LÁSZLÓ JÓZSEF

HANGKÁRTYA PROGRAMOZÁSA PASCAL ÉS ASSEMBLY NYELVEN

COMPUTERBOOKS

LÁSZLÓ JÓZSEF

HANGKÁRTYA PROGRAMOZÁSA PASCAL ÉS ASSEMBLY NYELVEN

LEKTOR
ÁRVAI LÁSZLÓ

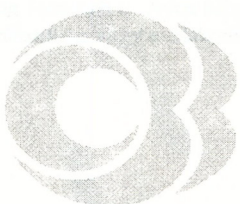
COMPUTERBOOKS
BUDAPEST,1996

A könyv készítése során a Szerzők és a Kiadó a legnagyobb gondossággal járt el. Ennek ellenére hibák előfordulása nem kizárható. Az ismeretanyag felhasználásának következményeiért sem a Kiadó sem a Szerző felelősséget nem vállal.

második változatlan kiadás

© Kiadó: ComputerBooks Kiadói Kft
1126 Bp., Tartsay Vilmos u. 12.
Tel.: 175-15-64; tel./fax: 175-35-91
Felelős kiadó: ComputerBooks Kft ügyvezetője
ISBN: 963 618 091 1
Borítógrafika: Székely Edith
Nyomdai munkák: PRESSMAN Bt.

HANGKÁRTYA PROGRAMOZÁSA PASCAL ÉS ASSEMBLY NYELVEN



COMPUTERBOOKS

Tartalom

Előszó **1**

Bevezetés **3**

1. Alapfogalmak **7**
1.1. Analóg jelek osztályozása 7
1.2. A hang 8
1.2.1. Hullámforma 10
1.3. AM, FM és a többiek 12
1.3.1. Amplitúdómoduláció (AM) 12
1.3.2. Frekvenciamoduláció (FM) 13
1.3.3. Hangszín 13
1.3.4. Spektrum 13
1.3.5. Burkológörbe, ADSR generátor 14
1.4. Analóg jelek digitális feldolgozása 18
1.4.1. Mintavételezés és kvantálás 18
1.4.2. Analóg-digitális átalakítás (ADC) 22
1.4.3. Digitális-analóg átalakítás (DAC) 24
1.4.4. Periodikus függvények Fourier-analízise (sorba fejtés) 25
1.5. Zene, zene, zene 29
1.5.1. Kotta és hangjegyek 29
1.5.2. Hangsorok, akkordok 30
1.5.3. Gitárakkordok 31

2. A beépített Assembler használata **33**
2.1. Assembly blokkok 33
2.2. Direktívák 34
2.3. Szimbólumok 35
2.4. Címkék 36
2.5. Assembly eljárások és függvények 37
2.5.1. Paraméterek és visszaadott értékek 38

3. Az ADLIB hangkártya **43**
3.1. Az ADLIB kártya regisztereinek elérése 43
3.2. Az ADLIB regiszterkészlete 46
3.3. Az ADLIB kártya programozása 54
3.3.1. Az ADLIB unit 54
3.3.2. Egyetlen hang kiadása 60
3.3.3. Több csatorna használata 61
3.3.4. A ritmushangszerek használata 63
3.4. Megszakításvezérelt lejátszóprogram 65
3.4.1. Események 66
3.4.2. Az adathalmaz formátuma 69
3.4.3. A lejátszó (*player*) 72

Típusok ... 72
Állandók ... 74
Változók ... 74
Eljárások ... 75
A főprogram ... 84
3.4.4. A fordító (*compiler*) ... 85
Típusok ... 86
Állandók ... 87
Változók ... 87
Eljárások ... 88
A főprogram ... 95
Szintaktikai szabályok ... 96
3.4.5. Példaprogramok a lejátszóhoz ... 97
Egyszerű hangsor egy csatornára ... 97
Összetett hangsor egy csatornára ... 98
Akkordok több csatornán ... 99

4. A SOUND BLASTER hangkártya ... 101
4.1. ADLIB kompatibilitás ... 101
4.2. A DSP egység ... 102
4.2.1. A DSP egység inicializálása ... 103
4.2.2. Parancs és adat kiírása a DSP egységre ... 103
4.2.3. Adat beolvasása a DSP-ről ... 104
4.2.4. DSP átviteli módok felvételkor ... 104
4.2.5. DSP átviteli módok lejátszáskor ... 104
4.2.6. ADPCM ... 105
4.2.7. DSP parancsok ... 105
4.3. A DSP egység programozása ... 106
4.3.1. A DSP unit ... 107
Típusok, állandók ... 107
Eljárások, függvények ... 109
4.3.2. Hullámformák előállítása a DSP egységgel ... 111
Változók ... 112
Eljárások ... 112
A főprogram ... 113
4.3.3. Hangfelvétel direkt módon a DSP-vel ... 115
4.3.4. Hangfelvételek visszajátszása direkt módon ... 117
4.3.5. Hosszabb hangfelvételek készítése ... 119
4.3.6. Visszajátszás ... 122
4.4. DMA ... 124
4.4.1. A DMA vezérlő programozása ... 125
4.4.2. A DMA átvitel korlátai ... 129
4.4.3. Az IRQ vezérlő programozása ... 130
4.4.4. Az SB_DMA unit ... 130
Állandók, változók ... 131
Eljárások ... 133
4.4.5. Felvétel DMA átvitellel ... 140
4.4.6. Lejátszás DMA átvitellel ... 141
4.4.7. Időállandók kiszámítása ... 142

4.4.8. Nagy sebességű átviteli módok ... 142
Nagy sebességű felvétel (ADC) ... 142
Nagy sebességű lejátszás (DAC) ... 143
4.5. Összegzés ... 143

5. A Sound Blaster Pro hangkártya ... 145
5.1. Hardverbeállítások ... 146
5.2. Csatlakozók ... 146
5.3. I/O-kiosztás ... 147
5.4. Az FM chip programozása ... 148
5.4.1. Egyéb módosítások ... 150
5.5. A keverő programozása ... 150
5.5.1. Regisztertükrök ... 155
5.5.2. Az SBPRO unit ... 156
5.6. Sztereó hang a kimeneten ... 161

6. A Sound Blaster 16 hangkártya ... 165
6.1. A Sound Blaster 16 lehetőségei ... 166
6.1.1. Hardverbeállítások ... 166
6.1.2. Csatlakozók ... 167
6.1.3. Telepítés (installálás) ... 168
6.2. Az OPL–3 FM chip programozása ... 168
6.2.1. Négyoperátoros FM hanggenerálás ... 169
6.2.2. OPL–3 regiszterkészlet ... 170
Állandók, változók ... 175
Eljárások ... 175
6.2.3. Négyoperátoros hang ... 176
6.3. A továbbfejlesztett DSP egység (*ASP*) ... 178
6.3.1. DSP parancsok ... 178
6.3.2. Bájtszekvenciális parancsok ... 179
6.4. Egyszerű egyciklusos DMA átvitel ... 182
6.5. DMA átvitel automatikus inicializálással ... 182
6.6. A keverő programozása ... 184
6.6.1. Az SB16MIX unit ... 189

7. Sound Blaster AWE 32 ... 197
7.1. Belső felépítés ... 197
7.1.1. Joystick port ... 198
7.1.2. MIDI port ... 198
7.1.3. Bus Interface ... 198
7.1.4. MCU (MIDI Controller Unit) ... 198
7.1.5. ASP (Advanced Sound Processor) ... 198
7.1.6. D/A ... 199
7.1.7. Mixer ... 199
7.1.8. Waveblaster ... 199
7.1.9. MCD Interface ... 199
7.1.10. Amplifier ... 199
7.1.11. EMU8000 subsystem ... 200

8. A GRAVIS ULTRASOUND hangkártya 205
8.1. A Gravis Ultrasound lehetőségei 205
8.2. Hardver áttekintés 206
8.2.1. Csatlakozók 206
8.2.2. I/O címek 207
8.2.3. A GUS DRAM memória 208
8.2.4. MIDI interfész 208
8.2.5. Joystick 208
8.2.6. A GF1 hangprocesszor (Voice Sound Synthetizer) 209
8.2.7. Különböző verziójú kártyák 210
8.3. A Gravis Ultrasound regiszterei 210
8.4. Regiszterek 211
8.4.1. MIDI regiszterek 212
8.4.2. GF1 globális regiszterek 213
8.4.3. GF1 csatornaparaméter regiszterek 217
8.4.4. Általános regiszterek 224
8.5. Törtszámok 229
8.6. Automatikus hangerő-szabályozás 230
8.7. „Megszakadok . . . " 231
8.8. A GUS unit 232
Állandók 233
Eljárások, függvények 234
8.8.1. A Gravis Ultrasound tesztelése 244
8.8.2. Egyszerű szinuszos hang előállítása 244
8.8.3. Oktávok 246
8.8.4. Hangminta lejátszása 247
8.8.5. Megszakítások 248

9. Zenefájl formátumok 251
9.1. Az SBI fájlok formátuma 251
9.1.1. Az SBI unit 256
9.2. A CMF fájlok formátuma 260
9.2.1. CMF fejléc 261
9.2.2. A CMF fájlok lejátszása 264
9.3. A VOC fájlok formátuma 273
9.4. A WAV fájlok formátuma 278
9.5. A MOD fájlok formátuma 281
9.5.1. Effektus parancsok 285
9.5.2. Extra effektus parancsok 289
9.5.3. Hangminták 293
9.5.4. MOD fájlok kilistázása 293
9.6. A MID fájlok formátuma 300
9.6.1. Az idő tárolása: dinamikus adathossz 301
9.6.2. MIDI események 302
9.6.3. Hang csatornaüzenetek 302
9.6.4. Rendszerüzenetek 304
9.6.5. MID fejléc 306
9.6.6. Exkluzív üzenetek 308
9.6.7. Metaesemények 309

9.6.8. A GENERAL MIDI szabvány 312
9.6.9. A ROLAND GS MIDI szabványa 312

10. MOD lejátszó Gravis kártyára 313
Hiányosságok 322

11. Összegzés 323
ADLIB 323
Sound Blaster 1.0 323
Sound Blaster 1.5 324
Sound Blaster 2.0 324
Sound Blaster Pro 1.0 324
Sound Blaster Pro 2.0 325
Sound Blaster 16 325
Sound Blaster AWE 32 325
Gravis Ultrasound 326

A függelék 327

B függelék 329

C függelék 339

D függelék 343

E függelék 347

F függelék 349

G függelék 351

Technikai adatok 353

Irodalomjegyzék 355

Tárgymutató 357

Előszó

Nem lehet kétséges, hogy korunk egyik legdinamikusabban fejlődő tudományága a számítógép-technika. A mikroelektronika fejlődése hívta életre ezt az igen fiatal tudományt, amely néhány évtized alatt meghódította a világot, eljutott a legkülönbözőbb felhasználási területekre, és szinte pillanatok alatt uralkodóvá vált mindenhol. Mai világunk már szinte működni sem tudna a számítógépek nélkül. Egyre többen ismerik meg a számítógépet, annak használatát, és ami még fontosabb, programozását is.

Ahogyan a számítógépek terjedtek és fejlődtek, úgy vált egyre nyilvánvalóbbá, hogy milyen sok mindenre is lehet használni őket. Tudnak statisztikákat készíteni és könyvelni, szöveget szerkeszteni és műszaki rajzot készíteni, tervezni, irányítani és ellenőrizni; képesek látni és hallani, vagy éppen megszólalni, ha arra van szükség. A számítógépes hangfeldolgozás igen széles skálájú tudomány, amely magában foglalja a *digitális jelfeldolgozást*, a *digitális beszédfeldolgozást*, a *hangszintetizálást* és a *hangfelismerést* is. Hogy mindezekre teljes részletességgel kitérjünk, ahhoz még tíz ilyen könyv is kevés lenne. Itt most csak az IBM PC hangkeltési lehetőségeiről lesz szó. Az IBM PC-hez több különböző típusú hangkártyát gyártanak, amelyek közül három-négy eléggé elterjedt Magyarországon is. Ezeknek a kártyáknak a programozását írjuk le ebben a könyvben.

A programozáshoz szükség lesz valamilyen programozási nyelvre is. A hangkártyákat szerintünk — mint általában a legtöbb hardverelemet — alacsony szintű nyelven lehet programozni a leghatékonyabban. Ezért a könyv példáinak megértéséhez elengedhetetlen az **ASSEMBLY** nyelv alapos ismerete. Programjainkat **BORLAND PASCAL**-ban fogjuk írni, amelyben lehetőség van assembly betétek elhelyezésére.

Az első fejezetekben a digitális jelfeldolgozás és a hangkeltés alapfogalmairól szólunk. Megnézzük majd, hogy mik is azok a jelalakok, milyen, zenében használatos fogalmakat kell ismerni, hogyan történik az analóg jelek mintavételezése és a digitális hangminták lejátszása. Ez után a különböző hangkártyák regisztereit és azok használatát ismertetjük. Rövid példaprogramok mutatják majd be az adott lehetőségeket. Nem állítjuk persze, hogy mindez könnyű lesz. Sajnos ahhoz, hogy valaki hatékonyan tudjon programozni egy hangkártyát, egy kicsit zenésznek, egy kicsit fizikusnak és matematikusnak és természetesen gyakorlott programozónak is kell lennie! Példáink éppen ezért sohasem törekednek majd optimális sebességre vagy méretre, mindig csak az adott feladatok egy-egy egyszerű megoldását adják.

Reméljük, hogy a könyv fejezetei hozzásegítik majd az Olvasót a hangkártyák használatának elsajátításához, és talán egy-egy apróbb programozói trükkel is gazdagítják tudását.

László József

ELENDER® COMPUTER

ELENDER

Budapesten :
VIII. Hungária krt. 8. Tel.: 114-0532, 134-5214 Fax: 133-4347
IX. Ferenc krt. 16. Tel./Fax: 218-2858 XIII. Csángó u. 13. Tel./Fax: 270-3097
Vidéken :
4025 Debrecen, Piac u. 57. Tel./Fax: (52) 413-795
6721 Szeged, Madách u. 15. Tel./Fax: (62) 310-269
8200 Veszprém, Zrínyi u. Botev üzletház Tel./Fax: (88) 428-235
9700 Szombathely, Hunyadi u. 45. Tel./Fax: (94) 312-265
7624 Pécs, Klimó Gy. u. 13. Tel./Fax: (72) 312-820

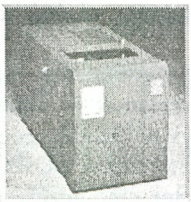
Maxtor

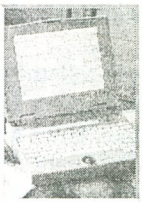
PCMCIA III.
*** winchesterek ***
memória kártyák *
garancia: 1 év
ENHANCED IDE
*** winchesterek ***
garancia: 2 év

*** Komplett számítógépek ***
*** hálózati elemek ***
*** perifériák * szoftverek ***
*** pénztárgépek ***
*** GSM telefonok ***

Maxoptix

OPTIKAI drive-ok
OPTIKAI lemezek
JukeBox-ok





Tulip computers

*** PCI alaplap ***
és
*** vezérlőkártyák ***
*** 3 év garancia ***
*** 486SX-Pentium 90 ***
*** Kedvező árak ***

Quantum

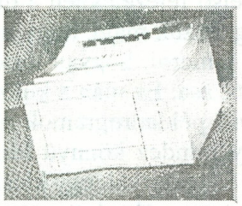
SCSI II.
winchesterek

540 MB - 4.3 GB

3-5 év garancia

hp

*** Lézer ***
és
*** Tintasugaras ***
nyomtatók
*** Scanner-ek ***



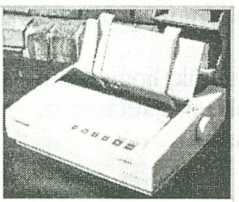
MDPT

Fast SCSI II.
vezérlőkártyák
minden jelenlegi
operációs rendszer támogatja
*** DOS * WINDOWS * Novell ***
*** OS/2 * WINDOWS NT * UNIX ***

PCI Apple Raid vezérlők

SAMSUNG

*** Számítógépek ***
*** mátrix és lézer ***
nyomtatók
*** telefonok ***
*** faxok ***
*** pénztárgépek ***



Bevezetés

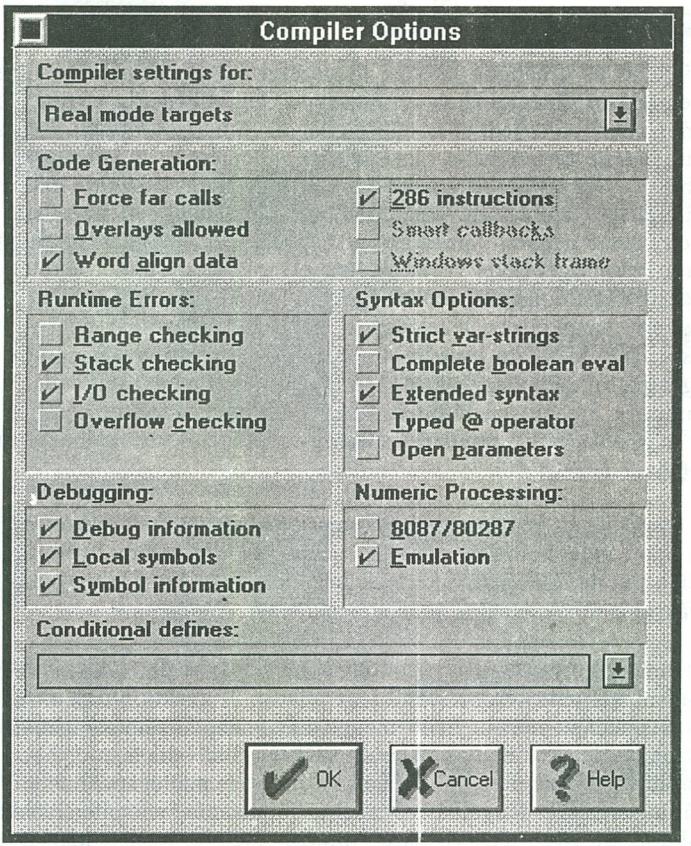
A hangfeldolgozás és a hangok tárolási módszerei az elmúlt néhány évben a digitális technikai megoldások felé tolódtak el. Az egyik legjobb példa erre a Compact Disk, amely rohamos ütemben terjedt el az egész világon. A hangok digitális úton való előállításáról és tárolásáról megoszlik a szakemberek véleménye. A konzervatívok szerint a digitális technika elvesz valamit a zene hangzásából, megváltoztatja azt. Mások szerint viszont a digitálisan tárolt hang jobb, pontosabb, könnyebben kezelhető. Bizonyára mindkét tábor véleményében van igazság. Ezt azonban nehéz megítélni. Ami viszont biztos: mára a számítógépekkel szemben már elvárás, hogy minél tökéletesebb és jobb hangképzési és hangfeldolgozási lehetőségekkel jeleskedjenek. Az IBM PC-t eredetileg csak igen szerény hangképzési lehetőséggel látták el tervezői. A gépben egy egyszerű hangszóró (*speaker*) található csupán, amelyen egy négyszögjel-generátor segítségével állandó hangerejű, változtatható magasságú hangot lehet megszólaltatni. Persze a találékony programozók hamarosan rájöttek, hogy más módszerekkel némileg többet is ki lehet hozni a *speaker*ből (impulzusszélesség-moduláció), de a hang még így sem tökéletes. :-(

Mivel a PC képességeit részint azok a kiegészítő kártyák határozzák meg, amelyeket külön-külön lehet megvásárolni és utólag beépíteni, ezért várható volt, hogy a hangkeltésre is előbb-utóbb különböző kártyák jelennek meg. Ez a könyv ezekről a leggyakoribb kártyákról szól.

A hangkártyák programozása — lévén, hogy a hangkártyák olyan kiegészítők, amelyeket alacsony szinten, I/O címeken keresztül lehet elérni — gyakorlatilag a regiszterek megismerését és használatát jelenti. Ezért megpróbáljuk részletesen és korrektül ismertetni majd a regiszterkészleteket. Sajnos az ehhez szükséges technikai információkat a gyártók nem szívesen teszik közzé. Ezért sok olyan információt adunk majd, amelyeket a könyv szerzője saját módszerrel — nyomkövetéssel és fáradságos visszafejtésekkel — „csikart ki" a kártyákból. Az ilyen „kísérletekre" minden esetben eredeti kártyákon került sor, ezért a kompatíbilis (nem az eredeti gyártótól származó) kártyák esetén apróbb eltérések fordulhatnak elő.

A könyv fejezetei tartalmazzák majd a kártyák programozásához szükséges ismereteket és módszereket. Az egyes lehetőségeket példaprogramok segítségével mutatjuk majd be, ezek lehetőleg rövidek és tömörek lesznek. A programok megértéséhez ismerni kell a *PASCAL* és az *ASSEMBLY* nyelv kisebb trükkjeit, például a dinamikus memóriakezelést vagy az Assembler betétek felépítését. A példákat kivétel nélkül *Borland Pascal 7.0*-ban írtuk. A fordító viselkedése a beállított fordítási direktíváktól függően más-más lehet, ezért itt röviden ismertetjük azokat a beállításokat, amelyek a programok írásakor érvényesek voltak. Javasoljuk, hogy az Olvasó az alábbi beállítások szerint — amelyek

egyébként általánosnak mondhatók — állítsa át saját fordítóját az *Options/Compiler* menüpontban a munka elkezdésekor.



1. ábra. A fordító beállításai

Néhány szó a könyvben használt betűtípusokról. A könyvben a programok és azok magyarázatai ugyan nem különülnek el szervesen, mégis könnyen megkülönböztethetők egymástól. A magyarázatokat mindig normál (times betűtípussal), míg a programokat mindenhol `program` stílussal (`courier` betűtípussal) írjuk majd. A programokra a Borland Pascal dokumentációjában található szabályok érvényesek: a megjegyzések dőlt betűvel, a fenntartott szavak vastagított betűvel, míg az egyéb szavak és szimbólumok normál betűvel szerepelnek, valahogy így:

```
{ megjegyzés }
procedure
writeln('Ez itt egy szöveg');
```

A könyvben előfordulnak olyan angol szavak, amelyeknek a lefordítását — magyarítását — nem tartottuk célszerűnek, mert a szakirodalomban szintén angolul találhatók meg. Az ilyen szavakat *dőlt (italic)* stílussal írjuk. Ugyancsak dőlt betűkkel jelöljük az egyes változókat, típusokat, eljárásneveket. Ha egy-egy szó egy mondatban **félkövér** stílusú, akkor az valamilyen fontosabb alapfogalom vagy megjegyzésre érdemes név. Előfordulhat, hogy a hangkártyák hardverének valamely magyarázatánál az alábbi jelölés található: (???). A kérdőjelekkel azt jelezzük, hogy az adott fogalom vagy funkció nem pontosan meghatározott, az irodalom nem definiálja korrektül, vagy kompatibilitási gondok léphetnek fel bizonyos hangkártyák esetén.

Az egyes szövegrészekben néhol előfordulhatnak bizonyos — az Olvasó számára talán — ismeretlen jelek, például: :–) , :–(. Ezek a jelek teljesen világosan és tömören kifejezik a szerző egyéni véleményét vagy hangulatát a szöveg írásakor. A jeleket az *Internet* használóitól (e könyv szerzője is közéjük tartozik) kölcsönöztük. Az értelmezésükről csak annyit, hogy a *UNIX*-alapú rendszerekben a karakteres megjelenítők és az ilyen formájú kommunikálás általános. A könyv 90 fokos jobbra fordításával a jelek világossá válnak. Például a :–) egy mosolygó arcot mutat, jelentése pedig az, hogy az adott szöveg valamiért mosolyt csal az ember arcára. A többi jel megfejtését az Olvasóra bízzuk.

A könyvben a sokféle szöveg összefolyhat: például programlista, regiszterleírás vagy fizikai összefüggések. Némelyik ezek közül fontosabb, mások nem annyira érdekesek. A fontosabb részeket egységesen az egész könyvben egy jellemző ábrával (ikonnal) jelezzük, így az esetlegesen összefolyó szövegrészek is jól elkülöníthetők lesznek:

hangkártyaregiszter

fontosabb matematikai vagy fizikai összefüggés

program (unit) forráslistája

fontos hangkártya I/O cím

zenefájlkomponens

A könyvben esetlegesen előforduló hibákkal vagy pontatlanságokkal kapcsolatban a könyv szerzője bármilyen kritikát és észrevételt szívesen fogad a következő *e-mail* címen:

sbook@malacka.iit.uni-miskolc.hu

Végezetül a szerző szeretne köszönetet mondani azoknak, akik e könyv megírásában segítségére voltak:

Köszönet illeti *Loránt Istvánt* irgalmat nem ismerő kritikai megjegyzéseiért és a javításra vonatkozó ötleteiért. (Ezek segítségével szakmai szempontból többszörösen át lehetett vizsgálni és újraértékelni a könyvet.)

Köszönet *Drótos Dánielnek* az *Internet* és a *UNIX* használata terén nyújtott segítségért.

Nagy-nagy köszönet *Árvai Lászlónak* az igen sok és magas színvonalú szakmai segítségéért, amely nélkül ez a könyv el sem készülhetett volna.

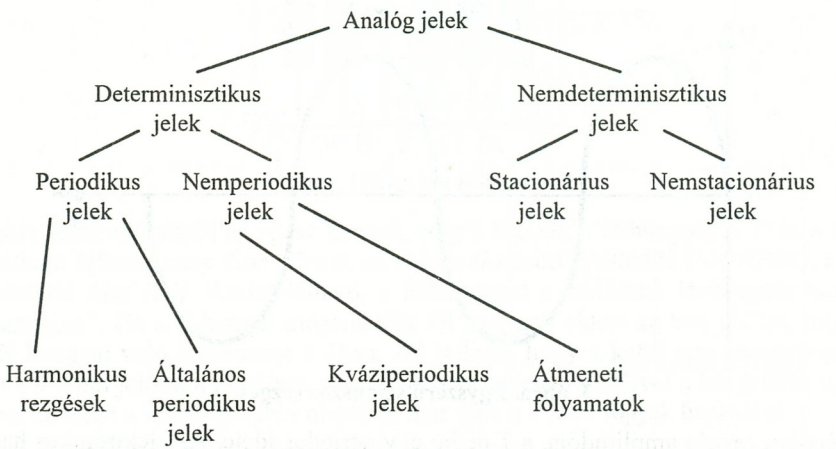
Végül, de nem utolsósorban köszönet illeti *Mihály Győzőt* és az *Elender Computer Kft.*-t, hogy a munkához szükséges hardvert a szerző rendelkezésére bocsátotta.

1. Alapfogalmak

Mielőtt rátérünk a hangkártyák konkrét programozására, ismerkedjünk meg néhány, a hangokkal, a zenével és a hangfeldolgozással kapcsolatos alapfogalommal. Arra nem lesz mód, hogy mindennel a lehető legrészletesebben foglalkozzunk, ezért egy kicsit nagyobb lépésekkel fogunk haladni, nem veszünk majd el az apró részletekben.

1.1. Analóg jelek osztályozása

Analóg jeleknek nevezzük azokat a jeleket, amelyek mind értéktartományukban (amplitúdójukban), mind az időtartományban folyamatosan változnak. Az analóg szó ógörög eredetű, jelentése: „megfelelő". Ezzel tehát azt fejezzük ki, hogy az analóg jel folyamatosan és megfelelően, tehát analóg módon képez le egy fizikai folyamatot. Az analóg jeleket fizikai tulajdonságaik és matematikai kezelhetőségük szerint szoktuk osztályozni. Az alábbi ábra az analóg jelek klasszikus osztályozását mutatja:



2. ábra. Az analóg jelek osztályozása

Egy jelet akkor nevezhetünk determinisztikusnak, ha értékének változását előre látjuk, azaz tetszőleges matematikai formulával bármely időpillanatban ki tudjuk számítani a jel értékét. A nemdeterminisztikus jelekről ez nem mondható el. Ezeket csak statisztikailag tudjuk kezelni a valószínűségszámítás eszközeivel.
A determinisztikus jelek a hangkártyák programozása szempontjából jelentősebbek számunkra. A harmonikus rezgések és az átmeneti folyamatok — ahogyan az később majd látható — programból vezérelhetők, így a hangkeltés főleg az ilyen jelek megfelelő beállításával valósul meg. Természetesen ez nem jelenti azt, hogy a nemdeterminisztikus jelek nem fontosak, csupán számunkra most kisebb jelentőségűek.

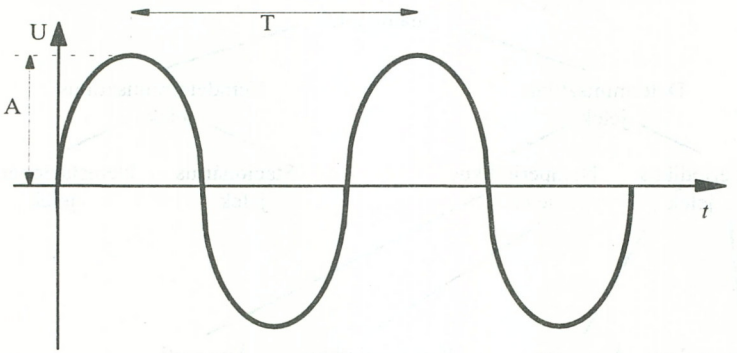
1.2. A hang

Az emberi fül által hallható hang nem más, mint valamely rezgő test által létrehozott, a levegőben terjedő nyomáshullám. A levegő nem az egyedüli közeg, ahol a hang terjedni képes, de a mi szempontunkból gyakorlatilag most csak ez számít. A rezgéseknek van néhány igen fontos fizikai jellemzője.

Frekvencia: az egy másodperc alatti rezgések száma. Frekvenciáról csak akkor illik beszélni, ha ún. periodikus (harmonikus) rezgésről van szó. A legegyszerűbb harmonikus rezgés a szinuszos rezgés. Az emberi hallószervek a 20 ... 20 000 Hz frekvencia-intervallumba eső rezgéseket érzékelik hangként. Az értékek természetesen egyénenként változhatnak, ezek csak statisztikailag kimutatott értékhatárok.

Amplitúdó: a rezgések legnagyobb kitérési értéke. Ez a hangerőt jelenti. Minél nagyobb egy hang amplitúdója, annál hangosabbnak halljuk.

Az alábbi ábrán egy egyszerű szinuszos rezgés két periódusa látható:



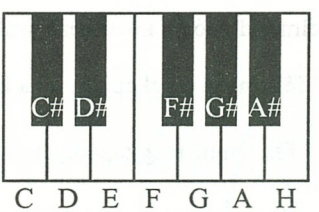
3. ábra. Egyszerű szinuszos rezgés

Az A érték a rezgés amplitúdója, a T pedig egy periódus ideje. Az elektronikus hangkeltés esetében a hang egy váltakozó feszültségnek felel meg, amelyet csak a hangszóró alakít át levegőben terjedő nyomáshullámokká. Ezért használtuk a függőleges tengelyen az U jelölést, amely a fizikában a feszültség jele. A vízszintes tengelyen az eltelt időt ábrázoljuk. Az ábra alapján érthetővé válik a **periódus** fogalma is, azaz a jelnek az az egy teljes szakasza, amely ismétlődik. A periódusidő, vagyis a T a két ismétlődés — két hullámcsúcs — között eltelt idő. A frekvencia és a periódusidő között az alábbi összefüggés áll fenn:

$$T = \frac{1}{f}$$

Azaz a periódusidő a frekvencia reciproka és fordítva. Például a hálózati feszültség névleges frekvenciája 50 Hz, így periódusideje 20 ms.

Nézzük tovább a hangokat! A zenében általában nem a hangok frekvenciájáról beszélünk, hanem a hangok magasságáról. Minél nagyobb egy hang frekvenciája, annál magasabbnak érzékeljük a hangot. A hangmagasság viszonylagos (relatív dolog). Ha csak egyetlen hangot hallunk, akkor nemigen tudjuk megmondani, hogy az melyik hangskála melyik hangja volt. (Persze vannak abszolút hallású emberek is, nekik ez nem okozna gondot.) A zenei hangok esetében a hangmagasságot a frekvencia logaritmusával arányosan érzékeljük. Tehát a hang magassága a frekvencia relatív megváltozásával arányosan változik. Mindezekért az ember kialakította a számára legmegfelelőbb hangskálát és hozzá a megfelelő hangtávolságot, az **oktávot**. Az oktávok egymáshoz való frekvenciaviszonya 1/2, és mindegyik oktávban tizenkét hang található. A hangok között vannak *egész* és *félhangok*. Minden hang a mellette lévőtől félhangnyira található. Nézzük meg, hogyan helyezkedik el egy oktáv a zongorán:



4. ábra. Hangok a zongorán

A zongora fehér billentyűi az egész hangok, míg a feketék a félhangok. A *C* és a *D* között található félhang neve *Cisz (Desz)*, az *F* és a *G* között találhatóé *Fisz (Gesz)*, a *G* és az *A* közöttié *Aisz (Bé)*. Amint látható, a félhangokat a mellettük lévő egész hangból „származtatjuk". Ha a *C* hangot megemeljük fél hanggal, akkor az lesz a *Cisz*, míg a *D* hang fél hanggal való leeresztése a *Desz*. Jól látható, hogy a kettő egy és ugyanaz. Az emelést a # jellel jelölik a kottában, míg a süllyesztés jele a *b*. Mivel a két jelölés ugyanazt fejezi ki, ezért a könyv további részében már csak a # jelet fogjuk használni.

Egy oktávon belül tizenkét, egymástól félhangnyira lévő hang található, és két oktáv frekvenciaviszonya 1/2; ezért két szomszédos hang frekvenciaviszonya állandó:

$$K = \sqrt[12]{2} = 1{,}059463$$

Ezt a hangskálát hívjuk **egyenletesen temperált** hangskálának. Az egyenletesen temperált skálán tehát minden egyes hang egy másikból tetszőlegesen meghatározható. Hogy a skála hangjai mindenhol azonosak legyenek, ezért kell lennie egy alaphangnak, amelyből bármelyik másik hangot elő tudjuk állítani. Ez a hang az ún. „kamara hang", vagyis a

normál A hang, melynek frekvenciája 440 Hz. Ezt az értéket szabványban is rögzítették. Nézzük meg a **normál A** oktávját és két szomszédos oktávjának frekvenciaértékeit:

Hangjegy	c oktáv	c^1 oktáv	c^2 oktáv
C	130.813	261.626	523.251
Cisz, Desz	138.591	277.183	554.365
D	146.832	293.665	587.330
Disz, Esz	155.563	311.127	622.254
E	164.814	329.628	659.628
F	174.614	349.228	698.456
Fisz, Gesz	184.997	369.994	739.989
G	195.998	391.995	783.991
Gisz, Asz	207.652	415.305	830.609
A	220.000	**440.000**	880.000
Aisz, Bé	233.082	466.164	932.328
H	246.942	493.883	987.767

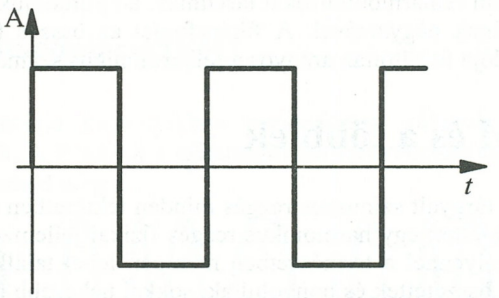
A fentiek alapján általánosan elmondhatók a következő szabályok:

– Egy hang felett lévő félhangot megkapjuk, ha a hang frekvenciáját megszorozzuk $\sqrt[12]{2}$-vel.
– Egy hang alatt lévő félhangot megkapjuk, ha a hang frekvenciáját elosztjuk $\sqrt[12]{2}$-vel.
– Egy hang egy oktávval magasabb megfelelőjét megkapjuk, ha a hang frekvenciáját megszorozzuk 2-vel.
– Egy hang egy oktávval alacsonyabb megfelelőjét megkapjuk, ha a hang frekvenciáját elosztjuk 2-vel.

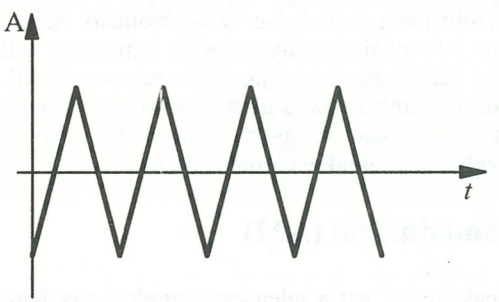
1.2.1. Hullámforma

A szinuszos rezgésen kívül van még néhány alapvető hullámalak, amelyek az elektronikus hangkeltésben fontosak. A természetben több hullám összegét halljuk. Ha egy hullám frekvenciájának egész számú többszöröse is megszólal, akkor azt felharmonikusnak hívjuk. Minél nagyobb számú többszöröse a felharmonikus frekvenciája az alaphullám — alapharmonikus — frekvenciájának, annál nagyobb számú felharmonikusról beszélhetünk. Az a felharmonikus a 2. felharmonikus, amelyiknek frekvenciája kétszerese az alapharmonikus frekvenciájának, az a 3. felharmonikus, amelynek frekvenciája háromszorosa az alapharmonikusnak és így tovább. Egy francia matematikus, **Jean Baptiste Joseph de Fourier** 1922-ben kimutatta, hogy bármilyen periodikus függvényt fel lehet bontani különböző amplitúdójú és fázisú periodikus függvényekre. Ez azt jelenti, hogy bármilyen periodikus jelfüggvény előállítható különböző szinuszhullámok súlyozott összegeként. Nézzük most meg a leggyakrabban használt hullámformákat és az azokat

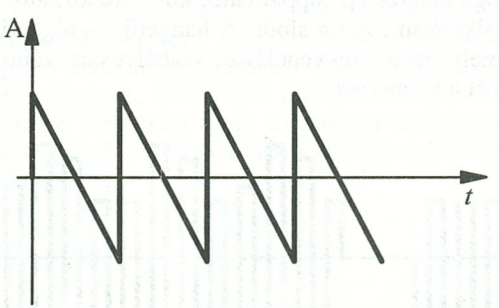
alkotó felharmonikusokat! Ezek a hullámok elektronikus úton könnyen előállíthatók, és így lehetőség nyílik olyan hangok keltésére is, amelyek a természetben nem találhatók meg.



5. ábra. Szimmetrikus négyszögjel



6. ábra. Háromszögjel



7. ábra. Fűrészfogjel

A különböző hangszerek különböző hullámalakokat keltenek. A szinuszhoz igen hasonló, egyszerű hangja van a fuvolának. A klarinét hangja a négyszögjelhez hasonlít.

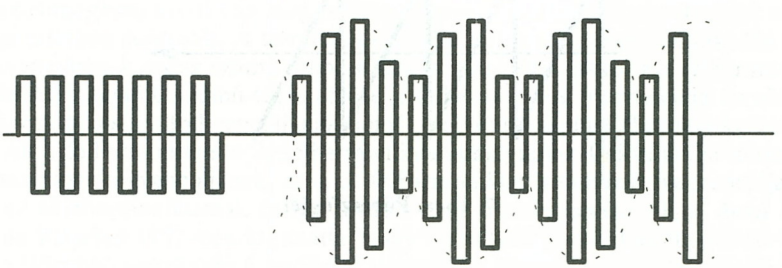
A hegedű a fűrészfogjelhez hasonló hangot ad ki. Minden hullámforma más-más felharmonikusokból tevődik össze. A négyszögjel csak páratlan felharmonikusokat tartalmaz, ezek amplitúdója fordítottan arányos a felharmonikus sorszámával. A háromszögjel szintén páratlan felharmonikusokat tartalmaz, amplitúdójuk fordítottan arányos a felharmonikus számának négyzetével. A fűrészfogjel az összes felharmonikust tartalmazza, ezek amplitúdója fordítottan arányos a felharmonikus számával.

1.3. AM, FM és a többiek

Az előző szakaszban tárgyalt szinuszos rezgés minden tekintetben ideális: egyszerű, érthető, ragyogóan szemlélteti egy harmonikus rezgés fizikai jellemzőit. Egy kis baj azonban mégis van vele: ilyennel a természetben nemigen lehet találkozni. Környező világunk hangjai és zajai összetettek és bonyolultak, sokkal nehezebb őket kezelni, mint egy egyszerű szinuszos rezgést. A hangszerek sem adnak ki ilyen egyszerű hangot. A gitár húrja lassan lecseng, a furulya hangereje tetszés szerint változtatható, a zongora egyik pedáljával az éppen szóló hang lecsengése befolyásolható. Szinte minden hangszernek van valamilyen sajátos jellemzője, amelyet fizikai ismeretek nélkül is lehet érzékelni. Vegyük csak példaként két hangszer hangjának összehasonlítását! Hiába szól ugyanaz a hang, hiába ugyanakkora a hangerejük, a gitár és az orgona hangja mégis megkülönböztethető. Vegyük tehát sorra azokat az egyéb jellemzőket, amelyek — a fenti fizikai jellemzőkön kívül — meghatározhatnak valamely hangot.

1.3.1. Amplitúdómoduláció (AM)

Amplitúdómodulációnak hívjuk azt a jelenséget, amikor egy hang amplitúdója valamilyen periódussal kismértékben, de szabályosan változik. A legegyszerűbb amplitúdómoduláció egy hétköznapi síppal bármikor létrehozható, ha valaki szabályosan hangosabban, majd halkabban fújja a sípot. A hangerő — függetlenül a hang alapfrekvenciájától — valamely más frekvenciával szabályosan változik. Zenész nyelven **tremolónak** nevezik ezt a jelenséget.

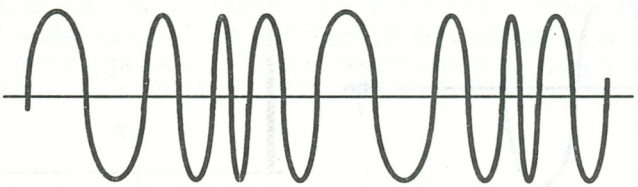


8. ábra. Amplitúdómoduláció

A *8. ábra* bal oldalán egy egyszerű négyszögjel látható, míg a jobb oldalon egy olyan négyszögjel, amelyet amplitúdómoduláltunk egy szinuszjellel. Így a négyszögjel frekvenciája az eredeti marad, de az amplitúdója a szinuszos jel szerint változik.

1.3.2. Frekvenciamoduláció (FM)

Ha egy hang frekvenciája kismértékben periodikusan változik, akkor frekvenciamodulációról beszélünk. A *9. ábrán* a szinuszhullám frekvenciája szakaszosan — periodikusan — lecsökken, majd megnő:



9. ábra. Frekvenciamoduláció

A zenében vibrációnak nevezik a frekvenciamodulációt. Ha egy hang ilyen módon van modulálva, akkor azt mondjuk, hogy a hang **vibrátóval** szól. Azok az eszközök, amelyek ezt elő tudják állítani, a vibrátorok. A vibráció tipikus esete a hápogtató kar használata a gitáron. A kar a húr feszességét változtatja, ezáltal változik a hang frekvenciája.

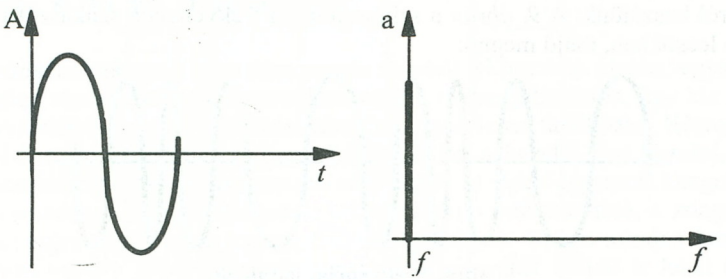
1.3.3. Hangszín

Mint említettük, a természetben általában az összetett hullámok találhatók meg gyakrabban. Az emberi beszéd vagy a hangszerek hangja bonyolult hanghullámok. A hangokat a **hangszín** szerint lehet megkülönböztetni. A hangszín nem pontosan definiált fizikai fogalom, leginkább szubjektív érzékelésen alapszik. Gyakorlatilag a hang összetevőinek és átmeneti (tranziens) jelenségeinek, különböző modulációinak összességét nevezhetjük hangszínnek. A hangszín sok mindentől függ: mechanikai kialakítástól, a hangkeltő eszköztől, az anyag tulajdonságaitól. Mindezen hatások miatt alakul ki minden hangszernek valamilyen jellegzetes hangszíne.

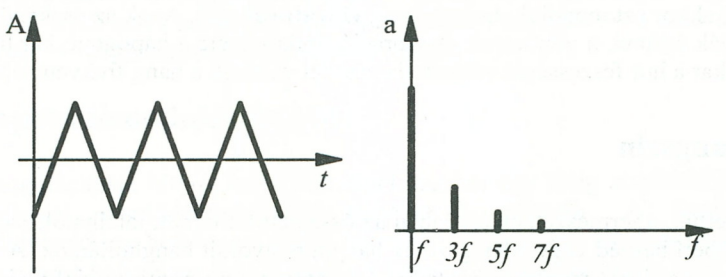
1.3.4. Spektrum

Ahhoz, hogy valamilyen összetett hullámot szintetikus úton elő tudjunk állítani, fontos, hogy meghatározzuk az adott hullámalak összetevőit. A módszert, amellyel ezt el lehet végezni, Fourier tiszteletére **Fourier-analízisnek** nevezték el. Fourier módszerével meghatározhatjuk, hogy egy összetett hullám milyen összetevőkből áll. Az összetevők egyszerű szinuszos hullámok, mindegyik valamilyen jellemző frekvenciával és

amplitúdóval. Ezeket az összetevőket adjuk meg egy olyan koordináta-rendszerben, amelynek függőleges tengelyén (ordinátatengelyén) az amplitúdót, vízszintes tengelyén (abszcisszatengelyén) pedig a frekvenciát ábrázoljuk. Ezt a megadási módot nevezzük az adott hullámalak spektrumának. A spektrum egyértelműen megmutatja, hogy milyen frekvenciájú és amplitúdójú szinuszos jeleket kell összegeznünk ahhoz, hogy megkapjuk az eredeti jelet. Példaképpen nézzük meg a szinuszhullám és a háromszöghullám spektrumát!



10. ábra. A szinuszjel és spektruma



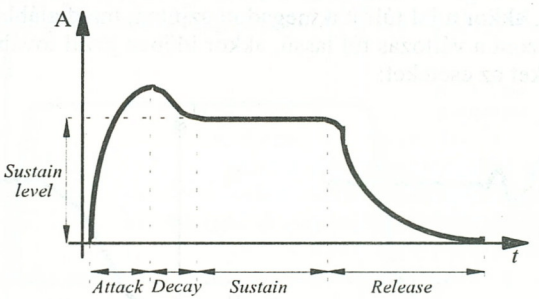
11. ábra. A háromszögjel és spektruma

1.3.5. Burkológörbe, ADSR generátor

Mindeddig nem esett még szó a hang átmeneti jelenségeiről. A hangszerek hangereje általában a megszólalástól az elhallgatásig folyamatosan változik. Legelsőként — például egy gitárhúr megpendítésekor — a hangerő felfut egy bizonyos maximális szintre. (Persze ez függ attól, hogy mekkora a megpendítés ereje, azaz mekkora mozgási energiát közlünk a húrral.) Ezt az időt nevezzük felfutási időnek, angolul ***attack time***-nak. Ez után a hang ereje valamilyen közbenső szintre esik vissza. Ezt hívjuk lecsengési időnek (***decay time***). Ezen a szinten a hang tovább szól, egy bizonyos ideig kitartva marad. Ezt az időt nevezzük kitartási időnek (***sustain time***), a szintet pedig, amelyen a hang ilyenkor

szól, kitartási szintnek (***sustain level***). A kitartási szintet általában a legnagyobb amplitúdóhoz viszonyítva, százalékosan szoktuk megadni. Egy 80%-os kitartási szint tehát azt jelenti, hogy a *decay* szakasz alatt az amplitúdó a maximális érték 80%-ára esik vissza. Végezetül a hang fokozatosan elhal — a húr elveszti mozgási energiáját —, azaz lecseng (***release time***). Az egész folyamat alatt tehát az amplitúdó fokozatosan változott. Ha ezt az amplitúdóváltozást az idő függvényében ábrázoljuk, akkor megkapjuk az adott hang **burkológörbéjét**.

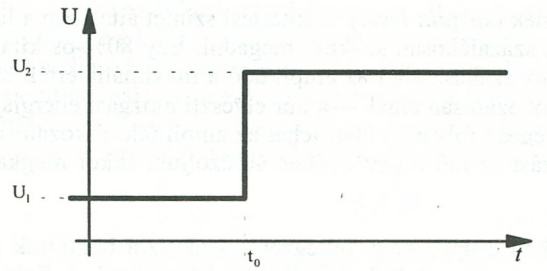
Bár ebben az esetben csupán négy jellegzetes szakaszra bontottuk a burkológörbét, el kell mondanunk, hogy nem csak ilyen felbontás képzelhető el. Sokkal több önálló szakaszt is meg lehet adni, ezzel a hang jóval közelebb lesz a valósághoz. Egy trombita hangja például sokkal bonyolultabb burkológörbét eredményezhet, hiszen a levegő áramlását az ember szabja meg. Az ilyen hangszerek hangját éppen ezért nehezebb előállítani egy négyszakaszos burkológörbével, mint mondjuk a zongoráét.



12. ábra. A burkológörbe jellegzetes szakaszai

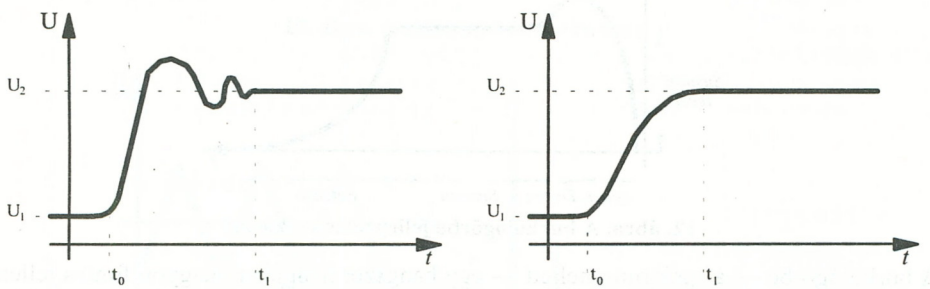
A burkológörbe — a spektrum mellett — egy hangszer hangjának nagyon fontos jellemzője. Ha egy hangszer hangját elektronikus úton kívánjuk előállítani, akkor a megfelelő hullámalak mellett gondoskodnunk kell a megfelelő burkológörbéről is. A burkológörbéket elektronikus úton az ún. burkológörbe-generátorokkal állítjuk elő. Ezeket a generátorokat — az egyes szakaszok angol nevei alapján — szokás **ADSR** generátoroknak is hívni. A későbbiekben ismertetésre kerülő hangkártyáknak van ADSR generátoruk. A programozható ADSR generátoroknál nagyon kell figyelni arra, hogy a *sustain* szakasz paramétere általában nem időtartamot, hanem jelszintet jelent. A kitartás idejét más módon szokták megadni.

Mielőtt továbbmennénk, tennünk kell egy kis kitérőt, hogy tisztázzuk a „pillanatszerű" fogalmát. Az elektronikában az egyik leggyakoribb jel az ún. **egységugrásjel**, amely tulajdonképpen egy adott jelszintről egy másik jelszintre történő átlépést jelent. Az ideális egységugrásjel a következő:



13. ábra. Az egységugrás függvény

A *13. ábra* szerint a következő történik: az U_1 szintről a t_0 időpillanatban az U_2 szintre ugrik fel a jel. Ez azonban ideális eset. A valóságban a jelszintek közötti áttérés nem történhet meg valamekkora idő nélkül. Ha a változás túl gyors — azaz túl nagy energiával megy végbe —, akkor a jel túlfut a megadott szinten, majd alább esik, és így fokozatosan áll be. Ha viszont a változás túl lassú, akkor időben jóval tovább tart. Az alábbi két ábra bemutatja ezeket az eseteket:

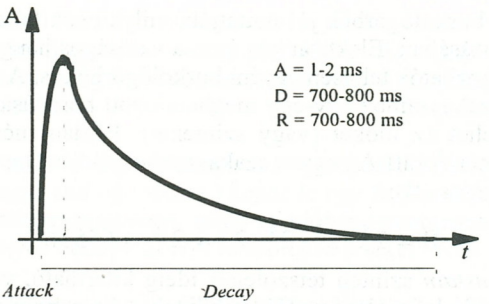


14. ábra. Tranziens (átmeneti) jelenségek

Ezekben az esetekben — azaz a valóságban — tehát az átugrás valamekkora időt vesz igénybe.

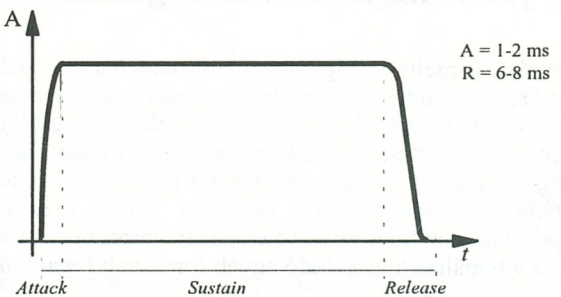
Nézzük meg példaképpen néhány hangszer jellegzetes burkológörbéjét! Az ütőhangszereknél — ilyen például a cimbalom és a dob — a burkológörbe közbenső két szakasza szinte teljesen elmarad. A hang pillanatszerűen felfut, majd csaknem ugyanolyan gyorsan le is cseng.

A cimbalom burkológörbéjénél látható, ahogy a hang az 1-2 ms-os felfutás után egyből elkezd lecsengeni, ez 700-800 ms-ig tart. A lecsengés itt tulajdonképpen megegyezik az elengedéssel, hiszen a jelnek nincs kitartási szintje. Ha a burkológörbének nincs kitartása — pontosabban szólva a kitartási szint és a kitartási idő is nulla —, akkor a jel már a *decay* szakaszban leesik a nulla szintre, így a *release* ciklus érdektelen.



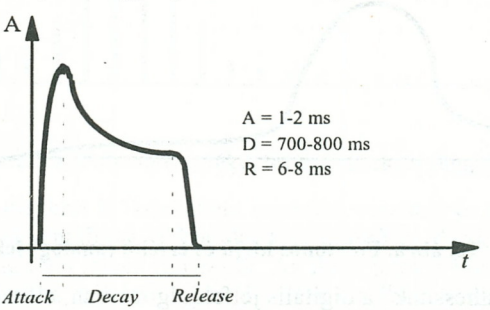
15. ábra. A cimbalom burkológörbéje

Az orgona burkológörbéje talán az egyik legegyszerűbb. A hang a billentyű lenyomásakor gyorsan (1-2 ms alatt) felfut a legnagyobb amplitúdóra, és ott is marad. Amíg az orgona billentyűje le van nyomva, a hang mindaddig ezen a szinten kitartva marad. A billentyű felengedésekor a hang azonnal (6-8 ms alatt) elhal.



16. ábra. Az orgona burkológörbéje

A zongoránál szintén igen gyors a jel felfutása, de a hang a cimbalomhoz hasonlóan azonnal elkezd lecsengeni. Ha a zongora billentyűjét felengedjük, mielőtt a hang teljesen lecsengett volna, akkor az amplitúdó azonnal leesik nullára. Ha a billentyűt mindvégig nyomva tartjuk, akkor a hang a *decay* ciklus alatt fokozatosan elhal, és ekkor a *release* ciklus érdektelenné válik (ez az egyik pedállal szabályozható).



17. ábra. A zongora burkológörbéje

A fentebb ismertetett burkológörbék jól mutatják, milyen sok lehetőség rejlik az ADSR generátorok programozásában. Elektronikus úton a valóságos hangszerek jól szimulálhatók, ugyanakkor létrehozhatók teljesen egyéni burkológörbék is. A legtöbb hangkártyán a burkológörbe egyes szakaszaihoz egy-egy meghatározott hosszúságú bitsorozat tartozik, amellyel definiálni lehet az időket (vagy szinteket). Például nézzük meg az ADLIB hangkártya ADSR generátorát! Az egyes szakaszokhoz 4-4 bit tartozik, azaz a létrehozható burkológörbék száma (**B**):

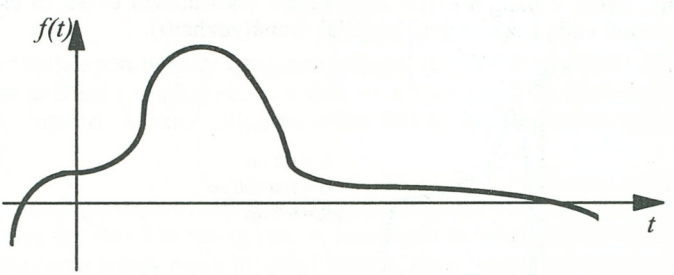
$$\mathbf{B} = 2^4 \cdot 2^4 \cdot 2^4 \cdot 2^4 = 2^{16} = 65536$$

Ezenkívül a hang a *sustain* szinten tetszőleges ideig kitartható, valamint lehetőség van különböző hullámalakok definiálására. Jól látszik tehát, hogy nagyon sok a lehetőség!

