tudja mérni. Más a helyzet a COM programoknál. Mivel ezek magukra a programokra vonatkozóan semmiféle információt sem tartalmaznak, a DOS a legrosszabb esetet, vagyis a lehetséges leghosszabb programot (64 kbájt) és a lehetséges legkisebb tárméretet (ugyancsak 64 kbájt) tételezi fel, azaz a rendelkezésre álló teljes RAM-tárat lefoglalja. Ekkor viszont nincs lehetőség arra, hogy a COM program az EXEC-funkcióval egy másik programot betöltsön, hiszen a tárban már nincs üres hely. Ugyanez vonatkozik – még ha más okok miatt is – számos EXE programra is. Ahhoz tehát, hogy egy program újabb programot be tudjon tölteni, a meghívó programnak az EXEC-funkció használata előtt az általa fölöslegesen lefoglalt tárterületet fel kell szabadítania. Az ehhez szükséges lépéseket a 2.4.2. és a 2.4.3 pontban már leírtuk.

Az EXEC-funkció használatához a 21(h) megszakítás meghívása előtt a funkció számát (4B(h)) most is az AH regiszterbe kell betölteni. Az AL regiszterbe a meghívás céljától függően 0-t vagy 3-at kell írni. A 0 azt jelenti, hogy a funkciónak az illető programot be kell töltenie, és azonnal végre is kell hajtania. Ha viszont az AL-ben 3 van, akkor az EXEC-funkció feladata csak az, hogy a megadott programot a szülőprogram ún. overlay-programjaként ez utóbbi által megadott helyre betöltse (erre később még visszatérünk). A DS:DX és az ES:BX regiszterpárokban két paraméterterület címét kell átadni. Ezek közül az egyik a végrehajtandó, ill. betöltendő program nevét tartalmazza, amelynek ASCII-karakterláncként kell a tárban lennie, és amelyet egy végjelnek (0 ASCII-kód) kell lezárnia. A másik paraméterterület az ún. paraméterblokk, amely a DOS által meghatározott felépítésben egy sor információt tartalmaz.

A programnév a meghajtóazonosítón kívül teljes útvonal-kijelölést tartalmazhat. Ennek utolsó elemként a meghívandó állomány nevét és – kötelezően – a kiterjesztését kell megadni. A megadásnak egyértelműnek kell lennie, tehát a programnév, ill. kiterjesztés jokerkaraktereket nem tartalmazhat! Az állomány csak COM vagy EXE kiter-

jesztésű lehet. Ha nem adunk meg meghajtóazonosítót vagy útvonal-kijelölést, akkor a rendszer az állományt az aktuális egység aktuális katalógusában keresi. Mivel az EXEC-funkció batch-állományt közvetlenül nem tud végrehajtani, BAT kiterjesztésű állomány nem adható meg.

Ha mégis egy batch-állományt akarunk végrehajtani, akkor programnévként először a parancsprocesszor nevét, tehát a COMMAND.COM nevet kell megadni. Ezzel elérjük, hogy az EXEC-funkció másolatot készít a tárban az eredeti COMMAND.COM állományról. Ha a parancssorba a COMMAND.COM mögé beírjuk még a /c paramétert (a command első betűjeként), akkor ezzel közölhetjük a parancsértelmező processzorral, hogy a parancssorban az e paraméter után álló parancsot hajtsa végre. Ez a parancs most már lehet BAT állomány neve, de bármely más, belső DOS parancs is (mint pl. a DIR, COPY stb.). A parancssorban a programnév után álló paramétereket nevezzük parancsparamétereknek (a parancssor ezen részének angol neve command tail).

Ha a parancsprocesszorról íly módon másolatot készítünk a tárban, akkor arra is van lehetőség, hogy egy program egy másik programot úgy hívjon meg, hogy nem adja meg a meghívandó program kiterjesztését. Ilyenkor ugyanis a parancsprocesszor a megadott névvel először egy COM, majd EXE, végül pedig BAT kiterjesztésű állományt keres. Ha a megadott (vagy az aktuális) katalógusban nem talál ilyet, akkor a keresést mindazon katalógusokban folytatja, amelyeket a PATH parancsban kijelöltünk. Egy program közvetlen – a parancsprocesszor megkerülésével történő – meghívása ezt a keresést nem teszi lehetővé.

Ahhoz persze, hogy az előbbi kereséshez vagy a belső DOS parancsok végrehajtásához a parancsprocesszort igénybe vehessük, azt is tudnunk kell, hogy éppen melyik meghajtóban, ill. katalógusban található. Ha a számítógéphez merevlemezes meghajtó is csatlakozik, akkor abból lehet kiindulni, hogy a COMMAND.COM ennek a meghajtónak a főkatalógusában van. A pontos helye a környezeti blokkba is be van írva egy karakterláncként a következőképpen:

COMSPEC = meghajtó azonosító\útvonal-kijelölés\COMMAND.COM

Természetesen nemcsak a parancsprocesszornak (ami az EXEC számára ugyanolyan program, mint bármely más „közönséges" program) adhatók át a programnév mögött paraméterek. A parancsparamétereket a rendszer pontosan úgy értelmezi, mintha azokat valamely program meghívásakor a billentyűzetről írtuk volna be.

Mielőtt megvizsgálnánk a paraméterátadás módját, előbb vegyük szemügyre a paraméterblokk felépítését (amikor az AL regiszterben 0 érték van). A paraméterblokk címét az ES:BX regiszterpárban adjuk át az EXEC-funkciónak. A paraméterblokk felépítése a következő:

Mező	Bájt	Tartalom
1.	0 – 1	Környezeti blokk szegmenscíme
2.	2 – 3	Parancsparaméterek ofszetcíme
3.	4 – 5	Parancsparaméterek szegmenscíme
4.	6 – 7	Első FCB ofszetcíme
5.	8 – 9	Első FCB szegmenscíme
6.	10 – 11	Második FCB ofszetcíme
7.	12 – 13	Második FCB szegmenscíme

Az első mező a gyermekprogram környezeti blokkjának szegmenscímét adja meg. A blokk ofszetcímét nem kell megadni, mert ez mindig 0. A környezeti blokk mindig paragrafushatáron (16-tal osztható tárcímen) kezdődő, ASCII-karakterláncokat tartalmazó tárterület. A környezeti blokkban lévő ASCII-karakterláncok a parancsprocesszor és más programok számára fontos információkat tartalmaznak, így például azt az útvonalkijelölést (PATH), amely a keresett állományokhoz vezet. A karakterláncok általános alakja:

NÉV = paraméter

A karakterláncot a végjel (0 ASCII-kód) zárja le. Az egyes karakterláncok közvetlenül egymás után következnek úgy, hogy egy végjel után rögtön jön a következő karakterlánc első karaktere. A környezeti blokk maximum 32 kbájt hosszú lehet, ami igen sok információ átadását teszi lehetővé. A blokk végét az utolsó karakterláncot lezáró végjel utáni végjel mutatja (vagyis két, egymást követő 00 bájt).

Felhasználói szintről a környezeti blokk a SET és a PATH DOS parancsokkal változtatható meg. Azokra a programokra, amelyek betöltődésük és végrehajtásuk után a tárban maradnak (rezidensekké válnak), a környezeti blokknak e két DOS paranccsal történő megváltoztatása nincs hatással.

Ha a szülőprogram e környezeti blokkon keresztül információkat akar a gyermekprogramnak átadni, akkor erre a célra létrehozhat egy új környezeti blokkot, vagy az átadni kívánt információkkal kiegészítheti a saját környezeti blokkját. Az első megoldásnál az új környezeti blokk szegmenscímét kell a paraméterblokk első mezejébe beírni. Ha viszont a gyermekprogramnak a szülőprogram környezeti blokkját kell megkapnia, akkor ebbe a mezőbe 0-t kell írni. Ebben az esetben az EXEC-funkció – még mielőtt a vezérlést átadná a gyermekprogramnak – a gyermekprogram előtt álló PSP 2C(h) ofszetcímére beírja a környezeti blokk szegmenscímét. Ilyenkor azt mondjuk, hogy a gyermekprogram „örökölte" a szülőprogram környezeti blokkját.

Ha a gyermekprogram részére új környezeti blokkot kell átadni, akkor ennek legkevesebb három karakterláncot kell tartalmaznia, amelyeket (általában) a szülőprogram környezeti blokkja is tartalmaz, és amelyek a parancsprocesszor számára alapvető fontosságúak. Ezek:

COMSPEC = paraméter
PATH = paraméter
PROMPT = paraméter

A DOS ezeket a karakterláncokat a rendszer inicializálásakor helyezi el a környezeti blokkban. Paraméterként vagy az alapértelmezés szerinti paramétereket használja, vagy azokat, amelyeket a CONFIG.SYS állományban és az – esetleges – AUTOEXEC.BAT állományban talál. A fenti három karakterlánc közül az első arról tájékoztatja a COMMAND.COM rezidens részét, hogy hol találhatók a tranziens részei (hogy szükség esetén újra betölthesse ezeket), a második azt az útvonalat tartalmazza, amely a külső parancsokig, ill. más végrehajtható állományokig vezet, míg a harmadik a rendszer prompt ját határozza meg.

Természetesen a gyermekprogram is megváltoztathatja a saját környezeti blokkját, azonban ezek a változások a vezérlés visszaadásakor a szülőprogram környezeti blokkjába nem kerülnek vissza.

A paraméterblokk 2. és 3. mezeje azoknak a parancsparamétereknek a címét adja meg, amelyek a végrehajtandó program PSP-jébe a 80(h) ofszetcímtől kezdve másolódnak át. Ezeknek a tárban pontosan olyan szerkezetűnek kell lenniük, amilyen szerkezetet a DOS a PSP-ben is vár. Az első bájt az ún. számlálóbájt, amelynek értéke a parancsparaméterek karaktereinek száma mínusz 1. Ezt követik a paraméterek ASCII-karakterek sorozataként. A parancsparamétereket egy kocsi vissza (CR; 13-as ASCII-kód) karakter zárja le (ez nem számít bele a karakterek számába). A gyermekprogram ezeket a parancsparamétereket úgyanúgy értelmezi, mintha a felhasználó ezeket a program meghívásakor a DOS promptjára a DOS parancssorába írta volna be.

Ahhoz, hogy egy batch-programot (amelynek legyen a neve pl. FUSS.BAT) a parancsprocesszor segítségével végre tudjunk hajtani, a parancsparamétereknek olyan formában kell a tárban lenniük, amelyet a következő assembly definíció hoz létre:

```
db      0Bh,"/c FUSS.BAT",0Dh
```

(Vegyük észre a /c paraméter és az állománynév közötti szóközt: mivel a parancssorban is be kellene írnunk ide egy szóközt, ez a tárbeli képből sem hiányozhat!).

Az EXEC-funkció a parancsparamétereket a PSP-be való bemásoláskor ellenőrzi, és mindazokat a paramétereket kiszűri, amelyek következménye a bevitel vagy a kivitel átirányítása lenne. Az EXEC az alapértelmezés szerinti bevitel, ill. kivitel parancssorból történő átirányítását figyelmen kívül hagyja. Átirányítást csak a szülőprogram, és csak az EXEC meghívása előtt végezhet.

A 4 – 7. mező a szülőprogram PSP-jében lévő két FCB-re vonatkozik. Ha az információ átadására nem ezt a két FCB-t kívánjuk használni, akkor ezekbe a mezőkbe mindig a –1 értéket (FFFF(h)) kell írni. Ha a gyermekprogram szemszögéből emulálni akarjuk a COMMAND.COM parancsértelmező funkcióját (pl. a billentyűzetről való beolvasást), akkor a szülőprogramnak a 21(h) megszakítás 29(h) funkciójával a parancssor első két paraméterét az EXEC meghívása előtt be kell olvasnia a két FCB-be. Mielőtt az

EXEC-funkció a vezérlést átadja a gyermekprogramnak, ezt a két FCB-t átmásolja a gyermekprogram PSP-jének 5C(h) és 6C(h) ofszetcímeire.

Most már addig eljutottunk, hogy valamennyi regiszterbe és a paraméterblokkba is betöltődtek a szükséges értékek, de az EXEC-funkció még mindig nem hívható meg. Meghívásakor ugyanis a CS és az IP regisztereket kivéve valamennyi regiszter tartalma megváltozik, ezért a regisztereknek a funkció meghívása előtti tartalmát a verembe kell menteni, majd az SS és az SP regiszterek tartalmát be kell írni a kódszegmensen belül elérhető változókba. Csak ez után lehet az EXEC-funkciót a 21(h) megszakításon keresztül meghívni. Az EXEC-funkció a visszatérését követően a megszokott módon, az átvitelbittel jelzi, hogy végrehajtása sikeres volt vagy nem. Mielőtt azonban a program végrehajtását folytatnánk, az SS és SP regiszterekbe vissza kell tölteni eredeti értéküket, amelyeket a funkció meghívása előtt a kódszegmensben lévő változókba írtunk. Ezután már a többi regiszter tartalma is elővehető a veremből.

Némileg más a feladata az EXEC-funkciónak akkor, ha a meghívásakor az AL regiszterben 3-at adunk át. Ilyenkor ugyanis csak az a dolga, hogy egy COM vagy egy EXE programot betöltsön a tárba anélkül, hogy azt végrehajtaná. A program betöltése után ezért a vezérlés azonnal visszakerül a meghívó programhoz, amelyik a most betöltött programot bármely, általa meghatározott időpontban végrehajthatja.

A 0. alfunkcióval szemben a meghívott program most a tár nem valamely tetszőleges címére, hanem a meghívó program által előre meghatározott címre töltődik be. Mivel most a meghívó program paramétereket nem ad át a meghívott programnak, a paraméterblokk szerkezete is más.

Mező	Bájt	Tartalom
1.	0 – 1	Szegmenscím, ahová az overlay program betöltődik
2.	2 – 3	Áthelyezési (relokációs) tényező

A funkció meghívása előtt a paraméterblokk első mezejébe azt a szegmenscímet kell írni, amelyre a mindenkori overlay programot be kell tölteni. Ha a meghívó program a meghívandó program számára nem biztosított elegendő helyet a tárban, akkor a tárkezelésre vonatkozó DOS-funkciók valamelyikével előbb tárhelyet kell foglalnia, majd ezt a meghívott program rendelkezésére kell bocsátania. A meghívott program elé most nem kerül PSP, úgyhogy ez közvetlenül a megadott szegmenscímtől kezdődően töltődik be, azaz az első utasítás a 0. ofszetcímen áll.

Az áthelyezési (relokációs) tényező a meghívott program szegmenscímeinek illesztésére szolgál. Mivel ennek csak EXE programnál van jelentősége, az áthelyezési tényező értéke COM programoknál mindig 0. Ezzel szemben az EXE programoknál azt a szegmenscímet kell megadni, amelyre a programnak be kell töltődnie (ill. a táron belül áthelyeződnie) ahhoz, hogy az EXE programon belüli szegmenskijelölések a betöltött program szegmenscímeihez illeszkedhessenek.

Miután a program betöltődött, a meghívó program előbb-utóbb végre is fogja hajtani (hiszen ezért töltötte be). A meghívott programot minden esetben a meghívó program alprogramjaként kell tekinteni, ezért meghívásuk a CALL (gépi kódú) utasítással történik. Az utasításnak mindig FAR CALL hívásnak kell lennie, mert ugyan lehetséges, hogy a betöltött program a tárban közvetlenül a meghívó program után áll, de a szegmenscímük sohasem egyezik meg. Egy COM programnál ennél a CALL utasításnál az ofszetcím mindig 100(h), mert a COM programok végrehajtása definíció szerint mindig közvetlenül a PSP után, a 100(h) címen kezdődik. Ez viszont problémát okoz, mert a 3-as alfunkció a programot PSP nélkül tölti be. Emiatt a COM program kódrésze nem a 100(h), hanem a 0(h) ofszetcímen kezdődik. Mivel azonban a COM programon belüli összes ugróutasítás és adathozzáférés nem a 0(h), hanem a 100(h)-s címre vonatkozik, egy FAR CALL utasítás végrehajtásához az sem megfelelő megoldás, ha szegmenscímként a betöltési szegmens címét, ofszetcímként pedig 0(h)-t adunk meg. Ehelyett a FAR CALL-hoz a szegmenscímet kell 100(h)-val „visszafelé" állítani, vagyis – figyelembe véve a szegmenscímképzésből ismert egy hexa helyértékkel történő eltolást – szegmenscímként a betöltési szegmenscím mínusz 10(h) címet, ofszetcímként pedig a 100(h) címet kell megadni.

Ha a COM program rendeltetése kizárólag az, hogy egy főprogram számára overlay programként szolgáljon, akkor természetesen a 100(h) címtől eltérő indulási címek is előfordulhatnak. Ennek ellenére a meghívásakor ebben az esetben is csak az ofszetcím változik. A szegmenscímnek most is 10(h)-val kisebbnek kell lennie a betöltési szegmenscímnél.

Az EXE programoknál valamelyest más a probléma. Ha ezek úgy töltődnek be (a 0 alfunkción keresztül), hogy rögtön végre is hajtódnak, akkor az EXEC-funkció a kódszegmenst és az utasításmutatót arra az utasításra állítja be, amely az assembler forrásprogramban első utasításként van deklarálva. Ezt a címet azonban a meghívó program, amely az EXE programot overlay programként betölti, nem ismeri. Ezen úgy segíthetünk, hogy az EXE programban lévő, első végrehajtandó utasítást az EXE program elejére írjuk, tehát az ofszetcíme 0(h) lesz. Ilyenkor a forrásprogramban elsőként a kódszegmenst kell definiálni, hogy ez garantáltan az EXE program elején álljon. Ekkor a FAR CALL utasítás szegmenscímeként a betöltési szegmens címét, ofszetcímként pedig a 0(h) címet használja.

6.8.1. Programpélda

Szemben a magasszintű programozási nyelvekkel az assembly nem teszi lehetővé programok egymásból történő parancsszintű meghívását, ezekben a programokban a 21(h) megszakításnak az éppen az előző fejezetben tárgyalt 4B(h) funkcióját közvetlenül kell alkalmazni. A példaprogram ezt a módszert szeretné kicsit közelebb hozni és megvilágítani.

A program a korábban ismertetett EXE típusú programvázra épül. A gyermekprogramot a példában az EXEPRG eljárás hívja meg, amelynek ehhez két paraméterre van szüksége. Az első paraméter példánkban „c:\command.com". A parancsprocesszor

neve előtt azért szerepel útvonalkijelölés, hogy particionált merevlemez esetén bármely logikai lemezegységről elérhető legyen a parancsprocesszor. A második paraméter a hívandó program nevét és paraméterét tartalmazza, tartalma „/c echo üzenet". A programhívást végző EXEPRG eljárás számára szükséges változók a kódszegmensben helyezkednek el. Az EXEPRG felé az EXEC-funkció az átvitelbitben beállított értékkel jelzi a végrehajtás sikeres ill. sikertelen voltát. Sikertelen végrehajtás esetén az átvitelbit értéke 1, az AX regiszter a DOS által beállított hibakódot tartalmazza, ellenkező esetben az átvitelbit 0 és AX tartalma a hívott program által beállított érték.

A példában az „echo" a hívott program, mely lehetőséget nyújt egy rövid, a program céljára utaló üzenet megjelenítésére.

```
;
;   ┌──────────────────────────────────────────────────────┐
;   │                    E X E A S M . A S M               │
;   │                                                      │
;   │  Funkciója : demonstrálja a DOS EXEC eljárás meghívásával
;   │              a programfuttatást. A példában az "ECHO"
;   │              program fut le az EXEASM programból meghívva.
;   └──────────────────────────────────────────────────────┘
;
;— ADATOK ---------------------------------------------------------------

adatok   segment para 'DATA'

prognev  db "c:ommand.com",0    ; A hívandó program neve.

progpar  db "/c echo Ezt az echo-t a DOS exec eljárás futtatta !! ",0

                                ; A hívandó program paraméterei.

adatok   ends

;— KÓD -----------------------------------------------------------------

kod      segment para 'CODE'

         assume cs:kod, ds:adatok, ss:verem

                                ; Szegmensregiszterek hozzárendelése
                                ; a szegmensekhez.

futtat   proc far

         mov  ax,adatok         ; Adatszegmens cím -> DS.
         mov  ds,ax

         call freemem           ; Felesleges memória felszabadítása.

         mov  dx,offset prognev ; A programnév címe a memóriában.
         mov  si,offset progpar ; A paraméterek címe a memóriában.
         call exeprg            ; A program felhívása.

         mov  ax,4c00h          ; 21(h) megszakítás, a program
         int  21h               ; 0 hibakóddal ér véget.

futtat   endp

;— FREEMEM : A felesleges tárterület felszabadítása ---------------------
;— BE      : ES = a PSP címe
;— KI      : —
;
;   Mivel megállapodás szerint a veremszegmens az .EXE állomány utol-
;   só szegmense, és a DOS az állomány betöltésekor az ES regisztert
;   az állomány elé elhelyezett PSP elejére, SS:SP-t pedig az állomány
;   végére állítja, ezekből az adatokból az állomány hossza kiszámít-
;   ható.
;
;— REG     : AX, BX, CL és FLAGS értéke megváltozik.
```

```
freemem  proc near

         mov bx,ss                  ; A két szegmens egymásból való kivo-
         mov ax,es                  ; nása megadja azoknak a paragrafu-
         sub bx,ax                  ; soknak a számát, amelyet az állo-
                                    ; mány a PSP kezdőcímétől a veremtár
                                    ; kezdetéig lefoglal.
         mov ax,sp                  ; Mivel a veremmutató a szegmens vé-
         mov cl,4                   ; gére mutat, a tartalma megadja a
         shr ax,cl                  ; verem hosszát. Ezt az előbb ki-
         add bx,ax                  ; számított paragrafusokhoz hozzá
                                    ; kell adni.

         inc bx                     ; A biztonság kedvéért egy paragra-
                                    ; fussal nagyobb hosszt veszünk.

         mov ah,4ah                 ; Tárterületet változtató funkció
         int 21h                    ; és megszakítás hívása.

         ret                        ; Visszatérés a meghívó programhoz.

freemem  endp                       ; A freemem eljárás vége.

;— EXEPRG : program felhívása ————————————————————————————————
;— BE     : DS:DX = A programnév címe
;           DS:SI = A paraméterek címe
;— KI     : Átvitelbit = 1 -> hiba lépett fel (AX = hibakód)
;
;   A programnév és a paraméterek ASCII karakterlánc formájában, egy 0
;   karakterrel lezárva szerepeljenek.
;
;— REG    : AX és FLAGS értéke megváltozik.

exeprg   proc near
                                    ; A paramétersor átmásolása a saját
                                    ; pufferbe, a karakterek számlálása.

         push bx                    ; A regiszterek elmentése.
         push cx
         push dx
         push di
         push si
         push bp
         push ds
         push es

         mov  di,offset comline+1       ; A parancssor címe.
         push cs                        ; CS --> ES.
         pop  es
         xor  bl,bl                 ; Karakterszámláló nullázása.

copy_par: lodsb                     ; Egy karakter olvasása.
         or   al,al                 ; Végjelző (0) karakter ?
         je   copy_veg              ; Ha igen, vége a parancssornak.
         stosb                      ; Ha nem, átmásolás az új pufferbe.
         inc  bl                    ; Parancssor következő karaktere.
         cmp  bl,126                ; Maximum (126) elérve ?
         jne  copy_par              ; Ha nem, folytatni a másolást.

copy_veg: mov  cs:comline,bl            ; A karakterszám eltárolása.
         mov  byte ptr es:[di],13       ; A parancssor lezárása.
```

```
            mov  cs:savess,ss       ; SS és SP eltárolása.
            mov  cs:savesp,sp

            mov  bx,offset parblokk; ES:BX a paraméterblokkra mutat.
            mov  ax,4b00h           ; EXEC funkciókód.
            int  21h                ; 21(h) megszakítás.

            cli                     ; A megszakítások letiltása.
            mov  ss,cs:savess       ; A veremszegmens és -mutató feltöl-
            mov  sp,cs:savesp       ; tése az eredeti értékekkel.
            sti                     ; A megszakítások engedélyezése.
            pop  es                 ; A regiszterek visszatöltése.
            pop  ds
            pop  bp
            pop  si
            pop  di
            pop  dx
            pop  cx
            pop  bx

            jc   exe_veg            ; Ha hiba lépett fel, akkor VÉGE,
            mov  ah,4dh             ; ellenkező esetben a végrehajtott
            int  21h                ; program hibakódjának lekérdezése.

exe_veg:    ret
                                    ; Az alábbi rutin változói csak
                                    ; CS-n át érhetőek el.

savess      dw (?)                  ; SS eltárolása.
savesp      dw (?)                  ; SP eltárolása.

parblokk    equ this word           ; Az EXEC-funkció paraméterblokkja.
            dw 0                    ; Ugyanaz az környezet blokk.
            dw offset comline       ; A parancssor ofszet- és
            dw seg kod              ; szegmenscíme.
            dd 0                    ; Nincs adat a FCB #1-ben.
            dd 0                    ; Nincs adat a FCB #2-ben.

comline     db 128 dup (?)          ; A parancssor eltárolása.

exeprg      endp

;— VEREM ———————————————————————————————————————————————————————

verem       segment para stack      ; A veremszegmens definiálása.

            dw 256 dup (?)          ; A verem hossza 256 szó.

verem       ends                    ; A veremszegmens vége.

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

kod         ends
            end  futtat
```

6.9. A RAM-tár kezelése

A DOS jelenlegi verziói 1 Mbájt nagyságú, összefüggő RAM-tár kezelésére képesek. Az IBM PC XT/AT gépeken a DOS és az egyéb programok a 00000 címen kezdődnek, és a 9FFFF(h) címig terjedő területet foglalhatják el. Ezt az összesen 640 kbájt hosszúságú területet hagyományos tárterületnek is nevezik. Az e fölötti területen az ugyancsak írható és olvasható video-RAM, a ROM-BIOS rutinjai, a képernyőkezelő rutinok, és más, szintén csak olvasható rutinok helyezkednek el.

A DOS a hagyományos, 640 kbájt nagyságú RAM-tárat két fő részre osztja fel: rendszerterületre és programterületre (ez utóbbit TPA-nak is nevezik, az angol Transient Program Area kifejezés rövidítéseként). A rendszerterület a 00000 címen kezdődik, és a megszakítási vektortáblát, az operációs rendszer megfelelő részeit, ezek belső tábláit, puffereit és különböző változókat tartalmazza. Az operációs rendszer itt elhelyezkedő része tartalmazza a parancsprocesszor (COMMAND.COM) rezidens részét, valamint a rezidens és az installálható eszközmeghajtókat (device driver). A rendszerterület hossza változó: függ a DOS verziójától, az installált eszközmeghajtóktól és más, egyéb tényezőktől.

