Agárdi Gábor

MBI IBM

Gyakorlati Assembly

Haladóknak

Lemezmelléklettel

Agárdi Gábor

Gyakorlati Assembly haladóknak

Lektorálta: Kaszanyiczki László

Nyitott rendszerű képzés - távoktatás -
oktatási segédlete * Tankönyv

LSI Oktatóközpont
A Mikroelektronika Alkalmazásának
Kúltúrájáért Alapítvány

ISBN 963 577 141 X

Kiadó: **LSI Oktatóközpont**
Felelõs vezetõ: **Dr. Kovács Magda**
Témafelelõs: **Flier István**

Készítette: FÜTI PRINT Kft. 95–32
Felelős vezető: Hegyesi László

Tartalomjegyzék

Előszó avagy Murphy és az optimizmus

Ezen könyv megírására hasonló dolog késztetett, mint elődjére. Célom egy kicsit tovább taglalni azokat a témákat, melyek már említésre kerültek előző könyvemben, azonban ott kevesebb helyet kaptak, illetve további új információkat is szeretnék közreadni. Továbbra sem célom, hogy egyfajta rutingyűjteményt hozzak létre. Annak ellenére, hogy vannak kész, felhasználható rutinok is, inkább csak ötleteket, algoritmusokat szeretnék átadni információ gyanánt. A könyv tartalmilag az **IBM PC Gyakorlati Assembly** című, már megjelent könyvre épül, így a rutinok egyes részei nem dokumentáltak, azok valószínűleg megtalálhatóak az említett kiadványban. Van azonban egy fontos dolog, amit jó ha mindenki figyelembe vesz egy program megírása során, miszerint Murphy igen optimista volt, mivel a hibalehetőségek száma négyzetesen arányos a begépelt karakterek számával, továbbá a számítógép a mi utasításainkat és nem gondolatainkat hajtja végre. Ezért ha a gép nem azt teszi amit mi szeretnénk, ne bántsuk, hanem inkább keressük meg, hol tévedtünk.

Agárdi Gábor

1. Fejezet

A PC (DOS) filekezelése

A könyv első részében már említésre került a PC lemezkezelése, azonban ott részletesebben nem foglalkoztam vele. Ezért megpróbálom egy kicsit bepótolni ezt. Egy file kezelése többnyire a DOS-on keresztül történik így meg kell tanulni annak korlátait leküzdve ésszerűen kezelni a file-okat.

Egy file létrehozása

Azért a lehetőségek nem olyan szűkösek, mint hinnénk. Mit is lehet tenni egy jólnevelt file-lal? Először is induljunk ki abból, hogy üres a lemez. Első lépésben tehát létre kell hozni azt. Ehhez szükség van az állomány leendő nevére, tulajdonságaira és egy programra ami ezekből elkészíti a file-t. A file nevét egészen egyszerűen letároljuk a programban mint egy stringet egy nullával a végén (ez nagyon fontos). Majd **ds:dx** regisztert ezen string címére állítjuk.

A file tulajdonságai már különbözőek lehetnek. Ezt a **cx** regiszter egyes bitjeinek beállításával kell majd megadni a gépnek. Ennek felépítése a következő:

0.bit Read only
1.bit Hidden
2.bit System
9.bit Archív

Tehát a megfelelő bit helyére 1-et írva érhetjük el a kívánt tulajdonságot. Ha **cx**=0 akkor normál attribútumot kap.

Ha mindezt beállítottuk, meg kell hívni a **3Ch** DOS funkciót, és ha minden rendben, a file létrejön. Az **ax** regiszterben megkapjuk a file azonosító kódot, amire a későbbiekben majd szükségünk lesz. Ha valami nem volt tökéletes, a carry flag 1 értéke jelzi és ekkor **ax** regiszterben a hiba kódját kapjuk, amelyek a következők lehetnek:

- **ax**=3 Hibás elérési útvonal
- **ax**=4 A file azonosító létrehozása nem lehetséges (pl.: túl sok nyitott file).
- **ax**=5 Tiltott hozzáférés (pl.: file már létezik és read-only vagy az aldirectory már tele van stb).

A másik kiindulási alap az, hogy már létezik a file. Ekkor is hasonlóan kell eljárnunk, mindössze néhány különbség van csak. Tehát a **ds:dx** regiszter itt is a file nevét tartalmazó memóriarészre mutat, a file megnyitási módját azonban itt az **al** regiszterbe kell írni, ahol az egyes biteknek szintén meg van a maguk jelentése:

File hozzáférési jogok

A 0. és 1. bit a file megnyitási módjára utalnak.

00 csak olvasható file
01 csak írható file
10 írható, olvasható file.

A 2. és 3. bit funkciója ismeretlen, értékük mindig 0.

A 4-7 bit a file-ok osztott elérésével kapcsolatosak, amikor egyidejűleg több program használja az adott állományt. A 7. bit az öröklés beállítására szolgál. Ez szabja meg, hogy az adott állományt használó feladaton belül indított újabb feladat örökölheti-e az állomány nyitottságát. A 4-6 bit az osztott elérésű megnyitás módjára utal:

000 Kompatibilis mód. Az így megnyitott file-t a továbbiakban más alkalmazások is megnyithatják, de csak 000 módban, annak lezárásáig.

001 Kizárólagos mód. Az így megnyitott állományt a továbbiakban semmilyen más alkalmazás nem nyithatja meg annak lezárásáig.

010 Lefoglalás írásra. Az így megnyitott állományt annak lezárásáig nem lehet újra megnyitni se kompatibilis, se írásra lefoglalt módon.

011 Lefoglalás olvasásra. Az így megnyitott állományt annak lezárásáig nem lehet újra megnyitni se kompatibilis, se olvasásra lefoglalt módon.

100 Nem lefoglalt mód. Az így megnyitott állományt a továbbiakban nem lehet megnyitni (annak lezárásáig) kompatibilis módon.

Ha mindezt beállítottuk, a DOS 3Dh funkciójával nyithatjuk meg az állományt.

Közös tulajdonsága a két funkciónak, hogy mind létrehozás, mind megnyitás után "nyitva" lesz az állomány a további műveletek elvégzéséhez. A legnagyobb különbség az, hogy ha létrehozunk egy file-t és már létezett azzal a névvel egy régebbi állomány, akkor annak hosszát 0-ra állítja, azaz mintha törölné. Ezzel szemben a file megnyitása csupán elérhetővé teszi az állományt. A file megnyitására már láthattunk példát a Gyakorlati Assembly könyvben, így most egy file létrehozására mutatok be egy példát.

[Program 01]

```
prog01    segment                  ;Szegmensdefiníció
          assume cs:prog01,ds:prog01 ;Cs és ds regiszterek
                                   ;hozzárendelése.

start:    push  cs                 ;Ds regiszterbe a cs
          pop   ds                 ;értékét tölti

          mov   ax,0b800h          ;A videomemória kezdőcímét
          mov   es,ax              ;es regiszterbe tölti.

          mov   ax,3               ;80*25 karakteres mód beállítása.
          int   10h

          mov   ah,3ch             ;A file létrehozása a filenév
```

```
                xor   cx,cx                ;címkénél tárolt néven,
                mov   dx,offset [filenev]  ;normál attributummal
                int   21h
                mov   bx,ax                ;Visszatéréskor ax regiszter
                                           ;tartalmazza a file azonosítót.

                xor   di,di                ;Az első szöveg kiíratása a
                                           ;bal felső saroktól történik.

                jnc   .4_prog01            ;Ha a c-bit 0, az azt jelenti, hogy
                                           ;a file létrehozása sikerült.

                cmp   ax,3                 ;Ha c=1 akkor a hiba kódját
                                           ;ax regiszter tartalmazza.
                jnz   .1_prog01            ;Ha ez nem 3, ugrik a következőre.

                mov   si,offset hiba_1     ;Az első hibaüzenet kiíratása.
                call  kiiro
                jmp   lezaras

.1_prog01:      cmp   ax,4                 ;Ha a hibakód nem 4, ugrás
                jnz   .2_prog01            ;a következő vizsgálatra.
                mov   si,offset hiba_2     ;Ha igen, a 2. hiba kiíratása.
                call  kiiro
                jmp   lezaras

.2_prog01:      cmp   ax,5
                jnz   .3_prog01
                mov   si,offset hiba_3
                call  kiiro
                jmp   lezaras

.3_prog01:      mov   si,offset hiba_4
                call  kiiro
                jmp   lezaras

.4_prog01:      mov   si,offset keszit_ok  ;Ha ide ugrott a program, azt
                call  kiiro                ;jelenti, a file létre lett hozva.

lezaras:        mov   ah,3eh               ;A file lezárása.
                int   21h

                mov   di,160               ;A második sor első pozíciója.
                jc    .5_prog01            ;Ha c=1 akkor hiba a lezárásnál.
```

```
                mov   si,offset lezar_ok      ;Különben ok.
                call  kiiro
                jmp   vege

.5_prog01:      mov   si,offset hiba_5
                call  kiiro

vege:           mov   ah,4ch                  ;Visszatérés az op. rendszerhez.
                int   21h

kiiro           proc

                mov   ah,15                   ;Az si által megadott címen lévő
.1_kiiro:       mov   al,[si]                 ;szöveg kiíratása di-től kezdve
                inc   si                      ;fényes fehér színnel.
                cmp   al,255                  ;A szöveg végét a 255 kód
                jz    .2_kiiro                ;jelzi.
                mov   es:[di],ax
                add   di,2
                jmp   .1_kiiro
.2_kiiro:       ret

kiiro           endp


hiba_1:         db    "Hibás elérési útvonal.",255
hiba_2:         db    "A file azonosítót nem lehet létrehozni.",255
hiba_3:         db    "Tiltott hozzáférés.",255
hiba_4:         db    "Egyéb file létrehozási hiba.",255
hiba_5:         db    "Hiba a file lezárásánál.",255
keszit_ok:      db    "A file létrehozása sikerült.",255
lezar_ok:       db    "A file lezárása sikerült.",255
filenev:        db    "proba01.aga",0

prog01          ends                          ;A szegmens vége.
                end   start                   ;A program vége.
```

Ez a példaprogram igen egyszerű, és nagyban hasonlít a file megnyitásánál megismertekhez. A különbség itt mindössze annyi, hogy nem a file hozzáférési jogait kell megadnunk, hanem annak attribútumait. A továbbiakban már minden a régi, a **ds:dx** regiszterrel mutatunk egy

filenévre, aminek a végét egy nulla jelzi. Ha a rutin végrehajtása sikeres volt, a carry flag nulla és az **ax** regiszter a file azonosító számot tartalmazza. Ha azonban valami nem volt rendben a file létrehozásánál, a carry flag 1 értéke jelzi és az **ax** regiszter az aktuális hibakódot tartalmazza.

File pointer mozgatása, buborék módszer

A előző könyvemben már esett pár szó adatok sorba rendezéséről. Ennek legegyszerűbb módja az úgynevezett buborék módszer. Lényege, hogy összehasonlítjuk az egymás utáni bejegyzéseket egymással. Tegyük fel, hogy növekvő ABC sorrendbe kívánunk rendezni egy állományt. Ekkor fogunk két egymás utáni adatot, és azoknak karaktereit sorban összehasonlítjuk egymással. Ha olyat találunk, ahol a második bejegyzés adott betűjének kódja alacsonyabb, meg kell cserélnünk a két rekordot egymással. Ezután jöhet a 2. és 3. majd a 3. és 4. adat, egészen az *utolsó-1* és az *utolsó* adatokig. Ha végigértünk a soron, elölről kell kezdeni mindezt és a *bejegyzések száma-1*-szer végre kell hajtani ezt a folyamatot. Nagy hátránya a dolognak, hogy a rendezési idő a rekordok számával négyzetesen nő, így hosszabb file-oknál nem célszerű használni.

Erre a feladatra kívánok bemutatni egy ötletet a 2. példa programban. Azzal a kis furcsasággal, hogy a program nem tölti be az egész állományt a memóriába, hanem mindig csak az aktuális 2 rekordot. Ez úgy lehetséges, hogy létezik a file-oknál egy úgynevezett file pointer amely arra a helyre mutat, ahová a következő írás illetve ahonnan a következő olvasás történni fog. Ezt lehetőség van programból elállítani oda ahová csak szeretnénk. Megadhatjuk, hogy az állítás mihez képest történjen. Lehet a program elejéhez, végéhez és az aktuális pozícióhoz képest is.

[Program 02]

```
prog02    segment                    ;Szegmensdefiníció
          assume cs:prog02, ds:prog02 ;Cs és ds regiszterek
                                      ;hozzárendelése.

start:    push  cs                   ;A cs regiszter értékét
          pop   ds                   ;áttölti a ds-be
```

```
        mov   ax,3d02h              ;A filenév címke alatt letárolt
        mov   dx,offset filenev     ;nevű file megnyitása írási
        int   21h                   ;és olvasási jogokkal.
        mov   word ptr [handle],ax  ;A file azonosító számot a
                                    ;handle változóba teszi.

        mov   cx,word ptr [bejegyzesek]
        dec   cx                    ;A programot a bejegyzések
                                    ;száma-1-szer kell lefuttatni.

.1_prog02: push  cx
        mov   cx,word ptr [bejegyzesek]
        dec   cx                    ;Hogy a teljes adatbázison
                                    ;végrehajtsuk a cserélgetést,
                                    ;a következő rutint szintén
                                    ;a bejegyzések száma-1-szer
                                    ;kell végrehajtani.

.2_prog02: push  cx
        mov   ax,4200h              ;A file mutató beállítása
        mov   bx,word ptr [handle]  ;A pointer által mutatott
        mov   cx,word ptr [pointer+2] ;címre (a file elejétől
        mov   dx,word ptr [pointer] ;számítva).
        int   21h

        mov   ah,3fh                ;A már megnyitott file-ból
        mov   cx,szohossz           ;2*szóhossz blokkot olvas
        shl   cx,1                  ;be és tárolja a puffer
        mov   dx,offset puffer      ;címkétől kezdődően.
        mov   bx,word ptr [handle]
        int   21h

        mov   si,offset puffer
        mov   bx,word ptr [szohossz]
        mov   cx,bx
.3_prog02: mov   al,[si+bx]         ;A két szóhossz hosszúságú
        cmp   al,[si]               ;adat karaktereit összehasonlítja
        jc    csere                 ;egymással, és ha olyat talál,
        inc   si                    ;ahol a második adat kisebb,
        loop  .3_prog02             ;ugrik a csere rutinra.
        jmp   .4_prog02             ;Ha nem volt ilyen, ugrik a
                                    ;következő adat beállítására.
```

```
jump:       jmp   .1_prog02              ;Segédugrás a loop-hoz.

csere:      mov   cx,bx                  ;A két rekord megcserélése
            mov   si,offset puffer       ;a memóriában.
.1_csere:   mov   al,[si]
            xchg  al,[si+bx]
            mov   [si],al
            inc   si
            loop  .1_csere

            mov   ax,4200h               ;A pointer beállítása.
            mov   bx,word ptr [handle]
            mov   cx,word ptr [pointer+2]
            mov   dx,word ptr [pointer]
            int   21h

            mov   ah,40h                 ;A puffer címkétől kezdődően
            mov   bx,word ptr [handle]   ;letárolt 2*szóhossz hosszú
            mov   cx,word ptr [szohossz] ;blokk visszaírása a file-ba.
            shl   cx,1
            mov   dx,offset puffer
            int   21h

.4_prog02:  pop   cx
            mov   bx,word ptr [szohossz] ;A pointert egy rekorddal
            add   word ptr [pointer],bx  ;arrébb állítja.
            adc   word ptr [pointer+2],0
            loop  .2_prog02

            mov   word ptr [pointer],0   ;A pointer visszaállítása a
            mov   word ptr [pointer+2],0 ;file elejére.
            pop   cx
            loop  jump

            mov   bx,word ptr [handle]   ;A file lezárása.
            mov   ah,3eh
            int   21h


            mov   ah,4ch                 ;Visszatérés a DOS-hoz.
            int   21h

pointer:    dd    0
filenev:    db    "datatext.aga",0
```

```
handle:     dw   0
bejegyzesek: dw  10
szohossz    dw   16
puffer:     db   ?

prog02      ends                    ;A programszegmens vége.
            end  start              ;A program vége.
```

A program egy írás-olvasásra történő file megnyitással kezdődik. Ez szükséges ahhoz, hogy a továbbiakban bármilyen műveletet végezzünk a file adataival. A pointer mozgatására a 42h DOS funkció szolgál, ahol az **ah** regiszter tartalmazza a 42h DOS funkció számát, az **al** a viszonyítási kódot, a **bx** a file azonosító számot és a **cx:dx** a relatív eltolás mértékét byte-ban. Visszatéréskor ha hiba történt (amit a carry jelez), annak kódját az **ax** regiszter tartalmazza. A file mutató abszolút helyzetét a **dx:ax** regiszterben kapjuk vissza sikeres végrehajtás esetén.

A viszonyítási kódok a következők lehetnek:

00h A file kezdetéhez viszonyítva.
01h A file mutató aktuális helyzetéhez viszonyítva.
02h A file végéhez viszonyítva.

Ezen módszerrel szóhosszonként léptetjük majd a file-pointert előre a file-ban. Így az összes rekord kiolvasható akár egyenként is, itt azonban egyszerre kettőt olvasunk be a memóriába, majd ezeket karakterenként összehasonlítjuk egymással, és az eredménytől függően ha kell, megcseréljük a két rekordot, és visszaírjuk a lemezre. Amint említettem, hosszabb adatbázisok esetén ez egy elég lassú módszer. Erre elég meggyőző az az adat, hogy az itt ismertetett program le lett futtatva egy 1000 rekordos adatbázison egy 16ms-es winchesterrel felfegyverzett 386-oson, és a teljes rendezés megközelítőleg 17 percet vett igénybe. Ezen egy kicsit segíthetünk úgy, hogy az egész adatbázist betöltjük a memóriába, és ott végezzük el a rendezést, de érdemesebb inkább más fajta rendezési módszerhez folyamodni.

File attribútumok kezelése

Említésre került már, hogy a file-oknak különböző tulajdonságokat adhatunk létrehozáskor. Lehetőség van azonban ennek lekérdezésére és megváltoztatására is pl.: Norton Commanderrel. Itt azonban az assembly szintű variálgatási lehetőségeket szeretném bemutatni. Erre a DOS-nak szintén van egy funkciója, mellyel mindezt könnyen megtehetjük. Ez nem más, mint a 43h funkció, amely mind az olvasási mind az írási műveleteket elvégzi. Paraméterezése a következő:

Lekérdezésnél

ax	4300h
ds:dx	A file nevét tartalmazó memóriacímre mutat.

Visszatérés:

carry	Hiba esetén 1, különben nulla.
ax	Hiba esetén a hiba kódja.
cl	A file tulajdonságai (attribútumait)
ch	00h

Beállításnál

ax	4301h
cl	A file beállítandó tulajdonsága
ch	00h
ds:dx	A file nevét tartalmazó memóriarekeszre mutat.

Visszatérés:

carry	Hiba esetén 1, különben nulla.
ax	Hiba esetén a hiba kódja.

Az attribútum byte felépítése

0. bit	Csak olvasható
1. bit	Rejtett file
2. bit	Rendszerfile

3. bit	Lemeznév
4. bit	Alkönyvtár
5. bit	Archív file
6. bit	?
7. bit	Hálózati file

Ezzel a funkcióval a 3., 4., 6. és 7. bit nem változtathatóak meg. A tulajdonságok lekérdezésére mutat egy példát a 3. és megváltoztatására a 4. program.

[Program 03]

```
prog03      segment                 ;Szegmensdefiníció.
            assume cs:prog03,ds:prog03  ;Cs és ds regiszterek
                                    ;hozzárendelése a programhoz.

start:      push  cs                ;Cs értékének áttöltése a
            pop   ds                ;ds regiszterbe.

            mov   ax,0b800h         ;A videomemória kezdőcímét az
            mov   es,ax             ;es regiszterbe tölti.
            xor   di,di

            mov   ax,3              ;80*25 karakteres mód.
            int   10h

            mov   si,offset text    ;A text cimke alatt tárolt
            mov   ah,15             ;szöveg kiíratása.
            xor   bx,bx
.1_prog03:  mov   al,[si]
            inc   si
            cmp   al,0              ;A szöveg végének figyelése.
            jz    .3_prog03
            cmp   al,13             ;Enter figyelése.
            jnz   .2_prog03
            add   di,160
            xor   bx,bx
            jmp   .1_prog03
.2_prog03:  mov   es:[di+bx],ax
            add   bx,2
            jmp   .1_prog03
```

```
.3_prog03:  mov  ax,4300h          ;A filenev nevű file
            mov  dx,offset filenev ;attribútumának lekérdezése.
            int  21h

            mov  bl,cl             ;Ezt az értéket áttöltjük
                                   ;a bl regiszterbe.

            mov  cx,8              ;8 bit.
            mov  ah,15             ;Világos fehér szín.
            mov  di,42             ;A 20. pozíció címe.

.4_prog03:  mov  al,"0"            ;Az Al regiszterb tölti a "0" karater
                                   ;kódját tölti.

            shr  bl,1              ;Jobbra léptetésnél, ha
            jnc  .5_prog03         ;a kilépő bit értéke, 1 akkor
            mov  al,"1"            ;az al regiszter tartalmát
                                   ;átírja az "1" kódjára,
                                   ;ha a kilépő bit értéke nulla, akkor
                                   átugorja ezt.

.5_prog03:  mov  es:[di],ax        ;Az elkészült érték kiírása.
            add  di,160            ;Új sor.
            loop .4_prog03

            xor  ax,ax             ;Billentyűvárás.
            int  16h

            mov  ax,4c00h          ;Visszatérés a DOS-hoz.
            int  21h

filenev:    db   "proba01.aga",0

text:       db   "Csak olvasható file:",13
            db   "Rejtett file:",13
            db   "Rendszerfile:",13
            db   "Lemezcimke:",13
            db   "Alkönyvtár:",13
            db   "Archiv file:",13
            db   "?:",13
            db   "Hálózati file:",0

prog03      ends                   ;A szegmens vége.
            end  start             ;A program vége.
```

[Program 04]

```
prog04    segment                    ;Szegmensdefiníció.
          assume cs:prog04,ds:prog04 ;Cs és ds regiszterek
                                     ;hozzárendelése a programhoz.

start:    push  cs                   ;A cs regiszter értékét a
          pop   ds                   ;vermen keresztül beírjuk
                                     ;a ds regiszterbe.

          mov   ax,4301h             ;A proba01.aga file
          mov   cx,00100111b         ;attributúmainak beállítása
          mov   dx,offset filenev    ;a cl regiszternek megfelelően.
          int   21h

          mov   ax,4c00h             ;Visszatérés a DOS-hoz.
          int   21h

filenev:  db    "proba01.aga",0

prog04    ends                       ;A szegmens vége.
          end   start                ;A program vége.
```

A lekérdező program egy már ismert szöveg kiíratórutinnal kezdődik, ami kiírja a lehetséges attribútumokat a képernyőre. Ezután következik a file tulajdonságainak lekérdezése, majd a bináris szám kiíratásnál megismert módon az egyes értékek kiíratása.

A beállító programban semmi extra dolog nincs, mindössze a DOS funkciót paraméterező és meghívó rutin.

File átnevezése

Vannak még további lehetőségei is a file-okkal való manipulációnak. Ezek közül az egyik a file átnevezése. Ez a DOS-ból már bizonyára jól ismert *rename* parancs. Ennek az assembly megfelelője a DOS 56h funkciója. Segítségével lehetőség van egy file átnevezésére illetve átmozgatására az adott lemezegységen belül bárhová. Feladatunk csupán

annyi, hogy megadjuk a régi illetve az új filenevet és ha szükséges, a hozzá tartozó elérési útvonalakat.

[Program 05]

```
prog05      segment                    ;Szegmensdefiníció.
            assume cs:prog05,ds:prog05 ;Cs és ds regiszterek
                                       ;hozzárendelése a kódhoz.

start:      mov   ax,cs                ;A cs regiszter értékét az
            mov   ds,ax                ;ax regiszteren keresztül
            mov   es,ax                ;beírja a ds, es regiszterekbe.

            mov   dx,offset reginev    ;A funkció paramétereinek
            mov   di,offset ujnev      ;beállítása és a funkció
            mov   ah,56h               ;meghívása.
            int   21h

            jnc   vege                 ;Ha a végrehajtás sikeres
                                       ;volt, ugrik a végére.

            mov   ax,0b800h            ;Ha a végrehalytás sikertelen,.
            mov   es,ax                ;a hibaüzenet kiírása történik.
            mov   ax,3
            int   10h
            xor   di,di
            mov   si,offset text
            mov   ah,15
.1_prog05:  mov   al,[si]
            inc   si
            cmp   al,0
            jz    vege
            mov   es:[di],ax
            add   di,2
            jmp   .1_prog05

vege:       mov   ah,4ch
            int   21h

text:       db    "A módosítás nem sikerült.",0
reginev:    db    "proba01.aga",0
ujnev:      db    "proba02.aga",0

prog05      ends                       ;A szegmens vége.
```

```
        end   start                    ;A program vége.
```

Amint az látható, a **ds:dx** regiszterben kell megadni a file régi nevét és **es:di** címen az új nevet. A hibát itt is a carry flag jelzi.

File törlése

Ha már nincs szükségünk egy file-ra, a 41h funkció segítségével törölhetjük azt. A **ds:dx** címen elhelyezett, **nullával lezárt** *filenév* által megjelölt file kerül majd törlésre a funkció meghívásakor. Joker karakterek használata csak akkor megengedett, ha az 5Dh funkción keresztül hajtottuk végre ezt.

[Program 06]

```
prog06      segment                    ;Szegmensdefiníció.
            assume cs:prog06,ds:prog06 ;Cs és ds regiszterek
                                       ;hozzárendelése a kódhoz.

start:      mov   ax,cs                ;A cs regiszter értékét az
            mov   ds,ax                ;ax regiszteren keresztül
                                       ;beírja a ds regiszterbe.

            mov   dx,offset filenev    ;A funkció paramétereinek
            mov   ah,41h               ;beállítása és a funkció
            int   21h                  ;meghívása.

            jnc   vege                 ;Ha a végrehajtás sikeres
                                       ;volt, ugrik a program végére.

            mov   ax,0b800h            ;Ha nem, hibaüzenet kiíratása.
            mov   es,ax
            mov   ax,3
            int   10h
            xor   di,di
            mov   si,offset text
            mov   ah,15
.1_prog06:  mov   al,[si]
            inc   si
            cmp   al,0
            jz    vege
```

```
                mov   cs:[di],ax
                add   di,2
                jmp   .1_prog06

vege:           mov   ah,4ch
                int   21h

text:           db    "A törlés nem sikerült.",0
filenev:        db    "proba01.aga",0

prog06          ends                          ;A szegmens vége.
                end   start                   ;A program vége.
```

File keresése

A DOS lehetőséget ad különböző szempontok szerinti file-keresésre a 4Eh illetve 4Fh funkciók segítségével. A keresésnél meg kell adni a keresendő file nevét, ami tartalmazhat * és ? karaktereket is. Így lehetőség nyílik akár directory készítésére is a *.* keresési direktívával. Továbbá meg lehet adni a keresendő file attribútumát is a **cl** regiszterbe. Ha ez 0, akkor csak a normál attribútumú file-ok között fog keresni (az archív bitet figyelmen kívül hagyja), de ha például beállítjuk a **system** bitet, akkor a normál file-ok mellett a **system** file-ok között is fog keresni. 08h jellemzőt megadva a lemezcimkét kereshetjük meg.

A funkció használata során először a 4Eh rutint kell végrehajtatni, ami megkeresi az első olyan file-t, ami megfelel a feltételnek. Ezután tetszőlegesen ismételhetjük a 4Fh funkciót, ami a következő file megkeresésére szolgál. Ha a gép nem talált több bejegyzést (vagy egyáltalán nem talált) akkor ezt a carry flag jelzi nekünk. Ha talált, annak adatai egy átmeneti tárolóba kerülnek (DTA), melynek címét a DOS 2Fh funkciója segítségével kérdezhetjük le.

A DTA két részből áll. Az első 21 byte-ból, ami DOS verzió függő felépítésű, és a további kereséshez szükséges információkat tartalmazza, valamint az e fölötti területből, ami a számunkra értékes információkat tartalmazza, és felépítése állandó.

A DTA felépítése

Cím	Méret (Byte)	Funkció
00h	14h	A további kereséshez szükséges információk.
15h	01h	A megtalált file attribútuma.
16h	02h	Az állomány létlehozásának ideje:
		0-4. bit másodperc/2
		5-10. bit perc
		11-15. bit óra
18h	02h	A létrehozás dátuma
		0-4. bit nap
		5-8. bit hónap
		9-15. bit év-1980
1Ah	04h	A megtalált file mérete
1Eh	0Dh	A megtalált file neve 00h kóddal a végén.

A keresés folytatásánál nem szükséges semmilyen bemenő paramétert megadni mivel a szükséges információkat már tartalmazza az a bizonyos 21 byte.

[Program 07]

```
prog07    segment                    ;Szegmensdefiníció.
          assume cs:prog07,ds:prog07 ;cs és ds regiszterek
                                     ;hozzárendelése a kódhoz.

start:    mov  ax,cs                 ;A cs regiszter értékét az
          mov  ds,ax                 ;ax regiszteren keresztül
                                     ;beírja a ds regiszterbe.

          xor  di,di                 ;Képernyőtörlés és az írási
          mov  ax,3                  ;pozíció beállítása.
          int  10h

          mov  ah,4eh                ;Az első bejegyzés keresése.
          mov  cx,0
          mov  dx,offset filenev
          int  21h

          jc   kiiro1                ;Ha nem talált bejegyzést, ugrás
                                     ;a kiíró1 rutinra.
```

```
                call    dta             ;Ha talált, akkor ezen két rutin
                call    kiiro2          ;segitségével kiirja azt.

.1_prog07:      mov     ah,4fh          ;A következő bejegyzés
                int     21h             ;keresése.

                jc      kiiro1          ;Ha nincs több, ugrás.

                call    dta             ;Egyébként a talált file
                call    kiiro2          ;nevének kiíratása.

                jmp     .1_prog07       ;A következő file keresése.

kiiro1:         mov     ax,0b800h       ;A nincs több file üzenet
                mov     es,ax           ;kiíratása.
                mov     si,offset text
                mov     ah,15
.1_kiiro1:      mov     al,[si]
                inc     si
                cmp     al,0
                jz      vege
                mov     es:[di],ax
                add     di,2
                jmp     .1_kiiro1

vege:           xor     ax,ax           ;Billentyűvárás.
                int     16h

                mov     ah,4ch          ;Visszatérés a DOS-hoz.
                int     21h

dta             proc

                mov     ah,2fh          ;A DTA címének lekérdezése.
                int     21h
                mov     bp,offset dtatext
                mov     cx,12           ;És a maximum 12 byte hosszú
                add     bx,1eh          ;filenév átmásolása a dtatext
.1_dta:         mov     al,es:[bx]      ;címkétől kezdődően.
                mov     [bp],al
                inc     bx
                inc     bp
                loop    .1_dta
```

```
            ret

dta         endp

kiiro2      proc

            mov   si,offset dtatext      ;A filenév kiíratása.
            mov   cx,12
            mov   ax,0b800h
            mov   es,ax
            mov   ah,15
            xor   bx,bx
.1_kiiro2:  mov   al,[si]
            inc   si
            cmp   al,0                   ;A filenév végét egy nulla
            jz    .2_kiiro2              ;kód jelzi.
            mov   es:[di+bx],ax
            add   bx,2
            loop  .1_kiiro2
.2_kiiro2:  add   di,160                 ;A következő nevet új sorba
            ret                          ;kezdi.

kiiro2      endp

filenev:    db    "*.aga",0
text:       db    "Nincs több file.",0
dtatext:    db    ?

prog07      ends                         ;A szegmens vége.
            end   start                  ;A program vége.
```

2. Fejezet

A DOS Könyvtárkezelése

Ebben a fejezetben a lemezkezelés egy másik fontos alkotóeleméről lesz szó, a könyvtárak kezeléséről. Hasonlóképpen mint a file-oknál, itt is létre lehet őket hozni, törölni lehet, illetve egyéb más műveleteket végezhetünk velük. Ezek közül mutatok most be egy párat, amikre elég gyakran lehet szükség.

Egy könyvtár létrehozása

Az első ilyen példa egy könyvtár létrehozása, amihez a könyvtár nevére és helyére van szükségünk. Egyébként a funkció megegyezik a DOS MD parancsával. A 21h megszakítás 39h rutinja végzi ezt a műveletet oly módon, hogy a **ds:dx** regiszterekben megadjuk a filenévhez hasonló módon a könyvtár elérési útját és nevét, aminek végét egy nulla kód jelzi. Hatására a gép létrehozza a bejegyzést, vagy ha nem sikerült, azt a carry flag beállított értéke jelzi.

[Program 08]

```
prog08    segment                    ;Szegmensdefiníció.
          assume cs:prog08,ds:prog08 ;Szegmensregiszterek
                                     ;hozzárendelése a kódhoz.

start:    push  cs                   ;Cs regiszter áttöltése a
          pop   ds                   ;vermen keresztül ds-be.

          mov   ah,39h               ;A dirnév címke alatt tárolt
          mov   dx,offset dirnev     ;könyvtár létrehozása.
          int   21h

          jnc   vege                 ;Ha nincs hiba akkor kilépés.

          mov   ax,3                 ;Hibaüzenet kiírása.
          int   10h
          mov   ax,0b800h
          mov   es,ax
          mov   si,offset text
```

```
            mov   ah,15
kiiro:      mov   al,[si]
            inc   si
            cmp   al,0
            jz    vege
            mov   es:[di],ax
            add   di,2
            jmp   kiiro

vege:       mov   ah,4ch
            int   21h


dirnev:     db    "c:\konyvtar",0          ;A létrehozandó könyvtár
                                           ;neve a végén egy nullával.

text:       db    "A könyvtár létrehozása nem lehetséges",0

prog08      ends                           ;Szegmens vége.
            end   start                    ;Program vége.
```

Egy könyvtár törlése

Egy tartalomjegyzék törlése teljesen megegyezik az előző művelettel, annyi különbséggel hogy nem a 39h hanem a 3Ah funkciót kell elindítani a művelet végrehajtásához. Ügyelni kell arra, hogy csak üres könyvtárat lehet törölni. Ha a kijelölt directory nem üres, a gép hibajelzéssel tér vissza.

[Program 09]

```
prog09      segment                        ;Szegmensdefiníció.
            assume cs:prog09,ds:prog09     ;Cs és ds regiszterek
                                           ;hozzárendelése a kódhoz.

start:      push  cs                       ;Cs regiszter áttöltése ds-be.
            pop   ds

            mov   ah,3Ah                   ;A directory törlése.
            mov   dx,offset dirnev
            int   21h

            jnc   vege                     ;Ha nincs hiba, ugrás a végére.
```

```
            mov   ax,3                  ;Hibaüzenet kiírása.
            int   10h
            mov   ax,0b800h
            mov   es,ax
            mov   si,offset text
            mov   ah,15
kiiro:      mov   al,[si]
            inc   si
            cmp   al,0
            jz    vege
            mov   es:[di],ax
            add   di,2
            jmp   kiiro

vege:       mov   ah,4ch
            int   21h


dirnev:     db    "c:\konyvtar",0       ;A törlendő könyvtár neve.
text:       db    "A könyvtár törlése nem lehetséges",0

prog09      ends                        ;A szegmens vége.
            end   start                 ;A program vége.
```

Aktuális pozíció meghatározása

A DOS-ban a pg prompt beállításnál a számítógép kiírja a sor elejére az aktuális lemezegység betűjelét és az aktuális könyvtár elérési útvonalát. Igen ám, de mi van akkor ha erre az információra egy programon belül van szükségünk (a gép nem nézi a képernyőt és nem kotorászik a gondolatainkban). Az egyetlen dolog amit tehetünk ilyenkor, hogy megkérdezzük a géptől aki mindent tud.

Az aktuális lemezegység lekérdezése valamivel egyszerűbb. A DOS 19h megszakítását elindítva, az **al** regiszterbe visszakapjuk az adott meghajtó sorszámát (A=0, B=1, C=2 stb.). A programban ehhez az információhoz hozzáadjuk az 'A' betű kódját, mire a 0 kódból 'A' betű lesz amit már ki tudunk írni a képernyőre. A directory lekérdezése valamivel összetettebb, ott meg kell adnunk **dl** regiszterben, hogy mely meghajtó

aktuális könyvtárára van szükségünk, ahol a 0 kód az aktuális és az 1 az 'A' meghajtó kódja. Továbbá meg kell még adnunk a **ds:si** regiszterekben egy 64 byte hosszú lefoglalt memóriarekesz címét, ahová a gép elhelyezi a könyvtár útvonalat a meghajtónév és a főkönyvtár jelölése nélkül.