1.4. Analóg jelek digitális feldolgozása

Az analóg hangszintetizálás mellett a digitális úton történő hangrögzítés és hangelőállítás egyre nagyobb teret kapott, ahogyan a digitális eszközök egyre gyorsabbak és pontosabbak lettek. Az átalakítók (konverterek), amelyek az analóg jeleket digitálissá alakították és fordítva, egyre inkább tökéletesedtek, a tárolóeszközök kapacitása pedig megnőtt, így lehetővé vált a hangok digitális tárolása. A legjobb példa erre az audio-CD (Compact Disk), amely digitális formában tárolja a hangokat. Ebben a szakaszban a digitális jelfeldolgozás alapjait ismertetjük: a mintavételezés módszereit és szabályait, az átalakítók működési elvét és a témához kapcsolódó egyéb fontosabb fogalmakat.

1.4.1. Mintavételezés és kvantálás



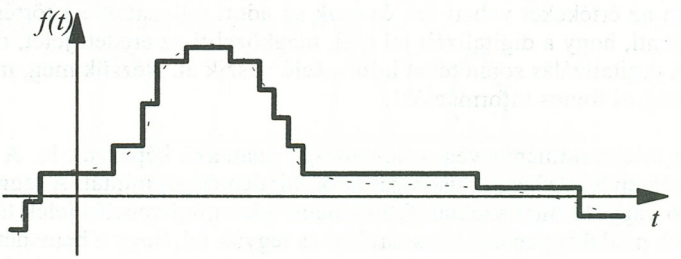
18. ábra. Folytonos idejű és értékű (analóg) jel

Ahhoz, hogy „elmerülhessünk" a digitális jelfeldolgozásban, először is a jelek típusait és struktúrájukat kell megismernünk. Az elektronikában (szűkebb értelemben a

híradástechnikában) általában úgy értelmezzük a jeleket, mint valamilyen időtől függő folyamat — azaz időfüggvény — amplitúdóját. Az amplitúdót most tekintsük feszültségnek, hiszen a hangok általában feszültség formájában terjednek az analóg elektromos rendszerekben.

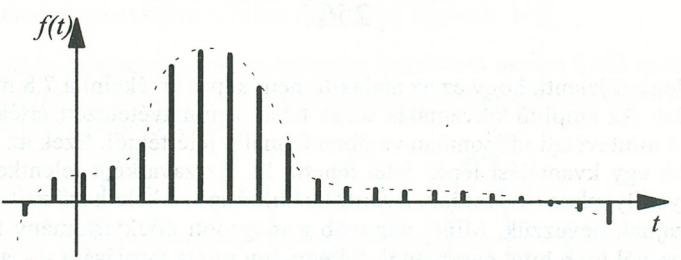
Az analóg jelek időben és értéktartományban folytonosak. Ez azt jelenti, hogy a jel megfelelően (ugyanúgy, azaz analóg módon) képez le egy folytonosan zajló fizikai folyamatot. A jel mind az értéktartományban, mind az időben tetszőleges értéket vehet fel, pontosabban szólva bizonyos határok között tetszőleges értéket vehet fel!

Léteznek azonban másfajta jelek is. Előfordulhat, hogy a jel időben folytonos, de csak meghatározott — pontosan definiált, diszkrét — értékeket vehet fel. Ezeket a jeleket időben folytonos, diszkrét értékű jeleknek nevezzük. Az előbbi analóg jelet alakítsuk át ilyen jelfüggvénnyé!



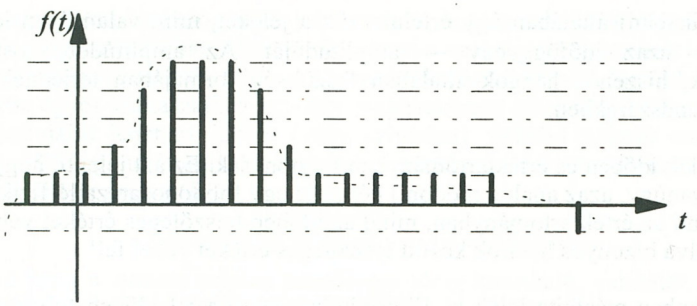
19. ábra. Időben folytonos, diszkrét értékű függvény

Egy másik lehetséges eset, hogy a jel folytonos értékű, ugyanakkor időben csak meghatározott pillanatokban jelentkezik, azaz időben diszkrét. A fenti függvény ilyen formában a következőképpen festene:



20. ábra. Folytonos értékű, időben diszkrét függvény

Ha egy analóg jelből diszkrét időközönként mintákat veszünk, és a jel értékét is diszkrét helyeken definiáljuk (kvantáljuk), akkor diszkrét értékű és diszkrét idejű jelet állítunk elő. Ezt a jelet már digitális jelnek nevezzük. Az ilyen jeleket könnyű tárolni a számítógép memóriájában, hiszen minden egyes diszkrét értékhez hozzá lehet rendelni valamilyen bináris kódot.



21. ábra. Diszkrét értékű és diszkrét idejű (digitális) jel

A *21. ábrán* jól láthatók azok a jelfokozatok (lépcsők), amelyeket definiáltunk, mint a függvény lehetséges értékeit. Ha az előbbi analóg jelet digitalizáljuk, akkor a digitalizált jel csak ezeket az értékeket veheti fel, és csak az adott pillanatokban történhet jelváltozás. Ez azt jelenti, hogy a digitalizált jel csak megközelíti az eredeti jelet, de nem írja le tökéletesen. A digitalizálás során tehát információ veszik el. Nézzük meg, miért vesznek, miért veszhetnek el fontos információk!

1. Az értéktartományt véges hosszúságú adatokká képezzük le. A memóriában valamilyen hosszúságú bitsorozat tárol minden egyes mintát. A leggyakrabban 8, 12, 16 vagy 24 bitet szoktak felhasználni a hangfrekvenciás jelek tárolására. Tekintsük példaképpen a 8-bites tárolást és tegyük fel, hogy a bemeneti analóg jel a –1 ... +1 V tartományba esik! Egy 8-bites átalakítás legnagyobb felbontása 256 lépcső (azaz a 2 nyolcadik hatványa), míg az értéktartomány 2 V. Ekkor tehát az egy lépcsőhöz tartozó feszültségérték:

$$\Delta U = \frac{2}{256} = 0.0078125$$

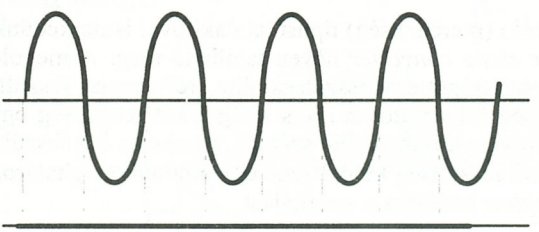
Mindez azt jelenti, hogy ez az átalakító nem képes érzékelni a 7.8 mV-nál kisebb eltérést. Az amplitúdókvantálás során tehát a mintavételezett értékek egy része eltér a mintavételi időpontban valóban fennálló jelértéktől. Ezek az eltérések legfeljebb egy kvantálási lépés felét tehetik ki. Ez zavarként jelentkezik. A hibát, amelyet ilyenkor elkövetünk, **kvantálási hibának**, a keletkező zajt pedig kvantálási zajnak nevezzük. Minél nagyobb a megadott értéktartomány felbontása — azaz minél több bitet használunk fel egy-egy minta tárolására —, annál kisebb a kvantálásból eredő hiba, de eltüntetni sajnos nem lehet!

2. A mintavételezést csak a kijelölt diszkrét időpillanatokban végezzük el. Ha két mintavétel közötti idő alatt gyors változás zajlik le, és ezt nem tudjuk érzékelni, akkor jelentős rész veszik el a jelből. A mintavétel gyakorisága és a mintavételezett jel jelváltozási sebessége között — azaz a frekvenciájuk között —

összefüggés van. Egyszerűen azt mondhatjuk, hogy a mintavételnek olyan gyorsnak kell lennie, hogy észlelni tudjuk a jelben előforduló leggyorsabb változást is. Definíciószerűen **Shannon** fogalmazta meg a mintavételi tételt:
A mintavételi frekvenciának a jelben előforduló legnagyobb frekvencia kétszeresénél nagyobbnak kell lennie ahhoz, hogy a jel által tartalmazott információ teljes mértékben megmaradjon, azaz a digitális mintákból az eredeti jel visszaállítható legyen.

Ha egy mintavételi rendszer megsérti a **mintavételezési tételt**, akkor nagymértékben torzulhat, esetleg a minták alapján visszaállított jel teljesen más lehet, mint az eredetileg mintavételezett jel.

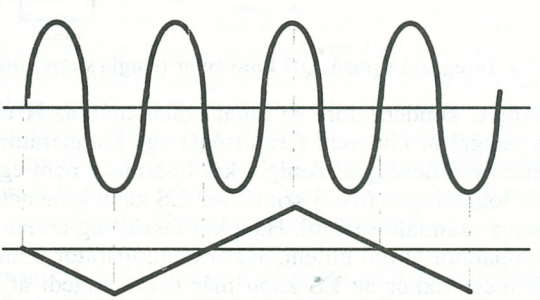
Hogy a mintavételi tétel megsértése milyen kellemetlenségeket okozhat, arra nézzünk két egyszerű példát! Elsőként képzeljünk el egy olyan rendszert, amelyben a mintavételezési frekvencia pontosan kétszerese a mintavételezett jel frekvenciájának. Legyen a digitalizálandó jel egy egyszerű szinuszos rezgés!



22. ábra. Hibás mintavételezés

A *22. ábrán* jól látható, hogy a mintavételezési időpontok pontosan az eredeti függvény nullátmeneti pontjaira esnek, és mivel a mintavételi frekvencia pontosan kétszerese a jel frekvenciájának, ezért ez így is marad mindvégig. Bár az eredeti jel szinuszos volt, a helytelen mintavétel miatt teljesen hibás függvényt kaptunk. ;–{(

A következő példa az alacsony mintavételezési frekvencia okozta hibát mutatja be. Képzeljünk el egy gyorsan változó jelet, amelyet a jel frekvenciájának kétszeresénél alacsonyabb frekvenciával mintavételezünk.



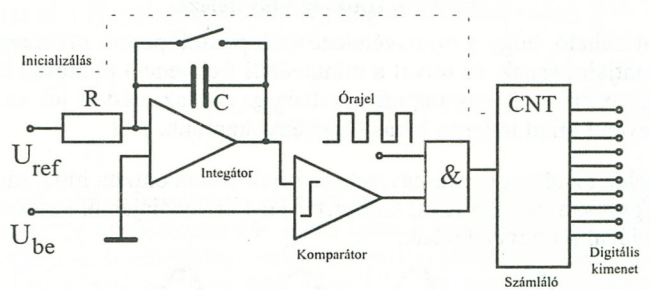
23. ábra. Hibás mintavételezés

A *23. ábra* jól szemlélteti, hogy miképpen veszett el a fontos információ, és hogyan jött létre egy teljesen hibás jelalak, amelynek nem sok köze van az eredeti jelhez. Ez olyan spektrális komponensek megjelenését okozza, amelyek nem szerepelnek az eredeti jel spektrumában. Ezt **aliasing** jelenségnek nevezzük.

1.4.2. Analóg-digitális átalakítás (ADC)

Az analóg jelek digitálissá alakításának többféle technikai megoldása is van. Azt, hogy egy adott rendszerben milyen típusú A/D konvertert alkalmaznak, mindig sajátos szempontok határozzák meg. Ilyen szempont lehet például az átalakítás pontossága, az átalakítás sebessége vagy az átalakító költsége. A következőkben a sokféle módszer közül csupán két megoldást kívánunk ismertetni, hiszen erről a témáról bőségesen található még információ az irodalomban.

Elsőként egy integráló (meredekség) típusú átalakítóval ismerkedünk meg. Az angol irodalomban ez *single slope converter* néven található meg. A megoldás lényege, hogy a bemeneti analóg feszültséget egy összehasonlító (referencia) feszültséggel vetjük össze. A referenciafeszültség folyamatosan nő, s amíg a két feszültség egyenlő nem lesz, egy digitális számláló folyamatosan felfelé számol. Amikor a két feszültség egyenlővé válik — pontosabban a két feszültség különbsége egy általunk meghatározott minimális intervallumba esik —, akkor leállítjuk a számlálást.



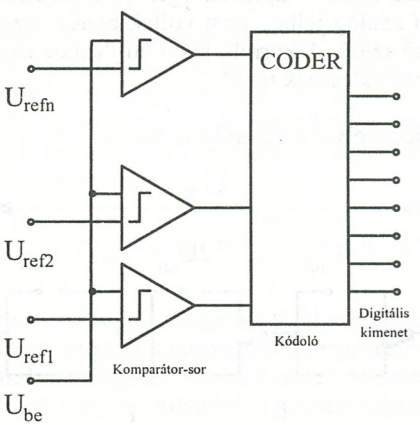
24. ábra. Integráló típusú A/D konverter (single slope converter)

Az integrátor áramkör **C** kondenzátora az inicializálás után az **R** ellenálláson keresztül elkezd töltődni. Az integrátor kimeneti feszültségét egy komparátor segítségével hasonlítjuk össze a bemeneti feszültséggel. Amíg a két feszültség nem egyenlő, addig a komparátor kimeneti jele logikai igaz (*true*) szintű. Az **ÉS** kapu kimenetén ilyenkor az órajel meg tud jelenni, azaz a számláló számol. Ha a két feszültség értéke annyira megközelíti egymást, hogy a komparátor át tud billeni, akkor a komparátor kimenetén logikai hamis (*false*) szint jelenik meg. Ekkor az **ÉS** kapu már nem „engedi át" az órajelet, tehát a számlálás leáll. A konverter tehát időbeli integrálást hajtott végre, a számláló digitális

kimenete arányos az integrálás idejével, az integrálás ideje pedig arányos a bemeneti feszültséggel. Az átalakítást az inicializálás indítja el, ekkor a kapcsoló alaphelyzetbe hozza az integrátort (azaz a kondenzátort kisüti), míg a számlálót lenullázza. Ez a típusú átalakítás tehát valamekkora időt vesz igénybe. Ez az idő az integrátor **R** és **C** tagjaitól függ. A kapcsolás előnye, hogy viszonylag egyszerű, és olcsón előállítható, hátránya viszont, hogy bizonyos frekvencia felett nem használható.

A második megoldás közvetlen feszültség-összehasonlítást végez. A bemeneti jelet közvetlenül több referenciafeszültséggel hasonlítjuk össze, amely referenciafeszültségek értékei csupán egy-egy kvantálási léptékkel térnek el egymástól.

A bemeneti feszültséget az összes komparátor összehasonlítja a saját referencia-feszültségével, és amely referenciafeszültségeknél a bemeneti feszültség nagyobb, ott az adott komparátor átbillen. A komparátorok kimeneteit tehát a bemeneti feszültségre jellemző digitális kódnak tekinthetjük. Ezt a kódot a kódoló áramkör a felbontásnak megfelelő bitszámú bináris számmá alakítja.



25. ábra. A/D átalakítás közvetlen feszültség-összehasonlítással

A módszer hátránya, hogy az átalakítónak **n** bit esetén 2^n-1 darab komparátort kell tartalmaznia, és a komparátoroknak igen pontosaknak kell lenniük. Ez a típusú átalakítás ezért sokkal költségesebb, mint az integráló módszer. Előnye viszont, hogy sokkal gyorsabb, hiszen az átalakítás idejét itt csak a komparátorok billenési ideje és a komparátorok kimenetének kódolási ideje szabja meg. Persze ennél a módszernél is van egy felső sebességhatár, de az integráló módszerhez képest jóval gyorsabban el tudja végezni az átalakítást. Az angol irodalomban — utalva a konverter sebességére — ezt a típusú konvertert villanásszerű, azaz *flash A/D converter*-nek nevezik.

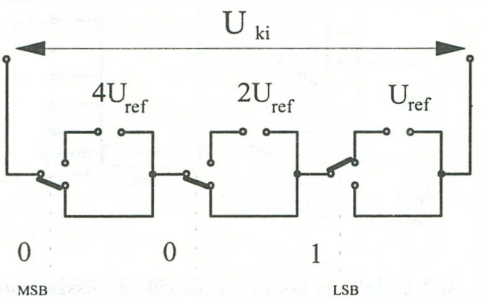
A fenti két megoldáson kívül még számos mód van az A/D átalakításra, ezeket azonban nem ismertetjük, hiszen ennek a fejezetnek csak az a célja, hogy bizonyos

alapismereteket adjon. A hangkártyák programozásához elegendő csupán annyit tudni, hogy a digitális hangcsatornákkal rendelkező kártyákon általában van valamilyen A/D konverter, így lehetőségünk van különböző hangok digitalizálására és tárolására.

1.4.3. Digitális-analóg átalakítás (DAC)

A memóriában tárolt, kódolt adatokat valahogyan ismét hanggá kell alakítani. Ezt — illetve tágabb értelemben bármilyen digitálisan tárolt jel visszaalakítását — végzik el a D/A konverterek. A jel visszaalakításakor a D/A átalakítás azonban csak az egyik lépés. A D/A konverterek kimenetén időben folytonos, de diszkrét értékű jel jelenik meg. Ezt a jelet egy megfelelően megválasztott aluláteresztő szűrőre kell vezetni ahhoz, hogy a jelből előálljon immár az eredeti jel. Miért van erre szükség? A D/A konverter kimenetén egy lépcsőzött — mondhatni szögletes — jel jelenik meg, azaz ebben a jelben gyors, ugrásszerű változások vannak. Ezek az ugrások a jel spektrumában olyan összetevőket keltenek, amelyek az eredeti analóg jelben nem voltak benne. Ezen spektrális összetevőket távolítja el az aluláteresztő szűrő. A korrekt helyreállításhoz nagyon fontos, hogy e szűrő határfrekvenciáját pontosan válasszák meg!

A D/A konverterek működési elvét a *26. ábra* mutatja:



26. ábra. Hárombites D/A konverter

A kapcsolásban bináris súlyozású referenciafeszültségek vannak sorba kötve. A digitális bemenet bitjei egy-egy ilyen referenciafeszültség-forrást kapcsolnak be a hálózatba, ha az adott bit 1 értékű, illetve ezeket kapcsolják ki a hálózatból, ha az adott bit 0 értékű. A bináris kód súlyozásának a feszültségértékek súlyozása természetesen megfelel. (Az LSB jelenti a legalacsonyabb helyi értékű bitet, míg az MSB a legmagasabb helyi értékűt.) Így az adott feszültségek összeadódnak, és a kimeneti feszültség a bekapcsolt források feszültségének összege lesz. Az ezen az elven működő D/A konverterek általában elég gyorsak a hangfrekvenciás felhasználáshoz. A későbbiekben még ismertetni fogunk egy olyan D/A konvertert, amelyet a PC *centronix* (párhuzamos) portjára lehet kötni, és így annak kimenetét felerősítve a digitalizált hangok hallhatók lesznek.

1.4.4. Periodikus függvények Fourier-analízise (sorba fejtés)

Az előzőekben már utaltunk a jelek spektrális összetevőinek meghatározási módszerére, vagyis a Fourier-analízisre. A módszert teljes egészében nem ismertetjük — hiszen ez igen terjedelmes lenne —, csupán az elméleti alapokat és néhány ismertebb jelalak jellegzetes spektrumát mutatjuk be. A Fourier által megfogalmazott tétel az alábbi matematikai formában írható le:

$$f(t) = \sum_{k=0}^{\infty} a_k \cos(\omega_k t) + \sum_{k=1}^{\infty} b_k \sin(\omega_k t)$$

Az *f(t)* periodikus időfüggvény, azaz teljesíti a következő feltételt:

$$f(t) = f(t + nT_0),$$

ahol az *n* egy egész szám, amely a negatív végtelentől a pozitív végtelenig terjed. Köznapi nyelven megfogalmazva ez azt jelenti, hogy a függvényérték a T_0 periódusidő többszöröse után ismétlődik. Némi kellemetlenséget okoz, hogy egy valódi függvény természetesen csak véges időtartamban létezik, és a valóságban negatív idők nem fordulnak elő. Ezt a problémát úgy hidaljuk át, hogy a függvényt eltoljuk az időtengellyel párhuzamosan negatív irányba, és feltételezzük, hogy a függvény, a jel már a 0 előtti időpillanatokban is létezett. A T_0 periódusidő a körfrekvenciával együtt lép fel, ezt a következőképpen lehet definiálni:

$$\omega_k = k\omega_0 = \frac{2\pi}{T_0} k = 2\pi f_0 k$$

Tehát kimondható, hogy egy periodikus függvény f_0 alapfrekvenciájú és T_0 egész számú többszöröseinek megfelelő frekvenciájú harmonikus rezgésre bontható, azaz sorba lehet fejteni. A sorba fejtett diszkrét rezgések fázishelyzetét és amplitúdóját az ún. Fourier-együtthatók, a_0, a_k és b_k adják meg. Az állandók a következőképpen számíthatók ki:

$$a_0 = \frac{1}{T_0} \int_{-T_0/2}^{T_0/2} f(t)dt,$$

$$a_k = \frac{2}{T_0} \int_{-T_0/2}^{T_0/2} f(t)\cos(k\omega_0 t)dt,$$

$$b_k = \frac{2}{T_0} \int_{-T_0/2}^{T_0/2} f(t)\sin(k\omega_0 t)dt.$$

Ezen alapösszefüggések alkalmazásával tehát bármilyen periodikus függvény sorba fejthető, ugyanakkor el kell mondani azt is, hogy ehhez eléggé magas — legalábbis a köznapi középfokú matematikai tudáshoz képest — matematikai képzettség szükséges. Az alábbiakban négy gyakori harmonikus rezgés — a szinusz-, a négyszög-, a háromszög- és a fűrészfogjel — előállításához szükséges sorozatokat ismertetjük. Ezek a sorozatok tulajdonképpen a spektrum egyes összetevőit adják meg.

1. Szinuszos rezgés:

$$f(t) = A\sin(\omega_0 t)$$

2. Szimmetrikus négyszögrezgés:

$$f(t) = \frac{4A}{\pi}\left[\cos(\omega_0 t) - \frac{1}{3}\cos(3\omega_0 t) + \frac{1}{5}\cos(5\omega_0 t) - \frac{1}{7}\cos(7\omega_0 t) + \ldots\right]$$

3. Háromszögrezgés:

$$f(t) = \frac{8A}{\pi^2}\left[\sin(\omega_0 t) - \frac{1}{3^2}\sin(3\omega_0 t) + \frac{1}{5^2}\sin(5\omega_0 t) - \ldots\right]$$

4. Fűrészfogrezgés:

$$f(t) = -\frac{2A}{\pi}\left[\sin(\omega_0 t) + \frac{1}{2}\sin(2\omega_0 t) + \frac{1}{3}\sin(3\omega_0 t) + \ldots.\right]$$

A következőkben nézzünk meg egy egyszerű Pascal programot, amely grafikusan is bemutatja ezeket az összefüggéseket. A fenti képletek alapján el lehet készíteni egy olyan függvényt, amely kiszámítja és összeadja az adott jelalak *n* darab összetevőjét. Ha ezt azután grafikusan ábrázoljuk, akkor jól meg lehet figyelni, hogy az egyes összetevő hullámok hozzáadásakor az eredeti szinuszos jel hogyan alakul mindinkább a kívánt formájú rezgéssé. Elsőként a négyszögjel előállítására nézzünk egy rövid programot! A program eredetileg VGA kártyára 640x480-as felbontásra készült, ezért más grafikus kártya esetében a rajzok elhelyezését esetleg módosítani kell.

```
program fourier;

uses crt,graph;

var gd,gm : integer;
      i,j : integer;

{ A next függvény állítja elő az n elemű sorozat értékét a t }
{ időpillanatban.                                             }
```

```
function next(t,n:integer):integer;
var i,j,k:integer;
        r:real;
begin
 k:=0; j:=1; i:=1; r:=0;
 repeat
  r:=r+j*cos(i*t*pi/90)/i;     { Egyetlen rezgés ...                }
  inc(i,2);                    { Az együtthatók 1/3, 1/5, 1/7 ... }
  j:=-j;                       { Az előjel váltogatása              }
  inc(k);
 until k=n;                    { Az összetevő rezgések összegzése }
 next:=round((240/pi)*r);      { A függvény egész alakban adja    }
                               { vissza az eredményt.             }
end;


BEGIN
{ Bekapcsoljuk a grafikus módot }
 gd:=detect;
 initgraph(gd,gm,'d:\bp\bgi');       { Ide a saját BGI-elérési   }
                                     { útvonalat kell írni !     }
 directvideo:=false;
 textattr:=14;
 writeln('Négyszögrezgés előállítása ...');

{ Az első hét hullámot rajzolja fel }
 for j:=1 to 7 do
  begin
   moveto(0,140+next(0,j));setcolor(8+j);
      { Egyetlen hullám felrajzolása }
   for i:=0 to 639 do lineto(i,140+next(i,j));
   delay(500);
      { Minden hullám kirajzolása után fél másodpercet vár }
  end;

{ Legvégül egy olyan hullámot rajzol a kép aljára, amelyben }
{ ötven önálló rezgés található!                            }

 j:=50;setcolor(7);
 moveto(0,340+next(0,j));
 for i:=0 to 639 do lineto(i,340+next(i,j));
 writeln('ENTER');
 readln;
 closegraph;
END. of program
```

Hogyan működik a program? A *next* eljárás feladata, hogy kiszámítsa az éppen következő értéket. Két paramétere van: *t*, azaz az aktuális időpillanat és *n*, azaz a sorozat elemi rezgéseinek száma. (A *t* itt tulajdonképpen egy képpont X koordinátájaként jelentkezik.)

A program a grafikus mód bekapcsolásával indul. A képernyő sikeres bekapcsolása után hét hullám kirajzolása következik különböző színekben. Az első hullám csupán egyetlen komponenst tartalmaz, ezért ez még tökéletes szinusz. A következőben már két komponens található, az azt követőben három és így tovább. A hullám a komponensek számának növelése miatt egyre inkább kezd hasonlítani a négyszögjelre. Végezetül egy 50 komponenst tartalmazó hullám felrajzolása következik. Ha a `j:=50` utasításban megnöveljük a j értékét, akkor a hullám kirajzolása tovább tart, de a formája jobban fog hasonlítani a négyszögre. Ahhoz persze, hogy ez a jel tökéletes négyszög legyen, végtelen sok hullámot kellene összegezni.

Háromszög és fűrészfog alakot a fentihez hasonló módon lehet előállítani, csupán a *next* függvényt kell a fenti sorozatok szerint átalakítani:

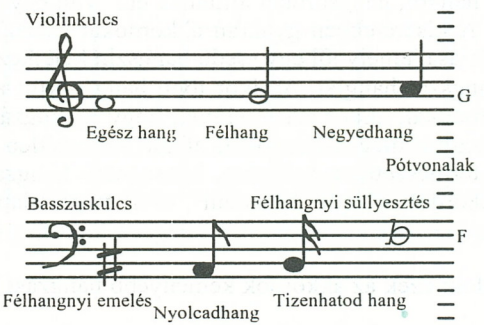
```
{ Háromszögjelet előállító függvény }
function next(t,n:integer):integer;
var i,k:integer;
      r:real;
begin
 k:=0; i:=1; r:=0;
 repeat
  r:=r+sin(i*t*pi/90)/i;
  inc(i); inc(k);
 until k=n;
 next:=round((440/(pi*pi))*r);
end;

{ Fűrészfogjelet előállító függvény }
function next(t,n:integer):integer;
var i,j,k:integer;
        r:real;
begin
 k:=0; j:=1; i:=1; r:=0;
 repeat
  r:=r+j*sin(i*t*pi/90)/(i*i);
  inc(i,2);
  j:=-j;
  inc(k);
 until k=n;
 next:=round((540/(pi*pi))*r);
end;
```

Tekintettel a képernyő felbontására, az amplitúdót meghatározó állandókat itt nem a sorozat számai alapján határoztuk meg, hanem egyéni — a képen jól elhelyezhető — értékeket választottunk.

1.5. Zene, zene, zene

1.5.1. Kotta és hangjegyek

A zene „írott" formája a kotta. A kotta tulajdonképpen teljes jelrendszer — ha úgy tetszik, külön nyelv —, amellyel a különböző zenei formákat, ritmusokat, hangokat szemléletesen és logikusan le lehet írni. Egy gyakorlott zenész kotta alapján képes bármilyen ismeretlen zenedarab lejátszására is. Ebben a könyvben nem akarjuk teljes részletességgel ismertetni a kottában előforduló összes jelet, csupán a legfontosabb részeket emeljük ki.



27. ábra. A kotta néhány fontosabb jelölése

Egy hangjegy két legfontosabb jellemzője a hang magassága és ideje. A kottán a magasságot a vonalak és a vonalközök jelentik. Ezek egy hang távolságra vannak egymástól. A felső violin- (vagy G-) kulcsos kottán a második vonal jelenti a G hangot, és ehhez viszonyítjuk a többit. Az első vonalköz egy hanggal van a G alatt, tehát ez az F hangot jelenti. A további vonalak és vonalközök logikusan követik egymást. Ha valamely hang túl magas vagy túl mély, akkor azokat pótvonalon (és pótvonalközökön) ábrázoljuk. A C hang például az első alsó pótvonalon található meg. A violinkulcsos kottán általában a szóló- (magasabb hangú) játékot írják le, míg a basszuskulcsos kotta a kísérő hangokat (basszus hangokat) írja le. Ez persze nem általános szabály. A basszuskulcs a mélyebb oktávok leírására alkalmas. A basszuskulcsot szokás F-kulcsnak is nevezni, mert a jel a negyedik vonalon található F hangra esik. Mindez persze a valóság eléggé pontatlan leírása, ezeket a fogalmakat és szabályokat zenei könyvekben lehet megtalálni pontosan.

A hang másik fontos jellemzője az ideje — a hossza —; ezt a hangjegyek különböző formájával szemléltetik, amint ez a *27. ábrán* is látható. Fontosak még a félhangok jelzéséhez a módosító jelek, a # és a *b* jel. A # — erről már volt szó korábban — egy félhanggal felemeli az adott hangjegyet, míg a *b* egy félhanggal lesüllyeszti. Ez vonatkozhat csak egyetlen hangra vagy esetleg egyetlen ütemre.

Nem tettünk említést igen sokfajta jelről, de ez a minimális ismeret is elég lesz a hangkártyák programozásához.

1.5.2. Hangsorok, akkordok

Egy zenében általában nem csak egyetlen hang szól, hanem a zene összetett, több hangszerrel egyszerre több hangot, hangsorokat állítanak elő. Van egy szólóhangszer és azt a többi hangszer kíséri. A kíséretekben gyakran akkordokat használnak. Az akkord több hang együttes megszólalása, amelyből egy eredő hangszín keletkezik, de ez jól kiegészíti — kíséri — a szólóhangszer hangját. Amikor több hang együtt az emberi fül számára kellemes összhatást produkál, akkor azt mondjuk, hogy a hangzás **konszonáns**. Ha viszont helytelenül választjuk meg a hangokat, akkor kellemetlen, **disszonáns** hangzást kapunk. Az akkordok természetesen kellemes, konszonáns hangzások. A legtöbb hanghoz többféle kísérő akkordot is lehet „készíteni", ezek mégis alapvetően két fajtába sorolhatók.

– „dúr" akkordok: ezek az akkordok keményebb hangzást kölcsönöznek az alaphangnak;
– „moll" akkordok: ezek az akkordok pedig lágyabban szólnak.

Furcsa módon a két hangzás között gyakran mindössze egy félhang különbség van, mégis igen jól megkülönböztethetők egymástól. Egy akkord már három hangból képezhető, de a hangszerek kialakításától függően több hangot is bele lehet venni egy akkordba. A zongorán négy hangot szoktak egy akkordként játszani, míg a gitáron hat hang is előfordulhat. Nézzük meg példaképpen a C-dúr és a C-moll akkordokat!

C-dúr: **C–E–G–C**
C-mol: **C–D#–G–C**

Az első és az utolsó C hang természetesen egy oktávnyi távolságra van egymástól. Mint látható, a két akkord csak egyetlen hangban tér el, de ezek is csupán félhang távolságra vannak egymástól. Ha egy zongorán egyszerre leütjük az adott hangokat, akkor mégis észrevehetően lágyabbnak érezzük a moll akkordot. **:–o**

1.5.3. Gitárakkordok

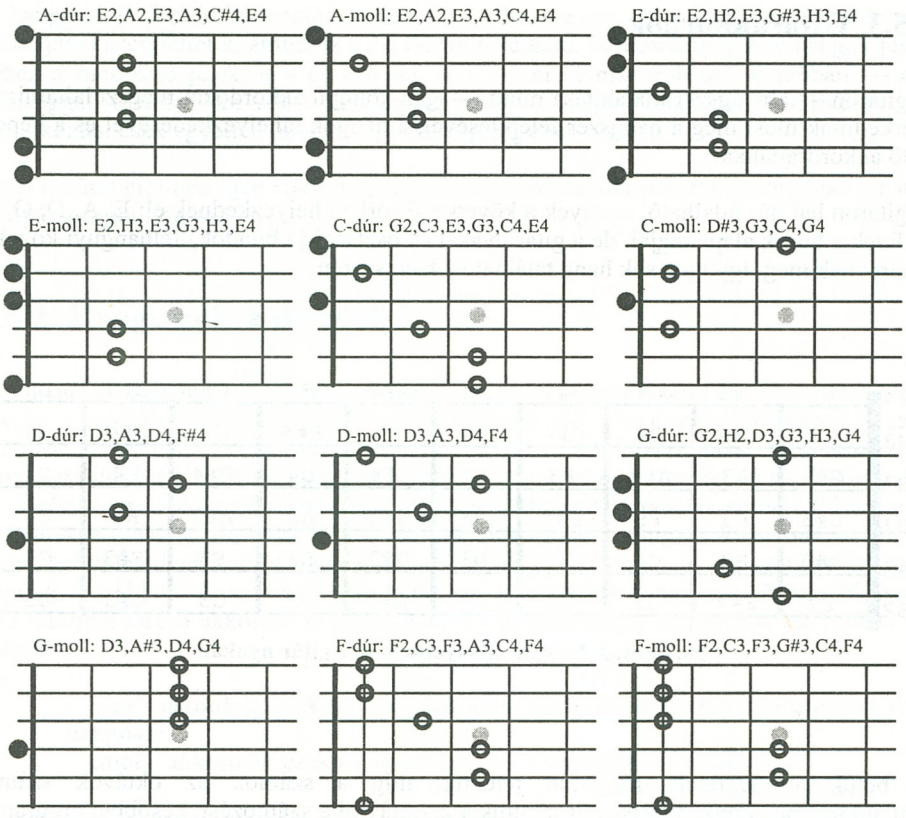
A gitáron — a hangszer kialakítása miatt — igen könnyű akkordokat megszólaltatni. Ismerkedjünk most meg a hangszer felépítésével, a hangok elhelyezkedésével és a képezhető akkordfajtákkal!

A gitáron hat húr található, amelyek a következő sorban helyezkednek el: E, A, D, G, H, E. Ezek a húrok alaphangjai, de a gitár nyakának osztásai (a bundok) félhangnyi közöket határoznak meg, így igen sok hang található a hangszeren.

E4	F4	F#4	G4	G#4	A4	A#4	H4	C5	C#5	D5
H3	C4	C#4	D4	D#4	E4	F4	F#4	G4	G#4	A4
G3	G#3	A3	A#3	H3	C4	C#4	D4	D#4	E4	F4
D3	D#3	E3	F3	F#3	G3	G#3	A3	A#3	H3	C4
A2	A#2	H2	C3	C#3	D3	D#3	E3	F3	F#3	G3
E2	F2	F#2	G2	G#2	A2	A#2	H2	C3	C#3	D3

28. ábra. A hangok elhelyezkedése a gitár nyakán

A betűk a megfelelő hangokat jelentik, míg a számok az oktávok számai. (Tulajdonképpen önkényesen választottuk a 2. oktávtól a számozást, később a programozásnál majd megértjük, hogy ez miért jó.) Látható, hogy ugyanazok a hangok több húron is előfordulnak. A hangszer kialakulásának okait és fejlődését most mellőzzük, bízzunk meg őseink tudásában, és fogadjuk el, hogy ez így van jól. A hangok elhelyezése mindenesetre olyan, hogy az akkordokat nehézség nélkül egy kézzel le lehet fogni, és a kéz fel-le csúsztatásával meg lehet szólaltatni más-más akkordokat.



29. ábra. Gitárakkordok

A *29. ábrán* néhány alapvető gitárakkord látható. Az üres körök a lefogás helyét mutatják, a tömör körökkel jelzett húrok lefogás nélkül is beletartoznak a hangsorba. Természetesen ez nem az összes akkord. Ha az Olvasó megfigyeli az E és az F akkordokat, akkor látható, hogy az F tulajdonképpen az E egy fogással eltolt változata. Ez az eltolás — csúsztatás — természetesen érvényes minden egyes akkordra. Ha például a H-moll akkordot akarjuk lefogni, akkor ezt az A-mollt két fogással felfelé csúsztatva kapjuk meg. Ha az F-dúrt eltoljuk egy fogással, akkor F#-dúrt kapunk, ha még eggyel, akkor G-dúrt és így tovább. Az egyes fogások elég nagy gyakorlatot és kézügyességet igényelnek, de gyakorlott gitárosoknak mindez persze nem okozhat gondot.

2. A beépített Assembler használata

A Turbo Pascal 6.0-s és 7.0-s fordítóprogramba beépítettek egy Assembler fordítót is. Ennek következtében tetszőleges Pascal eljárást vagy függvényt meg tudunk írni assembly nyelven a Pascal forrásszövegben, nincs szükség külön Assembler használatára és az objectfájlok összeszerkesztésére. A beépített Assembler szintaktikája túlnyomó részben megegyezik a legtöbb hagyományos Assembler szintaktikájával, csupán a Pascal nyelv, illetve az abból generált kód szerkezete miatt van némi eltérés — ha úgy tetszik ésszerűsítés — a megszokott Assembler fordítókhoz képest. Mivel a hangkártyákat általában alacsony szinten programozzák, ezért ebben a fejezetben a beépített (built-in) Assembler használatának szabályait ismertetjük, hiszen erre a későbbiekben mindenképpen szükség lesz. A 80x86-os processzorcsalád utasításkészletét nem ismertetjük, erről már bőségesen található irodalom.

2.1. Assembly blokkok

Assembler betéteket a Pascal programon belül bárhol lehet használni, ezeket a fordító egyetlen utasításblokknak tekinti. A betéteket az **asm** kulcsszóval kell kezdeni, és az **end** kulcsszóval kell lezárni:

```
begin
 writeln('Példa assembly betétre...');
 asm
   nop
   nop  { Assembly program }
 end;
end.
```

Ez a pár soros program már működőképes, noha sok értelme természetesen nincsen. Látható, hogy az assembly utasításokat egymás alá írtuk, ami egyébként természetes is, más programoknál is ez a megszokott. A hagyományos Assemblerhez képest eltérés viszont, hogy a megjegyzéseket (kommenteket) itt nem a ; jellel jelöljük, hanem a Pascal szintaxisának megfelelően kapcsos zárójelek közé tesszük. Ezen a módon bárhová elhelyezhetünk Assembler programot, akár a főprogramba, akár valamely eljárásba vagy függvénybe. A 80x86 regisztereit szabadon használhatjuk, kivéve a rendszer működéséhez szükséges regisztereket:

DS – alapesetben mindig a program adatszegmensére mutat, a program globális változói ebben a szegmensben találhatók.
SS – alapesetben a program veremterületére mutat, ide kerülnek a meghívott eljárások, függvények visszatérési címei, illetve ezek paraméterei és lokális változói.
SP – az **SS**-sel együtt a veremmemóriát címzi meg.
BP – az **SS**-sel együtt az eljárások és függvények lokális változóinak és paramétereinek elérésére használja a rendszer.

Ezen regiszterek közül az **SS:SP** párost más esetekben sem szokták használni, hiszen a verem sértetlensége a megszakítások működésének elengedhetetlen feltétele. A **DS** és a **BP** használata viszont annál inkább szükséges lehet. Ha tehát ezeket a regisztereket kell használni, akkor ügyelni kell arra, hogy az Assembler rutin befejezésekor visszakapják az eredeti értékeiket, máskülönben nagy lesz a baj! Arról sem szabad megfeledkezni, hogy ezek a regiszterek címezik meg a globális és a lokális változókat, átírásukkor tehát a program változói elérhetetlenek! A következő példa bemutat egy lehetséges használati módot:

```
asm
  push DS
       { Most szabadon használható a DS regiszter,          }
       { de a program globális változóit nem lehet elérni }
  pop  DS
  push bp
       { Most szabadon használható a BP regiszter,          }
       { de a program lokális változóit nem lehet elérni   }
  pop  bp
end;
```

A DS-t szándékosan írtuk nagybetűvel, ezzel is utalva arra, hogy ez nem általánosan használható regiszter, hanem olyan, amelynek a memóriacímzésben kitüntetett szerepe van. A könyv további részeiben is látható majd, hogy a szegmensregisztereket következetesen nagybetűvel jelöljük. Ez csak egy konvenció, amelyet nem szükséges betartani, bár szerintünk jól kifejezi a szegmensregiszterek kiemelt szerepét.

2.2. Direktívák

Az Assemblerek megszokott értékadó direktívái közül szabadon használhatók a következők:

```
DB    3Eh,100,200              { Bájtok definiálása       }
DW    4000,0F000h              { Szavak definiálása       }
DD    $1E00AAE0                { Dupla szavak definiálása }
DB    'E','A',"szöveg"         { Karakterek és sztringek  }
```

A fentiekkel tetszőleges adatokat definiálhatunk egy Assembler blokkon belül, így a program kódrészében adatokat tudunk elhelyezni. Nem megengedett viszont a következő direktívák használata:

```
EQU,PROC,STRUC,SEGMENT,MACRO
```

Egy egyszerű 8-bájtos tömb, amelyet a kódszegmensben helyezünk el:

```
procedure bitmasks;                                    assembler;
asm
  db   1,2,4,8,16,32,$40,080h
end;
```

A fenti példákból az is látható, hogy az Assembler kétféle írásmódban is elfogadja a hexadecimális számokat:

1. prefix alakban a **$** jellel: $1E00
2. postfix alakban a **h** jellel: 01E00h

Postfix alakban adhatók meg a bináris számrendszerbeli számok is: 11001101b

2.3. Szimbólumok

Az Assemblerben minden egyes eredeti Pascal szimbólumra lehet hivatkozni, azaz címkékre, változókra, állandókra, eljárásokra, függvényekre és típusokra. Ezeken kívül definiálva van még néhány szimbólum a programozás megkönnyítésére:

@code – az aktuális kódszegmens értéke;
@data – az aktuális adatszegmens értéke;
@result – egy függvény visszatérési értékének címe.

Használatukra nézzünk néhány példát:

```
asm
  mov  ax,SEG @data
  mov  ES,ax              { Az ES ráállítása az adatszegmensre }
  mov  ax,SEG @code
  mov  DS,ax              { A DS ráállítása a kódszegmensre    }
  les  di,@result         { Az ES:di pointer ráállítása a      }
                          { függvény visszatérési értékére     }

end;
```

Természetesen a más Assemblerekben megszokott *típusjelző* szimbólumok is használhatók:

```
PTR,BYTE,WORD,DWORD,QWORD,TBYTE,NEAR,FAR
```

Az alapértelmezett szegmensregiszterek felülbírálásához vezették be a következő prefixeket:

```
      SEGCS              { A kódszegmens definiálása          }
      SEGDS              { Az adatszegmens definiálása        }
      SEGES              { Az extraszegmens definiálása       }
      SEGSS              { A veremszegmens definiálása        }

asm
  segcs  mov  al,[bx]    { mov  al,CS:byte ptr [bx]           }
  seges  movsb           { movs byte ptr ES:[si],ES:[di]      }
  segds  mov  cx,[bp-2]  { mov  cx,word ptr DS:[bp-0002h]     }
end;
```

Ezen prefixek mellett az Assembler rugalmasan megengedi a megszokott szegmensfelülbíráló módszert is (CS:, ES:, SS:, DS:).

2.4. Címkék

Az Assemblerben valamely belépési vagy kilépési pontot, cikluskezdetet szoktunk címkével jelölni. A Pascal szintaxisa szerint a LABEL szóval definiált címkékre szabadon lehet hivatkozni az Assembler blokkon belül. Ha a blokknak saját — lokális — címkére van szüksége, akkor a címke nevét a @ jellel kell kezdenünk. Ebből a fordító tudni fogja, hogy a címkét az adott blokkon belül kell keresnie, illetve hibát jelez a fordítás közben, ha nem találja. A lokális címkékre szabadon lehet használni bármelyik feltételes vagy feltétel nélküli ugrást, de a globális címkékre, valamint az eljárásokra és a függvényekre csak feltétel nélküli ugróutasítással lehet ugrani (call, jmp). Nézzünk néhány egyszerű példát:

```
label global;
...
asm
  mov  cx,100
@local1:
  lodsb
  xor  ah,ah
  stosw
  loop @local1           { Ugrás a lokális címkére            }
  call global            { A globális pont meghívása helyesen }
@local2:
  dec  bh
  jnz  @local2           { Ez is lokális ugrás                }
  jc   global            { Ez hibás, nem fordítja le          }
end;
```

2.5. Assembly eljárások és függvények

Az eljárásokat és a függvényeket az alábbi formában lehet Assemblerben megadni:

```
procedure asmproc(a,b:word; c:char);                    assembler;
asm
   ...
end;          { Assembly eljárás  }

function asmfunc(a,b:word; s:string):byte;              assembler;
asm
   ...
end;          { Assembly függvény }
```

Az eljárások és a függvények fejléce után az **assembler** direktíva tehát tudatja a fordítóval, hogy az adott eljárás vagy függvény teljes egészében Assemblerben lesz írva. Ebben az esetben a **begin..end** páros elhagyható, és helyette az **asm..end** páros használható. Ha az eljárásnak lokális változói vannak, akkor azokat a megszokott szintaktika szerint lehet deklarálni az **asm** kulcsszó előtt. A paraméterekre és a lokális változókra egyszerűen a nevükkel lehet hivatkozni. Ekkor azonban két fontos szabályt be kell tartani:

1. Nem használható egy Assemblerben fenntartott szó szimbólumként, paraméter- vagy változónévként. Ha olyan változóval dolgozunk, amelynek neve megegyezik egy Assembler alapszóval: például AX, akkor azt az & jellel kell jelölni. A különbség jól érzékelhető:

```
mov    bx,ax        { Az AX regiszter betöltése a BX-be }
mov    bx,&ax       { Az AX változó   betöltése a BX-be }
```

2. Az adott változónak és regiszternek „méret szerint" egyeznie kell. Ha tehát a fenti **asmproc()** eljárást vesszük alapul, akkor jól látható, hogy az **a** és a **b** paraméter *word* típusú, míg a **c** *char*, amely megfelel a *bájt* típusnak.

```
mov    ax,a                 { Ez helyes,                          }
mov    bx,b                 { ez is helyes,                       }
mov    dl,c                 { és még ez is az                     }
mov    di,c                 { Típuskeveredés, nem fordítja le     }
mov    dh,byte ptr [ a]     { Típusátdefiniálás, helyes           }
mov    dh,a.byte            { Így is lehet típust átdefiniálni    }
mov    si,word ptr [ c]     { Típusátdefiniálás, lefordítja,      }
                            { de hibás eredményt adhat!           }
```

Ezek a példák jól szemléltetik a lehetőségeket és főképp az ezekben rejlő veszélyeket. Az Assembler a méretátdefiniáló operátorokkal megengedi, hogy bármely változót vagy paramétert más típusként érjünk el, mint ahogyan deklaráltuk, de nem ellenőrzi — hiszen

nem is ellenőrizheti —, hogy helyesen használjuk-e az átdefiniálást. A fenti példa ötödik sorában az átdefiniálás veszélytelen, mert egyértelmű. A **dh** regiszterbe az **a** paraméter alsó bájtja töltődik. Az utolsó sorban található átdefiniálás viszont beláthatatlan következményekkel járhat, mert egy bájt méretű változóra wordként hivatkozunk. Az **si** regiszter alsó bájtjában ugyan megjelenik a **c** paraméter, a felső bájt tartalmáról azonban semmit sem mondhatunk, abban bármi lehet. Az ilyen átdefiniálásokat éppen ezért csak nagyon alapos körültekintéssel célszerű használni!

2.5.1. Paraméterek és visszaadott értékek

A Borland Pascal 7.0-ban egy eljárásnak vagy függvénynek paraméterként értékeket háromféleképpen lehet átadni:

1. Érték szerint. Ebben az esetben a fordító a veremben megfelelő méretű memóriát foglal le a változó számára, majd a változó értékét a verembe másolja.
2. Cím szerint. Ilyenkor a fordító a változót nem másolja át a verembe, hanem csak a címét. Assembly szinten ez azt jelenti, hogy az eljárás egy mutatót kap, amely a megadott változóra mutat. A cím szerinti átadást a **var** kulcsszóval kell jeleznünk a deklarációs részben.
3. Cím szerint, konstans alakban. Ez assembly szinten megegyezik a normál cím szerinti átadással, különbség lehet azonban a változó eredeti címét — származását — illetően. Az állandókkal megadott paraméterek ugyanis mindig a kódszegmensben helyezkednek el, innen másolódnak át a verembe. A konstans alakú átadás — amelyet a **const** kulcsszóval kell jelezni — arra utasítja a fordítót, hogy a paramétert cím szerint adja át még akkor is, ha az a kódszegmensben található.

A fenti esetek megvilágításához nézzünk egy rövid példaprogramot:

```
var
      s1 : string;

procedure proc1(s:string);              { Érték szerinti átadás }
begin
  ...
end;

procedure proc2(var s:string);          { Cím szerinti átadás   }
begin
  ...
end;

procedure proc3(const s:string);        { Cím szerinti átadás   }
begin                                   { konstans alakban      }
  ...
end;
```

```
BEGIN
 proc1(s1);
 proc1('első hívás');
 proc2(s1);
 proc2('második hívás');        { ** Ez hibás ** }
 proc3(s1);
 proc3('harmadik hívás');
END.
```

A *proc1* eljárásnál a sztringet érték szerint adjuk át, ami azt jelenti, hogy minden híváskor a teljes szöveg — azaz esetleg 256 bájt — átmásolódik a verembe. Ez eléggé sok időt vesz igénybe, ezért az ilyen átadás csökkenti a program sebességét. A *proc2* már csak a szöveg címét adja át a verembe, ez minden esetben 4 bájt, tehát ez a módszer jóval gyorsabb az előzőnél. Ebben az esetben viszont nem használhatjuk az állandóként megadott szöveget, vagyis a *proc2* második hívása hibás. A fordító ezt a sort nem tudja lefordítani, mert egy változó címét várja, helyette azonban egy állandó szöveget kap. Egyik megoldás sem az igazi! A harmadik, vagyis a konstans szerinti átadás viszont ötvözi a kétféle átadás jó tulajdonságait: a szöveg címét adja át, azaz gyors, ugyanakkor elfogadja az állandóval megadott szöveget is. A *proc3* első hívásakor tehát a program az s1 változó címét adja át, míg a második hívás esetén a 'harmadik hívás' sztring címét, ami jelen esetben természetesen a kódszegmensben található. A **var** és **const** szerinti paraméterátadás másik nagy előnye, hogy a fordító az ilyen átadással deklarált paraméterek esetén nem követel típusazonosítást. Az ilyen deklarációval tehát tetszőleges hosszúságú és típusú változó is átadható. Ez nagyon kényelmes lehet bizonyos esetekben, amikor különböző típusokat akarunk átadni ugyanannak az eljárásnak vagy függvénynek. A következő példa bemutatja az ilyen „típus nélküli" paraméterátadás rugalmasságát:

```
var
      b : byte;          { Bájt változó          }
      c : char;          { Karakter változó      }
      s : shortint;      { Rövid egész változó   }

procedure decrement(var a); assembler;
asm
  les   di,a
  dec   byte ptr ES:[di]  { Csökkentjük az "a"-t }
end;

BEGIN
 decrement(b);           { Most egy bájtot csökkentünk          }
 decrement(c);           { Most egy karaktert csökkentünk       }
 decrement(s);           { Most egy rövid egészet csökkentünk }
END.
```

A fenti programban a *decrement* eljárás egy cím szerint átadott „a" nevű paramétert vár, típusára semmilyen megkötés nincs. A főprogramban elhelyezett mindhárom hívás tehát helyes lesz. Az ilyen paraméterek esetében azonban mindig a programozónak kell arról

gondoskodnia, hogy illegális érték ne kerüljön átadásra, hiszen a fordító ezt ilyenkor nem tudja ellenőrizni.

Nézzünk meg most már néhány, a gyakorlatban is használható egyszerű Assembler eljárást és függvényt!

1. Egy mutató (pointer) szegmens- és offszetrészének meghatározása:

```
procedure reptr(p:pointer; var s,o:word);          assembler;
asm
  les   dx,p              { A pointer az ES:dx regiszterpárba }
  mov   ax,ES             { kerül, majd az ax:dx tárolja      }
  les   di,s              { Az ES:di most az s-re mutat       }
  mov   ES:[di],ax        { A szegmensrész visszaadása        }
  les   di,o              { Az ES:di most az o-ra mutat       }
  mov   ES:[di],dx        { Az offszetrész visszaadása        }
end;
```

Az eljárás a **p** mutató szegmens- és offszetrészét az **s**,**o** szavakban helyezi el. Az **s** és az **o** két *word* típusú változó. Látható, hogy a **p**-t érték szerint, míg az **s**-et és az **o**-t cím szerint adtuk át. Ez az eljárás tulajdonképpen tehát egy olyan függvény, amelynek két visszatérési értéke van, de ezeket csak már definiált változókba tudja tenni.

2. Határérték-ellenőrzés.

```
function inrange(x,min,max:word):boolean;          assembler;
asm
  xor   al,al             { A visszatérési érték az AL(=0)    }
  mov   bx,x              { Az x a BX-be kerül                }
  cmp   bx,min            { Kisebb a minimumnál?              }
  jb    @exit             { Vége, ha igen                     }
  cmp   bx,max            { Nagyobb a maximumnál?             }
  ja    @exit             { Vége, ha igen                     }
  inc   al                { Az AL=1 a true érték              }
@exit:
end;
```

Ez a függvény egy *boolean* típusú értéket ad vissza, amely igaz, ha a $min<=x<=max$ feltétel teljesül, egyébként pedig hamis. Itt mindhárom paramétert érték szerint adtuk át. A visszatérési értéket az AL-ben helyeztük el.

A Pascalban a függvények visszatérési értékére a következő szabályok vonatkoznak:

- bájt méretű (byte, char, shortint) változókat az AL regiszterben kell visszaadni;
- szó méretű (word, integer) változókat az AX regiszterben kell visszaadni;
- dupla szó méretű (pointer, longint) változókat a DX:AX regiszterpárban kell visszaadni;

- real változókat a DX:BX:AX regiszterekben kell visszaadni;
- sztring változókat a **@result** szimbólumon keresztül kell visszaadni.

3. Két bájt „NEM-ÉS" (NAND) kapcsolata:

```
function nand(x,y:byte):byte;                    assembler;
asm
  mov  al,x                    { AL-be az x paraméter  }
  and  al,y                    { AL:=x and y           }
  not  al                      { AL:=not (x and y)     }
end;
```

4. Egy tömb legnagyobb elemének megkeresése:

```
function max(var i_array; no : integer):integer;      assembler;
asm
  les  si,i_array          { ES:SI a tömb első elemére mutat   }
  mov  cx,no               { CX-be kerül a darabszám           }
  xor  ax,ax               { A maximum indulási értéke 00      }
@find:
  cmp  ax,ES:[si]          { Összehasonlítás                   }
  jge  @great              { Ugrás, ha AX>=ES:[si],            }
  mov  ax,ES:[si]          { különben ez lesz az új maximum    }
@great:
  add  si,2                { A mutató növelése (integerek)     }
  loop @find               { Keresés tovább, amíg CX<>0        }
end;
```

Ez a függvény egy egészeket tartalmazó tömb legnagyobb elemét keresi meg. A tömböt természetesen nem adtuk át a függvénynek, hanem csak a címét. Ez a cím (*i_array*) kerül az ES:SI regiszterekbe. A CX-be betöltjük az elemek számát — ez a *no* paraméter, amelyet érték szerint adtunk át —, és a ciklus minden elemmel összehasonlítja az AX-et. Ha valamelyik elemet nagyobbnak találja az AX-nél, akkor az AX-be ezt az új értéket teszi. A ciklus lefutása után tehát az AX az addig talált legnagyobb elem értékét adja meg.