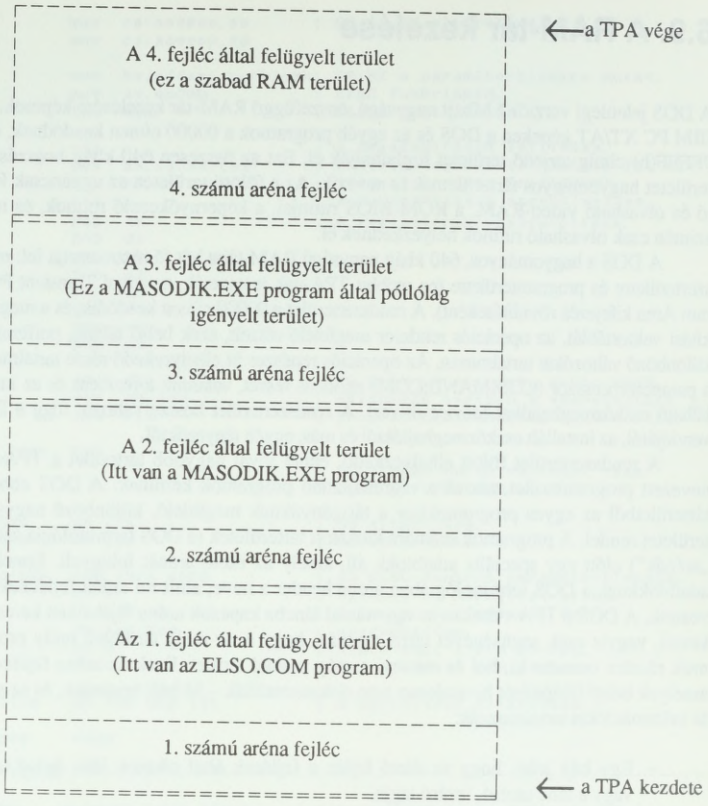
A rendszerterület fölött elhelyezkedő, ennél jóval nagyobb tárterület a TPA-nak nevezett programterület, ahová a végrehajtandó programok kerülnek. A DOS ebből a tárterületből az egyes programokhoz a tárigényüknek megfelelő, különböző nagyságú területet rendel. A programok számára kiosztott tárterületek (a DOS terminológia szerint „arénák") előtt egy speciális adatblokk áll, amely az illető arénát felügyeli. Ennek az adatblokknak a DOS terminológiában arena header a neve, amit mi aréna fejlécnek nevezünk. A DOS a TPA-t ezeken az egymással láncba kapcsolt aréna fejléceken keresztül kezeli, vagyis ezek segítségével tartja nyilván, hogy a tár mely területeit mely programok részére osztotta ki, hol és mennyi a még szabad terület. Ezek az aréna fejlécek – amelyek belső felépítését hivatalosan nem dokumentálták – 16 bájt hosszúak, és az alábbi információkat tartalmazzák:

- Egy bájt jelzi, hogy az illető fejléc a fejlécek által alkotott lánc belső tagja, vagy a lánc utolsó, lezáró tagja.

- Egy szó jelzi, hogy az illető fejléc által felügyelt aréna szabad, vagy egy programhoz hozzá van már rendelve (az utóbbi esetben a szó a program PSP-jére mutat).

- Egy szó adja meg paragrafusokban a fejléc által felügyelt aréna hosszát.

A fejléc kezdőcíme és az aréna hossza egy mutatót képez, amely a következő fejléc címére mutat. Ily módon az arénák egymással összekapcsolva, a teljes TPA-ra kiterjedő, folyamatos láncot alkotnak. Nézzük meg ezt egy példán keresztül (24. ábra):



24. ábra: A RAM–tár kezelése

Kiindulási helyzetként tételezzük fel, hogy a számítógépet éppen most kapcsoltuk be, és a RAM-tárban csak a rendszerterületen vannak adatok, a TPA teljesen üres. Most a DOS EXEC funkciójával betöltjük pl. az ELSO.COM programot. Ekkor a DOS a TPA legalsó címétől kezdődően inicializálja az 1. számú aréna fejlécet, amely után betölti az ELSO.COM nevű programot. Azt már a korábbiakból tudjuk, hogy a DOS egy COM program betöltésekor a rendelkezésre álló teljes tárterületet lefoglalja. Ahhoz tehát, hogy további programokat is be lehessen tölteni, az ELSO.COM programnak meg kell hívnia egy funkciót – a száma 4A(h) –, amely segítségével az általa nem használt tárterületet felszabadítja (az ezt végrehajtó rutint a COMVAZ.COM programpéldánk tartalmazza). Töltsük be most pl. a MASODIK.EXE nevű programot. A tárfoglalás ebben az esetben

másként történik. Az EXE állomány fejléce mindazon adatokat tartalmazza, amelyek a program "jólnevelt" betöltéséhez szükségesek, így a program tényleges hosszát és a végrehajtásához szükséges minimális és maximális tárigényt. Ugyan ez a maximális tárigény itt is FFFF(h) paragrafus (plusz természetesen magának a programnak a hossza), de – mivel ekkora terület nem áll rendelkezésre – a DOS beéri a minimális tárigénnyel, és ezt a program rendelkezésére bocsátja. A megfelelő adatok most beíródnak a MASODIK.EXE program előtt álló 2. számú aréna fejlécbe, és a program a tárba kerül (vele együtt még a környezeti blokkja is, de ezzel itt most nem foglalkozunk). Tegyük fel most, hogy a MASODIK.EXE program a végrehajtása során valamilyen közbenső adatok tárolására további tárigényt jelent be (ezt a 48(h) funkció meghívásával teheti meg). Ekkor a DOS e tárterület számára is elkészít egy (3. számú) aréna fejlécet, amely után a program rendelkezésére bocsátja a kért területet. A tárban az e terület után álló 4. számú fejléc most a lánc utolsó tagja, és – mivel egyelőre több terület nincs a tárban lefoglalva – a TPA teljes fennmaradó részét felügyeli, és mint szabad tárat tartja nyilván.

Az előzőekben két tárkezelő funkcióról már volt szó: a 48(h) és a 4A(h) funkciókról. A csoport harmadik tagja a 49(h) funkció. Nézzük meg most e funkciók használatát. Mindegyik funkció meghívásához a funkció számát az AH regiszterbe kell tölteni.

A 48(h) funkcióval tárterület foglalható le. A lefoglalandó paragrafusok számát a BX regiszterben kell átadni. Ha a tárban a kért hosszúságban van szabad terület, akkor a funkció a tárterületet lefoglalja, és a terület szegmenscímét az AX regiszterben szolgáltatja. Az átvitelbit értéke 0, jelezve, hogy a művelet sikeres volt. Ellenkező esetben az átvitelbit értéke 1, az AX regiszterbe hibakód, a BX regiszterbe pedig a még rendelkezésre álló tárterület nagysága kerül, ugyancsak paragrafusokban. Ez utóbbi esetben tárfoglalás még nem történt, úgyhogy a program a visszakapott adatok alapján eldöntheti, beéri-e a kisebb tárhellyel. Amennyiben ez is megfelel, akkor ismét meg kell hívnia a 48(h) funkciót, most már a lefoglalható paragrafusokkal a BX-ben, hogy a tárfoglalás ténylegesen megtörténjen.

Amikor a DOS tárkezelője egy terület iránti igény során számbaveszi a lehetséges szabad tárterületeket, akkor a következő stratégiák szerint járhat el.

Keresés az alacsonyabb címektől kezdődően (first fit)

A DOS a kért területnek megfelelő szabad területet a legalacsonyabb címeken lévő aréna fejlécekkel kezdi, és alulról felfelé haladva az első megfelelő méretű szabad területet a program rendelkezésére bocsátja.

Keresés a legjobban illeszkedő terület szerint (best fit)

A DOS a rendelkezésére álló szabad területek közül azt a területet bocsátja a program rendelkezésére, amelyik a lehető legkisebb mértékben haladja meg az igényelt területet. Ennél a keresésnél a kiutalandó területnek a teljes táron belüli elhelyezkedését figyelmen kívül hagyja.

Keresés a magasabb címektől kezdődően (last fit)

A DOS a kért területnek megfelelő szabad területet a legmagasabb címeken lévő aréna fejlécekkel kezdi, és felülről lefelé haladva az első megfelelő méretű szabad területet a program rendelkezésére bocsátja.

Ha a DOS által az előbbi stratégiák valamelyike szerint kiválasztott terület nagyobb az igényeltnél, akkor ezt a DOS két részre osztja: az igényelt méretű területre, amelyet a program számára lefoglal, és a fennmaradó részre, amelyet továbbra is szabad területként tart nyilván.

A DOS alapértelmezés szerint az elsőként említett stratégiát, vagyis az alacsonyabb címektől induló keresést használja. A 3.0 verziótól felfelé azonban a felhasználói program a 21(h) megszakítás 58(h) funkciójával saját maga is megválaszthatja a stratégiáját.

A 48(h) funkcióval ellentétes műveletet végez a 49(h) funkció: a 48(h) funkcióval lefoglalt tárterületet ismét felszabadítja. Az AH regiszterben a funkció számát kell megadni, az ES regiszterben pedig a felszabadítandó tárterületnek a szegmenscímét (amelyet az előzőleg meghívott, 48(h) funkció az AX regiszterbe írt). Mivel ezzel a funkcióval csak olyan területet lehet felszabadítani, amelyet előzőleg a 48(h) funkcióval foglaltunk le, a terület méretét a DOS ismeri, ezért a terület méretét felszabadításakor nem is kell megadni. A művelet sikeres végrehajtását itt is az átvitelbit 0 értéke jelzi. Hiba akkor fordulhat elő, ha valamelyik tárterülethez tartozó fejléc sérült, vagy az ES regiszterben átadott szegmenscím szabad területre mutat.

A csoport harmadik funkciójával egy már lefoglalt tárterület nagyságát változtathatjuk meg. A terület nagyságát növelni is, csökkenteni is lehet. A 4A(h) funkció meghívásához az ES regiszterben a módosítandó terület szegmenscímét, a BX regiszterben pedig az új terület nagyságát kell megadni paragrafusokban. A funkció meghívása után a regiszterek szereposztása megegyezik a 48(h) funkciónál leírtakkal.

6.10. Szűrők

Az operációs rendszerben a szűrők olyan programok, amelyek lényegében hasonló feladatot látnak el, mint a hétköznapi értelemben vett szűrők: az egyik oldalukon betöltünk valamit, amiből a másik oldalon valami más jön ki. Az operációs rendszer szűrőinek egyik oldalán karaktereket töltünk be, ezeken a szűrő különböző műveleteket végez, majd az így „megszűrt" karaktereket a szűrő másik oldalán visszakapjuk. A szűrők különböző műveleteket végezhetnek: a bevitt karaktert kiszűrhetik, lecserélhetik másik karakterre, de karakterláncok ábécé szerinti rendezését és karakterláncok keresését is lehetővé teszik. A DOS alapértelmezésben három szűrőt tartalmaz.

- SORT : karakterláncokat ábécé sorrendbe rendezi,
- FIND : szövegben adott karakterláncot keres,
- MORE : szöveget képernyőoldalanként jelenít meg.

A szűrőműveletek elvégzését a DOS 2.0 verziójában bevezetett két újdonság teszi lehetővé: az alapértelmezés szerinti egységek, valamint a bevitel/kivitel átirányíthatósága. Az alapértelmezés szerinti egységekhez öt, eleve kijelölt handle tartozik, amelyeket a rendszerben futó programok mindegyike azonos módon használhat. Ezek:

Handle	Alapértelmezés szerinti egység
0	Standard beviteli egység (CON)
1	Standard kiviteli egység (CON)
2	Standard hibajelző egység (CON)
3	Standard soros interfész (AUX)
4	Standard nyomtató (PRN)

A CON nevű egység alapértelmezés szerint a billentyűzetet és a képernyőt jelenti. Az AUX egység alapértelmezés szerint az első soros interfésznek (COM1), a PRN pedig az első párhuzamos nyomtatónak (LPT1) felel meg.

A bevitel/kivitel átirányíthatósága azt jelenti, hogy amikor a DOS-ból meghívunk egy programot, akkor a fenti, alapértelmezés szerinti egységeket a parancssorba írt speciális jelekkel megváltoztathatjuk. Így például megtehetjük, hogy a program által a billentyűzetről várt adatok egy lemezes állományból származzanak, vagy hogy a kivitel ne a képernyőre, hanem pl. a nyomtatóra vagy a soros interfészre kerüljön. Az átirányítást előíró jelek és hatásuk a következő:

Jel	Hatás
< állomány	Az adatok a billentyűzet helyett a megadott állományból származnak.
< egység	Az adatok a billentyűzet helyett a megadott egységről származnak.
> állomány	Az adatok a képernyő helyett a megadott állományba kerülnek.
>> állomány	Az adatok nem a képernyőre kerülnek, hanem a megadott állomány aktuális tartalmához hozzáíródnak (append).
> egység	Az adatok a képernyő helyett a megadott egységre kerülnek.
p1 : p2	A p1 program kimenetét a p2 program bemenetére irányítja (csatolás).

A fentiek megértésére vizsgáljuk meg a

C> SORT < BE.TXT > KI.TXT <RETURN>

parancssort, amely a SORT szűrőt és két átirányító jelet tartalmaz.

Ha a DOS promptjára beírtuk ezt a parancssort, és lenyomtuk a RETURN billentyűt, akkor a vezérlést átveszi a parancsprocesszor. Először azt állapítja meg, hogy meg kell hívnia a SORT nevű rendező programot. Ezután a < BE.TXT kifejezéssel találkozik, ami azt mondja neki, hogy az alapértelmezés szerinti bevitelt át kell irányítania egy állományra. Ezt úgy végzi el, hogy a billentyűzethez tartozó 0 handle-t most a BE.TXT nevű állományhoz kapcsolja. Ugyanez történik a „ > KI.TXT" hatására a kiviteli oldalon is azzal a különbséggel, hogy most nem a 0, hanem az 1-es handle-t irányítja át. A gyakorlatban ez úgy valósul meg, hogy az illető handle-t először lezárja, majd az állomány nevével együtt meghívja a handle-k megnyitására szolgáló funkciót. Mivel a DOS mindig a legkisebb, nem használt handle-t osztja ki, az állomány automatikusan az előbb lezárt handle-t kapja meg.

Ezzel a parancssor vizsgálata befejeződött, és a DOS az EXEC-funkció segítségével meghívja a SORT szűrőprogramot. Mivel az EXEC-funkcióval meghívott program a meghívó program valamennyi handle-jét örökli, a SORT program most a 0 handle-n keresztül az ehhez rendelt BE.TXT állományból olvas, és az 1-es handle-en keresztül a

KI.TXT állományba ír. A program szempontjából annak nincs jelentősége, hogy az adatok honnan érkeznek, ill. hova kerülnek, és ez számára láthatatlan is marad.

Miután a SORT program befejezte munkáját, visszaadja a vezérlést a parancsprocesszornak. Ez a bevitel és a kivitel átirányítását most érvényteleníti, vagyis visszaállítja az alapértelmezés szerinti állapotot.

Ezt a szűrési műveletet különösen az teszi hatékonnyá, hogy a DOS ún. csatolással (piping) is támogatja. A csatolás lényege az, hogy a DOS két program között olyan kapcsolatot hoz létre, amelyen keresztül az egyik program adatokat tud a másik programnak átadni.

A csatolás során az történik, hogy az adatokat átadó program az adatokat az alapértelmezés szerinti kiviteli egységre írja, a másik program pedig ezt az egységet a saját szempontjából alapértelmezés szerinti beviteli egységnek tekinti, és az adatokat innen olvassa be. A kivitel és a bevitel ily módon való összekapcsolásáról az összecsatolt programok egyike sem szerez tudomást. A szűrés és a csatolás között az az alapvető különbség, hogy míg a szűrésnél a bevitel/kivitel átirányításával az adatokat csak egy egységre vagy egy állományba lehet küldeni, a csatolással egy másik programba is.

A szűrés és a csatolás kombinálásával első ránézésre meglehetősen bonyolultnak tűnő feladatok is egyszerűen megoldhatók. Nézzünk erre egy példát. Legyen az a feladatunk, hogy az aktuális meghajtóban lévő aktuális katalógust ábécé sorrendbe rendezve a nyomtatóra kell küldenünk. Ehhez először ki kell adni a DOS DIR parancsát. Tudjuk, ha csak ezt a parancsot adnánk ki, akkor a katalóguslista a lemezre írás sorrendjében jelenne meg a képernyőn. Ahhoz, hogy ezt a listát ábécé szerint rendezzük, az előző parancshoz csatolni kell a SORT szűrőprogramot. Ez most a bemenetét a DIR parancstól kapja, a kimenete pedig – most már rendezve – a képernyőn jelenik meg. A parancssor tehát így nézne ki:

```
C>DIR : SORT
```

Ahhoz, hogy az ily módon rendezett listát a nyomtatóra küldjük, a SORT program kimenetét át kell irányítani a nyomtatóra:

```
C>DIR : SORT > PRN
```

Ezzel a kitűzött feladatot megoldottuk.

Ugyancsak jól használható a gyakorlati munkában a szűrőprogramoknak az a kombinációja, amely a katalógust ábécé sorrendben, és egyúttal képernyőoldalakra tördelve jeleníti meg, lehetővé téve az egyes tételek nyugodt olvasását. Ezt a következő parancssor beírásával érhetjük el:

```
C>DIR : SORT : MORE
```

A fenti példában két szűrőprogram végrehajtását írtuk elő egy parancssoron belül. Mivel a DOS jelenlegi verziói (a 4.0-ig bezárólag) nem támogatják a többfeladatos (multitasking) programvégrehajtást, a két szűrőprogram sem hajtható egyidejűleg végre.

Ezért a DOS először meghívja az első szűrőprogramot, végrehajtja, majd az eredményt elhelyezi egy ún. csatolóállományba (pipe-file). Ezután meghívja a második szűrőprogramot, amelynek a bemenetét azonban a csatolóállományra irányítja át, vagyis az előző szűrőprogram kimenetét olvassa. Ezzel a módszerrel halad végig mindegyik szűrőn. Ha az utolsó szűrőprogramot is végrehajtotta, akkor törli az átmenetileg létrehozott csatolóállományokat úgy, hogy a felhasználó ezeket soha nem is látja.

6.10.1. Programpéldák

Példaprogramjaink egy hasznos és a könyv készítése során gyakran használt szűrőfunkciót valósítanak meg. Míg a legtöbb nyomtatócsalád újabb tagjai már (többé-kevésbé) képesek a magyar karakterkészlet kezelésére, a régebbi típusoknál ez a lehetőség még nem automatikusan adott. Az egyik lehetséges megoldás a nyomtató hardverén végzett kisebb átalakítás révén valahol a 127-es ASCII kód feletti tartományban biztosítani a magyar ékezetes karakterek nyomtatását. Amennyiben az összes (18) ékezetes karakter nyomtatható, mindenképpen szükséges a billentyűzetről bevitt karakterek (amelyek kódjait a különböző billentyűzetszoftverek a használhatóságot szem előtt tartva definiálják) konvertálása abba a legtöbbször összefüggő kódtartományba, melyben a nyomtató működik.

A példákban a bevitt karakterek kódja 127 és 169 között van, a kinyomtatás pedig a görög karaktereknek megfelelő kódokkal történik. Maguk a programok igen egyszerűek. A C nyelvű program egyetlen paraméterrel, a konvertálni kívánt állomány nevével indítandó. A kimenet alapértelmezés szerint a képernyő, de a fent említett DOS átirányítási funkcióval ez tetszés szerint megváltoztatható. A Turbo Pascal program paraméterek nélkül indítható, futása elején bekéri a be- és kimenet azonosítóit.

Mindkét programban a gyorsabb működést támogatja, hogy a beérkező karakterek, amennyiben nem esnek bele a konvertálandó karakterek által kijelölt intervallumba, (a fenti tény eldöntése után) közvetlenül a kimenetre kerülnek. A konverzió mindkét esetben két párhuzamos tömb (táblázat stb.) felhasználásával történik, az egyik tömbben fellelve a beérkezett karaktert, annak pozíciója azonosítja a másik tömbben a kibocsátandó karaktert.

Előre jelezzük, hogy a BIOS nyomtató kezelő funkciók kapcsán be fogjuk mutatni a probléma általánosabban használható megoldását assembly nyelven.

```
/*                                                                    */
/*                                                                    */
/*                          K A R K O N V . C                         */
/*                                                                    */
/*   Funkciója: RX - 80 nyomtatón a magyar karakterkészlet megje-     */
/*              lenítéséhez szükséges konverzió elvégzése.            */
/*                                                                    */
/*                                                                    */

# include <stdio.h>

int  tab_1 [18] = { 160,130,161,148,   /* A billentyűzetről érkező   */
                    147,162,129,150,   /* NAGYBETOS,                 */
                    163,               /*      III.                  */
                    143,144,140,153,   /* kisbetűs                   */
                    167,149,154,152,   /* karakterek.                */
                    151
                  };

int  tab_2 [18] = { 235,236,237,239,   /* A nyomtatóra kerülő        */
                    242,238,246,247,   /* NAGYBETOS,                 */
                    243,               /*      III.                  */
                    224,225,226,228,   /* kisbetűs                   */
                    229,227,232,233,   /* karakterek.                */
                    230
                  };

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

void main(npar, infajl)
int npar;

char *infajl[];

{
FILE *fin, *fopen();
int   c;                               /* A konverziós program egy    */
int   i;                               /* paraméterrel hívandó, ami   */
                                       /* a konvertálandó állomány    */
if (npar != 2)                         /* neve (útvonallal).          */
{
printf("Érvénytelen paraméterezés ..");
return;
}

fin  = fopen(infajl[1], "r");

while ((c = getc(fin)) != EOF)
{                                      /* A konvertálandó karakterek  */
   if (c < 129 || c > 167)             /* intervallumán kívül nem kell*/
      putchar(c);                      /* végiglapozni a tab_1        */
   else                                /* táblázatot, a beolvasott    */
      {                                /* karakter közvetlenül kiírható.*/
      for (i=0; i<18; i++)
         if (c == tab_1[i])
            {
            putchar(tab_2[i]);         /* Konverzió a tab_1 és tab_2  */
            break;                     /* konverziós táblázatoknak meg-*/
            }                          /* felelően.                   */
         else if (i == 17)
            putchar(c);
      }
   }
fcloseall();
}
```

```
{ ┌──────────────────────────────────────────────────────────┐ }
{ │                                                          │ }
{ │                  K A R K O N V . P A S                   │ }
{ │                                                          │ }
{ │  Funkciója: RX - 80 nyomtatón a magyar karakterkészlet megje- │ }
{ │             lenítéséhez szükséges konverzió elvégzése.   │ }
{ │                                                          │ }
{ └──────────────────────────────────────────────────────────┘ }

program KARKONV;

const  tab_1 : array [1..18] of Char = 'áéíöőóüűúÁÉÍÖŐÓÜŰÚ';
       tab_2 : array [1..18] of Char = '∂∞∅∩≥∈÷≈≤αβΓΣσ▪δΘμ';

       { A billentyűzetről beérkező NAGYBETŰS, ill. kisbetűs }
       { karakterek vektorát a tab_1, a kimenetre kerülő ka- }
       { rakterek vektorát pedig a tab_2 karaktertömb tar-   }
       { talmazza.                                           }

       minkar     = 129;        { A legkisebb és legnagyobb    }
       maxkar     = 167;        { konvertálandó karakter kódja. }

       puffermeret = 1;         { Egyszerre 128 bájt olvasandó be }

var    forras      : file;
       forrasnev   : string[40];
       cel         : integer;   { Kimenet eszközének azonosítója. }
       ujkarakter  : char;
       puffer      : array [1..128] of byte;
       i           : integer;

{ ┌──────────────────────────────────────────┐ }
{ │ KARIR : EGY KARAKTER MEGJELENÍTÉSE A     │ }
{ │         KÉRT ESZKÖZÖN                    │ }
{ │    BE : A megjelenítendő karakter és     │ }
{ │         a kimeneti eszköz.               │ }
{ │    KI : -                                │ }
{ └──────────────────────────────────────────┘ }

procedure KARIR (eszkoz : integer; karakter : char);

begin
   if eszkoz = 1 then write(karakter);              { 1 : képernyő }
   if eszkoz = 2 then write(lst, karakter);         { 2 : nyomtató }
end;

{ ┌──────────────────────────────────────────┐ }
{ │ KONVERZ : EGY KARAKTER KONVERTÁLÁSA, HA  │ }
{ │           SZÜKSÉGES                      │ }
{ │      BE : A konvertálandó karakter.      │ }
{ │      KI : A konvertált karakter.         │ }
{ └──────────────────────────────────────────┘ }

function KONVERZ (karakter : char) : char;

var aktelem : integer;
    megvan  : boolean;
```

```
begin
   aktelem:=1;
   megvan:=false;
   if (karakter < chr(minkar)) or (karakter > chr(maxkar)) then
      KONVERZ := karakter

      { A konvertálandó karakterek intervallumán kívül nem kell }
      { végiglapozni a tab_1 táblázatot, a beolvasott karakter  }
      { közvetlenül kiírható.                                   }

   else

      { Konverzió a tab_1 és tab_2 konverziós táblázatoknak     }
      { megfelelően.                                            }

      begin
         while (aktelem < 19) and (megvan <> true) do
            begin
            if karakter <> tab_1[aktelem] then
               aktelem:= aktelem + 1
            else
               begin
                  KONVERZ := tab_2[aktelem];
                  megvan:=true;
               end
            end;
         if megvan = false then KONVERZ := karakter;
      end
end;

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

begin
   clrscr;
   write('Forrásállomány (max. 30 karakter)  : ');
   readln(forrasnev);
   writeln;
   write('Célészköz (1-képernyő, 2-nyomtató) : ');
   readln(cel);
   clrscr;
   if (cel = 1) or (cel = 2) then
      begin                                { A kimeneti eszköz korrekt }
      assign(forras, forrasnev);           { megadása esetén az állo-  }
      reset(forras);                       { mány olvasása és konver-  }
      while eof(forras) <> true do         { tálása 128 bájtonként.    }
         begin
         blockread(forras,puffer,puffermeret);
         for i:=1 to 128 do
            begin
            ujkarakter := KONVERZ(chr(puffer[i]));
            KARIR(cel, ujkarakter);
            end                            { A konvertált karakterek   }
         end                               { megjelenítése ciklusban.  }
      end;
   close(forras)
end.
```

6.11. Eszközmeghajtók

Eszközmeghajtóknak nevezik az operációs rendszer részét képező azon programok csoportját, amelyek közvetlen kapcsolatban állnak a hardverrel. (Mint már említettük, ezeket rendeltetésüknek megfelelően igazából perifériavezérlő programoknak kellene neveznünk, de a DOS terminológiában használt angol device driver elnevezésnek megfelelően, és nem utolsó sorban a rövidség kedvéért ezeket a továbbiakban is eszközmeghajtóknak nevezzük.) Az operációs rendszeren belül az eszközmeghajtók képezik azt a legalsó szintet, amely lehetővé teszi, hogy a felsőbb szintek a mindenkori hardvertől függetlenül végezhessék feladatukat. Úgy is felfogható, hogy az eszközmeghajtók olyan kapcsolatot alkotnak az operációs rendszer magja és a hardverspecifikus perifériák között, mint a szárazföldet a környező szigetekkel összekötő hidak: ezek biztosítják a szárazföld (a rendszer magja) és a szigetek (perifériák) közötti közvetlen kapcsolattartást.