[Program 10]

```
prog10    segment                    ;Szegmensdefiníció.
          assume cs:prog10,ds:prog10 ;Cs és ds regiszterek
                                     ;hozzárendelése a kódhoz.

start:    push  cs                   ;Cs szegmensregiszter áttöltése
          pop   ds                   ;a ds-be a vermen keresztül.

          mov   ax,0b800h            ;A videomemória kezdőcímét
          mov   es,ax                ;az es regiszterbe tölti,
          mov   ax,3                 ;Törli a képernyőt, illetve
          int   10h                  ;beállítja a 80*25 karakteres
          xor   di,di                ;képernyőn a bal felső sarok
                                     ;pozícióját a di regiszterbe.

          mov   si,offset text1      ;Az első szöveg kirakása.
          call  kiiro

          push  di

          mov   ah,19h               ;Az aktuális drive lekérdezése.
          int   21h

          pop   di                   ;A drive számához hozzáadjuk
          add   al,"A"               ;az "A" betű kódját, amit utána
          mov   ah,15                ;kirakunk az első szöveg
          mov   es:[di],ax           ;után.

          mov   di,160               ;A második sor elejétől kezdve
          mov   si,offset text2      ;kiírja a második üzenetet.
          call  kiiro
          push  di

          mov   ah,47h               ;Az aktuális könyvtár nevének
          xor   dl,dl                ;lekérdezése
          mov   si,offset directory
          int   21h
```

```
            pop   di                        ;és kiíratása a második szöveg
            mov   si,offset directory       ;után.
            call  kiiro

vege:       mov   ah,4ch                    ;Visszatérés a DOS-hoz.
            int   21h

kiiro       proc

            mov   ah,15                     ;Az si címen kezdődő, nullával
.1_kiiro:   mov   al,[si]                   ;lezárt szöveg kiíratása a di által
            inc   si                        ;mutatott címtől kezdődően.
            cmp   al,0
            jz    .2_kiiro
            mov   es:[di],ax
            add   di,2
            jmp   .1_kiiro
.2_kiiro:   ret

kiiro       endp

text1:      db    "Az aktuális drive : ",0
text2:      db    "Az aktuális directory : ",0
directory:  db    64 dup (0)

prog10      ends                            ;A szegmens vége
            end   start                     ;A program vége.
```

Nagyon sok dolgot művelhetünk még a könyvtárakkal és file-okkal, de ezek már kevésbé fontosak, illetve az eddig megismertek alapján bárki képes megírni az általa kitalált művelet elvégzésére szolgáló rutint. Ebben nagy segítséget nyújtanak a már megjelent könyvek illetve segédprogramok.

3. Fejezet

Batches rutinok

Továbbra is a DOS témakörénél maradva szeretnék egy pár apró ötletet, rutint adni azoknak akik adnak arra, hogy hogyan néz ki az álltaluk megírt batch program.

Scroll rutin

Talán a legfontosabb egy jó batch programnál (szerintem), hogy ne hasonlítson semmi olyanra, amiből már telített a piac. Tehát találjunk ki valami újat, érdekeset. Ennek nem feltétlenül kell nagyon bonyolultnak lennie, elég ha látványos mint például a 11. példaprogram, ami kiválóan alkalmas szerzői üzenetek illetve bármilyen rövid információ nem általános módon (echo) történő megjelenítésére.

[Program 11]

```
prog11    Segment                    ;Szegmensdefiníció
          assume cs:prog11,ds:nothing ;A szegmensregiszterek
                                      ;hozzárendelése a kódhoz.

sor       equ  0                     ;A szöveg helye a képen.
szin      equ  15                    ;A szöveg színe.

start:    xor  al,al                 ;Megvizsgálja, hogy van-e
          cmp  cs:[128],al           ;a parancsnak paraméterrésze.

          jz   Vege                  ;Ha nincs, ugrik a végére.

          mov  si,129                ;A szöveg kezdőcíme.
          mov  ax,0b800h             ;A videomemória szegmenscímét
          mov  es,ax                 ;az es regiszterbe tölti.

          mov  al,160                ;Kiszámolja a megadott sor
          mov  bl,sor                ;utolsó karakterének a címét.
          mul  bl
          add  ax,158
          mov  di,ax
```

```
                mov   ah,szin           ;Ah regiszterben tárolja a
                                        ;kiírandó szöveg színét.

.1_prog11:      xor   bx,bx             ;A mutató nullázása.

.2_prog11:      mov   al,[si+bx]        ;Kiolvassa az adott karaktert.
                cmp   al,13             ;Ha az enter, akkor elölről
                jz    .1_prog11         ;kezdi a kiíratást.
                mov   es:[di],ax        ;Ha nem, ki lehet írni a
                inc   bx                ;képernyőre az adott karaktert.

                mov   ax,100h           ;Figyeli, van-e lenyomott
                int   16h               ;billentyű.
                jnz   Vege              ;Ha igen, akkor kilépés.

                mov   cx,20
                mov   dx,3dah           ;Vertical Blank időzítés.
.3_prog11:      in    al,dx
                test  al,8
                jnz   .4_prog11
.4_prog11:      in    al,dx
                test  al,8
                jz    .4_prog11
                loop  .3_prog11

                mov   cl,79             ;A kijelölt sort egy
                sub   di,158            ;karakterrel balra tolja.
.5_prog11:      mov   ax,es:[di+2]
                mov   es:[di],ax
                add   di,2
                loop  .5_prog11
                jmp   .2_prog11

Vege:           mov   ax,4c00h          ;Kilépés a DOS-hoz.
                int   21h

prog11          ends                    ;A szegmens vége.
                end   start             ;A program vége.
```

A program elindulásakor megvizsgálja, hogy gépeltünk-e az utasítás után egyéb szöveget is vagy sem. Ennek ellenőrzésekor kihasználja a pc azon tulajdonságát, hogy egy .exe program elindulásakor **ds** és **es**

regiszterek egyaránt a psp kezdőcímére mutatnak. Ha nem, akkor a program futtatása értelmetlen, így ugrik a befejezésre. Ha van, akkor **si** regiszterbe tölti a szöveg kezdőcímét, valamint elvégzi a megfelelő paraméter beállításokat, mint a videomemória kezdőcíme stb. A képernyőcím számításakor a megadott sor utolsó karakterének címét számolja, mivel innen fog indulni a szöveg.

A szöveg kiíratása karakterenként történik, ügyelve az *enter* kódra. Minden betű kirakása után egy billentyűfigyelés és egy időzítés következik. Az időzítéshez a függőleges visszafutási időt használja ki, mivel ez körülbelül azonos minden gépnél, és használatával még a szöveg olvashatósága is javul. Ezt követően mozgatjuk egy karakterrel arrébb a kijelölt sort majd vissza az elejére.

A program használata igen egyszerű, a batch programunkba mindössze annyit kell beírni, hogy ***prog11 futtatandó szöveg*** és ennek végrehajtásakor, mindaddig míg le nem ütünk egy billentyűt, a futtatandó szöveg ott fog mászkálni a megadott sorban.

Grafika a szöveges képernyőn ?

Ezen rutin megírásához egy elég komoly multimédiás kártya adta az ötletet, ami egyébként a 94-es compfairen is megtekinthető volt. A történet lényege, hogy a kártyához volt egy pár demo CD, köztük például a teljes Top Gun film két darab CD-n stb. Volt azonban egy rövid animáció, amit ha DOS környezetből elindítottunk karakteres képernyőn, kinnmaradt minden szöveg, de egy pörgő, forgó animáció került a képre mindenféle hanghatás kíséretében. Amin én megdöbbentem, - annak ellenére, hogy a látvány sem volt elhanyagolandó - az volt, hogy hogyan rajzolnak a karakteres képernyőre grafikusan. Nos ez ott hardware-es úton volt megoldva, de elkezdtem gondolkodni, hogy hogyan lehetne ezt a trükköt software-es úton megvalósítani? A megoldás igen kézenfekvő volt, mivel karakteres képernyőre nem lehet rajzolni így a képernyőn lévő szöveget kell átvinni a grafikus felületre, amire ezután tetszőlegesen firkálhatunk. Erre a trükkre mutat egy példát a 12. program.

[Program 12]

```
prog12      segment                 ;Szegmensdefiníció.
            assume cs:prog12,ds:prog12  ;Szegmensregiszterek
                                    ;hozzárendelése a kódhoz.

start:      push  cs                ;Cs áttöltése ds-be a vermen
            pop   ds                ;keresztül.

            mov   ax,0b800h         ;A szöveges képernyő címét
            mov   es,ax             ;es regiszterbe teszi,
            xor   si,si             ;és beállítja a másolási
            mov   di,offset hely    ;paramétereket.
            mov   cx,80*25

.1_prog12:  mov   al,es:[si]        ;Minden karakterkód átmásolása
            mov   [di],al           ;a megadott helyre.
            add   si,2
            inc   di
            loop  .1_prog12

            mov   ax,0fh            ;640*350/2 üzemmód beállítása.
            int   10h

            mov   ax,1122h          ;8*14 pontos betükészlet
            int   10h               ;betöltése a memóriába.

            push  ds                ;Ds regiszter áttöltése es-be.
            pop   es

            mov   bx,1              ;A 0. lapra normál attribútummal
            mov   dx,0              ;a 0,0 ponttól kezdődően kiírja
            mov   cx,80*25          ;a 80*25 karakter méretű
            mov   bp,offset hely    ;hely cimkétől letárolt
            mov   ax,1300h          ;szöveget.
            int   10h

            mov   ax,0a000h         ;Az ega képernyő szegmenscíme.
            mov   es,ax

            mov   dx,3c4h           ;A 0. planc engedélyezése
            mov   ax,0102h
            out   dx,ax

            mov   cx,20             ;Az alakzat kirajzolása.
.2_prog12:  push  cx
```

```
            call    linc
            add     word ptr [xa],10
            add     word ptr [yb],5
            pop     cx
            loop    .2_prog12

            mov     cx,20
.3_prog12:  push    cx
            call    linc
            add     word ptr [xa],10
            sub     word ptr [yb],5
            pop     cx
            loop    .3_prog12

            mov     cx,20
.4_prog12:  push    cx
            call    linc
            sub     word ptr [xa],10
            sub     word ptr [yb],5
            pop     cx
            loop    .4_prog12

            mov     cx,20
.5_prog12:  push    cx
            call    linc
            sub     word ptr [xa],10
            add     word ptr [yb],5
            pop     cx
            loop    .5_prog12

            xor     ax,ax                   ;Billentyűvárás.
            int     16h

            mov     ax,3                    ;Képernyőtörlés és a
            int     10h                     ;80*25 karakteres mód
                                            ;visszaállítása.
            mov     ah,4ch                  ;Visszatérés a DOS-hoz.
            int     21h


linc        proc                            ;A vonalhúzó rutin kezdete

            mov     bx,word ptr [xa]        ;A vonal kezdőpontjának és
            mov     ax,word ptr [xb]        ;végének x koordinátáját
```

```
                                    ;beírja ax, bx regiszterekbe.

        mov   dx,1                  ;A vízszintes lépésköz beállítása.

        sub   ax,bx                 ;Kiszámítja a két pont közötti
                                    ;távolságot.

        jnc   .1_line               ;Ha ez nem negatív, átugorja
                                    ;a következő részt.

        neg   ax                    ;A távolság valódi értékének a
                                    ;kiszámítása.

        mov   dx,65535              ;A vízszintes lépésközt -1-re
                                    ;állítja.

.1_line: mov  word ptr [delta_x],ax ;A regisztereket beírja a
        mov   word ptr [pont_x],bx  ;memóriaváltozókba.
        mov   word ptr [sgn_x],dx

        mov   bx,word ptr [ya]      ;Ugyanaz, ami a vízszintes
        mov   ax,word ptr [yb]      ;értékeknél történt.
        mov   dx,1
        sub   ax,bx
        jnc   .2_line
        neg   ax
        mov   dx,65535

.2_line: mov  word ptr [delta_y],ax
        mov   word ptr [pont_y],bx
        mov   word ptr [sgn_y],dx

        cmp   ax,word ptr [delta_x] ;Eldönti, hogy az adott vonal milyen
                                    ;irányú.

        jc    .1_vizsz              ;Ha a vonal vízszintes összetevője
                                    ;a nagyobb, arra az eljárásra
                                    ;ugrik.

        mov   dx,ax                 ;Egyébként az algoritmus
        shr   dx,1                  ;növekményét számláló
                                    ;regiszterbe beírja a vonal
                                    ;vízszintes hosszának a felét.
```

```
;fuggoleges

                mov   cx,ax                     ;A ciklusszámlálóba pedig a
                inc   cx                        ;hossznál eggyel többet.

.1_fuggo:       push  dx                        ;A számláló elmentése.

                call  pont                      ;Az aktuális pont kirakása.

                pop   dx
                mov   ax,word ptr [sgn_y]       ;Az algoritmus szerinti
                add   word ptr [pont_y],ax      ;következő pont koordinátainak

                add   dx,word ptr [delta_x]     ;kiszámítása.
                cmp   dx,word ptr [delta_y]
                jc    .2_fuggo

                sub   dx,word ptr [delta_y]
                mov   ax,word ptr [sgn_x]
                add   word ptr [pont_x],ax
.2_fuggo:       loop  .1_fuggo

                ret

;vizszintes

.1_vizsz:       mov   cx,word ptr [delta_x]     ;Hasonlóan mint a függőleges
                mov   dx,cx                     ;eseteben történt, itt is
                inc   cx                        ;azok a műveletek hajtódnak
                shr   dx,1                      ;végre, csak az ellenkező

.2_vizsz:       push  dx                        ;koordinátákkal.
                call  pont
                pop   dx
                mov   ax,word ptr [sgn_x]
                add   word ptr [pont_x],ax
                add   dx,word ptr [delta_y]
                cmp   dx,word ptr [delta_x]
                jc    .3_vizsz

                sub   dx,word ptr [delta_x]
                mov   ax,word ptr [sgn_y]
                add   word ptr [pont_y],ax
.3_vizsz:       loop  .2_vizsz
```

```
                ret

linc            endp


pont            Proc

                push    cx
                mov     ax,word ptr [pont_x]    ;A pont koordinátáiból
                mov     bl,8                    ;kiszámítja a memóriacímet.
                div     bl
                mov     cl,ah
                xor     ah,ah
                mov     di,ax
                mov     ax,word ptr [pont_y]
                mov     bx,80
                mul     bx
                add     di,ax
                mov     al,128
                shr     al,cl
                or      es:[di],al              ;Majd kirakja a pontot
                                                ;a megfelelő helyre.
                pop     cx
                ret

pont            Endp


pont_x:         dw      100                     ;A kirakandó pont xy
pont_y:         dw      100                     ;koordinátája.

xa:             dw      120                     ;A vonal kezdőpontjának
ya:             dw      175                     ;koordinátai.

xb:             dw      320                     ;A vonal végpontjának
yb:             dw      175                     ;koordinátai.

delta_x:        dw      ?                       ;A vonal vízszintes mérete.
delta_y:        dw      ?                       ;A vonal függőleges mérete.

sgn_x:          dw      ?                       ;A vízszintes lépésköz és
                                                ;iránya.
```

```
sgn_y:    dw    ?                ;A függőleges lépésköz és
                                 ;iránya.

hely:     db    80*25 dup (0)

prog12    ends                   ;A szegmens vége.
          end   start            ;A program vége.
```

A program induláskor elmenti a szöveges képernyő minden karakterét színbyte-ok nélkül. Ezt egy 80*25 hosszú ciklusban egy egyszerű **mov** utasítással teszi meg.

Ha ez megtörtént, jöhet a grafikus képernyőre történő átváltás, a 8*14 pontos ega betűkészlet betöltése,majd a 10h megszakítás 13h rutinjával kirakjuk a képernyőre az előzőekben letárolt karakteres képernyőt, ami így pontosan olyan lesz, mint szöveges üzemmódú elődje.

Ezt követően le kell tiltanunk az összes plane-t a nulladik kivételével, mivel az ega két színű módja úgy működik, hogy a gép két plane-t használ, amivel lehetőség van normál, fényes illetve villogó megjelenítési mód beállítására. A használt lapok a 0. és a 2. A használható kombinációk eredménye pedig a következő:

0	0	Nincs megjelenítés,
0	1	normál megjelenítés,
1	0	villogó megjelenítés,
1	1	fényes megjelenítés.

Tehát mivel nekünk a normál megjelenítésre van szükségünk, ezért kizárólag a 0. lapra kell írni. Az alakzat kirajzolása négy ciklusból lett megoldva a valószínűleg már ismert vonalhúzó rutin segítségével.

A program egy billentyűvárással és a karakteres képernyő visszaállításával fejeződik be. A rutin hátránya, hogy a BIOS használata miatt elég lassú, ezért akinek van kedve, megírhatja például a karakterek kirakására szolgáló programot, és így mindjárt sokkal gyorsabb lesz a trükk. Tovább lehet gyorsítani a dolgot azzal, hogy a grafikus üzemmódba váltás előtt másoljuk a video-memóriába a karaktereket és a 8fh

üzemmódot kapcsoljuk be ami, annyiban különbözik a 0fh funkciótól, hogy mivel a legfelső bit beállítása a képernyőtörlést tiltja, a már kirajzolt kép jelenik meg egyből.

4. Fejezet

Mi van, ha többet akarunk ?

Kivel ne fordult volna elő az a helyzet, hogy kevés volt a regiszterek száma vagy a 16 bit. Vagy netán egy egyszerű műveletre is több sort kellett begépelnie mivel a 8086-nak nincs arra a feladatra célutasítása. Nos ebben a fejezetben egy kicsit elkalandozunk a 386 valós módja által nyújtott lehetőségek felé. A védett módról ebben a könyvben nem lesz szó mivel az inkább operációs rendszerek illetve sokkal összetettebb programok megírásánál szükséges csupán. Sok emberben él az a hiedelem, hogy minél jobb gépe van, annál jobb programokat tud majd írni a gépére. Nos ez félig-meddig így is van, azonban nagyon ritka az a program ami egy 386-os gép tudásánál többet igényelne (legfeljebb sebességben, de ott még a 100MHz-es Pentium is lassú...). A nagyságrendnyi ugrás az XT-286-386 lépcsők között van. Ezután már csak a Pentium az, ami többet nyújt, de nem igen van a piacon olyan program ami kihasználná képességeit (még).

Mivel a 386 elég sok értelmes dolgot tud, amit egy 286 vagy XT nem, ezért érdemesnek láttam megemlíteni ezek közül azokat, melyek néha nagyságrendekkel könnyítik meg munkánkat.

A 386 real mód szolgáltatásai

Az alapvető változás, hogy az alap **ax**, **bx**, **cx**, **dx**, **si**, **di**, **bp**, **sp**, **ds**, **cs**, **es**, **ss**, **ip** 16 bites regiszterek mellé jött még két új általános célú szegmens regiszter az **fs**, **gs** és az **ax-sp** regiszterek 32 bites változata az **eax-esp** regiszterek. Ezek tulajdonképpen a már meglévő 16 bites elődjeinek a felső 16 bittel kibővített rokona. Így ezentúl nem 65535 a felső számhatár egy regiszterben, hanem 4.294.967.295 -ugye mekkora változás

Itt csupán egy pár fontos utasításról lesz szó, de a teljes utasításkészlet megtalálható a Dr. Kovács Magda által írt 80386/80486 II. c. könyvben, beleértve a védett módú lehetőségeket is.

Fontosabb utasítások

A legfőbb kérdés a különböző hosszúságú regiszterek közötti adatátvitel megoldása. Erre a célra két utasítás is szolgál, az egyik előjeles számként viszi át az adatot, a másik pedig nullákkal tölti ki a fennmaradó részt. Ezek a **MOVSX** (előjeles átvitelnél) illetve a **MOVZX** (nullával történő pótlás esetén). Ezek használhatók 8 bites adatból 16 és 32-bitesbe illetve 16 bitesből 32 bitesbe történő mozgatásra. Illetve a byte-word, word-dword konvertáló **CBW** és **CWD** utasításokhoz hasonlóan a **CWDE** 16 bites regiszterből 32-bitest csinál, a **CDQ** pedig két darab 32 bitest. Ennek főként osztásnál van nagy szerepe.

Ezenkívül igen jól használható utasítások vannak a bittesztelésekre és beállításokra, ezért a **TEST** és **OR** műveleteket lassan el is felejthetjük (kivéve ha csak az jó). Ezek tulajdonsága, hogy az eredményt a carry flagben tárolja. A bitek tesztelésére a **BT** utasítás szolgál. Használatánál egy bázis és egy offsetcímet kell megadni. A bázis lehet regiszter illetve memóriacím, az offset - ami meghatározza, hogy hányadik bit értékére vagyunk kíváncsiak - egy bármilyen regiszter.

A **BTC** hasonlít az előzőhöz, de nem csak teszteli hanem komplementálja is a bit értékét.

A **BTR** segítségével nullára, a **BTS** utasítással pedig egyre lehet állítani egy kijelölt bit értékét.

Továbbá fontos még az új regiszterekkel kapcsolatos verem kezelés. A **POPA** utasítás 32 bites megfelelője a **POPAD** illetve a **PUSHA** mellett a **PUSHAD** segít a regiszterek elmentésében.

Formai változás a forrásszövegben a **.386** és a **segment** után írt **use16** kiegészítések amik jelzik a fordítónak hogy 386-os utasításokra is számíthat (**tasm**-nál).

5. Fejezet

Hogyan írjunk szép programot ?

Nos igen, egy programnál nem csak az számít, hogy mennyi mindent tud, hanem az is, hogy milyen a külső megjelenése. Sőt néha ez utóbbi kerül előtérbe, és válik egy program sokak által használttá pedig se nem jó, se nem tud sokat, de szép... Ennek főként olyan software-eknél van jelentősége, amit másnak írunk, legyen az a barátunk vagy egy vállalat stb.

Amint azt már előző könyvemben is említettem, a karakteres üzemmódokat akár el is felejthetjük, illetve az inkompatibilitás miatt az svga üzemmódokat úgyszintén (megoldás: VESA). Mivel már szinte minden helyen legalább 386DX40-es gép van 4MB RAM-mal és VGA monitorral, ezért ennél lejjebb csak akkor érdemes írni, ha az feltétlenül igény. Így a következő lehetőségeink maradtak a video szempontjából: 320*200/256, 640*480/2, 640*480/16. Ezek közül az elsőt inkább játékprogramok írásánál célszerű használni, ahol a sokszínűség a fontosabb és nem a nagy felbontás. A mono üzemmódot pedig csak olyan helyzetekben érdemes alkalmazni, ahol a megjelenítendő adatnak tökéletesen elegendő 2 szín. Marad tehát a 16 színű mód, ami általában minden igényt kielégít, csak a programozó idegeit borzolja össze egy kicsit, mire előhozza belőle a kívánt képet. De mivel ha ló nincs, a VGA/16 is jó, ezért talán maradjunk ennél.

Előkészületek

Egy felhasználói program megírásánál célszerű először a hátteret majd az összes használandó objektumot megrajzolni, és csak ezután kezdeni a program megírásához. Természetesen mindezek előtt nem árt ha átgondoljuk, mit is szeretnénk.

Ha az ábrák megvannak, szükség van egy kirakó rutinra, amely megfelelően paraméterezve képes kirakni az ábrákat a képernyőre. Ez természetesen a tárolás módjától is függ, amit a készítő program határoz meg. Itt a legkevésbé se ajánlom a BMP formátumot, hacsak nem a program mérete után fizetnek... (én továbbra is maradok a De Luxe Paint-nél). Esetleg a hátteret kirakhatjuk külön rutinnal is.

Munkánk megkönnyítése

Az ábrák megrajzolása során nagyban megkönnyíthetjük későbbi munkánkat, ha ügyelünk egy pár dologra. Mégpedig, hogy az objektumaink vízszintes elhelyezkedését illetően lehetőleg byte határon kezdődjön egy byte-os szélességű legyen, azaz 8 valamely többszöröse legyen a szélesség értéke (nem árt ha wordös szélességgel dolgozunk ld. később). Ez azért fontos, mert különben a kirakásnál különböző bitforgatási manipulációkat kéne végezni, ami sokkal lassabbá és bonyolultabbá tenné a programot.

Kiegészítések a BBM formátumhoz

Ha már említésre került a DP, még egy pár szó a BBM formátumról, ami az előző könyvből kimaradt. Az egyik legfontosabb dolog, amire vigyázni kell a képkirakásnál, hogy azokat az alakzatokat melyek szélessége nem éri el a 72 képpontot, nem tömörítve tárolja a program, hanem bittérképesen. Először az első sor 0. plane-je, majd az 1. 2. 3. plane-je és utána jön a második sor 0. stb plane-je. A 72 pontos vagy annál szélesebb alakzatokat az előző könyvben már leírt módon tömörítve tárolja. Továbbá van még egy nehezítés az ábrák felhasználásában, mivel a nem wordös szélességű ábrákat a DP kiegészíti wordösre. Így ha egy ábra például 24 pont széles volt, azt 32 pont szélesnek tárolja. Ezzel 72 pont alatt nincs is különösebb probléma, mert az utolsó byte-ot egyszerűen átugorjuk a programból, de egy tömörített állománynál ezt nem tehetjük meg, mivel lehet hogy az illető byte egy különálló adat, de az is előfordulhat, hogy hozzátömörítette a képinformációhoz. Itt alapvetően két dolgot tehetünk. Vagy a memóriába tömörítjük ki az ábrát és a szükséges részt tesszük csak ki a képre, vagy a tárolásnál ügyelünk arra, hogy a kilógó rész háttérszíne megegyező legyen a majdani háttérszínnel és így nyugodtan kitehetjük a képernyőre. Mindkét esetben ügyelni kell azonban arra, hogy wordös szélességgel számoljunk a kitömörítésnél.

Helymegtakarítás

Fontos lehet egy programnál az is, hogy fölösleges információk ne kerüljenek tárolásra. Mivel egy ilyen típusú programnál általában fix

palettakiosztással dolgozunk, amit elegendő egyszer beállítani, továbbá mivel mi készítettük, tudunk minden méretbeli és egyéb információt az ábrákról, így mindezeket le lehet vágni a képről. Ezzel rengeteg hely megtakarítható. Én ezt egy igen primitív módszerrel végeztem el: az NC editorába behívtam a csonkítani kívánt állományt, és addig nyomtam a *delete* gombot, amíg fel nem tűnt a BODY címke. Ezt és az utána következő négy karaktert még töröltem, és ezután visszamentettem a file-t. Ezzel levágtam minden olyan információt, ami csak a file hosszát növeli.

A 13. mintaprogramba, amely egy kicsit hosszabb a megszokottnál, beépítettem egy pár olyan rutint amelyek feltétlenül szükségesek a kultúrált megjelenéshez, továbbá különböző ötleteket adhatnak mindenkinek. A program teljesen moduláris felépítésű, így működésének leírása is ennek megfelelően részekre bontva történik majd. Maga a program egyébként semmi hasznosat nem csinál, mivel csupán egy használható felületet kívántam bemutatni vele.