5. Egy bájt bináris formátumának kiírása. Ez a példa a sztringeket visszaadó függvényekben a **@result** szimbólum használatát mutatja be.

```
function binary(b:byte):string;                  assembler;
asm
  mov  ah,b                { AH:= a bájt értéke                }
  les  di,@result          { Az ES:DI a visszaadandó szövegre  }
                           { mutat                             }
  cld                      { Irányjelző bit: D=0               }
  mov  cx,8                { Egy bájtban 8 bit található       }
  mov  al,cl
  stosb                    { A sztring első bájtja a hossza    }
@bits:
```

```
  shl   ah,1              { Eltolás, Cy:= a következő bit      }
  mov   al,'0'            { AL:= a '0' karakter kódja          }
  adc   al,0              { A bittől függően AL= '0'/'1'       }
  stosb                   { Kiírás a célsztringbe              }
  loop  @bits             { Nyolcszor végzi el                 }
end;
```

A függvényben úgy használtuk a **@result** szimbólumot, mintha egy mutató lenne. Valójában ez annak a 256 bájtos területnek a címét jelenti, amelyet a fordító a veremben foglalt le a függvény visszatérési értéke számára. Az átalakítás módszere egyszerű: minden bitet sorban kitolunk a *Cy* jelzőbitbe, és ettől függően '1'-et vagy '0'-t írunk a sztringbe.

Noha nem törekedtünk teljességre, azért reméljük, hogy az itt bemutatott példák megmutatták azokat az alapvető ismereteket, amelyekre a beépített Assembler használatakor szükség lehet. A későbbi programokban alaposan ki is fogjuk használni az adódó lehetőségeket, ezért talán nem haszontalan, ha az Olvasó is megpróbálkozik hasonló egyszerű függvények és eljárások megírásával.

3. Az ADLIB hangkártya

Az első jelentős audiokiegészítő, amelyet az IBM PC-hez kifejlesztettek, az ADLIB hangkártya volt. Éppen ezért szolgáltatásai meglehetősen szerénynek fognak tűnni a mostani modern hangkártyákéhoz képest. Az audiojeleket még — a mai technikákhoz képest hagyományos — analóg módon állították elő. (A kártyán digitálisan vezérelhető analóg hangkeltő áramkörök találhatók. Az eredeti FM chip az **YM–3812** típusjelű áramkör volt, de mostanában már a továbbfejlesztett **OPL–3** FM chipet használják. A két chip közötti eltérést a Sound Blaster 16 kártyánál ismertetjük.) A hangchip frekvencia-modulált hangcsatornákkal állítja elő a hangokat, amelyeket aztán egyszerűen összekever, és ez az eredő hang jut a kártya analóg kimenetére. A kártyán kilenc — egymástól független — csatorna található. Minden csatornához tartozik egy burkológörbegenerátor, amellyel tetszőlegesen meg lehet adni a burkológörbe négy szakaszának idejét. A csatornák kimenő hullámformáját négy rögzített hullámalak közül lehet kiválasztani. A csatornák programozása duplaoperátoros, ami azt jelenti, hogy a hang bizonyos visszacsatolási algoritmusok után modulálni tudja magát, így különleges, összetett hangzások is létrehozhatók. A csatornákat amplitúdómodulálni lehet, és vibrátorok is alkalmazhatók. Időzítési célokra két egyszerűen programozható időzítő egység található a kártyán, amelyek képesek megszakítást is adni. Összességében a kártya kiválóan alkalmas az „*FM voice*", azaz a frekvenciamodulált hangkeltésre.

3.1. Az ADLIB kártya regisztereinek elérése

Az ADLIB kártyát I/O csatornákon keresztül lehet elérni, minden adatforgalom így valósul meg. A kártya a 65536 I/O címből mindössze kettőt használ fel. A két I/O port címe a következő:

$388 – cím/állapotport *(Address/Status port)*
$389 – adatport *(Data port)*

A kártyán ezzel szemben egy programozói regiszterkészlet található, amely 244 belső regisztert tartalmaz. A regisztereket a fenti két I/O porton keresztül lehet programozni: először ki kell írni az adott regiszter címét a címregiszterbe ($388), majd az adatot az adatregiszterbe ($389). A kártya regiszterei jórészt csak írhatók, ezért érdemes létrehozni a memóriában egy — a regiszterekkel azonos kiosztású — memóriatömböt, amely „*shadow RAM*"-ként (árnyéktárként) tárolja a regiszterek értékét. A kártya a processzorhoz képest elég lassú, ezért a regiszterek írásakor bizonyos időzítési szabályokat be kell tartani. A címregiszterbe való beírás után várni kell tizenkét buszciklust, ennyi idő kell az ADLIB-nek, hogy a cím alapján el tudja érni a megfelelő belső regisztert. A tizenkét

buszciklus letelte után lehet kiírni az adatot az adatregiszterbe. Az adat kiírása után nyolcvannégy buszciklusnak kell eltelnie, amíg a kártya egy következő regiszterhozzáférést végezhet.

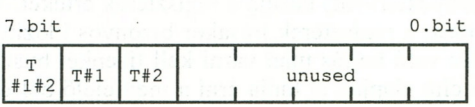
Ezek az idők különböző gépek esetében természetesen különbözőek lehetnek (mivel a busz sebessége gépenként változik), ezért inkább mikroszekundumban (µs) adják meg a várakozási időket: a cím elfogadásának ideje 3.3 mikroszekundum, míg a következő érvényes regiszteroperációt 23 mikroszekundum után lehet biztonságosan elvégezni. :–(A programokat általában úgy készítik, hogy egy viszonylag gyors gépen (pl. 486DX2—66 MHz-esen) beállítják a megfelelő várakozásokat. Ha a kártya egy ilyen gyors gépen is jól működik, akkor biztos, hogy a lassúbb gépeken is korrekt lesz a regiszterek írása.

A regiszterek írása a következő egyszerű eljárással valósítható meg:

```
procedure WriteReg(regnum,value:byte);          assembler;
asm
  mov  dx,$0388          { ADLIB I/O báziscím: $388          }
  mov  al,regnum         { A belső regiszter száma           }
  out  dx,al             { a címregiszterbe kerül            }
  mov  cx,6              { Az első várakozási konstans a CX-be }
@wait1:
  in   al,$00
  loop @wait1            { Az első várakozás                 }
  inc  dx                { Az adatregiszter címe: $389       }
  mov  al,value          { Az adat az AL-be kerül,           }
  out  dx,al             { onnan pedig az adatregiszterbe    }
  mov  cx,35             { A második várakozási konstans a CX-be }
@wait2:
  in   al,$00
  loop @wait2            { A második várakozás               }
end;
```

Az eljárásban szereplő időzítési állandókat (mov cx,6 és mov cx,35) egy 33 MHz-es 486-oson állítottuk be. Lehetséges, hogy ezeket az értékeket gyorsabb processzoroknál meg kell növelni! :–)

A kártya regiszterei — egy kivétellel — csak írhatók. Az egyetlen olvasható regiszter az állapotregiszter *(status register)*, amely a **$388** I/O címen érhető el. Innen egy egyszerű IN utasítással lehet beolvasni a kártya aktuális állapotát. Az állapotregiszter bitkiosztása a következő:



bit 0..4 – nincs definiálva (*not used*)
bit 5 – a második időzítő lejárt (*timer #1 has expired*)
bit 6 – az első időzítő lejárt (*timer #2 has expired*)
bit 7 – valamelyik időzítő lejárt (*either timer has expired*)

A többi — csak írható — regisztert különböző csoportokba szervezik, amelyek egy-egy csatorna jellemzőit tartalmazzák. A következő táblázat bemutatja ezeket a regiszter-csoportokat:

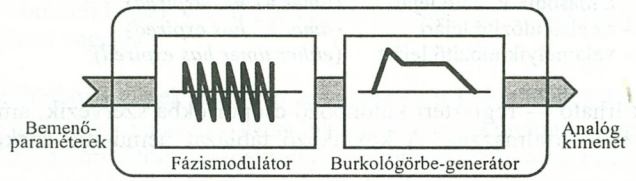
Belső regisztercím	Regiszterfunkció
$01	hullámforma-kiválasztás engedélyezése *(EWC)*
$02	első időzítő adat *(timer #1)*
$03	második időzítő adat *(timer #2)*
$04	időzítővezérlő *(timers control)*
$08	FM/beszédszintézis mód *(speech/FM select)*
$20..$35	effektusok *(AM/vibrato/multiple)*
$40..$55	kimeneti jelszint *(output level)*
$60..$75	burkológörbe *(attack/decay)*
$80..$95	burkológörbe *(sustain/release)*
$A0..$A8	frekvencia alsó bájt *(frequency-low)*
$B0..$B8	frekvencia felső bájt *(frequency-high/start sound)*
$BD	ritmusvezérlő *(rhytm control)*
$C0..$C8	visszacsatolás-vezérlő *(feedback control)*
$E0..$F5	hullámforma-vezérlő *(waveform type)*

A $20, $40, $60, $80, és a $E0 című regisztercsoportok két-két operátort tartalmaznak, mivel minden egyes FM csatornához két belső hanggenerátor tartozik. Az operátor elnevezés nem túl szerencsés, de az angol irodalomban mindenhol ilyen néven hivatkoznak a dupla hanggenerátorokra. A könyvben mindenesetre a következő módon definiáljuk a fogalmakat: minden csatornához két hanggenerátor tartozik, és a generátorokat vezérlő regisztereket nevezzük operátoroknak. A két operátor együttesen határozza meg a kimenő hullámalakot. A csatornákhoz a következő táblázat alapján lehet meghatározni a két operátor regisztercímét:

Csatorna	0	1	2	3	4	5	6	7	8
Operátor #1	$00	$01	$02	$08	$09	$0A	$10	$11	$12
Operátor #2	$03	$04	$05	$0B	$0C	$0D	$13	$14	$15

A két operátor hatását — pontosabban a második operátor elsőre való hatását — a visszacsatolás vezérlő regiszter egy bitje határozza meg. Az irodalomban sok helyen az első operátort *Modulator*-nak nevezik, a másodikat pedig *Carrier*-nek.

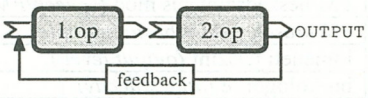
Az operátorok belső szerkezete valami ilyesmi:



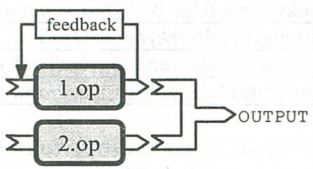
30. ábra. Egy operátor (hanggenerátor) belső felépítése

Az operátor a bemenőparaméterek alapján beállítja a belső eszközeit, és a kimenetén megjelenik az ennek megfelelő hullámalak. Egy csatorna két operátorát kétféle módon lehet egymáshoz kapcsolni, ettől függ a hangszintézis módja:

1. FM szintézis (soros)



2. Additív szintézis (párhuzamos)

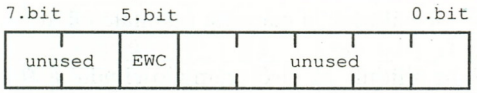


A kétféle összekapcsolásról bővebben majd a $C0...$C8 regiszterkészletnél lesz szó.

3.2. Az ADLIB regiszterkészlete

Ebben az alfejezetben az ADLIB regisztereit mutatjuk be. A regisztereknél ismertetjük az egyes bitek funkcióit, kitérve az esetleges összefüggésekre és mellékhatásokra, amelyekre a programozás során figyelni kell. Ezek közül az egyik legfontosabb, hogy egyes regiszterek kicsit másképp viselkedhetnek a felhasználó saját hangkártyáján, mint ahogyan azt itt leírjuk. A világon igen sok cég foglalkozik ADLIB kompatíbilis hangkártyák gyártásával. Bár minden egyes cég garantálja a kompatibilitást, azért némi eltérés mindig adódhat. Ez egyébként általánosan igaz valamennyi hangkártyára.


Regiszter $01: hullámforma-engedélyező regiszter *(Enable Waveform Control)*



A regiszter 5. bitje engedélyezi a hangcsatornák hullámformájának kiválasztását. Ha az 5. bit 1, akkor a *$E0..$F5 waveform type* regiszterek kiválasztják a megfelelő hullámformát az egyes csatornákhoz. Ha az 5. bit 0, akkor ezen regiszterek írása nincs engedélyezve.

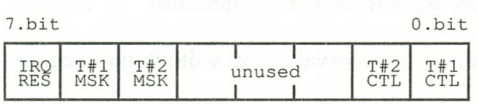

Regiszter $02: első időzítő adatregiszter *(Timer #1 Data)*


Regiszter $03: második időzítő adatregiszter *(Timer #2 Data)*

Mindkét időzítőhöz egy 8-bites számlálóérték tartozik. Ha az adott számláló engedélyezve van a *$08 timers control* regiszter megfelelő bitjének beállításával, akkor az időzítő adatregiszterében lévő érték elkezd növekedni. Ha az adatregiszter túlcsordul, akkor egy TIMER (INT $08) megszakítás generálódik, és az állapotregiszter megfelelő bitjei bebillennek. Az első időzítő a 7. és a 6. bitet, míg a második időzítő a 7. és az 5. bitet billenti 1-be az állapotregiszterben. Ha a megszakítások tiltva voltak — vagy a CLI utasítással, vagy a *timers control* regiszterrel —, akkor csak az állapotregiszter bitjei billennek be. Mindkét időzítőhöz tartozik egy adott időállandó, amelynek leteltével az időzítő adatregisztere növekedhet:

Időzítő	Időállandó	Minimális idő	Maximális idő
Timer #1	80 µs	80 µs	20480 µs
Timer #2	320 µs	320 µs	81920 µs

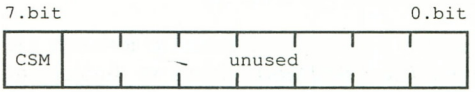

Regiszter $04: időzítő vezérlő *(Timers Control)*



IRQ,RES: A regiszter 7. bitje a megszakításokat és a regiszter többi bitjét maszkolja. Alapállapotba hozza mindkét időzítő bitjeit. Ha 1-be van állítva, akkor a regiszter többi bitjének állítása nem engedélyezett.
T#1 MSK: Az első időzítő engedélyezése/tiltása. Ha 1, akkor a 0. bit nem állítható.
T#2 MSK: A második időzítő engedélyezése/tiltása. Ha 1, akkor az 1. bit nem állítható.

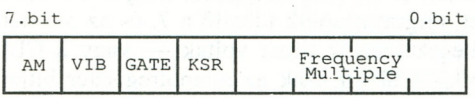
T#2 CTL: A második időzítő indítása. Amikor 1-be állítjuk, a második számláló elindul a *$03 timer #2 data* regiszter aktuális értékével.
T#1 CTL: Az első időzítő indítása. Amikor 1-be állítjuk, az első számláló elindul a *$02 timer #1 data* regiszter aktuális értékével.

Regiszter $08: FM/beszédszintézis mód *(Speech/FM Voice Mode Select)*



Ennek a regiszternek csak a 7. bitje definiált. Ha ez 1, akkor az ADLIB átkapcsol összetett szinuszhullám beszédszintézis *(composite sine-wave speech synthesis)* módba. Az FM csatornák használatakor ennek a bitnek 0-nak kell lennie!

Regiszter $20..$35: effektusok *(Frequency Mul/KSR/GATE/VIB/AM Mode)*



AM: Az adott csatorna amplitúdómodulációjának bekapcsolása. Az amplitúdómoduláció intenzitását a *$BD rhytm control* regiszterben lehet megadni.

VIB: Vibráto effektus bekapcsolása. A vibráció intenzitását a *$BD rhytm control* regiszterben lehet megadni.

GATE: A burkológörbe *sustain* szakaszának viselkedése. Ha ez a bit 0, akkor a *decay* ciklus után a hang a leesik *sustain* szintre, majd a *release* ciklussal a hang lecseng. Ha a bit értéke 1, akkor a hang a *sustain* szinten mindaddig kitartva marad, amíg a bit 0-vá válik. Ezzel egy burkológörbe a *sustain* szinten tetszőleges ideig kitartható!

KSR: Skálatényező. A burkológörbe ciklusának időintervallumát változtatja meg. Az időintervallumok rövidebbek, ha ez a bit 0.

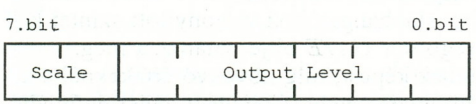
Frequency Multiple: Frekvencia szorzóérték összetett hanghatások előállításához. A regiszter alsó 4 bitje harmonikus operátorokat ad meg. Ezt különböző akkordok készítésére lehet felhasználni. Az alábbi értékeket a csatorna alapfrekvenciájához viszonyítva kell érteni:

Érték	Szorzó	Jelentés
0000 00	0.5	egy oktávval alatta
0001 01	1	a csatorna alapfrekvenciáján
0010 02	2	egy oktávval felette
0011 03	3	egy oktávval és öt félhanggal felette
0100 04	4	két oktávval felette
0101 05	5	két oktávval és három félhanggal felette (majorálva)
0110 06	6	két oktávval és öt félhanggal felette
0111 07	7	két oktávval és hét félhanggal felette (minorálva)
1000 08	8	három oktávval felette
1001 09	9	három oktávval és két félhanggal felette (majorálva)
1010 0A	10	három oktávval és három félhanggal felette (majorálva)
1011 0B	10	ugyanaz, mint az 1010 eset
1100 0C	12	három oktávval és öt félhanggal felette
1101 0D	12	ugyanaz, mint az 1100 eset
1110 0E	15	három oktávval és hét félhanggal felette (majorálva)
1111 0F	15	ugyanaz, mint az 1110 eset

Normál esetben, amikor a csatornát egyszerű hangkeltésre akarjuk használni, a 0001 és a 0010 érték a legkedvezőbb.

A regiszter dupla operátoros.

Regiszter $40..$55: kimeneti jelszint (*Output Level*)



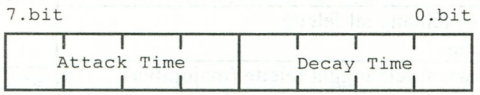
A felső két bit (*Scale*) a csatorna kimeneti szintjének csökkenését adja meg az oktávok megváltozásakor:

Scale bitek	Jelszintcsökkenés
00	nincs csökkentés
01	1.5 dB/oktáv
10	3 dB/oktáv
11	6 dB/oktáv

Az alsó hat bit (*Output Level*) a kimenet hangerejét adja meg, de egyes komplemens alakban, azaz inverz módon, mert ez tulajdonképpen a kimeneti jel csillapítása. A programozáshoz csupán annyit kell tudni, hogy a regiszter $00 értéke esetén a leghangosabb a kimenet, majd a regiszter értékét növelve a kimeneti jelszint fokozatosan csökken. Ha a regiszter értéke $FF, akkor gyakorlatilag nincs kimeneti jel.

A regiszter dupla operátoros.

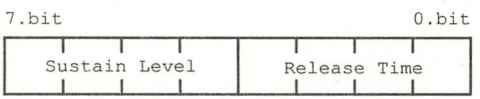
Regiszter $60..$75: a burkológörbe AD összetevője *(ADSR Generator Attack/ Decay)*



A regiszter felső 4 bitje a burkológörbe-generátor felfutási (*Attack*) szakaszának idejét, míg az alsó 4 bit a lecsengési (*Decay*) szakasz idejét adja meg. Az időintervallumok 0000 estén a leghosszabbak, és 1111 estén a legrövidebbek. Ez gyakorlatilag azt jelenti, hogy az egyes szakaszok az 1111 esetben teljesen eltűnnek, a 0000 esetben viszont nagyon hosszúra nyúlnak.

A regiszter dupla operátoros.

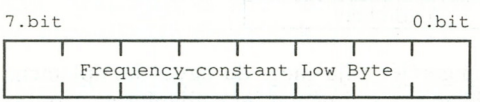
Regiszter $80..$95: a burkológörbe SR összetevője *(ADSR Generator Sustain/ Release)*



A felső négy bit a kitartás (*Sustain*) erősségét adja meg. Az ADSR görbe többi szakaszától eltérően itt nem időintervallumot, hanem az alaphangerőhöz viszonyított szintet kell megadni. (A *Sustain* idejét a *$20 effektusok* regiszter *GATE* bitje határozza meg.) 0000 esetben nincs hangerőcsökkenés a *Decay* szinthez képest, míg növekvő értékeknél a kitartási szint egyre lejjebb esik. Az alsó négy bit az elengedés (*Release*) idejét definiálja. Az AD szakaszokhoz hasonlóan itt is a 0000 jelenti a leglassúbb elengedést, míg az 1111 a leggyorsabbat.

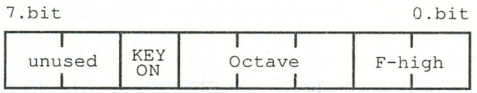
A regiszter dupla operátoros.

Regiszter $A0..$A8: a frekvenciakonstans alsó bájtja *(Frequency-low)*



Ez a regiszter a *$B0 Frequency-high* regiszterrel együtt állandó értéket határoz meg egy oktáv hangjaihoz. A regiszter értéke nem a frekvencia értéke, hanem egy ahhoz rendelt számérték.

Regiszter $B0..$B8: a frekvenciakonstans felső bájtja *(Frequency-high/Octave/ KEY ON)*



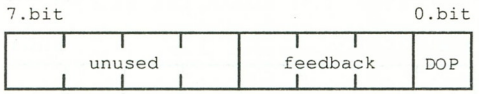
F-high: A frekvenciakonstans felső két bitje. A *$A0 Frequency-low* regiszterrel együtt egy 10-bites számot tartalmaz minden hanghoz. Az alábbi táblázat megmutatja a *normál A (440 Hz)* hangot tartalmazó oktáv hangjaihoz tartozó értékeket:

Hangjegy	Frekvencia	Frekvencia-konstans
C	261.6	$157
C#	277.2	$16B
D	293.7	$181
D#	311.1	$198
E	329.6	$1B0
F	349.2	$1CA
F#	370.0	$1E5
G	392.0	$202
G#	415.3	$220
A	440.0	$241
A#	466.2	$263
H	493.9	$287

Octave: A regiszter 2..4. bitjei az oktávszámot határozzák meg. Mivel ez hárombites érték, az ADLIB az adott frekvenciatáblázat nyolc oktávját képes előállítani. Ha megváltoztatjuk a frekvenciaszámokat, akkor a kártya természetesen más hangsor oktávjait állítja elő.

KEY ON: Az 5. bit egy ADSR ciklust indít el. Ha ez a bit magas, akkor elkezdődik egy ADSR ciklus, és a kimeneten megjelenik a hang. Ha a bitet bármikor — akár egy éppen aktív ADSR ciklus alatt — alacsonyra állítjuk, akkor a hang megszűnik. Ezzel a bittel tehát ki-be kapcsolgathatjuk a hangot. Ha az ADSR ciklus lefutott (a hang teljesen lecsengett), de a *KEY ON* bitet bekapcsolva hagytuk, akkor a csatorna addig nem fog hangot adni, amíg egyszer 0-ba nem áll a bit, majd újra 1-es nem lesz. Ennek az az oka, hogy a hangcsatornák indítása felfutó (pozitív) élvezérelt. Ezt a felfutó élt állítjuk elő a *KEY ON* bit 0→1 átmenetekor. A programozás szempontjából ez azt jelenti, hogy minden egyes hang kiadásakor először ki kell kapcsolni (*KEY ON* = 0) az adott csatornát, majd a paraméterek beállítása után (ADSR, hangerő, effektusok) újra be kell állítani (*KEY ON* = 1).

Regiszter $C0..$C8: visszacsatolási szint *(Feedback Strength)*

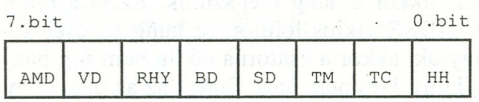


DOP (Double Operators): A 0. bit a kétoperátoros regiszterek operátorainak viszonyát határozza meg. Ha a bit értéke 0, akkor az 1. operátor modulálja a 2. operátort (ez a fejezet elején bemutatott FM szintézis). Ebben az esetben csak a 2. operátorhoz tartozó generátor kelti a hangot. Ha a bit 1, akkor mindkét operátor generátora közvetlenül generálja a hangot (azaz a generátorok kimeneti jelei össze vannak keverve, de nem modulálják egymást – ez az additív szintézis). Az összetett hangzások előállítása sokkal könnyebb abban az esetben, ha a bit értéke nulla. A szerző személyes tapasztalata: ha csak a 2. operátor generátora kelti a hangot, akkor annak olyan kicsi a kitöltési tényezője, hogy szinte alig hallható.

Visszacsatolás	Fáziseltolás
0 000	0
1 001	π/16
2 010	π/8
3 011	π/4
4 100	π/2
5 101	π
6 110	π*2
7 111	π*4

Feedback: Az 1..3 bitek a visszacsatolás erősségét határozzák meg. Ez az érték 3-bites, így nyolcféle visszacsatolási szint állítható be. Ha ez az érték 000, akkor nincs visszacsatolás. Más esetekben az első operátor hanggenerátora saját kimeneti jelének egy részét visszacsatolja a saját bemenetére additív szintézis esetén, illetve a második operátor jelét visszacsatolja az első bemenetére FM szintézis esetén. A 001 a legkevesebb részt jelenti, az 111 a legtöbbet. A fenti táblázatban a visszacsatolásokhoz tartozó fáziseltolásokat foglaltuk össze.

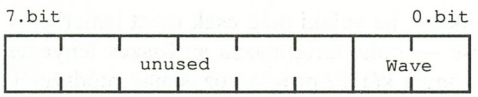
Regiszter $BD: ritmusvezérlő *(Rhytm Control)*



HH (Hi-Hat) : Magas hangú ütőhangszer.
TC (Top-Cymbal): Magas hangú cintányér.
TM (Tom-Tom): Tom-Tom dob.
SD (Snare Drum): Pergődob.
BD (Bass Drum): Basszusdob.
RHY (Rhytm Enable): Ritmuskeltő hangszerek engedélyezése. Ha a bit értéke 1, akkor a fenti ritmushangszerek használata engedélyezett. Ebben az esetben a csatornák száma 6-

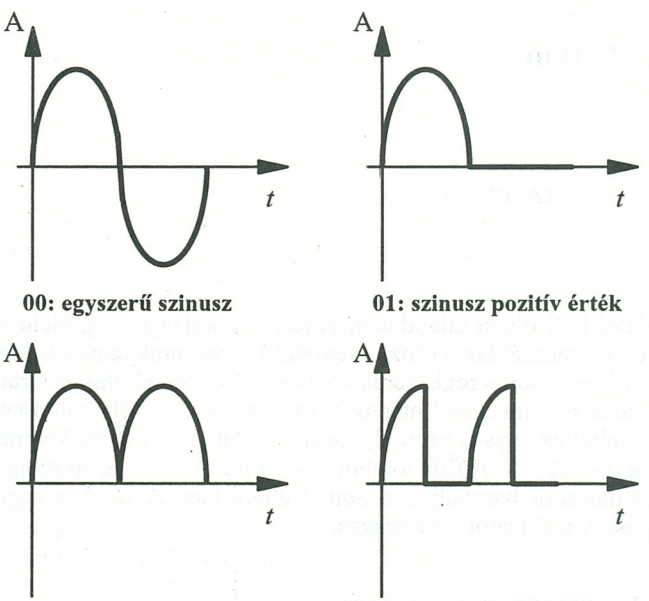
ra csökken, mert a ritmuskeltő eszközök az utolsó három csatorna operátorait használják. Ha a bit 0, akkor mind a 9 csatorna a megszokott módon használható. A programozáskor figyelni kell arra, hogy a ritmuskeltő eszközök használatakor a 6-os, 7-es és 8-as csatorna kikapcsolt állapotban legyen (*KEY ON=0*). A csatorna többi paramétere (ADSR) a megfelelő értékeket tartalmazhatja.
VD (*Vibrato Depth*): A vibrátor mélysége. Ha a bit 0, akkor a vibrátor mélysége 7, ha 1, akkor 14. A vibrátort a *$20 effektusok* regiszter 6. bitje kapcsolja be.
AMD (*AM Depth*): Az amplitúdómoduláció mélysége. Ha a bit 0, akkor az AM mélység 1 dB, egyébként 4.8 dB. Az amplitúdómodulációt a *$20 effektusok* regiszter 7. bitje kapcsolja be.

Regiszter $E0..$F5: hullámforma-kiválasztás *(Select Waveform Type)*



A regiszter alsó két bitje az aktuális hullámalakot határozza meg. Átállításának csak abban az esetben van hatása, ha a *$01 hullámforma-kiválasztás engedélyezése* regiszter 5. bitjét 1-be állítottuk. Az ADLIB a *31. ábrán* feltüntetett négy hullámformát tudja generálni.

A regiszter dupla operátoros.



00: egyszerű szinusz **01: szinusz pozitív érték**

10: szinusz abszolút érték **11: 50%-os fázishasított abszolút szinusz**

31. ábra. A kimenet szabványos jelalakjai

3.3. Az ADLIB kártya programozása

Most, hogy megismertük a kártya regisztereit, elkezdődhet a programozás. Kezdetben seregnyi kérdés vetődik fel a programozóban, körülbelül ilyesfélék:

- Miképpen lehet eldönteni, hogy van-e ADLIB kártya a gépben?
- Miképpen lehet a kártyát inicializálni?
- Mi a módja az egyszerű hangkeltésnek?
- Miképpen lehet a csatornákat egyenként vagy együtt használni?
- Melyek a legfontosabb szabályok, amelyeket be kell tartani?

Ezek a kérdések nagyon fontosak, különösen akkor, ha valaki még csak most ismerkedik a hangkártyával. A regiszterkészlet ismertetése — noha tartalmazza az összes lényeges információt — önmagában még nem mutatja meg a kártya programozásának módszerét. Egyébként is, bármiféle hardverelem programozását leginkább működő, a gyakorlatból vett egyszerű példák alapján lehet igazán megtanulni. Ezért most következzék egy rövid *unit* (egység), amelyben az alapvető feladatok elvégzéséhez szükséges egyszerű eljárások találhatók meg. Mivel ez az első igazi példaprogram, az eljárások többségét itt még Pascalban írtuk meg, de a későbbiekben lassan áttérünk az Assemblerre.

3.3.1. Az ADLIB unit

```
unit adlib; { ADLIB.PAS }

interface
```

Elsőként definiálni kell olyan állandókat, amelyek alapján a megfelelő hangokhoz tartozó értékeket és a csatornákhoz tartozó megfelelő regiszterek címeit ki tudjuk számítani. A *cnum* tömb a kétoperátoros regiszterek első operátorának eltolásait tartalmazza. A második operátor címe mindig 3-mal nagyobb, mint az elsőé, ezért külön nem érdemes tárolni. A *cnum* tömböt az egység azon eljárásai használják, amelyek kétoperátoros regisztereket állítanak be. A *Freq_Tab* tömböt — amint a neve is mutatja — egy oktáv hangjainak előállításakor használjuk a *Sound* eljárásban. A tömb a regiszterkészletnél ismertetett alapoktáv táblázatot tartalmazza.

```
const
      { Az 1. operátor eltolásai          }
    cnum : array[ 0..08] of byte=
           ($00,$01,$02,$08,$09,$0a,$10,$11,$12);
```

```
      { Frekvenciatáblázat egy oktávhoz }
Freq_Tab : array[ 0..11] of word=
           ($157,$16b,$181,$198,$1b0,$1ca,
            $1e5,$202,$220,$241,$263,$287);
```

A kártya regisztereinek írásakor előfordulhat, hogy egy-egy regisztert nem kell teljes egészében módosítani, hanem csak bizonyos biteket vagy bitcsoportokat kell átállítani. Mivel az ADLIB kártya regiszterei csak írhatók, ezért az ilyen műveleteknél a regiszter aktuális értéke nem hozzáférhető. Ezért létrehoztunk egy *Reg* nevű, 256 bájt elemű tömböt, amely árnyéktárként tartalmazza az éppen aktuális beállításokat. A *WriteReg* eljárás (amely egy regisztert állít be) gondoskodik arról, hogy egy regiszter beírásakor az árnyéktárba is belekerüljön az az érték, amelyet a kártyára küldünk.

```
var

      { Árnyéktár az ADLIB regiszterei számára }
    Reg : array[ 0..255] of byte;
```

A változók után következzenek az eljárások!

```
{ Egy ADLIB regiszter közvetlen beírása }
procedure WriteReg(regnum,value:byte);
{ Kétoperátoros regiszter beírása }
procedure DoubleOperator(regnum,value:byte);
{ Az ADLIB tesztelése }
function  TestADLIB:boolean;
{ Az ADLIB inicializálása }
procedure ResetADLIB;


{ Egy csatorna hangerejének beállítása }
procedure Volume(ch,vol:byte);
{ Egy csatorna ADSR generátorának beállítása }
procedure ADSR(ch:byte; ADSR:word);
{ Egy csatorna frekvenciaszorzójának beállítása }
procedure FreqMultiple(ch,fmul:byte);
{ Egy csatorna amplitúdómodulációjának beállítása }
procedure SetAM(ch:byte; amt,amd:boolean);
{ Egy csatorna frekvenciamodulációjának beállítása }
procedure SetFM(ch:byte; fmt,fmd:boolean);
{ Egy csatorna hullámformájának beállítása }
procedure SetWave(ch,wave:byte);
{ Egy csatorna kikapcsolása }
procedure Off(ch:byte);
{ Egy csatornán adott hang kiadása }
procedure Sound(ch,octave,note:byte);

implementation
```

```
procedure WriteReg(regnum,value:byte);                    assembler;
asm
  mov  dx,$388          { ADLIB I/O báziscím    }
  mov  al,regnum
  out  dx,al            { A regiszter száma     }
  mov  cx,6
@wait1:
  in   al,$00
  loop @wait1           { Első várakozás        }
  inc  dx               { DX=$389 (adat)        }
  mov  al,value
  out  dx,al            { A regiszter értéke    }
  mov  cx,35
@wait2:
  in   al,$00
  loop @wait2           { Második várakozás     }
  lea  di,Reg           { DI: az árnyéktár címe }
  mov  bl,regnum
  xor  bh,bh            { BX: a regiszter száma }
  mov  [bx+di],al       { Beírás az árnyéktárba }
end;
```

WriteReg: Az eljárás a *regnum* számú regiszterbe beírja a *value* értéket. Az adatot későbbi felhasználás céljából a *Reg* tömbben is elhelyezi. Gyakorlatilag ez az eljárás megegyezik a fejezet elején ismertetettel, kiegészítve az árnyékmemória megfelelő beállításával. Az egység további rutinjai mind ezen keresztül állítják a regisztereket, így az árnyéktár minden esetben az aktuális állapotot fogja tükrözni. Az eljárásban szereplő időzítési konstansok — a `mov cx,6` és a `mov cx,35` — gyors gépek esetében esetleg módosításra szorulhatnak. ;–(

```
procedure DoubleOperator(regnum,value:byte);
begin
 WriteReg(regnum,value);        { Az 1. operátor beállítása }
 WriteReg(regnum+3,value);      { A  2. operátor beállítása }
end;
```

DoubleOperator: Kétoperátoros regiszterek beállítása. Az eljárás egyszerűen meghívja a *WriteReg* eljárást a megfelelő regiszterszámokkal.

```
function TestADLIB:boolean;
var w : word;
begin
 TestADLIB:=false;
 w:=$0000;
 WriteReg($04,$60);       { A timer #1 inicializálása }
 WriteReg($02,$00);       { Timer #1 adat = 00 (256)  }
 WriteReg($04,$01);       { A számlálás elindítása    }
 if port[$388] and $40=$40 then exit;
 repeat      { Várakozás a timer #1 lejártára }
```

```
  inc(w);
 until (port[ $388] and $40=$40) or (w=$ffff);
 TestADLIB:=w<>$ffff;
end;
```

TestADLIB: A kártya tesztelése. Az eljárás teszteli, hogy a gépben van-e ADLIB (vagy ADLIB kompatíbilis) hangkártya. A teszteléshez az első időzítőt használjuk fel. Inicializáljuk, majd elindítjuk a számlálást az első időzítővel. Ha a számlálás rögtön az indítás után véget ért, vagy a megadott 65535 lépéses ciklus lefutása alatt sem ért véget, akkor nincs ADLIB kártya a gépben. A módszer eléggé egyszerű, ezért nem teljesen biztonságos. Az ADLIB báziscímén előfordulhat más hardverkiegészítő is, ami esetleg hibás tesztelésre vezet. Teljesen biztonságosan csak a számlálás közben eltelt idő pontos mérésével és mindkét számláló többszöri vizsgálatával lehet eldönteni, hogy a kártya valóban jelen van-e.

```
procedure ResetADLIB;
var i:integer;
begin
 { Minden ADLIB regiszter nullázása }
 for i:=1 to $f5 do writereg(i,$00);
end;
```

ResetADLIB: A kártya inicializálása (kiindulási állapotba hozása). Az eljárás minden egyes ADLIB regisztert nulláz, így az összes előző paraméter elvész, a kimenetek tiltott állapotba kerülnek. Nem állítjuk, hogy mindezt nem lehetne elegánsabban megoldani, de a gyakorlatban a módszer jól működik.

```
procedure Volume(ch,vol:byte);
begin
 DoubleOperator($40+cnum[ ch] ,(not (vol and 63)) or $C0 );
end;
```

Volume: A hangerő beállítása az adott csatornához. A *ch* számú csatornán beállítja a *vol* értékű hangerőt. A *ch* értéke 0..8 között lehet, míg a *vol 0..63* között változhat. A hangerő *vol=63* esetén a legnagyobb. Az eljárás az oktávonkénti jelszintcsökkenést is beállítja (or $C0) 6 dB/oktáv értékkel. Ha ezen módosítani akarunk, akkor a $C0 értéket kell megváltoztatni. Figyelni kell arra, hogy ennek a számnak csak a legfelső két bitjét szabad módosítani, a többi legyen mindig 0.

```
procedure ADSR(ch:byte; ADSR:word);
begin
 { ADSR attack/decay összetevő }
 DoubleOperator($60+cnum[ ch] ,hi(ADSR));
 { ADSR sustain/release összetevő}
 DoubleOperator($80+cnum[ ch] ,lo(ADSR));
end;
```

ADSR: Burkológörbe-definiálás. Az eljárás a *ch* számú csatornára az *ADSR* 16 bites értékkel megadott burkológörbét állítja be. A paraméter bitjei a következőképpen definiáltak:

0.. 3. bit: R (*Release*) elengedési idő
4.. 7. bit: S (*Sustain*) kitartási szint
8..11. bit: D (*Decay*) lecsengési idő
12..15. bit: A (*Attack*) felfutási idő

A kitartás idejét itt nem lehet definiálni !

```
procedure FreqMultiple(ch,fmul:byte);
begin
 { Frekvenciaszorzó-beállítás, de a többi bitet
   az árnyéktárból kell kivenni! }
 DoubleOperator
 ($20+cnum[ ch],(fmul and $0f)or(Reg[ $20+cnum[ ch]] and $f0));
end;
```

FreqMultiple: A frekvenciaszorzó állandó beállítása. Az eljárás a *ch* csatorna frekvenciaszorzóját állítja be ($20 regiszter alsó négy bit) az *fmul* értékre. Csak az *fmul* alsó négy bitje értelmezett.

```
procedure SetAM(ch:byte; amt,amd:boolean);
var b:byte;
begin
 if amd then b:=$80 else b:=$00;
 { Az AM mélységének beállítása }
 WriteReg($bd,(Reg[ $bd] and $7f) or b);
 if amt then b:=$80 else b:=$00;
 { Az adott csatornán az AM bekapcsolása }
 DoubleOperator($20+cnum[ ch],(Reg[ $20+cnum[ ch]] and $7f) or b);
end;
```

SetAM: Az amplitúdómoduláció beállítása egy csatornán. A *ch* a csatorna számát adja meg, az *amt* a moduláció ki-be kapcsolását vezérli, míg az *amd* a moduláció mélységét állítja be. A moduláció mélysége nem állítható csatornánként, hanem minden csatornára azonosan hat, ezért az átállítás után a többi csatorna AM mélysége is megváltozik.

```
procedure SetFM(ch:byte; fmt,fmd:boolean);
var b:byte;
begin
 if fmd then b:=$40 else b:=$00;
 { Az FM mélységének beállítása }
 WriteReg($bd,(Reg[ $bd] and $bf) or b);
 if fmt then b:=$40 else b:=$00;
 { Az adott csatornán az FM bekapcsolása }
 DoubleOperator($20+cnum[ ch],(Reg[ $20+cnum[ ch]] and $bf) or b);
end;
```

SetFM: A frekvenciamoduláció (vibráció) beállítása egy csatornán. A *ch* a csatorna számát adja meg, az *fmt* a moduláció ki-be kapcsolását vezérli, míg az *fmd* a moduláció mélységét állítja be. A moduláció mélysége nem állítható csatornánként, hanem minden csatornára azonosan hat, ezért az átállítás után a többi csatorna FM mélysége is megváltozik.

```
procedure SetWave(ch,wave:byte);
begin
 DoubleOperator($e0+cnum[ ch],wave and 3);
end;
```

SetWave: Egy csatorna hullámformájának beállítása. Az eljárás a *ch* számú csatorna kimenő hullámformáját beállítja a *wave* alsó két bitje szerint. A hullámforma beállítása csak akkor lehetséges, ha a *$01 Enable Waveform Control* regiszter 5. bitjével a hullámforma-vezérlés engedélyezve van.

```
procedure Off(ch:byte);
begin
 WriteReg($b0+ch,Reg[ $b0+ch] and $df);
end;
```

Off: Egy csatorna kimenetének tiltása, a hang kikapcsolása. Az eljárás a *ch* számú csatorna kimenetét tiltja, a hang azonnal — az ADSR ciklus állapotától függetlenül — elhallgat.

```
procedure Sound(ch,octave,note:byte);
begin
 Off(ch);
 WriteReg($a0+ch,lo(Freq_Tab[ note] ));
 WriteReg
 ($b0+ch,(hi(Freq_Tab[ note] ) and 3) or (octave shl 2) or $20);
end;
```

Sound: Egy csatorna megszólaltatása. Az eljárás a *ch* számú csatornán az *octave* oktávú, *note* számú hangot adja ki. Az *octave* 0..7 között lehet, a *note* pedig egy 0..11 közötti szám, amely megfelel egy oktáv hangjainak. A hangok számozása a következő:

Hang	Kód
C	00
C#	01
D	02
D#	03
E	04
F	05
F#	06
G	07
G#	08
A	09
A#	10
H	11

Az eljárás önmagában még nem elegendő egy hang kiadásához. A hang egyéb paramétereit a fent ismertetett eljárásokkal kell beállítani, még a *Sound* eljárás használata előtt.

A unit a program indulásakor nullázza az árnyéktárat:

```
BEGIN
 { Az ADLIB inicializálása }
 fillchar(Reg,sizeof(Reg),$00);
END. of unit
```

Az egység használatakor figyelni kell arra, hogy az eljárások nem ellenőrzik, vajon a paraméterek helyesek-e. A legfontosabb a csatornaszám helyes megadása! A többi paramétert az adott rutin általában bitenként maszkolja, de helytelen csatornaszám esetén a *cnum* tömb indexelése hibás lehet. Ezért a csatornaszám mindig essen a 0..8 intervallumba!

Mit is tesz ez a unit? Gyakorlatilag megtalálhatók benne az ADLIB csatornáinak programozásához szükséges legfontosabb rutinok, így alkalmas egyszerű főprogramból történő hangkeltésre. Az igazi profi alkalmazásokban persze a hangkeltés nem lehet fő feladat, ezért ez az egység inkább bemutató-oktató jellegű, mintsem hogy komolyabb programban fel lehetne használni. A *DoubleOperator* eljárás mindkét operátort azonosan állítja be, így sok lehetőséget nem lehet kihasználni, de az egyszerűség kedvéért jobbnak láttuk így megoldani a kétoperátoros regiszterek beállítását. (A valódi és igazán hatékony programok a hangkártyát megszakítás alatt kezelik, így a hangok kiadása a processzornak csak minimális idejét veszi el.) Nézzük meg néhány rövid példaprogram segítségével az egység rutinjainak használatát!

3.3.2. Egyetlen hang kiadása

Ez a rövid program csupán egyetlen hangot szólaltat meg az ADLIB 0-s csatornáján. A program bemutatja, melyek azok az alapvető paraméterek, amelyeket feltétlenül be kell állítani ahhoz, hogy a kártya hangot adjon.

```
uses crt,adlib;          { Használjuk az előző egységet }

BEGIN
 { Kilépés és hibaüzenet, ha a gépben nincs ADLIB hangkártya }
 if not TestAdlib then
  begin
   writeln('Nincs ADLIB kártya a gépben !!');
   halt;
  end;

 { Inicializáljuk a kártyát (nullázzuk a regisztereket) }
 ResetADLIB;
```

```
 { A hullámforma-vezérlés engedélyezése }
 WriteReg($01,$20);

 SetWave($00,$00);        { Egyszerű szinusz                    }
 Volume($00,$3f);         { Maximális hangerő                   }
 ADSR($00,$f1ff);         { Attack: 15 , Decay: 01              }
                          { Release: 15 , Sustain: 15           }
 FreqMultiple($00,$01);   { Frekvenciaszorzó: a saját oktáv     }
 Sound($00,$03,$00);      { A hangot kiadjuk                    }
END. of program
```

A legfontosabb a burkológörbe szakaszainak helyes megadása. Ha ezt hibásan adjuk meg, akkor általában csak egyetlen kattanás hallható, vagy még az sem. Érdemes megfigyelni, hogy a fenti példában a *Sustain* szint gyakorlatilag a legkisebb, így a *Release* rész tulajdonképpen érdektelen. (Lásd a cimbalom burkológörbéjét!) Az egység nem használja ki a kitartási szint idejének vezérlési lehetőségét (*GATE* bit) – ehhez komolyabb programot kellene írni, amely megszakítás alatt figyeli az idő lejártát. Érdemes kipróbálni a paraméterek módosítását. Érdekes hanghatások érhetők el például a hullámforma átváltásával vagy a visszacsatolás megváltoztatásával.

3.3.3. Több csatorna használata

Ez a példa egy C-dúr akkordot szólaltat meg, de egyelőre itt csak egyetlen csatornát használunk. Ennek az lesz az eredménye, hogy a hang kiadásakor elindul egy ADSR ciklus, majd a következő hang ezt a ciklust félbeszakítja és újat kezd. Ez persze csak akkor következik be, ha a hangok kiadása közötti idő kisebb, mint egy ADSR ciklus lefutási ideje. (Ez a jelen esetben fennáll!)

```
uses crt,adlib;

var i : integer;

BEGIN
 { A kártya tesztelése }
 if not TestAdlib then
  begin
   writeln('Nincs ADLIB kártya a gépben !!');
   halt;
  end;
 { Inicializálás }
 ResetADLIB;
 WriteReg($01,$20);
 { Az alapparaméterek beállítása }
 SetWave($00,$02);
 Volume($00,$3f);
 ADSR($00,$f1ff);
```

```
 FreqMultiple($00,$01);
 WriteReg($C0,$02);
 { Az akkord négy hangjának kiadása a 0. csatornán }
 {C } Sound($00,$03,$00); delay(400);
 {E } Sound($00,$03,$04); delay(400);
 {G } Sound($00,$03,$07); delay(400);
 {C'} Sound($00,$04,$00); delay(400);
END.
```

A hangok kiadására ebben az esetben 0.4 másodpercenként kerül sor. Az előbbiekben megadott ADSR ciklus ennél hosszabb, ezért az új hang mindig félbeszakítja az előzőt. Ha az ADLIB unitban a *Sound* eljárásból eltávolítjuk az *Off* eljárás meghívását, akkor a csatorna *KEY ON* bitje nem törlődik, ezért csak a legelső hang szólal meg. A *Sound* többi hívása beállítja ugyan az új paramétereket, de új hang már nem szólal meg. Mindezzel csupán a *KEY ON* bit helyes használatát szerettük volna bemutatni!

Több csatorna együttes használatával a hangzás sokkal „életszerűbb" lesz. A most következő példában a négy hang kiadásához külön-külön csatornákat használunk fel, így mindegyik ADSR ciklus teljesen le tud futni. A kezdőértékek megadásánál természetesen minden egyes csatorna megfelelő paraméterét be kell állítani.

```
uses crt,adlib;

var i : integer;

BEGIN
 { Tesztelés }
 if not TestAdlib then
  begin
   writeln('Nincs ADLIB kártya a gépben');
   halt;
  end;
 { Inicializálás }
 ResetADLIB;
 WriteReg($01,$20);
 { Azonos paraméterek megadása az első négy csatorna számára }
 for i:=0 to 3 do
  begin
   SetWave(i,$02);
   Volume(i,$3f);
   ADSR(i,$f1ff);
   FreqMultiple(i,$01);
  end;
 { A négy hang kiadása négy különböző csatornára }
 {C } Sound($00,$03,$00); delay(400);
 {E } Sound($01,$03,$04); delay(400);
 {G } Sound($02,$03,$07); delay(400);
```

```
  {C'} Sound($03,$04,$00); delay(400);
END. of program
```

3.3.4. A ritmushangszerek használata

A kártya utolsó három csatornáján lehetőség van különböző ütőhangszerek használatára (lásd a *$BD rhytm control* regisztert). Az operátorok viszont ebben az esetben valamivel bonyolultabb módon vannak összekapcsolva. A kártyán összesen 9 csatorna, azaz 18 operátor található — a ritmushangszerek ebből 6 operátort használnak fel. Így marad meg a 12 operátor — 6 csatorna — a normál hangok kiadásához. A további operátorok az öt ütőhangszerhez tartoznak, a basszusdobhoz kettő, a többi hangszerhez pedig egy-egy. Ezeknek a hangszereknek a megszólalását nem a *KEY ON* bit vezérli, hanem a *$BD rhytm control* regiszter megfelelő bitjének 1-be állításakor adja ki a hangot a kártya. Ezen biteknél — a *KEY ON* bithez hasonlóan — a kártya a 0→1 átmenetet tekinti a hangkiadás kezdetének (pozitív élvezérlés). A következő példa ezt mutatja be.

```
uses crt,adlib;

var  i : integer;

const
   vol : byte = $1f;            { A kezdeti hangerő              }

procedure Hit;                  { Egyetlen hang leütése          }
begin
 writereg($bd,$20);             { Az összes bit törlése,         }
 writereg($bd,$3f);             { majd az összes bit beállítása }
end;

procedure Snare(n:byte);        { A pergetett dob megvalósítása}
var i,j : integer;
begin
 for j:=1 to n do
  for i:=1 to 6 do
   begin
    Hit;                        { Egy hang kiadása               }
    delay(i*7+10);              { Növekvő szünetek               }
   end;
end;

procedure Short(n:byte);        { Egy rövid ritmus (ti) kiadása}
var i : integer;
begin
 for i:=1 to n do
  begin
```

```
    Hit;                          { Leütés                    }
    delay(200);                   { 0.2 másodperces várakozás }
  end;
end;

proce dure Long(n:byte);          { Egy hosszú ritmus (tá) kiadása}
var i : integer;
begin
 for i:=1 to n do
  begin
    Hit;                          { Leütés                    }
    delay(500);                   { 0.5 másodperces várakozás }
  end;
end;

BEGIN
 clrscr;
 { Tesztelés }
 if not TestAdlib then
  begin
    writeln('Nincs ADLIB kártya a gépben');
    halt;
  end;

 { Inicializálás }
 ResetADLIB;
 WriteReg($01,$20);

 WriteReg($bd,$20);

 { A paraméterek beállítása }
 for i:=6 to 8 do
  begin
    SetWave(i,$02);          { Hullámforma      }
    Volume(i,$1f);           { Hangerő          }
    ADSR(i,$f8ff);           { Burkológörbe     }
  end;

 Writeln('Bal, bal, ... bal, jobb, bal .... :-)');
 { Itt kezdődik a dobszó ... }
 repeat
  Snare(1);  Short(3);
  Snare(1);  Short(3);
  Snare(1);  Short(1);
  Snare(1);  Short(1);
  Snare(1);  Short(3);
  { A hangerő folyamatosan nő }
  if vol<$3f then inc(vol,8);
  for i:=6 to 8 do Volume(i,vol);
```

```
until keypressed;
{ Egy billentyűleütésre kilép }
while keypressed do readkey;

END. of program
```

A dob megszólalásáért a *hit* eljárás felelős. Az eljárásban a *$BD rhytm control* regiszter bitjeit állítjuk. A *hit* először 0-ba állítja ($20) minden egyes ritmuskeltő hangszer bitjét, majd az összeset ismét 1-be. Ezáltal minden egyes eszköz ad hangot, így egy összetett hangzást kapunk eredményül, amely leginkább a pergődob hangjára hasonlít. A *snare* egy pergő-hangsort állít elő, a *short* egy rövid kiütést (ti), a *long* pedig egy hosszabbat (tá). A program fő ciklusa ezeket az eljárásokat hívja, így kapjuk az egyszerű ritmust. Bármely billentyű lenyomására a program leáll. Érdemes megpróbálkozni az ADSR egyes szakaszainak átállításával; igen érdekes hangzások állíthatók elő!

3.4. Megszakításvezérelt lejátszóprogram

Ha olyan programot írunk, amely hangokat is kiad — tipikusan ilyen egy játékprogram —, akkor a hanggenerálás csupán egy mellékes feladat, éppen ezért minimális időt lehet csak ráfordítani. Gyakorlatilag a hanggenerálásnak a „háttérben", szinte észrevétlenül kell futnia. Ezt megszakításokkal tudjuk a legkönnyebben megoldani. A következőkben egy megszakításvezérelt zenelejátszó programot ismertetünk, amely alkalmas több csatorna egyidejű megszólaltatására, és nagyméretékben kihasználja az ADLIB lehetőségeit. Működéséről egyelőre annyit, hogy a timer megszakítást felhasználva bizonyos időegységenként megvizsgálja a zeneadathalmaz következő eseményét, és ettől függően állítja be az ADLIB egyes regisztereit. Egy időegységnek definiáljuk a két timer megszakítás között eltelt időt. Egyszerűbben fogalmazva: *a végrehajtás egy megszakításkor egy időegységnyit lép előre.* Fontos még tisztáznunk a *fizikai* és a *logikai* hangcsatorna fogalmát. Fizikai hangcsatornán az ADLIB valamely hangcsatornáját értjük, míg logikai hangcsatornán az éppen lejátszott zeneadathalmazt lejátszó programot és annak adatait. Ez majd a forrásszövegnél érthetőbbé válik. Egyelőre elég annyi, hogy a logikai csatorna nem feltétlenül csak egyetlen fizikai csatorna megszólaltatására képes! A program az előző példáknál kissé terjedelmesebb, és jórészt Assembler részekből áll, ezért megértése talán nehezebb lesz, de semmiképpen nem lehetetlen!

Programozás szempontjából a zenét — igencsak leegyszerűsítve — felfoghatjuk úgy, mint időben egymás után következő elemi eseményeket, amelyek rendre megváltoztatják a hangkártya állapotát (belső regisztereit). A különböző események között meghatározott idők (adott időegységek) telnek el. Ha például egyszerűen a skála egy oktávjának hangjait akarjuk kiadni, akkor ezt a következő eseményekkel írhatjuk le:

1. A **C** hang kiadása
2. Várakozás
3. A **D** hang kiadása
4. Várakozás
5. Az **E** hang kiadása
6. Várakozás

   . . . így tovább minden hanggal . . .

n. Befejezés

A hangkártya a hang kiadása után (a *KEY ON* bit beállítása) nem igényli a processzort, azaz a várakozás idejében — amíg egy újabb hang sorra nem kerül — bármit lehet tenni. Ezzel máris definiáltunk két elemi eseményt: a hangkiadást és a várakozást.

Mi más lehet még elemi esemény? Nos, ez teljes mértékben tőlünk függ, hiszen egy lejátszóprogram lehetőségeit annak írója szabja meg. Azt is mondhatnánk, hogy egy új programozási nyelvet definiálunk, amelynek az utasításait mi magunk határozzuk meg. :–) A példában a lehetőségek számát igyekeztünk minimálisra csökkenteni anélkül, hogy ez a teljesítmény rovására ment volna. Így talán a programban definiált események kevésnek tűnnek, de a bővítésnek nincs semmi akadálya. Most mindenesetre nézzük meg a definiált elemi események — ha úgy tetszik, zeneparancsok — pontos megadását, azután pedig azt, hogy ezekből hogyan lesz lejátszásra alkalmas zenefájl!

