IBM PC/XT

felhasználóknak és programozóknak

2

Pethő Ádám

A programozó és az MS-DOS

SZÁMALK

PROFESSZIONÁLIS SZEMÉLYI SZÁMÍTÓGÉPEK

Ádám Pethő

IBM PC/XT

FOR USERS AND PROGRAMMERS

2.

THE PROGRAMMER AND THE MS-DOS

Computing Applications and Service Company
Budapest, 1987

Pethő Ádám

IBM PC/XT

FELHASZNÁLÓKNAK
ÉS
PROGRAMOZÓKNAK

2.

A PROGRAMOZÓ ÉS AZ MS-DOS

Számítástechnika-alkalmazási Vállalat
Budapest, 1987

Lektorálta: **Radnóti László**
Supervised by **L. Radnóti**

ISBN 963 553 131 1

ISBN 963 553 123 0 (1. kötet)
ISBN 963 553 129 x (összkiadás)

Kiadja a Számítástechnika-alkalmazási Vállalat
Felelős kiadó: Havass Miklós vezérigazgató
Felelős szerkesztő: Lukács Erzsébet
Műszaki szerkesztő: Beleznai László
A fedelet tervezte: Molnár Zoltán
Megjelent: 32,175 (A/5) ív terjedelemben

87-1147 ARTUNION Kiadó és Nyomda Iroda
Budapest, 1987. Felelős vezető: Deák Gyula
1141/764

Tartalomjegyzék

Táblázatok és ábrák jegyzéke

Előszó

Ez a könyv a Számalk PPC sorozata "IBM-PC/XT felhasználóknak és programozóknak" című háromkötetesre tervezett könyvsorozatának második kötete. Az eredeti tervek szerint a három kötet sorban az "Assembly alapismerek", az "MS-DOS alapismeretek" és a "Programozás az IBM PC/XT-n" címet viselte volna. A második részt azonban érdemesnek láttuk két kötetre bontani; így alakult ki a:

2. kötet: "A programozó és az MS-DOS"
3. kötet: "A programozó és a ROM BIOS"

Tehát ez a kötet, melyet az Olvasó a kezében tart, a tervezett második kötetnek valamivel több mint a felét tartalmazza. Témája az MS-DOS operációs rendszer használata a programozó eszközeivel. A könyv megírása során igyekeztem összegyűjteni mindent, amit a program futása közben a programozó kaphat az operációs rendszertől. Csaknem minden lehetőséget példaprogramokkal is megvilágítok. E programok megírása közben az a szempont vezetett, hogy a "kényes kérdéseket" ne kerüljem meg. Az Olvasó talán elnézi, ha néhány egészen triviális lehetőség illusztrálása elmarad olyanok bővebb ismertetése kedvéért, amelyek igazán fontosak, és végrehajtásuk nem egészen nyilvánvaló.

Számos esetben fordul elő, hogy a rendelkezésre álló irodalom nem ad meggyőző választ az Olvasóban felmerülő kérdésekre. Tapasztalataim szerint a kézikönyvek alapos tanulmányozása után is marad több homályos pont. Nemegyszer találkoztam olyan problémákkal, melyeket csak "szájhagyomány útján" megszerzett információk birtokában, sőt magasszintű programozási nyelven készült programok disassemblálása révén tudtam leküzdeni. Az ilyen pontok felderítése érdekében a rendszer egyes részeinek visszafejtésétől sem riadtam vissza. A könyvben éppen azokra a részletekre szeretném a legnagyobb súlyt fektetni, amelyekre az irodalom nem ad elég alapos választ. Ez természetesen óriási munka, és egy pillanatig sem merem azt hinni, hogy akárcsak a legtöbb kérdést is meggyőzően meg tudnám világítani. Remélem azonban, hogy az Olvasó haszonnal forgathatja majd munkámat, és az itt leírt tapasztalatok birtokában többet fog tudni gépéről és annak operációs rendszeréről.

Mivel könyvemet tankönyvnek szántam, az ismeretek csoportosítása során logikai rendet igyekeztem követni; a függelékek segítségével viszonylag egyszerűen vissza tudjuk keresni az információkat másféle szempontból is. Az ismeretek leírása során

olyan alaposságra törekedtem, hogy kézikönyvként is használható legyen. Például az MS-DOS funkciókat nem kódjuk, hanem szerepük szerint csoportosítjuk; azonban az "MS-DOS funkciókódok" függelékből visszakereshetjük őket kódjuk szerint is.

A példaprogramok, mint az Olvasó látni fogja, nemcsak úgy lesznek egyre bonyolultabbak, hogy egyre bonyolultabb, összetettebb rendszerfunkciókkal foglalkoznak, hanem egyfajta programozói fejlődést is szemléltetnek: egy olyan utat, amelyet a legtöbb programozó néhány kitérővel bejár. Eleinte nagyon célratörően dolgozik, a lehető legkevesebbet foglalkozik a program formájával, szép, messzebbre vezető megfogalmazásával. Egyetlen célja, hogy az éppen előtte álló nehézséggel megbirkózzon. További munkája során számos sikerélmény és még több kudarc hatására érik, fejlődik programozói tudata. Látóköre szélesedik, egyre kevésbé fél az előtte álló technikai nehézségektől. Persze a nehézségek, amelyek régebben ijesztőnek látszottak, egyre zsugorodnak (hogy aztán nagyobbaknak adják át a helyüket). Eleinte probléma volt az, hogy miképpen is lehetne kibűvölni a képernyőre egy szöveget, hogy lehet file-okat kezelni. Később ezek a technikai nehézségek eltűnnek: egyszerűbbnek látszanak, egyre több ügyes kis rutin és makró áll rendelkezésünkre (melyekben megbízunk, hiszen mi írtuk őket). Lassan megbízunk majd a mások által írott programokban is, a csoportmunka összecsiszol más programozók eltérő stílusával, minden munkatársunktól (a nálunk gyengébbektől is) tanulunk ezt-azt, és eszköztárunk egyre gazdagabb. Gondjaink is összetettebbek és "magasabbrendűek" lesznek. Csuklóból oldjuk meg a technikai nehézségeket, és egyre inkább a programok komoly algoritmikus problémái foglalkoztatnak bennünket, az általánosabb eljárások, a különféle mellékhatások, a programok finom és egyre finomabb "hangolása".

Ennek az útnak a bejárása csodálatos élmény annak számára, aki figyel a munkájára, és igyekszik jobban és mindig jobban dolgozni. A programpéldák során ennek a végtelen hosszú útnak első néhány lépését tesszük meg a "fejjel a falnak" stílustól egy (talán) profi szinten is elfogadható programozási felfogásig. Természetesen teszünk bizonyos kitérőket, és utólag sokszor mosolygunk majd a korábbi programok nehézkes és nem túl elegáns, terjedelmes megfogalmazásán. Sőt, mint minden munkában, néha restellni is fogjuk a korábbi megoldásokat.

Arra kérem tehát az Olvasót, hogy nagyon gondosan tanulmányozza a programpéldákat. Főként ne arra figyeljen, hogy ezek mit csinálnak, hanem arra, hogyan. Igyekezzen belekötni minden

egyes programsorba, gondolkozzon el szebb, jobb, az ésszerűség határain belül tömörebb megfogalmazáson. Próbáljon meg minden kis programot (legalább gondolatban) fejleszteni, mégpedig nem a bonyolítás, hanem az általánosítás irányába. Gondolkozzon el azon, hol vannak a programokban olyan pontok, melyek "nyújthatók", és hol vannak azok a szinte hardware-akadályok, amelyeket (legalábbis a kód felülírása nélkül) nem lehet átlépni. Például érdemes arra törekedni, hogy egy adott szubrutint minél általánosabban fogalmazzunk meg. Tegyük fel, hogy át kell másolnunk egy string tartalmát az első kocsivissza karakterig egy másik területre (probléma az "életből ellesve": egyik nagyobb léleg-zetű példaprogramunknak lesz erre szüksége). Ekkor nem használjuk ki azt, hogy a zárókarakter éppen egy kocsivissza kód, hanem másolunk az első vezérlőkarakterig. Az így megfogalmazott rutin (mivel a kocsivissza is vezérlőkarakter) konkrét célunknak is megfelel, de általánosabb használatra is alkalmas. Amit csak lehet, paraméterként adunk át (pl. a stringek kezdőcímét). De nem lehet paraméter az, hogy szavanként vagy byte-onként menjen végbe a másolás; ez vagy nagyon bonyolult, vagy pedig önmódosító kódot eredményezne, és az álmoskönyv szerint egyik sem igazán szerencsés választás. Ez tehát lényegében egy hardware-akadály: az önmódosító kód írását erősen ellenjavallt cselekedetnek tarthatjuk.

A programok ilyen szempontok szerinti alapos átgondolása még annak is hasznára válhat, aki egyébként már jóval túllépett az ilyen egyszerű programok szintjén.

A példákat a könyvben megjelenő formában egytől egyig kipróbáltam. A kész programok szövegét természetesen nem kézzel gépeltem be a könyvbe; a később sikeresen lefordított és futtatott forrásnyelvű file-t szövegszerkesztővel másoltam be, tehát a programok kódja garantáltan megegyezik az eredeti és lefuttatott változattal.

Ennek ellenére a programok sikeres futásáért nem állhatok jót. Más MS-DOS verzióban, más ("kompatibilis") gépen a futás eredménye némileg eltérő vagy egészen más is lehet. Példaként az ANSI drivert említem, amely az MS-DOS operációs rendszer egyes (nem eredeti Microsoft) változataiban jóval többet tud az itt leírt verziónál. Ha valaki ezt a többletet kihasználja, csalódni fog a program más gépen való futtatásakor.

Budapest, 1987. június 15.

Kiknek szánjuk ezt a könyvet?

Jóllehet minden egyes példaprogram assemblyben íródott, a könyv nemcsak assemblyben és C-ben vagy egyéb gépközeli nyelven programozókhoz szól. A rendszerhívások, rendszerszolgáltatások alapos ismerete minden programozó számára fontos. A könyvet mindazon programozók haszonnal forgathatják, akik (bármilyen nyelven programoznak is) egy kicsit a dolgok mögé akarnak látni; akik tudni szeretnék, hogy körülbelül mi zajlik a rendszerben akkor, amikor programjuk egy adott műveletet végez.

Ugyancsak haszonnal forgathatják rendszerszervezők vagy rendszertervezők. Hiszen ők azok, akiknek igazán tisztában kell lenniük a gép valódi fizikai és rendszerlehetőségeivel. Igaz, hogy minden gépet és rendszert meg lehet erőszakolni és alaptermészetétől idegen dolgokat rákényszeríteni, de ez sohasem vezet jó eredményre. Ha valakinek, akkor a rendszertervezőnek pontosan tudnia kell, hogy mit várhat az operációs rendszertől. Egy program minősége sokkal inkább múlik a tervezésen, mint a megvalósításon. Pontosabban: rosszul tervezett programot a kódolás során megjavítani nem lehet, legfeljebb tovább rontani.

A könyv megértéséhez szükséges ismeretek

Elsődlegesen fontos, hogy az Olvasó felhasználói szinten ismerje az MS-DOS operációs rendszert, a rendszer belső és külső parancsait. Ismernie kell az olyan alapvető szolgáltatások alkalmazását és jelentőségét, mint a directory-struktúra, a beépített óra (DATE és TIME parancsok) stb.

Feltételezzük azt is, hogy az Olvasó ismer legalább egy szövegszerkesztő programot, és azzal képes programszövegek begépelésére és karbantartására (gyakran találkozunk olyan "programozókkal", akik igen gyengén editálnak - nem tudom, hogy a korszerű interaktív rendszerek világában hogyan élnek meg).

Végül pedig feltételezzük az "IBM PC/XT felhasználóknak és programozóknak" sorozat első kötetében foglaltak ismeretét. A példaprogramok során minden előzetes figyelmeztetés nélkül felhasználjuk az ebben foglalt utasításokat, assembler direktívákat és az ott leírt összes egyéb információt.

Bevezetés

A programozó számára létfontosságú, hogy tisztában legyen a használt számítógép és az operációs rendszer lehetőségeivel. Pontosan tudnia kell, mire képes a hardware, hol vannak azok a határok, amelyeket semmiképpen nem léphet át. Ugyanilyen fontos, hogy minél alaposabban ismerje az operációs rendszer lehetőségeit — mely funkciók állnak rendelkezésére, és mi az, amit neki kell megírnia.

A program "élete", futása nagyjából három részből áll: a belépés, amikor az operációs rendszer beolvassa a programot egy file-ból a memóriába, előkészíti a futását (pl. biztosítja számára a szükséges perifériális eszközöket, logikai csatornákat stb.), és a vezérlést a program belépési utasítására adja. A második szakasz a program futása. Egyfelhasználós, egyszerre egy programot futtató operációs rendszer esetében a vezérlés állandóan a program utasításain van, és az operációs rendszer egyes részei csak akkor aktivizálódnak, ha a program meghívja őket valamilyen célból.

> Megjegyzés: az egyidejűleg több programot futtató ún. multi-tasking rendszerek esetén a vezérlést időről időre az operációs rendszer kapja meg, amely váltja az éppen aktív programot. A program azonban erről általában nem szerez tudomást, úgy "alszik" a tárban, hogy felébredésekor (azaz újabb aktivizálásakor) ott folytatja a munkát, ahol abbahagyta. Hacsak nem akar közvetlenül kommunikálni egyéb programokkal, akkor számára multi-tasking rendszerben futni csaknem pontosan olyan, mintha egyedül lenne a rendszerben.

Végül a program futásának befejező szakasza a kilépés és annak előkészítése. A legtöbb esetben nagyon fontos, hogy programunk semmilyen nyomot ne hagyjon az operációs rendszerben, a gépben. Amit futás közben megváltoztattunk, azt kötelesek vagyunk helyreállítani (hacsak nem éppen annak az elemnek a megváltoztatása volt a cél). A program a kilépés után csaknem minden operációs rendszerben küldhet egy üzenetet az őt elindító programnak - ez szinte mindig a rendszer működését, a programok futását vezérlő program (ezt számos helyen nevezik supervisornak, felügyelőnek, "főnöknek"). Ezzel az üzenettel a program információt adhat futása sikeréről, esetleg a sikertelenség okáról.

Az MS-DOS operációs rendszer az IBM PC/XT (és AT) számítógépek standard operációs rendszere. Első változata 1980-ban jelent meg. Azóta több generációja terjedt el világszerte. Magyarországon jelenleg (1987 júliusában) a 3.20 verzió a legelterjedtebb.

Nem lévén a Microsoft Corporation munkatársa, nem vagyok tisztában az MS-DOS operációs rendszer keletkezésének körülményeivel, tervezőinek, megvalósítóinak szándékával. Azonban az eredmény és a rendszer egyes változatainak fejlődése arra enged következtetni, hogy kezdetben egy alaposan "felerősitett" CP/M operációs rendszerre gondoltak (melyet a Digital Research fejlesztett ki Intel 8080 alapú személyi számítógépekre a hetvenes évek közepén). Ezt támasztja alá a rendszer teljes külső megjelenése és programból hívható néhány alapvető funkciója. Tehát egyrészt maga a rendszer-prompt (A>, B> stb.), a logikai eszköznevek, a legfontosabb külső és belső parancsok neve és funkciói (maga a "külső" és "belső" parancs kifejezés is a CP/M-ből származik, jóllehet a fogalom ősidők óta ismert), másrészt pedig az illető rendszerhívások kódja, valamint teljes szintaktikai és logikai kompatibilitása a megfelelő CP/M funkciókkal. Az MS-DOS azonban számos területen túllépett a CP/M-en: például rugalmasabb és hatékonyabb file-rendszert alakitottak ki benne, bevezették a teljesen UNIX-szerű directory-struktúrát.

A későbbi MS-DOS verziók (jóllehet a Microsoft [itt-ott igen nagy nehézségek árán!] fenntartotta a felülről való kompatibilitást a korábbi verziókkal) egyre inkább a UNIX irányába léptek. A UNIX operációs rendszer számos olyan elemét találjuk meg az MS-DOS-ban, amelyek a CP/M-ben nyomokban sem léteztek: az operációs rendszer felszabdalását (ti. az MS-DOS nem annyira egy parancssor-értelmezővel, mint inkább egy parancs-interpreterrel rendelkezik, mivel az e funkciót betöltő COMMAND.COM programból újraindítható SHELL), az (átirányítható) standard file-okat, a "pipeline", a "csővezeték" bevezetését. Belső rendszerét tekintve az újabb verziókban megvalósított programszerkezettel, memória- és programvezérléssel, valamint a programból meghívható funkciókkal is egyre inkább a UNIX lehetőségei állnak a programozó rendelkezésére.

Mindezek ellenére le kell szögeznünk, hogy az MS-DOS nem a CP/M és a UNIX keveréke, hanem az IBM PC/XT hardware-lehetőségeire épülő, azokat elég jól kihasználó általános és önálló operációs rendszer, amely sok mindent vett át korábbi rendszerekből, de amelyben sok új elemmel is találkozhatunk. Elég ké-

nyelmes, barátságos, aránylag egyszerű és hatékony (különösen, ha megtámogatjuk néhány "mankóval", mint DOSEDIT stb.). Felhasználói szinten napok alatt ismerhető meg, programozói szinten kicsit nagyobb elmélyülést igényel. Azonban sokkalta gyorsabban és könnyebben tanulhatjuk meg, mint bármelyik nagygépes operációs rendszert vagy a UNIX-ot. Az ilyen kis rendszereknek ez felbecsülhetetlen előnye.

Az MS-DOS operációs rendszer általános ismertetéséhez hozzátartozik még annak ismételt hangsúlyozása, hogy az MS-DOS egyfelhasználós, egy időben egy programot futtató operációs rendszer. Egyfelhasználós, mert felhasználójával egy logikai csatornán kommunikál, és az egész rendszer úgy épül fel, hogy sem hardware-, sem pedig software-szempontból nem támogatja több egyenrangú logikai csatorna létrehozását, így nem teszi lehetővé több felhasználó egyidejű és egyenrangú kiszolgálását. Szigorúan egy programot futtat egy időben (annak ellenére, hogy több program lehet egyszerre a memóriában, és ezek akár hívogathatják is egymást), mert az MS-DOS nem reentráns, azaz nincs lehetőség arra, hogy rendszerszintről újra meghívjuk a rendszert. Ez lehetetlenné teszi azt, hogy bármilyen úton-módon is multiprogramozhatóvá tegyük az MS-DOS harmadik verziójáig ismeretes változatokat. Annak nem volna ugyan akadálya, hogy az időzítő áramkör megfelelő felhasználásával működtessünk egy diszpécserprogramot, amely időről időre cserélné az éppen aktív programot. De előfordulhatna az az eset, hogy a mostanáig aktív program éppen MS-DOS hívást hajt végre, mikor az időzítő megszakítja futását. Mivel a diszpécserprogram is kénytelen néhány MS-DOS hívást végrehajtani, meg kell szegnie a fenti törvényt.

Van néhány olyan kivételes MS-DOS funkció, amelyet mindig, vagy majdnem mindig meg lehet hívni. Ezekre a megfelelő helyen kitérünk majd.

Végezetül néhány olyan megállapodást közlünk, melyek fontosak a könyv olvasásához. A legtöbb megadott számérték (interruptok sorszáma, funkciókódok, hibakódok stb.) hexadecimális. Ezt a "H"-val csak a példaprogramokban emeljük ki. Azokat az értékeket írtuk csak decimális számrendszerben, amelyek szintel sem képzelhetők másként. Például a 64 Kb decimális, de az "a szó 12. bitje" kifejezésben a 12 is. Ebben a kérdésben, sajnos, az egész szakirodalom elég következetlen.

Az egyes fogalmakat, ahol csak értelmesen lehetett, magyarul írtuk. A lehető legtöbb helyen (de legalább az első előfordulásnál) megadjuk a kifejezések "hivatalos" angol alakját is.

Felhasznált irodalom

(1) IBM Personal Computer Software
DOS Technical Reference Manual, version 2.10

(2) IBM Personal Computer Software
Hardware Technical Reference Manual, version 2.10

(3) IBM Personal Computer Software
DOS Technical Reference Manual, version 3.00

(4) IBM Personal Computer Software
Hardware Technical Reference Manual, version 3.00

(5) IBM Personal Computer Software
Disk Operating System, version 3.00

(6) IBM Personal Computer Software
DOS Technical Reference Manual, version 3.10

(7) Microsoft Corporation
Microsoft MS-DOS 3.1 Programmer's Reference Manual

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

I. Amikor a program belép

Az MS-DOS beolvassa a felhasználó parancsát, amelynek első szava (az első szóközig vagy tabulátor karakterig) a parancs neve, a további szavak a paraméterek (lásd a fejezet végén a programindítási példákat). Ezután a következő módon végzi el a parancs értelmezését és végrehajtását:

(1) - a rendszer megvizsgálja, hogy a kapott parancs nem belső parancs-e. Ha igen, akkor azonnal végrehajtja. Ha nem belső parancs, akkor először a parancsban megadott nevű EXE, majd COM, végül pedig BAT típusú file-t keresi előbb a pillanatnyilag kiválasztott lemez kurrens directory-jában, majd pedig a rendszer PATH-ában szereplő aldirectorykban. Ha BAT típusú (parancs-) fileról van szó, azt a rendszer "batch" processzora fogja végrehajtani. Ha COM vagy EXE típusú file-t kell lefuttatni, akkor:

(2) - az MS-DOS létrehoz egy memóriablokkot a legalacsonyabb című szabad területen, és ide bemásolja saját környezetét (environment) az elindított program számára;

(3) - létrehoz egy újabb memóriablokkot, amely az első szabad paragrafustól a memória tetejéig tart. Ezt nevezzük programterületnek (eredeti [nem túl szerencsés] nevén Program Segment).

Megjegyzés: ismert, hogy az operációs rendszer egy része (lásd "MS-DOS memóriatérkép" függelék) a RAM terület legmagasabb címein helyezkedik el. Amennyiben a program módosítja ezt a területet (jogában áll, hiszen az övé), akkor kilépése után az MS-DOS újratölti önmagát. Az MS-DOS úgy érzékeli a módosítást, hogy megvizsgál egy ellenőrző összeget. Ha ez megváltozik, akkor tudja, hogy a megfelelő részeket újra be kell tölteni a lemezről. Ilyen esetben arról a lemezegységről kísérli meg az újratöltést, amelyről a rendszert utoljára beolvastuk (ha ez egy floppy volt, akkor oda egy rendszerlemezt kell berakni).

(4) - a program számára lefoglalt terület kezdetén előkészíti a programszegmenst leíró blokkot (angol nevén Program Segment Prefix, a továbbiakban PSP, lásd a hasonnevű fejezetben).

COM file esetén a PSP a program egyetlen szegmensének első 256 byte-ján, EXE file esetén külön 256 byte-nyi [16 paragrafusnyi] szegmensben helyezkedik el. A PSP-ben szerepel az MS-DOS hívási címe, a kilépési címek, az előbb előkészített környezet szegmenscíme, a program memóriablokkjának teteje (az első el nem érhető paragrafus sorszáma, azaz szegmenscíme stb.);

(5) - a PSP előkészítése során elvégzi a parancssor további szavainak feldolgozását. Ha a parancssor átirányítási paramétereket tartalmaz (lásd az "MS-DOS standard file-jai"), akkor elvégzi a standard file-ok átirányítását. Magukat az átirányítást vezérlő szavakat nem kapjuk meg paraméterként. Ha a parancssor file-névként értelmezhető szavakat tartalmaz, akkor előkészíti a PSP elsődleges és másodlagos FCB-területét. Végül magát a parancssort bemásolja a PSP-terület felső felére (részletesen lásd a "Program Segment Prefix" fejezetet);

(6) - beolvassa a COM vagy EXE file tartalmát a programterületnek a PSP-t követő területére;

(7) - EXE típusú file esetén elvégzi a program relokálását a tárban (lásd az "EXE típusú file-ok headerje" fejezet "Relokáció" alfejezetét);

(8) - a regiszterek tartalmát az alábbi módon készíti elő a program elindítása előtt:

AL - jelzi, hogy az első paraméter, mint file-specifikáció, tartalmaz-e illegális lemezegység-megadást (pl. "Q:"). AL értéke 00, ha nem, és FF, ha igen;

AH - jelzi, hogy az első paraméter, mint file-specifikáció, tartalmaz-e illegális lemezegység-megadást (pl. "T:"). AH értéke 00, ha nem, és FF, ha igen;

CX - a programfile hosszára utaló információt tartalmaz, a file hossza modulo 256 (ti. a memórialap, a PAGE hossza szerinti maradékot). Ezt programhelyesség-ellenőrzésre lehet felhasználni: ismerjük a program hosszát a fejlesztés befejezésének idején. A program belépésekor így ellenőrizhetjük a CX-ben kapott értéket. Ha valaki belenyúlt a kész programba, akkor a hossz változását észrevesszük.

Az egyéb regiszterek állása EXE és COM program esetén:
a) Ha a program EXE típusú:

DS, ES - a PSP-re mutatnak
SS:SP - a LINK által meghatározott értéket kapják, a stack-szegmens kezdőcímét, illetve a stack-terület tetejét címzik.
A többi regiszter értéke 0.

Végül egy hosszú ugrással eléri, hogy:
CS:IP - a program belépési pontját címzi

b) Ha a program COM típusú:

CS, SS, DS, ES - a közös program-, adat- és stack-szegmensre címeznek
SP - a stack-terület tetejére mutat (értéke a program hosszától függő páros szám)
A többi regiszter értéke 0.

Végül egy hosszú ugrással eléri, hogy:
CS:IP - a program belépési pontját címzi, azaz IP értéke 100H.

Programindítási példa

Mint minden program, az általunk írtak is tetszőleges számú paraméterrel hívhatók meg. Legyen a program neve PROG.EXE. Ezt (többek között) a következőképpen indíthatjuk el:

```
C>PROG                          nincs paraméter
C>PROG x@abc z,ert 56!w&        három paraméter
C>PROG egyik.txt semelyik.ppp   két file-név a paraméter
C>PROG a:egy.txt f:egy.ppp      hibás a második paraméter
```

Az első esetben a PSP nem tartalmaz semmilyen, a paraméterekre vonatkozó információt.

A második esetben három teljesen "értelmetlen" paramétert adtunk meg. Ezek a paraméterek nem értelmezhetők file-névként. Ebben az esetben a paraméterek a PSP felső felében, a hexadecimális 80. pozíción olvashatók el. Az első byte tartalmazza a

paraméterek összhosszúságát karakterekben (a példában a PROG parancsot követő szóközzel együtt ez 18). A további byte-okon (a hexadecimális 81. pozíciótól kezdve) következnek a paraméterek karakterei; az utolsó karakter (az "&" jel) után még találunk egy kocsivissza-karaktert is (ASCII-kódja decimális 13).

A harmadik esetben (ez a leggyakoribb) két szabályos filenevet adtunk meg paraméterként. Ekkor a PSP két helyen is tartalmaz a paraméterekre vonatkozó információt. Az egyik természetesen a hexadecimális 80. pozíció, ahol ugyanúgy megtalálunk minden egyes karaktert, mint a második esetben. De, mivel a paraméterek file-névként értelmezhetők, az MS-DOS kitölti a PSP elsődleges és másodlagos File Control Block-ját ("File Control Block" fejezet) az első és második paraméter alapján. Tehát a hexadecimális 5C-edik pozíción találhatunk egy 0-t (amely a kurrens drive-ra utal), és nyolc byte-on (szóközökkel kiegészítve) az "egy" stringet mint file-nevet, majd a "txt" stringet mint file-típust. A másodlagos File Control Block a hexadecimális 6C-edik pozíción kezdődik. Ide kódolja le a rendszer a második paramétert, ha az file-névként értelmezhető. Mint a "Hagyományos file-kezelés" fejezetben olvasható, az így előkészített File Control Block éppen egy file átnevezésére lenne alkalmas. Amennyiben külön akarjuk használni a file-okat, akkor persze mindkét FCB-t ki kell másolnunk valahová.

Végül a negyedik esetben a második paraméter illegális lemezkódot tartalmaz. Ezért a program belépésekor AH értéke FFH (azaz -1), és AL értéke 0.

A harmadik példa átgondolása elvezet a PSP használatának legnagyobb problémájához. EXE típusú file esetén a PSP szegmenscíme különbözik az adatszegmens(ek) címétől, így vagy az ES segítségével címezzük meg a PSP-t (ami egy kicsit kényelmetlen dolog), vagy a DS regisztert használjuk váltva a PSP és (valamelyik) adatszegmens megcímzésére (ekkor viszont nem látjuk egyszerre mindkettőt). A gyakorlat az lehetne, hogy a PSP legfontosabb elemeit a program belépése után átmásoljuk az adatszegmensre, hogy később ne legyünk kénytelenek a PSP-re hivatkozni.

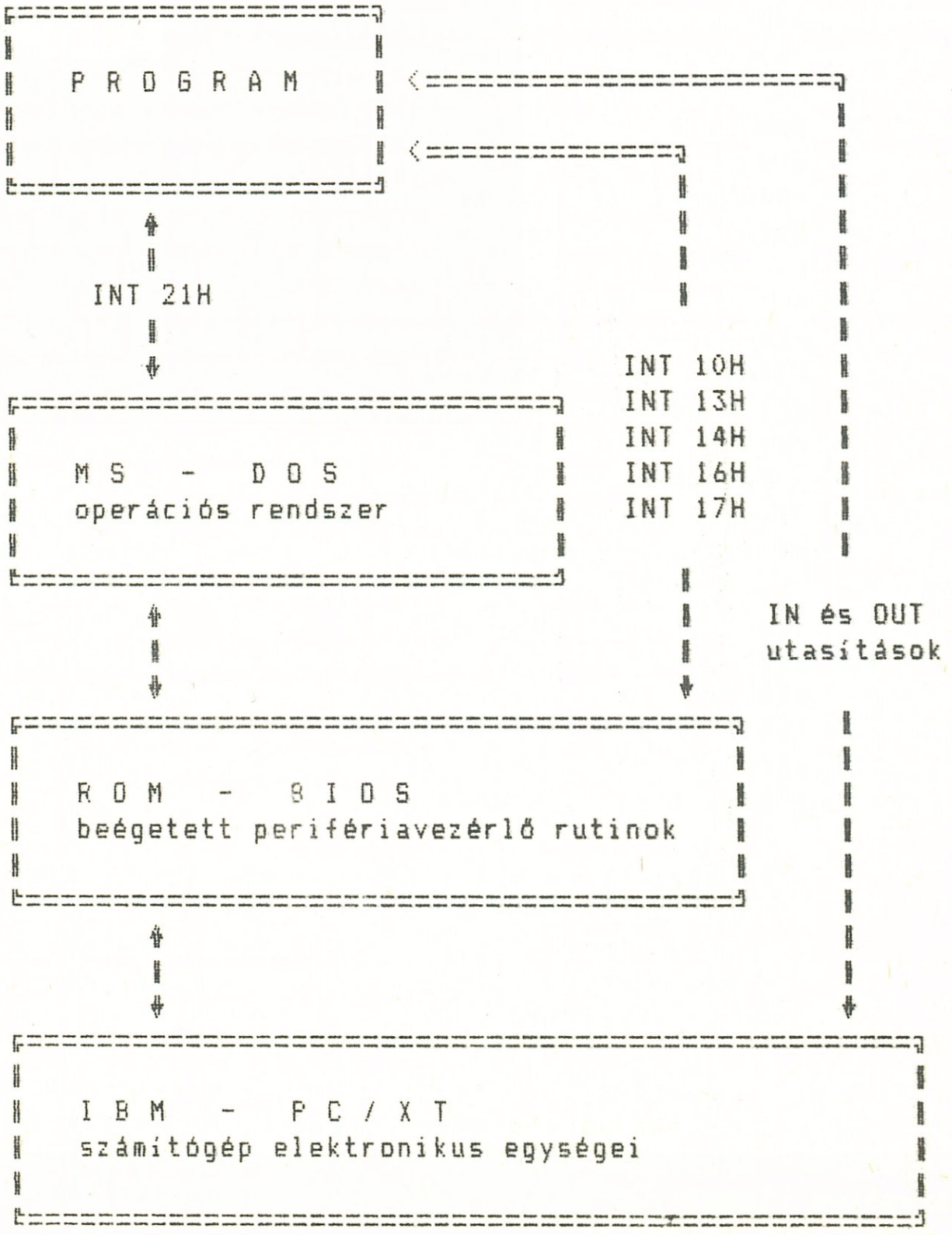
COM típusú file esetén nem szükséges ez a bonyodalmas előkészítés, hiszen minden szegmensregiszter ugyanarra a címre mutat; ez a cím pedig a PSP kezdőcíme.

A PSP számos olyan elemmel rendelkezik, amely túlmutat a program keretein. Ilyenek a kilépési címek a PSP elején (mint "Az interruptok funkciói" fejezetben olvasható, ezek a címek

bizonyos interrupt-vektorokban is megtalálhatók). A gyakorlatban ezeknél sokkal fontosabb a környezet (lásd ott). Szegmensének címét a PSP hexadecimális 2C-edik pozícióján egy szó tartalmazza. Ha a környezet tartalmát el akarjuk olvasni, ezt a szegmenscímet be kell töltenünk valamelyik szegmensregiszterbe, majd pedig a 0 offsettől kezdve elolvashatjuk a környezetet alkotó stringeket. A környezet stringjeinek elérésére lásd a "Memóriakezelés és programvezérlés" fejezetet, ahol szerepel egy olyan példaprogram, amely elolvassa és kinyomtatja környezetét.

II. Az aktív program és a rendszer

Egy MS-DOS alatt futó program, a rendszer és a gép kapcsolatait több részre bontva tárgyaljuk. A logikai kapcsolatokat a következő ábra szemlélteti:



Mint az ábráról látható, a számitógép elektronikus berendezéseinek közvetlen programozását az ún. ROM BIOS látja el. Az angol ROM rövidítés (Read Only Memory, csak olvasható memória) mutatja, hogy ezt a rutincsomagot felül nem irható memóriába égették be, igy mindig megtalálható a memória megfelelő területén (magas címeken). Maga a BIOS név Basic Input/Output System-et, be- és kiviteli alaprendszert jelent. Ez a rutincsomag tehát (egyebek között) azokat a programokat tartalmazza, amelyek a perifériavezérlők közvetlen programozását, "meghajtását" végzik. Ezek angol neve "driver". Maga az MS-DOS operációs rendszer minden I/O (input-output) tevékenységet szintén a BIOS drivereire bíz.

Az MS-DOS operációs rendszer lényeges részei a tár alacsony címeire töltődnek. Lényegében emiatt választjuk el élesen a ROM BIOS rutinoktól: míg a BIOS rutinok mindig elérhetők a tárban, az MS-DOS részei rendszerindításkor RAM területre töltődnek be a rendszerlemezről. Ez a legfontosabb különbség a BIOS és a DOS rutinjai között. Az MS-DOS legalapvetőbb szolgáltatásai a filekezelés, a processz-vezérlés és a memóriakezelés.

Az általunk irott program futás közben igénybe veheti az operációs rendszer szolgáltatásait, fordulhat közvetlenül a BIOS-hoz, vagy (ha ehhez a programozónak kellő felkészültsége van) input és output utasítások kiadásával közvetlenül is fordulhat a perifériavezérlőkhöz (és egyéb egységekhez, mint például az interrupt-vezérlő áramkör).

A ROM BIOS hívásait és a legfontosabb közvetlen programozási lehetőségeket a 3. kötetben tárgyaljuk. Ennek során olyan ismeretekre tehet szert az Olvasó, amelyek segitségével az MS-DOS input-output lehetőségeinél elegánsabban és rugalmasabban végezheti el a legfontosabb műveleteket.

> Megjegyzés: ha ezt kihasználjuk, akkor semmi garancia nincs arra, hogy programunk más (IBM-"kompatibilis") gépen is futni fog, vagy pontosan ugyanúgy fog futni! Az egyes utánzatokban a ROM BIOS lényegesen különbözhet az eredetitől.

Az olyan lehetőségek kihasználásához, mint a hanggenerálás vagy speciális hadware-elemek kezelése, közvetlen programozás szükséges. Általános alapelvként szögezhetjük le azonban, hogy közvetlen input és output utasitásokat, hacsak nem elengedhetetlen, sose adjunk ki.

A 3. kötetben szerepel egy olyan fejezet is, amely egyfelől az Intel8087 koprocesszort írja le, másfelől pedig programozástechnikai kérdésekkel (a számábrázolás problémáival) foglalkozik. Ezen ismeretek birtokában jobban ki tudjuk majd használni a gép aritmetikai sajátosságait még koprocesszor nélkül is.

Az általunk írt program az MS-DOS operációs rendszer egyes funkcióit és a ROM BIOS rutinjait a software interrupt rendszer segítségével hívhatja meg. Egy software interrupt lényegében egy távoli (azaz más szegmensre végrehajtott) indirekt szubrutinhívás. Ebben az esetben a hívó és a meghívott program regiszterekben kommunikál egymással. Általános megállapodás, hogy egy rutincsaládot egy adott sorszámú software interrupt aktivizálásával hívhatunk meg. Ilyen esetben az AH regiszter tartalmazza a család kért tagjának sorszámát, a többi regiszterben pedig az egyéb paramétereket adjuk meg.

Az MS-DOS funkciói lényegében mind a hexadecimális 21 interrupt aktivizálásával hívhatók meg. A funkciók kódjai pedig a 00-tól a hexadecimális 62-ig terjednek. Bemenetként általában egy adatot adhatunk még meg, ezt 8 bites adat esetén a DL, 16 bites adat esetén pedig a DX regiszterben kell elhelyezni. Ezután adhatjuk ki az INT 21H utasítást, amely a vezérlést a DOS megfelelő pontjára adja. Mikor a programunk futása folytatódik az INT-et követő utasítással, általában az AL regiszter tartalmaz egy visszatérési kódot. Ez megállapodás szerint 0, ha a DOS sikerrel hajtotta végre a kérést; -1, amennyiben valami hiba következett be. Számos olyan MS-DOS funkció van, ahol ez a szegényes (még a CP/M operációs rendszerből származó) megállapodás nem elegendő. Ilyenkor a DOS saját alapelveit megszegve további paramétereket vár más regiszterekben is (pl. a BX-ben és a CX-ben). A válaszként adott érték is lehet sokkal bonyolultabb; vannak esetek, mikor a DOS az AL regiszter mellett a STATUS-ban is szolgáltat információt: a Zero és a Carry bitek tartalmazhatnak logikai értékeket, amelyek módosíthatják az AL-ben vagy AX-ben visszaadott érték jelentését.

A ROM BIOS egy-egy funkciócsaládját (melyek az egyes perifériavezérlők programozására szolgálnak) különböző sorszámú interruptok aktivizálásával lehet meghívni. Ekkor is általános elv, hogy a család meghívott tagjának sorszámát helyezzük el az AH regiszterben. A többi regiszter a hívott rutin számára nélkülözhetetlen paramétereket tartalmazza. Ezek részletes leírását lásd a 3. kötet megfelelő fejezeteiben.

III. Az interruptok funkciói

Mint ismeretes, az Intel 8088 mikroprocesszor 256 szintű, vektoros interrupt rendszerrel rendelkezik (lásd pl. e sorozat első kötetét). Az MS-DOS operációs rendszer széleskörűen kihasználja az ebből eredő lehetőségeket. Szinte minden rendszerfunkció egy software interrupt aktivizálásával hívható meg. Nagyon fontos, hogy legalább nagy vonalakban tisztában legyünk az interruptok kiosztásával.

A memória elején elhelyezkedő interrupt-vektorok címzik meg az egyes interruptok aktivizálásakor belépő rutinokat. Igen fontos elv, hogy ha nem szükséges, az interrupt-vektorok tartalmát ne változtassuk meg, vagy ha ez elkerülhetetlen, akkor a program futása végén állítsuk helyre. Az interrupt-vektorokat programból direkt címzéssel ne módosítsuk, hanem használjuk fel az erre szolgáló MS-DOS funkciókat (lásd "Az interrupt-vektorok kezelése" fejezet).

III.1. Processzor-interruptok

Az alábbi interruptokat maga a processzor generálja bizonyos feltételek esetén. Más célra nem használhatjuk őket. Hardware-úton nem válthatjuk ki ezen interruptok bekövetkezését, és kerüljük az ezekre vonatkozó interrupt-utasítások kiadását is!

00 Osztási túlcsordulás, zerodivide
Ezt az interrupt-rutint a processzor abban az esetben hívja meg, ha ún. osztási túlcsordulás lép fel, azaz egy osztási utasítás (DIV vagy IDIV) végrehajtása után az eredmény nem fér el a számára kijelölt regiszterben. Az interrupt-vektor normális esetben egy DOS-rutint címez, amely abortálja az eseményt kiváltó programot.

01 Lépésenkénti megszakítás, single-step
Ha a STATUS regiszter T (trap) bitjét beállítottuk, akkor ez az interrupt aktivizálódik minden egyes utasítás végrehajtása után. Ha ezt az interrupt-vektort úgy készítettük elő, hogy egy értelmes nyomkövető rutint címezzen, akkor ezt egy valódi hardware-vezérelte nyomkövetésre használhatjuk.

02 NMI - nem maszkolható interrupt
Az NMI (Non Maskable Interrupt) egy sajátos hardware interrupt típus. Mint neve is mutatja, nem tiltható le. Külön csatlakozásra kell bekötni, és az összes többi interrupttól függetlenül éleződhet.
Az IBM-PC/XT-n nem használatos.

03 Töréspont, breakpoint
Ez az interrupt a paraméter nélküli interrupt, az

INT

utasítás hatására aktivizálódik. Mivel az utasításnak egy byte hosszú a gépikódja, bármely gépi utasítás operációs kódjának helyére "belopható". Mikor a tesztelni kívánt program eléri ezt az utasítást, automatikusan aktivizálódik a 03 interrupt, és a vezérlés a már előkészített nyomkövető rutinra adódik.
Megjegyezzük, hogy a DEBUG programban e célra az

INT 3

utasítást használhatjuk, ha "Assemble" módban valamilyen programot írunk, és valahová egy fix töréspontot akarunk beiktatni.

04 Túlcsordulási rutin hívása, overflow
Ezt az interrupt-rutint az e célra szolgáló

INTO

utasítással aktivizálhatjuk. Akkor hajtódik csak végre az interrupt, ha valamely korábbi aritmetikai művelet során a STATUS regiszter Overflow bitje egyes értéket kapott.
A vektort persze át kell állítanunk a saját, túlcsordulást lekezelő rutinunk címzésére.

Megjegyzés: alapértelmezésben a 01, 03 és 04 interrupt-vektorok a memória azonos pontjára mutatnak, ahol nemes egyszerűséggel egy IRET utasítás található.

III.2. A PC/XT BIOS interruptjai

Az itt felsorolt három interruptot (és az összes hátralevőt) már az operációs rendszer használja - felhasználásuk nem a gép tervezői, hanem a rendszert megalkotó programozók szándékán múlik. Egyes interruptok (melyeket szabad interruptoknak hívunk) software interruptokként rendelkezésünkre állanak.

05 Képernyőtartalom nyomtatása (hard copy)
A vektor egy ROM BIOS rutinra mutat, amely kiolvassa és nyomtatja a képernyő aktuális tartalmát.

06, 07 Nem használt

III.3. A PC/XT hardware interruptjai

A PC/XT hardware interruptjait az Intel 8259 típusjelű interrupt-vezérlő kezeli. Ennek nyolc vonalát hozzárendelhetjük bármely nyolccal osztható interrupt-címen kezdődő nyolc szomszédos interrupt-vektorhoz.

Az itt listázott interrupt-vektorokhoz a 8259 vonalai emelkedő sorrendben vannak hozzárendelve. Minden interrupt után megadjuk a 8259 megfelelő interrupt vonalának számát.

08 Idő: az óra kezelése
Az Intel 8253 típusjelű (háromcsatornás) timer áramkör másodpercenként kb. 18.2-szer generálja ezt az interruptot. Emögött egy ROM BIOS rutin van, amely kezeli a gép belső órájaként használt számlálókat.
Fontos tudnunk, hogy ez a rutin kilépése előtt még meghívja az 1C interruptot, amely alapértelmezésben üres, azaz a ROM BIOS "DUMMY_RETURN" nevű belépési pontjára mutat, ahol egy IRET utasítás szerepel. De a felhasználó átirányíthatja az 1C interruptot, ahol tetszőleges tevékenységet végezhet, például kiírhatja a rendszeridőt. Azonban ennek kihasználásával legyünk óvatosak, vigyázzunk arra, hogy csak igen rövid időt töltsünk el interrupt-szinten, mivel ez az interrupt igen gyakran aktivizálódik, és súlyos problémák léphetnek fel akkor, ha hosszabb időre megfogjuk a processzort.
(8529 IRQ0)

09 A klaviatúra hardware-interruptja
Beolvassa a klaviatúra által küldött kódot, értelmezi és elhelyezi egy ciklikus bufferben. Amikor bármely szabványos funkcióval (legyen az akár DOS, akár ROM BIOS funkció) beolvasást hajtunk végre, akkor ebből a bufferből kapjuk meg a legrégebben érkezett és eddig még ki nem olvasott karaktert.
Fontos tudnunk, hogy ez a rutin megvizsgálja a különböző speciális billentyűkombinációkat:

CTRL-ALT-DELETE
CTRL-BREAK
CTRL-NUMLOCK

A CTRL-ALT-DELETE billentyűkombináció hatására a rutin interrupt-szintről újraindítja a rendszert.
A CTRL-BREAK billentyűkombináció esetén (amely a futó program abortálását jelenti) kiüríti a klaviatúra buffert (!), és egy (0,0) byte-párt helyez el benne (innen egyébként kiolvasáskor felismerhető a CTRL-BREAK leütése), beállítja a BIOS_BREAK nevű belső változó egy bitjét, és (számunkra legfontosabb:) meghívja az 1B interrupt-vektor által címzett rutint. Ezt arra használhatjuk, hogy programunk kilövése esetén valami fontos tevékenységet még interrupt-szinten elvégezzünk. Erre a célra azonban inkább a 23 interruptot használjuk!
Végül a CTRL-NUMLOCK kombináció arra szolgál, hogy felfüggesszük az aktív program futását. Ekkor a rutin beállít egy belső flaget, és addig vár, míg le nem ütnek egy billentyűt, amely bármi lehet a NUMLOCK-ot, valamint a CTRL-ALT-DELETE és a CTRL-BREAK kombinációt kivéve. Ekkor törli a flaget, s az aktív program futása csak ezután folytatódhat.
(8529 IRQ1)

0A Nem használt
(8529 IRQ2)

0B Az aszinkron vonal hardware-interruptja
Bővebben a 3. kötet "Az aszinkron vonal (RS232)" fejezetében.
(8529 IRQ3)

0C A másodlagos aszinkron vonal hardware-interruptja
Bővebben a 3. kötet "Az aszinkron vonal (RS232)" fejezetében.
(8529 IRQ4)

0D A képernyő lefutási interruptja
Bővebben a 3. kötet "A képernyő" fejezetében.
(8259 IRQ5)

0E A diskette hardware interruptja
Bővebben a 3. kötet "Diskette-kezelés" fejezetében.
(8259 IRQ6)

0F A nyomtató hardware-interruptja
Alapértelmezésben egy IRET utasításra mutat, akár a 01, 03 és 04 interrupt-vektorok.
(8259 IRQ7)

III.4. BIOS belépési pontok

10 Képernyő-driver hivása
Részletes leírását lásd a 3. kötet "A képernyő" fejezetében.

11 A gép hardware-elemeinek lekérdezése
Ez az interrupt-hivás arra szolgál, hogy információt nyerjünk a géphez csatolt kiegészítő berendezésekről. Ezek listáját a bekapcsoláskor végrehajtott teljes ellenőrzés készíti elő. Részletes leírását lásd a 3. kötet "Egyéb ROM BIOS interruptok" fejezetében.

12 Memóriahossz ellenőrzése
Ez az interrupt megadja a RAM terület hosszát 1 Kb-os egységekben. Részletes leírását lásd a 3. kötet "Egyéb ROM BIOS interruptok" fejezetében.

13 Floppy-disk driver hivása
Részletes leírását lásd a 3. kötet "Diskette-kezelés" fejezetében.

14 Aszinkron vonali driver hívása
Részletes leírását lásd a 3. kötet "Az aszinkron vonal (RS232)" fejezetében.

15 Kazetta-driver hívása (csak az IBM PC-ben!)
Részletes leírását lásd a 3. kötet "Egyéb ROM BIOS interruptok" fejezetében.

16 Klaviatúra-driver hívása
Részletes leírását lásd a 3. kötet "A klaviatúra" fejezetében.

17 Nyomtató-driver hívása
Részletes leírását lásd a 3. kötet "A nyomtató" fejezetében.

18 ROM BASIC belépési pont

19 Rendszerindítás, bootstrap
Leírását lásd a 3. kötet "Egyéb ROM BIOS interruptok" fejezetében.

1A A belső óra lekérdezése vagy módosítása
Részletes leírását lásd a 3. kötet "Egyéb ROM BIOS interruptok" fejezetében.

1B CTRL-BREAK észlelésekor meghivandó felhasználói rutin
A klaviatúra hardware interrupt-rutinja aktivizálja CTRL-BREAK karakter észlelése esetén.

1C Timer felhasználói interrupt
A timer hardware interrupt-rutinja (08 interrupt) aktivizálja minden egyes lefutásakor. Programozóként speciális időzítési tevékenységekre használhatjuk fel.

III.5. A BIOS paraméterek címei

1D Video paramétertábla címe
Részletes leírását lásd a 3. kötet "A képernyő" fejezetében.

1E Diskette paramétertábla címe
Részletes leírását lásd a 3. kötet "Diskette-kezelés" fejezetében.

1F Video grafikus karaktergenerátor címe
Részletes leírását lásd a 3. kötet "A képernyő" fejezetében.

III.6. Az MS-DOS számára fenntartott interruptok

20 Kilépés hagyományos módon
Kilépéskor használhatjuk közvetlenül ezt az interruptot is. A DOS-hívások között is szerepel három kiléptető függvény (0, 4C és 31).
Fontos: ha így akarunk kilépni, akkor az interrupt aktivizálása előtt a CS regiszterben a Program Segment Prefix (lásd "Program Segment Prefix" fejezet) címének kell lennie. Ebből tulajdonképpen következik, hogy csak COM típusú programokból tudunk így kilépni. EXE típusú programokban ezt a kilépést csak trükkösen lehet kihasználni - lásd első kötet (Assembly alapismeretek), ahol kellő rendszerismeret hiján azt a trükköt vetettük harcba, hogy a vezérlést a Program Segment Prefix (lásd "Program Segment Prefix" fejezet) 0. offsetjére adtuk át, ahol egy INT 20 utasítás van.
Ha csak lehet (és miért ne lehetne? nincs semmi kizáró ok), ehelyett használjuk mindig a 4C kódú DOS funkciót!

21 MS-DOS funkciók hívása
Ezek leírását tartalmazza ez a kötet az V-XIII. fejezetekben. E kötet fő célja éppen e funkciók részletes ismertetése.

22 Kilépési cím
Ez a szegmens:offset pár az a cím, ahová a vezérlés a program kilépésekor adódik. Amikor a DOS elindít egy programot, akkor a 22 interrupt-vektor tartalmát átmásolja annak Program Segment Prefixére. A 22 interruptot nem szabad programból aktivizálni, ez az MS-DOS kizárólagos joga!

Amennyiben feltétlenül szükséges egy saját kilépési interrupt-rutin megírása, akkor a megfelelő MS-DOS funkciók segítségével mentsük el és változtassuk meg a 22 interrupt-vektor tartalmát.

23 A CTRL-BREAK kilépés (programkilövés)

Amikor a DOS elindít egy programot, akkor a 23 interrupt-vektor tartalmát átmásolja annak Program Segment Prefixére. Ezt az interruptot nem szabad programból aktivizálni, ez az MS-DOS kizárólagos joga!

Ha a felhasználó kiadja a CTRL-BREAK parancsot, akkor az MS-DOS, amilyen hamar csak lehet, aktivizálja a 23 interruptot. A DOS minden standard file-kezelő funkció futása során megvizsgálja, nem érkezett-e CTRL-BREAK a klaviatúráról. Amennyiben az MS-DOS BREAK kapcsolóját beállitottuk (lásd "Kisegítő MS-DOS funkciók" fejezet), akkor a rendszer minden funkcióhivás esetén végrehajtja ezt a vizsgálatot.

Amennyiben saját CTRL-BREAK rutint kell irnunk, akkor a megfelelő funkciókkal mentsük és módositsuk a 23 interrupt-vektor tartalmát úgy, hogy az saját rutinunk kezdőcímére mutasson. CTRL-BREAK észlelése esetén az operációs rendszer aktivizálja ezt az interruptot, és ott elvégezhetjük a szükséges tevékenységeket (pl. file-ok lezárása stb.).

Saját CTRL-BREAK kezelő rutinunkból, kivételesen, kétféleképpen térhetünk vissza. Ha minden regisztert mentettünk, akkor helyreállitásuk után kiadhatunk egy IRET utasitást. Ezt végrehajtva programunk futása azon rendszerhivás után folytatódik, amelynek során a rendszer a CTRL-BREAK lenyomását észlelte.

Visszatérhetünk egy hosszú RET utasitással is. Ekkor a Carry flaget használjuk fel annak jelzésére, hogy abortálni kell-e a programot, vagy folytatódhat a futása. Ha a Carry értéke 1, akkor az MS-DOS abortálja a programot, ellenkező esetben folytatódik a program futása.

A 23 interrupt felhasználására példát olvashatunk az "Ami a példaprogramokból kimaradt" fejezetben.

24 Kilépési cim kritikus hiba esetén

Amennyiben valamilyen input-output vagy memóriakezelő tevékenység során kritikus hiba lép fel (ez jellegzete-

sen lemezkezelési hiba), az MS-DOS aktivizálja a 24 interruptot. Amikor a DOS elindít egy programot, akkor a 24 interrupt-vektor tartalmát átmásolja annak Program Segment Prefixére. Ezt az interruptot nem szabad programból aktivizálni, ez az MS-DOS kizárólagos joga!
Egy példával illusztráljuk a kritikus hiba bekövetkezését és az akkor lejátszódó eseményeket. Floppy-diskről szeretnénk egy directory-listát kérni, de elfelejtettük lezárni a lemezegységet. A DIR parancs után a rendszer hibakezelő rutinjától a

Not ready error reading drive A
Abort, Retry, Ignore?

üzenetet kapjuk (miszerint az A: lemezegység nem üzemkész, nem lehet róla olvasni). Ekkor lezárjuk az ajtaját; jó lélekkel mondhatjuk azt, hogy "Retry", próbáljuk meg újra. Ha azonban a lemez formázatlan, akkor a rendszer a

Data error reading drive A
Abort, Retry, Ignore?

szöveget írja ki (olvasás közben adathiba lépett fel). Ekkor nincs értelme tovább kísérletezni, válaszunk csak "Abort" lehet. Ha egy olyan lemezt akarunk XT-n olvasni, melyet előzőleg AT-n (nagykapacitású lemezegységgel) írtunk, akkor érdemes megkísérelni a művelet folytatását ("Ignore"), mert elképzelhető, hogy a hibák ellenére legalább a directory-t ki tudjuk listázni.

> Megjegyzés: többször sikerült már nagykapacitású lemezegységen írt file-okat olvasni egy szokásos kapacitású egységen az IGNORE parancs sűrű használatával. A tapasztalat azonban azt mutatja, hogy ha a DOS kritikus hibát észlel, akkor ott valóban nagy baj van. Sokszor egy átmásolt filenak csak egy-egy byte-ja sérült, ezek azonban általában nehezebben javíthatók, mint ha hemzsegnének az átviteli hibák.

Lényegében négyféleképpen cselekedhetünk kritikus hiba esetén:

ABORT a programot abortálni kell
(FAIL) a művelet eredményéről lemondva folytathatjuk a munkát (csak belső lehetőség, a felhasználó ezt nem választhatja)
RETRY a műveletet újra meg kell kísérelni
IGNORE a művelet eredményét a hiba ellenére fel kell használni

A rendszer megvizsgálja, hogy válaszunk megfelel-e a hiba természetének. Ha "RETRY"-t választottunk, de az adott hiba esetén nem megengedett, akkor ezt a rendszer a "FAIL"-el helyettesíti. Ha "IGNORE"-t választottunk, és ez nem megengedett, a rendszer szintén "FAIL"-el helyettesíti. Végül, ha a hibakezelő FAIL-t választ, és ez nem megengedett, akkor a rendszer "ABORT"-ra váltja (amely mindig megengedett).

Ha a kritikus hiba lekezelésére saját rutint kell írnunk, akkor az erre szolgáló MS-DOS funkciók segítségével mentsük és változtassuk meg a 24 interrupt-vektor tartalmát úgy, hogy az saját rutinunk kezdőcímére mutasson.

A kritikus hiba-kezelő rutin a vezérlést az alábbi körülmények között kapja meg:

a) A rendszer háromszor kísérli meg a kívánt műveletet, majd a szükséges előkészítés után kiadja az INT 24 utasítást. A processzor "disable interrupt" módban van. AX a hiba helyére, DI a hiba okára utaló kódot tartalmaz.

<u>DI alsó byte-ja (a felső byte meghatározatlan):</u>

00 - írásvédett lemezt próbáltunk írni
01 - ismeretlen lemezegység
02 - a lemezegység nem üzemkész
03 - ismeretlen lemezkezelési parancs
04 - adathiba (CRC-hiba)
05 - a megadott I/O kérést leíró struktúra nem szabályos hosszúságú
06 - sávkeresési hiba
07 - ismeretlen lemezformátum
08 - szektorkeresési hiba
09 - a papír kifogyott a nyomtatóból
0A - lemezírási hiba
0B - lemezolvasási hiba
0C - általános hiba

E hibakódokat az 59 funkció (DOS_GTEXTERR) segítségével is lekérdezhetjük a kritikus hiba-kezelő rutinban. Ekkor ezeket a kódokat hexadecimális 13-al megnövelve kapjuk meg, hogy ne essenek egybe a normál hibakódokkal (melyek 1-től hexa 12-ig terjednek; lásd "Felhasználói hibakódok" fejezet). Az 59 funkció azonban a szigorúan vett hibakódok mellett számos egyéb információt is szolgáltat; lásd a "Kisegítő MS-DOS funkciók" fejezetet. A kritikus hiba kezelése során mindenképpen az 59 funkció által visszaadott információra érdemes támaszkodni.

Az AX-ben visszaadott hibakódok:

(1) Lemezkezelési hibák:

Ha AH 7. bitje 0, akkor lemezhiba következett be. Ekkor AL tartalmazza a lemezegység kódját (0=A:, 1=B: stb.). AH további bitjei a következőket jelentik:

0. bit: írás vagy olvasás közben következett-e be a hiba
0 -> olvasás
1 -> írás

2-1.bit: a lemez mely részén lépett fel a hiba:
00 -> DOS területen (rendszerfile-ok)
01 -> File Allocation Table-n
10 -> directory-területen
11 -> adatterületen

3.bit: mutatja, hogy a FAIL választása megengedett-e az adott hiba esetén
0 -> nem
1 -> igen

4.bit: mutatja, hogy a RETRY választása megengedett-e az adott hiba esetén
0 -> nem
1 -> igen

5.bit: mutatja, hogy az IGNORE választása megengedett-e az adott hiba esetén
0 -> nem
1 -> igen

(2) Hibakódok nem lemezkezelési hiba esetén:

Ha az AH 7. bitje 1, ez jelzi, hogy más természetű hiba következett be. Ez valamely karakteres átvitelt végző perifériális eszköz vagy a memóriába bemásolt File Allocation Table hibája lehet.
A szubrutin belépésekor a BP:SI regiszterpár azon eszköz leíró táblázatának (az ún. Device Block Headernek) kezdőcímét tartalmazza, amelynek kezelése során a kritikus hiba fellépett (BP tartalmazza a cím szegmes-, SI pedig az offsetrészét). A Device Block Header tartalmazza az egy szónyi hibakódot (melynek offsetje a struktúrán belül 4). A leíró blokk szerkezete a következő (elöl áll a blokk megfelelő elemének típusa [némileg pongyolán], ezt követi az elem tartalmának ismertetése):

DWORD a következő driverleíró blokk címe (és FFFFH, ha ez az utolsó)

WORD a perifériális eszköz attributuma:
15.bit 0 - blokkolt (lemez)
1 - karateres eszköz

Ha a 15. bit 1, akkor:
0.bit 1 - ez a kurrens standard input
1.bit 1 - ez a kurrens standard output
2.bit 1 - ez a kurrens NULL periféria
3.bit 1 - ez a kurrens CLOCK periféria

WORD a driver "stratégiai" (vezérlő) belépési pontja

WORD a driver interrupt belépési pontja

8 BYTE karakteres eszköz: eszköznév
lemezeszköz: a lemezegységek száma az első byte-on

A fenti bitek jelentésének részletesebb magyarázatát lásd a 44 funkció leirásánál.

b) Programunk veremterületén megtaláljuk az összes regiszternek a hibát okozó rendszerhivás előtti álla-

potát a következő sorrendben (elöl az SP által most címzett elem, tehát a verem teteje). A vezérlő és adat jellegű információkat szintezés választja el:

```
IP          az INT 24 utasitást kiadó
CS          MS-DOS program vissza-
FLAGS       térési információi
        AX
        BX
        CX      a hibát okozó DOS funkció
        DX      meghivásakor érvényes
        SI      regisztertartalmak
        DI      (funkciókód,
        BP      paraméterek stb.)
        DS
        ES
IP          a hibát okozó DOS-funkció
CS          visszatérési információi
FLAGS
```

c) Van lehetőség arra is, hogy az általunk írt kritikus hiba-kezelő rutin felhasználja a DOS azon rutinját, amely az

Abort, Retry, Ignore

üzenetet írja ki. Ehhez mindössze annyit kell tennünk, hogy egy PUSHF utasítással elmentjük a flageket, majd a vezérlést egy hosszú CALL-al átadjuk a DOS kritikus hiba-kezelő rutinjára. Ennek címe volt a 24 interrupt-vektorban a program belépésekor, s ha megváltoztattuk, akkor mentettük is (lásd feljebb). A vezérlést az elmentett címre kell átadni. Miután a felhasználó válaszolt a kérdésre, visszakapjuk a vezérlést. A felhasználó válaszának kódját a rendszer AL-ben adja vissza:

AL=0 - a válasz IGNORE volt
AL=1 - a válasz RETRY volt
AL=2 - a válasz ABORT volt
AL=3 - a válasz FAIL volt

Ezután végezhetjük el a kívánt tevékenységeket. Rutinunk nem használhat minden DOS-funkciót, csak a 01-től 0C-ig terjedőket es az 59-et (ezt csak 3.00 és későbbi MS-DOS verziókban).

Megjegyzés: ezt a korlátozást nem megkötésnek, hanem engedménynek kell tekintenünk, hiszen tudjuk, hogy az MS-DOS operációs rendszer nem reentráns, azaz MS-DOS szintről semmilyen MS-DOS interruptot nem szabad aktivizálni. A kritikus hiba kezelése közben pedig MS-DOS szinten vagyunk!

Rutinunknak szintén nincs joga megváltoztatni a BP:SI-ben megadott című Device Header tartalmát.

d) Ha a kritikus hiba-kezelő rutin vissza akar térni a programhoz, akkor minden elemet el kell vennie a veremről az utolsó három szót kivéve (mivel ez a valódi visszatérési cím), majd egy IRET utasítás kiadásával visszatérhet.

Megjegyzés: a 24 interrupt-vektor felhasználására két példát is találunk a kötet végén, az "Ami a példaprogramokból kimaradt" fejezetben.

25 Abszolút disk-olvasás - a DOS BIOS (Basic I/O System) hívása; az AL-ben megadott lemezről a DX-ben megadott logikai szektortól kezdve CX számú szektort olvas be a DS:BX-ben megcímzett területre. (A szektor, logikai szektor kifejezéseket lásd a "File Allocation Table" fejezetben.)

26 Abszolút disk-írás - a DOS BIOS hívása; a paraméterek ugyanazok, mint a 25 interruptnál.

Megjegyzés: a két utóbbi funkció arra szolgál, hogy a lemezt ne file-orientált egységként kezeljük, hanem fizikai címmel megadott szektor(oka)t írjunk vagy olvassunk. Ilyen feladatokat a programozó nem gyakran ad az operációs rendszernek, hiszen legtöbbször valamelyik, a lemezre felírt file egy adott részét szeretné elolvasni vagy módosítani.

Ezek a funkciók lehetővé teszik pl. a ROOT directory vagy akár a FAT (File Allocation Table) elolvasását is.

Rendkívüli gondossággal használjuk ezeket a funkciókat, mert egyetlen rossz kiírással jóvátehetetlenül elpusztíthatjuk egy egész lemez tartalmát!

Fontos tudnunk, hogy ez a két hívás a szegmensregiszterek és a Stack Pointer kivételével minden regiszter tartalmát elrontja.
A 25 és 26 interrupt kritikus hiba esetén nem indítja el a 24 interrupt-vektor által címzett hibakezelő rutint. Ha a funkció sikeresen futott le, akkor a Carry-ben 0 van; hiba esetén pedig 1. Ekkor AL-ben a következő hibakódokat kapjuk meg:

80 - a kontroller nem válaszolt
40 - seek-hiba: a sávkeresés sikertelen
08 - olvasás közben a rendszer CRC-hibát észlelt
04 - a rendszer a keresett szektort nem találta
03 - az írandó lemez írásvédett
02 - egyéb hiba

Az abszolút disk-kezelésre egy példát olvashatunk az "Ami a példaprogramokból kimaradt" fejezetben.

27 A program futásának megállítása úgy, hogy a program bentmarad a memóriában. A kilépés előtt DX-be kell beírni az első szabad byte címét (tehát CS:DX címzi az első szabad byte-ot).

Megjegyzés: EXE típusú program nem adhatja ki az INT 27 utasítást; a bentmaradó program legfeljebb 64 Kb hosszú lehet. Az előbbiek miatt inkább a 31 sorszámú MS-DOS funkciót használjuk erre a célra, mely mentes a fenti megkötésektől! Az INT 27 utasítás felhasználása (különösen újabb MS-DOS verziókban) ellenjavallt.

28-2E A DOS belső használatára fenntartott interruptok.

2F A 3.00 előtti verziókban a 2F interrupt is a DOS belső használatára szolgál.

MS-DOS 3.00 verzió

Aszinkron nyomtató driver (ez a PRINT külső MS-DOS parancs) vezérlésére szolgáló software-interrupt. Az alfunkciók az alábbiak (a kódot AL-ben kell megadni):

0 - a nyomtató driver állapotának lekérdezése.
Válaszok AL-ben:
00 - a driver nem installált, de installálható
01 - a driver nem installált és nem is installálható
FF - a driver installált

1 - file befűzése a nyomtatásra váró file-ok közé.
Belépéskor DS:DX címez egy "submit packet"-et. Ennek hossza 5 byte, felépítése a következő:

BYTE	a befűzési szint kódja (melyről többet semmiféle irodalomból kideríteni nem lehetett)
DWORD	a befűzni kívánt file teljes specifikációjának címe — nullával lezárt ASCII-string, amely tartalmazza a lemez, a directory-út (path) és a file nevét; a "*" és "?" karakterek (az ún. wildcard-ok) használata nem megengedett

2 - file törlése a nyomtatásra váró file-ok sorából.
DS:DX címzi a törlendő file nevét megadó, nullával lezárt ASCII-stringet. Itt megengedett a "*" és "?" karakterek használata.
Figyelem: míg az 1 alfunkció esetén a file-t egy további indirekcióval érjük el, a 2 funkciónál a DS:DX egyenesen a stringre címez!

3 - az összes file törlése a nyomtatásra váró file-ok listájából.

4 - az aszinkron nyomtató driver munkájának felfüggesztése, és a nyomtatandó file-okat leíró sor címének lekérdezése.
A rendszer megállítja az aszinkron nyomtatást, és a DS:SI regiszterben visszaadja a sor címét. Ez a sor 64 byte hosszú elemekből áll, amelyekben a nyomtatandó file teljes specifikációja olvasható nullával lezárva. Az utolsó bejegyzés 0-val kezdődik. Kiegészítésként DX-ben megkapjuk a printer működése közben bekövetkezett hibák számát is.
A nyomtatást bármely INT 2F hívás újraindítja.

5 - a 4 alfunkcióval felfüggesztett nyomtatás újraindítása minden egyéb hatás nélkül.

MS-DOS 3.1 és későbbi verziók

Multiplex interrupt - két, egyidejűleg a memóriában levő program kommunikációját támogató interrupt. Felhasználása az előbbiekben megadottakkal analóg módon, az egymással kommunikáló programok szándéka szerint lehetséges. AH-ban egy ún. multiplex csatornaszámot kell megadni, amely a kommunikálni kívánó programpárt azonosítja. Az egymást hívni akaró programok mindegyike minden más programmal egy ilyen kommunikációs csatornán tartja a kapcsolatot. Minden ilyen programpárhoz hozzá kell rendelni a 256 kommunikációs csatorna valamelyikét. A multiplex csatornaszámok közül a 00 és 7F közöttieket az MS-DOS használja, a többi áll az egymást hívni akaró felhasználói programok rendelkezésére. A hívó és a meghívott programnak kell gondoskodnia a hívás korrekt végrehajtásáról és fogadásáról.
AL egy funkciókódot tartalmaz, a további regiszterek a felhasználói programok igényei szerinti információkat tartalmazzák.
A következő funkciókódokat az MS-DOS előre definiáltnak veszi:

F.kód	Rövid leírás
0	Lekérdezi, hogy installált-e a csatorna
1	File befűzése egy várakozási sorba
2	File törlése egy várakozási sorból
4	Munka felüggesztése, státusz lekérdezése
5	Munka folytatása státuszlekérdezés után

Az AH=1-el kiválasztható csatorna a futó program és a PRINT aszinkron nyomtatóvezérlő közötti kommunikációra szolgál. Az egyes funkciók megegyeznek a 2F interrupt leírásában a 3.00 verzió alattiakkal (és az előbbi táblázatban megadott kódok kiterjesztésének tekinthetők).

Hibakódok (melyeket sikertelen futás után AX-ben kapunk vissza, illetve ott kell visszaadni):

H.kód	Rövid leírás
01	érvénytelen funkciókód
02	nemlétező file
03	nemlétező path, directory-út
04	túl sok megnyitott file van a rendszerben
05	meg nem engedett elérés
08	a sor betelt
09	a program foglalt, nem képes fogadni
0C	a megadott név túl hosszú
0F	érvénytelen lemezegység

Vö. a "Felhasználói hibakódok" fejezetben olvasható kódokkal; szinte teljes logikai analógiát figyelhetünk meg a 2F interrupt hibakódjai és a felhasználói hibakódok között.
Amennyiben saját programjaink közötti kommunikációra használjuk fel ezt az interruptot, lehetőség szerint tartsuk magunkat a fenti kódokhoz; egyéb hibákra más, ezektől eltérő értékeket használjunk.

Megjegyzés: a multiplex interrupt használata izgalmas és fontos téma. Mivel a rendelkezésemre álló irodalomból a fentieknél többet nem sikerült kideritenem, neki kellett esnem és kipróbálnom az interrupt meghívását. Ez sem segített, így vissza kellett fejtenem az interrupt-vektor által címzett kódot.
Sajnos, ebből sem tudtam meg nagyon sokat. Vázlatosan a következő történt: az interrupt mögötti

program belépett, végrehajtott egy csomó összehasonlítást, és ugrált fel s alá a memóriában. Az AH-ban hagyott egyes értékkel semmit nem törődött (persze, hiszen a PRINT program nem volt aktív). Ezután elindítottam a PRINT-et, és szép szabályosan installáltam. Ismét végrehajtottam egy megfelelően előkészített 2F interrupt-hivást, amely most máshová csapott be; elsőként összehasonlította az AH-t 1-gyel, és elugrott valahová. Itt sebtiben FF-el töltötte meg az AL-t (miszerint a nyomtató driver nem installált és nem is installálható - ??), majd visszatért.
Kipróbáltam további funkciókat is. Sikeresen kérdeztem le a nyomtatandó file-ok listáját, töröltem előbb egy, majd minden file-t a listából. A legfontosabb kérdésről azonban (hogy lehet a multiplex interrupt csatornái közé beiktatni egy újabbat?) semmi biztosat nem tudtam meg.
Erre nézve egyetlen ötletem van. Tegyük fel, hogy a hexadecimális 86 csatornát akarjuk használni. Ekkor kérdezzük le és mentsük az interrupt-vektor eredeti tartalmát, majd irányítsuk át a vektort a saját rutinunkra. Ez összehasonlítja az AH regisztert saját csatornánk számával (a 86-al). Ha megegyezik, akkor elugrik a programunk meghivandó részére (melyből természetesen IRET utasitással térünk vissza). Ha a csatorna száma nem a miénk, akkor pedig egy hosszú ugrással átadjuk a vezérlést a 2F interrupt-vektor eredeti tartalmára, azaz az addig ismert címre - csináljon az eredeti kezelő-rutin a kéréssel azt, amit akar vagy tud. Saját kezelő-rutinunkban pedig tartsuk be a fent leírt szabályokat: az előre definiált funkciókódokat arra használjuk, amire valók (vagy definiáljunk teljesen újakat), és tartsuk be a hibakódokra vonatkozó szabályokat is.
Azt hiszem, hogy a későbbi verziókban kevésbé anarchikusan fogják megoldani ezeket a kérdéseket. A fenti ötlet mindenesetre megy, ha nem is túl gyors és elegáns.

30-3F A DOS belső használatára fenntartva

III.7. Egyéb interruptok

40-7F Szabad interruptok

Megjegyzés: újabb MS-DOS verziók használják a 40-től 5F-ig terjedő interruptokat is.
A 60H-66H interruptok az ún. overlay interruptok. Ezek tartalmazzák a LINK által beépített automatikus overlay-kezelő rutinok címét. Akkor aktivizálódnak, amikor a program olyan rutint hív meg, amely éppen nincs a tárban.

80-F0 A BASIC interruptjai

Megjegyzés: a 86-F0 interruptok a futó BASIC számára foglaltak. Ha a BASIC nem aktiv, akkor bárki használhatja őket.

F1-FF Szabad interruptok

Az interrupt-vektorok vizsgálata során döbbenten tapasztalhatjuk, hogy a különféle szakkönyvekben "szabadnak" jelzett interrupt-vektorok közül számos teljesen értelmes és hasznosnak látszó kódra mutat, míg mások tartalma 0:0. Ez utóbbiak direkt meghívása végzetes lenne, hiszen a 0 ponton kezdődő szegmens 0. offsetjére adná át a vezérlést; itt pedig egy interrupt-vektor, tehát egy cím van! Ha pedig címet próbálunk meg programként végrehajtani, annak következményei beláthatatlanok. A tréfa kedvéért hadd iktassak ide egy "programot", amelyet a 0:0 cím visszafejtéséből kaptam:

```
XCHG    DL, [BX+DI-2AH]
ADD     AH, [BX+0CH]
JO      0008H
CMPSB
ADD     [BX+SI], SI
DB      67              ; ????
OR      AL, 70H
```

és hasonló értelemben tovább...

Eltérő MS-DOS verziókban természetesen teljesen más kódot kaphatunk, hiszen mások az egyes rutincímek.

IV. A program és az MS-DOS kapcsolatai

Az MS-DOS operációs rendszer a felhasználói programokkal software interruptok segítségével tartja a kapcsolatot. Minden rendszerfunkció egy megfelelően előkészített interrupt aktivizálásával hívható meg, amely lényegében egy szubrutinhívással egyenértékű. Felhívjuk a figyelmet arra, hogy az interrupt rutinok végén IRET utasítás van, amely három szót vesz ki a veremből. Ezért az interrupt-rutinok csak INT utasítással hívhatók meg, különben a verem végzetesen összegabalyodik.

Megjegyzés: az INT utasítás helyett használhatunk egy

```
PUSHF                   ;STATUS a stack-re
CLI                     ;Interruptok letiltása
CALL    FAR     ....    ;Rutinhívás
```

utasítássorozatot. Ettől azonban, amennyiben lehetséges, óvakodjunk.

A DOS a rendelkezésére álló hardware-eszközöket is interruptosan kezeli (pl. ha leütünk egy billentyűt, akkor a 09 interrupt éleződik). "Az interruptok funkciói" fejezetben olvashattuk az interruptok kiosztását; a fenntartott interruptokat a programozó természetesen nem használhatja.

A program a fordítás és szerkesztés folyamata után a DOS által biztosított környezetben fut. Az MS-DOS egy viszonylag kis operációs rendszer, amely "barátságos", de talán kicsit szegényes környezetet biztosít a vezérlése alatt futó programnak. Rokonszenvesen UNIX-szerű rendszer, filozófiája és szolgáltatásai nagyban emlékeztetnek a UNIX-éira.

A program indításkor a következőket kapja az operációs rendszertől (első közelítésben egy EXE típusú file-t feltételezve):

a) külön szegmensben a Program Segment Prefix-et, egy 100 (256) byte hosszúságú területet, amelyen számos információt talál (lásd "Program Segment Prefix" fejezet);

b) a gépen fizikailag rendelkezésre álló összes szabad memóriát (azaz, minthogy a DOS alulról felfelé építkezik, a DOS által szabadnak tudott első memóriabyte-tól a legmagasabb című, még létező RAM memóriabyte-ig, kiépítéstől függően maximum 640K-ig);

Megjegyzés: EXE file-ok esetén a LINK-nek meg lehet adni, hogy a program legalább és legfeljebb mennyi memóriát akar futás közben allokálni. Az átadott memóriaterület nagysága ettől is függ.

c) kaphat egy (legfeljebb 32Kb hosszú) ún. környezetet (environment), amelyben szöveges formában van leírva a programnak átpasszolni kívánt információ. Ez a szöveg nullával lezárt stringek nullával lezárt sorozata (lásd "Környezet (environment)" fejezet);

d) öt megnyitott file-t, amelyeket standard file-oknak nevezünk (lásd "Az MS-DOS standard file-jai" fejezet).

Az MS-DOS a program belépési pontjára adja a vezérlést. Ezután a program "szabadon garázdálkodhat" a memóriában. Az MS-DOS a továbbiakban csak a rendszerhívások esetén, külön felkérésre ad bármilyen támogatást a futó programnak.

Az EXE típusú file-ok több szegmensből állanak. Ezek közül fizikailag első a "Program Segment Prefix" (amelyet a továbbiakban PSP-nek nevezünk). Belépéskor erre mutat a DS és ES szegmensregiszter. A többi szegmens a programozó által előírt sorrendben helyezkedik el (kivétel a source-ban MEMORY néven definiált és minden osztálytól különböző osztályba sorolt szegmens, amely megállapodás szerint az utolsó helyre szerkesztődik).

Az EXE típusú file-ok elején a LINK egy ún. headert helyez el. Ez a header (lásd "Az EXE típusú file-ok headerje" fejezet) tartalmazza a program beolvasásához, futtatásához és a relokálás (azaz a program elindításakor a program futási helyétől függő adatok kitöltése) elvégzéséhez szükséges információkat. Ez a header persze nem része a programnak; a betöltés után a DOS többé nem használja.

Ha a programunk COM típusú (amely EXE típusúból az EXE2BIN programmal hozható létre, ha lehetséges), akkor a program egy szegmensből áll, minden szegmensregiszter egy pontra mutat. A program első 100 byte-ja kötelezően üres, mert ide helyezi el a DOS a PSP-t, a belépési pont pedig megállapodás szerint a hexadecimális 100. byte (CP/M-szerű programszerkezet). Ebben az egyetlen szegmensben el kell férnie a teljes program-, adat- és stack-területnek. Minden olyan program megvalósítható tehát, amely hagyományos 64Kb-os memóriában elfér. Vegyük észre azonban, hogy egy "igazi" 64Kb-os környezetben (pl. CP/M-ben) a rendszer memóriarezidens elemei — driverek, handlerek, file-kezelés — is benne vannak ebben a 64Kb-ban, a DOS ezen részei

viszont a memória egészen más pontján helyezkednek el, így egy COM típusú program valóban 64Kb hosszú lehet, leszámítva persze a 256 byte hosszúságú PSP-t.

A COM típusú program ugyancsak megkapja a teljes memóriát, bár alapértelmezésben csak hatvannégy kilobyte hosszúságú lehet. Természetesen éppen úgy átvehet környezetet, mint egy EXE típusú, és rendelkezésére állanak a standard file-ok is.

A standard file-ok (standard input és output, standard hiba, standard nyomtató és standard kiegészítő eszköz) kezelése rendkívül egyszerű, a DOS hívásainak segítségével oldható meg (lásd "Az MS-DOS standard file-jai" fejezet). Ezeket a file-okat nem kell sem megnyitni, sem lezárni. Természetük szerint szekvenciális file-ok, a standard input csak beolvasásra, a standard output, a standard nyomtató és a standard hiba csak kiírásra szolgál, a standard kiegészítő eszköz (az aszinkron kommunikációs port) kétirányú.

Az egyéb rendszerhívások segítségével tetszőleges file-okat kezelhetünk, viszonylag kényelmesen használhatjuk az összes beépített I/O eszközt, és számos vezérlési funkció áll rendelkezésünkre.

Könyvünkben a program és az MS-DOS kapcsolatát ismertető fejezetek felépítése a következő: egy rövidebb-hosszabb bevezető után az adott témához kapcsolódó "kemény" információt, azaz az odatartozó rendszerhívások ismertetését olvashatjuk - először az adott MS-DOS funkció sorszámát láthatjuk hexadecimális alakban. Ezzel egy sorban áll a funkció "hivatalos" angol neve (a Microsoft Corporation szóhasználata szerint, mely néha eltér az IBM kiadványokban szereplő nevektől!), majd pedig egy javasolt konstansnév, amelyet programjainkban az EQU operátor segítségével deklarálhatunk és használhatunk. A funkciók nem kódjuk szerint, hanem logikai sorrendben követik egymást. Az F függelék tartalmazza a funkciók felsorolását a részletes leírás oldalszámával együtt. A funkció ismertetésekor (ha ez lényeges) megemlítjük, hogy az MS-DOS mely verziójában használható. A második verzió egyes változataiban is elérhető funkciókat külön nem jelöljük meg. A 57-nél nagyobb kódú funkciók csak a 3.00 és későbbi verziókban használhatók. Ezeket az

MS-DOS 3.00 és későbbi verziók

vagy (ha csak a 3.1-ben szerepelnek)

MS-DOS 3.1 és későbbi verziók

sor vezeti be. Ez csak a közvetlenül alatta szereplő funkcióra vagy alfunkcióra vonatkozik.