[Program 13]

```
.386                                    ;A 386-os utasításkészlet
                                        ;használatának engedélyezése.

prog13      segment use16               ;Szegmensdefiníció 386-os
                                        ;módban.
            assume cs:prog13,ds:data13  ;A szegmensregiszterek
                                        ;hozzárendelése a programhoz

start:      mov   ax,data13             ;Ds értékének beállítása.
            mov   ds,ax

            mov   ax,3d00h              ;A firstname címkénél tárolt
            mov   dx,offset firstname   ;file-név alapján betölti az
            int   21h                   ;adatokat a space címkétől
            push  ax                    ;kezdődően.
            mov   bx,ax
            mov   ax,3f00h
            mov   cx,0ffffh
            mov   dx,offset space
            int   21h
            pop   bx
            mov   ax,3e00h
```

```
        int     21h

        mov     ax,12h          ;640*480/16 mód beállítása.
        int     10h

        mov     ax,0a000h       ;A videomemória kezdőcímét
        mov     es,ax           ;es regiszterbe tölti.

        call    set1            ;Set1 üzemmód beállítása.
        call    makepal         ;A szürkeárnyalatok beállítása.
        call    picout          ;A kezdőkép kitömörítése.

        call    textinit        ;A karakterkirakó inicializálása.

        mov     ax,3d00h        ;A nyomógombok betöltése.
        mov     dx,offset filename
        int     21h
        push    ax
        mov     bx,ax
        mov     ax,3f00h
        mov     cx,0ffffh
        mov     dx,offset space
        int     21h
        pop     bx
        mov     ax,3e00h
        int     21h

        mov     si,offset putmap1   ;Az első csoport kirakása.
        call    bbmguide

        call    initmouse       ;Az egér inicializálása.
        jz      end             ;Egérhiba esetén ugrik
                                ;a végére.

.1_prog13:  call    mouseguide  ;Az egérkezelő elindítása.

        jmp     bx              ;A lenyomott gombnak megfelelő
                                ;funkció végrehajtása.

end:    mov     ax,3            ;A 80*25 karakteres mód
        int     10h             ;visszaállítása.

        mov     ah,4ch          ;Visszatérés a DOS-hoz.
        int     21h
```

```
makepal   proc

          mov   si,offset colors      ;Az si regisztert a palettatábla
          mov   dx,3c8h               ;kezdőcímére állítja.

          mov   cx,16                 ;16 paletta módosítandó.
.1_mp:    mov   al,[si]               ;A palettaszám beolvasása
          out   dx,al                 ;és kiküldése a 3c8h porton.
          inc   dx
          mov   al,[si+1]             ;A piros összetevő beolvasása
          out   dx,al                 ;és kiküldése a 3c9h porton.
          mov   al,[si+2]
          out   dx,al                 ;Zöld.
          mov   al,[si+3]
          out   dx,al                 ;Kék.
          dec   dx
          add   si,4                  ;A következő adatcsomagra állítja
                                      ;si értékét.
          loop  .1_mp                 ;Folytatás, ha még nem telt le
                                      ;a ciklus.

          ret                         ;Különben visszatérés.

makepal   endp


picout    proc

          mov   si,offset space       ;A kirakandó kép kezdőcíme.
          xor   di,di
          mov   dx,3c4h               ;Dx regiszterbe tölti a sequencer
                                      ;regiszter portcímét.

          mov   cx,480                ;A kép 480 sorból áll.
.1_picout: push cx
          mov   cx,4                  ;A tárolásnál először az első
                                      ;sor 0., 1., 2., majd 3. plane-je
                                      ;került a memóriába.

          mov   ax,102h               ;A 0. plane engedélyezéséhez
                                      ;szükséges ax beállítás.
```

```
.2_picout:  push  cx di
            out   dx,ax              ;A kiválasztott plane engedélyezése.
            mov   bh,80              ;Egy sor 80 byte-ból áll.

.3_picout:  mov   cl,ds:[si]         ;A tömörített blokk egy byte-ját
                                     ;a cl regiszterbe tölti,
            inc   si                 ;és növeli si-t.

            test  cl,128             ;Ha az adat tömörített, akkor
            jnz   .5_picout          ;ugrik a .5_picout címkéhez.

            inc   cl
            sub   bh,cl              ;A sorhossz számláló csökkentése.
.4_picout:  mov   bl,ds:[si]         ;Cl darab byte kimásolása
            mov   es:[di],bl         ;az állományból a képernyőre.
            inc   si
            inc   di
            loop  .4_picout
            jmp   .8_picout          ;Ugrás a végére

.5_picout:  neg   cl                 ;Neg cl+1 darabot tesz a
            inc   cl                 ;következő adatból a képre.
            sub   bh,cl
.6_picout:  mov   bl,ds:[si]
            inc   si
.7_picout:  mov   es:[di],bl
            inc   di
            loop  .7_picout

.8_picout:  or    bh,bh              ;Ellenörzi, hogy bh nem
            jnz   .3_picout          ;érte-e el a nullát. Ha még nem
                                     ;ujabb byte-ot olvas be.

            shl   ah,1               ;A következő plane kiválasztása,
            pop   di cx              ;így a sor mind a 4 plane-je
            loop  .2_picout          ;a képernyőre kerül.

            add   di,80              ;Új sor.
            pop   cx
            loop  .1_picout

            ret                      ;Visszatérés a rutinból.

picout      endp
```

```
bbmguide    proc

            mov     word ptr [grup],si      ;A grup változóba letátolja
                                            ;a kiválasztott csoportot.

.1_bbmguide:mov     al,[si]                 ;A csoport adott elemét al-be
            inc     si                      ;tölti, majd növeli si-t.
            cmp     al,255                  ;Ha al=255, az a csoport végét
            jz      .2_bbmguide             ;jelenti. Ekkor ugrás a végére.
            push    si                      ;Különben si elmentése és
            call    putbbm                  ;a kiválasztott objektum kirakása
            pop     si                      ;és si visszatöltése.

            jmp     .1_bbmguide             ;Újabb mező lehívása.

.2_bbmguide:ret

bbmguide    endp


putbbm      proc
            push    ax
            call    set1                    ;A set1 üzemmód beállítása.
            pop     ax
            mov     si,offset space         ;Az al értékét 12-vel
            mov     bl,12                   ;megszorozva, és hozzáadva ezt
            mul     bl                      ;a brushmap címke címéhez.
            mov     bp,offset brushmap      ;a kirakandó objektumot leíró
            add     bp,ax                   ;táblázat kezdőcímét kapjuk
                                            ;eredményül.

            add     si,ds:[bp]              ;Ezen táblázat első 2 byte-ja
                                            ;a kiválasztott objektum címét
                                            ;adja (space-hez viszonyítva).

            mov     di,ds:[bp+2]            ;2. és 3. wordje a kirakási
            shr     di,3                    ;pozíció x és y koordinátája,
            mov     ax,ds:[bp+4]            ;amiből a gép kiszámolja di
            mov     bx,80                   ;értékét.
            mul     bx
            add     di,ax
            mov     al,72
```

```
                cmp   ds:[bp+6],al          ;A 6. word a széleséget mutatja,
                jnc   contbbm               ;ami ha kisebb mint 72 akkor a
                jmp   sortbbm               ;BBM nincs tömörítve.

contbbm:        mov   dx,3c4h
                mov   cx,word ptr ds:[bp+8] ;8.word = magasság.

.1_bbm:         push  cx                    ;Ez a rutin megegyezik a picout
                mov   cx,4                  ;rutinnal, mindössze a para-
                mov   ax,102h               ;méterezése más. És figyeli.
.2_bbm:         push  cx di dx              ;hogy word széleségű legyen
                out   dx,ax                 ;az objektum.
                mov   bx,word ptr ds:[bp+6]
                shr   bx,3
                xchg  bl,bh
                inc   bh
                and   bh,254
.3_bbm:         mov   cl,ds:[si]
                inc   si
                test  cl,128
                jnz   .5_bbm
                inc   cl
                sub   bh,cl
.4_bbm:         mov   bl,ds:[si]
                mov   es:[di],bl
                inc   si
                inc   di
                loop  .4_bbm
                jmp   .8_bbm
.5_bbm:         neg   cl
                inc   cl
                sub   bh,cl
.6_bbm:         mov   bl,ds:[si]
                inc   si
.7_bbm:         mov   es:[di],bl
                inc   di
                loop  .7_bbm
.8_bbm:         or    bh,bh
                jnz   .3_bbm

                shl   ah,1                  ;A következő plane.
                pop   dx di cx
                loop  .2_bbm
                add   di,80                 ;A következő sor.
```

```
            pop   cx
            loop  .1_bbm

            ret


sortbbm:    mov   cx,ds:[bp+8]          ;A brush magassága.
            mov   dx,3c4h
.1_sbbm:    push  cx
            mov   cx,4                  ;4 plane.
            mov   ax,102h
.2_sbbm:    push  cx di
            out   dx,ax
            mov   cx,ds:[bp+6]          ;A brush szélessége.
            shr   cx,3
.3_sbbm:    mov   bl,ds:[si]            ;ds:si-ről átmásol egy byte-ot
            mov   es:[di],bl            ;es:di-re, majd
            inc   si                    ;mindkét indexregiszter
            inc   di                    ;értékét növeli.
            loop  .3_sbbm

            mov   bx,ds:[bp+6]          ;Ha az alakzat nem wordös
            shr   bx,3                  ;széleségű, akkor növeli az
            adc   si,0                  ;si értékét, átlépve ezzel a
            shl   ah,1                  ;fölösleges részt.
            pop   di cx
            loop  .2_sbbm
            add   di,80
            pop   cx
            loop  .1_sbbm

            ret


putbbm      endp


getscreen   proc

            mov   ax,word ptr [mouse_y] ;Az egér koordinátáiból
            mov   bx,80                 ;kiszámolja si értékét.
            mul   bx
            mov   si,ax
            mov   ax,word ptr [mouse_x]
```

```
            shr   ax,3
            add   si,ax
            mov   di,offset screenspace   ;A tárolás a megadott címkétől
                                          ;kezdődik.

            mov   cx,4                    ;4 plane.
            mov   dx,3ceh
            mov   ax,0ff08h               ;Adatváltoztatás engedélyezése
            out   dx,ax                   ;a byte mind a 8 bitjén.

            mov   ax,3
            out   dx,ax                   ;0. írási mód.

            mov   ax,4                    ;A 0. bitsík kiválasztása
                                          ;olvasásra.

.1_gs:      push  cx si
            out   dx,ax

            inc   ah                      ;A következő plane beállítása.

            mov   cx,16                   ;16 darab dword-öt olvasunk
.2_gs:      mov   ebx,es:[si]             ;ki, és írunk be az új címre.
            mov   [di],ebx
            add   di,4
            add   si,80
            loop  .2_gs
            pop   si cx
            loop  .1_gs

            ret

getscreen   endp


putscreen   proc

            mov   ax,word ptr [mouse_y]   ;A forrás és cél regiszterek
            mov   bx,80                   ;beállítása.
            mul   bx
            mov   di,ax
            mov   ax,word ptr [mouse_x]
            shr   ax,3
            add   di,ax
```

```
                mov   si,offset screenspace

                mov   cx,4                      ;4 plane.

                mov   dx,3ceh                   ;A megfelelő írási mód
                mov   ax,0ff08h                 ;beállítása, és a plane
                out   dx,ax                     ;kiválasztása.
                mov   ax,3
                out   dx,ax
                mov   ax,1
                out   dx,ax
                dec   ax
                out   dx,ax
                mov   dx,3c4h
                mov   ax,102h

.1_ps:          push  cx di
                out   dx,ax
                shl   ah,1
                mov   cx,16                     ;16 darab dword kimásolása
.2_ps:          mov   ebx,[si]                  ;a képernyőre.
                mov   es:[di],ebx
                add   di,80
                add   si,4
                loop  .2_ps
                pop   di cx
                loop  .1_ps

                ret

putscreen       endp


putmask         proc

                mov   ax,word ptr [mouse_y]     ;A memóriacím kiszámítása.
                mov   bx,80
                mul   bx
                mov   di,ax
                mov   ax,word ptr [mouse_x]
                mov   bl,8
                div   bl
                mov   byte ptr [shift],ah       ;Az osztásnál képződött maradékot
                                                ;a shift változóba teszi. Ez
```

```
                                    ;szolgál a byte-on belüli
                                    ;pozíció meghatározására.
        xor   ah,ah
        add   di,ax

        mov   si,offset mask
        mov   ax,0f02h              ;Mind a 4 plane-t engedélyezi.
        mov   dx,3c4h
        out   dx,ax

        mov   cx,16
        mov   dx,3cch
        mov   ax,0000h              ;A set regiszterbe nullát tölt.
        out   dx,ax
        mov   ax,0f01h              ;és engedélyezi mind a 4 plane-re.
        out   dx,ax
        mov   al,8                  ;A 3ceh port 8. regisztere a
                                    ;byte-on belüli bit változta-
                                    ;tásokat engedélyezi illetve
                                    ;tiltja.

.1_m:   push  cx
        xor   bh,bh
        mov   bl,[si]               ;Betölti bl-be a mask alsó
                                    ;byte-ját,
        mov   cl,byte ptr [shift]   ;és elforgatja a shift által
        ror   bx,cl                 ;mutatott értékkel, aminek
                                    ;hatására az adott pozícióba
                                    ;kerülő pontok bl regiszterben
                                    ;maradnak, a következő byte
                                    ;elejét pedig a bh-ba átcsúszó
                                    ;rész alkotja.

        mov   ah,bl                 ;Az első byte kirakása.
        out   dx,ax
        or    es:[di],bl

        mov   ah,bh                 ;A 2. byte első felének kirakása.
        out   dx,ax
        or    es:[di+1],bh

        xor   bh,bh                 ;A 2. byte második felének
        mov   bl,[si+1]             ;kirakása.
        ror   bx,cl
```

```
            mov   ah,bl
            out   dx,ax
            or    es:[di+1],bl

            mov   ah,bh                 ;A 3. byte kirakása.
            out   dx,ax
            or    es:[di+2],bh

            add   si,2                  ;A következő sor.
            add   di,80
            pop   cx
            loop  .1_m
            ret

putmask     endp


putmouse    proc

            mov   dx,3ceh               ;Set-Reset tiltása.
            mov   ax,1
            out   dx,ax

            mov   ax,1003h              ;Or kirakási mód.
            out   dx,ax

            mov   ax,5                  ;0. írási mód.
            out   dx,ax

            mov   ax,word ptr [mouse_y] ;Az objektum címének
            mov   bx,80                 ;kiszámítása.
            mul   bx
            mov   di,ax
            mov   ax,word ptr [mouse_x]
            mov   bl,8
            div   bl
            mov   cl,ah                 ;A maradékot cl-be teszi.
            xor   ah,ah
            add   di,ax

            mov   si,offset mouse

            mov   bl,128                ;Segéd byte.
            shr   bl,cl
```

```
                mov   cx,16             ;A kirakandó ábra 16*16 pont.
.1_pm:          push  cx di
                mov   cx,16
.2_pm:          mov   ah,[si]           ;Az aktuális képpont beolvasása.

                cmp   ah,0              ;Ha nulla, akkor nem teszi ki.
                jz    .3_pm

                mov   al,2
                mov   dx,3c4h           ;A színnek megfelelő plane
                out   dx,ax             ;engedélyezése.

                mov   al,8              ;A bl által mutatott pozíció
                mov   ah,bl             ;engedélyezése
                mov   dx,3cch
                out   dx,ax

                or    es:[di],bl        ;és a kívánt színűre történő
                                        ;változtatása.

.3_pm:          ror   bl,1              ;A következő pont.
                jnc   .4_pm
                inc   di
.4_pm:          inc   si
                loop  .2_pm
                pop   di cx
                add   di,80             ;A következő sor.
                loop  .1_pm

                ret

putmouse        endp


initmouse       proc

                xor   ax,ax             ;Az egér inicializálása.
                int   33h

                or    ax,ax             ;Ha ax = 0 akkor az hibát jelent.
                jz    endinit

                mov   ax,4              ;A kezdő koordináták beállítása.
```

```
                mov   cx,320
                mov   dx,240
                int   33h

                mov   ax,8                    ;Az egér mozgási területének
                mov   cx,16                   ;korlátozása.
                mov   dx,464
                int   33h
                mov   ax,7
                mov   cx,16
                mov   dx,624
                int   33h

                mov   word ptr [mouse_x],320  ;A koordináták memóriába
                mov   word ptr [mouse_y],240  ;mentése.

                call  getscreen               ;A képernyőterület eltárolása.
                call  putmask                 ;A maszk kirakása.
                call  putmouse                ;Az egér kirakása.

endinit:        ret

initmouse       endp


mouseguide      proc

.1_guide:       mov   ax,3                    ;Az egérparaméterek lekérdezése.
                int   33h

                test  bx,3                    ;Ha van lenyomott gomb, ugrik
                jnz   .5_guide                ;a vizsgáló részre.

.2_guide:       cmp   word ptr [mouse_x],cx   ;különben az elmozdulást
                jnz   .3_guide                ;vizsgálja, és ha van, a
                cmp   word ptr [mouse_y],dx   ;.3_guide címkénél folytatja.
                jz    .4_guide

.3_guide:       push  cx dx
                call  putscreen               ;A régi képernyőtartalom
                                              ;kirakása a képre.
                pop   dx cx

                mov   word ptr [mouse_x],cx   ;Az új koordináták beállítása.
```

```
                mov     word ptr [mouse_y],dx

                call    getscreen               ;Az egér alatti terület
                                                ;eltárolása.
                call    putmask                 ;Az egérmaszk kirakása.
                call    putmouse                ;Az egér kirakása.

.4_guide:       jmp     .1_guide                ;Vissza az elejére.

.5_guide:       call    buttontest              ;Az ellenőrző rutin hívása.
                jnz     .2_guide                ;Ha az egér nem egy aktív mező
                                                ;fölött állt, akkor vissza a
                                                ;rutin elejére.

                ret                             ;Egyébként visszatérés a
                                                ;főprogramhoz.

mouseguide      endp


buttontest      proc

                push    cx dx
                mov     si,word ptr [grup]      ;Az aktív gombok táblázatcímének
                                                ;lekérdezése.

.1_bt:          mov     bp,offset brushmap
                mov     al,[si]                 ;Az aktuális objektum számának
                                                ;betöltése az al regiszterbe
                cmp     al,255                  ;A 255 érték a felsorolás végét jelzi.
                jz      .4_bt

                xor     ah,ah                   ;Az mezőt leíró tábla címének
                mov     bl,12                   ;kiszámítása.
                mul     bl
                add     bp,ax
                add     bp,2

                mov     ax,word ptr [mouse_x]   ;Ax, bx regiszterekbe az egér
                mov     bx,word ptr [mouse_y]
                mov     cx,ds:[bp]              ;cx, dx-be pedig a vizsgált mező
                mov     dx,ds:[bp+2]            ;bal felső sarkának koordinátái
                                                ;kerülnek.
```

```
            cmp   ax,cx                ;Ellenőrzi, hogy az egér helyc
            jc    .2_bt                ;nincs-e balra vagy följebb
            cmp   bx,dx                ;a kiválasztott ponttól mert
            jc    .2_bt                ;ha igen, az azt jelenti, hogy
                                       ;biztosan nincs a mező fölött.

            add   cx,word ptr ds:[bp+4] ;Az objetum koordinátáihoz azok
            add   dx,word ptr ds:[bp+6] ;széleségét illetve magasságát
                                       ;adja hozzá.

            cmp   cx,ax                ;Ellenőrzi, hogy az így
            jc    .2_bt                ;kiszámolt koordinátáktól
            cmp   dx,bx                ;nincs-e jobbra vagy alatta.
            jnc   .3_bt                ;mert ez szintén azt jelenti.
                                       ;hogy nincs az aktuális mező
                                       ;fölött.

.2_bt:      inc   si                   ;A következő mező vizsgálata.
            jmp   .1_bt

.3_bt:      mov   bx,ds:[bp+8]         ;Az adott mezőhöz tartozó
            pop   dx cx                ;ugrási címet a bx regiszterbe
            test  al,0                 ;tölti, és a z bitet 1-be
            ret                        ;állítja a visszatérés előtt.

.4_bt:      or    bl,bl                ;Törli a z bitet, és visszatér
            pop   dx cx                ;a hívó programhoz.
            ret

buttontest  endp


set1        proc

            mov   dx,3cch              ;Set-reset tiltása.
            mov   ax,1
            out   dx,ax

            mov   ax,3                 ;Felülírási mód.
            out   dx,ax

            mov   ax,5                 ;0. írási mód.
            out   dx,ax
```

```
                mov   ax,0ff08h          ;Adatváltoztatás engedélyezése
                out   dx,ax              ;minden biten.

                ret

set1            endp


set2            proc

                push  ax dx

                mov   dx,3c4h            ;Minden plane engedélyezése
                mov   ax,0f02h
                out   dx,ax

                mov   dx,3cch            ;Set-reset data = 0
                xor   ax,ax
                out   dx,ax

                mov   ax,0f01h           ;és engedélyezése az összes
                out   dx,ax              ;plane-re.

                pop   dx ax
                ret

set2            endp


clear1          proc

                call  putscreen

                mov   si,word ptr [grup] ;Az aktív csoport lehívása.

.1_clr:         mov   al,[si]            ;Egy mező lehívása.
                inc   si
                cmp   al,255
                jz    .3_clr

                mov   bp,offset brushmap ;Az aktuális mező táblacímének
                mov   bl,12              ;kiszámítása.
                mul   bl
                add   bp,ax
```

```
                mov   ax,ds:[bp+4]          ;A képernyőcím kiszámítása a
                mov   bx,80                 ;táblázat adatai alapján.
                mul   bx
                mov   di,ax
                mov   ax,ds:[bp+2]
                shr   ax,3
                add   di,ax

                call  set1                  ;A set1 mód beállítása.

                mov   dx,3c4h               ;Az összes plane engedélyezése.
                mov   ax,0f02h              ;(fehér kitöltőszín)
                out   dx,ax

                mov   cx,word ptr ds:[bp+8] ;A mező szélességének és
.2_clr:         push  cx di                 ;magasságának megfelelő
                mov   cx,word ptr ds:[bp+6] ;méretű rész kitöltése 255
                shr   cx,3                  ;azaz fehér színnel.
                mov   al,255
                rep   stosb
                pop   di cx
                add   di,80
                loop  .2_clr
                jmp   .1_clr

.3_clr:         call  getscreen
                call  putmask
                call  putmouse
                ret

clear1          endp


textinit        proc

                mov   ax,1130h              ;A 8*14 pontos betükészlet
                mov   bh,2                  ;címének lekérdezése és
                push  es                    ;betöltése az fs regiszterbe.
                int   10h
                push  es
                pop   fs
                pop   es
                mov   word ptr [fontstart],bp ;Az offset értéket egy memória
```

```
                                        ;változóban tároljuk.
            ret

textinit    endp


textguide   proc

            call    set2                ;A 2. beállítás aktíválása.
            mov     si,offset text1     ;A kirakandó szöveg címe.

            mov     ax,64               ;A megjelenítés x koordinátája.
            mov     bx,64               ;A megjelenítés y koordinátája.
            mov     cx,26               ;Egy sor hossza.

            shr     ax,3                ;A képernyőcím kiszámítása.
            mov     di,ax
            mov     ax,80
            mul     bx
            add     di,ax

.1_tg:      push    cx di
.2_tg:      mov     al,[si]             ;Egy karakter beolvasása.
            inc     si
            cmp     al,255              ;Ha ez 255 akkor vége a
            jz      .3_tg               ;kiírásnak.

            call    putchr              ;A beolvasott karakter kirakása.
            inc     di                  ;A következő pozíció.

            loop    .2_tg

            pop     di cx
            add     di,80*14            ;Új sor.
            jmp     .1_tg

.3_tg:      pop     cx cx               ;A kiírás vége.
            ret

textguide   endp


putchr      proc
```

```
                pusha                   ;Az összes regiszter elmentése.
                mov   si,word ptr [fontstart]
                mov   cx,14             ;A betűk 14 képpont magasak.
                mov   bl,14             ;Ha a betű kódját megszorozzuk
                mul   bl                ;14-gyel, akkor a kirakandó
                                        ;karakter memóriabeli címét
                                        ;kapjuk, amit hozzá kell adni
                add   si,ax             ;az offset értékhez.

                mov   dx,3ceh
                mov   al,8

.1_pc:          mov   ah,fs:[si]        ;Ah-ba az fs:si tartalmát, azaz
                                        ;a betű egyik sorát teszi.
                inc   si

                out   dx,ax             ;Az adatváltoztatást csak a
                                        ;sor aktív bitjeinél engedi.

                or    es:[di],ah        ;A karakter adott sorának
                                        ;kirakása.

                add   di,80             ;A következő pontsor.
                loop  .1_pc
                popa
                ret

putchr          endp


press           proc

                push  si                ;Az si regiszter az éppen
                call  putscreen         ;aktívált mező kódját
                pop   si                ;tartalmazza, így, ha a nála
                mov   al,[si]           ;eggyel nagyobb kódút kirakjuk,
                inc   al                ;akkor az annak a gombnak a
                call  putbbm            ;lenyomott változata.

.1_press:       mov   ax,3              ;Várakozás az egérgomb
                int   33h               ;felengedéséig.
                test  bx,3
                jnz   .1_press
```

```
                ret

press           endp


press2          proc

                call    getscreen
                call    putmask
                call    putmouse
                ret

press2          endp


routin1:        call    press                   ;A lenyomott gomb kirakása.
                mov     si,offset putmap2
                call    bbmguide                ;A 2. csoport megjelenítése.
                call    press2
                jmp     .1_prog13


routin2:        call    press
                mov     si,offset putmap3       ;A 3. csoport megjelenítése.
                call    bbmguide
                call    press2
                jmp     .1_prog13

routin3:        call    press
                call    textguide               ;A szöveg kirakása.
                mov     al,4
                call    putbbm
                call    press2
                jmp     .1_prog13

routin4:        jmp     end

routin5:
routin6:
routin7:
routin8:
routin9:
routin10:
routin11:
```

```
routin12:   call    clear1                  ;Az aktív csoport törlése
            mov     si,offset putmap1       ;és az 1. kijelölése aktívnak.
            mov     word ptr [grup],si
            call    set1
            call    bbmguide
            jmp     .1_prog13

prog13      ends

;*********************************************************

data13      segment use16

mask        db      11111111b,11110000b     ;Az egérkurzor maszkja
            db      11111111b,11100000b
            db      11111111b,11000000b
            db      11111111b,10000000b
            db      11111111b,00000000b
            db      11111111b,00000000b
            db      11111111b,10000000b
            db      11111111b,11000000b
            db      11110011b,11100000b
            db      11100001b,11110000b
            db      11000000b,11111000b
            db      10000000b,01111100b
            db      00000000b,00111110b
            db      00000000b,00011111b
            db      00000000b,00001111b
            db      00000000b,00000111b

mouse       db      15,15,15,15,15,15,15,15,15,15,15,15,0,0,0,0
            db      15,7,7,7,7,7,7,7,7,7,15,0,0,0,0,0
            db      15,7,7,7,7,7,7,7,7,7,15,0,0,0,0,0,0 ;Az egérkurzor képe.
            db      15,7,7,7,7,7,7,7,15,0,0,0,0,0,0,0
            db      15,7,7,7,7,7,7,7,15,0,0,0,0,0,0,0
            db      15,7,7,7,7,7,7,7,15,0,0,0,0,0,0,0
            db      15,7,7,7,7,7,7,7,15,0,0,0,0,0,0,0
            db      15,7,7,7,15,15,7,7,7,15,0,0,0,0,0,0
            db      15,7,7,15,0,0,15,7,7,7,15,0,0,0,0,0
            db      15,7,15,0,0,0,0,15,7,7,7,15,0,0,0,0
            db      15,15,0,0,0,0,0,0,15,7,7,7,15,0,0,0
            db      15,0,0,0,0,0,0,0,0,15,7,7,7,15,0,0
            db      0,0,0,0,0,0,0,0,0,0,15,7,7,7,15,0
            db      0,0,0,0,0,0,0,0,0,0,0,15,7,7,7,15
```

```
                db      0,0,0,0,0,0,0,0,0,0,0,0,15,7,7,15
                db      0,0,0,0,0,0,0,0,0,0,0,0,15,15,15

screenspace     dw      128 dup (?)

mouse_x         dw      320
mouse_y         dw      240

grup            dw      ?
shift           db      0
fontstart       dw      ?

putmap1         db      0,2,4,6,255
putmap2         db      8,10,12,14,255
putmap3         db      16,18,20,22,255

text1           db      "Ez egy bemutató program  "
                db      "ami az IBM PC Gyakorlati "
                db      "Assembly című könyv 13.  "
                db      "példaprogramjához készült."
                db      "  (C) 1994 The Aga",255

brushmap        dw      0,440,24,72,22,offset routin1
                dw      556,440,24,72,22,0          ;0 gomb1
                dw      1016,528,24,72,22,offset routin2
                dw      1572,528,24,72,22,0         ;2 gomb2
                dw      2030,440,48,72,22,offset routin3
                dw      2586,440,48,72,22,0         ;4 gomb3
                dw      3044,528,48,72,22,offset routin4
                dw      3600,528,48,72,22,0         ;6 gomb4
                dw      4058,440,72,72,22,offset routin5
                dw      4614,440,72,72,22,0         ;8 gomb5
                dw      5070,528,72,72,22,offset routin6
                dw      5626,528,72,72,22,0         ;10 gomb6
                dw      6084,440,96,72,22,offset routin7
                dw      6640,440,96,72,22,0         ;12 gomb7
                dw      7098,528,96,72,22,offset routin8
                dw      7654,528,96,72,22,0         ;14 gomb8
                dw      8112,440,120,72,22,offset routin9
                dw      8668,440,120,72,22,0        ;16 gomb9
                dw      9126,528,120,72,22,offset routin10
                dw      9682,528,120,72,22,0        ;18 gomb10
                dw      10140,440,48,72,22,offset routin11
```

```
            dw    10696,440,148,72,22,0      ;20 gomb11
            dw    11154,528,148,72,22,offset routin12
            dw    11710,528,148,72,22,0      ;22 gomb12

colors      db    0,0,0,0,1,4,4,4,2,9,9,9,3,13,13,13,4,17,17,17
            db    5,21,21,21,20,26,26,26,7,30,30,30,56,34,34,34
            db    57,38,38,38,58,43,43,43,59,47,47,47,60,51,51,51
            db    61,55,55,55,62,60,60,60,63,63,63,63

filename    db    "brush.aga",0
firstname   db    "kep13.aga",0
space       db    ?

data13      ends
            end   start
```

A vezérlőprogram működése

A program alapvetően három részre osztható:

1. Vezérlő rész,
2. Szubrutin gyűjtemény,
3. Adatterület.

A vezérlő programnak ebben az esetben nem sok szerep jutott, mindössze az alap paraméterezés és a megfelelő rutinok indítása.

Az első pár sor a már félig-meddig ismert fejléc és szegmensregiszter beállítás. Ezt követi a háttérképet tartalmazó file betöltése, itt tölti be a háttérkép tömörített és csonkított adatait. A művelet során a file hosszára egy fars értéket adtunk meg, de a betöltés így is tökéletes, mindössze a végrehajtás után visszakapjuk a valóban betöltött byte-ok számát, amire azonban semmi szükségünk sincs. Ezután bekapcsoljuk a 640*480/16 grafikus üzemmódot, beállítjuk a videomemória szegmenscímét, a megfelelő üzemmódot, a palettát, és meghívjuk a háttérképet kitömörítő rutint.

Visszatérés után elindítjuk a karakterkészlet paramétereket lekérdező rutint, majd betöltjük a gombokat tartalmazó file-t, és a **bbmguide** rutinnal

kirakjuk a képernyőre a **putmap1** által megjelölt mezőket. Ha ezzel végeztünk, az egér inicializálása következik. Ha visszatéréskor valami hibát jelez a gép, akkor visszatérünk a DOS-hoz. Ha minden rendben volt, indul az egérkezelő program, ahonnan csak akkor tér vissza, ha lenyomtunk egy gombot. Ekkor az indítandó rutin címét a **bx** regiszter mutatja, így egy **jmp bx** segítségével könnyedén elindíthatjuk azt.

A főprogram a képernyő üzemmódjának visszaállításával és a DOS-hoz való visszatéréssel ér véget.

A szubrutinok működése

A legelső rutin a **makepal** a megfelelő palettaszínek beállítását végzi el. Mivel az itt használt üzemmódban a palettaszínek nem egymás után következnek, hanem szétszórva helyezkednek el a 256 helyen, ezért azt a megoldást választottam a beállításra, hogy egy színhez 4 byte-ot rendelek. Ezek közül az első jelenti, hogy melyik palettahelyre kell beírni, a maradék három az RGB színinformáció. Ezen információk az adatszegmens **colors** címkéjétől kezdődően lettek letárolva.

A második eljárás (**picout**) egy kitömörítő rutin, amely a háttérkép kitömörítését végzi. Az algoritmust most nem ismertetem, mivel ezt már leírtam a már említett könyvemben. A rutin lényege, hogy mivel tudjuk a kép méreteit, így a kitömörítésnél csak ezt kell figyelemmel kísérni.

A következő igen fontos rutin a **bbmguide**, amely az ábrák kirakásának vezérlését végzi. Indítása előtt az **si** regiszterbe kell beállítani azt a csoportot, amit ki akarunk rakni a képre. Csoport alatt értem itt az összetartozó nyomógombokat, melyeket egyszerre kell megjeleníteni a képernyőn, és mindig együtt. Ezeket a **putmap1...x** címkéken tároltam le, egy 255 kóddal minden csoport végén. A vezérlő rutinnak ez jelzi, hogy elfogytak az ábrák, lehet visszatérni a rutinból.

Az ábrák kirakását a **putbbm** rutin végzi. Ezt használhatjuk a vezérlőtől függetlenül önállóan is, mindössze az **ax** regiszterbe kell beírni a kirakandó ábra sorszámát. Ezután a rutin ebből az információból kiszámítja az objektum fizikai címét oly módon, hogy az egyes ábrákat leíró **brushmap** legelső bejegyzése az adott objektum címe a file-on belül.

Kiszámítja továbbá azt is, hogy a képernyőn hová kell kirakni az adott alakzatot, mivel a 2. és 3. adat a kirakási pozíciót határozza meg. Figyelni kell arra is, hogy tömörített vagy tömörítetlen alakzatról van-e szó illetve hogy a szélessége 16-tal osztható-e vagy sem. Az ábra szélességét az előbb említett leírótábla 4. adatából tudja meg, amely az ábra szélességét tárolja. A kitömörítésnél szükség van az ábra magasságára is ami a 5. word. Lehetőség van továbbá minden mezőhöz egy rutint rendelni. Ezt oly módon tehetjük meg, hogy a 6. adat helyére az indítandó rutin offsetcímét írjuk be. Ha nem tartozik indítandó program az objektumhoz, akkor ide célszerűen egy nullát lehet írni. A lényeg az, hogy valamilyen adat mindenféleképpen legyen, mert különben a kép címének kiszámításakor hibás értéket kapnánk. Mivel a számításnál úgy kalkulál, hogy minden leíró tábla 6 word-ből áll.

A következő pár rutin szorosan összetartozik, mivel mindegyik az egér kezelésével kapcsolatos. A **getscreen** az egér alatti kép eltárolását végzi. Teszi ezt úgy, hogy kiszámítja az egér pozícióját viszont a byte-on belüli bitpozíciót figyelmen kívül hagyja. Ahhoz, hogy mégis tárolásra kerüljön az egész nyíl alatti terület, nem csak a feltétlen szükséges 16*16 képpontot tároljuk el (mivel ekkor különböző bitforgatásokra, stb. is szükség lenne), hanem 32*16 pontot. Erre kiválóan alkalmas a 386-os által biztosított 32 bites regiszterek. A 4 plane-t egymás után tároljuk le a kiolvasási lapozó regiszter megfelelő beállításával.

A **putscreen** nagyon hasonlít az előző rutinhoz, csupán a forrás és cél rekeszek cserélődtek fel, és nem a kiolvasási lapozó regisztert kell állítani, hanem az írás engedélyezést kell megadni az adott lapra.

Akkor, ha egy ábra kirakásánál nem használjuk ki egy byte összes bitjét, ügyelni kell arra, hogy csak azokat a biteket változtassuk meg, amelyeket szükséges. A probléma csupán annyi, hogy itt egy képpontot 4 bit határoz meg amik ráadásul egymás mögött vannak. Van azonban egy igen praktikus megoldás arra, hogy egyszerre nullázzuk ki az egér maszkját. Ezt a **set-reset** használatával könnyedén megtehetjük.

Ezután a **set-reset** regisztert letiltjuk, és beállítjuk **or** műveletre a kirajzolást. Majd a planek írását engedélyező regisztert felhasználva a színek beállítására, pontonként kirakjuk az egér képét.

Az egér inicializálása a 33h megszakítás segítségével történik. Itt állítjuk be, és tároljuk el a kezdő koordinátákat, korlátozzuk a mozgási területet, valamint a már ismertetett rutinok segítségével eltároljuk az egér alatti képet, és kirakjuk az egérkurzort.

A továbbiakban minden egérrel kapcsolatos vezérlést a **mouseguide** rutin végez. Ellenőrzi, hogy van-e lenyomott egérgomb, illetve, hogy történt-e elmozdulás az utolsó lekérdezés óta. Ha gomb lett lenyomva, ugrik az ellenőrző rutinra. Ha pedig elmozdulás történt, kirakja a régi háttérképet, megváltoztatja a koordinátákat, eltárolja az új képet és kirakja az egérkurzort az új pozícióba.

Az gombok vizsgálatát a **buttontest** rutin végzi. Ez az egyik legfontosabb eljárás a programban, ugyanis ez határozza meg, hogy az adott pozícióban volt-e olyan objektum, ami aktiválásakor indítani kell egy másik rutint. Teszi ezt úgy, hogy egy objektumcsoport kirakásakor eltárolódik az aktuális csoport listáját tartalmazó memóriaterület címe. Így csupán ezen objektumok területére kell a vizsgálatot végrehajtani. Mivel minden ábra leíró táblája tartalmazza annak kirakási pozícióját és méreteit, csupán ezeket kell lehívni, és összehasonlítani az egér pozíciójával. Amennyiben egy objektum fölött állt a kurzor, a **bx** regiszterbe beolvassa az indítandó rutin offsetcímét és a **z** bitet 1-be állítja, jelezve ezzel, hogy van lenyomott gomb. Ha a rutin úgy futott le, hogy az adott koordináta egyik mezőnek sem felelt meg, törli a **z** bitet, és így tér vissza.

A következő két rutin a megfelelő üzemmódok beállításáért felelős. A **set1** tiltja a **set-reset** funkciót, és beállítja a 0. üzemmódban a felülírási funkciót, továbbá az adatváltoztatást engedélyezi minden biten.

A **set2** rutin az összes plane-t egyszerre engedélyezi, és engedélyezi a **set-reset** funkciót az összes lapra nulla adattal.

A **clear1** rutin a **buttontest** rutinhoz hasonlóan az éppen aktív csoport alakzatait törli le a képernyőről.

A következő három eljárás a szöveges megjelenítést végzi. Az első (**textinit**) csupán lekérdezi a 8*14 pontos betűkészlet memóriabeli címét és

eltárolja ezt az **fs** szegmensregiszterbe. A készlet offsetcímét egy memóriaváltozóban tároljuk.

A **textguide** rutin a szöveg kiíratását végzi. Első lépésben beállítja a kettes üzemmódot, majd a kirakás paramétereit. Ezután a kirakandó karaktereket egyenként kiolvassa, és a **putchr** rutin meghívásával kirakja azokat.

A program úgy lett kialakítva, hogy az első három nyomógomb lenyomásakor a gomb láthatóan benyomódik, de amíg az egérgombot nem engedjük el, nem történik semmi. Ezt a **press1** rutinnal oldottam meg úgy, hogy mivel az egérgomb lenyomása utáni vizsgálat ha aktív pozícióban találta a kurzort, kiolvassa a hozzá tartozó rutin címét. Ezt viszont csak akkor tudja megtenni, ha tudja az adott alakzat leírótáblájának címét. Ha pedig tudja, akkor azt egy másik rutin is fel tudja használni. Így van ez ebben az esetben is és mivel a gombok úgy lettek eltárolva, hogy a gomb normál alakját annak lenyomott változata követi, mindössze meg kell növelni eggyel az alakzat sorszámát és kitenni a képernyőre. Ezzel elértük azt, hogy a gomb lenyomódott. Ezután csupán azt kell figyelemmel kísérni, hogy mikor engedjük fel az egér gombját. Amikor ez megtörtént, akkor indítjuk el a nyomógombhoz tartozó rutint.

A **press2** eljárás a megfelelő rutin végrehajtása utáni ismételt egérkirakást végzi el.

Ezzel a szó szerinti értelembe vett eljárásoknak vége, azonban mégis itt kaptak helyet azok a rutinok, amelyek az egyes gombok lenyomásakor végrehajtódnak.

Az első nyomógomb lenyomására megjelenik a 2. csoport és a gomb lenyomva marad. A 2. gomb hatása hasonló, csak a 3. csoport jelenik meg. A 3. gomb megnyomásának hatására az lenyomódik, kiíródik a megadott szöveg, és a gomb visszanyeri eredeti alakját. A 4. nyomógomb a kilépésre lett fenntartva. Az összes többi gomb hatására eltűnik az adott alcsoport, és ismét az 1. csoport gombjait figyeli a program.

A program adatmezője

Az adatok letárolásánál elsőként az egér maszkja kerül sorra. Ezt követi magának az egérnek a képe. Ezután különböző memóriaváltozók jönnek. Ezt követik az egyes csoportokat leíró táblák amelyek meghatározzák, hogy mely alakzatokat kell egyszerre kitenni a képre. A **text1** címke alatt a kirakandó szöveget tároltuk le. Ezt követi magukat az alakzatokat leíró tábla, a gombok normál és lenyomott változatával, a palettatérkép és a két betöltendő file neve. A file-ok betöltése a **space** címkétől kezdődik.

6. Fejezet

Az animáció

Az emberek egy programról alkotott véleményét, mint említettem, elsődlegesen az adott program arculata, megjelenése határozza meg, legyen az felhasználói, oktató vagy játék program. Ennek a megjelenésnek egy másik fontos momentuma a mozgás, ami főként az utóbbi két programtípusnál kap jelentős szerepet. Aprócska animációkkal hihetetlenül fel lehet dobni egy program tetszetősségét, azonban arra is vigyázni kell, hogy ne vigyük túlzásba a dolgot, mivel az már idegesítő lehet egyesek számára.

Mi is az az animáció ?

Nem más, mint képek egymásutánja, amelyek között valamilyen kapcsolat van. Ez legtöbbször egy mozgás, ami a legegyszerűbbtől egészen bonyolult is lehet. Gondoljunk csak bele, hogy mennyivel másabb egy *help*, ha miközben írja ki a gép a szöveget a képre, egy száj is tátog a képernyőn, vagy egyéb ötletes megoldások, mint például az oldalváltáskor látható a lapozás stb.

Az animációnak alapvetően két csoportja van. Az egyik, amit abban a pillanatban számol ki a gép, a másik, amikor csak előre eltárolt képeket jelenítünk meg egymás után (ez az egyszerűbb).

Minden mozgásnak van azonban egy nagyon fontos kitétele, mégpedig az, hogy a képváltásokat úgy kell megoldani, hogy a felhasználó mindíg folyamatosan mozgó ábrát lásson. Tehát az a bizonyos *vertical blank* időzítés itt kap igen nagy szerepet, mivel, ha ezt figyelmen kívül hagyjuk, nincs az a vak felhasználó aki észre ne venné a törésvonalakat. Van azonban ezzel egy kis probléma, mégpedig az, hogy ezek szerint egy kép kirakásának bele kell férnie egy visszafutási időbe, ami az egyre modernebb monitoroknál igen csekély idő. Tehát két dolgot tehetünk: vagy kis ábrákkal dolgozunk (kb. 50x50 képpont) vagy más módszert választunk. Nos ez a más megoldás az, hogy nem a látható képre rajzolunk, hanem egy másik lapra a háttérben, - mivel a video-kártyákon álltalában van elegendő memória -, és ha kirajzoltuk, akkor "egyszerűen"

megcseréljük a két ábrát. Ezzel elérjük azt, hogy tetszőleges méretű képet animálhatunk (legfeljebb egy kicsit lassú lesz ...Pentium...) mivel a szinkron visszafutási időbe csupán a cserének kell belefférnie.

Ez a csere azonban mégsem olyan egyszerű, mivel még egy **rep movsw** utasítással sem lehet (kivéve a gyorsabb gépeken) egy teljes képet átmozgatni az egyik helyről a másikra. Tehát más megoldást kell keresni. A PC-n lehetőség nyílik arra, hogy eltoljuk a képernyő kezdőcímét egy adott értékkel. Ennek az eltolásnak a növelésével egyszerűen megoldhatunk például egy felfelé scroll rutint. Ha azonban nem csak egy sorral, hanem egy egész szegmenssel toljuk el azt, akkor elértük, hogy a kép nem a 0a000h, hanem a 0b000h szegmenscímen fog kezdődni. Így megtehetjük azt, hogy amíg a felhasználó a 0a000h szegmenscímen lévő képet látja, mi megrajzoljuk a 0b000h-n lévőt majd a pointert ide állítjuk, illetve tesszük ezt fordítva.

Ezért a kezdőcímért a 3d4h port 0ch-0dh regisztere felelős. Azonban előbb be kell állítani, hogy a videomemóriánk (legalább) 128 Kb, hogy elférjen a két szegmens. Valamint a 0b000h címen kezdődő lapot fel kell tölteni nullákkal (így kitöröltük a fölösleges adatokat a képről).

A bemutatásra kerülő program kiválóan alkalmas akár képernyővédőnek is. Érdekesség továbbá, hogy nincs betöltve egyszerre az összes frame, hanem egyszerre csak egy. Ennek lényege, hogy így akármilyen hosszú animáció lejátszható kellően gyors winchesterrel vagy cache program segítségével. Fontos (célszerű) egy animációnál, hogy az egyes képek azonos színeket használjanak, azaz ne kelljen palettát válltani az egyes képek között, mivel ezt csak a visszafutási időbe tehetnénk meg, mert ez azonnali változást eredményez.

Az itt bemutatott program egy gömb körül forgat három gyűrűt ellentétes tengelyek körül, és közben az egész ábrát mozgatja a képen, mintha pattogna a kép szélei között.