A DOS 2.0 verziójában vezették be az ún. installálható eszközmeghajtókat, ami a rendszer rugalmasságát alapvetően megnövelte. Míg a korábbi rendszerekben az eszközmeghajtók a rendszer fix részét képezték, a 2.0 verziótól kezdődően a felhasználók saját maguk is készíthetnek eszközmeghajtókat, amelyeket a rendszerbe bekapcsolhatnak. Az előbbi hasonlattal élve, ha valaki egy új szigetet akar a szárazföldhöz kapcsolni, akkor a sziget és a szárazföld között egy új hidat építhet, és ezt bekapcsolhatja a rendszer teljes infrastuktúrájába. Mivel a DOS és a mindenkori eszközmeghajtó közötti kapcsolattartás funkcióhívások és adatstruktúrák viszonylag egyszerű sémáján alapszik, az assembly nyelvű programozásban kevésbé járatos programozók is kifejleszthetnek saját eszközmeghajtó programot, amellyel az operációs rendszerüket egy bizonyos perifériához vagy hardverhez illeszthetik.

A DOS az eszközmeghajtó programok két csoportját különbözteti meg:

– karakterkezelő eszközmeghajtó programok,
– blokk-kezelő eszközmeghajtó programok.

Az első csoportba tartozó programok a hardverrel karakterről karakterre (helyesebben: bájtról bájtra) tartják a kapcsolatot (billentyűzet, képernyő, nyomtató stb.), míg a blokkkezelő programok egyetlen funkció meghívásával karakterek egész csoportját (blokkját) tudják továbbítani (hajlékony- és merevlemezek, streamer stb.). Először vizsgáljuk meg a karakterkezelő eszközmeghajtókat.

6.11.1. Karakterkezelő eszközmeghajtók

A karakterkezelő eszközmeghajtók olyan perifériákat vezérelnek, amelyekre a bemenő, ill. kimenő adatok időben egymás után, bájtról bájtra érkeznek, ill. továbbítódnak. Általában az ilyen perifériák mindegyikéhez saját, csak az illető perifériát vezérlő eszközmeghajtó program tartozik. Ez azt jelenti, hogy a DOS-ban külön-külön vezérlő prog-

ramja van a billentyűzetnek, a képernyőnek, a nyomtatónak stb. Mindegyik karakterkezelő eszközmeghajtónak saját, 1 – 8 karakter hosszúságú logikai neve van, amelyen a felhasználói program az illető meghajtót, akárcsak egy állományt, megnyithat és lezárhat. Maga a név csak logikai név, amelyet a DOS az eszköz azonosítására használ; magához az eszközhöz nincs semmi köze.

A DOS-nak az ún. konzolt (billentyűzet és képernyő), a soros interfészt és a nyomtatót meghajtó, „beépített" programjai annyiban sajátosak, hogy ezekhez egy felhasználói program három módon is hozzáférhet:

- megnyithatja bemenetre és kimenetre nevük (CON, AUX, PRN stb.) szerint ugyanúgy, mint bármely más karakterkezelő meghajtót;

- meghívhatja a DOS 21(h) megszakítás 00(h) – 0C(h) speciális, karakterkezelő funkcióit;

- használhatja az alapértelmezés szerint kiosztott handle-ket (standard bemenet, standard kimenet, standard hibajelző, standard soros interfész és standard nyomtató) anélkül, hogy ezeket előzetesen meg kellene nyitnia.

Az installálható karakterkezelő eszközmeghajtók számát csak a rendelkezésre álló tárkapacitás korlátozza, valamint az a követelmény, hogy mindegyik meghajtónak saját, a többitől különböző nevének kell lennie. Ha egyidejűleg több, azonos nevű meghajtó létezik, akkor mindig a legutoljára betöltött meghajtó az érvényes, és a többi, azonos nevű meghajtó nem jut szerephez. Ebből a látszólagos hátrányból előnyt is lehet kovácsolni, amint ezt általában meg is tesszük: a rendszert konfiguráló CONFIG.SYS állományba a DEVICE=ANSI.SYS paranccsal bekapcsolt eszközmeghajtóval felülbíráljuk a rendszer eredeti, ehhez képest kevésbé hatékony CON meghajtóját.

A karakterkezelő eszközmeghajtók kétféle üzemmódban működtethetők: bináris (raw) és ASCII (cooked) üzemmódban. Maga a meghajtó mindkét üzemmódban azonosan működik, a kétféle üzemmód csak a DOS pufferkezelését változtatja meg az adatok beolvasásakor, ill. kiírásakor. (Erről a 2.5 alfejezetben már volt szó.) Ha az eszközmeghajtóról ASCII üzemmódban kell karaktereket beolvasni, akkor ezeket a DOS először egy belső pufferbe helyezi el, ahol megvizsgálja, hogy az éppen beolvasott karakter nem valamilyen vezérlőkarakter-e (Ctrl-P, Ctrl-S, RETURN stb.). Ha történetesen vezérlőkaraktert talált, akkor az ennek megfelelő műveletet végrehajtja, vagyis a RETURN karakter (0D(h) ASCII kód) észlelésekor a DOS befejezi a karakterek beolvasását, és a belső pufferében tárolt karaktereket átmásolja a meghívó program pufferébe. Hasonlóképpen, ha a beolvasott karakter a Ctrl-S (13(h) ASCII kód), felfüggeszti a képernyő görgetését, a Ctrl-C (03(h) ASCII kód) hatására pedig megszakítja a program futását. A Ctrl-P (10(h) ASCII kód) hatására a képernyőről a nyomtatóra irányítja a kivitelt. A nyomtatás a billentyű ismételt lenyomásakor befejeződik.

Bináris üzemmódban ezek a vizsgálatok elmaradnak. Ha pl. egy program 10 karaktert akar beolvasni, akkor pontosan 10 karaktert be is olvas, még akkor is, ha a program kezelője netán már a második karakterként a RETURN billentyűt nyomta volna le.

Ebben az üzemmódban a karakterek tehát nem teszik meg a belső pufferen át vezető kerülő utat, hanem közvetlenül a meghívó program pufferébe kerülnek. Kiíráskor a Ctrl-C, ill. a Ctrl-Break billentyű lenyomásának figyelése is elmarad.

6.11.2. Blokk-kezelő eszközmeghajtók

A blokk-kezelő eszközmeghajtók általában tömegtárakkal, így merevlemezzel, hajlékonylemezzel és streamerrel való kapcsolattartásra szolgálnak. Ezek minden egyes meghívásukkor egyidejűleg nem egy, hanem több karaktert továbbítanak. Az egy meghívással továbbított karakterek csoportját blokknak nevezzük. A blokkok mérete lehet rögzített (lemezes egységek), de lehet változó is (mágnesszalag).

A karakterkezelő eszközmeghajtókkal ellentétben a blokk-kezelő meghajtók egyidejűleg több egységet is képesek kiszolgálni. Arra is van lehetőség, hogy egy egységet több logikai egységre bontsunk: például a 40 Mbájt kapacitású merevlemez felosztható két, egyenként 20 – 20 Mbájtos logikai egységre, amelyeknek most C és D lesz az azonosítója. A DOS a logikai egységekhez most nem neveket, hanem az ábécé sorrendnek megfelelő betűket rendel, és ezután ez a betű lesz az illető egység (lemezmeghajtó) azonosítója (vagyis az első logikai egységé az A betű, a másodiké a B, majd C stb.). Ezeket az azonosítókat nem az eszközmeghajtó választja meg, hanem az határozza meg, hogy maga a program az eszközmeghajtók által alkotott láncon belül hol helyezkedik el. A rendszer mindegyik logikai egységet állománytároló egységnek tekint, és ezeket eszerint is kezeli. Ennek megfelelően mindegyik egységnek rendelkeznie kell egy betöltő szektorral, állományelhelyezési táblával (FAT), főkatalógussal stb.

A blokk-kezelő eszközmeghajtók – a karakterkezelő eszközmeghajtóktól eltérően – nem különböztetik meg a bináris és az ASCII üzemmódot. A blokkokat pontosan úgy olvassák be és írják ki, ahogyan azokat megkapják, és az átviendő adatokat semmilyen módon sem változtatják meg.

6.11.3. Az eszközmeghajtók felépítése

Bár a karakterkezelő és a blokk-kezelő eszközmeghajtók néhány fontos részletben különböznek egymástól, belső szerkezetük azonos, így mindkét típus együtt vizsgálható. Az eszközmeghajtók három, egymástól jól elkülöníthető részből állnak. Ezek:

– az eszközmeghajtó fejléce (device header),
– a stratégia rutin (strat),
– a megszakítási rutin (intr).

A következőkben külön-külön megvizsgáljuk ezeket az összetevőket.

Az eszközmeghajtó fejléce

Az eszközmeghajtó fejléce (angol nevén device header), amint a neve is mutatja, az eszközmeghajtó elején helyezkedik el, és mind a DOS, mind az eszközmeghajtó számára fontos információkat tartalmaz. Amint majd látni fogjuk, a DOS az eszközmeghajtókat egymással láncba kapcsolja. A fejléc első két mezőjén (00(h) – 03(h) ofszetcímek) a meghajtók láncában soron következő meghajtó ofszet- és szegmenscíme áll. Ezekre a címekre a megfelelő értékeket a DOS írja be akkor, amikor a rendszer indítása során az eszközmeghajtót bekapcsolja a láncba. A programozónak a meghajtó fejlesztésekor csak az ehhez szükséges helyet kell lefoglalnia. A fejléc 04(h) ofszetcímén az eszközmeghajtó ún. attribútum szava található, amelynek tartalmával részletesen megismerkedünk. A 06(h), ill. a 08(h) ofszetcímeken a stratégia rutin, ill. a megszakítási rutin ofszetcíme áll. Ezekre a címekre a DOS-nak van szüksége azért, hogy az illető rutinokat meghívhassa. (Szegmenscímek megadására nincs szükség, mert a rutinok szegmenscíme megegyezik a fejléc szegmenscímével.) A 0A(h) ofszetcímen kezdődő, 8 bájt hosszúságú mezőbe kerül az eszközmeghajtó azonosítója. Ha az illető meghajtó karakterkezelő, akkor ide a meghajtó logikai neve (mint pl. PRN vagy COM1) kerül. Ha a név 8 karakternél rövidebb, akkor az üresen maradt bájtokba 00-t kell írni. Ha a meghajtó blokk-kezelő, akkor a 0A(h) mezőbe azoknak a logikai egységeknek a száma kerül, amelyeket ez a blokk-kezelő vezérel. A többi 7 bájt foglalt terület. Az eszközmeghajtó fejlécének szerkezetét a 25. ábra mutatja.

00(h)	a következő eszközmeghajtó ofszetcíme
02(h)	a következő eszközmeghajtó szegmenscíme
04(h)	az eszközmeghajtó attribútum szava
06(h)	a stratégia rutin belépésének ofszetcíme
08(h)	megszakítási rutin belépésének ofszetcíme
0A(h)	– ha karakterkezelő, akkor az eszközmeghajtó logikai neve (max. 8 bájton) – ha blokk-kezelő, akkor az eszközmeghajtó által vezérelt logikai egységek száma (1 bájton)

25. ábra: Az eszközmeghajtó fejlécének szerkezete

Az eszközmeghajtó fejlécének 04(h) címén lévő attribútum szó az eszközmeghajtóról tartalmaz fontos információkat. Többek között innen szerez tudomást a DOS arról, hogy az illető meghajtó karakter- vagy blokkezelő meghajtó-e.

Bit	Jelentése
0.	= 1, ha az aktuális standard beviteli eszköz (stdin)
1.	= 1, ha az aktuális standard kiviteli eszköz (stdout)
2.	= 1, ha az aktuális nemlétező (NUL) eszköz
3.	= 1, ha az aktuális órameghajtó (CLOCK$) eszköz
4 – 10.	foglalt, 0-nak kell lenniük
11.	= 1, ha képes a cserélhető adathordozó eszköz cseréjének felismerésére (OCR bit)
12.	foglalt, 0-nak kell lennie
13.	blokk-kezelő eszközmeghajtó esetén: = 1, ha az adathordozó eszköz jellemzőinek megállapításához a betöltő szektorban lévő BIOS paraméterblokkot kell igénybe venni, = 0, ha az adathordozó eszköz azonosító (ID) bájtját kell használni karakterkezelő eszközmeghajtó esetén: = 1, ha az adatkivitelt az eszközmeghajtó FOGLALT állapotáig támogatja
14.	= 1, ha az IOCTL írás/olvasást támogatja (IOCTL bit)
15.	= 1, ha karakterkezelő eszközmeghajtó = 0, ha blokk-kezelő eszközmeghajtó

26. ábra: Az attribútum szó felépítése, és az egyes bitek jelentése

A blokk-kezelő eszközmeghajtók számára az attribútum szóban csak a 11. és a 13–15. biteknek van jelentőségük, a többinek 0-nak kell lennie.

A stratégia rutin (strat)

A DOS a stratégia rutint az eszközmeghajtó első betöltésekor és installálásakor hívja meg, valamint minden olyan esetben, amikor az alkalmazói program I/O kéréssel fordul az eszközmeghajtóhoz. A rutin meghívásakor a DOS a stratégia rutinnak egy kettős szóból álló mutatót ad át, amely egy adatszerkezetre mutat. Ezt az adatszerkezetet mi a továbbiakban „igénybejelentő fejlécnek", ill. röviden IBF-nek nevezzük (angol neve request header, ill. röviden RH). Az IBF az eszközmeghajtó által végrehajtandó műveletre vonatkozó információkat tartalmazza. A DOS jelenlegi verzióiban maga a stratégia rutin semmiféle műveletet sem hajt végre, a feladata mindössze az, hogy tárolja azt a mutatót, amit a DOS a részére átadott, és amely az IBF-re mutat. Azt követően, hogy megkapta az IBF címét, a rutin a vezérlést azonnal visszaadja a DOS-nak. A rutin semmiféle 21(h) megszakítást sem hívhat meg.

A következőkben vizsgáljuk meg közelebbről az eszközmeghajtók működésében igen fontos szerepet játszó igénybejelentő fejléc szerkezetét (lásd a 27. ábrát).

Az IBF első 13 bájtja valamennyi eszközmeghajtó funkció esetében azonos jelentésű, ezért ezt a fejléc statikus részének is nevezik. Az ezt követő bájtok száma és jelentése az igényelt funkciótól függően változó. Az IBF-ben lévő bájtokat egyaránt írja és olvassa a DOS is, és az eszközmeghajtó is. Az IBF talán legfontosabb része az ún. parancskód vagy funkciókód (angol neve command code), amelyet az IBF a harmadik bájtján ad át az eszközmeghajtó megszakítási rutinjának. Ez a parancskód dönti el, hogy az eszközmeghajtónak milyen funkciót kell meghívnia (írás, olvasás stb.). Az IBF többi bájtja egyéb információkat ad át az eszközmeghajtónak, amelyekre rövidesen kitérünk. A 27. ábrával kapcsolatban ismételten hangsúlyozzuk, hogy az IBF-nek csak az első 13 bájtja azonos jelentésű, míg a többi az igényelt funkciótól függően változó. Az ábrán bemutatott eset az eszközmeghajtó írási és olvasási funkciójának meghívásakor igaz.

00(h)	az IBF hossza bájtokban	1 bájt
01(h)	a megszólított egység azonosító száma	1 bájt
02(h)	az IBF parancskódja	1 bájt
03(h)	a visszatérési státusz-szó	1 szó
05(h)	foglalt adatterület	8 bájt
0D(h)	adathordozó leíró bájtja	1 bájt
0E(h)	átadandó tárterület ofszet- és szegmenscíme	2 szó
12(h)	bájtok/szektorok száma	1 szó
14(h)	kezdő szektor száma	1 szó

27. ábra: Az IBF (igénybejelentő fejléc) szerkezete.

A megszakítási rutin (intr)

Az eszközmeghajtó utolsó és egyúttal legbonyolultabb része az ún. megszakítási rutin, amelyet a DOS közvetlenül a stratégia rutin után hív meg. Tulajdonképpen ez az a rutin, amelyik igazából az eszközmeghajtó programnak tekinthető: ez hajtja (vagy hajtatja) végre az aktuális beviteli/kiviteli műveleteket azoknak az információknak az alapján, amelyeket az IBF adatszerkezetben megkap. A rutin – az inicializálásakor meghívható, igen korlátozott számú DOS megszakítástól eltekintve – 21(h) típusú megszakításokat nem hajthat végre.

Amikor befejezte a kívánt műveletet, a megszakítási rutin az IBF státusz mezőjébe beírt kóddal tájékoztatja a DOS-t a végrehajtás eredményéről. Emellett még más adatokat is beírhat az IBF-be (pl. az átvitt bájtok vagy szektorok számát).

Maga a rutin a végrehajtását tekintve három fő részre osztható:

- Első lépésben verembe menti a funkciók meghívásakor érintett regiszterek tartalmát, elolvassa az IBF-ben lévő parancskódot, majd – általában egy ugrótáblán keresztül – ráugrik a parancskód által kijelölt szubrutinra.

- A rutin második része az ezekhez a parancskódokhoz tartozó, és a DOS által várt szubrutinok, ill. funkciók csoportja.

- A rutin a harmadik, befejező részében beírja a visszatérési státuszt és a hibakódokat az IBF megfelelő mezőjébe, visszaállítja a regiszterek eredeti tartalmát, és visszaadja a vezérlést a DOS-nak.

Mielőtt rátérnénk az egyes funkciók ismertetésére, nézzük meg, milyen értéket ad vissza a megszakítási rutin az IBF visszatérési státusz-szó mezőjében.

A 16 bites státusz-szó a benne lévő információt tekintve két bájtra osztható: a felső bájt (8 – 15. bit) az eszközmeghajtó tényleges állapotáról (hibás végrehajtás, eszközmeghajtó foglalt, eszközmeghajtó kész), az alsó bájt (0 – 7. bit) pedig hibás végrehajtás esetén a hiba okáról ad felvilágosítást.

A 28. ábra az IBF visszatérési státusz-szavának felépítését, a 29. ábra pedig a státusz-szó alsó bájtjában lévő hibakódokat mutatja be.

Bit	Jelentése
15.	ha értéke 1, akkor hiba a végrehajtásban
12 – 14.	foglalt bitek
9.	ha értéke 1, akkor az eszközmeghajtó FOGLALT (busy)
8.	ha értéke 1, akkor az eszközmeghajtó KÉSZ (done)
0 – 7.	a hiba kódja, ha a 15. bit értéke 1

28. ábra: Az IBF visszatérési státusz-szavának felépítése

Kód	Jelentése
0	az adathordozó eszköz írásvédett
1	ismeretlen egység
2	lemezmeghajtó nem kész állapotú
3	ismeretlen parancs
4	adathiba (CRC)
5	hibás adathossz
6	keresési hiba
7	ismeretlen típusú adathordozó
8	szektort nem találta
9	nyomtatóból kifogyott a papír
10	íráshiba
11	olvasáshiba
12	általános hiba
13 – 14	foglalt kód
15	érvénytelen lemezcsere (csak a 3.0 verziótól)

29. ábra: Hibakódok az IBF visszatérési státusz-szavának alsó bájtjában

6.11.4. A parancskódok funkciói

A DOS a 3.0 verziójáig bezárólag összesen 17 olyan parancskódot, ill. az ezekhez tartozó funkciót definiált, amelyeket az eszközmeghajtóknak támogatniuk kell. Az itt következő ábrán felsoroljuk a parancskódokat, a hozzájuk tartozó funkciókat, továbbá azt is feltüntetjük, hogy az eszközmeghajtók mely típusánál játszanak szerepet, és melyik verziónál vezették be őket (30. ábra).

Parancs kód	A funkció feladata	Eszközmeghajtó típusa	DOS verziószám
0	inicializálás	kar/blokk	2.0
1	adathordozó vizsgálata	blokk	2.0
2	BIOS paraméterblokk (BPB) létrehozása	blokk	2.0
3	IOCTL olvasás	kar/blokk	2.0
4	olvasás	kar/blokk	2.0
5	megtartó olvasás	kar.	2.0
6	beviteli állapot vizsgálata	kar.	2.0
7	beviteli pufferek törlése	kar.	2.0
8	írás	kar/blokk	2.0
9	írás verifikálással	blokk	2.0
10	kiviteli állapot vizsgálata	kar.	2.0
11	kiviteli pufferek törlése	kar.	2.0
12	IOCTL írás	kar/blokk	2.0
13	eszköz megnyitása	kar/blokk	3.0
14	eszköz lezárása	kar/blokk	3.0
15	cserélhető adathordozó	blokk	3.0
16	kivitel míg foglalt	kar.	3.0

30. ábra: Az eszközmeghajtók parancskódjai és a hozzájuk tartozó funkciók (a kar. karakterkezelő, a blokk blokk-kezelő eszközmeghajtót jelent)

Az 0 – 12. sorszámú funkciókat valamennyi installálható eszközmeghajtónak támogatnia kell, még akkor is, ha ezek egyetlen tevékenysége mindössze az, hogy az IBF visszatérési státusz-szavában a KÉSZ bitet (8. bit, lásd a 28. ábrát) magasra állítják.

A DOS 3.0 verziójában ezt a 13 funkciót további 4-gyel bővítették ki, amelyeket az eszközmeghajtók támogathatnak, de ez nem kötelező. A DOS figyeli az eszközmeghajtó fejlécében lévő attribútum szó bitjeinek értékét, és innen tudja megállapítani, hogy az egyes eszközmeghajtók mely funkciókat támogatnak.

Az ábrából láthatjuk, hogy egyes funkciók csak a karakterkezelő, mások csak a blokk-kezelő, míg néhány mindkét típusú eszközmeghajtóra vonatkozik. Itt is érvényes az, hogy a nem használt funkciókhoz is tartozniuk kell végrehajtható rutinnak, még akkor is, ha ez a rutin nem tesz mást, minthogy a KÉSZ bitet magasra állítja.

A következőkben vizsgáljuk meg részletesen ezeket a funkciókat a számozásuk szerinti sorrendben. Az ismertetésre kerülő funkciók mindegyike a parancskódot és más paramétereket abból az IBF-ből (igénybejelentő fejléc, lásd a 27. ábrát) kapja, amelynek címét a DOS az ES:BX regiszterpárban a stratégia rutinnak átadta. A funkciók befejeződésükkor a visszatérési paramétereket ugyancsak ebben az IBF-ben helyezik el. Ezért az egyes funkcióknál ennek az IBF-nek a kezdetétől számított azon ofszetcímeit adjuk meg, amelyek a funkciók meghívásakor a funkciók számára paramétereket tartalmaznak, illetve amelyekbe a funkciók visszatérésükkor adatokat írnak. Az ofszetcímek mellett még azt is megadjuk, hogy az illető paraméter bájt vagy szó hosszúságú. Lesz még egy harmadik, „mutató" adattípusunk is, amely egy táblára mutat, és két, a tárban egymást követő szóból áll. Az első szó a tábla ofszetcímét, a második a szegmenscímét tartalmazza.

00 – 00(h) funkció: Az eszközmeghajtó inicializálása

Ezt a funkciót a DOS csak a rendszer indításakor hívja meg, amelynek során az eszközmeghajtót installálja és inicializálja (ekkor kapnak különböző belső változók, ill. megszakítási vektorok kezdőértéket). Mivel az inicializálás idején az operációs rendszer még nem épült fel teljes egészében, ez a rutin a DOS 21(h) megszakításnak csak a 01(h) – 0C(h) és a 30(h) funkcióit hívhatja meg. Ezek a funkciók azonban éppen elegendők ahhoz, hogy a DOS verzió számát le lehessen kérdezni, vagy a képernyőn üzenetet kapjunk az eszközmeghajtó installálásáról. Abban az esetben, ha az újonnan bekapcsolt eszközmeghajtó CON meghajtó lenne, az üzenetet még a régi CON meghajtó írja ki, mert az új meghajtó csak az inicializáló rutinjának befejeződése után épül be teljes egészében a rendszerbe.

Az inicializáló rutin az IBF-ből a parancskódon túlmenően két információt vesz át. Ezek közül az egyik egy tárcím, amely a CONFIG.SYS állományból arra a karakterláncra mutat, amely a rendszer indításakor az eszközmeghajtó installálását tulajdonképpen kezdeményezte. (Mint tudjuk, a DOS az indításakor figyeli a CONFIG.SYS állományban lévő DEVICE=XXX.SYS parancsokat, és a rendszert ezeknek megfelelően konfigurálja.) Így például a

```
DEVICE=ANSI.SYS
```

kijelölésben az ANSI.SYS az eszközmeghajtó neve, és az IBF-ből átvett tárcím az egyenlőségjel után álló első karakterre, vagyis az A betűre mutat. A második információra csak a 3.0 verziótól felfelé, és csak blokk-kezelő eszközmeghajtó esetén van szük-

ség. Ez az információ az ezáltal az eszközmeghajtó által vezérelt első logikai egység (lemezmeghajtó) azonosító számát adja meg (a 0 az A jelű, az 1 a B jelű stb. egységet jelenti. Amint erről már volt szó, ezt az azonosítót nem az eszközmeghajtó választja meg, hanem a DOS „osztja ki".)