A fejezetek utolsó részében egyszerű példaprogramok illusztrálják az odatartozó funkciók, lehetőségek felhasználását. (Az itt szereplő programokban a második részben megadott konstansneveket használjuk, általában külön deklaráció nélkül.) Ezek a kis programok önállóan futásra képesek, bár legtöbbjük, mint program, fabatkát sem ér, pusztán illusztrációnak tekintendők. A programok legnagyobb részét COM típusú programként írtuk meg, hogy ezáltal is kevesebbet kelljen kínlódni a szegmensekkel. Vannak esetek azonban, mikor a hangsúly épp azon van, hogy a feladatot EXE típusú file-ban kell (vagy lehet) megoldani. EXE programot csak ilyenkor írtunk.

> Megjegyzés: az MS-DOS operációs rendszer (mint a bevezetőben említettük) a CP/M-től indult. Innen ered a COM programformátum, amely minden ízében CP/M-szerű. A rendszer későbbi fejlődése azonban teljesen a UNIX irányába mutat. Minél fejlettebb verzióval dolgozunk, annál inkább támogatja a rendszer az EXE típusú programok megvalósítását, és egyre kevésbé szereti a COM alakot. Tehát egyes egyedül kényelmi (és terjedelmi) szempontok szólnak a COM típus mellett.

A példaprogramokban számos konstans-, makró-, struktúra- és rekorddefiníciót használunk. Az első néhány példaprogramot kivéve (melyek, az egészen kezdők kedvéért, teljes részletességgel szerepelnek) ezeket a deklarációkat meg sem adjuk. Ehelyett, megelőlegezve a harmadik kötetben irandókat, a könyv végén a "DOSCALL.INC" függelékben megadunk egy ún. include filet, mely az összes szükséges deklarációt tartalmazza. Ennek a file-nak a DOSCALL.INC nevet adtuk, s az

```
INCLUDE         DOSCALL.INC
```

utasítás segítségével lehet beépíteni a programunkba.

> Megjegyzés: az ilyen include file-ok használata nagyon hasznos gyakorlat, mert így egy helyen foghatjuk össze az alapvető fontosságú információkat. Igen nagy gondossággal kell azonban eljárnunk, mivel az összes korábban írt programot újra kell fordítani abban az esetben, ha valami lényegeset megváltoztatunk egy alapvető include file-ban, az ilyen újrafordítások

pedig gyakran okoznak nem várt nehézségeket. Férfiasan be kell vallanunk, hogy az include file-ok használata nem javítja a program olvashatóságát. Legkönnyebb egy olyan program olvasása lenne, amely makrók és szubrutinok, include-ok stb. nélkül egyszerűen és lineárisan adja meg az összes utasítást. Egy ilyen program hatékonyságáról azonban talán felesleges is beszélni - elképzelhetetlenül hosszú és rendkívül sok memóriát igényelne. Ezért is, és a program biztonsága, javíthatósága kedvéért is, amit csak lehet, szubrutinban vagy makró segítségével oldunk meg. A legfontosabb (és mindig szükséges) információkat természetesen include file-ból vesszük. Ez azonban azt az "egyszerű" feladatot rója a program olvasójára, netán karbantartójára, hogy fejből tanulja meg az összes deklarációt és szubrutinnevet és -szinopszist, különben folyton lapozgatnia kell gyakran két-háromszáz lapos programlistákat, és kisvártatva hónaljig, majd feje búbjáig tűnik el a mocsárban. De hát, utóvégre, ez a szép a programozásban...

Egy COM típusú példaprogram fordítása és lefuttatása a következő (természetesen a forrásnyelvű file létrehozása után):

```
C>MASM file;                    fordítás
C>LINK file;                    szerkesztés
Warning: no stack segment       panasza, hogy nincs
C>EXE2BIN file file.COM         konverzió EXE típusra
C>file.COM                      futtatás
```

A C> a rendszer promptja; a vastagon szedett szöveget a DOS vagy valamely rendszerprogram írja ki, a normál módon olvasható szöveget kell begépelni, az apróbetűk a szerző kommentárjai.

Az EXE típusú file-ok létrehozása természetesen analóg, egyszerűen elmarad az utolsó lépés, az EXE2BIN program meghívása; a LINK outputja azonnal futásra kész.

Természetesen a szükséges információkat tartalmazó include file-nak a MASM által elérhető helyen kell lennie (amennyiben a PE becenevű Personal Editort használjuk, akkor okszerűen abban az aldirectory-ban, amelyben a forrásnyelvű file-ok helyet kaptak, mivel editálás közben gyakran van szükségünk az alapfilera, és a PE csak egy directory-ban tud dolgozni).

A különböző funkciók minden egyes ravasz alkalmazási lehetőségét természetesen nem tudjuk illusztrálni. A példaprogramok csak a legfontosabb, a jellegzetes alkalmazást mutatják be. Ennél bővebbet a harmadik kötetben olvashatunk, ahol számos funkciót "életben" mutatunk be. Ez bizonyára többet is ér, mint az itteni, eléggé egyszerű kis példaprogramocskák. Viszont a harmadik kötet sem törekedhet teljességre. Sok funkció egyáltalán nem szerepel, csak azok, amelyek beilleszkednek a fő irányba.

IV.1. Az MS-DOS funkciók hívása

A DOS-funkciókat lényegében két nagy csoportra oszthatjuk: a hagyományos hívások és a kiterjesztett funkciók csoportjára. Figyeljük meg azt, hogy a hagyományos hívások legtöbbje lényegében megegyezik a CP/M-ben ismert függvényhívásokkal (persze az Intel 8080 és 8088 processzorok regiszterei közötti analógiát figyelembe véve). érdekes megfigyelni azt is, hogy a kiterjesztett funkciók egy része mintha UNIX-szerű lenne, hiszen rendkívüli módon támogatja a szabványos C könyvtári funkciók egyszerű, hatékony megvalósítását, és teljesen UNIX-szerűen kezeli az amúgy is onnan átvett directory-struktúrát és a fileokat, valamint az egyes eszközöket.

Azért vagyunk kénytelenek megkülönböztetni a hagyományos és a modern funkciókat, mert hívási stratégiájuk némileg eltér, a visszatérési információ pedig alapvetően különbözik.

A DOS-funkciók hívásához elő kell készíteni a szükséges paramétereket, majd a vezérlést valamilyen ügyes trükkel a DOS "function dispatcher" pontjára kell adni. Ez a pont a bejárat a DOS felhasználói számára. Itt a DOS ellenőrzi a paramétereket, majd végrehajtja a kért műveletet.

A megadandó értékek: egy funkciókód, amely megmondja azt, mit is kérünk a DOS-tól, valamint egy vagy több paraméter, amely arra utal, hogy mivel kell a műveletet elvégezni. A funkciókódot minden esetben AH-ba kell tölteni. Számos funkciónak több alfunkciója van. Az alfunkció kódját (ha van) az AL regiszterben kell megadni. A hagyományos funkciók legtöbbje vagy nem kap paramétert, vagy egyet vesz át. A paramétert (ha van) a DL vagy a DX regiszterben kell megadni, aszerint, hogy byte-os vagy szavas adatról van szó.

A hagyományos funkciók között is akad, amely több paramétert

vesz át, ezeket a megfelelő (a funkciónál megadott) regiszterekbe kell tölteni.

A modern funkciók általában több paramétert vesznek át; akad közöttük olyan, amely az összes általános regiszter mellett igénybe veszi a két indexregisztert is.

A függvények hívására három út van:

a) Az előkészítés után kiadunk egy

```
INT     21H
```

utasítást. Ez a gyakorlatban követett eljárás. Ha nagyon szépen akarjuk csinálni, akkor deklaráljunk egy ilyesféle makrót:

```
DOSCALL MACRO   CODE
                MOV     AH, CODE
                INT     21H
        ENDM
```

Ezt a makrót minden példaprogramban használni fogjuk. Az egyes függvények sorszámát az EQU segítségével konstansként (esetleg külön include file-ban) definiáljuk, majd pedig a makró hívásával aktivizálhatjuk a kívánt DOS funkciót.
Figyeljük meg, hogy nem mentjük az AX regisztert, hiszen az MS-DOS itt egy hibakódot adhat vissza, ezért a regiszter helyreállítása súlyos hiba volna!

b) A függvény sorszámát AH-ba töltjük, és egy hosszú hívást hajtunk végre a Program Segment Prefix területre, az 50 címre, ide a LINK előkeszített már egy

```
INT     21H
RETF
```

utasítássorozatot. Ezzel a lehetőséggel lényegében csak COM típusú program esetében élhetünk. Azonban nem biztos, hogy ez egy igazán jó eljárás, mert a távoli rutinhívás gépikódban hosszabb, mint az INT utasítás. Ezenkívül nem látszik igazán, mit is akarunk tenni. Az INT utasítás egyértelművé teszi a rendszerhívás szándékát, míg egy ilyen ködös rutinhívás elfedheti azt.

c) Már régebben írt programok használják ezt a kiegészítő lehetőséget. CL-be töltve a függvény sorszámát, rövid hívást hajtanak végre a hexadecimális 5 címre, ahol már elő van készítve egy hosszú ugrás a DOS-funkciók vezérlőrészére. Az AX regiszter tartalma elvész!
Ezt a lehetőséget a CP/M-el való kompatibilitás kedvéért őrizték meg. Ha élünk vele, hívásunk a következő módon jut el a DOS funkciók belépési pontjára:

(1) Programunk rövid hívást hajt végre a PSP 5. címére, ezzel a visszatérési cím offsetjét elmenti (egy szó).

(2) Az 5. címen végrehajtódik egy hosszú hívás. Ez két szót ment el, először a PSP szegmenscímét, majd a 0A értéket (a hosszú CALL utasítást követő byte címét).

(3) A meghívott ponton egy hosszú ugrás van, amely egy elég tisztességtelen szekvenciára vezeti a programot. Ez az utasítássorozat először kiveszi a veremből a 0A értéket (ezt el is dobja), majd felhozza a PSP szegmenscímét. A következő lépésben az általunk hagyott rövid visszatérési címet kiemeli egy változóba. Ezután elvermeli a STATUS-t, majd visszanyomja a verembe előbb a PSP-címet, majd az általunk hagyott offsetet, tehát egy rövid és egy hosszú hívásból összerakja a hosszú visszatérési címet, de még aládugja a STATUS-t is. Ezzel tökéletesen (?) szimulált egy interrupt utasítást. Eközben persze elrontotta az AX-et, és felhasznált egy belső memóriaváltozót.

(4) A CL-ben megadott funkciókódot áttölti az AH regiszterbe, és átadja a vezérlést oda, ahol a szokásos INT 21-el egyből beléptünk volna.

Figyeljük meg, hogy a vezérlés így nem a saját kódszegmensre, hanem a PSP-re adódik vissza, hiszen a rendszer annak szegmenscímét mentette el. Ezt a hívási stratégiát tehát csakis akkor használhatjuk, ha a PSP és a kódszegmens közös - magyarul csak COM típusú programokban. Azonban, mivel sokkal egyszerűbb eljárás is van a rendszer meghívására, ezt a módszert még ilyenkor se alkalmazzuk!

IV.2. Az MS-DOS funkciók által használt regiszterek

Nagyon fontos kérdés, hogy a meghívott DOS-funkciók mely regiszterek értékét őrzik meg a hívás után. Nos, természetesen visszakapjuk az összes szegmensregisztert, az SP értékét, és persze IP is a kellő helyre mutat. Egyebekben az az alapelv, hogy csak azon regiszterek tartalma vész el, ahol a DOS valamilyen visszatérési értéket szolgáltat. Tehát az AX regiszter nem marad sértetlen, de gyakorlatilag az összes többire számíthatunk. A flagek tartalma sem marad feltétlenül változatlan (mint a hibakódokról szóló fejezetben olvasható).

Jó tanácsként azt a gyakorlatot ajánlom, hogy ne számítsunk a szegmensregiszterek, az SP és az IP kivételével semmire, hanem mentsünk minden szükséges regisztertartalmat!

IV.3. Felhasználói hibakódok

A DOS-funkciók visszatéréskor az AL regiszterben és esetleg a Carry flagben adnak egy kódot, amely arról tájékoztat, milyen sikerrel hajtották végre a feladatukat.

A függvények "hagyományos" csoportja általában nem nyúl a flagekhez, az AL-ben nullát ad vissza sikeres futás esetén, és mínusz egyet, ha a futás sikertelen. Ez vonatkozik a standard file-okat kezelő, valamint a CP/M-szerű file-kezelő funkciókra (és néhány egyéb speciális funkcióra). Vannak azonban olyan funkciók (pl. 06, 0B), melyek az AL regiszteren kívül a Zero flagben is adnak válaszértéket!

A további funkciók az AX-ben adnak vissza valamilyen értéket vagy egy hibakódot. Esetükben a Carry flag vizsgálata mutatja meg, hogy a művelet sikeres volt-e vagy sem. Ha a Carry flag 0, akkor a művelet sikeres volt. Ez esetben az AL-ben (esetleg AX-ben) levő érték nem hibakód. Ha a művelet nem volt sikeres, akkor a Carry értéke 1, és AL a következő hibakódok egyikét tartalmazza (a hibakódok hexadecimális alakban szerepelnek). Az alábbiakban csak a "normál" hibakódokat adjuk meg, nem foglalkozunk a hálózatkezelési hibákkal.

01 - érvénytelen alfunkciókód

02 - a megadott file nem létezik

03 - a megadott directory-út (path) nem létezik

04 - túl sok file-t nyitottunk meg, nincs több szabad file-sorszám (file-handle)

05 - az elérés tiltott, írásvédett file-t akartunk írni

06 - érvénytelen file-sorszám - nem létező vagy meg nem nyitott file-sorszám (file-handle)

07 - a memórialeíró blokk megsérült - dinamikus memóriakezelés során hiba lépett fel

08 - nincs elég memória - dinamikus memóriakezelés során elfogyott a memória

09 - érvénytelen blokkcím - dinamikus memóriakezelés során érvénytelen címet adtunk meg

0A - nem szabályos környezet (environment) - programindítás során az átadandó környezet nem korrekt

0B - szabálytalan formátum - nem szabályos formátumú leíró blokkot adtunk meg

(Megjegyzés: ezt a kódot sem dokumentációban, sem programozás közben semmilyen funkcióval kapcsolatban nem láttam még.)

0C - szabálytalan elérési kód - file-nyitás során illegális file-elérési kódot adtunk meg

0D - érvénytelen adatok - perifériális eszköz direkt vezérlése során érvénytelen adatot adtunk meg

0E (nem használatos)

0F - érvénytelen lemezspecifikáció - nem létező lemezegységre hivatkoztunk

10 - a kurrens directory-t próbáltuk törölni

11 - nem ugyanaz az eszköz - file-név módosítása során eltérő eszköznevet adtunk meg

12 - nincs több file - nem talált több, a specifikációnak megfelelő file-t

IV.4. Kiterjesztett hibakódok

Míg a fenti hibakódokat az INT 21 utasítással meghívott rendszerfunkciók AL-ben adják vissza, az alább felsoroltakat csak a DOS 3.00 és későbbi verzióiban kaphatjuk meg kritikus hiba esetén. A kritikus hibát kezelő rutinnak meg kell hívnia az 59 funkciót, ezzel lekérdezni a kiterjesztett hibakódot. Az alábbi listát vessük össze a 24 interrupt-vektor leírásában megadott hibalistával (lásd "A DOS számára fenntartott interruptok" fejezetben). Tapasztalni fogjuk, hogy ezek a kódok pontosan megegyeznek az ottaniakkal, ha azok értékét hexa 13-al (19-el) megnöveljük: a kiterjesztett hibakódok a kritikus hiba esetén visszaadott hibakódoknak a felhasználói hibakódokra való leképezéséből származnak.

A továbbiakban megadott hibakódok csak az MS-DOS 3.0 és későbbi verzióiban élnek, és kritikus hiba esetén kaphatjuk meg őket.

13 - a lemez írásvédett - az írás a védelem miatt nem sikerült

14 - ismeretlen lemezegység

15 - a lemezegység nem üzemkész - például nincs rögzítve a lemez

16 - ismeretlen lemezkezelési parancs

17 - CRC-hiba - egy vagy több szektor CRC-je sérült (lásd "Az MS-DOS lemezek fizikai felépítése" fejezetben)

18 - a megadott I/O kérést leíró struktúra hossza érvénytelen

19 - keresési hiba - lemezkezelés közben nem sikerült a kívánt sávot megtalálni

1A - nem MS-DOS lemez - a felhasznált lemez nem MS-DOS formátumú

1B - a szektor nem található meg

1C - a papír kifogyott a nyomtatóból

1D - lemezírási hiba

1E - lemezolvasási hiba

1F - általános hiba

20 - illegális file-elérési kísérlet - az elindított program nem jogosult a file elérésére

21 - lezárt file - lezárt (lockolt) file-részt próbáltunk elérni

22 - rossz lemez

23 - nem létező File Control Block

A 24-4F közötti hibakódok csak hálózatkezelő programokkal kapcsolatban élnek MS-DOS 3.1 és későbbi verziókban, ezért ismertetésüktől ehelyütt eltekintünk.

IV.5. Hibakódok a 3.00 verzióban

Az alábbi kódokat olyan függvények adhatják vissza, amelyek csak az MS-DOS 3.00 és későbbi verzióiban használhatók. Ezért:

MS-DOS 3.00 és későbbi verziók

50 - a file már létezik

51 - fenntartott

52 - a "make" művelet nem sikeres

53 - 24-es interrupt hiba

Az 54-58 közötti hibakódok csak hálózatkezelő programokkal kapcsolatban élnek MS-DOS 3.1 és későbbi verziókban, ezért ismertetésüktől most eltekintünk.

V. Az MS-DOS standard file-jai

V.1. A standard file-ok fogalma

A DOS virtuális perifériákat (standard file-okat) bocsát a program rendelkezésére. Ezeket nem kell megnyitni. Segítségükkel a program elég kényelmesen végezheti el a legfontosabb input-output műveleteket. A standard file-ok filozófiája teljesen a UNIX operációs rendszerből származik. Ennek az a lényege, hogy egy interaktív program számára létfontosságú klaviatúra és képernyő felfoghatók úgy, mint szigorúan szekvenciális, csak olvasható vagy csak írható file-ok. S ha így fogjuk fel őket, semmi akadálya sincs annak, hogy a programot elindító felhasználó kívánsága szerint az output ne a képernyőre, hanem valami hasonlóan kezelhető file-ra (pl. a nyomtatóra, az aszinkron vonalra [tehát perifériális eszközökre], vagy pedig egy megnevezett lemezfile-ba) kerüljön. Hasonlóképpen átirányíthatjuk az inputot is; a program veheti a parancsokat egy megadott lemezfile-ból, az aszinkron vonalról vagy bármely egyéb eszközről, amely képes inputot produkálni.

Ez a filozófia rendkívül rugalmassá teszi a UNIX (és az azt lemásoló MS-DOS) operációs rendszert, hiszen az adott program teljesen különböző funkciók megvalósítására képes aszerint, hogy a standard input és output file-jai éppen mit jelentenek.

Példaképpen képzeljük el, hogy a programunk sorra visszaír a standard output file-ra minden karaktert, amit a standard input file-ból olvas. Ez első közelítésben bizony kicsit butácska program, semmi eredményt nem várhatunk tőle. Legyen a neve egyszerűen SCOPY.EXE. Ha elindítjuk, akkor még az is gondot okoz, hogyan tudunk megszabadulni tőle. Tegyük fel, hogy egy megadott karakterre, pl. a CTRL-Z-re kilép:

```
C>SCOPY
abc
1
234
^Z
C>
```

Jó esetben valami ilyesmit láthatunk. Ha azonban kicsit ravaszabbul hívjuk meg a programot, például így:

```
C>SCOPY >a.txt
```

akkor az SCOPY standard output file-ját (csak e futás idejére) átirányitottuk úgy, hogy a program a képernyő helyett az A.TXT file-ba fog irni. E file tartalma, ha ugyanezt gépeljük be:

```
abc
1
234
^Z
```

karaktersorozat lesz. Ha ezután ugyanezt a programot az

```
C>SCOPY <a.txt
```

hivással inditjuk, a program a standard input helyett fogja használni az "a.txt" file-t, végigolvassa tartalmát, és kiirja a standard outputra (ami most a képernyő). Végül az

```
C>SCOPY <a.txt >b.txt
```

inditás hatására a standard input az "a.txt", mig a standard output a "b.txt" file lesz, a program pedig átmásolja az első file-t a másodikba.

Az átirányitás egy jellegzetes alkalmazása lehet még az is, hogy valamelyik standard file-t nem egy lemezfile-ba, hanem egy perifériára irányitjuk át:

```
C>SCOPY >prn
```

Ekkor az SCOPY program a képernyő helyett a nyomtatóra küldi az összes kiirást (pontosabban az MS-DOS "eltéríti" a nyomtatóra az SCOPY-nak a standard outputra irányított kiirásait).

A standard file-okat egyébként (csakúgy, mint a lemezfile-okat és az egyéb perifériákat) az ún. file-sorszám segitségével azonositjuk. Ennek eredeti neve "file-handle", néha "token". Egy ilyen "file-handle" (melyet más operációs rendszerből vett kifejezéssel logikai csatornának is nevezhetnénk) egy sorszám, amely arra utal, hogy az MS-DOS hányadik leiró struktúrájával kezeli az adott file-t. Egyszerre természetesen több ilyen csatornát is kezelhet a programunk. Ezek számát a CONFIG.SYS file-ban adhatjuk meg a FILES=nn paranccsal. Az alapértelmezés sze-

rinti érték nyolc, tehát egyéb utasitás hiján egyidejűleg három csatornát tarthatunk megnyitva (az öt standard file mellett).

A standard file-ok, azaz a virtuális perifériák sorszáma a következő (zárójelben a standard file alapértelmezése):

0000 standard input file (klaviatúra), átirányítható

0001 standard output file (képernyő), átirányítható

0002 standard hibajelző file (képernyő), nem irányítható át

0003 standard kiegészítő eszköz (aszinkron kommunikációs adapter), nem irányítható át

0004 standard nyomtató, nem irányítható át

Megjegyzés: az átirányíthatóság pusztán azt jelenti, hogy az adott standard file-t MS-DOS szinten, a futás idejére átirányíthatjuk. Maga a program az erre szolgáló funkciók segítségével bármelyik standard (vagy egyéb) file-ját tetszőlegesen átirányíthatja (lásd "File-sorszámot módosító funkciók" fejezet).

Mielőtt részletesen tárgyalnánk a standard file-ok kezelésére szolgáló funkciókat, fontos megjegyeznünk, hogy az MS-DOS minden ilyen funkció végrehajtásakor CTRL-BREAK ellenőrzést hajt végre. Amennyiben a funkció aktivizálódása előtt a felhasználó leütött egy CTRL-BREAK-et, akkor a DOS végrehajt egy INT 23 utasítást, amely alapértelmezésben abortálja a programot (roppant hasznos segítség a programok tesztelése során!). Ezalól csak néhány standard file-kezelő funkció kivétel (a direkt konzolkezelő funkciók, 06 és 07); ott ezt külön kiírjuk.

Az MS-DOS-nak van egy BREAK kapcsolója. Ha ezt a megfelelő funkciókkal vagy a beépitett BREAK parancs segítségével bekapcsoljuk, akkor a rendszer nem csak a szokványos standard file-kezelő funkciók, hanem bármely rendszerfunkció végrehajtása során elvégzi a CTRL-BREAK ellenőrzést. A direkt konzolkezelő funkciók (06, 07) végrehajtása közben azonban a rendszer ekkor sem végez CTRL-BREAK ellenőrzést!

V.2. A standard file-okat kezelő funkciók

A standard file-ok kezelésére a következő DOS-funkciók szolgálnak:

01 Read Keyboard and Echo - DOS_STRDCHE
Beolvasás a standard input file-ról - vár egy karaktert, beolvassa, visszaírja a standard output file-ra (echo), és kódját AL-ben adja vissza.
Figyeli a CTRL-PRTSCR parancsot. Ha ilyen érkezik, akkor végrehajtja a képernyőtartalom másolását a nyomtatóra (hard copy).

Megjegyzés: a visszaadott érték a beolvasott karakter ASCII-kódja. Amennyiben olyan billentyűt ütünk le, amelyhez nem tartozik ASCII-kód (bármelyik funkcióbillentyű vagy a cursorvezérlő billentyűk), akkor a rendszer 0 értéket ad vissza AL-ben, majd a következő olvasásra megkapjuk a leütött billentyű "scan-kódját" (lásd a 3. kötet "A klaviatúra" fejezetét).

02 Display Character - DOS_STWRCH
Kiírás a standard output file-ra - DL-ben kell megadni a kiírandó karakter kódját.

03 Auxiliarry Input - DOS_AUXIN
Beolvasás a standard kiegészítő file-ról - vár egy karaktert, beolvassa, és kódját AL-ben adja vissza.

04 Auxiliarry Output - DOS_AUXOUT
Kiírás a standard kiegészítő file-ra - a kiírandó karakter kódját a DL-ben kell megadni.

05 Print Character - DOS_PRWRCH
Kiírás a standard nyomtatóra - a kiírandó karakter kódját a DL-ben kell megadni.

06 Direct Console I/O - DOS_STDIRIO
"Direkt konzol Input/Output" - kiírást vagy beolvasást egyaránt végezhet a standard outputra vagy a standard inputról.

Ha meghívásakor a DL tartalma FF (-1), akkor megvizsgálja a standard input eszköz bufferét. Ha van benne karakter, akkor a Z flag-et 0-ra állítja, és a beolvasott karakter kódját AL-ben adja vissza; ha nincs, akkor Z-ben 1-gyel válaszol. Ekkor AL értelmetlen.
Ha DL tartalma nem FF, akkor a DL-ben levő oktettet (bit-nyolcas, a legfelső bit levágása nélkül) kiírja a standard output eszközre.
Ez a funkció nem ellenőrzi a CTRL-BREAK és a CTRL-PRTSC kódokat!

Megjegyzés: a CP/M-ből megismert gyakorlat szerint három fontos különbség van a 01 vagy 02, valamint a velük analógnak tekinthető 06 funkció között.
Ezek közül az egyik az MS-DOS-ban merőben elméleti jellegű. Az e funkciók alapjául szolgáló CP/M operációs rendszerben a 01 és 02 funkció szabvány ASCII-kódokat olvas, 7 bitet, a legmagasabb helyiértékű bitet pedig paritásellenőrzésre használja. Tehát a CP/M-ben a kiírás (02 funkció) levágja a legmagasabb bitet. Ha nyolcbites kódot akarunk kiküldeni, akkor kénytelenek vagyunk a 06 funkciót használni (melyről persze látszik, hogy nem minden kód kiküldésére képes). Az MS-DOS-ban a bit levágása nem történik meg, ugyanúgy kiírhatunk bármilyen félgrafikus karaktert, mint a szabvány ASCII-kódkészlet karaktereit.
A másik fontos különbség az, hogy a 01 beolvasó funkció "szinkron", míg a 06 "aszinkron" - azaz a 01 funkció megvárja a karakter megérkezését, és csak sikeres olvasás után adja vissza a vezérlést a hívónak, míg a 06 funkció egy logikai értéket szolgáltat, és a vezérlés igen rövid időn belül visszakerül a hívóhoz.
A harmadik, amelyet már említettünk, hogy a 06 funkció nem végez CTRL-BREAK ellenőrzést.

07 Direct Console Input - DOS_STDIRIN

Direkt konzol input echo nélkül - vár egy karaktert a standard input file-ról, és a kapott kódot AL-ben adja vissza. Nincs CTRL-BREAK ellenőrzés.

08 Read Keyboard - DOS_STRDCH
Karakterbeolvasás echo nélkül - analóg 01-gyel, de echo nincs.

09 Display string - DOS_STWRSTR
Buffer (string) kiírása a standard output file-ra - a kiírandó stringet a "$" jel zárja le. A string címét a hívás előtt a DS:DX regiszterbe kell elhelyezni.

0A Buffered Keyboard Input - DOS_STRDSTR
Bufferelt klaviatúra input - híváskor a DS:DX egy olyan bufferre mutasson, melynek első két byte-ja vezérlőinformáció, a többi szolgál bufferként. Az első byte tartalmazza a valódi buffer hosszát byte-okban, melybe a beolvasott stringet lezáró ENTER-nek is bele kell férnie. A második byte-ban adja vissza a DOS a valóban olvasott karakterek számát.
Tehát a beolvasandó karaktersorozat lehetséges legnagyobb hosszát a felhasználó szabja meg. A DOS addig olvas, míg a buffer (egy karakter híján) be nem telik vagy egy ENTER kódot (0D) nem kap. Ha a buffer betelik, akkor a DOS hibajelzésként sípol egyet minden karakterre - csak az ENTER-t és az utolsó karakter törlésére szolgáló BACKSPACE-t fogadja el, amely lezárja a stringet. Tehát 17 karakter beolvasásához egy 18 byte-nyi bufferre van szükségünk, hogy beleférjen még a záró ENTER kódja (0D) is.
Igen kellemes tulajdonsága még a 0A funkciónak, hogy amíg le nem zárjuk a sort, addig a BACKSPACE billentyűvel visszamehetünk, törölhetjük és javíthatjuk az addig gépelt szöveget.

0B Check Keyboard Status - DOS_STINCHK
A standard input eszköz állapotának lekérdezése - megvizsgálja, van-e beolvasható karakter a standard input eszköz bufferében. Ha van, AL-ben FF-et ad vissza, ha nincs, AL tartalma 0.

> Megjegyzés: amennyiben a standard inputot egy lemezfile-ba irányítottuk át, akkor ez a funkció a file végének elérésekor ad vissza 0-t AL-ben, egyébként FF-et.

0C Flush Buffer, Read Keyboard - DOS_STINFLS
A klaviatúra bufferének törlése (ha a standard input file-t nem irányították át), és egy klaviatúra-funkció meghívása - törli a már beütött karaktereket, és meghívja a 01, 06, 07, 08 és 0A kódú DOS funkciók valamelyikét mint alfunkciót (a kódot AL-ben kell megadni). A rendszer (ha fizikai beolvasásra van szükség) várni fog a legközelebbi karakterre.

Megjegyzés: ez adja az egyetlen lehetőséget a klaviatúra bufferének törlésére - használjuk a 06 alfunkciót. Sajnos — elég érthetetlen módon — a ROM BIOS klaviatúra-driverben nincs lehetőség a buffer törlésére.

A fenti funkciók szimbolikus kódja azért tér el hivatalos angol nevüktől, mert sohasem a klaviatúrát vagy a képernyőt, hanem a standard input és output file-t kezelik.

Lássunk most néhány egyszerű példát! Ezek során a felhasznált funkcióhívásokat "A DOS funkciók hívása" fejezetben megadott DOSCALL nevű makróba ágyazzuk. A programok így a kezdők számára kicsit nehezebben, a rutinos programozó számára azonban könnyebben olvashatók. Mindenesetre a kezdőknek is mihamarabb meg kell barátkozniuk a makrókkal, és hozzá kell szokniuk ahhoz, hogy még igen egyszerű programban is deklaráljanak és használjanak makrókat, mert ez nagymértékben javítja a programozás biztonságát. E helyen még az szól a makrók használata mellett, hogy nélkülük a bemutatandó programok szövege rétestésztaként nyúlna, túl minden épeszű és elfogadható terjedelmen. Egyébként ezek az első példaprogramok teljes szövegek, ha így begépeljük őket, akkor futni fognak. A későbbiek során a rövidítés kedvéért include file-okat és az addig megadott példákra való hivatkozásokat is fogunk találni.

V.3. Példák a standard file-kezelő funkciókra

V.3.1. Beolvasás és kiírás karakterenként

Az első program addig olvassa a standard input file-t (méghozzá echo-val), amíg egyszer be nem olvas egy ESCAPE karaktert. Minden egyes beolvasott karaktert kiír a maga emberségéből is, így minden karaktert kétszer fogunk látni.

Megjegyzés: azért használjuk az ESCAPE-t a természetesebb CTRL-Z helyett, mert annak kiírása az átirányított standard outputra a file azonnali lezárását jelenti; ezt nem akarjuk spontán módon elvégezni.

Hajtsuk végre ezt a programot először átirányítás nélkül, majd irányítsuk át a standard output file-t pl. az X.X nevű lemezfile-ra. Következő lépésként a standard input file-t irányítsuk át az előbb létrehozott X.X-be, végül pedig mind a standard input, mind a standard output file-t irányítsuk át egy-egy lemezfile-ba.

A végső cél a teljes transzparencia elérése: az outputként létrehozott file egyezzen meg bitről bitre az inputként használt file-al. Ez a cél ezekkel az eszközökkel teljes általánosságban azért nem valósítható meg, mert a program nem képes felismerni, hogy az inputként megadott file véget ért. A maga részéről ő lelkesen várja a további karaktereket, amelyek azonban soha nem érkezhetnek meg. Első pillantásra úgy tűnik, hogy a file végének felismerésében segithet bennünket a 0B funkció, amely ellenőrzi a standard input státuszát. Ekkor azonban a programunk nem képes egyformán kezelni standard inputként a klaviatúrát és egy lemezfile-t, hiszen a klaviatúra esetén a 0 azt jelenti, hogy nincs beolvasható karakter (de még lesz), míg egy lemezfile esetén a file végének elérését.

Ezt a nehézséget úgy tudjuk lekuzdeni, hogy a "file vége" jelként értelmezett ESCAPE karaktert kiírjuk a file-ra. Ekkor viszont az inputként használt file a belsejében nem tartalmazhat ESCAPE karaktereket, hiszen a beolvasás az első ilyen karakternél véget ér. Ettől eltekintve azonban némi erőfeszitéssel tudunk irni egy csaknem teljesen transzparens programot.

```
TITLE   STDIN másolása STDOUT-ra

;
;Konstansok
;

DOS_STRDCHE     EQU     01H     ;Karakterolvasás STDIN-ről
DOS_STWRCH      EQU     02H     ;Karakterkiírás STDOUT-ra
CHR_CR          EQU     13      ;Kitüntetett karakterek: CR,
CHR_LF          EQU     10      ;LF - kocsivissza/soremelés
CHR_ESC         EQU     27      ;Kilépési parancs, ESCAPE
```

```
;
; Makrók
;
DOSCALL MACRO   CODE
                MOV     AH, CODE
                INT     21H
        ENDM
;
; Kód- és adatszegmens
;
CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:CODE, ES:CODE, SS:CODE
        ORG     100H

ENTRY:
;               JMP     START   ;Felesleges, nincs változó
;START:                          ;-------;
                                        ;
RD_CHR_LOOP:                            ;
                DOSCALL DOS_STRDCHE     ;karakter be + echo
                                        ;
                CMP     AL, CHR_LF      ;LF-et elnyeljük
                JE      RD_CHR_LOOP     ;
                                        ;
                MOV     DL, AL          ;
                DOSCALL DOS_STWRCH      ;Karakter kiírása
                                        ;
                CMP     DL, CHR_ESC     ;Kilépési parancs?
                JE      QUIT            ;Igen, kiugrunk
                                        ;
                CMP     DL, CHR_CR      ;CR volt?
                JNE     RD_CHR_LOOP     ;Ha nem, folytatjuk
                                        ;
                MOV     DL, CHR_LF      ;Igen, utánaküldünk
                DOSCALL DOS_STWRCH      ; egy LF karaktert
                JMP     RD_CHR_LOOP     ;Folytatjuk
QUIT:                                   ;
                INT     20H             ;Kilépés a DOS-hoz
                                        ;
CODE    ENDS                            ;
                                        ;
        END     ENTRY                   ;
```

Az ENTRY cimke utáni két "szabvány sort" azért láttuk el pontosvesszővel, mert nem lévén egyetlen adatbyte sem, nincs szükség a vezérlés ilyen átadására.

Kihasználtuk azt a tényt, hogy ezek a funkciók nem rontják el a DL regiszter értékét, tehát minden beolvasott karaktert előbb visszaírunk a standard output file-ra, és azután nézzük csak meg, mi is volt. Ez alól persze az LF (soremelés) kivétel, de vajon miért? Gondoljuk meg, hogy amikor mi RETURN vagy ENTER billentyűt ütünk, akkor a program egy CR (kocsivissza) kódot olvas. A visszaírásnál tehát kénytelen "utánaküldeni" egy LF-et, hogy a képernyőn a sorok ne kerüljenek egymásra. Ha most a standard output file-t átirányítjuk egy lemezfile-ba, akkor oda is kiíródik az LF karakter. Amikor pedig egy ilyen lemezfile-t használunk standard inputként, akkor minden CR után kiírunk egy LF-et, ezután pedig jön a file-ból olvasott LF. Ezt tehát "el kell nyelnünk", különben két LF karakter jelenik meg a mostani standard outputon, és oda a transzparencia. Persze az így is, úgy is odavan, mert ha egy file-ban véletlenül egy magányos LF van valahol (amit nem előz meg egy CR karakter), a programunk azt is éhesen elnyeli. Ha viszont az input file-ban több egymást követő CR karakter van, mindegyiket felbővítjük egy LF-el, és már megint baj van. E probléma teljes megoldását, amely láthatóan nem is olyan nagyon egyszerű, a lelkes Olvasóra bízzuk, azzal a jótanáccsal, hogy az itt használt beolvasó funkció helyett természetesen valamelyik echo nélküli olvasás ígér csak teljes sikert. Ez a program ugyanis mindent kétszer mond, kétszer mond...

V.3.2. Stringek beolvasása és kiírása

Ez a program beolvas egy nem nulla karakter hosszúságú stringet a standard inputról, majd visszaírja azt a standard outputra. Mielőtt azonban beolvasáshoz kezdene, kiír egy üzenetet, amely felszólítja a felhasználót néhány karakter begépelésére. Ha a felhasználó erre egy üres ENTER billentyűt üt, azaz egy nulla hosszúságú stringet ad be, akkor jól leszidja, és kéri újra a string begépelését. Végül pedig visszaírja a kapott stringet, és utánaküld még egy kocsivissza-soremelés karaktersorozatot is.

E program során találkozunk először azzal az érdekes jelenséggel, hogy egy ilyen egyszerű funkció megvalósítása néha kevésbé fáradságos, mint szépen megjelenő, precizen megadott üzenetek definiálása. Külön kell definiálnunk minden sort, mindegyiket el kell látnunk a megfelelő kocsivezérléssel stb. Ráadásul (hacsak nem akarunk csalni) kénytelenek vagyunk külön deklarálni még egy egyszerű kocsivissza-soremelés sorozatot is.

Megjegyzés: megtehetjük persze, hogy az egyik üzenet végét külön elnevezzük MSG_CRLF-nek, de az ilyesmi már kimeríti a "tisztességtelen haszonszerzés" fogalmát, mivel egyetlen dolgot kétféleképpen használunk fel. Felületes vizsgálat után a program későbbi módosításakor, valamit közéjük iktatva, esetleg elszakítjuk a kocsivissza-soremelés előtt levő szöveget a vezérlőkarakterektől. Ez pedig igen nehezen észrevehető hibákat fog okozni a programban. A tapasztalatok azt mutatják, hogy az ilyen pongyolaságokat a megrendelő fedezi fel a kész progam bemutatásának második-harmadik percében. Ez pedig nagyban csökkenti a programozó hitelét és a sikeres átadás esélyeit.

```
TITLE   String beolvasása és kiírása
;
; Konstansok
;
DOS_STWRSTR     EQU     09H             ;String kiírása
DOS_STRDSTR     EQU     0AH             ;String beolvasása
CHR_CR          EQU     13              ;Kocsivissza
CHR_LF          EQU     10              ;Soremelés
CHR_DOLLAR      EQU     '$'             ;Dollár karakter
;
; Makrók
;
DOSCALL MACRO   CODE
                MOV     AH, CODE
                INT     21H
        ENDM

PRTSTR  MACRO   STRING
                MOV     DX, OFFSET STRING
                DOSCALL DOS_STWRSTR
        ENDM
```

```
RDSTR   MACRO   BUFFER
                MOV     DX, OFFSET BUFFER
                DOSCALL DOS_STRDSTR
        ENDM
;
; Kód- és adatterületek
;
CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:CODE, ES:CODE, SS:CODE
        ORG     100H

ENTRY:
                JMP     START           ;Ugrás a belépésre
KB_BUF          DB      16, 0           ;Buffer a string be-
                DB      16  DUP ( ? )   ; olvasásához

;
; Kiirandó üzenetek (kocsivissza-soremeléssel)
; Mindegyik "$" jellel lezárt
;

QST_WRT         DB      'Adj be pár karaktert!'
                DB      CHR_CR, CHR_LF, CHR_DOLLAR
MSG_OK          DB      'A következőt gépelted: '
                DB      CHR_DOLLAR
MSG_ERR         DB      'Nem üres stringet kértem!
                DB      CHR_CR, CHR_LF, CHR_DOLLAR
MSG_CRLF        DB      CHR_CR, CHR_LF, CHR_DOLLAR

START:                                  ;
RD_STRING_LOOP:                         ;
                PRTSTR  QST_WRT         ;Bevezető üzenet ki
                                        ;
                RDSTR   KB_BUF          ;String beolvasása
                PRTSTR  MSG_CRLF        ;CR-LF ki
                                        ;
                MOV     BX, OFFSET KB_BUF
                CMP     BYTE PTR 1[ BX ], 0
                JNE     RD_STRING_OK    ;Hány karakter jött?
                                        ;
                PRTSTR  MSG_ERR         ;Nulla - hibaüzenet
                JMP     RD_STRING_LOOP  ;Olvasás még egyszer
```

```
RD_STRING_OK:                                   ;
                PRTSTR  MSG_OK                  ;Nyugtázás eleje ki
                MOV     CL, BYTE PTR  1[ BX ]
                XOR     CH, CH                  ;Az üzenet hossza CX.
                MOV     DI, CX                  ;majd DI regiszterbe
                MOV     BYTE PTR  2[ DI ][ BX ], CHR_DOLLAR
                                                ;Az üzenet lezárása
                MOV     DX, BX                  ;
                ADD     DX, 2                   ;Visszaírandó címe
                DOSCALL DOS_STWRSTR             ;A beolvasott string
                                                ;kiírása
                PRTSTR  MSG_CRLF                ;CR-LF még egyszer
                INT     20H                     ;Kilépés MS-DOS-hoz
CODE    ENDS
        END     ENTRY                           ;
```

V.3.3. A direkt input/output funkciók használata

A harmadik, a standard file-ok kezelését illusztráló program "ravaszabb" funkciókat használ, az ún. direkt konzol funkciókat. Az "ún." használata tudatos, mert ezek a funkciók sem a konzol kezelését végzik. Ugyanúgy a standard input file-t olvassák és a standard output file-t írják, mint az eddig bemutatott funkciók. Tehát ugyanúgy képesek az átirányított file-ok kezelésére, mint az összes többi itt megismert funkció.

```
TITLE   A direkt konzolkezelő funkciók használata

;
; Konstansok
;

DOS_STDIRIO     EQU      06H                    ;Direkt input/output
DOS_STDIRIN     EQU      07H                    ;Beolv. echo nélkül

DOS_STDIRIO_IN  EQU     0FFH                    ;Direkt beolvasás

CHR_CR          EQU     13                      ;Kocsivissza
CHR_LF          EQU     10                      ;Soremelés
CHR_ESC         EQU     27                      ;Kilépési parancs (ESC)
```

```
;
; Makrók
;
DOSCALL MACRO   CODE
                MOV     AH, CODE
                INT     21H
        ENDM

;
; Kódszegmens
;

CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:CODE, ES:CODE, SS:CODE
        ORG     100H
                                        ;
ENTRY:
;               JMP     START
;START:
LOOP:                                   ;
                MOV     DL, DOS_STDIRIO_IN
                DOSCALL DOS_STDIRIO     ;Direkt beolvasás
                JZ      LOOP            ;Várjuk a karaktert
                                        ;
                MOV     DL, AL          ;A karakter kiírása
                DOSCALL DOS_STDIRIO     ;
                                        ;
                CMP     DL, CHR_ESC     ;
                JNE     LOOP            ;
QUIT:                                   ;
                INT     20H             ;
                                        ;
CODE    ENDS                            ;
        END     ENTRY
```

Ez a program lényegesen egyszerűbb, mint a 01 és 02 funkciót illusztráló első példaprogram. Nem kezeltük le a kocsivissza karakter esetét; ezt az első példa alapján az Olvasó könnyen megteheti.

A program fejlesztésében továbbhaladni úgy lehetne talán, hogy a beolvasáshoz a 06 funkció helyett a 07-et használjuk. Ekkor persze felesleges a várakozási ciklus, mert a 07 funkció

"szinkronban" olvas. Viszont a program nem írja ki kétszer az összes bevitt karaktert, mert mind a 06, mind a 07 funkció echo nélkül olvas. Az első program teljes transzparenciát megközelítő változatában talán a 07 funkció lehet segítségünkre, mert nem "nyeli le" a CTRL-BREAK karaktereket.

V.3.4. A standard input bufferének törlése

Az alábbi program fontossága mindenki előtt világos. Akkor szükséges ez a trükk, ha valamilyen okból tiszta lappal kell indulnunk a standard input file olvasásában, és nem engedhető meg az, hogy a felhasználó által unaloműzésként bepöttyentett karakterek megjelenjenek a standard input file-on. E szubrutin fontosságát még csak növeli az a tény, hogy a ROM BIOS nem ad lehetőséget a klaviatúra-buffer törlésére.

```
TITLE   A standard input bufferének törlése

;
; Konstansok
;

DOS_STRDCHE     EQU      01H            ;Karakter be + echo
DOS_STWRSTR     EQU      09H            ;String kiírása

DOS_STINFLS     EQU      0CH            ;Fő funkciókód
DOS_STDIRIO     EQU      06H            ;Szükséges alfunkció
DOS_STDIRIO_IN  EQU     0FFH            ;Direkt I/O - input

CHR_ESC         EQU     27              ;ESCAPE kódja
CHR_CR          EQU     13              ;Kocsivissza
CHR_LF          EQU     10              ;Soremelés

;
; Makrók
;
DOSCALL MACRO   CODE
                MOV     AH, CODE
                INT     21H
        ENDM                            ;
```

```
PRTSTR  MACRO   STRING
                MOV     DX, OFFSET STRING
                DOSCALL DOS_STWRSTR
        ENDM
;
; Kódszegmens
;
CODE    SEGMENT PARA    PUBLIC  'CODE'

        ASSUME  CS:CODE, DS:CODE, ES:CODE, SS:CODE
        ORG     100H

ENTRY:                                  ;
                JMP     START           ;Ugrás a belépésre

TEXT1:          DB      'Nyomj le néhány billentyűt!'
                DB      CHR_CR, CHR_LF, '$'
TEXT2:          DB      'Ismét nyomj le néhány billentyűt!'
                DB      CHR_CR, CHR_LF, '$'

START:                                  ;
                PRTSTR  TEXT1           ;1. utasítás kiírása
                CALL    WAIT            ;
                CALL    CLRSTINBUF      ;Töröljük a buffert
                PRTSTR  TEXT2           ;2. utasítás kiírása
READ_LOOP:                              ;
                DOSCALL DOS_STRDCHE     ;Olvasás echóval
                CMP     AL, CHR_ESC     ;ESCAPE volt?
                JNE     READ_LOOP       ;
                                        ;
                INT     20H             ;Kilépés
                                        ;
;---------------------------------------;
;
; CLRSTINBUF    - a standard input bufferének törlése
;       Input:
;               semmi
;       Output:
;               semmi
;       Minden regiszter mentve
;
;---------------------------------------;
```

```
CLRSTINBUF      PROC    NEAR                    ;
                                                ;
                PUSH    AX                      ;
                PUSH    DX                      ;
                PUSHF                           ;
                MOV     AL, DOS_STDIRIO ;Alfunkció kódja
                MOV     DL, DOS_STDIRIO_IN
                DOSCALL DOS_STINFLS             ;
                POPF                            ;
                POP     DX                      ;
                POP     AX                      ;
                RET                             ;
CLRSTINBUF      ENDP                            ;
                                                ;
;-----------------------------------------;
; WAIT          - várakozás néhány másodpercig
;       Input:
;               semmi
;       Output:
;               semmi
;       Minden regiszter mentve
;-----------------------------------------;
                                                ;
WAIT    PROC    NEAR                            ;
                PUSH    BX                      ;WAIT-nek észrevehetet-
                PUSH    CX                      ; lennek kell lennie!
                                                ;
                MOV     BX, 40                  ;Külső ciklusszámláló
WAIT_LOOP1:                                     ;
                XOR     CX, CX                  ;Belső ciklusszámláló
WAIT_LOOP2:                                     ;
                LOOP    WAIT_LOOP2              ;Várunk egy kicsit
                DEC     BX                      ;
                JNZ     WAIT_LOOP1              ;meg még egy kicsit
                                                ;
                POP     CX                      ;
                POP     BX                      ;
                RET                             ;
WAIT    ENDP                                    ;
                                                ;
CODE    ENDS                                    ;
        END     ENTRY                           ;
```

A fenti kis program példáink sorában az első, ahol indokoltá vált szubrutin megírása, és mindjárt kettő is. Mindenki érzi, hogy mind a várakozás, mind pedig a standard input bufferének törlése eléggé fontos és önálló ahhoz, hogy ne a főprogramba írjuk, hanem külön rutinba, hogy bármikor, bárhonnan meghívhassuk.

A WAIT rutinhoz hasonló gyakran szerepel programokban. Itt azért van rá szükség, hogy a felhasználó az első utasítás után "teletömhesse" valamivel a standard input bufferét, és így meggyőződhessen arról, hogy a CLRBUF rutin, amelyet a második üzenet kiírása előtt hívtunk meg, valóban üríti a buffert.

Megjegyezzük még, hogy a WAIT rutinban a kettősciklus külső ciklusváltozója bemenő paraméter is lehetne, ezzel programból állíthatjuk be a várakozási időt.

V.3.5. Kiírás a standard nyomtató file-ba

Az alábbi példaprogram nem valamilyen szöveget, hanem egy ún. ESCAPE szekvenciát küld ki a standard printer file-ra (az ESCAPE szekvencia egy ESCAPE karakterrel kezdődő karaktersorozat, amely nem látható szöveg, hanem a periféria programozására szolgáló parancssorozat - lásd még a "Bővitett képernyő- és klaviatúrafunkciók" fejezetet). Tegyük fel, hogy gépünkhöz egy EPSON nyomtatót csatoltunk. Ekkor az alább "kinyomtatott" (azaz a standard nyomtató file-ra kiírt) szekvencia ezt a nyomtatót az "ELITE" módba állítja át. Ebben a nyomtatási módban a nyomtató keskenyebb, elegánsabb, "elit" minőségű betűket nyomtat.

Amennyiben valamilyen szöveget akarunk nyomtatni, akkor ugyanilyen módon, karakterenként kell a szöveget kiküldeni a standard nyomtató file-ra.

```
TITLE   Nyomtató átprogramozása ELITE módba
;
; Konstansok
;

DOS_PRWRCH      EQU     05H             ;Nyomtatás STDPRN-re
                                        ;
CHR_ESC         EQU     27              ;
PRT_SETELITE    EQU     'M'             ;
```

```
;
; Makrók
;
DOSCALL MACRO   CODE
                MOV     AH, CODE
                INT     21H
        ENDM
;
; Kódszegmens
;

CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME   CS:CODE
        ORG     100H
ENTRY:                                  ;
                                        ;
;               JMP     START           ;Nincs adatterület, ez
;START:                                 ; itt felesleges
                                        ;
                MOV     DL, CHR_ESC     ;ESCAPE karakter
                DOSCALL DOS_PRWRCH      ; kiírása
                                        ;
                MOV     DL, PRT_SETELITE;A módbeállitási
                DOSCALL DOS_PRWRCH      ; parancs kiírása
                                        ;
                INT     20H             ;Kilépés
                                        ;
CODE        ENDS                        ;

        END     ENTRY
```

Célszerűnek látszik egy olyan szubrutin megírása, amely nem a nyomtató felprogramozását végző ESCAPE szekvenciát, hanem valamilyen stringet ír ki a nyomtatóra. Ugyanezt nem felesleges megvalósítani esetleg más standard file-okra sem. A standard output string-nyomtató funkciója kicsit kényelmetlen, ugyanis a dollárjelet nem tudjuk vele kiíratni. Helyes lenne átvenni a C nyelv egyik okos trükkjét: a string az első nulla kódú karakterig tart. Ezzel a megállapodással minden, látható karakterekből álló stringet tudunk kezelni, hiszen nincs karaktervesztés. A nulla kódú karakter képe az üres karakter, de ennek nyomtatására rendelkezésünkre áll a 20 kód, a valódi "blank".

V.3.6. Az egyéb standard file-ok kezeléséről

A standard error file mögött átirányíthatatlanul a képernyő van. Használata akkor célszerű, ha olyan üzeneteket kell kiírnunk, amelyeknek mindenképpen a képernyőre kell kerülniük, például azoknak a fontos hibaüzeneteknek, amelyeket nem szabad átirányítani. Mint a fentiekből kiderül, nincsenek olyan MS-DOS funkciók, amelyek a standard hibajelző file-t kezelnék. Erre a file-ra az "MS-DOS file-kezelés" fejezetben leírt funkciók segítségével küldhetünk ki üzeneteket, ugyanúgy, ahogy bármely egyéb MS-DOS file-ra írnánk.

A standard kiegészítő file (a soros vonal) kezelése pontosan úgy lehetséges a 03 és 04 funkciók használatával, mint a standard input és output file-ok kezelése a 01 és 02 funkciókkal. Számos nehézséggel kell azonban megküzdenünk akkor, ha a vonalat kezelni akarjuk. Például vezérelnünk kell a vonal sebességét, a paritásellenőrzés módját stb. Ehhez azonban olyan eszközök szükségesek, amelyek e fejezet témakörébe nem illeszkednek, mert túlmutatnak a standard file-ok kezelésén. A szükséges eszközök egy részét az "MS-DOS file-kezelés", más részét pedig a 3. kötet "Az aszinkron vonal (RS232)" fejezetében fogjuk megismerni. Az alábbiakból kitűnik azonban, hogy a rendszer által biztosított eszközök nem elegendőek a probléma teljes megoldásához. Gyakorlatilag három út áll előttünk.

Az első, hogy az MS-DOS MODE nevű külső parancsával megszabjuk a vonal paramétereit, majd a vonalat standard file-ként kezeljük. Talán ez a legkevésbé nehéz, de ekkor a vonalat szinkron programozással kell kezelnünk, azaz minden egyes karakter megérkezését meg kell várnunk. Ez sajnos a gyakorlatban nem felel meg semmilyen értelmes célra. E módszernek másik hátránya, hogy a MODE parancsot kézzel vagy programból mindenképpen el kell indítani az átvitel megkezdése előtt. Ha pedig "ügyesebb" vonalkezelést akarunk megvalósítani, akkor bele kell kevernünk a standard file-kezelő funkciók közé az "MS-DOS file-kezelés" fejezetben ismertetett 44 funkció használatát.

A második lehetséges út az, hogy a vonalat a ROM BIOS megfelelő funkciói segítségével kezeljük. Meg kell azonban mondani azt, hogy a ROM driver sem elég hatékony a vonal jó kezeléséhez. Az első nehézség ugyanaz, mint a standard file-ként való kezelés esetén - csakis szinkronban olvashatunk a vonalról. A másik probléma is közös (nem véletlenül, hiszen a DOS is e driver segítségével kezeli a vonalat). Ez pedig az, hogy a vonalat

kezelő áramkör lehetőségeinek csak egy részét használja ki a ROM BIOS driver. Vessük össze a 3. kötet "Az aszinkron vonal (RS232)" fejezetben olvasható lehetőségeket a BASICA.COM lehetőségeivel. Ez a BASIC saját drivert használ, és sokkal többet tud, mint az eredeti driver.

A harmadik és leginkább célravezető (egyúttal persze legbonyolultabb) út pedig az, hogy magunk írunk egy vonali drivert. Ez a lehetőség azonban túl messzire vezet, könyvünk szűk keretei nem teszik lehetővé ennek ismertetését (eléggé leegyszerűsített példa olvasható a 3. kötet "Az aszinkron vonal (RS232)" fejezetében). Egy ilyen programnak nagyjából a következőket kell tudnia: felhasználója ne karakterek, hanem üzenetek átvitelében gondolkodhasson, és az üzenetek átvitele a "háttérben" menjen végbe. Az "üzenetben gondolkodás" körülbelül azt jelenti, hogy kiírásnál a felhasználó megadja egy üzenet memóriacímét és hosszát, majd meghívja a drivert, amely igen rövid időn belül visszaadja a vezérlést, miközben az üzenet átvitelét a háttérben folytatja. Vétel esetén pedig egy üres, eléggé hosszú adatterület címét kell megadni, ahová a vett üzenetet el kívánjuk helyezni. Meghívjuk a drivert, amely most is hamar visszaadja a vezérlést (hogy az üzenet vétele közben egyéb bokros teendőinket intézhessük), és akkor szól csak vissza, ha egy komplett üzenet megérkezett.

Mit jelent az, hogy a driver a háttérben dolgozik? "Egyszerűen" annyit, hogy a vonalat megszakítások, hardware interruptok segítségével kezeli, és két interrupt között (azaz a fizikai átvitel idején) a vezérlés a felhasználói program utasításain van, amely így végezheti saját munkáját, és nem kell áhítattal csüggenie a vonalon, sorsa nincs a vonal másik végén, a másik gépen futó program kezében. Ha ezek a feltételek nem teljesülnek, nem lehetünk elégedettek a vonali driverrel. Az olyan "részletkérdések" is rendkívül fontosak persze, mint az átvitel sebessége, az adatbiztonság megvalósítása, az átviteli hibák felismerése, a transzparencia - ezek azonban nem annyira elvi, mint inkább technikai nehézségek. Aki megpróbált valaha ilyen feltételeknek eleget tévő programot írni, az tisztában van a fent vázolt problémák bonyolultságával, és (ha csak lehet) inkább egy negyedik utat választ: megvásárol egy profi vonal- vagy méginkább hálózatkezelő programot, és nem kínlódik olyan nehézségek legyűrésével, amelyeket mások már jól (jobban) megoldottak.

VI. Bővitett képernyő- és klaviatúrafunkciók

VI.1. Az ANSI.SYS driver használata

A UNIX operációs rendszer nyomdokain haladva az MS-DOS is egy rendkivül egyszerű, ún. teletype operátori konzolt tételez fel, azaz egy egyszerű papíros terminált. Ennek képességei mindössze a látható karakterek kinyomtatására, valamint a sor végén soremelésre, és a kocsinak, az irófejnek sor elejére pozicionálására korlátozódnak. (Ez összhangban van azzal az elvvel is, hogy a standard output és hibafile-ok szigorúan szekvenciális file-ok).

Maga a gép persze egy korszerű katódsugárcsöves képernyőjű terminált tartalmaz, amelynek vezérlésére egy igen intelligens Motorola (!) áramkör szolgál. Ez a képernyő természetesen minden szépséget tud: az ernyő bármely pontjára vezérelhetjük a cursort, bárhová irhatunk, színes ernyő esetén bármelyik karaktert a kivánt szinűre szinezhetjük és igy tovább.

Mivel a beépitett DOS-driverek a fenti elv miatt nem használhatják ki ezeket a lehetőségeket, a rendszer egyik alapfilejaként létrehozták az ANSI.SYS file-t, amely nem más, mint egy majdnem ANSI szabvány szerint működő képernyő-driver. Segitségével csaknem úgy hajthatjuk meg a képernyőt, mint egy VT100 tipusú displayt. A képernyőt vezérlő áramkör természetesen ennek sokszorosára képes; lehetőségeinek egy részét a ROM BIOS rutinjai ki is használják. Erről bővebben olvashatunk sorozatunk 3. kötetében, "A képernyő" fejezetben.

Most azt irjuk le, miképpen használhatjuk az ANSI drivert a képernyő vezérlésére. Ehhez először is be kell épitenünk a drivert az MS-DOS rendszerbe. Ez nem túlzottan nehéz feladat. Csak annyit kell tennünk, hogy a CONFIG.SYS file-ba beirjuk a

```
DEVICE = ANSI.SYS
```

parancssort. A következő rendszertöltés után az ANSI driver már az operációs rendszer része lesz.

Ezután minden kiirást ugyanúgy végzünk el, mint eddig: egyszerűen végrehajtjuk a megfelelő DOS-funkciót, és minden eddigi lehetőségünk változatlanul megmarad. Ha azonban egy úgynevezett ESCAPE szekvenciát küldünk ki a standard output file-ra, akkor az ANSI driver "felébred", észreveszi, hogy nem valódi kiirást kivánunk végrehajtani, hanem egy neki szóló parancsot adtunk

ki; megpróbálja értelmezni és végrehajtani a parancsot. Ezekkel az ESCAPE szekvenciákkal például tetszőleges helyre állíthatjuk a cursort, törölhetjük a képernyő egy részét stb.

Az ESCAPE szekvencia egy olyan karaktersorozat, amely az ESCAPE karakterrel kezdődik (kódja decimális 27), és utána megadott rendben néhány megszabott karakter következik. Arra szolgál, hogy perifériavezérlő parancsokat továbbítson a periféria-drivernek (vagy, mint a nyomtató esetében láthattuk, magának a perifériának, amely kellően intelligens ilyen parancsok fogadására).

Amennyiben elvétjük az ESCAPE szekvenciát, vagyis nem legális karaktersorozatot írunk ki, akkor a driver a nem értelmezhető karaktereket továbbítja a standard output file-ra, és azok megjelennek. Ez a program felhasználóját könnyen készítetheti egy kis fejtörésre. Ugyanilyen meglepetéseket okozhat az is, ha a szekvencia elejéről elvész az ESCAPE karakter - ekkor a parancs karakterről karakterre kiíródik, és ilyenkor bizony elég értelmetlen "szöveget" láthatunk a képernyőn.

Az alábbiakban olvashatóak az ANSI driver parancsai. Használatuk elég kényelmetlennek látszik, és jóval kevesebbet érhetünk el segítségükkel, mint a ROM BIOS rutinok direkt meghívásával. Az ANSI driver egyik előnye, hogy míg a ROM BIOS rutinok a képernyőre írnak, az ANSI driver használata esetén a standard output file-ra írunk, ez pedig átirányítható. Ráadásul a ROM BIOS semmiféle bevett képernyőkezelési szabványhoz nem hasonlít, míg az ANSI driver segítségével csaknem pontosan egy igen elterjedt szabvány, a VT100 szerint kezelhetjük a képernyőt.

> Megjegyzés: az, hogy az ANSI driver "csaknem" VT100 szabvány szerint kezeli az ernyőt, azt jelenti, hogy a parancsai kompatibilisek a VT100 parancsokkal, de van néhány olyan VT100 parancs, amelyet az ANSI driver nem tud. Például egy igazi VT100 átállítható VT52 szabványra, az ANSI driver pedig nem. Mindez megfordítva is igaz: van olyan parancs az ANSI driverben, amelyet a szabványos VT100 nem ismer.

Ennek jelentősége felbecsülhetetlen, különösen magasszintű nyelvek használata esetén. Valószínű, hogy assemblerben megírt programot úgysem fogunk áthozni egy másik gépről. Nincs azonban akadálya annak, hogy egy szép COBOL, FORTRAN vagy egyéb programot áthozzunk. Ha pedig ez a program VT100 terminálra íródott, akkor (legalábbis ebből a szempontból) jól fog futni itt is.

VI.2. Az ANSI driver parancsai

A parancsok leirásánál használt jelölések:

    #   - numerikus paraméter, ASCII-karaterekkel leírt szám, amely minden esetben elhagyható. Az alapértelmezés ez esetben mindenütt 1.
    ESC - az ESCAPE karakter, kódja 27D (1BH)

Lássunk egy példát egy ANSI szekvencia assembly megjelenésére!

```
CUP1240         DB      27, '12;40H$'
```

ahol a teljes parancsot ASCII-karakterekkel adtuk meg, a numerikus paramétereket is, tehát nem bináris értékekből állitjuk össze a parancsot. Az utolsó dollárjel csak arra szolgál, hogy a 09 funkció számára lezárja a stringet.

Az egyes parancsok után megadjuk a parancsok szimbolikus nevét. Talán ezt célszerű névként használni, ha assemblerben definiáljuk és elnevezzük az egyes ESCAPE szekvenciákat.

VI.2.1. Cursorvezérlés

ESC [#;#H CUrsor Position, ANSI_CUP

A cursor pozicionálása adott képernyőpontra. Az első szám a sor, a második az oszlop pozíciója.
Alapértelmezés: #,# = 1,1 - azaz a bal felső sarok.

Megjegyzés: az ANSI driver számára a képernyő sorai 1-től 25-ig, míg a ROM BIOS driver számára 0-tól 24-ig értelmezettek. Az oszlopok számának jelölése hasonló eltérést mutat.

ESC [#A CUrsor Up, ANSI_CUU

"#" sorral feljebb mozgatja a cursort.
Alapértelmezés: # = 1.
Nem mozdul a cursor, ha a felső sorban volt, vagy odaért.

ESC [#B CUrsor Down, ANSI_CUD

"#" sorral lefelé mozgatja a cursort.
Alapértelmezés: # = 1.
Nem mozdul a cursor, ha az alsó sorban volt, vagy odaért.

ESC [#C CUrsor Forward, ANSI_CUF

"#" oszloppal jobbra ("előre") mozgatja a cursort.
Alapértelmezés: # = 1.
Nem mozdul a cursor, ha a jobb szélső oszlopban volt, vagy odaért.

ESC [#D CUrsor Backward, ANSI_CUB

"#" oszloppal balra ("vissza") mozgatja a cursort.
Alapértelmezés: # = 1.
Nem mozdul a cursor, ha a bal szélső oszlopban volt, vagy odaért.

ESC [#;#f Horiz. & Vertical Position, ANSI_HVP

A "#,#" pozícióba teszi a cursort (ugyanaz, mint "H").
Alapértelmezés: #,# = 1,1 - azaz a bal felső sarok.

ESC [6n Device Status Report, ANSI_DSR

A driver a standard input file-on egy olyan ESCAPE szekvenciát küld vissza, amelyből a cursor pozícióját tudhatjuk meg. Ez a szekvencia olvasható alább. Lásd "A cursor lekérdezése" példát!

ESC [#;#R Cursor Position Report, ANSI_CPR

Ezt az ESCAPE szekvenciát küldi vissza a driver a standard input file-on keresztül az előző hívásra. Lásd "A cursor lekérdezése" példát!

ESC [s Save Cursor Position, ANSI_SCP

A driver elmenti a cursor pillanatnyi pozícióját. Az elmentett pozícióra a következő hívással állíthatjuk vissza a cursort (ESC [u).

ESC [u Restore Cursor Position, ANSI_RCP

Az előző paranccsal elmentett pozícióra állítja vissza a cursort.

VI.2.2. Törlés a képernyőn

ESC [2J Erase in Display, ANSI_ED

Törli a teljes képernyőt, és a cursort a bal felső sarokba viszi.

Megjegyzés: meg kell mondanunk, hogy a "2" teljesen felesleges, az irodalom bájos tévedésének tekinthető. Ugyanez történik, bármit is írunk a paraméter helyére, vagy ha egészen elhagyjuk.

ESC [K Erase in Line, ANSI_EL

Törli a sor cursortól jobbra eső részét.

VI.2.3. A video-mód megváltoztatása

ESC [#;#;...;#m Set Graphic Rendition, ANSI_SGR

Grafikus sajátosságok beállítása - a paraméterek (amelyek tetszőleges sorrendben és számban adhatók meg) a következőket jelentik:

0 - minden kikapcsolva — normál fekete-fehér
1 - nagy intenzitás (előtér világosabb szinű)
4 - aláhúzás (csak IBM monochrom képernyőn)
5 - villogtatás
7 - inverz video-mód
8 - láthatatlan karakterek

Az előtér szinének (a karakterek szinének) beállitása:

30 - fekete előtér
31 - vörös előtér
32 - zöld előtér
33 - sárga előtér
34 - kék előtér
35 - lila előtér
36 - türkiz előtér
37 - fehér előtér

A háttér színének (az alapszínnek) beállitása:

40 - fekete háttér
41 - vörös háttér
42 - zöld háttér
43 - sárga háttér
44 - kék háttér
45 - lila háttér
46 - türkiz háttér
47 - fehér háttér

Az attributumok nem e parancs kiadásától változnak meg, hanem a további kiírások során jutnak érvényre. Tehát az ezután kiírt szöveget, és nem a teljes ernyőt tudjuk átszínezni.
Hozzá kell fűznünk, hogy sokkal kényelmesebb (bár lassúbb) az ANSI driver, mint a ROM BIOS használata, ha csak a kiírandó szöveget akarjuk másképpen színezni.

ESC [=#h vagy
ESC [?7h — Set Mode, ANSI_SM

Módváltás "#" szerint. "#" lehet:

0 - 40 x 25 fekete-fehér alfanumerikus
1 - 40 x 25 színes alfanumerikus
2 - 80 x 25 fekete-fehér alfanumerikus
3 - 80 x 25 színes alfanumerikus
4 - 320 x 200 színes
5 - 320 x 200 fekete-fehér
6 - 640 x 200 fekete-fehér
7 - a cursor leejtése - a sor végén a kiírás automatikusan a következő sorban folytatódik, a legalsó sorban felfelé mozgatja az ernyőt, és új sort nyit (rolloz).
Az alapértelmezés a leejtés.

ESC [=#l vagy
ESC [?7l — Reset Mode, ANSI_RM

Mint az előző funkció, csakhogy a 7 mód nem beállitja, hanem megszünteti a cursor új sorba való átdobását. Ekkor, ha egy kiírással túllépnénk a sor végét, a karaktereket a cursor mozgatása nélkül a sor utolsó pozíciójára irja (igy persze elég kevés látszik belőlük).

VI.2.4. Beolvasott kódok konvertálása

```
ESC [#;#;...#p          Keyboard Key Reassignment, ANSI_KKR
ESC [#;"string";p
ESC [#;"string1";#;..;"string2";...;#p
```

Az első "#" által megadott ASCII kódú billentyű leütése után a további "#"-ok által megadott kódsorozatot fogjuk megkapni a klaviatúra (pontosabban a standard input file) olvasásakor. Egy ilyen kódsorozat, mint a második és harmadik változat mutatja, amennyiben egy hosszabb stringet akarunk megadni, idézőjelek között is megadható.
Ha olyan billentyű jelentését akarjuk újradefiniálni, amelynek nincs ASCII-kódja (funkció- és numerikus billentyűk), akkor az első "#" 0 lesz, és egy pontosvessző után az illető billentyű scan-kódját adjuk meg (lásd az idevágó példaprogramot). Egy egyszerűbb hozzárendelési példát itt adunk meg, mivel a példa egy bonyolultabb esetet mutat be. Amennyiben kiírunk egy

```
ESC [65;81p     vagy     ESC [65;"Q";p
```

karaktersorozatot, akkor az "A" leütésekor bármely, a standard input file-t olvasó program "Q"-t fog kapni. Ha nem az ASCII-tartományba eső billentyűt akarunk átdefiniálni (funkcióbillentyűk, numerikus billentyűzet elemei), akkor az ASCII-kódot egy "0;scan-kód" sorozattal kell helyettesíteni (lásd "Funkcióbillentyűk újradefiniálása" példa).

VI.3. Példaprogramok az ANSI.SYS driverhez

VI.3.1. Cursormozgatás

Az alábbi kis program egy kis bújócskát játszik a képernyőn a cursorral. Először kiír egy szöveget ("Hihi, itt vagyok"), majd, mikor leütünk egy billentyűt, arrébb teszi a cursort, és újra kiírja a fenti mélyértelmű szöveget. Időnként beletöröl az ernyőbe, hogy az egész változatosabb legyen. A program rendki-

vül egyszerű, "munkájának" teljes kihüvelyezését az Olvasóra bízzuk.

```
TITLE   Cursorvezérlés az ANSI driverrel
;
; Konstansok
;
DOS_STRDCH      EQU     08H             ;Input echo nélkül
DOS_WAITKEY     EQU     DOS_STRDCH      ; és várakozás
DOS_STWRSTR     EQU     09H             ;Kiírás STDOUT-ra
CHR_ESC         EQU     27              ;ESCAPE kódja

;
; Makrók
;
DOSCALL MACRO   CODE
                MOV     AH, CODE
                INT     21H
        ENDM

PRTSTR  MACRO   STRING
                MOV     DX, OFFSET STRING
                DOSCALL DOS_STWRSTR
        ENDM

;
; Kódszegmens
;
CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:CODE, ES:CODE, SS:CODE
        ORG     100H

ENTRY:                                  ;
                JMP     START           ;-------;
                                                ;
TEXT            DB      'Hihi, itt vagyok!$'    ;Tesztszöveg
;-----------------------------------------------;
; Cursorpozicionáló parancsok
;
CP_1020         DB      CHR_ESC, '[10;20H', '$'
CP_HOME         DB      CHR_ESC, '[H', '$'      ;Mint '1;1' !
CR_UP1          DB      CHR_ESC, '[1A', '$'     ;
```

```
;
; Ernyőtörlési parancsok
;
ERS_SCR         DB      CHR_ESC, '[2J', '$'     ;'[J' is lehet!
ERS_ROW         DB      CHR_ESC, '[K', '$'

START:                                          ;
                PRTSTR  TEXT                    ;Tesztszöveg ki
                DOSCALL DOS_WAITKEY             ;Várunk
                                                ;
                PRTSTR  CP_1020                 ;Kb. az ernyő közepe
                PRTSTR  TEXT                    ;Tesztszöveg ki
                DOSCALL DOS_WAITKEY             ;Várunk
                                                ;
                PRTSTR  CP_HOME                 ;Bal felső sarok
                PRTSTR  TEXT                    ;Tesztszöveg ki
                DOSCALL DOS_WAITKEY             ;Várunk
                                                ;
                PRTSTR  CP_HOME                 ;Bal felső sarok
                PRTSTR  ERS_ROW                 ;Első sor törlése
                DOSCALL DOS_WAITKEY             ;Várunk
                                                ;
                PRTSTR  ERS_SCR                 ;Ernyő törlése
                DOSCALL DOS_WAITKEY             ;Várunk
                                                ;
                PRTSTR  CP_1020                 ;Az ernyő közepe
                PRTSTR  TEXT                    ;Tesztszöveg ki
                DOSCALL DOS_WAITKEY             ;Várunk
                                                ;
                PRTSTR  CR_UP1                  ;Egy sor fel
                PRTSTR  TEXT                    ;Tesztszöveg ki
                DOSCALL DOS_WAITKEY             ;Várunk
                                                ;
                INT     20H                     ;
CODE    ENDS                                    ;
        END     ENTRY                           ;
```

A fenti programmal kapcsolatban csak néhány kiegészítő megjegyzés kívánkozik ide. Az első egy konkrét dolog. Le kell szögeznünk, hogy a cursor ilyen módon való pozicionálása több mint kényelmetlen. Nem túl gyakori eset az, hogy előre tudjuk, hová is fogjuk pozicionálni a cursort. Ha pedig a kívánt pozíció ko-

ordinátái csak futás közben derülnek ki, akkor eléggé kényelmetlen kikeverni egy olyan ESCAPE szekvenciát, amelynek kiírása megvalósítja a kívánt pozicionálást, hiszen kénytelenek vagyunk egy decimális konverziót kétszer is végrehajtani, és szinte bitenként összeállítani egy karaktersorozatot.

A második megjegyzés egy teljesen elvi kérdésre vonatkozik. Figyeljük meg, hogy a program második konstansdefiníciója az elsőként definiált konstansra hivatkozik - az adott értéknek két nevet is adtunk. Lehetett volna a második definíció is egyszerűen

```
DOS_WAITKEY     EQU     8
```

de ekkor két "különböző" 8 érték szerepel a programban, nem látszik, hogy ugyanazt nevezzük kétféleképpen. Ez egy felesleges akadékoskodásnak tűnik, holott a "minden adatnak pontosan egy forrása legyen" elv egyik megjelenési formája.

Mielőtt pedig végképpen búcsút vennénk ettől a kis programtól, próbáljuk ki, mi történik, ha standard outputját átirányítjuk egy lemezfile-ba! Legyen a neve például ANSI.XXX. Ekkor persze a file karakterről karakterre tartalmazni fogja az összes ESCAPE szekvenciát, és ha a program lefuttatása után egy TYPE paranccsal kiíratjuk ezt a file-t a terminálra, akkor az egész program újra "le fog játszódni", csak sokkal gyorsabban, várakozás nélkül.

VI.3.2. A cursor lekérdezése

Az alábbi program lekérdezi, megváltoztatja, majd helyreállítja a cursor pozícióját.

```
TITLE   A cursor lekérdezése az ANSI driverrel
;
; Konstansok
;
DOS_STWRSTR     EQU     09H             ;String kiírása
DOS_STRDCH      EQU     08H             ;Karakter beolvasása
DOS_WAITKEY     EQU     DOS_STRDCH      ;Várakozás
CHR_CPR         EQU     'R'             ;CPR üzenet vége
CHR_CUP         EQU     'H'             ;CUP üzenet vége
CHR_DOLLAR      EQU     '$'             ;Dollárjel
CHR_ESC         EQU     27              ;ESC karakter
```

```
;
; Makrók
;

DOSCALL MACRO   CODE
                MOV     AH, CODE
                INT     21H
        ENDM

PRTSTR  MACRO   STRING
                MOV     DX, OFFSET STRING
                DOSCALL DOS_STWRSTR
        ENDM

;
; Kódszegmens
;

CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:CODE, ES:CODE, SS:CODE
        ORG     100H

ENTRY:                                  ;
                JMP     START           ;

TEXT1           DB      'Hihi, új cursorpozíció!$'
TEXT2           DB      'Hehe, régi cursorpozíció!$'

DSR             DB      CHR_ESC, '[6n$' ;Lekérdezési parancs
CPR             DB      10 DUP ( ? )    ;Buffer a válasznak
CP_1020         DB      CHR_ESC, '[10;20H', '$'

START:                                  ;
                PRTSTR  DSR             ;Cursor lekérdezése
                MOV     DI, OFFSET CPR  ;Buffercím be
GETCURPOS:                              ;
                DOSCALL DOS_STRDCH      ;Olvasunk, mig vége
                CMP     AL, CHR_CPR     ; nincs
                JE      GETCURPOS_END   ;-------;
                MOV     BYTE PTR [ DI ], AL     ;Elmentjük
                INC     DI              ; a karaktereket
                JMP     GETCURPOS       ;
```

```
GETCURPOS_END:                          ;
                MOV     BYTE PTR [ DI ], CHR_CUP
                INC     DI              ;CPR-CUP csere
                MOV     BYTE PTR [ DI ], CHR_DOLLAR
                                        ;String lezárása
                PRTSTR  CP_1020         ;Ernyő közepe
                PRTSTR  TEXT1           ;Tesztszöveg ki
                DOSCALL DOS_WAITKEY     ;Várunk
                PRTSTR  CPR             ;Cursor helyreáll
                PRTSTR  TEXT2           ;Tesztszöveg ki
                INT     20H             ;Kilépés
CODE    ENDS                            ;
        END     ENTRY
```

Egyetlen apró nehézséget kellett leküzdenünk: nem határozott számú karakter beolvasására volt szükség, hanem egy úgynevezett terminátorig, határolójelig olvastunk be. Ezt a terminátort helyettesítettük aztán egy másik ANSI parancs kódjával, hogy a visszakapott cursorpozíciót felhasználhassuk a cursor pozicionálására. A fenti feladatot azonban az ANSI_SCP és ANSI_RCP szekvenciákkal egyszerűbben is megoldhattuk volna - csak akkor nem kapjuk meg a cursor koordinátáit.