[Program 14]

```
.286
prog14      segment                     ;Szegmensdefiníció.
            assume cs:prog14,ds:prog14  ;A szegmensregiszterek
```

```
                                    ;hozzárendelése a kódhoz.

start:  mov   ax,prog14             ;Ds beállítása a kódlapra.
        mov   ds,ax

        mov   ax,13h                ;320*200/256 üzemmód
        int   10h                   ;beállítása.

        mov   dx,3ceh               ;A videómemória címtartományát
        mov   al,6                  ;128kb-ra állítja úgy, hogy a
        out   dx,al                 ;regiszter többi bitjét nem
        inc   dx                    ;válltoztatja meg.
        in    al,dx
        and   al,0011b
        out   dx,al

        mov   ax,0b000h             ;A 0b000h címen kezdődő
        mov   cx,320*200            ;szegmens kinullázása.
        xor   di,di
        mov   es,ax
        mov   al,0
        rep   stosb

        mov   ax,0a000h             ;* Inkompatibilitás esetén
        mov   es,ax                 ;* aktíválandó


        mov   dx,3d4h               ;A kijelzés beállítása a
        mov   al,0ch                ;normál címtől kezdődően.
        out   dx,al
        inc   dx
        mov   al,0
        out   dx,al

        mov   ax,3d00h              ;A színpaletta-file betöltése
        mov   dx,offset cfile
        int   21h
        push  ax
        mov   bx,ax

        mov   dx,offset cmap        ;a cmap címkéhez.
        mov   ax,3f00h
        mov   cx,768
        int   21h
```

```
                pop   bx
                mov   ax,3e00h
                int   21h

                call  cset              ;A paletta beállítása.

.1_prog14:      mov   ax,offset names   ;Az offset1 változót a képek
                mov   word ptr [offset1],ax ;nevét tartalmazó memória-
                                        ;rekesz kezdőcímére állítjuk.

                mov   cx,18             ;Az animáció 18 frame-ből áll.

load:           push  cx                ;A megfelelő képfile megnyitása,
                mov   dx,word ptr [offset1]
                mov   ax,3d00h
                int   21h
                push  ax
                mov   bx,ax

                mov   dx,offset file    ;betöltése,
                mov   ax,3f00h
                mov   cx,10240
                int   21h

                pop   bx                ;és lezárása.
                mov   ax,3e00h
                int   21h

                add   word ptr [offset1],11 ;A pointert a következő névre
                                        ;állítjuk.

                call  vertbl            ;Vertical blank időzítés.
                call  kirako            ;A betöltött kép kitömörítése.


                call  change            ;A kijelzés kezdőcímének
                                        ;megváltoztatása.

                                        ;* Inkompatibilitás esetén
                                        ;* törlendö

                call  vertbl            ;Verical blank időzítés.
```

```
                mov   ax,word ptr [sgn_x]     ;Az x koordináta növelése
                add   word ptr [koord_x],ax   ;(csökkentése) eggyel és
                cmp   word ptr [koord_x],175  ;a szélső helyzetek figyelése.
                jnz   .2_prog14               ;Ha az ábra a kép széléhez
                neg   ax                      ;ér, negálja a mozgási irányt.
                mov   word ptr [sgn_x],ax
.2_prog14:      cmp   word ptr [koord_x],5
                jnz   .3_prog14
                neg   ax
                mov   word ptr [sgn_x],ax

.3_prog14:      mov   ax,word ptr [sgn_y]     ;Ugyanaz, mint az x
                add   word ptr [koord_y],ax   ;koordináta esetén, csak más
                cmp   word ptr [koord_y],90   ;végértékekkel.
                jnz   .4_prog14
                neg   ax
                mov   word ptr [sgn_y],ax
.4_prog14:      cmp   word ptr [koord_y],0
                jnz   .5_prog14
                neg   ax
                mov   word ptr [sgn_y],ax

.5_prog14:      pop   cx                      ;A billentyűzet figyelés.
                mov   ax,100h
                int   16h                     ;Kilépés, ha van lenyomott
                jnz   end                     ;billentyű,

                loop  load                    ;egyébként a program folytatása.
                jmp   .1_prog14


end:            xor   ax,ax                   ;A lenyomott gomb kódjának
                int   16h                     ;kiolvasása.

                mov   ax,3                    ;80*25 karakteres mód beállítása.
                int   10h

                mov   ax,4c00h                ;Kilépés a DOS-ba.
                int   21h

;********** RUTINOK **********
```

```
kirako      proc

            mov   si,offset file          ;A kitömörítendő file címe.

            mov   ax,word ptr [koord_y]   ;A kirakási pozíció bal felső
            mov   bx,320                  ;sarkának címét di regiszterbe
            mul   bx                      ;helyezi.
            mov   di,ax
            add   di,word ptr [koord_x]

            mov   cx,110                  ;Az alakzat 110 pont magas
.1_kirako:  push  cx di
            mov   dl,140                  ;és 140 pont széles.

.2_kirako:  mov   cl,[si]
            inc   si
            test  cl,128
            jz    .3_kirako

            neg   cl
            inc   cl
            sub   dl,cl
            mov   al,[si]
            inc   si
            rep   stosb                   ;A file kitömörítése...
            jmp   .4_kirako

.3_kirako:  inc   cl
            sub   dl,cl
            rep   movsb

.4_kirako:  or    dl,dl
            jnz   .2_kirako
            pop   di cx
            add   di,320
            loop  .1_kirako

            mov   ax,es                   ;A videomemória szegmenscímét
            xor   ax,1000h                ;0a000-ról 0b000-ra illetve
            mov   es,ax                   ;fordítva változtatja.

                                          ;* Inkompatibilitás esetén
                                          ;* törlendö
```

```
            ret

kirako      endp


cset        proc

            mov   si,offset cmap        ;A megfelelő palettaszínek
            mov   dx,3c8h               ;beállítása.
            mov   al,0
            mov   cx,256

.1_cset:    push  ax
            out   dx,al
            inc   dx

            mov   al,[si]
            shr   al,2                  ;Mivel a színek 0-255-ig
            out   dx,al                 ;változó értékkel lettek
            mov   al,[si+1]             ;letárolva, ezért a beállítás
            shr   al,2                  ;során el kell őket tolni 2
            out   dx,al                 ;bittel, azaz osztani kell
            mov   al,[si+2]             ;őket néggyel.
            shr   al,2
            out   dx,al

            dec   dx
            pop   ax
            inc   al
            add   si,3
            loop  .1_cset
            ret

cset        endp


vertbl      proc

            mov   dx,3dah               ;Vertical blank időzítés.
.1_vb:      in    al,dx
            test  al,8
            jz    .1_vb
```

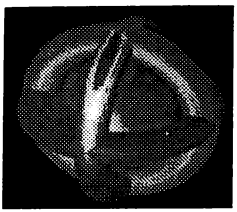
```
.2_vb:      in      al,dx
            test    al,8
            jnz     .2_vb

.3_vb:      in      al,dx
            test    al,8
            jz      .3_vb

            ret

vertbl      endp


change      proc

            mov     dx,3d4h             ;A 3d4h port 0ch regisztere
            mov     al,0ch              ;határozza meg a megjelenítés
            out     dx,al               ;kezdőcímének magasabb
            inc     dx                  ;helyiértékű byte-ját.
            mov     al,40h              ;Ha ezt xor-oljuk 40h-val akkor
            xor     byte ptr [select],al ;pont egy képernyőnyi (64K)
            mov     al,byte ptr [select] ;értékkel változtatjuk meg azt.
            out     dx,al               ;Így lehet a két képernyő
            ret                         ;között lapozgatni.

change      endp


;********** ADATOK **********

koord_x:    dw      5
koord_y:    dw      0

sgn_x:dw    1
sgn_y:dw    1

select:     db      0

offset1:    dw      ?

names:      db      "gomb01.aga",0
            db      "gomb02.aga",0
            db      "gomb03.aga",0
            db      "gomb04.aga",0
```

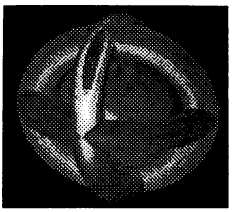
```
                db      "gomb05.aga",0
                db      "gomb06.aga",0
                db      "gomb07.aga",0
                db      "gomb08.aga",0
                db      "gomb09.aga",0
                db      "gomb10.aga",0
                db      "gomb11.aga",0
                db      "gomb12.aga",0
                db      "gomb13.aga",0
                db      "gomb14.aga",0
                db      "gomb15.aga",0
                db      "gomb16.aga",0
                db      "gomb17.aga",0
                db      "gomb18.aga",0

cfile:          db      "cmap.aga",0

cmap:           db      768 dup (0)             ;A színtérképnek lefoglalt hely.

file:           db      10240 dup (0)           ;A képnek lefoglalt hely.

prog14          ends                            ;A szegmens vége.
                end     start                   ;A program vége.
```

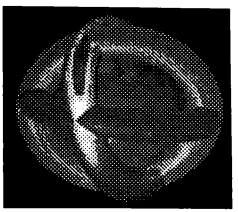
A program indulásakor elkövetjük azt a kis változtatást, amit írtam a program előtt, azaz engedélyezzük a 128KB-os video RAM kezelést, valamint az alap kijelzést a normál, de a szegmenscímet a második lapra állítjuk. Ezután betöltjük a színpalettát tartalmazó 768 byte-os file-t és a **cset** rutinnal beállítjuk a megfelelő színeket. Ezután egy 18-as ciklussal egyenként betöltjük az aktuális képet, kirakjuk a hátsó képre, *vertical blank* időzítés, a kijelzés megcserélése majd egy ismételt *VB* időzítés. A program sajnos eléggé videokártya specifikus. Ezért a forráskódban *-al jelölt részeken módosítani kell, ha az animáció nem megfelelő (vibrál, darabos stb). Ezzel a dupla képernyő használatát tiltjuk le.

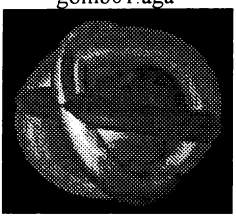
A program legfontosabb pontja a két kép megfelelő időben történő váltása. Ezt a megfelelő *vertical blank* időzítéssel lehet elérni.

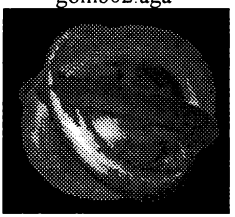
Az egyes frame-ek külön állományban lettek letárolva a lemezen:

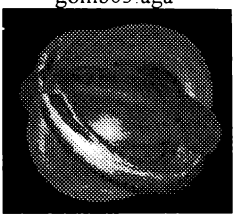

gomb01.aga

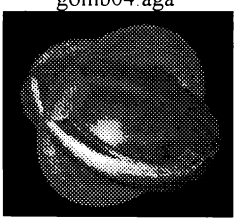

gomb02.aga

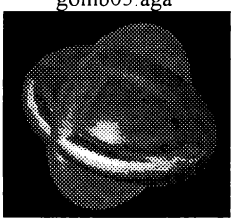

gomb03.aga

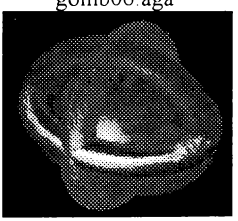

gomb04.aga

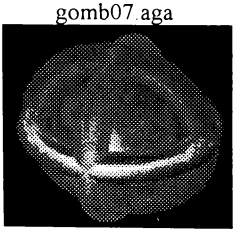

gomb05.aga

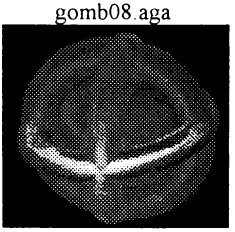

gomb06.aga

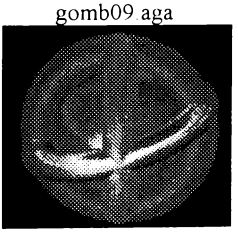

gomb07.aga

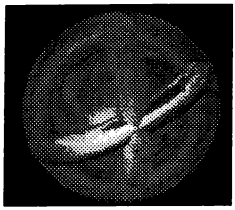

gomb08.aga


gomb09.aga

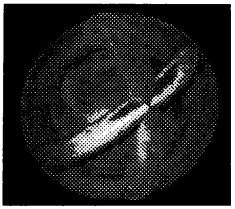

gomb10.aga


gomb11.aga

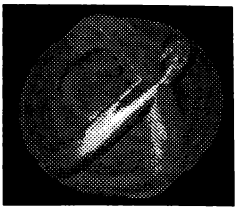

gomb12.aga



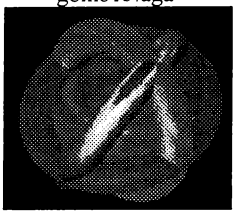
gomb13.aga



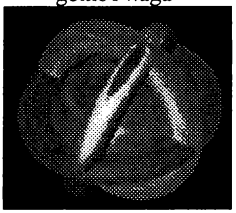
gomb14.aga



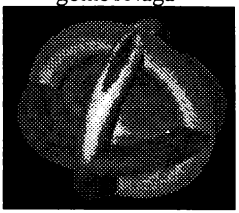
gomb15.aga



gomb16.aga



gomb17.aga



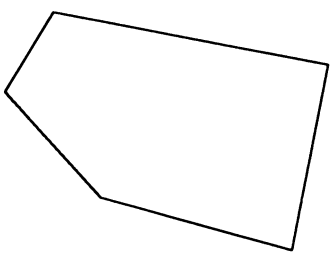
gomb18.aga

7. Fejezet

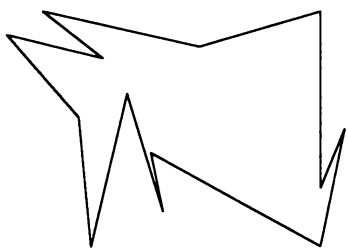
Egy zárt alakzat kifestése egy színnel

Zártnak nevezzük azt az alakzatot, melyet megszakítás nélküli folytonos vonal határol. Az ilyen alakzatok kifeshetők, ami a körvonalon belüli rész kitöltését jelenti egy megadott színnel. Ennek több módja is van, az alakzat típusától, formájától függően.

Így megkülönböztetünk egyszerű és szilánkos alakzatokat. A szilánkos alakzatoknak a legfontosabb ismérve, hogy egy lépcsőben nem lehet kifesteni őket. De milyenek is azok a normál és szilánkos alakzatok? Ezt a következő két ábrából könnyen megérthetjük:



Egyszerű alakzat



Szilánkos alakzat

Egyszerű alakzatok kifestése

Egy egyszerű alakzatot is többféle módszerrel lehet kitölteni. Az ismertebb módszer az, amikor megadunk egy pontot az alakzat belsejében és innen elindulva töltődik fel az alakzat. Hogy is működik ez? Induljunk ki abból, hogy van egy pont, amiről valaki azt állítja, hogy az egy alakzaton belül van.

Az első feladat, hogy megnézzük ennek a pontnak a színét, mivel az ilyen színű pontokat kell majd kifesteni, az ettől eltérő színű valószínűleg az alakzat határoló vonala, vagy egy másik alakzat.

Tehát ott tartottunk, hogy van egy pontunk, amelyiknek tudjuk a színét, és tudjuk róla, hogy ki kell cserélni azt egy előre megadott színre (cseréljük ki...). Ezek után meg kell nézni, hogy alatta, illetve fölötte milyen színű pont található, mert ha olyan, amit az első pont kiolvasása során kaptunk, akkor az egy majdan kifestendő pont, tehát tároljuk el az aktuális pozíciót. Ezek után nézzük meg, hogy például jobbra milyen pont van. Ha olyan, amelyet meg kell változtatni, ugorjunk arra a pontra, és kezdjük elölről az itt leírtakat mindaddig, amíg jobbra találunk kitöltendő pontot. Ha elfogytak, ugorjunk vissza oda, ahonnan elkezdtük a sort, és csináljuk meg mindezt balra is. Ha ezzel is végeztünk, akkor jönnek azok az adatok, amiket a felfelé és lefelé lévő pontok figyelésénél letároltunk. Vegyük először az utoljára letárolt felfelé lehetőséget, és csináljuk végig mindazt amit az előző vonalnál tettünk, és folytassuk ezt mindaddig, amíg van lehetőség a felfelé mozgásra. Ha ezek elfogytak, akkor ismét vissza a kezdő sorhoz, és az utoljára letárolt lefelé lehetőségnél folytassuk az előbbiek szerint.

Hogy mindezt könnyebb legyen megérteni, a mos következő programba beépítettem egy ciklust, amely a kiszínezést jól nyomonkövethetővé teszi. Ezt géptípusnak megfelelően gyorsítani, vagy lassítani lehet (vagy kivenni). Ennek segítségével megfigyelhetjük a rutin pontos működését.

[Program 15]

```
prog15    segment                    ;Szegmensdefiníció.
          assume cs:prog15,ds:prog15 ;A szegmensregiszterek
                                     ;hozzárendelése a kódhoz.

start:    push  cs                   ;Cs áttöltése ds-be a vermen
          pop   ds                   ;keresztül.

          mov   ax,0a000h            ;A videomemória kezdőcímének
          mov   es,ax                ;beállítása es regiszterbe.

          mov   ax,13h               ;320x200/256 grafikusd mód
          int   10h                  ;beállítása.

          call  box                  ;A szabálytalan négyszög
                                     ;felrajzolása a képernyőre.
```

```
                call    fill                    ;Az alakzat kitöltése.

                xor     ax,ax                   ;Billentyűvárás.
                int     16h

                mov     ax,3                    ;Text üzemmód visszaállítása.
                int     10h

                mov     ah,4ch                  ;Visszatérés a DOS-hoz.
                int     21h

;********** RUTINOK **********

fill            proc

                mov     di,word ptr [fill_x]    ;A kitöltés kezdőpontjának
                mov     ax,word ptr [fill_y]    ;képernyőcímét di regiszterbe
                mov     bx,320                  ;teszi.
                mul     bx
                add     di,ax

                mov     al,es:[di]              ;Kiolvassa ennek a pontnak a
                                                ;színét, mivel az ilyen színü
                                                ;pontokat kell kifesteni a
                                                ;határolóvonalon belül.

                mov     ah,byte ptr [color]     ;A kitöltés színét ah-ba teszi.

                xor     dx,dx                   ;Regiszterek nullázása.
                xor     bx,bx

.1_fill:        push    di                      ;Az aktuális di letárolása.

.2_fill:        push    bx                      ;A soron belüli pozició tárolása.

.3_fill:        mov     es:[di+bx],ah           ;Az adott pont kiszinezése a
                                                ;kitöltő szinre.

                mov     cx,02fffh               ;Lassító ciklus.
.1_wait:        loop    .1_wait
```

```
        cmp   es:[di-320+bx],al     ;Megvizsgálja, hogy az aktuális
        jnz   .4_fill               ;pont fölötti pozícióban milyen
        mov   si,bx                 ;színű pont található, és ha a pont
        mov   dl,255                ;majdan kifestendő, eltárolja
                                    ;az aktuális sorpozíciót, és
                                    ;a dl regisztert 255-re állítja.

.4_fill:cmp   es:[di+320+bx],al     ;Az aktuális pozíció alatti pont
        jnz   .5_fill               ;megvizsgálása az előzőekhez
        mov   bp,bx                 ;hasonlóan.
        mov   dh,255
.5_fill:inc   bx
        cmp   es:[di+bx],al
        jz    .3_fill

        pop   bx                    ;A sorkitöltés kezdőpontjától
.6_fill:mov   es:[di+bx],ah         ;balra lévő pontok kitöltése.

        mov   cx,02fffh             ;Lassítás.
.2_wait:loop  .2_wait

        cmp   es:[di-320+bx],al
        jnz   .7_fill
        mov   si,bx
        mov   dl,255
.7_fill:cmp   es:[di+320+bx],al
        jnz   .8_fill
        mov   bp,bx
        mov   dh,255
.8_fill:dec   bx
        cmp   es:[di+bx],al
        jz    .6_fill

        cmp   dl,0                  ;Figyeli, hogy volt-e még
        jz    .9_fill               ;felfelé kitöltetlen pont. Ha
        xor   dl,dl                 ;igen, törli dl-t és a di:bx
        sub   di,320                ;értékét erre a pozícióra állítja.
        mov   bx,si                 ;majd visszaugrik a sorkitöltő
        jmp   .2_fill               ;rutinhoz.

.9_fill:pop   di                    ;Ha elfogytak a felfelé lévő
        cmp   dh,0                  ;kitöltendő helyek, akkor megpróbálja
        jz    .10_fill              ;lefelé folytatni hasonlóan mint
        xor   dh,dh                 ;előbb.
```

```
                add   di,320
                mov   bx,bp
                jmp   .1_fill

.10_fill:       ret

fill            endp


box             proc

                mov   word ptr [xa],100      ;A megfelelő pontok összekötése.
                mov   word ptr [ya],50
                mov   word ptr [xb],190
                mov   word ptr [yb],30
                call  line
                mov   word ptr [xa],180
                mov   word ptr [ya],100
                call  line
                mov   word ptr [xb],90
                mov   word ptr [yb],110
                call  line
                mov   word ptr [xa],100
                mov   word ptr [ya],50
                call  line
                ret

box             endp


line            proc                         ;A vonalhúzó rutin kezdete.

                mov   bx,word ptr [xa]       ;A vonal kezdőpontjának és
                mov   ax,word ptr [xb]       ;végpontjának x koordinátáját
                                             ;beírja az ax,bx regiszterekbe.

                mov   dx,1                   ;A vízszintes lépésköz.

                sub   ax,bx                  ;Kiszámítja a két pont közötti
                                             ;távolságot.

                jnc   .1_line                ;Ha ez nem negatív, átugorja
                                             ;a következő részt.
```

```
                neg   ax                    ;A távolság valódi értékének a
                                             ;kiszámítása.

                mov   dx,65535               ;A vízszintes lépésközt -1-re
                                             ;állítja.

.1_line:        mov   word ptr [delta_x],ax  ;A regisztereket beírja a
                mov   word ptr [pont_x],bx   ;memóriaváltozókba.
                mov   word ptr [sgn_x],dx

                mov   bx,word ptr [ya]       ;Ugyanaz, ami a vízszintes
                mov   ax,word ptr [yb]       ;értékeknél történt.
                mov   dx,1
                sub   ax,bx
                jnc   .2_line
                neg   ax
                mov   dx,65535

.2_line:        mov   word ptr [delta_y],ax
                mov   word ptr [pont_y],bx
                mov   word ptr [sgn_y],dx

                cmp   ax,word ptr [delta_x]  ;Eldönti, hogy a vonal milyen
                                             ;irányú.

                jc    .1_vizsz               ;Ha a vízszintes összetevője
                                             ;a nagyobb, arra az eljárásra
                                             ;ugrik.

                mov   dx,ax                  ;Egyébként az algoritmus
                shr   dx,1                   ;növekményét számláló
                                             ;regiszterbe beírja a vonal
                                             ;vízszintes hosszának a felét.

;fuggoleges

                mov   cx,ax                  ;A ciklusszámlálóba pedig a
                inc   cx                     ;hossznál eggyel többet.

.1_fuggo:       push  dx                     ;Számláló elmentése.

                call  pont                   ;Az aktuális pont kirakása.
```

```
                pop   dx
                mov   ax,word ptr [sgn_y]      ;Az algoritmus szerinti
                add   word ptr [pont_y],ax     ;következő pont koordinátainak

                add   dx,word ptr [delta_x]    ;kiszámítása.
                cmp   dx,word ptr [delta_y]
                jc    .2_fuggo

                sub   dx,word ptr [delta_y]
                mov   ax,word ptr [sgn_x]
                add   word ptr [pont_x],ax
.2_fuggo:       loop  .1_fuggo

                ret

;vizszintes

.1_vizsz:       mov   cx,word ptr [delta_x]    ;Hasonlóan mint a függőleges
                mov   dx,cx                    ;esetében történt, itt is
                inc   cx                       ;azok a műveletek hajtódnak
                shr   dx,1                     ;végre, csak az ellenkező

.2_vizsz:       push  dx                       ;koordinátákkal.
                call  pont
                pop   dx
                mov   ax,word ptr [sgn_x]
                add   word ptr [pont_x],ax
                add   dx,word ptr [delta_y]
                cmp   dx,word ptr [delta_x]
                jc    .3_vizsz

                sub   dx,word ptr [delta_x]
                mov   ax,word ptr [sgn_y]
                add   word ptr [pont_y],ax
.3_vizsz:       loop  .2_vizsz

                ret

line            endp


pont            Proc

                mov   di,word ptr [pont_x]
```

```
                mov   ax,word ptr [pont_y]
                mov   bx,320
                mul   bx
                add   di,ax
                mov   al,7
                mov   es:[di],al

                ret

pont            Endp

;********** ADATOK **********

pont_x:         dw    ?                 ;A kirakandó pont xy
pont_y:         dw    ?                 ;koordinátája.

xa:             dw    0                 ;A vonal kezdőpontjának
ya:             dw    0                 ;koordinátái.

xb:             dw    120               ;A vonal végpontjának
yb:             dw    175               ;koordinátai.

delta_x:        dw    ?                 ;A vonal vízszintes mérete.
delta_y:        dw    ?                 ;A vonal függőleges mérete.

sgn_x:          dw    ?                 ;A vízszintes lépésköz és
                                        ;iránya.

sgn_y:          dw    ?                 ;A függőleges lépésköz és
                                        ;iránya.

fill_x:         dw    110               ;A kitöltés kezdő koordinátái.
fill_y:         dw    70

color:          db    3                 ;A kitöltő szín értéke.

prog15          ends                    ;A szegmens vége.
                end   start             ;A program vége.
```

A program egy, az IBM PC Gyakorlati Assembly című könyvben már ismertetett vonalrajzoló eljárás felhasználásával kirajzol egy

szabálytalan, de egyszerű alakzatot, majd a megadott kezdő koordinátákból kiszámolja a kezdőpont címét. Ezután következik a már leírt színkitöltési folyamat oly módon, hogy a felfelé illetve lefelé levő kitöltendő pont koordinátákat az **si** illetve **bp** regiszterekben tárolja, és mindig felülírja azokat a következő pont koordinátáival.

Egyszerű alakzatok kitöltése másként

Az előző módszernél gyorsabb, azonban nem mindenhol alkalmazható módszer, hogy egy téglalap alakú terület egyik sarkának pozícióját és magasságát adjuk meg kiinduló értékként. Fontos, hogy a függőleges pozíció megegyezzen a legfelső (legalsó) kitöltendő pont y koordinátájával, és célszerű a vízszintes pozíciót a legszélső még nem kifestendő pont-1 x koordinátájára venni. Legyen most ez például a bal felső sarok. Innen elindulva jobbra, megkeressük az első olyan pontot amelyik színe megegyezik a megadott ábra keretszínével. Ha ezt megtaláltuk, ezt át kell ugorni, azaz tovább kell lépkedni mindaddig, amíg kitöltendő pontot nem találunk. Innen elkezdődhet a pontok kirakása mindaddig, míg újból keret színű ponttal nem találkozunk. Ezután megismételjük az eljárást az eggyel alatta lévő soron is, folytatva ezt a megadott magasság értékének megfelelően.

[Program 16]

```
prog16    segment                    ;Szegmensdefiníció.
          assume cs:prog16,ds:prog16 ;A szegmensregiszterek
                                     ;hozzárendelése a kódhoz.

start:    push  cs                   ;Cs áttöltése ds-be a vermen
          pop   ds                   ;keresztül.

          mov   ax,0a000h            ;A videomemória kezdőcímének
          mov   es,ax                ;beállítása es regiszterbe.

          mov   ax,13h               ;320x200/256 grafikusd mód
          int   10h                  ;beállítása.

          call  box                  ;A szabálytalan négyszög
                                     ;felrajzolása a képre.
```

```
            call    fill                    ;Az alakzat kitöltése.

            xor     ax,ax                   ;Billentyűvárás.
            int     16h

            mov     ax,3                    ;Text üzemmód visszaállítása.
            int     10h

            mov     ah,4ch                  ;Visszatérés a DOS-hoz.
            int     21h

;********** RUTINOK **********

fill        proc

            mov     di,word ptr [fill_x]    ;A megadott kezdőpont koordináta
            mov     ax,word ptr [fill_y]    ;memóriacímének kiszámítása.
            mov     bx,320
            mul     bx
            add     di,ax

            mov     cx,word ptr [height]    ;Az alakzat magassága.

.1_fill:    push    di
            xor     bx,bx
            mov     al,byte ptr [keret]

.2_fill:    cmp     es:[di+bx],al           ;A keretvonal kezdetének
            jz      .3_fill                 ;megkeresése.
            inc     bx
            jmp     .2_fill

.3_fill:    cmp     es:[di+bx],al           ;Az első kitöltendő pont
            jnz     .4_fill                 ;keresése.
            inc     bx
            jmp     .3_fill

.4_fill:    mov     ah,byte ptr [color]     ;Kitöltés mindaddig, amíg
.5_fill:    mov     es:[di+bx],ah           ;keret színű ponttal nem
            inc     bx                      ;találkozik.
            cmp     es:[di+bx],al
            jnz     .5_fill

            pop     di                      ;A következő sor kitöltése.
```

```
            add   di,320
            loop  .1_fill

            ret

fill        endp


box         proc

            mov   word ptr [xa],100      ;A megfelelő pontok összekötése.
            mov   word ptr [ya],50
            mov   word ptr [xb],190
            mov   word ptr [yb],30
            call  line
            mov   word ptr [xa],180
            mov   word ptr [ya],100
            call  line
            mov   word ptr [xb],90
            mov   word ptr [yb],110
            call  line
            mov   word ptr [xa],100
            mov   word ptr [ya],50
            call  line
            ret

box         endp


line        proc                         ;A vonalhúzó rutin kezdete.

            mov   bx,word ptr [xa]       ;A vonal kezdőpontjának és
            mov   ax,word ptr [xb]       ;végpontjának x koordinátáját
                                         ;beírja az ax,bx regiszterekbe.

            mov   dx,1                   ;A vízszintes lépésköz.

            sub   ax,bx                  ;Kiszámítja a két pont közötti
                                         ;távolságot.

            jnc   .1_line                ;Ha ez nem negatív, átugorja
                                         ;a következő részt.

            neg   ax                     ;A távolság valódi értékének a
```

```
                                        ;kiszámítása.

        mov   dx,65535                  ;A vízszintes lépésközt -1-re
                                        ;állítja.

.1_line: mov  word ptr [delta_x],ax     ;A regisztereket beírja a
         mov  word ptr [pont_x],bx      ;memóriaváltozókba.
         mov  word ptr [sgn_x],dx

         mov  bx,word ptr [ya]          ;Ugyanaz, ami a vízszintes
         mov  ax,word ptr [yb]          ;értékeknél történt.
         mov  dx,1
         sub  ax,bx
         jnc  .2_line
         neg  ax
         mov  dx,65535

.2_line: mov  word ptr [delta_y],ax
         mov  word ptr [pont_y],bx
         mov  word ptr [sgn_y],dx

         cmp  ax,word ptr [delta_x]     ;Eldönti, hogy a vonal milyen
                                        ;irányú.

         jc   .1_vizsz                  ;Ha a vízszintes összetevője
                                        ;a nagyobb, arra az eljárásra
                                        ;ugrik.

         mov  dx,ax                     ;Egyébként az algoritmus a
         shr  dx,1                      ;növekményt számláló
                                        ;regiszterbe beírja a vonal
                                        ;vízszintes hosszának a felét.

;fuggoleges

         mov  cx,ax                     ;A ciklusszámlálóba pedig a
         inc  cx                        ;hossznál eggyel többet.

.1_fuggo: push dx                       ;A számláló elmentése.

         call pont                      ;Az aktuális pont kirakása.

         pop  dx
         mov  ax,word ptr [sgn_y]       ;Az algoritmus szerinti következő
```

```
            add   word ptr [pont_y],ax      ;pont koordinátainak kiszámítása

            add   dx,word ptr [delta_x]
            cmp   dx,word ptr [delta_y]
            jc    .2_fuggo

            sub   dx,word ptr [delta_y]
            mov   ax,word ptr [sgn_x]
            add   word ptr [pont_x],ax
.2_fuggo:   loop  .1_fuggo

            ret

;vizszintes

.1_vizsz:   mov   cx,word ptr [delta_x]   ;Hasonlóan, mint a függőleges
            mov   dx,cx                   ;eseteben történt, itt is azok
            inc   cx                      ;a műveletek hajtódnak végre,
            shr   dx,1                    ;csak az ellenkező koordinátákkal

.2_vizsz:   push  dx
            call  pont
            pop   dx
            mov   ax,word ptr [sgn_x]
            add   word ptr [pont_x],ax
            add   dx,word ptr [delta_y]
            cmp   dx,word ptr [delta_x]
            jc    .3_vizsz

            sub   dx,word ptr [delta_x]
            mov   ax,word ptr [sgn_y]
            add   word ptr [pont_y],ax
.3_vizsz:   loop  .2_vizsz

            ret

line        endp


pont        Proc

            mov   di,word ptr [pont_x]
            mov   ax,word ptr [pont_y]
            mov   bx,320
```

```
            mul   bx
            add   di,ax
            mov   al,7
            mov   es:[di],al

            ret

pont        Endp

;********** ADATOK **********

pont_x:     dw    ?             ;A kirakandó pont xy
pont_y:     dw    ?             ;koordinátája.

xa:         dw    0             ;A vonal kezdőpontjának
ya:         dw    0             ;koordinátái.

xb:         dw    120           ;A vonal végpontjának
yb:         dw    175           ;koordinátái.

delta_x:    dw    ?             ;A vonal vízszintes mérete.
delta_y:    dw    ?             ;A vonal függőleges mérete.

sgn_x:      dw    ?             ;A vízszintes lépésköz és
                                ;iránya.

sgn_y:      dw    ?             ;A függőleges lépésköz és
                                ;iránya.

fill_x:     dw    90            ;A kitöltés kezdő koordinátái.
fill_y:     dw    31

height:     dw    79            ;Az alakzat magassága.

color:      db    3             ;A kitöltő szín értéke.

keret:      db    7             ;Az alakzat határvonalának
                                ;színe.

prog16      ends                ;A szegmens vége.
            end   start         ;A program vége.
```

Szilánkos alakzatok kitöltése

Nos az előbbiekben láttuk az egyszerű alakzatokra alkalmazható fill eljárásokat. Ezeken kívül bizonyára jó pár algoritmus van ezen feladatok megoldására, de az elmélet megértéséhez elegendőnek érzem eme kettő bemutatását. És most térjünk át a bonyolultabb, szilánkos alakzatokra. Ezeknek olyan visszahajló, eldugott zugai vannak, amit az előbbi algoritmusok nem képesek teljes mértékben kifesteni. A 17. program leginkább az elsőre hasonlít, annyi különbséggel, hogy nem csupán egy alsó és egy felső pozíciót tárol el, hanem amint egy lezárt területtel találkozik, ott újabb címet tárol le. Ezt úgy tudjuk ellenőrizni, hogy miközben figyeljük az alsó, és felső pontok színét, ha olyannal találkozunk, ahol valamelyik irányba nem lehet tovább haladni, akkor az azt jelenti, hogy vége az előző mezőnek, és amint újból lehet abban az irányban festeni, újabb pozíciót kell eltárolni. Erre a tárolásra a legmegfelelőbb a verem, mivel a pointert automatikusan kezeli adat behelyezésekor illetve kivételekor. Mindössze azt kell figyelnünk, hogy mikor fogynak el az értékek a veremből.

Egy egyszerűsítés is be lett építve a programba, ami ugyan egy picit lassítja a futást, azonban összességében lehet, hogy gyorsabb lesz tőle a program, mégpedig az, hogy nem jobbra-balra kezeljük a pontokat egy adott soron belül, hanem megkeressük a bal szélsőt és innen csak jobbra kell pásztáznunk. Amennyiben egy megjelölt ponttól előbb az egyik majd a másik irányba indulnánk el, az elég sok ellenőrzési problémát adna, mivel előfordulhatna, hogy egy mezőhöz két letárolt pozíció tartozik, és egyéb más gondokat is okozna. Egyébként a programban csak a kirajzolt alakzat változott a kifestő rutin mellett.

[Program 17]

```
prog17      segment                 ;Szegmensdefiníció.
            assume cs:prog17,ds:prog17  ;A szegmensregiszterek
                                    ;hozzárendelése a kódhoz.

start:      push  cs                ;Cs áttöltése ds-be a vermen
            pop   ds                ;keresztül.
```

```
                mov   ax,0a000h          ;A videomemória kezdőcímének
                mov   es,ax              ;beállítása es regiszterbe.

                mov   ax,13h             ;320x200/256 grafikus mód
                int   10h                ;beállítása.

                call  box                ;A szabálytalan sokszög
                                         ;felrajzolása a képre.

                call  fill               ;Az alakzat kitöltése.

                xor   ax,ax              ;Billentyűvárás.
                int   16h

                mov   ax,3               ;Text üzemmód visszaállítása.
                int   10h

                mov   ah,4ch             ;Visszatérés a DOS-hoz.
                int   21h

;********** RUTINOK **********

fill            proc

                mov   di,word ptr [fill_x]   ;A kitöltés kezdő pozíciójának
                mov   ax,word ptr [fill_y]   ;kiszámítása.
                mov   bx,320
                mul   bx
                add   di,ax

                mov   al,es:[di]         ;A háttérszín beolvasása.
                mov   ah,byte ptr [color]    ;A kitöltőszín beolvasása.

                xor   bx,bx              ;A segédregiszter nullázása.

                mov   si,sp              ;A veremmutató étrékének
                                         ;eltárolása.

.1_fill:        cmp   es:[di-1],al       ;Az adott sorban az első
                jnz   .2_fill            ;kitöltendő pont megkeresése.
                dec   di
                jmp   .1_fill
```

```
.2_fill:    mov   es:[di],ah          ;A kitöltőszín kirakása az
                                      ;adott es:di pozícióba.

            mov   cx,2fffh            ;Lassítás.
.1_wait:    loop  .1_wait


            cmp   es:[di-320],al      ;Az aktuális pozíció fölötti
            jz    .3_fill             ;pont megvizsgálása.
                                      ;Ha a pont kitöltendő, ugrik.

            xor   bl,bl               ;Ha nem, törli a segédregisztert,
            jmp   .4_fill             ;és ugrik a következő vizsgálatra.

.3_fill:    cmp   bl,0                ;Ha bl=0, akkor ez már egy új
                                      ;terület, tehát le kell tárolni
                                      ;az aktuális pozíció fölötti
                                      ;címet.

            jnz   .4_fill             ;Ha nem nulla, akkor már le lett
                                      ;tárolva az adott mezőbe egy
                                      ;pozíció, ezért átugorja a
                                      ;letárolást.

            mov   bl,255              ;A segédregiszter 255-be
                                      ;állításával jelezzük, hogy
                                      ;le lett tárolva egy pozíció.

            sub   di,320              ;Az aktuális pont fölötti
            push  di                  ;cím letárolása a verembe.
            add   di,320

.4_fill:    cmp   es:[di+320],al      ;Hasonlóan az akatuális pont
            jz    .5_fill             ;fölötti pont figyeléséhez
            xor   bh,bh               ;itt az eljárás az alsó szomszédos
            jmp   .6_fill             ;pozícióval ismétlődik meg

.5_fill:    cmp   bh,0
            jnz   .6_fill
            mov   bh,255
            add   di,320
            push  di
            sub   di,320
```

```
.6_fill:    cmp   es:[di+1],al
            jnz   .7_fill
            inc   di
            jmp   .2_fill

.7_fill:    cmp   si,sp                ;Ha a stack pointer és az si
            jz    .9_fill              ;regiszter megegyezik egymással,
                                       ;az azt jelenti, hogy az alakzat
                                       ;ki van töltve, azaz nincs több
                                       ;kitöltendő mező.

            pop   cx                   ;Kiolvassuk az utoljára letárolt
            cmp   cx,di                ;pozíciót, és összehasonlítjuk
            jnc   .8_fill              ;az aktuális di értékkel. Ha a
                                       ;di értéke kisebb, a bh, ha
                                       ;nagyobb, a bl regisztert kell
                                       ;nullára állítani, hogy lehetőség
                                       ;legyen újabb pozíciók
                                       ;letárolására.

            xor   bl,bl
            mov   di,cx
            jmp   .1_fill
.8_fill:    xor   bh,bh
            mov   di,cx
            jmp   .1_fill

.9_fill:    ret

fill        endp


box         proc

            mov   word ptr [xa],100    ;A megfelelő pontok összekötése.
            mov   word ptr [ya],40
            mov   word ptr [xb],150
            mov   word ptr [yb],50
            call  line

            mov   word ptr [xa],190
            mov   word ptr [ya],45
            call  line
```

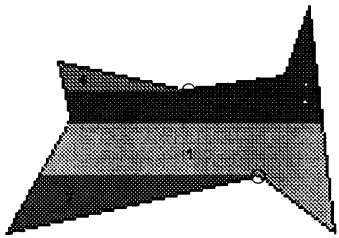
```
        mov   word ptr [xb],200
        mov   word ptr [yb],20
        call  line

        mov   word ptr [xa],210
        mov   word ptr [ya],100
        call  line

        mov   word ptr [xb],180
        mov   word ptr [yb],80
        call  line

        mov   word ptr [xa],80
        mov   word ptr [ya],100
        call  line
        mov   word ptr [xb],105
        mov   word ptr [yb],60
        call  line
        mov   word ptr [xa],100
        mov   word ptr [ya],40
        call  line
        ret

box     endp


line    proc                    ;A vonalhúzó rutin kezdete.

        mov   bx,word ptr [xa]  ;A vonal kezdőpontjának és
        mov   ax,word ptr [xb]  ;végpontjának x koordinátáját
                                ;beírja ax,bx regiszterekbe.

        mov   dx,1              ;A vízszintes lépésköz.

        sub   ax,bx             ;Kiszámítja a két pont közötti
                                ;távolságot.

        jnc   .1_line           ;Ha ez nem negatív, átugorja
                                ;a következő részt.

        neg   ax                ;A távolság valódi értékének a
                                ;kiszámítása.
```

```
                mov   dx,65535              ;A vízszintes lépésközt -1-re
                                            ;állítja.

.1_line:        mov   word ptr [delta_x],ax ;A regisztereket beírja a
                mov   word ptr [pont_x],bx  ;memóriaváltozókba.
                mov   word ptr [sgn_x],dx

                mov   bx,word ptr [ya]      ;Ugyanaz, ami a vízszintes
                mov   ax,word ptr [yb]      ;értékeknél történt.
                mov   dx,1
                sub   ax,bx
                jnc   .2_line
                neg   ax
                mov   dx,65535

.2_line:        mov   word ptr [delta_y],ax
                mov   word ptr [pont_y],bx
                mov   word ptr [sgn_y],dx

                cmp   ax,word ptr [delta_x] ;Eldönti, hogy a vonal milyen
                                            ;irányú.

                jc    .1_vizsz              ;Ha a vízszintes összetevője
                                            ;a nagyobb, arra az eljárásra
                                            ;ugrik.

                mov   dx,ax                 ;Egyébként az algoritmus
                shr   dx,1                  ;növekményét számláló
                                            ;regiszterbe beírja a vonal
                                            ;vízszintes hosszának a felét.

;fuggoleges

                mov   cx,ax                 ;A ciklusszámlálóba pedig a
                inc   cx                    ;hossznál eggyel többet.

.1_fuggo:       push  dx                    ;A számláló elmentése.

                call  pont                  ;Az aktuális pont kirakása.

                pop   dx
                mov   ax,word ptr [sgn_y]   ;Az algoritmus szerinti következő
                add   word ptr [pont_y],ax  ;pont koordinátainak kiszámítása.
```

```
                add   dx,word ptr [delta_x]
                cmp   dx,word ptr [delta_y]
                jc    .2_fuggo

                sub   dx,word ptr [delta_y]
                mov   ax,word ptr [sgn_x]
                add   word ptr [pont_x],ax
.2_fuggo:       loop  .1_fuggo

                ret

;vizszintes

.1_vizsz:       mov   cx,word ptr [delta_x]   ;Hasonlóan mint a függőleges
                mov   dx,cx                   ;esetében történt, itt is azok
                inc   cx                      ;a műveletek hajtódnak végre.
                shr   dx,1                    ;csak az ellenkező koordinátákkal

.2_vizsz:       push  dx
                call  pont
                pop   dx
                mov   ax,word ptr [sgn_x]
                add   word ptr [pont_x],ax
                add   dx,word ptr [delta_y]
                cmp   dx,word ptr [delta_x]
                jc    .3_vizsz

                sub   dx,word ptr [delta_x]
                mov   ax,word ptr [sgn_y]
                add   word ptr [pont_y],ax
.3_vizsz:       loop  .2_vizsz

                ret

linc            endp


pont            Proc

                mov   di,word ptr [pont_x]
                mov   ax,word ptr [pont_y]
                mov   bx,320
                mul   bx
                add   di,ax
```

```
                mov   al,7
                mov   es:[di],al

                ret

pont            Endp

;********** ADATOK **********

pont_x:         dw    ?             ;A kirakandó pont xy
pont_y:         dw    ?             ;koordinátája.

xa:             dw    0             ;A vonal kezdőpontjának
ya:             dw    0             ;koordinátái.

xb:             dw    120           ;A vonal végpontjának
yb:             dw    175           ;koordinátái.

delta_x:        dw    ?             ;A vonal vízszintes mérete.
delta_y:        dw    ?             ;A vonal függőleges mérete.

sgn_x:          dw    ?             ;A vízszintes lépésköz és
                                    ;iránya.

sgn_y:          dw    ?             ;A függőleges lépésköz és
                                    ;iránya.

fill_x:         dw    140           ;A kitöltés kezdő koordinátái.
fill_y:         dw    70

color:          db    3             ;A kitöltő szín értéke.

prog17          ends                ;A szegmens vége.
                end   start         ;A program vége.
```

A program működésének megértésében sokat segíthet a következő ábra, ami az egyes elkülönülő mezőket különböző árnyalatokkal tünteti fel.