3.4.1. Események

1. Hangjegy kiadása:

A legelemibb esemény a hangkiadás, hiszen ez generálja a hangot. Paraméterei a következők lehetnek: oktávszám, hangjegy, időtartam. Tágabb értelemben az időtartamot — a kitartási időtől függően — két további részre lehet osztani: az első szakaszban a hang a *sustain* szinten kitartva marad (*GATE=1*); a második szakaszban pedig a hang elengedi a kitartási szintet, és elkezdődik a *release* ciklus (*GATE=0*).
A parancs definíciója:

```
n o t  [tr]
```

Az *n* a hangjegy nevét adja meg (*note*), az *o* az oktáv számát (*octave*), a *t* pedig a hang idejét (*time*). Ha a *tr* paramétert is megadjuk, akkor a hang időtartama két részre osztott: a *t* a kitartás idejét, míg a *tr* az elengedés idejét adja meg. Pl.:

```
C 3 20              ; a C hang a 3. oktávban 20 időegységig
```

```
D# 2 8020 0010      ; a Disz hang a 2. oktávban
                    ; 20 időegységig kitartva
                    ; 10 időegység alatt elengedve
```

A későbbiekben majd kitérünk az utasítások részletes szintaktikájára, most egyelőre csak annyit, hogy a számok hexadecimális számrendszerben értendők, és a *tr* paraméter meglétét a *t* paraméter 15. bitje jelzi — ezért szerepel itt 8020 (a 15. bit 1). A hangokhoz az alábbi kulcsszavakat rendeltük:

```
C, C#, D, D#, E, F, G, G#, A, A#, H
```

2. Hangkártyaregiszter beállítása:

Ez az alapművelet mindenképpen fontos, hiszen bármelyik ADLIB regiszternek elérhetőnek kell lennie. Ennek a parancsnak logikusan két paramétere van: a beállítandó regiszter száma és a regiszter értéke. Mivel azonban egyetlen hang kiadásához általában több regisztert kell módosítanunk, érdemes ezt a parancsot úgy definiálnunk, hogy bármennyi regisztert át lehessen írni vele.
A parancs definíciója:

```
regset
 rn rv : rn rv : rn rv
.
```

Az *rn* az adott regiszter számát *(register number)*, az *rv* pedig a regiszterbe írandó értéket *(register value)* jelenti. Pl.:

```
regset
  01 20 : 04 60 : 04 80
.
```

3. Hangok közötti szünet:

A hangok között szükség lehet bizonyos üres idő beiktatására, amikor nincs semmiféle hangkiadás. Erre szolgál a szünet parancs, amelynek paramétere logikusan a szünet ideje; ezt időegységben adjuk meg.
A parancs definíciója:

```
pause time
```

A *time* az időegységek számát jelzi, de a legfelső bitnek kitüntetett szerepe van: ez vezérli az előző hang kikapcsolását. Ha ezt a bitet 1-be állítjuk, akkor a szünet kezdetekor az éppen szóló hang rögtön elhallgat *(KEY ON=0)*, egyébként pedig szabadon lecsenghet. Pl.:

```
pause 30          ; 30 időegységes szünet
pause 8030        ; 30 időegységes szünet, a hang kikapcsolása
```

4. Hangsorok ismétlése:

Gyakran előfordul, hogy a zene valamely szakaszát néhányszor ismételni kell. Ennek megvalósításához definiáltuk az ún. *repeatereket* (ismétlőket), amelyek képesek az adathalmaz bizonyos szakaszainak megadott számú ismétlésére. Egy-egy logikai csatornához négy *repeatert* definiáltunk.
A parancsok definíciója:

```
rpt r n
  ciklusmag
djnz n
```

Az *rpt* beállítja az *r* számú *repeater* ismétlési számát *n*-re, míg a *djnz* visszaugrik a ciklus elejére, és csökkenti a számlálót. Hogy mindez hogyan zajlik le, arra majd később részletesen kitérünk. Pl.:

```
rpt 0 5
 C 2 10 : D 2 10 : E 20 10
djnz 0       ; A C-D-E hangok ismétlése ötször a 0. repeaterrel
```

5. Fizikai csatorna megváltoztatása:

A zeneadathalmazban több hang kiadásakor szükség lehet több fizikai csatornára. Ilyenkor csatornaszámot váltunk. A csatornaváltást egyszerűen egy új fizikai csatornaszám megadásával végezzük el.
A parancs definíciója:

```
chnum cn
```

A *cn* a csatorna számát adja meg, amely az ADLIB esetében 0..8 lehet. Pl.:

```
chnum 0
C 3 10            ; a C hang a  0. csatornán szólal meg
chnum 1
D 3 10            ; a D hang az 1. csatornán szólal meg
chnum 2
E 3 10            ; az E hang a 2. csatornán szólal meg
```

6. Vezérlésátadás, a végrehajtási pont megváltoztatása:

Az adathalmazt nem szükségszerűen sorban kell végrehajtani, hiszen előfordulhatnak ismétlések, gyakori hangsorok. Ekkor célszerű lenne beavatkozni a végrehajtás menetébe. Erre a célra definiáltuk a vezérlésátadó utasításokat (az *assemblyből* kölcsönözve).

A parancsok definíciója:

```
jmp  address
call address
ret
```

Az *address* egy abszolút címet jelent az adathalmazban, gyakorlatilag az elejétől számított valamekkora eltolás.

7. A lejátszás befejezése, a logikai csatorna felszabadítása:

A lejátszás végét is jelezni kell egy paranccsal, ami leállítja a zeneadathalmaz feldolgozását, és kikapcsolja a logikai csatornát, ezután már nincs további hangkiadás.
A parancs definíciója:

```
end
```

Ezzel a néhány paranccsal tehát definiáltunk egy olyan egyszerű nyelvet, amely alkalmas zene lejátszására. Ily módon a zenét egy szövegfájlban lehet megadni, amelyben a fenti parancsok használhatók. Csakhogy ettől még a hangkártya nem fog megszólalni. Ehhez ugyanis készíteni kell két programot. Az egyiknek a fentiek alapján elő kell állítania egy olyan adatfájlt, amelyet a másik képes feldolgozni és lejátszani. Az első program tehát egy fordítóprogram (*compiler*), míg a második maga a lejátszó (*player*).

3.4.2. Az adathalmaz formátuma

Az adathalmaz célszerűen egy bináris fájl, amely egyszerűen bájtokat tartalmaz. Ezek a fenti parancsok kódjai és adatai. Minden parancsnak van egy adott kódja, és ettől függően lehet valamennyi paramétere. A következőkben a parancsok lefordított formátumát ismertetjük. Ez tulajdonképpen az a formátum, amelyet a lejátszó értelmez és végrehajt.

1. ***Hangok:*** a hangokat a kód 7. bitje különbözteti meg az egyéb parancsoktól. Ha a 7. bit 0, akkor az adott kód egy hangot jelent. A többi bitet a következőképpen definiáltuk:

bit: 0–3: a hangjegy kódja (0..11, azaz C ... H)
4–7: az oktáv száma (0..7)

Az első bájt tehát megadja a hangjegyet és az oktávot. A következő word az időtartamot definiálja. Itt az a szabály, hogy ha a 15. bit 1, akkor van még egy word, és ilyenkor az

időtartam kettéoszlik kitartási és elengedési időre, míg a 15. bit 0 állapota esetén nincs kitartás. Pl.:

```
C 0 20              ; lefordítva:     00 20 00
D 0 8020 10         ; lefordítva:     02 20 80 10 00
E 2 09              ; lefordítva:     24 09 00
F 3 8005 05         ; lefordítva:     35 05 80 05 00
```

Egyetlen hangjegy tehát 3 vagy 5 bájt hosszú lehet az adatfájlban.

2. End: Ez az utasítás egyetlen bájt, nincs semmilyen paramétere. A parancs kódja $80.

```
end                 ; lefordítva:     80
```

3. Jmp: Az utasításnak egyetlen word típusú paramétere van, amely megadja az új végrehajtási pontot. A parancs kódja $81.

```
jmp   1E00          ; lefordítva:     81 00 1E
```

4. Call: A *call* — hasonlóan a *jmp*-hoz — egyetlen word típusú paramétert igényel, amely megadja a szubrutin címét. A parancs kódja $82.

```
call  A010          ; lefordítva:     82 10 A0
```

5. Ret: Nincs paramétere, a parancs kódja $83.

```
ret                 ; lefordítva:     83
```

A *call–ret* utasításpár saját vermet használ, amelyet a *player* program definiál. A veremben alapesetben 32 cím helyezhető el, így a szubrutinok egymásba ágyazási mélysége legfeljebb 32 lehet.

6. RegSet: A regiszterek beállításánál több regisztert is meg lehet adni, így az utasítás hossza változó lehet. A beállítás kezdetét a $84 kód jelzi, majd az adatpárok következnek, a sort pedig egy $FFFF érték zárja, mivel nincs $FF számú ADLIB regiszter.

```
regset              ; lefordítva:     84
 01 20 : E0 03      ; lefordítva:     01 20 E0 03
.                   ; lefordítva:     FF FF
```

A : és . jelek használata a fordítóprogram miatt szükséges, a lefordított kódban nincs jelentőségük.

7. ***Rpt:*** A repeater ciklus kezdetét definiáló parancsnak két paramétere van: a *repeater* száma (0..3) és az értéke (1..63). Az utasítás kódja $85.

```
rpt 0 2          ; lefordítva:     85 02
rpt 1 5          ; lefordítva:     85 45
```

A két paramétert el lehet helyezni egyetlen bájtban, ezért az operandus bájt felső két bitje a *repeater* számát, az alsó hat bit pedig a kezdeti értéket tartalmazza. A cikluskezdet címének tárolása a *player* feladata, ez a kódban nem jelentkezik.

8. ***Djnz:*** Az ismétlő ciklus lezáró utasításának csupán a *repeater* számát kell megadni. A parancs kódja $86.

```
djnz  2          ; lefordítva:     86 02
```

Az *rpt–djnz* utasításpár a címet és az ismétlések számát a *player* belső változóiként kezeli, ezért ezeknek az adatok között nincs nyomuk. Figyelni kell arra, hogy a *repeater* száma ugyanaz legyen mindkét utasításnál, egyébként az ugrás címe — amelyet a *djnz* kezdeményez — bizonytalan, ezért hibás lejátszáshoz vezet. ;–(Ugyancsak hibás működéshez vezet a *djnz* használata az *rpt* nélkül. Ha csak az *rpt*-t használjuk, de nincs hozzá tartozó *djnz*, akkor nem történik baj. Ha egy *rpt*-t többször is újra megadunk, akkor mindig a legutolsó definíció lesz az érvényes. Több ismétlési ciklus egymásba ágyazható, ezért van négy különböző *repeater* egy logikai csatornán.

9. ***Pause:*** A szünetek esetében csak a várakozási időt kell megadni egyetlen word típusú paraméterrel. A 15. bit 1-es állapota esetén az éppen szóló hang elhallgat, egyébként szabadon lecseng. A parancs kódja $87.

```
pause 200        ; lefordítva:     87 00 02
pause 8210       ; lefordítva:     87 10 82
```

10. ***ChNum:*** A csatornaváltásnál egyetlen bájtparaméterrel az új csatorna számát kell megadni. A csatorna számának a 0..8 intervallumba kell esnie. A parancs kódja $88.

```
chnum 03         ; lefordítva:     88 03
```

A zeneadatfájl tehát ezeket a parancsokat tartalmazhatja. A lejátszó program az adatfájl utasításait — a mikroprocesszor működésének analógiájára — sorban értelmezi és végrehajtja. A végrehajtás sebességét a megszakítások száma szabja meg, amelyet a programban nagyjából 50 megszakítás/másodperc értékre állítottunk be. Ez alapján az időegységeket 20 ms-os lépésekben lehet megadni (1/50), azaz a legrövidebb időtartam 20 ms, míg a leghosszabb — mivel az időtartamokat csak 15 biten tároljuk — 32768*20 ms, azaz 655.36 másodperc, tehát nagyjából 11 perc. A definíciók szerint a program 8 oktáv 12 hangját tudja kiadni, ez 96 különböző magasságú hangot jelent. Az egyéb paramétereket (hullámforma, burkológörbe, visszacsatolás) a *RegSet* utasítással szabadon be lehet állítani, kivéve a *GATE* bit vezérlését, mert azt a hangjegyek önállóan elvégzik.

3.4.3. A lejátszó (*player*)

A kulcsszavak és az adatformátumok ismertetése után következzék a program, amely a fenti módon létrehozott fájlt képes lejátszani.

```
program player; { PLAYER.PAS }
{$g+}
uses crt,dos;       { A CRT és a DOS egységet használjuk }
```

Típusok

A legelső és legfontosabb lépés, hogy definiálunk egy olyan típust, amely egy logikai csatorna összes adatát képes tárolni. Ez a *ChannelDatas* rekord.

```
type
    ChannelDatas = record
     rpt0,rpt1,rpt2,rpt3      : byte;        { 00 }
     roff0,roff1,roff2,roff3  : word;        { 04 }
     PC     : pointer;                       { 12 }
     SP     : word;                          { 16 }
     flag   : byte;                          { 18 }
     cnum   : byte;                          { 19 }
     Time   : word;                          { 20 }
    end;
```

A rekord elemei a következők:

rpt0...rpt3: A csatorna *repeaterjeinek* számlálói. Ezekbe a mezőkbe kerülnek a számlálók adatai az *rpt* utasítás végrehajtásakor, és ezeket csökkenti a *djnz* utasítás.

Az *rpt* utasítás formátuma miatt csak 6 bitet használunk fel számlálásra, de még ez is 63 ismétlést tesz lehetővé.

roff0...roff3: A csatorna *repeaterjeinek* címtárolói. Amikor a *player* egy *rpt* utasítást hajt végre, akkor az aktuális címet ezekben a mezőkben tárolja. A *djnz* utasítás ez alapján tudja meg, hol van az ismétlési ciklus kezdete.

PC: Az adathalmaz aktuális mutatója. A PC mindig a következő végrehajtandó utasításra mutat. Mivel a PC mutató, ezért a zene adatait célszerűen el lehet helyezni a heapben, majd ráállítani a PC-t. A PC csak az alsó 16 bitet használja adatcímzésre, ezért egy zeneadat legnagyobb hossza 65536 bájt (64K).

SP: Az adathalmaz veremmutatója. Az SP-t a *call* használja a szubrutinok címének tárolására, míg a *ret* innen olvassa ki, hová kell visszatérnie. Az SP nem a címet, hanem csak a címet tároló tömb aktuális elemének címét tárolja. A vermet mindenképpen a főprogram adatszegmensében kell elhelyezni. Jelen esetben az *Stk* tömb jelenti a vermet (lásd később).

flag: Jelzőbitek. Ebben a mezőben különböző állapotbitek tárolhatók. Egyelőre csak a 7. és a 0. bitet használjuk ki. A 7. bit a különálló *sustain/release* ciklusok meglétét jelzi (amikor a hangjegynek két időparamétere van), míg a 0. bit a csatornát kapcsolja ki/be. Ha a 0. bit 1, akkor a lejátszás leáll. Ezt a bitet az *end* utasítás állítja be.

cnum: Az aktuális fizikai csatorna száma. Ez a mező tárolja annak a fizikai csatornának a számát, ahol a lejátszás éppen folyik. Ezt a mezőt a *chnum* utasítással lehet átállítani.

Time: A csatorna időzítője. Ebbe a mezőbe kerül az összes időparaméter. Minden egyes timer megszakítás eggyel csökkenti. Amikor 0-ra csökkent, akkor egy újabb hangot vagy parancsot kell végrehajtani. Ha a flag 7. bitje be van állítva, akkor a *player* tudja, hogy egy újabb időparamétert kell a *Time*-ba tölteni, és a *GATE* bitet törölni kell.

A csatornák *call–ret* utasításpárjainak működéséhez szükséges egy verem, ahol a szubrutinok címét tárolhatjuk. Erre a célra definiálunk egy verem típust:

```
Stack  = array[ 0..31] of word;
```

Ebben a veremben 32 cím tárolható, ezért legfeljebb 32 szubrutin ágyazható egymásba. Amint látható, nincs akadálya annak, hogy ezt megnöveljük, ha szükséges. A veremkezelés a mikroprocesszor veremkezelésével analóg, tehát betöltéskor (*call*) a veremmutató — azaz az SP — csökken, míg kiemeléskor (*ret*) a mutató nő. Ezért az inicializáláskor az SP-t a verem legutolsó szavára kell állítani.

Állandók

```
const
    Base   = $388;              { ADLIB I/O báziscím }

    ChanDataSize = SizeOf(ChannelDatas);
                                { Egy csatornarekord hossza }

    _rpt   = 00;
    _roff  = 04;
    _pc    = 12;
    _sp    = 16;
    _flag  = 18;
    _cnum  = 19;
    _Time  = 20;
```

Ezeket az állandókat az Assemblerben kell használni a logikai csatorna egyes mezőinek indexeléséhez. Az Assembler részben — amint az később látható — az egyes csatornák báziscímét mindig a BX regiszter tartalmazza, így ahhoz hozzáadva a fenti értékeket a megfelelő mező címét kapjuk meg. Így az Assembler olvashatóbb lesz, az egyes indexértékek már nevükkel is mutatják, miről is van szó.

Definiálunk még két inicializált tömböt is; a *FreqTAB* egy oktáv hangjaihoz tartalmazza a megfelelő frekvenciaállandókat, az *OperOff* pedig a kétoperátoros regisztereknél az első operátor eltolását.

```
FreqTAB : array[ 0..11] of word=
    ($2157,$216b,$2181,$2198,$21b0,$21ca,
     $21e5,$2202,$2220,$2241,$2263,$2287);

OperOff : array[ 0..8]  of byte=
    ($00,$01,$02,$08,$09,$0a,10,11,12);
```

Változók

```
var
    Effects : array[ $00..$15] of byte;
    Channel : array[ 0..3] of ChannelDatas;
    Stk     : array[ 0..3] of Stack;
    oldIRQ  : procedure;
```

Effects: Ez a 22 bájt az ADLIB $20...$35 regisztereinek pontos másolatát tartalmazza. Ezekben a regiszterekben találhatók meg a *GATE* bitek, amelyeket a különálló *sustain/release* időciklusok alatt változtatni kell. Az ADLIB regiszterei csak olvashatók, ezért ez a tömb árnyéktárként tartalmazza a $20...$35 regiszterekbe legutoljára

beírt értékeket. A *RegSet* eljárás (lásd később) másolatot készít minden adatról, ha erre a területre írunk.

Channel: A logikai csatornák definiálása. Most csak négy logikai csatornát definiáltunk, de gyakorlatilag tetszőleges számút írhatnánk elő, hiszen egy logikai csatorna bármely fizikai csatornát képes használni (lásd a *chnum* parancsot!). Megítélésünk szerint felesleges a túl sok csatorna használata, kilenc fölé semmiképpen nem érdemes menni, hiszen ennyi fizikai csatorna van.

Stk: A logikai csatornák vermei. Minden csatornához egy-egy verem tartozik, így ugyanannyi veremnek kell lennie, ahány logikai csatorna van.

oldIRQ: Ez a változó tárolja a timer megszakítás előző címét. Azért van rá szükség, hogy kilépéskor az eredeti rutin vissza tudja venni a megszakítás kezelését.

```
i,j,k   : integer;
music   : pointer;
size    : word;
f       : file;
```

A további változók már általános célt szolgálnak. Az *i,j,k* egészek bizonyos ciklusokban szerepelhetnek, a *music* a betöltött zenefájl heapbeli címét tartalmazza. A *size* a betöltött fájl hossza, míg az *f* egy egyszerű, típus nélküli fájlt azonosít. A programban csupán egyet használunk most fel a négy definiált csatorna közül, ezért van csak egyetlen *music* és *size*, de ha valaki több csatornára is ír programot, akkor értelemszerűen újabb változókat kell deklarálnia!

Eljárások

```
procedure RegSet;                                    assembler;
asm
  push  ax
  push  dx                    { A használt regiszterek tárolása }
  push  cx                    { a veremben                      }
  cmp   al,$20
  jc    @set                  { Ugrás, ha a regiszter száma<$20 }
  cmp   al,$36
  jnc   @set                  { Ugrás, ha a regiszter száma>$35 }
  mov   cx,bx
  lea   bx,Effects-$20        { A bx az árnyéktárra mutat       }
  mov   dx,ax
  xor   dh,dh
  add   bx,dx                 { Hozzáadjuk az eltolást          }
  mov   [bx],ah               { Beírjuk az értéket a tömbbe     }
  mov   bx,cx
```

```
@set:
  mov   dx,Base                { dx = ADLIB báziscím          }
  out   dx,al                  { A regiszter számát kiküldjük }
  mov   cx,6
@wait1:
  in    al,$00
  loop  @wait1                 { Várakozás                    }
  inc   dx                     { dx = dx + 1                  }
  mov   al,ah
  out   dx,al                  { A regiszter értékét kiküldjük }
  mov   cl,35
@wait2:
  in    al,$00
  loop  @wait2                 { Várakozás                    }
  pop   cx                     { A regiszterek eredeti értékét }
  pop   dx                     { visszatöltjük a veremből     }
  pop   ax
end;
```

RegSet: Egy ADLIB regiszter beállítása. Az eljárás az AL számú regiszterbe betölti az AH értéket. Ha a regiszter száma a $20...$35 területre esik, akkor az Effects árnyéktárban is tárolja az értéket, hogy a GATE bit állításakor a regiszter többi bitje ne változzon. A programban minden egyes ADLIB regisztert csak ezzel az eljárással lehet állítani, ezért a tárolt értékek mindig korrekten mutatják a legutoljára beírt állapotot. Az eljárás regisztereken keresztül kapja a paramétereit, ezért közvetlenül nem szabad meghívni, csak az Assembler betétekből.

A most következő *PlayNext* eljárás a program leglényegesebb része. A megszakítás akkor hívja meg ezt az eljárást, amikor egy következő parancsot kell lehívni és végrehajtani a zeneadatokból, így a mikroprocesszor utasításlehívó (*FETCH*) ciklusára hasonlít leginkább. Belépéskor a BX regiszternek a megfelelő logikai csatorna adatainak báziscímét kell tartalmaznia, a DS-nek pedig a program adatszegmensére kell mutatnia. (Talán feleslegesnek tűnik, hogy a DS tartalmát is megemlítettük, hiszen egy Pascal program alapesetben sohasem változtatja meg a DS regisztert, így az mindig a program adatszegmensére mutat. Ezt az eljárást azonban egy megszakítás hívja, amelynél nem lehet pontosan tudni, hogy a DS-ben éppen milyen érték van, ezért fontos, hogy a regisztert a hívás előtt biztosan beállítsuk!)

```
procedure PlayNext;                                    assembler;
asm
  push  ES                          { Az ES tárolása a veremben }
  mov   ES,[ bx+_pc+2]              { A PC értéke az ES:SI      }
  mov   si,[ bx+_pc]                { regiszterpárba kerül      }
```

Mint látható, az ES:SI regiszterpár mutat a következő végrehajtandó zenei utasításra. A *@nextcommand* az a hely, ahol az utasításkód lehívása megtörténik. Itt a 7. bit értéke eldönti, hogy hangjegy vagy parancs következik-e. Parancsok esetén ugrás a parancsok

végrehajtására, míg hangjegy esetében ki kell adni egy újabb hangot. A kód már implicit módon (befoglaltan) tartalmazza az oktávszámot (4–6. bit) és a hangjegy számát (alsó 4 bit).

```
@nextcommand:
  segES lodsb                          { A következő kód lehívása }
  test  al,10000000b                   { A 7. bit 1? (parancs?)    }
  jnz   @commands                      { Ugrás, ha parancs         }
  { Hangjegy kiadása következik, mert a 7. bit 0                   }
  mov   di,ax                          { DI=AX                     }
  and   di,$000f                       { DI=a hangjegy száma       }
  shl   di,1                           { DI=DI*2                   }
  add   di,offset FreqTAB              { DI a frekvenciatáblázat   }
                                       { megfelelő elemére mutat   }
  mov   cx,[ di]                       { CX=a hang frekvenciája    }
```

Hangjegy esetén az alsó négy bit alapján meghatározzuk a hanghoz tartozó frekvenciát. A DI-be betöltjük a hangjegy számát; ez mindenképp a 0..11 tartományba esik. A DI-t ezután eltoljuk balra 1 bittel — azaz megszorozzuk kettővel —, mivel a frekvenciatáblázatban word típusú adatok találhatók. A DI-hez hozzáadjuk a táblázat kezdőcímét, így a DI a hangjegynek megfelelő frekvenciaállandóra mutat. Ezt átmenetileg a CX regiszterbe töltjük.

```
  shr   al,2                           { Az oktávszámot a helyére  }
  and   al,00011100b                   { toljuk és kimaszkoljuk,   }
  or    ch,al                          { majd betesszük a CH-ba    }
  mov   ax,$00b0
  add   al,[ bx+_cnum]
  call  RegSet                         { A hangot kikapcsoljuk     }
  mov   al,$a0
  add   al,[ bx+_cnum]
  mov   ah,cl                          { Frekvencia alsó bájt      }
  call  RegSet
  mov   al,$b0
  add   al,[ bx+_cnum]
  mov   ah,ch                          { Frekvencia felső bitek+   }
  call  RegSet                         { oktáv+KEY ON bit          }
```

A frekvencia után az oktáv számát a megfelelő helyre tesszük, majd a frekvencia felső bájtjába írjuk. A táblázatban már eleve olyanok az értékek, hogy a *KEY ON* bit bekapcsolt állapotban van. Mielőtt azonban az új hangot kiküldenénk, az előzőt mindenképpen ki kell kapcsolni, hiszen a *KEY ON* bit egyelőre (az előző hang miatt) még 1-es állapotban van. A kikapcsolás után kiadjuk az új hangot. Érdemes megfigyelni, hogy minden egyes ADLIB regiszter írásakor az aktuális fizikai csatorna számát hozzáadjuk a regiszterek bázisértékéhez (add al,[bx+_cnum]) !

```
  segES lodsw                          { Az időállandó betöltése }
  mov   dl,ah                          { A felső bájtot tároljuk }
```

```
and   ax,$7fff
mov   [bx+_Time],ax                { Az idő betöltése          }
and   dl,10000000b                 { A legfelső bit a GATE     }
and   [bx+_flag].byte,01111111b
or    [bx+_flag],dl                { Betesszük a flag mezőbe }
```

A hang kiadása után be kell állítani a *Time* mezőt. Beolvassuk az időértéket, amely word típusú. Ezt tároljuk a *Time* mezőben, de a legfelső bit nélkül, mert az a következő időérték meglétét jelzi. Ha a legfelső bit 1, akkor még egyéb teendők is vannak, egyébként ki lehet lépni, hiszen ilyenkor nincs kitartási idő. A legfelső bitet tárolni kell a *flag* mezőben is!

```
test  dl,10000000b
jz    @exit                        { Vége, a legfelső bit 0  }
mov   cx,bx                        { A BX tárolása a CX-ben  }
mov   al,[bx+_cnum]                { A fizikai csatornaszám  }
lea   bx,OperOff                   { alapján meghatározzuk az }
xlat                               { első operátor címét     }
mov   bx,cx                        { A BX-et visszaállítjuk  }
xor   ah,ah
mov   di,ax                        { A regiszter eredeti     }
add   di,offset Effects            { értékét kivesszük az    }
mov   ah,[di]                      { árnyéktárból            }
or    ah,00100000b                 { GATE = 1                }
add   al,$20
call  RegSet                       { Az első operátor        }
add   al,$03
call  RegSet                       { A második operátor      }
jmp   @exit                        { Kilépés                 }
```

Ha van kitartási idő, akkor a *GATE* bitet 1-be kell állítani. Ilyenkor az *OperOff* tömb és a *cnum* mező alapján meghatározzuk az első operátor címét. Az *Effects* tömbben megtalálható a regiszterek eredeti értéke. Erre szükség van, hiszen mi csak a *GATE* bitet akarjuk módosítani. Ezért előhívjuk az eredeti értéket, és a *GATE* bitet beállítjuk benne, majd mindkét operátorba betöltjük. Így a hang a definiált időig kitartva marad, és a *flag* mező legfelső bitje jelzi, hogy most ilyen típusú hangot adunk ki. Ha az idő lejár, akkor a flag megvizsgálásával el lehet dönteni, hogy kell-e újabb időciklus. Ez a *PlayChannel* eljárásban történik meg.

Ha egy utasításkód 7. bitje 1, akkor valamilyen vezérlőparancsról van szó. A vezérlőparancsok címeit a *CMDtab* táblázat tartalmazza. A *@commands* ponton — a parancs kódja alapján — a program a megfelelő címre ugrik, és megkezdődik az adott parancs végrehajtása.

```
@commands:
  mov   di,ax                      { DI= utasításkód         }
  and   di,$007f                   { Kimaszkoljuk a kódot    }
  shl   di,1                       { DI=DI*2                 }
```

```
  jmp    CS:[ di+offset @CMDtab] .word { Ugrás a parancsra      }
@CMDtab: { A táblázat a parancsrutinok címeit tartalmazza      }
  dw     offset @CMDend,  offset @CMDjmp,    offset @CMDcall
  dw     offset @CMDret,  offset @CMDregset, offset @CMDrpt
  dw     offset @CMDdjnz, offset @CMDpause,  offset @CMDsetcnum

@CMDend:                                  { Az END parancs rutinja }
  or     [ bx+_flag] .byte,00000001b
  jmp    @exit
```

End: A parancs beállítja a flag mező 0. bitjét. A Play eljárás ezt teszteli, és ha 1-es állapotban találja, akkor már nem kerül sor újabb parancs lehívására, azaz a logikai csatorna befejezi a végrehajtást.

```
@CMDjmp:                                  { A JMP parancs rutinja  }
  mov    si,ES:[ si]
  jmp    @nextcommand
```

Jmp: Az utasítás lehív egy word típusú paramétert, ez az új végrehajtási cím. Ezt az SI regiszterbe tölti, majd visszaugrik a parancslehívásra. (Az újabb parancs természetesen már az új címről érkezik.)

```
@CMDcall:                                 { A CALL parancs rutinja  }
  segES lodsw
  sub    [ bx+_SP] .word,2                { Az SP csökkentése       }
  mov    di,[ bx+_SP]                     { A mutatott cím a DI-be  }
  mov    [ di] ,si                        { A régi PC a verembe     }
  mov    si,ax                            { Az új cím a PC-be       }
  jmp    @nextcommand
```

Call: A *call* esetében a *jmp*-hoz hasonlóan szintén egy word típusú paraméter adja a szubrutin címét, de itt az elugrás előtt még menteni kell a visszatérési címet. Ezért az *SP* mező alapján a *call* a verem megfelelő helyére teszi az SI eredeti értékét, hiszen ez a visszatérési cím. A verem ***predekrementális*** (előzetes csökkentés) elven működik, azaz a cím betöltése előtt a mutató kettővel csökken.

```
@CMDret:                                  { A RET parancs rutinja  }
  mov    di,[ bx+_SP]                     { DI-be a mutatott cím   }
  mov    si,[ di]                         { SI=visszatérési cím    }
  add    [ bx+_SP] .word,2                { A veremmutató növelése }
  jmp    @nextcommand
```

Ret: A *ret* utasítás a *call* „fordítottja". A veremből kiemel egy word adatot, ez a visszatérés címe. Ezt az SI regiszterbe tölti, majd visszaugrik a következő parancs lehívásához. A veremből egy word kiemelése mindig ***posztinkrementális*** (utólagos növelés) elven történik, azaz a cím kiolvasása után a veremmutató megnövekszik kettővel.

```
@CMDpause:                          { A PAUSE parancs rutinja  }
  segES lodsw                       { Az időállandó beolvasása }
  mov   dl,ah
  and   ax,$7fff                    { A legfelső bit nem kell  }
  mov   [bx+_Time],ax
  test  dl,10000000b                { Ki kell kapcsolni?       }
  jz    @exit                       { Ugrás, ha nem kell       }
  mov   ax,$00b0
  add   al,[bx+_cnum]               { A fizikai csatorna       }
  call  RegSet                      { kikapcsolása: KEY ON=0   }
  jmp   @exit
```

Pause: A parancsnak egy word típusú paramétere van, amely a szünet idejét definiálja. Ha a legfelső bit 1, akkor ez azt jelenti, hogy az éppen szóló hangot lecsengés nélkül azonnal ki kell kapcsolni. Ilyen esetben az utasítás kitörli a *KEY ON* bitet.

```
@CMDregset:                         { A REGSET parancs rutinja }
  segES lodsw                       { AX=következő adat        }
  cmp   ax,$ffff                    { Vége?                    }
  jnz   @setnext                    { Ugrás, ha nem            }
  mov   [bx+_Time].word,$0001
  jmp   @exit
@setnext:
  call  RegSet                      { Beállítás                }
  jmp   @CMDregset                  { Vissza az elejére        }
```

RegSet: Ennek az utasításnak akárhány paramétere lehet. Folyamatosan olvassa a word paramétereket és vizsgálja, hogy a beolvasott adat $FFFF-e. Ha igen, akkor vége a beállításoknak, ha viszont nem, akkor beállítja a megfelelő ADLIB regisztert. Látható, hogy ez a művelet sok beállítás esetén sokáig tarthat. A PlayNext rutint a timer megszakítás hívja, ez pedig nem lehet akármilyen hosszú, ezért a RegSet egyszeri használatánál nem szabad túl sok beállítást elvégezni, mert ha a megszakítások egymásba érnek, akkor a rendszer lefagy. Ezért nem érdemes egyszerre 20-30 regiszternél többet átállítani.

```
@CMDrpt:                            { Az RPT parancs rutinja   }
  segES lodsb                       { Repeater szám és érték  }
  mov   di,ax
  and   al,00111111b                { Az érték kimaszkolása    }
  shr   di,5
  and   di,$0006                    { DI=a repeater száma*2    }
  mov   [bx+di+_roff],si            { A cím tárolása           }
  shr   di,1
  mov   [bx+di],al                  { A számláló beállítása    }
  jmp   @nextcommand
```

Rpt: Az utasításnak egy bájt típusú paramétere van, amely tartalmazza a számlálási számot és a *repeater* számát is. A *repeater* száma alapján a megfelelő *roff* mezőbe beteszi az SI értékét, azaz a ciklus kezdőcímét, míg a megfelelő *rpt* mezőbe a számlálási értéket írja.

```
@CMDdjnz:                                { A DJNZ parancs rutinja }
  segES lodsb
  mov   di,ax
  and   di,$0003                         { DI =a repeater száma   }
  dec   [bx+di].byte                     { A számláló csökkentése }
  jz    @endrepeat                       { Ugrás, ha lejárt       }
  shl   di,1
  mov   si,[bx+di+_roff]                 { SI =a ciklus kezdőcíme }
@endrepeat:
  jmp   @nextcommand
```

Djnz: A *djnz*-nek szintén egyetlen bájt paramétere van, amely a *repeater* számát adja meg. Gyakorlatilag a paraméter alsó két bitje érvényes csak. Ennek alapján csökkenti a megfelelő *rpt* mező értékét. Ha a számláló még nem járt le, akkor a megfelelő *roff* mezőből az SI regiszterbe tölti a ciklus kezdőcímét. Ha a számláló lejárt, akkor a *djnz* nem módosítja az SI-t, tehát nincs újabb ismétlés.

```
@CMDsetcnum:                             { A CHNUM parancs rutinja }
  segES lodsb                            { AL =  új csatornaszám   }
  mov   [bx+_cnum],al                    { cnum = AL               }
  jmp   @nextcommand
```

ChNum: A csatornaszám beállításánál egy bájtparaméter adja meg az aktuális fizikai csatorna számát. Ez a *cnum* mezőbe kerül.

```
@exit:
  mov   [bx+_pc],si                      { A PC frissítése        }
  pop   ES                               { Az eredeti ES vissza   }
end;                                     { PlayNext kész          }
```

A *PlayNext* eljárás tehát az összes parancs végrehajtási rutinját tartalmazza. Figyeljük meg, hogy az utasítások kétféleképpen fejeződhetnek be: vagy visszaugrunk a *@nextcommand* pontra, hogy egy újabb parancsot lehívjunk, vagy kiugrunk az eljárásból az *@exit* ponton keresztül. Három parancsnál van közvetlen kiugrás: a *Pause*, az *End* és a *RegSet* esetében. A *Pause* ugyanúgy időegységek számlálását jelenti, mint az egyszerű hangjegy-megszólaltatás, ezért ez után nem kell már újabb parancsot lehívni, csak ha a beállított idő lejárt. Az *End* a csatorna működésének leállítását jelenti, ezért itt nem is szabad újabb parancsot lehívni. A *RegSet*-nél viszont nem lenne akadálya az újabb parancsok értelmezésének. Itt még egy egy időegységnyi késleltetés is van, hogy a beállítások után legalább egy megszakítás „üres" legyen. Ezt azért tartottuk célszerűnek, mert ez az utasítás egyébként is sokáig tarthat, tehát az újabb parancsok lehívása — ha például a következő utasítás szintén *RegSet* — túlságosan megnövelheti a végrehajtási időt, így lefagyáshoz vezethet.

```
procedure PlayChannel;                                  assembler;
asm
  dec   [bx+_time].word                  { Az  idő csökkentése      }
```

```
  jnz   @exit                          { Ugrás, ha még nem járt le }
  { Ha az idő lejárt, a flag 7. bitjétől függően vagy              }
  { újabb időadat, vagy újabb utasítás következik                  }
  test  [bx+_flag].byte,10000000b
  jz    @next                          { Ugrás, ha újabb utasítás  }
  push  ES
  mov   ES,[bx+_pc+2]                  { ES:SI = a PC értéke       }
  mov   si,[bx+_pc]
  add   [bx+_pc].word,$0002            { A PC növelése             }
  segES lodsw                          { AX = az új időadat        }
  mov   [bx+_time],ax                  { A Time mező feltöltése    }
  and   [bx+_flag].byte,01111111b      { 7. bit = 0                }
  { Most a GATE bit kikapcsolása következik }
  mov   cx,bx                          { A BX tárolása a CX-ben    }
  mov   al,[bx+_cnum]                  { Az első operátor címének  }
  lea   bx,OperOff                     { meghatározása a cnum és   }
  xlat                                 { az OperOff alapján (AL)   }
  mov   bx,cx                          { BX eredeti értéke vissza  }
  xor   ah,ah
  mov   di,ax
  add   di,offset Effects              { DI = az árnyéktár címe    }
  mov   ah,[di]                        { AH = eredeti érték        }
  and   ah,11011111b                   { GATE = 0                  }
  add   al,$20
  call  RegSet                         { Az első operátor          }
  add   al,$03
  call  RegSet                         { A második operátor        }
  pop   ES
  jmp   @exit                          { Kilépés                   }
@next:
  call  PlayNext                       { Újabb utasítás            }
@exit:
end;
```

PlayChannel: A *PlayChannel* az időzítéseket kezeli. Ez az eljárás csökkenti a *Time* mezőt, és kezeli az esetleges *GATE* bit beállítást. Ha az időzítő lejárt, akkor megvizsgálja a flag mező legfelső bitjét, ez mutatja, hogy egy újabb időzítő adat következik. Ha nincs újabb időzítő adat, akkor a *PlayNext* meghívásával folytatódhat a végrehajtás. Ha viszont van újabb időzítő adat, akkor ezt beolvassa, és egy újabb időzítési állandót állít be. Ekkor törli a megfelelő *GATE* bitet, így elkezdődhet a *release* ciklus. A *GATE* bit mellett a *flag* mező legfelső bitjét szintén törölni kell, különben a *PlayChannel* újabb időállandót olvasna be.

```
procedure Play;                                        assembler;
asm
  lea   bx,Channel                     { BX =a csatornaadatok címe }
  mov   cx,0001                        { CX számú logikai csatorna }
@playchannels:
  push  cx
```

```
  test  [bx+_flag].byte,00000001b    { 0. bit = 1                }
  jnz   @nextplay                    { Ugrás, ha a csatorna már  }
                                     { ki van kapcsolva,         }
  call  PlayChannel                  { egyébként pedig újabb     }
                                     { végrehajtás               }
@nextplay:
  add   bx,ChanDataSize              { A mutató növelése         }
  pop   cx
  loop  @playchannels                { Minden csatorna futtatása }
end;
```

Play: A *Play* lejátssza az összes logikai csatornát. Tulajdonképpen ez az az eljárás, amelyet a timer megszakítás közvetlenül meghív. Jelenleg csupán egyetlen logikai csatornát futtat (mov cx,0001), de ez természetesen módosítható. Az eljárás működése egyszerű: betölti a legelső csatorna adatainak címét a BX regiszterbe, majd futtatja azt. Ezt követően megvizsgálja a *flag* mező 0. bitjét, és csak akkor hívja a *PlayChannel*-t, ha ez a bit 0, azaz a csatorna be van kapcsolva. Az első csatorna után következne a többi, de jelen esetben nincsen több, mert a CX 1!

```
procedure WriteReg(regnum,value:byte);                    assembler;
asm
  mov   al,regnum
  mov   ah,value
  call  RegSet
end;
```

WriteReg: A *WriteReg* eljárást azért készítettük, hogy az ADLIB regisztereit közvetlenül is át lehessen állítani. A *RegSet* eljárás regiszterekben várja a paramétereket, ezért nem szabad közvetlenül hívni, hanem helyette a *WriteReg*-et kell használni, amely beállítja az AL és az AH regisztereket a megfelelő értékekre. A programban csak a *ResetADLIB* eljárás használja a kártya alaphelyzetbe állításához.

```
procedure newIRQ;                                         assembler;
asm
  push DS                            { A regiszterek tárolása }
  pusha
  mov  ax,Seg @data                  { A DS beállítása az     }
  mov  DS,ax                         { adatszegmensre         }
  call Play                          { A csatornák kezelése   }
  popa
  pop  DS                            { Regiszterek vissza     }
  iret                               { A megszakítás vége     }
end;
```

newIRQ: A *newIRQ* a mi saját timer megszakításunk, amely minden egyes megszakításkor kezeli az ADLIB kártyát, ha szükséges: Mint látható, az eljárásban egyszerűen beállítjuk a DS regisztert a program adatszegmensére, majd a *Play* rutint hívjuk. Mivel ez egy megszakítási rutin, ezért mindenképpen az IRET utasítással kell befejezni.

```
procedure SetTimer(p:word);                                    assembler;
asm
  cli                                    { Megszakítás tiltva          }
  mov   al,$36
  out   $43,al                           { Timer adatszó               }
  mov   ax,P
  out   $40,al                           { Az alsó bájt                }
  mov   al,ah
  out   $40,al                           { A felső bájt                }
  sti                                    { Megszakítás engedélyezve    }
end;
```

SetTimer: Ezzel az eljárással a timer időzítését lehet átállítani. Alapesetben a timer kb. 18.2 megszakítást kér másodpercenként. A programban ezt 50 megszakításra állítottuk át.

```
procedure ResetADLIB;
var i:byte;
begin
 for i:=1 to $F5 do WriteReg(i,$00);
end;
```

ResetADLIB: Az eljárással a kártyát alapállapotba lehet hozni. A rutin nullázza az összes ADLIB regisztert.

A főprogram

```
BEGIN
  clrscr;
  { Minden adat nullázása a logikai csatornákon                    }
  fillchar(Channel,Sizeof(Channel),$00);
  { Beolvassuk az adatfájlt                                        }
  assign(f,'proba.dat');
  reset(f,1);
  size:=filesize(f);
  getmem(music,size);
  blockread(f,music^,size);
  close(f);
  { A kártya regisztereinek inicializálása                         }
  ResetADLIB;

  { A 0. csatorna adatainak beállítása                             }
  with Channel[ 0] do
   begin
    PC:=music;                  { A PC a zeneadatokra mutat        }
    SP:=ofs(Stk[ 0,31] );       { A veremmutató beállítása         }
    Time:=1;                    { Ez nagyon fontos!                }
    cnum:=0;                    { A kezdeti fizikai csatorna:#0}
   end;
```

```
 getintvec($1c,@oldIRQ);          { A régi timer megszakítás címe}
  setintvec($1c,@newIRQ);         { Az új megszakítás beállítása }
  settimer(23862);                { 50 IRQ/sec }

  {*********************************}
  {*}  {Ez itt a főprogram helye}  {*}
  {*} repeat                      {*}
  {*} until keypressed;           {*}
  {*} while keypressed do readkey; {*}
  {*********************************}

  settimer(0);                    { 18.2 IRQ/sec                  }
  setintvec($1c,@oldIRQ);         { Az eredeti megszakítás címe   }
  freemem(music,size);            { A heap felszabadítása         }
  ResetADLIB;                     { A kártya nullázása            }
END. of program
```

A főprogramban elsőként nullázzuk a logikai csatornák adatait, majd beolvassuk a próbafájlt, amelyben egy egyszerű zeneprogram található. Ez után inicializáljuk a kártya regisztereit, majd feltöltjük a 0-s logikai csatorna legfontosabb mezőit. A *PC*-t ráállítjuk a beolvasott adatok kezdőcímére, az *SP*-t pedig az *Stk* tömbre. A *Time* mezőt 1-re kell állítani, hogy a következő megszakításnál a lejátszás elinduljon. Alapesetben a 0-s fizikai csatornát választjuk ki a lejátszásra, azaz *cnum*=0. A $1C megszakítás vektorát ráállítjuk a saját rutinunkra, és a megszakítások számát 50-re állítjuk. Ezzel automatikusan el is indul a lejátszás, és a főprogramban egy *repeat-until* ciklus figyeli a billentyűk lenyomását, míg a zene lejátszása a megszakítás feladata. Ha ilyenkor lenyomunk egy billentyűt, akkor a megszakítási vektor visszakapja az eredeti értékét, és a zene leáll.

Most már van egy jól definiált programnyelvünk és egy olyan programunk, amely ezen nyelv utasításait végre tudja hajtani. :–) Zeneprogramunk viszont még nincs. :–(A parancsokat kódolni kézi módszerrel bizony elég nehézkes és hálátlan feladat. Írjunk tehát egy fordítóprogramot is! Persze ez a fordítóprogram igen egyszerű lesz, ezért a forrásszövegben előforduló hibákat nem biztos, hogy fel tudja majd fedezni. A most következő programot tehát ki-ki ízlése szerint tovább bővítheti és alakíthatja!

3.4.4. A fordító (*compiler*)

A fordítónak egy szövegfájlból — az abban elhelyezett parancsok alapján — bináris kimeneti fájlt kell generálnia, amelyet a *player* be tud olvasni és le tud játszani. A feladat tehát szövegfeldolgozás. A program soronként beolvassa a forrásszöveget, és minden sort értelmezni próbál. A fent megadott parancsokhoz hozzárendeli a megfelelő kódot, és kiértékeli a paramétereket. Az igazi előnye mégis az, hogy automatikusan kiszámolja a forrásban előforduló ugrások, szubrutinok címeit. A fordítás két menetben történik. Az első menetben a program értelmezi a parancsokat és eltárolja a címkéket, de az ugrások

és szubrutinok címeit még nem határozza meg, csupán megjegyzi, hol talált *call/jmp* utasítást. A második menetben ezeket a helyeket a tárolt címkék alapján feltölti a megfelelő címmel.

```
program compiler; { COMP.PAS }
{$f+}
{ Az F direktívának bekapcsolt állapotban kell lennie!! }
uses crt;
```

Típusok

```
type
  amem = array[ 0..32766] of byte;    { Memória típus          }
  memptr = ^amem;                     { Memóriamutató típus    }

  awrec = record                      { Egy word adatrekord    }
  case byte of
    0: ( w : word);
    1: ( l : byte;
         h : byte);
  end;

  label_type = record                 { Címke típus            }
   name : string[ 8] ;
   offs : word;
  end;

  pcmod_type = record                 { Ugrás típus            }
   name : string[ 8] ;
   offs : word;
      l : word;
  end;

  pcm = array[ 0..1023] of pcmod_type;
  lbl = array[ 0..1023] of label_type;
  pcmptr = ^pcm;
  lblptr = ^lbl;
```

Az *amem* típus egy 32K nagyságú memóriát definiál, a *memptr* pedig egy erre mutató pointert. A program a lefordított kódot egy ilyen területre helyezi el. Az *awrec* egy word típusú paramétert jelent, amelynek a bájtjait az *l* és a *h* mezőben egyenként is el lehet érni. Azoknál az utasításoknál hasznos, ahol két bájtparaméter van. A programban előforduló címkéket a *label_type* típusú változókban tároljuk. Amikor a fordító az első menetben egy címkét talál, akkor megjegyzi annak nevét (*name*) és címét (*offs*), így az első menet végén minden címke ismert lesz. Hasonló célt szolgál a *pcmod_type* típus is, csak

ebben az előforduló *call/jmp* utasításokat tároljuk. Amikor a fordító az első menetben egy *call* vagy *jmp* utasítást talál, akkor megjegyzi a címet (*offs*), a címke nevét, amelyre az utasítás hivatkozik (*name*) és a forrássor számát, ahol az utasítást találta (*l*). Ezért az első menet végén az összes *call* és *jmp* utasítás ismert lesz. A *pcm* és az *lbl* az előző típusokat tartalmazó 1024 elemű tömböket definiál, míg a *pcmptr* és az *lblptr* ezekre a tömbökre mutató pointereket. (A programban azért definiálunk minden fontosabb típushoz mutatót, mert az ilyen változók dinamikusak lesznek, azaz a heapben tároljuk majd őket.)

Állandók

```
const
  keyword : array[ 0..21] of string[ 6] =(
   'C','C#','D','D#','E','F','F#','G','G#','A','A#','H',
   'END','JMP','CALL','RET','REGSET','RPT','DJNZ','PAUSE',
   'CHNUM','FINISH');

  etx : array[ 0..7] of string[ 20] =(
  'Ismeretlen utasítás','Túl sok a címke','Címkeismétlés',
  'Túl sok a CALL/JMP','Ugrások címke nélkül',
  'Nem definiált címke','Számelírás','Nyitott hangsor');
```

A *keyword* tömb a forrásszövegben előforduló kulcsszavakat tartalmazza. Ezek azok a szavak, amelyek alapján a fordító tudja majd, hogy milyen kódot kell generálnia, valamint hogy milyen paraméterek szükségesek az adott parancshoz. Az *etx* tömb a fordító által felfedezett hibákat tartalmazza. Természetesen a program igen rövid egy ilyen nagyságrendű feladat megoldásához, így ezen a téren jócskán akad még bővítenivaló!

Változók

```
var
      line : word;              { Az aktuális sorszám          }
     ns,nd : string[ 40];       { A forrás- és a célfájl neve  }
      jmps : pcmptr;            { Az ugrások tárolása,         }
      labs : lblptr;            { a címkék tárolása;           }
                                { mindkettő pointer!           }
  pc,jn,ln : word;              { PC, ugrásszám, címkeszám     }
    source : text;              { A forrásállomány             }
      dest : memptr;            { Célcím a memóriában(pointer) }
      { A további változókat általánosan használjuk          }
         f : file;
         s : string;            { A feldolgozás alatt álló sor }
   i,j,k,l : integer;
         q : byte;              { A sor aktuális karaktere     }
       par : awrec;
       ps1 : boolean;           { Az első menet végének jelzése }
```

Eljárások

```
procedure ErrorExit(er:byte);
begin
 sound(440);delay(100);nosound;          { Hangjelzés          }
 close(source);                          { A forrásfájl lezárása }
 writeln;                                { Kiírás a képernyőre  }
 writeln('Hiba a ',line,'. sorban: ',etx[er]);
 halt;       { A program leállítása, kilépés                   }
end;
```

ErrorExit: Az eljárás hívására akkor kerül sor, amikor a fordítás során valamilyen hiba történt. A hibát detektáló eljárás egy hibakódot küld, amelynek alapján az *ErrorExit* kinyomtatja a hibához tartozó szöveget. A szöveg kiírása után a fordító leáll, nem keletkezik célfájl.

```
procedure spaceskip;
begin                                    { A szóközkarakterek átlépése }
 while (s[q]=' ')and(s[q]<>#254) do inc(q);
end;
```

spaceskip: Átugorja a szövegben a szavak és paraméterek közötti szóközöket. A forrásban csak a szóközt lehet elválasztójelként használni!

```
function nextword:string;          { Lehívja a következő szót    }
var d:string;
begin
 if s='' then
  begin
   nextword:=#0; exit;
  end;
 d:=''; spaceskip;                 { Átlépi a szóközöket         }
 while (s[q] in [#33..#127]) do
  begin      { Kigyűjti a #33..#127 karaktereket; ez a szó      }
   d:=d+upcase(s[q]);
   inc(q);
  end;
 nextword:=d; spaceskip;           { Rááll a következő szóra     }
end;
```

nextword: Az éppen feldolgozott sor következő szavát adja. A fordító szónak tekint minden olyan karaktercsoportot, amelyet szóközök választanak el egymástól. A számokat szintén szónak tekinti. A *nextword* eljárást használjuk a kulcsszavak és a paraméterek lehívására.

```
function nextnum:byte;  { Lehív egy bájtparamétert                }
var d:string;
```

```
    b:byte;                { Bájt !! }
    i:integer;
begin
 d:=nextword; d:='$'+d; { Lehívja a következő szót          }
 val(d,b,i);              { Megpróbálja számmá átalakítani }
 if i<>0 then errorexit(6);      { Ha hiba történt, kilépés }
 nextnum:=b;                { Visszatér a számmal           }
end;
```

nextnum: Egy bájt beolvasása a szövegből. Ennek az eljárásnak a hívására akkor kerül sor, amikor egy bájt típusú paraméternek kell következnie. A *nextword*-öt használva behívja a következő szót, majd a *val* eljárással megpróbálja számmá alakítani. A szó elé egy '$' jelet tesz, így minden egyes számot hexadecimálisként próbál értelmezni. Ha a *val* nem tudja a szót számmá alakítani, akkor számelírás történt; ilyenkor az *ErrorExit*-et hívja.

```
function nextwnum:word; { Lehív egy word paramétert          }
var d:string;
    w:word;             { Word !! }
    i:integer;
begin
 d:=nextword; d:='$'+d; { Lehívja a következő szót          }
 val(d,w,i);            { Megpróbálja számmá átalakítani    }
 if i<>0 then errorexit(6);   { Ha  hiba történt, kilépés   }
 nextwnum:=w;               { Visszatér a számmal           }
end;
```

nextword: Egy word beolvasása a szövegből. Ugyanaz a célja, mint a *nextnum*-nak, csak ezzel word paramétereket lehet beolvasni. Működése megegyezik a *nextnum*-éval, de a visszaadott adat típusa nem bájt, hanem word.

```
procedure nextline;              { Beolvassa a következő sort  }
begin
 readln(source,s); inc(line); { Beolvassa a sort            }
 gotoxy(1,wherey);
 write('Forrássor: #',line);  { Kiírja a sorszámot          }
 s:=s+#254; q:=1; { Hozzátesz egy #254-es karaktert         }
end;
```

nextline: A következő sor beolvasása a forrásszövegből. Az eljárás az *s* változóba teszi a következő sort, amelyhez hozzáfűz egy #254-es karaktert, mert a fordító ezt értelmezi általánosan sorvégjelként. A *q* változó az éppen következő karaktert mutatja, ezért ezt 1-re állítja.

```
procedure twoparam;              { Két bájtparaméter lehívása  }
begin
 par.l:=nextnum; par.h:=nextnum;
end;
```

twoparam: Két bájtparaméter beolvasása. Ezt akkor használja a fordító, amikor két bájtparaméternek kell következnie.

```
procedure put(a:byte);  { Egy bájt kiküldése a célhelyre }
begin
 dest^[pc]:=a; inc(pc);
end;
```

put: Egy bájt kiküldése a célhelyre. A *dest* a lefordított kódra mutat. A bájt elhelyezése után megnöveli a *pc* változót, amely azt mutatja, hogy hány bájt hosszú már a lefordított zeneprogram.

```
procedure putw(a:word); { Egy word kiküldése a célhelyre }
begin
 put(lo(a)); put(hi(a));      { Alsó bájt, felső bájt        }
end;
```

putw: Egy word kiküldése a célhelyre. Az eljárás célja ugyanaz, mint a *put*-é.

```
procedure storelabel(lb:string);          { Egy címke tárolása }
var i:integer;
begin
 if ln=1023 then ErrorExit(1); { Túl sok címke }
 if ln<>$ffff then
  for i:=0 to ln do
   if labs^[ln].name=lb then ErrorExit(2); { Újradefiniálás }
 inc(ln); { Növeljük a címkék számát }
 with labs^[ln] do
  begin   { Beírjuk az új címkét }
   name:=lb; offs:=pc;
  end;
end;
```

storelabel: Egy címke tárolása. Az eljárás eltárol egy címkét a *labs^* tömbben. A fordító 1024 címkét képes tárolni. Ha a címke már definiált, vagy túl sok címke van, akkor az *ErrorExit*-et hívja.

```
procedure storejmp(lb:string);       { Egy call/jmp tárolása }
begin
 if jn=1023 then ErrorExit(3);       { Túl sok call/jmp }
 inc(jn);
 with jmps^[jn] do
  begin                  { Beírja az utasítást a jmps^ tömbbe }
   name:=lb; offs:=pc; l:=line;
  end;
end;
```

storejmp: Az ugrások tárolása. Az eljárás a *call/jmp* utasításokat tárolja. Ha már túl sok a tárolt adat (1024), akkor az *ErrorExit*-et hívja.