VI.3.3. Video-mód váltás

A most következő program különböző video-módokat állít be, és különböző tesztszövegek kiírásával mutatja be a beállított video-módokat. Lefuttatáskor figyeljük meg azt, hogy a program első néhány kiírása során a beállított színek nem az egész ernyőre, hanem csak a frissen kiírt karakterekre vonatkoznak. Ezt az eredményt direkt ROM BIOS hívásokkal elég fáradságos munkával tudnánk csak elérni.

```
TITLE   Video-mód váltás az ANSI driverrel
;
; Konstansok
;
DOS_STRDCH      EQU     08H
DOS_WAITKEY     EQU     DOS_STRDCH
DOS_STWRSTR     EQU     09H             ;String kiírása
```

```
CHR_ESC         EQU     27              ;ESCAPE kódja
CHR_CR          EQU     13              ;Kocsivissza
CHR_LF          EQU     10              ;Soremelés

;
; Makrók
;

DOSCALL MACRO   CODE
                MOV     AH, CODE
                INT     21H
        ENDM

PRTSTR  MACRO   STRING
                MOV     DX, OFFSET STRING
                DOSCALL DOS_STWRSTR
        ENDM
;
; Kódszegmens
;

CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:CODE, ES:CODE, SS:CODE
        ORG     100H

ENTRY:                                  ;
                JMP     START           ;

TEXT1   DB      'Piros betű kék alapon'
        DB      CHR_CR, CHR_LF, '$'
TEXT2   DB      'Villogó piros betű kék alapon'
        DB      CHR_CR, CHR_LF, '$'
TEXT3   DB      'Normál - fehér betű fekete alapon'
        DB      CHR_CR, CHR_LF, '$'
TEXT4   DB      'Ez a szöveg túlírja a sort, mert hosszú!$'
TEXT5   DB      'Ez a szöveg igen nagyon hosszú, és nem is'
        DB      ' látom a végét!$'
TEXT6   DB      'Alfanumerikus 40x25 színes ernyő$'
TEXT7   DB      '640x200 pontos grafika$'
TEXT8   DB      'Alfanumerikus 80x25 színes ernyő$'

TXTCRLF DB      CHR_CR, CHR_LF, '$'     ;Kocsivissza-soremelés
```

```
;
; Módváltási parancsok
;
SGR_RED_ON_BLUE DB      CHR_ESC, '[31;44m$'     ;Piros kékre
SGR_BLINK       DB      CHR_ESC, '[5m$'         ;Villogás
SGR_NORM        DB      CHR_ESC, '[0m$'         ;Normál mód

SM_WRAP         DB      CHR_ESC, '[?7h$'        ;Cursor átdob.
RM_TRWN         DB      CHR_ESC, '[?7l$'        ;Cursor marad

SM_4025C        DB      CHR_ESC, '[=1h$'        ;40x25 színes
SM_8025C        DB      CHR_ESC, '[=3h$'        ;80x25 színes
SM_GRLG         DB      CHR_ESC, '[=6h$'        ;640x200 graf.


START:                                  ;
                PRTSTR  SGR_RED_ON_BLUE ;Színbeállítás
                PRTSTR  TEXT1           ;Tesztszöveg ki
                DOSCALL DOS_WAITKEY     ;
                                        ;
                PRTSTR  SGR_BLINK       ;
                PRTSTR  TEXT2           ;Tesztszöveg ki
                DOSCALL DOS_WAITKEY     ;
                                        ;
                PRTSTR  SGR_NORM        ;
                PRTSTR  TEXT3           ;Tesztszöveg ki
                DOSCALL DOS_WAITKEY     ;
                                        ;
                PRTSTR  SM_WRAP         ;
                PRTSTR  TEXT4           ;Tesztszöveg ki
                PRTSTR  TEXT4           ; háromszor, hogy
                PRTSTR  TEXT4           ; lássuk az átdobást
                PRTSTR  TXTCRLF         ;Kocsivissza ki
                DOSCALL DOS_WAITKEY     ;
                                        ;
                PRTSTR  RM_TRWN         ;
                PRTSTR  TEXT5           ;Tesztszöveg ki
                PRTSTR  TEXT5           ; háromszor, hogy
                PRTSTR  TEXT5           ; lássuk: cursor marad
                PRTSTR  TXTCRLF         ;Kocsivissza ki
                DOSCALL DOS_WAITKEY     ;
```

```
                PRTSTR  SM_4025C        ;Módválasztás
                PRTSTR  SGR_RED_ON_BLUE ;Színezünk
                PRTSTR  TEXT6           ;Tesztszöveg ki
                DOSCALL DOS_WAITKEY     ;
                                        ;
                PRTSTR  SM_GRLG         ;Módválasztás
                PRTSTR  SGR_RED_ON_BLUE ;Színezünk, de hiába
                PRTSTR  TEXT7           ;Tesztszöveg ki
                DOSCALL DOS_WAITKEY     ;
                                        ;
                PRTSTR  SM_8025C        ;Módválasztás
                PRTSTR  SGR_RED_ON_BLUE ;Színezünk
                PRTSTR  TEXT8           ;Tesztszöveg ki
                DOSCALL DOS_WAITKEY     ;
                                        ;
                PRTSTR  SGR_NORM        ;Módválasztás
                PRTSTR  TXTCRLF         ;Kocsivissza ki
                                        ;
                INT     20H             ;Kilépés
                                        ;
CODE    ENDS                            ;
        END     ENTRY
```

VI.3.4. Funkcióbillentyűk újradefiniálása

Az alábbi program két billentyű hozzárendelését végzi el. Az egyik a gyakorlatban is fontos parancs kiadását segíti: az F10 funkcióbillentyűhöz a DIR C: rendszerparancsot rendeli hozzá. A másik hozzárendelés még sokkal fontosabb - az F9 funkcióbillentyű új tartalma: egy CD\GAMES és rögtön utána egy TETRIS parancs, melynek hatására bárhol voltunk is eddig, átlépünk a GAMES nevű directory-ba, és útnak indítjuk a népszerű TETRIS nevű játékprogramot (feltéve persze, hogy a kurrens lemezen ott van ez a fontos directory és benne a TETRIS játék). S mindezt egyetlen billentyű (meglehet, véletlen) megnyomásával! Komolyra fordítva: az utóbbi példa azt illusztrálja, hogy nemcsak egy, de több parancssort is hozzá tudunk rendelni egy billentyűhöz. Felhívjuk a figyelmet arra, hogy ez nem kötelezően egy funkcióbillentyű - csekély fáradsággal elérhetjük, hogy minden billentyű leütésére a TETRIS jelentkezzen. Ha az ezt megvalósító

programot gondosan lefuttatjuk az AUTOEXEC.BAT file-ban, akkor tréfás meglepetést szerezhetünk tájékozatlan kollégáinknak.

```
TITLE   Klaviatúrabillentyűk átdefiniálása
;
; Konstansok
;
DOS_STWRSTR     EQU     09H             ;String kiírása
CHR_ESC         EQU     27              ;ESCAPE kódja

;
; Makrók
;
DOSCALL MACRO   CODE
                MOV     AH, CODE
                INT     21H
        ENDM
PRTSTR  MACRO   STRING
                MOV     DX, OFFSET STRING
                DOSCALL DOS_STWRSTR
        ENDM
;
; Kódszegmens
;
CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:CODE, ES:CODE, SS:CODE
        ORG     100H
ENTRY:                                  ;
                JMP     START           ;Ugrás a belépésre
                                        ;
KKR_TETRIS      DB      CHR_ESC         ;A TETRIS elindítása
                DB      '[0;67;"CD\GAMES";13;"TETRIS";13p$'
KKR_DIRC        DB      CHR_ESC,        ;Directory-lista
                DB      '[0;68;"dir c:";13p$'
                                        ;
START:                                  ;
                PRTSTR  KKR_TETRIS      ;
                PRTSTR  KKR_DIRC        ;
                                        ;
                INT     20H             ;
CODE    ENDS                            ;
        END     ENTRY                   ;
```

Figyeljünk meg itt egy "apró" trükköt! Mint már emlitettük, a standard file-ok olvasása során a leütött billentyűhöz rendelt karakter ASCII-kódját kapjuk meg, ha van neki olyan. Azonban a speciális billentyűk nem adhatnak ASCII-kódot, igy a funkcióbillentyűk sem. Ilyen esetben a rendszer először egy nulla kódot ad be, majd a leütött billentyű scan-kódját (lásd a "Scan-kódok" függeléket) - a program erről ismerheti fel, hogy a felhasználó egy funkcióbillentyűt ütött le. Természetesen ilyen formában kell megadnunk a hozzárendelési parancsban a specifikálni kivánt billentyű kódját.

A fenti program, némi további ismerettel kiegészitve (melyekről a "Hagyományos file-kezelés" vagy az "MS-DOS file-kezelés" fejezetben olvashatunk) "késztermékké" fejleszthető. Talán úgy kell megfogalmazni, hogy olvasson be valamilyen szabványos szovegfile-t, melyet egy szövegszerkesztő programmal tarthatunk karban, és az egyes funkcióbillentyűkhöz rendelje hozzá az e file-ban megszabott parancsokat. Ezután editorral létrehozzuk a kivánt szövegfile-t, és tetszésünk szerinti parancssorozatokat aktivizálhatunk egyetlen funkcióbillentyű leütésével. Ezt a programot a negyedik kötetben olvashatjuk. Addig is érdemes vele önállóan megverekedni; hasznos és tanulságos feladat.

Fel kell azonban hívnom a figyelmet az ilyenfajta programok egyik hátrányára is. Minden olyan program, amely a DOS-funkciók segitségével olvas (például a programszövegek editálására elsőrangú PE), az általunk megszabott karaktersorozatot fogja olvasni, ha egy billentyűt átdefiniálunk. A kényelmes felhasználáshoz kell egy olyan program is, amely visszaállitja az eredeti értéket, hogy (ha kell) eredeti értelemben is használhassuk a funkcióbillentyűket.

VII. Hagyományos file-kezelés

Hagyományos file-kezelésen az MS-DOS "ásatag leleteiként" megmaradt, a régi CP/M operációs rendszer szabványai szerint működő file-kezelő funkciókat értjük, így különböztetve meg őket a "modern", azaz az MS-DOS valódi file-kezeléséhez idomuló funkcióktól.

VII.1. A hagyományos file-kezelés alapjai

A hagyományos file-kezelés során CP/M-szerűen használhatjuk a file-okat (bár a DOS file-rendszere lényegesen különbözik a CP/M-étől - rugalmasabb és hatékonyabb). Minden file kezeléséhez egy File Control Block-ot kell létrehoznunk (melyet a továbbiakban FCB-nek nevezünk, lásd a "File Control Block" fejezetben), és ki kell jelölnünk egy (elegendően nagy) Disk Transfer Area-t (a továbbiakban DTA). Az FCB leírja a file kezeléséhez szükséges paramétereket. Első lépésként kitöltjük az FCB elemeit, majd megnyitjuk a file-t. Minden további file-művelet elvégzéséhez is meg kell adnunk az FCB címét. Természetesen ezalatt csak néhány FCB-paramétert módosíthatunk. Végül a file-t le kell zárni, ezután az FCB-t "el lehet dobni".

Itt kell megemlítenünk azt, hogy az FCB-ben lényegében csak a file nevét és típusát kell megadnunk. Bizonyos esetekben (vö. 11, 12 és 13 funkció) a file-névnek nem kell egyértelműnek lenni, azaz tartalmazahat wildcard karaktereket is ("*" és "?").

Nemlétező file-ok megnyitása nem lehetséges. Egy külön DOS funkció segítségével kell létrehozni a file-t, amely ezután meg is van nyitva - azonnal írhatunk bele.

A file-ok írása és olvasása szekvenciálisan és direkt eléréssel is lehetséges. Egyszeri művelet során általában egyetlen "rekord"-ot írunk vagy olvasunk. A rekordot azért írtuk idézőjelben, mert nincs köze a más rendszerekből ismert valódi rekord fogalmához, amely valamilyen logikailag különálló egység, amelyet fizikailag is egységként kezelünk és amelyet utólag fel is lehet ismerni a file-ban. A rekord itt mindössze byte-ok folytonos csoportja; hossza a file megnyitása után az FCB-ben tetszőlegesen adható meg, de menet közben már nem módosítható. A kész file-ban többé nem lehet felismerni azt, hogy milyen rekordhosszúsággal írtuk ki a karaktereket, mert a file erre néz-

ve semmilyen információt nem tartalmaz. Direkt elérés során a rekordhossz és a kért rekord száma szabja meg, hogy a file melyik byte-jával kezdődik az átvitel.

> Megjegyzés: direkt elérést mondunk az eredeti "random" helyett. Ezt sokan fordítják pongyolán "véletlen" elérésnek. Itt inkább a szó második szótári jelentése, a "tetszőleges" fedi a valódi jelentést. Nem véletlenül érjük el ugyanis a file-t, hanem tetszőlegesen, azt a rekordot, amelyiket akarjuk.

Az átvitel (figyelmen kívül hagyva a rendszer file-kezelésre használt buffereit) a file és a Disk Transfer Area között megy végbe. Beolvasás után tehát a DTA-n találjuk a sikeresen beolvasott adatokat, kiírás előtt pedig ide kell elhelyezni a kiírandókat.

A DTA terület belépéskor automatikusan a PSP terület 80H pozícióján kap helyet, hossza hexadecimális 80 byte (ami decimálisan 128); a felhasználó természetesen tetszőleges memóriapozícióra helyezheti át.

Figyeljünk fel arra, hogy bár tetszőleges lemezt választhatunk a file-kezelés során, kizárólag az adott lemez kurrens directory-jában végezhetünk file-műveleteket.

Általános jellemzője még a hagyományos file-kezelésnek, hogy a visszatéréskor az AL regiszter tartalma tudósít a művelet sikeréről vagy hibájáról. Sikeres művelet esetén AL-ben 0 van, hiba után pedig általában FFH vagy valami 0-tól különböző hibakód (lásd az egyes funkciók leírásánál). A hiba okáról, természetéről első közelítésben (3.00 előtti verziókban) csakis előfordulásának helyéből következtethetünk: pl. ha egy file megnyitása sikertelen, akkor valószínű, hogy nincs ilyen a kurrens directory-ban. A 3.00 és későbbi verziókban sikertelen művelet után ajánlatos az 59 [DOS_GTEXTERR] funkció meghívása; az innen nyert információ bőségesen tájékoztat a hiba valódi okáról. Ez a szolgáltatás nagyon fontos a hagyományos file-kezelésben.

A hagyományos file-kezelésnek négy "fő csapásiránya" van. Az első az, amit file-kezelés alatt leginkább értünk: file-ok létrehozása, tartalmuk írása-olvasása. A második: létező file-ok törlése a kurrens directory-ból. A harmadik: file-ok keresése, a directory végigvizsgálása valamilyen szempontból. Végül a negyedik: létező file-ok nevének megváltoztatása. Eszerint csoportosítottuk a funkciókat is, így alakult ki négy plusz egy

csoport. Ezek közül négy megfelel a négy fő iránynak, míg az ötödik az információt szolgáltató funkciók csoportja.

Meg kell jegyeznünk azt, hogy a hagyományos file-kezelő funkciók segítségével nem tudjuk igazán kihasználni az MS-DOS többszintű directory-struktúráit és file-kezelésének rugalmasságát. Hacsak nincs nyomós okunk, akkor ne használjuk ezeket a funkciókat. Vállaljuk a nagyobb szabadság, a kényelmesebb programozás kedvéért a nagyobb felelősséget, amely az MS-DOS file-kezelési funkciók kihasználásával jár. A két funkciócsoport tagjai közötti megfeleltetést "A hagyományos és modern funkciók kapcsolata" függelék tartalmazza.

Egyetlen nagy előnye a hagyományos file-kezelő funkcióknak a modernekkel szemben, hogy bizonyos esetekben nem korlátozott az egyidejűleg megnyitva tartható file-ok száma, hiszen a kezelésükhöz szükséges adatstruktúrát (FCB) mi magunk biztosítjuk.

Az MS-DOS 3.00 előtti verzióiban nincs korlátozva az FCB-vel egyidejűleg megnyitható file-ok száma. Amennyiben a 3.00 és későbbi verziók bármelyikét a SHARE driver nélkül használjuk, nincs semmilyen korlát a file-ok számára nézve.

Ha a SHARE paranccsal betöltjük a megosztott file-elérést vezérlő drivert, akkor a CONFIG.SYS-ben megadott

```
FCBS=m,n
```

paranccsal szabhatjuk meg az FCB felhasználásával egyidejűleg megnyitható file-ok számát. Itt "m" jelenti az egyszerre megnyitva tartható file-ok számát (1 és 255 között), az "n" pedig azt adja meg, hogy ezek közül hányat nem zárhat le a rendszer "m" átlépése esetén (0 és 255 között). Alapértelmezések: az "m"-re 4, az "n"-re 0.

Ha egy program "m" file-t már megnyitott, és megkísérli egy újabb file megnyitását, akkor a rendszer automatikusan lezárja a legrégebben felhasznált FCB-t. Ha egy automatikusan lezárt file-t próbálunk meg írni vagy olvasni, akkor az 59 funkció segítségével a 23 hibakódot kapjuk meg (melynek jelentése: nem létező FCB).

Amennyiben megszabjuk az "n" paramétert, akkor a megnyitási sorrendben első "n" file lezárása tiltott, tehát a legfontosabb file-jainkat megóvhatjuk az (egyébként kissé anarchikus és igen nehezen ellenőrizhető) automatikus lezárástól.

VII.2. A hagyományos file-kezelési funkciók

VII.2.1. A szokásos file-kezelésre szolgáló funkciók

0D Disk Reset - DOS_DSKRES

A lemezkezelő rendszer alapállapotba hozása - kiönt minden buffert a file-okba. A módosított file-ok directory-bejegyzéseit nem módosítja a rendszer; a nyitva hagyott file-ok nem lesznek hibátlanok.

Megjegyzés: az írt file-ok explicit lezárása mindenképpen kötelező. Ha egy file írása közben a felhasználó floppyt cserél, akkor a rendszer nem fog ugyan írni az újonnan berakott diskette-re, de a régin a file természetesen nem lesz sértetlen. Ezt a hibát persze programból nem tudjuk lekezelni (gonoszság ellen nem lehet védekezni), de mielőtt utasítást adnánk a felhasználónak arra, hogy cseréljen lemezt, mindenképpen zárjuk le a file-okat.

0E Select Disk - DOS_DSKSEL

Aktív lemezegység kiválasztása - a DL-ben megadott kódú (0=A:, 1=B: stb.) lesz a kurrens lemez a hívás után. A logikai lemezegységek számát AL-ben kapjuk vissza (logikai lemezegység alatt valóban létező lemezt, létező egységre átirányított lemezt vagy pedig RAM-disket értünk). Jegyezzük meg, hogy itt másképpen kell a lemezeket kódolni, mint az FCB esetén!

Megjegyzés: a visszakapott érték a lemezegységek maximális száma, melyet egyébként a CONFIG.SYS file-ban a

LASTDRIVE=x

paranccsal módosíthatunk, ahol "x" az utolsó még legális lemezkódot jelenti (alapértelmezése E). Miért ez az érték az alapértelmezés? Először is, ha csak egy floppy-egységünk (A:) van, akkor sem illegális a B: egységre való hivatkozás: ekkor B: helyett is az A:-t fogjuk használni. Ezenkívül éppen úgy elérhetjük a RAM-disket, mint bármely fizikai lemezt. Az öt logikai lemezegység: két floppy (A: és B:), két Winchester (C: és esetleg D:) és egy RAM-disk (D: vagy E:).

Fontos megjegyezni azt is, hogy a rendszer nem figyelmeztet bennünket a sikertelen lemezválasztásra. Ha például tévesen a nemlétező 17. lemezre hivatkozunk, akkor nem történik semmi, az a lemez marad a kiválasztott, amelyik eddig az volt.

0F Open File - DOS_FOPEN

File-megnyitás - DS:DX mutat egy (meg nem nyitott) file FCB-jére, melyet szabályosan kitöltöttünk. A rendszer a kurrens directory-ban keresi a megadott file-t. Ha nem találja, a válasz AL-ben FFH. Ha megtalálta és a file megnyitása sikeres, a válasz 0, és az FCB-ben kitöltődnek a rendszer által kezelt paraméterek (például a file hossza, keletkezésének dátuma stb.).
Rejtett vagy rendszerfile-ok megnyitása ezzel a funkcióval nem lehetséges!
A rekordhosszt a rendszer az alapértelmezett 128-ra állítja be. Ha ez nem felel meg, akkor a kívánt rekordhosszt a file megnyitása és az első írás vagy olvasás között kell megadnunk (0E-0F offset).
Direkt file-műveletek előtt ki kell tölteni az FCB Random Record számlálóját.
MS-DOS 3.00 és későbbi verziók
Szekvenciális műveletek előtt ki kell tölteni az FCB decimális 32 (20H) offsetjén található rekordszámlálót.
MS-DOS 3.00 és későbbi verziók
Az összes FCB-vel megnyitott file-ok elérési módja: írás-olvasás; a file-ok öröklődnek, és megosztott elérési módjuk "kompatibilis" mód (vö. 3D - DOS_OPEN).

16 Create File - DOS_FCREAT

File létrehozása - DS:DX egy meg nem nyitott file FCB-jére mutat. Az FCB-ben adott paraméterek szerint létrehoz egy új file-t a lemezen. Ha a file létezik, akkor tartalma elvész. Ha nincs, akkor a DOS egy üres helyet keres a directory-ban. Ha talált ilyet, létrehozza a file-t, és AL-ben 0-t ad vissza, egyébként FF-t.
Kiterjesztett FCB segítségével írásvédett file-t is létrehozhatunk, de ebben nincs sok köszönet, hiszen a file rögtön megkapja ezt az attributumot, és semmit sem írhatunk bele. Ilyen file-okat az MS-DOS file-kezelés megfelelő funkcióival hozhatunk létre.

Hasonlóképpen, kiterjesztett FCB használatával hozhatunk létre lemezcimkét. Ha a lemezen van már cimke, a kísérlet sikertelen lesz. érdekesség még, hogy a lemezcimke mindig a ROOT directory-ban jön létre attól függetlenül, hogy melyik aldirectory-ban voltunk addig.

1A Set Disk Transfer Area - DOS_SETDTA
Disk transzfer terület átcímzése - az új címet a DS:DX szegmens- és címzőregiszterben adjuk meg.
Az alapértelmezett DTA a PSP hexadecimális 80 byte-ján kezdődik, hossza 128 byte. A DTA lekérdezése a 2F funkcióval lehetséges.

10 Close File - DOS_FCLOSE
File-lezárás - lezárja a file-t, a rendszer buffereiben levő utolsó rekordokat kiírja. Ha a file-t írtuk, akkor kötelesek vagyunk lezárni; csak olvasott file-ok lezárása nem kötelező (csak melegen ajánlott — a file-kezelés elég komoly feladat ahhoz, hogy ne bízzuk az eredményt a véletlenre). Belépéskor DS:DX címezi meg a megnyitott file FCB-jét. Hiba esetén AL-ben hexadecimális FF-t kapunk vissza; sikeres lezárás után pedig 0-t.

14 Read Sequential - DOS_FRDSEQ
Soros (szekvenciális) olvasás - DS:DX egy megnyitott file FCB-jére mutat. A rendszer az FCB-ben adott paraméterek szerint felolvassa a következő rekordot.
Ha elértük a file végét, és az olvasás már egy byte-ot sem tudott behozni, akkor a visszatérési kód 01. Ha a rekord beolvasása közben értük el a file végét, és egy "tört" rekordot még sikerült beolvasni, akkor AL-ben 03-at kapunk vissza. A visszatérési kód 02, ha a DTA nem elég hosszú a beolvasáshoz, és az olvasás emiatt befejeződött.

> Megjegyzés: a 02 hibakód előállítására irányuló erőfeszítéseim rövid történetét lásd a 27 funkciónál.

Sikeres olvasás esetén a válasz az AL regiszterben 0, és a rendszer megnöveli a rekordszámlálót.

15 Write Sequential - DOS_FWRSEQ
Soros (szekvenciális) írás - DS:DX egy megnyitott file FCB-jére mutat. A funkció kiírja a DTA-ról a soron következő rekordot.
Ha a lemez tele van, AL-ben 1-et, ha rendszerterületen nincs elég hely az átvitel számára, 2-t kapunk vissza. Sikeres kiírás esetén AL=0, és a rendszer megnöveli a rekordszámlálót.

24 Set Relative Record - DOS_SETRCD
A Random Record szám beállítása - DS:DX egy megnyitott file FCB-jére mutat. Ez a funkció úgy állítja be az FCB Random Record számát, hogy egy direkt művelettel (21, 22, 27 vagy 28 funkciók) ott folytathassuk a file írását vagy olvasását, ahol az addig elvégzett szekvenciális műveletekkel éppen tartottunk.

21 Read Random - DOS_FRDRND
Direkt olvasás - DS:DX egy megnyitott file FCB-jére mutat. Az FCB-ben adott paraméterek szerint direkt olvasást hajt végre. Hibakódok, mint a direkt olvasásnál, lásd 14 funkció.

22 Write Random - DOS_FWRRND
Direkt írás - DS:DX egy megnyitott file FCB-jére mutat. Az FCB-ben adott paraméterek szerint direkt kiírást hajt végre a file-ra. Hibakódok, mint a soros írásnál, lásd 15 funkció.

27 Random Block Read - DOS_FRDBRND
Direkt blokkolvasás - DS:DX egy megnyitott file FCB-jére mutat. Az FCB-ben adott paraméterek szerint olvas egy blokkot a file-ból a kurrens DTA-ra. A beolvasandó blokk hosszát CX tartalmazza rekordokban kifejezve. Hibakódok pontosan, mint a soros file-olvasásnál, lásd 14 funkció.
Ha CX=0, akkor egy rekordot sem olvas be.
Visszatérés után CX tartalmazza a valóban beolvasott rekordok számát.

Megjegyzés: a 02 hibakód teljesen ködösnek tűnik, mert a rendszer nem lehet tisztában a DTA hosszával, hacsak ... nem allokáltunk külön memóriát a

DTA számára, ahol legalább a sérült blokkhatárolók utalhatnak a DTA terület túlírására.
Elsőként azt kíséreltem meg a 02 előállításának érdekében, hogy lényegesen megnöveltem a rekordhosszt. Semmi eredmény - a rendszer szó nélkül túlírta a DTA területet a program kódjával együtt (a PSP-ben levő 128 byte hosszú DTA-t is, ami több mint kínos). Az olvasás "sikeres" volt, AL-ben 0-t találtam. A következő lépésben blokkolvasással próbálkoztam. Az eredmény ugyanaz. Egy újabb kísérlet, most már külön a DTA számára allokált memóriával (lásd "Memóriakezelés és programvezérlés" fejezet): először DOS_SETBLK-al átrendeztem a memóriát, majd DOS_MALLOC-al lefoglaltam pár tucat paragrafust, és a kapott területet beállítottam mint a DTA címét. Ezután megpróbáltam olvasni egy, a lefoglalt memóriát lényegesen meghaladó hosszúságú blokkot. Az olvasás ismét sikeres!
Végül feldühödtem, és a blokkszámlálót akkorára állítottam, hogy a beolvasandó blokk hossza meghaladta a 64 Kb-ot. Ekkor végre megkaptam az olyannyira áhított 02 hibakódot, de a programból kilépni már nem tudtam - a DOS teljesen lepusztult, és újra kellett tölteni. Mindenesetre a kritikus hívás után meghívtam még az 59 funkciót (ez csaknem mindig lehetséges), amellyel beolvastam a kiterjesztett hibakódot ("Kisegítő MS-DOS funkciók" fejezet). A visszaadott kód memóriahibára utalt (ami mélységesen igaz), a javasolt akció pedig:

Terminate with cleanup

azaz "takarits és takarodj" - ez valóban egy megszívlelendő tanács volt!
Mielőtt befejezném ezt a kisded Odüsszeiát, szeretném még egy apró "finomságra" felhívni az érdeklődő Olvasó figyelmét. Mikor COM file-t hozunk létre, akkor (természetesen) nincs külön STACK szegmensünk. A Stack Pointert a rendszer úgy állítja be, hogy az egy sem programok, sem adatok által le nem foglalt 256 byte hosszú területre mutasson, minden felhasznált területnél magasabb pozicióra. Ha a program jó hosszú, akkor kezdeti

értéke tapasztalataim szerint FEE6 (hogy miért? "...ne kutassa kíváncsi szó"). Mikor átrendezzük a memóriát, akkor persze a Stack Pointer nem változik, így veremterületünk már kívül esik a program memóriablokkjának határain! Ha pedig ezután memóriát foglalunk, sőt, a lefoglalt memóriát használjuk is, akkor elég hamar elpusztítjuk a vermet. Ezt a tragédiát elkerülendő, célszerű COM file-ban is néhány (de legalább 128) byte-ot lefoglalni a verem számára, és utasítással előkészíteni a Stack Pointert. Akkor ilyen csúnyaság nem esik meg, de nagyon ügyelni kell arra, hogy túl ne írjuk a vermet... Summa summarum: ne írjunk COM programot, vagy ha mégis, ne allokáljunk benne memóriát!

28 Random Block Write - DOS_FWRBRND
Direkt blokkírás - DS:DX egy megnyitott file FCB-jére mutat, CX szabja meg, hány rekordot kell kiírni a file-ba. Ha CX nem 0, akkor a rendszer az FCB-ben adott paraméterek szerint kiír CX számú blokkot a file-ba a kurrens DTA-ról. Ha a CX tartalma 0, akkor a rendszer átmásolja az FCB Random Record számát a directory-bejegyzés file-hossz mezőjébe, azaz a file hosszát módosítja. Ennek során persze felszabadítja a feleslegessé vált clustereket, vagy annyi clustert allokál, amennyi a file "meghosszabbításához" szükséges. Hibaüzenetek, mint soros írásnál, lásd 15 funkció. A 02 hibakód itt azt jelenti, hogy a kiírandó blokk túllépné a DTA szegmensének határait.
Visszatérés után CX tartalmazza a valóban kiírt rekordok számát.

Megjegyzés: a blokkműveletek több rekord egyidejű mozgatására szolgálnak.
Bármely írási vagy olvasási művelet esetén ügyelnünk kell arra, hogy a DTA elég hosszú legyen a kívánt számú byte befogadására, mert a rendszer erre nézve semmi ellenőrzést nem végez. Beolvasás esetén kegyetlenül felülírja a DTA terület mögötti területet, kiírásnál pedig a DTA utáni memória tartalmát is kiviszi a file-ra.

VII.2.2. Keresés a kurrens directory-ban

11 Search for First Entry - DOS_FSFRST

Az első megfelelő bejegyzés keresése - a DS:DX-ben megadott című FCB által megadott első megfelelő file-t keresi a kurrens directory-ban (ennek elsősorban nem egyértelmű file-definició esetén van értelme). Ha nem talált megfelelő bejegyzést, akkor az AL-ben FF-el tér vissza. Ha talált, akkor AL-ben 0 a válasz.
A rendszer a "Disk Transfer Area"-n hoz létre egy olyan FCB-t, amely a megtalált file adatai szerint van kitöltve és alkalmas a file kezelésére (lásd a következő bekezdést).

Megjegyzés: fontos tudnunk, hogy a DTA-n nem egyszerűen egy FCB-t, hanem egy teljes directory-bejegyzést (lásd "Directory-bejegyzés" fejezet) kapunk - a rendszer bemásolja a megtalált directory-bejegyzést, melyből a file-ról szinte mindent megtudhatunk. A directory-bejegyzés és az FCB gondos tanulmányozása megmutatja, hogy ha az előbbit egy byte-al eltoljuk (azaz ha elé írunk egy drive-kódot és a drive-kód címére hivatkozunk), akkor ez pontosan megfelel egy FCB-nek is, mert abban megnyitás előtt kötelezően csak a lemez kódját, a file nevét és típusát kell kitölteni, a további byte-ok tartalma tetszőleges lehet.

Ha a kereséshez kiterjesztett FCB-t használunk (lásd "File Control Block" fejezet), akkor annak attributum-byte-jában megadott értékkel a következőképpen módosíthatjuk a keresési stratégiát (ez esetben a kiterjesztés kezdőcímét kell DX-be tölteni!).
Ha az attributum

00	csak a normál (archív) file-okra keresünk; directory-file-okat, lemezcimkét, rejtett és rendszerfile-okat nem keres meg a DOS.
02,04,10	a keresés a normál file-ok mellett a megnevezett attributumú file-okra is vonatkozik: rejtett-, rendszer- és directory-file-ok. E biteket kombinálva is használhatjuk.
08	a keresés kizárólag a lemezcimkére vonatkozik (amelyből egy lemezen csak egy lehet).

Kiterjesztett FCB használata esetén a DTA területen elsőként a kiterjesztésünk másolatát fogjuk megtalálni, melynek hossza 7 byte, és csak ezután következik az FCB, amelynek második byte-jától kezdődik a megfelelő directory-bejegyzés másolata. Tehát elsőként egy FF-et tartalmazó byte-ot kapunk, melyet öt, 0 tartalmú byte követ. Ezután találjuk az általunk specifikált attributumot, majd a kurrens lemez kódját, amelyet a beolvasott directory-bejegyzés követ.

12 Search for Next Entry - DOS_FSNEXT

A következő megfelelő bejegyzés keresése - ha az előző funkcióval már megtaláltuk az első megfelelő bejegyzést, ezzel kereshetjük a következőt. Ekkor persze a 11 funkciónak korábban átadott FCB címét kell DS:DX-be tölteni.

Ha a 12 funkció talált új bejegyzést, AL-ben 0-t ad vissza, ha pedig nem, akkor FF-t.

A megtalált bejegyzést ugyanúgy a DTA-n adja vissza, mint a 11 funkció.

Példa: e funkciók a kurrens Disk Transfer Address területen adják vissza a megtalált bejegyzés valódi értékét; azaz, ha a keresett bejegyzés a

```
pro??.*
```

volt, és a C: lemez kurrens directory-jában szereplő, az általunk megadott határozatlan file-specifikációnak megfelelő file-ok rendre a

```
proba.asm
proba.bak
proba.obj
procc.txt
```

nevet viselik, a 11 [DOS_FSFRST] hívás után a

```
<3>proba   asm<20H>
```

az első 12 [DOS_FSNEXT] hívás után a

```
<3>proba   bak<20H>
```

azután

```
<3>proba   obj<20H>
```

végül a

```
<3>procc   txt<20H>
```

byte-okat találjuk a DTA-n, ahol <i> egyetlen, a megadott számértéket tartalmazó byte-ot jelent. A

<3> a lemez kódja (C:), a <20H> pedig a file attributuma (lásd "Directory-bejegyzés"). A következő hívás már FF-et ad vissza AL-ben. A név és a típus között természetesen ott találjuk a három-három szóközt. Az itt talált első 12 byte egyszerűen bemásolható egy FCB-be, s azután sorra meg is nyithatjuk a file-okat, végül pedig az utolsó file megtalálása után a 12H hívás AL-ben FFH-t ad vissza.
Természetesen a DTA területet magát is felhasználhatjuk mint FCB-t, de akkor a következő keresés elrontja FCB-nket (és ha még nem zártuk le a file-t, akkor... magunkra vessünk).

VII.2.3. File-ok törlése

13 Delete File - DOS_FDELET
File törlése — DS:DX egy olyan FCB-re mutat, amellyel nem nyitottunk meg file-t (tartalmazhat határozatlan file-nevet is, azaz "*" és "?" karaktereket). A rendszer minden, az FCB-ben megadott névnek és típusnak megfelelő bejegyzést töröl (ha a file-név nem egyértelmű). Ha nem talált megfelelő bejegyzést, AL-ben FF-t, ha törölt legalább egy file-t, 0-t ad vissza.

VII.2.4. File-ok átnevezése

17 Rename File - DOS_FRENAME
File átnevezése - DS:DX egy speciálisan kitöltött FCB-re mutat, amelyben az első (a régi) file-név végétől 6 byte-ra van az új file-név (DS:DX+11 hexa). A directory minden megfelelő elemét átnevezi az új névre, melyben lehetnek "?" karakterek - az eredeti file-név azon a pozíción változatlan marad.
AL-ben FF-t kapunk, ha nem talált "régi" nevű file-t, ha pedig talált, akkor 0-t.
Nem használható rendszer- és rejtett file-ok átnevezésére, sem pedig directory-k nevének megváltoztatására.

VII.2.5. Információs funkciók

29 Parse Filename - DOS_FNMINI

Parancssor feldolgozása, file-név előkészítése - belépéskor DS:SI mutat a parancssorra, ES:DI pedig arra a területre, ahol létre kell hozni egy FCB-t. Ebből talán már kiviláglik, mit is csinál ez a funkció: megvizsgálja, hogy a parancssor értelmezhető-e mint egy file-név, amelynek formátuma:

dr:file-név.típus

és ha igen, akkor létrehoz belőle egy szabályos FCB-t. AL alsó 4 bitje a következőt szabja meg:

0. bit: 0 átlépi a bevezető elválasztójeleket. Ezek: : . ; , = " [ ] < > / \ |, valamint szóköz és tabulátor;
1 az első bevezető elválasztójelre beszünteti a parancssor feldolgozását.

1. bit: 0 0-val tölti az FCB-ben a lemez kódját, ha a parancssorban nincs megadva;
1 nem változtatja meg az FCB-ben a lemez kódját, ha a parancssorban nincs megadva.

2. bit: 0 nyolc szóközzel írja felül a file-nevet az FCB-ben, ha a parancssorban nincs megadva;
1 nem változtatja meg a file-nevet az FCB-ben, ha a parancssorban nincs megadva.

3. bit: 0 három szóközzel írja felül a file-típust az FCB-ben, ha a parancssorban nincs megadva;
1 nem változtatja meg a file-típust az FCB-ben, ha a parancssorban nincs megadva.

Ha a parancssor értelmetlen (azaz nem tartalmaz filenevet), AL-ben FFH-t kapunk vissza.
Ha a file-név "*"-t vagy "?"-t tartalmaz, AL-ben 1-et kapunk vissza, egyébként pedig 0-t.
DS:SI a parancssor utolsó feldolgozott karaktere mögé mutat, így könnyen folytathatjuk a parancssor feldolgozását. ES:DI a létrehozott FCB-t címzi.

19 Get Current Disk - DOS_GETDSK

Kurrens lemez lekérdezése - AL-ben visszaadja a kurrens disk kódját (0=A:, 1=B: stb.). Jegyezzük meg, hogy itt a lemezegységek másképpen kódoltak, mint az FCB esetén!

1B Get Default Drive Data - DOS_GETDFAT
Információ a kurrens lemez allokációs táblájáról - a visszatérés után DS:BX megcímez egy byte-ot, amely a kurrens lemez FAT-azonosítóját tartalmazza (lásd a File Allocation Table fejezetet), DX-ben van az allokációs egységek ("clusterek") száma, AL a szektorok számát tartalmazza allokációs egységenként, CX pedig a szektorok fizikai hosszát. (A sáv, a szektor és az allokációs egység leírását lásd "A lemez fizikai felépítése" fejezetben.)

1C Get Drive Data - DOS_GETSFAT
Információ a DL-ben specifikált lemez allokációs táblájáról (0=kurrens lemez, 1=A: stb.) - pontosan ugyanazt kapjuk meg, mint 1B esetén a kurrens lemezről.

36 Get Disk Free Space - DOS_GETDFREE
Szabad lemezterület lekérdezése a DL-ben adott kódú (0=kurrens lemez, 1=A: stb.) lemezről - AX FFFF lesz, ha a lemez kódja érvénytelen volt. Egyébként BX a szabad clusterek számát tartalmazza, DX az összes cluster számát, AX a szektorok számát clusterenként, és CX a byte-ok számát szektoronként.

23 Get File Size - DOS_GETFSIZ
File-hosszúság lekérdezése - DS:DX egy meg nem nyitott file FCB-jére mutat, a file-névnek egyértelműnek kell lennie, nem tartalmazhat "*" és "?" karaktereket. Az FCB-ben megadott paraméterek szerint keres egy megfelelő directory-bejegyzést, és ha megtalálta, a "random record" field-be írja a file hosszát rekordokban, az AL-ben pedig 0-t ad vissza. Ha nem talál megfelelő directory-bejegyzést, AL-ben FFH-val felel. A hívás előtt feltétlenül ki kell tölteni a rekord hosszát! Ez a funkció tehát arra szolgál, hogy megtudjuk: hány adott hosszúságú rekordot tartalmaz a file. Csak akkor adja vissza a file valódi hosszát, ha a rekordhosszat 1-re állítjuk.

2F Get Disk Transfer Address - DOS_GETDTA
A DTA, a Disk Transfer Address lekérdezése - visszatéréskor ES:BX tartalmazza a címet.

VII.3. Példaprogramok a hagyományos file-kezeléshez

Az alábbiakban olyan példaprogramok következnek, amelyek a fenti funkciók használatát illusztrálják. Mivel minden egyes file-kezelést végző programban kénytelenek vagyunk egy (vagy néha több) FCB-t, valamint számos konstanst deklarálni, érdemes ezeket egy külön file-ban egyszer s mindenkorra megadni, és minden egyes programba fordítási időben "beemelni", hogy ezzel helyet és időt takaríthassunk meg. Ezzel egyúttal igen fontos gyakorlati módszerre látunk példát. Nagyon lényeges dolog egy ilyen include file létrehozása, mégpedig ismét a "minden adatnak pontosan egy forrása legyen" elv betartása kedvéért. Ha egyszer megírjuk (és persze gondosan karbantartjuk) include file-jainkat, sokkal ritkábban fogjuk eltéveszteni például egy DOS-funkció kódját, nem hagyunk ki elemet az FCB-k definíciójából stb. Tehát a makró-, struktúra- és egyéb deklarációk, include.file-ok fő haszna nem a hely- és munkamegtakarítás (bár ez is nagyon fontos szempont), hanem a programozás nagyobb biztonsága.

Lássunk akkor példát egy olyan include file-ra, amely a hagyományos file-kezelésben hasznos konstans-, makró- és struktúradefiníciók gyűjteménye!

VII.3.1. Deklarációs file

```
COMMENT *
Ez az include file a hagyományos file-kezeléshez szükséges leg-
fontosabb konstans-, makró-, struktúra- és rekorddefiníciókat
tartalmazza.
Felhasználása:
        INCLUDE         FCB.INC
a forráskód elején.
*

; Konstansok
;
; Karakterkonstansok
;

CHR_ESC         EQU     27      ;ESCAPE
CHR_CR          EQU     13      ;Kocsivissza
CHR_LF          EQU     10      ;Soremelés
```

```
;
; MS-DOS konstansok
;

        IFNDEF  DOS_INT
DOS_INT         EQU     21H
        ENDIF

;
; MS-DOS funkciót hívó makró
;

DOSCALL MACRO   CODE
                MOV     AH, CODE
                INT     DOS_INT
        ENDM

;
; Standard file-kezelési konstansok
;

DOS_STRDCHE     EQU     01H     ;Karakter be STDIN-ről + echo
DOS_STWRCH      EQU     02H     ;Karakter kiír. STDOUT-ra
DOS_STRDCH      EQU     08H     ;Beolvasás echo nélkül
DOS_STDIRIO     EQU     06H     ;Direkt input/output
        DOS_STDIRIO_IN  EQU     0FFH    ;Direkt I/O - input
DOS_STDIRIN     EQU     07H     ;Direkt beolvasás

DOS_STWRSTR     EQU     09H     ;String kiírása STDOUT-ra
DOS_STRDSTR     EQU     0AH     ;String beolvasása STDIN-ről

DOS_STINCHK     EQU     0BH     ;Státusz lekérdezése
DOS_STINFLS     EQU     0CH     ;Buffer törlése

;
; Standard file-kezelő makrók
;

PRTSTR  MACRO   STRING
                MOV     DX, OFFSET STRING       ;;String címe
                DOSCALL DOS_STWRSTR             ;;DOS-hívás
        ENDM                                    ;;
```

```
TRCMSG  MACRO   STRING
        IFDEF   TRACE                   ;;Csak ha külön kérik
                PUSH    DX
                PUSH    AX              ;;Regisztermentés
                MOV     DX, OFFSET STRING
                DOSCALL DOS_STWRSTR
                POP     AX
                POP     DX              ;;Regiszterek vissza
        ENDIF
        ENDM

;
; Makró üzenetek definiálására
;
NOCR            EQU     1       ;Kocsivezérlő konstans

MSGDEF  MACRO   NAME, MESSAGE, CRCONT
NAME            DB      MESSAGE
        IFNDEF   CRCONT         ;;Nem biztos, hogy kell CR-LF,
                DB      CHR_CR, CHR_LF
        ENDIF
                DB      '$'     ;; de dollárjel biztosan kell
        ENDM                    ;;

;
; FCB file-kezelési konstansok
;
DOS_DSKRES      EQU     0DH     ;Disk inicializálása
DOS_DSKSEL      EQU     0EH     ;Disk kiválasztása
DOS_FOPEN       EQU     0FH     ;File megnyitása
DOS_FCLOSE      EQU     10H     ;File lezárása
DOS_FSFRST      EQU     11H     ;Első bejegyzés keresése
DOS_FSNEXT      EQU     12H     ;További bejegyzések keresése
DOS_FDELET      EQU     13H     ;File törlése
DOS_FRDSEQ      EQU     14H     ;Szekvenciális olvasás
DOS_FWRSEQ      EQU     15H     ;Szekvenciális írás
DOS_FCREAT      EQU     16H     ;File létrehozása
DOS_FRENAME     EQU     17H     ;File átnevezése
DOS_GETDSK      EQU     19H     ;Kurrens lemez lekérdezése
DOS_SETDTA      EQU     1AH     ;DTA címének beállítása
DOS_GETDFAT     EQU     1BH     ;FAT inf. kurrens lemezről
DOS_GETSFAT     EQU     1CH     ;FAT inf. adott lemezről
```

```
DOS_FRDRND      EQU     21H     ;Direkt file-olvasás
DOS_FWRRND      EQU     22H     ;Direkt file-írás
DOS_GETFSIZ     EQU     23H     ;File-hossz lekérdezése
DOS_SETRCD      EQU     24H     ;Random rekordszám beállítása
DOS_FRDBRND     EQU     27H     ;Direkt blokkolvasás
DOS_FWRBRND     EQU     28H     ;Direkt blokkírás
DOS_FNMINI      EQU     29H     ;File-név feldolgozása
        DOS_FNMINI_SEP  EQU     01H     ;Elválasztás átlépése
        DOS_FNMINI_DR   EQU     02H     ;Lemeznév feldolgozása
        DOS_FNMINI_NAM  EQU     04H     ;Név feldolgozása
        DOS_FNMINI_TIP  EQU     08H     ;Típus feldolgozása
DOS_GETDTA      EQU     2FH     ;DTA címének lekérdezése
DOS_GETDFREE    EQU     36H     ;Szabad terület lekérdezése

;
; Hibakódok
;
RET_SUCC        EQU      00H    ;Sikeres végrehajtás
RET_NOBYTE      EQU      01H    ;File-vég, nincs byte
RET_NOSPACE     EQU      02H    ;Nincs hely az átvitelre
RET_FRAGM       EQU      03H    ;Tört rekord beolvasása
RET_UNSUCC      EQU     0FFH    ;Sikertelen végrehajtás

;
; Drive-kódok
;
; FCB-beli megadáskor használatos drive-kódok
;
DR_CURR         EQU     0       ;Kurrens drive
DR_A            EQU     1       ;A: drive
DR_B            EQU     2       ;B: drive
DR_C            EQU     3       ;C: drive
DR_D            EQU     4       ;D: drive
DR_E            EQU     5       ;E: drive
;
; Lekérdezéskor kapott drive-kódok
;
DR_A_RESP       EQU     0       ;A: drive
DR_B_RESP       EQU     1       ;B: drive
DR_C_RESP       EQU     2       ;C: drive
DR_D_RESP       EQU     3       ;D: drive
DR_E_RESP       EQU     4       ;E: drive
```

```
;
; FCB-s makrók, struktúrák és rekordok
;
FCBCLL  MACRO   CODE, FCB
                MOV     DX, OFFSET FCB
                DOSCALL CODE
                TEST    AL, AL
        ENDM

FCB     STRUC
        DRIVE   DB      0               ;;Drive kódja
        NAME    DB      '               ;;File-név
        EXT     DB      '               ;;File-típus
        BLOCK   DW      0               ;;Blokkszámláló
        RECSIZ  DW      0               ;;Rekordhossz
        FILSIZ  DD      0               ;;File hossza
        DATE    DW      0               ;;Utolsó írás dátuma
        TIME    DW      0               ;;Utolsó írás ideje
        RES_FCB DB      8 DUP ( ? )     ;;Fenntartott terület
        CURREC  DB      0               ;;Rekordszámláló
        RNDREC  DD      0               ;;Random rekordszámláló
FCB     ENDS                            ;

FCB_EXT STRUC
        EXT_SG  DB      0FFH            ;;Kiterjesztésjelző
        RES_EXT DB      5 DUP ( ? )     ;;Fenntartott terület
        ATTRIB  DB      0               ;;Attributum
FCB_EXT ENDS                            ;
;
; Kiterjesztett FCB definiálására szolgáló makró
;

FCBEXT  MACRO   NAME, FIL_DR, FIL_NM, FIL_EXT, FIL_ATR
NAME&_EXT       FCB_EXT <,,FIL_ATR>
NAME            FCB     < FIL_DR, FIL_NM, FIL_EXT >
        ENDM

;
; Attributum-konstansok
;
ATR_RDONLY      EQU     01H     ;Csak olvasható file
ATR_HIDDEN      EQU     02H     ;Rejtett file
```

```
ATR_SYSTEM      EQU     04H     ;Rendszerfile
ATR_VOLAB       EQU     08H     ;Lemezcimke "file"
ATR_DRTRY       EQU     10H     ;Directory
ATR_ARCHV       EQU     20H     ;Archiv file
;
; A DATE elem belső szerkezete
;
DATE_R  RECORD  YEAR:7, MONTH:4, DAY:5
;
; A TIME elem belső szerkezete
;
TIME_R  RECORD  HOUR:5, MINUTE:6,TWOSEC:5
```

Mielőtt továbbhaladnánk, szeretném az Olvasó figyelmét felhívni néhány nagyon fontos apróságra.

Kezdjük a file elején! A file természetesen egy bevezető megjegyzéssel kezdődik, amely legalább az itt megadott információkat tartalmazza: mi ez a file, mire szolgál, és hogyan lehet használni.

A néhány karakterkonstans csak a legfontosabbakra szorítkozik. Hasonlóképpen csak a legfontosabb standard file-kezelő konstansokat definiáltuk, hogy ez ne vonja el az Olvasó figyelmét a lényegesebb konstansdefiníciókról.

A DOS_INT nevű konstanst azért definiáljuk egy IFNDEF feltétel alatt, hogy ha már egy másik include-ból vagy egyéb helyről ismert lenne, ne kerüljön ellentmondásba a két deklaráció.

Az FCB file-kezelési konstansok most nem logikai, hanem nagyság szerinti sorrendben követik egymást. Ez egy vitatható pont, mivel nem igazán javítja az olvashatóságot, viszont így sokkal könnyebb karbantartani a file-t. Azt az egy apróságot figyeljük meg, hogy az alfunkciók kódját (ahol van ilyen) egy tabulátorpozícióval beljebb kezdtük, ezzel érzékeltetve alárendelt szerepüket.

A következő blokkban az itt lehetséges visszatérési kódok olvashatók. Ez alatt a használatos drive-kódok szerepelnek, melyek külön deklarálása azért célszerű, mert más-más logika érvényesül a beállításkor és lekérdezéskor, illetve az FCB-ben való felhasználáskor.

Most pedig lássuk az érdekesebb kérdéseket! Definiáltuk a már jót ismert DOSCALL és PRTSTR makrókat. Ezek semmiféle meglepetést nem tartogatnak, hacsak azt nem, hogy nem mentünk semmit (visszatérési információ!), és a DOSCALL makróban nem tesz-

teljük AL értékét, hiszen ez elrontaná a Zero flagben adott választ, ha van. Annál érdekesebbnek látszik a TRCMSG makró. Mi szükség van itt a regiszterek mentésére, pontosabban miért van két makró (látszólag) ugyanarra a funkcióra? A nevük mutatja meg a különbséget. Míg a PRTSTR a szabvány string-kiírás, a TRCMSG egy "trace-message"-t, egy ún. nyomkövető üzenetet ír ki. Az előbbit "önmagáért" használjuk (és így a mentés felesleges), az utóbbit viszont bármely két utasítás közé be akarjuk szúrni, tehát nem ronthat el semmit. Ezért van itt PUSH és POP. Erre persze meg lehet kérdezni, hogy mi lesz a visszatérési információval (amely egyébként e hívásnál nem értelmezett, de elmélet is van a világon). A válasz: ha a programunk már a standard output file-ra se tud írni, akkor úgy is régen rossz. Figyeljük meg azt is, hogy a makró belsejét egy feltételes fordítási blokkba foglaltuk. Ez nagyon hasznos lehet, mert a program elején (a file beemelése, include-olása előtt!) vagy definiáljuk a TRACE nevű szimbólumot, vagy nem. Ha definiáljuk, akkor minden TRCMSG makróhívás kiírja a maga üzenetét, ha pedig nem, akkor az üzenetek sem jönnek. Egy fejlesztés közben tartandó bemutató kedvéért nem kell kigyomlálnunk minden TRCMSG makróhívást a programból, hiszen egyetlen sor megváltoztatásával (és persze a programok újrafordításával és szerkesztésével) letilthatjuk vagy engedélyezhetjük a nyomkövetést segítő üzenetek kiírását.

A következő makró egy ravasz területfoglalás. Az ember gyorsan megunja, hogy folytonosan ismételgesse a CHR_CR, CHR_LF, '$' szekvenciát. Ezért definiál egy üzenetdefiniáló makrót, amely paraméterként veszi át az üzenet nevét, szövegét - és egy feltételt (CRCONT, azaz CaRriage CONTrol, kocsivezérlés). A makróba épített feltételes blokk beépít az üzenet végére egy kocsivissza-soremelés karakterpárt, hacsak ezt külön le nem tiltjuk. Ez a makró tehát definiálhat olyan üzeneteket is, amelyekből többet lehet egy sorba írni.

Az FCBCLL nevű makró a file-kezelő funkciók meghívására szolgál. Az egyetlen apró trükk, hogy AL-t meg is teszteli, azaz kilépése után a flagekben megtaláljuk a visszatérési feltételt: sikeresen futott-e a meghívott funkció vagy sem. Itt a Zero flag állása érdektelen, tehát a teszteléssel nem vesztünk információt.

Ezután nagyon fontos struktúra következik: egy teljes FCB struktúrája. Persze hogy célszerű ilyet definiálni, de miért éppen így? Miért szerepel NAME és EXT alatt idézőjelek között

nyolc, illetve három szóköz, és miért DUP operátorral definiáltuk RES_FCB-t? Egyszerűen azért, mert az idézőjelek között megadott stringet a struktúra meghívásakor módosíthatjuk, és így könnyedén megadhatjuk a file-nevet. A név balra igazítását és szóközökkel való kiegészítését a fordító gondosan elvégzi, nekünk ezzel nem kell kínlódni. Viszont nem módosíthatjuk a RES_FCB elem tartalmát - de ehhez nincs is jogunk. Tehát a különböző definíciós lehetőségek kihasználása kényelmessé és biztonságosabbá teszi munkánkat.

A kiterjesztett FCB elemei triviálisak, ezekről nincs semmi mondanivalónk. Azonban az FCB_EXT struktúra meghívásakor gondosan ügyelni kell arra, hogy ez pontosan a kiterjeszteni kívánt FCB előtt történjék. Mivel struktúra tagjaként struktúrát nem használhatunk, kézenfekvő módon nem oldható meg abszolút biztonsággal a kiterjesztett FCB definiálása. Ekkor ismét a makrók sietnek segítségünkre: makró belsejében lehet struktúra! Ez látható az FCBEXT makró esetén. Ez a makró két struktúrahívást tartalmaz, és garantálja, hogy korrektül definiáljunk egy kiterjesztett FCB-t. Itt csak egy kérdés van: a kívánt rekordhosszt miért nem állíthatjuk be? Azért, mert azt csak a sikeres nyitás és az első írás vagy olvasás között szabad módosítani.

Ezzel kapcsolatban még egy keserű tapasztalatról kell beszámolnom. A struktúrahívások során a fordító némelykor kényes arra, hogy az előkészítésben használunk-e szóközt, és ha igen, hol. Tekintsük például az FCBEXT makró FCB_EXT struktúrahívását. Itt az első két elemet nem szabad (a másodikat nem is lehet) módosítani. Ha viszont szóközt írtunk volna a vesszők elé, akkor ezt a fordító módosítási kísérletnek tekinti, goromba hibaüzenetet ad, de az utolsó - az ATTRIB nevű - elemet nem módosítja (ez valószínűleg fordítóhiba).

A file utolsó két eleme két rekorddefiníció, melyek teljesen triviális módon definiálják a dátumot, illetve az időt tartalmazó szavak belső struktúráját. Ezek felhasználására a harmadik kötet ad majd példát.

VII.3.2. File létrehozása FCB-vel

Az első példaprogram igazán az elején kezdi: létrehoz egy file-t, beleír néhány byte-nyi szöveget, majd lezárja a frissen létrehozott file-t.

```
TITLE   File létrehozása FCB-vel
;
; Konstansok, makrók, rekordok
;
TRACE   EQU     1
        INCLUDE         FCB.INC

DTA_SIZE        EQU     30      ;DTA terület hossza - 2

;
; Kódszegmens
;

CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:CODE, ES:CODE, SS:CODE
        ORG     100H

ENTRY:                                  ;
                JMP     START           ;

FCB_CREAT       FCB     < DR_CURR, 'NEWFILE', 'XXX' >
WORK_DTA        DB      DTA_SIZE DUP ( 'A' )
                DB      CHR_CR, CHR_LF  ;Rekord végén CR-LF!
        MSGDEF  MSG_FCREAT_ERR, 'Sikertelen file-létrehozás'
        MSGDEF  MSG_FWRITE_ERR, 'Sikertelen file-írás'
        MSGDEF  MSG_FCLOSE_ERR, 'Sikertelen file-lezárás'

START:                                  ;-------;
                FCBCLL  DOS_FCREAT, FCB_CREAT   ;Létrehozzuk
                JE      FCREAT_OK               ;Sikerült!
                PRTSTR  MSG_FCREAT_ERR  ;Nem sikerült, vége
                JMP     EXIT            ;
FCREAT_OK:                              ;-------;
                FCBCLL  DOS_SETDTA, WORK_DTA    ;DTA-címzés
                MOV     FCB_CREAT.RECSIZ, DTA_SIZE + 2
                                        ;Rekord hossza 32!
                MOV     CX, 3           ;3 rekord lesz
WRITE_LOOP:                             ;
                FCBCLL  DOS_FWRSEQ, FCB_CREAT   ;Köv. ki
                JE      WRITE_OK        ;Sikerült!
                PRTSTR  MSG_FWRITE_ERR  ;Nem sikerült, vége
                JMP     EXIT            ;
```

```
WRITE_OK:                                       ;
                LOOP    WRITE_LOOP              ;Ismétlés háromszor
                FCBCLL  DOS_FCLOSE, FCB_CREAT   ;Lezárás
                JE      CLOSE_OK                ;Lezárás sikerült
                PRTSTR  MSG_FCLOSE_ERR          ;Nem sikerült, vége
                JMP     EXIT                    ;Csak "szépség"!
CLOSE_OK:                                       ;
                                                ;
EXIT:                                           ;
                INT     20H                     ;Kilépés
                                                ;
CODE    ENDS

        END     ENTRY
```

Ez a kis program, úgy érzem, különösebb magyarázatot nem igényel. Szépen létrehozza a NEWFILE.XXX file-t, és kiír három rekordot, amelyek harminc-harminc A karaktert, valamint egy kocsivissza-soremelést tartalmaznak. A létrehozott file hossza így 96 byte lesz. Egyetlen apró inkorrektséget már csak azért is beépítettem a kódba, hogy illusztráljam: az ember furcsa helyzetbe kerülhet, ha nem igazán jól szabja meg makrói és szubrutinjai feladatkörét. Az FCBCLL makró megírása igazán kézenfekvő volt, de szigorúan véve nem használható a DTA terület átcímzésére. Mivel azonban a paramétereket pontosan ugyanúgy kell beállítani mindkét esetben, a fordító (és a rendszer) nem veszi észre a turpisságot, és minden rendben lesz, legfeljebb egy naivabb programozó fog elbotlani, ha olvassa a programunkat. Talán szerencsésebb lett volna a makrót másként nevezni!

Az utolsó JMP utasítás pusztán azért szerepel, hogy az egyes programelemek teljesen egységesek legyenek; nem használjuk ki azt a körülményt, hogy az utolsó rendszerhívás egyben a program utolsó érdemi utasítása is - ezzel pedig meghagyjuk annak lehetőségét, hogy a CLOSE_OK és az EXIT közé további utasításokat szúrjunk majd be.

Amennyiben valamilyen különleges dologot szeretnénk elérni, például rejtett file-t kívánunk létrehozni, kiterjesztett FCB-vel kell dolgoznunk. Ekkor a kiterjesztés címét kell DX-be tölteni, és a kiterjesztés attributum-elemébe kell beírni a kívánt értéket. Létrehozhatnánk így akár egy lemezcimkét is. A lemezcimke esetén vigyáznunk kell arra, hogy ne legyen több ilyen a lemezen!

VII.3.3. Egy csacska file-másoló program

A következő program egy file-másoló program, amely az előző programmal létrehozott NEWFILE.XXX file-t átmásolja egy másik file-ba, amelynek neve legyen például CPDFILE.XXX. Miért írtuk, hogy ez egy csacsi program? Mert szeretnénk rámutatni az FCB-s file-kezelés legnagyobb hátrányára. Ez pedig a következő: ha elkezdünk olvasni egy file-t adott rekordhosszúsággal, és a file hossza történetesen nem osztható ezzel a számmal, akkor az utolsó olvasásra csak egy "tört" rekord jön be. Ez még nem lenne baj; megkapjuk a 03 válaszkódot, a DTA-n pedig a rekord 0 kódokkal kiegészítve jelenik meg.

Igen ám, de mit tehetünk a másolás során, ha egyszer már elkezdtünk egy adott rekordhosszal írni? Semmit; az utolsó rekord értékes részét csak a kiegészítő 0 kódokkal együtt tudjuk kiírni az output file-ra, és ezzel a másolást mindjárt el is rontottuk, hiszen megváltozott a file hossza. Még súlyosabb a helyzet, ha olyan programot akarunk írni, amely egy adott file tartalmát megkutyulja úgy, hogy bizonyos csomagonként felolvassa, csinál vele valamit, majd a csomagot visszaírja oda, ahonnan felvette. Ezt a problémát vagy úgy oldhatjuk meg, hogy a rekordhosszúságot egy byte-ra állítjuk (ami nem egy igazán szerencsés lépés, mert a kiírások számát nagyon megnöveli), vagy pedig MS-DOS file-kezelő függvényeket használunk. Ott az ilyen nehézségek egyszerűen nem léteznek. Lássuk most a programot - a rekordhosszúságot "véletlenül" 17 byte-ban szabtuk meg, ami persze nem osztója a file hosszának, mivel az a 32-nek háromszorosa.

```
TITLE   File másolása FCB-vel
;
; Konstansok, makrók, rekordok
;
TRACE   EQU     1
        INCLUDE         FCB.INC
DTA_SIZE        EQU     17
;
; Kódszegmens
;
CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:CODE, ES:CODE, SS:CODE
        ORG     100H                    ;
```

```
ENTRY:                                  ;
                JMP     START           ;

FCB_INPUT       FCB     < DR_CURR, 'NEWFILE', 'XXX' >
FCB_OUTPUT      FCB     < DR_CURR, 'CPDFILE', 'XXX' >

WORK_DTA        DB      DTA_SIZE DUP ( 'A' )
                DB      '$'

        MSGDEF  MSG_FOPEN_ERR,  'Nyitási hiba'
        MSGDEF  MSG_FCREAT_ERR, 'Létrehozási hiba'
        MSGDEF  MSG_FREAD_ERR,  'Olvasási hiba'
        MSGDEF  MSG_FWRITE_ERR, 'File-írási hiba'
        MSGDEF  MSG_FCLOSO_ERR, 'Output file zárási hiba'
        MSGDEF  MSG_FCLOSI_ERR, 'Input file zárási hiba'
        MSGDEF  MSG_RDEOF,      'Vége az input file-nak'
        MSGDEF  MSG_RDFRAGM,    'Tört rekord olvasása'
        MSGDEF  MSG_CRLF,       ' '     ;Csak CR-LF
                                        ;
START:                                  ;
                FCBCLL  DOS_FCREAT, FCB_OUTPUT
                JE      FCREAT_OK       ;Létrehozás sikeres
                PRTSTR  MSG_FCREAT_ERR  ;Hibaüzenet, kilépés
                JMP     EXIT            ;
FCREAT_OK:                              ;
                FCBCLL  DOS_FOPEN, FCB_INPUT
                JE      FOPEN_OK        ;File-nyitás sikeres
                PRTSTR  MSG_FOPEN_ERR   ;Hibaüzenet, kilépés
                JMP     EXIT            ;
FOPEN_OK:                               ;
                FCBCLL  DOS_SETDTA, WORK_DTA
                MOV     FCB_INPUT.RECSIZ,  DTA_SIZE
                MOV     FCB_OUTPUT.RECSIZ, DTA_SIZE
COPY_LOOP:                              ;
                TRCMSG  DOS_FRDSEQ, FCB_INPUT
                JE      READ_OK         ;Olvasás sikeres
                CMP     AL, RET_NOBYTE  ;
                JE      READ_EOF        ;File vége
                CMP     AL, RET_FRAGM   ;
                JE      READ_FRAGMENT   ;Tört rekord jött
                PRTSTR  MSG_FREAD_ERR   ;Hibaüzenet, kilépés
                JMP     EXIT            ;
```

```
READ_EOF:                                       ;
                PRTSTR  MSG_CRLF                ;Ki az üzenet
                PRTSTR  MSG_RDEOF               ;
                JMP     FCLOSE                  ;Befejezzük
READ_FRAGMENT:                                  ;
                PRTSTR  MSG_CRLF                ;Ki az üzenet
                PRTSTR  MSG_RDFRAGM             ;
                JMP     FINISH                  ;Tört rekord ki
READ_OK:                                        ;
                TRCMSG  WORK_DTA                ;Rekord STDOUT-ra
                FCBCLL  DOS_FWRSEQ, FCB_OUTPUT
                JE      WRITE_OK                ;Sikeres kiírás
                PRTSTR  MSG_FWRITE_ERR          ;Hibaüzenet, kilépés
                JMP     EXIT                    ;
WRITE_OK:                                       ;Csak "szépség", az
                JMP     COPY_LOOP               ; egységesség kedvéért
FINISH:                                         ;
                FCBCLL  DOS_FWRSEQ, FCB_OUTPUT
                JE      WRITE_LAST_OK           ;Sikeres kiírás
                PRTSTR  MSG_FWRITE_ERR          ;Hibaüzenet, kilépés
                JMP     EXIT                    ;
WRITE_LAST_OK:                                  ;
FCLOSE:                                         ;
                FCBCLL  DOS_FCLOSE, FCB_OUTPUT
                JE      CLOSO_OK                ;Output lezárva
                PRTSTR  MSG_FCLOSO_ERR          ;Hibaüzenet, kilépés
                JMP     EXIT                    ;
CLOSO_OK:                                       ;
                FCBCLL  DOS_FCLOSE, FCB_INPUT
                JE      CLOSI_OK                ;Input lezárva
                PRTSTR  MSG_FCLOSI_ERR          ;Hibaüzenet, kilépés
                JMP     EXIT                    ;
CLOSI_OK:                                       ;
EXIT:                                           ;
                INT     20H                     ;Kilépés
CODE    ENDS
        END     ENTRY
```

Jóllehet láttunk már példákat erre, szeretném külön is felhívni a figyelmet a kettős cimkékre. Az FCLOSE cimke helyett használhattuk volna egyszerűen a WRITE_LAST_OK cimkét akkor, mikor a file végének elérésekor további kiírás nélkül akartunk

ugrani a lezárási szakaszra. Ekkor azonban elsikkadna az, hogy két különböző feltétel alapján lépünk rá ugyanarra a pontra. Emiatt persze az adott pont logikailag nem egy, hanem két belépés, és csak véletlen az, hogy ezek fizikailag egybeesnek.

Figyeljük meg a PRTSTR és a TRCMSG makrók használatát is! Az előbbivel írattuk ki a hibaüzeneteket, hiszen ezeknek mindig meg kell jelenniük. Azokat a szövegeket azonban, amelyek "készterméken" már nem szükségesek, a TRCMSG makróval írtuk ki. Mikor a program elkészül, akkor sokkal egyszerűbb feladat lesz így eltüntetni a nyomkövető üzenetek kiírását.

VII.3.4. File-ok keresése és listázása

A következő program a file-keresés és -listázás egy alkalmazását mutatja be. Az előző két programot lefuttatva két XXX típusú file-t hoztunk létre. Az alábbi program pedig megkeresi a kurrens directory-ban az összes ilyen típusú file-t, és kiírja a nevüket.

```
TITLE   File-ok keresése és listázása FCB-vel
;
; Konstansok, makrók, rekordok
;
TRACE   EQU     1
        INCLUDE         FCB.INC

;
; Kódszegmens
;
CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:CODE, ES:CODE, SS:CODE
        ORG     100H
ENTRY:                                  ;
                JMP     START           ;

FCB_SEARCH      FCB     < DR_CURR, '*', 'XXX' > ;Keresendő file
WORK_DTA        DB      40  DUP ( ? )
        MSGDEF  MSG_FILNAM, 'File neve: ', NOCR
        MSGDEF  MSG_NOFILE, 'Nincs ilyen file!'
        MSGDEF  MSG_NOMORE, 'Nincs több ilyen file!'
        MSGDEF  MSG_CRLF,   ' '         ;
```

```
START:                                  ;
                FCBCLL  DOS_SETDTA, WORK_DTA    ;DTA beállítás
                FCBCLL  DOS_FSFRST, FCB_SEARCH  ;Első keresése
                JE      FIRST_FOUND     ;Az első megvan
                PRTSTR  MSG_NOFILE      ;Nincs, üzenet
                JMP     EXIT            ; és vége
FIRST_FOUND:                            ;
SEARCH_LOOP:                            ;
                PRTSTR  MSG_FILNAM      ;Bevezető szöveg ki
                MOV     BYTE PTR WORK_DTA + 12, '$'
                PTRSTR  WORK_DTA+1      ;File-név és típus ki
                PTRSTR  MSG_CRLF        ;Kocsivissza ki
                FCBCLL  DOS_FSNEXT, FCB_SEARCH  ;Következő ker.
                JE      SEARCH_LOOP     ;Van még, folytatjuk
                PRTSTR  MSG_NOMORE      ;Nincs - üzenet és vége
EXIT:                                   ;
                INT     20H             ;Kilépés
                                        ;
CODE    ENDS

        END     ENTRY
```

Ez a program egyetlen szempontból érdekes még. Figyeljük meg azt, hogy az aránylag szerencsésnek mondható makródefiníciók felhasználásával egyes helyeken szinte magaszintű nyelven programozunk; egy sor többé nem egyetlen, hanem több gépi utasításra fordul le, és egyáltalán nem egy egyszerű, "gépikód-szerű" funkciót lát el, hanem egy viszonylag összetett műveletet végez (persze felhasználva a rendszer rutinjait és egyéb makró- és struktúradeklarációkat is). Ha assemblerben programozunk, akkor nagyon gyakran élünk ezzel az eszközzel. Jól definiált makró- és szubrutinrendszer lehetővé teszi az egyre magasabb szintű "nyelven" való programozást; és ezt a nyelvet mi magunk definiáljuk!

VII.3.5. File-ok törlése típus szerint

Az előbb nagy fáradsággal létrehozott XXX típusú file-okat törli ki az alábbi programocska. Rendkívül egyszerű, semmilyen megjegyzést nem kell hozzáfűzni.

```
TITLE   File(ok) törlése FCB-vel
;
; Konstansok, makrók, rekordok
;
TRACE   EQU     1
        INCLUDE         FCB.INC
;
; Kódszegmens
;

CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:CODE, ES:CODE, SS:CODE
        ORG     100H

ENTRY:                                  ;
                JMP     START           ;
                                        ;-------;
FCB_DELETE      FCB     < DR_CURR, '*', 'XXX' > ;Törlendő file
                                        ;specifikációja
        MSGDEF  MSG_NOFILE, 'Nincs ilyen file'
        MSGDEF  MSG_DELETE, 'Minden XXX file törölve'   ;

START:                                  ;
                FCBCLL  DOS_FDELET, FCB_DELETE  ;Törlés
                JE      DELETE_OK       ;Sikeres volt
                PRTSTR  MSG_NOFILE      ;Nincs file - üzenet,
                JMP     EXIT            ; kilépés
DELETE_OK:                              ;
                TRCMSG  MSG_DELETE      ;
EXIT:                                   ;
                INT     20H             ;Kilépés
                                        ;
CODE    ENDS
        END     ENTRY
```

VII.3.6. File-típusok módosítása

Futtassuk le újra az első két pédaprogramot, és hozzuk létre megint a NEWFILE.XXX és CPDFILE.XXX file-okat! Mielőtt azonban a negyedik példaprogram újabb futtatásával törölnénk őket, in-

dítsuk el az alábbi programot, amely a file-ok nevének vagy típusának (itt típusának) módosítását illusztrálja. Tegyük fel, hogy a program neve FREN.COM. Az indítás legyen a következő:

```
C>FREN                  hibás, paraméter nélkül
C>FREN *.XXX            hibás, egy paraméterrel
C>FREN *.XXX *.YYY      korrekt, két file-specifikáció
```

Az első két hívásra persze

Illegális paraméterek!

hibaüzeneteket kapunk, míg a harmadikra a

File-neveket módosítottam.

üzenet jön; a program minden XXX típusú file nevét (itt típusát) YYY-ra módosította. Ha ezután megkíséreljük a harmadik hívást ismét, megint hibaüzenet érkezik:

Nincs ilyen file!

Ezután megkísérelhetjük az összes XXX típusú file törlését a törlő példaprogrammal. Ha a program jó, akkor tőle is hibaüzenetet kapunk. Az FREN.COM program újabb futtatásával visszaállíthatjuk az eredeti file-neveket, ezután persze a törlő program is sikeresen fut majd le.

```
TITLE   File-átnevezés FCB-vel
;
; Konstansok, makrók, rekordok
;

TRACE   EQU     1
        INCLUDE         FCB.INC

PSP_PRM_FCB     EQU     5CH     ;Elsődleges FCB a PSP-ben
PSP_SEC_FCB     EQU     6CH     ;Másodlagos FCB a PSP-ben

;
; Kódszegmens
;

CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:CODE, ES:CODE, SS:CODE
        ORG     PSP_PRM_FCB
FCB_RENAME      FCB     <>              ;Paraméterblokk címe!
        ORG     100H                    ;
```

```
ENTRY:                                  ;
                JMP     START           ;Ugrás a belépésre

        MSGDEF  MSG_ILLPAR, 'Illegális paraméterek!'
        MSGDEF  MSG_NOFILE, 'Nincs ilyen file!'
        MSGDEF  MSG_RENAME, 'File-neveket módosítottam.'

START:
                MOV     BX, PSP_PRM_FCB + 1     ;Első név címe
                MOV     AL, BYTE PTR [ BX ]     ; első karakter
                CMP     AL, ' '                 ;Van ott valami?
                JE      ILLPAR                  ;Nincs - illegális par.
                MOV     BX, PSP_SEC_FCB + 1     ;Második név c.
                MOV     AH, BYTE PTR [ BX ]     ; első karakter
                CMP     AH, ' '                 ;Van ott valami?
                JE      ILLPAR                  ;Nincs - illegális par.
                FCBCLL  DOS_FRENAME, FCB_RENAME ;Átnevezzük
                JE      RENAME_OK               ;Sikeres volt, üzenet
                PRTSTR  MSG_NOFILE              ;Hibaüzenet és
                JMP     EXIT                    ; kilépés
RENAME_OK:                              ;
                TRCMSG  MSG_RENAME      ;
                JMP     EXIT            ;
ILLPAR:                                 ;
                PRTSTR  MSG_ILLPAR      ;Hibaüzenet, kilépés
EXIT:                                   ;
                INT     20H             ;Kilépés
                                        ;
CODE    ENDS
        END     ENTRY
```

Ebben a programban láttunk példát először a PSP-ben levő információk használatára. Talán ez a legegyszerűbb alkalmazás. Mindenesetre egy érdekes apróságot figyelhetünk meg. Jószerével többet kellett dolgoznunk a paraméterek helyességének ellenőrzésével, mint magával a programmal. Ez bonyolultabb paraméterek esetén még szembetűnőbb lesz. Vegyük figyelembe azt is, hogy az ellenőrzés legkínosabb részét az MS-DOS végezte el! Hiszen a rendszer vizsgálta meg, hogy a megadott (két) paraméter értelmezhető-e file-névként vagy sem (hiszen a PSP elsődleges és másodlagos FCB-jében csak akkor találunk szóköztől eltérő értéket, ha a paraméterek erre alkalmasak).

VIII. MS-DOS file-kezelés

VIII.1. Az MS-DOS file-kezelés alapjai

Az MS-DOS file-kezelésről vázlatosan a következőket kell tudni: megnyitás előtt egy, a file-t specifikáló stringet kell előkészítenünk, s ennek címét kell átadni a rendszernek. Válaszképpen egy sorszámot, a "file-handle"-t kapjuk meg; minden további file-művelet során erre kell hivatkoznunk. A megnyitás során utalnunk kell a kívánt műveletekre is (írás, olvasás vagy írás-olvasás). A "file-handle"-t a továbbiakban, mint már említettük, file-sorszámnak mondjuk majd.

A string tartalmazhatja a lemezegység, a teljes directory-út (path) specifikációját és a file nevét és típusát, sőt, számítógép-hálózatokban a célszámítógép hálózati nevét is, tehát egy teljes file-specifikációt. A string végét egy 0 byte-al kell jelezni.

A rendszer a file-t az ún. file pointer felhasználásával kezeli. Ez a pointer azt mutatja meg, hogy a file hányadik byteja a következő művelet "címzettje". A pointer a file megnyitása után 0 értékű, azaz a fizikailag első byte-ra mutat. Minden egyes művelet során automatikusan előre mozdul: ha tíz byte-ot olvasunk be, akkor tízzel. Tehát a következő olvasás a most felolvasottak után következő byte-o(ka)t fogja olvasni. A file pointert szabadon átállíthatjuk bárhová, így válik lehetővé a direkt elérés.

A hagyományos file-kezelés során a file-ra az ún. FCB, a File Control Block segítségével hivatkozunk, s ez teszi lehetővé a file hagyományos kezelését (rekordos direkt elérés). A file-sorszám egy UNIX-szerű file-elérést biztosít, melynek során egyfelől a file-t mint byte-folyamot tekintjük, amelyből "bárhonnan meríthetünk egy vödörrel", másfelől pedig a perifériális eszközöket (mint például a két részre választott konzolt) is file-szerűen kezeljük.

Ezek a funkciók az eddigiektől eltérően nem az AL regiszterben, hanem a STATUS Carry flagjében adnak jelzést arról, hogy sikeresen futottak-e le, vagy valami hiba következett be a művelet során. Amennyiben Carry értéke 0, akkor a művelet sikeres volt. Ha a Carry-ben 1-et találunk, akkor valamilyen hiba következett be, melynek kódját az AL regiszterben kapjuk meg. Azonban vannak olyan funkciók, amelyek sikeres futás esetén az AX regiszterben szolgáltatnak egyéb visszatérési értéket. Tehát

e funkciók használata után először a Carry flaget kell megvizsgálnunk. Ha ez 0, akkor minden kétkedés nélkül folytathatjuk munkánkat; az AX-ben levő értéket (ha van) eredeti jelentése szerint használhatjuk. Ha a Carry 1, akkor az AL-ben található érték egy hibakód. Ennek megfejtése után kell eldöntenie programunknak, hogy folytathatja-e munkáját, vagy ki kell lépnie. Hiba esetén hasznos gyakorlat az 59 funkció segítségével bővebb információt és tanácsot kérni a rendszertől.

Az MS-DOS file-kezelés során az FCB-s file-kezeléshez hasonló fő csapásirányokat ismerhetjük fel: a file hagyományos kezelése (írás-olvasás), file-ok keresése, törlése és nevük módosítása (amely itt bővebb lehetőségeket kínál: a directory-bejegyzés minden olyan elemét módosíthatjuk, amely értelmesen egyáltalán módosítható). Az ötödik fontos csoport a file-sorszámok módosítását (és így átirányítását) segítő funkciók csoportja. Végül a hatodik fontos terület a direkt device-vezérlés. Az erre szolgáló igen összetett 44 funkciót ismerteti a "Device-ok direkt vezérlése" fejezet.

A második, harmadik és negyedik csoport, úgy tűnik, sokkal inkább illik a directory-kezelő, mint a file-kezelő funkciók közé. Ezért, a "Hagyományos file-kezelés" fejezettől eltérően ezeket nem itt, hanem a "Directory-kezelés" fejezetben ismertetjük.

VIII.2. Az MS-DOS file-kezelésre szolgáló funkciók

VIII.2.1. File-kezelés file-sorszámok segítségével

3C Create Handle - DOS_CREAT

File létrehozása — hasonló a 16 funkcióhoz, de itt a belépéskor DS:DX a file specifikációját tartalmazó nullával lezárt ASCII-stringre kell mutasson. A file-specifikáció alakja (ASCII-karaktersorozat):

```
dr:\path\név.típus <0>
```

ahol "dr" a lemezegység neve, "path" a directory-út, "név" és "típus" a file neve és típusa, végül <0> a stringet lezáró nulla byte-ot jelenti. Mindezeket az MS-DOS felhasználói szintjén megszokott alakban kell megadni, például:

```
FILE1   DB  'C:\ASM\FIRST.DAT', 0
```

Megjegyzés: Ebből következik, hogy ilyen módon tetszőleges directory-ban hozhatunk létre file-okat. Amennyiben a "path" egészen elmarad, akkor a file a kurrens directory-ba kerül. Ha egy aldirectory-ban vagyunk, és a ROOT directory-ban akarunk file-t létrehozni, akkor a file-specifikáció alakja:

```
dr:\név.típus <0>
```

Például az AUTOEXEC.BAT megadása:

```
AESPEC  DB  'C:\AUTOEXEC.BAT', 0
```

CX-be írhatjuk be a file kívánt attributumát (lásd "Directory-bejegyzés" fejezet). Ha a specifikált file már létezik, akkor tartalma elvész, és írását elölről kezdhetjük. Ha nem létezik ilyen file, akkor a rendszer a megadott directory első szabad bejegyzésében létrehozza a file-ra mutató bejegyzést. Az új file elérési módja "írás/olvasás", a megosztási mód "kompatibilis", és a file-t az elindított programok öröklik (vö. 3D funkció, DOS_OPEN).
Sikeres művelet után a Carry értéke 0, ekkor az AX regiszter tartalmazza a file-sorszámot. Erre a sorszámra hivatkozunk minden további file-műveletnél, így mentése nagyon fontos.
Hibakódok AX-ben Carry=1 esetén:

03 - a megadott directory-út (path) nem létezik

04 - túl sok file-t nyitottunk meg, nincs több szabad file-sorszám

05 - nincs szabad directory-bejegyzés;
- a file létezik és írásvédett, tartalmát nem lehet törölni;
- van ilyen nevű aldirectory a directory-ban (csak akkor, ha a létrehozni kívánt file típusa üres, azaz három szóköz lesz belőle);
- directory-t próbáltunk létrehozni;
- lemezcimkét próbáltunk létrehozni, de már létezik.

Ha a megadott file-attributum 0, akkor közönséges archív file-t kapunk. Amennyiben az attributummal csak olvasható file-t hozunk létre, akkor a file ezt az att-

ributumot csak a létrehozás utáni lezáráskor kapja meg, tehát a létrehozás után még írhatunk a file-ba. Ez eltér az FCB-vel létrehozott file-októl; amennyiben a 16 funkcióval hozunk létre írásvédett file-t, akkor már mi sem írhatunk bele.