1. **mező**: a felhasználó által magadott pont sorának bal szélétől elindulva - mivel induláskor a segédregisztereket nulláztuk, és mind felfelé, mind lefelé van kitöltendő pont - a veremben eltárolja a kezdőpont sorának első szomszédos kitöltendő pont-koordinátáját, majd kitölti a sort. Ezután kiemeli a veremből az utoljára letárolt értéket ami jelen esetben az alatta lévő sort jelenti. Itt mivel felfelé már nem lehet kitölteni, csak egy koordinátát tárol el, és teszi ezt mindaddig, míg el nem ér a bekarikázott sarokponthoz. Itt mivel van egy olyan pont, ahol nem érzékel lefelé kitölthető pontot, nullázza a segédregisztert, ami az adott mező végét jelenti. A sarkor elhagyva ismét talál lefelé kitölthető pontot, így az első lehetőség koordinátáit eltárolja. A sor végére érve már ezt emeli le a veremről és innen folytatja. Ezért lett azonos árnyalattal jelölve ez a terület.

2. **mező**: Mivel a sarok elérése előtti utolsó letárolt koordináta a 2. mező bal felső saroka volt, ezért a kitöltést innen folytatja. Mivel felfelé nem fog sehol kitöltetlen pontot találni, a bal alsó sarok elérése után a legelső felfelé lévő pont koordinátáit fogja a verem tetején találni.

3. **mező**: mivel ezen ponttól lefelé már minden pont kitöltött, így csak felfelé irányuló pontokat fog a verembe helyezni. Amikor a 3-4 mező határához ér, ismét érzékelni fog egy pontot, ahol nem tud felfelé haladni, ezért itt nullázza a segédregisztert, majd a 3. mező nyúlványának első kitölthető pont-koordinátáját helyezi a verembe, így ez szintén a 3. mezővel együtt kerül kifestésre.

4. **mező**: A 3. - 4. mező határán lévő töréspont előtti utolsó letárolt koordináta a 4. mező bal alsó sarka. Így innen folytatja a kifestést.

Mint említettem a verem egy kiválló eszköz a koordináták tárolására, mivel nincs szükség külön pointer kezelésére, elegendő a kezdeti érték eltárolása. Ha a 4. mezőt is kifestette, akkor az aktuális veremmutató értéke meg kell hogy egyezzen az álltalunk eltárolt értékkel, és ha ez így van, vége a kitöltésnek.

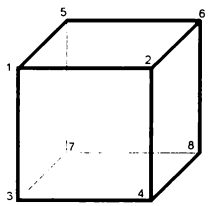
A program futásában meghatározó szerepet játszó töréspontok kis karikával lettek megjelölve. Ha a program futása egy ilyen ponthoz érkezik az jelzés a számára, hogy az előző mező lezárult, tehát újabb értéket kell majd betenni a verembe, amint újból szabad lesz a tér abban az irányban.

8. Fejezet

Háromdimenziós alakzatok kezelése

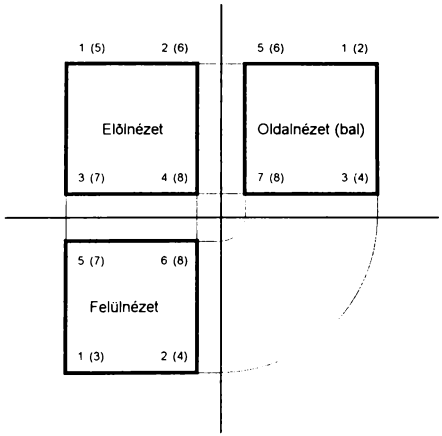
Az eddigiekben kizárólag kétdimenziós alakzatokkal foglalkoztunk. Bármennyire is úgy tűnik néna, hogy egy térbeli ábrát látunk (pl.: nyomógombok, gömbforgatás), azok csak előre elkészített ábrák voltak. Ahhoz, hogy egy kép háromdimenziós azaz térbeli kép legyen, alapfeltétel, hogy minden pontjához három koordináta tartozzon (X,Y,Z), és természetesen ennek megfelelően kell kezelni is. Azonban van egy kis probléma. Mégpedig az, hogy az a doboz, amin nézzük a képet, sajnos nem képes, csak síkbeli alakzatok megjelenítésére. Az egyetlen megoldás, hogy becsapjuk szemünket, és megpróbáljuk elhitetni vele, hogy például ő most nem egy négyzetet, hanem egy kockát lát. De hogyan is tehetjük meg ezt ?

Akik nem tanultak szakrajzot, valószínűleg azok is könnyen megértik a most következő pár ábrát.



Egy kocka térbeli ábrázolása.

A kocka egyes sarkait számokkal jelöltem a későbbi hivatkozások könnyebb megértésének érdekében. Ugyanezt a kockát ábrázolhatjuk úgy is, hogy az egyes nézeteket különbontva egy kétdimenziós vetületi ábrát készítünk belőle.



Egy kocka vetületi ábrázolása

Az előzőekben megszámozott sarkok itt kapnak szerepet, ugyanis a nem látható sarkokat jelölő számok zárójelek közé kerültek. Nos ez az, az ábrázolási mód, amire a monitor képes. De hogyan lesz ebből a három vetületből az előbb látott kocka ?

Térbeli alakzatok síkbeli leképzése

A megoldás az, hogy az egyes pontokhoz tartozó **X, Y, Z** koordinátákból ki kell számolni, hogy az a pont ami valahol van a térben, egy papírlapra vetítve hová esne, azaz a három koordinátából kettőt kell csinálni. Erre van egy már jól bevált képlet ami **x'** és **y'** koordinátákat számol az előbb említett háromból:

$$x' = \frac{m \cdot X}{p + Z}$$

$$y' = \frac{m \cdot Y}{p + Z}$$

A képletben használt **m** és **p** változók a nagyítást és a perspektívát határozzák meg. A megfelelő látvány eléréséhez ezeket a megfelelő értékre kell beállítani, de ez az ábrázolandó alakzattól is függ. Ezen képletek segítségével kiszámolhatjuk az alakzat minden pontjához tartozó vetületi koordinátákat, amit már tudunk ábrázolni a képernyőn.

Az ábrák forgatása

Sokféle dolgot lehet ezekkel a háromdimenziós alakzatokkal művelni. Mivel az egyes pontokat általában vonalak kötik össze, mindössze a pont koordinátáinak megváltoztatásával egy másik alakzatot állíthatunk elő. Továbbá igen könnyű például tükrözni, eltolni, nagyítani az ábrát, mivel előre megadott koordinátákat kell megváltoztatni. Ez a vektorgrafika előnye. Nagy hátránya azonban az, hogy sokkal nehezebb olyan látványos (szép) ábrák létrehozása, mint egy bittérképes grafikával.

Az előbb említett műveleteknél valamivel nehezebb a forgatás, mivel ehhez már szükség van a szögfüggvények alkalmazására, amit a számítógép (az alap processzor) nem ismer. Megoldásként szokták alkalmazni, hogy táblázatszerűen letárolnak egy bizonyos felbontású szinusz és koszinusz táblát 0-360°-ig. Ezután, ha egy szögfüggvény értékére van szükség, egyszerűen kiolvassák a táblázat megfelelő elemét. Egyszerűsíti a dolgot az is, hogy nem szükséges két teljes táblázat letárolása mivel a szinusz és koszinusz függvények értékében mindössze egy 90°-os eltolás van, tehát elegendő egy tábla létrehozása ennyivel megtoldva.

További problémát jelent, hogy a szögfüggvények értéke pozitív és negatív értékeket egyaránt felvehet, és értéke mindig -1 és 1 közé esik, azaz vagy egy lebegőpontos számábrázolásra van szükség, vagy másként kell megoldani a problémát. A gyakorlatban az utóbbit szokták alkalmazni oly módon, hogy a függvény értékeket felszorozva tárolják 15 biten, és a 16. bit az az előjel. A számolások során pedig mint előjeles számokat kell kezelni.

Egy meghatározott tengely körüli forgatáshoz mindig csak két koordinátára van szükség (például ha a **z** tengely körül akarjuk elforgatni az alakzatot akkor az **x** és az **y** koordinátákra).

Az egyes tengelyek körüli forgatást a következő képletek alapján tehetjük meg:

A **Z** tengely körüli forgatásnál:

$$x' = x \cdot \cos\alpha + y \cdot \sin\alpha$$
$$y' = y \cdot \cos\alpha - x \cdot \sin\alpha$$

Az **X** tengely körüli forgatásnál:

$$z' = z \cdot \cos\beta + y \cdot \sin\beta$$
$$y' = y \cdot \cos\beta - z \cdot \sin\beta$$

Az **Y** tengely körüli forgatásnál:

$$x' = x \cdot \cos\gamma + z \cdot \sin\gamma$$
$$z' = z \cdot \cos\gamma - x \cdot \sin\gamma$$

Amint az látható az egyenletekből, az új koordináta mindig a kilépő elem helyére kerül a képletben. A megfelelő műveletek végrehajtásával az ábránkat bármelyik irányban elforgathatjuk.

A most következő program erre a forgatásra mutat be egy példát.

[Program 18]

```
prog18    segment                 ;Szegmensdefiníció.
          assume cs:prog18,ds:prog18 ;A szegmensregiszterek
                                  ;hozzárendelése a kódhoz.

start:    push  cs                ;Cs áttöltése ds-be a vermen
          pop   ds                ;keresztül.

          mov   ax,13h            ;320*200/256 üzemmód
          int   10h               ;beállítása.

          mov   dx,3ceh           ;A videómemória címtartományát
          mov   al,6              ;128kb-ra állítja úgy, hogy a
```

```
                out     dx,al           ;regiszter többi bitjét ne
                inc     dx              ;változtassa meg.
                in      al,dx
                and     al,0011b
                out     dx,al

                mov     ax,0b000h       ;A 0b000h címen kezdődő
                mov     cx,320*200      ;szegmens kinullázása.
                xor     di,di
                mov     es,ax
                mov     al,0
                rep     stosb

                mov     dx,3d4h         ;A kijelzés beállítása a
                mov     al,0ch          ;normál címtől kezdődően.
                out     dx,al
                inc     dx
                mov     al,0
                out     dx,al

.0_prog18:      mov     word ptr [angle],0  ;A kezdő szögelfordulás
                                            ;beállítása nullára.

                call    copy            ;A koordinátamásolatok
                                        ;elkészítése, mivel egy adott
                                        ;tengely körüli forgatáskor
                                        ;csak két koordinátát változtatunk
                                        ;meg, de a leképzéshez mind
                                        ;a három (x,y,z) összetevőre
                                        ;szükség van.

                mov     cx,9            ;az elforgatás 9 lépésből
                                        ;lett megoldva.

.1_prog18:      push    cx

                call    rotate_x        ;Forgatás az X tengely körül.

                call    lekepz          ;A 3D alakzat 2D-s vetületének
                                        ;leképzése.

                call    box             ;A kocka felrajzolása.

                call    vert_bl         ;Vertical Blank időzítés.
```

```
            call    change              ;A kijelzés lapozása.

            mov     ax,100h             ;Billentyűfigyelés.
            int     16h
            jnz     end                 ;Ha van lenyomott gomb, ugrik
                                        ;a végére.

            pop     cx
            inc     word ptr [angle]    ;Az elforgatási szög megnövelése.
            loop    .1_prog18


            call    copy
            mov     cx,9
            mov     word ptr [angle],0
.2_prog18:  push    cx

            call    rotate_y
            call    lekepz
            call    box
            call    vert_bl
            call    change

            mov     ax,100h
            int     16h
            jnz     end

            pop     cx
            inc     word ptr [angle]
            loop    .2_prog18

            call    copy
            mov     cx,9
            mov     word ptr [angle],0
.3_prog18:  push    cx

            call    rotate_z
            call    lekepz
            call    box
            call    vert_bl
            call    change

            mov     ax,100h
```

```
                int   16h
                jnz   end

                pop   cx
                inc   word ptr [angle]
                loop  .3_prog18

                jmp   .0_prog18

end:            xor   ax,ax             ;Billentyűvárás.
                int   16h

                mov   ax,3              ;Text üzemmód visszaállítása.
                int   10h

                mov   ah,4ch            ;Visszatérés a DOS-hoz.
                int   21h

;********** RUTINOK **********

rotate_y        proc

                mov   ax,word ptr [angle]   ;A megadott szöghöz tartozó
                shl   ax,1                  ;memóriacím kiszámítása.
                lea   di,sin
                add   di,ax
                lea   si,oldkoords
                mov   cx,8

.1_ry:          mov   ax,[si]
                mov   bx,[di+20]
                imul  bx                ;X * COS α
                mov   bx,32767
                idiv  bx
                push  ax

                mov   ax,[si+4]         ;Z * SIN α
                mov   bx,[di]
                imul  bx
                mov   bx,32767
                idiv  bx
                pop   bx

                add   ax,bx             ;Z * SIN α+ X * COS α
```

```
                mov   [si+48],ax                ;X új értéke.

                mov   ax,[si]
                mov   bx,[di]
                imul  bx                        ;X * SIN α
                mov   bx,32767
                idiv  bx
                push  ax

                mov   ax,[si+4]
                mov   bx,[di+20]
                imul  bx                        ;Z * COS alfa
                mov   bx,32767
                idiv  bx

                pop   bx

                sub   ax,bx

                mov   [si+52],ax                ;Z új értéke.
                add   si,6
                loop  .1_ry

                ret

rotate_y        endp

;*******************************

rotate_z        proc

                mov   ax,word ptr [angle]
                shl   ax,1
                lea   di,sin
                add   di,ax
                lea   si,oldkoords
                mov   cx,8

.1_rz:          mov   ax,[si]
                mov   bx,[di+20]
                imul  bx                        ;X * COS α
                mov   bx,32767
                idiv  bx
                push  ax
```

```
                mov   ax,[si+2]                 ;Y * SIN α
                mov   bx,[di]
                imul  bx
                mov   bx,32767
                idiv  bx
                pop   bx

                add   ax,bx                     ;Y * SIN α+ X * COS α
                mov   [si+48],ax                ;X új értéke

                mov   ax,[si]
                mov   bx,[di]
                imul  bx                        ;X * SIN α
                mov   bx,32767
                idiv  bx
                push  ax

                mov   ax,[si+2]
                mov   bx,[di+20]
                imul  bx                        ;Y * COS α
                mov   bx,32767
                idiv  bx

                pop   bx

                sub   ax,bx

                mov   [si+50],ax                ;Y új értéke
                add   si,6
                loop  .1_rz

                ret

rotate_z        endp

;*****************************

rotate_x        proc

                mov   ax,word ptr [angle]
                shl   ax,1
                lea   di,sin
                add   di,ax
```

```
            lea   si,oldkoords
            mov   cx,8

.1_rx:      mov   ax,[si+2]
            mov   bx,[di+20]
            imul  bx                    ;Y * COS α
            mov   bx,32767
            idiv  bx
            push  ax

            mov   ax,[si+4]             ;Z * SIN α
            mov   bx,[di]
            imul  bx
            mov   bx,32767
            idiv  bx
            pop   bx

            add   ax,bx                 ;Z * SIN α+ Y * COS α
            mov   [si+50],ax            ;Y új értéke.

            mov   ax,[si+2]
            mov   bx,[di]
            imul  bx                    ;Y * SIN α
            mov   bx,32767
            idiv  bx
            push  ax

            mov   ax,[si+4]
            mov   bx,[di+20]
            imul  bx                    ;Z * COS α
            mov   bx,32767
            idiv  bx

            pop   bx

            sub   ax,bx

            mov   [si+52],ax            ;Z új értéke.
            add   si,6
            loop  .1_rx

            ret

rotate_x    endp
```

```
;*******************************

cls         proc

            mov   di,30*320          ;A képernyő középső sávjának
            mov   cx,140*160         ;törlése.
            xor   ax,ax
            rep   stosw
            ret
cls         endp

;*******************************

vert_bl     proc

            mov   dx,3dah            ;Vertical blank időzítés.
.1_vb:      in    al,dx
            test  al,8
            jz    .1_vb

.2_vb:      in    al,dx
            test  al,8
            jnz   .2_vb

.3_vb:      in    al,dx
            test  al,8
            jz    .3_vb

            ret

vert_bl     endp

;*******************************

copy        proc

            lea   si,oldkoords       ;A régi koordináták átmásolása
            lea   di,newkoords       ;az új helyre.
            mov   cx,24
.1_copy:    mov   ax,[si]
            mov   [di],ax
            add   di,2
            add   si,2
```

```
            loop    .1_copy
            ret
copy        endp

;*****************************

change      proc

            mov     dx,3d4h             ;A 3d4h porton elérhető (0ch)
            mov     al,0ch              ;video regiszter határozza meg a
            out     dx,al               ;megjelenítés kezdőcímének
            inc     dx                  ;magasabb helyiértékű byte-ját.
            mov     al,40h              ;Ha ezt xor-oljuk 40h-val, akkor
            xor     byte ptr [select],al ;pont egy képernyőnyi (64K)
            mov     al,byte ptr [select] ;értékkel változtatjuk meg azt.
            out     dx,al               ;Így lehet a két képernyő
                                        ;között lapozgatni.

            mov     ax,es               ;A 0A000h és 0B000h cím közötti
            xor     ax,1000h            ;váltás.
            mov     es,ax

            call    cls

            ret

change      endp

;*****************************

box         proc

            lea     si,xykoords         ;A kocka felrajzolása.
            mov     ax,[si]
            mov     word ptr [xa],ax
            mov     ax,[si+2]
            mov     word ptr [ya],ax
            mov     ax,[si+4]
            mov     word ptr [xb],ax
            mov     ax,[si+6]
            mov     word ptr [yb],ax
            call    line

            mov     ax,[si+16]
```

```
mov   word ptr [xb],ax
mov   ax,[si+18]
mov   word ptr [yb],ax
call  line

mov   ax,[si+12]
mov   word ptr [xb],ax
mov   ax,[si+14]
mov   word ptr [yb],ax
call  line


mov   ax,[si+28]
mov   word ptr [xa],ax
mov   ax,[si+30]
mov   word ptr [ya],ax
mov   ax,[si+24]
mov   word ptr [xb],ax
mov   ax,[si+26]
mov   word ptr [yb],ax
call  line

mov   ax,[si+16]
mov   word ptr [xb],ax
mov   ax,[si+18]
mov   word ptr [yb],ax
call  line

mov   ax,[si+12]
mov   word ptr [xb],ax
mov   ax,[si+14]
mov   word ptr [yb],ax
call  line


mov   ax,[si+20]
mov   word ptr [xa],ax
mov   ax,[si+22]
mov   word ptr [ya],ax
mov   ax,[si+24]
mov   word ptr [xb],ax
mov   ax,[si+26]
mov   word ptr [yb],ax
call  line
```

```
                mov   ax,[si+16]
                mov   word ptr [xb],ax
                mov   ax,[si+18]
                mov   word ptr [yb],ax
                call  line

                mov   ax,[si+4]
                mov   word ptr [xb],ax
                mov   ax,[si+6]
                mov   word ptr [yb],ax
                call  line

                mov   ax,[si+8]
                mov   word ptr [xa],ax
                mov   ax,[si+10]
                mov   word ptr [ya],ax
                mov   ax,[si+24]
                mov   word ptr [xb],ax
                mov   ax,[si+26]
                mov   word ptr [yb],ax
                call  line

                mov   ax,[si+12]
                mov   word ptr [xb],ax
                mov   ax,[si+14]
                mov   word ptr [yb],ax
                call  line

                mov   ax,[si+4]
                mov   word ptr [xb],ax
                mov   ax,[si+6]
                mov   word ptr [yb],ax
                call  line

                ret

box             endp

;*******************************

lekepz          proc

                mov   si,offset newkoords
```

```
                mov   di,offset xykoords
                mov   cx,8

.1_lekepz:      mov   ax,[si]                ;m*X
                mov   bl,100
                imul  bl
                cwd
                mov   bx,[si+4]              ;p+Z
                add   bx,200
                idiv  bx                     ;X=(m*X)/(p+Z)

                add   ax,160                 ;X középre pozícionálása.
                mov   [di],ax

                mov   ax,[si+2]              ;m*Y
                mov   bl,100
                imul  bl
                cwd
                mov   bx,[si+4]              ;p+Z
                add   bx,200
                idiv  bx                     ;Y=(m*Y)/(p+Z)

                add   ax,100                 ;Y középre pozícionálása.
                mov   [di+2],ax

                add   di,4
                add   si,6
                loop  .1_lekepz

                ret

lekepz          endp


line            proc                         ;A vonalhúzó rutin kezdete.

                mov   bx,word ptr [xa]       ;A vonal kezdőpontjának és
                mov   ax,word ptr [xb]       ;végpontjának x koordinátáját
                                             ;beírja az ax, bx regiszterekbe.

                mov   dx,1                   ;A vízszintes lépésköz.

                sub   ax,bx                  ;Kiszámítja a két pont közötti
                                             ;távolságot.
```

```
                jnc     .1_line                 ;Ha cz nem negatív, átugorja
                                                ;a következő részt.

                neg     ax                      ;A távolság valódi értékének a
                                                ;kiszámítása.

                mov     dx,65535                ;A vízszintes lépésközt -1-re
                                                ;állítja.

.1_line:        mov     word ptr [delta_x],ax   ;A regisztereket beírja a
                mov     word ptr [pont_x],bx    ;memóriaváltozókba.
                mov     word ptr [sgn_x],dx

                mov     bx,word ptr [ya]        ;Ugyanaz, ami a vízszintes
                mov     ax,word ptr [yb]        ;értékeknél történt.
                mov     dx,1
                sub     ax,bx
                jnc     .2_line
                neg     ax
                mov     dx,65535

.2_line:        mov     word ptr [delta_y],ax
                mov     word ptr [pont_y],bx
                mov     word ptr [sgn_y],dx

                cmp     ax,word ptr [delta_x]   ;Eldönti, hogy a vonal milyen
                                                ;irányú.

                jc      .1_vizsz                ;Ha a vízszintes összetevője
                                                ;a nagyobb, arra az eljárásra
                                                ;ugrik.

                mov     dx,ax                   ;Egyébként az algoritmus
                shr     dx,1                    ;növekményét számláló
                                                ;regiszterbe beírja a vonal
                                                ;vízszintes hosszának a felét,

;fuggoleges

                mov     cx,ax                   ;a ciklusszámlálóba pedig a
                inc     cx                      ;hossznál eggyel többet.

.1_fuggo:       push    dx                      ;A számláló elmentése.
```

```
            call    pont                    ;Az aktuális pont kirakása.

            pop     dx
            mov     ax,word ptr [sgn_y]     ;Az algoritmus szerinti következő
            add     word ptr [pont_y],ax    ;pont koordinátainak kiszámítása

            add     dx,word ptr [delta_x]
            cmp     dx,word ptr [delta_y]
            jc      .2_fuggo

            sub     dx,word ptr [delta_y]
            mov     ax,word ptr [sgn_x]
            add     word ptr [pont_x],ax
.2_fuggo:   loop    .1_fuggo

            ret

;vizszintes

.1_vizsz:   mov     cx,word ptr [delta_x]   ;Hasonlóan, mint a függőleges
            mov     dx,cx                   ;esetben történt, itt is azok
            inc     cx                      ;a művcletek hajtódnak végre,
            shr     dx,1                    ;csak az ellenkező koordinátákkal

.2_vizsz:   push    dx
            call    pont
            pop     dx
            mov     ax,word ptr [sgn_x]
            add     word ptr [pont_x],ax
            add     dx,word ptr [delta_y]
            cmp     dx,word ptr [delta_x]
            jc      .3_vizsz

            sub     dx,word ptr [delta_x]
            mov     ax,word ptr [sgn_y]
            add     word ptr [pont_y],ax
.3_vizsz:   loop    .2_vizsz

            ret

line        endp
```

```
pont    Proc

        mov   di,word ptr [pont_x]
        mov   ax,word ptr [pont_y]
        mov   bx,320
        mul   bx
        add   di,ax
        mov   al,7
        mov   es:[di],al

        ret

pont    Endp

;********** ADATOK **********


pont_x:   dw   ?              ;A kirakandó pont xy
pont_y:   dw   ?              ;koordinátája.

xa:       dw   0              ;A vonal kezdőpontjának
ya:       dw   0              ;koordinátai.

xb:       dw   120            ;A vonal végpontjának
yb:       dw   175            ;koordinátai.

delta_x:  dw   ?              ;A vonal vízszintes mérete.
delta_y:  dw   ?              ;A vonal függőleges mérete.

sgn_x:    dw   ?              ;A vízszintes lépésköz és
                              ;iránya.

sgn_y:    dw   ?              ;A függőleges lépésköz és
                              ;iránya.

select:   db   0


sin:      dw   0,163ah,2bc7h,4000h,5246h,620dh,6ed9h,7847h,7e0eh,7fffh

cos:      dw   7fffh,7e0eh,7847h,6ed9h,620dh,5246h,4000h,2bc7h,163ah,0

angle:    dw   0
```

```
oldkoords:  dw   -50,-50,-50        ;A kocka X, Y, Z koordinátái.
            dw   50,-50,-50
            dw   50,50,-50
            dw   -50,50,-50

            dw   -50,-50,50
            dw   50,-50,50
            dw   50,50,50
            dw   -50,50,50

newkoords:  dw   24 dup (?)
xykoords:   dw   16 dup (0)

prog18      ends                    ;A szegmens vége.
            end  start              ;A program vége.
```

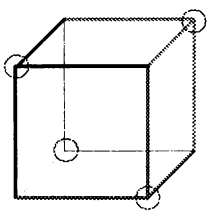
A program felváltva forgatja az x - y - z tengelyek körül a kockát, azonban minden forgatás végrehajtása azonos lépésekből áll. Ezek: A kezdő koordináták átmásolása arra a helyre, ahol majdan az új koordináták képződnek mivel ha egy tengely körül elforgatunk egy kockát, annak csak 2 koordináta összetevőjét kell hogy megváltoztassuk, azonban a 3D-2D leképzéshez mind a három összetevőre szükség van. Igy a legegyszerűbb volt ezt egy rutinnal megoldani, ami mind a három összetevőt másolja.

A második lépés a forgatás végrehajtása. Itt most csupán egy 10 elemű szinusz illetve koszinusz táblázatot készítettem mivel az elmélet bemutatásához elegendő a 90°-os forgatást 10°-os lépésekben bemutatni. Az első lépésben kiszámítjuk az aktuális elfordulási szöghöz tartozó táblázatcímet, amit a **di** regiszterbe helyezünk. Ezután pl.: a **z** tengely körüli forgatásnál kiszámoljuk az $x \cdot \cos\alpha$ illetve az $y \cdot \sin\alpha$ értékeket és

összeadjuk egymással. Ezzel megkaptuk az új x koordináta értékét. Az y kiszámításához először az $x \cdot \sin\alpha$ majd az $y \cdot \cos\alpha$ értékeket határozzuk meg, és ez utóbbiból kivonjuk az előzőt, hogy megkapjuk az y' értékét. Ezen műveletek minden forgatás során hasonlóan történnek.

A következő lépés az alakzat 2D-s vetületének elkészítése. A **leképz** eljárás a már említett egyenletet használja ennek megvalósítására m=100 és p=200 paraméterekkel. Az alakzat koordinátáinak meghatározásakor ügyelni kell arra, hogy az előbbiekben említett képletek a 0,0 pontot veszik középpontnak, tehát ezen pont körül forgatnak, és ezt veszik szemtengely középpontnak is. Ezért a kocka sarok koordinátáinak meghatározásakor + és - koordinátákat alkalmaztam, amit a **leképz** rutinban kompenzálok oly módon, hogy az x koordináta értékhez 160-at, az y-hoz pedig 100-at adok.

A 3D-2D átalakítás után következik az alakzat képernyőre (azaz mögé) rakása. Egy kockát többféleképpen is fel lehet rajzolni. Akkor, ha nem akarjuk kitölteni annak oldalait, hanem csak egy vonalrajzot szeretnénk készíteni, akkor az a cél, hogy minél kevesebb koordináta megadással tudjuk ezt megtenni. Erre viszont már kevesebb variáció van. Nem a leggyorsabb, de azért már optimális megoldásnak számít az, amit ebben a programban is alkalmaztam, de ez a következő ábrából könnyebben megérthető.



A kocka felrajzolásának módja

A bekarikázott sarkok az úgynevezett kiindulási pontok. Minden egyes ilyen pontból három vonal indul ki. Ezzel a módszerrel a kocka 12 élét 16 koordináta megadásával tudjuk felrajzolni. A legkevesebb pontmegadás (12), amit úgy tudjuk elérni, hogy ha ezen módszer szerint

haladva az utolsó vonal végpontjából húzzuk tovább a bekarikázott ponthoz a vonalat.

Ha a kép már a háttérképernyőn van, akkor egy *vertical blank* időzítés után már csak a kijelzést kell megcserélni, és természetesen törölni a háttérképet. Ez a módszer azonban (ebben a formában) nem alkalmas csak egy tengely körüli forgatásra. Ha egyszerre több irányba akarunk forgatni, akkor vagy sokkal pontosabb számításokra van szükség, vagy egy lépésben kell kiszámolni a három irányú forgatás eredményét. Ez utóbbi egy kicsivel bonyolultabb képletet igényel viszont, gyorsabb és hosszú távon hatékonyabb módszer.

Forgatás mindhárom tengelyen

Tulajdonképpen nem történik más, mint a három forgatási eljárást összevonjuk egy képletbe. Így keletkeztek a következő képletek:

$$x' = x\cdot(\cos\alpha\cdot\cos\gamma - \sin\alpha\cdot\sin\beta\cdot\sin\gamma) - y\cdot(\sin\alpha\cdot\cos\alpha\cdot\sin\gamma + \cos\gamma\cdot\sin\alpha) + z\cdot(\sin\gamma\cdot\cos\beta)$$
$$y' = x\cdot(\sin\alpha\cdot\cos\beta) + y\cdot(\cos\beta\cdot\cos\alpha) + z\cdot(\sin\beta)$$
$$z' = x\cdot(\sin\gamma\cdot\cos\alpha + \cos\gamma\cdot\sin\beta\cdot\sin\alpha) + y\cdot(\cos\gamma\cdot\sin\beta\cdot\cos\alpha - \sin\gamma\cdot\sin\alpha) + z\cdot(\cos\beta\cdot\cos\gamma)$$

Ez első ránézésre elég bonyolultnak tűnhet (másodikra még inkább), azonban ha jobban megnézzük a képleteket, észrevehetjük, hogy ismétlődő részeket tartalmaz, így szét lehet bontani részekre amiket elegendő egyszer kiszámolni és behelyettesíteni a képletbe. De ez a következő programból jobban megérthetővé válik.