A következő eljárások a fordítást végzik. Minden kulcsszóhoz tartozik egy-egy eljárás, amely kezeli az adott utasítás paramétereit. Bár a fordítás sororientált, a hangjegyeknél és a *RegSet* utasításnál lehetőség van egy sorban több utasítást is megadni.

```
procedure note(n:byte); { A hangjegyek fordítása }
var  w:word;
     sl:string;
    i,j:integer;
begin
 repeat
  put(n or ((nextnum and 7) shl 4) );
  w:=nextwnum; { Az első időállandó }
  putw(w);
  if w and $8000=$8000 then putw(nextwnum); { Két időállandó }
  sl:=nextword;
  if sl=':' then { Az egy sorban lévő hangok kódolása }
   begin
    sl:=nextword; j:=$ff;
    for i:=0 to 11 do if sl=keyword[ i] then j:=i;
    if j=$ff then ErrorExit(7) else n:=j;
    sl:=':';
   end;
 until sl<>':';
end;
```

note: Hangjegyek befordítása. Az eljárás a hangokat fordítja megfelelő kódra. A hangok esetén lehetőség nyílik arra, hogy a : jellel elválasztva több rekordot definiáljunk egy soron belül. Itt is figyelni kell a szóközök meglétére, különben a fordító hibát jelez. A későbbi példákban részletesen kitérünk majd a tipikus hibákra.

```
procedure cmd_REGSET;          { A regset parancs fordítása      }
var h,l,c:byte;
        b:boolean;
       sl:string;
begin
 put($84); b:=false;                  { Kiküldi a parancs kódját }
 repeat
  repeat           { Átugorja az üres és a megjegyzés sorokat    }
   nextline; spaceskip;
  until not (s[ q] in [ #254,';'] );
  spaceskip;
  if s[ q] <>'.' then      { Keresi a végjelző karaktert          }
   begin
    repeat                 { Kódolás, amíg nem talál végjelzőt    }
     twoparam; putw(word(par));
```

```
    sl:=nextword;
   until sl<>':';
  end else b:=true;
 until b;
 putw($ffff);                    { Kiküldi a végjelző adatot    }
end;
```

cmd_REGSET: A *RegSet* parancs lefordítása. A parancs több sorból áll. A beállítások végét egy '.'-tal kezdődő sor jelzi.

```
procedure cmd_RPT;               { Az rpt parancs fordítása     }
var b:byte;
begin
 put($85);                       { Kiküldi a parancs kódját     }
 twoparam;                       { Lehívja a két paramétert     }
 b:=((par.l and 3) shl 6) or (par.h and $3f);
 put(b);                         { Kiküldi az operandust        }
end;
```

cmd_RPT: Az *rpt* parancs lefordítása.

```
procedure cmd_DJNZ;              { A djnz parancs fordítása     }
begin
 put($86);                       { Kiküldi a parancs kódját     }
 put(nextnum and 3);             { Kiküldi az operandust        }
end;
```

cmd_DJNZ: A djnz parancs lefordítása.

```
procedure cmd_JMP;               { A jmp parancs fordítása      }
var sz:string[8];
begin
 put($81); sz:=nextword;         { Kiküldi a parancs kódját     }
 storejmp(sz);                   { Tárolja az utasítást         }
 putw($0000);                    { Az operandus egyelőre $0000  }
end;
```

cmd_JMP: A *jmp* parancs lefordítása. A címet még nem számolja ki, csupán tárolja az utasítást a *jmps^* tömbben. A címet a második menetben definiáljuk.

```
procedure cmd_CALL;              { A call parancs fordítása     }
var sz:string[8];
begin
 put($82); sz:=nextword;         { Kiküldi a parancs kódját     }
 storejmp(sz);                   { Tárolja az utasítást         }
 putw($0000);                    { Az operandus egyelőre $0000  }
end;
```

cmd_CALL: A *call* parancs lefordítása. A címet még nem számolja ki, csupán tárolja az utasítást a *jmps^* tömbben. A címet a második menetben definiáljuk.

```
procedure cmd_RET;                { A ret parancs fordítása      }
begin
 put($83);                        { Kiküldi a parancs kódját     }
end;
```

cmd_RET: A *ret* parancs lefordítása.

```
procedure cmd_PAUSE;              { A pause parancs fordítása    }
begin
 put($87);                        { Kiküldi a parancs kódját     }
 putw(nextwnum);                  { Kiküldi az operandust        }
end;
```

cmd_PAUSE: A *pause* parancs lefordítása.

```
procedure cmd_END;                { Az end parancs fordítása     }
begin
 put($80);                        { Kiküldi a parancs kódját     }
end;
```

cmd_END: Az *end* parancs lefordítása.

```
procedure cmd_CHNUM;              { A chnum parancs fordítása    }
begin
 put($88);                        { Kiküldi a parancs kódját     }
 put(nextnum);                    { Kiküldi az operandust        }
end;
```

cmd_CHNUM: A *chnum* parancs lefordítása.

```
procedure cmd_FINISH;             { A fordítást lezáró FINISH    }
begin
 put($80); ps1:=true;    { Az első menet végének jelzése !!   }
end;
```

cmd_FINISH: Az *end* paranccsal analóg. Ez az utasítás nem létezik a *player*-ben, csupán a fordító igényli. Az első menet addig folytatódik, amíg a fordító megtalálja a *FINISH* kulcsszót. Ez csak egy formai követelmény, de a későbbiekben látni fogjuk az értelmét.

```
const
  routine : array[ 0..9] of pointer=(
   @cmd_END,@cmd_JMP,@cmd_CALL,@cmd_RET,@cmd_REGSET,
   @cmd_RPT,@cmd_DJNZ,@cmd_PAUSE,
   @cmd_CHNUM,@cmd_FINISH);
```

```
procedure command(cm:string);
var i,j:byte;
      p:procedure;
begin
 j:=$ff;
 { A beolvasott szó keresése a kulcsszavak között                    }
 for i:=0 to 21 do if cm=keyword[ i] then j:=i;
 if j=$ff then ErrorExit(0);          { A kulcsszó ismeretlen }
 case j of
  0..11  : note(j);      { Ugrás, ha a szó egy hangjegy          }
  12..28 : begin
             @p:=routine[ j-12] ; { p = a megfelelő rutin        }
              p;  { Meghívjuk a kulcsszóhoz tartozó              }
                  { fordító eljárást                             }
           end;
 end;
end;
```

routine,command:
A *routine* tömb tartalmazza az előbbi fordító eljárások címeit. A *command* eljárás a beolvasott kulcsszó alapján kiveszi a megfelelő címet a *routine* tömbből, és meghívja az adott eljárást.

```
procedure compilepass_1;
var inst : string;
begin
 ps1:=false; jn:=$ffff;ln:=$ffff;
 repeat
  nextline;
  inst:=#0; inst:=nextword;
  case inst[ 1] of
   ';',#13,#0,#254 :;          { Ezek a megjegyzéssorok          }
   '@' : storelabel(inst);     { @ esetén a szó címke,           }
   else command(inst);         { különben parancs                }
  end;
 until ps1;
 writeln; writeln('Az első menet kész!');
end;
```

compilepass_1: A fordítás első menete. Az eljárás sorban olvassa a forrásszöveg sorait, és minden sorra hívja a *command* eljárást, kivéve, ha a sor megjegyzés vagy címke. A megjegyzéssoroknak — hasonlóan az Assemblerhez — a ; jellel kell kezdődniük, míg a címkéket a @ jellel kell kezdeni.

```
procedure compilepass_2;
var i,j:integer;
      b:boolean;
begin
```

```
 if jn<>$ffff then
 begin
  if ln=$ffff then ErrorExit(4); { Nincs címke, de van ugrás }
  for i:=0 to jn do
   begin
    b:=true;
    for j:=0 to ln do
     if jmps^[i].name=labs^[j].name then
      begin
       b:=false;
       dest^[jmps^[i].offs]:=labs^[j].offs;
      end;
    if b then
     begin
      line:=jmps^[i].l;
      ErrorExit(5); { Nem található meg az adott címke }
     end;
   end;
 end;
 writeln('A második menet kész!');
end;
```

compilepass_2: A fordítás második menete. Az eljárás a tárolt *call/jmp* utasításokhoz megkeresi a megfelelő tárolt címkét, és az utasítások címeit beírja a célhelyre. Ha valamelyik *call/jmp*-hoz nincs megfelelő címke, akkor az *ErrorExit*-et hívja.

A főprogram

```
BEGIN
 clrscr;
 { Beolvassa a fájlok neveit }
 write('A forrásfájl neve:'); readln(ns);
 write('A célfájl neve:');        readln(nd);
 { Létrehozza és inicializálja a dinamikus változókat }
 new(jmps); new(labs); new(dest);
 fillchar(jmps^,sizeof(jmps^),0);
 fillchar(labs^,sizeof(labs^),0);
 fillchar(dest^,sizeof(dest^),0);
 { Megnyitja a forrásfájlt }
 assign(source,ns);
 reset(source);
 { A fordítás két menetben történik }
 line:=0;
 compilepass_1;    { Fordítás: első menet    }
 compilepass_2;    { Fordítás: második menet }
 close(source);
 { Az adatok kiírása a célfájlba }
 assign(f,nd);
```

```
 rewrite(f,1);
 blockwrite(f,dest^,pc);
 close(f);
 { Felszabadítja a heapet, a fordítás sikeres volt }
 dispose(dest); dispose(labs); dispose(jmps);
END. of program
```

A főprogram elég rövid, hiszen az eljárások az összes fontos tennivalót elvégzik. Először bekérjük a fájlok neveit. A program nem ellenőrzi a megadott neveket, ezért óvatosnak kell lennünk, mert fontos fájlokat is letörölhetünk, ha hibás kimeneti nevet adunk meg. A nevek bekérése után létrehozunk három dinamikus változót a *new* eljárással. A *jmps* a *call/jmp* utasításokat tároló tömbre, a *labs* pedig a címkéket tároló tömbre mutat. A *dest* egy 32K méretű bájt tömbre mutat, itt helyezzük el a fordítás eredményét. Megnyitjuk a forrásszöveget, és elindítjuk a fordítás első menetét (*compilepass_1*). Ha a fordító itt hibát észlel, akkor az *ErrorExit* eljáráson keresztül befejeződik a fordítás. Ha nincs hiba, akkor elindul a második menet is. A második menetben meg kell találni minden egyes *call/jmp* utasításhoz a megfelelő címkét. Hiba esetén szintén leállás az eredmény. Ha nincs hiba, akkor a főprogram a megadott néven kimenti az eredményt lemezre, felszabadítja a heapet, és kilép.

A fordító képes észlelni bizonyos hibákat, mégis — mint minden nyelv esetén — elő kell írni néhány szintaktikai szabályt, és ezeket feltétlenül be kell tartani a forrásszövegek írásakor!

Szintaktikai szabályok

– A megjegyzések csak egy sorra vonatkoznak. Ha egy sor a ; karakterrel kezdődik, akkor a fordító nem veszi figyelembe azt a sort, de csakis azt az egyet nem!
– Az elválasztókarakter a szóköz. Minden kulcsszó és paraméter közé legalább egy szóközt kell írni.
– A *FINISH* kulcsszót a forrásszöveg végén kötelező használni. A fordító a *FINISH* szóig fordítja a szöveget.
– A számokat hexadecimálisan kell megadni mindenhol.
– A *REGSET* utasítás formája a következő:

```
regset
 xx yy : xx yy : xx yy
 xx yy : xx yy
.
```

A *regset* kulcsszó után nem állhat paraméter. A következő sorokban kettősponttal elválasztva sorakozhatnak a beállítások, de a legutolsó számpár után nem állhat kettőspont. A beállítás végét a pont jelzi, ezt kötelező kiírni, és mindig új sorban kell elhelyezni.

– A hangjegyek elválasztása egy sorban a következő:

```
C 4 10 : C# 5 8010 0010 : D 0 3
```

Hangjegyeket csak kettősponttal elválasztva szabad felsorolni, de a legutolsó hang után nem állhat kettőspont.

– Az előző két eset kivételével egy sorban csak egyetlen utasítás állhat.
– Egy sor legfeljebb 254 karakterből állhat.
– A kettősponttal való elválasztás esetén a kettőspont elé és mögé kötelező legalább egy-egy szóközkaraktert írni.
– A címkék nevei 8 karakter hosszúak lehetnek — több betűt a fordító nem vesz figyelembe —, és minden címkének kötelezően a @ jellel kell kezdődnie.
– A címkéket egy sorban kell elhelyezni, és a címkék után már nem állhat más utasítás.
– Legfeljebb 1024 címke és 1024 ugrás (*call/jmp*) fordulhat elő a programban.

Mindezeket a szabályokat néhány rövid példával illusztráljuk!

3.4.5. Példaprogramok a lejátszóhoz

Egyszerű hangsor egy csatornára

```
; próbafájl az FM compilerhez
regset                                  ; regiszterbeállítás
 01 20 ; a hullámforma-vezérlés engedélyezése
 20 01 : 40 00 : 60 F8 : 80 12 ; az első operátor paraméterei
 23 01 : 43 00 : 63 F8 : 83 12 ; a második operátor paraméterei
. ; a regiszterbeállítás vége

rpt 0 2
  call @felfele          ; hangsor felfelé
  pause 20               ; szünet
  call @lefele           ; hangsor lefelé
  pause 20               ; szünet
djnz 0                   ; imétlés

end

@felfele
  C 2 0F : D 2 0F : E 2 0F : F 2 0F
  G 2 0F : A 2 0F : H 2 0F : C 3 0F
  ret
```

```
@lefele
  C 3 0F : H 2 0F : A 2 0F : G 2 0F
  F 2 0F : E 2 0F : D 2 0F : C 2 0F
  ret

finish
```

A program csupán a 0. fizikai csatornát használja. A legelső teendő az alapvető operátor-regiszterek beállítása: ADSR, hangerő és frekvenciaszorzó. (Az egyszerűség kedvéért most mindkét operátort azonosan töltöttük fel.) Ez után kettőt töltünk a 0. *repeaterbe*, így a ciklus kétszer fog lefutni. A ciklus belsejéből meghívjuk azt a két szubrutint, amelyek kiadják a hangokat. A programot az *end* utasítással fejezzük be.

Összetett hangsor egy csatornára

A következő program *Morricone* egyik filmzenéjét „játssza". A programban szintén csak a 0. fizikai csatornát használjuk.

```
; próbafájl az FM compilerhez
; Morricone: Godfather
regset
    01 20
    20 01 : 40 00 : 60 ff : 80 22
    23 01 : 43 00 : 63 ff : 83 22
.

 rpt 0 2
  call @part1
  pause 20
  call @part2
  pause 20
  call @part3
  pause 20
  call @part4
  pause 20
  call @part5
  pause 15
   regset
    20 02 : 23 02
   .
 djnz 0

end

@part1
 H 3 10 : E 4 10 : G 4 10 : F# 4 10 : E 4 10 : G 4 10
 pause 4
 E 4 0c : F# 4 10 : E 4 0c : C 4 10
```

```
 pause 6
 D 4 08 : H 3 15
ret

@part2
 H 3 10 : E 4 10 : G 4 10 : F# 4 10 : E 4 10 : G 4 10
 pause 4
 E 4 0c : F# 4 10 : E 4 0c
 pause 4
 H 3 0a : A# 3 0a : A 3 15
ret

@part3
 A 3 10 : H 3 10 : C 4 10 : F# 4 8015 10
ret

@part4
 G 3 10 : A 3 10 : H 3 10 : E 4 8015 10
ret

@part5
 E 3 10 : G 3 10 : D 4 10 : C 4 10 : H 3 10 : D 4 10
 pause 8
 C 4 8 : C 4 10 : H 3 8
 pause 8
 H 3 10
 pause d
 D# 3 8 : E 3 20
ret

finish ;                                                    :-(
```

Akkordok több csatornán

Végül nézzünk meg egy olyan példát, amely több csatornát is megszólaltat a *chnum* parancs használatával. (A négy akkord egyébként egy részlet a *Scorpions* együttes *Born To Touch Your Feelings* című dalából.)

```
; próbafájl az FM compilerhez
; Scorpions: Born To Touch Your Feelings (részlet)
regset
  01 20
  20 01 : 40 00 : 60 f1 : 80 03 : 23 01 : 43 00 : 63 f1 : 83 03
  21 01 : 41 00 : 61 f1 : 81 03 : 24 01 : 44 00 : 64 f1 : 84 03
  22 01 : 42 00 : 62 f1 : 82 03 : 25 01 : 45 00 : 65 f1 : 85 03
  28 01 : 48 00 : 68 f1 : 88 03 : 2b 01 : 4b 00 : 6b f1 : 8b 03
  c0 09 : c1 09 : c2 09 : c3 09
```

```
.
 rpt 1 2
  call @elso
  call @masodik
  call @harmadik
  call @negyedik
  regset
    c0 00 : c1 00 : c2 00 : c3 00
  .
 djnz 1
end
```

A szubrutinokat — helytakarékosság miatt — egymás mellett soroltuk fel, de a begépelés során ezeket természetesen egymás után kell írni!

```
@elso
 rpt 0 2
 chnum 3
 E 3 10
 chnum 1
 H 3 10
 chnum 2
 G 3 10
 chnum 0
 E 4 10
 chnum 1
 H 3 10
 chnum 2
 G 3 10
 djnz 0
ret

@masodik
 rpt 0 2
 chnum 3
 E 3 10
 chnum 1
 C 4 10
 chnum 2
 A 3 10
 chnum 0
 E 4 10
 chnum 1
 C 4 10
 chnum 2
 A 3 10
 djnz 0
ret

@harmadik
 rpt 0 2
 chnum 3
 F# 3 10
 chnum 1
 D 4 10
 chnum 2
 A 3 10
 chnum 0
 F# 4 10
 chnum 1
 D 4 10
 chnum 2
 A 3 10
 djnz 0
ret

@negyedik
 rpt 0 2
 chnum 3
 G 3 10
 chnum 1
 E 4 10
 chnum 2
 H 3 10
 chnum 0
 G 4 10
 chnum 1
 E 4 10
 chnum 2
 H 3 10
 djnz 0
ret

finish ;                                    :-)
```

Nem állítjuk, hogy a dallamokat és a ritmusokat tökéletesen sikerült eltalálnunk, de azért a példák felismerhetők. ;–) Ezekkel a rövid programokkal csupán az volt a célunk, hogy bemutassuk a szintaktikai szabályokat és a különböző lehetőségeket. Sem a példák, sem a programok — a lejátszó és a fordító — nem makulátlanok és nem is optimálisak. Ki-ki próbálja tökéletesíteni és fejleszteni őket!

Az ADLIB kártya — illetve tágabb értelemben az *FM-voice* típusú hanggenerátorok — programozását ezzel be is fejezzük. Reméljük, hogy a példák és a programok jól szemléltetik, mi mindent lehet — és főképpen, mi mindent nem lehet — megoldani az ADLIB-bel. A kártyát talán ma már kissé „butának" nevezhetik egyesek, de véleményünk szerint a feladatát tökéletesen ellátja. Az sem utolsó szempont, hogy a későbbi fejlettebb hangkártyák valamilyen fokon igyekeztek kompatíbilisek maradni az ADLIB-bel, ami azt jelenti, hogy még ma is számolni kell vele. Végeredményben — persze itt szubjektívek leszünk — érdemes időt és energiát áldozni az ilyen hagyományos hangkeltési módszerek megismerésére.

4. A SOUND BLASTER hangkártya

A Sound Blaster a hangdigitalizálás és a digitális hangminták lejátszása terén hozott újat az ADLIB kártyához képest. A kártyán található egy digitális hangprocesszor, amely 8-bites egyszerű hangfeldolgozási alapfeladatok ellátására képes. A kártya ezenfelül természetesen kompatíbilis az ADLIB-bel, tehát megtalálható rajta az összes lehetőség, amelyet az ADLIB programozása során ismertettünk.

Amikor a hangkártyát beépítjük a gépbe, három fontos hardverjellemzőt kell beállítani:

– A hangkártya I/O báziscímét. Ez $220 vagy $240 lehet, a gyári beállítás általában $220. A kártya I/O portjait ehhez a bázishoz képest relatív címükkel szokták megadni. A báziscímet a kártyán *jumperrel* lehet átállítani, erről bővebb információ a kártyához mellékelt *(User's Manual)* kézikönyvben található. A továbbiakban mindenhol azt feltételezzük, hogy a kártya I/O bázisa a $220, így az összes többi I/O címet ennek megfelelően adjuk meg!

– A hangkártya által használt megszakítási vonal számát. A Sound Blaster képes bizonyos műveletek elvégzése után megszakítást kérni. A megszakítási cím a kártyán szintén *jumperrel* állítható, a legtöbb esetben az IRQ5 vagy az IRQ7 a gyári beállítás.

– A hangkártya által használt DMA csatorna számát. A Sound Blaster képes a hangok mintavételezését, illetve a hangminták lejátszását DMA ciklusok alatt végezni, ezáltal a processzor nagymértékben tehermentesíthető. Az alapbeállítású DMA csatorna általában a DMA 1, az előző két esettől eltérően ezt nem lehet megváltoztatni, legfeljebb letiltani a megfelelő *jumperrel*. A későbbi kártyáknál már ez is módosítható, ezért érdemes úgy tekinteni, hogy változó lehet! (A DMA átvitelt később részletesen ismertetjük.)

4.1. ADLIB kompatibilitás

A Sound Blaster „*FM voice*" egysége az ADLIB-hez hasonlóan a $388, $389 I/O címeken érhető el, de a kártya báziscímén — azaz a $220,$221 címen — szintén lehetőség van az FM chip programozására. A lefuttatott programok alapján mindkét cím tökéletesen használható. Az egyetlen jelentős probléma, hogy az FM jelszint alacsonynak tűnik a DSP egység kimeneti jelszintjéhez képest. Ezért a hangerőt szabályozó potenciométert folyamatosan állítgatni kell, ha egyszerre használjuk a DSP és az FM egységet. :–(

4.2. A DSP egység

Az eredeti Sound Blaster kártyákon a digitális egység a *CT-DSP 1321*-es típusszámú digitális hangprocesszor (***Digital Sound Processor***). Az elnevezés nem túl szerencsés, mert a szakemberek a DSP-n általában nem hangprocesszort, hanem jelprocesszort (***Digital Signal Processor***) értenek. A mi esetünkben viszont a DSP ezután mindenhol a hangprocesszort jelenti!

A DSP legfontosabb tulajdonságai a következők:

- 8-bites D/A konverter a hangminták lejátszásához;
- 8-bites A/D konverter a hangok digitalizálásához;
- direkt és DMA adatátvitel mindkét művelethez;
- állítható mintavételi frekvencia a felvételhez 4 kHz-től 15 kHz-ig;
- állítható mintavételi frekvencia a lejátszáshoz 4 kHz-től 44.1 kHz-ig;
- automatikus megszakításkérés a felvétel/lejátszás végén;
- hardveres hangkitömörítési (dekompressziós) eljárások (1:2, 1:3, 1:4) lejátszás-kor;
- MIDI csatlakoztatási és vezérlési lehetőség.

A DSP a következő I/O címeken érhető el:

Relatív I/O cím	Alapbeállítású I/O cím	DSP funkció
Base+$06	$226 *(write only)*	*DSP Reset* A DSP egység alapállapotba hozása
Base+$0A	$22A *(read only)*	*Read Data* Adat olvasása az ADC-ről
Base+$0C	$22C *(read/write)*	*Write command/data* Parancs/adat írása a DAC-ra *Read Buffer Status* Bufferállapot olvasása
Base+$0E	$22E *(read only)*	*Data Available, IRQ acknowledge* Adatérvényesség olvasása

Az egyes alapműveleteknek — például alapállapotba hozás, direkt írás és olvasás — szigorú szabályai vannak, amelyeket feltétlenül be kell tartani, máskülönben a DSP hibásan fog működni. A fejezet további részében ezeket a szabályokat és különböző példákat ismertetünk, amelyek bemutatják a DSP megfelelő használatát.

4.2.1. A DSP egység inicializálása

Az inicializálási folyamat két dologra is jó: egyrészt — mint ahogyan azt a neve is mutatja — alapállapotba hozza a DSP egységet, másrészt kitűnő tesztelési lehetőség a DSP egység meglétének detektálására. Ezzel a módszerrel a Sound Blaster kártya jelenléte is detektálható. Az inicializálás folyamata a következő:

1. Ki kell küldeni a RESET portra a $01 értéket. Ez jelzi a DSP számára az inicializálási folyamat kezdetét.
2. Várakozni kell legalább 3 μs ideig a parancs elfogadására.
3. A RESET portra ki kell írni egy $00 értéket.
4. Folyamatosan figyelni kell a DATA AVAILABLE port 7. bitjét, hogy az adat érvényessé válik-e. Az adat érvényességét a 7. bit 1-es állapota jelzi. Ha ez nagyjából 100-200 leolvasás után sem következik be, akkor valószínűleg nincs DSP egység az adott báziscímen.
5. Az adatérvényesség bekövetkezése után folyamatosan figyelni kell a READ DATA portot, ahol $AA értéknek kell megjelennie. Ezzel jelzi a DSP az inicializálás hibátlan végrehajtását. Ha a $AA érték kb. 10 000 olvasás után sem érkezik meg, akkor az inicializálás nem sikerült, vagy nincs DSP egység.

Az inicializálási folyamat 100–120 μs-ig tart. A fenti feladatokat megvalósító eljárást a később ismertetésre kerülő *DSP.PAS* unit tartalmazza.

4.2.2. Parancs és adat kiírása a DSP egységre

Az írás a WRITE COMMAND/DATA porton keresztül történik. Ez a port írható és olvasható is (R/W). Olvasáskor a 7. bit 1-es állapota a regiszter foglaltságát jelzi, azaz a DSP ilyenkor még az előző parancson dolgozik. Íráskor ezért mindenképpen meg kell várni, amíg ez a bit 0-ba áll, máskülönben a kiírt újabb parancs az előzőt megzavarja. Az írás folyamata tehát a következő:

1. Meg kell várni, amíg a READ BUFFER STATUS regiszter 7. bitje 0-ba áll.
2. Ki kell írni az új parancsot vagy adatot a WRITE COMMAND/DATA portra.

Amint azt a táblázatban láthattuk, mindkét port a $22C címen érhető el, de az egyik csak olvasható, míg a másik csak írható.

4.2.3. Adat beolvasása a DSP-ről

Az adatbeolvasás folyamata az inicializálás utolsó két lépéséhez hasonló. Bármilyen adat beolvasása előtt meg kell várni, hogy a DATA AVAILABLE port 7. bitje 1-be álljon. Ez a bit jelzi az adat érvényességét. Az olvasás így az alábbi részfeladatokra bontható:

1. Várakozni kell, amíg a DATA AVAILABLE port 7. bitje 1-es állapotba kerül.
2. Be kell olvasni az adatot a READ DATA portról.

4.2.4. DSP átviteli módok felvételkor

A hangok digitalizálása kétféle módon történhet: direkt módon és DMA átvitellel. Az első esetben a program saját maga kezeli a READ DATA portot, és így közvetlenül minden adatot egyenként kell beolvasni és eltárolni. A második esetben a DMA vezérlőt kell a megfelelő módon programozni. A hangkártya a megszabott mintavételi időközönként DMA átvitelt kér, így a digitalizálás közvetetten, a processzor kikerülésével zajlik le. A digitalizálás során a Sound Blaster nem képes hardverkompresszióra, így az csak 8-bites normál módon valósul meg mindkét esetben. A tipikus mintavételezési-frekvencia-értékeket a következő táblázat tartalmazza:

Az adatátvitel fajtája	A mintavételezési frekvencia
8-bites normál	4 kHz ... 13 kHz
8-bites nagy sebességű	13 kHz ... 15 kHz (???)

4.2.5. DSP átviteli módok lejátszáskor

A már digitalizált hangok lejátszása — a felvételhez hasonlóan — szintén történhet direkt módon és DMA ciklusok alatt. Az első esetben a hangmintákat sorban ki kell küldeni a WRITE COMMAND/DATA portra, míg a második esetben ezt a DMA vezérlő teszi meg. A lejátszáskor választani lehet a normál 8-bites és a tömörített hangminták lejátszása között. A tömörítést a kártya hardver úton oldja meg, ezt nevezzük ADPCM kódolásnak. A hangminták tömörítését szoftver úton kell megoldanunk, mert ezt a kártya nem teszi meg, visszajátszáskor viszont hardver segíti a tömörített hangminták átalakítását. Az eljárást hardveres dekompressziónak nevezzük, a kódolás pedig ADPCM rendszerű eljárással történik.

4.2.6. ADPCM

Az ADPCM (*Adaptive Pulse Code Modulation*) egy olyan egyszerű tömörítési eljárás, amelyet előszeretettel alkalmaznak digitális hangminták tárolásakor. A tömörítés célja a felhasznált memória méretének csökkentése. Az eljárás lényege, hogy a digitalizált hangminta nem az egyes mintavételi időkben vett minták abszolút értékeit tartalmazza, hanem mindig az előző és a következő minta közötti különbséget. Vegyünk például egy egyszerű mintasorozatot, ahol a tárolt bájtok abszolút értékei a következők:

```
$03, $07, $01, $02, $08, $00
```

Ha most ezen értékek különbségeit tároljuk, akkor a következő sorozatot kapjuk:

```
+4,  -6,  +1,  +6,  -8
```

A minták közötti különbséget általában egy előre meghatározott bitszámon tárolják, így akkor kapnak új mintát, ha a jel túllépi a legnagyobb differenciát. Amint látható, a fenti különbségsorozat semmit nem ér akkor, ha nincs meg az a kezdeti kiinduló érték, amelyhez azután a többi értéket viszonyítani lehet. Ezt az alapértéket **referenciabájtnak** hívjuk. A hangkártya lehetőséget nyújt referenciabájtos és referenciabájt nélküli átvitelre is. A referenciabájt nélküli átvitel akkor lehet érdekes, ha az előző átvitel utolsó értéke beleesik az ábrázolható differenciába. Ilyenkor nincs szükség újabb referenciabájt megadására, hanem az előző átvitel végét tekinthetjük alapjelszintnek. Az alábbi táblázatban a DMA lejátszásokat és azok legnagyobb mintavételezési frekvenciáit tüntettük fel:

Az adatátvitel fajtája	A mintavételezési frekvencia
8-bites normál	4 kHz ... 23 kHz
4-bites ADPCM	4 kHz ... 12 kHz
2.6 bites ADPCM	4 kHz ... 13 kHz
2-bites ADPCM	4 kHz ... 11 kHz
8-bites nagy sebességű (???)	23 kHz ... 44 kHz (???)

4.2.7. DSP parancsok

A DSP programozása alapvetően parancsvezérelt. Ez annyit jelent, hogy a DSP-t a WRITE COMMAND/DATA portra írt vezérlőparanccsal utasítani kell bármely tevékenység elkezdésére és végrehajtására. A vezérlőparancsok 8-bites konstans értékek, amelyek azután a DSP-t felkészítik a további beállításokra vagy egyéb vezérlőadatokra.

Az alábbi táblázatban a DSP parancsait foglaltuk össze:

Kód	Parancs	Parancsfunkció
$40	*Set Sample Rate*	a mintavételezési idő beállítása
$D1	*Speaker On*	a DSP kimenetének bekapcsolása
$D3	*Speaker Off*	a DSP kimenetének kikapcsolása
$D8	*Get Speaker Status*	a kimenet állapotának lekérdezése
$10	*Direct Play (DAC)*	direkt módú lejátszás (DAC)
$20	*Direct Record (ADC)*	direkt módú felvétel (ADC)
$D0	*DMA Transfer Stop*	a DMA átvitel felfüggesztése
$D4	*DMA Transfer Cont.*	a DMA átvitel folytatása
$14	*Normal DMA Play*	normál 8-bites DMA lejátszás (DAC)
$24	*Normal DMA Record*	normál 8-bites DMA felvétel (ADC)
$74	*ADPCM 4 bit*	4-bites ADPCM lejátszás DMA alatt
$75	*ADPCM 4 bit, ref.*	4-bites ADPCM referenciabájt-beállítás
$76	*ADPCM 2.6 bit*	2.6 bites ADPCM lejátszás DMA alatt
$77	*ADPCM 2.6 bit, ref.*	2.6 bites ADPCM referenciabájt-beállítás
$16	*ADPCM 2 bit*	2-bites ADPCM lejátszás DMA alatt
$17	*ADPCM 2 bit, ref.*	2-bites ADPCM referenciabájt-beállítás
$E1	*Get DSP Version*	a DSP verziószámának lekérdezése
$30	*MIDI Read Poll*	MIDI olvasás
$31	*MIDI Read IRQ*	MIDI megszakításos olvasás
$38	*MIDI Write Poll*	MIDI írás

A kártya verziószámától függően esetleg más parancsok is előfordulhatnak. A 2.0-s vagy annál magasabb verziójú DSP-nél a nagy sebességű átvitelhez és a DSP azonosításához bevezettek néhány újabb DSP parancsot:

Kód	Parancs	Parancsfunkció
$E0	*DSP ID*	a DSP egység azonosítása (???)
$48	*Set High Speed Size*	nagy sebességű átvitel méret (???)
$91	*High Speed Play*	nagy sebességű lejátszás DMA alatt (???)
$99	*High Speed Record*	nagy sebességű felvétel DMA alatt (???)

4.3. A DSP egység programozása

A programozás megkönnyítéséhez készítettünk egy unitot, amelyben a fent leírt alapfeladatokhoz megtalálhatók a megfelelő eljárások. A unitban definiáltuk azokat a leggyakrabban használt állandókat, amelyek a DSP programozásakor szükségesek lehetnek.

4.3.1. A DSP unit

```
unit dsp; { DSP.PAS }

interface
```

Típusok, állandók

```
type
     { Egy 64K széles bájt tömb típus definiálása          }
     memory = array[ 0..65534] of byte;
     { Mutató definiálása a 64K széles bájt tömb számára   }
     memptr = ^memory;
```

memory: A *memory* egy 65535 elemű bájtokat tartalmazó tömb. Ilyen tömb kialakítására csak a heapben van lehetőség, ezért célszerű létrehozni egy olyan pointert, amely erre a tömbre mutat.

memptr: A *memory* tömbre mutató pointer. A pointer használatával lehetőség nyílik ilyen nagy méretű összefüggő területek kezelésére. A heapet a *getmem* vagy a *new* eljárással lehet lefoglalni, és a *freemem* vagy a *dispose* eljárással felszabadítani.

A uniton belül nem használjuk ezeket a típusokat, de a későbbi programokban még szükség lesz rájuk.

```
const
     { A SoundBlaster kártya rendszerinformációi          }
     BaseAddr   : word = $220; { SB I/O báziscím          }
     IRQnumber  : byte =  $05; { SB megszakítási vonal    }
     DMAchannel : byte =  $01; { SB DMA csatornaszám      }
```

BaseAddr: A kártyán aktuálisan beállított I/O báziscím. A unit minden eljárása és függvénye ez alapján számolja a DSP regisztereinek címeit, így egy esetleges más beállítású kártyán a programban csak ezt a változót kell módosítani ahhoz, hogy az eljárások más I/O címeket használjanak.

IRQnumber: A kártyán aktuálisan beállított megszakítási vonal. A megszakítási vonal határozza meg, hogy a Sound Blaster milyen számú INT utasítást generál.

DMAchannel: Az aktuálisan beállított DMA csatornaszám.

A következő állandók a DSP parancsait definiálják. A parancsokhoz rendelt kódokat kell kiírni a WRITE COMMAND/DATA portra, hogy a DSP a megfelelő üzemmódba kapcsoljon. Egyes parancsoknál egyéb paraméterek is lehetnek, ezeket majd a példaprogramokban mutatjuk be.

```
    { DSP parancsok }

    SetSampleRate = $40;      { A mintavételezési frekvencia  }
                              { beállítása                    }

    { A DSP kimenetével kapcsolatos parancsok                 }
    SpeakerOn     = $d1;      { A kimenet bekapcsolása        }
    SpeakerOff    = $d3;      { A kimenet kikapcsolása        }
    SpeakerStatus = $d8;      { A kimenet állapota            }

    { Direkt módú lejátszás és felvétel                       }
    DirectDAC     = $10;      { Direkt lejátszás (DAC)        }
    DirectADC     = $20;      { Direkt felvétel  (ADC)        }

    { A DMA átvitellel kapcsolatos parancsok                  }
    DSP_DMA_stop  = $d0;      { A DMA átvitel felfüggesztése  }
    DSP_DMA_cont  = $d4;      { A DMA átvitel folytatása      }

    NormalDMA_DAC = $14;      { Normál 8-bites lejátszás (DAC)}
    NormalDMA_ADC = $24;      { Normál 8-bites felvétel (ADC) }

    { Impulzuskód-modulált DMA átviteli parancsok             }
    ADPCM4_DMA    = $74;      { 4-bites ADPCM                 }
    ADPCM4r_DMA   = $75;      { 4-bites ADPCM referenciával   }
    ADPCM26_DMA   = $76;      { 2.6 bites ADPCM               }
    ADPCM26r_DMA  = $77;      { 2.6 bites ADPCM referenciával }
    ADPCM2_DMA    = $16;      { 2-bites ADPCM                 }
    ADPCM2r_DMA   = $17;      { 2-bites ADPCM referenciával   }

    GetDSPver     = $e1;      { A DSP chip verziószámának     }
                              {lekérdezése                    }

function  DSPreset:boolean;           { DSP inicializálás     }
procedure DSPwrite(DSPdata:byte);{ DSP parancs/adat           }
function  DSPread:byte;          { DSP adatbeolvasás          }
procedure DSPspeakerOn;          { DSP kimenet bekapcsolása   }
procedure DSPspeakerOff;         { DSP kimenet kikapcsolása   }
procedure Pause(Atime:word);     { Várakozás                  }

implementation
```

Eljárások, függvények

```
function DSPreset:boolean;                    assembler;
asm
  mov   dx,BaseAddr
  add   dl,$06                    { DX=$226, DSP RESET          }
  mov   al,$01
  out   dx,al                     { Egy $01 érték kiküldése     }
  mov   cx,100
  rep lodsb                       { Várakozás                   }
  xor   al,al
  out   dx,al                     { Egy $00 érték kiküldése     }
  add   dl,$08                    { DX=$22E, DATA AVAILABLE     }
  mov   cx,128
@wait1:                           { Várakozás, amíg az adat     }
  in    al,dx                     { érvényes lesz               }
  and   al,$80
  jnz   @test2                    { Kiugrás, ha érvényes        }
  loop  @wait1
  jmp   @bad                      { Hiba, ha lejárt a ciklus    }
@test2:
  mov   cx,10240
  sub   dl,$04                    { DX=$22A, DSP READ DATA      }
@wait2:
  in    al,dx
  cmp   al,0aah                   { A $AA értéket kell adnia    }
  jz    @good                     { Ugrás, ha a RESET sikeres   }
  loop  @wait2
@bad:
  xor   al,al                     { Hiba: AL = false            }
  jmp   @exit
@good:
  mov   al,1                      { Nincs hiba: AL = true       }
@exit:
end;
```

DSPreset: A Sound Blaster DSP egységének alapállapotba hozása. A függvény — az előbbiekben részletesen ismertetett módon — megpróbálja inicializálni a DSP egységet. Ha ez sikerül, vagyis a beállított időintervallumokon belül a megfelelő értékeket észleli, akkor igaz értékkel tér vissza. Ha az időzítési ciklusok lejártával sem érzékeli a megfelelő eredményeket, akkor a DSP nem működik helyesen, ilyenkor hamis értékkel tér vissza.

```
procedure DSPwrite(DSPdata:byte);         assembler;
asm
  mov   dx,BaseAddr
  add   dl,0ch                    { DX=$22C, READ BUFFER STATUS }
```

```
@c1:
  in    al,dx
  and   al,80h                { Szabad már a regiszter?     }
  jnz   @c1                   { Várakozás, ha még nem       }
  mov   al,DSPdata            { AL=adat/parancs             }
  out   dx,al                 { WRITE COMMAND/DATA          }
end;
```

DSPwrite: Egy adat/parancs kiküldése a DSP-nek. Az eljárás az előbbiekben ismertetett elven kiküld egy bájtot a WRITE COMMAND/DATA regiszterbe.

```
function DSPread:byte;                          assembler;
asm
  mov   dx,BaseAddr
  add   dl,0eh                { DX=$22E, DATA AVAILABLE     }
@wait:
  in    al,dx                 { Várakozás, amíg az adat     }
  and   al,80h                { érvényes lesz               }
  jz    @wait
  sub   dl,4                  { DX=$22A, DSP READ DATA      }
  in    al,dx                 { Az adat beolvasása az AL-be }
end;
```

DSPread: Egy adat beolvasása a DSP-ről. A függvény a DSP READ DATA regiszterét olvassa be az előbbiekben ismertetett módon.

```
procedure DSPspeakerOn;
begin
 DSPwrite(SpeakerOn);         { A kimenet bekapcsolása      }
end;
```

DSPspeakerOn: A DSP kimenetének bekapcsolása, a hangkiadás engedélyezett.

```
procedure DSPspeakerOff;
begin
 DSPwrite(SpeakerOff);   { A kimenet kikapcsolása          }
end;
```

DSPspeakerOff: A DSP kimenetének kikapcsolása, a hangkiadás tiltott.

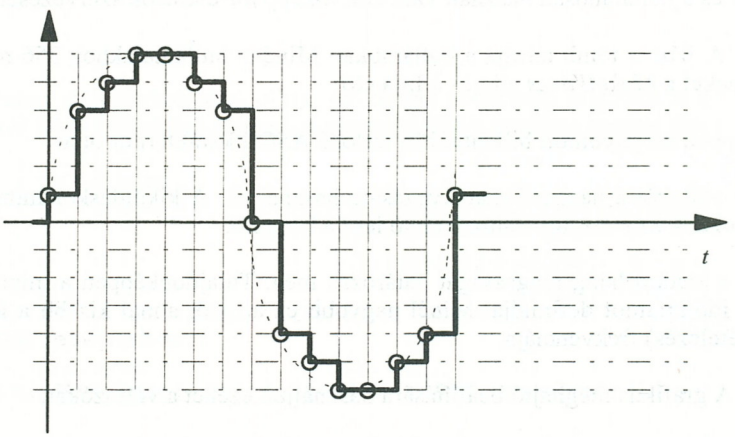
```
procedure Pause(Atime:word);                    assembler;
asm
  mov  cx,Atime               { A várakozási állandó a CX-be}
@wait:
  nop
  nop
  loop @wait                  { Várakozás                   }
end;
```

Pause: várakozás adott ideig. Az eljárás *Atime* számú üres ciklust hajt végre. Akkor érdemes alkalmazni, amikor ms-nál rövidebb időket kell várni.

```
end. of unit
```

4.3.2. Hullámformák előállítása a DSP egységgel

Az első példában a leggyakoribb hullámformákat állítjuk elő a DSP-vel. A hullámok definiálása igen egyszerű: a hullámgörbe bizonyos — általunk meghatározott — pontjaiban a függvény értékeit egyszerűen tároljuk egy bájt típusú tömbben. Ha ezután ezeket az értékeket sorban kiküldjük a DSP-re, akkor a kimeneten az adott hullám jelenik meg. Példaképpen nézzük meg a szinuszhullámot:



32. ábra. A szinuszhullám diszkrét értékű pontjai

A pontozott vonal az ideális szinuszhullámot jelöli, a kis körök pedig azokat a helyeket, amelyeket ehhez képest a DSP-vel elő tudunk állítani. (Emlékezzünk vissza a kvantálási hibára!) A kimeneti függvényt — ez kerül a hangszóróra — a vastag vonal jelenti. Ez az ábra természetesen igen elnagyolt, hiszen a valóságban sokkal több állapotot tudunk meghatározni, egész pontosan 256-ot.

Fontos tudni, hogy a DSP a hangmintákat 128 egységgel eltolva értelmezi. Ez azt jelenti, hogy a kimeneten akkor jelenik meg 0 szint, amikor a DSP-re 128-at küldünk ki, míg 000 esetén a negatív maximum, 255 esetén pedig a pozitív maximum jelenik meg. A programban ezért a hullámokat shortint típusú (ez bájtot jelent) tömbökben tároljuk, és a minták kiküldésekor az értékeket korrigáljuk, azaz minden mintát megnövelünk 128-cal.

```
program WavesForDSP; { waves.pas }

uses crt,dsp,graph;
```

Változók

```
var
   i,j : word;
 Waves : array[ 0..3,0..255] of shortint;
     c : char;
     r : real;
     p : integer;
gd,gm : integer;
```

i,j: Az *i* és a *j* általánosan használt változók főképp *for* ciklusok szervezéséhez.

Waves: A *Waves* tömb tárolja a hullámokat. Minden hullámalakhoz 256 mintát tárolunk, ezeket a *MakeWaves* eljárás állítja elő.

c: Az éppen megnyomott billentyűhöz tartozó ASCII karaktert tárolja.

r: Az *r* változót a hangerő szabályozására használjuk. A kiküldéskor minden mintát megszorzunk az *r*-rel, így változtatható lesz a hangerő.

p: A *p* a kiadott hang magasságát határozza meg. Tulajdonképpen a minták kiadása közötti időtartamot definiálja. Minél nagyobb ez az idő, annál kisebb a lejátszás (a mintavételezés) frekvenciája.

gd,gm: A grafikus meghajtó beállítására használjuk ezeket a változókat.

Eljárások

```
procedure MakeWaves;        { A megfelelő hullámformák generálása }
begin
 for i:=0 to 255 do
  begin
     { Szinusz    }
   Waves[ 0,i] :=round(32*sin(i*pi/32));
     { Fűrészfog }
   Waves[ 1,i] :=(i and 63)-31;
     { Háromszög }
   if (i and 32)=0 then Waves[ 2,i] :=(i and 31)*2-32
                   else Waves[ 2,i] :=32-(i and 31)*2;
     { Négyszög   }
   if (i and 32)=0 then Waves[ 3,i] :=32
```

```
                    else Waves[ 3,i] :=-32;
  end;
end;
```

MakeWaves: Ez az eljárás állítja elő a hullámokat a Waves tömbben.

```
procedure DrawDiagrams;{ A jelalakok grafikus megjelenítése }
begin
 for j:=0 to 3 do
  begin
   setcolor(white);
   line(0,60+j*120,255,60+j*120);
   moveto(0,60+j*120-Waves[ j,0] );
   setcolor(j+11);
   for i:=0 to 255 do
     lineto(i,60+j*120-Waves[ j,i] );
  end;
end;
```

DrawDiagrams: A jelalakok ábrázolása. Az eljárás grafikusan megjeleníti a négy kiszámolt jelalakot négy különböző színnel. A program VGA grafikus kártyát feltételez!

A főprogram

```
BEGIN
 clrscr;
 { Inicializálás és kilépés, ha hiba történt     }
 if not DSPreset then
  begin
   writeln('Nem lehet inicializálni a DSP egységet!');
   halt;
  end;
```

Hívjuk a DSP egységet. Ha inicializálás közben hiba lépett fel, akkor a program leáll. Ha nincs hiba, akkor beállítjuk a grafikus képet, kiszámítjuk és kirajzoljuk az egyes jelalakokat.

```
 gd:=detect;
 { Ide a saját BGI elérési útvonalat kell írni! }
 initgraph(gd,gm,'d:\bp\bgi');

 MakeWaves;          { Előállítja a hullámokat      }
 DrawDiagrams;       { Kirajzolja a hullámokat      }

 DSPspeakerOn;       { Bekapcsolja a DSP kimenetét }

 { Alapbeállítások }
```

```
 j:=0;       { Hullámszám      0-3      }
 r:=1;       { Hangerő         0.1-3.7  }
 p:=100;     { Hangmagasság    10-2000  }
```

A kezdeti paraméterek beállítása után egy ciklus következik, amely megvizsgálja a billentyűzetet. Ha volt billentyűlenyomás, akkor a megfelelő műveletet el kell végezni, ha viszont nem volt, akkor következhet a hangminták kiküldése a DSP-re.

```
 repeat
  { Billentyűzetlekérdezés }
  if keypressed then c:=upcase(readkey)
                else c:=#0;
  case c of
   '1'..'4' : begin              { Hullámalak-kiválasztás }
                DSPspeakerOff;
                j:=ord(c)-ord('1');
                Pause(1000);
                DSPspeakerOn;
              end;
   '+' : if r<3.7 then r:=r+0.1;     { Hangerőnövelés          }
   '-' : if r>0.1 then r:=r-0.1;     { Hangerőcsökkentés       }
   ',' : if p>10   then dec(p,5);    { Frekvenciacsökkentés    }
   '.' : if p<2000 then inc(p,5);    { Frekvencianövelés       }
  end;

  { A következő ciklus a hangot adja ki                        }
  for i:=0 to 255 do
   begin
    DSPwrite(DirectDAC);                      { Direkt DSP írás     }
    DSPwrite(round(r*Waves[ j,i ])+128);      { A következő minta   }
    Pause(p);                                 { Várakozás két minta között}
   end;
 until c=#27; { Kilép az Esc gomb megnyomásakor }
```

Amint látható, a hangminták kiadásakor mindig két bájtot kell kiküldeni a DSP-re. Az első bájt a parancs, azaz a 8-bites direkt módú lejátszás parancsa, a második bájt pedig a hullámalak következő adata. A minta kiadása után a *Pause* eljárással várakozunk, ez a két minta közötti időtartamot szabja meg.

```
 DSPspeakerOff;
 closegraph;
END. of program
```

A program a következő billentyűket használja:

1–4:	Hullámforma-kiválasztás
+:	A hangerő növelése
–:	A hangerő csökkentése

>: A frekvencia növelése
<: A frekvencia csökkentése
ESC: Kilépés

Figyeljük meg, hogy a szinusz- és a háromszöghullám nagyon hasonló hangot ad. A hullámformák előállításánál láthattuk, hogy a háromszög már igen kevés összetevő hullám után létrejön, ebből ered a hasonlóság. Az is érdekes, hogy a szinusz és a háromszög halkabbnak — kisebb teljesítményűnek — hallatszik, mint a másik két hullám. Ez a hullámok kitöltési tényezőjétől és a felharmonikusok arányától függ. Célszerű kísérletezni esetleg más hullámformák kialakításával is, érdekes hanghatásokat lehet elérni, hiszen a hullám alakja teljesen szabálytalan is lehet.

4.3.3. Hangfelvétel direkt módon a DSP-vel

A közvetlen programvezérelt felvétel nagyon egyszerű: minden egyes mintavételkor a DSP számára ki kell küldeni egy *Normal Record ($20)* parancsot, amely elindít egy A/D konverziót. Ez után be kell olvasni a digitalizált mintát a READ DATA portról:

```
DSPwrite(DirectADC);
sample:=DSPread;
```

A *sample* a beolvasott minta, típusa bájt. A most következő példa ezt a fajta felvételt mutatja be. A program beolvas 65520 mintát, majd ezt tárolja egy fájlban.

```
program DirectDSPrecord;        { DIR_REC.PAS }

uses crt,dsp;

 var
     f : file;              { Egy típus nélküli fájlváltozó a      }
                            { minták tárolásához                   }
     i : word;              { Az i a mintákat számolja (0..65519)  }
     m : memptr;            { Egy 65520 bájtos terület mutatója;   }
                            { a területet a heapben foglaljuk le   }
BEGIN
 clrscr;
 writeln('*** Direkt módú felvétel a DSP-vel ***');

 { Inicializálás és a DSP veziószám lekérdezése                    }
 if DSPreset then
  begin                     { Ha a DSP helyesen működik...         }
   DSPwrite(GetDSPver);     { Verziószám-lekérdezés                }
                            { Lekérdezzük mindkét számot           }
   writeln('DSP verziószám:',DSPread,'.',DSPread);
```

```
  end
 else
  begin                          { Ha a DSP hibásan működött      }
   writeln('Nem lehet inicializálni a DSP egységet!');
   halt;                         { Kilépés a programból           }
  end;

 getmem(m,65520); { A memóriaterület lefoglalása                  }
 fillchar(m^,65520,0);  { és nullázása                            }

 write('Felvétel indul az Enter lenyomásakor ... ');
 readln;
 writeln('Felvétel!');

 asm cli end;                   { A megszakításokat a felvétel    }
                                { idejére letiltjuk               }

 for i:=0 to 65519 do           { 65520 mintavétel                }
  begin
   DSPwrite(DirectADC); { Direkt mintavétel ... }
   m^[ i] :=DSPread;    { Az adat tárolása      }
   pause(10);           { Várakozás             }
  end;

 asm sti end;                   { A megszakítások engedélyezése}

 writeln('Kész!');

 { A felvett pár másodperces hangminta lemezre mentése            }
 assign(f,'proba.sam');
 rewrite(f,1);
 blockwrite(f,m^,65520);
 close(f);

 write('[ENTER]');
 readln;
 freemem(m,65520);        { A heap felszabadítása, kilépés       }
END. of program
```

Hogyan működik ez az igen bonyolult program? Nézzük sorban! A DSP egység sikertelen inicializálása esetén a program azonnal kilép, hiszen ekkor nemigen lehetne felvételt készíteni. Ha a DSP megfelelően működik, akkor a *GetDSPver* paranccsal lekérdezzük a DSP egység verziószámát. Amint látható, a parancs után a DSP kétszeri olvasásával áll elő a fő- és az alverzió szám. Egy Sound Blaster 2.0 esetén ez 2.0, esetleg 2.1 lehet. (A verziószám lekérdezése egyébként nem szükséges a felvételhez, csupán azért írtuk be a programba, hogy megismertessük ennek a módját is.) Ez után a *getmem* eljárással lefoglalunk 65520 bájt memóriát a heapben, ez tárolja majd a mintákat. A felvétel idejére letiltjuk a megszakításokat, hogy a két mintavétel között eltelt idő minden esetben azonos

legyen. A felvételt egy egyszerű *for* ciklus végzi: a *Pause* utasítás által megadott idő lejártával mindig újabb mintát veszünk, és ezt tároljuk a memóriában. Ha a tömb megtelik, újra engedélyezzük a megszakításokat, és a hangmintát lemezre mentjük *proba.sam* névvel.

A program — noha működik — sok olyan hátrányos tulajdonságot rejt magában, amely miatt profi hangfelvételi célokra nem alkalmas. Nézzük meg most ezeket a hátrányos tulajdonságokat!

– A felvétel ideje alatt a megszakítások tiltva vannak, ezért a futásba nem lehet beavatkozni. Hiába nyomogatjuk a *Break* gombot, a programra ez nincs túl sok hatással. Ezt ki lehet küszöbölni, ha a megszakításokat a felvétel idejére nem tiltjuk le, de ekkor elronthatjuk az egyenletes mintavételi időt.

– A hangfelvétel rövid ideig tart. Sajnos a hangminták tárolásához igen nagy mennyiségű memória kell. Az igazi megoldás az lenne, ha a digitalizált hang egyből a háttértárolóra kerülne, de ez a direkt felvételnél nem megoldható, mert a lemez írása több időt vesz igénybe, mint amennyi két mintavétel között rendelkezésünkre áll.

– A felvétel a processzor teljes idejét elveszi. Ez a PC DOS esetén nem túl zavaró, hiszen úgyis csak egyetlen programot lehet futtatni egy időben, más multitaszkos operációs rendszerek esetén azonban az ilyen drasztikus processzor-leterhelés nem megengedett.

– A felvétel hosszúságát és a végrehajtás sebességét a processzor órajele határozza meg. Gyorsabb gépeken a felvétel rövidebb ideig tart, de a mintavételi frekvencia nagyobb, míg lassabb gépeknél a mintavételezés frekvenciája kisebb, ezért a felvétel tovább tart.

4.3.4. Hangfelvételek visszajátszása direkt módon

A közvetlen hanglejátszás a felvételhez hasonló elven történik. A hullámformák előállításánál már ezt tettük: egy memóriában lévő hang mintáit egyesével kiküldtük a DSP-re. A felvételek lejátszásának is ez a módja. Az alábbi programmal az előző felvevőprogrammal felvett hangmintát tudjuk visszahallgatni.

```
program DirectDSPplay; { DIR_PLAY.PAS }

uses crt,dsp;

 var
     f : file;
```

```
    l : longint;
    i : word;
    m : memptr;

BEGIN
 clrscr;
 writeln ('* Hangminta lejátszása direkt módon a DSP-vel *');
 assign(f,'proba.sam');
 reset(f,1);
 l:=filesize(f);
 if l>65520 then          { Kilépés, ha a fájl túl hosszú  }
  begin
   writeln('Túl hosszú a fájl! (Max 65520 bájt lehet)');
   close(f);
   halt;
  end;

 if DSPreset then
  begin          { A DSP működik: verziószám-lekérdezés }
   DSPwrite(GetDSPver);
   writeln('DSP verziószám:',DSPread,'.',DSPread);
  end
 else            { A DSP nem működik: kilépés          }
  begin
   writeln('Nem lehet inicializálni a DSP egységet!');
   close(f);
   halt;
  end;

 { A memória lefoglalása és az adatok betöltése }
 getmem(m,l);
 blockread(f,m^,l);
 close(f);

 write('Lejátszás indul az Enter lenyomásakor ... ');
 readln;
 writeln('Lejátszás !');

 asm cli end;      { A megszakítások tiltása a lejátszás alatt }

 DSPspeakerOn;           { A kimenet engedélyezése     }

 for i:=0 to l-1 do           { Lejátszás              }
  begin
   DSPwrite(DirectDAC); { DSP direkt írás        }
   DSPwrite(m^[ i] );   { A következő minta      }
   Pause(100);          { Várakozás              }
  end;

 DSPspeakerOff;               { A kimenet letiltása   }
```

```
 asm sti end;        { A megszakítások engedélyezése      }

 write('Kész. [Enter]');

 readln;
 freemem(m,l);       { A heap felszabadítása, kilépés }
END. of program
```

A program eléggé egyszerű, hiszen az előzőekben elvégzett feladatok logikusan megfordulnak a lejátszás alatt. Lényeges, hogy a mintavételi idő ebben az esetben jóval nagyobb, mert a DSP „sokkal" gyorsabban tud lejátszani, mint felvenni. Ezért, hogy a lejátszás ideje nagyjából ugyanaz legyen, mint a felvételé, itt a *Pause(100)* utasítás nagyobb időközöket szab meg.