MS-DOS 3.00 és későbbi verziók

5A Create Temporary File, DOS_CRTEMP
Ideiglenes file létrehozása egyedi névvel. Bemenetként DS:DX egy olyan területre mutat, amelyben egy 0-val lezárt string megad egy directory-utat (path). A string utolsó értékes karakterének a "\"-nek (backslash) kell lennie. A lezáró 0-t követően még 13 byte szabad helyre van szükség a létrehozandó file neve számára. A CX regiszter tartalmazza a kívánt attributumot (lásd a "Directory-bejegyzés" fejezetben).
A rendszer a megadott directory-ban létrehoz egy egyedi nevű file-t (a név létrehozási módja garantálja, hogy ameddig a directory be nem telik, mindig létre lehet hozni ilyen file-nevet). Az új file nevét a rendszer közvetlenül a directory-út mögé írja, és a teljes file-specifikáció címét a (változatlan) DS:DX regiszterben kapjuk vissza.
Sikeres létrehozás után az új file-sorszámot AX-ben találjuk meg. A file elérési módja írás-olvasás, a megosztási módja kompatibilis; az elindított programok öröklik file-t (vö. 3D - DOS_OPEN).
Felhívjuk a figyelmet arra, hogy az ideiglenes file-ok törléséről a rendszer nem gondoskodik automatikusan, ezt programunknak kell elvégeznie. Ez kissé kényelmetlennek látszik, de nem nagy ügy, hiszen a rendszer úgy adja vissza a file-nevet, hogy annak felhasználásával bármikor meghívhatjuk a file-törlési funkciót (41, lásd a "Directory-kezelés" fejezetben). A látszólagos hiányosság valódi oka az, hogy belövés idején meg tudjuk nézni, mit is hagyott a program kilépése után az ideiglenesnek szánt file-okban. A kész programban kilépés előtt természetesen töröljük majd e file-okat.
Hibakódok AX-ben Carry=1 esetén mint 3C-nél, és:

05 - tiltott elérés - a directory betelt

MS-DOS 3.00 és későbbi verziók

5B Create New File, DOS_CRNEWF

Ez a funkció pontosan megegyezik a 3C (DOS_CREAT) funkcióval azt kivéve, hogy ha a megadott file létezik, akkor annak lecsonkítása helyett hibakóddal tér vissza. Sikeres file-létrehozás esetén Carry-ben 0 van, AX tartalmazza az új file-sorszámot. A létrehozott file elérési kódja írás-olvasás, megosztási módja kompatibilis (vö. 3D - DOS_OPEN).
Hibakódok AX-ben Carry=1 esetén mint 3C-nél és:

50 - ugyanazon specifikációjú file már létezik

3D Open Handle, DOS_OPEN

File megnyitása - hasonló a 0F funkcióhoz, de itt a belépéskor DS:DX egy, a file nevét tartalmazó (nullával lezárt) ASCII-stringet címez. AL-ben adjuk meg az elérési kódot:

0 - olvasás
1 - írás
2 - írás/olvasás

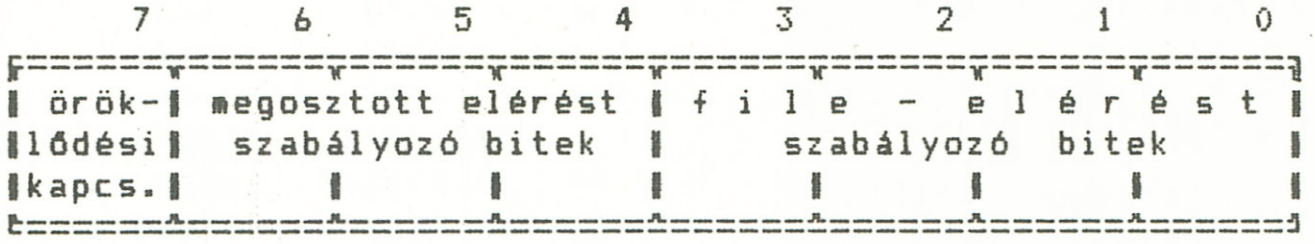
Ezzel a funkcióval csak létező file-t nyithatunk meg. Sikeres lefutás esetén Carry értéke 0, ekkor AX tartalmazza a file-sorszámot.
Hibakódok AX-ben Carry=1 esetén:

01 - illegális elérési vagy megosztási kód
02 - a megadott file nem létezik
03 - a megadott directory-út (path) nem létezik
04 - túl sok file-t nyitottunk meg, nincs több szabad file-sorszám
05 - az elérés tiltott, írásvédett file-t akartunk írásra megnyitni;
- megosztott elérés esetén nem megengedett megosztási módon próbáltuk megnyitni a file-t
0C - szabálytalan elérési kód, nemlétező file-elérési kódot vagy megosztási módot adtunk meg

Sikeres megnyitás után az írási/olvasási pointer a file első byte-jára mutat (módosítása: 42 funkció).

MS-DOS 3.00 és későbbi verziók

Az MS-DOS 3.00 és későbbi verziói (nagy lépést téve a multiprogramozás felé) megengedik a file megosztott kezelését. Az osztott elérés vezérléséhez el kell indítani az MS-DOS SHARE nevű programját. Ez aktivizálja az MS-DOS osztott file-elérést vezérlő driverét, és létrehozza az ehhez szükséges adatterületeket.
Az AL további bitjeinek felhasználásával szabhatjuk meg azt, hogy a programunkból elindított további programok hozzáférhetnek-e az itt megnyitott file-sorszámokhoz, és ha igen, hogyan. A bitek kiosztása a következő:



Ha az öröklődési kapcsoló (inherit bit) értéke
0 - az elindított programok "öröklik" a file-t. Ha az elindított programnak valamilyen ravasz módon átpasszoljuk a megnyitott file sorszámát, akkor ez a program ugyanúgy kezelheti a file-t, mint az indító program (lásd a "Memóriakezelés és programvezérlés" fejezetet, a "Saját program elindítása programból" példát, amelyben elkövetjük ezt a nem éppen erkölcsös trükköt).
1 - az elindított programok nem öröklik a file-t. Hiába passzoljuk át az elindított programnak a megnyitott file sorszámát, az nem kezelheti a file-t. Minden kísérletnél a 06 hibakódot kapja: a file-sorszám nem létezik.

A megosztott elérést (sharing mode) szabályozó bitek:
000 - kompatibilis mód - bármely program (beleértve a file-t először megnyitó programot is) akárhányszor újranyithatja a file-t, ha a nyitáskor és az újranyitáskor a megosztási bitekben 000 van. A megnyitás minden más esetben sikertelen.

001 - írásra-olvasásra tiltott - ha a file-t először így nyitottuk meg, akkor más program semmilyen elérési és megosztási módon nem nyithatja meg a file-t.
Ez tehát az öröklléstől függetlenül biztosítja azt, hogy a file-t csak az elsőként megnyitó program használhatja, az ellenben többször is megnyithatja.

010 - írásra tiltott - ha a file-t először így nyitottuk meg, akkor más program a file-t írásra vagy írásra és olvasásra már nem nyithatja meg.

011 - olvasásra tiltott - ha a file-t először így nyitottuk meg, akkor más program olvasásra vagy írásra és olvasásra már nem nyithatja meg.

100 - nincs tiltás - ha a file-t először így nyitottuk meg, akkor más program kompatibilis módban már nem nyithatja meg.

Amennyiben a file újra megnyitása tiltott, akkor a DOS vagy a 05 felhasználói hibakóddal tér vissza, vagy pedig, indokolt esetben, egy INT 24-et hajt végre (kritikus hiba). Itt az 59 funkció meghívásával kérdezhetjük le a hiba pontosabb okát.
Az alábbi táblázat mutatja be, hogy milyen elérési és milyen megosztási módok mellett milyen újabb elérési móddal nyitható meg a file. Tehát a táblázat soraiban szerepelnek a lehetséges első megnyitási módok (kompatibilis módon írásra vagy olvasásra és így tovább), az oszlopokban pedig a második (esetleg harmadik stb.) megnyitás lehetséges módjai. A sorok és oszlopok találkozásánál olvashatjuk, milyen eredményt várhatunk a megosztott elérési kísérletektől. Rövidítések:

i - a file-t írásra nyitottuk meg
o - a file-t olvasásra nyitottuk meg
m - a file-t írásra-olvasásra nyitottuk meg
"." - a második megnyitási kísérlet illegális, és Carry=1 mellett felhasználói hibakódot kapunk
"*" - a második megnyitási kísérlet illegális, és kritikus hibát okoz (INT 24)
"I" - a második megnyitás megengedett

			Ismételt megnyitás													
			komp. mód			i/o tilt.			írás tilt			olv. tilt			nincs t.	
Első	mód	mód-elérés	i	o	m	i	o	m	i	o	m	i	o	m	i	o	m
	komp. mód	i	I	I	I	.	.	.	.	.	.	.	.	.	.	.	.
		o	I	I	I	.	.	.	.	.	.	.	.	.	.	.	.
		m	I	I	I	.	.	.	.	.	.	.	.	.	.	.	.
	írás/olv. tilt.	i	*	*	*	.	.	.	.	.	.	.	.	.	.	.	.
		o	*	*	*	.	.	.	.	.	.	.	.	.	.	.	.
		m	*	*	*	.	.	.	.	.	.	.	.	.	.	.	.
	írás tilt.	i	*	*	*	.	.	.	.	.	.	.	I	.	.	I	.
		o	*	*	*	.	.	.	.	I	.	.	.	.	.	I	.
		m	*	*	*	.	.	.	.	.	.	.	.	.	.	I	.
	olvasás tilt.	i	*	*	*	.	.	.	.	.	.	I	.	.	I	.	.
		o	*	*	*	.	.	.	I	.	.	.	.	.	I	.	.
		m	*	*	*	.	.	.	.	.	.	.	.	.	I	.	.
	nincs til-tás	i	*	*	*	.	.	.	.	.	.	I	I	I	I	I	I
		o	*	*	*	.	.	.	I	I	I	.	.	.	I	I	I
		m	*	*	*	.	.	.	.	.	.	.	.	.	I	I	I

Ez az ábra mindent elmond az egyes megnyitási lehetőségekről. Mint látható, a táblázat igen szép tengelyes szimmetriát mutat. Ezalól lényegében csak az a "fránya" kompatibilis mód kivétel: míg nem kompatibilis módon megnyitott file-t nem szerencsés kompatibilis módon újra megnyitni, hiszen az kritikus hibát okoz és a program abortálását eredményezi, addig az először kompatibilis módon megnyitott file-t csak kompatibilis módon nyithatjuk meg újra, minden egyéb kísérlet sikertelen (de nem okoz kritikus hibát).

Ennek okát könnyen megérthetjük. Ha a második program valamilyen ravasz megosztási móddal próbálja meg a file újramegnyitását, akkor feltehető, hogy tisztában van

jogaival, csak éppen a hívó program nem volt elég precíz a file első megnyitásánál. Ha azonban az első megnyitásnál az első program valamilyen megszorítást tett, és a hívott program kompatibilis módon próbálja újra megnyitni a file-t, akkor valószínű, hogy felelőtlen vigéckedésről van szó - abortálni kell a hívott programot.
A szimmetria egyébként a programok tökéletes egyenrangúságát mutatja. A meghívott program is élhet megszorításokkal a file-ra nézve a további megnyitásokat illetően. Tegyük fel például, hogy az első program azt gondolja: nekem mindegy, nem tiltok semmit, kivéve a kompatibilis módot (minden szamár azért ne érhesse el ezt a file-t), és megnyitja "semmi sem tiltott" módban, csak olvasásra. Azután meghív egy programot, s az már nem ilyen engedékeny: ő bizony megtiltja a file írását. Az első program, ideiglenesen visszakapva a vezérlést, meggondolja magát, és megpróbálja újranyitni a file-t, mert szeretne beleírni valamit. Ekkor azonban hibakódot kap, mivel a file elérése már "írásra tiltott".
Egy elérési korlátozás addig van érvényben, amíg valamelyik, a file-t használó program le nem zárja azt a sorszámot, amellyel a file-t megnyitotta és egyúttal korlátozta is annak további felhasználását. Ha a file-t egy vagy több program többször nyitotta meg, akkor több sorszám van életben, amelyek ugyanarra a file-ra mutatnak. A korlátozások sorszámokra és file-okra együtt érvényesek, tehát egy adott korlátozás nem szűnik meg attól, hogy egy másik, ugyanarra a file-ra mutató filesorszámot lezárunk.
A file-ok programok között megosztott elérését a "Memóriakezelés és programvezérlés" és az "Ami a példaprogramokból kimaradt" fejezetek példái illusztrálják.

3F Read Handle, DOS_READ
Olvasás egy file-ról vagy perifériális eszközről - BX-ben megadott sorszámú file-ról vagy eszközről olvas be a rendszer CX-ben megadott számú byte-ot, mégpedig a DS:DX-ben megadott című bufferbe. Visszatéréskor sikeres olvasás (Carry=0) esetén AX-ben van a ténylegesen beolvasott byte-ok száma. Ha ez 0, akkor a file végét akartuk éppen túlolvasni.

Hibakódok AX-ben Carry=1 esetén:

05 - tiltott elérés - a file-t olvasásra nem nyitottuk meg
06 - érvénytelen file-sorszám - nemlétező vagy meg nem nyitott file-sorszám

Nem biztos, hogy a rendszer CX számú byte-ot fog olvasni. Amennyiben az olvasott "file" az ASCII-módban kezelt klaviatúra (lásd 44 funkció), a rendszer legalább egy kocsivissza karakterrel lezárt sort beolvas!

40 Write Handle, DOS_WRITE

Kiírás file-ra vagy perifériális eszközre - BX-ben megadott sorszámú file-ra CX-ben adott számú byte-ot ír ki a DS:DX-ben adott című bufferből. Visszatéréskor sikeres művelet esetén (Carry=0) AX-ben van a kiírt byte-ok száma. Ha ez nem egyezik a CX-el (gyakorlatilag csak kisebb lehet), akkor valamilyen hiba következett be; valószínű, hogy a lemez megtelt.
Ha belépéskor a CX regiszter tartalma 0 volt, akkor a rendszer a file-pointer pillanatnyi értékét tölti a directory-bejegyzésbe mint a file hosszát. Ha a file mérete emiatt csökken, akkor a kieső clustereket a rendszer felszabadítja, ha pedig nő a file-hossz, akkor lefoglalja a szükséges számú clustert.

Megjegyzés: ez a lehetőség elsősorban arra szolgál, hogy fizikai kiírás nélkül lefoglaljuk a helyet egy olyan direkt eléréssel kezelendő file számára, melynél pontosan tudjuk a rekordhosszt és az összesen kiírni kívánt rekordok számát. Ez esetben előre mozdítjuk a File Pointert, és elvégezzük az üres kiírást. Ezután le kell kérdeznünk a file hosszát (lásd 42 funkció). Ha a kiírás sikeres volt, akkor a file hossza megegyezik az általunk definiált értékkel, és így lefoglaltuk a megfelelő területet a file számára. Nem kell attól félnünk, hogy valamelyik későbbi, valódi kiírás lesz sikertelen, mert nincs elég hely a lemezen. Ha a kiírás nem volt sikeres, és a file hossza nem egyezik meg az általunk megkívánttal, akkor nincs elég hely a file számára.

Hibakódok AX-ben Carry=1 esetén:

05 - tiltott elérés, a file-t írásra nem nyitottuk meg
06 - érvénytelen file-sorszám - nemlétező vagy meg nem nyitott file-sorszám

42 Move File Pointer, DOS_MOVPNT

A file írási/olvasási pointerének elmozdítása - a file három "pointerrel" rendelkezik. Az első mindig a file kezdetét címzi, értéke tehát 0. A második a kurrens pointer, a harmadik pedig mindig a file végére mutat (értéke megegyezik a file byte-okban kifejezett hosszával). A kurrens pointer mozgatása mindig úgy történik, hogy a rendszer egy általunk megadott duplaszavas értéket ad hozzá valamelyik pointerhez.
A file-sorszámot a BX regiszterben adjuk meg, CX:DX-ben a léptetés hosszát byte-okban (a magasabb helyiértékű szó CX-ben), AL-ben pedig az alapként használt pointer kódját.

0 - a file kezdetéhez viszonyítva (abszolút pozicionálás)
1 - a pillanatnyi pozícióhoz viszonyítva (relatív pozicionálás)
2 - a file végéhez viszonyítva (relatív pozicionálás)

Megjegyzés: ez utóbbi CX:DX=0 mellett alkalmas a file hosszának meghatározására.

Visszatéréskor a DX:AX regiszterpár tartalmazza a mozgatás utáni file-pointert.
Hibakódok AX-ben Carry=1 esetén:

01 - érvénytelen kód
06 - érvénytelen file-sorszám - nemlétező vagy meg nem nyitott file-sorszám

A file pointert tetszőleges helyre pozicionálhatjuk, akár a file végén túlra is. Ez nem okoz hibát (vö. 40 funkció), és módot ad arra, hogy a file méretét kiírás nélkül megnöveljük.

MS-DOS 3.00 és későbbi verziók

5C Lock/Unlock, DOS_PROTCT

File adott részének elzárása, vagy az elzárás feloldása. Belépéskor BX tartalmazza a file kezelésére használt file-sorszámot, a CX:DX regiszterpár az elzárandó vagy feloldandó terület kezdetének címét a file-on belül (CX a felső szó), SI:DI pedig a terület hosszát byte-okban (SI a felső szó).

Ezt a funkciót a megnyitásra vonatkozó, elég bonyolult feltételek mellett lehet használni:

- ha a file-t olvasásra, "olvasásra tiltott" vagy "nincs tiltás" megosztási módban nyitottuk meg;
- írásra és olvasásra, vagy csak írásra nyitottuk meg, "írásra tiltott" megosztási módban.

Felhasználásának csakis akkor van értelme, ha a SHARE paranccsal aktivizáltuk az MS-DOS megosztott file-elérést vezérlő driverét.

Az alfunkciók:

a) Elzárás (LOCK)

AL=0 - a rendszer elzárja a specifikált file megadott területét. Semmilyen más program nem férhet az elzárt területhez, sem írás, sem olvasás céljából. Egy file-ban egyidejűleg több egymásba nem nyúló részt is lezárhatunk.

Amennyiben más program kísérli meg a file elzárt részének elérését, a rendszer három próbálkozás után (hátha közben feloldják az elzárást) egy INT 24-et hajt végre. A kísérletek számát és két kísérlet közötti időt a 44 MS-DOS funkció 0B alfunkciójával lehet módosítani; lásd ott.

Az elzárást nem célszerű hosszú időn át fenntartani, a file lezárása vagy a program kilépése előtt pedig mindenképpen fel kell oldani.

Ha megkettőzzük a file-sorszámot (45 funkció), az új sorszám pontosan olyan elérési joggal rendelkezik, mint az eredeti. A 4B funkcióval elindított program, ha a file-t örökli is, nem örökli az elérés jogát.

A kritikus hiba-kezelő, valamint a CTRL-BREAK-et kezelő rutinnak a program abortálása előtt fel kell oldania a file elzárását.

Hibakódok AX-ben Carry=1 esetén:

01 - megosztási mód-hiba - a SHARE drivert aktivizálni kell e funkció használatához.
06 - érvénytelen file-sorszám - a BX-ben megadott file-sorszám nincs megnyitva
21 - a megadott terület vagy egy része már el van zárva (kritikus hibát vált ki; az 59 funkció hívásával kaphatjuk meg)

b) Az elzárás feloldása (UNLOCK)
AL=1 - feloldja a file adott területének elzárását.
Hibakódok AX-ben Carry=1 esetén:

01 - megosztási mód-hiba - a SHARE drivert aktivizálni kell e funkció használatához.
06 - érvénytelen file-sorszám - a BX-ben megadott file-sorszám nincs megnyitva
21 - a megadott terület nem egyezik meg semelyik elzárt területtel (kritikus hibát vált ki; az 59 funkció meghívásával kaphatjuk meg)

3E Close Handle, DOS_CLOSE
Lezárja a BX-ben megadott file-sorszámmal kezelt (3D, 3C, 5A vagy 5B valamelyikével megnyitott vagy létrehozott) file-t.
Hibakód (Carry=1 esetén):

06 - érvénytelen file-sorszám - nemlétező vagy meg nem nyitott file-sorszám

VIII.2.2. File-sorszámot módosító funkciók

45 Duplicate File Handle, DOS_DUPHND
File-sorszám megkétszerezése - belépéskor BX-ben egy megnyitott file-sorszámot adunk meg, a rendszer pedig AX-ben ad vissza egy újat, amely ugyanazon file ugyanazon pontjára mutat. Ez a funkció egy file többszörös felhasználását teszi lehetővé.

Megjegyzés: a megkettőzött file-sorszámok mögötti file-pointer ugyanaz, ha tehát az egyik sorszám felhasználásával módosítjuk annak file-pointerét (akár írással vagy olvasással, akár a pointer direkt címzésével), az egyúttal a másik sorszám mögötti pointert is érinti.

Hibakódok AX-ben Carry=1 esetén:

04 - túl sok file-t nyitottunk meg, nincs szabad file-sorszám

06 - érvénytelen file-sorszám - a régi egy nemlétező vagy meg nem nyitott file-sorszám

Force Duplicated File Handle, DOS_FRCHND

File-sorszám átirányítása - belépéskor BX-ben adjuk meg az eredeti, CX-ben pedig az átirányítani kívánt (már létező) file-sorszámot. A kilépés után a CX-ben levő pontosan oda fog mutatni, ahová a BX-ben levő. Ha CX éppen egy megnyitott file-ra mutatott, a rendszer az átirányítás előtt automatikusan lezárja.

Hibakódok AX-ben Carry=1 esetén:

06 - érvénytelen file-sorszám - nemlétező vagy meg nèm nyitott file-sorszám

Megjegyzés: ezek a funkciók akkor használhatók, ha egy megnyitott file-sorszámmal végzett műveleteket "át akarunk irányítani" egy másik file-ba. Tegyük fel például, hogy a standard hibajelző file-t szeretnénk a program futása közben ideiglenesen átirányítani egy lemezfile-ba úgy, hogy a futás egy későbbi szakaszában vissza akarunk térni a standard hibajelző file eredeti értelmezéséhez. Mit tehetünk ekkor? A DOS_CREATE funkcióval létrehozzuk a kívánt lemezfile-t, azután "mentjük" a standard hibajelzőt (0002 file-sorszám) a DOS_DUPHND funkcióval. Ezután átirányítjuk a 0002 sorszámot a lemezfile-ba. Amikor vissza akarjuk állítani a 0002 sorszám eredeti értelmezését, egyszerűen visszairányítjuk őt a sorszám megkettőzésekor kapott sorszámra (lásd a "Standard hibajelző file átirányítása" példát).

VIII.2.3. Device-ok direkt vezérlése

44 I/O Control for Device, DOS_IOCTL
Device-ok I/O vezérlése, azaz a perifériális eszközök be- és kiviteli műveleteinek közvetlen vezérlése vagy paramétereik lekérdezése - belépéskor BX tartalmazza a file-sorszámot, AL pedig egy alfunkció kódját.
Alfunkciók:

00 - DOS_IOCTL_GDAT
információkérés a sorszám által kezelt file-ról vagy eszközről DX-be

01 - DOS_IOCTL_SDAT
paraméterbeállítás a sorszám által kezelt eszközre vonatkozóan DX szerint (DH - a felső nyolc bit - csak 0 lehet!)

A 0. és 1. alfunkció által kezelt egyszavas adat bitkiosztása (melyet DX-ben kapunk vagy adunk meg):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
r	C							I	E	B	r	C	N	C	C
e	T	reserved						S	O	I	e	L	U	O	I
s	L							D	F	N	s	K	L	T	N

ahol:
res - reserved, fenntartott bitek
CTL - 1, ha a BX-ben megadott sorszám olyan eszközt jelent, amelyet e funkció 02 és 03 alfunkcióival vezérelni lehet, 0 különben
ISD - 1, ha a BX-ben megadott sorszám egy perifériális eszközt jelent; 0, ha egy file-t.

Ha a lekérdezett sorszám mögött egy lemezfile van (ISD bit=0), akkor lekérdezés után a 0-5 bitek a lemezegység sorszámát jelentik (0=A:, 1=B: stb.), EOF=0 pedig azt jelzi, hogy a sorszámot írtuk-e.

Ha ISD=1, azaz a sorszám egy perifériális eszközt jelent, akkor a következőket:

EOF - 0, ha beolvasás során elértük a file végét, és 1, ha nem
BIN - 1, ha az adatokat "nyersen" (a vezérlőkarakterek ellenőrzése nélkül) továbbítja a rendszer, és 0 egyébként. A standard input és output file esetén e bit jelentését lásd a 44 funkció ismertetésének végén.
CLK - 1, ha a BX-ben megadott sorszám a CLOCK$ eszközt jelenti (lásd a 44 funkció ismertetésének végén)
NUL - 1, ha a BX-ben megadott sorszám a NUL logikai eszközt jelenti (lásd a 44 funkció ismertetésének végén)
COT - 1, ha a BX-ben megadott sorszám a standard output file-t jelenti
CIN - 1, ha a BX-ben megadott sorszám a standard input file-t jelenti

További alfunkciók:

02 - DOS_IOCTL_RCNT
CX-ben megadott számú byte olvasása a sorszám által kezelt eszköz vezérlőcsatornájáról a DS:DX-ben megadott című bufferbe.

03 - DOS_IOCTL_WCNT
CX-ben megadott számú byte kiírása a sorszám által kezelt eszköz vezérlőcsatornájára a DS:DX-ben megadott című bufferből.

04 - DOS_IOCTL_RDSK
ugyanaz, mint 02, de file-sorszám helyett a BL-ben megadott kódú lemezegység vezérlőcsatornáját olvassuk. (0 - kurrens, 1 - A stb.)

05 - DOS_IOCTL_WDSK
ugyanaz, mint 03, de file-sorszám helyett a BL-ben megadott kódú lemezegység vezérlőcsatornáját írjuk. (0 - kurrens, 1 - A stb.)

E funkciókkal természetesen csak olyan eszközöket kezelhetünk, melyeknek létezik vezérlőcsatornája. Ezt előre le kell kérdezni; lásd 00 alfunkció.

06 - DOS_IOCTL_GINS
a handle input-státuszának lekérdezése. Válaszértékek AL-ben:

Válasz	perif. eszköz esetén	input file esetén	output file esetén
00	nem kész	file vége	írásra kész
FF	működéskész	olvasásra kész	írásra kész

07 - DOS_IOCTL_GOUTS
a handle output-státuszának lekérdezése

A státuszlekérdező alfunkciók megmondják, hogy az eszköz vagy file készen áll-e karakter(ek) beadására (06), illetve fogadására (07). Ha AL-ben FF-et kapunk vissza, akkor készen áll, ha pedig 0-t, akkor nem.

MS-DOS 3.00 és későbbi verziók

08 - DOS_IOCTL_DSKCH
lekérdezi, cserélhető-e a device (azaz floppy disk-e vagy sem). A válaszok:

AX	Jelentése
00	a lemez cserélhető (floppy)
01	a lemez nem cserélhető (merevl.)
0F	nem lemezegységet kérdeztünk le

MS-DOS 3.1 és későbbi verziók

09 - DOS_IOCTL_REDBK
Megvizsgálja, hogy a BL-ben megadott kódú lemez lokális-e a hálózatban, vagy egy távoli állomáson található.
Felhasználói programokból nem tanácsos használni. Egy ilyen programra nézve mindegy, hogy a lemez át van-e irányítva vagy sem.

MS-DOS 3.1 és későbbi verziók

0A - DOS_IOCTL_REDHD
Megvizsgálja, hogy a BX-ben megadott filesorszámmal kezelt file vagy perifériális eszköz lokális-e a hólózatban, vagy pedig átirányították-e egy távoli állomásra.
Felhasználói programokból nem tanácsos használni. Egy ilyen programra nézve mindegy, hogy a file át van-e irányítva vagy sem.

MS-DOS 3.00 és későbbi verziók

0B - DOS_IOCTL_RTRYC
azt szabhatjuk meg ezzel az alfunkcióval, hogy egy osztott elérésű file esetén hányszor és milyen időközönként ismételje meg a DOS a sikertelen elérési kísérletet, mielőtt meghívná a kritikus hiba-kezelő rutint (INT 24). DX-ben adhatjuk meg a végrehajtandó kísérletek számát, CX-ben pedig egy időtartamra utaló mennyiséget. Minél nagyobb értéket adunk meg, annál hosszabb lesz ez az idő, de legnagyobb a 0, mivel CX mint számláló szerepel. Alapértelmezések: a kísérletek száma három, a számláló értéke 1.

Ez a funkció, mint láthatjuk, elsősorban perifériális eszközök direkt vezérlésére szolgál. Alkalmas például az aszinkron vonal felprogramozására. A kisegítő eszköz (file-sorszáma 0003) vezérlőcsatornáját, melyen a csatoló működését közvetlenül programozni tudjuk, ezzel a funkcióval írhatjuk-olvashatjuk. Ugyanakkor lekérdezhetjük az eszköz státuszát, tehát közvetlenül vezérelhetjük a vonalat - időnként "ránézve" pl. az input státuszra, megtudhatjuk, van-e már beolvasható karakter a vonalon.

Hibakódok AX-ben Carry=1 esetén:

01 - érvénytelen alfunkciókód, vagy (0A esetén) a hálózati program (Microsoft Networks) program nem aktív

05 - (csak 04 és 05 esetén) az elérés tiltott, érvénytelen lemezkód
06 - érvénytelen file-sorszám - nemlétező vagy meg nem nyitott file-sorszám
0D - (csak 06 és 07 esetén) érvénytelen adatok
0F - (csak 08 és 09 esetén) érvénytelen lemezkód

Ismerkedjünk meg közelebbről azokkal a speciális eszköznevekkel, amelyekre a 00 és 01 alfunkció leírásánál hivatkoztunk!

A NUL eszköz belövési célokra alkalmazható. Amennyiben egy file-sorszámhoz a NUL eszközt rendeljük, akkor minden I/O művelet sikeres lesz. Beolvasás esetén azonnal EOF-ot generál, azaz úgy tesz, mintha rögtön elértük volna a file végét. Kiírás esetén pedig annyi byte sikeres kiírását "ismeri el", amennyit ki akartunk írni.

A CLOCK$ (óra) eszköz tulajdonképpen egy speciális driver, amely nem a DATE és TIME parancsok szintaxisa szerint kezeli a gép belső óráját. Akár beolvasást, akár kiírást hajtunk végre, mindig a következő hat byte átvitele következik be:

az 1980. jan. 1. óta eltelt napok száma (1 szó)	percek óra:00 óta	órák 0 óra óta	század-másod- percek	másod- percek

A standard input és output file-ok esetén az eszközleíró szó BINARY bitjének jelentése a következő:

Standard input:

Ha a BINARY bit 1, akkor a file-t binárisan kezeljük, tehát:
- a CTRL-S (a kiírás, az echo felfüggesztése), a CTRL-P (a standard output kiírásának printerre való másolása) és a CTRL-BREAK (a program kilövése) karaktereket nem teszteli;
- nincs nyomtatóra másolás;
- nincs echo a standard outputra;
- a megadott számú byte beolvasása után azonnal visszatér a beolvasó funkció.

Ha a BINARY bit 0, akkor pedig szövegfile-ként kezeljük, tehát:
- a CTRL-S, CTRL-P és CTRL-BREAK karaktereket a rendszer vizsgálja;

- a beolvasott karaktereket a standard output file-ra kiírja (echo);
- a beolvasás visszaadja a régebbről a bufferben maradt karaktereket, és előre mozdítja a file-pointert;
- a beolvasás az első sorhatároló karakterig tart. A begépelt szöveget a funkcióbillentyűkkel szerkeszthetjük;
- ha a kocsivissza-soremelés karakterek előtt egy CTRL-Z-t adunk meg, akkor a rendszer a CTRL-Z-t, a kocsivissza és a soremelés karaktereket egy soremeléssel helyettesíti.

Standard output:

Ha a BINARY bit 1, akkor a file-t binárisan kezeljük, tehát:
- a rendszer nem vizsgálja a CTRL-S, CTRL-P és CTRL-BREAK karakterek leütését;
- nincs nyomtatóra másolás;
- a megadott számú karaktert írja ki tekintet nélkül a kiírás közben érkező vezérlőkarakterekre;
- a vezérlőkaraktereket a "^" (kalap) karakter nélkül írja ki;
- a tabulátorkaraktereket nem helyettesíti szóközökkel.

Ha a BINARY bit 0, akkor pedig szövegfile-ként kezeljük, tehát:
- a rendszer megvizsgálja a CTRL-S, CTRL-P és CTRL-BREAK karakterek leütését, és végrehajtja a megfelelő parancsokat;
- szóközökkel helyettesíti a tabulátor karaktereket;
- a vezérlőkaraktereket direkt kiírásuk helyett a "^" jellel írja ki (pl. a 03 karaktert ^C-ként);
- a kiírást a felhasználó által megadott számú karakterig vagy az első CTRL-Z (kódja decimális 26) eléréséig folytatja.

VIII.3. Példaprogramok az MS-DOS file-kezeléshez

Az alábbiakban néhány olyan példaprogram következik, amely illusztrálja a fenti funkciók alkalmazását. Az előző fejezetben már bevált gyakorlatot fogjuk követni: létrehozunk egy olyan file-t, amely tartalmazza az összes konstans-, makró-, struktúra- és rekorddeklarációt. Ezúttal azonban ismét továbblépünk egy kicsit. A mostantól használt file neve DOSCALL.INC (teljes terjedelmében lásd a "DOSCALL.INC" függelékben), és nem csupán az itt leírtakhoz szükséges deklarációkat tartalmazza, hanem minden, a rendszerhívások során felmerülő konstansét, struktúráét, rekordét, makróét. Persze a "minden" relatív fogalom; csak annyit értünk alatta, hogy e file megfogalmazása közben

nem voltunk tekintettel az Olvasó memóriájának, befogadóképességének véges voltára, hanem irgalom nélkül összeszedtünk minden szükségesnek látszó deklarációt.

Soha többé nem adjuk meg programpéldában a rendszerkonstansok és egyebek deklarációját. Programjaink így lényegesen rövidebbek lesznek ugyan, de kicsit nehezebben olvashatók. E gyakorlat követése azonban mindenképpen helyes, mert kényelmesebb, biztonságosabb programozást tesz lehetővé.

VIII.3.1. Segédrutinok

Nem indokolt a hibaüzenetekkel a programban kínlódni úgy, hogy a visszatérés után vizsgálgatjuk a visszakapott kódot, és össze-vissza ugrálunk a különböző esetekben. Ehelyett kifejlesztünk egy hibakód-értelmező rutint. Ezt egyszerűen akkor hívjuk meg, ha egy modern DOS-funkció Carry=1-el, azaz hibakóddal tért vissza. A hibakód-értelmező rutin AL alapján kiír egy hibaüzenetet.

> Megjegyzés: ilyen rutinra eddig nem volt szükségünk, hiszen nem volt értelmes hibakód. A hiba okára nem a DOS által visszaadott értékből, hanem csakis előfordulásának helyéből következtethettünk. A modern funkciók esetén már sokkal többet tudhatunk a hiba okáról. Ez is e funkciók használata mellett szól a hagyományosakkal szemben!

A rutint PRTERRMSG-nek nevezzük el. Vizsgáljuk meg e rutin tevékenységét! Először meghívja a DOSCALL.INC file-ban definiált ERRMSG makrót (lásd "DOSCALL.INC" függelék). Ebben egyfelől a hibakódok, másfelől a hibaüzenetek deklarációja van, végül pedig egy cím- és hossztáblázat. Ez tartalmazza a definiált hibaüzenetek címét és hosszát, szavas formában. A rutin AL-ben átveszi a hibakódot, megszorozza kettővel, a kapott értékkel beleindexel a címtáblázatba, és kiveszi belőle a megfelelő hibaüzenet címét. Ezután kinyomtatja az üzenetet a standard outputra, és minden regisztert helyreállítva visszatér.

Az Olvasó kiemelt megjegyzésekkel találkozik az alábbi kódban. Ezek szövege: "EXE specifikus kód!". Ez azt jelenti, hogy az adott sorok azért szerepelnek, hogy a hibakezelő rutint EXE program részeként is használhassuk. Ott a nehézség az, hogy a

könyvtári rutin megírása során nem ismerjük a később írandó EXE programok adatszegmensének nevét. Sőt, ha egy programban több adatszegmens van, akkor azt sem tudhatjuk, hogy a rutin meghívásakor éppen melyik adatszegmens aktív. Ezért a hibaüzeneteket csakis a kódszegmensben helyezhetjük el. Emiatt (mentés után) át kell állítani a DS-t úgy, hogy az a CS által címzett (tehát pillanatnyilag aktív) kódszegmensre mutasson. Persze felmerülhet, hogy címezni a CS regiszterrel is lehet. Stringeket kinyomtatni azonban így nem tudunk; a rendszerhívások paraméterigénye kényszerít bennünket a DS szegmensregiszter felhasználására. Ezen és még néhány hasonló problémán érdemes elgondolkodnia annak, aki jól használható könyvtárat akar létrehozni.

Ezt a rutint az egyszerűség és a rövidebb kódolás kedvéért együtt használjuk az EXIT nevű makróval, amelyet a DOSCALL.INC file-ban talál az Olvasó. Ez a makró két cimkét deklarál, az ERR_EXIT-et és a SUCC_EXIT-et. Ezekre a programból rá tudunk ugrani. Az ERR_EXIT mögött a PRTERRMSG rutin hívása áll; ezután a TRM_DTERR kilépési kóddal ugrik rá a kiléptető funkció meghívására. A SUCC_EXIT mögött nics semmi ravaszság: egyszerű kilépés a TRM_SUCC kilépési kóddal. Befejezésül csak arra szeretnénk felhívni a figyelmet, hogy az INT 20 utasitást a további példákban már nem használjuk.

```
TITLE   Hibakód-értelmezés és -kiírás STDOUT-ra

        PUBLIC  PRTERRMSG       ;Ez lesz a belépési pont
;
; Konstansok, makrók, rekordok
;
        .XLIST                  ;DOSCALL.INC-et nem listázzuk!
        INCLUDE         DOSCALL.INC
        .LIST
;
; Kódszegmens
;
CODE    SEGMENT BYTE    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:CODE, ES:CODE, SS:CODE
;-------------------------------;
; PRTERRMSG     - hibakód értelmezése és kiírása
;       Input:  AL      - hibakód
;       Output: semmi
;       Minden regiszter mentve. Nincs külön adatterület!
```

```
PRTERRMSG       PROC    NEAR                ;
                PUSHF                       ;
                PUSH    DS                  ;EXE specifikus kód!
                PUSH    DX                  ;
                PUSH    BX                  ;
                PUSH    AX                  ;Mentés
                PUSH    CS                  ;EXE specifikus kód!
                POP     DS                  ;EXE specifikus kód!
                MOV     BL, AL              ;
                XOR     BH, BH              ;Hibakód BX-ben
                ROL     BX, 1               ;Szavas index
                MOV     DX, ERRMSG_TABLE[ BX ] ;Hibaüz.címe (*)
                DOSCALL DOS_STWRSTR         ;Hibaüzenet kiírása (*)
                POP     AX                  ;
                POP     BX                  ;
                POP     DX                  ;Visszatöltés
                POP     DS                  ;EXE specifikus kód!
                POPF                        ;
                RET                         ;
                                  ;-------;
        ERRMSG                    ;Az ERRMSG makró kifejtése
PRTERRMSG       ENDP              ;

CODE    ENDS
        END
```

A hibakódok kiírását a standard output file helyett a standard hibajelző file-ba is irányíthatnánk. Azért nem így cselekszünk, mert akkor elveszítjük az átirányíthatóságot és azt a lehetőséget, hogy a belövés ideje alatt megkapott hibaüzeneteket egy lemezfile-ba írhassuk ki. A kész programban azonban már káros lenne a komoly hibaüzeneteket a standard outputra küldeni, minden ilyet szigorúan a standard hibajelző file-ra írunk ki. Nagyon hasznos, hogy az egyszerű információs üzenetek és a komoly hibákról vagy fontos tevékenységekről hírt adók logikailag különböző helyre küldhetők. Ez is egyike a UNIX-ból származó igen rokonszenves sajátosságoknak!

A programot viszonylag egyszerűen módosíthatjuk úgy, hogy a hibajelző file-ba küldje ezeket az üzeneteket. A (*)-al jelölt sorok helyett az

```
ERRWRIT ERRMSG_TAB[ BX ], ERRMSG_SIZ_TAB[ BX ]
```

makróhívást kell alkalmaznunk (lásd "DOSCALL.INC" függelék).

Még egy fontos segédrutint kell megírnunk, ez pedig egy paraméter-értelmező rutin. Mint tudjuk, a felhasználó számára igen kényelmes, ha a programnak szóló parancsait hívási paraméterként adhatja meg. A paraméterek listája a PSP 80. (128.) byte-ján kezdődik a string karakterekben számított hosszával (lásd "Amikor a program belép" fejezet). Az alábbi rutin egy duplaszónyi struktúrába tölti le a soron következő paraméter címét, és magát a paramétert egy nulla byte-al zárja le a szóköz helyén. Hasonló feladatot lát el, mint a 29 (DOS_FNMINI) funkció, bár annál sokkal kevesebbet tud (pl. nem lépi át a terminátorokat stb.). Miért csináltuk meg mégis? Egyszerűen azért, mert a 29 funkció nem paramétereket, hanem file-neveket készít elő, és nem hajlandó a lemezek, valamint a directory-út kezelésére. A későbbi példaprogramokban pedig éppen az a lényeg, hogy directory-kat is kezelünk.

Bemenetként DS:DI címzi meg azt duplaszót, ahová a következő paraméter teljes címét (szegmens:offset alakban) le akarjuk tölteni, ES-ben pedig a PSP szegmenscímét. Figyeljük meg, hogy ebben a megfogalmazásban a rutin egyaránt alkalmas COM és EXE típusú file-okban való felhasználásra is. COM file-okban a DS és ES regiszterek ugyanoda mutatnak: a PSP kezdőcímére. Az EXE programok belépése után (amikor a paramétereket fel akarjuk dolgozni) DS már az adatszegmensre mutat, de ES általában még a PSP-re. A rutin használata tehát elég kényelmes.

A rutin működése a következő: van egy belső változója, a GETPAR_PNT, amelyet arra használ, hogy nyilvántartsa, hol jár a parancssoron belül. Ez a változó egy mutató. Fordítási időben 0-ra van előkészítve (nem szeretjük az ilyen trükköket, de néha nagyon kényelmesek). Az első belépéskor innen ismeri fel, hogy még egyetlen paramétert sem kérdeztek le. Ekkor előkészíti a mutatót, hogy az első értékes karakterre mutasson. Ezután átlépi az esetleges kezdő szóközöket és tabulátorokat. Az első látható karakter címét adja vissza a duplaszóban. Végül pedig addig viszi előre a mutatót, míg el nem éri a paramétert követő első szóközt vagy tabulátort, és azt egy 0-val helyettesíti, hogy lezárja a paramétert. Amennyiben a paramétert lezáró karakter egy kocsivissza, akkor a mutatót -1-re állítja (ezzel jegyzi meg, hogy nincs több paraméter). Minden sikeresen elkülönített paraméter után Carry=0-val tér vissza; az utolsó paraméter elérésekor Zero=1-el jelzi, hogy a mostani keresés még sikeres volt ugyan, de ez volt az utolsó paraméter. Minden további hívásra már Carry=1-el mint hibajelzéssel tér vissza. (Ha

a belépéskor a mutató eleve -1, akkor az előző hívás már levá-lasztotta és visszaadta az utolsó paramétert.)

Egy alrutint is definiáltunk, melynek neve WHITE_TST. Ez az AL-ben kapott karaktert ellenőrzi, hogy nem egyezik-e meg vala-melyik láthatatlan karakterrel. Ha igen, akkor Zero=1-el tér vissza. Amennyiben emellett még sorhatároló jel is (kocsivissza vagy soremelés vagy lapdobás), akkor beállítja a Carry flaget is. Ez a kis rutin önmagában is hasznos lehet.

> Megjegyzés: az ismételt SCASB utasítás segítségével flexibilisebben is megírhattuk volna ezt a rutint, de (próbálják csak ki) ilyen kevés karakter esetén ez kódban is, futásidőben is hosszabb lenne!
> A programozás során gyakran kötünk magunkkal ilyen kompromisszumot. A lehető legnagyobb flexibilitás megvalósítása igen fontos követelmény, de az egysze-rűség, tömörség és gyorsaság is az. Emlékezzünk csak az első kötet egy kitételére, mely szerint a progra-mozás elveinek még az ellenkezője sem igaz; nem sza-bad nagyon mereven ragaszkodni a néha öncélú szépség-hez, mert ennek az elkészült program minősége láthat-ja kárát.

```
TITLE   Paraméterfeldolgozó rutin

        PUBLIC  GETPAR          ;Ez lesz a belépési pont
;
; Konstansok, makrók, rekordok
;
        .XLIST                  ;DOSCALL.INC-et nem listázzuk!
        INCLUDE         DOSCALL.INC
        .LIST
;
; Vezérlő konstansok
;

IND_NOPAR       EQU     -1      ;Jelzi, hogy elfogyott

CODE    SEGMENT BYTE    PUBLIC  'CODE'

        ASSUME  CS:CODE, DS:NOTHING, SS:NOTHING, ES:NOTHING
                                ;Nem tudjuk a szegmensneveket
```

```
;-----------------------------------;
; GETPAR      - futtatási paraméter előállítása
;       egy hosszú mutató előkészítése a PSP-ben megadott
;       következő paraméterre, plusz a paraméter elkülönítése,
;       azaz nulla karakterrel való lezárása
;       Input:
;               DS:DI   - a mutató (duplaszó) címe
;               ES      - a PSP címe
;       Output:
;               Carry   - 1  ha nincs több paraméter
;               Carry   - 0  ha van még
;                       Zero - 1  ha ez az utolsó, 0 különben
;                       [ DI ]  - pointer a következő param.-re
;                       CX      - a paraméter hossza byte-okban
;       Belső változók:
;       GETPAR_PNT      - mutató a parancssor köv. karakterére
;
;-----------------------------------;
                                    ;
GETPAR          PROC    NEAR        ;
                PUSH    AX          ;
                PUSH    BX          ;
                PUSH    SI          ;Regiszterek mentése
                XOR     SI, SI      ;Offs. PSP kezeléshez
                XOR     CX, CX      ;
                MOV     BX, CS:GETPAR_PNT
                TEST    BX, BX      ;Pointer előkészitett?
                JS      GETPAR_ERROR    ;-1, nincs több par.
                JNE     GETPAR_PROC     ;Poz., előkészitve
GETPAR_INIT:                        ;0, nincs előkészitve
                MOV     BX, PSP_CMDLIN  ;Pointer előkész. BX-be
GETPAR_PROC:                        ;
GETPAR_LOOP1:                       ;Láthatatlanok átlépése
                MOV     AL, ES:[ BX ][ SI ]
                CALL    WHITE_TST       ;Látható karakter?
                JNE     GETPAR_SETPNT   ;Z=0 - látható
                JC      GETPAR_END_NOPAR    ;C=1, sorvég volt
                INC     BX              ;Z=1 - nem látható
                JMP     GETPAR_LOOP1    ;Első láthatóig folyt.
GETPAR_SETPNT:                      ;Megvan a par. kezdete
                MOV     OFFS.[ DI ], BX ;Pointer beállitása a
                MOV     SEGM.[ DI ], ES ; paraméter kezdetére
```

```
GETPAR_READPAR:                         ;Par. végének keresése
                INC     CX              ;Számoljuk a hosszát
                INC     BX              ;
                MOV     AL, ES:[ BX ][ SI ]  ;Köv. kar AL-be
                CALL    WHITE_TST       ;Láthatatlan?
                JNE     GETPAR_READPAR  ;Folyatjuk, míg nem az
                MOV     BYTE PTR ES:[BX][SI], CHR_EOS   ;Lezár.
                INC     BX              ;Zárókarakter átlépése
                MOV     CS:GETPAR_PNT, BX    ;Módosított p. ki
                JNC     GETPAR_SUCC     ;Ha C=0, kész, van még
                MOV     CS:GETPAR_PNT, IND_NOPAR        ;Nincs
                CLC                     ;
                JMP     GETPAR_QUIT     ;Z=1! jelzi az utolsót
GETPAR_END_NOPAR:                       ;Vége és nincs több
                MOV     CS:GETPAR_PNT, IND_NOPAR
GETPAR_ERROR:                           ;
                TEST    CX, CX          ;Z és C beál-
                STC                     ; lítása
                JMP     GETPAR_QUIT     ;
GETPAR_SUCC:                            ;Sikeres, van paraméter
                TEST    CX, CX          ;Z és C törlése
GETPAR_QUIT:                            ;
                POP     SI              ;
                POP     BX              ;
                POP     AX              ;Regiszterek helyreáll.
                RET                     ;
                                        ;
GETPAR_PNT      DW      0               ;Belső munkapointer
                                        ;
GETPAR          ENDP                    ;
                                        ;
;---------------------------------------;
; WHITE_TST     - megvizsgálja, hogy AL látható karakter-e
;       Input:
;               AL      - a karakter kódja
;       Output:
;               ZERO    - 0, ha a karakter nem láthatatlan
;               ZERO    - 1, ha a karakter láthathatlan
;                         (BLANK, TAB, CR, LF, FF, 0)
;               CARRY   - 1, ha egy kocsivezérlő karakter
;                         (CR, LF, FF, 0)
;---------------------------------------;
```

```
WHITE_TST       PROC    NEAR                    ;
                                                ;
                CMP     AL, CHR_BLANK           ;BLANK?
                JE      WHITESPACE              ;
                CMP     AL, CHR_TAB             ;TAB?
                JE      WHITESPACE              ;
                CMP     AL, CHR_CR              ;Kocsivissza?
                JE      CARRIAGECONT            ;
                CMP     AL, CHR_LF              ;Soremelés?
                JE      CARRIAGECONT            ;
                CMP     AL, CHR_FF              ;Lapdobás?
                JE      CARRIAGECONT            ;
                TEST    AL, AL                  ;Zérókarakter?
                JE      CARRIAGECONT            ;
WHITESPACE:                                     ;
                CLC                             ;BLANK vagy TAB; Z=1!!
                JMP     WHITE_TST_QUIT          ;
CARRIAGECONT:                                   ;Kocsivez.; CARRY beáll.
                STC                             ;
WHITE_TST_QUIT:                                 ;
                RET                             ;
                                                ;
WHITE_TST       ENDP                            ;
                                                ;
;-----------------------------------------------;
;

CODE    ENDS

        END
```

A fenti kis programokat önállóan fordítjuk le. Ezután kettőt tehetünk velük. Az egyik, hogy minden olyan programunkhoz hozzászerkesztjük object változatukat, amelyben használni akarjuk őket; a másik, hogy létrehozunk egy felhasználói könyvtárt a LIB.EXE programmal, és ezt szerkesztjük minden programunkhoz. Mondanom sem kell, ez utóbbi a profi megoldás. A LIB.EXE program felhasználásával, a könyvtárak szervezésének szempontjaival a harmadik kötetben foglalkozunk bővebben. Itt illusztrációképpen csak a legszükségesebb parancsokat soroljuk fel. Tegyük fel, hogy a fenti programok a DOS_ERR.ASM és a GETPAR.ASM forrásnyelvű file-okban vannak.

```
C>MASM DOS_ERR;                          DOS_ERR fordítása
C>MASM GETPAR;                           GETPAR fordítása
C>LIB                                    Hívjuk a LIB-et
Microsoft Library Manager
Version 3.00 (C) Copyright Microsoft Corp 1983, 1984, 1985

Libary name: ownrut.lib                  Könyvtár neve
Library does not exist. Create? y        Létrehozzuk őt
Operations: +dos_err +getpar             Befűzzük a rutinokat
List file: ownrut.ext                    Listát is kérünk róla
C>
```

Ezzel a parancssorozattal máris létrehoztunk egy könyvtárat, melyet a LINK negyedik kérdésére, vagy negyedik paramétereként adunk majd meg. További modulokat (object-file-okat) hasonlóképpen fűzhetünk be a könyvtárba. Modulok törlése a "-" (mínusz) paranccsal lehetséges.

VIII.3.2. Egy precíz file-másoló program

Az alábbiakban olvasható program végre megvalósítja régi vágyunkat, egy file teljesen precíz átmásolását. Mint láttuk, az e fejezet előtt ismertetett funkciók nem alkalmasak erre a feladatra, mivel olyan megkötésekkel terhesek, amelyek nem látszanak nagyon súlyosaknak (pl. rögzített rekordhossz), de lehetetlenné teszik egy ilyen nagyon egyszerű probléma megoldását.

```
TITLE   File másolása file handle segítségével
;
; Külső objektumok
;
        EXTRN   GETPAR:NEAR, PRTERRMSG:NEAR
;
; Konstansok, makrók, rekordok
;
        .XLIST                      ;DOSCALL.INC-et nem listázzuk!
        INCLUDE         DOSCALL.INC
        .LIST
SIZ_BLOCK       EQU     512
CR_ATTRIB       EQU     ATR_ARCHV       ;
```

```
;
; Kódszegmens
;
CODE    SEGMENT PARA    PUBLIC  'CODE'

        ASSUME  CS:CODE, DS:CODE, ES:CODE, SS:CODE
        ORG     100H
ENTRY:                                  ;
                JMP     START           ;
                                        ;
COPY_BUFF       DW      SIZ_BLOCK  DUP ( ? )
OLD_HND         DW      0               ;Régi file-sorszám
NEW_HND         DW      0               ;Másolt file-sorszám
                                        ;
OLD_PNT         LONGPNT         <>      ;Régi file-sp. címe
NEW_PNT         LONGPNT         <>      ;Másolt file-sp. címe
                                        ;
REQ_BYTES       DW      0               ;Munkaváltozók: az
READ_BYTES      DW      0               ; olvasandó és olvasott
                                        ; byte-ok száma
START:                                  ;
                MOV     DI, OFFSET OLD_PNT
                CALL    GETPAR          ;Első a régi file-sp.
                HNDCTL  DOS_OPEN, OLD_PNT.OFFS
                JC      ERR_EXIT        ;Régi file megnyitása
                MOV     OLD_HND, AX     ;File handle mentése
                MOV     DI, OFFSET NEW_PNT
                CALL    GETPAR          ;Második az új file-sp.
                MOV     CX, CR_ATTRIB   ;Az új file attributuma
                HNDCTL  DOS_CREAT, NEW_PNT.OFFS
                JC      ERR_EXIT        ;Az új file megnyitása
                MOV     NEW_HND, AX     ;File handle mentése
                MOV     REQ_BYTES, SIZ_BLOCK
COPY_LOOP:                              ;
                READ_H  OFFSET! COPY_BUFF, REQ_BYTES, OLD_HND
                JC      ERR_EXIT        ;Köv. 512 byte be
                MOV     READ_BYTES, AX  ;Olv. byte-ok száma
                WRIT_H  OFFSET! COPY_BUFF, AX, NEW_HND
                JC      ERR_EXIT        ;Kiírjuk a byte-okat
                MOV     AX, READ_BYTES  ;
                CMP     AX, REQ_BYTES   ;Tört rekord volt?
                JE      COPY_LOOP       ;Ismétlés, míg van adat
```

```
CLOSE:                                  ;
                HNDCLL  DOS_CLOSE, OLD_HND
                JC      ERR_EXIT        ;Régi file-t lezárjuk
                HNDCLL  DOS_CLOSE, NEW_HND
                JNC     SUCC_EXIT       ;Az új file-t lezárjuk
                                        ;
                EXIT                    ;Kilépés makróval
CODE    ENDS                            ;
        END     ENTRY
```

A fenti program nem kíván semmilyen bonyolult magyarázatot. Annyit figyeljen meg a makrók használatában járatlan Olvasó, hogy a READ_H és a WRIT_H makrókat (melyek csaknem betűre azonosak - lásd "DOSCALL.INC" függelék) változatos címzési módú paraméterekkel használtuk. A MASM egyik nagy hibája az, hogy a makró paramétereinek határolására a szóközt is elfogadja, nem csak a ","-t. Ezért, ha valamelyik paraméterünk helyére egy összetett kifejezést kell írnunk (pl.OFFSET COPY_BUFF), akkor kénytelenek vagyunk vagy beírni az OFFSET kulcsszót a makróba (ezzel erősen megkötve saját kezünket a címzési módok használatát illetően), vagy pedig a "!" operátort használni azért, hogy a kétszavas paramétert elválasztó szóköz mint szóköz és ne mint terminátor számítson. Ezt a kis trükköt érdemes talán megjegyezni. Figyeljük meg azt is, hogy a READ_H és WRIT_H makrókban a második paraméter helyén egyszer memóriaváltozót, egyszer regisztert használtunk. Itt látszik, milyen kényelmes is a makró.

Maga a program egy kívülről vezérelt végtelen ciklusban dolgozik; addig olvas, amíg csak el nem éri az első tört "rekordot", amikor a file-ból már nem sikerült 512 byte-ot beolvasni. Itt csak annyit kell átgondolni, hogy mi lesz, ha a file hossza osztható 512-vel, és utoljára egy 0 byte hosszúságú csomagot olvastunk be. Tudjuk, hogy ha CX-be kiírás során 0-t teszünk, az a file hosszának módosítását jelenti. Ebben az esetben ez nem is baj, hiszen a File Pointer az utolsó kiírt byte mögé mutat - pontosan a file végére.

Ennél nagyobb gondnak látszik az az eset, mikor a kiírásra kapunk az AX-ben 0-t válaszul (ez azt jelenti, hogy betelt a lemez). Az Olvasóra bízom annak eldöntését, mit is tesz ilyenkor: törli a részben átmásolt file-t, vagy lezárja ott, ahol éppen tart - ez felfogás kérdése. A rendszer COPY parancsa igen logikusan az elsőt választja (hiszen egy részben átmásolt bináris file semmire se jó).

VIII.3.3. Egy pici "adatbázis" kezelése

Az itt olvasható program első nagyobb méretű és bonyolultságú közös vállalkozásunk. Az "adatbázis"-kezelő program fő célja a direkt file-elérést illusztrálása. Azonban a programvezérléssel kapcsolatban a file-ok megosztott elérésére mutat példát.

Adatbázisunk személyek adatainak nyilvántartására szolgál. Azonos szerkezetű és hosszúságú rekordokból áll, kivéve a 0. rekordot, amely felépítésében eltér, és lényegesen rövidebb, mint a hasznos rekordok. (Nézze el az Olvasó, hogy az adatbázis rekordjainak belső szerkezete rögzített és nagyon egyszerű; nem írhattuk meg példaprogramként a "dBASE III PLUS"-t). A 0. rekord az ún. header, amely az adatbázis állapotát írja le: hány hasznos rekordot tartalmaz, és melyik ezek közül a kurrens rekord. Megjegyezzük, hogy a kurrens rekord száma csak egy esetleges későbbi felhasználás céljára van itt; lehetővé tenné pl. egy adatrekord módosítását. Az adatrekordok öt elemet tartalmaznak: a személy nevét, személyi számát, lakcímét, munkahelyi és otthoni telefonszámát. Ezeket az adatokat adott hosszúságú ASCII-stringekben tartjuk nyilván. A fel nem használt karakterek nullák. A rekord végén egy kocsivissza-soremelés pár lesz. Ezzel a trükkel elérjük, hogy az MS-DOS SORT programjával rendezhetjük az adatbázist (ami megvan, minek írjuk meg?). Viszont gondoskodni kell arról, hogy a header-rekord mindig előre kerüljön. Ezt úgy érjük el, hogy tíz szóköz-karakterrel kezdjük el (erre egyébként semmi szükség nem lenne). Az adatbázis rendezésére azért van szükség, mert az adatokat intervallumfelezéssel akarjuk keresni. Ez igen gyors eljárás, és adatstruktúránk segítségével elég kényelmesen megoldhatjuk.

Az adatbáziskezelő két programból áll (és felhasználja harmadikként a SORT segédprogramot). A főprogram megnyitja az adatbázist, melynek neve az egyszerűség kedvéért be van drótozva a szövegbe: PERSONAL.DTB (javítási lehetőség!). Ha nem létezik ilyen nevű file, akkor létrehozza azt. Ezután egy négysoros menüt ír ki, ahonnan a felhasználó választhat: új elemet iktathat be, rendezheti az adatbázist, megkeresheti egy személy adatait, vagy kiléphet.

Az új elem befűzését a főprogram végzi, az egyéb tevékenységekhez segédprogramokat használ. A felhasználó kívánságára meghívja a SORT.EXE programot, hogy az adatbázis rekordjait rendezze, vagy "magára húzza" az alprogramot (melynek file-neve DBC-2.EXE).

Az alprogram olvasásra megnyitja az adatbázist, kér egy személynevet, és megkeresi az ilyen nevű személy adatait az adatbázisban. Ha megtalálta, akkor kiírja az egész rekordot, ha nem, akkor egy hibaüzenetet ír ki, és vár az első ENTER-re. Ha ezt megkapja, akkor visszatér a főprogramhoz.

Ezután ismét a főprogram várja a felhasználói parancsokat.

Ismerkedjünk meg részletesebben a főprogram elemeivel. Ez már csak azért is hasznos lesz, mert példát láthatunk egy viszonylag nagyobb lélegzetű és összetettebb program ésszerű szegmentálására, rutinokra bontására. Ezt minden programozó másképpen csinálja; vizsgáljuk meg az alábbi megoldást, és próbáljunk javítani rajta (igen sok helyen lehet).

Első lépés a külső szimbólumok deklarációja. Ezek mind szubrutinok, melyeket egy könyvtárból vagy más object file-ból szerkesztünk programunkhoz. Az ismert rutinok mellett találunk egy újat, melyet e fejezet végén adunk majd meg: az STRCOPYC rutint, amely egy stringet másol az első vezérlőkarakterig. Ez az adatbázis elemeinek szerkesztéséhez szükséges. Felhasználjuk a "Standard file-ok kezelése" fejezetben megismert CLRSTINBUF rutint is.

Ezután include utasítások következnek. Itt csak arra kell figyelni, hogy két file-unk van: az általános DOSCALL.INC a rendszerkonstansok deklarációjával, és a DBC.INC, amely az adatbázis kezeléséhez szükséges konstans- és struktúradeklarációkat tartalmazza (lásd e fejezet végén). Tehát különválasztjuk az általános és a feladatspecifikus deklarációkat.

Ezután néhány többé-kevésbé ravasz makró deklarációja következik. Az ANSI-driver kihasználása kedvéért a 09 funkciót választjuk a szövegek kiírására. Egyes szövegek előtt pedig megadjuk az "ESC [2J" karaktersorozatot, amely törli az ernyőt, és a cursort a bal felső sarokba viszi. Ezzel sok munkát takarítunk meg.

Figyeljük meg az "&" operátor használatát az EDMSG és az EDIT makrókban, valamint azt, hogy az adatbázis-elemek editálását lényegében már a makrókkal megoldottuk.

A program talán leglényegesebb része az adatterületek definíciója. Egy jól megszabott adatstruktúra megkönnyíti a program megírását és kiteljesítését, míg a rosszul megválasztott adatszerkezet egy programot végzetesen tönkretehet. Itt elég egyszerűen megadott szerkezetekkel dolgozunk. Nincs semmi meglepő, hacsak az ERR_IND változó nem. Ennek szerepe a program belső szerkezetének megismeréséből fog megvilágosodni.

A program számos szubrutint tartalmaz, amelyek különböző "szinteken" dolgoznak. Maga a program tulajdonképpen egy rutin, amely "magasszintű" rutinokat hív. Ezek alacsonyabb szintű rutinokat hívnak és így tovább. A főrutin nagyon rövid, mindössze néhány rutinhivásból áll. A már megszokott előkészítés (PSP elmentése, a memória felszabadítása) után megnyitja vagy létrehozza az adatbázist. Az előkészítésben csak az lehet érdekes, hogy elmentjük a DOS-tól kapott környezet kezdőcímét, hogy átadhassuk a meghívott segédprogramoknak a DOS környezetét; ki tudja, hátha kell nekik valamire.

Megjegyzés: a COMMAND.COM nagyon is igényli az eredeti környezetet, amelynek PATH kezdetű sora szabja meg, hogy mely aldirectory-kban kell keresni az elindítandó programokat.

Ezután egy ravasz rutin következik: a MENU_SEL. Kiírja a négysoros menüt, és beolvassa, ellenőrzi és "dekódolja" a felhasználó válaszát. Minden alfunkciót egy-egy szubrutin hajt végre; MENU_SEL a kiválasztott funkciót végrehajtó rutin címét adja vissza BX-ben. A főprogram ezután egy CALL BX utasítással indítja el a kért műveletet elvégző szubrutint.

Ezek a rutinok mind egyenrangúak. Protokolljuk a következő: kötelesek Carry=0-val kilépni, kivéve a program befejezését előkészítő rutint. A főprogram innen tudja, hogy abbahagyhatja a kiválasztás és végrehajtás ismétlését. Ha a rutinok valamilyen DOS-hibával találkoznak, akkor az ERR_IND változóba teszik a hibakódot. A főprogram ennek ismeretében dönthet, hogy folytathatja-e a munkát vagy sem.

Talán egyetlen rutin érdekes némileg: az SRT_DBS rutin. Miért nem hívhatjuk meg egyenesen a SORT.EXE programot, valóban szükséges a COMMAND.COM egy újabb verziójának elindítása? Mert sajnos nem egyszerű úgy meghívni a SORT-ot programból, hogy elfogadja az átirányításokat. Gondoljuk át, hogy a standard input és output file-ok öröklődnek, tehát a meghívott program standard intputja és outputja megegyezik a hívóéval. Ez a módszer azonban eredményre vezetett, hiszen az újabb COMMAND.COM újabb standard file-okkal rendelkezik. Ezt az eljárást érdemes megjegyezni, mivel hasonló feladatot gyakran kell megoldanunk.

A további rutinok már nem tartogatnak különösebb meglepetéseket. Figyelmesen olvassuk el őket, és tegyük magunkévá (vagy komoly érvekkel vessük el) azt a koncepciót, amelynek alapján a feladatokat megosztottuk az egyes rutinok között.

```
TITLE   Adatbázis-vezérlés, létrehozás, bővítés

        EXTRN   CUT_EXE:NEAR, PRTERRMSG:NEAR
        EXTRN   STRCOPYC:NEAR
        EXTRN   CLRSTINBUF:NEAR

;
; Konstansok, makrók, rekordok
;
        .XLIST                      ;DOSCALL.INC-et nem listázzuk!
        INCLUDE         DOSCALL.INC
        .LIST

        INCLUDE         DBC.INC

;-----------------------------------------------;
;
; Makrók
;
; INSTR - megadott hosszúságú string beolvasása
;
;-----------------------------------------------;

INSTR   MACRO   SIZE
                MOV     IN_BUFF_SIZE, SIZE      ;;String hossza
                MOV     DX, OFFSET IN_BUFF_ADDR ;;Buffer címe
                DOSCALL DOS_STRDSTR             ;;
        ENDM

;-----------------------------------------------;
;
; EDMSG - szerkesztési üzenet definíciója
;
;-----------------------------------------------;

EDMSG   MACRO   EL, MESS
DB_ED_&EL       DB      CHR_ESC, '[2J'          ;;Ernyőtörlés
                DB      4  DUP  ( CHR_LF )      ;;4 x soremelés
                DB      MESS                    ;;Az üzenet
                DB      '$'                     ;;Dollárjel
        SIZ_DB_ED_&EL   EQU     $ - DB_ED_&EL   ;;Az üz. hossza
        ENDM                                    ;
```

```
;-------------------------------------------------;
; EDIT - elem editálása DOS-funkciók segítségével
;-------------------------------------------------;
                                                  ;
EDIT    MACRO   EL                                ;
                MOV     DX, OFFSET DB_ED_&EL      ;;Elemcím DX-be
                DOSCALL DOS_STWRSTR               ;;Kiírjuk
                CALL    CLRSTINBUF                ;;Buffertörlés
                INSTR   SIZ_&EL + 1               ;;Elem beolv.
                MOV     SI, OFFSET IN_BUFF        ;;
                MOV     DI, OFFSET DB_ELEMENT.D_&EL
                CALL    STRCOPYC                  ;;Bemásoljuk
        ENDM                                      ;
                                                  ;
;-------------------------------------------------;
; Adatterületek
;-------------------------------------------------;

DATA    SEGMENT BYTE    PUBLIC  'DATA'

                ; Adatbázis-kezelő változók
                ;-------------------------;
                                          ;
DB_PATH         DB      'C:PERSONAL.DTB', 0
DB_ACCESS       DB      DOS_OPEN_RW + DOS_OPEN_DNWRIT
DB_HND          DW      0                 ;File handle
ERR_IND         DB      0                 ;Hibajelző (0 - 1)
                                          ;
                ;-------------------------;
                ; Adatbázis-bufferek
                ;-------------------------;
                                          ;
DB_HEADER       DB_HDR  <>                ;Header struktúra
DB_ELEMENT      DB_ELM  <>                ;Kurrens elem
                                          ;
                ;-------------------------;
                ; Beolvasó buffer
                ;-------------------------;
                                          ;
IN_BUFF_ADDR    LABEL   BYTE              ;Buffer neve
IN_BUFF_SIZE    DB      0, 0              ;Hossz + válasz hossza
IN_BUFF         DB      SIZ_BUFF DUP (0);Buffer
```

```
                ;------------------------;
                ; Kiírandó üzenetek
                ;------------------------;
                                         ;
MSG_MENU        DB      CHR_ESC, '[2J'   ;Ernyő törlése, plusz
 DB             CHR_CR                   ;
 DB  10 DUP     (CHR_LF)                 ;menü az ernyő közepére
 DB  CHR_TAB, CHR_TAB, '(1) Egy új elem befűzése'
 DB  CHR_CR,  CHR_LF
 DB  CHR_TAB, CHR_TAB, '(2) Az adatbázis rendezése'
 DB  CHR_CR,  CHR_LF
 DB  CHR_TAB, CHR_TAB, '(3) Keresés  név szerint'
 DB  CHR_CR,  CHR_LF
 DB  CHR_TAB, CHR_TAB, '(4) Kilépés az adatbáziskezelőből'
 DB  CHR_CR,  CHR_LF
 DB  CHR_TAB, CHR_TAB
 DB  '$'
       SIZ_MSG_MENU    EQU     $ - MSG_MENU
                                         ;
                ;------------------------;
                ; Editálási szövegek
                ;------------------------;
                                         ;
       EDMSG    NAM, 'A személy neve (25): '
       EDMSG    PNM, 'Személyi száma (11): '
       EDMSG    ADR  'Lakcíme (43): '
       EDMSG    PHO, 'Hivatali telefonszáma (12): '
       EDMSG    PHP, 'Otthoni telefonszáma (12): '

                ;------------------------;
                ; Segéd- és alprogram-indítási blokkok
                ; A SORT indításához
                ;------------------------;
                                         ;
PATH_SORT       DB 'C:\COMMAND.COM', 0   ;Program path-a !
EX_SORT         EX_BLK  <>               ;Paraméterblokk
                                         ;
CMDL_SORT       DB      SIZ_CMDL_SORT    ;Parancssor hossza
                DB      ' /C SORT <PERSONAL.DTB >PERSONAL.DTB'
       SIZ_CMDL_SORT    EQU     $ - CMDL_SORT
                DB      CHR_CR, 0        ;CR nem számít bele!
                                         ;
```

```
                ;------------------------;
                ; A DBC-2 indításához
                ;------------------------;
PATH_SEARCH     DB      'DBC-2.EXE', 0  ;Program path-a
EX_SEARCH       EX_BLK  <>              ;Paraméterblokk
                                        ;
CMDL_SEARCH   DB      SIZ_CMDL_SEARCH   ;Parancssor (üres)
              DB      ' '               ;
      SIZ_CMDL_SEARCH EQU     $ - CMDL_SEARCH
              DB      CHR_CR, 0

DATA    ENDS

;---------------------------------------;
; Stack-terület
;---------------------------------------;

STACK   SEGMENT PARA    STACK   'STACK'
        DW      100     DUP ( ? )       ;Hossza 100 szó
STACK   ENDS

;---------------------------------------;
; Kódszegmens, benne belső változók
;-------------------------------;

CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:DATA, SS:STACK, ES:DATA
PSP_SEG         DW      0       ;Területek a PSP címének,
PRG_ENV         DW      0       ; a program környezete címének,
SAVE_SS         DW      0       ; a Stack Segment és
SAVE_SP         DW      0       ; a Stack Pointer mentésére
                                ;
START:                          ;-------;
                MOV     CS:PSP_SEG, DS  ;
                XOR     SI, SI          ;Environment offsetje
                MOV     AX, PSP_ENVSEG[ SI ]
                MOV     CS:PRG_ENV, AX  ;Elmentjük
                MOV     AX, DATA        ;
                MOV     DS, AX          ;DS előkészítése
                CALL    CUT_EXE         ;Memóriafelszabadítás
                PUSH    DS              ;
                POP     ES              ;ES -> DATA szegmens
```

```
                CALL    DB_OPEN         ;Adatbázis ny./létreh.
                JC      ERR_EXIT        ;Nagy baj van
DB_LOOP:                                ;
                CALL    MENU_SEL        ;Menü + parancsbeolv.
                CALL    BX              ;Parancs végrehajtása
                JC      DB_QUIT         ;Kilépési parancs volt
                MOV     AL, ERR_IND     ;Hiba ellenőrzése
                TEST    AL, AL          ;Volt valami hiba?
                JZ      DB_LOOP         ;Ha 0, nem
                CALL    PRTERRMSG       ;Hibaüzenet ki
                MOV     ERR_IND, 0      ;Elfelejtjük, nem volt
                JMP     DB_LOOP         ; végzetes
DB_QUIT:                                ;
                MOV     AL, ERR_IND     ;Hibavizsgálat
                TEST    AL, AL          ;Lezárás sikeres volt?
                JZ      SUCC_EXIT       ;
                EXIT                    ;Kilépés
                                        ;
;---------------------------------------;
; MENU_SEL      - menü kiírása és választás a menüből
;       Input:
;               semmi
;       Output:
;               BX      - a kért rutin címe
;---------------------------------------;
MENU_SEL        PROC    NEAR            ;
MENU_SEL_LOOP:                          ;
                CALL    CLRSTINBUF      ;Input buffer törlése
                MOV     DX, OFFSET MSG_MENU
                DOSCALL DOS_STWRSTR     ;Menü kiírása
                INSTR   SIZ_CMD         ;Parancs beolvasása
                MOV     AL, IN_BUFF     ;Parancskód AL-be
                SUB     AL, '0'         ;Konverzió
                JLE     MENU_SEL_LOOP   ;Hibás választás (<=0)
                DEC     AL              ;Index 0-3, pcs. 1-4!
                CMP     AL, NUM_FUNCT   ;Nem túl nagy?
                JGE     MENU_SEL_LOOP   ;Hibás választás (>4)
                MOV     BL, AL          ;
                XOR     BH, BH          ;Indexelés előkészítése
                SHL     BX, 1           ;Szorzás 2-vel, szavas
                MOV     BX, CS:SEL_TABLE[ BX ]  ;BX <- rutincím
                RET                     ;
```

```
                ;-------------------------;
                ;Címtáblázat
                ;-------------------------;
SEL_TABLE       DW      NEW_ELEMENT     ;Egy új elem beírása
                DW      SRT_DBS         ;Az adatbázis rendezése
                DW      SRC_ELEMENT     ;Elem keresése
                DW      PROG_EXIT       ;Kilépés
                                        ;
MENU_SEL        ENDP                    ;
                                        ;
;---------------------------------------;
; NEW_ELEMENT   - új elem befűzése az adatbázisba
;---------------------------------------;
                                        ;
NEW_ELEMENT     PROC    NEAR            ;
                                        ;
                CALL    EDIT_ELEMENT    ;Beolv. az új elemet
                CALL    WR_LAST_EL      ;Kiírjuk az adatbázisba
                JC      NEW_ELEMENT_ERR ;Hiba - lekezeljük
                INC     DB_HEADER.H_DBN ;Elemszám növelése
                CALL    WR_HEADER       ;Header frissítése
                JNC     NEW_ELEMENT_SUCC;Sikeres - visszatérünk
NEW_ELEMENT_ERR:                        ;
                MOV     ERR_IND, AL     ;Hibakód mentése
NEW_ELEMENT_SUCC:                       ;Visszatérünk
                CLC                     ;Nem volt kilépés
NEW_ELEMENT_RET:                        ;
                RET                     ;
                                        ;
NEW_ELEMENT     ENDP                    ;
                                        ;
;---------------------------------------;
; SRT_DBS       - rendezi az adatbázist
;---------------------------------------;
                                        ;
SRT_DBS PROC    NEAR                    ;
                                        ;
                CALL    DB_CLOSE        ;Előtte le kell zárni
                                        ; az adatbázist!
                JC      SRT_DBS_ERR     ;Nem sikerült, baj van
                MOV     BX, OFFSET EX_SORT
                MOV     AX, CS:PRG_ENV  ;Eredeti ENV szegmense
```

```
                MOV     EX_S_ENV.[ BX ], AX
                MOV     AX, OFFSET CMDL_SORT
                MOV     EX_P_PAR.[ BX + OFFS ], AX
                MOV     EX_P_PAR.[ BX + SEGM ], DS
                MOV     DX, OFFSET PATH_SORT
                MOV     CS:SAVE_SS, SS  ;Stack Segment és
                MOV     CS:SAVE_SP, SP  ; Pointer mentése
                DOSCALL DOS_EXEC, DOS_EXEC_PRC
                MOV     SS, CS:SAVE_SS  ;Stack Segment és
                MOV     SP, CS:SAVE_SP  ; Pointer vissza
                JNC     SRT_DBS_SUCC    ;
SRT_DBS_ERR:                            ;Hiba van
                MOV     ERR_IND, AL     ;Hibakód mentése
                JMP     SRT_DBS_RET     ;Visszatérés
SRT_DBS_SUCC:                           ;Sikeres művelet
                CALL    DB_OPEN         ;Megint meg kell nyitni
                JC      SRT_DBS_ERR     ;Nagy baj van
SRT_DBS_RET:                            ;Visszatérés
                CLC                     ;Nem lépünk ki
                RET                     ;
                                        ;
SRT_DBS ENDP                            ;
                                        ;
;---------------------------------------;
; SRC_ELEMENT   - elem keresése név szerint, alprogrammal
;---------------------------------------;
                                        ;
SRC_ELEMENT     PROC    NEAR            ;
                                        ;
                MOV     BX, OFFSET EX_SEARCH
                MOV     AX, CS:PRG_ENV  ;Eredeti environment
                MOV     EX_S_ENV.[ BX ], AX
                MOV     AX, OFFSET CMDL_SEARCH
                MOV     EX_P_PAR.[ BX + OFFS ], AX
                MOV     EX_P_PAR.[ BX + SEGM ], DS
                MOV     DX, OFFSET PATH_SEARCH
                MOV     CS:SAVE_SS, SS  ;
                MOV     CS:SAVE_SP, SP  ;
                DOSCALL DOS_EXEC, DOS_EXEC_PRC
                MOV     SS, CS:SAVE_SS  ;
                MOV     SP, CS:SAVE_SP  ;
                JNC     SRC_ELEMENT_SUCC;
```

```
SRC_ELEMENT_ERR:                        ;
                MOV     ERR_IND, AL     ;
                JMP     SRC_ELEMENT_RET ;
SRC_ELEMENT_SUCC:                       ;
                CLC                     ;
SRC_ELEMENT_RET:                        ;
                RET                     ;
                                        ;
SRC_ELEMENT     ENDP                    ;
                                        ;
;---------------------------------------;
; PROG_EXIT     - kilépés a programból
;---------------------------------------;
                                        ;
PROG_EXIT       PROC    NEAR            ;
                CALL    DB_CLOSE        ;Lezárjuk az adatbázist
                JNC     PROG_EXIT_SUCC  ;Nem sikeres
PROG_EXIT_UNSUCC:                       ;
                MOV     ERR_IND, AL     ;Hibakód mentése
PROG_EXIT_SUCC:                         ;
                STC                     ;Jelezzük a kilépést
                RET                     ;
PROG_EXIT       ENDP                    ;
                                        ;
;---------------------------------------;
; DB_OPEN       - az adatbázis megnyitása vagy létrehozása
;---------------------------------------;
                                        ;
DB_OPEN         PROC    NEAR            ;
                                        ;
                MOV     DX, OFFSET DB_PATH
                MOV     CX, ATR_ARCHV   ;Keresés előkészitése
                DOSCALL DOS_FDFRST      ;Létezik az adatbázis?
                JNC     DB_OPEN_FILE    ;Igen, megnyitjuk
                CMP     AL, ERR_NOMFIL  ;Nem létezik, vagy más?
                JZ      DB_OPEN_CREATE  ;Nincs adatbázis!
                JMP     DB_OPEN_RET     ;Más baj van
DB_OPEN_FILE:                           ;Megnyitás következik
                MOV     DX, OFFSET DB_PATH
                DOSCALL DOS_OPEN, DB_ACCESS
                JC      DB_OPEN_RET     ;Lehetetlen eset
                MOV     DB_HND, AX      ;File handle mentése
```

```
                CALL    RD_HEADER       ;Header beolvasása
                JMP     DB_OPEN_RET     ;Válasz Carry-ben!
DB_OPEN_CREATE:                         ;Adatbázis létrehozása
                MOV     DX, OFFSET DB_PATH
                MOV     CX, ATR_ARCHV   ;
                DOSCALL DOS_CREAT       ;Létrehozzuk
                JC      DB_OPEN_RET     ;Válasz Carry-ben!
                MOV     DB_HND, AX      ;File handle mentése
                MOV     DB_HEADER.H_DBS, 0      ;A header
                MOV     DB_HEADER.H_DBN, 0      ;előkészítése
                MOV     DB_HEADER.H_CEL, 0      ;alapállapotra
                CALL    WR_HEADER       ;A header kiírása
DB_OPEN_RET:                            ;Válasz Carry-ben!
                RET                     ;
                                        ;
DB_OPEN         ENDP                    ;
                                        ;
;---------------------------------------;
; EDIT_ELEMENT  - egy új elem tartalmának beeditálása
;---------------------------------------;
                                        ;
EDIT_ELEMENT    PROC    NEAR            ;
                                        ;
                EDIT    NAM             ;Sorra kiírjuk
                EDIT    PNM             ; az összes elemre
                EDIT    ADR             ; vonatkozó kérdést,
                EDIT    PHO             ; és beolvassuk
                EDIT    PHP             ; a válaszokat
                                        ;
                RET                     ;
                                        ;
EDIT_ELEMENT    ENDP                    ;
                                        ;
;---------------------------------------;
; DB_CLOSE      - az adatbázis lezárása
;---------------------------------------;
                                        ;
DB_CLOSE        PROC    NEAR            ;
                MOV     BX, DB_HND      ;
                DOSCALL DOS_CLOSE       ;
                RET                     ;Válaszkód Carry-ben!
DB_CLOSE        ENDP                    ;
```

```
;-----------------------------------------;
; RD_HEADER     - adatbázis header-jének beolvasása
;-----------------------------------------;
                                          ;
RD_HEADER       PROC    NEAR              ;
                CALL    MOV_PNT_0         ;File pointer előre
                READ_H  OFFSET! DB_HEADER, SIZ_DB_HD, DB_HND
                RET                       ;Válaszkód Carry-ben!
RD_HEADER       ENDP                      ;
                                          ;
;-----------------------------------------;
; WR_HEADER     - adatbázis header-jének kiírása
;-----------------------------------------;
                                          ;
WR_HEADER       PROC    NEAR              ;
                CALL    MOV_PNT_0         ;File pointer előre
                WRIT_H  OFFSET! DB_HEADER, SIZ_DB_HD, DB_HND
                RET                       ;Válaszkód Carry-ben!
WR_HEADER       ENDP                      ;
                                          ;
;-----------------------------------------;
; MOV_PNT_0     - file pointer a 0 pozícióra
;-----------------------------------------;
                                          ;
MOV_PNT_0       PROC    NEAR              ;
                XOR     CX, CX            ;
                XOR     DX, DX            ;
                MOV     BX, DB_HND        ;Pointer előre
                DOSCALL DOS_MOVPNT, DOS_MOVPNT_ST
                RET                       ;
MOV_PNT_0       ENDP                      ;
                                          ;
;-----------------------------------------;
; WR_CUR_EL     - kurrens elem kiírása
;-----------------------------------------;
                                          ;
WR_CUR_EL       PROC    NEAR              ;Ez a program igazából
                                          ; fel nem használt
                CALL    MOV_PNT_CURR      ; "mellékterméke"!
                WRIT_H  OFFSET! DB_ELEMENT, SIZ_DB_ELM, DB_HND
                RET                       ;
WR_CUR_EL       ENDP                      ;
```

```
;-----------------------------------------;
; WR_LAST_EL    - elem kiírása az utolsó helyre
;-----------------------------------------;
                                          ;
WR_LAST_EL      PROC    NEAR              ;
                MOV     AX, DB_HEADER.H_DBN
                MOV     DB_HEADER.H_CEL, AX     ;Kurrens !
                CALL    WR_CUR_EL         ;
                RET                       ;
WR_LAST_EL      ENDP                      ;
                                          ;
;-----------------------------------------;
; MOV_PNT_CURR  - file pointer a kurrens blokk elejére
;-----------------------------------------;
                                          ;
MOV_PNT_CURR    PROC    NEAR              ;
                MOV     AX, SIZ_DB_ELM  ;Elemhossz AX-be
                MUL     DB_HEADER.H_CEL ;Szorozva indexszel
                ADD     AX, SIZ_DB_HD   ;Plusz header hossza
                ADC     DX, 0           ;Ez egy duplaszó ám!
                MOV     CX, AX            ;
                XCHG    CX, DX          ;File pointer CX:DX-be
                MOV     BX, DB_HND        ;
                DOSCALL DOS_MOVPNT, DOS_MOVPNT_ST
                RET                       ;
MOV_PNT_CURR    ENDP                      ;
                                          ;
;-----------------------------------------;
;

CODE    ENDS
        END     START
```

Lássuk most az alprogramot! A már megismert rutinok mellett itt az STRCMPC rutint hívjuk kívülről. Ez a rutin bemenetként egy elsődleges és egy másodlagos stringet kap DS:SI-ben, illetve ES:DI-ben. Addig folytatja a stringek összehasonlítását, míg különböző karaktereket nem talál, vagy az elsődleges szövegben el nem éri az első vezérlőkaraktert (ez itt egy kocsivissza lesz, lásd 0A funkció). Tehát tulajdonképpen azt vizsgálja meg, hogy az elsődleges string része-e a másodlagosnak. Ezzel azt érjük el, hogy a keresendő név meghatározásánál nem kell megad-

ni a teljes nevet, csak annyi karaktert, amennyi egyértelműen azonosítja a személy nevét.