[Program 19]

```
.386
prog19      segment use16                ;Szegmensdefiníció.
            assume cs:prog19,ds:prog19   ;A szegmensregiszterek
                                         ;hozzárendelése a kódhoz.

start:      push  cs                     ;Cs áttöltése ds-be a vermen
            pop   ds                     ;keresztül.

            mov   ax,13h                 ;320*200/256 üzemmód
            int   10h                    ;beállítása.
```

```
            mov   dx,3ceh              ;A videómemória címtartományát
            mov   al,6                 ;128kbyte-ra állítja úgy, hogy a
            out   dx,al                ;regiszter többi bitjét nem
            inc   dx                   ;válltoztatja meg.
            in    al,dx
            and   al,0011b
            out   dx,al

            mov   ax,0b000h            ;A 0b000h címen kezdődő
            mov   cx,320*200           ;szegmens kinullázása.
            xor   di,di
            mov   es,ax
            mov   al,0
            rep   stosb

            mov   dx,3d4h              ;A kijelzés beállítása a
            mov   al,0ch               ;normál címtől kezdődően.
            out   dx,al
            inc   dx
            mov   al,0
            out   dx,al

            lea   bp,ds:morfdata
.1_prog19:  mov   cx,360
.2_prog19:  push  cx
            call  rotate
            call  lekepz
            call  box
            call  change
            inc   word ptr [alfa]
            inc   word ptr [beta]
            inc   word ptr [gamma]
            call  morf
            pop   cx
            mov   ax,100h              ;Billentyűfigyelés.
            int   16h
            jnz   end
            loop  .2_prog19
            mov   word ptr [alfa],0
            mov   word ptr [beta],0
            mov   word ptr [gamma],0
            jmp   .1_prog19

end:        xor   ax,ax                ;A billentyű kiolvasása.
```

```
                int     16h

                mov     ax,3                    ;Text üzemmód visszaállítása.
                int     10h

                mov     ah,4ch                  ;Visszatérés a DOS-hoz.
                int     21h

;********** RUTINOK **********

rotate          proc

                mov     bx,word ptr [alfa]      ;Sin αés cos αkiszámítása.
                lea     si,sin8
                movsx   eax,byte ptr [si+bx]
                mov     dword ptr [sinalfa],eax
                movsx   eax,byte ptr [si+bx+90]
                mov     dword ptr [cosalfa],eax

                mov     bx,word ptr [beta]      ;Sin β, cos β
                movsx   eax,byte ptr [si+bx]
                mov     dword ptr [sinbeta],eax
                movsx   eax,byte ptr [si+bx+90]
                mov     dword ptr [cosbeta],eax

                mov     bx,word ptr [gamma]     ;Sin γ, cos γ
                movsx   eax,byte ptr [si+bx]
                mov     dword ptr [singamma],eax
                movsx   eax,byte ptr [si+bx+90]
                mov     dword ptr [cosgamma],eax

                mov     ebx,127

                mov     eax,dword ptr [cosalfa] ;Cα*Cγ-Sα*Sβ*Sγ
                imul    dword ptr [cosgamma]
                idiv    ebx
                mov     ecx,eax
                mov     eax,dword ptr [sinalfa]
                imul    dword ptr [sinbeta]
                idiv    ebx
                imul    dword ptr [singamma]
                idiv    ebx
                sub     ecx,eax
                mov     dword ptr [xx],ecx
```

```
mov   eax,dword ptr [sinbeta]    ;-(Sα*Cα*Sγ+Cγ*Sα)
imul  dword ptr [cosalfa]
idiv  ebx
imul  dword ptr [singamma]
idiv  ebx
mov   ecx,eax
mov   eax,dword ptr [cosgamma]
imul  dword ptr [sinalfa]
idiv  ebx
add   eax,ecx
neg   eax
mov   dword ptr [xy],eax

mov   eax,dword ptr [singamma];Sγ*Cβ
imul  dword ptr [cosbeta]
idiv  ebx
mov   dword ptr [xz],eax

mov   eax,dword ptr [sinalfa]    ;Sα*Cβ
imul  dword ptr [cosbeta]
idiv  ebx
mov   dword ptr [yx],eax

mov   eax,dword ptr [cosalfa]    ;Cα*Cβ
imul  dword ptr [cosbeta]
idiv  ebx
mov   dword ptr [yy],eax

mov   eax,dword ptr [sinbeta]    ;Sβ
mov   dword ptr [yz],eax

mov   eax,dword ptr [singamma];Sγ*Cα+Cγ*Sβ*Sα
imul  dword ptr [cosalfa]
idiv  ebx
mov   ecx,eax
mov   eax,dword ptr [cosgamma]
imul  dword ptr [sinbeta]
idiv  ebx
imul  dword ptr [sinalfa]
idiv  ebx
add   eax,ecx
mov   dword ptr [zx],eax
```

```
        mov   eax,dword ptr [cosgamma];Cγ*Sβ*Cα-Sγ*Sα
        imul  dword ptr [sinbeta]
        idiv  ebx
        imul  dword ptr [cosalfa]
        idiv  ebx
        mov   ecx,eax
        mov   eax,dword ptr [singamma]
        imul  dword ptr [sinalfa]
        idiv  ebx
        sub   ecx,eax
        mov   dword ptr [zy],ecx

        mov   eax,dword ptr [cosgamma];-Cγ*Cβ
        imul  dword ptr [cosbeta]
        idiv  ebx
        neg   eax
        mov   dword ptr [zz],eax

        mov   cx,8
        lea   si,oldkoords
        mov   bx,127

rotation: movsx eax,word ptr [si]          ;x*xx
        imul  dword ptr [xx]
        mov   edi,eax
        movsx eax,word ptr [si+2]          ;y*xy
        imul  dword ptr [xy]
        add   edi,eax
        movsx eax,word ptr [si+4]          ;z*xz
        imul  dword ptr [xz]
        add   eax,edi
        cwd
        idiv  bx
        mov   [si+48],ax

        movsx eax,word ptr [si]            ;x*yx
        imul  dword ptr [yx]
        mov   edi,eax
        movsx eax,word ptr [si+2]          ;y*yy
        imul  dword ptr [yy]
        add   edi,eax
        movsx eax,word ptr [si+4]          ;z*yz
        imul  dword ptr [yz]
        add   eax,edi
```

```
                cwd
                idiv    bx
                mov     [si+50],ax

                movsx eax,word ptr [si]         ;x*zx
                imul    dword ptr [zx]
                mov     edi,eax
                movsx eax,word ptr [si+2]       ;y*zy
                imul    dword ptr [zy]
                add     edi,eax
                movsx eax,word ptr [si+4]       ;z*zz
                imul    dword ptr [zz]
                add     eax,edi
                cwd
                idiv    bx
                mov     [si+52],ax

                add     si,6
                sub     cx,1
                jz      endrotate
                jmp     rotation
endrotate:      ret

rotate          endp

;*******************************

cls             proc

                mov     di,30*320+90            ;A képernyő középső sávjának
                mov     cx,140                  ;törlése.
                xor     ax,ax
.1_cls:         push    cx
                mov     cx,70
                rep     stosw
                add     di,180
                pop     cx
                loop    .1_cls
                ret

cls             endp

;*******************************
```

```
vert_bl     proc

            mov   dx,3dah          ;Vertical blank időzítés.
.1_vb:      in    al,dx
            test  al,8
            jz    .1_vb

.2_vb:      in    al,dx
            test  al,8
            jnz   .2_vb

.3_vb:      in    al,dx
            test  al,8
            jz    .3_vb

            ret

vert_bl     endp

;*****************************

change      proc

            call  vert_bl
            mov   dx,3d4h          ;A 3d4h porton elérhető 0ch
            mov   al,0ch           ;regisztere határozza meg a
            out   dx,al            ;megjelenítés kezdőcímének
            inc   dx               ;magasabb helyiértékű byte-ját.
            mov   al,40h           ;Ha ezt xor-oljuk 40h-val, akkor
            xor   byte ptr [select],al ;pont egy képernyőnyi (64K)
            mov   al,byte ptr [select] ;értékkel válltoztatjuk meg azt.
            out   dx,al            ;Így lehet a két képernyő
                                   ;között lapozgatni.

            mov   ax,es            ;A 0A000h és 0B000h cím közötti
            xor   ax,1000h         ;váltás.
            mov   es,ax
            call  vert_bl
            call  cls

            ret

change      endp
```

```
;*****************************

box     proc

        lea   si,xykoords          ;A kocka felrajzolása.

        mov   ax,[si+4]            ;A forgásszög megállapítása,
        sub   ax,[si]              ;ha negatív, akkor ugrin a
        mov   bx,[si+10]           ;a következő oldal rajzolására.
        sub   bx,[si+6]
        imul  bx
        mov   cx,ax
        mov   ax,[si+6]
        sub   ax,[si+2]
        mov   bx,[si+8]
        sub   bx,[si+4]
        imul  bx
        add   cx,32768
        add   ax,32768
        cmp   ax,cx
        jnc   .1_box

        mov   ax,[si]              ;Egy oldal felrajzolása.
        mov   word ptr [xa],ax
        mov   ax,[si+2]
        mov   word ptr [ya],ax
        mov   ax,[si+4]
        mov   word ptr [xb],ax
        mov   ax,[si+6]
        mov   word ptr [yb],ax
        call  line
        mov   ax,[si+8]
        mov   word ptr [xa],ax
        mov   ax,[si+10]
        mov   word ptr [ya],ax
        call  line
        mov   ax,[si+12]
        mov   word ptr [xb],ax
        mov   ax,[si+14]
        mov   word ptr [yb],ax
        call  line
        mov   ax,[si]
        mov   word ptr [xa],ax
        mov   ax,[si+2]
```

```
            mov   word ptr [ya],ax
            call  line


.1_box:     mov   ax,[si+20]
            sub   ax,[si+8]
            mov   bx,[si+26]
            sub   bx,[si+22]
            imul  bx
            mov   cx,ax
            mov   ax,[si+22]
            sub   ax,[si+10]
            mov   bx,[si+24]
            sub   bx,[si+20]
            imul  bx
            add   cx,32768
            add   ax,32768
            cmp   ax,cx
            jnc   .2_box


            mov   ax,[si+4]
            mov   word ptr [xa],ax
            mov   ax,[si+6]
            mov   word ptr [ya],ax
            mov   ax,[si+20]
            mov   word ptr [xb],ax
            mov   ax,[si+22]
            mov   word ptr [yb],ax
            call  line
            mov   ax,[si+24]
            mov   word ptr [xa],ax
            mov   ax,[si+26]
            mov   word ptr [ya],ax
            call  line
            mov   ax,[si+8]
            mov   word ptr [xb],ax
            mov   ax,[si+10]
            mov   word ptr [yb],ax
            call  line
            mov   ax,[si+4]
            mov   word ptr [xa],ax
            mov   ax,[si+6]
            mov   word ptr [ya],ax
```

```
        call    line


.2_box: mov     ax,[si+16]
        sub     ax,[si+20]
        mov     bx,[si+30]
        sub     bx,[si+18]
        imul    bx
        mov     cx,ax
        mov     ax,[si+18]
        sub     ax,[si+22]
        mov     bx,[si+28]
        sub     bx,[si+16]
        imul    bx
        add     cx,32768
        add     ax,32768
        cmp     ax,cx
        jnc     .3_box

        mov     ax,[si+28]
        mov     word ptr [xa],ax
        mov     ax,[si+30]
        mov     word ptr [ya],ax
        mov     ax,[si+24]
        mov     word ptr [xb],ax
        mov     ax,[si+26]
        mov     word ptr [yb],ax
        call    line
        mov     ax,[si+20]
        mov     word ptr [xa],ax
        mov     ax,[si+22]
        mov     word ptr [ya],ax
        call    line
        mov     ax,[si+16]
        mov     word ptr [xb],ax
        mov     ax,[si+18]
        mov     word ptr [yb],ax
        call    line
        mov     ax,[si+28]
        mov     word ptr [xa],ax
        mov     ax,[si+30]
        mov     word ptr [ya],ax
        call    line
```

```
.3_box:   mov  ax,[si]
          sub  ax,[si+16]
          mov  bx,[si+14]
          sub  bx,[si+2]
          imul bx
          mov  cx,ax
          mov  ax,[si+2]
          sub  ax,[si+18]
          mov  bx,[si+12]
          sub  bx,[si]
          imul bx
          add  cx,32768
          add  ax,32768
          cmp  ax,cx
          jnc  .4_box

          mov  ax,[si+16]
          mov  word ptr [xa],ax
          mov  ax,[si+18]
          mov  word ptr [ya],ax
          mov  ax,[si]
          mov  word ptr [xb],ax
          mov  ax,[si+2]
          mov  word ptr [yb],ax
          call line
          mov  ax,[si+12]
          mov  word ptr [xa],ax
          mov  ax,[si+14]
          mov  word ptr [ya],ax
          call line
          mov  ax,[si+28]
          mov  word ptr [xb],ax
          mov  ax,[si+30]
          mov  word ptr [yb],ax
          call line
          mov  ax,[si+16]
          mov  word ptr [xa],ax
          mov  ax,[si+18]
          mov  word ptr [ya],ax
          call line

.4_box:   mov  ax,[si+20]
          sub  ax,[si+16]
          mov  bx,[si+6]
```

```
        sub   bx,[si+22]
        imul  bx
        mov   cx,ax
        mov   ax,[si+22]
        sub   ax,[si+18]
        mov   bx,[si+4]
        sub   bx,[si+20]
        imul  bx
        add   cx,32768
        add   ax,32768
        cmp   ax,cx
        jnc   .5_box

        mov   ax,[si+16]
        mov   word ptr [xa],ax
        mov   ax,[si+18]
        mov   word ptr [ya],ax
        mov   ax,[si+20]
        mov   word ptr [xb],ax
        mov   ax,[si+22]
        mov   word ptr [yb],ax
        call  line
        mov   ax,[si+4]
        mov   word ptr [xa],ax
        mov   ax,[si+6]
        mov   word ptr [ya],ax
        call  line
        mov   ax,[si]
        mov   word ptr [xb],ax
        mov   ax,[si+2]
        mov   word ptr [yb],ax
        call  line
        mov   ax,[si+16]
        mov   word ptr [xa],ax
        mov   ax,[si+18]
        mov   word ptr [ya],ax
        call  line

.5_box: mov   ax,[si+8]
        sub   ax,[si+12]
        mov   bx,[si+26]
        sub   bx,[si+10]
        imul  bx
        mov   cx,ax
```

```
            mov   ax,[si+10]
            sub   ax,[si+14]
            mov   bx,[si+24]
            sub   bx,[si+8]
            imul  bx
            add   cx,32768
            add   ax,32768
            cmp   ax,cx
            jnc   endbox

            mov   ax,[si+12]
            mov   word ptr [xa],ax
            mov   ax,[si+14]
            mov   word ptr [ya],ax
            mov   ax,[si+8]
            mov   word ptr [xb],ax
            mov   ax,[si+10]
            mov   word ptr [yb],ax
            call  line
            mov   ax,[si+24]
            mov   word ptr [xa],ax
            mov   ax,[si+26]
            mov   word ptr [ya],ax
            call  line
            mov   ax,[si+28]
            mov   word ptr [xb],ax
            mov   ax,[si+30]
            mov   word ptr [yb],ax
            call  line
            mov   ax,[si+12]
            mov   word ptr [xa],ax
            mov   ax,[si+14]
            mov   word ptr [ya],ax
            call  line

endbox:     ret

box         endp

;*****************************

lekepz      proc

            mov   si,offset newkoords
```

```
                mov   di,offset xykoords
                mov   cx,8

.1_lekepz:      mov   ax,[si+4]          ;m*X
                mov   bx,720
                imul  bx
                mov   bx,[si]            ;p+Z
                add   bx,1000
                idiv  bx                 ;X=(m*X)/(p+Z)
                add   ax,160             ;X középre pozícionálása.
                mov   [di],ax

                mov   ax,[si+2]          ;m*Y
                mov   bx,600
                imul  bx
                mov   bx,[si]            ;p+Z
                add   bx,1000
                idiv  bx                 ;Y=(m*Y)/(p+Z)

                add   ax,100             ;Y középre pozícionálása.
                mov   [di+2],ax

                add   di,4
                add   si,6
                loop  .1_lekepz

                ret

lekepz          endp

;*****************************

line            proc                     ;A vonalhúzó rutin kezdete.

                mov   bx,word ptr [xa]   ;A vonal kezdőpontjának és
                mov   ax,word ptr [xb]   ;végpontjának x koordinátáját
                                         ;beírja az ax, bx regiszterekbe.

                mov   dx,1               ;A vízszintes lépésköz.

                sub   ax,bx              ;Kiszámítja a két pont közötti
                                         ;távolságot.

                jnc   .1_line            ;Ha ez nem negatív, átugorja
```

```
                                        ;a következő részt.

            neg   ax                    ;A távolság valódi értékének a
                                        ;kiszámítása.

            mov   dx,65535              ;A vízszintes lépésközt -1-re
                                        ;állítja.

.1_line:    mov   word ptr [delta_x],ax ;A regiszterek értékét beírja a
            mov   word ptr [pont_x],bx  ;memóriaváltozókba.
            mov   word ptr [sgn_x],dx

            mov   bx,word ptr [ya]      ;Ugyanaz, ami a vízszintes
            mov   ax,word ptr [yb]      ;értékeknél történt.
            mov   dx,1
            sub   ax,bx
            jnc   .2_line
            neg   ax
            mov   dx,65535

.2_line:    mov   word ptr [delta_y],ax
            mov   word ptr [pont_y],bx
            mov   word ptr [sgn_y],dx

            cmp   ax,word ptr [delta_x] ;Eldönti, hogy a vonal milyen
                                        ;irányú.

            jc    .1_vizsz              ;Ha a vízszintes összetevője
                                        ;a nagyobb, arra az eljárásra
                                        ;ugrik.

            mov   dx,ax                 ;Egyébként az algoritmus
            shr   dx,1                  ;növekményét számláló
                                        ;regiszterbe beírja a vonal
                                        ;vízszintes hosszának a felét.

;fuggoleges

            mov   cx,ax                 ;A ciklusszámlálóba pedig a
            inc   cx                    ;hossznál eggyel többet.

.1_fuggo:   push  dx                    ;A számláló elmentése.

            call  pont                  ;Az aktuális pont kirakása.
```

```
                pop   dx
                mov   ax,word ptr [sgn_y]      ;Az algoritmus szerinti következő
                add   word ptr [pont_y],ax     ;pont koordinátainak kiszámítása

                add   dx,word ptr [delta_x]
                cmp   dx,word ptr [delta_y]
                jc    .2_fuggo

                sub   dx,word ptr [delta_y]
                mov   ax,word ptr [sgn_x]
                add   word ptr [pont_x],ax
.2_fuggo:       loop  .1_fuggo

                ret

;vizszintes

.1_vizsz:       mov   cx,word ptr [delta_x]    ;Hasonlóan mint a függőleges
                mov   dx,cx                    ;eseteben történt, itt is
                inc   cx                       ;azok a műveletek hajtódnak
                shr   dx,1                     ;végre, csak az x

.2_vizsz:       push  dx                       ;koordinátákkal.
                call  pont
                pop   dx
                mov   ax,word ptr [sgn_x]
                add   word ptr [pont_x],ax
                add   dx,word ptr [delta_y]
                cmp   dx,word ptr [delta_x]
                jc    .3_vizsz

                sub   dx,word ptr [delta_x]
                mov   ax,word ptr [sgn_y]
                add   word ptr [pont_y],ax
.3_vizsz:       loop  .2_vizsz

                ret

line            endp


pont            Proc
```

```
                mov   di,word ptr [pont_x]
                mov   ax,word ptr [pont_y]
                mov   bx,320
                mul   bx
                add   di,ax
                mov   es:[di],byte ptr 7

                ret

pont            Endp

;*****************************

morf            proc

                xor   dx,dx             ;Változást ellenőrző flag.
                mov   cx,24             ;24 koordinátát kell
                                        ;megváltoztatni.

                lea   si,oldkoords      ;A báziskoordináták címe.

.1_morf:        mov   ax,ds:[bp]        ;Ax-be tölti a morf alakzat
                                        ;adott koordinátáját és

                cmp   [si],ax           ;ezt összehasonlítja az eredeti
                jnc   .11_morf          ;alakzat koordinátájával.

                inc   word ptr [si]     ;Ha az kisebb, növeli,
                mov   dx,255
.11_morf:       cmp   ax,[si]
                jnc   .2_morf
                dec   word ptr [si]     ;ha pedig nagyobb, csökkenti,
                mov   dx,255            ;és jelzi a válltoztatást.

.2_morf:        add   si,2              ;A következő koordináta.
                add   bp,2
                loop  .1_morf

                or    dx,dx             ;Ha történt változtatás,
                jz    .3_morf
                sub   bp,48             ;visszaállítja a bp-t a morf
                                        ;alakzat kezdőcímére, és
                ret                     ;visszatér a hívó programhoz.
```

```
.3_morf:    inc     byte ptr [morfcount]      ;A következő alakzat.

            cmp     byte ptr [morfcount],8    ;Ha elérte az utolsót, vissza
            jnz     .4_morf                   ;az elejére.
            lea     bp,ds:morfdata
            mov     byte ptr [morfcount],0
.4_morf:    ret                               ;Visszatérés a hívóhoz.

morf        endp

;********** ADATOK **********

pont_x:     dw      ?                         ;A kirakandó pont xy
pont_y:     dw      ?                         ;koordinátája.
color:      db      ?                         ;A vonal színe
xa:         dw      0                         ;A vonal kezdőpontjának
ya:         dw      0                         ;koordinátái.

xb:         dw      120                       ;A vonal végpontjának
yb:         dw      175                       ;koordinátái.

delta_x:    dw      ?                         ;A vonal vízszintes mérete.
delta_y:    dw      ?                         ;A vonal függőleges mérete.

sgn_x:      dw      ?                         ;A vízszintes lépésköz és
                                              ;iránya.

sgn_y:      dw      ?                         ;A függőleges lépésköz és
                                              ;iránya.

select:     db      0                         ;Kiválasztás a képernyő-
                                              ;lapozáshoz.

morfcount:  db      0                         ;Morfózis fázis.

oldkoords:  dw      -50,-50,-50               ;A kocka X, Y, Z koordinátái.
            dw      50,-50,-50
            dw      50,50,-50
            dw      -50,50,-50
            dw      -50,-50,50
            dw      50,-50,50
            dw      50,50,50
            dw      -50,50,50
```

```
newkoords:  dw   24 dup (0)          ;Elforgatott koordináták.

xykoords:   dw   16 dup (0)          ;Leképzett koordináták.

morfdata:   dw   -10,-10,-10         ;Morfózis fázisok.
            dw   10,-10,-10
            dw   10,10,-10
            dw   -10,10,-10
            dw   -10,-10,10
            dw   10,-10,10
            dw   10,10,10
            dw   -10,10,10

            dw   -60,-20,-20
            dw   60,-20,-20
            dw   60,20,-20
            dw   -60,20,-20
            dw   -60,-20,20
            dw   60,-20,20
            dw   60,20,20
            dw   -60,20,20

            dw   -60,-10,-10
            dw   60,-10,-10
            dw   60,30,-50
            dw   -60,30,-50
            dw   -60,-10,60
            dw   60,-10,60
            dw   60,10,50
            dw   -60,10,50

            dw   -50,-50,-10
            dw   50,-50,-10
            dw   50,50,-10
            dw   -50,50,-10
            dw   -50,-50,10
            dw   50,-50,10
            dw   50,50,10
            dw   -50,50,10

            dw   -60,-10,-60
            dw   60,-10,-60
            dw   60,30,-10
```

```
                dw      -60,30,-10
                dw      -60,-10,60
                dw      60,-10,60
                dw      60,10,50
                dw      -60,10,50

                dw      -50,-10,-50
                dw      50,-10,-50
                dw      50,10,-50
                dw      -50,10,-50
                dw      -50,-10,50
                dw      50,-10,50
                dw      50,10,50
                dw      -50,10,50

                dw      -50,-50,-5
                dw      50,-50,-5
                dw      50,50,-50
                dw      -50,50,-50
                dw      -50,-5,50
                dw      50,-5,50
                dw      50,5,50
                dw      -50,5,50

                dw      -50,-50,-50
                dw      50,-50,-50
                dw      50,50,-50
                dw      -50,50,-50
                dw      -50,-50,50
                dw      50,-50,50
                dw      50,50,50
                dw      -50,50,50

alfa:           dw      0
beta:           dw      0
gamma:          dw      0

sinalfa:        dd      ?
sinbeta:        dd      ?
singamma:       dd      ?

cosalfa:        dd      ?
cosbeta:        dd      ?
cosgamma:       dd      ?
```

```
xx:     dd  ?
xy:     dd  ?
xz:     dd  ?

yx:     dd  ?
yy:     dd  ?
yz:     dd  ?

zx:     dd  ?
zy:     dd  ?
zz:     dd  ?

sin8:   db  000h,002h,004h,007h,009h,00Bh,00Dh,00Fh,012h,014h
        db  016h,018h,01Ah,01Dh,01Fh,021h,023h,025h,027h,029h
        db  02Bh,02Eh,030h,032h,034h,036h,038h,03Ah,03Ch,03Eh
        db  040h,041h,043h,045h,047h,049h,04Bh,04Ch,04Eh,050h
        db  052h,053h,055h,057h,058h,05Ah,05Bh,05Dh,05Eh,060h
        db  061h,063h,064h,065h,067h,068h,069h,06Bh,06Ch,06Dh
        db  06Eh,06Fh,070h,071h,072h,073h,074h,075h,076h,077h
        db  077h,078h,079h,079h,07Ah,07Bh,07Bh,07Ch,07Ch,07Dh
        db  07Dh,07Dh,07Eh,07Eh,07Eh,07Fh,07Fh,07Fh,07Fh,07Fh
cos8:   db  07Fh,07Fh,07Fh,07Fh,07Fh,07Fh,07Eh,07Eh,07Eh,07Dh
        db  07Dh,07Dh,07Ch,07Ch,07Bh,07Bh,07Ah,079h,079h,078h
        db  077h,077h,076h,075h,074h,073h,072h,071h,070h,06Fh
        db  06Eh,06Dh,06Ch,06Bh,069h,068h,067h,065h,064h,063h
        db  061h,060h,05Eh,05Dh,05Bh,05Ah,058h,057h,055h,053h
        db  052h,050h,04Eh,04Ch,04Bh,049h,047h,045h,043h,041h
        db  040h,03Eh,03Ch,03Ah,038h,036h,034h,032h,030h,02Eh
        db  02Bh,029h,027h,025h,023h,021h,01Fh,01Dh,01Ah,018h
        db  016h,014h,012h,00Fh,00Dh,00Bh,009h,007h,004h,002h
        db  000h,0FEh,0FCh,0F9h,0F7h,0F5h,0F3h,0F1h,0EEh,0ECh
        db  0EAh,0E8h,0E6h,0E3h,0E1h,0DFh,0DDh,0DBh,0D9h,0D7h
        db  0D5h,0D2h,0D0h,0CEh,0CCh,0CAh,0C8h,0C6h,0C4h,0C2h
        db  0C1h,0BFh,0BDh,0BBh,0B9h,0B7h,0B5h,0B4h,0B2h,0B0h
        db  0AEh,0ADh,0ABh,0A9h,0A8h,0A6h,0A5h,0A3h,0A2h,0A0h
        db  09Fh,09Dh,09Ch,09Bh,099h,098h,097h,095h,094h,093h
        db  092h,091h,090h,08Fh,08Eh,08Dh,08Ch,08Bh,08Ah,089h
        db  089h,088h,087h,087h,086h,085h,085h,084h,084h,083h
        db  083h,083h,082h,082h,082h,081h,081h,081h,081h,081h
        db  081h,081h,081h,081h,081h,081h,082h,082h,082h,083h
        db  083h,083h,084h,084h,085h,085h,086h,087h,087h,088h
        db  089h,089h,08Ah,08Bh,08Ch,08Dh,08Eh,08Fh,090h,091h
        db  092h,093h,094h,095h,097h,098h,099h,09Bh,09Ch,09Dh
```

```
        db    09Fh,0A0h,0A2h,0A3h,0A5h,0A6h,0A8h,0A9h,0ABh,0ADh
        db    0AEh,0B0h,0B2h,0B4h,0B5h,0B7h,0B9h,0BBh,0BDh,0BFh
        db    0C0h,0C2h,0C4h,0C6h,0C8h,0CAh,0CCh,0CEh,0D0h,0D2h
        db    0D5h,0D7h,0D9h,0DBh,0DDh,0DFh,0E1h,0E3h,0E6h,0E8h
        db    0EAh,0ECh,0EEh,0F1h,0F3h,0F5h,0F7h,0F9h,0FCh,0FEh
        db    000h,002h,004h,007h,009h,00Bh,00Dh,00Fh,012h,014h
        db    016h,018h,01Ah,01Dh,01Fh,021h,023h,025h,027h,029h
        db    02Bh,02Eh,030h,032h,034h,036h,038h,03Ah,03Ch,03Eh
        db    040h,041h,043h,045h,047h,049h,04Bh,04Ch,04Eh,050h
        db    052h,053h,055h,057h,058h,05Ah,05Bh,05Dh,05Eh,060h
        db    061h,063h,064h,065h,067h,068h,069h,06Bh,06Ch,06Dh
        db    06Eh,06Fh,070h,071h,072h,073h,074h,075h,076h,077h
        db    077h,078h,079h,079h,07Ah,07Bh,07Bh,07Ch,07Ch,07Dh
        db    07Dh,07Dh,07Eh,07Eh,07Eh,07Fh,07Fh,07Fh,07Fh,07Fh

prog19  ends                        ;A szegmens vége.
        end   start                 ;A program vége.
```

A program vezérlő része hasonlít az előzőre, mivel a feladatuk szinte azonos. A legnagyobb eltérés a forgató rutinban található. Itt első lépésben kiszámítjuk mindhárom egyenlet x, y, z összetevőjét. Fontos ez azért, mert ezek csak szögfüggvényeket tartalmaznak, amelyeket egy forgatási fázis kiszámítása során nem változtatunk meg, így a kocka mind a 8 pontjára alkalmazhatóak. Így az egyes fázisok kiszámítása már gyorsan megy, mivel csak be kell helyettesíteni a kiszámolt értékeket az egyes szorzatokba. Tehát a képlet leegyszerűsödik a következő alakra:

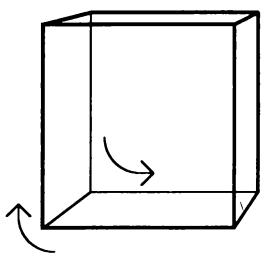
$$x' = x \cdot xx + y \cdot xy + z \cdot xz$$
$$y' = x \cdot yx + y \cdot yy + z \cdot yz$$
$$z' = x \cdot zx + y \cdot zy + z \cdot zz$$

Ahol csak az x, y, z értékeket kell változtatni egy fázison belül. A frame kiszámítása után itt is a 3D-2D leképzés következik. Ez csupán annyiban különbözik az előbbitől, hogy a perspektíva egy kicsivel hátrébb lett állítva, hogy élethűbb legyen, valamint kompenzálja a 320x200-as felbontás torzítását.

Az alakzat valós képének előállítása

A következő nagy eltérés az alakzat felrajzolásában van. Ugyanis, ha azt szeretnénk, hogy csak azok az élek látszódjanak, amelyek a valóságban

is láthatóak (ez egyébként az esetleges fillezés egyik alapfeltétele), akkor ezt az előbbi módszerrel nem lehet megoldani. Itt az a teendőnk, hogy oldalanként rajzoljuk fel az ábrát. Természetesen csak a megfelelő oldalakat. Azt pedig, hogy melyik oldal van elöl és melyik hátul, avagy mi az, amit láthatunk, úgy dönthetjük el, hogy figyeljük az oldal éleinek felrajzolási irányát. Ha pozitív, akkor felrajzolható, ha pedig negatív akkor az egy láthatatlan oldal.



A felrajzolási irány megállapítása

A forgásirány meghatározásához három pont koordinátáira van szükségünk (ha az alakzat nem tartalmaz 180°-nál nagyobb vagy egyenlő szöget). Vegyük például azt az esetet, ha a három pont: 0,0 - 0,10 - 10,10 a pozitív forgási iránynál és 10,10 - 0,10 - 0,0 a negatív forgási iránynál. A feladatunk az, hogy a három pont közötti két-két szakaszt, mint vektort számítsuk ki, amit az adott **x,** illetve **y** koordináták különbségéből kapunk.

0,0		10,10	
	→ 0,10		→ -10,0
0,10		0,10	
	→ 10,0		→ 0,-10
10,10		0,0	

Ha az igy kapott két-két vektor ellentétes koordinátáit összeszorozzuk (0x0,10x10 - -10x-10, 0x0), akkor arra a megállapításra

juthatunk, hogy a pozitív forgási iránynál a második szorzat mindig nagyobb. Tehát egy adott forgási irányt úgy lehet meghatározni, hogy kiszámoljuk az előbbi szorzatokat, és összehasonlítjuk őket egymással. Ha az első szorzat a nagyobb, akkor negatív, amennyiben a második, akkor pedig pozitív a felrajzolási irány.

A látványosság kedvéért beépítettem egy érdekes hatást keltő, de egyszerű rutin. A művelet csupán annyi, hogy két értéket összehasonlítunk egymással, és ha az adott érték kisebb, mint a másik, növeljük, ha pedig nagyobb, akkor csökkentjük az értékét. A hatás annak köszönhető, hogy ezek az értékek az egyes pontok és egy másik alakzat koordinátáinak értéke, így egyfajta metamorfózist érhetünk el. Csupán arra kell ügyelni, hogy ne képezzünk 180°-nál nagyobb vagy egyenlő szöget. Ezenkívül a morfózis képzése során figyelni kell, hogy változtattunk-e értéket, vagy sem, mert ha nem, akkor a következő fázisra kell ugrani.

További változtatás, hogy már teljes szinusz és koszinusz táblázatot használunk. Ami azonban csak ± 127 értékek között mozog, de ez a célnak tökéletesen megfelel.

9. Fejezet

A vírusok lélektana

Ki ne hallott volna már a számítógépes vírusokról. Sajnos legtöbben már találkoztunk is velük több-kevesebb örömmel. De mik is ezek a vírusok? A válasz nem túl egyszerű, de mielőtt mélyebben belemerülnénk a kérdésbe, felhívom a figyelmet arra, - mivel az itt leírt információk alapján akár mi magunk is írhatunk vírust - hogy igen erkölcstelen dolog a tudást ártó céllal felhasználni. Ezért kérek mindenkit, hogy a fejezet elolvasása után ne kezdjen el senki vírusokat gyártani, hanem az itt leírtakat inkább az ellenük való védekezésre használja fel.

Tulajdonképpen több féle vírustípus van amelyek leginkább kártevési módukban különböznek egymástól. Vannak egész szerények, amelyek csak viccelődnek (tegyél egy hamburgert az 'A' meghajtóba), vannak azonban olyanok is amelyek képesek magát a számítógépet is megrongálni (szerencsére ezek igen ritkák).

A vírusok működése

Ahhoz, hogy egy vírus terjedni tudjon, először el kell valahogyan indítani azt. Nos ez általában úgy történik, hogy hozzáfűzi magát egy .com, .exe stb. programhoz, amelynek elindulásakor először a vírus aktivizálódik és így elvégezheti teendőjét (a fertőzést) majd visszaadja a vezérlést az elindított programnak. Ugye milyen egyszerű? (Nem az) Ha az volna, akkor azok írtása is ilyen egyszerű lenne. A vírusprogramozók sajnos igen leleményesek, és elég hatásosan megnehezítik az általuk írt vírus felderítését.

Működési feltételek biztosítása

Ahhoz, hogy a felhasználó ne jöjjön rá azonnal, hogy a gépe megbetegedett, szükség van egy bizonyos lappangási időre is, amit a vírus a terjedéshez használ fel. Ezt azonban csak úgy érheti el, hogyha a fertőzött program továbbra is futtatható. Tehát célszerűen nem felülírják a programokat, hanem például a végéhez fűzik magukat. Van azonban egy kis probléma ezen feltétel teljesítésénél. Mégpedig az, hogy egy forráskód

lefordításakor az offsetcímek az adott szegmens elejéhez viszonyulnak. Tehát, ha a programot véletlenszerűen töltjük be valahová a szegmensbe, akkor a címeket ennek megfelelően kell módosítani. Nos némely vírus igen leleményes módszerrel oldja meg ezt a problémát. A vírus kódja a következő pár sorral kezdődik:

```
start:    call label1
label1:   pop bp
          sub bp,offset label1
```

Nos ez a pár, látszatra értelmetlen utasítás a következőket csinálja: elindulva a **start** címkénél ugrik a következő sorra. (ugye milyen értelmetlen?) Ennek az ugrásnak azonban nem vezérlésátadási szerepe van, hanem az, hogy egy **call** utasítás végrehajtásakor a hívási cím a verembe íródik a későbbi visszatérés érdekében. Ezt a címet olvassa ki a **pop bp** sorral, és vonja ki belőle a **label1** címke eltolási értékét. Ezzel pontosan meghatározta, hogy hol van a kód a memóriában, és a továbbiakban minden változó címzésénél **bp**-vel indexelve címez pl.: **lea si,[bp+buffer]**. Ez a pár sor azonban csak a vírus működésének biztosításához szükséges.

Védekezés a víruskeresők ellen

Azért, hogy az adott vírus a kicsit gyakorlottabb programozók elől is el tudjon rejtőzni, a vírusírók különféle cseleket alkalmaznak a vírus megírásakor. A legegyszerűbb, ami azonban a vírus keresők ellen bizonyos mértékig hatásos, hogy elinduláskor egy látszólag értelmetlennek tűnő adathalmazt, a teljes vírust át **xor**-olja egy bizonyos értékkel, ami lehet akár a file dátuma, vagy egy letárolt adat. Az így módosított program már a működőképes kód. A módszer lényege, hogy ha valamit kétszer ugyanazzal a számmal **xor**-olunk meg, ismét az eredeti számot kapjuk. Kódolt állapotban azonban a víruskeresők nem ismerik fel, főleg, ha a kódolást egy, a víruskeresőkbe előre nem beprogramozható adattal végzik (dátum, hossz, stb)

Védekezés a Debuggerek ellen

Ezt azonban mi is szimulálhatjuk egy debugger programmal, és már is nem ér semmit a védelem. De ez nem ilyen egyszerű mivel a

programozók erre is gondoltak, és bevetettek egy másik védelmet is, ami védi az előzőt, és egyben az egész vírus "biztonságát". Ezt a következő pár sorral oldották meg:

```
label1:   mov   cx,09EBh
          mov   ax,0FE05h
          jmp   $-2
          add   ah,03Bh
          jmp   $-10
          lea   bx,[di + label2]
          push  cs
          pop   es
          int   021h
          mov   al,1
          int   021h
          jmp   short label3
label2:   jmp   $
label3:   mov   byte ptr [di + lock_keys + 1],130
          mov   al,128
          out   021h,al
```

Nos ugyebár ez még értelmetlenebb mint az előző. Ez is volt a cél, mivel ez a programrészlet nem az aminek látszik, hanem egy egészen más, nagyon is ésszerű, ötletes megoldás. Ezt a következő debugg-olt listából jobban megérthetjük:

```
012C: B9EB09      MOV     CX,09EB
012F: B805FE      MOV     AX,FE05
0132: EBFC        JMP     0130
0130: 05FEEB      ADD     AX,EBFE
0133: FC          CLD
0134: 80C43B      ADD     AH,3B
0137: EBF4        JMP     012D
012D: EB09        JMP     0138
0138: F4          HLT
0139: 8D9D4701    LEA     BX,[DI+0147]
013D: 0E          PUSH    CS
013E: 07          POP     ES
013F: CD21        INT     21
0141: B001        MOV     AL,1
0143: CD21        INT     21
0145: EB02        JMP     0149
0149: C6854F0182  MOV     [DI+014F],82
```

```
014E: B082      MOV     AL,82
0150: E621      OUT     [21],AL
```

Nos... Hasonlít? Szerintem elég csekély mértékben, pedig ugyanarról a programról van szó. Na de hogyan is működik ez a dolog? Az első két sor teljesen hétköznapi módon betölt egy értéket az **ax** és a **cx** regiszterekbe. A következő ugró utasítással azonban a **mov ax,fe05** három byte-os utasítás második byte-jára ugrik vissza, minek hatására egy másik utasításnak értelmezi azt, mégpedig egy összeadásnak. Ha megnézzük a hexadecimális utasításkódokat, akkor abból előtűnik, hogy valójában hogyan is lehetséges a dolog. A következő sor egy lényegtelen **cld** parancs, aminek semmi kihatása nincs az adott rutinban (csupán így jöttek ki a byte-ok). Az **add ah,3d** szintén szerepel a forráslistában. Végrehajtása után az **ah** regiszter tartalma 25h, az **al** regiszteré pedig 03h lesz, ami majdan a 21h DOS megszakítás 25h rutinját fogja jelenteni, ami nem más, mint a megszakításvektorok beállítása. A következő két sorban szintén ugrásokkal találkozhatunk, amik nehezítik a papíron történő nyomkeresést, majd egy HLT, amelyek a debugg-olást nehezíti még akkor is, ha nem aktív a vírus. Ezt követően a **bx** regiszterbe beírja a 0147h rekesz valós címét (**di** szerepe megegyezik az előzőekben említett **bp**-vel), majd a programszegmens értékét áttölti az **es** regiszterbe, és meghívja a DOS megszakítást. Ezután az **al**-be 1-et tölt, és ismét meghívja a DOS funkciót. De mi ennek az értelme? Nem más, mint a legtöbb debugger által használt **single step** és **break point,** azaz a lépésenkénti programvégrehajtás és a töréspont-elhelyezés megszakítások átirányítása az **es:bx** által mutatott címre, ami jelen esetben a lista végére írt önmagára ugró **jmp** utasítás (itt már csak a reset segít). Ezáltal, ha megpróbálnánk visszafejteni ezt a ránézésre értelmetlen kódot, akkor abba beletörne a bicskánk, mivel az első lépés után leállna a számítógép. További szépsége a programnak, hogy a forráskódban **mov al,128** formában szereplő sort egy ügyes húzással **mov al,130**-ra változtatja, ami ugyan csekély változás, arra viszont éppen elegendő, hogy az így kiadott **out** utasítás hatására letiltja a billentyűzet-megszakítást.