Az inicializáló rutinnak 4 paramétert kell visszaadnia az őt meghívó DOS funkciónak. Az első – mint minden más funkciónál – itt is a visszatérési kód, vagyis az arra vonatkozó információ, hogy a funkció végrehajtása sikeres volt-e. Blokk-kezelő eszközmeghajtó esetén szolgáltatnia kell továbbá az általa vezérelt logikai egységek számát. A DOS a blokk-kezelő eszközmeghajtó által az IBF-be visszaírt egységek számát használja fel arra, hogy kiossza számukra az egységazonosítót. Ha pl. egy új eszközmeghajtó installálásakor az előzőleg már kiosztott legnagyobb sorszám a 3-as volt (ami a D betűnek felel meg), és az új meghajtó két logikai egységet vezérel, akkor ezek a 4-es és az 5-ös sorszámot, vagyis az E és az F betűazonosítót kapják. (Csak megjegyezzük, hogy a vezérelt egységek számát a DOS az eszközmeghajtó fejlécéből is kiolvashatná, mégsem teszi.)

A funkciónak az eszközmeghajtó által elfoglalt RAM-terület utáni első szabad címet (break address) is vissza kell adnia az IBF-ben, hogy a DOS tudja, hová helyezheti el – néhány saját, belső adatszerkezetet követően – a láncban következő eszközmeghajtót.

Végül, ha blokk-kezelő eszközmeghajtóról van szó, akkor utolsó paraméterként át kell adnia egy mutatót, ami egy tábla címére mutat. Ez a tábla ofszetcímekként maga is mutatókat tartalmaz, amelyek további adatszerkezetekre, az ún. BIOS paraméterblokkokra (a továbbiakban BPB) mutatnak. Egy ilyen BPB, amelynek szerkezetét a 31. ábra mutatja, egy eszközmeghajtó által vezérelt logikai egységgel kapcsolatos legfontosabb információkat tartalmazza. Mindegyik eszközmeghajtóhoz tartozik egy BPB. A BPB címtáblájában az első cím az első eszközmeghajtóhoz tartozó BPB-re, a második a második eszközmeghajtóhoz tartozó BPB-re stb. mutat. Ha viszont az eszközmeghajtók által vezérelt logikai egységek közül néhánynak (vagy mindegyiknek) azonos a fizikai felépítése (pl. két azonos hajlékonylemezes meghajtó), akkor a BPB-k címeit tartalmazó mutatók ugyanarra a BPB-re is mutathatnak.

Bájtok	Jelentése
00(h) – 01(h)	bájt/szektor
02(h)	szektor/klaszter (2 hatványa)
03(h) – 04(h)	foglalt szektorok száma (a betöltő szektort is beleértve)
05(h)	állományelhelyezési táblák (FAT) száma
06(h) – 07(h)	főkatalógus bejegyzéseinek száma (max.)
08(h) – 09(h)	szektorok száma az adathordozón
0A(h)	adathordozó leíró bájtja
0B(h) – 0C(h)	egy FAT által elfoglalt szektorok száma
0D(h) – 0E(h)	szektor/sáv (3.0 verziótól)
0F(h) – 10(h)	író/olvasófejek száma (3.0 verziótól)
11(h) – 12(h)	rejtett szektorok száma
13(h) – 1E(H)	foglalt a későbbi verziók számára

31. ábra: A BPB (BIOS paraméterblokk) szerkezete

A 00(h) funkció hívása (az eszközmeghajtó inicializálása)

Bemeneti paraméterek:

IBF + 2	bájt	Parancskód száma = 0
IBF + 18	mutató	Annak a karakternek az ofszet- és szegmenscíme, amelyik a CONFIG.SYS állományban lévő DEVICE paraméter utáni egyenlőségjel után áll (csak olvasható információ)
IBF + 22	bájt	A blokk-kezelő eszközmeghajtó által vezérelt első logikai egység azonosító száma (0 = A, 1 = B stb.). (Csak blokk-kezelőknél, és csak a 3.0 verziótól.)

Visszatérési paraméterek:

IBF + 3	szó	Visszatérési státusz-szó

IBF + 13	bájt	Egységek száma (csak blokk-kezelőknél)
IBF + 14	mutató	Az eszközmeghajtó utáni első szabad tárhely címe (break address)
IBF + 18	mutató	A BPB címtáblájának kezdőcíme (csak blokk-kezelőknél)

01 – 01(h) funkció: Adathordozó vizsgálata

Ez a funkció csak a blokk-kezelő eszközmeghajtóknál jut szerephez. A karakterkezelő meghajtóknál nincs más dolga, mint hogy az IBF státusz-szavában lévő KÉSZ bitet magasra állítsa. A DOS ezt a funkciót akkor hívja meg, ha az adathordozóra irányuló meghívás nemcsak egyszerű írási vagy olvasási műveletre vonatkozik. Ezzel a funkcióhívással a DOS azt állapítja meg, hogy két adatátviteli művelet közben vajon kicserélődött-e az adathordozó egység (kézenfekvően a hajlékonylemez). Ha a rutin feltételezi, hogy az utolsó hozzáférés óta nem történt csere, akkor a DOS a további műveletekhez a még tárban lévő FAT tartalmát használja. Így megtakarítja az ismételt beolvasást, és javítja a saját hatékonyságát.

Előfordulhat, hogy néhány esetben, főként hajlékonylemezes egységeknél a cserére vonatkozó kérdésre csak bizonytalan válasz adható. Ezért a DOS ennek a funkciónak azt is megengedi, hogy a kérdésre az igen és a nem mellett a számítógépes környezetben nem éppen szokásos „talán"-nal is válaszolhasson. A DOS a feltett kérdésre kapott választól függően az alábbiak szerint cselekszik:

- Ha az adathordozó nem cserélődött ki, akkor a maga által felállított szabályok szerint folytatja a munkáját.

- Ha a funkció lemezcseréről tájékoztatta, akkor az adott logikai egységgel (lemezzel) kapcsolatos valamennyi belső puffert kiüríti (beleértve azokat is, amelyeket a kicserélt lemezre kellett volna írnia), meghívja a BPB-t létrehozó 02(h) funkciót, majd betölti az új adathordozó FAT tábláját és főkatalógusát. A pufferek kiürítésekor a bennük tárolt adatok természetesen elvesznek.

- Ha a kapott válasz a „talán", akkor a DOS további lépései a pufferek állapotától függenek:

- Ha a pufferek üresek, ill. azok tartalmát a DOS kiírta a lemezre, akkor a DOS abból indul ki, hogy az adathordozó kicserélődött, és úgy viselkedik, mintha a funkció igennel válaszolt volna.

- Ha a pufferekben még vannak olyan adatok, amelyeket ki kellene írnia, akkor a DOS azt feltételezi, hogy nem történt csere, és az éppen aktuális adathordozóra ráírja a pufferekben lévő adatokat. Ez az eljárás magában rejti annak a veszélyét, hogy – amennyiben az adathordozó mégis kicserélődött – a ráírt adatok a

kicserélt adathordozó adatszerkezetét összezavarják, és ezáltal az egészet használhatatlanná teszik.

Ha az eszközmeghajtó fejlécében lévő attribútum szó 11. bitjének értéke 1 (tehát az eszközmeghajtó támogatja a cserélhető adathordozó eszköz cseréjének felismerését), akkor a használt DOS verzió 3.0 vagy e fölötti. Ha most a funkció a visszatérésekor lemezcseréről értesíti a DOS-t, azaz az IBF-be a lemezcsere megtörténtét jelző –1 kódot írja vissza, akkor a funkciónak még azt a szegmens- és ofszetcímet is vissza kell adnia, ami a lemezmeghajtóban előzőleg volt lemez kötetcímkéjének ASCII karakterláncára mutat. (Amennyiben a lemeznek nem volt kötetcímkéje, akkor a funkció egy NO_NAME nevű ASCII karakterláncra mutathat. A karakterláncokat egy végjelnek – a 00(h) ASCII kódnak – kell lezárnia.)

Ha a DOS azt állapítja meg, hogy a lemezcserére anélkül került sor, hogy a belső puffereiben lévő adatokat a lemezre írhatta volna, akkor ezt a 0F(h) kritikus hiba kódjának kibocsátásával jelzi (érvénytelen lemezcsere). A felhasználói programok ezt a kódot lekérdezhetik, és a hibára ennek megfelelően reagálhatnak (pl. felszólítják a felhasználót, hogy helyezze vissza az előző lemezt a meghajtóba).

Miután megvizsgáltuk a DOS döntési mechanizmusát, vegyük szemügyre a funkciónak átadott, és a funkció által visszaírt paramétereket.

Természetesen most is át kell adni a parancskód számát. Ezen túlmenően meg kell adni a logikai egység azonosító kódját, továbbá az illető logikai egységben utoljára használt adathordozónak a leíró bájtját. Ez az adat csak azoknál az egységeknél érdekes, amelyek többféle formátumot képesek kezelni. (Ilyenek az IBM PC/AT hajlékonylemezes egységei, amelyek 360 kbájtos és 1.2 Mbájtos lemezekkel is tudnak dolgozni.) A DOS 3.0 verziójáig bezárólag támogatott adathordozók leíró bájtjainak értékét, ill. ezek jelentését a 32. ábra mutatja.

Kód	Jelentése
0F8(h)	merevlemez
0F9(h)	5.25"-os, kétoldalas, 15 szektoros hajlékonylemez
0FC(h)	5.25"-os, egyoldalas, 9 szektoros hajlékonylemez
0FD(h)	5.25"-os, kétoldalas, 9 szektoros hajlékonylemez
0FE(h)	5.25"-os, egyoldalas, 8 szektoros hajlékonylemez
0FF(h)	5.25"-os, kétoldalas, 8 szektoros hajlékonylemez

32. ábra: A DOS 3.0 verziójáig bezárólag támogatott adathordozók leíró bájtjai

A 01(h) funkció hívása (az adathordozó vizsgálata)

Bemeneti paraméterek:

IBF + 1	bájt	Egység kódja
IBF + 2	bájt	Parancskód = 1
IBF + 13	bájt	Adathordozó leíró bájtja

Visszatérési paraméterek:

IBF + 3	szó	Visszatérési státusz-szó
IBF + 14	bájt	Adathordozó csere bitje: –1, ha volt lemezcsere 0, ha „talán" volt lemezcsere 1, ha nem volt lemezcsere
IBF + 15	mutató	Mutató az előző lemez kötetcímkéjére, ha az eszközmeghajtó attribútum-szavában lévő 11. bit értéke 1, és lemezcsere történt (csak a 3.0 verziótól felfelé)

02 – 02(h) funkció: BIOS paraméterblokk (BPB) létrehozása

Ez a funkció csak a blokk-kezelő eszközmeghajtóknál jut szerephez. A karakterkezelő meghajtóknál nincs más dolga, minthogy az IBF státusz-szavában lévő KÉSZ bitet magasra állítsa. A DOS ezt a funkciót arra használja, hogy rendelkezésére álljon az a mutató, amely az aktuális lemez érvényes BPB-jére mutat, és akkor hívja meg, ha a 01(h) számú, adathordozót vizsgáló funkció lemezcserét jelzett, vagy ha „talán"-nal válaszolt, és a DOS belső puffereiben nem maradtak olyan adatok, amelyeket az esetleg kicserélt lemezre kellett volna írnia (vagyis röviden szólva a lemezcsere érvényes volt).

A funkció a meghívása előtt az IBF-ben egy mutatót kap, amely egy szektor hosszúságú (512 bájt) pufferre mutat. Ha az eszközmeghajtó fejléc attribútum szavában a 13. bit értéke 0, akkor ebben a pufferben a FAT első szektora van (ami tartalmazza az adathordozó leíró bájtját). Az eszközmeghajtónak ezt a puffert nem szabad felülírnia. Ha viszont a bit értéke 1, akkor a puffer munkaterületként használható.

A funkciónak a DOS 3.0 verziójától felfelé, ha az eszközmeghajtó fejléc attribútum szavában a 11. bit értéke 1, akkor az adathordozó kötetcímkéjét is olvasnia és tárolnia kell. Ha ugyanis az egységben az adathordozó kicserélődött, akkor ezt a 01(h) funkció rendelkezésére kell bocsátania.

A 02(h) funkció hívása: (BIOS paraméterblokk létrehozása)

Bemeneti paraméterek:

IBF + 1	bájt	Egység kódja
IBF + 2	bájt	Parancskód = 2

IBF + 13	bájt	Adathordozó leíró bájtja
IBF + 14	mutató	Puffer címe

Visszatérési paraméterek:

IBF + 3	szó	Visszatérési státusz-szó
IBF + 18	mutató	Az új BPB mutatója

03 – 03(h) funkció: IOCTL olvasás

Az IOCTL olvasás funkció – amelyet mind a karakter-, mind a blokk-kezelő eszközmeghajtók használnak – lehetővé teszi, hogy az eszközmeghajtó az információkat közvetlenül továbbítsa a felhasználói programokhoz. A funkció a 21(h) megszakítás 44(h) funkcióján keresztül akkor hívható meg, ha az eszközmeghajtó attribútum szavában lévő IOCTL bit (14. bit) magas.

A 03(h) funkció hívása (IOCTL olvasás)

Bemeneti paraméterek:

IBF + 1	bájt	Egység kódja (ha blokk-kezelő)
IBF + 2	bájt	Parancskód = 3
IBF + 13	bájt	Adathordozó leíró bájtja
IBF + 14	mutató	Átviteli tárterület címe
IBF + 18	szó	Olvasandó bájtok/szektorok száma
IBF + 20	szó	Kezdő szektor száma (ha blokk-kezelő)

Visszatérési paraméterek:

IBF + 3	szó	Visszatérési státusz-szó
IBF + 18	szó	Beolvasott bájtok/szektorok száma

04 – 04(h) funkció: Olvasás

Az olvasás funkció – amelyet mind a karakter-, mind a blokk-kezelő eszközmeghajtók használnak – a megadott eszközről adatokat olvas be egy megadott pufferbe. Ha az olvasás során hiba történt, akkor a funkciónak azon túlmenően, hogy a visszatérési státusz-szóban a hibajelző bitet magasra kell állítania, még a hiba nélkül beolvasott bájtok vagy szektorok számát is vissza kell írnia az IBF-be.

A DOS 3.0 verziójától felfelé ez a funkció a nyitó és záró funkciók (0D(h) és 0E(h)) által beállított nyitott állományok számának és az adathordozó leíró bájtjának a segítségével azt is meg tudja határozni, hogy történt-e nem megengedett lemezcsere.

A 04(h) funkció hívása (olvasás)

Bemeneti paraméterek:

IBF + 1	bájt	Egység kódja (ha blokk-kezelő)
IBF + 2	bájt	Parancskód = 4
IBF + 13	bájt	Adathordozó leíró bájtja
IBF + 14	mutató	Átviteli tárterület címe
IBF + 18	szó	Olvasandó bájtok/szektorok száma
IBF + 20	szó	Kezdő szektor száma (ha blokk-kezelő)

Visszatérési paraméterek:

IBF + 3	szó	Visszatérési státusz-szó
IBF + 18	szó	Beolvasott bájtok/szektorok száma
IBF + 22	mutató	Mutató a kötetcímkére, ha a visszatérési hibakód 0F(h) (3.0 verziótól felfelé).

05 – 05(h) funkció: **Megtartó olvasás**

A megtartó olvasás funkció csak a karakterkezelő eszközmeghajtóknál jut szerephez. A blokk-kezelő meghajtóknál nincs más dolga, minthogy az IBF státusz-szavában lévő KÉSZ bitet magasra állítsa. Ez a funkció azzal a karakterrel tér vissza, amelyet a 4-es parancskódú olvasási funkció olvasna ki az eszközmeghajtó belső pufferéből. Az utóbbitól eltérően azonban ez a megtartó funkció úgy olvassa a karaktert, hogy a pufferből nem távolítja el. A DOS ezt a funkciót arra használja, hogy – mialatt más műveletek folynak – figyelje, nem kerültek-e Ctrl-C karakterek a billentyűzetmeghajtó pufferébe.

A 05(h) funkció hívása: (megtartó olvasás)

Bemeneti paraméterek:

IBF + 2	bájt	Parancskód = 5

Visszatérési paraméterek:

IBF + 3	szó	Visszatérési státusz-szó: ha a FOGLALT bit = 0, akkor legalább egy karakter vár a sorban, ha a FOGLALT bit = 1, akkor nincs karakter a sorban.
IBF + 13	bájt	Beolvasott karakter (ha a FOGLALT bit = 0).

06 – 06(h) funkció: **Beviteli állapot vizsgálata**

Ezt a funkciót is csak a karakterkezelő eszközmeghajtóknak kell támogatniuk. A blokk-kezelők feladata mindössze annyi, hogy a KÉSZ bitet magasra kell állítaniuk.

Ez a funkció az eszközmeghajtó aktuális állapotával tér vissza, lehetővé téve, hogy a DOS megvizsgálja, várakoznak-e karakterek az eszközmeghajtó úgynevezett előretartó pufferében. Ha az eszközmeghajtónak nincs ilyen előretartó puffere, akkor a beviteli állapotot vizsgáló funkciónak visszatéréskor a FOGLALT bitet mindig 0-ra kell állítania, hogy a DOS ne várjon a végtelenségig az olvasó (04(h)) vagy a megtartó olvasó (05(h)) funkció meghívására.

A 06(h) funkció hívása (beviteli állapot vizsgálata)

Bemeneti paraméterek:

IBF + 2	bájt	Parancskód = 6

Visszatérési paraméter:

IBF + 3	szó	Visszatérési státusz-szó: ha a FOGLALT bit = 1, akkor olvasás kérési funkció megy a fizikai egységre, ha a FOGLALT bit = 0, akkor már vannak karakterek a sorban, és rövidesen megérkezik az olvasás kérési funkció.

07 – 07(h) funkció: **Beviteli pufferek törlése**

Ezt a funkciót is csak a karakterkezelő eszközmeghajtóknak kell támogatniuk. A blokk-kezelők feladata csak az, hogy a KÉSZ bitet magasra állítsák.

A funkció meghívásakor a beviteli pufferekben lévő adatokat törli, így ezek a karakterek elvesznek.

A 07(h) funkció hívása (beviteli pufferek törlése)

Bemeneti paraméter:

IBF + 2	bájt	Parancskód = 7

Visszatérési paraméter:

IBF + 3	szó	Visszatérési státusz-szó

08 – 08(h) funkció: **Írás**

Az írás funkció – amelyet mind a karakter-, mind a blokk-kezelő eszközmeghajtók használnak – egy megadott pufferből adatokat ír a megadott eszközre. Ha az írás során hiba történt, akkor a funkciónak azon túlmenően, hogy a visszatérési státusz-szóban a hibajelző bitet magasra kell állítania, még a hiba nélkül kiírt bájtok vagy szektorok számát is vissza kell adnia az IBF-ben.

A DOS 3.0 verziójától felfelé ez a funkció a nyitó és záró funkciók (0D(h) és 0E(h)) által beállított nyitott állományok számának és az adathordozó leíró bájtjának a segítségével azt is meg tudja határozni, hogy történt-e nem megengedett lemezcsere.

A 08(h) funkció hívása (írás)

Bemeneti paraméterek:

IBF + 1	bájt	Egység kódja (ha blokk-kezelő)
IBF + 2	bájt	Parancskód = 8
IBF + 13	bájt	Adathordozó leíró bájtja
IBF + 14	mutató	Átviteli tárterület címe
IBF + 18	szó	Írandó bájtok/szektorok száma
IBF + 20	szó	Kezdő szektor száma (ha blokk-kezelő)

Visszatérési paraméterek:

IBF + 3	szó	Visszatérési státusz-szó
IBF + 18	szó	Kiírt bájtok/szektorok száma
IBF + 22	mutató	Mutató a kötetcímkére, ha a visszatérési hibakód 0F(h) (3.0 verziótól felfelé).

09 – 09(h) funkció: **Írás verifikálással**

Ez a funkció meghívását és feladatát tekintve megegyezik a 08(h) funkcióval azzal a különbséggel, hogy az adatok kiírását követően még egy olvasási műveletet is végrehajt, és így ellenőrzi a kiírt adatok helyességét.

10 – 0A(h) funkció: **Kiviteli állapot vizsgálata**

Ezt a funkciót is csak a karakterkezelő eszközmeghajtóknak kell támogatniuk. A blokk-kezelők feladata mindössze annyi, hogy a KÉSZ bitet magasra állítsák.

Ez a funkció az eszközmeghajtó aktuális kiviteli állapotával tér vissza, lehetővé téve, hogy a DOS megvizsgálja, vannak-e még karakterek az eszközmeghajtó kiviteli pufferében. A DOS e funkció eredményéből állapítja meg, hogy hívhatja-e ismét az írás funkciót.

A 0A(h) funkció hívása (kiviteli állapot vizsgálata)

Bemeneti paraméter:

IBF + 2	bájt	Parancskód = 10

Visszatérési paraméter:

IBF + 3	szó	Visszatérési státusz-szó: Ha a FOGLALT bit = 1, akkor az írás funkció hívása addig vár, míg az előző funkció befejezi a munkáját. Ha a FOGLALT bit = 0, akkor azonnal hívható az írás funkció.

11 – 0B(h) funkció: Kiviteli pufferek törlése

Ezt a funkciót is csak a karakterkezelő eszközmeghajtóknak kell támogatniuk. A blokk-kezelők feladata csak az, hogy a KÉSZ bitet magasra állítsák.

A funkció a meghívásakor a kiviteli pufferekben lévő adatokat törli, így ezek a karakterek elvesznek. A funkció az aktuális kiviteli kéréseket figyelmen kívül hagyja.

A 0B(h) funkció hívása (kiviteli pufferek törlése)

Bemeneti paraméter:

IBF + 2	bájt	Parancskód = 11

Visszatérési paraméter:

IBF + 3	szó	Visszatérési státusz-szó

12 – 0C(h) funkció: IOCTL írás

Az IOCTL írás funkció – amelyet mind a karakter-, mind a blokk-kezelő eszközmeghajtók használnak – lehetővé teszi, hogy egy felhasználói program közvetlenül küldhessen vezérlő információkat egy eszközmeghajtóhoz (pl. beállíthatja a baudráta értékét a soros interfészen). A funkció a 21(h) megszakítás 44(h) funkcióján keresztül akkor hívható meg, ha az eszközmeghajtó attribútum szavában lévő IOCTL bit (14. bit) magas.

A 0C(h) funkció hívása (IOCTL írás)

Bemeneti paraméterek:

IBF + 1	bájt	Egység kódja (ha blokk-kezelő)
IBF + 2	bájt	Parancskód = 12
IBF + 13	bájt	Adathordozó leíró bájtja

IBF + 14	mutató	Átviteli tárterület címe
IBF + 18	szó	Írandó bájtok/szektorok száma
IBF + 20	szó	Kezdő szektor száma (ha blokk-kezelő)

Visszatérési paraméterek:

IBF + 3	szó	Visszatérési státusz-szó
IBF + 18	szó	Kiírt bájtok/szektorok száma

13 – 0D(h) funkció: Eszköz megnyitása

Ezt a funkciót csak a DOS 3.O és későbbi verziói támogatják, és csak akkor hívják meg, ha az eszközmeghajtó fejlécében lévő attribútum szó 11. bitje (az ún. OCR bit, open/close/removable bit) magas.

Blokk-kezelő eszközmeghajtó esetén a DOS ezt a rutint minden egyes állomány-megnyitáskor meghívja. Így a rutin segítségével megállapíthatja, hány állomány van nyitva azon az egységen, amelynek kódját a rutin meghívásakor megkapta. E lehetőség kihasználásakor azonban óvatosnak kell lennünk, mert azok a programok, amelyek FCB-ken keresztül fordulnak állományokhoz, hajlamosak arra, hogy a megnyitott állományokat nem zárják le, amiből következik, hogy a nyitott állományokat számláló szám nem lesz igaz. Ezzel a veszéllyel szemben úgy lehet védekezni, hogy minden olyan esetben, amikor az adathordozó cseréjét kérdező funkció igen válasszal tér vissza, és a DOS ezt követően meghívja a BPB-t létrehozó funkciót, akkor az eszközmeghajtó ezt a számot nullázza anélkül, hogy kiürítené a puffereket.

Karakterkezelő eszközmeghajtónál a funkcióval pl. egy inicializáló karakterlánc küldhető a nyomtatóra. Ez a karakterlánc felhasználói programból adható át az eszközmeghajtónak a 21(h) DOS megszakítás IOCTL írás alfunkciójának segítségével. A funkcióval az is megakadályozható, hogy az eszközmeghajtót egyidejűleg egynél több művelet vegye igénybe (pl. hálózatban), és ezzel egymás munkáját zavarják.

Ezt a funkciót a CON, AUX és a PRN egységek megnyitásához természetesen nem kell meghívni, hiszen azok mindig nyitva vannak.

A 0D(h) funkció hívása (Eszköz megnyitása)

Bemeneti paraméterek:

IBF + 1	bájt	Egység kódja (ha blokk-kezelő)
IBF + 2	bájt	Parancskód = 13

Visszatérési paraméter:

IBF + 3	szó	Visszatérési státusz-szó

14 – 0E(h) funkció: Eszköz lezárása

Ezt a funkciót csak a DOS 3.O és későbbi verziói támogatják, és csak akkor hívják meg, ha az eszközmeghajtó fejlécében lévő attribútum szó 11. bitje (az ún. OCR bit, open/close/removable bit) magas.

Blokk-kezelő eszközmeghajtó esetén a DOS ezt a rutint minden egyes állomány lezárásakor meghívja, és a nyitott állományokat nyilvántartó szám értékét 1-gyel csökkenti. Így egyszerűen megállapíthatja, hány állomány van nyitva azon az egységen, amelynek kódját a rutin meghívásakor megkapta. Amikor ennek a számnak az értéke eléri a 0-t, akkor ez azt jelenti, hogy nincs már nyitott állomány, és az eszközmeghajtónak ki kell ürítenie a belső puffereket, mert a felhasználó esetleg lemezt akar cserélni.

Karakterkezelő eszközmeghajtónál a funkcióval pl. egy lezáró karakter vagy karakterlánc küldhető a nyomtatóra. Egy szöveg kinyomtatása után például így dobhatunk lapot. Ez a karakter(lánc) most is felhasználói programból adható át az eszközmeghajtónak a 21(h) DOS megszakítás IOCTL írás alfunkciójának segítségével.