A makrók nem ravaszabbak, mint a főprogram makrói; ugyanazokkal a trükkökkel dolgozva ezúttal az adatbázis egy rekordjának elemenkénti kiírását oldjuk meg segítségükkel.

Az adatterületen annyit kell megfigyelni, hogy más file-elérési módot adunk meg. Az SRC_LOWER, SRC_MIDDLE és SRC_UPPER változók az intervallumfelezési eljárásban szerepelnek.

A program a szokásos előkészítés után bekéri a keresendő nevet, kikeresi az adatbázisból, és kilép. Egyetlen fontos rutin van az egész programban: az SRCNAME rutin, amely a program fő tevékenységét végzi. Ennek menete a következő: az SRC_LOWER változót nullára, az SRC_UPPER-t pedig a rekordok számára állítja. Ezután mindig megfelezi az SRC_LOWER és az SRC_UPPER közötti távolságot, ezt helyezi el az SRC_MIDDLE változóba. Felolvassa az így kapott sorszámú rekordot, és összehasonlítja a keresendő névvel. Ha egyezik, készen vagyunk. Ha a keresett név betűrendben előbb van, akkor az SRC_MIDDLE-t az SRC_UPPER-be tölti, és folytatja az eljárást, hacsak az intervallum hossza nem csökkent 0-ra; ekkor be kell fejezni, a keresett személy nincs az adatbázisban. Ha a keresett név betűrendben hátrább van, akkor egy kicsit trükköznie kell. Ha ugyanis a felezések folytán az intervallum hossza 1-re csökkent, akkor az újabb felezés mindig az alsó határt adja, és végtelen ciklusba esünk. Ennek kivédésére nem azt vizsgáljuk, hogy az intervallum hossza nulla-e, hanem azt, hogy egy-e.

Elemezzük ezt a lehetőséget! Tegyük fel először, hogy a keresett név szerepel az adatbázisban. Ha az utolsó elemről van szó, akkor a felezések végén az alsó határ az utolsó index, a felső pedig eggyel nagyobb, mert innen indultunk el. Az utolsó összehasonlítás rátalál a névre, mert az összehasonlítás után vizsgáljuk az interrvallum hosszát. Ha nem az utolsó elemet keressük, akkor a felezések során nyilvánvalóan rábukkanunk.

Tegyük fel most, hogy a keresett név nincs az adatbázisban. Ha betűrendben az összes névnél hátrább van, akkor mindig a felső félintervallumon folytatjuk a munkát, és az utolsó elem (melynek indexe eggyel kisebb a rekordok számánál, mivel az indexek nullától indulnak) vizsgálata után az SRC_LOWER és az SRC_UPPER közötti különbség 1, és az összehasonlítás sikertelen. Ekkor ki kell lépni, nincs értelme a további keresésnek. Ha a név két bentlévő név közé esik, akkor vagy az utolsó elem előtt van, vagy valahol középen. Az első esetben az utolsó

előtti elem indexe lesz az SRC_LOWER és az SRC_MIDDLE változókban. Az utolsó összehasonlítás "kisebb" eredményt ad, ekkor az intervallum hossza nullára csökken. A második esetben viszont a keresés közben SRC_UPPER ellépett a kezdeti értékéről, hiszen legalább egyszer "kisebbnek" találtuk a keresett elemet. Amikor az intervallum hossza egyre csökkent, még végzünk egy összehasonlítást; ekkor SRC_LOWER értéke megegyezik SRC_MIDDLE-ével; a keresett név ennél nagyobb, de az SRC_UPPER indexűnél kisebb; ekkor sem kell tovább keresni.

```
TITLE   Keresés az adatbázisban

        EXTRN   CUT_EXE:NEAR, PRTERRMSG:NEAR
        EXTRN   STRCMPC:NEAR
;
; Konstansok, makrók, rekordok
;
        .XLIST                  ;DOSCALL.INC-et nem listázzuk!
        INCLUDE         DOSCALL.INC
        .LIST
        INCLUDE         DBC.INC

;-----------------------------------------------;
; Makrók
; INSTR - adott hosszúságú string beolvasása
;-----------------------------------------------;

INSTR   MACRO   SIZE
                MOV     IN_BUFF_SIZE, SIZE      ;;String hossza
                MOV     DX, OFFSET IN_BUFF_ADDR ;;Buffer címe
                DOSCALL DOS_STRDSTR             ;;
        ENDM                                    ;;
;-----------------------------------------------;
; OUTMSG - üzenetdefiniáló makró
;-----------------------------------------------;

OUTMSG  MACRO   EL, MESS
DB_OUT_&EL      DB      CHR_CR, CHR_LF, CHR_TAB, CHR_TAB
                DB      MESS
                DB      '$'
        SIZ_DB_OUT_&EL  EQU     $ - DB_OUT_&EL
        ENDM                                    ;
```

```
;-------------------------------------------------;
; OUTEL - elemkiíró makró
;-------------------------------------------------;

OUTEL   MACRO   EL
                MOV     DX, OFFSET DB_OUT_&EL
                DOSCALL DOS_STWRSTR
                OUTWRIT DB_ELEMENT.D_&EL, SIZ_&EL
        ENDM

;-------------------------------------------------;
; Adatterületek
;-------------------------------------------------;
DATA    SEGMENT BYTE    PUBLIC  'DATA'
                ;-----------------------;
                ; Adatbázis-kezelő változók
                ;-----------------------;
DB_PATH         DB      'C:PERSONAL.DTB', 0
DB_ACCESS       DB      DOS_OPEN_READ + DOS_OPEN_DNWRIT
DB_HND          DW      0               ;File handle
                ;-----------------------;
                ; Keresési változók
                ;-----------------------;
SRC_LOWER       DW      0               ;Alsó határ
SRC_MIDDLE      DW      0               ;Intervallum közepe
SRC_UPPER       DW      0               ;Felső határ
                ;-----------------------;
                ; Adatbázis-bufferek
                ;-----------------------;
DB_HEADER       DB_HDR  <>              ;Header-struktúra
DB_ELEMENT      DB_ELM  <>              ;Kurrens elem
                ;-----------------------;
                ; Kiírandó üzenetek
                ;-----------------------;
MSG_NAME        DB      CHR_ESC, '[2J'  ;Ernyőtörlés
 DB     4       DUP     (CHR_LF)        ;4 x soremelés
 DB     'Keresett személy neve (24): '  ;Kérdés szövege
        SIZ_MSG_NAME    EQU     $ - MSG_NAME
MSG_NOT_F       DB      CHR_ESC, '[2J'  ;Nem találtuk meg
 DB     4       DUP     (CHR_LF)
 DB     'A keresett személy nincs az adatbázisban.'
        SIZ_MSG_NOT_F   EQU     $ - MSG_NOT_F   ;
```

```
MSG_F           DB      CHR_ESC, '[2J'  ;Megtaláltuk
 DB     4       DUP     (CHR_LF)
 DB     'A keresett személy adatai: '
        SIZ_MSG_F       EQU     $ - MSG_F
                ;-----------------------;
                ; Elem-megjelenítési üzenetek
                ;-------------------------------;
        OUTMSG  NAM, 'Név:                '     ;Név,
        OUTMSG  PNM, 'Személyi szám:      '     ; személyi szám
        OUTMSG  ADR, 'Lakcím:             '     ;
        OUTMSG  PHO, 'Hivatali telefon:   '     ; stb.
        OUTMSG  PHP, 'Otthoni  telefon:   '     ;
                ;-------------------------------;
                ; Beolvasó buffer
                ;-----------------------;
IN_BUFF_ADDR    LABEL   BYTE            ;Struktúra neve
IN_BUFF_SIZE    DB      0, 0            ;Hossz, válasz hossza
IN_BUFF         DB      SIZ_BUFF DUP (0);Buffer

DATA    ENDS


;---------------------------------------;
; Stack-terület
;---------------------------------------;

STACK   SEGMENT PARA    STACK   'STACK'
        DW      100     DUP ( ? )       ;Hossza 100 szó
STACK   ENDS                            ;

;---------------------------------------;
; Kódszegmens
;---------------------------------------;

CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:DATA, SS:STACK, ES:DATA

PSP_SEG         DW      0               ;Mentési területek
PRG_ENV         DW      0               ;
                                        ;
SAVE_SS         DW      0               ;
SAVE_SP         DW      0               ;
```

```
START:                                  ;Belépési pont
                                        ;
                MOV     CS:PSP_SEG, DS  ;Szokásos előkészités:
                XOR     SI, SI          ;
                MOV     AX, PSP_ENVSEG[ SI ]    ;Environment c.
                MOV     CS:PRG_ENV, AX  ;Elmentjük
                                        ;
                MOV     AX, DATA        ;
                MOV     DS, AX          ;DS -> DATA szegmens
                CALL    CUT_EXE         ;Memória felszabaditása
                PUSH    DS              ;
                POP     ES              ;ES -> DATA szegmens
                                        ;
                CALL    DB_OPEN         ;Megnyitjuk az adatb.-t
                JC      ERR_EXIT        ;Hiba esetén kilépünk
                                        ;
                CALL    RDNAME          ;Keresendő név beolv.
                CALL    SRCNAME         ;Név keresése
                INSTR   1               ;Várunk CHR_CR-re
                                        ;
                CALL    DB_CLOSE        ;Lezárjuk az adatb.-t
                JNC     SUCC_EXIT       ;
                                        ;
                EXIT                    ;
                                        ;
;---------------------------------------;
;
; RDNAME        - keresendő név beolvasása
;       Input:
;               semmi
;       Output:
;               IN_BUFF - a keresendő név
;
;---------------------------------------;
                                        ;
RDNAME          PROC    NEAR            ;
                                        ;
                OUTWRIT MSG_NAME        ;Kérdés kiirása
                INSTR   SIZ_BUFF        ;Név beolvasása
                RET                     ;
                                        ;
RDNAME          ENDP                    ;
```

```
;-----------------------------------------;
; SRCNAME       - a név keresése az adatbázisban
;       Input:
;               semmi
;       Output:
;               semmi
;       Ha megtalálja az elemet, kiírja az adatokat;
;       ha nem találja meg, akkor hibaüzenetet ír ki.
;       Keresési eljárás: intervallumfelezés
;-----------------------------------------;
                                        ;
SRCNAME         PROC    NEAR            ;
                                        ;
                XOR     AX, AX          ;
                MOV     SRC_LOWER, AX   ;
                MOV     AX, DB_HEADER.H_DBN
                MOV     SRC_UPPER, AX   ;
                JMP     SRCNAME_CONT    ;
SRCNAME_LOOP:                           ;
                CALL    RD_MIDDLE_EL    ;Középső elem beolv.
                CALL    CMP_NAME        ;
                JE      SRCNAME_FOUND   ;
                JG      SRCNAME_UPPER   ;
SRCNAME_LOWER:                          ;Kisebb, mint középső
                MOV     AX, SRC_MIDDLE  ;Az alsó felén foly-
                MOV     SRC_UPPER, AX   ; tatjuk
                JMP     SRCNAME_CONT    ;
SRCNAME_UPPER:                          ;Nagyobb, mint középső
                MOV     AX, SRC_UPPER   ;Mostani legmagasabb
                DEC     AX              ; rekordszám
                CMP     AX, SRC_MIDDLE  ;Ez volt az utolsó?
                LCJ     E, SRCNAME_NFOUND   ;Igen, nincs benne
                MOV     AX, SRC_MIDDLE  ;A felső felén foly-
                MOV     SRC_LOWER, AX   ; tatjuk
SRCNAME_CONT:                           ;
                MOV     AX, SRC_LOWER   ;
                CMP     AX, SRC_UPPER   ;Elfogyott?
                JE      SRCNAME_NFOUND  ;
                ADD     AX, SRC_UPPER   ;
                SHR     AX, 1           ;Ez az új középső elem
                MOV     SRC_MIDDLE, AX  ;
                JMP     SRCNAME_LOOP    ;Folytatjuk
```

```
SRCNAME_FOUND:                                  ;
                OUTWRIT MSG_F                   ;Megtalálta. Fő üzenet
                OUTEL   NAM                     ; és az egyes
                OUTEL   PNM                     ; elemek értékének
                OUTEL   ADR                     ; kiírása
                OUTEL   PHO                     ;
                OUTEL   PHP                     ;
                JMP     SRCNAME_RET             ;
SRCNAME_NFOUND:                                 ;
                OUTWRIT MSG_NOT_F               ;Nincs meg, üzenet ki
SRCNAME_RET:                                    ;
                RET                             ;
SRCNAME         ENDP                            ;
                                                ;
;-----------------------------------------------;
; CMP_NAME      - a nevek összehasonlítása
;       Input:
;               semmi
;       Output:
;               flagek  - mutatják az összehasonlítás
;                         eredményét.
;-----------------------------------------------;
                                                ;
CMP_NAME        PROC    NEAR                    ;
                MOV     SI, OFFSET IN_BUFF
                MOV     DI, OFFSET DB_ELEMENT.D_NAM
                CALL    STRCMPC                 ;Vezérlőkarakterig öh.
                RET                             ;
CMP_NAME        ENDP                            ;
;-----------------------------------------------;
; DB_OPEN       - az adatbázis megnyitása
;-----------------------------------------------;
DB_OPEN         PROC    NEAR                    ;
DB_OPEN_FILE:                                   ;
                MOV     DX, OFFSET DB_PATH
                DOSCALL DOS_OPEN, DB_ACCESS
                JC      DB_OPEN_RET             ;Hiba, nem ide tartozik
                MOV     DB_HND, AX              ;File handle mentése
                CALL    RD_HEADER               ;Header beolvasása
DB_OPEN_RET:                                    ;Válasz Carry-ben!
                RET                             ;
DB_OPEN         ENDP                            ;
```

```
;-----------------------------------------;
; DB_CLOSE      - az adatbázis lezárása
;       pontosan mint DBC-1-ben
; RD_HEADER     - adatbázis header-jének beolvasása
;       pontosan mint DBC-1-ben
; MOV_PNT_0     - file pointer a 0 pozícióra
;       pontosan mint DBC-1-ben
;
; RD_MIDDLE_EL  - középső elem beolvasása
;-----------------------------------------;
                                          ;
RD_MIDDLE_EL    PROC    NEAR              ;
                CALL    MOV_PNT_CURR      ;
                READ_H  OFFSET! DB_ELEMENT, SIZ_DB_ELM, DB_HND
                RET                       ;
RD_MIDDLE_EL    ENDP                      ;
                                          ;
;-----------------------------------------;
; MOV_PNT_MIDDLE - file pointer a "középső" blokk elejére
;-----------------------------------------;
                                          ;
MOV_PNT_MIDDLE  PROC    NEAR              ;
                                          ;
                MOV     AX, SIZ_DB_ELM    ;Elemhossz,
                MUL     SRC_MIDDLE        ; szorozva kívánttal
                ADD     AX, SIZ_DB_HD     ; plusz a header
                ADC     DX, 0             ; hossza (duplaszó)
                MOV     CX, AX            ;
                XCHG    CX, DX            ;Az érték CX:DX-be
                MOV     BX, DB_HND        ;
                DOSCALL DOS_MOVPNT, DOS_MOVPNT_ST
                RET                       ;
MOV_PNT_MIDDLE  ENDP                      ;
                                          ;
;-----------------------------------------;
;
CODE    ENDS
        END     START
```

Mielőtt elköszönnénk legnagyobb (és talán legérdekesebb) példaprogramunktól, törlesszük két adósságunkat. Az első a DBC.INC file, a második a felhasznált segédrutinok listája. A

DBC.INC file néhány konstans- és két struktúradeklarációt tartalmaz, valamint a közösen használt file specifikációját. Itt **mindössze** annyit kell megfigyelnünk, hogy a DATA szegmens kétszer szerepel, a második esetben nem PARA, hanem BYTE igazítási típussal!

```
COMMENT *
Ez az include file tartalmazza az adatbázis kezeléséhez szüksé-
ges legfontosabb konstans- és makródeklarációkat.
*
;-----------------------------------------------;
; Adatbázis-konstansok
;-----------------------------------------;
NUM_FUNCT       EQU     4               ;Funkciók száma
                ;----------------------;
                ; Elemek hossza
                ;------------------------;
                ;
SIZ_NAM         EQU     24              ;Név hossza
SIZ_PNM         EQU     11              ;Személyi szám hossza
SIZ_ADR         EQU     43              ;Cím hossza
SIZ_PHO         EQU     12              ;Hivatali telefon
SIZ_PHP         EQU     12              ;Otthoni telefonszám
                ;
SIZ_HNM         EQU     10              ;Header-név hossza
SIZ_CMD         EQU      2              ;Parancs hossza
SIZ_BUFF        EQU     SIZ_ADR + 1     ;Buffer hossza
                ;
;---------------------------------------;
; Struktúrák
;---------------------------------------;
                ;
DB_ELM  STRUC   ;
        D_NAM   DB  SIZ_NAM  DUP  ( 0 ) ;;Név
        D_PNM   DB  SIZ_PNM  DUP  ( 0 ) ;;Személyi szám
        D_ADR   DB  SIZ_ADR  DUP  ( 0 ) ;;Cím
        D_PHO   DB  SIZ_PHO  DUP  ( 0 ) ;;Munkahelyi tel.szám
        D_PHP   DB  SIZ_PHP  DUP  ( 0 ) ;;Otthoni telefonszám
        D_END   DB  CHR_CR, CHR_LF      ;;Zárókarakterek
DB_ELM  ENDS    ;
                ;
    SIZ_DB_ELM  EQU  TYPE  DB_ELM       ;Alapstruktúra hossza
```

```
DB_HDR  STRUC                                   ;
        H_HNM   DB  SIZ_HNM  DUP  (' ') ;;Hogy első maradjon
        H_DBS   DW  0                   ;;Adatbázis hossza
        H_DBN   DW  0                   ;;Elemek száma
        H_CEL   DW  0                   ;;Kurrens elem száma
      _ H_END   DB  CHR_CR, CHR_LF      ;;Zárókarakterek
DB_HDR  ENDS                            ;
    SIZ_DB_HD   EQU  TYPE  DB_HDR       ;Headerstruktúra hossza
;---------------------------------------;
; Adatszegmens közösen használt része
;---------------;
DATA    SEGMENT PARA    PUBLIC  'DATA'
                ; Közösen használt file specifikációja
                ;---------------------;
                                        ;EQU-val is def.-ható
DB_PATH         DB      'C:PERSONAL.DTB', 0
DATA    ENDS
```

Végül lássuk a segédrutinokat. Ezeket azért emeltük ki a programokból, mert teljesen általános rutinok lévén könyvtárban van a helyük. Az első egy egyszerű másolórutin, amely addig viszi át a byte-okat az egyik területről a másikra, míg el nem éri az első vezérlőkaraktert (jelen felhasználásában a beolvasó bufferben a rendszer által elhelyezett kocsivissza karaktert).

```
;---------------------------------------;
; STRCOPYC      - String másolása az első vezérlőkód eléréséig
;       Input:
;               DS:SI   - kiindulási string
;               ES:DI   - célstring
;       Output:
;               CX      - átmásolt karakterek száma a záró-
;                         karakter nélkül
;               Minden egyéb regiszter mentve
;---------------------------------------;
                                        ;
STRCOPYC        PROC    NEAR            ;
                                        ;
                PUSHF                   ;
                PUSH    AX              ;
                PUSH    SI              ;
                PUSH    DI              ;Regiszterek mentése
```

```
                CLD                             ;
                XOR     CX, CX                  ;
STRCOPYC_LOOP:                                  ;
                LODSB                           ;Következő karakter be
                CALL    CHK_CONTR               ;Vezérlőkarakter?
                JC      STRCOPYC_RET            ;Ha Carry=1, igen
                STOSB                           ;Kiírás
                LOOP    STRCOPYC_LOOP           ;Számlálás is!
STRCOPYC_RET:                                   ;
                NOT     CX                      ;Másolt karakterek sz.
                POP     DI                      ;NEG-el zárókarakter is
                POP     SI                      ;benne volna!
                POP     AX                      ;
                POPF                            ;
                RET                             ;
STRCOPYC        ENDP                            ;
```

A második rutin stringek összehasonlítását végzi addig, amig különböző karaktereket nem talál, vagy az elsődleges stringben el nem éri az első vezérlőkaraktert (jelen felhasználásban a beolvasó bufferben a rendszer által elhelyezett kocsivissza karaktert).

```
;-----------------------------------------------;
; STRCMPC       - Két string összehasonlítása vezérlőkarakterig
;       Input:
;               DS:SI   - elsődleges string kezdőcíme
;               ES:DI   - másodlagos string kezdőcíme
;       Output:
;               Flagek  - az összehas. eredménye, mely előjel
;                         nélküli vizsgálatokkal tesztelhető
;               Minden egyéb regiszter változatlan.
;       Ez a rutin nem nézi végig a stringeket, hanem ha az
;       elsődleges stringben vezérlőkaraktert talál, és a
;       stringek addig megegyeznek, akkor "egyenlővel" válaszol.
;---------------------------------------;
STRCMPC PROC    NEAR                            ;
                PUSH    AX                      ;A felhasznált
                PUSH    CX                      ; regiszterek
                PUSH    SI                      ; mentése
                PUSH    DI                      ;
                CLD                             ;Felfelé megyünk
```

```
                XOR     CX, CX          ;Nehogy CX elfogyjon!
STRCMPC_LOOP:                           ;
                LODSB                   ;Egyik string köv. eleme
                CALL    CHK_CONTR       ;Vezérlőkarakter?
                JC      STRCMPC_EQU     ;A rész-string pontos!
                SCASB                   ;Ez az összehasonlítás
                JNE     STRCMPC_DIFF    ;Nem egyenlő, vége
                LOOP    STRCMPC_LOOP    ;
STRCMPC_EQU:                            ;Egyenlő,
                XOR     AX, AX          ; Zero flag beállítás
STRCMPC_DIFF:                           ;A flagek jók!
                POP     DI              ;
                POP     SI              ;
                POP     CX              ;
                POP     AX              ;
                RET                     ;
STRCMPC ENDP                            ;
```

éd-segédrutin;
megvizsgálja,
sem.

Az utolsó segédrutin tulajdonképpen egy seg
csak a fenti két rutin használja fel. Egyszerűen
hogy az AL-ben megadott kód vezérlőkarakter-e vagy

```
;---------------------------------------;                    kter-e
; CHK_CONTR     - Karaktervizsgálat; AL vezérlőkara
;       Input:
;               AL      - karakter kódja (8 bites kó        d!)
;       Output:
;               Carry   - 1, ha vezérlőkarakter; 0, ha      a nem
;---------------------------------------;
CHK_CONTR       PROC    NEAR            ;
                CMP     AL, CHR_BLANK   ;JL 8 bites kc      dra hely-
                JB      CHK_CONTR_YES   ; telen eredmé      nyt adna!
                CLC                     ;A Carry csal!
                JMP     CHK_CONTR_RET   ;                   r,
CHK_CONTR_YES:                          ;Vezérlőkaraktei
                STC                     ; Carry=1
CHK_CONTR_RET:                          ;
                RET                     ;
CHK_CONTR       ENDP                    ;
```

Mint a megjegyzésekből kiolvasható, a vezérlőkaraktere
ismerésére azért kell külön rutint írni, hogy a kiterjes
ASCII-készlet elemei ne okozzanak hibát.

t fel-
ztett

VIII.3.4. A standard hibajelző file átirányítása

Ez a példaprogram egyfelől illusztrálja a standard hibajelző file kezelését (amellyel analóg módon kezelhető a standard kisegítő file is), másfelől pedig bemutatja, hogyan lehet programból átirányítani egy standard file-t egy lemezen levő fileba. Az itt látható eljárás azért is hasznos, mert az átirányító és helyreállító rutint szinte bármelyik programunkban jól felhasználhatjuk. Az ilyen hasznos kis programocskákat érdemes egy saját könyvtárba szervezni, és szerkesztési időben hozzáadni a készülő programhoz. Erről bővebben sorozatunk utolsó kötetében olvashatunk majd. Ez a példaprogram egyébként ismét COM formátumú.

```
TITLE   Standard hibajelző átirányítása
;
; Konstansok, makrók, rekordok
;

        .XLIST                  ;DOSCALL.INC-et nem listázzuk!
        INCLUDE         DOSCALL.INC
        .LIST

;
; Kódszegmens
;

CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:CODE, ES:CODE, SS:CODE
        ORG     100H

ENTRY:
                JMP     START

NEWERR  DB      'NEWERR.TXT', 0         ;Hibafile a lemezen
                        ;---------------;
                        ;Kiírandó üzenetek
                        ;---------------;
MSGDEF  MSG_FRST, 'Ez   még  STDERR-re   megy',,NODOLLAR
MSGDEF  MSG_SECD, 'Ez   már  NEWERR-re   megy',,NODOLLAR
MSGDEF  MSG_THRD, 'Ez   is   NEWERR-re   megy',,NODOLLAR
MSGDEF  MSG_FORT, 'Ez megint STDERR-re   megy',,NODOLLAR
```

```
START:                                  ;
                ERRWRIT MSG_FRST        ;Első hibaüzenet ki
                                        ;
                MOV     DX, OFFSET NEWERR
                CALL    REDIR_STDERR    ;Hibajelző file átir.
                                        ;
                ERRWRIT MSG_SECD        ;További hibaüzenetek
                ERRWRIT MSG_THRD        ;
                                        ;
                CALL    RESET_STDERR    ;Hibajelző file vissza
                                        ;
                ERRWRIT MSG_FORT        ;Hibaüz. az eredetire
                JMP     SUCC_EXIT       ;
                                        ;
                EXIT                    ;Kilépés makróval
                                        ;
;---------------------------------------;
; REDIR_STDERR  - a standard error átirányítása
;       Input:
;               DS:DX   - file-specifikáció címe
;---------------------------------------;
                                        ;
REDIR_STDERR    PROC    NEAR            ;
                                        ;
                PUSH    DX              ;File-spec. mentése
                MOV     BX, HND_STD_ERR ;
                DOSCALL DOS_DUPHND      ;Handle-kétszerezés
                MOV     CS:HND_SAVERR, AX
                POP     DX              ;File-spec. címe vissza
                MOV     CX, ATR_ARCHV   ;File attributuma
                DOSCALL DOS_CREAT       ;Létrehozzuk NEWERR-t
                MOV     CS:HND_NEWERR, AX
                MOV     BX, CS:HND_NEWERR
                MOV     CX, HND_STD_ERR ;
                DOSCALL DOS_FRCHND      ;Átirányítás
                                        ;
                RET                     ;
                                ;-------;
HND_NEWERR      DW      0       ;File handle a lemezfile-nak
HND_SAVERR      DW      0       ;File handle a mentéshez
                                ;
REDIR_STDERR    ENDP            ;
```

```
;-----------------------------------;
; RESET_STDERR  - standard error helyreállítása
;-----------------------------------------;
                                          ;
RESET_STDERR    PROC    NEAR              ;
                                          ;
                MOV     BX, CS:HND_SAVERR       ;Mentő handle
                MOV     CX, HND_STD_ERR   ;
                DOSCALL DOS_FRCHND        ;Vissza az egész
                                          ;
;-----------------------------------------;
;               MOV     BX, CS:HND_NEWERR
;               DOSCALL DOS_CLOSE         ;NEWERR lezárása
;-----------------------------------------;
                                          ;
                RET                       ;
                                          ;
RESET_STDERR    ENDP                      ;
                                          ;

CODE    ENDS
        END     ENTRY
```

E programban kevesebb dolgot kell megfigyelni, de talán fontosabbakat. Az első, hogy szubrutinokat írtunk - ezeket bárhol (EXE file-ban is) lehet használni.

A másik, hogy a szubrutinok "eldugják" a mentésre és az átirányításra használt file-sorszámokat. Ez az ő kizárólagos magánügyük. A program folyton a standard hibajelző file-ra ír, és ha (általában tesztelési céllal) meghívjuk a REDIR_STDERR rutint, senki sem fogja észrevenni a turpisságot - csak éppen kinyomtathatóan, lemezfile-on kapjuk a hibaüzeneteket. A belügyek elrejtésére, mint már nem egyszer, ismét a programszegmensben, a szubrutinok között elhelyezett változókat használtunk fel.

A harmadik megfigyelni való a RESET_STDERR rutinban van. A lemezfile lezárása elmaradt, mivel a ráirányított file-sorszám újabb átirányítása a file-t automatikusan lezárja. Persze az eredeti standard hibajelző lezárására (lévén nem file, hanem perifériális eszköz) nem kerül sor.

Fontos emlékeztetnünk arra, hogy file-sorszámban oly szegény napjainkban egy standard file átirányítása bizony nem olcsó mulatság, hiszen a korlátozott számban rendelkezésünkre álló sorszámok közül további kettőt foglal le!

VIII.3.5. Ideiglenes file kezelése

Az alábbi példaprogram egy ideiglenes file létrehozását, kezelését és végül törlését mutatja be. Sok mondanivalónk erről nincs - egyszerű, bár egy kicsit munkaigényes programocskát olvashatunk. Főleg a megfelelő makrók definiálása igényel egy kis figyelmet és pluszmunkát.

Annyit érdemes elöljáróban megjegyezni, hogy itt egyszerű példát látunk a File Pointer mozgatására.

```
TITLE   Ideiglenes file kezelése
;
; Külső objektumok
;
        EXTRN   GETPAR:NEAR, PRTERRMSG:NEAR
; Konstansok, makrók, rekordok
        .XLIST                  ;DOSCALL.INC-et nem listázzuk!*
        INCLUDE         DOSCALL.INC
        .LIST
SIZ_BUF         EQU     20
FILE_POS        EQU     12

;
; Kódszegmens
;
CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:CODE, ES:CODE, SS:CODE
        ORG     100H
ENTRY:                                  ;
                JMP     START           ;

BUFFER          DW      SIZ_BUF DUP ( ? )
HND_TEMP        DW      0
        TMPBLK  TEMP1, \TEMP    ;A file a TEMP dir.-ban lesz
MSGDEF  MSG_FRST, 'Egy sor az ideigl. file-ba',, NODOLLAR
MSGDEF  MSG_SECD, 'Még egy sor az ideigl. file-ba',, NODOLLAR
MSGDEF  MSG_MIDL, 'Bele a közepébe',, NODOLLAR

START:                                  ;
                MOV     DX, OFFSET TEMP1;
                MOV     CX, ATR_ARCHV   ;
                DOSCALL DOS_CRTEMP      ;Ideigl. file létrehoz.
```

```
                MOV     HND_TEMP, AX    ;File handle elmentése
                MOV     CX, SIZ_MSG_FRST + SIZ_MSG_SECD
                MOV     DX, OFFSET MSG_FRST
                MOV     BX, HND_TEMP    ;
                DOSCALL DOS_WRITE       ;Két sor írása együtt!
                                        ;
                MOV     CX, 0           ;
                MOV     DX, FILE_POS    ;File-pozíció duplaszó
                MOV     BX, HND_TEMP    ;
                MOV     AL, DOS_MOVPNT_ST       ;Alfunkciókód
                DOSCALL DOS_MOVPNT      ;File Pointer mozgatás
                                        ;
                MOV     CX, SIZ_BUF     ;
                MOV     DX, OFFSET BUFFER
                MOV     BX, HND_TEMP    ;
                DOSCALL DOS_READ        ;Olv. egyet középről
                MOV     CX, SIZ_BUF     ;
                MOV     DX, OFFSET BUFFER
                MOV     BX, HND_STD_OUT ;
                DOSCALL DOS_WRITE       ;Olvasás után beleírunk
                                        ;
                MOV     BX, HND_TEMP    ;
                DOSCALL DOS_CLOSE       ;Ideiglenes file lezár.
                                        ;
; (*)           MOV     DX, OFFSET TEMP1;
; (*)           DOSCALL DOS_DELENTRY    ;Töröljük a file-t
                                        ;
                EXIT                    ;Kilépés makróval
CODE    ENDS                            ;
        END     ENTRY
```

A fent olvasható kis program össze-vissza írja és olvassa az általa létrehozott ideiglenes file-t (hiszen azzal mindent szabad csinálni). Egyetlen apró megjegyzés még: a végén a törlési parancs (ugye, milyen egyszerű?) azért van megjegyzésben, hogy a program első futtatása után láthassuk az eredményt. Másodszorra kieditáljuk a "*"-al jelölt pontosvesszőket (persze a "*"-okkal együtt!), és megnézzük, valóban törli-e a rendszer az újabban létrehozott ideiglenes file-t.

Talán mondanunk sem kell, hogy ez a program a ROOT directory alatt elhelyezkedő TEMP nevű directory-ban hozza létre az ideiglenes file-t (tehát nem a kurrensben). Ha nincs ilyen direc-

tory, a program nem fog lefutni, és még csak nem is szól semmit, mivel (eléggé el nem ítélhető módon) nem foglalkoztunk a hibák kezelésével.

VIII.3.6. Lemezcimke létrehozása

Lemezcimkét FCB-vel és DOS file-kezelési funkcióval egyformán létrehozhatunk. Ha FCB funkciót használunk, akkor a kiterjesztett FCB-ben megadjuk az ATR_VOLAB attributumot, az FCB file-név és file-típus mezőire beírjuk a kívánt (legfeljebb 11 karakteres) FCB-t, a DOS_FCREAT funkcióval létrehozzuk a lemezcimkét, és készen is vagyunk. DOS file-kezelési funkció esetén egy fokkal könnyebb a dolgunk - az attributumot CX-ben adhatjuk meg.

```
TITLE   Lemezcimke létrehozása
;
; Külső objektumok
;
        EXTRN   PRTERRMSG:NEAR
;
; Konstansok, makrók, rekordok
;

        INCLUDE         DOSCALL.INC

DIR_ATTRIB      EQU     ATR_VOLAB

;
; Kódszegmens
;

CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:CODE, ES:CODE, SS:CODE
        ORG     100H

ENTRY:                                  ;
                JMP     START           ;

FILNAM  DB      'JOLLY-JO.KER', 0       ;Lemezcimke két részben
```

```
START:                                          ;
                MOV     CX, DIR_ATTRIB          ;
                HNDCTL  DOS_CREAT, OFFSET! FILNAM
                JC      ERR_EXIT                ;Létrehozzuk a cimkét
                HNDCLL  DOS_CLOSE, AX           ;Le is illik zárni
                JNC     SUCC_EXIT               ;
                EXIT                            ;Kilépés makróval
CODE    ENDS                                    ;
        END     ENTRY
```

Ebben a programban annyit figyeljünk meg, hogy miképpen adjuk meg a lemezcimke tizenegy karakterét a file-specifikációban. Az első nyolc karakter után el kell helyezni egy pontot, hogy a rendszer ne találja hibásnak a file megadását. Csakis így érhetjük el, hogy mind a tizenegy karakter felkerüljön a lemezre.

IX. Directory-kezelés

IX.1. A directory-kezelés alapfogalmai

A directory-kezelő funkciókat két csoportban tárgyaljuk. Az első csoport a directory-val mint egésszel foglalkozik, és a directory-k létrehozását, törlését stb. foglalja magában. A második csoport pedig a directory tartalmával, a bejegyzésekkel foglalkozó funkciók csoportja: bejegyzések keresésére, megváltoztatására szolgálnak az idetartozó függvények.

IX.2. Directory-kezelésre szolgáló funkciók

IX.2.1. Teljes directory-k kezelése

39 Make Directory - DOS_MKDIR
Subdirectory létrehozása - DS:DX-ben adjuk meg a létrehozandó directory-t specifikáló string címét. Sikeres művelet után AL-ben 0-t kapunk vissza.
Hibakódok AX-ben, Carry=1 esetén:

03 - a megadott directory-út (path) nem létezik
05 - tiltott elérés - már létező directory-t próbáltunk újra létrehozni

3A Remove Directory, DOS_RMDIR
Subdirectory törlése - DS:DX-ben adjuk meg a törlendő directory-t specifikáló string címét. Sikeres művelet után AL-ben 0-t kapunk vissza.
Hibakódok AX-ben, Carry=1 esetén:

03 - a megadott directory-út (path) nem létezik
05 - az elérés tiltott, a directory nem üres

3B Change Current Directory, DOS_CHDIR
Aldirectory-váltás - DS:DX-ben adjuk meg a kívánt directory-t specifikáló string címét. Sikeres művelet után AL-ben 0-t kapunk vissza.

Hibakódok AX-ben Carry=1 esetén:

03 - a megadott directory-út (path) nem létezik

47 Get Current Directory, DOS_GTCDIR
A kurrens directory lekérdezése - belépéskor DL-ben adjuk meg a kívánt lemezegység kódját (0 - kurrens lemez, 1 - A:, 2 - B: stb.), DS:SI-ben pedig egy 64 byte-nyi hosszúságú memóriablokk címét. A teljes "path"-t fogja ide beírni a rendszer a lemezegység megadása, valamint a bevezető és záró "\" karakterek nélkül.
Hibakódok AX-ben Carry=1 esetén:

0F - érvénytelen lemezkód

IX.2.2. Directory-bejegyzések kezelése

a) File-keresési funkciók

4E Find First File - DOS_FDFRST
File első előfordulásának keresése - DS:DX-ben megadjuk egy file-specifikáció címét. A file-specifikációban a file-név és a típus tartalmazhat wildcard-karaktereket ("*" és "?") is. CX-ben adhatjuk meg a keresendő file(ok) attributumát (lásd "Directory-bejegyzés" fejezet). Ekkor a rendszer a 11 funkcióhoz hasonlóan megkeresi a megadott directory-ban az első megfelelő file-t, és visszatéréskor a kurrens DTA-ra (lásd "Hagyományos file-kezelés" fejezet) a következő információkat tölti:
21 byte - foglalt, a további keresések használják;
1 byte - a file attributuma;
2 byte - a file létrehozásának ideje;
2 byte - a file létrehozásának dátuma;
4 byte - a file hossza byte-okban;
13 byte - a file neve és típusa: 0-val lezárt string, melyben a név és a típus a szokásos alakban, pont-karakterrel elválasztva, szóközök nélkül jelenik meg.
Figyeljük meg, hogy ez a funkció valamivel kevesebbet ad vissza a 11 funkciónál, de könnyebben használható formában.

Hibakódok AX-ben, Carry=1 esetén:

03 - a megadott directory-út (path) nem létezik
12 - nincs több file - nem talált több, a specifikációnak megfelelő file-t

4F Find Next File, DOS_FDNEXT
A következő megfelelő file megkeresése - hasonlóan viszonyul 4E-hez, mint a 12 funkció a 11-hez. Vagyis, ha a 4E-vel már megtaláltuk a keresett bejegyzések közül az elsőt, akkor ugyanazon paraméterekkel a 4F funkció ismételt hívásaival kereshetjük meg sorban a többit.
Hibakódok AX-ben, Carry=1 esetén:

03 - a megadott directory-út (path) nem létezik
12 - nincs több file - nem talált több, a specifikációnak megfelelő file-t

b) File-ok törlése tetszőleges directory-ból

41 Delete Directory Entry, DOS_DELENTRY
File törlése - a DS:DX-ben kell megadnunk egy 0-val lezárt ASCII-string címét. Ebben adjuk meg a törölni kívánt file(ok) teljes specifikációját. A "*" és "?" karakterek használata csak a file-névben és -típusban megengedett. Az írásvédett file-ok törlése e funkcióval nem lehetséges. (A törlés előtt a 43 funkcióval meg kell változtatnunk e file-ok attributumát.)
Hibakódok AX-ben, Carry=1 esetén:

02 - a megadott file nem létezik
05 - tiltott elérés - írásvédett file-t akartunk törölni

c) Directory-bejegyzés módosítása

56 Change Directory Entry - DOS_CHENTRY
File átnevezése - belépéskor DS:DX-ben megadjuk a megváltoztatandó file-t specifikáló ASCII-string, ES:DI-ben pedig az új nevet tartalmazó string címét. Mindkét

stringet 0-val kell lezárni. Ha megadunk drive-nevet, annak mindkét stringben azonosnak kell lennie.
Ha nem adunk directory-nevet, akkor a kurrens directory-ban a megadott file nevét módosítja a rendszer.
Megadhatunk tetszőleges directory-t is. Ha a két file-specifikációban a directory-k neve azonos, akkor szintén a file-név módosítására kerül sor. Ha különböző directory-kat adunk meg, akkor a rendszer átmásolja az első directory megadott bejegyzését a másodikban megadott directory-ba anélkül, hogy a file adatrészét átmásolná. A UNIX-ban ezt a műveletet MV-nek ("move" rövidítése), mozgatásnak nevezik. Az MS-DOS felhasználói szinten nem ismeri ezt a parancsot.
Megjegyezzük még, hogy (eltérően a 17 funkciótól) határozatlan file-név használata nem megengedett; sehol nem adhatunk meg "*" és "?" karaktereket.
Hibakódok AX-ben Carry=1 esetén:

02 - a megadott file nem létezik
03 - a megadott directory-út (path) nem létezik
05 - az elérés tiltott:
  - az első specifikáció directory-t ad meg
  - a második specifikáció létező file-t ad meg
  - a második directory-bejegyzést nem lehet megnyitni
11 - nem ugyanaz az eszköz: különböző lemezt adtunk meg

43 Get/Set File Attribute, DOS_GSATTR
File-attributum lekérdezése/megváltoztatása - DS:DX mutat a file nevét tartalmazó, 0-val lezárt stringre. Ha AL-ben 0-t adunk meg, a rendszer CX-ben adja vissza a file attributumát. Ha AL-ben 1-et adunk meg, a rendszer a megadott file attributumát a CX-ben átadottra változtatja (lásd a "Directory-bejegyzés" fejezetben).
Hibakódok AX-ben Carry=1 esetén:

02 - a megadott file nem létezik
03 - a megadott directory-út (path) nem létezik
05 - tiltott elérés - lemezcimke vagy directory attributumának 08 illetve 10 bitjét akartuk megváltoztatni, ami illegális

57 Get/Set Date & Time, DOS_GSDTTM
File keletkezési dátumának és idejének lekérdezése vagy beállítása - belépéskor BX-ben egy megnyitott file-sorszámot adunk meg.
AL=0 - az idő lekérdezése; ekkor DX-ben kapjuk vissza a dátumot, és CX-ben az időt.
AL=1 - az idő módosítása; DX-ben adjuk meg a kívánt új dátumot, CX-ben pedig az időt.
A dátum és az idő szerkezete megfelel a directory-bejegyzésben megadott alaknak (lásd "Directory-bejegyzés" fejezet).
Hibakódok AX-ben Carry=1 esetén:

01 - az alfunkció kódja illegális
06 - érvénytelen file-sorszám - a sorszám nem létezik vagy nincs megnyitva

IX.3. Példaprogramok a directory-kezeléshez

IX.3.1. Globális directory-kezelés

A directory-król szóló fejezet első példája egyszerűen létrehoz egy subdirectory-t a kurrens directory alatt, belelép (azaz kiválasztja mint kurrens directory-t), lekérdezi, és kiírja a standard output file-ra a kurrens directory nevét, majd visszalép, és törli az általa létrehozott directory-t.

```
TITLE   Directory-létrehozás, -váltás és -törlés
;
; Külső szimbólumok
;
        EXTRN   PRTERRMSG:NEAR
;
; Konstansok, makrók, rekordok
;
        .XLIST                  ;DOSCALL.INC-et nem listázzuk!
        INCLUDE         DOSCALL.INC
        .LIST                   ;
```

```
;
; Kódszegmens
;
CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:CODE, ES:CODE, SS:CODE
        ORG     100H
ENTRY:                                  ;
                JMP     START           ;

NEWDIR          DB      'newdir', 0     ;Az új directory neve
PREVDIR         DB      '..', 0         ;Fölöttünk levő dir. !!
DISK            DB      'C:\'
CURDIR          DB      SIZ_PATH DUP ( ? )      ;Lekérd.buffer

START:                                  ;
                MOV     DX, OFFSET NEWDIR
                DOSCALL DOS_MKDIR       ;Létrehozzuk
                JC      ERR_EXIT        ;
                MOV     DX, OFFSET NEWDIR
                DOSCALL DOS_CHDIR       ;Beleváltunk
                JC      ERR_EXIT        ;
                MOV     DL, DR_C        ;
                MOV     SI, OFFSET CURDIR
                DOSCALL DOS_GTCDIR      ;Lekérdezzük
                JC      ERR_EXIT        ;
                XOR     AL, AL          ;
                MOV     CX, SIZ_PATH    ;
                MOV     DI, OFFSET CURDIR
        REPNE   SCASB                   ;Keressük a 0-t
                DEC     DI              ;
                MOV     BYTE PTR [ DI ], CHR_DOLLAR ;Nevét lez.
                PRTSTR  DISK            ; és kinyomtatjuk
                MOV     DX, OFFSET PREVDIR
                DOSCALL DOS_CHDIR       ;Vissza egy szintet
                JC      ERR_EXIT        ;
                MOV     DX, OFFSET NEWDIR
                DOSCALL DOS_RMDIR       ;Töröljük az új dir.-t
                JC      ERR_EXIT        ;
                JMP     SUCC_EXIT       ;
                EXIT                    ;EXIT makró hívása
CODE    ENDS                            ;
        END     ENTRY                   ;
```

Egyetlen apró megjegyzés kívánkozik e program mögé. Mit kell tennünk akkor, ha ezt az egészet nem a kurrens, hanem a ROOT directory alatt akarjuk elvégezni? Nos, a dolog nagyon egyszerű: a "path" elé oda kell biggyesztenünk egy "\" (backslash) karaktert, amely jelzi a rendszernek, hogy az egész műveletet a ROOT directory-ban kell kezdeni.

IX.3.2. File-név módosítása - file-mozgatás

Ez a program egyszerűen módosítja az első és a második paraméterben megadott file nevét. Ennél azonban sokkal-sokkal fontosabb és hasznosabb a "melléktermék": ha a paraméterekben két különböző directory nevét adjuk meg, akkor a program "átemeli" a bejegyzést az egyik directory-ból a másikba a file testének (adatterületének) másolása nélkül. Ezt a lehetőséget az MS-DOS felhasználói felülete fájdalmasan nélkülözi!

```
TITLE   File átnevezése vagy mozgatása
;
; Külső objektumok
;
        EXTRN   GETPAR:NEAR, PRTERRMSG:NEAR
;
; Konstansok, makrók, rekordok
;
        .XLIST                  ;DOSCALL.INC-et nem listázzuk!
        INCLUDE         DOSCALL.INC
        .LIST                   ;
;
; Kódszegmens
;

CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:CODE, ES:CODE, SS:CODE
        ORG     100H

ENTRY:                                  ;
                JMP     START           ;

OLD_PNT         LONGPNT         <>      ;Első paraméter címe
NEW_PNT         LONGPNT         <>      ;Második paraméter címe
```

```
START:                                  ;
                MOV     DI, OFFSET OLD_PNT
                CALL    GETPAR          ;Első par. elkülönítése
                MOV     DI, OFFSET NEW_PNT
                CALL    GETPAR          ;Második par. elk.
                LES     DI, NEW_PNT     ;ES:DI-be a második,
                LDS     DX, OLD_PNT     ;DS:DX-be az első címe
                DOSCALL DOS_CHENTRY     ;Bejegyzés módosítása
                JC      ERR_EXIT        ;
                EXIT                    ;Kilépés makróval
CODE    ENDS                            ;
        END     ENTRY
```

IX.3.3. File keletkezési idejének frissítése

Felületes kollégák örömére készült a most olvasható program. Egyszerűen annyit tesz, hogy a paraméterként megadott file keletkezésének dátumát és idejét a kurrens dátumra és időre módosítja (lásd még "Kisegítő MS-DOS funkciók" fejezet). Ez akkor lehet hasznos, ha a rendszer újratöltése után (ami programfejlesztés közben tízpercenként bekövetkezik) elmulasztottuk a dátum és az idő beállítását, de az elkészült és elmentendő fileokon precíz dátumot akarunk kapni.

```
TITLE   File keletkezési idejének módosítása
        EXTRN   GETPAR:NEAR, PRTERRMSG:NEAR
;
; Konstansok, makrók, rekordok
;
        .XLIST                  ;DOSCALL.INC-et nem listázzuk!
        INCLUDE         DOSCALL.INC
        .LIST
;
; Kódszegmens
;

CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:CODE, ES:CODE, SS:CODE

        ORG     100H                    ;
```

```
ENTRY:                                  ;
                JMP     START           ;

PAR_ADR         LONGPNT         <>
HND             DW      0
DATE_CURR       DATE_R          <>
TIME_CURR       TIME_R          <>

        MSGDEF  ERR_INVDT, 'Dátum vagy idő rossz!'

START:                                  ;-------;
                DOSCALL DOS_GTDATE              ;Belső dátum
                SUB     CX, DOS_GSDTTM_BASE     ;Kiigazítjuk
                SET_EL  DATE_CURR, CX, YEAR     ;Előkészítjük
                MOV     AL, DH                  ;
                CBW                             ;
                SET_EL  DATE_CURR, AX, MONTH    ; a DATE_CURR
                MOV     AL, DL                  ;
                CBW                             ;
                SET_EL  DATE_CURR, AX, DAY      ; változót
                DOSCALL DOS_GTTIME              ;Belső idő lek.
                MOV     AL, CH                  ;
                CBW                             ;
                SET_EL  TIME_CURR, AX, HOUR     ;Előkészítjük
                MOV     AL, CL                  ;
                CBW                             ;
                SET_EL  TIME_CURR, AX, MINUTE   ; a TIME_CURR
                MOV     AL, DH                  ;
                CBW                             ;
                SHR     AX, 1                   ;
                SET_EL  TIME_CURR, AX, TWOSEC   ; változót
                                                ;
                MOV     DI, OFFSET PAR_ADR      ;
                                                ;File-név
                CALL    GETPAR                  ; paraméterként
                MOV     DX, WORD PTR PAR_ADR[ OFFS ]
                DOSCALL DOS_OPEN, 0             ;Megnyitjuk
                JC      ERR_EXIT                ;
                MOV     HND, AX                 ;
                MOV     BX, AX                  ;Lek. a dátumát
                DOSCALL DOS_GSDTTM, DOS_GSDTTM_GET  ; és idejét
                JC      ERR_EXIT                ;
```

```
                CMP     DX, DATE_CURR           ;Kur. korábbi?
                JA      ILL_DATE                ;Igen, baj van!
                JB      DTTM_OK                 ;
                CMP     CX, TIME_CURR           ;
                JA      ILL_TIME                ;
DTTM_OK:                                        ;
                MOV     DX, DATE_CURR           ;
                MOV     CX, TIME_CURR           ;
                MOV     BX, HND                 ;Mód. a dátumát
                DOSCALL DOS_GSDTTM, DOS_GSDTTM_SET  ; és idejét
                JC      ERR_EXIT                ;
                JMP     SUCC_EXIT       ;-------;
ILL_DATE:                               ;Dátum vagy idő illeg.,
ILL_TIME:                               ; mert a file "később"
                PRTSTR  ERR_INVDT       ; keletkezett, mint a
                JMP     SUCC_EXIT       ; kurrens dátum!
                                        ;
                EXIT                    ;
CODE    ENDS                            ;

        END     ENTRY
```

A fenti kis program egy kis aprósággal kombinálva hasznos kis segédprogram lehet! Módosítsuk úgy, hogy egy szövegfile-ból olvassa ki azon file-ok specifikációját, amelyeknek keletkezési idejét rendszeresen frissíteni kívánjuk. Kérdezzen rá határozottan a dátumra és az időre (esetleg összevetheti a választ a file-ok valódi keletkezési idejével), majd állítsa át minden, a szövegfile-ban megadott file keletkezési idejét a kapott adatok szerint. Ezután minden fontos file-unk keletkezési ideje aránylag rendben lesz.

IX.3.4. Rejtett file-ok és directory-k listázása

Az operációs rendszerről eddig szerzett ismereteinket úgy is kamatoztathatjuk, hogy gondosan felderítünk minden olyan rejtett file-t, amelyet kollégáink nagy fáradsággal igyekeztek eldugni a kandi szemek elől. Ebben lehet segítségünkre az alábbi program (mely COM formátumú - ilyennel úgy is régen találkoztunk).

```
TITLE   Láthatatlan és rendszerfile-ok listázása
        EXTRN   GETPAR:NEAR, PRTERRMSG:NEAR
;
; Konstansok, makrók, rekordok
;
        .XLIST                      ;DOSCALL.INC-Et nem listázzuk!
        INCLUDE         DOSCALL.INC
        .LIST
;
; Kódszegmens
;
CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:CODE, ES:CODE, SS:CODE
        ORG     100H

ENTRY:                                  ;
                JMP     START           ;

FNDEN           FDENT   <>              ;Struktúra a kereséshez
                DB      2 DUP ( CHR_BLANK )  ;csak elválasztás
PATH            DB      '*.*', CHR_EOS  ;File-specifikáció
        MSGDEF  MSG_VOL, 'lemezcimke  ', NOCR, NODOLLAR ;Attri-
        MSGDEF  MSG_DIR, 'directory   ', NOCR, NODOLLAR ; butum-
        MSGDEF  MSG_ARC, 'archív      ', NOCR, NODOLLAR ; nevek
        MSGDEF  MSG_RDO, 'védett    '  , NOCR, NODOLLAR
        MSGDEF  MSG_HID, 'rejtett   '  , NOCR, NODOLLAR
        MSGDEF  MSG_SYS, 'rendszer  '  , NOCR, NODOLLAR
        MSGDEF  MSG_CR,  ' '           ,     , NODOLLAR ;

START:                                  ;
                MOV     DX, OFFSET FNDEN;
                DOSCALL DOS_SETDTA      ;
FIND_FIRST:                             ;
                MOV     CX, ATR_ALL     ;
                MOV     DX, OFFSET PATH ;
                DOSCALL DOS_FDFRST      ;Első keresése
                LCJ     C, ERR_EXIT     ;Hosszú feltételes ugr.
FIND_LOOP:                              ;
                CMP     FNDEN.FDENT_ATTR, ATR_ARCHV
                LCJ     E, FIND_LOOP_CONT       ;Archív, nem l.
                OUTWRIT FNDEN.FDENT_NAME, SIZ_FILSPEC+1
                                        ;File neve ki
```

```
                TEST    BYTE PTR FNDEN.FDENT_ATTR, ATR_ARCHV
                JE      NO_ARCHV        ;
                OUTWRIT MSG_ARC         ;Archiv is, kiirjuk
NO_ARCHV:                               ;
                TEST    BYTE PTR FNDEN.FDENT_ATTR, ATR_DRTRY
                JE      NO_DRTRY        ;
                OUTWRIT MSG_DIR         ;Directory
NO_DRTRY:                               ;
                TEST    BYTE PTR FNDEN.FDENT_ATTR, ATR_VOLAB
                JE      NO_VOLAB        ;
                OUTWRIT MSG_VOL         ;Lemezcimke
NO_VOLAB:                               ;
                TEST    BYTE PTR FNDEN.FDENT_ATTR, ATR_RDONLY
                JE      NO_RDONLY       ;
                OUTWRIT MSG_RDO         ;Védett file
NO_RDONLY:                              ;
                TEST    BYTE PTR FNDEN.FDENT_ATTR, ATR_HIDDEN
                JE      NO_HIDDEN       ;
                OUTWRIT MSG_HID         ;Rejtett file
NO_HIDDEN:                              ;
                TEST    BYTE PTR FNDEN.FDENT_ATTR, ATR_SYSTEM
                JE      NO_SYSTEM       ;
                OUTWRIT MSG_SYS         ;Rendszerfile
NO_SYSTEM:                              ;
                OUTWRIT MSG_CR          ;Soremelés ki
                CALL    FNMINI          ;File-név szóközökkel!
FIND_LOOP_CONT:                         ;
                MOV     DX, OFFSET PATH ;
                DOSCALL DOS_FDNEXT      ;Ker. a köv.-t míg van
                JC      ERR_EXIT        ;
                JMP     FIND_LOOP       ;
                EXIT                    ;
                                        ;
;---------------------------------------;
; FNMINI        - file-név előkészitése szóközökkel
;               FNDEN file-név elemét ismételt string-
;               utasitással készíti elő, csupa szóközzel
;               felülírva.
;               Speciális célú rutin, csak itt használható.
;               Módosítja a Direction flaget, valamint a
;               DI, CX és AL regisztereket.
;---------------------------------------;
```

```
FNMINI  PROC    NEAR                            ;
                                                ;
                CLD                             ;Előre haladunk
                MOV     DI, OFFSET FNDEN.FDENT_NAME
                MOV     CX, SIZ_FILSPEC + 1
                MOV     AL, CHR_BLANK           ;
        REP     STOSB                           ;
                RET                             ;
                                                ;
FNMINI  ENDP                                    ;
                                                ;
CODE    ENDS                                    ;
                                                ;
        END     ENTRY
```

Természetesen mindenki előtt világos, hogy ezt a rutint sokkal általánosabban kellene megírni. Azonban akkor, ha a lehető legnagyobb gyorsaságra van szükségünk, nem mindig célszerű nagyon elegánsan programozni.

Milyen is lenne az általános megoldás? Először is, a szóközökkel előkészítendő string címét és hosszát paraméterként kellene átadni. Sőt esetleg célszerű azt a karaktert is paraméterként kezelni, amellyel fel akarjuk tölteni a buffert. Ekkor természetesen a rutin hívása előtt kell előkészíteni minden szükséges paramétert, ami megnöveli a kód hosszát akkor, ha a rutint több helyről hívjuk meg (figyelem: a futásidőt nem, hiszen a paramétereket így is, úgy is be kell állítani).

Maga a rutin menti a regisztereket (elsősorban a STATUS-t, hiszen kénytelen módosítani a Direction flaget, amihez tulajdonképpen nincs is joga). Igen hasznos lehet a CX ellenőrzése: ha a CX értéke 0, akkor az ES által címzett teljes szegmenst felülírnánk a kívánt karakterrel. Ilyen esetben tehát ki kell lépnünk a rutinból. Ezután hozzáláthatunk a munkához, a visszatérés előtt pedig gondosan helyreállítjuk a használt regiszterek tartalmát. Ez a sok munka persze a futásidőt is növeli.

Végül egy nagyon fontos megjegyzés a fenti programban elhelyezett "apró" csapdáról. Alaposan kihasználtuk azt, hogy programunk COM típusú, azaz ES és DS ugyanoda mutat. Ha EXE típusú programban csinálnánk az FNMINI rutinhoz hasonló tisztátalan trükköket, kellemetlen (mert nehezen felderíthető) hibák következnének be. Különösen veszélyes ez akkor, ha egy régebben megírt COM programot alakítunk EXE típusúvá.

X. Memóriakezelés és programvezérlés

X.1. Az MS-DOS memóriakezelése

Az MS-DOS a memóriát blokkokra osztja. Minden blokk előtt egy paragrafusnyi leíróterület van, amely szegmens:offset formában tartalmazza a blokk kezdőcímét, valamint a blokk paragrafusokban számított hosszát. A fix memóriaterületeken kívül (interrupt-vektorok, ROM BIOS adatterület és a rendszer állandóan rezidens részei) allokált blokkok szolgálnak minden célra. Például programunk elindításakor az első lépés a szükséges memória allokálása.

A memóriával három műveletet végezhetünk: megváltoztathajuk egy rendelkezésünkre álló blokk méretét, allokálhatunk egy szabad blokkot, és felszabadíthatunk egy foglalt memóriaterületet.

A memória dinamikus kezelése igen fontos és hasznos lehet olyan esetekben, amikor nem tudjuk előre, hogy a program egyes futásai során valójában mekkora területet fog használni. Ekkor jobban járunk, ha nem fordítási időben jelölünk ki feltehetően elég hosszú területet, hanem futás közben foglalunk annyi memóriát, amennyire ténylegesen szükség van.

A memória dinamikus kezelését az alábbiak szerint képzeljük el: a memória 16-tal osztható címeken kezdődő 16 byte-os egységekre oszlik. Egy ilyen egység a paragrafus (ez nem egy igazán "hardware"-meghatározott egység; azért érdemes definiálni, mert szegmens csak paragrafushatáron kezdődhet, és ennek megfelelően csak egész számú paragrafust tartalmazhat). A memória az MS-DOS számára folytonos paragrafusokból álló blokkokra oszlik. Ezek a blokkok láncba szervezettek. Egy blokkot az első paragrafusában megadott struktúra ír le; a blokk adatrésze a második paragrafuson kezdődik. A struktúra öt byte-ból áll. Első eleme byte-os, és egy speciális értéket tartalmaz. Ez 4E akkor, ha a blokk a blokkok láncában valamely közbeeső elem; 5A, ha az utolsó.

A struktúra következő két byte-ja (1 szó) a blokk tulajdonosára utal, míg az utolsó két byte (1 szó) a láncolási információ: ez tartalmazza a blokk hosszát paragrafusokban. A blokk kezdő paragrafusának címe és hossza pedig egyértelműen megadja a következő blokk kezdő paragrafusát. A hosszba egyébként nem számítják be a blokk leíró paragrafusát.

A blokk tulajdonosa mindig egy program. Alapesetben (mikor csak az MS-DOS aktív) a memória három blokkra oszlik. Az első blokk (amely a memóriában rezidensen elhelyezkedő programok fe-

lett helyezkedik el [lásd MS-DOS memóriatérképek]) tartalmazza a COMMAND.COM környezetét, a második magát a COMMAND.COM-ot, míg a harmadik a szabad terület. Mindhárom blokk tulajdonosa a COMMAND.COM, tehát a tulajdonosra utaló szó a COMMAND.COM memóriablokkja kezdő paragrafusának címét tartalmazza (nem a leíró paragrafus címét, hanem a valódi kezdőcímet; ez előtt egy paragrafussal találjuk meg a leíró struktúrát).

Ha elindítunk egy programot, akkor az MS-DOS két részre osztja a szabad területet, és mindkettőt átadja a programnak. Az első új blokkba az MS-DOS átmásolja saját környezetét az elindított programnak, a második blokk tartalmazza magát a programot. Mindkét blokk tulajdonosa a program. Amikor pedig ez átrendezi a memóriát, majd elkezd területeket foglalni és felszabadítani, akkor az MS-DOS újra léterhoz egy szabad blokkot, és ennek további bontásával állítja elő a kért szabad blokkokat. Az MS-DOS úgy jelöli meg a felszabadított blokkokat, hogy egyszerűen törli a tulajdonosra utaló szót. Amelyik blokk leíró struktúrájában ez 0, az szabad blokk.

A memóriakezelő rendszer nagy hiányossága (legalábbis a 3.20 verzióig), hogy a rendszer nem olvasztja össze a szomszédos felszabadított blokkokat. Ebből az következik, hogy a memóriafoglalás úgy szétszabdalhatja a memóriát, hogy hiába szabadítunk fel minden allokált területet, már egy rövid folytonos darabot sem tudunk foglalni.

A program kilépésekor a rendszer felszabadítja az összes olyan blokkot, amelynek a kilépő program a tulajdonosa. Azonban a memória szabdalt volta nem szűnik meg, ha allokált és fel nem szabadított blokkokat hagyunk meg. Ez egy elég komoly veszélyforrás, amellyel számolnunk kell.

Az ilyen szerencsétlen esetekben meg lehet kísérelni a memóriablokk hosszúságát megnövelni (4A [DOS_SETBLK] funkció). Ha a blokk mögötti területen van elég hosszú folytonos terület, akkor a DOS megnöveli a blokk hosszát, azaz összeolvaszt két (vagy több) szabad blokkot.

> Ide kell még iktatnom egy személyes megjegyzést, mivel könyvem megírása során a fentiek leírásával merészkedtem a legcsúszósabb terepre. Ezeket az információkat nem dokumentációból, hanem az MS-DOS visszafejtéséből és a memóriában való turkálásból szereztem, tehát könnyen előfordulhatnak kisebb-nagyobb félreértések és tévedések. Ráadásul éppen ezeket a

részeket elég nehéz nyomon követni, mivel a DEBUG és az egyéb segédeszközök (elrejtendő saját memóriakezelésüket) sok mindent módosítanak a memóriában. Az eredmények javát az AFD.COM nevű futáskövetővel értem el; azonban kínos meglepetéseket már ez is okozott. Tehát elnézést kérek azokért a csacsiságokért, amelyek esetleg bekerültek a fenti (jól tudom, hogy hiányos) leírásba.

Az allokált blokkokkal rendkívül gondosan kell bánnunk! Sem "eléjük", sem pedig "mögéjük" írni nem szabad. Gyakorlatilag persze csak a blokk túlírása reális veszély, hiszen a lefoglalt blokk valódi atatterületének szegmenscímét kapjuk meg. A blokk túlírása arra vezet, hogy a következő blokk leíró része sérül meg, ez pedig végzetes MS-DOS hibát okozhat. A lefoglalt területeket minden esetben ajánlatos felszabadítani a program kilépése előtt, bár az MS-DOS felszabadítja a program által lefoglalt blokkokat.

X.2. Memóriakezelő funkciók

48 Allocate Memory, DOS_MALLOC
Memóriafoglalás - belépéskor BX tartalmazza a kért paragrafusok (dec. 16 byte-os blokkok) számát. Visszatérés után AX:0 a memóriablokk valódi kezdőcíme, tehát AX tartalmazza a szegmensregiszterbe töltendő értéket (AX tehát nem a leíró blokkra, hanem a használható terület kezdetére mutat!). Ha a területfoglalás nem sikeres, BX-ben kapjuk vissza a legnagyobb memóriablokk paragrafusokban mért hosszát.

Megjegyzés: a rendszer a program belépésekor minden rendelkezésre álló memóriát lefoglal a program számára. A program tehát az utolsó memóriablokkban kap helyet, mely a RAM terület tetejéig tart. Ezért, mielőtt bármilyen dinamikus memóriakezelést igénylő műveletbe fognánk, a felesleges memóriaterületet a 4A [DOS_SETBLK] funkcióval fel kell szabadítanunk, azaz az utolsó blokk hosszát le kell csökkenteni.

Hibakódok AX-ben Carry=1 esetén:

07 - a memórialeíró blokk megsérült - dinamikus memóriakezelés során hiba lépett fel
08 - nincs elég memória - dinamikus memóriakezelés során elfogyott a memória (ekkor BX tartalmazza a legnagyobb elérhető blokk hosszát)

49 Free Allocated Memory, DOS_MFREE
A lefoglalt blokk felszabadítása - belépéskor ES-ben adjuk meg a felszabadítani kívánt (előzőleg 48-al allokált) blokk paragrafuscímét, melyet a 48 funkció AX-be adott vissza; tehát nem a leíró blokk, hanem a valóban szabad memória kezdőcímét. Fontos tudnunk, hogy a programunk által a 48 funkcióval allokált memóriát szabad csak így felszabadítani!
Hibakódok AX-ben Carry=1 esetén:

07 - a memórialeíró blokk megsérült - dinamikus memóriakezelés során hiba lépett fel
09 - érvénytelen blokkcím - dinamikus memóriakezelés során érvénytelen címet adtunk meg

4A Set Memory Block Size, DOS_SETBLK
Memóriablokk hosszának módosítása - belépéskor ES-ben adjuk meg a módosítani kívánt memóriablokk paragrafuscímét (nem a leíró blokk címét), BX pedig tartalmazza a kívánt blokkhosszúságot paragrafusokban. A rendszer megkísérli a kívánt csökkentést vagy növelést. Ha a művelet nem sikeres (csak növelés során képzelhető el, hiszen csökkentés csak a leíró blokkok teljes lepusztítása után lehet sikertelen), BX-ben kapjuk meg a lehetséges legnagyobb növelés mértékét paragrafusokban.
E funkció alkalmas arra, hogy két szomszédos memóriablokkot összeragasszon, ha az alacsonyabb című a miénk és a magasabban lévő szabad.

Megjegyzés: EXE programban hasznos trükk lehet, hogy definiálunk egy MEMORY nevű szegmenst, amely a program végén kap helyet, és teljesen üres. Ekkor ennek a címe megmutatja programunk fizikai végét a

tárban. Ezt használhatjuk fel a feltétlenül szükséges paragrafusok számának meghatározására.

```
MOV     BX, SEG MEMORY  ;Program vége
MOV     AX, DS
SUB     BX, AX          ;PSP paragrafusa
MOV     AX, 4A00H       ;Funkciókód és 0
INT     21H
```

Ez a kód természetesen közvetlenül a program belépése után alkalmazható így, ameddig egy szegmensregiszter tartalmát sem módosítottuk, és DS és ES a PSP-re mutat!

COM típusú programban úgy járhatunk el, hogy a CODE szegmens végére elhelyezünk egy címkét, mely az utolsó felhasznált byte mögé mutat. Ezt használjuk fel aztán a program hosszának meghatározására. Gondoskodnunk kell azonban arról, hogy a verem területe a megváltoztatott méretű blokkon belül maradjon. Ne felejtsük el azt sem, hogy az esetlegesen beszerkesztett egyéb modulok vagy könyvtári függvények a mi utolsó pozíciónk után kaphatnak helyet a memóriában. A program valódi hosszának meghatározása ilyenkor ugyancsak bonyodalmas feladat (lásd a példaprogramokat)!

Hibakódok AX-ben Carry=1 esetén:

07 - a memórialeíró blokk megsérült - a memóriakezelés során hiba lépett fel (gyakorlatilag valamelyik blokk túlírása következtében)

08 - nincs elég memória - elfogyott a memória, pontosabban nincs eléggé hosszú szabad terület a blokk méretének megnövelésére

09 - érvénytelen blokkcím - dinamikus memóriakezelés során érvénytelen címet adtunk meg

MS-DOS 3.1 és későbbi verziók

58 Get/Set Allocation Strategy, DOS_GSALSTR

Amennyiben valamilyen célra (pl. egy program betöltésére) memóriát kell allokálni, az MS-DOS a szabad terület egy blokkját használja fel erre a célra. Ha a szabad blokk nagyobb, mint kellene, akkor "elvágja", és két kisebb blokkot készít belőle.

Ezek a verziók háromféle stratégiát követhetnek a memóriafoglalás során. A statégiák és kódjaik a következők:

0 - First fit - az első (legalacsonyabb című) megfelelő blokk kiválasztása
1 - Best fit - a legkisebb, még elegendően hosszú blokk kiválasztása
2 - Last fit - az utolsó (legmagasabb című) megfelelő blokk kiválasztása

Alfunkciók:
AL=0 - a funkció AX-ben adja vissza a pillanatnyilag kiválasztott stratégia kódját.
AL=1 - BX-ben adjuk meg a használni kívánt stratégia kódját.

Hibakód Carry=1 esetén:

01 - érvénytelen alfunkciókód

Megjegyzés: az IBM kézikönyvekben úgy szerepel, mint a DOS belső használatára fenntartott funkció; ismertetését csak Microsoft kiadványban találtam meg. Tehát olyan "félig dokumentált" funkciónak tekinthető.

X.3. Programvezérlés programból

Az MS-DOS egyik legkellemesebb sajátossága, hogy jóllehet egy program futtatására képes (hiszen nem multiprogramozott rendszer), mégis támogatja azt, hogy több program legyen egyszerre a memóriában. Igaz, hogy a programok vezérlését nekünk kell megvalósítani, az is igaz, hogy (általában) nem egymástól független programok futnak így, de mégis van lehetőség programunkból újabb programok futtatására, sőt e programok egymással interaktív kapcsolatban is állhatnak.

Az egyidejűleg futó programok egymás közötti kommunikációját semmiképpen ne direkt rutinhívásokkal valósítsuk meg! Erre a célra szolgál a software interrupt, és 3.1 és későbbi MS-DOS verziókban az erre kijelölt multiplex interrupt, melynek sorszáma 2F (lásd "Az interruptok kiosztása" fejezetet).

X.4. Programvezérlési funkciók

4B Load and Execute Program, DOS_EXEC
Egy program betöltése vagy futtatása - kettős célú DOS hívás. Belépéskor DS:DX-ben adjuk meg egy 0-val lezárt ASCII-string címét, amely a futtatni kívánt program teljes specifikációját tartalmazza.

Megjegyzés: a rendszer számára megadott PATH nem vonatkozik a mi programunkra. Ha ezt közvetlenül a DOS indította el, akkor a rendszer átmásolja saját környezetét nekünk, melyben megtaláljuk az érvényes PATH-t. Innen ismerhetjük az aktivált directory-kat, bár ez elég barátságtalan munka.

ES:BX-ben egy paraméterblokk címét kell megadnunk, amelynek szerkezete függ az alfunkcióktól.

Alfunkciók és a hozzájuk tartozó blokkok:

AL=0 - programvégrehajtás esetén (javasolt konstansnév az alfunkció számára DOS_EXEC_PRC):

WORD	- az elindított programnak átadandó környezet szegmenscíme
DWORD	- pointer az elindított program PSP-je 80. pozícióján elhelyezendő parancssorra
DWORD	- pointer az elindított program PSP-je 5C. pozícióján elhelyezendő (elsődleges) FCB-re
DWORD	- pointer az elindított program PSP-je 6C. pozícióján elhelyezendő (másodlagos) FCB-re

A rendszer betölti a kívánt programot úgy, hogy szabványos PSP-t készít számára, és ugyanúgy indítja el a programot, mintha a parent, az elindító program maga a COMMAND.COM volna (kivéve persze a környezet kezelését - nem a saját eredeti környezetét adja át a programnak).

AL=3 - overlay ág beolvasása esetén (javasolt konstansnév az alfunkció számára DOS_EXEC_OVL):

WORD - a betöltés helyének szegmense
WORD - az alkalmazandó relokációs faktor

Ovarlay-ág betöltése esetén nem készül szabványos PSP, a kért program csak betöltődik a tár általunk megadott címére, és a rendszer elvégzi a relokálást. A betöltött program elindításáról egy megfelelő vezérlésátadó utasítással a parent program gondoskodik a kívánt időben.
Ez a funkció alkalmas arra, hogy egy COM típusú programot úgy futtassunk le, mintha saját programunk része volna: saját kódszegmensünkbe töltjük be, és (mivel nem készül PSP) az elindított program a mi PSP-nket és a mi környezetünket használja. Teljesen olyan, mintha egy szubrutin volna.

Fontos megjegyzés: ha lefuttatunk egy programot, minden regiszter, a Stack Segment és a Stack Pointer értéke is megváltozhat, ezért programhívás előtt legalább SS-t és SP-t el kell menteni saját adatterületünkre!
Kevésbé fontos megjegyzés: tapasztalatok szerint a 3.00 és későbbi verziójú rendszerek nem rontják el a Stack Segment és a Stack Pointer értékét.

MS-DOS 3.00 és későbbi verziók
Megjegyezzük még, hogy a 3.00 feletti rendszerek LINK programjai lehetővé teszik az overlay megvalósítását minden ilyen trükk nélkül - lásd a megfelelő kézikönyveket (Disk Operating System Manual).

4D Get Return Code of Child Process, DOS_GTRETCD
A program által elindított másik program (sub-process) visszatérési kódjának lekérdezése - AX-ben találjuk a kódot: AL származik az elindított programtól, AH pedig a program kilépésének okáról ad felvilágosítást:

00 - normális kilépés
01 - kilépés CTRL-BREAK-kel
02 - kilépés kritikus device-hiba miatt
03 - kilépés a 31 funkcióval

26 Create New Program Segment, DOS_CRPRSEG
Egy új programszegmens létrehozása - DX-ben egy szegmenscímet kell átadni, ahol az új programszegmenst létre kívánjuk hozni. A kurrens programszegmens első 100H byte-ját a rendszer átmásolja az új szegmensbe, azaz az elindító program PSP-je kerül át az új programszegmensbe. Ez a funkció overlay-ágak vagy egyéb programok beolvasása, lefuttatása előtt használatos. Minthogy azonban erre a célra rendelkezésünkre áll egy sokkal jobb lehetőség (4B), kerüljük e funkció használatát!

X.5. Memóriakezelési és programvezérlési példák

X.5.1. Memóriafoglalás és -felszabadítás

E fejezet első példája mindjárt két példaprogram. Mindkettő egyszerűen annyit tesz, hogy a számára felesleges memória felszabadítása után allokál egy szabad memóriablokkot, teleírja, kinyomtatja a standard outputra, és felszabadítja a lefoglalt memóriablokkot. Miért kellett rögtön két változatban megírni ezt a programot? Az ok egyszerű: egészen más stratégiát kell követnie egy EXE, mint egy COM típusú programnak.

```
TITLE   Memóriafoglalás és felszabadítás EXE programban
;
; Konstansok, makrok, rekordok
;
        .XLIST                      ;DOSCALL.INC-et nem listázzuk!
        INCLUDE         DOSCALL.INC
        .LIST

BLK_SIZE        EQU     'z' - ' '       ;Fontos kar.-ek száma
BLK_PARN        EQU     BLK_SIZE/16 + 1 ;Paragrafusokban
;
; Adatterületek
;
DATA    SEGMENT PARA    PUBLIC  'DATA'
BLK_ADDRESS     DW      0       ;Blokk szegmenscíme
DATA    ENDS                    ;
```

```
;---------------------------------;
; Stack-terület
;-----------------------------------------;

STACK   SEGMENT PARA    STACK   'STACK'
        DW      100     DUP ( ? )       ;Hossza 100 szó
STACK   ENDS

;
; Kódszegmens
;

CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:DATA, SS:STACK, ES:NOTHING

PSP_SEG         DW      0               ;
                                        ;
START:                                  ;
                MOV     CS:PSP_SEG, DS  ;PSP címét mentjük
                MOV     AX, DATA        ;
                MOV     DS, AX          ;DS előkészítése
                CALL    CUT_EXE         ;Memóriablokk levágása
                JC      ERR_EXIT        ;Hiba - nagy baj van!
                MOV     BX, BLK_PARN    ;
                DOSCALL DOS_MALLOC      ;Allokálunk egy blokkot
                JC      ERR_EXIT        ;Itt se lehet hiba!
                MOV     BLK_ADDRESS, AX ;Elmentjük a címet
                MOV     ES, AX          ;Letöltjük ES-be
                ;-----------------------;
                ; Kéne egy ASSUME, de mihez rendeljük ES-t ??
                ;-----------------------;
                MOV     CX, 'z' - ' '   ;Ciklusszámláló    (**)
                MOV     DI, 0           ;ES:0 offset beállítása
                MOV     AL, ' '         ;Első látható karakter
FILL_LOOP:                              ;
                STOSB                   ;Kiírjuk
                INC     AL              ;Kódot megnöveljük
                LOOP    FILL_LOOP       ;Ism. a blokk végéig
                                        ;
                PUSH    DS              ;
                PUSH    ES              ;
                POP     DS              ;DS <-> ES csere !
```

```
                OUTWRIT 0, BLK_SIZE, HND_STD_OUT ;Blokk-kiírás
                POP     DS               ;DS helyreállítása (**)
                                         ;
                MOV     AX, BLK_ADDRESS  ;
                MOV     ES, AX           ;
                DOSCALL DOS_MFREE        ;A blokk felszabadítása
                                         ;
                EXIT                     ;Kilépés makróval
                                         ;
;----------------------------------------;
; CUT_EXE - Felesleges memória felszabadítása EXE programban
;       Input:
;               semmi
;       Output:
;               Carry   - 0, ha sikeres
;               Carry   - 1, ha nem, ekkor
;                         BX - szabad paragrafusok száma
;----------------------------------------;
CUT_EXE         PROC    NEAR             ;
                                         ;
                PUSH    CX               ;
                PUSH    ES               ;
                MOV     CX, CS:PSP_SEG   ;Programblokk kezdete
                MOV     BX, SEG MEMORY   ;Programblokk vége
                SUB     BX, CX           ;Hossz-számítás
                MOV     ES, CX           ;Kezdőcím ES-be
                DOSCALL DOS_SETBLK       ;Blokkhossz csökkentése
                POP     ES               ;
                POP     CX               ;
                RET                      ;
                                         ;
CUT_EXE         ENDP                     ;
                                         ;
CODE    ENDS                             ;
                                         ;
;----------------------------------------;
; MEMORY - üres szegmens legfelülre
;----------------------------------------;

MEMORY  SEGMENT PARA    MEMORY  'MEMORY'
MEMORY  ENDS
        END     START
```

Néhány apró megjegyzés: Miért írtuk azt megjegyzésként, hogy "nagy baj van"? Mert tényleg nagyon nagy baj van, ha egy létező memóriablokk hosszát már csökkenteni se tudjuk. Ez egy lehetetlen esemény. Ha mégis bekövetkezik, akkor pl. memóriahiba van, ami valóban nagyon nagy baj. Hasonlóan súlyos probléma az, ha felszabadítunk pár száz (ezer?) paragrafust, és nem tudunk hatot allokálni. Komoly programban persze csak az első igazán lehetetlen eset; az utóbbi elég könnyen előfordulhat, ha az ember nagy tömböknek foglal helyet.