4.3.5. Hosszabb hangfelvételek készítése

Az előző programok legfeljebb 64 Kbájt méretű hangfájlokat tudtak kezelni. Próbáljunk meg hosszabb hangfelvételeket készíteni! A Pascal szabad memóriáját a heap mérete szabja meg. Ez *real* módban (a processzor normál üzemmódjában) nagyjából 580-600 Kbájt, hiszen ilyenkor a maximálisan megcímezhető memória 1 Mbájt. Ezzel szemben *protected* módban (a processzor védett üzemmódjában) a gépben lévő összes memória nagy része kihasználható (akár 15 Mbájt is). Hosszabb felvételek készítésénél tehát le kell foglalni a heapből a lehető legnagyobb területet, és oda kell tölteni a digitalizált hangmintát. Sajnos azonban az Intel processzorok szegmentált memóriakezelése miatt a Pascal *getmem* és *new* területfoglaló eljárásai legfeljebb 64 Kbájt méretű összefüggő területeket tudnak lefoglalni. A mintavételezés során tehát ezek között a területek között váltogatni kell, amikor valamelyik megtelt. Ez egyenetlenné teszi a mintavételezést, hiszen az átváltások több időt igényelnek, mint az egyszerű tárolás. Ha a tárolást ezen a módon oldjuk meg, akkor a visszajátszás során a hangmintában akadozások, zavarok jelentkeznek 64 Kbájtonként. Ezért meg kell próbálni úgy tárolni a mintákat, hogy az minden esetben azonos időt vegyen igénybe. A megoldást a Pascal heapkezelésében kell keresni. Ha egy olyan programot indítunk el, amelyben memóriafoglalások vannak, akkor ezen foglalások növekvő cím szerint egymás mögött veszik igénybe a memóriát. Ez csak és csakis akkor igaz, amikor a program elkezd futni, tehát a heap még érintetlen. Ha tehát egy programmal mondjuk három 64 Kbájtos területet foglalunk le, akkor biztos, hogy ezek lineáris, összefüggő területen helyezkednek el. Az ilyen összefüggő területeket pedig egy rövid gépi kódú rutinnal könnyen tudjuk kezelni.

Az a módszer, amelyet most ismertetünk, nem éppen elegáns megoldás. Kis túlzással azt is mondhatnánk, hogy a legelemibb Pascal konvencióknak és szabályoknak a lábbal tiprása, olyan durva megoldás, amelytől egy szolid Pascal programozó a kalapját a földhöz csapja. ;–)) Ilyet nem szabad csinálni! De lehet!

```
program LongRecord; { LONGREC.PAS }

uses crt,dsp;

 var
     f : file;
   i,j : word;
   k,l : longint;
     m : memptr;
     p : array[ 0..255] of memptr;

{ Egyetlen minta eltárolása a memóriában }
procedure StoreSample(sample:byte); assembler;
asm
  les   di,m              { A memóriamutató az ES:DI-be kerül      }
  mov   al,sample
  mov   ES:[ di] ,al      { Tároljuk a mintát                      }
  mov   ax,ES             { AX = ES: a szegmensrész az AX-be       }
  inc   di                { Most már az AX:DI-ben van a mutató,    }
                          { és a következő bájtra mutat            }
  jnz   @exit             { Ugrás, ha az offszet nem csordult túl  }
  add   ax,$1000          { Egyébként növeljük a szegmensrészt is  }
@exit:
  mov   m.word,di         { Tároljuk a mutató új értékét           }
  mov   m.word+2,ax
end;
```

StoreSample: Egy hangminta tárolása a memóriában. Az eljárás az *m* pointert használja a memória címzésére, és feltételezi, hogy a memóriát lineárisan foglaltuk le. A minta tárolása után növeljük a mutató offszet részét. Ha az indexregiszter (DI) túlcsordul, akkor a szegmensrészt meg kell növelni 4096-al, ez a fizikai címzés szempontjából 65536 bájtot jelent. A növelés után a mutatót tároljuk az *m*-ben.

```
BEGIN
 clrscr;
 { A DSP lekérdezése }
 if not DSPreset then
  begin
   writeln('Nem lehet inicializálni a DSP egységet!');
   halt;
  end;
 { A felhasználható memória hosszának megállapítása }
 i:=maxavail div 65536;
 l:=i*65536;
 writeln('A maximálisan felhasználható memória: ',l,' bájt');
 { A heap lefoglalása }
 for j:=0 to i-1 do getmem(p[ j] ,65535);
```

A program legelőször inicializálja a DSP-t az ismert módon. Ez után a *maxavail* függvény segítségével megállapítjuk a lefoglalható terület nagyságát, és le is foglaljuk a heapet. A *getmem*-nél láthatjuk, hogy csak 65535 bájtot foglalunk le egy terület számára. Itt azt használjuk ki, hogy a memóriafoglalás mindig megpróbál 0000 offszet részű pointereket visszaadni, ezért a fennmaradó 1 bájt két lefoglalt blokk között szabad marad. Ennek egyébként semmiféle nyoma nincs a programban, és a heapkezelésében sem mutatkozik meg.

```
write('Felvétel indul az Enter lenyomásakor ... ');
readln;
textattr:=$cf;
write('  Felvétel!  ');
textattr:=$07;

m:=p[ 0] ; { A célbájt mutatójának beállítása }

asm cli end;

for k:=1 to l do { !! Felvétel !! }
 begin
  DSPwrite(DirectADC);
  StoreSample(DSPread);
 end;

asm sti end;
```

A felvétel az ismert egyszerű módon történik, de a ciklus itt persze jóval tovább fut, mint az előző programokban. A mintákat az előbbi *StoreSample* eljárással tároljuk.

```
writeln('Felvétel kész!');
writeln('Az adatok lemezre mentése ...');

assign(f,'proba2.sam');
rewrite(f,1);
for j:=0 to i-1 do
 begin
  blockwrite(f,p[ j] ,32768);
  blockwrite(f,p[ j] ^[ 32768] ,32768);
  freemem(p[ j] ,65535);
 end;
close(f);

write('[ENTER]');
readln;
END. of program
```

A felvétel után a program *proba2.sam* néven menti a felvett hangot. Ez normál esetben — ha nem a Pascal editorból, hanem parancssorból futtatjuk a programot — kb. 600 Kbájt hosszú szokott lenni.

Még egyszer szeretnénk felhívni az Olvasó figyelmét arra, hogy a memória ilyesfajta elérése ellenjavalt, ha nincs szükség rá, ne alkalmazzuk ezt a módszert!

4.3.6. Visszajátszás

A létrehozott hosszabb felvétel természetesen semmit sem ér, ha nem tudjuk meghallgatni. Írjuk meg a visszajátszó programot is! A feladat világos: hasonló módon, mint a felvételnél, itt is kell írnunk egy rövid Assembler függvényt, amely a következő mintát előveszi a memóriából. A helyfoglalás és az adatok letöltése ugyanazon az elven történik, mint a felvételnél.

```
program LongPlay; { LONGPLAY.PAS }

uses crt,dsp;

 var
     f : file;
   i,j : word;
   k,l : longint;
     m : memptr;
     p : array[ 0..255] of memptr;

function NextData:byte; assembler;
asm
  les   di,m              { A memóriamutató az ES:DI-be kerül  }
  mov   al,ES:[ di]       { Hívjuk a következő mintát (AL)     }
  mov   bx,ES             { A BX-be tesszük a szegmensrészt    }
  inc   di                { Növeljük az offszet részt          }
  jnz   @exit
  add   bx,$1000          { Növeljük a szegmensrészt, ha kell  }
@exit:
  mov   m.word,di         { Tároljuk a mutatót az m-ben        }
  mov   m.word+2,bx
end;
```

NextData: A következő minta kiolvasása a memóriából. Hasonlóan a felvételnél megismert *StoreSample* eljáráshoz, ez a függvény is lineárisan éri el a memóriát, amelynek következő bájtját az *m* pointer mutatja. A függvény által visszaadott bájt kerül a DSP kimenetére.

```
BEGIN
 clrscr;
 { Inicializáljuk a DSP-t, kilépés, ha hiba történt }
 if not DSPreset then
  begin
   writeln('Nem lehet inicializálni a DSP egységet!');
   halt;
  end;

 assign(f,'proba2.sam');
 reset(f,1);
 l:=filesize(f);          { Megállapítjuk a fájl hosszát      }
 i:=l div 65536;
 if maxavail<l then       { Kilépés, ha nincs elég memória    }
  begin
   writeln('Nincs elég memória a hangfájl letöltéséhez!');
   close(f);
   halt;
  end;

 writeln('Adatbetöltés a lemezről ...');
 { Lefoglaljuk a memóriát, és betöltjük az adatokat }
 for j:=0 to i-1 do
  begin
   getmem(p[j],65535);
   blockread(f,p[j]^,32768);
   blockread(f,p[j]^[32768],32768);
  end;
 close(f);

 write('Lejátszás indul az Enter lenyomásakor ... ');
 readln;
 textattr:=$cf;
 write('  Lejátszás! ');
 textattr:=$07;

 DSPspeakerOn;      { Engedélyezzük a DSP kimenetét          }
 m:=p[0];           { A mutató beállítása az adatok elejére  }

 asm cli end;       { Letiltjuk a megszakításokat }

 for k:=1 to l do         { Lejátszás }
  begin
   DSPwrite(DirectDAC);
   DSPwrite(NextData);
   pause(80);             { A mintavételek közötti idő }
  end;

 asm sti end;       { A megszakítások engedélyezése }
```

```
 DSPspeakerOff;    { Letiltjuk a DSP kimenetét          }
 writeln;
 writeln('Kész! [Enter]');
 readln;
 { Felszabadítjuk a heapet, kilépés }
 for j:=0 to i-1 do freemem(p[j],65535);
END. of program
```

A lejátszás ebben az esetben is gyorsabban történik, mint a felvétel, ezért a *Pause* utasítással itt nagyobb időközöket szabunk meg. (A felvételnél egyáltalán nem is volt késleltetés két mintavétel között.) A felvétel és a lejátszás közötti időkülönbségek kiszámítása egyszerű feladat, úgyhogy ezt az Olvasóra bízzuk.

Sajnos ezek a megoldások *protected* módban nem működnek, mert olyankor a szegmensregiszterek nem fizikai címösszetevőt, hanem szelektorokat tartalmaznak a megfelelő deszkriptortáblákhoz. Ezért *protected* módban más megoldást kell keresni a hosszabb felvételekhez és lejátszásokhoz.

4.4. DMA

A Sound Blaster DMA csatornán keresztül is tud hangokat felvenni és lejátszani. A DMA vezérlést a PC-kben az Intel 8237A típusjelű áramkör valósítja meg. Az XT-ben csak egyetlen DMA vezérlő található, amely négy független DMA csatorna kezelésére képes. Ezek a csatornák egyszerre legfeljebb 64 Kbájt adatot tudnak átvinni, és az ún. lapregiszterek segítségével — amelyek 4-bitesek — 1 Mbájt területen tudnak dolgozni. Sajnos az XT esetében elég komoly megkötések vannak. A rendszer csupán az 1-es csatornát hagyja szabadon, így azt kell használnia minden kártyának, amely DMA átvitelt akar megvalósítani. Az AT gépekben a programozók már fejlettebb DMA rendszert használhatnak. Ezekben két 8237A vezérlő található kaszkádosítva, így hét független DMA csatorna kezelhető (egy csatorna a kaszkádosítást látja el). Az első négy csatorna az XT-vel kompatíbilis 8-bites átvitelre képes, míg a második négy 16 bites átvitelt valósít meg. Ilyenkor egyszerre 128 Kbájt adat kezelhető. Az AT-ben a lapregiszterek 8-bitesek, így 24 bites fizikai címeket lehet előállítani, azaz az egyes csatornák 16 Mbájt területet tudnak megcímezni.

A Sound Blaster 8-bites átvitelre képes, ezért a leggyakoribb az 1-es DMA csatorna felhasználása. (A Sound Blaster Pro és a Sound Blaster 16 esetében ez változtatható, a 2.0-s verzióig általában az 1-es csatornát használják.) A DMA vezérelt hangfelvétel és lejátszás előkészítése bonyolultabb, mint a direkt módé, hiszen ebben az esetben a DMA vezérlőt és a hangkártyát is programozni kell. De hogyan is működik pontosan ez az átvitel? A vezérlő és a hangkártya megfelelő beállítása után a hangkártya a mintavételi frekvencia által megszabott időközökben DMA átvitelt kér, és elvégzi az adott feladatot

(felvétel/lejátszás). Ezt a processzor nem veszi észre, hiszen a DMA (*Direct Memory Access*, azaz közvetlen memória-hozzáférés) átvitelnek éppen az a lényege, hogy az adatforgalom a processzor megkerülésével zajlik a memória és az I/O egység között. Amikor a DMA a megadott összes bájtot átvitte, akkor a Sound Blaster egy megszakítást generál, így tudatja a processzorral, hogy az átvitel befejeződött és kész újabb átvitelre. Ez azt jelenti, hogy a kártya és a DMA vezérlő programozása mellett a megszakítás-vezérlőt is be kell állítani. Fontos tehát a vezérlők regisztereit legalább azon a szinten megismerni, amennyi a hangkártyák programozásához szükséges. Nézzük meg ezért a DMA és az IRQ vezérlő alapvető regisztereit!

4.4.1. A DMA vezérlő programozása

Az első DMA vezérlőt az I/O címterület legelején lehet elérni a $00–$0F címeken. A második vezérlő elérése valamivel összetettebb feladat, mert — néhány hardver szempont miatt — ennek a regiszterei páros I/O címeken találhatók meg a $C0–$DE tartományban.

8237A vezérlőregiszterek:

1. vezérlő (AT/XT)	2. vezérlő (AT)	Regiszterfunkció
$00	$C0	0. csatorna cím (*offset*)
$01	$C2	0. csatorna számláló (*size*)
$02	$C4	1. csatorna cím (*offset*)
$03	$C6	1. csatorna számláló (*size*)
$04	$C8	2. csatorna cím (*offset*)
$05	$CA	2. csatorna számláló (*size*)
$06	$CC	3. csatorna cím (*offset*)
$07	$CE	3. csatorna számláló (*size*)
$08	$D0	parancsregiszter (*command*)
$08	$D0	státusregiszter (*status*)
$09	$D2	kérésregiszter (*request*)
$0A	$D4	bitenkénti maszkolás (*mask set/res*)
$0B	$D6	módregiszter (*mode*)
$0C	$D8	a bájtmutató törlése (*clr byte ptr*)
$0D	$DA	inicializálás (*reset controller*)
$0D	$DA	átmeneti tároló (*latch*)
$0E	$DC	a maszkregiszter törlése (*mask reset*)
$0F	$DE	maszkregiszter (*masks*)

A cím- és a számlálóregiszterek 16 bitesek, és az adott port kétszeri írásával (először az alsó, majd a felső bájt kerül sorra) lehet beállítani őket. A további regiszterek egyes bitjei határozzák meg a DMA átvitel módját és irányát. A következőkben csak az egyes vezérlő programozásáról lesz szó, de egészen a Sound Blaster Pro kártyáig ez elég is. A 16 bites átvitelnél a második vezérlő használata is szükséges.

DMA lapregiszterek:

Csatorna	Lapregiszter I/O cím
0 XT/AT	$87
1 XT/AT	$83
2 XT/AT	$81
3 XT/AT	$82
4 AT	$8F (kaszkádosítás)
5 AT	$8B
6 AT	$89
7 AT	$8A

A lapregiszterek adják a fizikai cím felső bitjeit. Az XT esetében csak az első négy regiszter létezik, és csak az alsó 4 bit érvényes. A XT lapregisztereket csak írni lehet. Az AT esetében a lapregiszterek 8-bitesek és olvashatók is. Ezek a regiszterek egyébként nem tartoznak fizikailag a 8237A áramkörhöz, hanem az alaplapon más áramkörökkel alakítják ki őket.

Regiszter $08 : Parancsregiszter, csak írható

7.bit 0.bit

DACK sen	DRQ sen	WRT CTRL	PRI CTRL	TMR CTRL	E/D CTRL	CH0 ADR	MEM MEM

DACK sense: A DACK jelek aktív szintje
0 – alacsony (L)
1 – magas (H)
DRQ sense: A DRQ jelek aktív szintje
0 – alacsony (L)
1 – magas (H)
Write Control: Az írásjelek időzítése
0 – normál
1 – bővített

Priority Control: A prioritási mód megadása
0 – fix
1 – körforgó
Timer Control: Időzítési mód
0 – normál
1 – sűrített
Enable/Disable Control: A vezérlő engedélyezése/tiltása
Channel 0 Address mode: A 0. csatorna címkezelési módja
0 – címmódosító
1 – nem címmódosító
Memory to Memory: Memória↔memória átvitel


Regiszter $08: Sátusregiszter, csak olvasható

7.bit	0.bit
DMA request states	Terminal count states

DMA request states: Az egyes bitek a DMA-kérést mutatják. Akkor 1 az adott csatornához tartozó bit, ha már érkezett átvitelkérés. A 4. bit a 0. csatornához tartozik, az 5. bit az 1. csatornához és így tovább.
Terminal count states: A számlálók állapotának jelzése. Azon bitek lesznek 1 állapotúak, amelyeknél a számláló már elérte a nullát. A 0. bit a 0. csatornához tartozik, az 1. bit az 1. csatornához és így tovább.

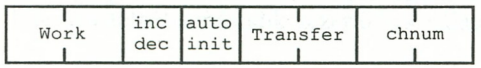

Regiszter $09: Kérésregiszter, csak írható

7.bit		0.bit
unused	new bit	chnum

new bit: A bit új értéke
chnum: A csatorna száma (0..3)


Regiszter $0A: A maszkregiszter bitenkénti írása, csak írható. A bitek kiosztása megegyezik a $09 regiszternél ismertetettel.

 **Regiszter $0B:** Módregiszter, csak írható



Work: üzemmód. A négy lehetséges üzemmód:
- lekérdezés (00)
- egyes átvitel (01)
- blokk átvitel (10)
- kaszkád mód (11)

Inc/Dec: A címmódosítás iránya
- növelés (0)
- csökkentés (1)

Auto Init: Az automatikus inicializálás engedélyezése
- egyciklusú (*single cycle*) típusú átvitel (0)
- automatikus inicializáló (*auto-init*) típusú átvitel (1)

Transfer: Az átvitel módja. A három lehetséges átviteli mód:
- ellenőrzés (00)
- I/O-ról memóriába (01)
- memóriából I/O-ra (10)
- érvénytelen kombináció (11)

 **Regiszter $0C:** A bájtmutató törlése, csak írható. Bármely 16 bites regiszter írása előtt ebbe a regiszterbe $00-t kell írni.

 **Regiszter $0D:** Inicializálás, csak írható.

 **Regiszter $0D:** Átmeneti tároló regiszter. A memória↔memória másolás utolsó átvitt bájtját tárolja.

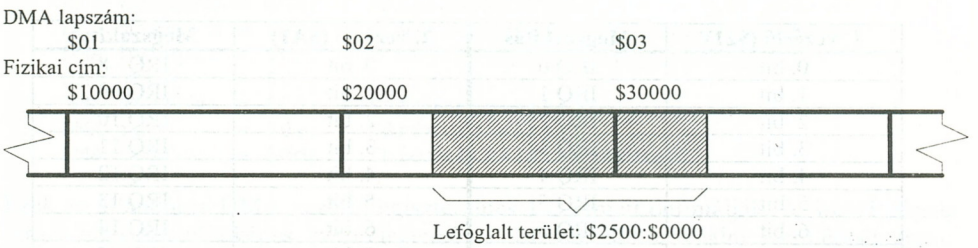
 **Regiszter $0E:** A maszkregiszter törlése, csak írható. A csatornákat engedélyezi.

 **Regiszter $0F:** Maszkregiszter, csak írható. Az alsó négy bit az egyes csatornákat maszkolja.

Az itt ismertetett I/O címek az első DMA vezérlőre vonatkoznak, az AT gépeknél a második vezérlő címeit az előbbi táblázat mutatja. A példaprogramban részletesen bemutatjuk majd, hogy mely regisztert hogyan kell programozni.

4.4.2. A DMA átvitel korlátai

A DMA átvitelnek sajnos van néhány korlátja. Az első, hogy a legfeljebb átvihető adatmennyiség 64 Kbájt, illetve a 16 bites átvitelnél 128 Kbájt. Ha 16 bites átvitellel dolgozunk, akkor arra is figyelni kell, hogy a fizikai cím 0. bitje mindig nulla. Ezért csak páros memóriacím érhető el ebben az esetben. Ezek a problémák gondot okoznak a programozás során, mégis sokkal kisebb a jelentőségük, mint a laphatárok kötöttségének. A másik probléma, hogy a lapregiszterek 64 Kbájtonként tudják elérni a memóriát, és ezek között a lapok között nem lehet átlapolás. A processzor viszont bármely memóriaterületet kényelmesen elérhet, hiszen az egyes szegmensek 16 bájtos határokkal átlapolhatók. A Pascal memóriaallokáló függvényei mindig normalizálják a visszaadott pointert, amikor csak lehet. Ez azt jelenti, hogy a visszaadott mutató offszetrésze a legtöbb esetben $0000, esetleg kivételesen $0008. Amikor lefoglalunk egy 64 Kbájt nagyságú memóriaterületet, akkor az nagy valószínűséggel nem fog pontosan laphatárra esni. Az alábbi ábra alapján a probléma világossá válik:



33. ábra. A DMA laphatárának átlépése

A lefoglalt terület általában két egymást követő DMA lapra esik. Előfordulhat, hogy a lefoglalt terület elfér egyetlen lapon, de ehhez az szükséges, hogy kis területekkel dolgozzunk. Jól látható tehát, hogy nagyobb hangminták esetében mindenképpen lapot kell váltani. A problémának többféle megoldása is létezik. A legegyszerűbb, de szerintünk a legcsúnyább megoldás az, hogy nem egy, hanem két 64 Kbájtos területet foglalunk le egymás mögött, így biztos, hogy a lefoglalt terület magában foglal egy teljes lapot. Ez azonban pazarlás. A másik megoldás pedig az, hogy a laphatár elérésekor újraprogramozzuk a DMA vezérlőt, hogy most már a következő lapon a maradék adatokat vigye át. Ez elegánsabb megoldás, de ilyenkor két DMA átvitel történik, tehát a Sound Blaster

megszakítását erre fel kell készíteni. A később ismertetésre kerülő *SB_DMA* unitban ezt a megoldást használjuk.

4.4.3. Az IRQ vezérlő programozása

A megszakításokat a PC-kben az Intel 8259A típusjelű megszakításvezérlő áramkör kezeli. Az XT-ben egy, az AT-ben két ilyen vezérlő található. Ezek a vezérlők egy-egy INT utasítást rendelnek az egyes hardvermegszakításokhoz, a következők szerint:

IRQ0 ... IRQ7: INT $08 ... INT $0F
IRQ8 .. IRQ15: INT $70 ... INT $77

Az áramköröket többféle üzemmódban lehet programozni, de ezeket most nem ismertetjük, mert a hangkártyák szempontjából nincs jelentőségük. A következő lényeges információk mindenkor elegendőek:

A vezérlőket négy I/O címen lehet elérni. Az első vezérlő a $20 és $21 címeket, míg a második az $A0 és $A1 címeket használja. A $20 és $A0 címre kiírt $20 érték jelzi a vezérlőnek a megszakítás elfogadását, pontosabban a kérés törlését. A $21 és $A1 portok egyes bitjei engedélyezik és tiltják a hardvermegszakításokat:

1. vezérlő ($21)	Megszakítás	2. vezérlő ($A1)	Megszakítás
0. bit	IRQ 0	0. bit	IRQ 8
1. bit	IRQ 1	1. bit	IRQ 9
2. bit	IRQ 2	2. bit	IRQ 10
3. bit	IRQ 3	3. bit	IRQ 11
4. bit	IRQ 4	4. bit	IRQ 12
5. bit	IRQ 5	5. bit	IRQ 13
6. bit	IRQ 6	6. bit	IRQ 14
7. bit	IRQ 7	7. bit	IRQ 15

Egy hardvermegszakítás átvételéhez tehát át kell írni a megfelelő megszakítási vektort a memóriában, és engedélyezni kell a hozzá tartozó csatornát. A Sound Blaster esetében a megszakítási vonal száma állítható, ezért olyan programot kell készíteni, amely képes bármelyik csatorna korrekt beállítására.

4.4.4. Az SB_DMA unit

A Sound Blaster DMA alatti használatához készült a következő unit. Azokat az alapvető eljárásokat gyűjtöttük össze benne, amelyek a DMA csatorna egyszerű programozását teszik lehetővé. A feladatokat több részfeladatra lehet bontani:

– a DMA vezérlő beállítása az átvitelhez;
– az IRQ vezérlő beállítása a megfelelő megszakítási vonal használatához;
– a Sound Blaster programozása a DMA átvitelhez.

Az eljárások mellett definiáltunk néhány olyan változót és konstanst, amelyek a programozás során hasznosak lehetnek. Az eddigi programokhoz képest ez a unit talán terjedelmesebb lesz, de a felvétel és a lejátszás szempontjából sokkal hatékonyabb eszközöket és lehetőségeket nyújt, mint a DSP direkt módú elérése.

Állandók, változók

```
{$g+}
unit sb_dma;        { SB_DMA.PAS }

interface uses dsp;              { Használjuk a DSP unitot }

const
    Rec  = $00;                  { A felvétel jelzése     }
    Play = $01;                  { A lejátszás jelzése    }

    { DMA vezérlőregiszterek }
    DMA_command = $08; { Parancsregiszter                          }
    DMA_mask    = $0a; { A maszkregiszter bitenkénti írása         }
    DMA_byteptr = $0c; { A bájtmutató törlése                      }
    DMA_mode    = $0b; { Módregiszter                              }
    DMA_status  = $08; { Státusregiszter                           }
    DMA_req     = $09; { Kérésregiszter                            }
    DMA_init    = $0d; { Inicializáló regiszter                    }
```

Ezek az állandók a DMA vezérlőregisztereinek I/O címeit definiálják. A Sound Blaster csak 8-bites átvitelre képes, ezért elegendő az első vezérlőt használni. Nem éppen aranyszabály, de szerintünk követésre méltó konvenció, hogy a fix I/O címeket mindig célszerű konstansokkal definiálni, és a programon belül mindenhol ezeket a szimbólumokat használni, ahol az adott I/O címet el kell érni. Ha esetleg a későbbiekben ezek a címek változnának, akkor elegendő az állandóknak új értéket adni, s nem kell fáradságos munkával a teljes programban javításokat végezni.

```
    { Bitmaszkértékek a megszakítási csatornához }
    BitMasks : array[ 0..7] of byte=
                        ($01,$02,$04,$08,$10,$20,$40,$80);

    { Az eredeti megszakítási vektor tárolása }
    OldIRQofs : word=00;
    OldIRQseg : word=00;
```

A *BitMasks* tömb a megszakítási vonalakhoz tartozó bitek állítását segíti, míg az *OldIRQofs* és az *OldIRQseg* az eredeti megszakítási vektort tárolják.

```
{ A DMAchannel értékétől függő DMA I/O címek táblázatai }
PageTAB : array[ 0..3] of byte=($87,$83,$81,$82);
OffsTAB : array[ 0..3] of byte=($00,$02,$04,$06);
SizeTAB : array[ 0..3] of byte=($01,$03,$05,$07);
```

A *DMAchannel* értéke változó lehet, ezért minden egyes hangkártya esetleg más-más DMA csatornát használhat. A különböző csatornákhoz pedig különböző I/O címek tartoznak. Ezért ezek a táblázatok megadják a négy lehetséges csatornához tartozó I/O címeket. A *Refresh* eljárás ezek alapján beállítja az aktuális (*Current*) értékeket.

```
var
    DMAdir     : byte; { Rec/Play (00/01)                    }
    DMApage    : byte; { A DMA lap száma                     }
    DMAoffset  : word; { A DMA kezdőcíme                     }
    DMAlength  : word; { Az átvitel hossza                   }
    DMAmax     : word; { Maximális átviteli hossz az         }
                       { aktuális lapon                      }

    DMAnextpg  : boolean; { Szükség van-e következő átvitelre? }
```

DMAdir: Az átvitel irányának jelzése a megszakítási rutin számára. A megszakítást úgy írtuk meg, hogy az mind lejátszásra, mind felvételre alkalmas legyen. Amikor egy DMA átvitel véget ér, de a következő lapon még van adat, akkor a DMAdir mondja meg, hogy lejátszás vagy felvétel következik-e.

DMApage,DMAoffset,DMAlength: Az átvitel memóriacíme (*DMApage, DMAoffset*) és az adatok hossza (*DMAlength*).

DMAmax,DMAnextpg: Az aktuális lapon legfeljebb használható bájtok száma. Ha a *DMAlength* — vagyis az adathalmaz hossza — nagyobb a *DMAmax*-nál, akkor laphatárátlépés történt, ilyenkor két DMA átvitelre van szükség. A második átvitel szükségességét a *DMAnextpg* jelzi.

```
{ A DMAchannel értékétől függő adatok                    }
CurrentPage : word; { Lapregiszter I/O cím               }
CurrentOffs : word; { Címregiszter I/O cím               }
CurrentSize : word; { Számláló  I/O cím                  }
PlayMode    : byte; { A lejátszás vezérlőbájtja          }
RecMode     : byte; { A felvétel vezérlőbájtja           }
DisDMA      : byte; { A DMA csatorna tiltása             }
EnDMA       : byte; { A DMA csatorna engedélyezése       }
```

Az aktuális DMA csatornához tartozó I/O címek és vezérlőbájtok. A unit a *DMAchannel* változó értékétől függően ezeket az értékeket automatikusan beállítja a *Refresh* eljárással. Ha a futás alatt mégis újra kellene állítani őket, akkor a *Refresh* bármikor meghívható. A *DMAchannel* változó a *DSP.PAS* unitban található.

Eljárások

```
procedure StartPlay(sr:byte; song:pointer; size:word);
procedure StartRec (sr:byte; song:pointer; size:word);
procedure Refresh;
```

Külső programokból — egyelőre — csak három eljárás hívható: a *StartPlay* elindít egy lejátszást, a *StartRec* elindít egy felvételt, míg a *Refresh* az alapvető beállításokat végzi el. Nincs semmi akadálya, hogy egyéb eljárásokat is ismertté tegyünk a külső programok számára, de arra vigyázni kell, hogy nem minden eljárást lehet csak úgy „hipp-hopp!" minden következmény nélkül meghívni.

```
implementation

procedure PageMaker;                          assembler;
asm
  mov   cl,ah           { CL=a szegmensrész felső bitjei     }
  shr   cl,4            { Csak a legfelső 4 bit számít       }
  shl   ax,4            { AX=16*szegmensrész                 }
  add   dx,ax           { DX=szegmens+offszet                }
  mov   ah,cl           { A lapszám betöltése az AH-ba       }
  adc   ah,00           { Növeljük, ha volt átvitel          }
  mov   cx,dx           { Az összeg negáltja a legfeljebb    }
  neg   cx              { felhasználható bájtok száma        }
end;
```

PageMaker: Egy pointer átalakítása *page* és *offset* összetevőre. Az eljárás az AX:DX regiszterpárban megadott pointert átalakítja a DMA vezérlő számára használható lap és lapeltolás formára. Visszatéréskor az AH tartalmazza a lapszámot, a DX a lapeltolást, míg a CX a lapból maximálisan felhasználható bájtok számát adja. Az eljárás nem hívható közvetlenül!

```
procedure ResetIRQ;                                    forward;
procedure PlayDMA(page:byte; offs,size:word);          forward;
procedure RecDMA (page:byte; offs,size:word);          forward;

procedure SB_IRQ;                                      assembler;
asm
  push  DS
  push  ES
  pusha                          { Tároljuk a regisztereket    }
  mov   ax,Seg @data
```

```
  mov   DS,ax                  { A DS az adatszegmensre mutat}
  mov   dx,BaseAddr
  add   dx,$0e                 { DX=$22E, DSP Data Available }
  in    al,dx                  { A megszakítás nyugtázása    }
  test  DMAnextpg,1            { Kell új DMA átvitel?        }
  jnz   @nextDMA               { Ugrás, ha kell              }
@endDMA:
  push  DSP_DMA_stop           { Leállítjuk a DMA átvitelt   }
  call  DSPwrite
  call  DSPspeakerOff          { Letiltjuk a DSP kimenetét   }
  call  ResetIRQ               { Visszaírjuk az eredeti      }
                               { megszakítási vektort        }
  jmp   @endIRQ                { Kilépünk a megszakításból   }
@nextDMA:
  mov   DMAnextpg,0            { Már nem lesz több átvitel   }
  mov   al,DMApage
  inc   al                     { Növeljük a DMA lapszámot    }
  push  ax
  push  $0000                  { A lapeltolás most $0000     }
  mov   ax,DMAlength
  sub   ax,DMAmax              { Kiszámítjuk a maradék hosszt}
  push  ax
  cmp   DMAdir,Rec
  jz    @x2                    { Felvétel esetén ugrás       }
  call  PlayDMA                { Egyébként lejátszás         }
  jmp   @endIRQ
@x2:
  call  RecDMA                 { Felvétel                    }
@endIRQ:
  mov   al,$20                 { A "megszakítás vége" jelzés }
  out   $20,al                 { az IRQ vezérlőnek           }
  popa
  pop   ES
  pop   DS                     { Regiszterek vissza, vége    }
  iret
end;
```

SB_IRQ: Megszakítás. Az *SB_IRQ* a Sound Blaster megszakítását kezeli. A program akkor hívja ezt a rutint, ha egy DMA átvitel befejeződött. Először a hangkártya *Data Available* portját olvassuk be, ebből tudja meg a kártya, hogy a processzor elfogadta a megszakítását. Utána eldöntjük a *DMAnextpg* értéke alapján, hogy kell-e újabb átvitelt kezdeményezni. Ha nem szükséges több átvitel, akkor a rutin leállítja a hangkártyát, kikapcsolja DSP kimenetét, és meghívja a *ResetIRQ* eljárást. A *ResetIRQ* visszaállítja a megszakítás eredeti állapotát. Ha a *DMAnextpg* szerint még kell átvitel, akkor újra be kell állítani a DMA vezérlőt. Legelőször is kitöröljük a *DMAnextpg* változót, hiszen ez után már nem szabad újabb átvitelt kezdeményezni. A lapszám eggyel nő, míg a lapeltolás $0000 lesz, hiszen az adatok most a lap elején kezdődnek. A hossz ilyenkor a még hátralévő bájtok száma, azaz a *DMAlength–DMAmax* különbség. Mindezen adatok kiszámítása után — a *DMAdir* értékétől függően — a program vagy a *PlayDMA*, vagy a

RecDMA eljárást hívja meg, amelyek elvégzik a beállításokat, és elindítják az újabb DMA ciklust. A megszakítás végén küldünk egy nyugtázást az IRQ vezérlőnek, és visszatérünk az eredeti programhoz. Az eljárás nem hívható közvetlenül!

```
procedure SetIRQ;                                    assembler;
asm
  cli
  mov   al,IRQnumber          { Az aktuális megszakítási vonal }
  or    al,8                  { alapján az INT száma *4        }
  cbw
  shl   ax,2                  { a vektor elhelyezése miatt     }
  mov   bx,ax
  xor   ax,ax
  mov   ES,ax                 { ES:BX a vektor helyére mutat   }
  mov   dx,ES:[bx]
  mov   OldIRQofs,dx          { Tároljuk az offszet részt      }
  mov   dx,ES:[bx+2]
  mov   OldIRQseg,dx          { Tároljuk a szegmensrészt       }
  lea   dx,SB_IRQ
  mov   ES:[bx],dx            { Beírjuk az új offszetet        }
  mov   ES:[bx+2],CS          { Beírjuk az új szegmenst        }
  mov   al,IRQnumber
  lea   bx,Bitmasks
  xlat                        { Meghatározzuk a bitmaszkot     }
  not   al                    { A megfelelő bit 0, a többi 1   }
  mov   ah,al
  in    al,$21                { AL=az eredeti maszk            }
  and   al,ah                 { Engedélyezzük a csatornát      }
  out   $21,al                { Kiírjuk az új maszkot          }
  sti
end;
```

SetIRQ: A megszakítás beállítása a Sound Blaster számára. Az *IRQnumber* alapján — amely a *DSP.PAS* unitban található — meghatározzuk a megszakítási vektor címét. Az eredeti vektort tároljuk az *OldIRQofs* és *OldIRQseg* változókban. Az *SB_IRQ* címét beírjuk az eredeti vektor helyére, és engedélyezzük a megfelelő megszakítási vonalat az IRQ vezérlő beállításával. Mindez idő alatt a megszakításokat természetesen tiltani kell!

```
procedure ResetIRQ;                                  assembler;
asm
  cli
  mov   al,IRQnumber        { Az aktuális megszakítási vonal *4 }
  or    al,8
  cbw
  shl   ax,2                  { a vektor elhelyezése miatt     }
  mov   bx,ax
  xor   ax,ax
  mov   ES,ax                 { ES:BX a vektor helyére mutat   }
  mov   ax,OldIRQofs
```

```
  mov   ES:[bx],ax
  mov   ax,OldIRQseg
  mov   ES:[bx+2],ax          { Visszaírjuk az eredeti vektort }
  mov   al,IRQnumber
  lea   bx,Bitmasks
  xlat                        { Meghatározzuk a bitmaszkot     }
  mov   ah,al
  in    al,$21                { AL=az aktuális maszk           }
  or    al,ah                 { Letiltjuk a csatornát          }
  out   $21,al                { Kiírjuk az új maszkot          }
  xor   ax,ax
  mov   OldIRQseg,ax          { OldIRQseg=$0000                }
  sti
end;
```

ResetIRQ: Az eredeti megszakítási állapotok visszaállítása. Az eljárás visszaállítja a megfelelő megszakítási vektort, és letiltja az IRQ vezérlő Sound Blasterhez tartozó vonalát. Ezt az eljárást az *SB_IRQ* megszakítási rutin hívja akkor, amikor az átvitel véget ér, és már nincs szükség a megszakításra.

```
procedure SetOffsSizePage;                              assembler;
asm
  push  dx                    { Tároljuk a DX-et               }
  mov   dx,CurrentPage        { Az aktuális lapregiszter címe  }
  out   dx,al                 { A lap beállítása               }
  mov   dx,CurrentSize        { Az aktuális számláló címe      }
  mov   al,bl
  out   dx,al                 { Alsó bájt                      }
  mov   al,bh
  out   dx,al                 { Felső bájt                     }
  mov   dx,CurrentOffs        { Az aktuális eltolás címe       }
  mov   al,cl
  out   dx,al                 { Alsó bájt                      }
  mov   al,ch
  out   dx,al                 { Felső bájt                     }
  pop   dx                    { Visszatöltjük a DX-et          }
end;
```

SetOffsSizePage: Az aktuális DMA csatorna paramétereinek beállítása. Az eljárással a megadott DMA csatornához tartozó lapregisztert, eltolás- és számlálóregisztert állítjuk be az átvitel megkezdésekor. A lapszámot az AL, a lapon belüli címeltolást a CX, míg a hosszúságot a BX tartalmazza. Az eljárás nem hívható közvetlenül!

```
procedure PlayDMA(page:byte; offs,size:word);    assembler;
asm
  dec   size
  mov   al,DisDMA
  out   DMA_mask,al           { Tiltjuk a DMA átvitelt         }
  xor   al,al
```

```
  out   DMA_byteptr,al          { Töröljük a bájtmutatót          }
  mov   al,PlayMode
  out   DMA_mode,al             { Beállítjuk lejátszásra a DMA-t  }
  mov   al,page
  mov   bx,size
  mov   cx,offs
  call  SetOffsSizePage         { Beállítjuk a csatorna           }
                                { alapvető paramétereit           }
  mov   al,EnDMA
  out   DMA_mask,al             { Engedélyezzük a DMA átvitelt    }
  push  NormalDMA_DAC
  call  DSPwrite                { DMAplay-re állítjuk a DSP-t     }
  push  bx
  call  DSPwrite                { A hossz alsó bájtja             }
  mov   bl,bh
  push  bx
  call  DSPwrite                { A hossz felső bájtja            }
                                { A DMA lejátszás elindul         }
end;
```

PlayDMA: A memóriában lévő hangminta lejátszása. Az eljárás elindítja a DMA alatti lejátszást. A *page* paraméterrel a DMA lap számát, az *offs* paraméterrel a lapon belül a kezdőcímet, míg a *size* paraméterrel a lejátszandó terület hosszát kell megadni. Az eljárás hívható ugyan közvetlenül, de nem szabad megfeledkezni az IRQ vezérlő beállításáról. A rutin beállítja a DMA vezérlőt és a DSP-t is ennek megfelelően. A DMA átvitel a DSP programozása után megkezdődik, ezért fontos, hogy először mindig a vezérlőt programozzuk, s csak utána a DSP-t.

```
procedure RecDMA(page:byte; offs,size:word);     assembler;
asm
  dec   size
  mov   al,DisDMA
  out   DMA_mask,al             { Tiltjuk a DMA átvitelt          }
  xor   al,al
  out   DMA_byteptr,al          { Töröljük a bájtmutatót          }
  mov   al,RecMode
  out   DMA_mode,al             { Beállítjuk felvételre a DMA-t   }
  mov   al,page
  mov   bx,size
  mov   cx,offs
  call  SetOffsSizePage         { Beállítjuk a csatorna           }
                                { alapvető paramétereit           }
  mov   al,EnDMA
  out   DMA_mask,al             { Engedélyezzük a DMA átvitelt    }
  push  NormalDMA_ADC
  call  DSPwrite                { DMArecordra állítjuk a DSP-t    }
  push  bx
  call  DSPwrite                { A hossz alsó bájtja             }
  mov   bl,bh
  push  bx
```

```
  call  DSPwrite                { A hossz felső bájtja          }
                                { A DMA felvétel elindul        }
end;
```

RecDMA: Hangminta digitalizálása és felvétele a memóriába. Az eljárás a *PlayDMA* fordítottja. A megadott DMA lapra (*page*) a megadott kezdőcímre (*offs*) a megadott hosszúságú (*size*) adatot felveszi a DSP bemenetéről. Szerkezetileg szinte azonos a *PlayDMA* eljárással, csupán a fordított adatirány miatt mind a DMA vezérlőt, mind a DSP-t más üzemmódban használja.

```
procedure StartPlay(sr:byte; song:pointer; size:word);
begin
 DMAdir:=Play;                  { Az adatmozgás iránya MEM->I/O }
 DMAlength:=size;               { Az adatmennyiség hossza       }
 asm
   les  dx,song                 { ES:DX a memóriára mutat       }
   mov  ax,ES
   call PageMaker               { Átalakítjuk a címet a DMA-nak }
   mov  DMApage,ah              { Lapszám                       }
   mov  DMAoffset,dx            { Lapon belüli eltolás (cím)    }
   mov  DMAmax,cx               { Maximális hely a lapon        }
 end;
 DMAnextpg:=DMAmax<DMAlength;   { Kell újabb DMA átvitel?       }
 if not DMAnextpg then DMAmax:=DMAlength; { A hossz             }
 DSPwrite(SetSampleRate);
 DSPwrite(sr);      { Beállítjuk a mintavételi frekvenciát      }
 DSPspeakerOn;      { Bekapcsoljuk a DSP kimenetét              }
 SetIRQ;            { Beállítjuk a megszakítási szintet         }
 PlayDMA(DMApage,DMAoffset,DMAmax); { Lejátszás indul           }
end;
```

StartPlay: Lejátszás. Az eljárás a *song* pointer által mutatott *size* hosszúságú hangmintát lejátssza *sr* mintavételezési frekvenciával DMA alatt. A rutinon belül minden szükséges beállítás megtörténik. Az *sr* paraméter a mintavételezési frekvenciával arányos bájtkonstans, amely $FF esetén a legnagyobb, míg $00 esetén a legkisebb minta-vételezési frekvenciát adja.

```
procedure StartRec(sr:byte; song:pointer; size:word);
begin
 DMAdir:=Rec;                   { Az adatmozgás iránya I/O->MEM }
 DMAlength:=size;               { Az adatmennyiség hossza       }
 asm
   les  dx,song                 { ES:DX a memóriára mutat       }
   mov  ax,ES
   call PageMaker               { Átalakítjuk a címet a DMA-nak }
   mov  DMApage,ah              { Lapszám                       }
   mov  DMAoffset,dx            { Lapon belüli eltolás (cím)    }
   mov  DMAmax,cx               { Maximális hely a lapon        }
 end;
```

```
 DMAnextpg:=DMAmax<DMAlength; { Kell újabb DMA átvitel?      }
 if not DMAnextpg then DMAmax:=DMAlength; { A hossz          }
 DSPwrite(SetSampleRate);
 DSPwrite(sr);      { Beállítjuk a mintavételi frekvenciát   }
 SetIRQ;            { Beállítjuk a megszakítási szintet      }
 RecDMA(DMApage,DMAoffset,DMAmax); { Felvétel indul          }
end;
```

StartREC: Felvétel. Az eljárás a *song* pointer által mutatott memóriacímre mintavételez egy *size* hosszúságú hangmintát *sr* mintavételi frekvenciával. A felvétel DMA alatt történik. Az *sr* paraméter a mintavételi frekvenciával arányos bájtkonstans, amely $FF esetén a legnagyobb, míg $00 esetén a legkisebb mintavételezési frekvenciát adja.

```
procedure Refresh;
begin
 { Aktuális lapregiszter I/O                                 }
 CurrentPage:=PageTAB[ DMAchannel];
 { Aktuális címregiszter I/O                                 }
 CurrentOffs:=OffsTAB[ DMAchannel];
 { Aktuális számlálóregiszter I/O                            }
 CurrentSize:=SizeTAB[ DMAchannel];
 { Lejátszásvezérlő bájt a DMA számára                       }
 PlayMode:=$48 or DMAchannel;
 { Felvételvezérlő bájt a DMA számára                        }
 RecMode:=$44 or DMAchannel;
 { Vezérlőbájtok a DMA engedélyezése/letiltása számára       }
 DisDMA:=DMAchannel or 4;
 EnDMA:=DMAchannel and 3;
end;
```

Refresh: Újrabeállítás. Az eljárás beállítja a DMA csatorna számától függő I/O címeket és üzemmód bájtokat.

```
BEGIN
 Refresh;   { Alapbeállítások a DSP.PAS adatai szerint       }
END. of unit
```

Amint látható, a unit két eljárását — a *StartPlay* és a *StartRec* nevűt — úgy írtuk meg, hogy kényelmesen használhatók legyenek az egyszerű lejátszásra és felvételre. Ha valaki ennél bonyolultabb feladatot szeretne megoldani — például nagyobb memóriaterületen mintavételezni, vagy közvetlenül a merevlemezre írni az adatokat —, akkor néhány helyen módosítania kell a unit egyes eljárásait.

4.4.5. Felvétel DMA átvitellel

A következő program — az előző *SB_DMA* unit felhasználásával — egy 65520 bájtos felvételt készít DMA átvitellel. A program — amint látható — igen egyszerű, hiszen az összes lényeges beállítást a unit végzi el.

```
program DmaRec; { DMAREC.PAS }

uses crt,dsp,sbdma;

var
    f : file;
    i : word;
    p : memptr;

BEGIN
 clrscr;
 if not DSPreset then
  begin
   writeln('Nem lehet inicializálni a DSP egységet!');
   halt;
  end;

 { Memóriafoglalás a felvétel adatai számára }
 getmem(p,65520);
 write('Az Enter lenyomására a felvétel elindul ... ');
 readln;
 textattr:=$CF;
 writeln('  DMA felvétel!  ');
 textattr:=$07;
 StartRec($C0,p,65520); { Felvétel DMA vezérelt módon }
 { A program itt várakozik a felvétel befejezésére }
 while OldIRQseg<>0 do inc(mem[ $b800:0] );
 writeln('Kész !');
 { Az adatokat lemezre mentjük }
 assign(f,'proba.sam');
 rewrite(f,1);
 blockwrite(f,p^,65520);
 close(f);
 { Felszabadítjuk a heapet, kilépés }
 freemem(p,i);
END. of program
```

Lényeges lehet, hogy a felvétel végét az *OldIRQseg* változó $0000 értéke jelzi. Alapesetben ugyanis ez semmiképpen nem lehet $0000, hiszen a megszakítás szegmenscíme az aktuális kódszegmens értéke. A változót a *ResIRQ* eljárás nullázza az átvitel

befejezésekor. Igazából ez nem éppen a legelegánsabb módszer a felvétel befejezésének jelzésére, de mindenesetre működik!

4.4.6. Lejátszás DMA átvitellel

Az *SB_DMA* unit segítségével a lejátszás is éppen olyan egyszerű, mint a felvétel. A lejátszáshoz nem kell mást tenni, mint a hangminta betöltése után meghívni a *StartPlay* eljárást a megfelelő paraméterekkel.

```
program DmaPlay; { DMAPLAY.PAS }

uses crt,dsp,sb_dma;

var
    f : file;
    i : word;
    p : memptr;

BEGIN
 assign(f,'proba.sam'); { Betöltjük a hangmintát      }
 reset(f,1);
 i:=filesize(f);
 getmem(p,i);
 blockread(f,p^,i);
 close(f);

 if not DSPreset then          { Inicializáljuk a DSP-t  }
  begin
   writeln('Nem lehet inicializálni a DSP egységet!');
   exit;
  end;

 write('Lejátszás indul ... ');
 StartPlay($b0,p,i);           { Elindítjuk a lejátszást }

 writeln('Kész. [Enter]');
 readln;

 freemem(p,i);     { Felszabadítjuk a heapet, kilépés }
END. of program
```

4.4.7. Időállandók kiszámítása

Az előző programokban gyakran használtuk a mintavételezési frekvencia fogalmát, az időzítéseket azonban nem adtuk meg pontosan. Nos, azért nem tartottuk célszerűnek konkrét értékekkel dolgozni, mert ezek az értékek mindig az adott üzemmódtól függenek. A felvétel például kisebb frekvenciával történik, mint a lejátszás, és a nagy sebességű részeknél még egyéb változások is vannak. A legegyszerűbb, a normál DMA lejátszás időállandója az alábbi képlettel számítható:

$$T_c = 256 - \frac{1000000}{sr} = 256 - \frac{1000000}{8000} = 131$$

A T_c az időállandót jelenti, az *sr* pedig az aktuális mintavételi frekvenciát. A példában a 8 kHz-hez tartozó állandó kiszámítását mutattuk be.

A nagy sebességű DMA lejátszáshoz a következő módon lehet meghatározni az időállandót (44,1 kHz-hez):

$$T_c = HIGH\left(65536 - \frac{256000000}{sr}\right) = HIGH\left(65536 - \frac{256000000}{44100}\right) = 0E9h$$

A *HIGH* függvénnyel azt jeleztük, hogy számunkra a kiszámított érték felső bájtja érdekes. Az, hogy a kiszámított értéknél nagyobb érték is létezik, még nem jelenti azt, hogy van nagyobb mintavételi frekvencia! Egy bizonyos határ után hiába változtatjuk az értékeket, hiszen a legnagyobb frekvenciát a hardver határozza meg, nem a beírt érték.

4.4.8. Nagy sebességű átviteli módok

A normál DMA átvitelhez teljesen hasonló módon történik a nagy sebességű átvitel is, csak ilyenkor a mintavételi frekvenciák más intervallumokban mozognak. Ezekre a módokra már nem mutatunk be külön programot, csupán a programozás metodikáját ismertetjük.

Nagy sebességű felvétel (ADC)

– Állítsuk be a megszakításvezérlőt a megfelelő módon (SB_IRQ)!
– A *SetSampleRate ($40)* DSP paranccsal állítsuk be a nagy sebességű mintavételi frekvenciához tartozó időállandót!
– Programozzuk a DMA vezérlőt az átvitelhez!
– Írjuk ki a *Set High Speed Size ($48)* parancsot a DSP-re!

– Írjuk ki a hossz alsó, majd felső bájtját a DSP-re!
– Írjuk ki a *High Speed Record ($99)* parancsot a DSP-re!

Nagy sebességű lejátszás (DAC)

– Kapcsoljuk be a DSP kimenetét a *SpeakerOn* ($D1) paranccsal!
– Állítsuk be a megszakításvezérlőt a megfelelő módon (SB_IRQ)!
– A *SetSampleRate ($40)* DSP paranccsal állítsuk be a nagy sebességű mintavételi frekvenciához tartozó időállandót!
– Programozzuk a DMA vezérlőt az átvitelhez!
– Írjuk ki a *Set High Speed Size ($48)* parancsot a DSP-re!
– Írjuk ki a hossz alsó, majd felső bájtját a DSP-re!
– Írjuk ki a *High Speed Play ($91)* parancsot a DSP-re!

Mindkét átvitel végén a normál módhoz hasonlóan állítsuk vissza a megszakítást, és kapcsoljuk ki a DSP kimenetét — ez csak a lejátszásnál érdekes — a *SpeakerOff $D3* paranccsal.

4.5. Összegzés

Ha az Olvasó figyelmesen elolvasta az eddigi fejezeteket, és megértette a példákat, akkor már elég jól ismeri a DSP-t. Természetesen nem volt szó *mindenről*, amire ezt az okos kis egységet fel lehet használni. Ha az Olvasó jártas a digitális jelfeldolgozásban, akkor jól tudja, hogy a digitális hangmintákat igen kényelmesen lehet kezelni. Algoritmikus úton lehet őket szűrni, keverni, a hangerőt és a frekvenciát változtatni, vagy éppen analizálni, ha arra van szükség. Mindez persze egyáltalán nem könnyű feladat, bármelyik alaposan megdolgoztatja a programozót. Ezért azt javasoljuk, hogy mindenki próbálkozzék ilyen jellegű feladatokkal, és kísérletezzen a DSP-vel! Nagyon sokat ki lehet hozni a hangkártyából egy jól átgondolt programmal! És ami a leglényegesebb: a türelem! ; –))

5. A Sound Blaster Pro hangkártya

A Sound Blaster Pro hangkártya a Sound Blaster továbbfejlesztett sztereó változata. A programok átvihetősége érdekében a kártya kompatíbilis maradt az ADLIB és a Sound Blaster kártyákkal, ugyanakkor sok újdonságot is hozott az elődeihez képest. Vegyük sorra az általános jellemzőket:

- teljes kompatibilitás az előző kártyákkal;
- mono és sztereó FM csatornák (mono az alapbeállítás);
- mono és sztereó digitális csatornák (mono az alapbeállítás);
- állítható mintavételi frekvencia 4–44.1 kHz-ig;
- mono és sztereó felvétel (alapesetben mono felvételeket lehet készíteni);
- mono és sztereó lejátszás (alapesetben mono a lejátszás);
- beépített sztereó erősítő (4 watt/csatorna);
- beépített CD-ROM vezérlő (Creative/Panasonic CD-ROM meghajtóhoz);
- sztereó csatlakozó az audio-CD-k erősítéséhez;
- sztereó vonalbemenet más audioeszközökhöz (Line In);
- mikrofonbemenet (Mic);
- állítható mintavételi forráseszköz (Line In, CD, Mic);
- állítható kimeneti forráseszköz (MIDI, CD, Line In);
- MIDI-csatlakoztatási lehetőség;
- beépített programozható keverőáramkör a jelforrások keveréséhez;
- joystick-csatlakoztatási lehetőség;
- PC–Speaker-csatlakoztatási lehetőség (???);
- DMA hardver dekompresszió (lejátszásnál).

Mindezen jellemzőket részletesen ismerteti a kártyához tartozó felhasználói kézikönyv (*User's Guide*). Ezen tulajdonságai mellett a kártyának természetesen megvan az összes ADLIB és Sound Blaster lehetősége. A kártya mellé bőséges programcsomagot kapunk az egyes alapfeladatok ellátásához. Ezek az ún. *utilityk* DOS és Windows alá, segítségükkel egyszerűen lehet felvenni és lejátszani.