A 0E(h) funkció hívása (eszköz lezárása)

Bemeneti paraméterek:

IBF + 1	bájt	Egység kódja (ha blokk-kezelő)
IBF + 2	bájt	Parancskód = 14

Visszatérési paraméter:

IBF + 3	szó	Visszatérési státusz-szó

15 – 0F(h) funkció: Cserélhető adathordozó

Ezt a funkciót csak a DOS 3.O és későbbi verziói támogatják, és csak blokk-kezelő eszközmeghajtókon. A karakterkezelő eszközmeghajtóknak csak annyi a feladatuk, hogy a KÉSZ bitet magasra állítsák. A DOS a funkciót csak akkor hívja meg, ha az eszközmeghajtó fejlécében lévő attribútum szó 11. bitje (az ún. OCR bit, open/close/removable bit) magas.

A 0F(h) funkció hívása (cserélhető adathordozó)

Bemeneti paraméterek:

IBF + 1	bájt	Egység kódja
IBF + 2	bájt	Parancskód = 15

Visszatérési paraméter:

IBF + 3	szó	Visszatérési státusz-szó: Ha a FOGLALT bit = 1, akkor az adathordozó nem cserélhető. Ha a FOGLALT bit = 0, akkor az adathordozó cserélhető.

16 – 10(h) funkció: **Kivitel míg nem foglalt**

Ezt a funkciót csak a DOS 3.0 és későbbi verziói támogatják, és csak karakterkezelő eszközmeghajtókon. A blokk-kezelő eszközmeghajtóknak csak annyi a feladatuk, hogy a KÉSZ bitet magasra állítsák. A DOS a funkciót csak akkor hívja meg, ha az eszközmeghajtó fejlécében lévő attribútum szó 13. bitje magas.

A funkció egy pufferből adatokat továbbít egy egységre mindaddig, amig az egység nem foglalt, vagyis amig képes karakterek fogadására. E tulajdonsága következtében lehet ezzel a funkcióval az ún. spooling eljárást megvalósítani: a „háttérben" állományokat lehet nyomtatóra küldeni, miközben az „előtérben" valamilyen program fut a számítógépen. Nem tekintendő a funkció hibás végrehajtásának az, ha visszatérésekor azt jelzi, hogy az igényeltnél kevesebb bájtot továbbított. Ha az átvitel során az egység jelzi, hogy további karaktereket már nem képes fogadni, akkor az addig az időpontig átvitt bájtok számát visszaírja a meghívó funkció IBF-jébe.

A 10(h) funkció hívása (kivitel míg nem foglalt)

Bemeneti paraméterek:

IBF + 2	bájt	Parancskód = 16
IBF + 14	mutató	Átviteli tárterület címe
IBF + 18	szó	Kiírandó bájtok száma

Visszatérési paraméterek:

IBF + 3	szó	Visszatérési státusz-szó
IBF + 18	szó	Átvitt bájtok száma

6.11.5. Órameghajtó

Az eszközmeghajtó programok közül speciális feladatot lát el az ún. órameghajtó program (angol nevén clock driver). A program olyan, karakterkezelő eszközmeghajtó, amelynek az az egyedüli feladata, hogy a DOS által az FCB-kben és a katalógusbejegyzésekben használt dátumot és időpontot az órameghajtóról beolvassa vagy azon beállítsa. Ebből a célból a DOS az indításakor betölt egy CLOCK$ nevű óravezérlőt. A programnak ettől eltérő neve is lehet, mivel a DOS az óraegységet nem a neve alapján, hanem aszerint azonosítja, hogy a fejlécé-

nek attribútum szavában lévő 3. bit (CLOCK$ egység) magasra van állítva. Mivel az órameghajtó karakterkezelő program, a 15. bitnek is magasnak kell lennie. Az óravezérlőt a DOS 21(h) megszakításának 2A(h) – 2D(h) funkciói hívják meg, amelyeken keresztül a dátum és az idő beolvasható, ill. beállítható. Az óravezérlőnek ehhez a 0-s, inicializáló IOCTL-funkció mellett csak a 4-es és a 8-as funkciókat kell támogatnia. A 4-es, olvasási funkció az óravezérlőből a dátumot és az időt beolvassa a DOS-ba (lekérdezés), míg a 8-as, írási funkció ezeket átveszi a DOS-ból, és beírja az óravezérlőbe (beállítás). A dátumot és az időt mindkét funkció egy 6 bájt hosszú pufferben adja át. A puffer címét az IBF 0E(h) ofszetcímén lévő mutató tartalmazza (27. ábra). Az óravezérlő adatait tartalmazó puffer szerkezetét a 33. ábra mutatja:

00(h)	1980. jan. 1-jétől eltelt napok száma	2 bájt
02(h)	percek	1 bájt
03(h)	órák	1 bájt
04(h)	századmásodpercek	1 bájt
05(h)	másodpercek	1 bájt

33. ábra: Az óravezérlő pufferének szerkezete

A puffer a dátumot nem évek, hónapok és napok bontásban, hanem az 1980. január 1. óta eltelt napok számaként adja át, 16 biten. Ennek a számnak az évekre, hónapokra és napokra való visszafejtéséhez egy meglehetősen bonyolult képletet kell használni, hiszen a szökőéveket is figyelembe kell venni.

Az óravezérlő az idő olvasásához és beállításához általában a BIOS 1A(h) megszakítás 00(h) és 01(h) funkcióit használja. Ezekkel a funkciókkal azonban csak a szoftveroldalról vezérelt időszámlálóhoz lehet hozzáférni, ami azt jelenti, hogy az óra csak a számítógép bekapcsolásának (pontosabban szólva az óravezérlő installálásának) pillanatától kezdődően kezd el „ketyegni", és a gép kikapcsolásakor elveszti a tartalmát. Mint ismeretes, az AT gépek már tartalmaznak beépített, telepről táplált elektronikus órát. Ez az óra a gép kikapcsolása után is „ketyeg", azaz az újbóli bekapcsoláskor – amennyiben előzőleg pontosan volt beállítva – a tényleges időt mutatja, és az órameghajtó már ezzel az ún. valós idővel tud dolgozni. (Ezeket az órákat valós idejű óráknak nevezik. Angol nevük real time clock, rövidítve RTC). A valós idejű órát lekérdező, ill. beállító óravezérlőkhöz a DOS a BIOS 1A(h) megszakítás 02(h) – 05(h) funkcióin keresztül fér hozzá. Ugyanezen megszakítás 06(h) és 07(h) funkcióival még egy előre megadott (riasztási) időpont is beállítható, ill. törölhető. Az idő mérésére ebben az esetben is a valós idejű óra szolgál, úgyhogy erre is csak az AT típusú gépeknél van lehetőség.

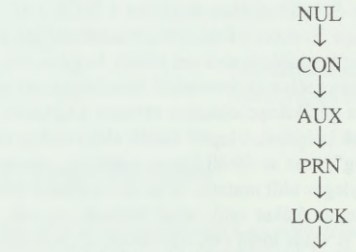
6.11.6. Az eszközmeghajtók installálása

Miután az előzőekben részletesen megismerkedtünk az eszközmeghajtók felépítésével és azokkal a funkciókkal, amelyek végrehajtását a DOS elvárja tőlük, tekintsük át azt a folyamatot, amely az operációs rendszer indulásával kezdődik, és amely során az operációs rendszer az installálandó eszközmeghajtót mintegy a saját részévé teszi.

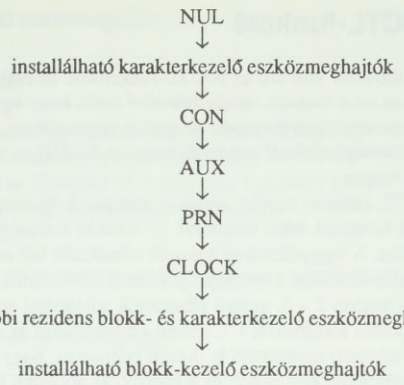
Feltételezzük, hogy előzőleg már elkészült egy (vagy több) eszközmeghajtó program, amelye(ke)t a CONFIG.SYS állomány tartalmaz. (Megjegyezzük, hogy a DEVICE=eszközmeghajtó kijelölés egység és útvonal specifikációt is tartalmazhat, vagyis az eszközmeghajtó programnak nem feltétlenül az aktuális egységben és a főkatalógusban kell lennie.)

Ha most bekapcsoljuk a számítógépet, akkor megkezdődik az operációs rendszer betöltése. Amint erről a 2.3. alfejezetben már volt szó, az IBMBIO.COM első modulja, a DOS-BIOS – többek között – betölti a rezidens eszközmeghajtókat (NUL, CON, AUX, PRN, CLOCK valamint a hajlékony- és merevlemezes egységek meghajtói) úgy, hogy ezek a tárban egymást követik, és sorban a következőre mutatva összefüggő láncot alkotnak. Ezt követően a SYSINIT (amely az IBMBIO.COM második modulja), beolvassa és feldolgozza a CONFIG.SYS állományt. Az ebben megnevezett első (majd második stb.) eszközmeghajtót betölti a tárba, és megvizsgálja a fejlécét. Ha azt állapítja meg, hogy ez karakterkezelő meghajtó, akkor ezt a láncban lévő többi karakterkezelő meghajtó elé (közvetlenül a NUL után) kapcsolja; ha blokk-kezelő meghajtó, akkor az összes, előzőleg installált valamint rezidens blokk-kezelő után, vagyis a lánc végére helyezi el. Ezt követően aktualizálja a láncot oly módon, hogy mindegyik eszközmeghajtó fejlécébe beírja a láncban őt követő eszközmeghajtó ofszet- és szegmenscímét. A láncban utolsó eszközmeghajtó fejlécébe csatoló címként –1, –1 (FFFF(h)) kerül.

A 34 – 35. ábrán eszközmeghajtók láncát mutatjuk be. A 34. ábra az installálható eszközmeghajtók betöltése előtti, a 35. ábra a betöltés utáni elrendezést szemlélteti.



34. ábra: Eszközmeghajtók lánca az installálható eszközmeghajtók betöltése előtt



35. ábra: Eszközmeghajtók lánca az installálható eszközmeghajtók betöltése után

Most a SYSINIT meghívja az eszközmeghajtó stratégia rutinját, és átadja részére azt a mutatót, amely az IBF nevű adatszerkezetre mutat. Az IBF-ben ekkor a parancskód értéke 0. Ezután a SYSINIT meghívja az eszközmeghajtó megszakítási rutinját, amely az IBF-ben lévő 0 parancskódnak megfelelően végrehajtja az inicializáló funkciót. A funkció visszatérésekor visszaírja az IBF-be az eszközmeghajtó által elfoglalt RAM terület utáni első szabad címet (break address), hogy a DOS tudja, hová helyezheti a láncban a következő eszközmeghajtót. Ezek után a DOS rátérhet a CONFIG.SYS-ben specifikált következő eszközmeghajtó installálására.

Az eszközmeghajtók installálásával kapcsolatban ismét megemlítjük a karakter- és a blokk-kezelő eszközmeghajtók közötti számos különbség egyikét: az operációs rendszer beépített blokk-kezelő eszközmeghajtói az installálható blokk-kezelő eszközmeghajtókkal nem írhatók felül, hanem csak kiegészíthetők. Ettől eltérő a helyzet a karakterkezelő eszközmeghajtók esetén: a rendszer alapértelmezés szerinti karakterkezelő eszközmeghajtói (mint pl. a CON) felülírható, egészen pontosan „kikerülhető" a felhasználó által írt, installálható eszközmeghajtókkal, amennyiben ezeknek a fejlécében ugyanazt a logikai nevet adjuk meg, mint ami a „kikerülendő" eszközmeghajtó eredeti neve. Ezt a kikerülést az teszi lehetővé, hogy a DOS egy karakter be- vagy kivitele iránti igény teljesítésekor minden esetben először az installált eszközmeghajtók listáját „futja át", és ha talált megfelelőt, akkor ezt hívja meg. A felülírás tehát képletesen értendő: a rendszerben benne marad az eredeti, alapértelmezés szerinti eszközmeghajtó is, de ha a felhasználó ugyanilyen néven egy másikat installált, akkor a DOS az eredetit „mellőzi". (Van ilyen...)

6.11.7. Az IOCTL-funkció

Az előzőekben már többször volt szó az IOCTL-funkcióról. Itt csak a rend kedvéért jegyezzük meg, hogy ez az a funkció, amely lehetővé teszi, hogy egy felhasználói programból közvetlen kapcsolatot tarthassunk az eszközmeghajtókkal. Erre természetesen csak azoknál az eszközmeghajtóknál van mód, amelyek fejlécében az attribútum szó 14. bitje (az IOCTL-bit) magas.

Maga az IOCTL-funkció egyike annak a számos funkciónak, amelyet a DOS 21(h) megszakításán keresztül lehet meghívni. A funkció száma 44(h), és egy sor alfunkciót foglal magába. A leggyakrabban használt alfunkciók három csoportba sorolhatók: a 0 – 1. számú alfunkciókkal a meghajtó jellemzői kérdezhetők le, ill. állíthatók be, a második csoportba tartozó 2 – 5. számú alfunkciók adatátvitel lebonyolítását végzik, míg a harmadik csoportba sorolható 6 – 7. számú alfunkciókkal az eszközmeghajtó be-, ill. kiviteli állapota (státusza) kérdezhető le. Közös bennük az, hogy az alfunkció számát mindenkor az AL regiszterbe kell beírni, és a funkció az átvitelbit értékével jelzi a művelet végrehajtásának sikerét. Ha a bit értéke 1, akkor a művelet rendben megtörtént, ellenkező esetben valamilyen hiba történt, és a hiba kódja az AX regiszterbe kerül.

Nézzük meg először azokat az alfunkciókat, amelyek egy eszközmeghajtó jellemzőivel kapcsolatosak. (Felhívjuk a figyelmet arra, hogy az a 16 bites szó, amely ezeket a jellemzőket tartalmazza, az igen erős hasonlóság ellenére sem egyezik meg az eszközmeghajtó fejlécében lévő attribútum szóval, ezért ezek egymástól szigorúan megkülönböztetendők!)

Az eszközmeghajtó jellemzői (az angol nyelvű DOS-irodalom erre a „device information" elnevezést használja) a 0. számú alfunkcióval kérdezhetők le, és az 1. számúval állíthatók be. Meghívásukhoz a funkció számát (44(h)) az AH, az alfunkció számát az AL regiszterbe, a meghajtóhoz tartozó handle számát pedig a BX regiszterbe kell beírni. A meghajtó jellemzőit a 0. számú, lekérdező alfunkció a DX regiszterben adja vissza. Ezt követően a jellemzők már megváltoztathatók, és az 1. számú alfunkció meghívásával visszaadhatók a meghajtónak. Egyedül ez az alfunkció teszi lehetővé, hogy az eszközmeghajtót a bináris és az ASCII üzemmód között átkapcsolhassuk. A bináris üzemmódba való átkapcsolással az adatátvitel felgyorsítható: ekkor ugyanis a DOS nem „vesztegeti" az idejét azzal, hogy megvizsgálja, van-e az adatfolyamban speciális vezérlő karakter (mint pl. Ctrl-C, Ctrl-S stb.), hanem válogatás nélkül mindent elfogad.

Az eszközmeghajtó jellemzőit tartalmazó szó felépítését és az egyes bitek jelentését a 36. ábra mutatja.

Ha blokk-kezelő eszközmeghajtó:

Bit	Jelentése
0 – 5.	a lemezmeghajtó azonosítója (0 = A, 1 = B stb.)
6.	= 0, ha az állomány írása befejeződött
	= 1, ha az állomány írása még nem fejeződött be
7.	= 0 jelzi, hogy blokk-kezelő meghajtó
8 – 15.	foglalt

Ha karakterkezelő eszközmeghajtó:

Bit	Jelentése
0.	= 1, ha az aktuális standard beviteli eszköz (stdin)
1.	= 1, ha az aktuális standard kiviteli eszköz (stdout)
2.	= 1, ha az aktuális nemlétező (NUL) eszköz
3.	= 1, ha az aktuális órameghajtó (CLOCK$) eszköz
4.	foglalt
5.	= 0, ha a meghajtó ASCII üzemmódban dolgozik
	= 1, ha a meghajtó bináris üzemmódban dolgozik
6.	= 0 lekérdezéskor, ha a bemeneten az állomány vége, és beállításkor minden esetben
7.	= 1 jelzi, hogy karakterkezelő meghajtó
8 – 13.	foglalt
14.	= 1, ha az IOCTL írás/olvasást támogatja (IOCTL bit) (ez a bit csak olvasható, és beállításkor 0-nak kell lennie)
15.	foglalt

36. ábra: Az eszközmeghajtó jellemzőit tartalmazó szó

Nézzük meg most az adatátvitel lebonyolítására alkalmas alfunkciókat. A 2 – 3. alfunkciók karakterkezelő meghajtókkal kapcsolatosak. A 2. számú alfunkcióval a meghajtóról előre megadott számú karakter olvasható be a meghívó program által kijelölt pufferbe. A handle-t a BX, a beolvasandó bájtok számát a CX regiszterbe, a puffer címét pedig a DS:DX regiszterpárba kell beírni. Visszatéréskor a funkció a beolvasott bájtok számát az AX regiszterben adja vissza. A 3. számú funkció az előzőnek az ellentetje: azonos regiszterkiosztás mellett egy kijelölt pufferből adott számú karaktert továbbít az eszközmeghajtóra.

Ugyanígy hívhatók a 4 – 5. számú alfunkciók azzal a különbséggel, hogy ezek blokk-kezelő meghajtókkal kapcsolatosak. Ennek megfelelően ezeknél a BX regiszterbe nem a handle-t, hanem a BL regiszterbe a lemezmeghajtó azonosító számát kell beírni (0 = A, 1 = B stb.).

A 6 – 7. számú alfunkciókkal az eszközmeghajtók állapota kérdezhető le. A 6. számúval az állapítható meg, hogy készen áll-e a meghajtó adatok fogadására, a 7. pedig a meghajtó kiviteli állapotáról, vagyis arról ad tájékoztatást, hogy kész-e adatok küldésére. Mindkét alfunkció részére a BX regiszterben át kell adni a meghajtóhoz tartozó handle-t. Ha tehát egy meghajtó állapotát akarjuk lekérdezni, akkor ehhez először a megfelelő funkcióval meg kell nyitni egy handle-t. A megnyitáshoz állománynévként a megszólítandó eszközmeghajtó logikai nevét, ill. amennyiben a handle egy állományhoz kapcsolódik, akkor az állomány nevét kell megadni. A standard be- és kiviteli egységek esetén a handle megnyitására természetesen nincs szükség, hiszen ezek mindig nyitva vannak. Az alfunkciók visszatéréskor az FF(h) értéket írják az AL regiszterbe, ha a meghajtó készenlétben áll. A 0 érték azt jelenti, hogy a meghajtó még foglalt.

További lehetőséget ad az eszközmeghajtókkal való közvetlen kapcsolattartásra a 25(h) és a 26(h) megszakítás, amelyek azonban csak blokk-kezelőkkel kapcsolatban használhatók. Ezek segítségével egy lemez szektorai közvetlenül olvashatók (25(h) megszakítás), ill. írhatók (26(h) megszakítás). A DOS a megszakítás hívásakor az átadott szektorszámot nem módosítja, úgyhogy a hozzáférés pontosan a megadott szektorra történik. Ezekkel a funkciókkal lehet a lemez ún. állományelhelyezési tábláját (FAT) és katalógusát olvasni, ill. módosítani. A megszakítások meghívásához az AL regiszterbe a lemezmeghajtó azonosító számát (0 = A, 1 = B stb.), a CX regiszterbe az olvasni vagy írni kívánt szektorok számát, a DX regiszterbe az első logikai szektor számát, végül a DS:BX regiszterpárba egy puffer címét kell beírni. A megszakítások sikeres végrehajtását az átvitelbit 0 értéke jelzi. Ellenkező esetben a bit magas, és a hiba kódja az AX regiszterbe kerül.

6.11.8. Programpélda

A blokk-kezelő eszközmeghajtók egyik jellemző példája az operatív tárban létrehozott virtuális lemezt vezérlő meghajtó. Egy ilyen meghajtóprogramot mutat be példánk.

A program egy 160 kilobájtos lemezt hajt meg. Ez az érték természetesen a számítógépben rendelkezésre álló operatív tár méretének függvényében megnövelhető. Az AT típusú gépeknél az 1 MB feletti tartományban is installálható a virtuális lemez, ebben az esetben a lemez és a DOS közötti adatforgalmat a 15(h) megszakítás 87(h) funkciójával kell megszervezni.

A meghajtóprogramban elhelyezett magyarázatok elegendőnek tűnnek a megértéshez, néhány kulcsfontosságú szempontra, ill. eljárásra azonban részletesebben is kitérünk.

Az inicializáló eljárás első lépésben a 21(h) megszakítás 30(h) funkciójával megállapítja az operációs rendszer verzióját. Amennyiben legalább 3.0 a verziószám, a DOS által átadott adatblokkban szereplő adat alapján beilleszti a sikeres installálást jelző üze-

netbe a létrehozandó lemez betűjelét. A következő lépés a lemezterület vége címének meghatározása. Mivel a lemez adatterülete közvetlenül a meghajtó utolsó eljárása után kezdődik, ez a cím a meghajtó végének címéből számítható. Átadásra kerül a DOS-nak a lemez formátumát leíró BPB címét tartalmazó változó (bpb_cim) címe. Esetünkben a lemezen egy szektor 512 bájtot tartalmaz, csak egy foglalt (betöltő) szektor és egy FAT van, a főkatalógusban legfeljebb 64 bejegyzés szerepelhet, összesen 320 szektor hozza létre a 160 kilobájt tárolókapacitást, az anyaghordozó leíró bájt pedig FE(h) (40 sáv, sávonként 8 szektorral).

A lemez formálása során az első lépés a betöltő szektor létrehozása, amelynek a tárban kialakított lemez esetén csupán az első 15 szava bír a DOS számára jelentőséggel. Ebben az esetben félrevezető a „betöltő" szektor név, hiszen a memórialemezről nyilván nem lehet rendszert tölteni. A formálás következő lépése a FAT létrehozása a második szektorban. Az anyaghordozó leíró bájton kívül csupa 0 értékkel kell feltölteni, hiszen induláskor egyetlen foglalt klaszter sincs. A formálás a főkatalógus létrehozásával ér véget.