Ismét felhívom a figyelmet a MEMORY szegmenssel elkövetett "trükkre". Ha a MEMORY típusú szegmens osztályban is különbözik valamennyi egyéb szegmenstől, a LINK a program tetejére, a legmagasabb címre fogja szerkeszteni. Tehát a MEMORY szegmenscíme összegezve paragrafusokban mért hosszával megadja a program feletti első szabad paragrafus címét. Példánkban a MEMORY szegmens hossza 0 volt.

Egy nagyon fontos dologra kell még felfigyelnünk. Most találkoztunk először olyan feladattal, amit nem lehet COM programba változatlanul átvinni. Ez arra fog indítani bennünket, hogy esetleg hozzunk létre még egy könyvtárat az EXE-specifikus szubrutinok számára. Ha ebben elhelyezzük majd CUT_EXE-t, talán nem butaság mindjárt a MEMORY szegmenst is odafordítani!

Persze így (legalább) három könyvtárra lesz szükségünk: egyegyre az általános, a COM és az EXE specifikus rutinok számára. Mit jelent az, hogy "legalább"? Eddig minden rutinunk típusa NEAR volt. De már sejtjük a szörnyű jövőt: a fejlődés során nem kerülhetjük el, hogy programjaink mérete meghaladja az egy szegmensnyit. Ekkor persze a rutinokat "távolról" fogjuk hívni, tehát olyan könyvtárak is kellenek mindhárom eddigiből, melyekben a rutinok (egy része) távoli hívású. Ez persze magával rántja az egész könyvtárat, és olyan problémákat zúdít a nyakunkba, amelyekről e pillanatban nem is álmodunk.

```
TITLE   Memóriaallokálás és felszabadítás COM programban
;;;;    EXTRN   PRTERRMSG:NEAR          ;Nem használható !!

;
; Konstansok, makrók, rekordok
;
        .XLIST                  ;DOSCALL.INC-et nem listázzuk!
        INCLUDE         DOSCALL.INC
        .LIST                   ;
```

```
BLK_SIZE        EQU     'z' - ' '       ;Fontos kar.-ek száma
BLK_PARN        EQU     BLK_SIZE/16 + 1 ;Hossz paragrafusokban

;
; Kódszegmens
;
CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:CODE, SS:CODE, ES:NOTHING
                ORG     100H

ENTRY:                                  ;
                JMP     START           ;Ugrás a valódi startra
                                        ;
BLK_ADDRESS     DW      0               ;Blokk szegmenscíme
                    ;-------------------;
        EVEN                            ;Stack páros címre !!!
STACK           DW      100  DUP ( ? )  ;Hossza 100 szó
STACK_TOP       LABEL   WORD            ;Most ez is kell!
                    ;-------------------;
START:                                  ;
                MOV     SP, OFFSET STACK_TOP    ;Stack áthely.
                CALL    CUT_COM         ;Memóriablokk levágása
                JC      ERR_EXIT        ;Hiba - nagy baj van!
                MOV     BX, BLK_PARN
                DOSCALL DOS_MALLOC      ;Blokk lefoglalás
                JC      ERR_EXIT        ;Hiba - ez is nagy baj!
                MOV     BLK_ADDRESS, AX ;Blokkcím mentése
;---------------------------------------;
; Lásd előző példát, a (**)-al jelzett sorokat
;---------------------------------------;
                MOV     AX, BLK_ADDRESS ;
                MOV     ES, AX          ;
                DOSCALL DOS_MFREE       ;A blokk felszabadítása
                JNC     SUCC_EXIT       ;
ERR_EXIT:                               ;
;               CALL    PRTERRMSG       ;Nem létezik, lehetett
                MOV     AL, TRM_DTERR   ; volna EXIT NOMESS
                JMP     PROG_EXIT       ;
SUCC_EXIT:                              ;
                MOV     AL, TRM_SUCC    ;
PROG_EXIT:                              ;
                DOSCALL DOS_EXIT        ;
```

```
;-----------------------------------------;
; CUT_COM - Felesleges memória felszabadítása COM programban
;       Input:
;               semmi
;       Output:
;               Carry   - 0, ha sikeres
;               Carry   - 1, ha nem; ekkor
;                         BX - szabad paragrafusok száma
;-----------------------------------------;
CUT_COM         PROC    NEAR            ;
                                        ;
                PUSH    CX              ;
                                        ;
                MOV     BX, OFFSET LAST_POS  ;Programhossz (?)
                MOV     CL, 4           ;
                SHR     BX, CL          ;(LAST_POS /4) + 1 adja
                INC     BX              ; a paragrafusok számát
                DOSCALL DOS_SETBLK      ;ES -> PSP !!
                                        ;
                POP     CX              ;
                                        ;
                RET                     ;
                                        ;
CUT_COM         ENDP                    ;
                                        ;
LAST_POS        LABEL   NEAR            ;"Itt a végcsapás, nem
CODE    ENDS                            ; Világoson!"

        END     ENTRY
```

Természetesen van mondanivalónk ezzel a kóddal kapcsolatban is. Eddig olyan jól bevált a PRTERRMSG és az EXIT használata; miért nem élünk ezzel a lehetőséggel most is? Nos, képzeljük el, hogy a PRTERRMSG rutint tartalmazó könyvtárat hozzászerkesztjük a programhoz. Kódja persze az általunk definiált területek mögé kerül majd, és ... LAST_POS cimkénk nem az utolsó értékes byte mögé, hanem a PRTERRMSG első byte-jára mutat. Amikor pedig meghívjuk a CUT_COM rutint, akkor lemondunk a PRTERRMSG-t tartalmazó területről is. Amikor allokálunk, akkor visszakapjuk ennek a területnek a roncsait (a rendszer beletette a blokkheadert!). Programunk viszont ismeri a régi (és már elpusztított) PRTERRMSG offsetjét, és mit sem sejtve meghívja, ha valami hiba

van; ekkor a vezérlés az allokált blokk headerjére adódik. Erről talán ennyit. Itt tehát, ha baj van, akkor még az EXE változatnál is nagyobb baj van! Tanulság: "suszter maradjon a kaptafánál", COM program ne allokáljon memóriát, mert ez szükségképpen a szegmensregiszterek módosításával jár, ami pedig túllépi egy COM program hatáskörét, és nem is igen illik.

X.5.2. Egy interaktív SHELL parancs

Okosabb programok (pl. jobb szövegszerkesztők stb.) általában rendelkeznek olyan lehetőséggel, hogy ideiglenesen kiléphetünk belőlük az MS-DOS-ba, és lefuttathatunk néhány MS-DOS parancsot (pl. formázhatunk egy lemezt, vagy egyéb műveletet végezhetünk) anélkül, hogy a használt programból végleg kilépnénk. Amikor befejeztük az MS-DOS-al végzendő műveleteket, egy EXIT parancs kiadásával visszaléphetünk abba a programba, amelyben voltunk, és folytathatjuk a munkát. Ezt a parancsot DOS-nak vagy SHELL-nek nevezik. Vizsgáljuk meg, mi is történik ebben az esetben, miért nevezik ezt a parancsot SHELL-nek!

A SHELL jelentése "kagyló"; programozástechnikai jelentése (mint annyi minden) a UNIX operációs rendszerből származik. A UNIX-beli shell egy parancs-interpreter, amelyet a felhasználó magával a bejelentkezéssel indít el. Ez menedzseli parancsait: elindítja a szükséges programokat, elvégzi a szükséges adminisztrációs tevékenységet stb. Azért interpreter, mert saját programozási nyelvén, parancsfile-okkal vagy terminálról vezérelve egészen komoly programozási feladatokat oldhatunk meg anélkül, hogy valódi, lefordítandó programot kellene írnuk. Ilyen persze régebben is volt; a UNIX shelljeiben (több is van, C-shell, Bourne-shell stb.) az a zseniális, hogy az interpreter parancsai keverhetők segédprogramok futtatásával (és a segédprogramok futásának eredménye sokkal könnyebben használható fel további feldolgozásra, mint régebbi rendszerekben), így egy nagymenő UNIX-os elképesztő dolgokat tud produkálni anélkül, hogy egyetlen sor valódi programot írna; már minden csillog-villog, de ő még mindig csak a shellt és a meglevő mintegy 300 segédprogramot használta. A UNIX shell fogalom ezen a téren több a minden interaktív operációs rendszerben ismert parancssor-értelmezőnél. A SHELL tehát az operációs rendszer parancskörnyezetét jelenti.

A UNIX felhasználója menet közben válthat a shellek között. Ha pl. Bourne-shellből elindítja a C-shellt, akkor ez utóbbi úgy lép be, mint akármelyik másik program. Az eredeti shell volt a "parent", a "szülő" program, míg az új shell a "child", a "gyermek". Ebből persze újabb programok indíthatók, többek között újabb shellek, "...s megint elölről". Amikor a felhasználó megunta az éppen használt shellt, kijelentkezik. Ekkor a shell kilép, visszatér az őt elindító programhoz (ami a fenti példa szerint egy korábban elindított shell). Ha elég szorgosan váltjuk a különféle shelleket, akár tizenhatszor is kiléphetünk a UNIX-ból, mire végre hazamehetünk. A lényeg tehát az, hogy a shell is csak egy program.

Mint fent olvasható, a DOS programvezérlése némileg emlékeztet a UNIX-éra. Itt is van "parent", azaz szülő, és "child", gyerek program. A másik közös elem az, hogy a shell (alapértelmezésben a COMMAND.COM) itt sem igazán kitüntetett program. Az operációs rendszer enélkül persze nem működik, de ha az igazi COMMAND.COM segítségével elindítottunk egy programot, akkor az elindított program, mint gyermek, elindíthatja a COMMAND.COM-ot (amelyből majd egy EXIT paranccsal tudunk kilépni).

Az alábbi programot csak EXE változatban írtuk meg, mert egyúttal azt is szeretnénk bemutatni, hogyan lehet programban környezetet definiálni és azt egy meghívott programnak átadni. Ehhez persze legcélszerűbb egy olyan programot írni, amelyben több szegmens van, köztük egy külön szegmens a környezet számára. Ez pedig COM programban nem igazán volna szép.

A környezet átadására azért is szükség van, mert a második COMMAND.COM feltűnően hasonlít az elsőre. Az alábbi program olyan jól álcázza magát, hogy felületes szemlélő észre sem veszi fáradságos munkáját. Legfeljebb akkor lepődik meg, ha a régebben beállított PATH helyett egészen mást talál a COMMAND.COM környezetében. Még nagyobb a meglepetés, amikor a DOS használata közben végre kiadja az EXIT parancsot. Az általunk behívott COMMAND.COM ekkor kilép (azaz visszaadja a vezérlést nekünk), programunk nyomtalanul eltűnik, és megint az eredeti környezetet kapjuk vissza (a COMMAND.COM a környezetét a SET parancsra kilistázza). Az alábbi program lefuttatásának fontos feltétele, hogy legyen C: lemezünk, és annak ROOT directory-jában legyen egy COMMAND.COM file. Más konfigurációban a (**)-al jelzett sort értelemszerűen módosítani kell.

Megjegyzés: a feladat megoldható COM programban is, csak arról kell gondoskodni, hogy a környezet terüle-

te paragrafushatáron kezdődjön. Ez egy jól irányzott ORG direktivával aránylag egyszerűen kivihető.

```
TITLE   COMMAND.COM elindítása EXE programból
;
; Konstansok, makrók, rekordok
;
        .XLIST                  ;DOSCALL.INC-et nem listázzuk!
        INCLUDE         DOSCALL.INC
        .LIST
;---------------------------------------;
; Adatterületek
;---------------------------------------;

DATA    SEGMENT PARA    PUBLIC  'DATA'
FILESPEC        DB      'C:\COMMAND.COM', 0     ;       (**)
COMLIN          DB      0, CHR_CR       ;Egy üres parancssor
FCBNULL         FCB     <>              ;Egy üres FCB
A_COMLIN        DD      COMLIN          ;Pointerek az elemekre
A_FCB1          DD      FCBNULL
A_FCB2          DD      FCBNULL

A_EXEC_BLK      DD      EXEC_BLK        ;Végrehajtási blokk
A_FILESPEC      DD      FILESPEC

EXEC_BLK        EX_BLK  <>              ;Adatbl. programhíváshoz
DATA    ENDS
;---------------------------------------;
; Környezetszegmens
;---------------------------------------;

ENV     SEGMENT PARA    PUBLIC  'DATA'
MSGDEF  ENV_1, 'Environment   első   sor', NOCR, NODOLLAR
        DB      0               ;Záró zéró
MSGDEF  ENV_2, 'Environment második  sor', NOCR, NODOLLAR
        DB      0               ;Záró zéró
MSGDEF  ENV_3, 'Environment harmadik sor', NOCR, NODOLLAR
        DB      0               ;Záró zéró
MSGDEF  ENV_4, 'Environment negyedik sor', NOCR, NODOLLAR
        DB      0               ;Záró zéró
        DB      0               ;Záró záró zéró
ENV     ENDS                    ;
```

```
;-------------------------------
; Stack-terület
;---------------------------------------;

STACK   SEGMENT PARA    STACK   'STACK'
        DW      100     DUP ( ? )       ;Hossza 100 szó
STACK   ENDS

;---------------------------------------;
; Kódszegmens
;---------------------------------------;

CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:DATA, SS:STACK, ES:NOTHING
PSP_SEG         DW      0               ;PSP-cím
SAVE_SP         DW      0               ;Mentő terület a
SAVE_SS         DW      0               ; stack-regiszstereknek
START:                                  ;
                MOV     CS:PSP_SEG, DS  ;PSP mentése
                MOV     AX, DATA        ;
                MOV     DS, AX          ;DS előkészítése
                CALL    CUT_EXE         ;Memória átrendezése
                JC      ERR_EXIT        ;Lehetetlen esemény
        EX_BLK_INI      EXEC_BLK, ENV, A_COMLIN, A_FCB1, A_FCB2
                                ;Sajnos, egy sorba csak így fér
                MOV     CS:SAVE_SP, SP  ;
                MOV     CS:SAVE_SS, SS  ;Stack reg. mentése
                LES     BX, A_EXEC_BLK  ;
                LDS     DX, A_FILESPEC  ;
                MOV     AL, DOS_EXEC_PRC;
                DOSCALL DOS_EXEC        ;Processzhivás
                MOV     SS, CS:SAVE_SS  ;
                MOV     SP, CS:SAVE_SP  ;Stack reg. helyreáll.
                JC      ERR_EXIT        ;Sikertelen volt
                DOSCALL DOS_GTRETCD     ;Válaszkód lekérdezése
                JMP     SUCC_EXIT       ;
                EXIT                    ;Kilépés makróval
                                        ;
;---------------------------------------;
;  CUT_EXE és MEMORY definicióját lásd a "Memóriakezelés EXE
;  programból" példában
        END     START                   ;
```

A program egyetlen apró érdekessége az EX_BLK_INI makró kifejtésében áll. Tekintsük a következő sort:

```
        MOV     EX_P_PAR.[ BX + SEGM ], ES
```

Itt egy indexelt címzést alkalmaztunk, eléggé ravasz formában. Két indexet is használtunk. Az EX_BLK struktúra EX_P_PAR tagja egy duplaszó. Ennek egyik vagy másik felét még egy pótlólagos offsettel lehet elérni, amely a LONG_PNT struktúra egyik vagy másik tagja. A két tagot persze egyszerűen összeadhatnánk a címzés során:

```
        MOV     EX_P_PAR+SEGM[ BX ], ES
```

de különválasztva is írhatjuk. A MASM egyik kellemes tulajdonsága, hogy ilyen szabadon választhatunk a címzési módok leírásában az egyes lehetőségek közül. Amennyiben ezzel a szabadsággal körültekintően élünk, programjaink olvashatóbbak lesznek, jobban le tudjuk írni az egyes algoritmusok belső logikáját. A fenti példa egyébként azt is mutatja, hogyan lehet struktúra belsejében másik struktúrát használni.

X.5.3. Beépített parancs végrehajtása programból

Az MS-DOS egyik felhasználói szempontból igen kellemes (és kardinális fontosságú) tulajdonsága az, hogy vannak beépített parancsai (DIR, DEL, COPY, CHDIR stb.), melyekhez nem kell a programok kódját tartalmazó lemez. Ez a sajátosság azonban kissé kellemetlen a programozónak, mert ő nem tud könnyen elindítani egy belső parancsot. Ezen a hiányosságon segít az a lehetőség, hogy a COMMAND.COM file-t nem csak hosszú időre hívhatjuk be a tárba. Felkérhetjük őt "egy táncra" is: ha átadunk egy paracssort, melyben szerepel a "/C" (Command) paraméter, akkor a COMMAND.COM tudni fogja, hogy mandátuma csak a parancssorban megszabott tevékenység elvégzésére szól; utána "a mór megtette kötelességét..." jelszóval távoznia kell, visszaadva a vezérlést az őt hívó programnak. Ezt láthatjuk az alábbiakban.

```
TITLE   Belső parancs végrehajtása EXE programból
;
; Konstansok, makrók, rekordok
;
        .XLIST                  ;DOSCALL.INC-et nem listázzuk!
        INCLUDE         DOSCALL.INC
        .LIST                   ;
```

```
;---------------------------------;
; Adatterületek
;-----------------------------------------;

DATA    SEGMENT PARA    PUBLIC  'DATA'

FILESPEC        DB      'C:\COMMAND.COM', 0
COMLIN          DB      SIZ_COMLIN      ;Hosszt MASM betölti
                DB      '/c dir *.asm'  ;
        SIZ_COMLIN      EQU     $ - COMLIN
                DB      CHR_CR          ;CHR_CR nem számít bele
A_COMLIN        DD      COMLIN          ;Pointerek az elemekre
A_FCB           DD      0               ;
                                        ;
EXEC_BLK        EX_BLK  <>              ;Adatbl. pr.-híváshoz
                                        ;
DATA    ENDS                            ;
;-----------------------------------------;
; Környezet
;---------------------------------;

ENV     SEGMENT PARA    PUBLIC  'DATA'
        DB      0, 0                    ;Ez egy üres környezet
ENV     ENDS

;---------------------------------;
; Stack-terület
;-----------------------------------------;

STACK   SEGMENT PARA    STACK   'STACK'
        DW      100     DUP ( ? )       ;Hossza 100 szó
STACK   ENDS

;-----------------------------------------;
; Kódszegmens
;-----------------------------------------;

CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:DATA, SS:STACK, ES:NOTHING
PSP_SEG         DW      0               ;
SAVE_SP         DW      0               ;
SAVE_SS         DW      0               ;
```

```
START:                                          ;
                MOV     CS:PSP_SEG, DS  ;PSP címének mentése
                MOV     AX, DATA        ;
                MOV     DS, AX          ;DS előkészítés
                CALL    CUT_EXE         ;Memória felszabadítása
                JC      ERR_EXIT        ;Lehetetlen esemény
                PUSH    DS              ;
                POP     ES              ;DS átmásolása ES-be
        EX_BLK_INI      EXEC_BLK, ENV, A_COMLIN, A_FCB, A_FCB
                                ;Másként nem fér ki egy sorba
                MOV     CS:SAVE_SP, SP  ;Stack Segment és
                MOV     CS:SAVE_SS, SS  ; Pointer mentése
                                        ;
                MOV     BX, OFFSET EXEC_BLK
                MOV     DX, OFFSET FILESPEC
                MOV     AL, DOS_EXEC_PRC;
                DOSCALL DOS_EXEC        ;Processzhivás
                                        ;
                MOV     SS, CS:SAVE_SS  ;
                MOV     SP, CS:SAVE_SP  ;Stack reg. helyreáll.
                JC      ERR_EXIT        ;
                DOSCALL DOS_GTRETCD     ;Visszatérési kód lek.
                JMP     SUCC_EXIT       ;
                                        ;
                EXIT                    ;Kilépés makróval
                                        ;
;---------------------------------------;
; CUT_EXE és MEMORY definicióját lásd a "Memóriakezelés EXE
;  programból" példában
;
        END     START
```

X.5.4. Saját program elindítása programból

A fejezet utolsó példaprogrampárja a környezet egy igen ravasz (és talán nem is túl tisztességes) felhasználását mutatja be. Az első program létrehoz egy file-t, és elindítja a második programot. Az elindított programnak átad egy környezetet, melynek első byte-jára letette a létrehozott file sorszámát. A második program ezt a byte-ot elveszi, majd kiírja a környezetet

alkotó stringeket a standard output file-ra és az átpasszolt sorszámú lemezfile-ra is, majd visszatérési kóddal kilép. A hívó program kiírja a visszatérési kódot, és szintén kilép.

Ezt a trükköt egyébként két okból tarthatjuk tisztességtelennek. Az első (elméleti jellegű) az, hogy az átpasszolni kívánt file-sorszám semmiképpen nem valódi ASCII-karakter, hanem egy vezérlőkarakter; az ilyenek valójában nem szerepelhetnek ASCII-stringben. A második pedig az, hogy egy file-sorszám lehet 0 is, ez pedig könnyen vezethet a környezet teljes félreértelmezésére.

```
TITLE   Környezet és benne file-handle átadása
PAGE    60,132
        EXTRN   PRTERRMSG:NEAR, CUT_EXE:NEAR
;----------------------------------------;
; Konstansok
;----------------------------------------;
        .XLIST                  ;DOSCALL.INC-et nem listázzuk!
        INCLUDE         DOSCALL.INC
        .LIST
;----------------------------------------;
; Adatterület
;----------------------------------------;

DATA    SEGMENT PARA    PUBLIC  'DATA'

PATH    DB      'GETENV.EXE', 0         ;Alprogram-specifikáció
EXEC_BLK        EX_BLK  <>              ;Végrehajtási blokk
                                        ;
COMFILE DB      'COMMON.TXT', 0         ;Közös file neve
COM_HND DW      0                       ;Handle az ő számára
                                        ;Bejelentkezési üzenet
MSGDEF  MSG_LOGIN, 'Hahó! Itt a parent program', , NODOLLAR
                                        ;Válaszkód kiírása
MSGDEF  MSG_RET,    'Válaszkód: ',      NOCR, NODOLLAR
MSG_RET_TRMCD   DB 0                    ;Itt a válaszkód
MSGDEF  MSG_RET2,  ', Kilépési kód: ', NOCR, NODOLLAR
MSG_RET_STATUS  DB 0                    ;Itt a kilépés módja
MSGDEF  MSG_CRLF    ' ', , NODOLLAR
        MSG_RET_SIZ     EQU     $ - MSG_RET

DATA    ENDS                            ;
```

```
;-----------------------------------------;
;Külön környezetszegmens
;-----------------------------------------;

ENV     SEGMENT PARA    PUBLIC  'DATA'
HND     DB      0               ;Átpasszolandó file-handle
MSGDEF  ENV_1, 'Ez az első sor',    , NODOLLAR
        DB      0
MSGDEF  ENV_2, 'Ez a második sor',  , NODOLLAR
        DB      0
MSGDEF  ENV_3, 'Ez a harmadik sor', , NODOLLAR
        DB      0, 0
ENV     ENDS
;-----------------------------------------;
; Stack-terület
;-----------------------------------------;
STACK   SEGMENT PARA    STACK   'STACK'
                DW      100     DUP     (?)     ;Hossza 100 szó
STACK   ENDS

;-----------------------------------------;
; Kódszegmens
;-----------------------------------------;

CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE,DS:DATA,SS:STACK,ES:NOTHING

PSP_SEG         DW      0               ;
                                        ;
START:                                  ;
                MOV     CS:PSP_SEG, DS  ;PSP elmentése
                                        ;
                MOV     AX, DATA        ;
                MOV     DS, AX          ;DS előkészitése
                                        ;
                CALL    CUT_EXE         ;Memória levágása
                LCJ     C,ERR_EXIT      ;(Lehetetlen esemény)
                MOV     AX, SEG ENV     ;
                MOV     ES, AX          ;
        ASSUME  ES:ENV                  ;
                                        ;
                OUTWRIT MSG_LOGIN       ;Belépési üzenet ki
```

```
                MOV     CX, ATR_ARCHV   ;
                MOV     DX, OFFSET COMFILE
                DOSCALL DOS_CREAT       ;File létrehozása
                JC      ERR_EXIT        ;
                MOV     COM_HND, AX     ;File-handle mentése és
                MOV     ES:HND, AL      ; !!Horrorrr!!  átadása
                                        ;
                MOV     CS:SAVE_SS, SS  ;SS mentése
                MOV     CS:SAVE_SP, SP  ;SP mentése
                MOV     DX, OFFSET PATH ;---;Program neve
                MOV     BX, OFFSET EXEC_BLK ;Paraméter-tábla
                MOV     EX_S_ENV.[ BX ], SEG ENV ;ENV elők.
                PUSH    DS              ;--------;
                POP     ES              ;ES:BX címez!
                DOSCALL DOS_EXEC, DOS_EXEC_PRC
                MOV     SS, CS:SAVE_SS  ;
                MOV     SP, CS:SAVE_SP  ;SS és SP helyeráll.
                                        ;
                JC      ERR_EXIT        ;
                DOSCALL DOS_GTRETCD     ;Válaszkód lekérd.
                ADD     AH, '0'         ; és kiírása
                MOV     MSG_RET_STATUS, AH
                ADD     AL, '0'         ;
                MOV     MSG_RET_TRMCD,  AL
                OUTWRIT MSG_LOGIN       ;
                OUTWRIT MSG_RET, MSG_RET_SIZ
                JMP     SUCC_EXIT       ;
                EXIT                    ;
                                        ;
SAVE_SS         DW      0               ;
SAVE_SP         DW      0               ;
                                        ;
CODE    ENDS                            ;
                                        ;
        END     START
```

Lássuk most a második programot, amely ügyesen "ellopja" az első által neki átadott file-sorszámot, sőt orvul bele is ír az így eltulajdonított file-ba. A memória átrendezése itt elmarad, mivel ez a program semmi mást nem akar csinálni, mint kiolvasni a közös file-sorszámot a környezetből, majd beleírni a file-ba. Erre a célra legcélszerűbb egy indirekt címzést alkalmazni.

```
TITLE   Környezet és benne file-sorszám átvétele
PAGE    60,132

        EXTRN   PRTERRMSG:NEAR, GTSTRSIZ:NEAR
;
; Konstansok és makrók
;
        .XLIST                  ;DOSCALL.INC-et nem listázzuk!
        INCLUDE         DOSCALL.INC
        .LIST
;---------------------------------------;
; Adatterület
;---------------------------------------;

DATA    SEGMENT PARA    PUBLIC  'DATA'

MSGDEF  MSG_LOGIN, 'Itt a handle-lopó child!', , NODOLLAR

MSG_HANDLE      DB      'Az átvett file-sorszám: '
HND_GOT         DB      0, CHR_CR, CHR_LF
        MSG_HANDLE_SIZ  EQU     $ - MSG_HANDLE

DATA    ENDS

;---------------------------------------;
; Stack-terület
;---------------------------------------;

STACK   SEGMENT PARA    STACK   'STACK'
                DW      100     DUP     (?)     ;Hossza 100 szó
STACK   ENDS

;---------------------------------------;
; Kódszegmens
;---------------------------------------;

CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:DATA, SS:STACK, ES:NOTHING
START:                                  ;
                MOV     CS:SAVE_PSP, DS ;
                MOV     AX, DATA        ;
                MOV     DS, AX          ;
```

```
                OUTWRIT MSG_LOG'          ;
                MOV     BX, PSP_ENVSEG    ;Környezet címe
                MOV     AX, WORD PTR ES:[BX]
                MOV     ES, AX            ;Környezet címe ES-be
                XOR     SI, SI            ;SI-vel címzünk ENV-ben
                MOV     AL, BYTE PTR ES:[SI]
                XOR     AH, AH            ;Közös handle kiolv.
                MOV     CS:COM_HND, AX    ; és mentése
                ADD     AL, '0'           ;Kódkonverzió
                MOV     HND_GOT, AL       ;Kiírjuk a handle-t
                OUTWRIT MSG_HANDLE, MSG_HANDLE_SIZ
                WRIT_H  OFFSET! MSG_LOG,SIZ_MSG_LOG,CS:COM_HND
                INC     SI                ;Átlépjük a handle-t
                MOV     DI, SI            ;
                PUSH    ES                ;
                POP     DS                ;ES -> DS
        ASSUME  DS:NOTHING                ;DS a környezetre címez
                                          ;
LOOP:                                     ;
                CALL    GTSTRSIZ          ;String hosszának lek.
                JE      SUCC_EXIT         ;Kilépés, ha 0
                WRIT_H  SI, CX, HND_STD_OUT
                JC      ERR_EXIT          ;
                WRIT_H  SI, CX, CS:COM_HND
                JC      ERR_EXIT          ;
                MOV     SI, DI            ;Következő SI-be
                JMP     LOOP              ;
                                          ;
                EXIT                      ;
                                          ;
SAVE_PSP        DW      0                 ;
COM_HND         DW      0                 ;
                                          ;
CODE    ENDS                              ;
                                          ;
;-----------------------------------------;
;
        END     START
```

Ide már csak egy apróság kívánkozik: a GTSTRSIZ rutin leírása és kódja. Ez, mint neve is mutatja, az ES:DI-ben adott című, 0-val lezárt string hosszát adja vissza karakterekben.

```
;-----------------------------------------;
; GTSTRSIZ      - String hosszának lekérdezése
;       Input:
;               ES:DI   - a string kezdete
;       Output:
;               CX      - a string hossza (záró 0 nem számit)
;               Carry   - 1, ha a string 64Kb hosszú
;               Zero    - 1, ha a string 0 hosszú
;               DI      - a string mögé mutat
;-----------------------------------------;
GTSTRSIZ        PROC    NEAR            ;
                                        ;
                PUSH    AX              ;
                XOR     CX, CX          ;Számlálást elők.
                XOR     AL, AL          ;Ezt keressük
GTSTRSIZ_LOOP:                          ;
                SCASB                   ;Keresés
                LOOPNE  GTSTRSIZ_LOOP   ; míg el nem érjük
                JNE     GTSTRSIZ_ILLG   ;CX fogyott el, hiba
                NOT     CX              ;NEG venné a 0-t is!
                TEST    CX, CX          ;
GTSTRSIZ_QUIT:                          ;
                POP     AX              ;
                RET                     ;
GTSTRSIZ_ILLG:                          ;Ez eléggé kilóg, nem
                STC                     ;fordulhat elő
                JMP     GTSTRSIZ_QUIT   ;
                                        ;
GTSTRSIZ        ENDP                    ;
```

Eddig mindig ügyeltünk arra is, hogy a rutin utolsó utasítása a RET legyen. Ez talán összeszedettebbé, olvashatóbbá teszi a programot. Itt azért mondtunk le erről, mert az az eset, hogy CX 0-ról indulva elfogyjon, és közben ne találjunk egyetlen 0 byte-ot sem, gyakorlatilag lehetetlen még akkor is, ha nem jól előkészített stringre hívjuk meg a rutint. Ez nem egyebet jelentene, mint azt, hogy egy egész 64 kilobyte hosszú szegmensben nincs egyetlen egy nulla byte sem. Azzal, hogy a szinte lehetetlen esetet kiraktuk a rutin mögé, megspóroltunk egy mindig végrehajtandó ugrást, ez pedig valamivel csökkenti a rutin futásidejét. Ilyen fontos és gyakran használt rutin esetén ez is egy nagyon fontos szempont!

XI. Interrupt-vektorok kezelése

Mivel tisztában vagyunk azzal, hol helyezkednek el az interrupt-vektorok a tárban, semmi akadálya sincs annak, hogy programból egyszerűen megcímezzük őket, és kiolvassuk, illetve módosítsuk tartalmukat. Ez a beavatkozás azonban igen komoly, hiszen egyenesen a rendszer "lelkébe", a legfontosabb táblázatokba nyúlunk bele. Tehát ilyen műveleteket csak rendkívül gondos előkészítés után végezzünk, és ha lehet, kerüljük el az interrupt-vektorok direkt írását; használjuk ilyen esetben az MS-DOS erre szolgáló funkcióit!

Kultúrember persze elsődleges kötelességének érzi a módosítani kívánt interrupt-vektor tartalmának mentését, és a program kilépése előtt helyreállítja a módosított vektort. Ezzel eleget tesz annak a fontos elvnek, hogy a program a tárban nem hagyhat maga után mást, mint amit ott talált.

XI.1. Funkciók az interrupt-vektorok kezelésére

25 Set Interrupt Vector, DOS_SETITVECT
Interrupt-vektor módosítása - az AL-ben adott sorszámú interrupt-vektort a DS:DX tartalmával írja felül. Nem győzzük hangsúlyozni, hogy a vektor eredeti tartalmát a 35 hívás adja meg, s azt tanácsos menteni, majd a program kilépése előtt helyreállítani.

35 Get Interrupt Vector, DOS_GETITVECT
Interrupt-vektor kiolvasása - AL-ben adjuk meg a vektor számát. ES:BX-ben kapjuk vissza a vektor szegmens- és offsetrészét.

XI.2. Példaprogram: a interrupt-vektorok listázása

A fejezet egyetlen példaprogramja lekérdezi és listázza az összes interrupt-vektort. Felhasználási területe egyetlenegy van: elindítunk néhány memóriarezidens programot (pl. SIDEKICK, BLANK, DOSEDIT), és kaján örömmel tanulmányozzuk, hogyan ragadják el egymás elől az interruptokat.

```
TITLE   Interrupt-vektorok listázása
PAGE    60,132
;       EXTRN   PRTERRMSG       ;Nincs hibalehetőség
        .XLIST                  ;DOSCALL.INC-et nem listázzuk!
        INCLUDE         DOSCALL.INC
        .LIST

;
; Adatterület
;

DATA    SEGMENT PARA    PUBLIC  'DATA'

HXTAB   DB      '0123456789ABCDEF'      ;Konverziós tábla

IT_VECTOR_SEG   DW      0               ;Munkaváltozók
IT_VECTOR_OFS   DW      0
IT_VECTOR_NUM   DW      0

BUFFER  DB      '    . IT vektor = '    ;Vektorleíró üzenet
BUF_SEG DB      '    :'
BUF_OFS DB      '    ', 13,10, '$'

DATA    ENDS

;-----------------------------------------;
; Stack-terület
;-----------------------------------------;

STACK   SEGMENT PARA    STACK   'STACK'
        DW      100     DUP     (?)     ;Hossza 100 szó
STACK   ENDS

;-----------------------------------------;
; Kódszegmens
;-----------------------------------------;

CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:DATA, SS:STACK, ES:NOTHING
START:                                  ;
                MOV     AX, DATA        ;
                MOV     DS, AX          ;DS előkészítése
```

```
                XOR     AL, AL          ;Első vektor száma 0
                MOV     BYTE PTR IT_VECTOR_NUM, AL
LOOP:                                   ;
                MOV     AL, BYTE PTR IT_VECTOR_NUM
                DOSCALL DOS_GETITVECT   ;
                MOV     IT_VECTOR_SEG, ES       ;A szegmens és
                MOV     IT_VECTOR_OFS, BX       ; offset elm.
                                        ;
                MOV     DX, IT_VECTOR_NUM
                MOV     DI, OFFSET BUFFER
                MOV     BX, 4           ;IT vektor számát
                CALL    CONVBA          ; konvertáljuk
                                        ;
                MOV     DX, IT_VECTOR_SEG
                MOV     DI, OFFSET BUF_SEG
                MOV     BX, 4           ;IT vektor szegmensét
                CALL    CONVBA          ; konvertáljuk
                                        ;
                MOV     DX, IT_VECTOR_OFS
                MOV     DI, OFFSET BUF_OFS
                MOV     BX, 4           ;IT vektor offsetjét
                CALL    CONVBA          ; konvertáljuk
                                        ;
                PRTSTR  BUFFER          ;Kiírjuk a szöveget
                                        ;Következő vektor
                INC     BYTE PTR IT_VECTOR_NUM
                JNE     LOOP            ;Folytatjuk, amíg van
                JMP     SUCC_EXIT       ;
                EXIT    NOMESS          ;Kilépés üzenet nélkül
                                        ;
;---------------------------------------;
; CONVBA        - Bináris érték hexadecimális ASCII konverziója
;       Input:
;               DX      - szám (16 bit előjel nélkül)
;               BX      - számjegyek maximális száma
;               DI      - buffer címe
;       Output:
;               konvertált string a bufferben
; Konverziós konstansok
;---------------------------------;
NUM_MASK        EQU     0FH     ;Egy számjegy elkülönítéséhez
CNT_SHIFT       EQU      4H     ;Egy számjegy kipörgetéséhez
```

```
CONVBA          PROC    NEAR                ;
                PUSH    AX                  ;
                PUSH    CX                  ;
                CLD                         ;Előre haladunk
                MOV     AL,                 ;Szóközökkel elő-
                MOV     CX, BX              ; készítjük a buffert,
        REP     STOSB                       ; DI mögé mutat!
                MOV     CL, CNT_SHIFT       ;Shiftek száma
CONV_LOOP:                                  ;
                MOV     AX, DX              ;Szám AX-be
                AND     AL, NUM_MASK        ;Elkülönítjük a köv.-t
                CALL    CONV                ;Konvertáljuk
                DEC     DI                  ;
                MOV     [ DI ], AL          ;Kiírjuk a karaktert
                SHR     DX, CL              ;A számot léptetjük
                JZ      CONV_QUIT           ;Ha 0, befejezzük
                DEC     BX                  ;Van még hely?
                JNZ     CONV_LOOP           ;Igen, folytatjuk
CONV_QUIT:                                  ;
                POP     CX                  ;
                POP     AX                  ;
                RET                         ;
CONVBA          ENDP                        ;
;-------------------------------------------;
;CONV           - egy 4 bites szám konverziója karakterré
;       Input:  AL      - konvertálandó érték
;                         ( 0 <= AL <= 15 )
;
;       Output: AL      - hozzá tartozó ASCII-kód
;-------------------------------------------;
CONV    PROC    NEAR                        ;
                PUSH    BX                  ;Regisztermentés
                MOV     BX, OFFSET HXTAB;Konverzió előkészítése
                XLAT                        ;Kódkonverzió
                POP     BX                  ;Regiszter helyreáll.
                RET                         ;
CONV    ENDP                                ;
                                            ;
CODE    ENDS                                ;
        END     START                       ;
```

Megjegyezzük, hogy az interrupt-vektorok felulírása nem kockázatmentes lépés, ilyen esetben kellő óvatossággal járjunk el!

XII. Kiegészitő MS-DOS funkciók

XII.1. Az MS-DOS egyéb lehetőségei

Ezek a funkciók (az 59, 5E, 5F és 62 funkciók kivételével, amelyek 3.00 és későbbi verziók használata során igen lényegesek) nem túlzottan fontos, ám hasznos segítséget adnak a programozónak. Azt hiszem, legtöbbjét ritkán használjuk majd, de egy operációs rendszer "szépségét" éppen az adja, hogy a legapróbb kis funkciónak is van valami jelentősége; aki nem ismeri valamennyit, annak a mozgástere beszűkül. Példának okáért viszonylag ritkán kerül sor arra, hogy programból beállítsuk a gép által ismert dátumot, bár nagyon hasznos lehet. Talán érdemes írni egy olyan programot, amely (amennyire ez software eszközökkel lehetséges) karbantartja a napi dátumot. A dátum precizitása nagyon fontos kérdés a mentések szempontjából. Egy komolyabb munka során az ember több ízben is archiválja a munkát alkotó file-okat, hogy egy-egy kockázatosabb lépcsőfok előtt biztosítsa a már elért eredményeket, az előző állapotra való visszatérés lehetőségét. Nos, ha nem tartjuk karban a dátumot, akkor a gép számára mindig 1980. január 1. lesz (mivel ez áll be alapértelmezésként); akkor pedig a számos mentés közül a jóisten sem fogja felismerni az 1987. április 26.-it. Az a lehetőség, hogy egy tizenöt-húszezer soros program szövegét olvasgatva ráismerjünk arra, hogy ez az öt lépcsőfok közül melyik, a gyakorlatban nem lehetőség, erre sohasem leszünk képesek, vagy csak aránytalanul nagy fáradsággal. Ha viszont precízen karbantartjuk a dátumot, akkor a file-ok keletkezésének ideje meg fogja mutatni, melyik a kívánt verzió.

Lássuk akkor a DOS-funkciók befejezéseképpen a néhány hátralévő lehetőséget!

XII.2. A kisegítő funkciók ismertetése

MS-DOS 3.00 és későbbi verziók

59 Get Extended Error, DOS_GTEXTERR
Ez a funkció egy kiterjesztett hibakód lekérdezésére szolgál. Akkor kell használnunk, ha kritikus hiba kezelő rutint írunk. Ekkor legcélszerűbb, ha ezzel a funk-

cióval lekérdezzük a hibakódot, mert így nemcsak teljes képet kapunk a hibáról, hanem információt a hiba előfordulásának helyéről, sőt még tanácsot is: mit lehet csinálni az adott esetben (magyarul: ki kell-e lépni, fel kell-e adni minden reményt, vagy programból lehetséges a hiba korrekcója).
A BX regiszterben bemenetkor el kell helyezni az MS-DOS verziószámát (milyen verzió konvenciói szerint akarjuk lekérdezni a hibát). A DOS 3.00 és 3.1 verziója számára BX-ben 0-t kell megadni.
A kiterjesztett hibakódok lekérdezésére két esetben van valóban szükség. Az egyik egy kritikus hibát kezelő rutin, amelyben célszerű kikérni a rendszer tanácsait. A másik jellegzetes felhasználási terület az FCB file-kezelési funkciók területe. Itt, mint emlékezhetünk rá, nem kapunk vissza hibakódot, csak AL-ben egy 0-t sikeres, és FFH-t sikertelen művelet után. Ha a hiba okára kíváncsiak vagyunk, nem tehetünk mást, mint hogy lekérdezzük a kiterjesztett hibakódot. Ebből elég pontos információt szerezhetünk a hiba valódi okáról. Szokásos felhasználási lehetőség még, hogy bármilyen hibás funkciólefutás után (a visszakapott hibakódot figyelmen kívül hagyva) lekérdezzük a kiterjesztett hibakódot. Azonban, mint látni fogjuk, a hibalehetőségek köre elég széles ahhoz, hogy akárcsak a legfontosabb eseteket is kényelmesen és könnyen lekezelhessük programból.

A visszatéréskor kapott információk:

- AX - kiterjesztett hibakód (amely megegyezhet az MS-DOS-tól szokásos úton visszakapott hibakóddal, de kritikus hiba esetén a kiterjesztett hibakódok valamelyikét kapjuk meg).
- BH - a hiba osztálya, amely a hiba természetére utal
- BL - javaslat a folytatásra - mit tehet a program az adott hiba után
- CH - a hiba helye - memória vagy periféria

A kiterjesztett hibakódokat lásd a "Felhasználói hibakódok", a "Kiterjesztett hibakódok" és a "Hibakódok a 3.00 verzióban" c. fejezetben.

Hibaosztályok (59 funkció)
(válaszérték BH-ban)

- 01 - Out of Resource - az "erőforrások" elfogytak, nincs terület, átviteli csatorna stb.
- 02 - Temporary Situation - olyan feltétel, amely valamennyi idő múlva bizonyára megszűnik
- 03 - Authorization - hozzáférési joggal kapcsolatos probléma
- 04 - Internal - az MS-DOS belső programhibája
- 05 - Hardware Failure - valamilyen komoly hardware hiba okozza a sikertelen műveletet
- 06 - System Failure - nem a program által okozott rendszerhiba (pl. a konfigurációs file hibás)
- 07 - Application Program Error - a felhasználói program által okozott komoly hiba (pl. teljesen értelmetlen kérés stb.)
- 08 - Not Found - a file-t vagy bejegyzést nem lehet megtalálni
- 09 - Bad Format - a file vagy bejegyzés formátuma illegális
- 0A - Locked - a file vagy egy része le van zárva
- 0B - Media - az adattárolási eszköz hibája (CRC hiba a lemezen stb.)
- 0C - Already Exists - létező bejegyzéssel megegyezőt akartunk létrehozni (csak IBM dok.-ban!)
- 0D - Unknown - nem sorolható a fenti osztályokba

Javasolt tevékenységek (59 funkció)
(válaszérték BL-ban)

- 01 - Retry - néhány kísérlet után a felhasználót kell megkérdezni, ki akar-e lépni vagy sem
- 02 - Delay Retry - bizonyos idő eltelte után újra meg kell kísérelni a műveletet
- 03 - User - a felhasználót meg kell kérni, ismételje a beadott információkat
- 04 - Abort (Terminate with Cleanup) - rendet teszünk magunk után, és kilépünk
- 05 - Immediate Exit - azonnal kilépni (és tanácsos új rendszert hívni)
- 06 - Ignore - a hibát figyelmen kívül hagyhatjuk
- 07 - Retry After User Intervention - a felhasználó beavatkozása után újra meg kell kísérelni a műveletet

Hibaelőfordulási helyek (59 funkció)
(válaszérték CH-ban)

01 - Unknown - nem megállapítható vagy nem jellemző a hiba helye
02 - Block Device - a hiba lemezkezelés közben következett be
03 - Reserved - nem használt, későbbi felhasználásra fenntartva (hálózati hiba **MS-DOS 3.1 és későbbi verziókban**)
04 - Serial Device - a hiba soros kezelésű perifériális eszközön következett be
05 - Memory - memóriahiba

MS-DOS 3.00 és későbbi verziók

62 Get PSP, DOS_GETPSP

Visszaadja a pillanatnyilag aktív program PSP-jének szegmenscímét a BX regiszterben.
Hasznos segédeszköz lehet, használata azonban assemblyprogramban nem javasolt. Magasszintű nyelven írt programban (amennyiben hivatkozni akarunk a PSP-re) nemigen juthatunk másképpen a PSP címéhez.

MS-DOS 3.1 és későbbi verziók

5E Get Machine Name/Printer Setup, DOS_MNMPRT

Hálózati funkció. Az egyik alfunkció a gépünk hálózatban kapott nevét adja vissza, a másik pedig a hálózati printer felprogramozására szolgál.
A hálózatkezelő programnak aktívnak kell lennie!

Alfunkciók:

00 - a gép hálózati nevének lekérdezése
DS:DX mutasson egy 16 byte hosszú bufferre, amelyben a funkció lefutása után megkapjuk gépünk hálózati nevét.
02 - olyan szekvenciát adhatunk meg, amelyet minden egyes, a hálózati printerre kiírt file előtt el kívánunk küldeni.

Ezzel a funkcióval, mivel hálózati felhasználással kapcsolatos, e könyv keretein belül nem tudunk érdemben foglalkozni.

MS-DOS 3.1 és későbbi verziók

5F Get/Make/Cancel Assign List Entry, DOS_ASSIGN

A hálózati hozzárendelési lista adott elemének lekérdezése, új elem létrehozása, és elem törlése a hozzárendelési listából.
A "Microsoft Networks" programnak aktívnak kell lennie!
Alfunkciók:

02 - a hozzárendelési lista elemének lekérdezése
03 - új elem létrehozása a hozzárendelési listában
04 - a hozzárendelési lista egy elemének törlése

Ezzel a funkcióval, mivel hálózati felhasználással kapcsolatos, e könyv keretein belül nem tudunk érdemben foglalkozni.

2E Set Verify State, DOS_STVFYST
A "verify" kapcsoló beállítása - a verify kapcsoló szabályozza, hogy végezzen-e a rendszer lemezíráskor ellenőrzést vagy sem.
AL-ben 0-t adunk meg, ha ki akarjuk kapcsolni az ellenőrzést; 1-et akkor, ha be.

54 Get Verify State, DOS_GTVFYST
Verify státusz lekérdezése - visszatéréskor AL-ben 0 lesz, ha a kapcsoló állása "nem" (azaz a rendszer nem végzi el az ellenőrzést); 1, ha "igen".

33 Get/Set Control-Break Check, DOS_GSBREAK
A CTRL-BREAK kapcsoló vezérlése - a CTRL-BREAK kapcsoló azt szabályozza, hogy a rendszer minden funkcióhívás esetén ellenőrizze-e a CTRL-BREAK billentyűkombinácó leütését (kivéve 06 és 07), vagy csak a standard fileok kezelése során (01-0C funkciók, kivéve 06 és 07).
AL=0 - lekérdezi a CTRL-BREAK ellenőrzést. Válaszként a DL regiszterben (!) 1-et kapunk, ha a rendszer minden funkcióhívásкor végez CTRL-BREAK ellenőrzést; és 0-t, ha csak a standard file-kezelő funkciók hívása során.
AL=1 - esetén a kapcsoló beállítását végezhetjük el. DL-ben 1-et megadva bekapcsoljuk, 0-t megadva pedig kikapcsoljuk a CTRL-BREAK kapcsolót.

Sikertelen hívás esetén AL-ben FF-et kapunk válaszként. Ez csak akkor következhet be, ha bemenetként AL-ben nem nullát vagy egyet adtunk meg.

2A Get Date, DOS_GTDATE
Dátum lekérdezése - CX:DX tartalmazza a dátumot (CX az évet, DH a hónapot, DL a napot tartalmazza binárisan). Az AL regiszterben a nap kódját kapjuk meg:
0 - vasárnap
1 - hétfő stb.

2B Set Date, DOS_STDATE
Dátum felülírása - CX:DX-ben kell megadni a dátumot a fenti formában. CX-be kell töltenünk az évet (amelynek 1980 és 2099 közé kell esnie), DH-ba a hónapot (értelemszerűen 1 és 12 között), és DL-be a napot (1 és 31 között).
A dátum sikeres módosítása után AL-ben 0-t, hibás dátum megadása után pedig FFH-t kapunk vissza. Ekkor a dátum persze nem változik.
A rendszer olyan intelligens, hogy ellenőrzi: van-e az adott hónapban a megadott nap: február 30. dátumként nem állítható be! A rendszer a szökőéveket is ismeri. Például 1987-ben nincs február 29., és ilyen dátumbeállítást nem is fogad el, de 1988-ban már igen.

2C Get Time, DOS_GTTIME
Idő lekérdezése - CX:DX-ben kapjuk vissza az időt. Az AL regiszterben a nap kódját találjuk:
0 - vasárnap
1 - hétfő
2 - kedd
stb.;
a CH regiszter tartalmazza az órákat, CL a perceket, DH a másodperceket, és DL a századmásodperceket.

2D Set Time, DOS_STTIME
Idő felülírása - CX:DX-ben kell megadni a fenti formában a kívánt új időt. CH-ban adjuk meg az órát 0 és 23 között, CL-ben a perceket 0 és 59 között, DH-ban a másodperceket (0-59), végül DL-ben a századmásodperceket (0-99).

Ha bármelyik paraméter illegális, akkor az AL regiszterben FFH-t kapunk vissza, egyébként 0-t.

Feladat: írjunk egy olyan programot, amely egy nulla hosszúságú file segítségével karbantartja a gép által ismert dátumot és időt. Az alapötlet a következő: létrehozunk valahol, persze jól elrejtve, egy file-t, és azon nyomban le is zárjuk. Ekkor ez csak egy directory-bejegyzést foglal le, adatterülete nincs is. Keletkezésének ideje azonban megmutatja, hogy mi volt az utolsó dátum és idő, amelyet helyesnek fogadtunk el. Az újabb futás során megnyitjuk ezt a file-t, és ha utolsó írásának ideje későbbi, mint a pillanatnyi rendszeridő, akkor ez utóbbi nyilván inkorrekt. Tehát ekkor beállítjuk a rendszeridőt valamilyen későbbi dátumra. Végül újra létrehozzuk a file-t, mely már a legújabb dátumot fogja őrizni.
Ez persze nem a korrekt paraméterek beállítását eredményezi, de garantálja a dátum és idő folytonos növekedését - és ez a legfontosabb.

30 Get DOS Version Number, DOS_GTDOSVR
A DOS verziószámának lekérdezése - visszatéréskor AL tartalmazza a verziószámot, AH pedig a változatot.

Megjegyzés: ez a funkció olyan programok számára lehet fontos, amelyek kihasználnak olyan lehetőségeket, amelyek csak bizonyos MS-DOS verziókban léteznek. Ha a verziószám nem megfelelő, akkor a programunk súlyosabb következmények nélkül "veheti a kalapját" és kiléphet, avagy (ha kellően intelligens) lemondhat e funkciók használatáról.

38 Get/Set Country Data, DOS_GSCNTRY
Országfüggő információk lekérdezése és beállítása. Az MS-DOS számos ország szabványai szerint képes kezelni a dátumot, az időt stb. E szabványra vonatkozó információk lekérdezésére vagy egy adott szabvány kiválasztására szolgál ez a funkció.
Megjegyezzük, hogy egy ország kódja általában az adott ország nemzetközi távhívási száma.

a) Országfüggő információ lekérdezése

AL=0 - a pillanatnyilag beállított országra jellemző adatok lekérdezése;

AL<255 - az AL-ben megadott kódú országra jellemző adatok lekérdezése

ha AL=255, akkor

- a BX-ben megadott (255-nél nagyobb) kódú országra jellemző adatok lekérdezése

DS:DX - 32 byte-os buffer címe az adatok befogadására

A visszatérés után a memóriablokkban a következőket találjuk:

MS-DOS 2.xx-ig:

1 szó - dátum/idő formátum kódja

0 - amerikai szabvány óra:perc:másodperc hónap/nap/év;

1 - európai szabvány óra:perc:másodperc nap/hónap/év;

2 - japán szabvány óra:perc:másodperc nap:hónap:év;

4 byte - a valuta szimbóluma (pl. "$ <0>"); nullával lezárt karakterstring

2 byte - ezreseket elválasztó jel kódja (lehet " ", ".", "," stb.), 0-val lezárva

2 byte - tizedesjegyeket elválasztó jel kódja (lehet ".", "," stb.), 0-val lezárva

24 byte foglalt későbbi felhasználásra.

MS-DOS 3.00 és későbbi verziókban:

1 szó - dátum/idő formátum kódja

0 - amerikai szabvány óra:perc:másodperc hónap/nap/év;

1 - európai szabvány óra:perc:másodperc nap/hónap/év;

2 - japán szabvány óra:perc:másodperc nap:hónap:év;

5 byte - a valuta szimbóluma (pl. "$ <0>" stb.), nullával lezárt karakterstring

2 byte - ezreseket elválasztó jel kódja (lehet " ", ".", "," stb.), 0-val lezárva

2 byte - tizedesjegyeket elválasztó jel kódja (lehet ".", "," stb.), 0-val lezárva

2 byte - a dátum elemeit elválasztó jel kódja (lehet "/", "-" stb.), 0-val lezárva

2 byte - az idő elemeit elválasztó jel kódja (lehet ":" stb.), 0-val lezárva

1 byte - valutakiírás módja:
- 0. bit - 0: a jel megelőzi az értéket
- 1: a jel követi az értéket
- 1. bit - 0: az érték és a jel között nincs szóköz
- 1: az érték és a jel között egy szóköz van

1 byte - idő formátuma:
- 0. bit - 0: 12 órás formátum
- 1: 24 órás formátum

2 szó - kisbetűket nagybetűvé konvertáló rutin címe.
Ez a rutin a 128-nál magasabb kódú karakterek esetén végzi el a konverziót. Bemenetként AL-ben kell megadni a konvertálandó kódot. Ugyancsak AL-ben kapjuk vissza a konvertált karaktert. Ha a kód 128-nál kisebb, vagy nincs nagybetűs megfelelője, akkor AL nem változik.

2 byte - adatlistaelemeket elválasztó karakter kódja, 0-val lezárt string

5 szó - fenntartott terület

b) Országfüggő információ beállítása

DX - 0FFFFH (jelzi, hogy nincs buffer)
AL - kiválasztott kód (0-254) vagy ha
Ha AL=255, akkor
BX - kiterjesztett országkód

Hibakód Carry=1 esetén:

02 - Illegális országkód

XII.3. Nem használt funkciókódok

Az alábbiakban az MS-DOS belső használatára fenntartott funkciókódokat adjuk meg. Ezek közül egyesek a programozó számára is hasznosak lehetnének; folyóiratokban lehet itt-ott olvasni egyes ilyen interruptokról. Használatuk azonban még akkor sem ajánlott, ha azt hisszük, hogy tudjuk, mit csinálnak, mert megváltoztathatnak egyet-mást az operációs rendszer belső táblázataiban, és a következményekkel valószínűleg nem tudunk mit kezdeni. A fenntartott funkciókódok:

18, 1D, 1E, 1F, 20, 32, 34, 37, 50, 51, 52, 53, 55, 5D, 60, 61

XII.4. Példaprogramok a kisegítő funkciókhoz

XII.4.1. Ország lekérdezése és módosítása

Az alábbi program lekérdezi és elmenti a kurrens ország kódját, beállít néhány szabványt - német, angol, svéd (itt egyezik meg a dátum kiírása a nálunk megszokottal) - és elindítja a COMMAND.COM-ot egy DATE paranccsal, hogy megismerkedhessünk az adott szabványból legalább a dátum formátumával.

```
TITLE   Országfüggő információ kezelése

        EXTRN   GETPAR:NEAR, PRTERRMSG:NEAR

        EXTRN   CUT_EXE:NEAR

;
; Konstansok, makrók, rekordok
;

        .XLIST                      ;DOSCALL.INC-et nem listázzuk!
        INCLUDE         DOSCALL.INC
        .LIST

GERMANY         EQU     49          ;A kivánt
GREAT_BR        EQU     44          ; országok
SWEDEN          EQU     46          ; kódja
```

```
;---------------------------------;
; Adatterületek
;---------------------------------;

DATA    SEGMENT PARA    PUBLIC  'DATA'
                ;---------------;
                ;Paraméterek a DATE parancs lefuttatásához
                ;-----------------------;
FILESPEC        DB      'C:\COMMAND.COM', 0
COMLIN          DB      SIZ_COMLIN
                DB      '/C DATE'
        SIZ_COMLIN      EQU     $ - COMLIN
                DB      CHR_CR
EX_DATE         EX_BLK  <>
                ;-----------------------;
                ; Paraméterek az országfüggő információhoz
                ;-----------------------;
CURR_COUNTRY    DB      0               ;Kurrens ország kódja
CURR_C_BLOCK    DB      32 DUP  ( 0 )   ;Buffer
                ;-----------------------;
                ; Felhasznált üzenetek
                ;-----------------------;
MSGDEF  MSG_GERM, 'Német szabvány',  , NODOLLAR
MSGDEF  MSG_GRBR, 'Angol szabvány',  , NODOLLAR
MSGDEF  MSG_SWED, 'Svéd szabvány',   , NODOLLAR
MSGDEF  MSG_ORIG, 'Eredeti állapot', , NODOLLAR
DATA    ENDS
;---------------------------------;
; Stack-terület
;---------------------------------;
STACK   SEGMENT PARA    STACK   'STACK'
        DW      100     DUP ( ? )       ;Hossza 100 szó
STACK   ENDS
;---------------------------------;
; Kódszegmens
;---------------------------------;
CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:DATA, SS:STACK, ES:DATA

PSP_SEG         DW      0       ;
SAVE_SS         DW      0       ;
SAVE_SP         DW      0       ;-------;
```

```
START:                                  ;
                MOV     CS:PSP_SEG, DS  ;PSP címének mentése
                XOR     SI, SI          ;
                MOV     AX, PSP_ENVSEG[ SI ]
                MOV     AX, DATA        ;
                MOV     DS, AX          ;DS -> DATA szegmens
                CALL    CUT_EXE         ;Memória átrendezése
                PUSH    DS              ;
                POP     ES              ;Szokásos előkészítés
                                        ;
                MOV     DX, OFFSET CURR_C_BLOCK
                DOSCALL DOS_GSCNTRY, 0  ;Kurrens ország lekérd.
                MOV     CURR_COUNTRY, AL; és mentése
                                        ;
                OUTWRIT MSG_GERM        ;A német szabvány
                MOV     DX, -1          ; beállitása és
                DOSCALL DOS_GSCNTRY, GERMANY
                CALL    DATE_REQ        ; ellenőrzése
                                        ;
                OUTWRIT MSG_GRBR        ;Az angol szabvány
                MOV     DX, -1          ; beállitása és
                DOSCALL DOS_GSCNTRY, GREAT_BR
                CALL    DATE_REQ        ; ellenőrzése
                                        ;
                OUTWRIT MSG_SWED        ;A svéd szabvány
                MOV     DX, -1          ; beállitása és
                DOSCALL DOS_GSCNTRY, SWEDEN
                CALL    DATE_REQ        ; ellenőrzése
                                        ;
                OUTWRIT MSG_ORIG        ;Az eredeti állapot
                MOV     DX, -1          ; beállitása és
                DOSCALL DOS_GSCNTRY, CURR_COUNTRY
                CALL    DATE_REQ        ; lekérdezése
                JMP     SUCC_EXIT       ;
                EXIT                    ;
;---------------------------------------;
; DATE_REQ      - elinditja a DATE belső parancsot
;       (lásd a "Memóriakezelés és programvezérlés" fejezet
;       "Belső parancs elindítása programból" példáját)
;---------------------------------------;
CODE    ENDS                            ;
        END     START                   ;
```

XII.4.2. A dátum megnövelése

Az alábbi program lekérdezi a kurrens dátumot, és megnöveli egy nappal. Egyetlen apró ötletként kihasználjuk azt, hogy a rendszer az illegális dátumbeállitási kísérletre hibakódot ad vissza. Tehát egyszerűen lekérdezzük a dátumot, megnöveljük a napot, és megpróbáljuk ezt beállitani mint dátumot. Ha sikerül, minden rendben van. Ha nem, akkor a napot 1-re állítjuk, és megnöveljük a hónapot. Ha így sem sikerül, akkor a hónapot is egyre állítjuk, és megnöveljük az évet. Ha most sem sikerül, akkor az Úr kétezeregyszázadik esztendejében járunk. Ez esetben pedig a helyes dátumbeállítás legyen az ükunokáink gondja.

```
TITLE   Dátum megnövelése
        EXTRN   GETPAR:NEAR, PRTERRMSG:NEAR
        EXTRN   CUT_EXE:NEAR
;
; Konstansok, makrók, rekordok
;
        .XLIST                  ;DOSCALL.INC-et nem listázzuk!
        INCLUDE         DOSCALL.INC
        .LIST
;---------------------------------;
; Adatterületek
;-----------------------------------------;
DATA    SEGMENT PARA    PUBLIC  'DATA'
                ;-----------------------;
                ;Paraméterek a DATE parancs lefuttatásához
                ;Lásd az előző példát
                ; Kiirandó üzenetek
                ;-----------------------;
MSGDEF  MSG_CDATE, 'Mai dátum növelés előtt', , NODOLLAR
MSGDEF  MSG_IDATE, 'Megnövelt dátum', , NODOLLAR
MSGDEF  MSG_NODATE, 'Ez sajnos lehetetlen dátum', , NODOLLAR
DATA    ENDS

;-----------------------------------------;
; Stack-terület
;-----------------------------------------;
STACK   SEGMENT PARA    STACK   'STACK'
        DW      100     DUP ( ? )       ;Hossza 100 szó
STACK   ENDS                            ;
```

```
;-----------------------------------------;
; Kódszegmens
;-----------------------------------------;

CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:DATA, SS:STACK, ES:DATA

PSP_SEG         DW      0               ;
PRG_ENV         DW      0               ;
                                        ;
SAVE_SS         DW      0               ;
SAVE_SP         DW      0               ;
                                        ;
START:                                  ;
                MOV     CS:PSP_SEG, DS  ;
                XOR     SI, SI          ;
                MOV     AX, PSP_ENVSEG[ SI ]
                MOV     CS:PRG_ENV, AX  ;
                MOV     AX, DATA        ;
                MOV     DS, AX          ;
                                        ;
                CALL    CUT_EXE         ;
                PUSH    DS              ;
                POP     ES              ;Szokásos előkészítés
                                        ;
                OUTWRIT MSG_CDATE       ;
                CALL    DATE_REQ        ;
                DOSCALL DOS_GTDATE      ;Dátum lekérdezése
                INC     DL              ;Nap megnövelése
                DOSCALL DOS_STDATE      ;
                TEST    AL, AL          ;
                JNS     DATE_SUCC       ;
                MOV     DL, 1           ;
                INC     DH              ;Hónap megnövelése
                DOSCALL DOS_STDATE      ;
                TEST    AL, AL          ;
                JNS     DATE_SUCC       ;
                MOV     DH, 1           ;
                INC     CX              ;Az év megnövelése
                DOSCALL DOS_STDATE      ;
                TEST    AL, AL          ;
                JS      DATE_UNS        ;Nincs mit tennünk
```

```
DATE_SUCC:                                      ;
                OUTWRIT MSG_IDATE               ;
                CALL    DATE_REQ                ;
                JMP     SUCC_EXIT               ;
DATE_UNS:                                       ;
                OUTWRIT MSG_NODATE              ;
                JMP     SUCC_EXIT               ;
                                                ;
                EXIT                            ;
                                                ;
;-----------------------------------------------;
; DATE_REQ      - elindítja a DATE belső parancsot
;       (lásd a "Memóriakezelés és programvezérlés" fejezet
;       "Belső parancs elindítása programból" példáját)
;-----------------------------------------------;
CODE    ENDS
        END     START
```

Valószínű, hogy az eddigi hányattatások megedzették már annyira az Olvasót, hogy ezt a "hihetetlenül bonyolult" programot sem kell agyonmagyarázni.

XIII. Kilépés a programból

Az ember hajlamos lenne azt hinni, hogy mi sem egyszerűbb, mint kilépni egy programból. Azonban a rutinosabb programozók és különösen azok, akik már programoztak assemblerben, tudják, hogy ez korántsem egyszerű feladat. Először is a kilépés csak magasszintű nyelvekben látszik trivialitásnak. Az ember kiad egy "exit" vagy valamilyen hasonló utasítást, és már el is van intézve a dolog. Udvariasabb rendszerek és nyelvek még azt is biztosítják, hogy a programozó által megnyitott file-ok automatikusan lezáródjanak, és eltakarítják az általa okozott zűrök nagy részét. De aki assemblerben programoz, mint látni fogjuk, jónéhány kellemetlen mellékkörülménnyel kénytelen számolni.

00 Exit from the Program, DOS_COMEXIT
Kilépés a programból. Pontosan megegyezik a 20-as interrupttal. Abban is teljes az egyezés, hogy a kilépési funkció meghívása előtt gondoskodnunk kell arról, hogy a CS regiszter a PSP-t címezze meg - ez a funkció tehát leginkább COM típusú programokban használható, de ha lehet (miért ne lehetne?), ott is kerüljük.

31 Exit but Stay Resident, DOS_STAYRES
Kilépés "bentmaradva" - visszaadja a vezérlést a rendszernek úgy, hogy a program bent marad a memóriában. AL-be kell tenni a visszatérési kódot, melyet az indító program megkap, DX-be pedig a bentmaradó program számára fenntartandó memória hosszát paragrafusokban.
Hasonló lehetőséget nyújt a 27 interrupt, csakhogy annak használata során kissé másként kell megadni a fenntartandó memóriaterület hosszát. A rendszer nem a 27 interrupt, hanem e függvény használatát támogatja!

Megjegyzés: ez a funkció bizonyos multi-tasking, pontosabban multi-rezidens programozást tesz lehetővé. Tehát több program is bent lehet egy időben a memóriában, amelyek egymást meghívhatják (vö. a 2F interrupt használatával 3.1 és későbbi verziókban).
Ajánlható módszer a következő: egy vezérlőprogram rendre beolvassa a 4B funckióval a kívánt programokat, amelyeknek ekkor egyetlen tevékenysége,

hogy a számukra kijelölt felhasználói interrupt-vektorának tartalmát elmentik, és felülírják a saját tényleges belépési pontjukkal. Ezután a 31 funkcióval visszatérnek a főprogramhoz, amely megkezdi a tényleges munkát. Valahányszor egy alprogramra van szüksége, a hozzárendelt interrupt segítségével aktivizálja azt. A főprogram és az alprogramok kommunikációjára az itt megismert eljárások kényelmesek lehetnek: a meghívás előtt egy funkciókódot az AH, a paramétereket a többi regiszterbe helyezzük; a meghívott program szintén a regiszterekben válaszol.
Végül a főprogram a teljes kilépés előtt sorra meghív minden alprogramot azzal a funkciókóddal, hogy az alprogram állítsa helyre az interrupt-vektort, és lépjen ki. Ha az alprogramokat fordított sorrendben hívjuk meg, és azok normális kilépéssel térnek vissza, akkor helyreállítottuk a DOS belső táblázatait is.
Ezt a rendkívül fontos funkciót nem illusztráljuk példaprogrammal egész egyszerűen azért, mert erre igen nehéz rövid és egyszerű példát adni, amelynek még valami értelme is van; egy SIDEKICK-rész megírása kicsit meghaladná könyvünk kereteit. Ehelyett a 3. kötetre hivatkozunk, ahol példát olvashatunk egy klaviatúra-driverre. Ez a driver bentmarad a tárban; egy driver jellegzetesen rezidens program.

4C End Process, DOS_ENDPRC
Kilépés visszatérési kód küldésével, valamint minden, a 3D-vel megnyitott file lezárásával. A kívánt visszatérési kódot AL-be kell tenni.
A hívó processz a visszatérési kódot a 4D funkció segítségével kérdezheti le (lásd "Memóriakezelés és programvezérlés" fejezet).

A régebbi verziójú MS-DOS operációs rendszerekben a COM típusú programok szokásos kilépési eljárása az

```
INT     20H
```

utasítás kiadása. Ennél a kilépésnél azonban alapfeltétel, hogy a CS szegmensregiszter az utasítás kiadásakor a Program Segment

Prefixre mutasson. Ezért EXE típusú programokból csak az "IBM PC/XT felhasználóknak és programozóknak" sorozat első kötetében bemutatott nem túlzottan erkölcsös előkészítés után léphetünk így ki. Emlékezetbe idézzük az előkészítést:

```
START           PROC    FAR
                PUSH    DS          ;PSP címe a stack-re
                XOR     AX, AX      ;AX = 0
                PUSH    AX          ;0. offset elmentése;
                .                   ; hosszú visszatérési cím!
                .                   ;Főprogram
                .                   ;
                RET                 ;Visszatérés PSP 0. offsetjére
START           ENDP
```

A "PROC FAR" biztosítja, hogy a fenti RET utasítás távoli visszatérés lesz, vagyis valóban kiveszi a veremből a számára előkészített két szót, és átadja a vezérlést a PSP 0. offsetjére, ahol egy INT 20H utasítás van.

A gyakorlatban alkalmazott (és a rendszer által támogatott) eljárás minden programban a 4C [DOS EXIT] funkció használata, amely még visszatérési kód küldését is megengedi!

A programból való kilépés csaknem minden változatát illusztrálják eddigi példaprogramjaink. Semmi ok nincs arra, hogy más, még egyszerűbb példával hozakodjunk elő. Jellegzetes példák olvashatók a "Memóriakezelés és programvezérlés" és az "MS-DOS file-kezelés" fejezetben.

XIV. Az MS-DOS táblázatai

Ez a fejezet a DOS táblázatai a közül számunkra legérdekesebbek, a File Control Block (FCB), a directory-entry (directory-bejegyzés) és a Program Segment Prefix (PSP) szerkezetét tartalmazza. Itt olvasható továbbá a EXE típusú file-ok header-jének, valamint a relokációs táblának szerkezete is.

Első fontos témakörünk az MS-DOS lemezeinek logikai és fizikai felépítése. Teljes pompájában, sajnos, nem ismerkedhetünk meg ezzel a fontos területtel, a terjedelem csak a legfontosabb elemekkel és elvekkel való megismerkedést teszi lehetővé. Ennél valamivel többet olvashatunk a 3. kötet "Diskette-kezelés" fejezetében.

XIV.1. Az MS-DOS lemezek fizikai felépítése

Bevezetésképpen néhány szó a lemezek (nem MS-DOS specifikus) fizikai felépítéséről. Egy mágneslemez egy vagy több egy tengelyen forgó lemezből áll (az utóbbit kevésbé pongyolán lemezcsomagnak is szokták mondani). A legtöbb lemeznek legalább két oldala (angolul side) van. Minden oldal sávokra (track) oszlik. Egy sáv egy gyűrű a lemez felületén, az adatokat erre a gyűrűre írjuk. A sávok szektorokra (sector) oszlanak. A legtöbb esetben a szektorok hossza azonos, vagyis ugyanannyi adatbyte írható fel rájuk. Általában sávonként azonos számú szektort szoktak elhelyezni, függetlenül attól, hogy a sáv a lemezoldal külső vagy belső részén helyezkedik el, és így fizikailag hosszabb vagy rövidebb.

Ezek szerint egy, a lemezre felírt adatbyte helye a következőképpen adható meg: melyik oldal hányadik sávján, a sáv hányadik szektorában, a szektoron belül hol helyezkedik el. Mivel azonban a legtöbb lemezegység szektoronként írja és olvassa a lemez területét, az utolsó adatra nincs szükség. Egy szektor helye tulajdonképpen egy háromdimenziós vektor:

(oldal, sáv, szektor)

Az MS-DOS az oldalakat és a sávokat nullától, míg a szektorokat egytől számozza. Az alacsonyabb sorszámú sávok kijjebb, a magasabbak pedig beljebb helyezkednek el a lemezen.

Az IBM PC/XT/AT lemezei alapvetően kétfélék: floppy-diskek

és merevlemezek (melyeket Winchesternek is szoktak mondani). Tudnunk kell, hogy az MS-DOS minden lemeztípuson 512 byte-os hosszúságú szektorokkal dolgozik (jóllehet a lemezkontrollerek általában többféle szektorhosszúságú lemez kezelésére képesek).

Egy szektor nemcsak a hasznos 512 byte-nyi információt tartalmazza. A szektorok elején egy szektor-headert találunk, mely az adott szektor fizikai sajátosságait írja le (erről bővebben - a floppy-diskeket illetően - lásd a 3. kötet "A diskette" c. fejezetét). A szektor végén egy ellenőrző információ, a CRC (Cyclic Redundancy Check, ciklikus ellenőrző adat) kap helyet. Ez két byte hosszú, és úgy keletkezik, hogy a szektor tartalmát, mint egy 512x16 bites számot, elosztják egy 16 bites számmal, és az osztás maradékát írják fel a lemezre. Olvasás közben a kontroller újra kiszámítja az adatterület CRC-jét, és összeveti a lemezről olvasott CRC-vel. Ha a lemezen vagy az átvitel során valamilyen adathiba következik be, akkor a CRC nagy valószínűséggel sérült, tehát a számított CRC nem egyezik a feljegyzettel. Ez az ellenőrzés fokozza az adatbiztonságot. CRC hiba esetén csak a második (harmadik stb.) file-másolat lehet segítségünkre; valószínű, hogy a hibás lemezt nem tudjuk többé rendbehozni. A "ciklikus" szó arra utal, hogy a maradékot (egy okos matematikai trükk segítségével) úgy lehet képezni, hogy byte-ról byte-ra haladunk, és az új byte-ot egy logikai művelettel hozzávehetjük az eddig kiszámított CRC-hez. Bármelyik byte-nál szakítjuk meg a munkát, a kapott eredmény az addig feldolgozott akárhány byte-os csomag CRC-je lesz.

A floppy-disk egy viszonylag kisméretű és -kapacitású lemez, amelyet cserélhetünk a lemezegységben. Nagyjából a következő típusai ismeretesek: egy- vagy kétoldalas, 40 vagy 80 sávos, sávonként 8, 9 vagy 15 szektoros lemezek. Egy diskette kapacitása így 320 szektortól (azaz 160 Kb-tól) 2400 szektorig (1200 Kb-ig) terjedhet. A floppy-diskekről meg kell még jegyezni, hogy az adatelérés ideje igen nagy, emiatt olyan gépen dolgozni, amelyen csak floppy-disk egységek vannak (tehát klasszikus IBM PC vagy hasonló berendezés), igen barátságtalan. Arról nem is beszélve, hogy a korszerű programcsomagok szinte mindegyike feltételezi egy nagykapacitású és jó elérési idejű merevlemez létét. Ezek nélkül egy olyan fordítóprogram használata, mint pl. a Microsoft C-fordítójának 3.00 (és későbbi) változatai, gyakorlatilag lehetetlen.

A merevlemezek általában három lemezből álló lemezcsomagok, amelyeket fixen beépítettek a lemezegységbe. Ezek kapacitása és

elérési ideje lényegesen meghaladja a floppy-diskekét. A lehetséges hat oldalból négyet használnak ki. A sávok száma 305-től a csillagos égig terjed. Ez idő szerint Magyarországon a 615 sávos (21 Mb-os) lemez a legelterjedtebb, de elérhetők 60 Mb és még ennél is nagyobb méretű lemezek. Más berendezésekhez pedig már láttam 600 Mb-os merevlemezt is!

Beszélhetünk egy újabb, nem ennyire természetes lemezterület-egységről, a cilinderről (cylinder) is. Ez az adott fejállás mellett egyszerre elérhető (tehát egymás felett elhelyezkedő) sávok összessége. Egyoldalas floppy-n a cilinder eléggé elméleti jellegű területegység, kétoldalas floppy-n már értelmezhető. Valójában azonban a különféle lemezcsomagok esetében van nagyobb jelentősége, ahol a cilinder az egymás fölött elhelyezkedő 4 (vagy a lemezcsomag típusától függően 8, 18 stb.) sávot jelenti.

Az MS-DOS a lemezterület fent megadott fizikai felépítése mellett egy logikai szektorsorrendet is ismer. Ez esetben a szektorokat egytől számozzák a következőképpen: az 1. számú szektor a 0. oldal 0. sávjának 1. szektora, a 2. számú ugyanott a 2. szektor és így tovább. Ezután az 1. oldal 1. szektora következik, majd ugyanott a második stb.

XIV.2. Az MS-DOS lemezek logikai felépítése

Az MS-DOS négy fő logikai részre bontja az általa használt lemezek területét.

Boot record - a fizikailag első szektor, amely egy apró program mellett a lemez típusára utaló információkat tartalmaz. Ez a program vagy a rendszer betöltésére szolgál, vagy pedig azt írja ki, hogy a lemez nem tartalmaz tölthető operációs rendszert. A boot record a lemez DOS-számozás szerint első szektora.

FAT - (File Allocation Table) - ez a nagyon fontos táblázat a lemez adatterületének foglaltságát és szervezését írja le (lásd a "File Allocation Table" fejezetet).

Root directory - Tartalomjegyzék - 32 byte hosszúságú elemekből álló táblázat, amely a lemezen található file-okat írja le.

Adatterület - A lemeznek ez a "hasznos" területe, itt helyezkedik el a lemezen található összes file "teste", adatrésze.

A fent felsorolt elemek hossza lemeztípustól függően változik. A "boot record" hossza mindig egy szektor, azaz 512 byte. A FAT hossza annál nagyobb, minél nagyobb a lemez adatterülete. Megjegyezzük, hogy a FAT csaknem minden lemezen kétszer szerepel a nagyobb biztonság céljából. Ez elsősorban hibafelismerési, esetleg (igen intelligens programok számára) hibajavítási lehetőségeket nyújt. A "root directory" hossza szabja meg, hogy hány file-t lehet a lemezre felírni aldirectory-k használata nélkül (a szokásos floppy-disken 112, a merev lemezeken általában 508 bejegyzés számára van hely).

Végül az adatterület hossza és szervezése szintén erősen függ a lemez típusától. Az adatterületet az MS-DOS úgynevezett clusterekre, vagy allocation unit-okra, magyarul területegységekre osztja. A továbbiakban az egyszerűség kedvéért megmaradunk a cluster kifejezés mellett. Egy ilyen cluster hossza egy vagy több szektor. A szokásos kétoldalas, 40 sávos, sávonként 9 darab 512 byte-os szektorra formázott floppy-disk esetén egy cluster 2 szektor. Egy merevlemez clusterei 4 vagy 8 szektorosak lehetnek. A rendszer a file-ok számára clusterekben (és nem szektorokban, esetleg még annál is kisebb egységekben) foglal helyet. Tehát egy rövid, pl. 10-20 byte hosszúságú file is egy teljes clustert foglal le. A clusterek a lemezen láncokba szervezettek; ezt a "File Allocation Table" fejezetben olvashatjuk.