Védekezés a Ctrl+Alt+Del ellen

Egyes vírusok még a Ctrl+Alt+Del billentyűkombináció igen fontos vírusterjedés megelőző hatását is figyelmen kívül hagyják, egyszerűen

elfelejtenek reagálni a kérésünkre (pofátlanság...) Na azért ez nem ilyen egyszerű. A megoldás kulcsa a billentyűzet-kezelő megszakításban rejlik. Ha ezt átírják egy saját kezelőprogram címére, akkor lehetőség nyílik az említett billentyűk figyelésére azok hatásának végrehajtása előtt. Tehát az első lépés, hogy kivesszük a gép kezéből a 09-es billentyűzet-kezelő megszakítás használatát, és átirányítjuk azt a saját rutinunkra, aminek ellenőriznie kell a három billentyű együttes lenyomását. Ezt két lépcsőből tudjuk megtenni. Az egyik az **Alt** és a **Ctrl** billentyűk vizsgálata, a második pedig a **Del** gomb ellenőrzése. Ha nincs egyszerre lenyomva a három billentyű, akkor visszaadhatjuk a vezérlést a ROM rutinnak, ellenkező esetben meg kell szabadulni a gombok lenyomási státuszától, amit egy egyszerű **out** utasítással megtehetünk (ld. függelék). Ezekután szabad a terep, olyan rutint építhetünk be, amit csak akarunk (lásd Windows) majd természetesen visszaadjuk a vezérlést a hívó programnak, de már üres bufferrel. Erre mutat egy példát a most következő program.

[Program 20]

```
prog20    segment
          assume cs:prog20,ds:nothing

keyb      dd    ?                 ;A régi vektor tárolására
                                  ;szolgáló hely.

start:    jmp   init              ;A program rezidenssé tétele.

progi:    pushf                   ;A flag elmentése
          push  ds ax si          ;Használt regiszterek elmentése
          mov   ax,40h            ;Ds regiszterbe 40h értéket
          mov   ds,ax             ;tölt, amely a BIOS paraméter-
                                  ;tábla szegmenscíme.

          mov   si,17h            ;A 17h címen a 3-2. bit az
          mov   al,[si]           ;Alt és a Ctrl billentyűk
          and   al,0ch            ;státuszát mutatja.
          cmp   al,0ch
          jnz   cont              ;Ha a kettő nincs együtt
                                  ;lenyomva, ugrik a befejezésre.

          in    al,60h            ;A billentyűzetről beolvassa
          cmp   al,53h            ;az utoljára lenyomott gomb
```

```
        jnz     cont                    ;scan kódját. Ha ez a kód egyezik
                                        ;meg a Del gomb kódjával, akkor
                                        ;ugrik a befejezésre.

        mov     al,[si]
        and     al,11110011b
        mov     [si],al

        in      al,61h
        mov     ah,al
        or      al,80h
        out     61h,al
        xchg    al,ah
        out     61h,al

        mov     al,20h                  ;Megszakítási mód vége.
        out     20h,al

        pop     si ax ds                ;Visszatérés a hívó programhoz.
        popf
        iret

cont:   pop     si ax ds                ;A használt regiszterek eredeti
                                        ;értékének visszatöltése.

        call    cs:keyb                 ;Az eredeti rutin indítása.
        iret                            ;Visszatérés.

init:   cli                             ;A megszakítások tiltása.
        xor     ax,ax                   ;Ds-t a 0. szegmensre állítja.
        mov     ds,ax
        mov     di,24h                  ;A keyboard megszakítás eredeti
        mov     ax,[di]                 ;értékét letárolja a keyb címkénél.
        mov     cs:word ptr [keyb],ax
        mov     ax,[di+2]
        mov     cs:word ptr [keyb+2],ax
        lea     ax,progi                ;Majd felülírja a címet a saját
        mov     word ptr [di],ax        ;rutinunk címével.
        mov     word ptr [di+2],cs

        sti                             ;Megszakítások engedélyezése.
        lea     dx,init                 ;A program végének címét a dx
        add     dx,256                  ;regiszterbe tölti, és hozzáad
                                        ;256-ot (fejléc + stack)
```

```
        int   27h                ;Rezidens kilépés.

prog20  ends                     ;A szegmens vége.
        end   start              ;A program vége.
```

A program érdekessége csupán annyi, hogy a megszakítás vektor kezelését nem egy DOS hívással oldja meg, hanem közvetlenül a memóriában módosítja azt. Az egyes interruptok címét könnyen kiszámolhatjuk, mivel azt tudjuk, hogy a címeiket tartalmazó táblázat a 0. szegmensen van letárolva, és egy vektorhoz 4 byte tartozik. Így a memóriacím: 0: INT x 4. Ez a megoldás egyébként teljes mértékben megegyezik a DOS-os módszerrel.

A fertőzés

Többek között ez az, amiért vírusnak nevezünk egy programot. A legegyszerűbb a .COM file-ok fertőzése, mivel ezek hossza nem haladhatja meg a szegmensméretet, és általában van még annyi hely, hogy egy aprócska rutin befészkelje magát. Tehát vegyük alapul azt az esetet, hogy valahogyan hozzánk került egy vírussal fertőzött file, és mi ezt (semmi rosszat nem sejtve) elindítjuk. Ekkor ugyebár aktiváljuk a vírust. A vírus ezután a következőket cselekszi:

1. a betöltési offsetcím kiszámítása,
2. dekódolás (XOR),
3. a betöltött program eredeti kezdőbyte-jainak visszaírása a későbbi elindítás érdekében,
4. az aktuális directory lekérése és eltárolása,
5. a főkönyvtárból kiindulva fertőzhető (.com) file-ok keresése,
6. találat esetén ellenőrzi, hogy az adott program nem e fertőzött már, ha igen, akkor újat keres,
7. ha még nem, beolvassa a program első 3 byte-ját,
8. megnézi a program hosszát (ellenőrzi, hogy elfér-e mögötte és kiszámítja a szükséges ugrási címet),
9. kiolvassa, letárolja és törli a file attribútumait,
10. az első három byte kicserélése egy JMP **vírus**-ra,
11. a file pointert a file végére állítja, és hozzáírja magát a file-hoz,
12. visszaállítja az eredeti file dátum, idő és attribútum információkat,

13. esetleges további fertőzés,
14. régi directory-pozíció visszaállítása,
15. esetleges rombolás,
16. a valós program elindítása.

Mivel a vírusoknál nagyon sokat számít a rutin hossza, igyekeznek minden rövidítési lehetőséget kihasználni, így például a DTA-t a stack-be töltik, és ott vizsgálják, nem pedig egy fenntartott adatterületen.

A vírusok között igen gyakoriak a rezidens programok. Ebben az esetben azonban a vírus indulásakor ellenőriznie kell, hogy nincs-e már a memóriában. Erre egy kiváló módszer a nem használt megszakítás vektorok kihasználása. Így egy INT hívással megvizsgálhatják, hogy van-e már aktív vírus a RAM-ban. Ezek közül elég sokat megtalálhatunk a következő listában.

A vírusok által használt megszakításvektorok

INT 21 - VIRUS - "**Perfume**" - INSTALLÁLTSÁG ELLENÖRZÉSE
    AX = 0B56h
Vissza: AX = 4952h, ha rezidens
Kapcsolódó funkciók: AX=0D20h,INT 12/AX=4350h/BX=4920h

INT 21 - VIRUS - "**Crazy Imp**" - INSTALLÁLTSÁG ELLENŐRZÉSE
    AX = 0D20h
Vissza: AX = 1971h, ha rezidens
Kapcsolódó funkciók: AX=0B56h,AH=30h/DX=ABCDh

INT 21 - VIRUS - "**Maltese Amoeba**" - INSTALLÁLTSÁG ELLENŐRZÉSE
    AX = 2B16h
    CX = 0643h
Vissza: AX = 1603h, ha installált

INT 21 - VIRUS - "**Possessed**" - INSTALLÁLTSÁG ELLENŐRZÉSE
    AH = 30h
    DX = ABCDh
Vissza: DX = DCBAh, ha installált
Kapcsolódó funkciók: AX=0D20h,AX=30F1h

INT 21 - VIRUS - "**Dutch-555**"/"**Quit 1992**" - INSTALLÁLTSÁG ELLENŐRZÉSE
    AX = 30F1h
Vissza: AL = 00h, ha rezidens
Kapcsolódó funkciók: AH=30h/DX=ABCDh,AX=330Fh

INT 21 - VIRUS - "**Burghofer**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 330Fh
Vissza: AL = 0Fh, ha rezidens (DOS Vissza AL=FFh)
Kapcsolódó funkciók: AX=30F1h,AX=33E0h

INT 21 - VIRUS - "**Oropax**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 33E0h
Vissza: AL = E0h, ha rezidens (DOS Vissza AL=FFh)
Kapcsolódó funkciók: AX=330Fh,AX=357Fh

INT 21 - VIRUS - "**Agiplan**"/"**Month 4-6**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 357Fh
Vissza: DX = FFFFh, ha installált
Kapcsolódó funkciók: AX=33E0h,AX=3DFFh

INT 21 - VIRUS - "**JD-448**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 3DFFh
Vissza: AX = 4A44h, ha rezidens
Kapcsolódó funkciók: AX=357Fh,AX=4203h

INT 21 - VIRUS - "**Shake**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 4203h
Vissza: AX = 1234h, ha rezidens
Kapcsolódó funkciók: AX=3DFFh,AX=4243h

INT 21 - VIRUS - "**Invader**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 4243h
Vissza: AX = 5678h, ha rezidens
Kapcsolódó funkciók: AX=4203h,AX=4B04h

INT 21 - VIRUS **???**
AH = 4Ah
BX = 00B6h
ES = CX
Vissza: ???
Megjegyzés: Ezt a vektort elfogja a Search&Destroy SDRes v27.03 a Novell DOS 7-tel, és feltételezhetően egyéb virusoknak is ellenörzés céljául szolgál.
Kapcsolódó funkciók: AH=0Eh/DL=ADh,AH=4Ah/BX=FFFFh,AH=D2h"VIRUS"

INT 21 - VIRUS **???**
AH = 4Ah
BX = FFFFh
CX = 0568h
SI = 0129h
DI = 0000h

ES = BP
Vissza: ???
Megjegyzés: Ezt a vektort elfogja a Search&Destroy SDRes v27.03 a Novell DOS 7-tel, és feltételezhetően egyéb virusoknak is szolgál ellenörző célból.
Kapcsolódó funkciók: AH=0Eh/DL=ADh,AH=4Ah/BX=00B6h

INT 21 - VIRUS - "**MG**", "**699**"/"**Thirteen Minutes**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 4B04h
Vissza: CF törölt, ha "MG" rezidens
AX = 044Bh if "699"/"Thirteen Minutes" rezidens
Kapcsolódó funkciók: AX=4243h,AH=4Ah/BX=FFFFh,AX=4B21h

INT 21 - VIRUS - "**Holocaust**"/"**Telefonica**" - ???
AX = 4B20h
Kapcsolódó funkciók: AX=4B04h,AX=4B21h

INT 21 C - VIRUS - "**Holocaust**"/"**Telefonica**" - **???**
AX = 4B21h
Megjegyzés: A vírus installálásának befejezésekor hívja
Kapcsolódó funkciók: AX=4B04h,AX=4B20h,AX=4B25h

INT 21 - VIRUS - "**1063**"/"**Mono**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 4B25h
Vissza: DI = 1234h, ha rezidens
Kapcsolódó funkciók: AX=4B21h,AX=4B40h

INT 21 - VIRUS - "**Plastique**"/"**AntiCad**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 4B40h
Vissza: AX = 5678h, ha rezidens
Kapcsolódó funkciók: AX=4B25h,AX=4B41h,AX=4B4Ah

INT 21 - VIRUS - "**Plastique**"/"**AntiCad**" - ???
AX = 4B41h
???
Vissza: ???
Kapcsolódó funkciók: AX=4B40h

INT 21 - VIRUS - "**Jabberwocky**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 4B4Ah
Vissza: AL = 57h, ha rezidens
Kapcsolódó funkciók: AX=4B40h,AX=4B4Bh

INT 21 - VIRUS - "**Horse-2**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 4B4Bh
Vissza: CF törölt, ha rezidens

Kapcsolódó funkciók: AX=4B4Ah,AX=4B4Dh

INT 21 - VIRUS - "**Murphy-2**", "**Patricia**"/"**Smack**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 4B4Dh
Vissza: CF törölt, ha rezidens
Kapcsolódó funkciók: AX=4B4Ah,AX=4B50h

INT 21 - VIRUS - "**Plastique-2576**"/"**AntiCad-2576**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 4B50h
Vissza: AX = 1234h, ha rezidens
Kapcsolódó funkciók: AX=4B4Dh,AX=4B53h,AX=4B60h

INT 21 - VIRUS - "**Horse**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 4B53h
Vissza: CF törölt, ha rezidens
Kapcsolódó funkciók: AX=4B50h,AX=4B55h

INT 21 - VIRUS - "**Sparse**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 4B55h
Vissza: AX = 1231h, ha rezidens
Kapcsolódó funkciók: AX=4B53h,AX=4B59h

INT 21 - VIRUS - "**Murphy-1**", "**Murphy-4**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 4B59h
Vissza: CF törölt, ha rezidens
Kapcsolódó funkciók: AX=4B50h,AX=4B5Eh

INT 21 - VIRUS - "**Brothers**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 4B5Eh
Vissza: CF törölt, ha rezidens
Kapcsolódó funkciók: AX=4B59h,AX=4B87h

INT 21 - VIRUS - "**Plastique-2576**"/"**AntiCad-2576**" - ???
AX = 4B60h
???
Vissza: ???
Kapcsolódó funkciók: AX=4B50h

INT 21 - VIRUS - "**Shirley**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 4B87h
Vissza: AX = 6663h, ha rezidens
Kapcsolódó funkciók: AX=4B5Eh,AX=4B95h

INT 21 - VIRUS - "**Zherkov-1882**" - INSTALLÁLTSÁG ELLENŐRZÉSE

AX = 4B95h
Vissza: AX = 1973h, ha rezidens
Kapcsolódó funkciók: AX=4B87h,AX=4BA7h

INT 21 - VIRUS - "**1876**"/"**Dash-em**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 4BA7h
Vissza: AX = B459h, ha rezidens
Kapcsolódó funkciók: AX=4B95h,AX=4BAAh

INT 21 - VIRUS - "**Nomenklatura**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 4BAAh
Vissza: CF törölt, ha rezidens
Kapcsolódó funkciók: AX=4BA7h,AX=4BAFh

INT 21 - VIRUS - "**948**"/"**Screenplus1**", "**Magnitogorsk**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 4BAFh
Vissza: AL = AFh, ha "Magnitogorsk" rezidens
AL = FAh, ha "948"/"Screenplus1" rezidens
Kapcsolódó funkciók: AX=4BAAh,AX=4BDDh

INT 21 - VIRUS - "**Lozinsky**"/"**Zherkov**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 4BDDh
Vissza: AX = 1234h
Kapcsolódó funkciók: AX=4BAFh,AX=4BFEh

INT 21 - VIRUS - "**Hitchcock**", "**Dark Avenger-1028**", "**1193**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 4BFEh
Vissza: AX = 1234h if "Hitchcock" rezidens
AX = ABCDh if "1193"/"Copyright" rezidens
DI = 55BBh, ha "Dark Avenger-1028" rezidens
Kapcsolódó funkciók: AX=4BDDh,AX=4BFFh"Justice"

INT 21 - VIRUS - "**USSR-707**", "**Justice**", "**Europe 92**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 4BFFh
Vissza: BL = FFh, ha "USSR-707" rezidens
DI = 55AAh, ha "Justice" rezidens
CF törölt, ha "Europe 92" rezidens
Kapcsolódó funkciók: AX=4BFEh,AX=4BFFh"Cascade",AX=5252h

INT 21 - VIRUS - "**Cascade**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 4BFFh
SI = 0000h
DI = 0000h

Vissza: DI = 55AAh, ha installált
Kapcsolódó funkciók: AX=4BFFh"Justice",AX=5252h

INT 21 - VIRUS - "**516**"/"**Leapfrog**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 5252h
Vissza: BX = FFEEh, ha rezidens
Kapcsolódó funkciók: AX=4BFFh "Cascade", AX=58CCh

INT 21 - VIRUS - "**1067**"/"**Headcrash**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 58CCh
Vissza: CF törölt, ha rezidens
Kapcsolódó funkciók: AX=5252h,AX=58DDh,AX=6969h

INT 21 - VIRUS - "**1067**"/"**Headcrash**" - EREDETI 21h VEKTOR
AX = 58DDh
Vissza: CX = A vírus kódszegmense
ES:BX = régi INT 21h vektor
Kapcsolódó funkciók: AX=5252h,AX=58CCh,AX=6969h

INT 21 - VIRUS - "**Rape-747**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 6969h
Vissza: AX = 0666h, ha rezidens
Kapcsolódó funkciók: AX=58CCh,AH=76h"VIRUS"

INT 21 - VIRUS - "**Klaeren**"/"**Hate**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AH = 76h
Vissza: AL = 48h, ha rezidens
Kapcsolódó funkciók: AX=6969h,AX=7700h"VIRUS"

INT 21 - VIRUS - "**Growing Block**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 7700h
Vissza: AX = 0920h, ha rezidens
Kapcsolódó funkciók: AH=76h,AH=7Fh

INT 21 - VIRUS - "**Squeaker**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AH = 7Fh
Vissza: AH = 80h, ha rezidens
Kapcsolódó funkciók: AX=7700h,AH=83h"VIRUS"

INT 21 - VIRUS - "**SVC**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AH = 83h
Vissza: DX = 1990h, ha rezidens
Kapcsolódó funkciók: AH=76h,AH=84h"VIRUS"

INT 21 - VIRUS - "**SVC 5.0**" or "**SVC 6.0**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AH = 84h

Vissza: DX = 1990h, ha rezidens
BH = verzió szám
Kapcsolódó funkciók: AH=83h"VIRUS",AH=89h"VIRUS"

INT 21 - VIRUS - "**Vriest**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AH = 89h
Vissza: AX = 0123h, ha rezidens
Kapcsolódó funkciók: AH=84h"VIRUS",AH=90h"VIRUS"

INT 21 - VIRUS - "**Carioca**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AH = 90h
Vissza: AH = 01h, ha rezidens
Kapcsolódó funkciók: AH=89h"VIRUS",AX=9753h"VIRUS"

INT 21 - VIRUS - "**Nina**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = 9753h
Vissza: soha (eredeti program végrehajtása), ha a vírus rezidens
Kapcsolódó funkciók: AH=90h"VIRUS",AX=A1D5h"VIRUS"

INT 21 - VIRUS - "**789**"/"**Filehider**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = A1D5h
Vissza: AX = 900Dh, ha rezidens
Kapcsolódó funkciók: AX=9753h,AX=A55Ah

INT 21 - VIRUS - "**Eddie-2**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = A55Ah
Vissza: AX = 5AA5h, ha rezidens
Kapcsolódó funkciók: AX=A1D5h,AX=AA00h

INT 21 - VIRUS - "**Blinker**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = AA00h
Vissza: AX = 00AAh, ha rezidens
Kapcsolódó funkciók: AX=A55Ah,AX=AA03h

INT 21 - VIRUS - "**Backtime**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = AA03h
Vissza: AX = 03AAh, ha rezidens
Kapcsolódó funkciók: AX=AA00h,AH=ABh

INT 21 - VIRUS - "**600**" vagy "**Voronezh**"-család - INSTALLÁLTSÁG ELLENŐRZÉSE
AH = ABh
Vissza: AX = 5555h, ha rezidens
Kapcsolódó funkciók: AX=AA03h,AX=BBBBh"VIRUS"

INT 21 - VIRUS - "**Hey You**" - INSTALLÁLTSÁG ELLENŐRZÉSE

AX = BBBBh
Vissza: AX = 6969h
Kapcsolódó funkciók: AH=ABh"VIRUS",AH=BEh"VIRUS"

INT 21 - VIRUS - "**Datalock**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AH = BEh
Vissza: AX = 1234h, ha rezidens
Kapcsolódó funkciók: AX=BBBBh,AX=BE00h

INT 21 - VIRUS - "**USSR-1049**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = BE00h
CF set
Vissza: CF törölt, ha rezidens
Kapcsolódó funkciók: AH=BEh"VIRUS",AH=C0h"VIRUS"

INT 21 - VIRUS - "**Slow**"/"**Zerotime**", "**Solano**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AH = C0h
Vissza: AX = 0300h, ha "Slow"/"Zerotime" rezidens
AX = 1234h, ha "Solano" rezidens
Kapcsolódó funkciók: AX=BE00h,AH=C1h"VIRUS",AX=C301h"VIRUS"

INT 21 - VIRUS - "**Solano**" - ???
AH = C1h
???
Vissza: ???
Kapcsolódó funkciók: AH=C0h"VIRUS"

INT 21 - VIRUS - "**Scott's Valley**" - ???
AH = C2h
???
Vissza: ???
Kapcsolódó funkciók: AH=C0h"VIRUS"

INT 21 - VIRUS - "**905**"/"**Backfont**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = C301h
DX = F1F1h
Vissza: DX = 0E0Eh, ha rezidens
Kapcsolódó funkciók: AH=C0h"VIRUS",AX=C500h"VIRUS"

INT 21 - VIRUS - "**Sverdlov**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = C500h
Vissza: AX = 6731h, ha rezidens
Kapcsolódó funkciók: AX=C301h"VIRUS",AH=C6h"VIRUS"

INT 21 - VIRUS - "**Socha**" - INSTALLÁLTSÁG ELLENŐRZÉSE

AH = C6h
Vissza: AL = 55h, ha rezidens
Kapcsolódó funkciók: AX=C500h"VIRUS",AX=C603h

INT 21 - VIRUS - "**Yankee**" vagy "**MLTI**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = C603h
CF beállítva
Vissza: CF törölt, ha rezidens
Kapcsolódó funkciók: AX=C500h"VIRUS",AX=C700h"VIRUS"

INT 21 - VIRUS - "**MH-757**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = C700h
Vissza: AL = 07h ha, rezidens
Kapcsolódó funkciók: AX=C603h"VIRUS",AH=CBh"VIRUS"

INT 21 - VIRUS - "**Piter**" - ???
AX = CA15h
???
Vissza: ???
Kapcsolódó funkciók: AH=CCh"VIRUS"

INT 21 - VIRUS - "**Milous**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AH = CBh
Vissza: AL = 07h ha rezidens
Kapcsolódó funkciók: AX=C700h"VIRUS",AX=CB02h

INT 21 - VIRUS - "**Witcode**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = CB02h
Vissza: AX = 02CBh ha rezidens
Kapcsolódó funkciók: AH=CBh"VIRUS",AH=CCh"VIRUS"

INT 21 - VIRUS - "**Westwood**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AH = CCh
Vissza: AX = 0700h ha rezidens
Kapcsolódó funkciók: AX=CB02h,AH=CDh"VIRUS",AX=D000h"VIRUS"

INT 21 - VIRUS - "**Westwood**" - ???
AH = CDh
???
Vissza: ???
Kapcsolódó funkciók: AH=CCh"VIRUS"

INT 21 - VIRUS - "**Fellowship**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = D000h
Vissza: BX = 1234h ha rezidens
Kapcsolódó funkciók: AH=CCh"VIRUS",AH=D5h"VIRUS",AX=D5AAh

INT 21 - VIRUS **???**
AH = D2h
???
Vissza: ???
Megjegyzés: Ezt a vektort elfogja a Search&Destroy SDRes v27.03 a Novell DOS 7-tel, és feltételezhetően egyéb virusoknak is szolgál ellenörzés céljául.
Kapcsolódó funkciók: AH=4Ah/BX=00B6h

INT 21 - VIRUS - **"Carfield"** - ???
AH = D5h
???
Vissza: ???
Kapcsolódó funkciók: AX=D5AAh,AH=F3h"Carfield"

INT 21 - VIRUS - **"Diamond-A"**, **"Diamond-B"** - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = D5AAh
Vissza: AX = 2A55h ,ha "Diamond-A" rezidens,
AX = 2A03h, ha "Diamond-B"-család vírus rezidens
Kapcsolódó funkciók: AX=D000h,AH=D5h"VIRUS",AX=D5AAh/BP=DEAAh

INT 21 - VIRUS - **"Dir"** - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = D5AAh
BP = DEAAh
Vissza: SI = 4321h, ha rezidens
Kapcsolódó funkciók: AX=D5AAh,AX=DADAh"VIRUS"

INT 21 - VIRUS - **"Gotcha"** - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = DADAh
Vissza: AH = A5h
Kapcsolódó funkciók: AX=D5AAh,AX=DAFEh"VIRUS"

INT 21 - VIRUS - **"Plovdiv 1.3"** - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = DAFEh
Vissza: AX = 1234h, ha rezidens
Kapcsolódó funkciók: AX=DADAh,AH=DDh"VIRUS",AH=DEh"VIRUS"

INT 21 - VIRUS - **"Jerusalem"**-család - ÁTHEJEZŐ VIRUS???
AH = DDh
CX = a másolt byte-ok száma
DS:SI -> a másolás forrása
ES:DI -> a másolás célja
Vissza: ???
Kapcsolódó funkciók: AH=E0h"VIRUS",AH=EEh"VIRUS"

INT 21 - VIRUS - "**Durban**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AH = DEh
Vissza: AH = DFh, ha rezidens
Kapcsolódó funkciók: AX=DAFEh,AX=DEDEh"VIRUS"

INT 21 - VIRUS - "**April 1st EXE**" - ???
AH = DEh
???
Vissza: ???

INT 21 - VIRUS - "**Brothers**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = DEDEh
Vissza: AH = 41h, ha rezidens
Kapcsolódó funkciók: AH=DEh"VIRUS",AH=E0h"VIRUS"

INT 21 - VIRUS - "**Jerusalem**", "**Armagedon**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AH = E0h
Vissza: AX = 0300h, ha "Jerusalem" rezidens
AX = DADAh, ha "Armagedon" rezidens
Kapcsolódó funkciók: AH=DEh"VIRUS",AX=DEDEh"VIRUS",AX=E00Fh

INT 21 - VIRUS - "**8-tunes**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = E00Fh
Vissza: AX = 4C31h, ha rezidens
Kapcsolódó funkciók: AH=E0h"VIRUS",AH=E1h"VIRUS"

INT 21 - VIRUS - "**Mendoza**", "**Fu Manchu**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AH = E1h
Vissza: AX = 0300h, ha "Mendoza" rezidens
AX = 0400h, ha "Fu Manchu" rezidens
Kapcsolódó funkciók: AX=E00Fh,AH=E4h"VIRUS"

INT 21 - VIRUS - "**Anarkia**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AH = E4h
Vissza: AH = 04h, ha rezidens
Kapcsolódó funkciók: AH=E1h"VIRUS",AH=E7h"VIRUS"

INT 21 - VIRUS - "**Spyer**"/"**Kiev**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AH = E7h
Vissza: AH = 78h, ha rezidens
Kapcsolódó funkciók: AH=E4h"VIRUS",AX=EC59h

INT 21 - VIRUS - "**Terror**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = EC59h
Vissza: BP = EC59h, ha rezidens
Kapcsolódó funkciók: AH=E7h"VIRUS",AH=EEh"VIRUS"

INT 21 - VIRUS - "**Jerusalem-G**", "**Pregnant**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AH = EEh
Vissza: AX = 0300h, ha "Jerusalem-G" rezidens
AL = 05h, ha "Pregnant" rezidens
Kapcsolódó funkciók: AH=DDh"VIRUS",AX=EC59h,AH=F0h"VIRUS"

INT 21 - VIRUS - "**Frere Jacques**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AH = F0h
Vissza: AX = 0300h, ha rezidens
Kapcsolódó funkciók: AH=EEh"VIRUS",AH=F1h"VIRUS"

INT 21 - VIRUS - "**F1-337**" - **???**
AH = F1h
???
Vissza: ???
Kapcsolódó funkciók: AH=F0h"VIRUS",AX=F1E9h

INT 21 - VIRUS - "**Tremor**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = F1E9h
Vissza: AX = installálási státusz
CADEh installált, és a hívó program megfertőzve
F100h nem installált (normál DOS visszatérési érték)
különben installált, de a hívó program nem fertőzött
Kapcsolódó funkciók: AH=F1h"VIRUS",AX=F2AAh

INT 21 - VIRUS - "**PcVrsDs**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = F2AAh
Vissza: AH = AAh, ha rezidens
Kapcsolódó funkciók: AH=F1h"VIRUS",AH=F3h"VIRUS"

INT 21 - VIRUS - "**Carfield**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AH = F3h
Vissza: AX = 0400h, ha rezidens
Kapcsolódó funkciók: AH=D5h"Carfield",AX=F2AAh,AH=F7h"VIRUS"

INT 21 - VIRUS - "**GP1**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AH = F7h
Vissza: AX = 0300h, ha rezidens
Kapcsolódó funkciók: AH=F0h"VIRUS",AH=FBh"VIRUS"

INT 21 - VIRUS - "**Cinderella**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AH = FBh
Vissza: AH = 00h, ha rezidens
Kapcsolódó funkciók: AH=F7h"VIRUS",AX=FB0Ah

INT 21 - VIRUS - "**dBASE**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = FB0Ah
Vissza: AX = 0AFBh, ha rezidens
Kapcsolódó funkciók: AH=FBh"VIRUS",AH=FCh"VIRUS"

INT 21 - VIRUS - "**Troi**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AH = FCh
Vissza: AL = A5h, ha rezidens
Kapcsolódó funkciók: AX=FB0Ah"VIRUS",AH=FDh"VIRUS"

INT 21 - VIRUS - "**Border**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AH = FDh
Vissza: AH = 13h, ha rezidens
Kapcsolódó funkciók: AH=FCh"VIRUS",AH=FEh"VIRUS"

INT 21 - VIRUS - "**483**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AH = FEh
Vissza: AH = 00h, ha rezidens
Kapcsolódó funkciók: AH=FDh"VIRUS",AX=FE01h

INT 21 - VIRUS - "**Flip**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = FE01h
Vissza: AX = 01FEh, ha rezidens
Kapcsolódó funkciók: AH=FEh"VIRUS",AX=FE02h

INT 21 - VIRUS - "**2468**"/"**Tequila**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = FE02h
Vissza: AX = 01FDh, ha rezidens
Kapcsolódó funkciók: AX=FE01h,AX=FE03h,AX=FEDCh"VIRUS"

INT 21 - VIRUS - "**2468**"/"**Tequila**" - VÍRUS ÜZENET MEGJELENÍTŐ
AX = FE03h
Kapcsolódó funkciók: AX=FE02h

INT 21 - VIRUS - "**Black Monday**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = FEDCh
Vissza: AL = DCh, ha rezidens
Kapcsolódó funkciók: AX=FE02h,AH=FFh"VIRUS"

INT 21 - VIRUS - "**Sunday**", "Tumen 0.5", "Hero" - INSTALLÁLTSÁG ELLENŐRZÉSE
AH = FFh
Vissza: AH = 00h, ha "Tumen 0.5" or "Hero" rezidens
AX = 0400h, ha "Sunday" rezidens
Kapcsolódó funkciók: AX=FEDCh"VIRUS",AX=FF0Fh

INT 21 - VIRUS - "**Twins**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = FF10h
Vissza: AL = 07h, ha rezidens
Kapcsolódó funkciók: AX=FF0Fh,AX=FFFEh

INT 21 - VIRUS - "**08/15**"/"**Many Fingers**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = FFFEh
Vissza: AX = 0815h, ha rezidens
Kapcsolódó funkciók: AX=FF10h,AX=FFFFh

INT 21 - VIRUS - "**Ontario**", "**Year 1992**"/"**B1M92**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = FFFFh
Vissza: AX = 0000h, ha "Ontario" resident
AX = 1992h, ha "Year 1992"/"B1M92" resident
Kapcsolódó funkciók: AX=FF0Fh,AX=FFFFh/CX=0000h,INT 6B"VIRUS"

INT 21 - VIRUS - "**Revenge**" - INSTALLÁLTSÁG ELLENŐRZÉSE
AX = FFFFh
CX = 0000h
Vissza: CX = 0006h, ha rezidens
Kapcsolódó funkciók: AX=FFFFh,INT 6B"VIRUS"

INT 31 - VIRUS - "**Vacsina**" széria - INSTALLÁLTSÁG ELLENŐRZÉSE (NOT A VECTOR!)
Megjegyzés: ha a Vacsina vírusból egy installált, a megszakítás alsó byte-ja mindig magában foglalja az INT 30 CP/M JMP utolsó információ byte-ját a maradék három byte a 7Fh 39h és ezt követően a Vacsina verzió száma
Kapcsolódó funkciók: INT 21/AX=FFFFh"VIRUS",INT 32"VIRUS"

INT 32 - VIRUS - állítólag a "**Tiny**" Vírusok használják
Kapcsolódó funkciók: INT 21/AX=FFFFh"VIRUS",INT 31"VIRUS",INT 44"VIRUS"

INT 32 - VIRUS - "**Plovdiv 1.3**"/"**Damage 1.3**" - EREDETI INT 21h VEKTOR
Kapcsolódó funkciók: INT 31"VIRUS",INT 9E"VIRUS"

INT 44 - VIRUS - "**Lehigh**" - EREDETI INT 21h VEKTOR
Kapcsolódó funkciók: INT 32"VIRUS",INT 60"VIRUS",INT 70"VIRUS",INT 9E"VIRUS"

INT 60 - VIRUS - "**Zero Bug**" - INSTALLÁLTSÁG ELLENŐRZÉSE
Installált, ha a kezelő szegmensének offset 103h tartalmazza a "ZE" byte-okat
Kapcsolódó funkciók: INT 32,INT 44"VIRUS",INT 61"SEMTEX"

INT 61 - VIRUS - "**SEMTEX**"/"**Screen Trasher**" - INT 21h helyettesítő
Megjegyzés: Az eredeti 21h vektor vírus másolata a 61h megszakításban

Kapcsolódó funkciók: INT 21h,INT 60"VIRUS",INT 6B"VIRUS"

INT 6B - VIRUS - "**Saddam**" - EREDETI INT 21h VEKTOR
Kapcsolódó funkciók: INT 21/AX=FFFFh,INT 61"VIRUS",INT 70"VIRUS"

INT 70 - VIRUS - "**Stupid**" - EREDETI INT 21h VEKTOR
Kapcsolódó funkciók: INT 6B"VIRUS",INT 9E"VIRUS",INT E0"VIRUS"

INT 87 - VIRUS - "**ZeroHunt**" - VIRUSOS KÓD (Nem vektor!)
Megjegyzés: A ZeroHunt vírus másolata rezidens kód a 0000h:021Ch címtől kezdődően
Kapcsolódó funkciók: INT 8B"VIRUS"

INT 8B - VIRUS - "**ZeroHunt**" - INSTALLÁLTSÁG ELLENŐRZÉSE (Nem vektor!)
Megjegyzés: ha a ZeroHunt vírus rezidens, akkor ez a vektor a következők egyike:
EE83h:019Bh (ZH-411) vagy EE83h:019Fh (ZH-415)
Kapcsolódó funkciók: INT 70"VIRUS",INT 87"VIRUS",INT 9C"VIRUS"

INT 9C - VIRUS - "**INT13**" - EREDETI INT 13h VEKTOR
Kapcsolódó funkciók: INT 8B"VIRUS",INT 9D"VIRUS",INT 9E"VIRUS",INT 9F"VIRUS"

INT 9D - VIRUS - "**INT13**" - ROM INT 13h BELÉPÉSI PONT
Megjegyzés: Ezt a vektort használja az INT 2F/AH=13h hívás eredményének tárolására
Kapcsolódó funkciók: INT 2F/AH=13h,INT 9C"VIRUS",INT 9E"VIRUS",INT 9F"VIRUS"

INT 9E - VIRUS - "**INT13**" - EREDETI INT 21h VEKTOR
Kapcsolódó funkciók: INT 70"VIRUS",INT 9C"VIRUS",INT 9D"VIRUS",INT E0"VIRUS"

INT 9F - VIRUS - "**INT13**" - INT 13h VEKTOR TÁROLÁSA
Megjegyzés: amikor fertőz egy file-t, az INT13 vírus megragadja az INT13 megszakítást és ezt a vektort használja a létező INT13 vektor későbbi visszaállításához.
Kapcsolódó funkciók: INT 9C"VIRUS",INT 9D"VIRUS"

INT E0 - VIRUS - "**Micro-128**" - ???
Megjegyzés: Micro-128 szintén felülírja az interrupt tábla felső részét
Kapcsolódó funkciók: INT 9E"VIRUS",INT F1"VIRUS"

INT F1 - VIRUS - "**Violetta**" - ???
Kapcsolódó funkciók: INT E0"VIRUS",INT FF"VIRUS"

INT FF - VIRUS - "**Violetta**" - ???
Kapcsolódó funkciók: INT E0"VIRUS",INT F1"VIRUS"

Amint az látható, elég változatos az megszakítások felhasználási területe. A legérdekesebb talán a vírus futtatására való felhasználásuk. Mivel minden vektorhoz egy 4 byte-os mutató tartozik, és elég sok nem használt megszakítás található még, tökéletesen megfelel a vektortáblázat felső része egy rövid víruskód befogadására. Ez a táblázat azonban ahhoz is segítséget nyújthat, hogy az itt említett kórokozókat idő előtt felderíthessük. A vírusok elleni küzdelemhez mindenkinek sok sikert és kitartást...