A lemezzel kapcsolatos műveletek közül a két legfontosabb az írás és az olvasás. Ezen műveletek mögött a MOZGAT eljárás áll, amely mind az olvasás, mind az írás során adateltolást végez. Az írást, ill. olvasást kezdeményező eljárások csupán a BP regiszter tartalmát készítik elő az adateltolás számára. A regiszter az adatmozgatás irányát definiálja. Az írás verifikálás nélkül történik, mivel klasszikus értelemben vett lemezhiba a tárban kialakított lemezen nem léphet fel. A MOZGAT eljárás az összes többi adatot, információt a DOS által átadott adatblokkból nyeri.

```
;
;   ┌──────────────────────────────────────────────────────────┐
;   │                                                          │
;   │                    M E M D I S K . A S M                 │
;   │                                                          │
;   │  Funkciója : 160 KB-os memórialemez (RAM - disk) meghajtó.
;   │                                                          │
;   │  FONTOS    : a létrejövő MEMDISK.EXE futtatható programból az
;   │              EXE2BIN programmal a MEMDISK.SYS programot kell
;   │              létrehozni, azt a C:\ katalógusba kell átmásolni,
;   │              a CONFIG.SYS állományba pedig be kell szúrni a
;   │              következő sort :
;   │                                                          │
;   │              DEVICE = MEMDISK.SYS
;   │                                                          │
;   └──────────────────────────────────────────────────────────┘
;

;— KÓD ------------------------------------------------------------

kod     segment

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

        org 0                       ; A programnak nincs PSP-je, ezért
                                    ; a 0-as offszeten kezdődik.

;— Konstansok -----------------------------------------------------

                                    ; Adatblokkbeli címek :

parancs equ 2                       ;     parancsmező
allapot equ 3                       ;     állapotmező
eszkoz  equ 13                      ;     eszközszám
lemcser equ 14                      ;     lemezcsere
veg_cim equ 14                      ;     meghajtó végcím
puf_cim equ 14                      ;     puffercím címe az adatblokkban
funszam equ 16                      ;     16 műveletre képes a meghajtó
szamlal equ 18                      ;     számláló
bpb_cim equ 18                      ;     BPB cím
szektor equ 20                      ;     első szektorszám címe
eszknev equ 22                      ;     eszköznév cím

CR      equ 13                      ; Kocsivissza ASCII kódja.
LF      equ 10                      ; Soremelés ASCII kódja.
KVEG    equ "$"                     ; Karakterlánc vége jelzés.

;— Adatok ---------------------------------------------------------

firstb  equ this byte               ; A meghajtó első bájtja.

;— A meghajtó fejrésze --------------------------------------------

        dw -1,-1                    ; Kapcsolat a következő meghajtóval.
        dw 0100100000000000b        ; Meghajtó állapot.
        dw offset strat             ; Mutató a stratégia rutinra.
        dw offset msz               ; Mutató a meghajtó rutinra.
        db 1                        ; 1 eszköz.
        db 7 dup (0)

;— Az egyes műveletek ugrótáblázata -------------------------------

                                    ; Műv.kód Művelet

muv_tab dw offset init              ;    0    Inicializálás.
        dw offset med_test          ;    1    Adathordozó vizsgálata.
```

```
        dw offset get_bpb       ;   2     BPB létrehozása.
        dw offset olvas         ;   3     IOCTL olvasás.
        dw offset olvas         ;   4     Olvasás.
        dw offset dummy         ;   5     Megtartó olvasás.
        dw offset dummy         ;   6     Bemeneti státusz.
        dw offset dummy         ;   7     Bemeneti puffer törlése.
        dw offset iras          ;   8     írás.
        dw offset iras          ;   9     írás verifikálással.
        dw offset dummy         ;  10     Kimeneti állapot vizsgálat.
        dw offset dummy         ;  11     Kimeneti pufferek törlése.
        dw offset iras          ;  12     IOCTL írás.
        dw offset dummy         ;  13     Eszköz megnyitása.
        dw offset dummy         ;  14     Eszköz lezárása.
        dw offset no_rem        ;  15     Cserélhető adathordozó.
        dw offset iras          ;  16     írás, amíg foglalt.

ab_cim  dw (?),(?)              ; Az átadott adatblokk címe.
ml_szeg dw (?)                  ; A memórialemez szegmense.

bpb_off dw offset bpb,(?)       ; A BPB címe.

bootszek db 3 dup (0)           ; Itt lenne az ugrás a betöltő
                                ; rutinra.

        db "XXXX n.n"           ; Gyártónév és verziószám.
bpb     dw 512                  ; 512 bájtos szektorok.
        db 1                    ; 1 szektoros klaszterek.
        dw 1                    ; 1 foglalt betöltő szektor.
        db 1                    ; 1 FAT.
        dw 64                   ; Max. 64 hely a gyökérkatalógusban.
        dw 320                  ; összesen 320 szektor = 160 KB.
        db 0FEh                 ; Anyaghordozó leíró bájt
                                ; (1 oldal, 40 sáv, 8 szektor).
        dw 1                    ; Minden FAT egy szektort foglal el.

kot_nev db "--MEMORIA--"        ; A kötetazonosító.
        db 8                    ; A kötetazonosító attribútum bájtja.

;— A meghajtó rutinjai és funkciói ————————————————————————————

strat   proc far                ; Stratégia rutin.

        mov  cs:ab_cim,bx       ; Az adatblokk címének eltárolása.
        mov  cs:ab_cim+2,es

        ret                     ; Visszatérés a hívóhoz.

strat   endp

;————————————————————————————————————————————————————————————

msz     proc far                ; Megszakítás rutin.

        push ax                 ; Regiszterek tartalmának mentése.
        push bx
        push cx
        push dx
        push di
        push si
        push bp
        push ds
        push es
```

```
                pushf                   ; Az állapotjelző regiszter is
                                        ; elmentődik.
                push cs                 ; Adatszegmens regiszter beállítása,
                pop  ds                 ; a kódszegmens megegyezik az adat-
                                        ; szegmenssel.

                les  di,dword ptr ab_cim; Az adatblokk címe ES:DI-be.
                mov  bl,es:[di+parancs] ; Műveletkód betöltése.
                cmp  bl,funszam         ; Érvényes műveletkód ?
                jle  pk_ok              ;     Igen : folytatható.

                mov  ax,8003h           ; "Érvénytelen művelet" kóddal
                ret                     ; visszatérés a hívóhoz.

pk_ok:          shl  bl,1               ; Mutató az ugrótáblára.
                mov  bh,0
                call [muv_tab+bx]       ; A művelet meghívása.

msz_vege        label near
                push cs                 ; Adatszegmens regiszter beállítása,
                pop  ds                 ; a kódszegmens megegyezik az adat-
                                        ; szegmenssel.

                les  di,dword ptr ab_cim        ; Az adatblokk címe ES:DI-be.

                or   ax,0100h           ; Kész bit bebillentése
                mov  es:[di+allapot],ax ; és az állapotmezőbe töltése.

                popf                    ; A regiszterek tartalmának
                pop  es                 ; helyreállítása.
                pop  ds
                pop  bp
                pop  si
                pop  di
                pop  dx
                pop  cx
                pop  bx
                pop  ax

                ret                     ; Visszatérés a hívóhoz.

msz             endp

;-------------------------------------------------------------------------

init            proc near               ; Inicializáló rutin.

                ;-- A memórialemez eszköznevének megállapítása ------------

                mov  ah,30h             ; DOS verzió megállapítás funkció.
                int  21h                ; 21(h) DOS megszakítás.
                cmp  al,3
                jb   kiiras             ; 3-as verzió esetén folytatás.

                mov  al,es:[di+eszknev] ; Eszköznév betöltése,
                add  al,"A"             ; betűvé alakítása és
                mov  betu,al            ; beillesztése a kezdőszövegbe.

kiiras:         mov  dx,offset szoveg   ; Az indítási üzenet címe.
                mov  ah,9               ; Karakterlánc kiírás funkció.
                int  21h                ; 21(h) DOS megszakítás.

                ;-- A memórialemez utáni első bájt címének kiszámítása ----

                mov  word ptr es:[di+veg_cim],offset ramdisk+8000h
```

```
        mov  ax,cs                      ; A memórialemez 32 KB +
        add  ax,2000h                   ; 2 * 64 KB hosszú.
        mov  es:[di+veg_cim+2],ax
        mov  byte ptr es:[di+eszkoz],1  ; 1 eszköz
        mov  word ptr es:[di+bpb_cim],offset bpb_off    ; A BPB mutató
        mov  es:[di+bpb_cim+2],ds                       ; címe.
        mov  ax,cs                  ; A memórialemez kezdetének
                                    ; szegmenscíme.
        mov  bpb_off+2,ds           ; A BPB szegmenscíme.
        mov  dx,offset ramdisk
        mov  cl,4                   ; Szegmenscímmé alakítás.
        shr  dx,cl
        add  ax,dx                  ; A két szegmenscím összeadása és
        mov  ml_szeg,ax             ; eltárolása.

        ;— A betöltő szektor felírása ————————————————————————

        mov  es,ax                  ; Szegmenscím --> ES.
        xor  di,di                  ; A betöltő szektor az 1. bájton
                                    ; kezdődik.
        mov  si,offset bootszek     ; A betöltő szektor címe.
        mov  cx,15                  ; Az első 15 szó használatos.
        rep  movsw                  ; A betöltő szektor másolása a
                                    ; memórialemezre.

        ;— A FAT felírása ————————————————————————————————————

        mov  di,512                 ; A FAT a memórialemez 512. bájtján
                                    ; kezdődik.
        mov  al,0FEh                ; Az anyaghordozó leíró bájt a
        stosb                       ; a FAT 1. bájtjába kerül.
        mov  ax,0FFFFh              ; A FAT 2. es 3. bájtja FFh.
        stosw
        mov  cx,236                 ; A FAT további 236 szavát 0-val
        inc  ax                     ; feltölteni.
        rep  stosw

        ;— Kötetazonosító beírása a gyökérkatalógusba ————————————

        mov  di,1024                ; A gyökér a 3. szektorban kezdődik.
        mov  si,offset kot_nev      ; A kötetazonosító címe.
        mov  cx,6                   ; A kötetazonosító 6 szó hosszú.
        rep  movsw                  ; írás a memórialemezre.

        mov  cx,1017                ; A katalógus többi részének
        mov  ax,0                   ; feltöltése 0-val.
        rep  stosw

        mov  ax,0                   ; Nincs hiba.
        ret                         ; Visszatérés a hívóhoz.

init    endp

;————————————————————————————————————————————————————————————

dummy   proc near                   ; Ez a rutin semmit sem csinál.

        mov  ax,0                   ; A foglaltságjelző bit törlése.
        ret                         ; Visszatérés a hívóhoz.

dummy   endp

;————————————————————————————————————————————————————————————
```

```
med_test proc near                  ; A memórialemez nem cserélhető.

         mov  byte ptr es:[di+lemcser],1

         mov  ax,0                  ; A foglaltságjelző bit törlése.
         ret                        ; Visszatérés a hívóhoz.

med_test endp

;-------------------------------------------------------------------------

get_bpb  proc near                  ; A BPB cím átadása a DOS-nak.

         mov  word ptr es:[di+bpb_cim],offset bpb
         mov  byte ptr es:[di+bpb_cim+2],ds

         mov  ax,0                  ; A foglaltságjelző bit törlése.
         ret                        ; Visszatérés a hívóhoz.

get_bpb  endp

;-------------------------------------------------------------------------

no_rem   proc near                  ; A memórialemez nem cserélhető.

         mov  ax,0                  ; A foglaltságjelző bit törlése.
         ret                        ; Visszatérés a hívóhoz.

no_rem   endp

;-------------------------------------------------------------------------

iras     proc near

         mov  bp,0                  ; Másolás DOS --> memórialemez.
         jmp  short mozgat          ; Adatok másolása.

iras     endp

;-------------------------------------------------------------------------

olvas    proc near

         mov  bp,1                  ; Másolás memórialemez --> DOS.

olvas    endp

;-- MOZGAT: adott számú szektor mozgatása a memórialemez és DOS között
;-- BE : BP = 0 : DOS --> memórialemez (írás)
;             1 : memórialemez --> DOS (olvasás)
;-- KI : -
;-- Regiszterek : AX, BX, CX, DX, SI, DI, ES, DS és FLAGS megváltoznak

mozgat   proc near

         mov  bx,es:[di+szamlal] ; A szektorok száma.
         mov  dx,es:[di+szektor] ; Az első szektor szektorszáma.
         les  di,es:[di+puf_cim] ; A puffer címe ES:DI-be.

mozg_1:  or   bx,bx                 ; További szektorok ?
         je   vege                  ;     Nem : vége.
         mov  ax,dx                 ; Szektorszám --> AX.
         mov  cl,5                  ; A paragrafusok számának kiszámí-
```

```
          shl   ax,cl               ; tása 32-vel való szorzással, hoz-
          add   ax,cs:ml_szeg       ; záadása a memórialemez szegmens-
          mov   ds,ax               ; kezdetéhez majd DS-be másolása.
          mov   si,0                ; Ofszetcím = 0
          mov   ax,bx               ; Az olvasandó szektorok száma -> AX.
          cmp   ax,128              ;     Ha ez több, mint 128, akkor az
          jbe   mozg_2              ;     összes szektor, ha nem, akkor
          mov   ax,128              ;     128 szektor (64 KB) olvasása.
mozg_2:   sub   bx,ax               ; A már mozgatott szektorok számának
                                    ; kivonása.
          add   dx,ax               ; A most mozgatandó szektorok
                                    ; számával növelés.
          mov   ch,al               ;
          mov   cl,0                ; CX tartalmazza a szektorszámlálót.
          or    bp,bp               ; Olvasás?
          jne   mozg_3              ;     Nem: tovabb
          mov   ax,es               ; ES eltárolása AX-ben.
          push  ds                  ; ES = DS.
          pop   es                  ;
          mov   ds,ax               ; DS = ES.
          xchg  si,di               ; SI = DI, DI = SI.
mozg_3:   rep   movsw               ; Adatok a DOS-pufferbe.
          or    bp,bp               ; Olvasás?
          jne   mozg_1              ;     Nem: tovább.
          mov   ax,es               ; ES eltárolása AX-ben.
          push  ds                  ; ES = DS.
          pop   es
          mov   ds,ax               ; DS = ES.
          xchg  si,di               ; SI = DI, DI = SI.
          jmp   short mozg_1        ; További szektorok mozgatása.

vege:     xor   ax,ax               ; Nincs hiba.
          ret                       ; Visszatérés a hívóhoz.

mozgat    endp

;-- Itt kezdődik a tulajdonképpeni memórialemez ---------------------

          if ($-firstb) mod 16                    ; A paragrafushatár
          org ($-firstb) + 16 - (($-firstb) mod 16) ; biztosítása.
          endif

ramdisk   equ this byte

szoveg    db CR,LF,"160 KB-os memórialemez "
betu      db "?"
          db ": meghajtóként installálva.",CR,LF,KVEG

;-----------------------------------------------------------------------

kod       ends
          end
```

6.12. Lemezkezelés

Ahhoz, hogy a DOS a lemezen tárolt információkhoz gyorsan és biztonságosan hozzáférjen, a lemezre íráskor és olvasáskor igen szigorú rend szerint kell eljárnia. Különben honnan is tudná, hogy mi hol van, mit hova írhat, mit honnan törölhet stb.? Gondoljunk csak bele: ha egy lemezről be akarunk tölteni egy állományt, akkor a rendszernek azt a válaszát, miszerint nem találja, még csak-csak elfogadjuk (tudván, hogy erről mi tehetünk: hibásan adtuk meg az állomány nevét vagy az útvonal-kijelölést, az állományt esetleg előtte töröltük stb.), de egy olyan válaszon, hogy: „bocsánat, elfelejtettem, hová is tettem", igencsak megütköznénk. Röviden szólva: a lemezen rendnek kell lennie.

A DOS a lemezen lévő állományokról, programokról, a velük, valamint magával a lemezzel kapcsolatos adatokról pontos és naprakész (helyesebben: azonnal kész) nyilvántartást vezet, amit magára a lemezre felír. Sőt, ha szükséges, még a nyilvántartásokról is nyilvántartásokat vezet. Ezek a nyilvántartások pontosan meghatározott adatszerkezetek, amelyek úgy és addig épülnek egymásra, míg egy olyan, a DOS számára egyértelmű adatszerkezetig nem jutnak, amelytől elindulva a DOS bármely keresett (és valóban létező) adatot megtalál. (Hiába a technika: úgy látszik, a DOS-nak is kell egy kályha. A fiatalok kedvéért: az egykori tánciskolában az egykori fiatalok a kályhától kiindulva tanulták az egykori táncokat.)

Ezekről az adatszerkezetekről – így például a FAT-ról, a katalógusokról vagy az alkatalógusokról – a korábbiakban már volt szó, ezért ezek a fogalmak bizonyára ismerősek. Most ezeket vizsgáljuk meg részleteikben és összefüggéseikben.

(Előbb azonban egy kis kitérőt teszünk. Úgy gondoljuk, hogy a lemezazonosító és a lemezmeghajtó azonosítója közötti különbség tisztázására érdemes néhány mondatot fordítani.)

A lemezazonosítóról:

A DOS a felhasználó szemszögéből a – fizikailag létező – adattároló vagy -hordozó mágneslemezeket attól függetlenül, hogy ezek hajlékony- vagy merevlemezek, (sőt még a fizikailag nem létező, azaz virtuális, vagy más szóval elektronikus lemezeket is) ún. köteteknek (angolul: volume) tekinti, és ezeket a nevük, vagy más szóhasználattal a címkéjük (angolul: volume label) szerint azonosítja. Egyszerűbben: mindegyik lemeznek saját neve van (hogy még se legyen olyan egyszerű: a név megadása nem kötelező, csak ajánlott). A helyzetet tovább bonyolítja, hogy egy valamilyen névre „keresztelt" lemez több részlemezből, azaz részkötetből is állhat (ezeket a részköteteket partícióknak nevezik), de ezeknek már nem lehet saját nevük.

Összefoglalva:

– egy lemeznek lehet neve, de a megadása nem kötelező,
– a lemez állhat egy vagy több partícióból, de ezeknek nem lehet önálló nevük.

A továbbiakban a lemez és a kötet szavakat azonos értelemben használjuk.

A lemezeknek részkötetekre, azaz partíciókra való bontásának a nagy kapacitású merevlemezeknél van jelentősége. Egy nagy kapacitású lemez ugyanis már „feldarabolva" is használható: egyik része pl. a DOS, másik része a UNIX operációs rendszer felügyelete alatt futtatható programokat, ill. magukat az operációs rendszereket tartalmazhatja. A partíciókat a DOS FDISK nevű segédprogramjával lehet létrehozni, és számukat csak a lemez tároló kapacitása és az ésszerűség korlátozza.

A lemezmeghajtó azonosítóról:

A lemezmeghajtó azonosítója (angol nevén drive identifier) ahhoz a – fizikailag ugyancsak létező – eszközhöz kapcsolódik, amely az előzőekben emlegetett lemezt valójában forgatja. (Itt is kivételt képez az elektronikus lemezmeghajtó, de csak annyiban, hogy ez a tárban létrehozott lemezt természetesen csak képletesen forgatja. A lemezazonosító, sőt akár a lemeznév azonban ennek is „kijár".) Mint ismeretes, a – fizikailag is létező – lemezmeghajtók hajlékony- vagy merevlemezes meghajtók lehetnek. A DOS a hajlékonylemezes meghajtókat általában az A, ill. B betűvel, a merevlemezes meghajtót a C, ill. D betűvel azonosítja. Ezen a helyen célszerűbb lenne a hajlékonylemezes meghajtó helyett cserélhető lemezes meghajtót, a merevlemezes meghajtó helyett pedig fixen beépített lemezes meghajtót mondani: ugyanis míg a cserélhető lemezes meghajtóban a lemezek – és ezzel együtt a kötetcímkék is – cserélődhetnek, ill. változhatnak, a fixen beépített lemezes meghajtóban sem a lemez, sem a címke nem változik.

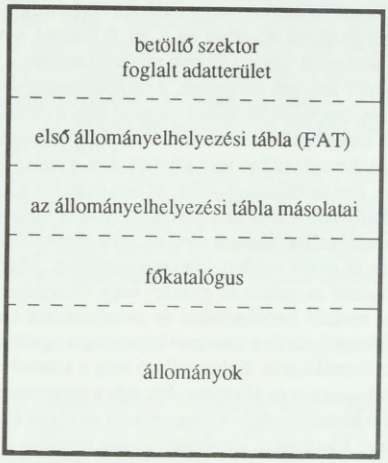
Végül is ez az, ami problémát okozhat, és ami miatt ezt a kitérőt megtettük: a fix lemez az az adathordozó, amit – még ha több részlemezre is osztunk – mindig ugyanazzal a lemezmeghajtó azonosítóval szólíthatunk meg, és egyik részlemeznek sincs saját neve.

Térjünk vissza az előbb említett adatszerkezetekhez. Minden lemeznek van kötetneve (amit nem kötelező megadni) és gyökér- vagy főkatalógusa (angol nevén root directory). Ez utóbbi további katalógusokat és természetesen magukat az állományokat tartalmazhatja. A főkatalógust és a lemezzel kapcsolatos egyéb adatszerkezeteket a DOS hozza létre a lemez formálásakor. Ekkor adható meg a kötetnév is. A főkatalóguson belüli további (al)katalógusokat és állományokat már a programozó hozza létre. A DOS a 21(h) megszakításon keresztül segíti a programozót az olyan állománykezelő műveletek elvégzésében, mint az állományok létrehozása, írása, olvasása, módosítása, törlése stb. A DOS-ba beépített lemezkezelő eszközmeghajtók lehetővé teszik, hogy a programozónak e műveletek végrehajtásakor ne kelljen azzal törődnie, a művelet hajlékony- vagy merevlemezre vonatkozik-e. Amikor a programozó egy állomány betöltését kéri, nem kell tudnia (és általában nem is tudja), hogy az a lemez mely részein, más szóval szektorain található. A DOS a megadott állománynév és a lemezen lévő információk alapján elvégzi a szükséges számításokat, és betölti azoknak a szektoroknak a tartalmát, amelyeken a kért állomány elhelyezkedik. Itt csak megemlítjük, hogy a DOS az állományokat az ún. állományelhelyezési táblában (FAT) tartja nyilván. A későbbiekben ezzel részletesen foglalkozunk.

A DOS a lemezeket formáláskor logikai szektorokra osztja, amelyeket 0-tól kezdődően folyamatosan sorszámoz. Ezeket azért nevezzük logikai szektoroknak, mert a sorszámuk szerint azonosítjuk őket, míg a fizikai szektorokat aszerint, hogy a lemezen

fizikailag hol helyezkednek el, azaz a lemez melyik oldalán, azon belül melyik sávban és mely szektorokban. A DOS az így létrehozott logikai szektorokat két fő részre osztja: a szektorok egy részét saját részére foglalja le, és itt helyezi el a lemez kezeléséhez szükséges különböző adatszerkezeteket, míg a többi szektor a programok, ill. állományok tárolására használható. Mivel az így elosztott területek mérete számos tényezőtől – pl. a lemez típusától, a DOS verziójától, a formálási paraméterektől stb. – függ, kell lennie egy olyan adatszerkezetnek, amelyet a DOS minden lemezen azonos helyen talál meg, és amely a lemezzel kapcsolatos valamennyi fontos információt tartalmazza (csak emlékeztetőül: ez az a bizonyos „kályha". Ha ezt a DOS megtalálja, indulhat a tánc). Ezt az adatterületet minden lemez 0. logikai szektora tartalmazza, és ugyancsak a lemez formálásakor íródik rá. Magát a szektort betöltő szektornak nevezik (angolul: boot sector).

A 37. ábra alapján röviden tekintsük át a DOS operációs rendszer alatt megformált lemez felépítését.



37. ábra: Egy DOS lemez felépítése

6.12.1. A betöltő szektor

A betöltő szektor onnan kapta a nevét, hogy ez az a szektor, amely az operációs rendszert a számítógép tárába tölti. Maga az operációs rendszer ugyanis nincs fixen beépítve a gépbe, hanem lemezről külön be kell tölteni. Ezért a gép bekapcsolásakor a ROM-tárba beépített BIOS rutin a gép tesztelését követően betölti a valamelyik meghajtóban lévő lemez 0. logikai szektorát, és megkezdi a 0. címen lévő program végrehajtását.

Ezért a betöltő szektor 0. címén mindig egy gépi kódú ugró utasításnak (JMP) kell lennie, amely hatására a vezérlés a betöltő rutinra kerül. Az ugrás lehet hosszú (műveleti kódja E9(h)), amelyet egy 16 bites címeltolás követ, vagy rövid (műveleti kódja E8(h)), amelyet egy 8 bites címeltolás követ. Ez utóbbi esetben a harmadik bájtra a 90(h) műveleti kódú üres utasítás (NOP) kerül (l. a 38. ábrát). Ha a BIOS rutin nem találja itt ezen ugró utasítások valamelyikét – vagy azért, mert a lemez még nem volt formálva, vagy más rendszer alatt lett formálva – akkor nem tudja végrehajtani a betöltő rutint, és természetesen az operációs rendszert sem. (Magának az ugró utasításnak a megléte persze önmagában még nem biztosíték arra, hogy a lemez DOS alatt lett formálva.)

Az ugró utasítás három bájtja után következő, 8 bájt hosszú mező a számítógép gyártójának nevét, és a formálást végző DOS verziószámát tartalmazza.

Az ezt követő harmadik, nagyobb mező az ún. BIOS paraméter blokk (BPB), amely a 0B(h) ofszetcímen kezdődik, és a DOS 2.0 verziójában a 17(h), a 3.0 verzióban pedig az 1D(h) ofszetcímig tart, és a lemezre vonatkozó legfontosabb fizikai adatokat tartalmazza. Ezeket az adatokat mind az eszközmeghajtók, mind a DOS használja: az eszközmeghajtók ezek alapján számítják át a logikai szektorszámokat fizikai szektorokra, a DOS pedig ezek alapján tudja megállapítani a különböző adatszerkezetek (FAT, főkatalógus) helyét. A BPB egyes bejegyzéseinek jelentését ugyancsak a 37. ábra mutatja. A BPB-t a verzióktól függő, különböző hosszúságú foglalt adatterület követi.

Ezután következik az ún. betöltő rutin (angol nevén bootstrap routine), amelyet a számítógép bekapcsolásakor a ROM betöltő rutinja indít el a betöltő szektor 0. ofszetcímén lévő ugró utasításon keresztül. A ROM rutinnak azonban csak annyi „esze" van, hogy a lemezmeghajtó író/olvasó fejét a kiinduló helyzetbe (a 0. sávra) pozícionálja, beolvassa az itt található első fizikai szektort a RAM meghatározott helyére, és átadja neki a vezérlést (mint láttuk, ez éppen egy ugró utasítás). A lemezen lévő betöltő rutin ennél többet „tud": kiszámítja azt a fizikai címet, ahol a lemezen az állományok elhelyezésére szolgáló terület kezdődik, és beolvassa az itt kezdődő két állományt a tárba (betölthető lemezek esetén ez a két állomány természetesen az operációs rendszert tartalmazó IBMBIO.COM és az IBMDOS.COM rendszerállomány). Ezt követően átadja a vezérlést a betöltött BIOS modulnak.

A betöltő szektor egyébként a rendszer által lefoglalt területnek csak az egyik szektora. A rendszer egynél több szektort is lefoglalhat a saját céljaira. A lefoglalt szektorok számát a BPB tartalmazza a betöltő szektor 0E(h) ofszetcímén. Az itt található érték a betöltő szektort is magában foglalja, úgyhogy az 1-es érték azt jelenti, hogy a betöltő szektoron túlmenően a lemezen nincs további foglalt szektor.

00(h)	ugró utasítás E9 xx xx vagy E8 xx 90	3 bájt
03(h)	gyártó neve és a verzió száma	8 bájt
0B(h)	bájt/szektor	2 bájt
0D(h)	szektor/klaszter	1 bájt
0E(h)	foglalt szektorok száma	2 bájt
10(h)	FAT-ok száma	1 bájt
11(h)	főkatalógus bejegyzéseinek száma	2 bájt
13(h)	szektorok száma az adathordozón	2 bájt
15(h)	adathordozó leíró bájtja	1 bájt
16(h)	egy FAT által elfoglalt szektorok száma	2 bájt
18(h)	szektor/sáv (3.0 verziótól)	2 bájt
1A(h)	író/olvasófejek száma (3.0 verziótól)	2 bájt
1C(h)	rejtett szektorok száma (3.0 verziótól)	2 bájt
1E(h)	verziótól függően foglalt adatterület	
	betöltő rutin	

BIOS paraméterblokk

38. ábra: Egy DOS lemez betöltő szektorának felépítése

6.12.2. Az állományelhelyezési tábla (FAT)

Az állományelhelyezési tábla – amit az egyszerűség és az egyértelműség kedvéért a továbbiakban az angol nevéből (file allocation table) képezett FAT rövidítésnek megfelelően mi is így fogunk nevezni – az a nyilvántartás, amely alapján a DOS nyomon tudja követni, hogy a lemezen az állományok elhelyezésére szolgáló adatterület mely szektorai foglaltak ill. üresek, továbbá azt is, hogy hol találhatók az egyes állományokat tartalmazó szektorok. (Ezek ugyanis többnyire nem egy összefüggő területen helyezkednek el, hiszen a törlés vagy bővítés következtében „szétszóródnak".)

Amikor a programozó a lemezen létrehoz vagy bővít egy állományt, akkor a DOS a lemezen lefoglalja az ehhez szükséges területet, és az állományhoz rendeli. Bár a lemezen az írás/olvasási műveletek során az adatmozgatás szektoronként történik (ez általában 512 bájtot jelent), a DOS a területkiosztást nem szektoronként, hanem ún. elhelyezési egységenként, más kifejezéssel klaszterenként (cluster) végzi. Az egy klaszterbe összefogott szektorok száma csak 2 valamely hatványa lehet, a szám pedig a betöltő szektor BPB-jében, a 0D(h) címen található. Az egy klaszterbe összefogott szektorok természetesen folyamatos logikai sorszámúak.