Megjegyezzük még, hogy a merevlemez ún. partíciókra osztható. Ennek leírására egy külön logikai egység, a partíciótábla szolgál, amely a lemez ún. Master Boot Record-jában (a fő boot recordban) foglal helyet. Ez egy olyan logikai egység, amely minden fent felsorolt logikai egység előtt helyezkedik el a merevlemezen. A partíciók létrehozásuk után külön életet élnek. Különböző rendszereket tartalmazhatnak, és az adott rendszer logikája szerint épülnek fel. Például gyakran osztják fel úgy a lemezt, hogy az egyik partíció MS-DOS, a másik XENIX operációs rendszert tartalmaz. Ennek fő örömei abban állnak, hogy, mint minden kompromisszum esetén, mind a két fél rosszul jár, mindkettő számára igen hamar szűknek bizonyul az adott partíció.

XIV.3. Directory-bejegyzés

A directory-bejegyzés a file eléréséhez szükséges információkat tartalmazza. A lemezen levő minden file-hoz tartozik egy directory-bejegyzés, melynek hossza 32 byte, és a directory file-ban helyezkedik el. Ez a directory lehet a ROOT directory vagy valamelyik aldirectory. Megjegyezzük, hogy a ROOT directory a lemez típusától függő, de rögzített helyen van, és semmilyen különleges bejegyzést nem tartalmaz, míg minden aldirectory egy önálló file-ként helyezkedik el a lemez adatterületén. Első két bejegyzése adminisztrációs célokra foglalt. Az első directory-bejegyzés saját magára vonatkozik. A "Név" mezőben egy "." helyezkedik el tíz szóközzel kiegészítve. A második bejegyzés pedig a közvetlenül az aldirectory felett lévő directory-ra utal. Itt a "Név" mezőben ".."-ot találunk kilenc szóközzel kiegészítve. A directory hossza (így a beleírható file-ok száma is) szintén lemeztípustól függő rögzített érték.
A directory-bejegyzés felépítése a következő (egy téglalap egy byte-ot jelent, az offsetek a pontosvessző előtt decimálisak, mögötte pedig hexadecimálisak):

0- 7;00-07	file - név								
8-10;08-0A	típus								
11 ;0B	attr								
12-21;0C-15	lefoglalt terület								
22-23;16-17	kel.idő								
24-25;18-19	kel.dátum								
26-27;1A-1B	cluster								
28-31;1C-1F	file-hossz (byte)								

File-név
Legfeljebb 8 karakter, balra igazítva, szóközökkel kiegészítve. A megengedett karakterek: nagybetűk, számjegyek és néhány egyéb

karakter, pl. "-" (kötőjel), "_" (underscore). A file-név belsejében nem szerepelhet szóköz.
A név első karaktere speciális információt hordozhat:

00	Ez a directory-bejegyzés még nem használt, soha nem tartalmazott file-t (directory logikai vége)
E5	Ez a directory-bejegyzés egy törölt file-t tartalmaz; a directory-bejegyzés többi eleme ép
2E	Ez a directory-bejegyzés az aldirectory adminisztrációs bejegyzése (. vagy .. "file")

Megjegyzés: file-ok hatékony elrejtésére és védelmére megfelelő eljárás, ha valamilyen módon kisbetűket csempészünk a file-névbe vagy -tipusba (saját programunk segítségével, vagy valamilyen "lemezoperáló" programmal, mint a Norton Utilities vagy PCTOOLS). Ez egyébként gyakorlatilag csak a directory területének direkt írásával valósítható meg (lásd "Az interruptok funkciói" fejezet, 25 és 26 interrupt). A védelem így biztos, olyannyira, hogy mi sem tudunk programjainkból a file tartalmához férni, mivel a rendszer minden megnyitási stb. kísérletnél a file-névben megadott kisbetűket nagybetűkké konvertálja. A védelem feloldása csakis újabb direkt directory-írással lehetséges. Ezért (bár a "trükk" beválik) jobb lesz más utat keresni a file-ok védelmére.

File-típus

Legfeljebb 3 karakter, balra igazítva, belső szóköz nélkül, szóközökkel kiegészítve. Megengedett karakterek mint fent.

A file-attribútum bitkiosztása

7	6	5	4	3	2	1	0
x	x	a	d	v	s	h	r/o

x - nem használt bitek

r - read only, csak olvasható file. A file-t sem rendszerparanccsal, sem programból nem tudjuk törölni.

h - hidden file, rejtett file. A rendszer listázó parancsai ezt a file-t vagy directory-t nem mutatják meg. Amennyiben programból akarunk directory-t listázni, akkor programunk csak kiterjesztett FCB vagy speciális attributum használatával "látja meg" ezeket a file-okat.

s - system file, rendszerfile. A file programként nem indítható el. A rendszer- és programparancsok nem látják az ilyen file-t, úgyhogy törlésük is lehetetlen, előbb meg kell változtatni attributumukat.

v - volume label, a lemez címkéjét tartalmazó "file": egy olyan directory-bejegyzés, amelyhez nem tartozik valódi file, egyetlen adatbyte-ja sincs. Itt a bejegyzés név- és típusmezeje adja a 11 karakteres címkét.
Megjegyezzük még, hogy lemezcímkéből egy lemezen csak egy lehet, és csak a lemez ROOT directory-jában.

d - directory-file. File-ként nem listázhatjuk és nem törölhetjük, kezelésére mind rendszerből, mind programból külön parancsok szolgálnak.

a - archive bit, jelzi, hogy ez egy közönséges file. Minden általunk létrehozott file ilyen "közönséges" file, akár forráskód, object-, program- vagy adatfile.
Ezt a bitet a BACKUP és RESTORE programpár használja. Amikor BACKUP segítségével mentünk, akkor a mentett file-ok attributumát a BACKUP 00-ra állítja, azaz törli az archív bitet. Minden módosítás automatikusan újratölti az archív bitet; tehát e bit voltaképpen azt jelzi, hogy kell-e menteni az adott file-t vagy nem, mert az utolsó mentés óta nem változott a tartalma.

Megjegyzés: a BACKUP program használata egyébként nem kockázatmentes; inkább mentsünk sokat és alaposan a

COPY paranccsal, mert ha a BACKUP által felírt lemezek egyike-másika megsérül, akkor sokkal nehezebb a mentett file-ok helyreállítása. File-jaink biztonságba helyezése során tartsuk szem előtt az "Egy mentés nem mentés" elvet, és legalább három példányban másoljunk le minden fontos file-t.

A bitek közül az alsó három megengedő vagy kapcsolatban áll egymással, azaz bármelyikük bármilyen értéket kaphat. A felső három értelemszerűen kizáró vagy kapcsolatban áll egymással: egy file vagy normális file, vagy directory file, vagy lemezcimke.

Megjegyzés: érdekes módon (noha ez nem logikus) a lemezcimke-bejegyzésben állhat együtt is az "a" és a "v" bit - ez attól függ, hogy hogyan hoztuk létre a lemezcimkét. Amennyiben az MS-DOS FORMAT programjával adtuk meg a cimkét, akkor a lemezcimke attributuma "archive" és "volume label"; ha valamilyen egyéb úton (például a méltán népszerű "NORTON utilities" megfelelő programjával), akkor csak "volume label". Ez a kis eltérés azonban a rendszert nem zavarja.
Ide kívánkozó megjegyzés még, hogy az MS-DOS ATTRIB parancsa segítségével változtathatjuk meg a file-ok "r/o" attributumát. Ezzel védetté tehetünk file-okat, vagy megszüntethetjük védelmüket.

Az időt tartalmazó szó szerkezete

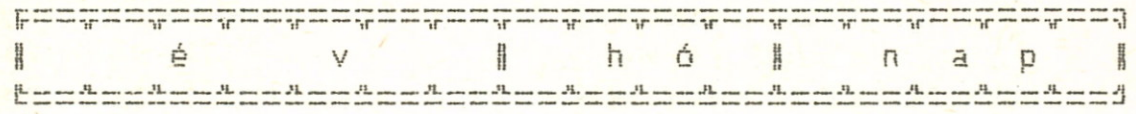
(a jobb szélső bit a legalacsonyabb helyiértékű)

15 14 . . . 9 8 7 6 . . . 1 0
óra	perc	2 másodperc

a legalsó 5 bit a keletkezés másodpercére utal, két másodperces növekményben feljegyezve (hiszen öt biten nem tudjuk a 60 másodpercet megadni).

Megjegyzés: a "DOSCALL.INC" függelék egy include file, amely a dátum és az idő kezelését megkönnyítő rekordokat és makrókat is tartalmaza.

A dátumot tartalmazó szó szerkezete
(a jobb szélső bit a legalacsonyabb helyiértékű)



File-hossz

A file byte-okban mért hossza (duplaszó). Az alacsonyabb helyiértékű szó helyezkedik el az alacsonyabb címen.

Cluster

A file kezdő clusterének, allokációs egységének sorszáma. A további allokációs egységek a FAT-ban vannak láncszerűen felfűzve (lásd "File Allocation Table" fejezet).

A fentiekből számos hasznos következtetés adódik az MS-DOS file-kezeléséről. Az első és legfontosabb az, hogy mi történik egy file törlése esetén. A rendszer felszabadítja a lefoglalt clustereket azzal, hogy a lánc minden elemét kinullázza, majd a directory-bejegyzés (és benne a file-név) első byte-ját az E5 karakterrel írja felül, ezzel jelezve azt, hogy ez a directory-bejegyzés már szabad. Ebből adódik az is, hogy a file törlése és az első rákövetkező írás között a file sértetlenül megvan a lemezen, csak helyre kell állítani a directory-bejegyzést, és "össze kell vadászni" a clustereket. Amennyiben a file folytonos volt (a rendszer, ha csak lehet, törekszik erre), ez utóbbi nem túl bonyolult feladat. Amennyiben egy hosszú file nem volt folytonos, akkor elég nehéz lehet a clusterek összeszedése. Egy hosszú bináris file helyreállítása pedig szinte reménytelen.

> Megjegyzés: ha az MS-DOS logikai lemezkezelése kicsit más lenne, akkor nagyobb esélyt lehetne biztosítani a törölt file-ok helyreállítására. Az ötlet az, hogy a frissen formázott lemezen létre kellene hozni egy file-t, amely az összes szabad clustert tartalmazza. Ennek cluster-láncából lehetne aztán a file-ok számára lemezterületet szerezni, a törölt vagy lecsonkított file-okból felszabaduló clustereket pedig a szabad clusterek láncának végére fűzni. Ezzel garantáljuk azt, hogy

- megmarad a láncolási információ;
- a törölt file-ok területét használjuk utoljára;
- a teljes lemezfelületet egyenletesen használjuk, de
- a legtöbb file nem lesz folytonos (a rövidek sem);
- nagyon sok fejmozgatás kell az adateléréshez.

Valószínű, hogy a rendszer megvalósítói a két utóbbi (és igen súlyos) negatívum miatt mondtak le az első három pozitívumról.

File-ok létrehozása során a rendszer arra törekszik, hogy a file-ok lehetőleg alacsonyabb sorszámú clustereket foglaljanak el, a directory-bejegyzések pedig folytonosan helyezkedjenek el, tehát ne legyen a directory belsejében soha nem használt bejegyzés. A különféle, directory-ban kereső funkciók abbahagyhatják tevékenységüket, amint rábukkannak az első soha nem használt bejegyzésre. File létrehozása esetén tehát a rendszer megkeresi az első, pillanatnyilag nem használt directory-bejegyzést (a gyakorlatban ez legtöbbször egy törölt file E5 karakterrel kezdődő bejegyzése), ide beírja a file adatait, majd clustereket foglal a FAT alapján. A szabad clusterek keresését szintén az alacsonyabb számú clustereken kezdi. Ha új lemezt kezdünk írni, akkor (az első törlésig) minden file folytonos lesz. Ha már megkezdjük a file-ok törlését is, akkor az így felszabaduló clusterekre kerülnek az újabb file-ok. Ezek már valószínűleg nem lesznek folytonosak.

XIV.4. File Allocation Table

A File Allocation Table, a file-területfoglalási táblázat, amelyet egyszerűen FAT-nak nevezünk, egy egész számokból álló táblázat, amelynek minden egyes eleme hozzá van rendelve a lemez adatclustereihez. Az első két elem (indexük 0 és 1) speciális információt hordoz, a továbbiak pedig indexük szerint tartoznak a 2., 3. stb. adatclusterhez (az adatclusterek számozása 2-től indul).

(1) A FAT 0. indexű tagja (melynek külön neve FAT-azonosító) a lemez típusát jelenti, arra utal, hogy a lemez hány oldalas, oldalanként hány sávos, sávonként hány szektort tartalmaz.

Lássuk például a floppy-diskek FAT-azonosítóit. Az alábbi táblázat tartalmazza a FAT-terület és a directory hosszát szektorokban, valamint a directory-terület kezdőszektorát DOS-számozás szerint:

Oldalak száma	Sávok száma	Szektorok sz.	Típusjel	FAT-azonosító	FAT	Dir.	Dir.
2	40	8	D-8	-1	2	7	3
1	40	8	S-8	-2	2	4	3
2	40	9	D-9	-3	4	7	5
1	40	9	S-9	-4	4	4	5
2	80	9	QD-9	-7	10	7	11
2	80	15	QS-15	-7	14	14	15

Merevlemezek FAT-azonosítója: -8

A merevlemezek FAT-területének, directory-jának adatait nem tudjuk egyszerűen összefoglalni, mert függ a lemez méretétől és particionálásától is.

(2) A következő (az 1 indexű) FAT-elem értéke mindig -1.

(3) A további FAT-bejegyzések tartalmazzák a láncolási információt. A file-hoz tartozó clusterek közül az első számát adja meg a directory-bejegyzés. Az ehhez rendelt FAT-bejegyzés tartalma adja meg a file második clusterének sorszámát, a másodikhoz rendelt a harmadikét stb. Az utolsó clusterhez tartozó FAT elem értéke -1.

(4) Amennyiben a lemez valamelyik clustere hibásnak bizonyul, ezt az MS-DOS egy -9 értékkel jelöli meg a hozzárendelt FAT elemben. A szabad clusterekhez tartozó FAT elem 0-t tartalmaz.

Ebből látható, hogy egy file tetszőlegesen hosszú lehet, clusterei nem szükségképpen folytonosan helyezkednek el, és fizikailag akármilyen sorrendben követhetik egymást. Az is látható, hogy nem kell az egész file-t végigolvasni ahhoz, hogy a végén levő byte-okhoz férjünk, egyszerűen végig lehet lépkedni a FAT-on is.

A FAT egy elemének mérete függ a lemez típusától. Floppy-diskeken egy elem hossza 12 bit, tehát három szomszédos byte két FAT-bejegyzést tartalmaz. Egy ilyen FAT-bejegyzéssel elvben

4096 cluster címezhető meg. A 12 bites FAT-bejegyzés fő "előnye" (a helytakarékosság mellet) az, hogy a felhasználó nem szívesen vállalja a bitek közvetlen kezelését, és remélhető, hogy nem kotorász a FAT-ban; ezt pedig nagyon szeretik a rendszerprogramozók! Egy 10 Mb-os Winchester kb. 2500 clustert tartalmaz, ezért az ő FAT-ja szintén lehet 12 bites méretű. Az ennél nagyobb kapacitású merevlemezek clustereinek száma azonban meghaladja a 12 biten megcímezhető mennyiséget (kb. 4000), így itt a FAT méretét 16 bitre növelték.

Mivel a FAT-elem hossza a lemeztípustól függ, nem adtuk meg korábban hexadecimálisan az egyes FAT elemek speciális tartalmát; pl. a file-t záró -1 ábrázolása FFFH vagy FFFFH is lehet.

XIV.5. File Control Block

XIV.5.1. A kiterjesztett FCB elemei

-7	0FFH	jelzi, hogy ez kiterjesztés
-6 - -2	lefoglalt terület	
-1	attr.	file-attributum

Igen fontos megjegyezni, hogy kiterjesztett FCB használata esetén a kiterjesztés offsetjét kell betölteni a DX regiszterbe annak ellenére, hogy az offsetek a szabványos FCB kezdőcímének megadását sugallják.

XIV.5.2. A szabványos FCB elemei

Azokat az elemeket, ahol a (DOS) megjegyzés szerepel, a rendszer tölti ki a file megnyitásakor. A (*)-al megjelölt elemeket a file megnyitása előtt, a (**)-al jelölteket közvetlenül utána a programnak kell kitöltenie. A (***)-al jelölt elem bármikor módosítható. Egy téglalap egy byte-ot jelent, az offsetek a pontosvessző előtt decimálisak, mögötte pedig hexadecimálisak.

0 ;00	dr	drive kódja	(*)
1- 8;01-08	f i l e - n é v (8 k a r.)		név (*)
9-11;09-0B	t i p u s	típus	(*)
12-13;0C-0D	blokk	kurrens blokk	(DOS)
14-15;0E-0F	rekordh.	rekordhosszúság (DOS)	(**)
16-19;10-13	file-hossz	file-hossz	(DOS)
20-21;14-15	dátum	keletkezés dátuma	(DOS)
22-23;16-17	idő	keletkezés órája-perce	(DOS)
24-31;18-1F	l e f o g l a l t t e r ü l e t		
32 ;20	r.sz.	rekordszám a blokkban	(DOS)
33-36;21-24	random rekord szám	direkt rekord száma	(***)

File-attributum

Lásd "Directory-bejegyzés" fejezet.

Drive kódja

A "dr" egy sorszám, amely a használni kívánt lemezegység kódja. File-nyitás előtt töltendő ki. Jelentése a következő:

0 - a kurrens lemez
1 - A:
2 - B:
stb.

Név és típus

Lásd "Directory-bejegyzés" fejezet.

Kurrens blokkszám

Egy blokk 128 rekordból áll, melyek hosszát a logikai rekord-hossz-mező tartalmazza. Szekvenciális műveleteknél használatos,

eléggé elméleti jellegű egység. Kezdetben 0-ra áll be. Minden 128. rekord elérése után eggyel megnövekszik, a rekordszámláló pedig (amely a blokkon belüli rekordokat számlálja) visszaáll nullára.

Logikai rekord hossza
A logikai rekord hosszát a file megnyitása és az első beolvasás vagy kiírás között a felhasználó tetszőlegesen beállíthatja. A file megnyitásakor automatikusan hexadecimális 80-ra áll be.

File-hossz
A file hossza byte-okban. Megnyitás közben a rendszer kitölti; lásd "Directory-bejegyzés" fejezet.

Dátum és idő
Az utolsó írás dátuma és ideje. Megnyitás közben a rendszer kitölti; lásd "Directory-bejegyzés" fejezet.

Rekordszám a blokkban
0-tól 127-ig növekvő egész, amely azt mutatja, a blokkon belül hányadik rekord kezelésénél tartunk éppen. Ha növekedés közben eléri a 128-at, akkor a blokksorszám eggyel megnő, ez a mező pedig visszaáll 0-ra.

Random rekord sorszáma
A random (direkt) file-kezelés során van szerepe. Itt állíthatjuk be, hogy a rekordhosszban megszabott hosszúságú rekordok közül hányadikat akarjuk beolvasni. Tehát összesen hat byte-nyi helyünk van a file-pozíció megadására: a szavas rekordhossz és a rekord duplaszavas sorszáma. Ezek szorzata a file-pozíció.

XIV.6. Program Segment Prefix, PSP

Futtatáskor ezt a táblát helyezi el a rendszer minden program elején (lásd "Amikor a program belép" fejezet). Ez a terület a rendszer és a program kapcsolatát biztosítja. Az indításkor a rendszer megkeresi azt a legalacsonyabb szabad területet, ahová a program betölthető, itt építi fel a PSP-t, bemásolja a programot a PSP mögé, ha szükséges, elvégzi a relokációt, majd átadja a vezérlést a program kezdőcímére.

Pozíció dec.	Pozíció hexad.	Tartalom
0-1	00-01	CD 20, INT 20 utasítás, kilépés a programból
2-3	02-03	a memória tetejének paragrafuscíme
4	04	foglalt
5-9	05-09	hosszú ugrás a DOS funkciók futását előkészítő kódra (lásd "A futó program és az MS-DOS" fejezetet)
10-13	0A-0D	kilépési cím (alul IP, felül CS)
14-17	0E-11	CTRL-BREAK kilépési cím (alul IP, felül CS)
18-21	12-15	kilépési cím kritikus hiba esetén (alul IP, felül CS)

Megjegyzés: ezeket a címeket a programozó akkor használhatja fel, ha valamilyen okból azt akarja, hogy kilövése vagy kritikus hiba miatt történő kilépése esetén a vezérlés ne a rendszer, hanem saját megfelelő rutinjaira kerüljön, felülírja a 23 és 24 interrupt-vektorokat. Normális kilépés esetén PSP-jéből is helyreállíthatja e vektorok eredeti tartalmát.
A vektorok átírására akkor van szükség, ha a program számít olyan kritikus hibára, melyet saját hatáskörében kell rendeznie. Másik jellegzetes eset az, ha olyan programo(ka)t akar elindítani, melyeket a felhasználó kilőhet, vagy amelyek kritikus hibákat válthatnak ki, és ezeket az eseményeket valamilyen okból a vezérlőprogramnak (tehát a kilőtt programot elindí-

tó programnak) és nem a rendszernek kell lekezelnie. Ekkor az elindított program PSP-jében persze az átírt értékek szerepelnek majd.

22-91	16-5B	A DOS használatára fenntartva; 2C-2D tartalmazza a környezet szegmenscímét (lásd lejjebb).

Megjegyzés: 50H-n egy

```
INT     21H
RETF
```

utasítássorozatot találunk, ez (távoli hívással) alkalmas a DOS-funkciók aktivizálására. Nem támogatott DOS-hívási eljárás!

Fontos megszorítás: a PSP eddigi részeit (a 00 offsettől az 5B offsetig) nem szabad felülírni!

92-107	5C-6B	formázott FCB - a parancssorban megadott első file-névvel. "Parancssor" alatt a programot elindító parancssort értjük.
108-127	6C-7F	formázott FCB - a parancssorban megadott második file-névvel. Ez felülíródik, ha az elsődleges FCB-vel file-t nyitunk meg!
128-255	80-FF	formázatlan paraméterterület - ez az alapértelmezés szerinti DTA. A formázatlan paraméterterület első byte-ja (a 80H offseten) tartalmazza a paraméterterület hosszát byte-okban, a 81H byte-tól kezdve pedig a programot elindító parancssor paraméterei olvashatók a program neve nélkül. A szöveget a CR (Carriage Return) karakter zárja le, de ez a karakter nem számít bele a sor hosszába.

128-255	80-FF	A parancssor bármelyik pontján megadott átirányítási paraméter(ek)et értelmezi az MS-DOS, de a programnak nem továbbítja, azaz nem másolja be a paraméterterületre. Ez összhangban van azzal az elvvel, hogy a program nem tudhatja meg, mi a standard file-ok valódi jelentése. Természetesen van arra lehetősége, hogy valamilyen úton kinyomozza, mi is van valóban a standard input vagy output mögött, de ez nem számít "törvényes" lehetőségnek.
(folytatás az előző oldalról)		

XIV.7. Környezet (environment)

A környezet egy legfeljebb 32 Kbyte-os terület, amelyen 0-val lezárt ASCII-stringek ugyancsak 0-val lezárt sorozata helyezkedik el általában (bár nem szükségképpen) a

NAME=string

formában. Ezt a területet az MS-DOS programbetöltő rutinja vagy a más programot meghívó felhasználói program készíti elő, és szegmenscímét a rendszer a program hívásakor a PSP 2C-edik szaván adja át a programnak. A környezet definiálására és tartalmának elolvasására példaként lásd a "Memóriakezelés és programvezérlés" fejezet programjait.

A rendszer szokásos környezete alapértelmezésben a

COMSPEC=... a COMMAND.COM helye
PATH=... aktivizált directory-k
(PROMPT=...) a PROMPT paranccsal megadott prompt

sorokat tartalmazza, valamint mindazon szövegeket, melyeket a SET paranccsal a felhasználó épít be a környezetbe. Például a Microsoft 3.00 vagy 4.00 verziójú C-fordítója (és a hozzá tartozó LINK) a PATH mellett a következő parancssorokat tudja értelmezni:

INCLUDE=dr:path
LIB=dr:path
TEMP=dr:path

amelyek közül az első sor az include file-ok fellelési helyét adja meg, a második sor a könyvtárakét, az utolsó az ideiglene-

sen létrehozandó munkafile-ok számára fenntartott helyet. Ez a példa jól illusztrálja a környezet felhasználási lehetőségeit: olyan információkat adunk így át a programnak, amelyeket nem kell "bedrótozni", hiszen a szervezéstől, a felhasználás módjától stb. függően akár minden futásnál más és más lehet ez az információ. Ez tehát valóban a program "környezete", azon adatok gyűjteménye, amelyek leírják a program futásának pillanatnyi körülményeit.

XIV.8. Az EXE típusú file-ok headerje

Egy EXE file két részből áll. Az első az a header, amely a file belső formátumát írja le, a második pedig a lefordított és összeszerkesztett szegmensek másolata. Ez utóbbit kell betölteni a tárba a program futtatásához. Az első rész pedig azt írja le, hogyan kell értelmezni a másodikat. Az elsőt a továbbiakban headernek, a másodikat pedig programmásolatnak nevezzük.

A memória byte-jairól ránézésre nem lehet megállapítani, hogy egy utasítás részei-e, vagy valamilyen numerikus vagy karakteres jellegű adaté. Szinte minden byte így is, úgy is értelmezhető. Tartalmat csak az ad neki, hogyan használjuk fel.

A programkódot az adatoktól mindössze az különbözteti meg, hogy a program végrehajtódik, azaz valamilyen úton rá adódik a vezérlés. Ha programunk most megcímez bizonyos memóriaterületeket, és tartalmukat aritmetikai műveletek elvégzésére használja, akkor az adott byte-okat numerikus adatként kezeltük. De ugyanúgy összeadhatunk két karakterkódot vagy utasításrészletet, mint két bármilyen egyéb byte-ot, hiszen csak byte-okról van szó. Ha a program ehelyett byte-onként kiküldi az adott terület tartalmát a terminálra, akkor karaktersorozatnak fogtuk föl a megcímzett memóriaterületet, bármi volt is tartalma. Természetesen programként végre lehet hajtani egy adatterület tartalmát is, ettől azonban nem várhatunk jó eredményeket.

Amikor megírunk és összeszerkesztünk egy programot, akkor az elkészült kódot a LINK kimásolja egy file-ba. A byte-halmazban a kódot csak az tünteti ki, hogy betöltéskor valamelyik utasításán (valamelyik byte-ján) kezdődik a program végrehajtása.

A headerben talán a legfontosabb két kitüntetett terület leírása. Ezek: a fő kódszegmens (ahol a belépési utasítás van) és a verem. Az összes többi byte értelmezése már a kódtól függ.

0- 1	5AH 4DH	jelzi, hogy ez egy EXE file
2- 3	maradék	a file hossza modulo 256
4- 5	lapszám	a file hossza 256 byte-os lapokban
6- 7	rel.t.h.	relokációs tábla hossza
8- 9	header h.	a header hossza paragrafusokban
10-11	minimum	a program legalább ennyi memóriát allokál még
12-13	maximum	a program legfeljebb
14-15	stack táv.	stack-terület távolsága (paragrafus)
16-17	SP érték	Stack Pointer kezdeti értéke
18-19	checksum	ellenőrző összeg
20-21	IP érték	Instruction Pointer kezdeti értéke
22-23	kód táv.	kódterület távolsága (paragrafus)
24-25	rel. táv.	első relokációs bejegyzés távolsága
26-27	overlay k.	overlay-kód (főmodul esetén 0).

Az első szó a LINK "aláírása", mely jelzi, hogy ez egy legális EXE típusú file. A következő két szó ("maradék" és "lapszám") tartalmazza a program hosszát lapokban, illetve a maradékot. Ez a maradék kerül majd be CX-be.

A "rel.t.h." a relokációs tábla bejegyzéseinek számát tartalmazza (lásd lejjebb). A "header h." a headerterület hossza paragrafusokban, azaz 16 byte-os egységekben. Ebből határozza meg a rendszer, hol kezdődik a file-ban valójában a programmásolat. A "minimum" és a "maximum" a program által igényelt plusz memóriaterület szükséges legkisebb, és lehetséges legnagyobb értékét adja meg paragrafusokban (értéküket alapértelmezés szerint 0000H

és FFFFH; a LINK korszerű verzióiban az utóbbi a /CPARMAXALLOC kapcsolóval szabályozható; lásd az MS-DOS kézikönyveket).

A "stack táv." és "SP értéke" a vermet írja le: megadja a verem távolságát a programmásolat kezdetétől paragrafusokban (azaz szegmensformában), valamint a Stack Pointer kívánt kezdeti értékét. A rendszer a következőképpen határozza meg a Stack Segment regiszter kezdeti értékét: összeadja a veremszegmensnek a programmásolat kezdetétől paragrafusokban mért távolságát és a betöltési cím szegmensét.

A "checksum" úgy keletkezik, hogy a LINK összeadja a file szavait, a túlcsordulást figyelmen kívül hagyva.

"IP érték" és "kód táv." a CS és az IP tartalmát írja elő. A CS kezdeti értékét egy összeg adja: a kódszegmensnek a programmásolat kezdetétől vett távolsága és a betöltési cím szegmense.

A "rel.táv." byte-okban adja meg a relokációs tábla első elemének távolságát a file kezdetétől. Végül az "overlay k." az overlay-struktúrák kialakításában játszik szerepet, az overlay-szintet határozza meg. A főprogram esetén ez a szint 0.

Relokáció

Mint tudjuk, a rendszer csak a betöltés során határozza meg, hol fut a program (általában a legalsó szabad memóriaterületre helyezi el). Ahhoz azonban, hogy a program futás közben be tudja tölteni a szegmenscímeket a szegmensregiszterekbe, tudnia kell, hogy pontosan milyen címeken fut a program.

A LINK a programmásolatot úgy készíti el, mintha a program a 00000H címen kezdődne (az öt nulla nem véletlen: 20 bites abszolút címről van szó!). Valahányszor immediate címzéssel hivatkozunk egy szegmensre, a LINK a kért szegmensnek a 00000H címtől vett távolságát tölti az utasítás operandusmezejébe.

A betöltés során a rendszer tudja már, hogy hol fut a program, tehát minden egyes esetben, amikor szegmensre hivatkozik a program, a valódi kezdőcímet hozzá kell adni az EXE file-ban megadott címhez. Ez történik az SS és CS kezdeti értékének meghatározásakor is, a relatív távolságot adják a fizikai kezdőcímhez, és ez az összeg kerül be a szegmensregiszterekbe.

A további szegmenshivatkozások a kódban vannak elszórva egyrészt beépített adatcímzésű utasítások operandusmezejében, másrészt pedig a szegmensnevekre hivatkozó szavas és duplaszavas változókban. Természetesen ezekhez is hozzá kell adni a fizikai kezdőcímet. Erre szolgál a relokációs tábla, amely a header

fent leírt formázott része után helyezkedik el. Elemeinek számát és első elemének a file kezdetétől vett távolságát a header tartalmazza. A relokációs tábla elemeinek száma megegyezik a szegmenshivatkozások számával (ez teszi változó hosszúságúvá az EXE headert).

A relokációs tábla egy eleme két szavas. Ez tartalmazza a szegmensre hivatkozó utasítás távolságát a programterület kezdetétől offset/szegmens formában (azért kell két szó, hogy a programterület hossza túlléphesse a 64Kb-ot). A betöltés utolsó lépéseként a rendszer sorra "felkeresi" a relokációs tábla elemei által címzett szegmeshivatkozásokat, és mindegyikhez hozzáadja a fizikai kezdőcímet, tehát a DOS minden egyes betöltéskor felülírja a programterület egy részét. Ez nem látszik elegáns megoldásnak, de ha elég nagy biztonsággal működik (márpedig a DOS nagyon riktán ront el valamit), akkor a célnak megfelel. Arról nem is beszélve, hogy ennél szebb megoldás aligha képzelhető el. A megoldandó feladat ugyanis nem egyszerű: úgy kell adatok regiszterbe töltését biztosítani, hogy azok a szegmensregiszterek állásától függetlenül mindig rendelkezésre álljanak. Ez csak beépített adatcímzéssel (immediate címzéssel) képzelhető el, mert CS mutat csak mindig ismert helyre (ti. arra az utasításra [pontosabban annak szegmensére]), melyet a processzor éppen végrehajt. Ha viszont az adatot beépített adatként kezeljük, akkor szó sem lehet semmiféle elegáns címtáblázatról, és betöltéskor felül kell írni a programkódot, hacsak nem akarunk lemondani a program áthelyezhetőségéről, relokálhatóságáról. Erről azonban nem mondhatunk le, mert ezzel elveszne a programok verziófüggetlensége és a program programból való elindíthatósága; hiszen akkor a program csak azon a memóriaterületen futhatna, ahová az adott MS-DOS verzió LINK programja szerkesztette.

XV. Ami a példaprogramokból kimaradt

Ebben a fejezetben néhány olyan példát olvashatunk, amelyek (természetesen) hiányoznak az egyes funkciócsaládok bemutatásánál, és nem kaptak helyet ott, ahol a logikusan kellett volna.

Az alábbi programok az MS-DOS-nak a 21H-tól különböző interruptjainak felhasználását illusztrálják. Igen nyomós okaink vannak arra, hogy ezeket ne zsúfoljuk bele elvileg logikus helyükre, az adott interrupt ismertetéséhez. Az első az, hogy a példák ott (szerteágazó voltuk miatt) sokkal jobban elvonták volna a figyelmet a lényegről. A másik pedig az, hogy e programok megvalósítása közben használni akarjuk mindazon eredményeket, amelyeket az eddigi példaprogramok során folyamatos fejlődéssel elértünk. Ha e programok így szerepeltek volna a könyv elején, legalábbis meghökkentették volna a programozásban nem elég jártas Olvasót. Ha viszont nem vetünk be minden, eddig kimunkált eszközt, akkor túlzottan bonyolult kódolásra lett volna szükség, ami többszörösére növelte volna a programok méretét. Reméljük, hogy "annyi balszerencse közt, oly sok viszály után" már nem lesz megrázó az a néhány apró ravaszság, amit az itteni példaprogramokban (eddigi munkánk megkoronázásaként) itt megengedünk magunknak.

XV.1. Egy "kilőhetetlen" program

Próbáljunk meg először egy olyan programot írni, amelyet nem lehet kilőni, azaz a felhasználó hiába ad ki egy CTRL-BREAK-et, a program nem hajlandó kilépni. Erre több lehetőségünk van. Az egyik egy eléggé veszélyes módszer: a processzort egy CLI utasítással "disable interrupt" módba léptetjük és kiadunk egy HLT utasítást. Garantált, hogy csak "hardware reset"-el (azaz a gép ki- és bekapcsolásával) szabadulhatunk meg a programtól. Ez egy valóban kilőhetetlen program, mivel a CLI utasítás miatt még a klaviatúra hardware-interruptja sem jut érvényre, így nem segít a CTRL-ALT-DELETE kombináció sem. Egy másik lehetőség az, hogy a programot egy olyan végtelen ciklusba léptetjük, amely egyetlen utasításból áll:

```
LOOP:
        JMP     LOOP
```

Ez a program is kilőhetetlen, mivel nem hívja meg az operációs rendszert, tehát sosincs CTRL-BREAK ellenőrzés. Ha előzőleg CLI utasítást is kiadtunk volna, akkor még a CTRL-ALT-DELETE sem segítene.

Akinek ezek az utak túl gorombának tűnnek (nekem is), kénytelen egy ennél elegánsabb, de kissé munkaigényesebb programot írni. Ez a program elmenti a 23 interrupt-vektor tartalmát, és saját rutinjának címével írja felül. A rendszer a CTRL-BREAK észlelése után aktivizálja a 23 interrupt-rutint, amelynek eredetije abortálná a programot. A mi rutinunk azonban csak annyit tesz, hogy kiírja:

Juszt se lépek ki!

és csak szépszerével, a megfelelő szöveg beolvasása után hajlandó távozni a rendszerből. Ez nem különösebben nagy művészet; lássuk megvalósítva!

```
TITLE   CTRL-BREAK interrupt használata
        EXTRN   PRTERRMSG:NEAR, STRCMP:NEAR
;
; Konstansok, makrók, rekordok
;
        .XLIST                  ;DOSCALL.INC-et nem listázzuk!
        INCLUDE         DOSCALL.INC
        .LIST
;--------------------------------;
; Adatterületek
;--------------------------------;

DATA    SEGMENT PARA    PUBLIC  'DATA'
        MSGDEF  MSG_INT, 'Juszt se lépek ki!'
        MSGDEF  MSG_CRLF
PASSWD  DB      'Isten veled', 0        ;Kilépési jelszó
        KEYBUF  RD, 12                  ;Input buffer
DATA    ENDS
;--------------------------------;
; Stack-terület
;--------------------------------;
STACK   SEGMENT PARA    STACK   'STACK'
        DW      100     DUP ( ? )       ;Hossza 100 szó
STACK   ENDS                    ;-------;
```

```
;---------------------------------;
; Kódszegmens
;-----------------------------------------;

CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:DATA, SS:STACK, ES:NOTHING

PSP_SEG         DW      0               ;
                                        ;
START:                                  ;
                MOV     CS:PSP_SEG, DS  ;
                MOV     AX, DATA        ;
                MOV     DS, AX          ;
                                        ;
                CALL    SETINT23        ;23 vektor előkészítése
                PUSH    DS              ;
                POP     ES              ;ES -> DATA szegmens
LOOP:                                   ;
                MOV     DX, OFFSET  RD_BUFFER
                DOSCALL DOS_STRDSTR     ;Beolv. egy stringet
                MOV     BL, RD_CHN      ;
                XOR     BH, BH          ;-------;
                MOV     RD_BUF[ BX ], CHR_EOS   ;Lezárjuk
                MOV     SI, OFFSET PASSWD       ;STRCMP számára
                MOV     DI, OFFSET RD_BUF       ;
                CALL    STRCMP          ;-------;A jelszó volt?
                JE      SUCC_QUIT       ;Igen, kilépünk
                PRTSTR  MSG_CRLF        ;Nem, CR-LF és folyt.
                JMP     LOOP            ;
SUCC_QUIT:                              ;
                CALL    RESETINT23      ;23. vektor helyreáll.
                JMP     SUCC_EXIT       ;
                EXIT                    ;
                                        ;
;---------------------------------------;
; SETINT23      - A CTRL-BREAK interrupt előkészítése
;       Ez a rutin fix értékre állítja be a CTRL-BREAK
;       interrupt-vektort az eredeti érték biztos helyre
;       mentésével. Változat: egy szegmensregiszter-regiszter
;       párban (pl. kis programoknál CS:BX) paraméterként
;       kaphatja meg az interrupt-rutin címét.
;---------------------------------------;
```

```
SETINT23        PROC    NEAR            ;
                PUSH    AX              ;
                PUSH    BX              ;
                PUSH    DX              ;
                PUSH    DS              ;Regiszterek mentése
                PUSH    ES              ;----------------;
                MOV     AL, INT_CTRB_ADDR               ;(*)
                DOSCALL DOS_GETITVECT           ;-------;
                MOV     CS:SAV_INT23.OFFS, BX   ;Eredeti ér-
                MOV     CS:SAV_INT23.SEGM, ES   ; tékek mentése
                LDS     DX, CS:OWN_INT23_ADD    ;-------;(*)
                MOV     AL, INT_CTRB_ADDR               ;(*)
                DOSCALL DOS_SETITVECT   ;Saját cim letöltése
                                        ;
                POP     ES              ;
                POP     DS              ;
                POP     DX              ;
                POP     BX              ;
                POP     AX              ;Regiszterek helyreáll.
                RET                     ;
SETINT23        END                     ;
                                        ;
SAV_INT23       LONGPNT <>              ;
OWN_INT23_ADD   DD      OWN_INT23       ;
                                        ;
;---------------------------------------;
; RESETINT23    - A CTRL-BREAK interrupt-vektor helyreállítása
;---------------------------------------;
RESETINT23      PROC    NEAR            ;
                PUSH    AX              ;
                PUSH    DX              ;
                PUSH    DS              ;Regiszterek mentése
                                        ;----------------;
                LDS     DX, DWORD PTR CS:SAV_INT23      ;(*)
                MOV     AL, INT_CTRB_ADDR               ;(*)
                DOSCALL DOS_SETITVECT   ;Eredeti 23. vektor
                                        ;
                POP     DS              ;
                POP     DX              ;
                POP     AX              ;
                RET                     ;
RESETINT23      ENDP                    ;
```

```
;-----------------------------------------;
; OWN_INT23     - saját CTRL-BREAK rutin
;-----------------------------------------;
                                          ;
OWN_INT23       PROC    FAR               ;
                PUSH    AX                ;Minden regisztert
                PUSH    BX                ; menteni kell(ene),
                PUSH    CX                ; különösen akkor,
                PUSH    DX                ; ha lenne program is!
                PRTSTR  MSG_INT           ;Csak egy üzenet
                POP     DX                ;
                POP     CX                ;
                POP     BX                ;
                POP     AX                ;
                CLC                       ;Nem lépünk ki!
                RET                       ;Ez egy hosszú RET
                                          ;
OWN_INT23       ENDP                      ;
CODE    ENDS                              ;
        END     START
```

Talán említeni sem kell, hogy a 23 interrupt-vektor kezelésére szolgáló kódot azért érdemes szubrutin formájában megírni, hogy (esetleg egy könyvtárból) más programban is kényelmesen használhassuk.

Ha a fenti programot lefuttatjuk, akkor azt is megértjük, miért szerepelt a "kilőhetetlen" szó idézőjelben a fejezet címében: ez a program korántsem kilőhetetlen, csak a CTRL-BREAK-el nem tudjuk távozásra bírni. A melegindítás, a CTRL-ALT-DELETE természetesen hatásos lesz. Ha a melegindítást is meg akarjuk akadályozni, akkor vagy le kell tiltanunk a klaviatúra hardware interruptjának fogadását, vagy pedig saját kezünkbe kell vennünk ezen interrupt kezelését. Az első út azért járhatatlan, mert ebben az esetben a program (legalábbis a szokásos módon) nem tud semmilyen inputot végezni. Az utóbbi út némely ismeretek megszerzése után nem nagyon nehéz. Erre a 3. kötet "A klaviatúra" c. fejezetében olvashatunk példát.

Iktassuk még ide a stringeket összehasonlító rutin kódját! Ez egy nagyon egyszerű kis rutin, de alaposan át kell gondolni, hogy jól kezeli-e a különféle eseteket, pl. ha az egyik string "része" a másiknak, azaz pont olyan, mint az, csak előbb ér véget. Ezt a "bonyolult" vizsgálatot az Olvasóra bízzuk.

```
;-----------------------------------------;
; STRCMP        - Két, 0-val lezárt string összehasonlítása
;       Input:
;               DS:SI   - egyik string kezdőcíme
;               ES:DI   - másik string kezdőcíme
;       Output:
;               Flagek  - az összehasonlítás eredménye, mely
;                         ASCII string esetén előjeles, kiter-
;                         jesztett jelkészlettel előjel nélküli
;                         feltételvizsgálatokkal tesztelendő.
;               A Direction flag állása UP (0)
;               Minden egyéb regiszter változatlan
;-----------------------------------------;
                                        ;
STRCMP  PROC    NEAR                    ;
                PUSH    AX              ;
                PUSH    CX              ;
                PUSH    SI              ;
                PUSH    DI              ;Regiszterek mentése
                CLD                     ;Felfelé haladunk
                XOR     CX, CX          ;Stringek hossza 64 Kb!
STRCMP_LOOP:                            ;
                LODSB                   ;Egyik string köv.eleme
                SCASB                   ;Másikkal összehas.
                JNE     STRCMP_DIFF     ;Nem egyenlő - vége
                TEST    AL, AL          ;CHR_EOS-on állunk?
                JZ      STRCMP_EQU      ;Ha igen, egyenlőek
                LOOP    STRCMP_LOOP     ;Ha nem, folytatjuk
STRCMP_NOEOF:                           ;
STRCMP_DIFF:                            ;A flagek helyesen
STRCMP_EQU:                             ; állnak!!
                POP     DI              ;
                POP     SI              ;
                POP     CX              ;
                POP     AX              ;
                RET                     ;
STRCMP  ENDP                            ;
```

Talán magyarázni sem kell, hogy STRCMP_NOEOF, STRCMP_EQU és STRCMP_DIFF logikailag különböző belépési pontok, de gyakorlatilag egybeesnek, hiszen az STRCMP_EQU-ra mindenképpen Zero=1 esetén ugrunk, STRCMP_DIFF-re olyan összehasonlítás után, ami-

kor különböző karaktereket találtunk. Végül STRCMP_NOEOF a ciklusból való kicsorgás esetére vonatkozik, amikor végig azonosak voltak a stringek, de nem találtuk meg a CHR_EOS-t, azaz 64 Kbon át nem szerepelt egyetlen 0 sem. Tekintettel arra, hogy egy string hossza a gyakorlatban nem haladhatja meg ezt a méretet, ilyenkor nyugodt lélekkel mondhatjuk azonosnak a két stringet.

Egy nagyon fontos dologra utal a szinopszisban a feltételvizsgálatokra vonatkozó kitétel. Arra hívja fel a rutin felhasználójának figyelmét, hogy az eredménnyel óvatosan kell bánnia, hiszen ha a kiterjesztett jelkészletet használja, azaz a stringekben félgrafikus karakterek is szerepelnek, akkor a flagekben visszaadott válasz előjeles vizsgálatokkal való tesztelése hamis eredményre vezet.

XV.2. Kezeljünk kritikus hibát

Következő feladatunk bonyolultabb: kritikus hibát akarunk kezelni. Kritikus hibát elég könnyű előállítani: kiadunk bármilyen file-kezelési műveletet az A: lemezegységre irányítva, de nem teszünk bele lemezt, vagy nem csukjuk le a lemezegység ajtaját. Olyan kritikus lesz a hiba, hogy az csak na! A 24 interrupt villámként csap be; más dolgunk nincs, mint hogy valamilyen "villámhárítóval" levezessük a rendszer dühét.

Komolyra fordítva a szót: lehetnek esetek, amikor nem engedhetjük meg magunknak azt a luxust, hogy a programunk kritikus hiba esetén "kardcsapás nélkül kapituláljon", azaz engedje magát kilőni anélkül, hogy megkísérelné elhárítani vagy legalábbis mérsékelni a kritikus hiba okozta károkat. Például egy jól fejlett szövegszerkesztő program nem hagyja abortálni magát akkor, ha nem tud menteni az A: lemezre, hiszen ez azzal járna, hogy a felhasználó munkája kárbavész. Ehelyett (az editált szöveg megőrzésével) addig kell gyötörnie a felhasználót különféle hibaüzenetekkel, amíg be nem rak végre egy lemezt a floppy-egységbe, vagy el nem határozza, hogy a merevlemezre ment, vagy bármi egyebet tesz, ami mégis megőrzi a munka eredményét.

A mi példaprogramunk el szeretné olvasni a diskette lemezcimkéjét, és akkor kell kritikus hibát kezelnie, ha a felhasználó nem tesz be lemezt. A kritikus hiba-kezelő rutin lekérdezi a kiterjesztett hibakódot, és megnézi, hogy le tudja-e kezelni a hibát. Ha igen (azaz az ERR_DRNRDY hibaüzenetet kapta meg,

amely szerint a lemezegység nem üzemkész és a felhasználó tud segíteni a problémán), kiír egy üzenetet, hogy

Kellene A:-ba egy lemez!

és meghívja a DOS kritikus hiba promptját kiíró rutint, felkínálva a választás lehetőségét. A vezérlést a hibát kiváltó INT 21 utáni utasításra adja vissza, de különleges módon. Ha a felhasználó észbekap (betesz egy lemezt), és azt feleli, hogy RETRY, akkor a Carry-ben 1-et ad ugyan vissza, de a hibakód 0, ami sikeres végrehajtásra utal. Ezzel jelzi, hogy nincs nagy baj, csak meg kell ismételni a műveletet.

Ha a felhasználó IGNORE-t válaszol (ez persze nem megengedett), akkor Carry=1 mellett a hibakód -1 (ami nem esik egybe egyetlen, a rendszer által visszaadott hibakóddal sem). A főprogram ebből tudja, hogy a felhasználó ismét hibázott, tehát nem szükséges a művelet megismétlése; kiírja, hogy az IGNORE nem megengedett, és kilép. Végül, ha a kritikus hiba nem a várt okból következett be, hanem akármi más miatt (pl. valami hardware-hiba), vagy a felhasználó ABORT-al válaszolt, akkor ugyancsak Carry=1 mellett a rendes hibakódot adja vissza a főprogramnak, amely ezt a hibakódot a szokásos módon értelmezi, a hozzá tartozó szöveget kiírja, és kilép.

```
TITLE   Kritikus hiba-kezelő rutin
        EXTRN   PRTERRMSG:NEAR
;
; Konstansok, makrók, rekordok
;
        .XLIST                  ;DOSCALL.INC-et nem listázzuk!
        INCLUDE         DOSCALL.INC
        .LIST
;--------------------------------;
; Adatterületek
;--------------------------------;

DATA    SEGMENT PARA    PUBLIC  'DATA'  ;
ALLFILES        DB      'A:*.*', 0      ;Ezt keressük majd
                                        ;-------;
MSGDEF  MSG_CLSDSK, 'Kellene A:-ba egy lemez!'  ;Kritikus h.üz.
MSGDEF  MSG_VOLAB,  'A lemez cimkéje', NOCR, NODOLLAR
MSGDEF  MSG_NOLAB,  'A lemeznek nincs cimkéje', , NODOLLAR
MSGDEF  MSG_NOIGN,  'Az IGNORE nem megengedett!', , NODOLLAR
MSGDEF  MSG_CRLF                        ;
```

```
                ;---------------------------------;
                ; Munkaterületek
                ;---------------;
FOUND   FDENT   <>              ;Keresési buffer
                                ;
DATA    ENDS                    ;
;---------------------------------;
; Stack-terület
;---------------------------------;

STACK   SEGMENT PARA    STACK   'STACK'
        DW      100     DUP ( ? )       ;Hossza 100 szó
STACK   ENDS

;---------------------------------;
; Kódszegmens
;-----------------------------------------;

CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:DATA, SS:STACK, ES:NOTHING

        PSP_SEG         DW      0       ;
START:                                  ;
                MOV     CS:PSP_SEG, DS  ;
                MOV     AX, DATA        ;
                MOV     DS, AX          ;
                MOV     ES, AX          ;
                CALL    SETIT24         ;24 int. vektor elők.
                MOV     DX, OFFSET FOUND;
                DOSCALL DOS_SETDTA      ;DTA a kereséshez
LOOP:                                   ;
                MOV     CX, ATR_VOLAB + ATR_ARCHV
                MOV     DX, OFFSET ALLFILES
                DOSCALL DOS_FDFRST      ;Keresünk, hiba lesz
                JNC     SUCC_QUIT       ;Sikerült, cimke ki
                CMP     AL, ERR_NOMFIL  ;Várt hibakód?
                JNE     CRIT_ERR_TST    ;Nem, lehet kritikus is
                OUTWRIT MSG_NOLAB       ;Igen, nincs cimke
                JMP     SUCC_EXIT       ;
CRIT_ERR_TST:                           ;
                TEST    AL, AL          ;Kritikus hiba volt?
                JZ      LOOP            ;0 - ismétlés
```

```
                JNS     CRIT_QUIT       ;Lehet, kilépni
                OUTWRIT MSG_NOIGN       ;IGNORE, figyelmezt. ki
CRIT_QUIT:                              ;
                CALL    RESETIT24       ;24 vektor helyeráll.
                JMP     SUCC_EXIT       ;ABORT vagy IGNORE
SUCC_QUIT:                              ;
                OUTWRIT MSG_VOLAB       ;Cimke megvan, kiirjuk
                OUTWRIT FOUND.FDENT_NAME, SIZ_FILSPEC
                OUTWRIT MSG_CRLF        ;
                CALL    RESETIT24       ;24 vektor helyeráll.
                JMP     SUCC_EXIT       ;Kilépés
                EXIT    NOERR           ;Nincs üzenet,
                                        ; OWN_IT24 kiirta!
;---------------------------------------;
; SETIT24       - Kritikus hibakezelő előkészitése
;       Lásd a SETIT23 rutint más konstanssal (*) helyen
; RESETIT24     - Kritikus hibakezelő helyreállítása
;       Lásd a RESETIT23 rutint más konstanssal (*) helyen
;---------------------------------------;
; OWN_INT24     - saját kritikus hibakezelő rutin
;-----------------------------------------------;
OWN_INT24       PROC    FAR                     ;
                MOV     CS:SAVE_DEVHND.OFFS, BP ;Megkapott
                MOV     CS:SAVE_DEVHND.SEGM, SI ; információ
                MOV     CS:SAVE_ERRCND, AX      ; elmentése
                MOV     CS:SAVE_ERRCOD, DI      ;
                                        ;-------;
                DOSCALL DOS_GTEXTERR    ;Kiterj. hibakód lek.
                MOV     CS:SAVE_AX, AX  ;Hibakód mentése
                CALL    PRTERRMSG       ;Hibajelenség kiirása
                MOV     AX, CS:SAVE_AX  ;Hibakód visszatölt.
                CMP     AL, ERR_DRNRDY  ;A lemez nem üzemkész?
                JNE     OWN_INT24_ABORT ;Nem, abortáljuk
                                        ; a programot
                PUSH    AX              ;
                PUSH    DS              ;
                MOV     AX, DATA        ;
                MOV     DS, AX          ;DS regiszter előkész.
        ASSUME  DS:DATA                 ;
                PRTSTR  MSG_CLSDSK      ;Saját hibaüz. ki
                POP     DS              ;Regiszterek vissza
                POP     AX              ;-------;
```

```
                MOV     BP, CS:SAVE_DEVHND.OFFS ;Mentett inf.
                MOV     SI, CS:SAVE_DEVHND.SEGM ; helyreáll.
                MOV     AX, CS:SAVE_ERRCND      ; DOS-rutin
                MOV     DI, CS:SAVE_ERRCOD      ; számára
                                                ;-------;
                PUSHF                           ;IRET lesz! prompt ki,
                CALL    CS:SAV_INT24            ; válasz lekérdezése
                                                ;
                MOV     BP, SP                  ;Stack-címzés előkészí-
                                                ; tése
                CMP     AL, CE_RETRY            ;RETRY-t válaszolt?
                JE      OWN_INT24_RETRY         ; igen, vissza
                                                ;
                CMP     AL, CE_IGNORE           ;IGNORE-t válaszolt?
                JE      OWN_INT24_IGNORE        ; igen, abortáljuk
                                                ;
                JMP     OWN_INT24_ABORT         ;Más, abortáljuk
                                                ;
OWN_INT24_RETRY:                                ;Rendben, folytatjuk;
                MOV     CE_AX.[ BP ], 0         ; hibakód 0!
                JMP     OWN_INT24_QUIT          ;
OWN_INT24_IGNORE:                               ;Illegális válasz,
                MOV     CE_AX.[ BP ], -1        ; hibakód -1!
                JMP     OWN_INT24_QUIT          ;
OWN_INT24_ABORT:                                ;Nincs folytatás,
                MOV     AX, CS:SAVE_AX          ;
                MOV     CE_AX.[ BP ], AX        ; rendes hibakód
OWN_INT24_QUIT:                                 ;-------;
                OR      CE_FLP.[ BP ], FL_CARRY ;Carry=1
                ADD     SP, CE_RETADDR_SIZ      ;Címek ki
                                                ;-------;
                POP     AX                      ;
                POP     BX                      ;Az összes
                POP     CX                      ;
                POP     DX                      ; elmentett
                POP     SI                      ;
                POP     DI                      ; regiszter
                POP     BP                      ;
                POP     DS                      ; helyreállítása
                POP     ES                      ;
                                                ;
                IRET                            ;Vissza az INT 21-hez
```

```
;-----------------------------------------;
; Kódterületen elhelyezendő fontos változók
;-----------------------------------------;
                                          ;
SAV_INT24       LONGPNT <>                ;Mentett rutincím
OWN_INT24_ADR   DD      OWN_INT24         ;Saját rutincím
                                          ;
SAVE_AX         DW      0                 ;AX mentésére  és
SAVE_DEVHND     LONGPNT <>                ; a válaszinformáció
SAVE_ERRCOD     DW      0                 ; mentésére szolgáló
SAVE_ERRCND     DW      0                 ; területek
                                          ;
OWN_INT24       ENDP                      ;
                                          ;
CODE    ENDS                              ;
        END     START
```

Figyeljük meg azt, hogy a hibakezelő rutin első dolga a kapott információk mentése volt. Figyelem! Ezeket most nem a veremre helyezte el, mert az most nagyon fontos információkat tartalmaz! A mentés után kiírja a kiterjesztett hibakódhoz tartozó szokásos hibaüzenetet (PRTERRMSG rutin). Megvizsgálja, hogy a várt hibajelenség következett-e be (minden hibát nehezen kezelhetünk le programból), és ha nem, akkor rögtön a visszatérési szakaszra lép. Ha tudja kezelni a hibát, akkor kiírja saját üzenetét (csakis standard file-kezelő műveleteket használva!), és visszaállítja a belépéskor kapott információkat a megfelelő regiszterekbe. Ezután hívhatja meg a rendszer kritikus hibakezelő rutinját elég ravasz módon, végül értelmezi a kapott választ, és ennek megfelelő adatokat tölt a regiszterek stacken levő másolatába. Itt egy struktúra ügyes használatát és a vermen levő STATUS módosítását érdemes megfigyelni. Végül, közös pályafutásunk első IRET utasításával, visszatér a hívóhoz.

XV.3. Amit a megosztott file-elérésről tudhatunk

Az alábbi programpár "csak" annyit tesz, hogy minden lehetséges eléréssel és minden megosztási módon megnyit egy file-t, kiírva, hogy sikeres volt-e a megnyitás vagy nem. Ha nem, akkor mond valamit a hiba okáról is. Amennyiben az öröklődési bitet

nem keverjük bele a dologba, akkor ez összesen 15x15, azaz 225 esetet jelent. A programok azonban néhány ügyes makródeklaráció felhasználásával nem is mondhatók hosszúnak. Lefuttatásukkal ellenőrizhetjük az "MS-DOS file-kezelés" fejezetben megadott táblázatot.

A programok nem túlságosan bonyolultak (eltekintve a file megnyitását végző, kissé trükkös makróktól). A főprogram megnyitja a file-t valamely elérési mód és megosztási mód kombinációjával, majd meghívja az alprogramot, amely az egyszer már megnyitott file-t újra megnyitja minden lehetséges kombinációban, és a művelet sikerességéről vagy kudarcáról üzenetet küld. Egyetlen nehézséggel kell megküzdenünk, amely azonban igazi problémát nem okozhat. Ez abban áll, hogy a második megnyitás bizonyos esetekben kritikus hibát okoz, tehát a teljes megoldás egy kritikus hibakezelő rutin megírását igényli. Ha ezt nem tesszük meg, akkor az elindított alprogram nem fog végig lefutni, hiszen az első kritikus hibánál kilép. Ekkor pedig nem próbáltunk ki minden egyes lehetőséget.

A nyitási kombinációk sorrendje mindkét program esetében a következő: olvasásra nyitunk "kompatibilis", "írásra-olvasásra tiltott", "írásra tiltott", "olvasásra tiltott" és végül "semmi sem tiltott" megosztási módban. Ugyanezt megismételjük írásra, majd írásra-olvasásra nyitott file-al. Az alprogram minden hívásakor ugyanezt a processzust játssza végig, ebből adódik a 225 kombináció.

```
TITLE   Megosztott file-elérés, főprogram

PAGE    60,132

COMMENT *

        A file neve:            SH-1.ASM
        Szerző:                 Pethő Ádám
        Dátum:                  1987. júl. 17.
        Módosítások:
                1987. aug. 23.
                (Magyar nyelvű megjegyzések, Pethő Ádám)
```

Ez a program a file-ok megosztott elérését illusztráló programpár főprogramja. Fordítása és szerkesztése a szokásos módon lehetséges.

Futtatása csak 3.00 és későbbi MS-DOS verziók alatt megengedett. A futtatás előtt a SHARE drivert aktivizálni kell! A futtatás során a kurrens directory-ban kell lennie a programpár alprogramjának, valamint a közösen felhasznált file-nak

SH-2.EXE, illetve
COMMON.TXT

néven.*

```
        EXTRN   PRTERRMSG:NEAR, CUT_EXE:NEAR
;-----------------------------------------;
; Konstansok és makrók
;-----------------------------------------;
        .XLIST                  ;DOSCALL.INC-et nem listázzuk!
        INCLUDE         DOSCALL.INC
        .LIST
;-----------------------------------------;
; File-megnyitási makró
;-----------------------------------------;
OPEN    MACRO   MODE, ACCESS
                MOV     AL, DOS_OPEN_&MODE      ;;Elérési mód
                OR      AL, DOS_OPEN_&ACCESS    ;;Megoszt. mód
                MOV     DX, OFFSET COMFILE      ;;File-név címe
                DOSCALL DOS_OPEN                ;;
                LCJ     C, ERR_EXIT             ;;Lehetetlen !
                MOV     COM_HND, AX             ;;Handle ment.
                OUTWRIT MSG_OPEN                ;;Inf. kiírása:
                OUTWRIT MSG_&MODE               ;; elérés és
                OUTWRIT MSG_&ACCESS             ;; megosztás
        ENDM

;-----------------------------------------;
; Megnyitások sorozata adott elérési mód mellett minden
; megosztási módon
;-----------------------------------------;

OPEN_SEQ        MACRO   MODE
        IRP     ACCESS, <COMP, DNRW, DNWRIT, DNREAD, DNNONE>
                OPEN    MODE, ACCESS    ;;Menyitás adott módon
                CALL    EXEC_SH2        ;;Alprogram meghivása
                MOV     BX, COM_HND     ;;
                DOSCALL DOS_CLOSE       ;;File lezárása
        ENDM                    ;;Ismétlés minden megosztással
        ENDM                    ;;
```

```
;-----------------------------------;
; Adatterületek
;-----------------------------------------;

DATA    SEGMENT PARA    PUBLIC  'DATA'

PATH    DB      'SH-2.EXE', 0           ;Alprogram-specifikáció

EXEC_BLK        EX_BLK  <>              ;Végrehajtási blokk

COMFILE DB      'COMMON.TXT', 0         ;Közös file-spec.
COM_HND DW      0                       ;Terület a handle-nak

MSGDEF  MSG_LOGIN,    'Hahó! Itt a parent program!', , NODOLLAR
MSGDEF  MSG_OPEN,     'COMMON.TXT megnyitva', NOCR, NODOLLAR
MSGDEF  MSG_READ,     ' olvasásra,',          NOCR, NODOLLAR
MSGDEF  MSG_WRIT,     ' írásra,',             NOCR, NODOLLAR
MSGDEF  MSG_RW,       ' írásra/olvasásra,',   NOCR, NODOLLAR

MSGDEF  MSG_COMP,     ' kompatibilis módban', , NODOLLAR
MSGDEF  MSG_DNRW,     ' írás/olvasás tiltva', , NODOLLAR
MSGDEF  MSG_DNWRIT,   ' írás tiltva', , NODOLLAR
MSGDEF  MSG_DNREAD,   ' olvasás tiltva', , NODOLLAR
MSGDEF  MSG_DNNONE,   ' semmi sincs  tiltva', , NODOLLAR

MSGDEF  MSG_LOG1,     'SH-2  után a parent program!', , NODOLLAR
MSGDEF  MSG_EXECERR, 'SH-2 végrehajtása sikertelen', , NODOLLAR

MSG_RET         DB    'Válaszkód: '
MSG_RET_STATUS  DB    0
                DB    ', Kilépési kód: '
MSG_RET_TRMCD   DB    0, CHR_CR, CHR_LF
        MSG_RET_SIZ   EQU       $ - MSG_RET

DATA    ENDS

;-----------------------------------------;
; Stack-terület
;-----------------------------------------;
STACK   SEGMENT PARA    STACK   'STACK'
        DW      0100H   DUP     (?)     ;Hossza 256 szó
STACK   ENDS                            ;
```

```
;-----------------------------------------;
; Kódterület
;-----------------------------------------;

CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE,DS:DATA,SS:STACK,ES:NOTHING

START:                                  ;
                MOV     CS:PSP_SEG, DS  ;PSP-cím mentése
                                        ;
                MOV     AX, DATA        ;
                MOV     DS, AX          ;DS előkészítése
                                        ;
                CALL    CUT_EXE         ;Memória felszabadítása
                LCJ     C,ERR_EXIT      ;(Lehetetlen esemény)
                OUTWRIT MSG_LOGIN       ;Bejelentkező üzenet ki
                                        ;-------;
                OPEN_SEQ        READ            ;Ennyi
                OPEN_SEQ        WRIT            ; maga a
                OPEN_SEQ        RW              ; program!
                                        ;-------;
                JMP     SUCC_EXIT       ;Sikeres kilépés
                EXIT                    ;
PSP_SEG         DW      0               ;
SAVE_SS         DW      0               ;
SAVE_SP         DW      0               ;
                                        ;
;-----------------------------------------;
; SH-2.EXE végrehajtása
;-----------------------------------------;
                                        ;
EXEC_SH2        PROC    NEAR            ;
                                        ;
                MOV     CS:SAVE_SS, SS  ;Stack Segment mentése
                MOV     CS:SAVE_SP, SP  ;Stack Pointer mentése
                MOV     DX, OFFSET PATH ;---;Programfile-név
                MOV     BX, OFFSET EXEC_BLK ;Paraméter-tábla
                PUSH    DS              ;---;
                POP     ES              ;ES:BX címez!
                DOSCALL DOS_EXEC, DOS_EXEC_PRC
                MOV     SS, CS:SAVE_SS  ;Stack Segment és
                MOV     SP, CS:SAVE_SP  ;Stack Pointer vissza
```

```
                LCJ     C, EXEC_SH2_ERR_EXIT    ;Sikertelen!
                DOSCALL DOS_GTRETCD     ;Válaszkód lekérdezése,
                ADD     AH, '0'         ; előkészítése és
                MOV     MSG_RET_STATUS, AH
                ADD     AL, '0'         ; kiírása
                MOV     MSG_RET_TRMCD,  AL
                OUTWRIT MSG_LOG1        ;
                OUTWRIT MSG_RET, MSG_RET_SIZ
                JMP     EXEC_SH2_QUIT   ;Sikeres végrehajtás
                                        ;
EXEC_SH2_ERR_EXIT:                      ;Sikertelen program-
                OUTWRIT MSG_EXECERR     ; végrehajtás, üzenet
                CALL    PRTERRMSG       ; és hibakód feldolg.
EXEC_SH2_QUIT:                          ;
                RET                     ;
                                        ;
EXEC_SH2        ENDP                    ;
                                        ;
CODE    ENDS                            ;
                                        ;
        END     START
```

Néhány apró megjegyzés: figyeljük meg, hogy az összetett makródeklarációk felhasználásával maga a program mindössze három sor hosszúságú lett. Minden algoritmikus "nehézségnél" fáradságosabb volt a viszonylag esztétikus üzenetkiírás megvalósítása!

A makrókat azért deklaráltuk éppen így, mert az OPEN makró jól használható egészen eltérő esetekben is; az OPEN_SEQ makró fő célja az, hogy írásmunkától szabadítson meg bennünket. Azért nem használtunk egy újabb IRP blokkot e makró hívására, hogy a program olvasója jobban lássa, mi történik: egy adott elérési mód mellett minden megosztási módot kipróbálunk. Ez rontja a ugyan program szimmetriáját, de talán egy picit olvashatóbbá teszi. Valószínűleg ez a program legtámadhatóbb pontja.

Végül a programot bevezető COMMENT-re szeretnénk felhívni a figyelmet. Minden egyes programfile-nak tartalmaznia kell egy ilyet, amelyben legalább ennyi információ szerepel a file-ról: a file neve, a szerző (ki tartja a hátát a hibákért), a keletkezés és a módosítások ideje, a módosítások pár szavas leírása, és a módosításokat végző személy neve, a fordítás és szerkesztés módja, a futtatáshoz szükséges tudnivalók.

```
TITLE   Megosztott file-elérés, alprogram
PAGE    60,132
COMMENT *
        Ez a file:              SH-2.ASM
        Szerző:                 Pethő Ádám
        Dátum:                  1987. júl. 17.
        Módosítások:
                1987. aug. 23.
                (Magyar nyelvű megjegyzések, Pethő Ádám)

Ez  a program a megosztott file-kezelést illusztráló programpár
alprogramja.  Fordítása  és  szerkesztése a szokásos. Futtatása
csak  3.00 és későbbi MS-DOS verziók alatt megengedett. Elindí-
tása felhasználói beavatkozás nélkül, az
        SH-1.EXE
programból történik.
*
        EXTRN   PRTERRMSG:NEAR, CUT_EXE:NEAR

;---------------------------------------;
; Konstansok és makrók
; .
        .XLIST                  ;DOSCALL.INC-et nem listázzuk!
        INCLUDE         DOSCALL.INC
        .LIST
;
; Megnyitási makró
;---------------------------------------;

OPEN    MACRO   MODE, ACCESS
        LOCAL   SUCC, OPEXIT, UNS_CRIT
                OUTWRIT MSG_OPEN        ;;Megnyitás módjának
                OUTWRIT MSG_&MODE       ;; és az elérési módnak
                OUTWRIT MSG_&ACCESS     ;; kiírása
                MOV     AL, DOS_OPEN_&MODE
                ADD     AL, DOS_OPEN_&ACCESS
                MOV     DX, OFFSET COMFILE
                DOSCALL DOS_OPEN        ;;File-megnyitás
                JNC     SUCC            ;;Sikeres művelet
                TEST    AL, AL
                JS      UNS_CRIT        ;;Kritikus hiba volt
                CALL    PRTERRMSG       ;;Normál hiba volt
```

```
UNS_CRIT:
                MOV     COM_HND, -1     ;;Nemlétező handle
                JMP     OPEXIT
SUCC:
                MOV     COM_HND, AX     ;;Handle mentése
                OUTWRIT MSG_SUCC        ;;Siker-üzenet ki
OPEXIT:
        ENDM

;----------------------------------------;
; File-lezárás - makró, hogy ellenőrizzük, létezik-e file
;----------------------------------------;

CLOSE   MACRO
        LOCAL   NOFILE
                MOV     BX, COM_HND     ;;Handle BX-be
                TEST    BX, BX          ;;Létezik?
                JS      NOFILE          ;;Negatív, nem létezik
                DOSCALL DOS_CLOSE
NOFILE:
        ENDM

;----------------------------------------;
; Sorozatos file-nyitás minden megosztási móddal
;----------------------------------------;

OPEN_SEQ        MACRO   MODE
        IRP     ACCCESS, <COMP, DNRW, DNWRIT, DNREAD, DNNONE>
                OPEN    MODE, ACCESS
                CLOSE
        ENDM            ;;Ismétlés minden megosztási móddal

        ENDM

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

DATA    SEGMENT PARA    PUBLIC  'DATA'

COMFILE DB      'COMMON.TXT', 0         ;Közös file-spec. és
COM_HND DW      0                       ; file-handle
```

```
                ;-------------------------;
                ; Kiírandó üzenetek
                ;-------------------------;
MSGDEF  MSG_LOGIN, '    Haho! Itt a child program!', , NODOLLAR
MSGDEF  MSG_OPEN,  '    COMMON.TXT megnyitása', NOCR, NODOLLAR
MSGDEF  MSG_READ,  ' olvasásra,',          NOCR, NODOLLAR
MSGDEF  MSG_WRIT,  ' írásra,',             NOCR, NODOLLAR
MSGDEF  MSG_RW,    ' írásra/olvasásra,',   NOCR, NODOLLAR

MSGDEF  MSG_COMP,    ' kompatibilis módban', , NODOLLAR
MSGDEF  MSG_DNRW,    ' írás/olvasás tiltva', , NODOLLAR
MSGDEF  MSG_DNWRIT,  ' írás tiltva', , NODOLLAR
MSGDEF  MSG_DNREAD,  ' olvasás tiltva', , NODOLLAR
MSGDEF  MSG_DNNONE,  ' semmi sincs tiltva', , NODOLLAR

MSGDEF  MSG_SUCC,    '00 - sikeres megnyitás', , NODOLLAR

DATA    ENDS

;-----------------------------------------;
; Stack-terület
;-----------------------------------------;

STACK   SEGMENT PARA    STACK   'STACK'
        DW      0100H   DUP     (?)     ;Hossza 256 szó
STACK   ENDS

;-----------------------------------------;
; Kódterület
;-----------------------------------------;

CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE,DS:DATA,SS:STACK,ES:NOTHING

START:                                  ;
                MOV     CS:PSP_SEG, DS  ;PSP-cim mentése
                MOV     AX, DATA        ;
                MOV     DS, AX          ;DS előkészitése
                                        ;
                CALL    CUT_EXE         ;Memória felszabadítása
                LCJ     C,ERR_EXIT      ;(Lehetetlen esemény)
                                        ;
```

```
                CALL    SETIT24         ;Kritikus hiba fogadása
                                        ;
                OUTWRIT MSG_LOGIN       ;Bejelentkezési üzenet
                                        ;-------;
                OPEN_SEQ        READ            ;Ennyi
                OPEN_SEQ        WRIT            ; maga a
                OPEN_SEQ        RW              ; program
                                        ;-------;
                CALL    RESETIT24       ;24 interrupt helyreáll
                                        ;
                JMP     SUCC_EXIT       ;
                EXIT                    ;
                                        ;
PSP_SEG         DW      0               ;
                                        ;
;---------------------------------------;
; SETIT24       - Kritikus hibakezelő előkészítése
;       (lásd az "Egy kilőhetetlen program"
;        példa egy módosítását)
;
; RESETIT24     - Kritikus hibakezelő helyreállítása
;       (lásd az "Egy kilőhetetlen program"
;        példa egy módosítását)
;
; Kritikus hiba - csak egy üzenetet ír ki
;---------------------------------------;
                                        ;
OWN_INT24       PROC    FAR             ;
                                        ;-------;
                MOV     CS:SAVE_DEVHND.OFFS, BP ;Megkapott
                MOV     CS:SAVE_DEVHND.SEGM, SI ; információ
                MOV     CS:SAVE_ERRCND, AX      ; mentése
                MOV     CS:SAVE_ERRCOD, DI      ;
                                        ;-------;
                DOSCALL DOS_GTEXTERR    ;Kiterj. hibakód lek.
                MOV     CS:SAVE_AX, AX  ;Hibakód mentése
                CALL    PRTERRMSG       ;Hibajelenség kiírása
                                        ;
                MOV     BP, SP          ;Stack-címzés elők.
                OR      CE_FLP.[ BP ], FL_CARRY ;Carry=1
                MOV     CE_AX.[ BP ], -1        ;Hibakód AX-be
                ADD     SP, CE_RETADDR_SIZ      ;Címek ki
```

```
                POP     AX              ;-------;
                POP     BX              ;Az összes
                POP     CX              ;
                POP     DX              ; elmentett
                POP     SI              ;
                POP     DI              ; regiszter
                POP     BP              ;
                POP     DS              ; helyreállítása
                POP     ES              ;
                                        ;
                IRET                    ;Vissza az INT 21 mögé
                                        ;
SAVE_BP         DW      0               ;
SAVE_AX         DW      0               ;Munkaterületek
                                        ;
SAVE_DEVHND     LONGPNT <>              ;Válaszinformáció
SAVE_ERRCOD     DW      0               ; mentésére szolgáló
SAVE_ERRCND     DW      0               ; területek
                                        ;
OWN_INT24       ENDP                    ;
                                        ;
CODE    ENDS                            ;
                                        ;
        END     START
```

A megfigyelésre érdemes dolgok itt: az egyszerű kritikus hiba kezelő rutin, amely megakadályozza az alprogram kilövését, a főprogramhoz nagyon hasonló, de attól mégis eltérő makrók deklarálása és használata.

A fenti programpár egy roppant fontos melléktermékkel is szolgál. Ez pedig a következő: futtassuk le a programokat minden trükk nélkül! Ekkor az összes kiírás a standard outputra, azaz a képernyőre irányul. Ezzel viszonylag keveset tudunk kezdeni, hiszen 225 megnyitási kísérlet eredményét látjuk végigrohanni a képernyőn. Nosza, irányítsuk át az SH-1 standard outputját a nyomtatóra!

```
C>SH-1 >PRN:
```

Ekkor persze átirányítottuk a parent program (SH-1) kiírásait, de mi fog történni a child program (SH-2) kiírásaival? Magyarul: függetlenek-e a parent és a child program standard file-

jai, vagy sem; öröklődnek-e, vagy nem? A programpár futása közben minden egyes kiírást a nyomtatóra végez, tehát az egymásból indított programok standard file-jai nem függetlenek, hanem öröklődnek.

Általában elmondható, hogy a file-sorszám egy adott file-hoz kötődik, nem pedig egy adott programnak egy adott file-ját jelenti. Ez természetes, hiszen az MS-DOS operációs rendszer szigorúan egyfelhasználós, és egy program futtatására képes. Egy program persze elindíthat egy másik programot, de attól fogva a vezérlés addig van az elindított program utasításain, amíg az a "maga jószántából" vissza nem adja; az egyidejűleg a memóriában levő programok korántsem függetlenek egymástól, tehát nem független a környezetük sem. Ez filozófiai, nem pedig programozástechnikai kérdés; nem tehetünk mást, mint hogy elfogadjuk.

XV.4. A floppy-disk directory-ja

Ez a példa a 25 interrupt használatát illusztrálja. A kitűzött cél egy tetszőleges típusú floppy-disk directory-jának elolvasása.

A feladat maga roppant egyszerű: elolvassuk a lemez FAT-ját, abból előállítjuk a FAT-azonosítót, hogy megtudjuk: hányadik szektorral kell kezdeni az olvasást, és hány szektort kell ahhoz elolvasni, hogy a directory-területet teljes egészében megkapjuk. Beolvassuk a directory-t, és igen egyszerűen listázzuk ki. Nem vacakolunk a file-név és -típus elválasztásával, az egyes elemek értelmezésével és konverziójával, hanem egyszerűen kiírjuk minden nem üres bejegyzés első 11 karakterét.

Eddig az elképzelés. Hanem hát ember tervez, MS-DOS végez; a program (természetesen) nem működik. Ahhoz, hogy kifogástalan legyen, valamilyen értelemben "alapállapotba" kellene hozni a drivert, ez azonban nem dokumentált lépés, az MS-DOS egyik nem dokumentált belügye. A nehézség abban áll, hogy a driver megjegyzi, hogy milyen típusú lemez volt először a lemezegységben, és aszerint folytatja a munkát akkor is, ha közben a lemezt kicseréltük. Kell lennie valami trükknek, amelynek segítségével a lemezcserét a driver tudomására lehet hozni, azaz arra késztetni a drivert, hogy rendezze át belső táblázatait a lemez típusának megfelelően. Azonban a dokumentált rendszerfunkciók között ilyen nincsen (nem jó a 0D [DOS_DSKRES] funkció sem).