Számunkra persze ez kevés. Mi nem csak használni, hanem programozni is szeretnénk a kártyát! Sajnos a Sound Blaster Pro programozásáról jóval kevesebb információ van, mint az elődeiéről. (A gyártó egyszerűen *nem hajlandó* kiadni semmiféle technikai információt.) Ezért a könyvben ismertetésre kerülő információk mellett létezhetnek egyéb olyan regiszterek, parancsok, lehetőségek, amelyek számunkra még ismeretlenek. :–(

5.1. Hardverbeállítások

I/O báziscím: Az alapbeállítás $220, ahogyan a Sound Blaster esetében is. A Sound Blaster Pro több I/O címet is felhasznál, ezért ezt a beállítást általában I/O címtartományban szokták megadni:

$220 – $23F
$240 – $25F

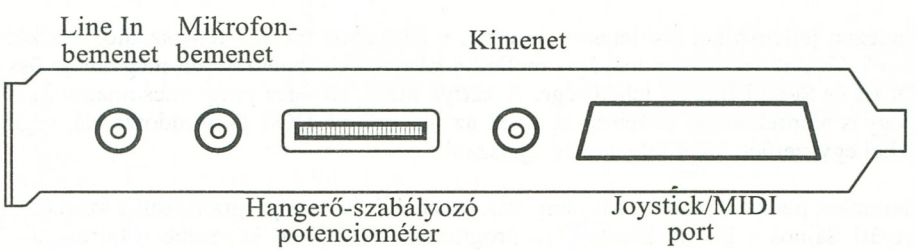
A DSP és az FM chipek programozására az első 16 I/O cím van fenntartva ($220–$22F), míg a CD-ROM vezérlő a következő 4 címet használja ($230–$233). Az ADLIB kompatibilitás miatt az FM chipek elérhetők a $388,$389 I/O címen is.

Megszakítási szint: A kártya többféle megszakítási szintet is megenged (2, 5, 7, 10), ezek közül általában az IRQ 5 az alapbeállítás.

DMA csatornaszám: Alapesetben az 1-es DMA csatorna van beállítva, de ezenkívül használható még a 0-s és a 3-as is.

5.2. Csatlakozók

A kártyán több csatlakozó is található, a legtöbb esetben az alábbi elrendezésben:



34. ábra. A hangkártya csatlakozói

Ezeken kívül a kártyán is található néhány belső csatlakozó a CD-ROM-vezérlő kábel számára, a CD-ROM audiokimenete számára, esetleg egyes újabb verziókon a PC-Speaker számára is. A *User's Guide* könyvben ennél részletesebb rajz és magyarázat található a csatlakozásokról és a *jumperek* beállításáról. Ezt egyébként érdemes figyelmesen átolvasni, és az útmutatásai alapján installálni a kártyát! A szoftver installálásával

nem lesz gond, a telepítő program minden fontosabb feladatot elvégez; felmásolja a *utilityket* és módosítja a rendszerfájlokat (autoexec.bat, config.sys). Ha a gépen Windows is van, akkor azt is a megfelelően beállítja, és létrehoz egy csoportot a programjai számára.

5.3. I/O-kiosztás

A kártya címkiosztása a Sound Blasterhez hasonlóan ebben az esetben is beállításfüggő, ezért az alábbi táblázatban megadott értékek a báziscímhez viszonyulnak. (Értelemszerűen a *w* betű írást, míg az *r* betű olvasást jelent.)

I/O relatív cím	I/O port funkció
Base+$00, w	FM chip bal csatorna cím *(Left FM address)*
Base+$00, r	FM chip bal csatorna állapot *(Left FM status)*
Base+$01, w	FM chip bal csatorna adat *(Left FM data)*
Base+$02, w	FM chip jobb csatorna cím *(Right FM address)*
Base+$02, r	FM chip jobb csatorna állapot *(Right FM status)*
Base+$03, w	FM chip jobb csatorna adat *(Right FM data)*
Base+$04, w	keverőchip cím *(Pro-Mixer address)*
Base+$05, r/w	keverőchip adat *(Pro-Mixer data)*
Base+$06, w	DSP inicializálás *(DSP Reset)*
Base+$08, w	FM chip mindkét csatorna cím *(FM address)*
Base+$08, r	FM chip mindkét csatorna állapot *(FM status)*
Base+$09, w	FM chip mindkét csatorna adat *(FM data)*
Base+$0A, r	DSP adat olvasás *(DSP read data)*
Base+$0C, w	DSP parancs/adat írás *(DSP write command/data)*
Base+$0C, r	DSP bufferállapot olvasás *(DSP read buffer status)*
Base+$0E, r	DSP adatérvényesség olvasás *(DSP data available)*
Base+$10, w	CD-ROM vezérlő adat *(CD-ROM data)*
Base+$11, r	CD-ROM vezérlő állapot *(CD-ROM status)*
Base+$12, w	CD-ROM vezérlő inicializálás *(CD-ROM reset)*
Base+$13, w	CD-ROM vezérlő engedélyezés *(CD-ROM enable)*
Adlib $388, w	ADLIB FM chip cím *(ADLIB FM address)*
Adlib $388, r	ADLIB FM chip állapot *(ADLIB FM status)*
Adlib $389, w	ADLIB FM chip adat *(ADLIB FM data)*

A címek és a regiszterek funkciói általában a fenti táblázat szerint alakulnak, de — mert a kártyának többféle verziószámú kiadása is létezik — apróbb eltérések előfordulhatnak. Ez leginkább az FM chip programozásánál fordul elő. (A kártyán egyébként két *YM–3812*

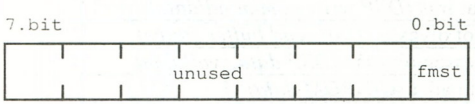
FM synthesis egység található, de áramkörileg ezek nem különülnek el egymástól. A továbbfejlesztett változatnak *OPL FM synthesis* chip a neve.) Az ADLIB kompatíbilis I/O címek nem változtathatók, azokat csak a rögzített értékekkel lehet használni.

5.4. Az FM chip programozása

Az FM chipet több I/O címen is el lehet érni. A $388,$389-es I/O címeken ADLIB kompatíbilis programozásra van lehetőség. Ha ezeket használjuk, akkor a kártya csak mono hangot ad ki, és csak az ADLIB-nél ismertetett lehetőségeket tudjuk kihasználni. Hasonlóan mindkét csatornát megszólaltatja a $228-as és $229-es port is. A sztereó FM hangzás előállítása valamivel összetettebb feladat. A hangkártya verziószámától függően többféle megoldás is létezik. A régebben kiadott kártyáknál a táblázat szerint a $220 és $221 címeken a bal, míg a $222 és $223 címeken a jobb csatorna FM egységét lehetett elérni, azaz a kártyán két független FM chip volt. Ezt a megoldást az 1.0-s verziószámú kártyáknál alkalmazták.

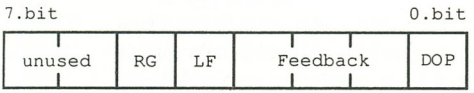
Az újabb kártyák esetében — a 2.0-s verziószámtól — az I/O cím már nincs szorosan hozzákötve a jobb vagy a bal csatornához, mert ezekbe a kártyákba az OPL–3 chipet építették. Bármelyik FM csatornát meg lehet szólaltatni mindkét oldalon. A $220/$221 és a $222/$223 I/O címekhez tehát az OPL–3 egy-egy FM egysége tartozik, és ezeket egymástól függetlenül lehet programozni. Alapesetben (bekapcsolás után) viszont ezek még nincsenek szétválasztva, ezért bármelyik portra írunk is, mindkét csatorna megszólal, azaz a hang még mono. A két egységet egy új regiszter választja szét, amely viszont kötötten mindig a $222-es báziscímen érhető el.

Regiszter $05 (bázis I/O:$222): Pro FM sztereó engedélyezés (*OPL–3*)



A regiszter 0. bitje engedélyezi a két FM chip különválasztását. Ha ezt a bitet 1-be állítjuk, akkor a kártya már külön kezeli a két I/O címre kiírt értékeket, de a hangok még mindig mindkét oldalon megszólalnak. Azt, hogy egy FM csatorna melyik oldalon szóljon, a *$C0..$C8 Feedback* regiszter két új bitje határozza meg.

Regiszter $C0..$C8: Visszacsatolás, Pro jobb/bal oldal engedélyezés



DOP: A két operátor viszonyát határozza meg. (Részletesebben lásd az ADLIB regisztereinél.)
Feedback: A visszacsatolás erősségét határozza meg. (Részletesebben lásd az ADLIB regisztereinél.)
LF: Az FM csatorna megszólalását engedélyezi a bal oldalon.
RG: Az FM csatorna megszólalását engedélyezi a jobb oldalon.
Ha az FM egységet sztereóban akarjuk használni, akkor ezen bitekkel lehet kiválasztani az adott csatorna megszólalását. Fontos, hogy alapesetben (mono módban) mindkét bitet be kell állítani, máskülönben a kimenet nem szólal meg.

Az alábbi programban ezeknek a regisztereknek a használatát mutatjuk be. A program csak a $220/$221 I/O címeket használja, mégis vezérelni lehet, hogy melyik oldal szólaljon meg. A próbák szerint a 2.0-s vagy annál nagyobb verziószámú Sound Blaster Pro kártyáknál működik így az FM chip. A programban az ADLIB kártyához készült ADLIB unitot használjuk fel, itt csupán az *AdlibBase* változót kell átírni, ha más I/O címet akarunk használni.

```
program stereo_output;

uses crt,adlib;

const
     Note : array[ 0..7] of byte=(0,2,4,5,7,9,11,0);
     Oct  : array[ 0..7] of byte=(2,2,2,2,2,2,2,3);
var
     i : integer;

{ Az eljárás lejátssza egy oktáv hangjait }
procedure Sounds(o:byte);
begin
 for i:=0 to 7 do
  begin
   Sound($00,o+Oct[ i] ,Note[ i] );
   delay(200);
  end;
 delay(200);
end;

BEGIN
 ResetADLIB;       { Inicializálás a $388/$389 I/O címen      }
 AdlibBase:=$220; { Most már $220 a báziscím                  }
 { Beállítjuk a hanggenerátor alapvető paramétereit }
 SetWave($00,$00);
```

```
 Volume($00,$ff);
 ADSR($00,$f1ff);
 FreqMultiple($00,$01);

 writeln('Mindkét oldal szól ...');
 Sounds(0);
 AdlibBase:=$222;
 WriteReg($05,$01);      { Különválasztjuk a $220 és a $222  }
                         { I/O címeken elérhető FM egységeket}
 AdlibBase:=$220;

 WriteReg($C0,$10);      { Csak a bal oldalt engedélyezzük   }
 writeln('Csak a bal oldal szól ...');
 Sounds(1);              { Kiadjuk a hangokat                }

 WriteReg($C0,$20);      { Csak a jobb oldalt engedélyezzük  }
 writeln('Csak a jobb oldal szól ...');
 Sounds(2);              { Kiadjuk a hangokat                }

 AdlibBase:=$222;
 WriteReg($05,$00);      { Tiltjuk a sztereó FM lejátszást   }
 AdlibBase:=$220;
 WriteReg($C0,$30);      { Mindkét oldalt engedélyezzük      }
 writeln('Mindkét oldal szól ...');
 Sounds(3);          { Kiadunk még egy hangsort              }
END.
```

5.4.1. Egyéb módosítások

A hangkártya kiadási évétől függően újabb módosítások is előfordulhatnak az FM chip regisztereiben. A 2.0-s verziószámú kártyáknál már az *OPL–3 FM synthesis* chipet használják, amelynek még több lehetősége van. Ezt az áramkört majd a Sound Blaster 16 hangkártyánál ismertetjük részletesen. (Ha a kártyán *OPL–3* van, akkor lehetőség kínálkozik négyoperátoros hanggenerálásra is. A négyoperátoros üzemmód használatakor a csatornák száma viszont lecsökken.)

5.5. A keverő programozása

A Sound Blaster Pro nagy előnye elődeihez képest, hogy a kártyán található audio-forrásokból keverni tudja a kimenetre kerülő jelet. A keverő chip — a továbbiakban Mixer — programozható, így lehetőség van a hangerő és egyéb paraméterek szoftverből történő állítására. A Mixer az alábbi feladatok ellátására képes:

– sztereó hangerő-szabályozás minden különálló audioegységnek –
FM chip, CD, Line In, mikrofon, kimenet (*master*);

– audioszűrő-vezérlés a bemenetekhez;
– audioszűrő-vezérlés a kimenethez;
– sztereó/mono működési mód kiválasztása;
– audiobemenet kiválasztása a felvételhez.

A Mixer programozása két I/O címen keresztül történik:

Regiszter I/O cím	Alapbeállítású I/O cím	Funkció
Base+$04	$224 (w)	Mixer címregiszter (*address*)
Base+$05	$225 (r/w)	Mixer adatregiszter (*data*)

A címregiszterbe kell írni a megfelelő Mixer regiszter számát, majd az adatregiszterből lehet leolvasni az aktuális értékét vagy beírni az újat. Különböző hardverokok miatt a címek legalsó bitje mindig 0, ezért az összes Mixer regiszter páros címen található! (Ez általánosan igaz, de előfordulhat olyan kártya is, amelynél a páratlan címek is írhatók. Sajnos a különböző verziószámú kártyák általában egy kicsit mindig különböznek egymástól, hiszen folyamatosan fejlesztik őket. A programok átvihetősége érdekében azonban nem javasoljuk az ilyen extra funkciók kihasználását, már ha egyáltalán az Olvasó ilyen pluszinformációk birtokába jut!)

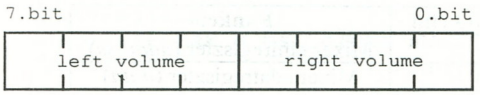
Regiszter	Regiszterfunkció
$00 *Mixer reset*	a Mixer inicializálása
$04 *DSP voice volume*	DSP kimenet hangerő
$06 *FM control*	FM vezérlés
$0A *Microphone volume*	mikrofon hangerő
$0C *Record source/filter*	felvételi eszköz és szűrő beállítása
$0E *DSP Play filter/mode*	lejátszási szűrő és mód beállítása
$20 *Input echo*	a bemenetek ki/be kapcsolása
$22 *Master volume*	kimeneti hangerő
$26 *FM/MIDI volume*	FM(MIDI) hangerő
$28 *CD volume*	CD-ROM hangerő
$2E *Line In volume*	vonali bemenet hangerő

A Mixer ezenkívül más regisztereket is tartalmazhat, ezzel azonban a gyártók nem szoktak dicsekedni. A fenti táblázat $06-os és $20-as regisztere sem volt dokumentálva, azokat a szerző a saját módszereivel fedezte fel. Számos címen előfordulhat ismétlés is, így például a fő hangerő-beállító $22 *Master volume* regiszter beállítását ki lehetett olvasni a $02,$12 és $24-es címről is. Ugyanígy a *Microphone volume* regiszter értéke a $0A cím mellett az $1A címen is megjelenik. A $30-as címtől felfelé a későbbi kártyák bővítései lehetnek, erről majd a 16 bites Sound Blasternél még lesz szó.

Regiszter $00: A Mixer inicializálása (*Mixer reset*)

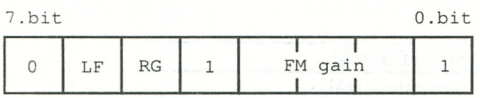
A regiszterbe írt $00 érték alapállapotba hozza a Mixert. A belső inicializálási folyamat az egyes regisztereket feltölti bizonyos előre meghatározott értékekkel.

Regiszter $04: A DSP kimeneti hangerő-beállítása (*DSP voice volume*)



A regiszter alsó négy bitje a jobb oldal, a felső négy bit pedig a bal oldal hangerejét szabályozza. A szabályozás lineáris, az 1111 jelenti a legnagyobb hangerőt, a 0000 pedig a teljesen lehalkított szintet.

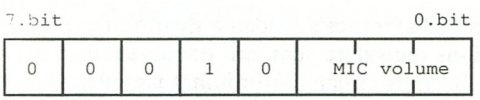
Regiszter $06: FM chip vezérlés (*FM control*)



A regiszter 1–3 bitjei az FM kimenet erősítését határozzák meg nyolclépéses lineáris alakban. A regiszter 6. bitje az FM kimenet bal csatornáját, az 5. bit pedig a jobb csatornát kapcsolja ki/be. A bitek invertáltak!

0 – bekapcsolás (*FM channel ON*)
1 – kikapcsolás (*FM channel OFF*)

Regiszter $0A: Mikrofon hangerő (*Microphone volume*)



A regiszter a mikrofonbemenet hangerejét definiálja az alsó három biten. E hangerő befolyásolja a mikrofon hangját a kimeneten, ha a mikrofon erősítése engedélyezett, és befolyásolja a felvétel hangerejét is.

Regiszter $0C: Felvételi forráseszköz és szűrő kiválasztása (*Record Source/Filter*)

7.bit				0.bit
???	???	Filter	Source	???

Ez a regiszter szolgál a bemeneti eszköz és szűrő kiválasztására a felvételhez. A *Source* mezővel a felvétel forrását lehet beállítani:

00 – mikrofon (*Microphone*)
01 – audio-CD (*CD Audio input*)
10 – nem használt (*unused*)
11 – vonali bemenet (*Line In*)

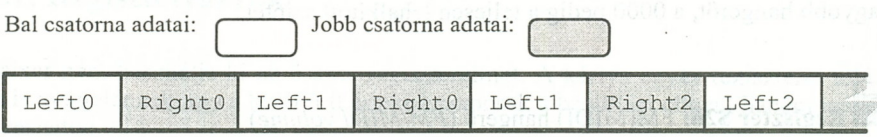
A *Filter* mező kiválasztja a felvételhez használt szűrő típusát:

001 – magas (*Input Filter High*)
010 – alacsony (*Input Filter Low*)
110 – nincs szűrés (*Input Filter Off*)
Bármely más kombináció érvénytelen!

Regiszter $0E: DSP lejátszási mód és szűrő kiválasztása (*DSP Play Mode/Filter*)

7.bit							0.bit
???	???	Flt	1	???	???	Vstc	1

Ezzel a regiszterrel a DSP lejátszási módját és a kimenet szűrését tudjuk beállítani. A DSP normál esetben a Sound Blasterrel kompatíbilis mono módban dolgozik. A sztereó lejátszást csak DMA átvitellel lehet megvalósítani, és akkor is egy sajátos formában. Ha a *Vstc* (*Voice Stereo Channels*) bit be van állítva, akkor a hangmintákat felváltva egyszer a bal, majd a jobb csatornára küldi ki a kártya. Egy-egy minta egybájtos, ezért ilyenkor a következő formában kell tárolni a hangot:



35. ábra. Sztereó hangmintaformátum

Az ábra szerint tehát minden páros számú bájt a bal csatornára, minden páratlan bájt pedig a jobb csatornára kerül. Ha egy adott hang lejátszását mono módról sztereó módra

változtatjuk, akkor a mintavételi frekvenciát is át kell állítani, hiszen ha azt nem változtatnánk meg, akkor a mintavételezés feleakkora frekvenciával folyna tovább!

Vstc: 0 – mono mód
1 – sztereó mód

A regiszter *Flt* bitje a kimeneti szűrőt kapcsolja ki és be:

0 – kimeneti szűrés engedélyezve (*Output Filter On*)
1 – kimeneti szűrés tiltva (*Output Filter Off*)

Regiszter $20: Bemenetek erősítésének engedélyezése (*Input echo*)

7.bit							0.bit
???	???	???	1	Line	CD	Mic	1

Az összes bemeneti vonal a kimeneten erősítve hallgatható. Hogy ezen vonalak közül a Mixer mely vonalat erősítse, azt ez a regiszter adja meg:

Mic – a mikrofonbemenet erősítése (0: engedélyezve, 1: tiltva)
CD – a CD audiobemenet erősítése (0: engedélyezve, 1: tiltva)
Line – a vonali bemenet erősítése (0: engedélyezve, 1: tiltva)

Ha a regiszter megfelelő bitjét 1-be állítjuk, akkor ezzel kikapcsoljuk az adott eszköz hangját.

Regiszter $22: Kimeneti hangerő (*Master Volume*)

7.bit							0.bit
left volume				right volume			

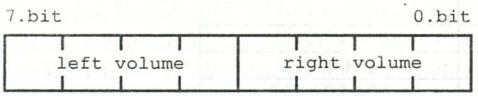
Ez a regiszter egyszerre állítja az összes eszköz hangerejét. A hangerő a jobb és a bal oldal számára külön négy-négy biten állítható. A szabályozás lineáris, az 1111 jelenti a legnagyobb hangerőt, a 0000 pedig a teljesen lehalkított szintet.

Regiszter $26: FM(MIDI) hangerő (*FM/MIDI volume*)

7.bit							0.bit
left volume				right volume			

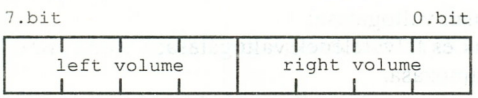
Az FM chip hangerejének beállítása. A hangerő a jobb és a bal oldal számára külön négy-négy biten állítható függetlenül attól, hogy az FM chip sztereó vagy mono módban van-e. A szabályozás lineáris, az 1111 jelenti a legnagyobb hangerőt, a 0000 pedig a teljesen lehalkított szintet.

Regiszter $28: CD hangerő (*CD volume*)



Az audio-CD bemenetre kapcsolt CD erősítése. A kereskedelemben kapható CD-ROM olvasók általában le tudják játszani az audio-CD-ket is. A hangkártyák többségén található egy bemenet a CD audiojelének erősítéséhez. Az egyszerűbb kártyákon a hangerő csak a hangerő-szabályozó potenciométer segítségével állítható, a Sound Blaster Pro viszont lehetőséget nyújt a szoftverből történő szabályozásra is. A hangerő a jobb és a bal oldal számára külön négy-négy biten állítható. A szabályozás lineáris, az 1111 jelenti a legnagyobb hangerőt, a 0000 pedig a teljesen lehalkított szintet.

Regiszter $2E: Vonali bemenet hangerő (*Line In volume*)



A Line In bemenet hangerejének szabályozása. A hangerő a jobb és a bal oldal számára külön négy-négy biten állítható. A szabályozás lineáris, az 1111 jelenti a legnagyobb hangerőt, a 0000 pedig a teljesen lehalkított szintet.

5.5.1. Regisztertükrök

A Mixer nem használja ki az összes regisztercímet. A kártya egyes regiszterei több helyen is megjelennek, így a beállított értékek nemcsak a beállítási címen, hanem máshol is kiolvashatók.

Ezeket a tükörcímeket tüntettük fel az alábbi táblázatban:

Eredeti regiszter	Tükörregiszter
$00 *Mixer reset*	–
$04 *DSP voice volume*	– (???)
$06 *FM control*	–
$0A *Microphone volume*	$1A
$0C *Record source/filter*	–
$0E *DSP Play filter/mode*	– (???)
$20 *Input echo*	$10 (???)
$22 *Master volume*	$24,$02,$12
$26 *FM/MIDI volume*	$16
$28 *CD volume*	$08,$18,2A$2C
$2E *Line In volume*	$1E (???)

5.5.2. Az SBPRO unit

A hangkártya programozásához készítettünk egy rövid unitot, amellyel kihasználhatók a Sound Blaster Pro új lehetőségei. A unit együtt használható a DSP és az SB_DMA unitokkal. A következő újításokat kellett bevezetni:

– az FM chip sztereó és mono módjának váltogatása;
– a DSP sztereó és mono lejátszásának és felvételének váltogatása;
– a Mixer egyes regisztereinek programozása.

Az eljárásokon kívül definiáltuk a Sound Blaster Próhoz szükséges állandókat — a DSP parancsokat és az eszközök azonosítóit —, amelyek a programozás során hasznosak lehetnek.

```
unit sbpro; { SBPRO.PAS }

interface uses dsp;

 const
       { Sound Blaster Pro új DSP parancsok                    }
       DSPirqReq          = $f2;  { Szoftvermegszakítás-kérés  }
       DMAmonoRecord      = $a0;  { Mono módú DMA felvétel     }
       DMAstereoRecord    = $a8;  { Sztereó módú DMA felvétel  }
       DMAhighSpeedplay   = $90;  { Nagy sebességű lejátszás   }
       DSPsetBlockSize    = $48;  { Nagy sebességű blokkméret  }
       DSPhsDMAplay       = $91;  { Nagy sebességű lejátszás   }
```

```
DSPhsDMArecord    = $99;  { Nagy sebességű felvétel      }
DSPid             = $e0;  { DSP-azonosítás               }
```

A kártya DSP egysége általában 3.00 vagy magasabb verziószámú. A sztereó felvétel és lejátszás, valamint a nagyobb mintavételezési frekvencia miatt bevezettek néhány újabb DSP parancsot.

DSPirqReq ($F2): DSP megszakításkérés programból. A parancs hatására a DSP egy megszakítást kér. Ez akkor lehet hasznos, ha a beállított megszakítási vonalat programból akarjuk letapogatni. Ehhez beállítjuk a megszakításvezérlőt, engedélyezzük az összes lehetséges Sound Blaster Pro megszakítási vonalat, és mindegyikhez írunk egy rövid kiszolgáló rutint. A *DSPirqReq* parancs hatására a program a megfelelő beállított vonal rutinját hívja meg, így a beállítás detektálható.

DMAmonoRecord ($A0): Mono felvételi módba kapcsolja a DSP egységet. Ez egyébként az alapbeállítás is. A parancs természetesen csak akkor hatásos, ha a DSP előzőleg sztereó módban volt.

DMAstereoRecord ($A8): Sztereó felvételi módba kapcsolja a DSP egységet. A kártya verziószámától függően itt két mód is kínálkozik a sztereó felvételre:

1. bájtszekvencia: $48, LSB, MSB, $99
2. bájtszekvencia: $A8, $99, LSB, MSB

(Véleményünk szerint a digitalizálást felesleges az Olvasónak megoldania, mert a mellékelt programok ezt tökéletesebben megteszik. A hangminták lejátszása persze már egészen más.)

DSPid ($E0): A DSP egység azonosítása és inicializálása.

```
{ Kapcsolóállapotok                          }
On     = true;        { Bekapcsolt állapot    }
Off    = false;       { Kikapcsolt állapot    }

{ A bemeneti szűrő üzemmódjai     }
fLow   = $00;         { Alacsony   }
fHigh  = $01;         { Magas      }
fOff   = $02;         { Kikapcsolt }

{ Bemeneti eszközök a felvétel forrásához }
iMic   = $00;         { Mikrofon   }
iCD    = $01;         { Audio CD   }
iLine  = $03;         { Line In    }

{ A bemeneti eszközök erősítése   }
oMic   = $02;         { Mikrofon   }
oCD    = $04;         { Audio CD   }
oLine  = $08;         { Line In    }
```

```
{ Alacsony szintű eljárások a Mixerhez    }
{ Regiszterbeállítás                      }
procedure SetMixer(regnum,value:byte);
{ Regiszterlekérdezés                     }
function  GetMixer(regnum:byte):byte;

{ Magas szintű rutinok a Mixer egyes elemeihez }
{ Inicializálás                           }
procedure ResetMixer;
{ A kimeneti hangerő beállítása }
procedure SetMasterVolume(volume:byte);
{ A DSP hangerő beállítása      }
procedure SetDSPVolume(volume:byte);
{ A Line In hangerő beállítása  }
procedure SetLineInVolume(volume:byte);
{ A CD hangerő beállítása       }
procedure SetCDVolume(volume:byte);
{ Az FM chip hangerő beállítása }
procedure SetFMVolume(volume:byte);
{ Sztereó/Mono kimenetválasztás }
procedure StereoOutput(state:boolean);
{ A kimeneti szűrő állapota     }
procedure OutputFilter(state:boolean);
{ Mikrofon hangerő              }
procedure MicLevel(level:byte);
{ Bemeneti forrás és szűrő      }
procedure InputDevice(device,filter:byte);
{ A sztereó FM beállítása       }
procedure StereoFM(state:boolean);
{ FM vezérlés                   }
procedure FMcontrol(left,right:boolean; gain:byte);
{ Kimeneti erősítések           }
procedure OutputSwitch(device:byte; state:boolean);

implementation

procedure SetMixer(regnum,value:byte); assembler;
asm
  mov   dx,BaseAddr          { SB Pro báziscím ($220)      }
  add   dx,$0004             { Mixer címregiszter ($224)   }
  mov   al,regnum
  out   dx,al                { Kiküldjük a címet           }
  inc   dx                   { Mixer adatregiszter ($225)  }
  mov   al,value
  out   dx,al                { Kiküldjük az adatot         }
end;

function  GetMixer(regnum:byte):byte; assembler;
asm
```

```
  mov   dx,BaseAddr            { SB Pro báziscím ($220)      }
  add   dx,$0004               { Mixer címregiszter ($224)   }
  mov   al,regnum
  out   dx,al                  { Kiküldjük a címet           }
  inc   dx                     { Mixer adatregiszter ($225)  }
  in    al,dx                  { Beolvassuk az adatot        }
end;
```

SetMixer,GetMixer: A Mixer alacsony szintű elérése. A két eljárással be tudjuk állítani és le tudjuk kérdezni bármelyik Mixer regiszter értékét.

```
procedure ResetMixer;
begin
 SetMixer($00,$00);
end;
```

ResetMixer: A Mixer alapállapotba hozása. Az eljárás inicializálja a Mixert, és a regisztereket feltölti a hardver által meghatározott értékekkel.

```
procedure SetMasterVolume(volume:byte);
begin
 SetMixer($22,volume);
end;

procedure SetDSPVolume(volume:byte);
begin
 SetMixer($04,volume);
end;

procedure SetLineInVolume(volume:byte);
begin
 SetMixer($2e,volume);
end;

procedure SetCDVolume(volume:byte);
begin
 SetMixer($28,volume);
end;

procedure SetFMVolume(volume:byte);
begin
 SetMixer($26,volume);
end;
```

Hangerő-beállítások. A fenti eljárásokkal az egyes Mixer-források hangerejét lehet beállítani. A hangerőt egy bájtparaméter definiálja, ahol az alsó négy bit a jobb oldal hangerejét, a felső négy bit pedig a bal oldal hangerejét adja meg.

```
procedure StereoOutput(state:boolean);
var b:byte;
begin
 b:=GetMixer($0e);
 if state then b:=b or $02 else b:=b and $fd;
 SetMixer($0e,b);
end;
```

StereoOutput: A DSP kimenetének sztereó módba kapcsolása. Az eljárás a DSP egységet sztereó üzemmódba kapcsolja át. A sztereó üzemmód csak DMA átvitellel valósítható meg! Az átállításhoz használjuk a definiált *On*, *Off* állandókat!

```
procedure OutputFilter(state:boolean);
var b:byte;
begin
 b:=GetMixer($0e);
 if state then b:=b or $20 else b:=b and $df;
 SetMixer($0e,b);
end;
```

OutputFilter: A kimeneti szűrő be/ki kapcsolása. Az eljárással a kimenet szűrését lehet vezérelni. Használjuk az *On*, *Off* állandókat!

```
procedure MicLevel(level:byte);
begin
 SetMixer($0a,level and 7);
end;
```

MicLevel: A mikrofon hangerejének beállítása. A hangerőt csak a *level* paraméter alsó három bitje definiálja, a többi bit értéke érdektelen.

```
procedure InputDevice(device,filter:byte);
begin
 SetMixer($0c,((device and 3) shl 1)or((filter and 3) shl 3));
end;
```

InputDevice: A bemeneti eszköz kiválasztása és a bemeneti szűrő beállítása. Az eljáráshoz használjuk a definiált *iMic*, *iCD*, *iLine* (bementi eszköz) és *fLow*, *fHigh*, *fOff* (szűrőtípus) állandókat!

```
procedure StereoFM(state:boolean);
begin
 port[ BaseAddr+2] :=$05;                { $222 : FM chip ! }
 pause(25);
 port[ BaseAddr+3] :=byte(state) and 1; { $223, 0.bit = 1   }
 pause(100);
end;
```

StereoFM: Az FM chip kimenetét állítja át. A beállítástól függően az FM $220/$221 és $222/$223 I/O címeken található FM egységeket lehet külön-külön elérni. A beállításhoz használjuk az *On*, *Off* állandókat!

```
procedure FMcontrol(left,right:boolean; gain:byte);
var lf,rg,gn:byte;
begin
 lf:=(byte(left)  and 1) shl 6;              { 6. bit   }
 rg:=(byte(right) and 1) shl 5;              { 5. bit   }
 gn:=(gain and 7) shl 1;                     { 1-3. bit }
 SetMixer($06,lf or rg or gn);
end;
```

FMcontrol: Az FM chip vezérlése. Az eljárással a $06-os Mixer regisztert tudjuk beállítani. Az erősítést 3 biten kell megadni, a bal és a jobb csatorna kapcsolgatásához használjuk a definiált *On*, *Off* állandókat!

```
procedure OutputSwitch(device:byte; state:boolean);
var old : byte;
begin
 old:=GetMixer($20);
 old:=old and not device;
 if not state then old:=old or device;
 SetMixer($20,old);
end;
```

OutputSwitch: A bemenetek erősítésének vezérlése. Ezzel az eljárással azt lehet megadni, hogy a bemenetek közül melyik legyen aktív. (Természetesen egyszerre több bemenet is aktív lehet.) Az eszköz (*device*) megadásához használjuk az *oMic*, *oCD*, *oLine* definiált állandókat, az állapotot (*state*) az *On*, *Off* állandókkal adhatjuk meg.

```
END. of unit
```

5.6. Sztereó hang a kimeneten

A most következő rövid programmal a kártya sztereó használatát mutatjuk be. A sztereó mód bekapcsolása a fenti *StereoOutput* eljárással igazán egyszerű. Az üzemmód átállítása után már ugyanazokat a feladatokat kell csak elvégezni, mint a normál Sound Blaster esetében, ezért az SB_DMA unit tökéletesen megfelel!

```
program stereo_out;

uses crt,dsp,sb_dma,sbpro;
```

```
var
    i : word;
    p : memptr;

BEGIN
 clrscr;
 { Lefoglalunk egy 32K hosszúságú helyet a hangmintának }
 getmem(p,32768);
 fillchar(p^,32768,$80);
 for i:=0 to 16383 do
  begin
   { A bal oldali minta szinuszhullám   }
   p^[ i*2] :=128+round(60*sin(i*pi/8));
   { A jobb oldali minta négyszöghullám }
   p^[ i*2+1] :=128-60*byte(i and 16=0);
  end;

 { A DSP inicializálása }
 if not DSPreset then
  begin
   writeln('Nem lehet inicializálni a DSP egységet!');
   halt;
  end;

 { A Mixer inicializálása }
 ResetMixer;
 SetMasterVolume($88);          { Fél hangerő a kimeneten      }
 SetDSPVolume($FF);             { A DSP hangereje maximális    }
 StereoOutput(On);              { Sztereó üzzemmód             }
 writeln('Lejátszás indul, mindkét csatorna szól');
 StartPlay($c0,p,i);            { Lejátszás                    }
 while OldIRQseg<>$00 do;       { Várakozás a DMA átvitel végére }
 delay(1000);                   { Szünet                       }

 SetDSPVolume($F0);             { A jobb oldal lehalkítása     }
 StereoOutput(On);
 writeln('Lejátszás indul, csak a bal csatorna szól');
 StartPlay($c0,p,i);            { Lejátszás                    }

 while OldIRQseg<>$00 do;       { Várakozás a DMA átvitel végére }
 delay(1000);                   { Szünet                       }

 SetDSPVolume($0F);             { A bal oldal lehalkítása      }
 StereoOutput(On);
 writeln('Lejátszás indul, csak a jobb csatorna szól');
 StartPlay($c0,p,i);            { Lejátszás                    }
 while OldIRQseg<>$00 do;       { Várakozás a DMA átvitel végére }

 writeln('Kész. [Enter]');
 readln;
```

```
 freemem(p,32768);                { Felszabadítjuk a heapet, kilépés }
END. of program
```

Hogyan működik a program? Az első feladat a hangminták előállítása. Ehhez egy 32K hosszúságú területet használunk a memóriában. A $0E regiszter leírásánál megadtuk az adatformátumot: minden páros bájt a bal csatornára, minden páratlan bájt a jobb csatornára kerül. Ezért minden páros helyre egy szinuszhullám mintáit, míg minden páratlan helyre egy négyszöghullám mintáit helyezi el a program elején található *for* ciklus. A hang előállítása után inicializáljuk a DSP egységet és a Mixert. A kimenet hangerejét 50%-os kivezérlésre állítottuk ($88), ez természetesen tetszés szerint módosítható. A DSP hangereje az első esetben mindkét oldalon maximális ($FF). Ez után már csak a sztereó üzemmódot kell bekapcsolni a *StereoOutput(On)* utasítással, és indulhat a lejátszás.

A kimenet háromszor szólal meg. Az első esetben mindkét oldal szól, de a bal csatornán egy szinusz, a jobb csatornán pedig egy négyszöghullám hallható. A második esetben a jobb csatorna hangerejét levettük, így csak a bal oldali hangot lehet hallani. Végül a jobb oldali hang hallható. Ha a *StereoOutput(On)* utasítást *StereoOutput(Off)*-ra cseréljük, akkor a lejátszás mono, tehát minden egyes minta kikerül mindkét oldalra. Ez a hullám igen sok felharmonikust tartalmazó bonyolult jel, ezért a hang ilyenkor érezhetően más.

Befejezésül még egy jelenségre szeretnénk felhívni a figyelmet. Ha a heapfoglalás DMA laphatárt sért — azaz az átvitelben van egy megszakítás —, akkor a hang közepében egy jól hallható kattanás van. Ennek az az oka, hogy a megszakítás idején a DMA átvitel szünetel. Az SB_DMA unit megszakítása meglehetősen sokáig tart, ezért ez a hangban is érezhető hibát okoz. A hiba kiküszöbölésére az egyetlen megoldás egy nagyon gyors megszakítás, amely képes rendkívül rövid idő alatt újraprogramozni a DMA vezérlőt. Sajnos ez tipikus hiba, amely már az átvitel módjából adódik, és igen komoly gondot tud okozni. A Sound Blaster 16 kártyánál szó lesz a kettős buffer (*auto-init* típusú DMA) technikáról, amely megoldja ezt a problémát.

6. A Sound Blaster 16 hangkártya

A könyvben eddig ismertetett hangkártyák csupán 8-bites hangdigitalizálásra és 8-bites hangminták visszajátszására voltak képesek. A hifi hangminőség eléréséhez a 8-bites felbontás kevés. A Sound Blaster család újabb tagja, a Sound Blaster 16 viszont — ahogyan azt a neve is mutatja — 8 és 16 biten is tud mintavételezni és lejátszani. A hifi minőségű hangok digitális tárolásához van néhány szabványban rögzített érték, amelyet feltétlenül be kell tartania minden ilyen eszközöket gyártó cégnek. Ezek közül a legfontosabb szabványelőírás:

– 16 bites minták (tárolás és visszajátszás);
– sztereó hang; :^)
– 44.1 kHz-es mintavételi frekvencia mindkét csatornához;
– megfelelő D/A konverter és aluláteresztő szűrő a kimenetre.

Az audio-CD lejátszók általában teljesítik ezen követelményeket, ami persze attól is függ, hogy ki gyártja a lejátszót, és milyen elemekből építi fel. A Sound Blaster 16 hangkártya szintén teljesíti ezeket az előírásokat. A hangkártya természetesen teljesen kompatíbilis az előzőekben tárgyalt kártyákkal — azaz az ADLIB-bel, a Sound Blasterrel, és Sound Blaster 2.0-val —, ugyanakkor számos új lehetőség található rajta. A Sound Blaster Próval viszont nem kompatíbilis, pontosabban nem DSP parancs kompatíbilis! A kártya tudja mindazt, amit az SB Pro, de egyes feladatokat más DSP parancsokkal kell megadnunk. Ez sajnos éppen a sztereó lejátszásnál jelentkezik leginkább. Tapasztalataink szerint a kártya az SB Pro meghajtóit sem tudja használni, így valószínű, hogy a DSP parancsokon kívül egyéb eltérések is vannak. :^(

Nos, a 16 bites hangminta szép és jó, de azért álljunk meg egy pillanatra! A CD minőségű hangnak ára van. Végezzünk el egy rövid számítást! Tegyük fel, hogy egy sztereó hangot 16 biten mintavételezünk 44.1 kHz-cel. Ekkor egy másodperc alatt n bájtot kell tárolnunk, ahol

$$n = SampleRate \times channels \times recsize = 44100 \cdot 2 \cdot 2 = 176400$$

Azaz a mintavételezési frekvencia szorozva a csatornák számával és szorozva egy minta hosszával. Jelen esetben a mintavételi frekvencia 44.1 kHz, a csatornák száma 2, hiszen sztereó, és az adathossz is 2, mert 16 bit az két bájt. Láthatjuk, hogy egy másodpercnyi hang tárolásához nagyjából 172 Kbájt szükséges. Tehát az ilyen hangminták igen sok memóriát igényelnek. Egy 60 másodperces hangminta tárolásához 10.09 Mbájt memória szükséges, ami meglehetősen sok! A hangminták felbontása és a felhasznált memória tekintélyes mérete miatt kompromisszumot kell találni a kívánt minőség eléréséhez.

Tehát a CD minőség ára az óriási memóriafelhasználás. Nem véletlen, hogy a CD-k nagy mennyiségű adat tárolására képesek (500-600 Mbájt).

6.1. A Sound Blaster 16 lehetőségei

Foglaljuk össze röviden, milyen lehetőségei vannak a Sound Blaster 16 hangkártyának:

- **FM chip**
  - bővített OPL–3 FM chip, amely négyoperátoros hangkeltésre is alkalmas
  - teljes kompatibilitás az előző ADLIB és Sound Blaster változatokkal
  - 18 kétoperátoros vagy 6 négyoperátoros hangcsatorna
- **Továbbfejlesztett DSP egység** (*ASP, Advanced Sound Processor*)
  - 8- és 16 bites felvétel és lejátszás
  - sztereó és mono felvétel és lejátszás mind 8-, mind 16 bites módban
  - dinamikus szűrők a felvételhez és a visszajátszáshoz
  - két DMA csatorna a DMA vezérelt felvételhez és lejátszáshoz
  - kibővített DSP parancskészlet
  - lineáris lépésekben programozható mintavételi frekvencia 4..45 kHz-ig
- **Bővített keverő** (Mixer) chip az audioforrások keveréséhez
  - a bemeneti eszközök tetszőleges keverése
  - magas és mély állítási lehetőség (*Treble/Bass*)
- **Szoftverből állítható hangerő minden bemenethez**
  - kimeneti hangerő, FM chip, vonali bemenet, mikrofon, CD, Speaker
- **Beépített sztereó erősítő**
  - 4 watt/csatorna, 4 ohmos hangszórókhoz
  - automatikus vagy fix erősítés a mikrofonhoz (*AGC/FGA Automatic Gain Control/Fixed Gain Control*)
  - külső vagy belső erősítő a kimenetekhez
- **MIDI interfész**
  - Sound Blaster vagy MPU401–UART kompatíbilis mód
- **CD-ROM interfész**
- **Hardverbeállítások szoftverből**
  - szoftverből állítható megszakítási szint
  - szoftverből állítható 8- és 16 bites DMA csatorna szám

6.1.1. Hardverbeállítások

I/O báziscím: A kártya báziscímét az előző kártyáknál megismert módon, *jumperekkel* lehet beállítani. A Sound Blaster 16 négy lehetséges báziscímet tud kezelni:

I/O bázis	I/O címtartomány
$220	$220 ... $233
$240	$240 ... $253
$260	$260 ... $273
$280	$280 ... $293

Ezeket a címeket az IOS0,IOS1 nevű *jumperekkel* tudjuk beállítani. A megadott tartományban csak a hangkeltéshez tartozó programozható eszközök érhetők el, tehát a DSP egység, a Mixer és az FM chip. A kártya egyéb részeit más-más I/O területre helyezték el.

UART chip:

MPU–401 UART I/O bázis	UART címtartomány
$300	$300 ... $301
$330	$330 ... $331

A további beállításhoz még néhány *jumper* használható:

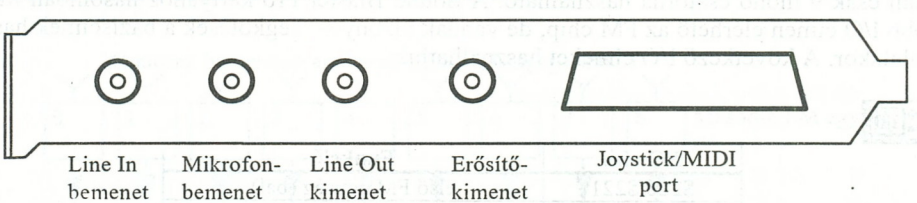
A jumper neve	A jumper funkciója
MSEL	MPU UART I/O cím kiválasztás
MPUEN	MPU UART engedélyezés/tiltás
CRE	CD-ROM vezérlő engedélyezés/tiltás
JYEN	joystick engedélyezés/tiltás
OPSL	bal oldali kimeneti erősítő kiválasztás (külső/belső)
OPSR	jobb oldali kimeneti erősítő kiválasztás (külső/belső)

DMA csatorna és IRQ szint kiválasztás:

A hangkártya által használt DMA csatornákat és a megszakítási szintet szoftverből tudjuk beállítani a megfelelő regiszterben. Az installáló szoftver elvégzi a beállítást, amelyet azután a különböző rendszerállományok átírásával lehet megváltoztatni. Erről bővebben olvashatunk a *User's Guide* felhasználói kézikönyvben.

6.1.2. Csatlakozók

Az alábbi ábrán a Sound Blaster 16 csatlakozói láthatók:



36. ábra. A hangkártya csatlakozói

A leglényegesebb változás az előző kártyákhoz képest a hangerő-szabályozó potenciométer elmaradása. A gyártók bizonyára úgy gondolták, hogy a szoftver-szabályozás

elegendő, így a potenciométer valójában felesleges. Ez így is van, de sajnos egyes programok nem állítják be megfelelően a hangerőt, így az vagy nagyon halk, vagy nagyon hangos. (Ez persze inkább a programok hibája, semmint a kártyáé! Egy jól átgondolt programnál ez nem fordulhat elő!) A másik újdonság a vonali kimenet, amellyel a kártya kimeneti jelét más audioeszközökbe lehet bevezetni. A kártyán is található csatlakozó az audio-CD és a speaker számára. (A régebbi kártyáknál előfordulhat még a hangerő-szabályozó potenciométer, ezeknél azonban egy kicsit más a csatlakozók elrendezése! Ha ilyen kártyánk van, akkor a kézikönyvben részletesebben megtalálható a kiosztás.)

6.1.3. Telepítés (installálás)

A hangkártya telepítése előtt célszerű figyelmesen elolvasni a *User's Guide* kézikönyv utasításait, és annak megfelelően eljárni. A szoftverek installálása itt is — ahogyan a Sound Blaster Pro esetében — kényelmes, mert az installáló program minden szükséges beállítást elvégez, és megfelelően módosítja a rendszerfájlokat. (*autoexec.bat*, *config.sys*, *win.ini* stb...).

A kártyához mellékelt szoftvereket úgy készítették el, hogy azok az alacsony szintű *driver* programokat használják. Ezzel elérték, hogy a programok viszonylag rövidek, viszont szükség van a memóriarezidens meghajtókra. Ez kellemetlen lehet, hiszen az ilyen programok állandóan lefoglalnak bizonyos területeket a gép memóriájából. Mint említettük, a telepítő program ezeket a meghajtókat automatikusan elhelyezi a *config.sys* fájlban. A játékprogramok általában nem igénylik őket, ezért hasznos lehet Boot menüt készíteni a gépre, hogy felálláskor a gép mindig csak az éppen szükséges eszközöket helyezze el a memóriában.

6.2. Az OPL–3 FM chip programozása

Az FM egység alapesetben ADLIB kompatíbilis módban dolgozik, azaz bekapcsolás után csak 9 mono csatorna használható. A Sound Blaster Pro kártyához hasonlóan itt is több I/O címen elérhető az FM chip, de vannak bizonyos megkötések a báziscímek használatakor. A következő I/O címeket használhatjuk:

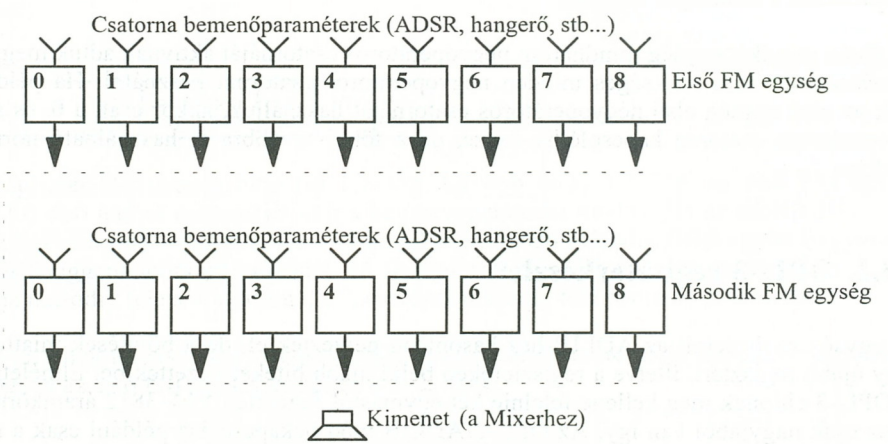
I/O cím	Funkció
$220/$221	első FM egység (bal)
$222/$223	második FM egység (jobb)
$228/$229	mindkét FM egység
$388/$389	első FM egység (bal)
$38A/$38B	második FM egység (jobb)

Zárójelben jeleztük, hogy az egység melyik oldalhoz tartozik. Ez egyébként nem éppen a legszerencsésebb felosztás, hiszen mindkét egység mindkét oldalon képes megszólalni. Ezért — noha az angol irodalomban mindenhol a $220/$221 I/O címet használják bal, a $222/$223 I/O címet pedig jobb oldalként — mi mégis az „első FM egység" ($220/$221) és „második FM egység" ($222/$223) elnevezéseket fogjuk használni.

6.2.1. Négyoperátoros FM hanggenerálás

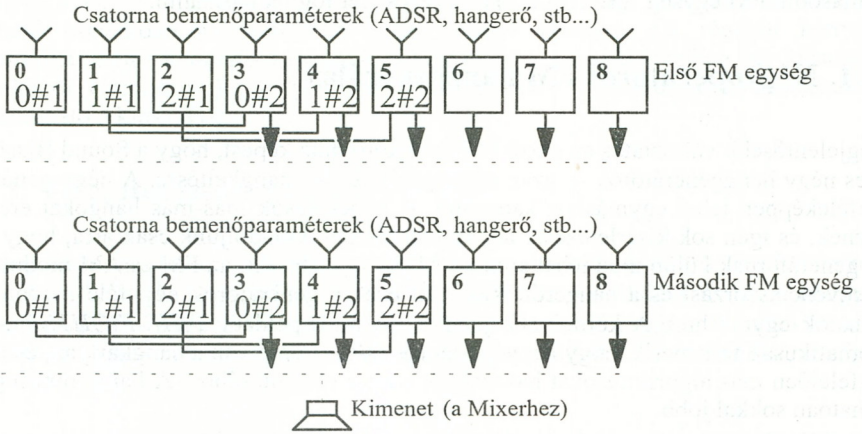
A legjelentősebb változtatás az előző FM áramkörökhöz képest, hogy a Sound Blaster 16 képes négy hanggenerátoros — azaz négyoperátoros — hangkeltésre. A négy generátort többféleképpen lehet egymáshoz kapcsolni. A kapcsolások más-más hangokat eredményeznek, és igen sok kísérletezésre adnak lehetőséget. Gondoljunk csak arra, hogy minden generátornak külön meg lehet adni az ADSR összetevőit, az FM és AM modulációt, a frekvenciaszorzást és a hangerőt. Ráadásul ezek a generátorok négyféle módon kapcsolhatók egymáshoz. A komolyabb játékok — mint például a *TIE-FIGHTER* :-) — automatikusan felismerik, hogy négyoperátoros *OPL–3* chip van a hangkártyán, és ennek megfelelően más algoritmusokat használnak a zene előállításához. A hang minősége jól hallhatóan sokkal jobb.

Sajnos a négyoperátoros hangok előállításához dupla annyi generátor kell, mint a kétoperátoros hangokhoz, ezért ilyenkor az FM csatornák száma lecsökken. Alapesetben 9 csatorna van mindkét FM egységben, azaz 18 csatornát lehet használni. Ha négyoperátoros módba kapcsolunk, akkor egy egységen három-három csatorna kapcsolódik össze, így ilyenkor csak 6 csatorna van egységenként. Először nézzük meg, hogyan vannak összekapcsolva a generátorok normál kétoperátoros módban !



37. ábra. Az FM egységek kétoperátoros módban

Négyoperátoros módban viszont mindkét egység első hat csatornája páronként összekapcsolódik, ezért olyankor csupán hat-hat — három négyoperátoros és három kétoperátoros — különálló csatornát lehet használni.



38. ábra. Az FM egységek négyoperátoros módban

Az *n#m* kifejezés az *n*-edik négyoperátoros csatorna *m*-edik egységét jelenti. Az ilyen összekapcsolásban az elsőt szoktuk „első félnek", a másodikat pedig „második félnek" nevezni (vaslogika ;–)). A „fél" elnevezés azért alakult ki, mert ebben az esetben egyetlen csatorna csak fél csatornának számít, hiszen csak két operátort — két hanggenerátort — tartalmaz a szükséges négyből.

Az ábrán mindkét egység mindhárom négyoperátoros csatornáját aktívnak adtuk meg, de természetesen nem szükséges minden négyoperátoros csatornát használni. Ha például csak az első egység első négyoperátoros csatornáját használjuk, akkor csak a 0. és a 3. kétoperátoros csatorna kapcsolódik össze, de a többi továbbra is használható normál módban.

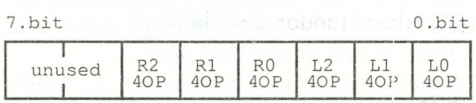
6.2.2. OPL–3 regiszterkészlet

Az egység regisztereit az ADLIB-hez hasonlóan helyezték el, de a bővítések miatt néhány újabb regisztert, illetve a regisztereken belül újabb biteket vezettek be. Elméletileg az OPL–3 chipnek meg kellene felelnie két egymástól független YM–3812 áramkörnek, de ez csak nagyjából van így. Az OPL–3/ADLIB mód bekapcsolása például csak a második egységgel lehetséges, míg az időzítők csak az első egységben találhatók meg.

Ugyanígy nem lehet ütőhangszereket használni a második egységnél, viszont a négyoperátoros mód bekapcsolásához egy második egységbeli regisztert kell módosítani. A következő táblázatban összefoglaltuk a regisztereket mindkét egységhez, megemlítve az eltéréseket is. A már ismert regiszterek bitkiosztásaira nem térünk ki, azok az ADLIB kártyáról szóló fejezetben részletesen megtalálhatók.

Regiszter	Regiszterfunkció	Első egység	Második egység
$01	hullámforma-kiválasztás *(EWC)*	van	van
$02	első időzítő adat *(timer #1)*	van	nincs (??)
$03	második időzítő adat *(timer #2)*	van	nincs (??)
$04	időzítő vezérlő *(timers control)*	van	nincs
$04	négyoperátoros mód vezérlő *(4OP)*	nincs	van
$05	OPL–3/ADLIB kiválasztás *(OPL–3)*	nincs	van
$08	FM/beszédszintézis mód *(FM sel)*	van	nincs
$20..$35	effektusok *(AM/vibrato/multiple)*	van	van
$40..$55	kimeneti jelszint *(output level)*	van	van
$60..$75	burkológörbe *(Attack/Decay)*	van	van
$80..$95	burkológörbe *(Sustain/Release)*	van	van
$A0..$A8	frekvencia alsó bájt *(frequency low)*	van	van
$B0..$B8	frekvencia felső bájt *(frequency hi)*	van	van
$BD	ritmusvezérlő *(rhytm control)*	van	nincs
$C0..$C8	visszacsatolás-vezérlő *(feedback)*	van	van
$E0..$F5	hullámforma-vezérlő *(waveform)*	van	van

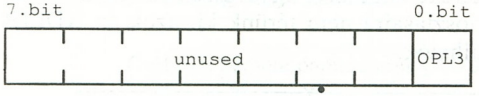
Regiszter $04 ($222): Négyoperátoros mód vezérlő (*4OP control, connection select*)



A regiszter alsó három bitje (az L0–4op, L1–4op és az L2–4op) az első FM egység ($220) első három csatornáját állítja be négyoperátoros módba. Ha az adott bit 1, akkor az adott csatorna — az első fél — a hozzá kapcsolódó második féllel együtt négyoperátoros hanggenerátorként működik. (A 0. csatorna második fele a 3. csatorna, az 1. csatorna második fele a 4. csatorna, a 2. csatorna második fele pedig az 5. csatorna.)