10. Fejezet

A megfelelő időzítés

Igen sok programban van szükség valamilyen időzítésre, lassításra, várakozásra stb. Ennek több módja is van, melyek közül a legegyszerűbb (igen rövid időzítésnél) a **nop**, vagy hosszabb esetén egy ciklus. Azonban ezek mindegyike a processzor sebességétől függő. Ahhoz, hogy minden géptípuson azonos sebességgel fusson az adott program, a CPU órajelétől független dologhoz kell viszonyítanunk. A legkézenfekvőbb a belső óra használata, ugyanis ez egy olyan szerkezet, ami a processzortól függetlenül ketyeg, és másodpercenként 18.2 -szer növeli meg egy számláló értékét.

"Black Screen" Képernyővédő

Ha elegendő ez az időosztás akkor kényelmesen használhatjuk eme adottságot. Sőt, van a PC-n egy **timer** megszakítás is, ami azonos ütemben hajtódik végre. Ha az adott feladatban éppen erre van szükség, akkor akár egy rezidens programot is indíthatunk ezzel a módszerrel. A 21. példaprogram ennek az időzítésnek a felhasználására mutat be egy példát, annyi kiegészítéssel, hogy nem csak a **timer** megszakításra "ül" rá, hanem a billentyűzetet kezelő rutin elé is befészkeli magát.

[Program 21]

```
.386
prog21      segment use16
            assume cs:prog21,ds:prog21

start:      jmp   init                  ;Ugrás az inicializálásra.

timer       dd    ?                     ;A 08h ROM rutin címe.
keyb        dd    ?                     ;A 09h ROM rutin címe.
flag        db    0                     ;A saver aktiválása.
counter           dw                    ;Időzítés számláló.
colors      db    768 dup (0)           ;A paletta számára fenntartott
                                        ;terület.

progi:      pushf
            pusha
```

```
                push  ds

                push  cs                    ;Cs áttöltése ds-be.
                pop   ds

                inc   word ptr [counter]    ;A számláló értékének növelése.
                cmp   word ptr [counter],1*18*60
                jnz   end                   ;Ellenőrzi, hogy elteltt-e a
                                            ;megadott időtartam.

                mov   cx,768                ;Ha igen, kiolvassa a paletta
                lea   di,colors             ;értékeit a lefoglalt memória-
                mov   dx,3c7h               ;területre
                mov   al,0
                out   dx,al
                add   dx,2

get_pal:        in    al,dx
                mov   [di],al
                inc   di
                loop  get_pal

                mov   cx,768
                mov   dx,3c8h
                mov   al,0
                out   dx,al
                inc   dx

clr_pal:        mov   al,0                  ;majd kinullázza azokat, ezzel
                out   dx,al                 ;minden színt feketére állít.
                loop  clr_pal
                mov   byte ptr [flag],255   ;Jelzi, hogy a képernyővédő
                                            ;működésbe lépett.

end:            pop   ds                    ;A timer-rel indított rutin vége.
                popa

                call  cs:timer              ;Az eredeti rutinra ugrás.
                iret

keyboard:       pushf                       ;Minden billentyű leütéskor
                pusha                       ;végrehajtódik ez a rutin.
                push  ds
```

```
            push  cs
            pop   ds

            cmp   byte ptr [flag],0       ;Ellenőrzi, hogy aktív-e a
            jz    endkey                  ;saver, ha nem, akkor a paletta
                                          ;visszaállító rutint átugorja.

            mov   byte ptr [flag],0       ;Az eredeti paletta színértékek
            mov   cx,768                  ;visszaállítása.
            lea   di,colors
            mov   dx,3c8h
            mov   al,0
            out   dx,al
            inc   dx

put_pal:    mov   al,[di]
            out   dx,al
            inc   di
            loop  put_pal

endkey:     mov   word ptr [counter],0    ;A számláló nullázása.
            pop   ds                      ;A saját billentyűkezelő rutin
            popa                          ;vége.

            call  cs:keyb                 ;Az eredeti keyboard rutin hívása.
            iret

init:       push  cs
            pop   ds
            cli
            mov   ax,3508h                ;A 08h vektor címének kiolvasása
            int   21h
            mov   word ptr [timer],bx
            mov   word ptr [timer+2],es

            mov   ax,2508h                ;és átírása.
            lea   dx,progi
            int   21h

            mov   ax,3509h                ;A 09h vektor címének kiolvasása
            int   21h
            mov   word ptr [keyb],bx
            mov   word ptr [keyb+2],es
```

```
        mov   ax,2509h          ;és átírása.
        lea   dx,keyboard
        int   21h

        sti

        lea   dx,init           ;Rezidenssé válás.
        add   dx,512
        int   27h

prog21  ends
        end   start
```

Ez a rövid kis program (egy kicsit tovább fejlesztve) igen hasznos segítség lehet a monitor élettartamának növelésében. A program működése azon alapszik, hogy a *timer* megszakítás átirányításával indítunk egy programot, ami növeli egy számláló értékét. Ha ez az érték elérte az általunk megadott határt, akkor végrehajt egy rutint, ami jelen esetben a palettaszínek letárolása és kinullázása, de ide bármilyen egyéb program is beilleszthető. A program azonban csak úgy tudja ellátni feladatát, ha egy további megszakítást is átveszünk a géptől, amivel ellenőrizzük a billentyűzetet. Ez azért szükséges, mert különben bármit csinálunk, az idő leteltével elsötétülne a kép, és úgy is maradna, így azonban minden billentyű leütésénél nullázzuk a számláló értékét, továbbá ha már elindult a képernyő védő funkció, a palettaszíneket is visszaállítjuk az eredetire.

"Zene" készítése PC-n

Egy másfajta időzítésre mutat példát a 22. példaprogram. Amint azt már említettem, a PC-k tartalmaznak egy időzítő áramkört, ami mellesleg akár programozható is. Ennek az időzítőnek az egyik csatornáját használja a belső óra. A másik kettő közül az egyik a RAM frissítésért, és a DMA kezelésért felelős, a másik pedig a PC hangszórójának szolgáltat a jelet. Most ez utóbbit fogjuk egy kicsit megpiszkálni, egy valószínűleg mindenki által jól ismert dallammal, a "Yankee Doodle" című világslágert fogja eljátszani a következő kis program. Itt az időzítés két formájával is találkozhatunk. Az egyik a hang magasságának beállítása, a másik pedig az egyes hangok hosszának meghatározása.

[Program 22]

```
prog22      segment
            assume cs:prog22, ds:prog22

prg_8255    equ   61h          ;A 8255 programozási portcíme.
prg_8253    equ   43h          ;Az osztó programozási portcíme.
timer       equ   42h          ;Az osztó osztási értéket beállító
                               ;címe.

start:      push  cs           ;A cs regiszter áttöltése ds-be
            pop   ds           ;a vermen keresztül.

            lea   si,[data]    ;A zene adatainak kezdőcímét
                               ;si regiszterbe tölti.

player:     mov   al,0b6h      ;Az osztó kiválasztása.
            out   prg_8253,al

            mov   bx,word ptr [si]   ;A hangmagasság betöltése.

            or    bx,bx        ;Ha nulla, akkor vége.
            jz    end_play

            mov   ax,034ddh    ;Az 1193181.7Hz alapfrekvenciához
            mov   dx,12h       ;szükséges osztási arány
            div   bx           ;kiszámolása azért, hogy az
            out   timer,al     ;általunk beállított érték
            mov   al,ah        ;szólaljon meg a hangszórón.
            out   timer,al

            in    al,prg_8255  ;A hangszóró bekapcsolása
            or    al,3
            out   prg_8255,al

            xor   ah,ah        ;A gép belső órájának aktuális
            mov   bx,[si+2]    ;értékéhez hozzáadjuk a hang
            int   1ah          ;hosszának értékét (1/18 scc),
            add   bx,dx        ;és addig várunk, amíg az óra
wait_loop:  int   1ah          ;el nem éri ezt az értéket.
            cmp   dx,bx
            jnz   wait_loop

            in    al,prg_8255  ;A hangszóró kikapcsolása.
```

```
            and   al,0fch
            out   prg_8255,al

            add   si,4                  ;A következő hang.
            jmp   player

end_play:   mov   ah,4ch                ;Visszatétés az operációs
            int   21h                   ;rendszerhez.

data:       dw    262,6,262,6,293,6,329,6,262,6,329,6,293,6,196,6
            dw    262,6,262,6,293,6,329,6,262,12,262,12
            dw    262,6,262,6,293,6,329,6,349,6,329,6,293,6,262,6
            dw    246,6,196,6,220,6,246,6,262,12,262,12
            dw    220,6,246,6,220,6,174,6,220,6,246,6,262,6,220,6
            dw    196,6,220,6,196,6,174,6,164,6,174,6,196,7
            dw    220,6,246,6,220,6,174,6,220,6,246,6,262,6,220,7
            dw    196,6,262,6,246,6,293,6,262,12,262,12
            dw    0,0
prog22      ends                        ;A szegmens vége.
            end   start                 ;A program vége.
```

Ahhoz, hogy a számítógépet hang kiadására bírjuk, először be kell állítani az áramkört arra az üzemmódra, amit mi használni szeretnénk, jelen esetben frekvencia osztásra. Ebben az üzemmódban az alap órajelet (ami 1193181.7Hz), osztja el az általunk beállított 16 bites értékkel. Így a legkisebb frekvencia, amit kiadhatunk, 18.2Hz. De az osztási értékek letárolása (mint hangmagasság), igen babrás munka mivel ez az érték ráadásul fordítottan arányos a frekvenciával, így ezt senkinek sem tanácsolom. Tehát a megoldás az, hogy az általunk beírt értékből, ami jelen esetben a hang frekvenciája, kiszámítunk egy osztási értéket, aminek hatására a gép az általunk kívánt hangot szólaltatja meg. Ezt úgy tehetjük meg, hogy egy fix értéket elosztunk a kívánt hangmagassággal. Ezt az értéket természetesen úgy kell megválasztani, hogy az eredményképpen kapott szám valóban az legyen, amit szeretnénk. Ha a hangok magasságát frekvenciában adjuk meg, akkor 1234DDh ez az érték. A dallam esetleges hamisságáért elnézést kérek, de az eredeti vírusból lett kibányászva a dallam adat része.

Saját dallam írásánál nagy segítség a következő táblázat, amely az egyes hangokhoz tartozó frekvenciaértékeket tartalmazza.

Frekvencia táblázat

	C_2 szub-kontra	C_1 kontra	C nagy
C	16,352	32,703	65,406
Cisz	17,324	34,648	69,296
D	18,354	36,708	73,416
Disz	19,445	38,891	77,782
E	20,602	41,203	82,407
F	21,827	43,654	87,307
Fisz	23,125	46,249	92,499
G	24,499	48,999	97,999
Gisz	25,957	51,913	103,826
A	27,500	55,000	110,000
Aisz	29,135	58,270	116,541
H	30,868	61,735	123,471

	C kis	C egyvonalas	C kétvonalas
C	130,813	161,626	523,251
Cisz	138,591	277,183	554,365
D	146,832	293,665	587,330
Disz	155,563	311,127	622,254
E	164,814	329,628	659,255
F	174,614	349,228	698,456
Fisz	184,997	369,994	739,989
G	195,998	391,995	783,991
Gisz	207,652	415,395	830,609
A	220,000	440,000	880,000
Aisz	233,082	466,164	932,328
H	246,492	493,883	987,767

	C háromvonalas	C négyvonalas	C ötvonalas
C	1016,502	2093,005	4186,009
Cisz	1108,731	2217,461	4434,922

D	1174,659	2349,318	4698,363
Disz	1244,508	2489,016	4978,032
E	1318,510	2637,021	5274,042
F	1396,913	2793,826	5587,652
Fisz	1479,978	2959,955	5919,910
G	1567,982	3135,964	6271,928
Gisz	1661,219	3322,438	6644,876
A	1760,000	3520,000	7040,000
Aisz	1864,655	3729,310	7458,620
H	1975,533	3951,066	7902,132

A megfelelő hangmagasság beállítása mellett a másik nagy feladat az, hogy az a megadott hang pontosan addig szóljon, ameddig mi szeretnénk. Itt ezt most a belső óra BIOS-ból történő kiolvasásával oldottam meg. Mivel értékét ez is növeli másodpercenként 18.2-szer, amit közelítőleg 18-nak vehetünk, a fél hangok hosszát így a 9, a negyedekét pedig a 4.5 kb.: 5 érték jelenti, amit úgy oldunk meg, hogy lehívjuk az óra aktuális értékét, hozzáadjuk a megadott hang hosszúságot és addig várunk, míg a számláló el nem éri a kívánt értéket.

Egyedi hang effektusok létrehozása

Mi van akkor, ha nekünk nem elegendő a 18.2-szeri megszakítás, esetleg gyorsabbra lenne szükségünk? Ekkor sincs probléma, mert ez is programozható. Mindössze arra kell ügyelni, hogy program befejezése előtt visszaállítsuk az eredeti értékeket. A 23. program ezen lehetőség felhasználásával mutat be egy hangeffektust előállító eljárást, ami az előző két programban már megismert módszereken alapszik.

[Program 23]

```
prog23      segment
            assume cs:prog23, ds:prog23

prg_8255    equ   61h          ;A 8255 programozási portcíme.
prg_8253    equ   43h          ;Az osztó programozási portcíme.
timer       equ   42h          ;Az osztó oztási érték beállító
```

```
                                    ;címe.

oldtimer    dd      ?               ;A timer eredeti vektora.

start:      push    cs              ;Cs áttöltése ds-be.
            pop     ds

            cli                     ;A megszakítás tiltása.

            mov     ax,3508h        ;A timer megszakítási vektor
            int     21h             ;értékének letárolása az
            mov     word ptr [oldtimer],bx  ;oldtimer címkénél.
            mov     word ptr [oldtimer+2],es
            mov     ax,2508h        ;Az új cím beállítása.
            lea     dx,progi
            int     21h
            sti                     ;A megszakítás engedélyezése.

            in      al,prg_8255     ;A hangszóró bekapcsolása.
            or      al,3
            out     prg_8255,al

            mov     al,36h          ;A timer átállítása 1/18-as
            out     43h,al          ;értékről 1/100-as értékre.
            mov     ax,11932
            out     40h,al
            mov     al,ah
            out     40h,al

player:     mov     al,0b6h         ;Az osztó kiválasztása.
            out     prg_8253,al

            mov     bx,word ptr [tone]  ;A hangmagasság betöltése.
            push    bx

            cmp     bx,400          ;400Hz-nél hagyja abba a rutint.
            jc      end_play

            mov     ax,034ddh       ;Az 1193181.7Hz alapfrekvenciához
            mov     dx,12h          ;szükséges osztási arány
            div     bx              ;kiszámolása azért, hogy az
            out     timer,al        ;álltalunk beállított érték
            mov     al,ah           ;szólaljon meg a hangszórón.
            out     timer,al
```

```
            pop   bx                        ;Addig vár, amíg a tone értéke
test:       cmp   word ptr [tone],bx        ;nem csökken.
            jz    test

            jmp   player                    ;Vissza a hangképzés elejére.

end_play:   cli                             ;A megszakítások tiltása.
            in    al,prg_8255               ;A hangszóró kikapcsolása.
            and   al,0fch
            out   prg_8255,al

            mov   dx,word ptr [oldtimer]    ;Az eredeti vektorcím visszaállítása
            mov   ax,word ptr [oldtimer+2]
            mov   ds,ax
            mov   ax,2508h
            int   21h

            mov   al,36h                    ;A számlálási időzítés
            out   43h,al                    ;visszaállítása 1/18-ra.
            mov   al,0
            out   40h,al
            out   40h,al

            sti                             ;A megszakítások engedélyezése.

            mov   ah,4ch                    ;Visszatétés az operációs
            int   21h                       ;rendszerhez.


progi:      pushf                           ;Megszakítás kezelő rutin, ami
            sub   word ptr cs:[tone],100    ;csökkenti a frekvencia
            call  cs:oldtimer               ;értékét.
            iret

tone:       dw    2000                      ;A hang kezdőértéke.

prog23      ends                            ;A szegmens vége.
            end   start                     ;A program vége.
```

A hangszóróhoz tartozó osztót itt is azonosan kezeljük, mint az előző programban. A különbség csupán annyi, hogy a hang magasságát nem előre letárolt értékek egymásutánja határozza meg, hanem egy

folyamatosan csökkenő érték, amelynek kezdőértéke jelen esetben 2kHz és a vége pedig 400Hz. A csökkentést egy, a 08h megszakítással vezérlet rutin végzi, melynek sebességét 18.2Hz-ről átállítottuk 100Hz-re, így másodpercenként 100-szor csökkenti százzal a számláló értékét. Így kb. 1.6 másodpercig tart a program végrehajtása. A pontos időzítésre itt azért van szükség, mert ha egy ciklus segítségével írnánk meg ezt a rutint például egy 386DX40-es gépen, akkor az egy 100MHz-es Pentiumon csak egy rövid csippanást eredményezne, ami nem az igazi cél.

11. Fejezet

Ellipszis rajzolása

Visszatérve még egy kicsit a grafikához, az IBM PC Gyakorlati Assembly című könyvemben már szó esett a vonal, illetve kör rajzolásáról. A probléma mindkét műveletnél a lebegőpontos számok kikerülése volt. Végülis egy rekurzív ciklus segítségével oldottuk meg a kérdést. Azonban előfordulhat az az eset is, amikor mi egy kört szeretnénk rajzolni, de az ami a képernyőre kerül az a legnagyobb jóindulattal is csak egy tojáshoz hasonlít. Mi is ennek az oka? Mint tudjuk, az egyes video üzemmódokhoz különböző felbontási értékek tartoznak. Ez még nem is volna nagy probléma.

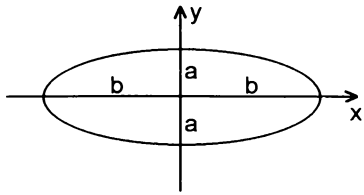
Képtorzítás

A baj akkor kezdődik, ha ugyanazt az ábrát, amit pl.: 640*200-as felbontásnál megrajzoltunk, megnézünk 640*480-ba. (Elképzelhető, hogy ráismerünk...) Sajnos az egyes felbontások igen nagy torzítási tényezőket tartalmaznak. Az ideális képernyő átló arány kb.: 1:1.3 - 1:1,333 aminek a 640*480, 800*600, 1024*768 felbontások felelnek meg. Az összes többi standard felbontás bizonyos mértékben torzít. Tehát, ha nekünk egy kört kell rajzolni a képernyőre, akkor számolni kell ezzel a torzítási tényezővel, azaz egy ellipszist kell a kör helyett számolni. De természetesen az is előfordulhat, hogy valóban egy ellipszisre van szükségünk.

Az ellipszis matematikai egyenlete

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

ahol:



Növekményes algoritmus

Azonban ahhoz, hogy ezen képlet alapján számoljuk ki az összes pontot, szükségünk lenne egy gyökvonó algoritmusra, és egy nyugágyra. Célszerűbb inkább egy hasonló algoritmushoz folyamodni, mint a már említett két esetben (kör, vonal). Továbbá mivel az ellipszis egy szimmetrikus alakzat, elegendő csupán 1/4-ét kiszámítani, és a maradékot x,y tükrözéssel a képernyőre rajzolni. Az algoritmus lényege, hogy futás közben a kínálkozó lehetőségek közül dönti el, hogy melyik van a legközelebb a valós értékhez (oldal vagy függőleges irányba lépjünk-e tovább). A felhasznált képlet a következőképpen néz ki:

$$b^2 \cdot x^2 + a^2 \cdot y^2 - a^2 \cdot b^2 = 0$$

Azért, hogy az algoritmus működése könnyebben érthető legyen, először egy magasabb szintű nyelvben megírt programot mutatok be, csak aztán az assembly rutint.

A rutin 'C' nyelven

```
Ellipse( xc, yc, a0, b0 )
int     xc,yc;                          /* az ellipszis középpontja */
int     a0,b0;                          /* átlók */
{
        int     x = 0;
        int     y = b0;

        long    a = a0;                 /* 32 bites pontosság */
        long    b = b0;

        long    sqra = a * a;           /* a hurok szélső értékeinek*/
        long    twosqra = 2 * sqra;     /* megadása*/
```

```
    long   sqrb = b * b;
    long   twosqrb = 2 * sqrb;

    long   d;
    long   dx,dy;

    d = sqrb - sqra*b + sqra/4L;
    dx = 0;
    dy = twosqra * b;

    while (dx<dy)
    {
     Set4Pixels( x, y, xc, yc, PixelValue );

     if (d > 0L)
     {
      --y;
      dy -= twosqra;
      d -= dy;
     }

     ++x;
     dx += twosqrb;
     d += sqrb + dx;
    }

    d += (3L*(sqra-sqrb)/2L - (dx+dy)) / 2L;

    while (y>=0)
    {
     Set4Pixels( x, y, xc, yc, PixelValue );

     if (d < 0L)
     {
      ++x;
      dx += twosqrb;
      d += dx;
     }

     --y;
     dy -= twosqra;
     d += sqra - dy;
    }
}
```

```
Set4Pixels( x, y, xc, yc, n )
int     x,y;
int     xc,yc;
int     n;
{
        SetPixel( xc+x, yc+y, n );
        SetPixel( xc-x, yc+y, n );
        SetPixel( xc+x, yc-y, n );
        SetPixel( xc-x, yc-y, n );
}
```

Ennek ismeretében talán valamivel könnyebb lesz megérteni az assembly forráslistát, amit természetesen úgy készítettem el, hogy ne gyors, hanem inkább érthető legyen. Ezért, ha valaki fel kívánja használni ezt a rutint, akkor a memória-változókat célszerű kicserélni regiszterekre és egy kicsit ajánlatos optimalizálni az eljárást.

[Program 24]

```
.386
prog24      segment use16
            assume cs:prog24,ds:prog24

xc          equ   dword ptr 160          ;A középpont koordinátái.
yc          equ   dword ptr 100
a           equ   dword ptr 100          ;A tengelyek értékei.
b           equ   dword ptr 50
sqra        equ   dword ptr a*a
sqrb        equ   dword ptr b*b
twosqra     equ   dword ptr sqra*2
twosqrb     equ   dword ptr sqrb*2

x           dd    0
y           dd    b
deltax      dd    0
deltay      dd    twosqra*b
d           dd    sqrb-sqra*b+sqra/4

start:      push  cs                     ;Cs áttöltése ds-be
            pop   ds

            mov   ax,0a000h              ;A videoparaméterek
```

```
                mov   es,ax                   ;beállítása.
                mov   ax,13h
                int   10h

                mov   edi,xc                  ;A középpont memóriacímének
                mov   eax,yc                  ;kiszámítása.
                mov   ebx,320
                mul   ebx
                add   edi,eax

.1_prog24:      mov   eax,dword ptr [deltax]  ;while (dx<dy) ...
                cmp   eax,dword ptr [deltay]
                jnc   .3_prog24
                call  set4pixels

                mov   eax,dword ptr [d]       ;if d>0 ...
                or    eax,eax
                jle   .2_prog24
                dec   dword ptr [y]
                sub   dword ptr [deltay],twosqra
                mov   eax,dword ptr [deltay]
                sub   dword ptr [d],eax
.2_prog24:      inc   dword ptr [x]
                add   dword ptr [deltax],twosqrb
                mov   eax,sqrb
                add   eax,dword ptr [deltax]
                add   dword ptr [d],eax
                jmp   .1_prog24

.3_prog24:      mov   eax,3*(sqra-sqrb)/2
                sub   eax,dword ptr [deltax]
                sub   eax,dword ptr [deltay]
                cdq
                mov   ebx,2
                idiv  ebx
                add   dword ptr [d],eax

.5_prog24:      mov   eax,dword ptr [y]       ;while y>=0 ...
                or    eax,eax
                js    .6_prog24
                call  set4pixels

                mov   eax,dword ptr [d]       ;if d<0
                or    eax,eax
```

```
            jns     .5_prog24
            inc     dword ptr [x]
            add     dword ptr [deltax],twosqrb
            mov     eax,dword ptr [deltax]
            add     dword ptr [d],eax

.5_prog24:  dec     dword ptr [y]
            sub     dword ptr [deltay],twosqra
            mov     eax,sqra
            sub     eax,dword ptr [deltay]
            add     dword ptr [d],eax
            jmp     .4_prog24

.6_prog24:  xor     ax,ax                   ;Visszatérés az op. rendszerhez.
            int     16h
            mov     ax,3
            int     10h
            mov     ax,4c00h
            int     21h

set4pixels  proc

            mov     ebx,dword ptr [y]       ;A 4 pont felrajzolása.
            mov     eax,320
            mul     ebx
            add     eax,dword ptr [x]
            mov     ebx,eax
            mov     al,7
            mov     es:[di+bx],al
            neg     bx
            mov     es:[di+bx],al

            mov     ebx,dword ptr [y]
            neg     ebx
            mov     eax,320
            mul     ebx
            add     eax,dword ptr [x]
            mov     ebx,eax
            mov     al,7
            mov     es:[di+bx],al
            neg     bx
            mov     es:[di+bx],al

            ret
```

```
set4pixels    endp

prog24        ends
              end   start
```

További lehetőségek

Amennyiben grafikai programot írunk, akár azt is megtehetjük, hogy keresünk egy üres interrupt területet, és egy saját rutingyűjteményt készítünk a programjaink számára, amelyeket szükség esetén egy **int** hívással aktiválhatunk. Ilyen módszerrel igen gyorsan meg lehet írni az egyes feladatokat. Csupán a saját rutin gyűjtemény megírása tart valamivel tovább,de ezt csak egyszer kell megtennünk.

12. Fejezet

Az installálás

A könyv utolsó témájaként egy olyan dolgot szeretnék bemutatni, amire minden olyan programnál szükség van, amit nem a magunk szórakoztatására írunk. Nos ez nem más, mint az installáló program.

Vannak, akik ezt egyszerűen megoldják egy **copy** paranccsal, vannak azonban olyanok is akik egy bonyolult grafikai programot írnak, ami esetenként meghaladja az installálni kívánt program hosszát. A magam részéről szeretnék valahol a két szélső érték között maradni.

Egy installáló program feladata

Az installáló programnak alapvetően 2 feladata van. Az első, hogy előkészítse a célegységet a program befogadására, a második pedig, hogy a lemezről, CD-ről stb. a megfelelő helyre juttassa a kívánt állományokat. Ezeket az egységeket további apróbb részekre bonthatjuk. Ezek a következők:

Az előkészítés

Mint említettem, ez több fázisból áll. Befolyásolja a lehetőségeket, hogy mekkora beavatkozási lehetőséget adunk a felhasználónak (minél kevesebbet, annál könnyebb). Ha a lehetőségek száma nulla, akkor elegendő megvizsgálni a célegységet, hogy az létezik-e, van-e elegendő szabad hely a telepítéshez. Ha a feltételeknek megfelel, létrehozhatjuk a megfelelő alkönyvtárakat, és jöhet a 2. lépcső. Ha megengedjük a felhasználónak, hogy ő adja meg a telepítés célegységét, akkor annak értelmezése után kell eldönteni, hogy létezik-e az az egység stb.

A telepítés

Általában nem azért írnak installáló programot, hogy egyszerűen átmásoljanak egy programot a megfelelő helyre, mivel ezt egy batch file-ból is megtehetnénk. Alapvetően két cél van. Az egyik a tömörített tárolás, a másik a másolandó (kitömörítendő) file-ok számának minimalizálása.

Mivel ha 234 darab 10 byte-os file-t másolnánk át az egyik helyről a másikra, az többször annyi ideig tartana, mint egy 2340 byte-os file átmásolása. Az ok a drive fejének a tartalomjegyzék és az adatblokk között fölöslegesen megtett mozgása. Ha ezt kiküszöböljük, gyorsíthatjuk a telepítést. A másik szempont a tárolás helyének minimalizálása. Nem mindegy, hogy egy program tíz lemezen vagy csak kettőn van rajta, ezt azonban már nehezebb megoldani, mivel vagy egy külső tömörítő programot vezérelünk, vagy saját algoritmust írunk (az utóbbi sose lesz olyan jó...) Tehát marad a külső program használata. Mivel igen sok a shareware program e téren, van válogatási lehetőségünk, azonban fontos, hogy a tömörítő program az aktuális csomag árát nem befolyásolhatja. A leginkább használt tömörítők a **pkzip** és az **arj** programok. A most bemutatásra kerülő rutin megegyezik a könyvhöz tartozó lemez-mellékleten használttal.

[Program 25]

```
.386
prog25      segment use16
            assume cs:prog25,ds:prog25

rss         dw   ?                  ;A stack szegmenscímének
rsp         dw   ?                  ;és mutatójának átmeneti
                                    ;tárolására szolgáló hely.

start:      push cs                 ;Ds beállítása cs-re
            pop  ds

            lea  bx,endprog25       ;A programon kívüli memória
            add  bx,512             ;felszabadítása a külső
            shr  bx,4               ;program számára.
            mov  ah,4ah
            int  21h

            mov  ax,3               ;Képernyő inicializálás.
            int  10h
            mov  ax,0b800h
            mov  es,ax

            lea  si,text1           ;Az első szöveg kiíratása.
            call write
```

```
                mov   ah,36h                ;A rendelkezésre álló szabad
                mov   dl,3                  ;terület ellenörzése a 'C'
                int   21h                   ;winchesteren.
                mul   ebx
                mov   ebx,ecx
                mul   ebx
                cmp   eax,400*1024          ;Ha nincs 400 Kbyte
                jnc   .1_prog25             ;szabad hely a merevlemezen, akkor
                jmp   .3_prog25             ;a telepítés nem lehetséges.

.1_prog25:      lea   si,text2              ;A text2 kiíratása.
                call  write

                mov   dx,3d4h               ;A kurzor eltüntetése.
                mov   ax,0ff0eh
                out   dx,ax

                xor   ax,ax                 ;Billentyűvárás.
                int   16h

                mov   ah,39h                ;Az asmbook2 könyvtár
                lea   dx,dirname            ;létrehozása.
                int   21h

                mov   ax,0501h              ;Átváltás az 1-es képernyő-
                int   10h                   ;lapra.

                mov   word ptr cs:[rss],ss  ;A stack címének lementése.
                mov   word ptr cs:[rsp],sp

                lea   si,command            ;Külső parancs indítása.
                int   2eh

                mov   ss,word ptr cs:[rss]  ;A stack címének visszatöltése.
                mov   sp,word ptr cs:[rsp]

                mov   ax,0500h              ;A 0. képernyőlap visszaállítása.
                int   10h

                mov   ax,0b800h             ;A 3. szöveg kiírása.
                mov   es,ax
                push  cs
                pop   ds
                lea   si,text3
```

```
                call    write

                mov     dx,3d4h         ;A kurzor eltüntetése.
                mov     ax,0ff0eh
                out     dx,ax

.2_prog25:      xor     ax,ax           ;Várakozás egy billentyű
                int     16h             ;leütésre.

                mov     ax,3            ;Képernyőtörlés
                int     10h

                mov     ah,4ch          ;Visszatérés az op. rendszerhez.
                int     21h

.3_prog25:      lea     si,text4        ;A hibaüzenet kiírása.
                call    write
                jmp     .2_prog25

write           proc

.1_write:       mov     al,[si]
                inc     si
                cmp     al,255          ;A 255 a koordinátamegadást
                jnz     .2_write        ;jelöli.
                mov     al,[si]
                inc     si
                shl     al,1
                xor     ah,ah
                mov     di,ax
                mov     al,[si]
                inc     si
                mov     bl,160
                mul     bl
                add     di,ax
                jmp     .1_write

.2_write:       cmp     al,254          ;A 254 kódnál a color értékébe
                jnz     .3_write        ;írja a következő adatot.
                mov     al,[si]
                inc     si
                mov     byte ptr [color],al
                jmp     .1_write
```

```
.3_write:   cmp   al,0                    ;A nulla érték, a text
            jz    .4_write                ;végét jelöli.
            mov   ah,byte ptr [color]
            mov   es:[di],ax
            add   di,2
            jmp   .1_write

.4_write:   ret

write       endp


text1:      db    254,9,255,5,2,"+",68 dup ("-"),"+",255,5,3
            db    5 dup ("¦",68 dup (32),"¦",10 dup (32))
            db    255,5,8,"+",68 dup ("-"),"+"

            db    254,12,255,21,3,"IBM PC Gyakorlati Assembly Haladóknak"
            db    255,24,4,"Lemezmelléklet telepítö program"
            db    254,11,255,21,6,"A példaprogramokat és a könyvet írta:"
            db    255,34,7,"Agárdi Gábor",0

text2:      db    254,9,255,10,20,"+",58 dup ("-"),"+",255,10,21
            db    "¦",58 dup (32),"¦"
            db    255,10,22,"+",58 dup ("-"),"+"
            db    254,138,255,15,21,"Nyomjon meg egy gombot a telepítés"
            db    " megkezdéséhez !",0

text3:      db    255,15,21,"      A telepítés befejezödött...      ",0

text4:      db    254,9,255,9,20,"+",60 dup ("-"),"+",255,9,21
            db    "¦",60 dup (32),"¦"
            db    255,9,22,"+",60 dup ("-"),"+"
            db    254,138,255,11,21,"A telepítéshez nincs elegendö szabad hely"
            db    "a winchesteren !",0


command:    db    30,"arj x -y book.arj c:\asmbook2",13
dirname:    db    "c:\asmbook2",0
color:      db    ?
endprog25:

prog25      ends
            end   start
```

A memória felszabadítása

A program első lépésben felszabadítja a fölöslegesen lefoglalt memóriát, mivel egy program indításakor a DOS az összes rendelkezésre álló memóriát lefoglalja. Ha be szeretnénk tölteni egy külső programot, először helyet kell biztosítani neki a memóriában. Ezt a 4ah DOS rutin segítségével tehetjük meg, ahol a **bx** regiszterben kell megadni a lefoglaltan tartott memória méretét paragrafusokban (16 byte).

A második lépés a text képernyő inicializálása és a *Copyright* szöveg kiíratása.

A szabad hely ellenőrzése

Ezután történhet a rendelkezésre álló szabad terület méretének lekérdezése. Erre a célra a DOS 36h rutinja szolgál, ahol **dl** regiszterbe kell megadni azt, hogy melyik meghajtó adataira vagyunk kíváncsiak. Válaszul a gép az **ax** regiszterben a clusterenkénti szektorszámot (vagy 0ffffh-t hiba esetén), **bx**-ben a lehetséges clusterek számát, **cx**-ben szektoronkénti byte-ok számát és a **dx** regiszterben a drive-on található clusterek számát. Ezen adatok birtokában már ki tudjuk számítani a rendelkezésre álló terület méretét oly módon, hogy az **ax, bx, cx** regiszterek értékét összeszorozzuk egymással. Az eredmény a szabad kapacitás byte-ban. Ha ez nagyobb vagy egyenlő az általunk megadottnál, akkor elkezdődhet a telepítés, egyébként hibaüzenetet kell kiírni és a telepítés félbeszakad.

A következő lépcső a **text2** felirat kiíratása és egy billentyű-leütés várása az induláshoz. Esztétikai szempontból az ilyen várakozásoknál a kurzort jó messzire pozícionáljuk, hogy még véletlenül se villogjon bele a képbe.

A telepítés első lépcsője a tartalomjegyzék bejegyzés létrehozása, majd, hogy az előbb elkészített képernyőnk megmaradjon, átlapozunk az 1-es képernyőlapra.

Külső parancs indítása

A következő funkció a lelke az egész programnak. A DOS lehetőséget ad egy DOS parancs végrehajtására, legyen az külső vagy belső, teljesen mindegy. A dolog egyetlen szépséghibája, hogy a regiszterek értéke visszatéréskor bizonytalan, nem meghatározható. Éppen ezért a hívás előtt le kell tárolni a stack pontos helyzetét, mivel itt van tárolva az összes információ a visszatéréshez. Az egyetlen biztos pont a **cs** regiszter. Tehát a címzést ehhez kell viszonyítani. A művelet elvégzése után vissza lehet tölteni a stack értékét, és lehet tovább folytatni a programot.

Az int 2eh használata

Az int 2eh megszakítás szolgál arra, hogy bármilyen DOS műveletet végrehajthassunk a program belsejéből. A használata igen egyszerű, a **ds:si** regiszterbe kell tölteni egy memóriarekesz címét, ahol az elindítandó program és a további paraméterek találhatók. Ezen blokk felépítése a következő: az 1. byte a string hosszát határozza meg (*enterrel* együtt). Ezután következik maga a parancs, majd egy lezáró 13 (enter) kód. Tapasztalataim szerint, ha az első byte értéke nagyobb, mint a valós érték, nem okoz problémát, ellenkező esetben azonban furcsán viselkedhet.

A telepítés elvégzése után nincs más dolgunk, mint az utolsó szöveget is a képernyőre írni, majd egy gombnyomásra visszatérni a dos-hoz.

Tárgymutató

Ára 1.076,–Ft