Ahelyett, hogy mindenféle ködös, dokumentálatlan tájra tévednénk, a 25 interrupt illusztrálása (és a fő directory felolvasása) kedvéért harcba vetünk egy eléggé alvilági trükköt: meghívunk egy directory-kezelő DOS-funkciót, hogy magát a rendszert kényszerítsük a belső táblázatok átrendezésére. Egyszerűen keresünk egy (egyébként biztosan nem létező) bejegyzést; ekkor a rendszer kénytelen felolvasni a directory-t, és átrendezi a belső táblázatokat. Elismerjük, hogy ez nem egy szép eljárás, de a programozó sajnos gyakran kényszerül ilyen trükkökre, ha nem tud eleget a rendszerről (vagy a rendszer hiányos, vagy egyenesen hibás). Természetesen az előbbi a jellemző eset; mi is amiatt kényszerülünk ilyen trükkök bevetésére, mert ismereteink nem eléggé bőségesek. Más kérdés, hogy gyakran a gyenge rendszerdokumentáció miatt tudunk keveset; de mégiscsak keveset tudunk.

```
TITLE   A lemez direkt kezelése

        EXTRN   GETPAR:NEAR, PRTERRMSG:NEAR
;---------------------------------------;
; Konstansok, makrók, rekordok
;
        .XLIST                  ;DOSCALL.INC-et nem listázzuk!
        INCLUDE         DOSCALL.INC
        .LIST
;
; Disk-választó konstansok
;---------------------------------------;
                                        ;
DR_A_DIR        EQU     DR_A_RESP       ;
DR_B_DIR        EQU     DR_B_RESP       ;
DR_C_DIR        EQU     DR_C_RESP       ;
                                        ;
;---------------------------------------;
; Disk fizikai konstansok
;---------------------------------------;
                                        ;
SIZ_SECTOR      EQU     512             ;Szektorhossz
NUM_DIRENT_SEC  EQU     SIZ_SECTOR/SIZ_DIRENT   ;Bejegyzések
                                        ;száma szektoronként
VAL_NOUSED      EQU      00H            ;Nem használt bejegyzés
VAL_ERASED      EQU     0E5H            ;Törölt bejegyzés
```

```
;-----------------------------------------;
; FAT-azonosító felolvasására szolgáló konstansok
;-----------------------------------------;
                                          ;
FAT_ID          EQU     1                 ;FAT-terület kezdete
FAT_ID_SZ       EQU     2                 ; és minimális hossza
                                          ;
;-----------------------------------------;
;
; Disktípus leíró konstansok
; Ismert diskette-típusok:
;       D8      - kétoldalas, 40 sávos, sávonként 8 szektor
;       S8      - egyoldalas, 40 sávos, sávonként 8 szektor
;       D9      - kétoldalas, 40 sávos, sávonként 9 szektor
;       S9      - egyoldalas, 40 sávos, sávonként 9 szektor
;       Q9      - kétoldalas, 80 sávos, sávonként 9 szektor
;       QF      - kétoldalas, 80 sávos, sávonként 15 szektor
; A leíró konstanshalmaz elemei:
;       FAT-azonosító
;               FAT kezdőszektora
;               FAT hossza szektorokban
;               Directory kezdő szektora
;               Directory hossza szektorokban
;-----------------------------------------;
                                          ;
DSK_ID_D8       EQU     -1                ;
        D8_FST          EQU     1         ;
        D8_FSZ          EQU     2         ;
        D8_DST          EQU     3         ;
        D8_DSZ          EQU     7         ;
                                          ;
DSK_ID_S8       EQU     -2                ;
        S8_FST          EQU     1         ;
        S8_FSZ          EQU     2         ;
        S8_DST          EQU     3         ;
        S8_DSZ          EQU     4         ;
                                          ;
DSK_ID_D9       EQU     -3                ;
        D9_FST          EQU     1         ;
        D9_FSZ          EQU     4         ;
        D9_DST          EQU     5         ;
        D9_DSZ          EQU     7         ;
```

```
DSK_ID_S9       EQU     -4                      ;
        S9_FST          EQU      1              ;
        S9_FSZ          EQU      4              ;
        S9_DST          EQU      5              ;
        S9_DSZ          EQU      4              ;
                                                ;
DSK_ID_Q9       EQU     -7                      ;
        Q9_FST          EQU      1              ;
        Q9_FSZ          EQU     10              ;
        Q9_DST          EQU     11              ;
        Q9_DSZ          EQU      7              ;
                                                ;
DSK_ID_QF       EQU     -7                      ;
        QF_FST          EQU      1              ;
        QF_FSZ          EQU     14              ;
        QF_DST          EQU     15              ;
        QF_DSZ          EQU     14              ;
                                                ;
;---------------------------------------;
; Disk-leíró blokk struktúrája
;-------------------------------;
                                ;
DSKDESC STRUC                   ;
                                ;
ID      DB      0               ;;Azonosító
FST     DW      0               ;;FAT kezdő szektora
FSZ     DW      0               ;;FAT hossza szektorokban
DST     DW      0               ;;Directory kezdő szektora
DSZ     DW      0               ;;Directory hossza szektorokban
                                ;
DSKDESC ENDS                    ;

        SIZ_DSK_DESC    EQU     TYPE    DSKDESC


;-------------------------------;
; Disk-leíró struktúrát definiáló makró
;---------------------------------------;

DSKDSC  MACRO   NAME, TP
NAME    DSKDESC <DSK_ID_&TP,TP&_FST,TP&_FSZ,TP&_DST,TP&_DSZ>
        ENDM                            ;
```

```
;-----------------------------------------;
; Adatterületek
;-----------------------------------------;

DATA    SEGMENT PARA    PUBLIC  'DATA'

DSK_BLK_AREA    LABEL   BYTE                    ;
        IRP     X, <D8, S8, D9, S9, Q9, QF>
        DSKDSC  X&_TAB, X                       ;Minden típushoz
        ENDM                                    ; egy leíró struktúra
                                                ;
DSK_BLK_NUM     DW      ($-DSK_BLK_AREA)/SIZ_DSK_DESC
                                                ;Leíró struktúrák száma
DSK_DESC_ADDR   DW      0                       ; és a kurrens címe
DSK_CODE        DB      DR_A_DIR                ;Olvasandó lemez kódja

NONEX_PATH  DB  'A:--------.---', 0     ;Nemlétező path

MSGDEF  MSG_FATDSK, 'Fatális lemezolvasási hiba', , NODOLLAR
MSGDEF  MSG_ILLTYP, 'Illegális lemeztípus', , NODOLLAR
MSGDEF  MSG_CRLF, , , NODOLLAR

DSK_BUFFER      DB    20*SIZ_SECTOR     DUP     ( -1 )

DATA    ENDS

;-----------------------------------------;
; Stack-terület
;-----------------------------------------;

STACK   SEGMENT PARA    STACK   'STACK'
        DW      100     DUP ( ? )       ;Hossza 100 szó
STACK   ENDS

;-----------------------------------------;
; Kódszegmens
;-----------------------------------------;

CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  CS:CODE, DS:DATA, SS:STACK, ES:NOTHING

PSP_SEG         DW      0               ;PSP mentése
```

```
START:                                  ;
                MOV     CS:PSP_SEG, DS  ;
                MOV     AX, DATA        ;
                MOV     DS, AX          ;
                                        ;-------;
                MOV     CX, ATR_VOLAB           ;Ez egy le- !!
                MOV     DX, OFFSET NONEX_PATH   ; hetlen    !!
                DOSCALL DOS_FDFRST              ; keresés   !!
                                        ;-------;
                MOV     AL, DSK_CODE    ;
                MOV     DX, FAT_ID      ;Kezdő szektor
                MOV     CX, FAT_ID_SZ   ;Szektorok száma
                MOV     BX, OFFSET DSK_BUFFER
                INT     INT_DSKRD       ;FAT-azonosító beolv.
                JC      DSK_ERROR       ;
                POPF                    ;Lenthagyott STATUS ki
                                        ;-------;
                MOV     AL, DSK_BUFFER[512]     ;FAT-ID AL-be
                MOV     CX, DSK_BLK_NUM         ;Leírók száma
                MOV     BX, OFFSET DSK_BLK_AREA ; és címe
                                        ;
DSK_TYPE_SEARCH:                        ;-------;
                CMP     AL, ID.[ BX ]   ;Keressük őt a blokkban
                JZ      DSK_TYPE_FOUND  ;Megvan a típus
                ADD     BX, SIZ_DSK_DESC;
                LOOP    DSK_TYPE_SEARCH ;Folytatjuk a keresést
                OUTWRIT MSG_ILLTYP      ;Nem találtuk - hiba
                JMP     ERR_EXIT        ;
DSK_TYPE_FOUND:                         ;-------;
                MOV     DSK_DESC_ADDR, BX       ;Cím mentése
                MOV     AL, DSK_CODE    ;-------;Lemezkód
                MOV     DX, DST.[ BX ]  ;Directory-kezdőszektor
                MOV     CX, DSZ.[ BX ]  ;Directory hossza
                MOV     BX, OFFSET DSK_BUFFER
                INT     INT_DSKRD       ;Directory beolvasása
                JC      DSK_ERROR       ;
                POPF                    ;Lenthagyott STATUS (*)
                JMP     DIR_LIST        ;
DSK_ERROR:                              ;
                POPF                    ;Lenthagyott STATUS (*)
                OUTWRIT MSG_FATDSK      ;Hibaüzenet kiírása
                JMP     ERR_EXIT        ;
```

```
DIR_LIST:                                   ;-------;
                MOV     BX, DSK_DESC_ADDR           ;
                MOV     AX, NUM_DIRENT_SEC          ;Bejegyzések
                MUL     DSZ.[ BX ]          ;-------; száma
                MOV     CX, AX              ;Ez a ciklusszámláló
                MOV     BX, OFFSET DSK_BUFFER
DIR_LIST_LOOP:                              ;
                PUSH    CX                  ;
                PUSH    BX                  ;
                CMP     BYTE PTR [ BX ], VAL_NOUSED
                JE      DIR_LIST_CONT       ;Nem használt bejegyzés
                CMP     BYTE PTR [ BX ], VAL_ERASED
                JE      DIR_LIST_CONT       ;Törölt bejegyzés
                WRIT_H  BX, SIZ_FILSPEC, HND_STD_OUT
                OUTWRIT MSG_CRLF            ;Bejegyzés és CR-LF
DIR_LIST_CONT:                              ;
                POP     BX                  ;
                POP     CX                  ;
                ADD     BX, SIZ_DIRENT      ;
                LOOP    DIR_LIST_LOOP       ;Folytatjuk, míg van
                JMP     SUCC_EXIT           ;
                                            ;
                EXIT    NOERR               ;Kilépés hibaüz. nélkül
                                            ;
CODE    ENDS                                ;
        END     START
```

A "!!"-el megjelölt sorok végzik a program bevezetésében említett keresést. Ez tehát az az "alvilági" trükk, amit célunk elérése érdekében kénytelenek voltunk felhasználni. Ezenkívül egyetlen megjegyzés kívánkozik ide: figyeljük meg, hogy milyen kellemetlen dolog, ha a vermen kapunk vissza bármilyen információt. Különösen kínos ez az eset, mikor valójában két STATUS-t kaptunk vissza: egyet a helyén, egyet pedig e vermen. Emiatt kellett a (*)-al megjelölt két POPF (és csak azért ennyi, mert a programunk egyszerű). Ha több ágra bontanánk a programot, akkor még több és még zűrösebb helyreállításra lenne szükség, és jaj nekünk, ha csak egyszer is elhibázzuk!

Mindenesetre érdemes lenne átgondolni, mit tehetünk a program kiteljesítése érdekében. Például kírhatjuk a file attribútumát, hosszát, a lefoglalt és valóban felhasznált terület ará-

nyát. Esetleg érdekes lehet egy térkép a file-ról: mely clustereket foglal le, folytonos-e stb. Igen fontos a keletkezés dátumának és idejének megjelenítése.

Azonban a fentiek megvalósításához jónéhány algoritmikus nehézséget kellene leküzdenünk. A "Directory-kezelés" fejezet egyik példája mutatja, hogy milyen hálátlan (bár nem bonyolult) dolog az attributumot értelmezni. Annál nehezebb a file hosszának kiírása. A file hossza ugyanis egy duplaszóban foglal helyet, és (ha a lemez mérete ezt megengedi) meghaladhatja a 64 kilobyte tízszeresét. Gondoljunk utána, hogy ez a legnagyobb szám, amit tízzel el tudunk úgy osztani, hogy nem keletkezik osztási túlcsordulás. Az ezt meghaladó értékek konverziója bizony trükkösebb módszereket igényel, az egyszerű maradékos osztás nem vezet célra.

Nagyon hasznos volna azonban egy olyan program, amely egy alapos directory-listázó funkciót valósít meg. Nem ritka például az az eset, mikor meg kell tudnunk, hogy a lemez egy adott pontján melyik file foglal helyet, például abban az esetben, ha rendszert akarunk cserélni a merevlemezen, az adott directory- és file-struktúra módosítása nélkül. Amennyiben az új rendszer több szektort igényel a lemez adatterületének elején, mint az előző (márpedig újabb MS-DOS verziók esetén ez a helyzet), akkor törölni kell az IBMBIO.COM és IBMDOS.COM file-ok mögött elhelyezkedő file-t. Ha ez a file az előző COMMAND.COM, akkor nincsen semmi probléma. Ezt töröljük, majd a SYS parancs lefuttatása után újra felmásoljuk. Ekkor persze más helyre fog kerülni, valahová az adatterület mélyére. A következő rendszerfrissítésnél pedig már az a bonyolultabb szituáció áll elő, hogy van valamilyen file a két láthatatlan rendszerfile mögött, de az ördög tudja, hogy mi. Ennek felderítésére alkalmas lehet egy olyan segédprogram, amely gondosan végigjárja a directory-struktúrát, és feltérképezi minden file helyét. A program megvalósítása során számos gyönyörű eszközt lehet harcba vetni: a rekurzív rutinhívás alkalmas a fastruktúra végigjárására, csodás alkalmak nyílnak memória-allokálásra, ravasz rendezésekre és az ezzel járó többszintű pointerezésre. Mindenesetre, ha egy mód van rá, egy ilyen programot százszor inkább C nyelven vagy TurboPascal-ban írjunk meg, mint assemblyben. Az e kötetben részletezett ismeretek segítségével a program technikailag már megvalósítható.

Az interrupt-vektorok kiosztása

Processzor-interruptok

00	Osztási túlcsordulás, zerodivide
01	Lépésenkénti megszakitás, single-step
02	NMI - nem maszkolható interrupt
03	Töréspont, breakpoint
04	Túlcsordulási rutin hívása, overflow

Alap BIOS interruptok

05	Képernyőtartalom nyomtatása (hard copy)
06, 07	Nem használt interruptok

Hardware-interruptok

08	Timer hardware-interruptja ----------	(8529 IRQ0)
09	A klaviatúra hardware-interruptja ---------	(8529 IRQ1)
0A	Nem használt ----------	(8529 IRQ2)
0B	Az aszinkron vonal hardware-interruptja ---	(8529 IRQ3)
0C	Második asz. vonal hardware-interruptja ---	(8529 IRQ4)
0D	A képernyő lefutási interruptja ----------	(8259 IRQ5)
0E	A diskette hardware-interruptja ----------	(8259 IRQ6)
0F	A nyomtató hardware-interruptja ----------	(8259 IRQ7)

BIOS kommunikációs interruptok

10	Képernyő-driver hívása
11	Kiegészitő berendezések listája
12	Memóriahossz ellenőrzése
13	Floppy-disk driver hívása
14	Aszinkron vonali driver hívása
15	Kazetta-driver hívása
16	Klaviatúra-driver hívása
17	Nyomtató-driver hívása
18	ROM BASIC belépési pont
19	Rendszerindítás, bootstrap
1A	A belső óra lekérdezése vagy módosítása
1B	CTRL-BREAK felhasználói rutin címe
1C	Timer felhasználói interrupt-rutin címe

BIOS paramétertáblázatok címei

1D	Video paramétertábla címe
1E	Diskette paramétertábla címe
1F	Video grafikus karaktergenerátor címe

MS-DOS interruptok

20	Kilépés hagyományos módon
21	DOS-funkciók hívása
22	Kilépési cím
23	A CTRL-BREAK kilépési cím (programkilövés)
24	Kilépési cím kritikus DOS-hiba esetén
25	Abszolút disk-olvasás
26	Abszolút disk-írás
27	Kilépés, a program a memóriában marad
28-2E	A DOS belső használatára fenntartott interruptok
2F	Multiplex interrupt, nyomtató (3.00 és későbbi verziók)
30-3F	A DOS belső használatára fenntartott interruptok

Felhasználói és BASIC interruptok

40-7F	Szabad interruptok
80-F0	A BASIC interruptjai
F1-FF	Szabad interruptok

Az interrupt-vektorok kiosztása és funkcióik részletes leírása a hasonnevű fejezetben, sorszám szerint emelkedő sorrendben olvasható. Ezért itt az oldalszámokat külön nem ismertetjük.

A hagyományos és modern DOS funkciók kapcsolata

Ez a függelék a hagyományos és a modern DOS funkciók megfeleltetését tartalmazza. Mint a megfelelő részekben hangsúlyoztuk, az MS-DOS újabb verziói egyre kevésbé támogatják a hagyományos, és egyre inkább a modern DOS-funkciók használatát.

Egyes hagyományos funkciókhoz több modern tartozik, amelyek valamelyikét vagy mindegyikét használhatjuk az adott hagyományos funkció helyett.

Hagyományos DOS-funkció	Modern
00 Terminate Process	4C End Process
0F Open File	3D Open Handle
10 Close File	3E Close Handle
11 Search for First	4E Find First File
12 Search for Next	4F Find Next File
13 Delete File	41 Delete Directory Entry
14 Sequential Read	3F Read Handle
15 Sequential Write	40 Write Handle
16 Create File	3C Create Handle
	5A Create Temporary File (3.00)
	5B Create New File (3.00)
17 Rename File	56 Change Directory Entry
21 Random Read	3F Read Handle
22 Random Write	40 Write Handle
	42 Move File Pointer
23 Get File Size	42 Move File Pointer
24 Set Relative Record	42 Move File Pointer
26 Create New PSP	4B Load and Execute Program
27 Random Block Read	3F Read Handle
28 Random Block Write	40 Write Handle

A DOSCALL.INC közhasznú deklarációs file

Ez a függelék egy olyan include file-t tartalmaz, amelyben megadjuk az összes DOS funkció, valamint a használatukhoz szükséges vagy hasznos konstans-, makró-, struktúra- és rekorddeklarációkat. Elsőként magát a file-t olvashatjuk, ezt követően néhány, a bonyolultabb elemekhez fűzött magyarázó megjegyzést.

```
COMMENT *

        A file neve:              DOSCALL.INC
        Szerző:                   Pethő Ádám
        Keletkezés dátuma:        1987-ápr-27
        Utolsó módosítás:         1987-júl-09
                (magyar nyelvű megjegyzések, Pethő Ádám)

 Ez a file az MS-DOS funkciók hívásához használható
 legfontosabb konstans-, struktúra, rekord- és
 makródefiníciókat tartalmazza

*

;---------------------------------------;
; Karakterkonstansok
;-------------------------------;
                                ;
CHR_CTRLC       EQU      3      ;Control-C
CHR_CTRLZ       EQU     26      ;Control-Z
CHR_ESC         EQU     27      ;ESCAPE
CHR_CR          EQU     13      ;Kocsivissza
CHR_LF          EQU     10      ;Soremelés
CHR_TAB         EQU      9      ;Tabulátor
CHR_FF          EQU     14      ;Lapdobás
CHR_BLANK       EQU     ' '     ;Szóköz
CHR_EOS         EQU      0      ;End Of String, string vége
CHR_DOLLAR      EQU     '$'     ;Dollárjel
CHR_NUMST       EQU     '0'     ;Számok blokkjának kezdete
CHR_NUMEND      EQU     '9'     ;Számok blokkjának vége
CHR_CAPST       EQU     'A'     ;Nagybetűk blokkjának kezdete
CHR_CAPEND      EQU     'Z'     ;Nagybetűk blokkjának vége
CHR_NRMST       EQU     'a'     ;Kisbetűk blokkjának kezdete
CHR_NRMEND      EQU     'z'     ;Kisbetűk blokkjának vége
```

```
;---------------------------------;
; MS-DOS interrupt konstansok
;---------------------------------;
                                  ;
INT_EXIT        EQU     20H       ;Normál kilépés
                                  ;
INT_DOS         EQU     21H       ;A DOS hívása
                                  ;
INT_EXIT_ADDR   EQU     22H       ;Kilépési cím
INT_CTRB_ADDR   EQU     23H       ;CTRL-BREAK kilépési cím
INT_CRER_ADDR   EQU     24H       ;Kritikus hiba-kezelő címe
                                  ;
INT_DSKRD       EQU     25H       ;Direkt disk-olvasás
INT_DSKWR       EQU     26H       ;Direkt disk-írás
                                  ;
INT_STAY        EQU     27H       ;Kilépés bentmaradva
                                  ;
;---------------------------------;
; MS-DOS hívási makró
;-----------------------------------------;
                                          ;
DOSCALL MACRO   FNC_CODE, SUBFNC_CODE     ;
        IFNB    <SUBFNC_CODE>             ;;
                MOV     AL, SUBFNC_CODE   ;;Alfunkció kódja
        ENDIF                             ;;
                MOV     AH, FNC_CODE      ;;Funkciókód
                INT     INT_DOS           ;;DOS-hívás
        ENDM                              ;
                                          ;
;-----------------------------------------;
; Távoli pointer-struktúra
;-----------------------------------------;
                                          ;
LONGPNT STRUC                             ;
        OFFS    DW      0                 ;;A pointer offset-
        SEGM    DW      0                 ;; és szegmensrésze
LONGPNT ENDS                              ;
```

```
;----------------------------------------;
; Standard file-kezelési konstansok
;--------------------------------;
                                 ;
DOS_STRDCHE     EQU     01H      ;Karakter-beolvasás STDIN-ről
DOS_STWRCH      EQU     02H      ;Karakter-kiírás STDOUT-ra
DOS_AUXIN       EQU     03H      ;Karakter-beolvasás STDAUX-ról
DOS_AUXOUT      EQU     04H      ;Karakter-kiírás STDAUX-ra
DOS_PRWRCH      EQU     05H      ;Karakter-kiírás STDPRN-re
DOS_STDIRIO     EQU     06H      ;Direkt input/output
        DOS_STDIRIO_IN  EQU     0FFH    ;Direkt I/O - input
                                 ;
DOS_STDIRIN     EQU     07H      ;Direkt beolvasás
DOS_STRDCH      EQU     08H      ;Beolvasás echo nélkül
DOS_STWRSTR     EQU     09H      ;String-kiírás STDOUT-ra
DOS_STRDSTR     EQU     0AH      ;String-beolvasás STDIN-ről
DOS_STINCHK     EQU     0BH      ;Státusz lekérdezése
DOS_STINFLS     EQU     0CH      ;Buffer törlés
                                 ;
;--------------------------------;
; A standard file-ok sorszáma
;--------------------------------;
                                 ;
HND_STD_IN      EQU     00H      ;Standard input
HND_STD_OUT     EQU     01H      ;Standard output
HND_STD_ERR     EQU     02H      ;Standard hibajelző
HND_STD_AUX     EQU     03H      ;Standard kisegítő eszköz
HND_STD_PRN     EQU     04H      ;Standard nyomtató
                                 ;
;--------------------------------;
; Standard input/output kezelési makrók
;-------------------------------------------------;
                                                   ;
PRTSTR  MACRO   STRING                             ;
                MOV     DX, OFFSET STRING          ;;String címe
                DOSCALL DOS_STWRSTR                ;;
        ENDM                                       ;
```

```
TRCMSG  MACRO   STRING                                  ;
        IFDEF   TRACE                                   ;;Ha kell trace
                PUSH    DX                              ;; mentés,
                PUSH    AX                              ;;
                MOV     DX, OFFSET STRING               ;; string címe,
                DOSCALL DOS_STWRSTR                     ;; kiírás
                POP     AX                              ;; helyre-
                POP     DX                              ;; állítás
        ENDIF                                           ;
        ENDM                                            ;
                                                        ;
;-----------------------------------------------------;
; Szövegdefiniáló makró
;-----------------------------------------------------;
                                                        ;
NOCR            EQU     1                               ;Kocsivezérlő konstans
NODOLLAR        EQU     1                               ;Dollárjel vez.konstans
                                                        ;
MSGDEF  MACRO   NAME, MESSAGE, CRCONT, DOLLAR
        IFB     <MESSAGE>                               ;;Nincs szöveg, CR-LF
NAME    LABEL   BYTE                                    ;;A szöveg neve
                DB      CHR_CR, CHR_LF                  ;;
            IFB     <DOLLAR>                            ;;
                DB      CHR_DOLLAR                      ;;Dollárjel a végére
            ELSE                                        ;;
                DB      CHR_EOS                         ;;Ha nem, akkor CHR_EOS
            ENDIF                                       ;;
        ELSE                                            ;;Van szöveg
NAME            DB      MESSAGE                         ;;
            IFB     <CRCONT>                            ;;
                DB      CHR_CR, CHR_LF                  ;;CR-LF sorozat
            ENDIF                                       ;;
            IFB     <DOLLAR>                            ;;
                DB      CHR_DOLLAR                      ;;Dollárjel a végére
            ELSE                                        ;;
                DB      CHR_EOS                         ;;Ha nem, akkor CHR_EOS
            ENDIF                                       ;;
        ENDIF                                           ;;
SIZ_&NAME       EQU     $ - NAME                        ;;A szöveg hossza
        ENDM                                            ;
                                                        ;
```

```
;-----------------------------------------;
; Klaviatúra-buffert definiáló makró
;-----------------------------------------;
                                          ;
KEYBUF  MACRO   NAME, SIZE                ;
NAME&_BUFFER    LABEL   BYTE              ;;Buffer neve
NAME&_SIZ       DB      SIZE              ;;Buffer hossza
NAME&_CHN       DB      0                 ;;Válasz hossza
NAME&_BUF       DB      SIZE  DUP ( ? )   ;;Buffer
        ENDM                              ;
                                          ;
;-----------------------------------------;
; Ez a fejezet a hagyományos file-kezeléshez szükséges
; legfontosabb konstans-, makró-, struktúra- és rekorddefiní-
; ciókat tartalmazza.
;---------------------------------;
; FCB file-kezelési konstansok    ;
                                  ;
DOS_DSKRES      EQU     0DH       ;Disk-inicializálás
DOS_DSKSEL      EQU     0EH       ;Disk-kiválasztás
DOS_FOPEN       EQU     0FH       ;File-megnyitás
DOS_FCLOSE      EQU     10H       ;File-lezárás
DOS_FSFRST      EQU     11H       ;Első bejegyzés keresése
DOS_FSNEXT      EQU     12H       ;További bejegyzések keresése
DOS_FDELET      EQU     13H       ;File-törlés
DOS_FRDSEQ      EQU     14H       ;Szekvenciális olvasás
DOS_FWRSEQ      EQU     15H       ;Szekvenciális írás
DOS_FCREAT      EQU     16H       ;File-létrehozás
DOS_FRENAME     EQU     17H       ;File-átnevezés
DOS_GETDSK      EQU     19H       ;Kurrens lemez lekérdezése
DOS_SETDTA      EQU     1AH       ;DTA címének beállítása
DOS_GETDFAT     EQU     1BH       ;FAT inf. kurrens lemezről
DOS_GETSFAT     EQU     1CH       ;FAT inf. adott lemezről
DOS_FRDRND      EQU     21H       ;Direkt file-olvasás
DOS_FWRRND      EQU     22H       ;Direkt file-írás
DOS_GETFSIZ     EQU     23H       ;File-hossz lekérdezés (rekord)
DOS_SETRCD      EQU     24H       ;Random rekordszám beállítás
DOS_FRDBRND     EQU     27H       ;Direkt blokkolvasás
DOS_FWRBRND     EQU     28H       ;Direkt blokkírás
```

```
DOS_FNMINI      EQU     29H     ;File-név feldolgozása
        DOS_FNMINI_SEP  EQU     01H     ;Elválasztás átlépése
        DOS_FNMINI_DR   EQU     02H     ;Lemeznév feldolgozása
        DOS_FNMINI_NAM  EQU     04H     ;Név feldolgozása
        DOS_FNMINI_TIP  EQU     08H     ;Típus feldolgozása
                                        ;-------;
DOS_GETDTA      EQU     2FH     ;DTA címének lekérdezése
DOS_GETDFREE    EQU     36H     ;Szabad terület lekérdezése
                                ;
;-------------------------------;
; Hibakódok FCB file-kezeléshez
;-------------------------------;
                                ;
RET_SUCC        EQU      00H    ;Sikeres végrehajtás
                                ;
RET_NOBYTE      EQU      01H    ;File vége, nincs byte
RET_NOSPACE     EQU      02H    ;Nincs hely az átvitelre
RET_FRAGM       EQU      03H    ;Tört rekord beolvasás
                                ;
RET_UNSUCC      EQU     0FFH    ;Sikertelen végrehajtás
                                ;
;-------------------------------;
; Drive kódok
; FCB-ben használatos drive-kódok
;-------------------------------;
                                ;
DR_CURR         EQU     0       ;Kurrens drive
DR_A            EQU     1       ;A: drive
DR_B            EQU     2       ;B: drive
DR_C            EQU     3       ;C: drive
DR_D            EQU     4       ;D: drive
DR_E            EQU     5       ;E: drive
                                ;
;-------------------------------;
; Lekérdezéskor kapott drive-kódok
;-------------------------------;
                                ;
DR_A_RESP       EQU     0       ;A: drive
DR_B_RESP       EQU     1       ;B: drive
DR_C_RESP       EQU     2       ;C: drive
DR_D_RESP       EQU     3       ;D: drive
DR_E_RESP       EQU     4       ;E: drive
```

```
;-----------------------------------;
; FCB file-kezeléshez használt makrók, struktúrák és rekordok
;-------------------------------------------;
                                            ;
FCBCLL  MACRO   CODE, FCB                   ;
                MOV     DX, OFFSET FCB      ;;FCB címe DX-be
                DOSCALL CODE                ;;DOS hívása
                TEST    AL, AL              ;;Válasz tesztelése
        ENDM                                ;
                                            ;
;-------------------------------------------;
;FCB-t definiáló struktúra
;-------------------------------------------;
                                            ;
FCB     STRUC                               ;
        DRIVE   DB      0                   ;;Drive kódja
        NAME    DB      '        '          ;;File-név  .
        EXT     DB      '   '               ;;File-típus
        BLOCK   DW      0                   ;;Blokkszámláló
        RECSIZ  DW      80H                 ;;Rekordhossz
        FILSIZ  DD      0                   ;;File hossza
        DATE    DW      0                   ;;Utolsó írás dátuma
        TIME    DW      0                   ;;Utolsó írás ideje
        RES_FCB DB      8 DUP ( 0 )         ;;Fenntartott terület
        CURREC  DB      0                   ;;Rekordszámláló
        RNDREC  DD      0                   ;;Random rekordszámláló
FCB     ENDS                                ;
                                            ;
;-------------------------------------------;
; FCB-kiterjesztést definiáló struktúra
;-------------------------------------------;
                                            ;
FCB_EXT STRUC                               ;
        EXT_SG  DB      0FFH                ;;Kiterjesztésjelző
        RES_EXT DB      5 DUP ( 0 )         ;;Fenntartott terület
        ATTRIB  DB      0                   ;;Attributum
FCB_EXT ENDS                                ;
                                            ;
```

```
;-----------------------------------------;
; Kiterjesztett FCB-t definiáló makró
;-----------------------------------------;
                                          ;
FCBEXT  MACRO   NAME, FIL_DR, FIL_NM, FIL_EXT, FIL_ATR
NAME&_EXT       FCB_EXT <,,FIL_ATR>     ;;
NAME            FCB     <FIL_DR, FIL_NM, FIL_EXT>
        ENDM                            ;
                                        ;
;-----------------------------------------;
; Directory-bejegyzés belső szerkezete
;-----------------------------------------;
                                        ;
DIRENT  STRUC                           ;
        D_NAME  DB       8 DUP  ( ? )   ;;File-név
        D_TYPE  DB       3 DUP  ( ? )   ;;File-típus
        D_ATTR  DB       0              ;;File-attributum
        D_RES   DB      10 DUP  ( ? )   ;;Fenntartott terület
        D_TIME  DW       0              ;;Utolsó írás ideje
        D_DATE  DW       0              ;;Utolsó írás dátuma
        D_CLUST DW       0              ;;A file első clustere
        D_SIZE  DD       0              ;;A file hossza
DIRENT  ENDS                            ;
                                        ;
SIZ_DIRENT      EQU     TYPE DIRENT     ;
                                        ;
;-----------------------------------------;
; File-attributum konstansok
;---------------------------------;
                                  ;
ATR_RDONLY      EQU     01H       ;Csak olvasható file
ATR_HIDDEN      EQU     02H       ;Láthatatlan file
ATR_SYSTEM      EQU     04H       ;System file
                                  ;
ATR_VOLAB       EQU     08H       ;Lemezcimke
ATR_DRTRY       EQU     10H       ;Directory
ATR_ARCHV       EQU     20H       ;Archív file
                                  ;
ATR_DIR         EQU     ATR_DRTRY OR ATR_HIDDEN OR ATR_SYSTEM
ATR_ALL         EQU     ATR_DIR OR ATR_VOLAB OR ATR_ARCHV
                                  ;
```

```
;---------------------------------;
; A DATE elem belső szerkezete
;---------------------------------;
                                  ;
DATE_R  RECORD  YEAR:7, MONTH:4, DAY:5
                                  ;
;---------------------------------;
; A TIME elem belső szerkezete
;---------------------------------;
                                  ;
TIME_R  RECORD  HOUR:5, MINUTE:6,TWOSEC:5
                                  ;
;---------------------------------;
; Filenévkezelő konstansok
;---------------------------------;
                                  ;
SIZ_FILSPEC     EQU     12        ;File-specifikáció hossza
SIZ_FILSP       EQU     11        ;Belső file-specifikáció h.
SIZ_FILNAME     EQU      8        ;File-név hossza
SIZ_FILTYPE     EQU      3        ;File-típus hossza
                                  ;
;---------------------------------;
; Ez a fejezet a DOS file-kezeléshez szükséges
; legfontosabb konstans-, makró-, struktúra- és rekorddefini-
; ciókat tartalmazza.
;-------------------------------------;
; File-handle kezelési konstansok     ;
;                                     ;
                                  ;-------;
DOS_CREAT       EQU     3CH       ;File-handle létreh. (csonkit)
DOS_OPEN        EQU     3DH       ;File vagy device megnyitása
        DOS_OPEN_READ   EQU     0       ;olvasásra
        DOS_OPEN_WRIT   EQU     1       ;írásra
        DOS_OPEN_RW     EQU     2       ;írásra és olvasásra
                                        ;
        DOS_OPEN_NOINH  EQU     80H     ;nem öröklődik
                                        ;
        DOS_OPEN_COMP   EQU     00H     ;kompatibilis mód
        DOS_OPEN_DNRW   EQU     10H     ;írás-olvasás tiltott
        DOS_OPEN_DNWRIT EQU     20H     ;írás tiltott
        DOS_OPEN_DNREAD EQU     30H     ;olvasás tiltott
        DOS_OPEN_DNNONE EQU     40H     ;semmi sem tiltott
```

```
DOS_CLOSE       EQU     3EH     ;File-handle lezárása
DOS_READ        EQU     3FH     ;Olvasás file-handle-ről
DOS_WRITE       EQU     40H     ;Írás file-handle-re
DOS_MOVPNT      EQU     42H     ;File pointer módosítása
        DOS_MOVPNT_ST   EQU     00      ;file kezdetéhez
        DOS_MOVPNT_PT   EQU     01      ;kurrens pointerhez
        DOS_MOVPNT_END  EQU     02      ;file végéhez visz.
                                ;-------;
DOS_IOCTL       EQU     44H     ;Input-output vezérlés
        DOS_IOCTL_GDAT  EQU     00H     ;Handle lekérdezése
        DOS_IOCTL_SDAT  EQU     01H     ;Handle beállítása
        DOS_IOCTL_RCNT  EQU     02H     ;Vezérlőcsatorna olv.
        DOS_IOCTL_WCNT  EQU     03H     ;Vezérlőcsatorna-írás
        DOS_IOCTL_RDSK  EQU     04H     ;Disk vez. csat. olv.
        DOS_IOCTL_WDSK  EQU     05H     ;Disk vez. csat. írás
        DOS_IOCTL_GINS  EQU     06H     ;Input státusz lekérd.
        DOS_IOCTL_GOUTS EQU     07H     ;Output státusz lekérd.
;MS-DOS 3.00 és későbbi verziók          ;
        DOS_IOCTL_DSKCH EQU     08H     ;Lemez cserélhető?
;MS-DOS 3.1 és későbbi verziók           ;
        DOS_IOCTL_REDBK EQU     09H     ;Átirányított lemez?
;MS-DOS 3.1 és későbbi verziók           ;
        DOS_IOCTL_REDHD EQU     0AH     ;Hálózatra átir. handle?
;MS-DOS 3.00 és későbbi verziók          ;
        DOS_IOCTL_RTRYC EQU     08H     ;Próbálkozások száma
                                ;-------;
DOS_DUPHND      EQU     45H     ;Handle megkétszerezése
DOS_FRCHND      EQU     46H     ;Handle átirányítása
                                ;
;--------------------------------;
;MS-DOS 3.00 és későbbi verziók
;--------------------------------;
                                ;
DOS_CRTEMP      EQU     5AH     ;Ideiglenes file létrehozása
DOS_CRNEWF      EQU     5BH     ;Nem létező file létrehozása
DOS_PROTCT      EQU     5CH     ;File részének zárása/feloldása
        DOS_PROTCT_LOCK EQU     0       ;lezárás
        DOS_PROTCT_UNL  EQU     1       ;feloldás
                                ;
```

```
;-----------------------------------------;
; File-handle kezelési makrók
; File-handle létrehozó makró
;-----------------------------------------;
                                          ;
HNDCTL  MACRO   CODE, FILESPEC            ;
                MOV     DX, FILESPEC      ;;Filespec. címe
                DOSCALL CODE              ;;
        ENDM                              ;
                                          ;
;-----------------------------------------;
; File-handle funkció hívó makró
;-----------------------------------------;
                                          ;
HNDCLL  MACRO   CODE, HANDLE              ;
                MOV     BX, HANDLE        ;;Handle BX-be
                DOSCALL CODE              ;;
        ENDM                              ;
                                          ;
;-----------------------------------------;
; Rekord az elérési mód byte-hoz
;-----------------------------------------;
                                          ;
ACCMOD  RECORD  INH:1, SHARE:3, ACC_RES:1=0, ACCESS:3;
                                          ;
;-----------------------------------------;
; Kisegítő makrók a standard file-ok használatához
; MESS-t
;       MSGDEF név, szöveg, [NOCR], NODOLLAR
; makróhívással kényelmes deklarálni
;-----------------------------------------;
                                          ;
INREAD  MACRO   BUFF, SIZE                ;
        IFNB    <SIZE>                    ;;Megadott hossz
                READ_H  OFFSET! BUFF, SIZE, HND_STD_IN
        ELSE                              ;;Default-hossz
                READ_H  OFFSET! BUFF, SIZ_&BUFF, HND_STD_IN
        ENDIF                             ;;
        ENDM                              ;
                                          ;
```

```
OUTWRIT MACRO   MESS, SIZE                      ;
        IFNB    <SIZE>                          ;;Megadott hossz
                WRIT_H  OFFSET! MESS, SIZE,     HND_STD_OUT
        ELSE                                    ;;Default-hossz
                WRIT_H  OFFSET! MESS, SIZ_&MESS, HND_STD_OUT
        ENDIF                                   ;;
        ENDM                                    ;
                                                ;
ERRWRIT MACRO   MESS                            ;
        IFNB    <SIZE>                          ;;Megadott hossz
                WRIT_H  OFFSET! MESS, SIZE,     HND_STD_ERR
        ELSE                                    ;;Default-hossz
                WRIT_H  OFFSET! MESS, SIZ_&MESS, HND_STD_ERR
        ENDIF                                   ;;
        ENDM                                    ;
                                                ;
;---------------------------------------;
; Kisegítő makrók egy file-handle írásához/olvasásához
;---------------------------------------;
                                                ;
READ_H  MACRO   BUFF, SIZE, HANDLE              ;
                MOV     DX, BUFF                ;;Buffer címe
        IFNB    <SIZE>                          ;;Megadott hossz
                MOV     CX, SIZE                ;;
        ELSE                                    ;;Default-hossz
                MOV     CX, SIZ_&BUFF           ;;
        ENDIF                                   ;;
                HNDCLL  DOS_READ, HANDLE;;
        ENDM                                    ;
                                                ;
WRIT_H  MACRO   BUFF, SIZE, HANDLE              ;
                MOV     DX, BUFF                ;;Buffer címe
        IFNB    <SIZE>                          ;;Megadott hossz
                MOV     CX, SIZE                ;;
        ELSE                                    ;;Default-hossz
                MOV     CX, SIZ_&BUFF           ;;
        ENDIF                                   ;;
                HNDCLL  DOS_WRITE, HANDLE;
        ENDM                                    ;
                                                ;
```

```
;-----------------------------------------;
;MS-DOS 3.00 és későbbi verziók
; Temporális file létrehozásához adatblokk definíció
;-----------------------------------------------;
                                                ;
TMPBLK  MACRO   TEMPSPEC, DIRECTORY             ;
TEMPSPEC        DB      '&DIRECTORY&', 0        ;;Directory-név
TEMPSPEC_NAME   DB      13      DUP ( 0 )       ;;File-név
        ENDM                                    ;
                                                ;
;-----------------------------------------------;
; Ez a fejezet a directory és directory-bejegyzések
; kezeléséhez szükséges konstansokat tartalmazza.
;-----------------------------------;
; Directory-kezelő konstansok       ;
                                    ;
DOS_MKDIR       EQU     39H         ;Directory-létrehozás
DOS_RMDIR       EQU     3AH         ;Directory-törlés
DOS_CHDIR       EQU     3BH         ;Kurrens directory váltása
                                    ;
DOS_GTCDIR      EQU     47H         ;Kurrens directory lekérdezése
                                    ;
DOS_DELENTRY    EQU     41H         ;Directory-bejegyzés(ek) törl.
DOS_GSATTR      EQU     43H         ;File-attr. lekérdezés/módosítás
                                    ;
DOS_FDFRST      EQU     4EH         ;Első bejegyzés megkeresése
DOS_FDNEXT      EQU     4FH         ;Következő bejegyz. megkeresése
DOS_CHENTRY     EQU     56H         ;Directory-bejegyzés módosítása
DOS_GSDTTM      EQU     57H         ;File dátumának/idejének mód.
        DOS_GSDTTM_GET  EQU     0       ;Lekérdezés
        DOS_GSDTTM_SET  EQU     1       ;Módosítás
        DOS_GSDTTM_BASE EQU     1980    ;Alapév
                                        ;
;---------------------------------------;
; Kisegítő konstans- és struktúradeklarációk
;-----------------------------------;
                                    ;
SIZ_PATH        EQU     64          ;Path-lekérdező blokk hossza
                                    ;
```

```
FDENT   STRUC                                   ;
        FDENT_RES       DB  21 DUP (?)  ;;Fenntartott terület
        FDENT_ATTR      DB   0          ;;A file attributuma
        FDENT_TIME      DW   0          ;;A file utolsó írásá-
        FDENT_DATE      DW   0          ;; nak ideje és dátuma
        FDENT_SIZE      DD   0          ;;A file hossza
        FDENT_NAME      DB  13 DUP (?)  ;;A file neve és típusa
FDENT   ENDS                                    ;
                                                ;
;---------------------------------------;
; Ez a fejezet a memória és a programvezérlési funkciók
; kezeléséhez szükséges konstansokat tartalmazza.
;-------------------------------;
; Memória- és processz-vezérlés ;
                                ;
DOS_MALLOC      EQU     48H     ;Memória allokálása
DOS_MFREE       EQU     49H     ;Allokált memória felszab.
DOS_SETBLK      EQU     4AH     ;Memóriablokk módosítása
                                ;
DOS_EXEC        EQU     4BH     ;Programbeolvasás
        DOS_EXEC_PRC    EQU     0       ; végrehajtásra
        DOS_EXEC_OVL    EQU     3       ; overlay-ágként
                                ;-------;
DOS_GTRETCD     EQU     4DH     ;Processz válaszkódja lekérd.
DOS_CRPRSEG     EQU     26H     ;Programszegmens létrehozása
                                ;
;-------------------------------;
; Processzhívást segítő makrók és struktúrák
; Processzvezérlő blokk
;-------------------------------;
                                ;
EX_BLK  STRUC                   ;
        EX_S_ENV        DW  0   ;;Környezet szegmenscíme
        EX_P_PAR        DD  0   ;;Paraméterblokk teljes címe
        EX_P_FCB1       DD  0   ;;Elsődleges FCB teljes címe
        EX_P_FCB2       DD  0   ;;Másodlagos FCB teljes címe
EX_BLK  ENDS                    ;
                                ;
```

```
;-----------------------------------;
; Vezérlőblokk kitöltésére szolgáló makrók
; Blokk-kitöltés futásidőben
;-------------------------------------------;
                                            ;
EX_BLK_INI      MACRO   BLOCK, ENV, PARLIN, P_FCB, S_FCB
                                            ;
                PUSH    ES                  ;;
                PUSH    BX                  ;;
                PUSH    SI                  ;;Regiszterek mentése
                                            ;;
                MOV     BX, OFFSET BLOCK    ;;
                MOV     SI, SEG ENV         ;;Environment szegmense
                MOV     EX_S_ENV.[ BX ], SI
                LES     SI, PARLIN          ;;P.sor címe
                MOV     EX_P_PAR.[ BX + OFFS ], SI
                MOV     EX_P_PAR.[ BX + SEGM ], ES
                LES     SI, P_FCB           ;;Elsődl. FCB címe
                MOV     EX_P_FCB1.[ BX + OFFS ], SI
                MOV     EX_P_FCB1.[ BX + SEGM ], ES
                LES     SI, S_FCB           ;;Másodl. FCB címe
                MOV     EX_P_FCB2.[ BX + OFFS ], SI
                MOV     EX_P_FCB2.[ BX + SEGM ], ES
                                            ;;
                POP     SI                  ;;
                POP     BX                  ;;
                POP     ES                  ;;Helyreállítás
        ENDM                                ;
                                            ;
;-------------------------------------------;
; Overlay-ág betöltését segítő struktúrák és makrók
;-------------------------------------------;
                                            ;
OV_BLK  STRUC                               ;
        OV_LOADS        DW      0           ;;Betöltés szegmense
        OV_RELOC        DW      0           ;;Relokációs faktor
OV_BLK  ENDS                                ;
                                            ;
```

```
;-----------------------------------------;
; Blokkdefiniálás fordítási időben
;---------------------------------;
                                  ;
OV_BLK_DEF      MACRO   BLOCKNAME, LOAD_ADDR, REL_FACT
BLOCKNAME       OV_BLK  <SEG LOAD_ADDR, REL_FACT>
        ENDM                      ;
                                  ;
;---------------------------------;
; Ez a fejezet az interrupt-vektorok kezeléséhez
; szükséges konstansdeklarációkat tartalmazza.
;-----------------------------------------;
; Interrupt-vektor kezelési konstansok  ;
                                  ;-------;
DOS_GETITVECT   EQU     35H       ;IT vektor beolvasása
DOS_SETITVECT   EQU     25H       ;IT vektor módosítása
                                  ;
;---------------------------------;
; Ez a fejezet a kisegítő DOS-funkciók használatához
; szükséges konstansdeklarációkat tartalmazza
;---------------------------------;
; Kisegitő MS-DOS funkciók        ;
                                  ;
DOS_STVFYST     EQU     2EH       ;VERIFY státusz beállítása
DOS_GTVFYST     EQU     54H       ;VERIFY státusz lekérdezése
DOS_GSBREAK     EQU     33H       ;BREAK státus lekérd./beáll.
        DOS_GSBREAK_GET EQU     00H     ;Lekérdezés
        DOS_GSBREAK_SET EQU     00H     ;Beállítás
                                  ;-------;
DOS_GTDATE      EQU     2AH       ;Dátum lekérdezése
DOS_STDATE      EQU     2BH       ;Dátum beállítása
DOS_GTTIME      EQU     2CH       ;Idő lekérdezése
DOS_STTIME      EQU     2DH       ;Idő beállítása
                                  ;
DOS_GTDOSVR     EQU     30H       ;DOS verziószám lekérdezése
                                  ;
DOS_GSCNTRY     EQU     38H       ;Országfüggő paraméterek kez.
        DOS_GSCNTRY_GET EQU     0       ;Lekérdezés
        DOS_GSCNTRY_SET EQU     -1      ;Beállítás
                                  ;-------;
; MS-DOS 3.1 és későbbi verziók ;
DOS_GSALLSTR    EQU     58H       ;Memóriakezelési stratégia
```

```
; MS-DOS 3.00 és későbbi verziók;
DOS_GTEXTERR    EQU     59H     ;Kiterjesztett hiba lekérdezése
; MS-DOS 3.1 és későbbi verziók, Microsoft Networks
DOS_MNMPRT      EQU     5EH     ;Gép neve/printer programozása
        DOS_MNMPRT_NAME         EQU     00      ;Gép neve
        DOS_MNMPRT_SETUP        EQU     02      ;Printer progr.
; MS-DOS 3.1 és későbbi verziók, Microsoft Networks
DOS_ASSIGN      EQU     5FH     ;Hozzárendelési lista kezelése
        DOS_ASSIGN_GET          EQU     02H     ;Elem lekérd.
        DOS_ASSIGN_MAKE         EQU     03H     ;Elem létrehoz.
        DOS_ASSIGN_CANC         EQU     04H     ;Elem törlése
; MS-DOS 3.00 és későbbi verziók;
DOS_GETPSP      EQU     62H     ;PSP cím lekérdezése
                                ;
;--------------------------------;
; Ez a fejezet a kilépési funkciókhoz szükséges
; konstansdeklarációkat tartalmazza.
;--------------------------------;
; Kilépési funkciók             ;
                                ;
DOS_COMEXIT     EQU     00H     ;Kilépés (COM) programból
DOS_ENDPRC      EQU     4CH     ;Kilépés (EXE) programból
DOS_STAYRES     EQU     31H     ;Kilépés bentmaradva
                                ;
;--------------------------------;
; Szabványos kilépési kódok
;--------------------------------;
                                ;
TRM_SUCC        EQU     0       ;Sikeres futás kódja
TRM_DTERR       EQU     1       ;Sikertelen adathiba miatt
TRM_PRERR       EQU     2       ;Sikertelen programhiba miatt
                                ;
;--------------------------------;
; Kilépési makró - két címkéje ERR_EXIT és SUCC_EXIT; ezekre
; rá lehet ugrani. ERR_EXIT esetén feltételes blokkból
; meghívja PRTERRMSG-t.
;--------------------------------;
                                ;
NOMESS  EQU     1               ;Kilépés üzenet nélkül
NOTERM  EQU     1               ;Kilépés 0 kilépési kóddal
```

```
EXIT    MACRO   NOMSG, TERM_CODE_SPEC   ;
        LOCAL   QUIT                    ;;
ERR_EXIT:                               ;;Hibás kilépés
        IFB     <NOMSG>                 ;;Hibaértelmezés kell
                CALL    PRTERRMSG       ;;
        ENDIF                           ;;
        IFNB    <TERM_CODE_SPEC>        ;;Terminálási kód kell
                MOV     AL, TRM_DTERR   ;;Terminálási kód AL-be
        ELSE                            ;;
                MOV     AL, TRM_SUCC    ;;Ne szemét legyen AL!
        ENDIF                           ;;
                JMP     QUIT            ;;
SUCC_EXIT:                              ;;Sikeres kilépés
                MOV     AL, TRM_SUCC    ;;
QUIT:                                   ;;
                DOSCALL DOS_ENDPRC      ;;
        ENDM                            ;
                                        ;
;---------------------------------------;
; Ez a fejezet az új MS-DOS funkciók hibakódjainak
; kezeléséhez szükséges konstansdeklarációkat tartalmazza.
;---------------------------------------;
; Az új MS-DOS funckiók hibakódjai      ;
                                ;-------;
ERR_ILLFNC      EQU     01H     ;Alfunkció kódja illegális
ERR_NOFILE      EQU     02H     ;Nincs ilyen file
ERR_NOPATH      EQU     03H     ;Nincs ilyen path
ERR_NOHND       EQU     04H     ;Nincs több file-sorszám
ERR_ACDND       EQU     05H     ;Tiltott elérés
ERR_INVHND      EQU     06H     ;A file-sorszám érvénytelen
ERR_MCBINV      EQU     07H     ;Memórialeíró blokk sérült
ERR_INSMEM      EQU     08H     ;Nincs elég memória
ERR_MBAINV      EQU     09H     ;Memóriablokk-cím érvénytelen
ERR_ENVILL      EQU     0AH     ;A környezet illegális
ERR_FRMINV      EQU     0BH     ;Illegális formátum
ERR_ACCINV      EQU     0CH     ;Elérési kód illegális
ERR_DATINV      EQU     0DH     ;Illegális adatok
ERR_DRVINV      EQU     0FH     ;Illegális drive
ERR_CURDIR      EQU     10H     ;Kurrens directory törlése
ERR_DFDRV       EQU     11H     ;Különböző drive-ok
ERR_NOMFIL      EQU     12H     ;Nincs több file
```

```
;---------------------------------;
; MS-DOS 3.00 és későbbi verziók hibakódjai
;---------------------------------;
                                  ;
ERR_WRTPR       EQU     13H       ;Írásvédett lemez
ERR_BADDSK      EQU     14H       ;Rossz lemez
ERR_DRNRDY      EQU     15H       ;A drive nem üzemkész
ERR_INVDKC      EQU     16H       ;A lemezkez. parancs illegális
ERR_CRC         EQU     17H       ;CRC hiba
ERR_INVLNG      EQU     18H       ;Hosszparaméter érvénytelen
ERR_SEEK        EQU     19H       ;Sikertelen sávkeresés
ERR_NOTDOS      EQU     1AH       ;Nem MS-DOS lemez
ERR_SECNFD      EQU     1BH       ;Sikertelen szektorkeresés
ERR_OUTPPR      EQU     1CH       ;Papír kifogyott
ERR_WRFLT       EQU     1DH       ;Írási hiba
ERR_RDFLT       EQU     1EH       ;Olvasási hiba
ERR_GNFAIL      EQU     1FH       ;Általános hiba
ERR_SHVLT       EQU     20H       ;Illegális file-megosztás
ERR_LCKVLT      EQU     21H       ;Lock-olt file
ERR_WRGDSK      EQU     22H       ;Rossz lemez
ERR_FCNNAV      EQU     23H       ;Nem elérhető FCB

;---------------------------------;
; Hibaüzeneteket és az üzenetek címtáblázatát definiáló
; makró
;---------------------------------;
                                  ;
ERRMSG  MACRO   CR                ;
                                  ;;
MSGDEF  DE_ILMSG,  '?? - Illegális hibaüzenet', CR
MSGDEF  DE_ILLFNC, '01 - Alfunkció kódja érvénytelen', CR
MSGDEF  DE_NOFILE, '02 - Nincs ilyen file', CR
MSGDEF  DE_NOPATH, '03 - Nincs ilyen path', CR
MSGDEF  DE_NOHND,  '04 - Nincs több file-sorszám', CR
MSGDEF  DE_ACDND,  '05 - Tiltott elérés', CR
MSGDEF  DE_INVHND, '06 - Nemlétező sorszám', CR
MSGDEF  DE_MCBINV, '07 - Sérült memórialeíró blokk', CR
MSGDEF  DE_INSMEM, '08 - Nincs elég memória', CR
MSGDEF  DE_MBAINV, '09 - Illegális memóriablokkcím', CR
MSGDEF  DE_ENVILL, '0A - Illegális környezet', CR
MSGDEF  DE_FRMINV, '0B - érvénytelen formátum', CR
MSGDEF  DE_ACCINV, '0C - Illegális elérési mód', CR
```

```
MSGDEF  DE_DATINV, '0D - Illegális adatok', CR
MSGDEF  DE_DRVINV, '0F - Illegális drive', CR
MSGDEF  DE_CURDIR, '10 - Kurrens dir. nem törölhető', CR
MSGDEF  DE_DFDRV,  '11 - Különböző drive-ok', CR
MSGDEF  DE_NOMFIL, '12 - Nincs több ilyen file', CR
MSGDEF  DE_WRTPR,  '13 - Írásvedett lemez', CR
MSGDEF  DE_BADDSK, '14 - Rossz lemez', CR
MSGDEF  DE_DRNRDY, '15 - A drive nem üzemkész', CR
MSGDEF  DE_INVDKC, '16 - Lemezkez. parancs illegális', CR
MSGDEF  DE_CRC,    '17 - CRC hiba', CR
MSGDEF  DE_INVLNG, '18 - Hosszparaméter érvénytelen', CR
MSGDEF  DE_SEEK,   '19 - Sikertelen sávkeresés', CR
MSGDEF  DE_NOTDOS, '1A - Nem MS-DOS lemez', CR
MSGDEF  DE_SECNFD, '1B - Sikertelen szektorkeresés', CR
MSGDEF  DE_OUTPPR, '1C - Papir kifogyott', CR
MSGDEF  DE_WRFLT,  '1D - Írási hiba , CR
MSGDEF  DE_RDFLT,  '1E - Olvasási hiba', CR
MSGDEF  DE_GNFAIL, '1F - Általános hiba', CR
MSGDEF  DE_SHVLT,  '20 - Illegális file-megosztás', CR
MSGDEF  DE_LCKVLT, '21 - Lock-olt file', CR
MSGDEF  DE_WRGDSK, '22 - Rossz lemez', CR
MSGDEF  DE_FCNNAV, '23 - Nem elérhető FCB', CR
                                    ;;
;;----------------------------------;
;; Kezelőtábla a hibaüzenetek kiírásához
;;----------------------------------;
                                    ;;
ERRMSG_TAB      DW      DE_ILMSG
                                    ;;Címtáblázat
DW  DE_ILLFNC, DE_NOFILE, DE_NOPATH, DE_NOHND
DW  DE_ACDND,  DE_INVHND, DE_MCBINV, DE_INSMEM
DW  DE_MBAINV, DE_ENVILL, DE_FRMINV, DE_ACCINV
DW  DE_DATINV, DE_DRVINV, DE_ILMSG,  DE_CURDIR
DW  DE_DFDRV,  DE_NOMFIL, DE_WRTPR,  DE_BADDSK
DW  DE_DRNRDY, DE_INVDKC, DE_CRC,    DE_INVLNG
DW  DE_SEEK,   DE_NOTDOS, DE_SECNFD, DE_OUTPPR
DW  DE_WRFLT,  DE_RDFLT,  DE_GNFAIL, DE_SHVLT
DW  DE_LCKVLT, DE_WRGDSK, DE_FCNNAV
                                    ;;------;
ERRMSG_SIZ_TAB  DW      SIZ_DE_ILMSG    ;;
```

```
                                        ;;Hossztáblázat
DW   SIZ_DE_ILLFNC, SIZ_DE_NOFILE, SIZ_DE_NOPATH, SIZ_DE_NOHND
DW   SIZ_DE_ACDND,  SIZ_DE_INVHND, SIZ_DE_MCBINV, SIZ_DE_INSMEM
DW   SIZ_DE_MBAINV, SIZ_DE_ENVILL, SIZ_DE_FRMINV, SIZ_DE_ACCINV
DW   SIZ_DE_DATINV, SIZ_DE_DRVINV, SIZ_DE_ILMSG,  SIZ_DE_CURDIR
DW   SIZ_DE_DFDRV,  SIZ_DE_NOMFIL, SIZ_DE_WRTPR,  SIZ_DE_BADDSK
DW   SIZ_DE_DRNRDY, SIZ_DE_INVDKC, SIZ_DE_CRC,    SIZ_DE_INVLNG
DW   SIZ_DE_SEEK,   SIZ_DE_NOTDOS, SIZ_DE_SECNFD, SIZ_DE_OUTPPR
DW   SIZ_DE_WRFLT,  SIZ_DE_RDFLT,  SIZ_DE_GNFAIL, SIZ_DE_SHVLT
DW   SIZ_DE_LCKVLT, SIZ_DE_WRGDSK, SIZ_DE_FCNNAV
                                       ;;
         ENDM                          ;
                                       ;
;-------------------------------;
; Ez a fejezet kisegitő konstansdeklarációkat tartalmaz
;-----------------------------------------------;
; PSP (Program Segment Prefix) konstansok       ;
                                       ;---------------;
PSP_EXIT          EQU     00H          ;Kilépési utasitás cime
PSP_TOPMRY        EQU     02H          ;Memória teteje
                                       ;
PSP_DOSJMP        EQU     05H          ;Ugrás a DOS-funkciókra
PSP_TERMADR       EQU     0AH          ;Kilépési cim
PSP_CTLCADR       EQU     0EH          ;Control-BREAK kilépési cim
PSP_CRERADR       EQU     12H          ;Kilépési cim krit. hiba esetén
                                       ;
PSP_ENVSEG        EQU     2CH          ;Environment szegmenscime
                                       ;
PSP_DOSCALL       EQU     50H          ;DOS-funkció hivása
                                       ;
PSP_FCB1          EQU     5CH          ;Elsődleges FCB
PSP_FCB2          EQU     6CH          ;Másodlagos FCB
                                       ;
PSP_DTA           EQU     80H          ;Disk Transfer Address
PSP_CMDLINSIZ     EQU     80H          ;Paramétersor hossza
PSP_CMDLIN        EQU     81H          ;Paramétersor
                                       ;
```

```
;---------------------------------;
; Kritikus hiba kezelési konstansok és struktúrák
;       (hibakódokat lásd a "Hibakódok" fejezetben)
; A stack szerkezete kritikus hiba kezelése közben
;---------------------------------;
                                  ;
CRERRST STRUC                     ;
        CE_IPD  DW      0         ;;DOS IP-e
        CE_CSD  DW      0         ;;DOS CS-e
        CE_FLD  DW      0         ;;DOS FLAG-jei
                                  ;;
                CE_AX   DW  0     ;;Program AX-e
                CE_BX   DW  0     ;;Program BX-e
                CE_CX   DW  0     ;;Program CX-e
                CE_DX   DW  0     ;;Program DX-e
                CE_SI   DW  0     ;;Program SI-e
                CE_DI   DW  0     ;;Program DI-e
                CE_BP   DW  0     ;;Program BP-e
                CE_DS   DW  0     ;;Program DS-e
                CE_ES   DW  0     ;;Program ES-e
        CE_IPP  DW      0         ;;Program IP-e
        CE_CSP  DW      0         ;;Program CS-e
        CE_FLP  DW      0         ;;Program FLAG-jei
CRERRST ENDS                      ;
                                  ;
;---------------------------------;
; Konstansok a kritikus hiba kezeléséhez
;-------------------------------------------------;
                                                  ;
CE_STACK_SIZ            EQU     (3+9+3)*WORD      ;Inf. hossz
CE_USERSTACK_SIZ        EQU     (3+9)*WORD        ;User inf. hossz
CE_USERREG_SIZ          EQU     9*WORD            ;Regiszterek
CE_RETADDR_SIZ          EQU     3*WORD            ;Visszatérés
                                          ;-------;
BIT_DISK_ERR            EQU     80H       ;Lemezhiba-bit
                                          ;
BIT_DISK_WRITE          EQU     01H       ;Hiba írás közben
                                          ;
```

```
FLD_DISK_DOS            EQU     00H     ;Hiba DOS-területen
FLD_DISK_FAT            EQU     01H     ;Hiba FAT-területen
FLD_DISK_DIR            EQU     02H     ;Hiba directory-n
FLD_DISK_DATA           EQU     03H     ;Hiba adatterületen
                                        ;
BIT_DISK_FAIL_ALLW      EQU     08H     ;FAIL megengedett
BIT_DISK_RETRY_ALLW     EQU     10H     ;RETRY megengedett
BIT_DISK_IGNORE_ALLW    EQU     20H     ;INGORE megengedett
                                        ;
;---------------------------------------;
; DOS válaszértékek kritikus hibát kezelő rutinban
;---------------------------------------;
                                        ;
CE_IGNORE       EQU     0               ;
CE_RETRY        EQU     1               ;
CE_ABORT        EQU     2               ;
CE_FAIL         EQU     3               ;
                                        ;
;---------------------------------------;
; Lemezhiba-byte szerkezete
;---------------------------------------;
                                        ;
DSKERR  RECORD  D_RW:1,D_A:2,D_F:1,D_R:1,D_I:1,D_RS:1,D_DSK:1
                                        ;
;---------------------------------------;
; A STATUS flagjei a stacken történő bit-manipulációhoz
;---------------------------------------;
                                        ;
FL_CARRY        EQU     0001H           ;Carry bit maszkja
FL_PARITY       EQU     0004H           ;Paritásbit maszkja
FL_ACARRY       EQU     0010H           ;Kis. carry-bit maszkja
FL_ZERO         EQU     0040H           ;Zéró bit maszkja
FL_SIGN         EQU     0080H           ;Előjelbit maszkja

FL_TRAP         EQU     0100H           ;Trap bit maszkja
FL_INT          EQU     0200H           ;Interrupt bit maszkja
FL_DIR          EQU     0400H           ;Irány bit maszkja
FL_OFL          EQU     0800H           ;Túlcsord. bit maszkja
                                        ;
```

```
;-----------------------------------------;
; Közhasznú makrók
;
; Feltételes hosszú ugrást végrehajtó makró
;       (Long Conditional Jump)
;---------------------------------;
                                  ;
LCJ     MACRO   COND, ADDR        ;
        LOCAL   NOCOND            ;;
                JN&COND NOCOND    ;;Ellentétes rövid ugrás
                JMP     ADDR      ;;Feltétel nélküli hosszú ugrás
NOCOND:                           ;;
        ENDM                      ;
                                  ;
;---------------------------------;
; Rekordkezelő makrók
; A REC nevű makró EL_NAME elemének beállítása NUM értékre
;-----------------------------------------;
                                          ;
SET_EL  MACRO   REC, NUM, EL_NAME         ;
                PUSH    AX                ;;
                PUSH    BX                ;;
                PUSH    CX                ;;Regiszterek mentése
                                          ;;
                MOV     BX, NUM           ;;NUM BX-be
                MOV     AX, REC           ;;REC AX-be
                AND     AX, NOT MASK EL_NAME    ;;EL_NAME törl.
                MOV     CL, EL_NAME       ;;
                SHL     BX, CL            ;;NUM pörg. EL_NAME-ra
                OR      AX, BX            ;;EL_NAME beállítása
                MOV     REC, AX           ;;REC módosítása
                                          ;;
                POP     CX                ;;
                POP     BX                ;;
                POP     AX                ;;Regiszterek vissza
        ENDM                              ;
                                          ;
```

```
;-----------------------------------------;
; A REC nevű rekord EL_NAME elemének előállítása AX-ben
;-----------------------------------------;
                                          ;
GET_EL  MACRO   REC, EL_NAME              ;
                PUSH    CX                ;;Regiszter mentés
                MOV     CL, EL_NAME       ;;
                MOV     AX, REC           ;;REC AX-be
                AND     AX, MASK EL_NAME  ;;Egyéb részek törlése
                SHR     AX, CL            ;;EL_NAME előállítása
                POP     CX                ;;Regiszter vissza
        ENDM                              ;
```

Nem minden dolgot fogunk részletesen elmagyarázni, hiszen a file egyfelől tele van teljesen természetes dolgokkal, másfelől pedig számos apróságot elmagyaráztunk már az FCBCLL.INC file-al kapcsolatban is.

A magyarázó megjegyzéseket kezdjük a file elején! A file természetesen egy bevezető megjegyzéssel kezdődik, amely legalább az itt megadott információkat tartalmazza: mi ez a file, mire szolgál, hogy lehet használni.

A karakterkonstansok közül talán a CHR_NUMST-CHR_NUMEND és a másik két hasonló pár szorul magyarázatra. Amennyiben meg kell vizsgálnunk, hogy szám-, nagybetű- vagy kisbetű-karakterrel van-e dolgunk, ezek a konstansok kényelmessé teszik a feltételvizsgálatot.

A számtalanszor használt DOSCALL makró ilyen megfogalmazása azért kényelmes, mert egy vagy két paraméterrel is meghívhatjuk (ti. a funkciókód mellett az alfunkció kódját is megadhatjuk, ha van).

A LONGPNT struktúra nagyon hasznos, mert nem kell fejben tartani, hogy a duplaszavas címben az offset és a szegmens hol helyezkednek el. E struktúrát egyébként szinte soha nem használjuk területfoglalásra, pusztán a tagok nevét használjuk, mint offseteket.

Az üzenetdefiniáló makró megfogalmazása itt azért egészen bonyolult, hogy ugyanazon makróval tudjunk definiálni egy üres CR-LF-et, vagy egy tetszőleges üzenetet CR-LF-el, vagy anélkül és záró dollárjellel vagy anélkül.

Klaviatúra-buffer definiálására írhatnánk egy ravaszul makróba ágyazott struktúrát, amely elegánsabb felhasználást, hivatkozást tenne lehetővé; céljainknak azonban ez is megfelel.

A file további részein, egészen az OUTWRIT, INREAD, ERRWRIT, READ_H és WRIT_H makrókig nincs semmi ravaszság. Ezekben a makrókban figyeljük meg az alapértelmezett vagy egy explicit üzenethossz felhasználását, amely igen kényelmes és rugalmas makróhívást tesz lehetővé. Ha MSGDEF-el vagy ahhoz hasonló módon definiáltuk a kiírandó blokkot, nem kell a blokkhosszúságot expliciten megadnunk, de bármikor megadhatjuk azt is.

Az ideiglenes file kezelésére használatos blokkot definiáló TMPBLK makróban az "&" makró operátor felhasználását figyelhetjük meg. Talán kényelmesebb a PATH nevét idézőjelek nélkül megadni hívás során. Ez azonban nagyon vitatható pont. Leginkább az operátor használatának illusztrálására került be a file-ba.

Az EX_BLK_INI makró az eddigi leghosszabb (és munkáját tekintve a legösszetettebb) makrónk. Ezekben a címzési módok felhasználására és ravasz, lényegre mutató leírására figyeljünk!

Első ránézésre az OV_BLK-t előkészítő makróhoz hasonlóan az EX_BLK fordítási időben való előkészítése sem látszik reménytelennek. Sajnos azonban struktúrába foglalt duplaszót a fordító (ki tudja, miért? valószínűleg hiba) egyszerűen nem tud előkészíteni a <> zárójelek közé foglalt értékre. Ezért maradt el az EX_BLK fordítási időben való előkészítése.

A kilépési makró újfent érdekes feltételes blokkokat tartalmaz. Ennek az a célja, hogy hibaüzenet és kilépési kód küldése nélkül is használhassuk az EXIT makrót.

A hibaüzeneteket és az üzenetkiíráshoz szükséges indextáblázatot definiáló makró a PRTERRMSG rutinnal együtt válik világossá.

A kritikus hiba lekezelése során, mint "Az interruptok funkciói" fejezetben olvasható, a vermen kapjuk meg a regiszterek eredeti értékét. Ha valamelyiket vagy a flagek bármelyikét módosítani kívánjuk (lásd a "Kezeljünk kritikus hibát" példaprogramot), akkor használhatjuk a CRERRST struktúrát a verem direkt megcímzéséhez. A LONGPNT struktúrához hasonlóan ezt a struktúrát sem használjuk területfoglalásra, csak a tagok nevét építjük be a BP-relatív indexelt címzés indexeként.

A feltételes hosszú ugrást végrehajtó makró rendkívül hasznos, mert lehetővé teszi az INTEL 8086 assembler egyik legnagyobb hiányosságának áthidalását, ti. azt, hogy a feltételes ugrások csak -128-tól 127-ig terjedő távolságok áthidalására képesek. Kényelmesebb lenne minden utasításra külön makrót írni, ekkor pl. JC helyett LJC-t, Long Jump if Carry-t használhatnánk. Ez azonban rengeteg makrót jelentene. Az itteni defi-

nició mint definíció kényelmesebb, a makró használata viszont egy fokkal nehézkesebb. Ha már itt tartunk, felmerülhet feltételes szubrutinhívási és -visszatérési makrók definíciója is, az INTEL 8080-hoz és Z80-hoz szokott programozók ezeket valószínűleg igen lelkesen üdvözölnék. A saját programozói gyakorlatomból mindenesetre úgy emlékszem, hogy ezeket elég ritkán használtam, mert egy nagyon fontos elv súlyos megsértésére vezetnek. Ez az elv úgy szól, hogy a szubrutinnak pontosan egy belépési és egy kilépési pontja van. A feltételes visszatérés természetesen egy alternatív kilépési pontot jelent. Ha a szubrutinban regisztereket mentettünk, akkor máris nagyon kockázatos dolog több kilépési pontot definiálni!

A file végén rekordkezelő makrókat olvashatunk. Ezek hasznosságáról szólni kell; a makrók maguk nem bonyolultak, használatuk eléggé kényelmes. Arra kell csak figyelnünk, hogy egy AX-ben vagy BX-ben megadott értéket nem tölthetünk be a rekord egy mezőjére a SET_EL makró segítségével!

Az IBM PC-XT memóriatérképei

Ebben a függelékben egy vázlatos MS-DOS memóriatérképet olvashatunk. Azért nem lehet ezt a térképet teljes precizitással megadni, mert nem csak a használt MS-DOS verziótól, de a rendszer generálásától is függhet bizonyos elemek elhelyezése.

Az "MS-DOS" memóriatérképet két részben adjuk meg. A teljesség kedvéért kapott itt helyet az irodalomban "ROM BIOS" memóriatérképként ismert ábra; azért adjuk meg ebben a kötetben is, hogy ne essen szét a szervesen egybetartozó két dolog. Ezen a térképen a memória rögzítetten beépített részeit ábrázoljuk: az interrupt-vektorok, a BIOS RAM-ja, a képernyőterület és a ROM-területek helyét. A memória egyéb részei tartoznak a szigorúan vett MS-DOS hatáskörébe; ezeket külön ábrán részletezzük. Mint ebből is következik, az IBM PC memóriájának felosztása első közelítésben "hardware" felosztás (azért használunk idézőjelet, mert más gépeken a memóriát sokkal keményebb eszközökkel, például prioritásos lapregiszterekkel vagy egyéb hardware-eszközökkel is fel tudják osztani). E felosztás egyik elemét tovább osztjuk egy "software" felbontással. Aszerint választjuk el a két felosztást, hogy ROM memóriába égetett elemekről, vagy a processzor vagy egyéb áramköri elem által rögzítetten használt memóriáról van szó (interrupt-vektorok, képernyő-file területe), vagy pedig a lemezről betöltött operációs rendszer részei használják az adott területet.

0000:0000 - 0000:03FF	Processzor-, BIOS-, MS-DOS, felhasználói és egyéb interrupt-vektorok
0040:0000 - 0040:00FF	BIOS programok adatterülete
0050:0000 - RAM teteje (128) 1000:FFFF (256) 3000:FFFF (512) 7000:FFFF (640) 9000:FFFF	"Felhasználói" memóriaterület: MS-DOS adatterület, memóriarezidens elemek, installált driverek és a felhasználói programok számára (részletesen lásd külön ábrán)
B000:0000 - B000:3FFF	Fekete-fehér képernyő RAM területe (hossza 16 Kilobyte)
B800:0000 - B800:3FFF	Színes képernyő RAM területe (hossza 16 Kilobyte)
Kontrollerfüggő helyen	Merev lemezt kezelő programok (ROM a kontrollerkártyán)
F400:0000 - F400:1FFF	Felhasználói ROM terület (hossza 8 Kilobyte)
F600:0000 - F600:7FFF	Beégetett BASIC interpreter (hossza 32 Kilobyte)
FE00:0000 - FE00:1FEF	ROM BIOS programterület (hossza 8 Kilobyte)
FE00:1FF0 - FE00:1FFF	Belépési cím bekapcsolás után (ROM, ugrás a BIOS igazi belépési pontjára

Mint látható, a fenti táblázat tartalmaz néhány bizonytalan elemet. Először is, a memória felső határa függ a kiépítéstől. A merevlemez vezérlőprogramjának helye bizonytalan; van olyan konfiguráció, ahol a C000:0000 ponton helyezkedik el, de nem mindig. Van, ahol van BASIC interpreter, van, ahol nincs; nem mindenki használja ki a felhasználói memóriát sem.

0050:0000 - 0000:03FF	MS-DOS adatterület (hossza függ a verziószámtól és a generálási adatoktól)
xxxx:0000	IBMBIO.COM - az MS-DOS és a ROM BIOS rutinok közötti interface
xxxx:0000	IBMDOS.COM - MS-DOS interrupt-rutinok, INT 21 rutinok
xxxx:0000	MS-DOS bufferek, installált eszközdriverek
xxxx:0000	A COMMAND.COM memóriarezidens részei egyéb interruptok, felső rész betöltése
xxxx:0000	Felhasználói EXE vagy COM programok számára használható terület
xxxx:0000	COM programok veremterülete (256 byte)
xxxx:0000	A felhasználói memória tetejére töltött programok (/HIGH-al szerkesztve)
xxxx:0000	A COMMAND.COM felső (tranziens) része. Belső parancsok stb. (RAM felső részén)

Mint látható, ez az ábra még több bizonytalanságot tartalmaz, mint az előző. Ez persze természetes is, hiszen minden eleme függ a MS-DOS használt verziójától, az adott verzió generálási paramétereitől (hány buffert foglalunk, hány file-t akarunk egyidejűleg nyitva tartani stb.). Végül pedig az egész függ a rendelkezésre álló RAM-terület méretétől. Nagyobb memóriában természetesen hosszabb lesz a felhasználói programok számára fenntartott terület azonos és ugyanúgy generált MS-DOS verzió használata esetén is.

A scan-kódok táblázata

A szabványos IBM-klaviatúra scan-kódjainak táblázatában felül olvasható a billentyűre írt betű, alul a scan-kód.

ESC	1	2	3	4	5	6	7	8	9	0	-	=	<---
01	02	03	04	05	06	07	08	09	10	11	12	13	14

TAB	Q	W	E	R	T	Y	U	I	O	P	[	]	EN-TER
15	16	17	18	19	20	21	22	23	24	25	26	27	

CTRL	A	S	D	F	G	H	J	K	L	;	'	`	28
29	30	31	32	33	34	35	36	37	38	39	40	41	

SH-L	\	Z	X	C	V	B	N	M	,	.	/	SH-R	*
42	43	44	45	46	47	48	49	50	51	52	53	54	55

ALTMODE	S P A C E B A R	CAPS LOCK
56	57	59

F1	F2
59	60
F3	F4
61	62
F5	F6
63	64
F7	F7
65	66
F9	F10
67	68

NUM LOCK	SCR.LOCK
69	70

HOME	^	PGUP	-
71	72	73	74
<-	5	->	
75	76	77	+
END	v	PGDN	78
79	80	81	
INS		DEL	
82		83	

ahol ESC az ESCAPE, "<---" a visszatörlés, TAB a tabulátor, CTRL a kontroll-billentyű, SH-L és SH-R a bal és jobb SHIFT, F1, F2, ... a funkcióbillentyűk, és SCR.LOCK a SCROLL LOCK.

Az MS-DOS funkciók kód szerinti listája

Ebben a függelékben az MS-DOS funkciók kód szerint emelkedő sorrendben megadott listáját olvashatjuk a következők alakban:

H.k. D.k. A funkció szimbolikus neve
A funkció rövid leírása ..... oldalszám

ahol a "H.k." a funkció hexadecimális, a "D.k." pedig decimális kódja. Ezután a DOSCALL.INC file-ban használt szimbólum, végül a részletes leírás helye olvasható.

24	36	DOS_SETRCD Random record beállítás, FCB 104
25	37	DOS_SETITVECT Interrupt-vektor módosítása 236
26	38	DOS_CRPRSEG Egy új programszegmens létrehozása 216
27	39	DOS_FRDBRND Direkt blokkolvasás, FCB 104
28	40	DOS_FWRBRND Direkt blokkírás, FCB 106
29	41	DOS_FNMINI File-név előkészítése 110
2A	42	DOS_GTDATE Dátum lekérdezése 245
2B	43	DOS_STDATE Dátum módosítása 245
2C	44	DOS_GTTIME Idő lekérdezése 245
2D	45	DOS_STTIME Idő felülírása 245
2E	46	DOS_STVFYST Verify kapcsoló beállítása 244
2F	47	DOS_GETDTA DTA lekérdezés 111
30	48	DOS_GTDOSVR A DOS verziószámának lekérdezése 246
31	49	DOS_STAYRES Program elindítása 256
32	50	nincs szimbolikus név Az MS-DOS belső használatára fenntartva ... 249
33	51	DOS_GSBREAK CTRL-BREAK kapcsoló kezelése 244
34	52	nincs szimbolikus név Az MS-DOS belső használatára fenntartva ... 249
35	53	DOS_GETITVECT Interrupt-vektor lekérdezése 236
36	54	DOS_GETDFREE Szabad lemezterület lekérdezése 111
37	55	nincs szimbolikus név Az MS-DOS belső használatára fenntartva ... 249
38	56	DOS_GSCNTRY Országfüggő információ kezelése 246

Az IBM PC/XT felhasználóknak és programozóknak

2. kötet

A programozó és az MS-DOS

Sorozatunk második kötete az MS-DOS operációs rendszert ismerteti a programozó szemszögéből: milyen lehetőségeket, szolgáltatásokat nyújt ez az operációs rendszer a programoknak.

A kötet anyaga három alapvető részre oszlik: hogyan lép be a program az operációs rendszerbe, milyen rendszerfunkciók segítik munkája elvégzésében, és végül, hogyan fejezi be a futást. A rendszer bemutatását a fontos táblázatok megadása és függelék teszi teljessé. A könyv tárgyalásmódját tekintve tankönyv. Az ismereteket logikai sorrendbe csoportosítva tárgyalja, így folyamatosan is olvasható. Az anyag ismertetése ugyanakkor kézikönyv-igényességű.

Minden lehetőséget (az egészen nyilvánvalók kivételével) bőséges példaanyag szemléltet. A példaprogramok nemcsak a lehetőségek kihasználását mutatják be; igényes megfogalmazásukkal és az egyes programozási lépések részletes és alapos ismertetésével nagyban elősegíthetik a gyakorlatlan programozók stílusának javítását.

A könyvben számos olyan információ is olvasható, amelyet a korábban megjelent ismertetők nem tartalmaznak. Ezek részint az MS-DOS egyes részeinek visszafejtéséből, részint a szerző programozási gyakorlatából származnak.

Jóllehet minden példaprogram assembly nyelvű, a könyvet minden programozónak, valamint szervezőknek és rendszertervezőknek ajánljuk. Az előbbiek magától értetődő módon használhatják fel a leírtakat programozási munkájuk során, míg a szervezőknek és rendszertervezőknek igen alaposan kell ismerniük a rendszer lehetőségeit.

The IBM PC/XT for Users and Programmers

2. volume

The Programmer and the MS-DOS

The second volume introduces the MS-DOS operating system from the viewpoint of the programmer. It describes the possibilities and facilities of the programs running under this operating system.

The material of the volume is divided into three parts: how the program enters the operating system, which system functions can be called to help the operation of the program and how the program can exit. The book also contains the description of the important system tables and a number of appendices. This book is a text-book, so it contains its material in logical order to facilitate continuous reading. Simultaneously, it gives a precise description of the sytem as a manual.

All of the possibilities (excluding the quite trivial ones) are illustrated by sample programs. The sample programs illustrate not only the system facilities but also a good programming style. The important programming steps are discussed in detail so the samples could be very useful for novice programmers to improve their programming style.

The book also contains some information not published in earlier literature. They have originated from the disassembling of some parts of the operating system and from the author's programming experience.

Although all sample programs are written in assembly, this book is recommended to programmers working in any programming language and also to system managers and system analysts.

IMPULZUS

Ne hagyja KIHASZNÁLATLANUL BERENDEZÉSEIT,
hiszen a manuális kezelési fázisok automatizálásával
azok teljesítőképessége növelhető.

Ne hagyja ELVESZNI INFORMÁCIÓIT,
hiszen a számítógépes adatfeldolgozással
nagyobb információmennyiség kezelhető,
az eddig „kiselejtezett" adatokat is figyelembe veheti.

Ne hagyja ELVESZNI IDEJÉT,
hiszen a fokozott teljesítőképességű berendezésekkel,
a nagyobb sebességű, teljesebb és pontosabb adatfeldolgozással
időt nyerhet.

Segítünk Önnek!

Hiszen eddigi munkánk során tapasztalatot szereztünk
a következő feladatok megoldásában:

- egyedi mérőautomaták korszerűsítése, kiegészítése, adatfeldolgozó szolgáltatásainak bővítése
- adatgyűjtő, tároló, feldolgozó és továbbító célgépek készítése a felhasználó igényei szerint
- különféle készülékek illesztése számítógépekhez
- egyedi készülékek karbantartása, javítása.

VEGYE IGÉNYBE SZOLGÁLTATÁSAINKAT!

IMPULZUS
ELEKTRONIKAI FEJLESZTŐ ÉS SZOLGÁLTATÓ GAZDASÁGI MUNKAKÖZÖSSÉG
1221 Budapest, Leányka u. 32. Telefon: 385-208

215,–Ft