A különböző lemezeknél használt szektor/klaszterszámok:

Lemez	Szektor/klaszter
5.25"-os, egyoldalas hajlékonylemez	1
5.25"-os, kétoldalas hajlékonylemez	2
a PC/AT merevlemeze	4
a PC/XT merevlemeze	8

A szektoroknak klaszterekké való összefogása egyébként annak a kompromisszumnak a következménye, amelyet a rendszer tervezői a lemezterület minél gazdaságosabb kihasználása és a hozzáférési idő közötti „érdekkülönbségek" csökkentésére kötöttek. Ha ugyanis a helykiosztás szektoronként történne, akkor nem maradna feleslegesen lefoglalt, üres szektor, vagyis kedvezőbb lenne a területkihasználás, viszont nagyon megnőne a hozzáférési idő (extrém esetben egy állományhoz tartozó szektorok között még véletlenül sem lenne két szomszédos szektor, ezek „összekeresése" pedig igen sok fejmozgatást igényelne). Ha viszont a gyorsaságot emelnénk ki, akkor ésszerűtlenül nagy klaszterekkel kellene dolgozni: ekkor ugyan az egymáshoz tartozó szektorok összekeresése gyorsan menne, de egy pár soros program is ugyanakkora lemezterületet „bitorolna", mint egy nála jóval hosszabb.

A FAT mezőkre van osztva, amelyek mindegyike közvetlenül egy klaszterhez van rendelve. Az egyes mezők hossza a DOS 1.0 és 2.0 verziójában 12 bit, a 3.0 verzióban pedig a lemez méretétől függ: ha a klaszterek száma 4087-nél kisebb, akkor 12 bit, egyébként 16 bit. A DOS a mezők szélességét ismét csak a BPB-ben található adatokból tudja kiszámítani.

A FAT első két mezője (ami 12 bites mezőhossz esetén 24 bit, azaz 3 bájt, 16 bites mezőhossznál pedig 32 bit, azaz 4 bájt hosszú) mindig foglalt, és nincs köze az állományterülethez. Ennek a foglalt adatterületnek az első bájtja IBM kompatibilis lemezek esetén az adathordozónak a BPB-ben is megtalálható leíró bájtját tartalmazza. A második és a harmadik (ill. amennyiben van, akkor a negyedik) bájton mindig FF(h) áll.

Az adathordozók leíró bájtjainak jelentését a 39. ábra mutatja.

Bájt	Jelentése
0F8(h)	merevlemez
0F9(h)	5.25"-os, kétoldalas, 80 sávos, 15 szektoros hajlékonylemez (csak AT)
0FC(h)	5.25"-os, egyoldalas, 40 sávos, 9 szektoros hajlékonylemez
0FD(h)	5.25"-os, kétoldalas, 40 sávos, 9 szektoros hajlékonylemez
0FE(h)	5.25"-os, egyoldalas, 40 sávos, 8 szektoros hajlékonylemez
0FF(h)	5.25"-os, kétoldalas, 40 sávos, 8 szektoros hajlékonylemez

39. ábra: A DOS 3.0 verziójáig bezárólag támogatott adathordozók leíró bájtjai

Az eddigiek alapján egy táblázatban már összefoglalhatjuk a DOS 3.0 verziójáig támogatott hajlékonylemezek legfontosabb formálási jellemzőit (40. ábra).

DOS verziószám	1.00	1.10	2.00	2.00	3.00 (AT)
Leíró bájt	FE	FF	FC	FD	F9
Író/olvasófejek száma	1	2	1	2	2
Sáv/fej 40	40	40	40	40	80
Szektor/sáv	8	8	9	9	15
Bájt/szektor	512	512	512	512	512
Szektor/klaszter	1	2	1	2	1
Foglalt szektorok száma	1	1	1	1	1
Szektor/FAT	1	1	2	2	7
FAT-ok száma	2	2	2	2	2
Főkatalógusban lévő szektorok száma	4	7	4	7	14
Főkatalógus bejegyzéseinek száma (max.)	64	112	64	112	224
Szektorok száma (max.)	320	640	360	720	2400
Szabadon használható szektorok száma	313	630	351	708	2371
Klaszterek száma	313	315	351	354	2371
Összkapacitás	160 kb	320 kb	180 kb	360 kb	1.2 Mb
Szabadon használható kapacitás	156.5 kb	315 kb	175.5 kb	354 kb	1.185 Mb

40. ábra: A hajlékonylemezek legfontosabb formálási jellemzői

Térjünk vissza a FAT-hoz. A FAT további mezőin lévő bejegyzések arra vonatkozó információkat tartalmaznak, hogy a hozzájuk tartozó klaszterek mit reprezentálnak. Egy klaszter természetesen lehet üres, tartalmazhatja egy állomány adatait, jelezheti, hogy az állomány utolsó klasztere, végül lehet olyan, amelyet a rendszer saját céljaira foglal le (pl. az (F)FF7(h) hibás klasztert jelöl). Míg a szabad klasztert egyértelműen a (0)000(h) érték jelöli, a rendszer egy állomány utolsó klaszterének, valamint a saját célokra lefoglalt klaszterek jelölésére 8 – 8 érték használatát engedi meg, feltehetően a további bővítési lehetőségekre gondolva.

A 41. ábrán bemutatjuk a lehetséges FAT-bejegyzések értékeit és jelentésüket. (A zárójelben lévő hexadecimális számjegy 16 bites bejegyzésekre vonatkozik.)

FAT-bejegyzés értéke	Jelentése
(0)000(h)	szabad klaszter
(F)FF0(h) – (F)FF6(h)	a rendszer által lefoglalt klaszter
(F)FF7(h)	hibás klaszter, nem része egy láncnak
(F)FF8(h) – (F)FFF(h)	az állomány utolsó klasztere
(X)XXX(h)	az állomány valamelyik klaszterének száma

41. ábra: A FAT-bejegyzések értéke és jelentése

Ahhoz, hogy a rendszer az egy állományhoz tartozó klasztereket egyetlen, összefüggő egységgé tudja összefűzni, ismét a láncolási eljárást használja. Ennek az a lényege, hogy a rendszer a FAT-ban két, egymásra következő mezőn lévő bejegyzést egy tétellé fog össze. Az így létrehozott tételben az első bejegyzés az állomány valamelyik klaszterének számát, a második pedig az állomány folytató klaszterének számát tartalmazza. Ha valahonnan megtudhatnánk egy állomány kezdő klaszterének számát – amit (mint rövidesen kiderül) természetesen meg is tudunk –, akkor már szinte gyerekjáték végighaladni az így létrejött láncon: elindulunk a kezdő klasztertől, megnézzük az ehhez tartozó FAT tételt, ami megmutatja, hol kell keresnünk a következő klasztert, és így tovább, míg a lánc végéhez nem érünk. Mint már tudjuk, az utolsó klasztert külön kód jelöli, így – ha nem tévedtünk – nem tévedünk el.

A FAT visszafejtése és egy ilyen lánc végigkövetése van annyira érdekes (és bizony bonyolult is), hogy ezzel külön fejezetben foglalkozzunk.

A DOS lehetővé teszi, hogy a számítógépgyártók a lemezen a FAT-ról több másolatot is tárolhassanak. A DOS egy állomány felírásakor vagy a katalógus módosításakor az első FAT aktualizálásával egyidejűleg a másolatokat is aktualizálja, úgyhogy ezek tartalma hibamentes körülmények között azonos. Ha a DOS a FAT beolvasásakor mégis valamilyen hibát találna, akkor elolvassa a lemezen lévő többi másolatot mindaddig, amíg a beolvasás nem hibátlan, vagy elfogytak a másolatok. Így tehát még abban az esetben is, ha valamilyen véletlen folytán meghibásodna az egyik FAT, a DOS a másolat segítségével még megmentheti az állományt.

A DOS CHKDSK (check disk) lemezvizsgáló parancsa a lemezen lévő FAT-ok tartalmát hasonlítja össze.

6.12.3. A főkatalógus

A főkatalógus közvetlenül a FAT utolsó másolata után következik. A főkatalógus 32 bájtos bejegyzésekből áll, amelyek az állományokkal, (al)katalógusokkal kapcsolatos adatokat, valamint az opcionálisan megadható kötetnevet tartalmazza. A bejegyzések lehetséges maximális számát ugyancsak a BPB tartalmazza a 11(h) címen. Ez az érték szintén formáláskor kerül a lemezre. A katalógusbejegyzések felépítését a 42. ábra mutatja.

00(h)	állománynév	8 bájt
08(h)	állománynév kiterjesztése	3 bájt
0B(h)	állomány attribútuma	1 bájt
0C(h)	foglalt	10 bájt
16(h)	létrehozás, ill. utolsó módosítás időpontja	2 bájt
18(h)	létrehozás, ill. utolsó módosítás dátuma	2 bájt
1A(h)	állomány első klasztere	2 bájt
1C(h)	állomány hossza	4 bájt

42. ábra: Egy katalógusbejegyzés felépítése

Nézzük meg most a bejegyzések egyes elemeinek belső szerkezetét. Az első 8 bájt általában az illető állomány nevét tartalmazza, balra igazítva. Ha az állomány neve 8 karakternél kevesebb, akkor az üres helyekre szóköz (space, 32-es ASCII kód) karakter kerül. Ha a katalógusbejegyzés nem egy állományra vonatkozó információkat tartalmaz, hanem más a feladata, akkor az állománynév első karakterének a helyére speciális kód kerül:

00(h), ha az illető bejegyzésmező még sohasem tartalmazott bejegyzést, azaz a főkatalógus üres területe,

05(h), ha az állománynév első karaktere az E5(h) ASCII kódja,

2E(h), ha a bejegyzés az aktuális vagy a szülő katalógust jelenti (az egy ponttal jelölt katalógus). Ha a következő bájton is 2E(h) áll, akkor a bejegyzés klasztermezője a szülő katalógus klaszterszámát tartalmazza; ha a szülő katalógus a főkatalógusban van, akkor ez az érték 00(h).

E5(h), ha a bejegyzés törölt állományra vonatkozik.

A bejegyzés második mezeje a állománynév kiterjesztését tartalmazza, ugyancsak balra igazítva, szóköz karakterekkel kiegészítve. Amint látható, az állomány neve és a kiterjesztés között nincs elválasztó pont (ezt a DOS külön kezeli és figyeli, de nem tárolja).

A következő 1 bájton az állomány attribútuma található. Az attribútum bájt értéke az állomány bizonyos tulajdonságait határozza meg. Az egyes bitek jelentését a 43. ábra mutatja.

Bit	Értéke	Jelentése
0.	1	az állomány csak olvasható, írásra és törlésre nem nyitható meg (read-only)
1.	1	rejtett állomány, normál kereséskor nem jelenik meg (hidden)
2.	1	rendszerállomány, normál kereséskor nem jelenik meg (system)
3.	1	a lemez kötetneve (volume label)
4.	1	a bejegyzés alkatalógust jelöl (directory)
5.	0	az állomány archivált (archive), az utolsó módosítás óta nem változott
	1	az állomány a lemezen tárolt tartalmához képest megváltozott, ezért ismét lemezre kell írni
6 – 7.		foglalt

43. ábra: Egy állomány attribútum bájtja

Néhány megjegyzés az attribútum bájt bitjeivel kapcsolatban:

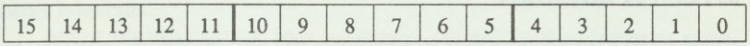
- A 0 – 2. bitek kombinálhatók, vagyis egymástól függetlenül lehet 0 vagy 1 az értékük.

- A 3 – 5. bitek nem kombinálhatók: egy bejegyzés értelemszerűen vagy kötetnevet, vagy katalógust, vagy normál állományt jelöl.

- A 0. bit (csak olvasható attribútum) értéke a DOS ATTRIB parancsával felhasználói szinten megváltoztatható.

A kötetnévre és a katalógusbejegyzésekre rövidesen visszatérünk.

Az attribútum bájt után 10 bájt hosszú foglalt mező következik, amelyet a rendszer saját céljaira használ. Ezt követi az állomány létrehozásának vagy utolsó módosításának időpontját és dátumát tartalmazó 1–1 szó. Ezek felépítését a 44., ill. a 45. ábra mutatja.

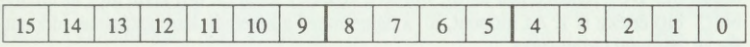


órák (0 – 23)

percek (0 – 59)

másodpercek 2 másodperces lépésekben (0 – 29)

44. ábra: Az időpontot tartalmazó szó szerkezete



észám (1980-tól számítva)

hónap (1 – 12)

nap (1 – 31)

45. ábra: A dátumot tartalmazó szó szerkezete

A következő mezőn annak a klaszternek a száma áll, amelyen az illető állomány kezdődik. Természetesen ehhez a klaszterhez is tartozik egy FAT bejegyzés, amely egyúttal – mint majd látni fogjuk – az állomány következő klaszterének számát is megadja. Az állomány első klaszterének fontosságát aligha kell hangsúlyozni: a DOS – miután megadtuk az állomány nevét – megkeresi az erre vonatkozó bejegyzést, és az itt talált klaszterszámon kezdi el az állomány adatait tartalmazó további klaszterek összegyűjtését. Ezt rövidesen még közelebbről is megnézzük.

A katalógusbejegyzés az utolsó négy bájton (az 1C(h) – 1F(h) ofszetcímeken) az állomány hosszát tartalmazza ugyanolyan elrendezésben, mint az FCB adatszerkezetben (l. 2.6.2. szakaszt, FCB funkciók): a legalsó címen a legkisebb helyértékű bájt van. A 4 bájtból a következő képlet segítségével számítható ki az állomány hossza:

1C(h)_cím + 1D(h)_cím * 256 + (1E(h)_cím + 1F(h)_cím * 256) * 65536

Ezzel elérkeztünk a katalógusbejegyzések belső felépítését ismertető fejezetrész végéhez. Ígéretünkhöz híven most visszatérünk az előzőekben már említett, a normál állományokra vonatkozó bejegyzésektől eltérő két különleges bejegyzés ismertetésére. Ilyen különleges bejegyzés a kötetnévbejegyzés, valamint azok a katalógusbejegyzések,

amelyek nem állományokra, hanem további katalógusbejegyzésekre mutatnak (ez utóbbiakat nevezzük alkatalógusoknak).

6.12.4. A kötetnév

A kötetnévről a lemezkezelés című fejezetben már elég sok szó esett. Az ott leírtakat kiegészítve a kötetnévvel kapcsolatos tudnivalókat a következőkben foglaljuk össze:

Egy lemeznek – attól függetlenül, hogy cserélhető vagy nem, vagy hogy hány partíciót tartalmaz – egyetlen kötetneve lehet. A kötetnevet a lemez formálásakor lehet – de nem kötelező – megadni. A kötetnév utólag is megadható. A DOS a kötetnevet – attól függetlenül, hogy mikor adjuk meg, és hogy a megadásakor melyik az aktuális katalógus – mindig a főkatalógusban helyezi el, és ott is keresi.

A kötetnevet a főkatalógus – a normál bejegyzésekhez hasonlóan – 32 bájt hosszú bejegyzésen tartalmazza. A kötetnév hossza – egy normál állománynév 8 bájtjának és az állománynév-kiterjesztés 3 bájtjának összevonásaként – 11 bájt lehet. Az attribútum bájt értéke MS-DOS rendszerben 28(h), vagyis a kötetnév biten (3. bit) kívül még az archivbit (5. bit) értéke is 1. Az időpont, ill. a dátummező a kötetnév létrehozásának időpontját, ill. dátumát tartalmazza.

A DOS 2.0 verziójáig bezárólag a lemez kötetnevének nincs jelentősége, és a rendszer a vele kapcsolatos műveleteket a handle funkciókon keresztül nem is támogatja (csak a bővített FCB funkciók használhatók). A 3.0 verziótól felfelé a rendszer a kötetnév alapján – amennyiben ez a képesség a rendszerbe be van építve – képes annak megállapítására, hogy egy állományra irányuló két művelet között történt-e lemezcsere. A kötetnévvel kapcsolatos – és handle funkciókon keresztül elvégezhető – műveletek köre azonban a 3.0 verzióban sem teljes: a 21(h) megszakítás 3C(h) funkciójával ugyan létrehozható 8-as attribútummal egy állomány (azaz kötetnév), sőt ez meg is keresheto a handle-funkcióval támogatott „első katalógusbejegyzés keresése" nevű, 4E(h) funkcióval, de a kötetnév megváltoztatásához továbbra is a bővített FCB funkciókat kell igénybe venni.

A kötetnév felhasználói szintről a DOS FORMAT parancsának paramétereként, vagy utólag, a LABEL parancs kiadásával hozható létre. A kötetnév a DIR, TREE, CHKDSK és a VOL parancsokkal megjeleníthető.

6.12.5. Katalógusbejegyzések

Már maga a fejezetcím is megtévesztő lehet, hiszen katalógusbejegyzésekről beszéltünk a főkatalógusról írt fejezetben is. Hogyan is van ez akkor?

Emlékezzünk vissza a 6.7. Katalógusok és állományok keresése című alfejezetben írtakra. Ott elmondtuk, hogy a főkatalógusban lévő bejegyzések vonatkozhatnak egy állományra vagy egy alkatalógusra, amely utóbbi ismét tartalmazhat állományokat, ill. további alkatalógusokat. Az egyes alkatalógusok tetszőleges mélységig egymásba

ágyazhatók, és tetszőleges számuk lehet. Így alakul ki a katalógusok és az állományok hierarchikus felépítése, amit egy fa struktúrával ábrázolhatunk. (l. a 23. ábrát). Itt most azokról a katalógusbejegyzésekről lesz szó, amelyek újabb katalógusbejegyzésekre mutatnak. Ebben az összefüggésben azokat a katalógusokat, amelyek egy másik katalógusra mutatnak, szülő katalógusnak, azokat pedig, amelyekre mutatnak, gyermek katalógusoknak is nevezik.

Annyit még megjegyzünk, hogy valamennyi „ősszülő" katalógusbejegyzés természetesen a főkatalógusban van, míg az ezekből kiinduló alkatalógusbejegyzések a főkatalógust követő lemezterületen, vagyis ott, ahol az állományok is találhatók.

Ezeknek a katalógusbejegyzéseknek a felépítése megegyezik az állományokra vonatkozó bejegyzések felépítésével azzal a különbséggel, hogy az előzőek attribútum bájtjában lévő 4. bit értéke 1, az állomány hosszát megadó bájtok értéke zérus, az időpont és a dátum mezők pedig a katalógus létrehozásának időpontját és dátumát tartalmazzák. A klasztermezőn annak a klaszternek a száma áll, amelyben az illető katalógus található.

A DOS egy (al)katalógus létrehozásakor a katalógus bejegyzésével egyidőben a katalógushoz hozzárendel egy klasztert, amelyben két bejegyzést hoz létre. Az első bejegyzés az ún. egypontos (.) bejegyzés, amely lényegében a katalógusbejegyzés másolata azzal a különbséggel, hogy a katalógus neve helyett az első bájton a 2E(h) érték áll, a többi pedig szóközökkel van feltöltve. A klasztermezőn lévő szám is ugyanaz, mint ami a katalógusbejegyzés klasztermezőjén áll, vagyis ez éppen a szóban forgó katalógus klaszterszáma.

A második bejegyzés az ún. kétpontos bejegyzés (..), ami annyiban különbözik az előzőtől, hogy a katalógusnévnek nemcsak az első, hanem a második bájtján is a 2E(h) érték áll, a klasztermező pedig annak a klaszternek a számát tartalmazza, amelyben az illető katalógus szülő katalógusa található. Amennyiben ez az érték zérus, úgy a szülő katalógus egyúttal a főkatalógus is.

Az előbbi két bejegyzés értelmét és szerepét könnyen belátjuk, ha emlékezetünkbe idézzük a katalógusok fa struktúráját: a DOS az egypontos bejegyzésből azt tudja megállapítani, hogy a szóban forgó katalógus a fa melyik ágán található, a kétpontosból pedig azt, hogy melyik az a csomópont, ahonnan ez az ág kiindul. (Ha még azt is elképzeljük, hogy az ágak végén lévő levelek a katalógusokban lévő állományoknak feleltethetők meg, akkor teljes a kép: a DOS a gyökértől kiindulva az ágakkal és csomópontokkal megadott úton végighaladva a fa bármely leveléhez elérhet.)

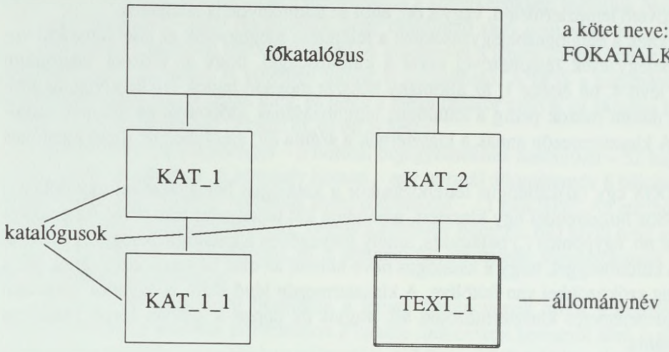
Az egy- és kétpontos bejegyzések nem törölhetők, és egy (al)katalógus is csak akkor törölhető, ha nem tartalmaz állományt (a törlés DOS parancsa: RD).

A fentiek illusztrálására és a jobb érthetőség kedvéért a főkatalógus – alkatalógus – állomány kapcsolatot egy gyakorlati példán keresztül is bemutatjuk. A példabeli lépéseket egy frissen formált, üres hajlékonylemezen végeztük el, a következők szerint:

1. A lemeznek kötetnevet adtunk. A kötetnév: FOKATALKAT.
2. Létrehoztunk egy katalógust KAT_1 néven.
3. Létrehoztunk egy másik katalógust KAT_2 néven.

4. Átváltottunk a KAT_1 katalógusra, és ebben létrehoztunk egy újabb katalógust, KAT_1_1 néven.
5. Átváltottunk a KAT_2 katalógusba, ahol létrehoztunk egy állományt TEXT_1 néven, amelybe írtunk is valamit.

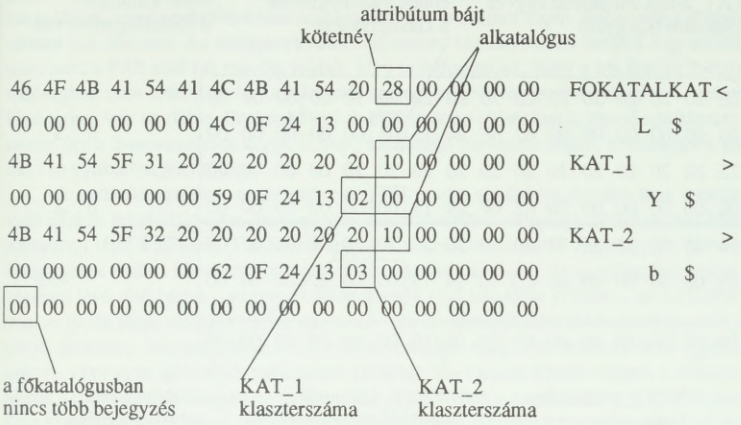
E lépések hatására a lemezen a következő elrendezés jött létre (46. ábra):



46. ábra: Katalógusbejegyzések elrendezése (példa)

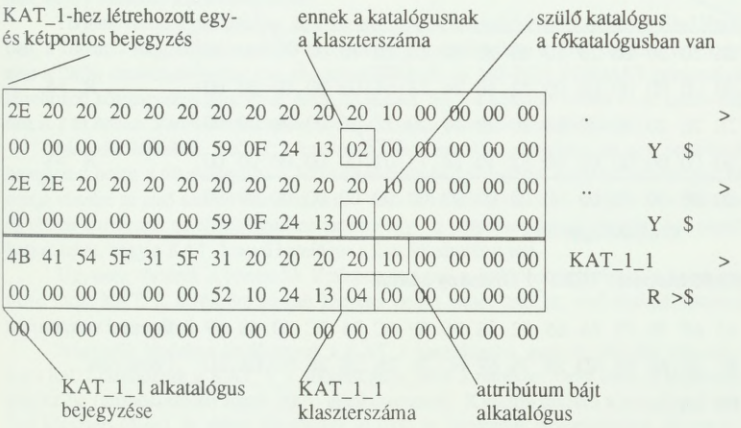
Ennek az elrendezésnek megfelelő lemeztartalmat a 47. ábra mutatja. (Ofszetcímeket nem írtunk ki, mert ezeknek itt nincs jelentőségük.)

FŐKATALÓGUS:

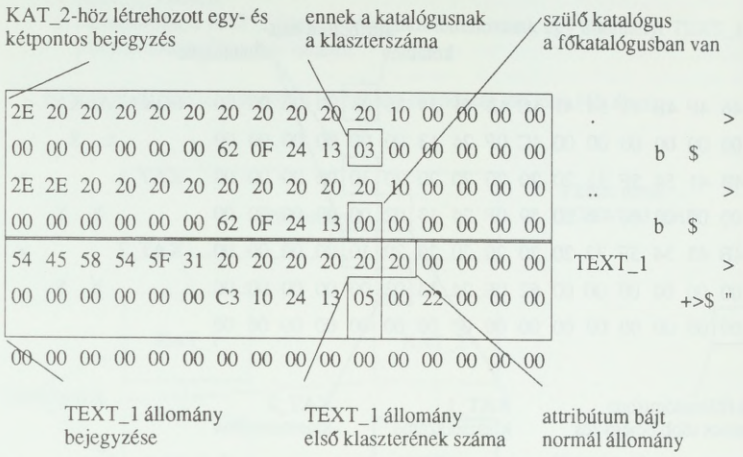


ÁLLOMÁNYOK TERÜLETE:

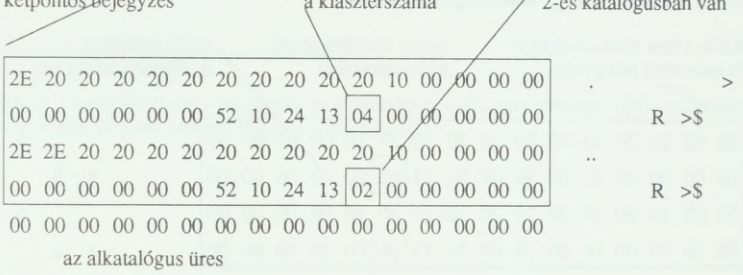
00002 klaszter: KAT_1 alkatalógus klasztere



00003 klaszter: KAT_2 alkatalógus klasztere



00004 klaszter: KAT_1_1 alkatalógus klasztere



00005 klaszter: TEXT_1 állomány adatai

45 67 79 73 7A 65 72 20 76 6F 6C 74 2C 20 68 6F Egyszer volt, ho
6C 20 6E 65 6D 20 76 6F 6C 74 2E 2E 2E 0D 0A 0D l nem volt…
0A 1A .....

47. ábra: Katalógusbejegyzések lemezen tárolt képe (példa).

Amint látható, a DOS a lemezterületet az első üres klasztertől kezdődően a katalógusok és az állomány létrehozásának sorrendjében, folyamatosan osztja ki. A példánkból azt is látjuk, hogy az elsőként kiosztott klaszter a 2-es számot kapta. Ehhez némi magyarázatot kell fűznünk: Az állományterület első szabad klasztere azért kapja a 2-es sorszámot, mert a FAT első két mezője foglalt. Úgy is felfoghatjuk, hogy a két foglalt mező a valóságban nem létező, de a sorszámozás szempontjából tekintetbe vett, 0 – 1. számú klaszterhez van rendelve. Ezeket követi az első, már a valóságban is létező adatklaszter, amely így a 2-es sorszámot kapja. Ennek, mint majd rövidesen látjuk, a számítások során van gyakorlati jelentősége.

A területkiosztásnak ez az imént említett rendje egyébként minden üres lemezre való íráskor mindaddig igaz, amíg el nem kezdünk törölni, vagy egy előzőleg már felírt állományt nem bővítünk. Törlésekor ugyanis a rendszer a törölt állományhoz tartozó klasztereket szabad klaszterekként jegyzi be a nyilvántartásába, és az állomány bejegyzésében lévő első bájtra – vagyis az állománynév első karaktere helyére – az E5 értéket írja. A törlés tehát valójában nem semmisíti meg az állományban lévő adatokat, ezért a törölt állomány helyreállítható, ha a törlést követően még nem hoztunk létre új állományt, vagy nem bővítettük valamelyik korábbit. Ha viszont írtunk valamit a lemezre, akkor a rendszer ezt – legyen ez új állomány vagy bővítés – a nyilvántartása szerinti első üres klaszterbe fogja beírni, ami már igazi törlést jelent. Természetesen a korábbi helykiosztási rend is „felborul".

A DOS a 3.0 verziótól felfelé egy állomány bővítésekor más módszer szerint osztja ki a klasztereket: a bővítéseket nem az első szabad klaszterbe, hanem a legutoljára felszabadított (törölt) klaszterbe írja be. E módszer szerint nagyobb annak a valószínűsége, hogy az egy állományhoz tartozó klaszterek viszonylag közel helyezkednek el egymáshoz, és így csökken a hozzáférési idő.

Térjünk vissza a példánkhoz, és kövessük nyomon a helykiosztás menetét: Első lépésben, a kötetnév megadásakor a DOS létrehozza a főkatalógusban az erre vonatkozó bejegyzést a 28(h) attribútum bájttal (ez az attribútum érték az MS-DOS FORMAT parancsával létrehozott kötetnevekre igaz: ez az érték azt jelenti, hogy a kötetnév bitnek és az archiv-bitnek is 1 az értéke. Vannak rendszerek, amelyek csak a kötetnév bitet állítják be).

Második lépésben, a KAT_1 katalógus létrehozásakor, a rendszer az erre vonatkozó bejegyzést beírja a főkatalógusba a 10(h) attribútum bájttal jelezve, hogy ez alkatalógus, kiosztja részére az első szabad klasztert, ami – üres lemezről lévén szó – most a 2-es klaszter, és ebben létrehozza az alkatalógushoz tartozó egy- és kétpontos bejegyzéseket. Azt mondhatjuk tehát, hogy a KAT_1 nevű katalógus a 2-es klaszterben van.

Ugyanez történik a harmadik lépésben is azzal a különbséggel, hogy a katalógus neve most KAT_2, a részére kiutalt klaszter pedig ismét csak az első szabad klaszter, ami most a 3-as számú.

Negyedik lépésben átváltottunk a KAT_1 katalógusba, azaz az aktuális klaszter a 2-es lett. Létrehoztuk a KAT_1_1 alkatalógust, amit a rendszer most ebbe a klaszterbe jegyez be (természetesen ismét 10(h) attribútummal). Kiosztja részére a következő szabad klasztert (4-es), és ebben létrehozza az egy- és kétpontos bejegyzéseket. Ez tehát a KAT_1_1 alkatalógus klasztere.

Végül az ötödik lépésben áttértünk a KAT_2 katalógusba (3-as klaszter), és itt létrehoztunk TEXT_1 néven egy normál állományt. A DOS beírta ebbe a klaszterbe a megfelelő bejegyzést (20(h) attribútummal), és kiutalta számára az 5-ös klasztert. Mivel a TEXT_1 nem alkatalógust, hanem közönséges állományt jelent, az 5-ös klaszter közvetlenül az állomány adatait tartalmazza.

6.12.6. A FAT visszafejtése

Mielőtt nekikezdenénk ahhoz, hogy egy állomány elhelyezkedését a FAT alapján nyomon kövessük, röviden foglaljuk össze a FAT-tal kapcsolatos eddigi ismereteinket.

A FAT az a nyilvántartás, amely alapján a DOS nyomon követi, hogy az állományok elhelyezésére szolgáló adatterület mely klaszterei foglaltak, ill. üresek, továbbá azt is, hogy hol találhatók az egyes állományokat tartalmazó klaszterek.

A FAT verziótól, ill. lemezmérettől függően 12, ill. 16 bites mezőkre van felosztva, amelyek különböző bejegyzéseket tartalmaznak. Az első két mezőn lévő bejegyzéseknek nincs közük az állományok területéhez: ennek a 12 bites mező esetén 3, 16 bites bejegyzés esetén pedig 4 bájt hosszú adatterületnek az első bájtján az adathordozó leíró bájtja, a továbbiakon pedig FF(h) áll.

A többi mező mindegyike egy-egy klaszterre mutat. A mezőkön lévő bejegyzések arról adnak tájékoztatást, hogy a mezőkhöz tartozó klaszterek mit reprezentálnak. A bejegyzések lehetséges jelentéseit a 41. ábrán mutattuk be.

Ennyi ismeret birtokában már hozzáláthatunk ahhoz, hogy egy állományt a FAT bejegyzések alapján „felgöngyölítsünk". A feladatot először elméletben végezzük el, hogy néhány általánosítható megállapítást is megfogalmazhassunk. Ezt követően az elméletben megismerteket egy gyakorlati példán keresztül is bemutatjuk.

Egy állomány visszafejtése alapvetően két lépésre bontható: első lépésben az állomány kezdő klaszterszámát, a továbbiakban pedig a folytató klaszterszámokat kell megállapítani.

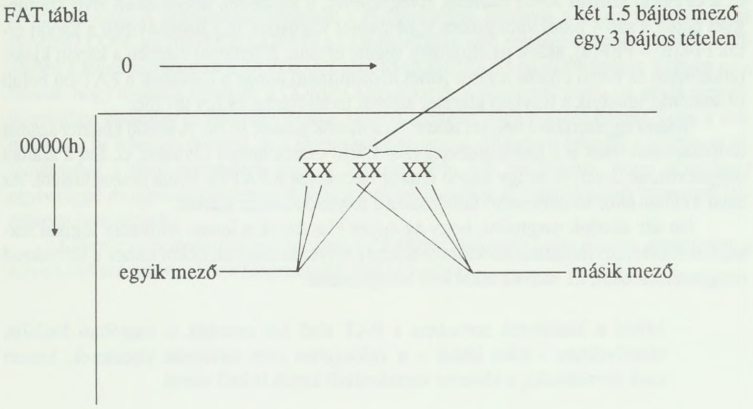
Mind a kezdő, mind az első folytató klaszter számának megállapításához az állomány katalógusbejegyzésének az 1A(h) ofszetcímén lévő, 16 bites bejegyzésből kell kiindulni. Ez a bejegyzés ugyanis egyrészt azt adja meg, hogy az állomány az állományterület hányadik klaszterén kezdődik, másrészt pedig azt, hogy a FAT-on belül melyik ofszetcímen található az a mező, amelyik az állomány folytató klaszterének számát tartalmazza. A többi folytató klaszter számának megállapításához már csak a FAT-on belüli bejegyzéseket kell használni. Ez eddig meglehetősen egyszerűnek tűnhet. Kicsit bonyolultabbá válik a dolog, ha a számításokat közelebbről is szemügyre vesszük.

Állapítsuk meg először a kezdő klaszter számát. Ez a feladat valóban egyszerű, hiszen ehhez nem kell mást tennünk, mint hogy kiolvassuk a katalógusbejegyzés 1A(h) ofszetcímén lévő szót. Az itt található érték közvetlenül a kezdő klaszter számát adja meg. Ennél egy kicsit bonyolultabb a folytató klaszterek számának megállapítása. Azért csak fogjunk hozzá.

Amint mondtuk, a katalógusbejegyzés 1A(h) ofszetcímén lévő szó egyúttal azt is meg-

adja, hogy a FAT-on belül melyik ofszetcímen található az a mező, amelyik az állomány folytató klaszterének számát tartalmazza. Számítsuk ki ezt az ofszetcímet.

A számítások menete innen kezdve eltérő attól függően, hogy a mezők hossza 12 vagy 16 bites. Lássuk először azt az esetet, amelyben a mező 12 bites (ez a bonyolultabb). Ebben az esetben a rendszer a két egymásra következő mezőn lévő bejegyzést egy $2 * 1.5 = 3$ bájtos tétellé kapcsolja össze. Gondolhatnánk, hogy ez az összekapcsolás úgy megy, hogy az egyik mezőt a tétel első 1.5 bájtja, a másikat pedig a második 1.5 bájtja tartalmazza. Nem így van. A rendszer a mezőket a valóságban a 48. ábrán látható módon kapcsolja össze.



48. ábra: Egy FAT tétel mezői

Az első folytató klaszter számát tartalmazó mezőnek a FAT-on belüli, bájtokban számított címét úgy számíthatjuk ki, hogy a katalógusbeli klaszterszámot megszorozzuk 1.5-del. Az így kapott szorzatnak az egész része a FAT-on belül annak a mezőnek a (0. címtől számított) ofszetcímére mutat, amely a folytató klaszter számát tartalmazza. Mivel azonban a FAT – más adatszerkezetekhez hasonlóan – az adatokat nem fél, hanem egész bájtokként tárolja, az 1.5 bájtnyi adat tárolásához is két bájtot használ fel. Ebből 1.5 bájt az egyik mezőhöz tartozó bejegyzést, a maradék 0.5 bájt pedig a másik mezőhöz tartozó bejegyzés egy „részét" tartalmazza. Úgy is fogalmazhatunk, hogy a két mező a 3 bájtos tételen a következőképpen „osztozkodik": míg az első bájtot kizárólag az első, a harmadikat pedig kizárólag a második bejegyzés „birtokolja", a középső bájt fele-fele arányban „közös tulajdon". A rendszer ezzel a „furfanggal" teszi lehetővé, hogy egy másfél bájtos információt a könnyebben kezelhető, 2 bájton tárolja, ugyanakkor egy fél bájtnyi hely se menjen veszendőbe. (Hogy ezzel nekünk okoz gondot, az a mi bajunk.) A két bájton tárolt információból tehát valahogyan ki kell „vennünk" a számunkra érdekes 1.5 bájtnyi információt. Amint a 48. ábrán is látható, ez nem megy úgy, hogy veszszük az első bájtot, meg a következő bájt első felét.

A megoldás

Az előbbi szorzat egész részét ofszetcímként használva eljutottunk ahhoz a két bájthoz, amely a folytató klaszter számát tartalmazza. Ebből a két bájtból a szokásos ábrázolásmódnak megfelelően (alsó bájt, felső bájt) egy szót képezünk. A most következő lépés attól függ, hogy a szorzatként kapott szám egész vagy tört. Ha egész, akkor a szóból a legnagyobb helyértékén lévő hexadecimális számjegyet, ha pedig tört, akkor a legkisebb helyértéken lévő hexadecimális számjegyet vágjuk le. A kapott érték mindegyik esetben a folytató klaszter számát tartalmazza. (Programban ezeket a levágásokat természetesen „elegánsabban" végezzük el: a legnagyobb helyértéken lévő számjegyet a szó és a 0FFF(h) közötti AND művelet elvégzésével, a legkisebb helyértéken lévő számjegyet pedig a szó 4 bittel való jobbra léptetésével vághatjuk le.) Amennyiben a kapott érték FF8(h) – FFF(h), akkor az állomány végére értünk. Ellenkező esetben a kapott klaszterszámból az előző eljárás szerint ismét kiszámítható annak a mezőnek a FAT-on belüli ofszetcíme, amelyik a folytató klaszter számát tartalmazza, és így tovább.

Sokkal egyszerűbb a helyzet akkor, ha a mezők hossza 16 bit. A kezdő klaszter számát természetesen most is a katalógusbejegyzés 1A(h) ofszetcíméről olvassuk ki. Ezt a számot megszorozzuk 2-vel, és az így kapott értéket használjuk a FAT-on belüli ofszetcímként. Az ezen a címen álló, 16 bites mező tartalmazza a folytató klaszter számát.

Ha azt akarjuk megtudni, hogy az egyes klaszterek a lemez hányadik logikai sorszámú szektorain (abszolút szektorszámokon) helyezkednek el, akkor ehhez a következő meggondolásokat, ill. műveleteket kell elvégeznünk:

- Mivel a klaszterek sorszáma a FAT első két mezőjét is magában foglalja, (amelyekhez – mint láttuk – a valóságban nem tartoznak klaszterek, hanem csak sorszámok), a klaszter sorszámából kettőt le kell vonni.

- A BPB-ben lévő adatokból megállapítható, hogy egy klaszter hány szektort tartalmaz. A fenti művelet eredményét ezzel a számmal meg kell szorozni.

- A BPB-ből az is kideríthető, hogy a rendszer a lemez elején hány szektort foglal le a lemezkezeléssel kapcsolatos célokra.

Számoljunk utána egy kétoldalas, 40 sávos és 9 szektoros hajlékonylemez példáján:

- Tudjuk, hogy a 0. szektoron a betöltő szektornak kell lennie, ami összesen 1 szektornyi foglalt lemezterületet jelent (itt további foglalt szektor nincs).

- A betöltő szektort a FAT követi, amely két példányban összesen 2 * 2 = 4 szektort foglal el.

- Végül a főkatalógus következik, amely összesen 7 szektort foglal el. (A teljesség kedvéért: egy katalógusbejegyzés hossza 32 bájt, a bejegyzések száma ma-

ximum 112, ehhez összesen 112 * 32 = 3584 bájtra van szükség, ami 3584/512 = 7 szektornak felel meg.)

A rendszer által lefoglalt szektorok száma tehát összesen 1 + 4 + 7 = 12, a szektorok logikai sorszáma pedig 0 – 11. Ezt a 12 foglalt szektort hozzá kell adni az előbb kiszámított szektorszámhoz.

Ezeket a műveleteket elvégezve megkapjuk annak a szektornak a logikai sorszámát, amelyen az illető klaszter kezdődik.

Következzék most az ígért gyakorlati példa: Adathordozóként kétoldalas, 9 szektorra és 40 sávra formált hajlékonylemezt választottunk. A lemezt a DOS FORMAT parancsával formáltuk meg. A parancsban megadtuk a /S paramétert, úgyhogy a lemez tartalmazza az IBMBIO.COM, az IBMDOS.COM és a COMMAND.COM rendszerállományokat. Ezt követően a lemezre másoltuk a DOS részét képező COMP.COM nevű állományt. Legyen az a feladat, hogy nyomon követjük a COMP.COM állomány elhelyezkedését. (A példa megválasztásában kizárólag az állomány hossza játszott szerepet: legyen olyan hosszú, hogy a számításokat be lehessen mutatni, de a türelmünket ne vegye túlságosan igénybe.) Miután az állományokat a fenti sorrendben írtuk a lemezre, nem lesz meglepetés, hogy a rendszer egybefüggő és egymásra következő lemezterületeket osztott ki számukra.) A lemeznek kötetnevet nem adtunk.

Először is bemutatjuk a FAT-nak és a főkatalógusnak azon részeit, amelyeket a következő számításainkhoz használnunk kell (49. ábra).

A FAT TÁBLA:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	FD	FF	FF	03	40	00	05	60	00	07	80	00	09	A0	00	0B
0010	C0	00	0D	E0	00	0F	00	01	11	20	01	13	40	01	15	60
0020	01	17	F0	FF	19	A0	01	1B	C0	01	1D	E0	01	1F	00	02
0030	21	20	02	23	40	02	25	60	02	27	80	02	29	A0	02	2B
0040	C0	02	2D	E0	02	2F	00	03	31	20	03	33	40	03	35	F0
0050	FF	37	80	03	39	A0	03	3B	C0	03	3D	E0	03	3F	00	04
0060	41	20	04	43	40	04	45	60	04	47	F0	FF	49	A0	04	4B
0070	C0	04	FF	0F	00	00	00	00	00	00	00	00	00	00	00	00

:

A FŐKATALÓGUS:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	49	42	4D	42	49	4F	20	20	43	4F	4D	27	00	00	00	00
0010	00	00	00	00	00	00	00	60	72	0E	02	00	54	56	00	00
0020	49	42	4D	44	4F	53	20	20	43	4F	4D	27	00	00	00	00
0030	00	00	00	00	00	00	00	60	71	0E	18	00	CF	75	00	00
0040	43	4F	4D	4D	41	4E	44	20	43	4F	4D	20	00	00	00	00
0050	00	00	00	00	00	00	00	60	3E	08	36	00	80	45	00	00
0060	43	4F	4D	50	20	20	20	20	43	4F	4D	20	00	00	00	00
0070	00	00	00	00	00	00	00	60	71	0E	48	00	76	10	00	00

:

49. ábra: A FAT és a főkatalógus részlete (példa)

Az ábrán látható, hogy a COMP.COM főkatalógusbeli bejegyzésének 1A(h) ofszetcímén a 0048(h) áll. Az állomány tehát a 48(h), azaz a decimális 72. számú klaszteren kezdődik.

Az első folytató klaszter FAT-on belüli ofszetcímének számításához szorozzuk meg ezt az értéket 1.5-del. Az eredmény:

72 * 1.5 = 108, azaz 6C(h)

Ezen az ofszetcímen az A049(h) szó áll. Mivel a szorzat egész szám, a szóból a legnagyobb helyértéken álló hexadecimális számjegyet kell levágnunk: marad a 049(h), azaz a decimális 73, ami az első folytató klaszter száma.

A következő folytató klaszter számának meghatározásához most ezt az értéket használjuk. Megszorozva 1.5-del, az eredmény:

73 * 1.5 = 109.5

Ennek egész része 109, azaz 6D(h).

Ezen az ofszetcímen a 04A0(h) szó áll. Mivel a szorzat tört szám, a szó legkisebb helyértékén álló hexadecimális számjegyet vágjuk le: marad a 04A(h), azaz a decimális 74, ami a folytató klaszter száma. Ezt 1.5-del szorozva:

74 * 1.5 = 111, azaz 6F(h)

Ezen az ofszetcímen a C04B(h) szó áll. A szorzat egész szám, tehát a legnagyobb helyértéken álló hexadecimális számjegyet vágjuk le: marad a 04B(h), azaz a decimális 75, ami a következő folytató klaszter száma. Szorozva 1.5-del:

75 * 1.5 = 112.5

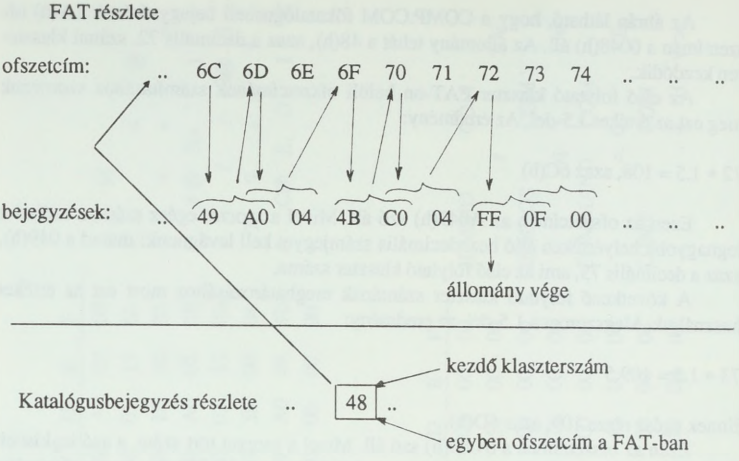
Ennek egész része 112, azaz 70(h).

Ezen az ofszetcímen a 04C0(h) szó áll. A szorzat eredménye tört szám volt, ezért a legkisebb helyértéken álló számjegyet vágjuk le: marad 04C(h), ami a decimális 76. számú klasztert jelenti. Ezt is szorozzuk meg 1.5-del:

76 * 1.5 = 114, azaz 72(h)

Ezen a címen a 0FFF(h) szó áll, amelyből – mivel a szorzat egész szám – a legnagyobb helyértéken álló számjegyet levágva az FFF(h) marad, ez pedig azt jelenti, hogy az állomány végére értünk.

A fejezet lezárásaként érdemes még egyszer szemügyre venni, miként épül fel az állományt tartalmazó klaszterek lánca. A katalógusbejegyzésből kiindulva átlépünk a FAT-ba, és itt folytatjuk a láncszemek keresését. Mindezt az 50. ábrán követhetjük nyomon.



50. ábra: Egy állomány klasztereinek lánca (példa)

Még néhány gondolat erejéig maradjunk a FAT táblánál. A korábbiakban már elmondtuk, hogy a FAT minden egyes mezője – ez első kettő kivételével – közvetlenül hozzá van rendelve egy klaszterhez. Ebben az összefüggésben akkor vajon hogyan értelmezhető pl. a 49. ábra szerinti FAT 22(h) ofszetcímén lévő FFF(h) érték, amely egyébként, mint tudjuk, egy állomány utolsó klaszterét jelzi? Ehhez a mezőhöz melyik klaszter tartozik? Ekkora sorszámú klaszter a lemezen nincs is! E átszólagos ellentmondás feloldása a következő.

Az FFF(h) érték természetesen nem egy klaszter sorszámát jelenti, hanem a rendszer arra használja, hogy a FAT-ban az egymásra következő állományok végét és elejét szét tudja választani. Ennek ellenére van olyan klaszter, amely a 22(h) ofszetcímű mezőhöz rendelhető: ez az a kezdő klaszter, amelynek sorszámát az a katalógusbejegyzés tartalmazza, amely arra az állományra vonatkozik, amelynek adatait a következő ofszetcímekről kiolvasható klaszterek tartalmazzák. Ez így nagyon bonyolultnak tűnhet, ezért ismét nézzük meg a 49. ábrát, hogy könnyebben megérthessük az összefüggéseket.

Amint látjuk, a 22(h) ofszetcímen lévő érték FFF(h). Azt biztosan állíthatjuk, hogy ez az érték valóban a mondott FFF(h), és nem FF0(h). Az előzőekben bemutatott számítások alapján egészen biztos ugyanis, hogy az ofszetcímen álló szónak nem a legnagyobb, hanem a legkisebb helyértékén lévő számjegyét kell levágnunk.

Itt térünk ki egy olyan eljárás bemutatására, amely nagyon egyszerűvé és szemléletessé teszi egy 3 bájtos FAT tétel 1.5 – 1.5 bájtos mezőkre való szétbontását, ill. összerakását. Lássuk előbb a szétbontást:

Példaként nézzük a 03(h) ofszetcímen lévő 3 bájtos tételt.

A három bájt:	03 40 00
Cseréljük fel az első és harmadik bájtot:	00 40 03
majd az első és a második 1.5 – 1.5 bájtot:	003 004

Az eredmény: az első 1.5 bájt egy klaszter, a második 1.5 bájt a folytató klaszter számát adja.

Ugyanilyen egyszerű a művelet megfordítása, vagyis két 1.5 bájtos mező összerakása egy három bájtos tétellé:
Ehhez vegyük példaként a 5-ös és a 6-os klaszterszámot tartalmazó mezők összerakását:

A két mező tartalma:	005 006
Megcserélve és bájtokba csoportosítva:	00 60 05
Végül az első és harmadik bájtot megcserélve:	05 60 00

Eredményül a FAT 06(h) ofszetcímén lévő három bájtos tételt kapjuk (ez nem lehet meglepetés, hiszen tudtuk, hogy az állomány a lemezen egybefüggő területen helyezkedik el).

Megjegyezzük, hogy ez az eljárás a FAT bármelyik tételére alkalmazható, de arról nem szabad megfeledkezni, hogy egy (három bájtos) tétel csak olyan ofszetcímen kezdődhet, amely 3 egész számú többszöröse!)

Ezek után visszatérhetünk a 22(h) ofszetcímhez. A fenti eljárással könnyen megállapíthatjuk, hogy az előző mező a 17(h) klaszterszámot, az utána következő pedig 19(h) klaszterszámot tartalmazza. Itt tehát látszólag egy „hézag" van, pedig tudjuk, hogy a rendszer a klasztereket folyamatosan osztotta ki. Ebbe a hézagba illeszkedik bele a következő állomány kezdő klasztere: a 21(h) ofszetcím annak a klaszternek a számát tartalmazza, amelyen befejeződött az IBMBIO.COM állomány, a 22(h) ofszetcímre a következő állomány, vagyis az IBMDOS.COM kezdő klaszterének száma illeszkedik („kölcsönvéve" a katalógusbejegyzésből), majd a 23(h) ofszetcímen már ennek a folytatása következik.

Ez a gondolatsor ötletet adhat egy olyan program megírásához, amely két állományt egy állománnyá fűz össze. Ha a kezdő állomány utolsó klaszterét jelző FFF(h) bejegyzés helyére a hozzáfűzendő állomány katalógusbejegyzésében lévő kezdő klaszterszámot írjuk be, akkor a FAT-on belül megtettük a szükséges intézkedést. Természetesen igazítani kell még az állomány hosszán, az állomány végét jelző bájton, stb. Meg lehet próbálni...

Ezzel a FAT táblában tett felfedező útunk végére értünk. Reméljük, hasznos volt.

A továbbiakban a rendszerprogramok másik nagy csoportjával, a BIOS-szal foglalkozunk.

2302-3/a


NOVOTRADE

IBM PC XT/AT rendszerprogramozás I.