A 3–5. bitek a második FM egység ($222) első három csatornáját állítják be ugyanúgy, ahogyan az első FM egység esetében a 0–2. bitek.

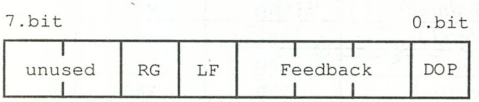
Regiszter $05 ($222): OPL–3/ADLIB-kiválasztás (*Enable OPL–3 mode*)



A regiszter az egyszerű ADLIB kompatíbilis FM mód vagy a bővített OPL–3 FM mód között választ.

OPL3=0	ADLIB YM3812 kompatíbilis üzemmód (mono kétoperátoros)
OPL3=1	OPL–3 bővített üzemmód (sztereó két/négyoperátoros)

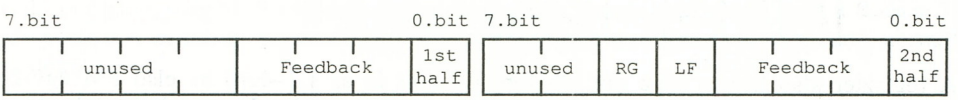
Regiszter $C0..$C8: Visszacsatolás, csatornakiválasztás, összekapcsolás típusa (*Feedback/Left-Right Select/Connection type*)



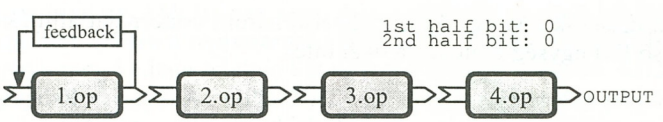
Normál kétoperátoros módban a regiszter bitkiosztása megegyezik a Sound Blaster Pro kártyánál ismertetett kiosztással, négyoperátoros módban viszont van némi eltérés. Ha négy operátort használunk, akkor mindig két regisztert kell megfelelően beállítani:

Négyoperátoros generátor	A generátor első fele	A generátor második fele
0. 4-OP csatorna	$C0	$C3
1. 4-OP csatorna	$C1	$C4
2. 4-OP csatorna	$C2	$C5

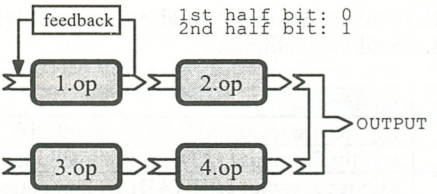
Ilyenkor a két összekapcsolt regiszter bitjeit a következő módon definiálják:



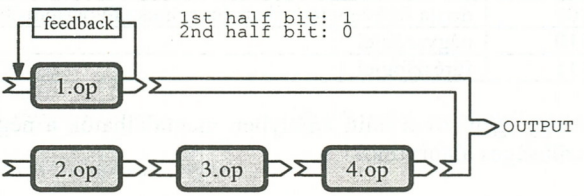
Az első félben a sztereó módot vezérlő bitek ebben az esetben nincsenek kihasználva. A hanggenerátorok összekapcsolási módját a *1st half* és a *2nd half* bitek adják meg. A bitek alapján négy összekapcsolási mód létezik.



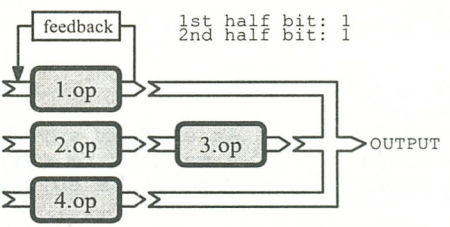
39. ábra. A 0–0 összekapcsolási mód



40. ábra. A 0–1 összekapcsolási mód



41. ábra. Az 1–0 összekapcsolási mód

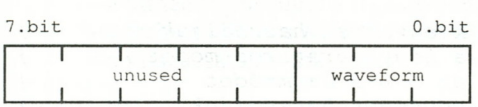


42. ábra. Az 1–1 összekapcsolási mód

Az összekapcsolt csatornák használatakor van néhány módosítás más regiszterekben is, amelyekre figyelni kell:

– A hang kiadását az első fél *KEY ON* bitje szabja meg.
– A visszacsatolás értékét a kártya csak az első fél esetében veszi figyelembe.
– A sztereó csatornákat vezérlő biteket a kártya csak a második félben veszi figyelembe.

Regiszter $E0..$F5: Hullámforma-kiválasztás (*Waveform select*)



Az eredeti YM3812-es hullámkészletet négy újabb alakkal bővítették, így 8 különböző előre definiált hullámalak közül választhatunk:

Bináris kód	Hullámalak
000	egyszerű szinusz (ADLIB kompatíbilis)
001	pozitív félszinusz (ADLIB kompatíbilis)
010	szinusz abszolút érték (ADLIB kompatíbilis)
011	50%-os fázishasított szinusz (ADLIB kompatíbilis)
100	dupla frekvenciájú szinusz félperiódusban
101	dupla frekvenciájú abszolút szinusz félperiódusban
110	négyszögjel
111	fűrészfogjel

Következzen most egy igen rövid unit, amelyben megtalálhatók a négyoperátoros mód bekapcsolásához szükséges alaprutinok!

```
unit opl3; {OPL3.PAS }

interface uses Adlib;

const
        On  = true;
        Off = false;
        FM1stIO  = $220;        { Az első FM egység I/O címe      }
        FM2ndIO  = $222;        { A második FM egység I/O címe    }
        FMbothIO = $228;        { Mindkét FM egység I/O címe      }
var
        OPLmode : boolean;      { Az OPL-3 üzemmód jelzése        }

procedure InitFMchip;                   { OPL-3 mód inicializálás  }
procedure Opl3chip(state:boolean);      { OPL-3 mód engedélyezés   }
procedure Switch4OP(mask:byte);         { Négyoperátoros mód       }

implementation

procedure InitFMchip;
var i : integer;
begin
 OPLmode:=false;
 ADLIBbase:=FM1stIO;     { Az első egységet használjuk          }
 WriteReg($04,$80);      { Timer-inicializálás                  }
 WriteReg($04,$60);
 ADLIBbase:=FM2ndIO;     { A második egységet használjuk        }
 WriteReg($04,$00);      { Tiltjuk a négyoperátoros módot       }
 WriteReg($05,$00);      { Tiltjuk az OPL-3 üzemmódot           }
 ADLIBbase:=FMbothIO;    { Mindkét egységet használjuk          }
 WriteReg($01,$20);      { Engedélyezzük a hullámforma-vezérlést }
 { Az alapvető paramétereket inicializálni kell!                }
```

```
 for i:=0 to 8 do        { 9 db csatorna                    }
  begin
   DoubleOperator($20+cnum[ i],$01); { Frekvenciaszorzó      }
   DoubleOperator($40+cnum[ i],$00); { Hangerő                }
   WriteReg($a0+i,$00);              { Frekvencia alsó bájt   }
   WriteReg($b0+i,$00);              { Frekvencia felső bájt  }
   WriteReg($c0+i,$00);              { Visszacsatolás         }
  end;
end;

procedure Opl3chip(state:boolean);
begin
 ADLIBbase:=FM2ndIO;
 OPLmode:=state;
 WriteReg($05,byte(OPLmode)); { OPL-3 mód engedélyezés/tiltás }
end;
procedure Switch4OP(mask:byte);
begin
 if not OPLmode then exit;
 ADLIBbase:=FM2ndIO;
 WriteReg($04,mask);          { Négyoperátoros mód }
end;

end. of unit
```

Állandók, változók

Az egység az ADLIB unitot használja, ezért minden regisztermódosítás előtt be kell állítani az *ADLIBbase* változót. Erre a célra definiáltuk az *FM1stIO*, *FM2ndIO* és az *FMbothIO* állandókat. Az *OPLmode* logikai változó az FM chip állapotát jelzi. Ha OPL–3 módba kapcsolunk, akkor ez a változó *true* lesz.

Eljárások

InitFMchip: Az FM chip inicializálása. Az eljárás alapállapotba hozza az FM egységeket, kikapcsolja az időzítőket, és egyszerű ADLIB kompatíbilis módot állít be. Ezenkívül a következő regisztereket tölti fel:

- a frekvenciaszorzó értékét 1-re állítja ($20);
- a kimeneti jelszintet a legnagyobbra állítja ($40);
- a frekvenciát és a *KEY ON* bitet nullázza ($A0,$B0);
- a visszacsatolást nullázza.

OPL3chip: A bővített OPL–3 üzemmód be/kikapcsolása. Az FM egységet ezzel az eljárással lehet a megfelelő üzemmódba kapcsolni:

OPL3chip(On) – OPL–3 bővített üzemmód;
OPL3chip(Off) – ADLIB kompatíbilis üzemmód.

Switch4OP: Négyoperátoros csatornák engedélyezése és tiltása. Ezzel az eljárással tudjuk megadni, hogy a két FM egység mely csatornái működjenek négyoperátoros módban. Az eljárás bemenőparamétere egy bitmaszk (*mask*), amelyben a megfelelő bitet 1-be kell állítani ahhoz, hogy az adott egység adott csatornája négyoperátoros legyen. A bitek kiosztása a következő:

Bit	Funkció
0.	első FM egység 0. csatorna (a 0. és a 3. összekapcsolva)
1.	első FM egység 1. csatorna (az 1. és a 4. összekapcsolva)
2.	első FM egység 2. csatorna (a 2. és az 5. összekapcsolva)
3.	második FM egység 0. csatorna (a 0. és a 3. összekapcsolva)
4.	második FM egység 1. csatorna (az 1. és a 4. összekapcsolva)
5.	második FM egység 2. csatorna (a 2. és az 5. összekapcsolva)
6.	nem definiált (redundáns)
7.	nem definiált (redundáns)

6.2.3. Négyoperátoros hang

Az OPL3 unitot felhasználva készítettünk egy rövid példaprogramot, amely az első FM egység 0. csatornáját átkapcsolja négyoperátoros módba, és egy egyszerű hangot ad ki.

```
uses crt,SbPRO,ADLIB,Opl3;

BEGIN
 clrscr;
 InitFMchip;              { Inicializáljuk az FM egységeket   }
 Opl3chip(On);            { OPL-3 módba kapcsolunk            }
 Switch4OP($00);          { Nincs négyoperátoros csatorna     }

 ResetMixer;              { Inicializáljuk a Mixert           }
 SetMasterVolume($ff);    { Legnagyobb kimeneti hangerő       }
 SetFMvolume($ff);        { Legnagyobb MIDI/FM hangerő        }

 { Az első FM egység 0. és 3. csatornáját fogjuk használni  }
 ADLIBbase:=FM1stIO;      { Az első egységet használjuk       }

 { 0. csatorna }
 SetWave($00,$00);        { Hullámforma                       }
 Volume($00,$ff);         { Hangerő                           }
 ADSR($00,$ff03);         { Burkológörbe                      }
 WriteReg($c0,$10);       { A bal oldalon szóljon             }
```

```
{ 3. csatorna }
SetWave($03,$00);       { Hullámforma                          }
Volume($03,$ff);        { Hangerő                              }
ADSR($03,$ff03);        { Burkológörbe                         }
WriteReg($c3,$20);      { A jobb oldalon szóljon               }

{ Először megszólaltatjuk normál módon }
write('A két hang most normál kétoperátoros módban szól... ');
Sound($00,$03,$04);     { A 0. csatorna hangja (bal oldal)     }
delay(1000);
Sound($03,$03,$04);     { A 3. csatorna hangja (jobb oldal)    }
delay(1000);
write('[Enter]');
readln;

{ Most összekapcsoljuk őket egy négyoperátoros hanghoz         }
write('Most négyoperátoros módban szól... ');
Switch4OP($01);         { Összekapcsolás                       }
ADLIBbase:=FM1stIO;
WriteReg($c0,$00);
WriteReg($c3,$30);      { Mindkét oldalon szólni fog           }
Sound($00,$03,$04);     { Ez a négyoperátoros hang             }
delay(1500);
write('[Enter]');
readln;
InitFMchip;
END.
```

Az FM chip inicializálása után egy hosszan lecsengő hanghoz beprogramozzuk a 0. és a 3. csatornát. A 0. csatorna hangját csak a bal, a 3. csatorna hangját pedig csak a jobb oldalon szólaltatjuk meg. Ekkor a hang még kétoperátoros. A két hang megszólalása után az Enter billentyűt kell lenyomni. Az Enter lenyomása után összekapcsoljuk a két csatornát egyetlen négyoperátoros hang előállításához. Az összekapcsolás típusát a $C0 és a $C3 regiszter legalsó bitjei adják. Jelen esetben mindkettő nulla, tehát a hanggenerátorok a 0-0 összekapcsolási séma szerint kapcsolódnak egymáshoz. A hang teljesen más lesz, mint az előbb, és — amint azt látható — csupán a 0. csatorna *KEY ON* bitjét kapcsoljuk be (csak egyetlen *Sound* utasítás van), mégis minden generátor megszólal.

Sem ez a program, sem az OPL3 unit nem tökéletes, igazából nem is használhatók komoly feladatra. (Az FM hanggenerálást az ADLIB kártyánál már részletesen ismertettük, ezért itt ezzel már nem foglalkozunk.) A példa alapján látható a négyoperátoros mód használata, a szükséges beállítások és szabályok. A további kísérletezgetést az Olvasóra bízzuk. **:–o**

6.3. A továbbfejlesztett DSP egység (*ASP*)

I/O relatív cím	I/O alapbeállítású cím	DSP funkció
Base+$06	$226 *(write only)*	A DSP egység alapállapotba hozása *(DSP Reset)*
Base+$0A	$22A *(read only)*	Adat olvasása az ADC-ről *(Read Data)*
Base+$0C	$22C *(read/write)*	Parancs/adat írása a DAC-ra *(Write command/data)* Bufferállapot olvasása *(Read Buffer Status)*
Base+$0E	$22E *(read only)*	Adatérvényesség olvasása *(Data Available, 8 bit IRQ acknowledge)*
Base+$0F	$22F *(read only)*	A 16 bites megszakítás nyugtázása *(16 bit IRQ acknowledge)*

A kártyán az FM chip mellett a DSP egységet is továbbfejlesztették (*Advanced Sound Processor*), így ez is sokkal többet tud, mint elődei. A felvétel és a lejátszás esetén kiválasztható a minták hosszúsága (8 vagy 16 bit), az adatátvitel módja (sztereó vagy mono) és a minták típusa (előjeles vagy előjel nélküli). Ezenkívül szabadon programozható a mintavételi frekvencia 5 kHz-től egészen 48 kHz-ig. Természetesen mind a felvétel, mind a lejátszás DMA átvitellel valósul meg. A kártya két DMA csatornát tud használni. Az alacsony DMA-t (*Low DMA channel*) 8-bites átvitelre, a magas DMA-t (*High DMA channel*) pedig 16 bites átvitelre használhatjuk.

A DSP egység — hasonlóan a Sound Blasterhez — több I/O címen érhető el. Ezekben nincs sok változtatás, csupán a 16 bites adatátvitel által kért megszakítás nyugtázásához vezettek be egy újabb portot. Az inicializálás, valamint az olvasás és az írás ugyanúgy történik, ahogy a Sound Blaster kártyánál. (Javasoljuk emlékeztetőül még egyszer elolvasni a könyvnek ezt a részét!)

6.3.1. DSP parancsok

Az előző kártyák tárgyalásakor már jó néhány DSP parancsot ismertettünk, így azokat már itt nem ismételjük meg, csupán a teljesen új lehetőségeket mutatjuk be. A kártya természetesen ismeri a korábbi DSP műveleteket, de szerintünk — és talán a gyártók szerint is — érdemesebb a saját SB 16 parancsokat használni.

Kód	Parancs	Parancsfunkció
$D0	*Pause 8 bit DMA*	a 8-bites DMA átvitel felfüggesztése
$D4	*Continue 8 bit DMA*	a 8-bites DMA átvitel folytatása
$D5	*Pause 16 bit DMA*	a 16 bites DMA átvitel felfüggesztése
$D6	*Continue 16 bit DMA*	a 16 bites DMA átvitel folytatása
$D9	*Exit 16 bit auto-init DMA*	a 16 bites *auto-init* típusú DMA leállítása
$DA	*Exit 8 bit auto-init DMA*	a 8-bites *auto-init* típusú DMA leállítása
$E1	*Get DSP version*	a DSP verziószámának lekérdezése
$41	*Set play sample rate*	mintavételi frekvencia beállítása lejátszáshoz
$42	*Set record sample rate*	mintavételi frekvencia beállítása felvételhez
$B6	*16 bit auto-init DMA play*	16 bites *auto-init* típusú DMA lejátszás
$BE	*16 bit auto-init DMA record*	16 bites *auto-init* típusú DMA felvétel
$B2	*16 bit normal DMA play*	16 bites *single-cycle* típusú DMA lejátszás
$BA	*16 bit normal DMA record*	16 bites *single-cycle* típusú DMA felvétel
$C6	*8 bit auto-init DMA play*	8-bites *auto-init* típusú DMA lejátszás
$CE	*8 bit auto-initDMA record*	8-bites *auto-init* típusú DMA felvétel
$C2	*8 bit normal DMA play*	8-bites *single-cycle* típusú DMA lejátszás
$CA	*8 bit normal DMA record*	8-bites *single-cycle* típusú DMA felvétel

Bizonyos DSP parancsok nemcsak egyetlen bájtból állnak, hanem egységes bájtszekvenciákat — bájtsorozatokat — kérnek. Az ilyen parancsok kiadásánál ezért vigyázni kell, hogy minden egyes bájt kikerüljön a DSP-re, máskülönben a parancs hatástalan lesz, vagy hibás működést eredményez.

6.3.2. Bájtszekvenciális parancsok

$41: A mintavételi frekvencia beállítása lejátszáshoz (*Set Play Sample Rate*).

0. bájt:	1. bájt:	2. bájt:
$41 (SPSR)	SR High	SR Low

$42: A mintavételi frekvencia beállítása felvételhez (*Set Record Sample Rate*).

0. bájt:	1. bájt:	2. bájt:
$42 (SRSR)	SR High	SR Low

A Sound Blaster 16 a mintavételi frekvencia beállításánál sokkal nagyobb szabadságot enged, mint az előző kártyák. Itt ugyanis közvetlenül egy 16 bites word adattal kell megadni a mintavételi frekvenciát. (Az előző kártyák csak egy 8-bites állandó értéket engedtek meg, amely arányos volt a frekvenciával.) A felvételhez és a lejátszáshoz egyaránt három bájtot kell kiküldeni a DSP-re: először a parancs kódját ($41/$42), majd a 16 bites

állandót. Figyeljünk arra, hogy ebben az esetben a felső bájtot kell először kiküldeni, majd utána az alsót!

Felvétel és lejátszás: A DMA átvitellel történő felvételhez és lejátszáshoz minden esetben négy bájtot kell kiküldeni a DSP-re a következő formában:

0.bájt:	1. bájt:	2. bájt:	3. bájt:
T-mode	M/S-mode	Size Low	Size High

A *T-mode* (*Transfer mode*) bájt adja meg az átvitel irányát és típusát. A fenti DSP parancsokat tartalmazó táblázat egyes parancsait ez a bájt határozza meg. A bitek kiosztása a következő:

7.bit							0.bit
1	0	1	1	P/R	A/S	fifo	0

43. ábra. A T-mode bájt bitkiosztása

A felső négy bit meghatározza az adatszélességet. Az ábrán a felső négy bit 1011, azaz $B; ez a 16 bites átvitel. Ha 8-bites átvitelt akarunk megvalósítani, akkor ezt a bitmezőt 1100-ra, azaz $C-re kell állítani.

P/R: Az adatátvitel irányának kiválasztása (*Play/Record select*)
P/R = 0 Play (*DAC, MEM to IO mode*) lejátszás
P/R = 1 Record (*ADC, IO to MEM mode*) felvétel

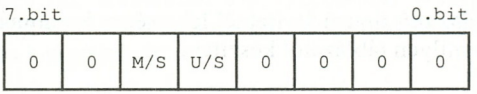
A/S: A DMA átvitel típusának kiválasztása (*DMA transfer type select*)
A/S = 0 Normál DMA átvitel (*Single-cycle DMA*)
A/S = 1 Automatikus inicializálásos DMA átvitel (*Auto-init DMA*)

fifo: Belső adatfeldolgozás (First In First Out type). Ezt a bitet mindig 1-be kell állítani!

Ezeknek a biteknek a megfelelő beállításával tehát tetszőleges átvitelt lehet megadni. Nézzük meg példaképpen a 16 bites *auto-init* típusú DMA átvitellel történő lejátszás parancsának kódolását:

Az átvitel 16 bites, ezért a felső négy bit 1011, azaz $B. Mivel lejátszásról van szó, ezért a P/R bit 0. Az átvitel *auto-init* típusú, ezért az A/S bit 1. A fifo bitet mindig 1-re állítjuk, így a kódolt érték: 10110110, tehát $B6 (lásd az előző táblázatot).

A következő bájt a sorozatban az *M/S-mode*. Ezzel a bájttal kell megadni, hogy az átvitel sztereó legyen-e vagy mono és azt is, hogy milyen adatokkal dolgozunk. Nyolcbites esetben a kártya előjelesként értelmezi a hangmintákat, míg tizenhat bites esetben nincs előjel (ez általánosan elfogadott szabály).



44. ábra. Az M/S-mode bájt bitkiosztása

M/S: Mono/sztereó választás (*Mono/Stereo select*)
M/S = 0 Mono üzemmód (*Mono mode*)
M/S = 1 Sztereó üzemmód (*Stereo mode*)

U/S: Az adattípus kiválasztása (*Unsigned/Signed data type select*)
U/S = 0 Előjel nélküli, ezt a 16 bites átvitelhez alkalmazzuk (*Unsigned datas*)
U/S = 1 Előjeles, ezt a 8-bites átvitelhez használjuk (*Signed datas*)

8/16 bit	Felvétel / Lejátszás	Mono / Sztereó	Normál / Előjeles	T-mode bájt	M/S-mode bájt
8 bit	felvétel	mono	normál	nincs	nincs
8 bit	felvétel	mono	előjeles	$CE	$10
8 bit	felvétel	sztereó	normál	nincs	nincs
8 bit	felvétel	sztereó	előjeles	$CE	$30
8 bit	lejátszás	mono	normál	nincs	nincs
8 bit	lejátszás	mono	előjeles	$C6	$10
8 bit	lejátszás	sztereó	normál	nincs	nincs
8 bit	lejátszás	sztereó	előjeles	$C6	$30
16 bit	felvétel	mono	normál	$BE	$00
16 bit	felvétel	mono	előjeles	nincs	nincs
16 bit	felvétel	sztereó	normál	$BE	$20
16 bit	felvétel	sztereó	előjeles	nincs	nincs
16 bit	lejátszás	mono	normál	$B6	$00
16 bit	lejátszás	mono	előjeles	nincs	nincs
16 bit	lejátszás	sztereó	normál	$B6	$20
16 bit	lejátszás	sztereó	előjeles	nincs	nincs

A két bit négyféle kombinációt eredményez, de ezeket nem kombinálhatjuk szabadon az átvitel bitszélességével (kompatibilitási gondok léphetnek fel), ezért nem lehet minden lehetőséget kihasználni. Az előző táblázat a variációkat mutatja, de csak az *auto-init* típusú átvitelhez.

A táblázat **nincs** mezői azokat a kombinációkat mutatják, amelyeket nem ajánlatos használni. A Sound Blaster 16 tudja ezeket a kombinációkat, de más kártyák esetében ez egyáltalán nem biztos, ezért csak a szabványos variációkat ajánlatos használni. Ha például 8-bites előjel nélküli felvételt készítünk, akkor azt a normál Sound Blaster nem tudja lejátszani!

Ezek tehát csupán az *auto-init* típusú átvitelnél lehetséges kombinációk, de a *single-cycle* típushoz is lehetne ugyanilyen táblázatot készíteni.

Size Low, Size High: A bájtsorozat utolsó két bájtja az átvitel hosszát adja meg. Itt először mindig az alsó bájtot, majd utána a felső bájtot kell kiküldeni, tehát éppen fordítva, mint a mintavételi frekvencia megadásánál. :–(.

6.4. Egyszerű egyciklusos DMA átvitel

Normál DMA átvitelnél — ahogyan a Sound Blaster kártyánál — csak néhány egyszerű beállítást kell elvégezni a felvételnél és a lejátszásnál is:

– Foglaljunk le egy buffert, amely lehetőleg pontosan DMA laphatárra esik, és az sem baj, ha minél hosszabb (max. 64K)!
– Készítsük el a megfelelő megszakítási kiszolgáló rutint, és programozzuk be a megszakításvezérlőt!
– Állítsuk be a DMA vezérlőt a *single-cycle* típusú átvitelre!
– A megfelelő bájtszekvenciális DSP paranccsal állítsuk be a mintavételi frekvenciát!
– A megfelelő bájtszekvenciális DSP paranccsal (felvétel/lejátszás) indítsuk el az átvitelt!

A megszakítási rutinban szintén van néhány fontos tennivalónk:

– Állítsuk be a DMA vezérlőt a következő átvitelre!
– Ugyanígy programozzuk a DSP-t is, hogy az átvitel elinduljon!
– Nyugtázzuk a megszakításkérést a $22E (8-bites esetben) vagy a $22F (16 bites esetben) I/O port beolvasásával!
– Jelezzük a megszakításvezérlőnek, hogy a megszakítást kezeltük a $20-as portra kiírt $20-as értékkel (vagy a $A0 portra kiírt $20-as értékkel, ha az IRQ8...IRQ15 megszakítási vonalakat használjuk)!

(A megszakítási vonal és a DMA csatorna számának detektálásáról, illetve a beállításukról majd a Mixer programozásánál lesz szó részletesebben.)

6.5. DMA átvitel automatikus inicializálással

Az előzőekben többször is említettük az *auto-init* típusú DMA átvitelt. Ezt a fajta átvitelt a DMA vezérlő ismeri, de a Sound Blaster családból csak a Sound Blaster 16 — és a későbbi típusok — tudják kezelni. Az előző kártyáknál éppen ezért nem is említettük az

átvitel típusát, azt mindenhol normálnak, azaz egyciklusú —*single-cycle* — átvitelnek értelmeztük. Nézzük meg most tehát az *auto-init* típusú átvitelt és azt, hogy miért is jó ez a hangok felvételénél és lejátszásánál!

Amikor egy DMA átvitel befejeződik, a DSP egy megszakítást generál. A megszakításnak kell újraprogramoznia a DMA vezérlőt és a hangkártyát az újabb átvitelre. Akármilyen gyors is a gép, ez jelentős időt elvesz. Ilyenkor nincs átvitel, tehát lejátszásnál a kimenet a megszakítás — az újraprogramozás — ideje alatt nem szól, ezért a hangban apró kattanás hallható. Felvételnél viszont ez alatt az idő alatt nincs hangdigitalizálás, azaz a hangból értékes minták vesznek el. Mindkettő igen kellemetlen, káros jelenség.

Az automatikus inicializáló módban a DMA vezérlő az átvitel végén — tehát amikor a csatornaszámláló regisztere nullára csökken — a kezdéskor beírt értékekkel tölti fel ismét a regisztereket, azaz automatikusan alapállapotba hozza magát. Az inicializálás után természetesen az átvitel is folytatódik. A vezérlő mindezt hardverből oldja meg, azaz jóval gyorsabban, mintsem azt szoftverből megtehetnénk. Ezért sem lejátszáskor nem hallatszik a kimeneten a zavar, sem a felvételnél nem maradnak ki hangminták. Mivel a hangkártya tudja, hogy *auto-init* típusú átvitel van folyamatban (a *T-mode* bájt *A/S* bitje 1-be van állítva), ezért a mintavételi frekvencia által meghatározott időkben továbbra is DMA átvitel történik. (Ezt azért tartjuk fontosnak megemlíteni, mert sokszor hibásan értelmezik a DMA átvitelt. A DMA vezérlő nem kér átvitelt, azt minden esetben a hangkártyának kell megtennie a megfelelő DRQ láb aktívvá tételével. Ezért az ilyen átvitelhez nem elég a vezérlőt *auto-init* módba programozni, a DMA-t kezdeményező áramkörnek — ami esetünkben a hangkártya — is tudnia kell ilyen módon dolgozni.)

Persze joggal kérdezheti valaki, hogy mit ér ez az egész — mármint hogy az átvitel továbbra is folytatódik —, hiszen így a már felvett mintára veszünk fel ismét más mintákat, illetve a már lejátszott hangot kezdjük el újra lejátszani. Nos, nem teljesen ez a helyzet. Az *auto-init* típusú átvitelhez ugyanis osztott buffer technikát — kettős buffer technikát — alkalmazunk. A módszer lényege a következő: a DMA vezérlő számára lefoglalunk egy teljes buffert (azaz egy 64K méretű DMA laphatárra eső memóriaterületet), de a hangkártya számára csak feleakkora adatátviteli hosszúságot adunk meg. Mindkét eszközt *auto-init* típusú átvitelre programozzuk. Amikor a belső számlálója lejár — tehát amikor 32K adat átvitele már megtörtént —, a hangkártya egy megszakítást kér, ugyanakkor a DMA folytatódik tovább, hiszen a vezérlő szerint még ugyanennyi bájtot át kell vinni, a hangkártya pedig tudja, hogy az átvitel *auto-init* típusú, tehát a megszakítás után is folytatni kell. A buffer első fele tehát feltöltődött felvétel esetén, illetve lejátszódott lejátszás esetén. Ezt a területet tehát el lehet kezdeni frissíteni, tárolni, vagy új mintákat betölteni. A megszakítás alatt ez zavartalanul folyhat, hiszen az átvitel ez alatt nem szünetel. Ugyanígy a buffer második felének végén is kér egy megszakítást a kártya, de ilyenkor a DMA vezérlő az automatikus inicializálás miatt a legelején kezdi az újabb átvitelt. Az egész folyamat lényege tehát az, hogy két megszakítás érkezik — az egyik az átvitel „félidejében", a másik pedig az átvitel végén —, de mindkét esetben már a fél buffer frissíthető; a DMA átvitel pedig nem szakad meg.

A programozást hasonlóan kell elvégezni, mint a *single-cycle* típusú átvitelnél, csak mind a DMA vezérlőt, mind a hangkártyát *auto-init* módba kell kapcsolni, és a hangkártyának a buffer felét kell megadni hosszként. A megszakítást kiszolgáló rutinnak ilyenkor más teendői vannak:

– Frissítsük a buffert a következő átvitelhez (tároljuk a tartalmát felvételkor, illetve töltsük fel az újabb hangrészlettel lejátszáskor)!
– Nyugtázzuk a megszakításkérést a $22E (8-bites esetben) vagy a $22F (16 bites esetben) I/O port beolvasásával!
– Jelezzük a megszakításvezérlőnek, hogy a megszakítást kezeltük a $20-as portra kiírt $20-as értékkel (vagy a $A0 portra kiírt $20-as értékkel, ha az IRQ8...IRQ15 megszakítási vonalakat használjuk)!

(A megszakítási vonal és a DMA csatorna számának detektálásáról, illetve a beállításukról majd a Mixer programozásánál lesz szó részletesebben.)

Ha az átvitel befejeződött, akkor le kell állítani. Ezt kétféleképpen lehet megtenni: közvetlenül, vagy a legutolsó blokk végén automatikusan. Közvetlenül a *Pause 8 bit DMA ($D0)*, illetve a *Pause 16 bit DMA ($D5)*, DSP parancsokkal lehet megállítani az átvitelt. Ilyenkor nincs megszakítás, az átvitel befejeződik. Ha a legutolsó blokkot még kezelni kell, akkor az *Exit 8 bit auto-init DMA ($D9)* és az *Exit 16 bit auto-init DMA ($DA)* DSP parancsokat kell használnunk. Ilyenkor a legutolsó megszakítás elfogadásakor ér véget az átvitel.

6.6. A keverő programozása

A Sound Blaster 16 Mixer áramköre — néhány apró eltérés kivételével — kompatíbilis maradt a Sound Blaster Pro keverőjével, de jó néhány újítást is találunk rajta. Az új Mixer nem 4 bitet használ a hangerő-szabályozáshoz, hanem minden audioforrás mindkét oldalához egy-egy külön regisztert. A hangszín szabályozásához négy új regisztert vezettek be, kettőt a magas szint beállításához (*Treble left, Treble right*) és kettőt a mély szint beállításához (*Bass left, Bass right*). Minden sztereó forráseszköz két oldala szabadon ki/be kapcsolható, illetve a felvételnél kiválasztható, hogy a felvett hangmintában melyik oldalra kerüljön. A PC-Speaker hangerejének szabályozására egy új regisztert vezettek be. A kimeneti erősítési szint négy lépcsőben adható meg. A mikrofonbemenethez beépítettek egy automatikus erősítésszabályozó áramkört (*Automatic Gain Control*). Sok tehát az újdonság, és még több a lehetőség, ahogyan a Mixert használni lehet.

A Mixert a Sound Blaster Próhoz hasonlóan itt is a *Base+$04*, *Base+$05* címeken lehet elérni:

Regiszter I/O cím	Alapbeállítású I/O cím	Funkció
Base+$04	$224 (w)	Mixer címregiszter (*address*)
Base+$05	$225 (r/w)	Mixer adatregiszter (*data*)

A belső regiszterek a $00...$2F tartományban a Sound Blaster Pro regisztereivel egyeznek meg. A $0E regiszter *VSTC* bitje itt nem működik, mert a sztereó/mono átváltást a DSP megfelelő programozásával lehet megtenni. Az egyéb regiszterek általában működnek, bár nem javasoljuk a használatukat. Amikor csak lehet, célszerű inkább az új SB 16 Mixer regisztereket használni. A regisztereket az alábbi táblázatban foglaltuk össze:

Regiszter	Regiszterfunkció
$30 *Master Volume Left*	kimeneti hangerő, bal oldal
$31 *Master Volume Right*	kimeneti hangerő, jobb oldal
$32 *DSP Voice Volume Left*	DSP hangerő, bal oldal
$33 *DSP Voice Volume Right*	DSP hangerő, jobb oldal
$34 *MIDI/FM Volume Left*	MIDI/FM egység hangerő, bal oldal
$35 *MIDI/FM Volume Right*	MIDI/FM egység hangerő, jobb oldal
$36 *CD Volume Left*	audio-CD hangerő, bal oldal
$37 *CD Volume Right*	audio-CD hangerő, jobb oldal
$38 *Line In Volume Left*	vonali bemenet hangerő, bal oldal
$39 *Line In Volume Right*	vonali bemenet hangerő, jobb oldal
$3A *Microphone Volume*	mikrofon hangerő
$3B *PC-Speaker Volume*	PC-Speaker hangerő ;–)
$3C *Output Select*	a kimeneti eszközök kiválasztása
$3D *Left Input Device Select*	a bal oldali bemeneti eszköz kiválasztása
$3E *Right Input Device Select*	a jobb oldali bemeneti eszköz kiválasztása
$3F *Left Input Gain Level*	bal oldali bemeneti erősítési szint
$40 *Right Input Gain Level*	jobb oldali bemeneti erősítési szint
$41 *Left Output Gain Level*	bal oldali kimeneti erősítési szint
$42 *Right Output Gain Level*	jobb oldali kimeneti erősítési szint
$43 *Microphone AGC Control*	mikrofon AGC engedélyezés/tiltás
$44 *Left Treble Level*	bal oldali magas szint szabályozás
$45 *Right Treble Level*	jobb oldali magas szint szabályozás
$46 *Left Bass Level*	bal oldali mély szint szabályozás
$47 *Right Bass Level*	jobb oldali mély szint szabályozás
$80 *Interrupt Level Select*	a megszakítási vonal kiválasztása
$81 *DMA Channels Select*	a DMA csatornák kiválasztása (8,16 bit)
$82 *Interrupt Request Status*	megszakításkérés állapot

Regiszter $30...$3B: Hangerő-szabályozás

A hangerőket szabályozó regiszterek 8-bitesek ugyan, de tapasztalataink szerint nem lehet 256 lépésben szabályozni a hangerőt. A kártya bizonyára figyelmen kívül hagyja a regiszterek alsó bitjeit (az alsó három bitet). A programozás során ezzel nem kell törődnünk, nyugodtan használhatjuk őket normál 8-bites regiszterekként.

Regiszter $3C: A kimeneti eszközök kiválasztása (*Output Select*)

7.bit							0.bit
unused			Line Lf	Line Rg	CD Lf	CD Rg	Mic

A regiszter a kimeneten megszólaló eszközöket választja ki.
Mic: A mikrofon erősítésének engedélyezése/tiltása.
CD Rg: Az audio-CD jobb oldalának engedélyezése/tiltása.
CD Lf: Az audio-CD bal oldalának engedélyezése/tiltása.
Line Rg: A jobb oldali vonali bemenet erősítésének engedélyezése/tiltása.
Line Lf: A bal oldali vonali bemenet erősítésének engedélyezése/tiltása.
Az állapotok minden bit esetében: 0 – tiltás, 1– engedélyezés.

Regiszter $3D: A bal oldali bemeneti eszközök kiválasztása (*Left Input Select*)

7.bit							0.bit
0	MIDI Lf	MIDI Rg	Line Lf	Line Rg	CD Lf	CD Rg	Mic

A regiszter a bal oldali bemeneti eszközt (eszközöket) választja ki a felvételhez. Amint látható, minden eszköz tetszőleges oldala kiválasztható, és természetesen egyszerre több eszköz is lehet a felvétel forrása.
Mic: A mikrofon engedélyezése/tiltása.
CD Rg: Az audio-CD jobb oldalának engedélyezése/tiltása.
CD Lf: Az audio-CD bal oldalának engedélyezése/tiltása.
Line Rg: A jobb oldali vonali bemenet engedélyezése/tiltása.
Line Lf: A bal oldali vonali bemenet engedélyezése/tiltása.
MIDI Rg: A MIDI/FM egység jobb oldalának engedélyezése/tiltása.
MIDI Lf: A MIDI/FM egység bal oldalának engedélyezése/tiltása.
Az állapotok minden bit esetében: 0 – tiltás, 1– engedélyezés.

Regiszter $3E: A jobb oldali bemeneti eszközök kiválasztása (*Right Input Select*)

7.bit							0.bit
0	MIDI Lf	MIDI Rg	Line Lf	Line Rg	CD Lf	CD Rg	Mic

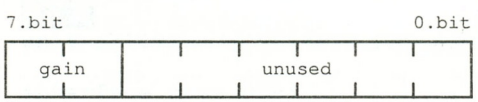
A regiszter a jobb oldali bemeneti eszközt (eszközöket) választja ki a felvételhez.
Mic: A mikrofon engedélyezése/tiltása.
CD Rg: Az audio-CD jobb oldalának engedélyezése/tiltása.
CD Lf: Az audio-CD bal oldalának engedélyezése/tiltása.
Line Rg: A jobb oldali vonali bemenet engedélyezése/tiltása.
Line Lf: A bal oldali vonali bemenet engedélyezése/tiltása.
MIDI Rg: A MIDI/FM egység jobb oldalának engedélyezése/tiltása.
MIDI Lf: A MIDI/FM egység bal oldalának engedélyezése/tiltása.
Az állapotok minden bit esetében: 0 – tiltás, 1– engedélyezés.

Regiszter $3F: A bal oldali bemeneti erősítési szint beállítása (*Left Output Gain Level*)

Regiszter $40: A jobb oldali bemeneti erősítési szint beállítása (*Right Output Gain Level*)

Regiszter $41: A bal oldali kimeneti erősítési szint beállítása (*Left Input Gain Level*)

Regiszter $42: A jobb oldali kimeneti erősítési szint beállítása (*Right Input Gain Level*)



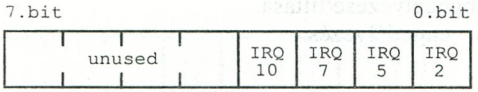
Mind a négy erősítésszabályozó regiszter azonos felépítésű: a felső két bit adja meg az erősítést, pontosabban mind a négy állapothoz tartozik egy előre definiált erősítési szint:

00 – egyszeres normál erősítés	(*1x gain, $00*)
01 – kétszeres erősítés	(*2x gain, $40*)
10 – négyszeres erősítés	(*4x gain, $80*)
11 – nyolcszoros erősítés	(*8x gain, $c0*)

Regiszter $43: Mikrofon FGC/AGC kiválasztás (*Fix/Automatic Gain Control*). Ezzel a regiszterrel lehet beállítani a mikrofon erősítésének vezérlését.
00h – fix erősítés (*AGC off*)
01h – automatikus erősítésszabályozás (*AGC on*)

Regiszter $44..$47: Magas és mély szabályozás. A magas és mély erőssége a hangerőkhöz hasonlóan egy 8-bites értékkel adható meg, de itt sincs kihasználva a regiszterek néhány alsó bitje.

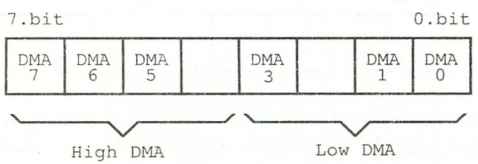
Regiszter $80: A megszakítási vonal kiválasztása (*Interrupt Level Select*)



A regiszter alsó négy bitje a megfelelő megszakítási szintet állítja be. A DSP és a MIDI interfész ezen a vonalon fog megszakítást kérni. Figyelni kell arra, hogy mindig csak egyetlen bit legyen 1-ben!

IRQ 10: a 10-es megszakítási vonal (IRQ 10, INT $72)
IRQ 7: a 7-es megszakítási vonal (IRQ 7, INT $0F)
IRQ 5: az 5-ös megszakítási vonal (IRQ 5, INT $0D)
(általában ez az alapbeállítás!)
IRQ 2: a 2-es megszakítási vonal (IRQ 2, INT $0A)

Regiszter $81: A DMA csatornák kiválasztása (*DMA Channels Select*)



Ezzel a regiszterrel a két DMA csatornát kell beállítani. Az alsó négy bit a 8-bites csatorna számát, a felső négy bit pedig a 16 bites csatorna számát adja meg. Ha a felső négy bit 0000, akkor nincs 16 bites átvitel, pontosabban ez is a 8-bites csatornán zajlik.

DMA 7: a 16 bites aktív DMA csatorna a 7-es.
DMA 6: a 16 bites aktív DMA csatorna a 6-os.
DMA 5: a 16 bites aktív DMA csatorna az 5-ös.
DMA 3: a 8-bites aktív DMA csatorna a 3-as.
DMA 1: a 8-bites aktív DMA csatorna az 1-es (általában ez az alapbeállítás).
DMA 0: a 8-bites aktív DMA csatorna a 0-s.
Itt is figyeljünk arra, hogy egyszerre mindkét félben csupán egyetlen bitet lehet 1-be állítani!

Regiszter $82: Megszakításkérés állapot (*Interrupt Request Status*)

7.bit							0.bit
1	0	0	0	?	UART	DSP 16	DSP 8

Ezt a regisztert csak olvasni lehet. A hangkártya megszakítását három esemény válthatja ki:

- a 8-bites DSP átvitel befejezése (DMA felvételnél vagy lejátszásnál);
- a 16 bites DSP átvitel befejezése (DMA felvételnél vagy lejátszásnál);
- a MIDI (UART) interfész egy eseménye.

A regiszter alsó három bitje azt adja meg, hogy melyik eszköz okozta a megszakítást. Ezt az értéket a megszakítási rutinban közvetlenül a megszakítás elfogadása után kell kiolvasni, hiszen az egyes típusokhoz más-más kezelési rutinok tartozhatnak.

6.6.1. Az SB16MIX unit

A korábbi fejezetekhez hasonlóan ehhez a kártyához is készítettünk egy rövid — ám annál hasznosabb — unitot, amelyben a kezeléshez szükséges alapvető eljárások, valamint az előre definiált (később jól felhasználható) állandók és változók vannak. Következzen tehát a Sound Blaster 16-hoz is egy ilyen rövid unit, amelynek segítségével könnyen lehet programozni a kártyát.

```
unit sb16mix; { SB16MIX.PAS }

interface uses DSP;

const
  { Hardver alapbeállítások }
  Base      : word = $220; { I/O báziscím              }
  IRQnumber : byte = $05;  { Megszakítási vonal        }
  LowDMAch  : byte = $01;  { 8-bites DMA csatornaszám  }
  HighDMAch : byte = $05;  { 16 bites DMA csatornaszám }

  On    = true;        { Bekapcsolás (AGC) }
  Off   = false;       { Kikapcsolás (AGC) }

  { Az analóg eszközöket definiáló konstansok (dmXXXX) }
  dmMic       = $01; { Mikrofon              }
  dmCDright   = $02; { Audio-CD bal oldal    }
  dmCDleft    = $04; { Audio-CD jobb oldal   }
  dmLineLeft  = $08; { Vonali bemenet bal oldal  }
  dmLineRight = $10; { Vonali bemenet jobb oldal }
  dmMIDIleft  = $20; { FM/MIDI bal oldal         }
```

```
dmMIDIright = $40; { FM/MIDI jobb oldal              }
dmOutputAll = $1F; { Minden kierősített eszköz       }
dmInputAll  = $7F; { Minden bemeneti eszköz          }
dmCD        = $06; { Audio-CD mindkét oldal          }
dmLine      = $18; { Vonali bemenet mindkét oldal    }
dmMIDI      = $60; { FM/MIDI mindkét oldal           }
dmLeftAll   = $55; { Minden bal oldali eszköz        }
dmRightAll  = $2A; { Minden jobb oldali eszköz       }

Gain_1x = $00; { Normál egyszeres erősítés }
Gain_2x = $40; { Kétszeres erősítés        }
Gain_4x = $80; { Négyszeres erősítés       }
Gain_8x = $C0; { Nyolcszoros erősítés      }

{ A megszakítási vonalat definiáló állandók (imXXXX) }
imIRQ10 = $08; { IRQ 10, INT $72 }
imIRQ7  = $04; { IRQ  7, INT $0F }
imIRQ5  = $02; { IRQ  5, INT $0D }
imIRQ2  = $01; { IRQ  2, INT $0A }
{ A DMA csatornákat definiáló állandók (dcXXXX)      }
dcDMA0    = $01; { DMA 0 }
dcDMA1    = $02; { DMA 1 }
dcDMA3    = $08; { DMA 3 }
dcDMA5    = $20; { DMA 5 }
dcDMA6    = $40; { DMA 6 }
dcDMA7    = $80; { DMA 7 }
dcDMAlow  = $00; { Low   }

{ DSP definíciók (dspXXXX)                           }
{ T-mode bájt }
dspMode8bit     = $C2; { 8-bites átvitel          }
dspMode16bit    = $B2; { 16 bites átvitel         }
dspRecord       = $08; { Felvétel                 }
dspPlay         = $00; { Lejátszás                }
dspAutoInit     = $04; { Auto-init típusú DMA     }
dspSingleCycle  = $00; { Single-cycle típusú DMA  }
{ M/S-mode bájt }
dspStereo       = $20; { Sztereó üzemmód          }
dspMono         = $00; { Mono üzemmód             }
dspSigned       = $10; { Előjeles adatok          }
dspUnsigned     = $00; { Előjel nélküli adatok    }
```

Az itt definiált állandók a beállításokat végző eljárásokhoz használhatók, így nem kell mindig elvégezni a fáradságos bitmaszkolásokat és számításokat. Minden fontosabb eljáráshoz előre megadtuk a beállítható paramétereket. Ezek általában egyszerű bitmaszkok, így a logikai kapcsolatokkal könnyedén megadhatók megfelelő kombinációk. Ha például a bemeneti eszközöket akarjuk beállítani, akkor azt a következő egyszerű módon tehetjük meg:

```
InputSelect(dmLineLeft or dmCDleft, dmLineRight or dmCDright);
```

Ez a bemeneti eszközök közül engedélyezi mindkét oldalra a vonali bemenetet és a CD-t. Ugyanígy használhatjuk a többi állandót is. A Turbo Vision példájára mi is az első néhány betűvel jeleztük, hogy mely állandó mely beállításban szerepelhet. Így a program jól olvasható és áttekinthető lesz (sokkal inkább, mintha egyszerű hexadecimális paramétereket adnánk meg).

Csoport	Funkció
dmXXXX (*Device Mask*)	bemeneti és kimeneti eszközök beállítása
dcXXXX (*DMA Channel*)	DMA csatornák beállítása
imXXXX (*Interrupt Mask*)	a megszakítási vonal beállítása
dspXXXX (*DSP Command*)	DSP parancsok összeállítása

```
{ ******** Alacsony szintű rutinok ******** }
procedure ResetMixer;                        { Mixer inicializálás      }
procedure SetMixer(RegNum,Value:byte);       { Mixer regiszter írás     }
function  GetMixer(Regnum:byte):byte;        { Mixer regiszter olvasás  }
```

Ezeket az eljárásokat akkor használhatjuk, amikor valamely Mixer regisztert közvetlenül akarjuk elérni. A *ResetMixer* inicializálja a Mixert, a *SetMixer*-rel egy adott regisztert lehet beállítani, míg a *GetMixer* segítségével egy regiszter értékét kérdezhetjük le. Az ilyen közvetlen elérésnél figyelni kell az egyes regiszterek beállításának szabályaira (például a megszakítási szint beállítására).

```
{ ****** Hangerő-szabályozó eljárások ****** }
procedure SetMasterVolume(Left,Right:byte); { Kimeneti hangerő }
procedure SetDSPVolume(Left,Right:byte);    { DSP hangerő      }
procedure SetMIDIVolume(Left,Right:byte);   { MIDI/FM hangerő  }
procedure SetCDVolume(Left,Right:byte);     { Audio-CD hangerő }
procedure SetLineVolume(Left,Right:byte);   { Vonali hangerő   }
procedure SetMicVolume(Volume:byte);        { Mikrofon hangerő }
procedure SetSpeakerVolume(Volume:byte);    { Speaker hangerő  }
```

A hangerő-szabályozó eljárásokkal az egyes audioeszközök hangerejét tudjuk beállítani. A hangerőt mindenhol 8 biten (bájt paraméterrel) adhatjuk meg, de figyelni kell arra, hogy a hangerőket nem lehet 256 lépésben szabályozni, mert a regiszterek néhány alsó bitje nincs kihasználva. Ugyanez igaz a magas és mély szintek beállítására is.

Eszköz	Szabályozás	Bitkiosztás
Kimenet (*Master*)	32 szint, 2 dB-es lépésekben	csak a felső 5 bit
DSP (*DSP voice*)	32 szint, 2 dB-es lépésekben	csak a felső 5 bit
audio-CD (*Audio CD*)	32 szint, 2 dB-es lépésekben	csak a felső 5 bit
MIDI/FM (*MIDI*)	32 szint, 2 dB-es lépésekben	csak a felső 5 bit
Vonali bemenet (*Line In*)	32 szint, 2 dB-es lépésekben	csak a felső 5 bit
Mikrofon (*Microphone*)	32 szint, 2 dB-es lépésekben	csak a felső 5 bit
Speaker (*PC-Speaker*)	4 szint, 6 dB-es lépésekben	csak a felső 2 bit
Magas (*Treble level*)	15 szint, –14 dB-től 14 dB-ig	csak a felső 4 bit
Mély (*Bass level*)	15 szint, –14 dB-től 14 dB-ig	csak a felső 4 bit

```
{ ********** Egyéb beállítások ********** }
{ A kimeneten erősített eszközök kiválasztása }
procedure OutputSelect(mask:byte);
{ A bemeneti eszközök kiválasztása }
procedure InputSelect(leftmask,rightmask:byte);
{ A bemeneti erősítés kiválasztása }
procedure InputGain(LeftGain,RightGain:byte);
{ A kimeneti erősítés kiválasztása }
procedure OutputGain(LeftGain,RightGain:byte);
{ A mikrofon FGC/AGC erősítési mód beállítása }
procedure MicAGC(state:boolean);
{ A magas (treble) szint beállítása }
procedure TrebleLevel(Left,Right:byte);
{ A mély (bass) szint beállítása }
procedure BassLevel(Left,Right:byte);
{ Az aktív megszakítási vonal kiválasztása }
procedure IRQselect(mask:byte);
{ Az aktív DMA csatornák kiválasztása }
procedure DMAselect(HighDMA,LowDMA:byte);

{ **** Néhány eljárás a DSP kezeléséhez **** }
{ Mintavételi frekvencia beállítása lejátszáshoz }
procedure SetPlaySampleRate(SR:word);
{ Mintavételi frekvencia beállítása felvételhez }
procedure SetRecordSampleRate(SR:word);
{ DMA ciklusos DSP átvitel elkezdése }
procedure StartDSPtransfer(Tmode,MSmode:byte; Size:word);
```

A hangkártya egyéb jellemzőit és paramétereit (erősítés, megszakítási vonal, kimenetek, bemenetek) lehet megadni ezzel a néhány rutinnal. Az eljárások nevei — úgy gondoljuk — magukért beszélnek. A beállításoknál célszerű mindig az előre definiált állandókat használni!

A rutinok megvalósítását nem részletezzük, hiszen igen egyszerű valamennyi eljárás. :~)

```
implementation

procedure SetMixer(RegNum,Value:byte);      assembler;
asm
  mov  dx,Base
  add  dx,$0004
  mov  al,RegNum
  out  dx,al
  inc  dx
  mov  al,Value
  out  dx,al
end;
```

```
function  GetMixer(Regnum:byte):byte;        assembler;
asm
  mov  dx,Base
  add  dx,$0004
  mov  al,Regnum
  out  dx,al
  inc  dx
  in   al,dx
end;

procedure ResetMixer;
begin
 SetMixer($00,$00);
end;

procedure SetMasterVolume(Left,Right:byte);
begin
 SetMixer($30,Left);
 SetMixer($31,Right);
end;

procedure SetDSPVolume(Left,Right:byte);
begin
 SetMixer($32,Left);
 SetMixer($33,Right);
end;

procedure SetMIDIVolume(Left,Right:byte);
begin
 SetMixer($34,Left);
 SetMixer($35,Right);
end;

procedure SetCDVolume(Left,Right:byte);
begin
 SetMixer($36,Left);
 SetMixer($37,Right);
end;

procedure SetLineVolume(Left,Right:byte);
begin
 SetMixer($38,Left);
 SetMixer($39,Right);
end;

procedure SetMicVolume(Volume:byte);
begin
 SetMixer($3A,Volume);
end;
```

```
procedure SetSpeakerVolume(Volume:byte);
begin
 SetMixer($3B,Volume);
end;

procedure OutputSelect(Mask:byte);
begin
 SetMixer($3C,Mask and $1F);
end;

procedure InputSelect(LeftMask,RightMask:byte);
begin
 SetMixer($3D,LeftMask  and $7F);
 SetMixer($3E,RightMask and $7F);
end;

procedure InputGain(LeftGain,RightGain:byte);
begin
 SetMixer($3F,LeftGain  and $C0);
 SetMixer($40,RightGain and $C0);
end;

procedure OutputGain(LeftGain,RightGain:byte);
begin
 SetMixer($41,LeftGain  and $C0);
 SetMixer($42,RightGain and $C0);
end;

procedure MicAGC(State:boolean);
begin
 SetMixer($43,byte(State) and 1);
end;

procedure TrebleLevel(Left,Right:byte);
begin
 SetMixer($44,Left);
 SetMixer($45,Right);
end;

procedure BassLevel(Left,Right:byte);
begin
 SetMixer($46,Left);
 SetMixer($47,Right);
end;

procedure IRQselect(Mask:byte);
begin
 SetMixer($80,Mask);
 case Mask of
```

```
   imIRQ10 : IRQnumber:=10;
   imIRQ7  : IRQnumber:=7;
   imIRQ5  : IRQnumber:=5;
   imIRQ2  : IRQnumber:=2;
  end;
end;

procedure DMAselect(HighDMA,LowDMA:byte);
begin
 SetMixer($81,(HighDMA shl 4) or (LowDMA and 15));
 LowDMAch:=LowDMA and 15;
 HighDMAch:=HighDMA and 15;
end;

procedure SetPlaySampleRate(SR:word);
begin
 DSPwrite($41);
 DSPwrite(hi(SR));
 DSPwrite(lo(SR));
end;

procedure SetRecordSampleRate(SR:word);
begin
 DSPwrite($42);
 DSPwrite(hi(SR));
 DSPwrite(lo(SR));
end;

procedure StartDSPtransfer(Tmode,MSmode:byte; Size:word);
begin
 DSPwrite(Tmode);
 DSPwrite(MSmode);
 DSPwrite(lo(Size));
 DSPwrite(hi(Size));
end;


BEGIN
 ResetMixer;
 IRQselect(imIRQ5); DMAselect(dcDMA5,dcDMA1);
 OutputSelect(dmOutputAll);
 InputSelect(dmLeftAll,dmRightAll);
 InputGain(Gain_1x,Gain_1x); OutputGain(Gain_1x,Gain_1x);
 MicAGC(Off);
END. of unit
```

A unit főrésze a program elején lefut, és beállítja a következő paramétereket:

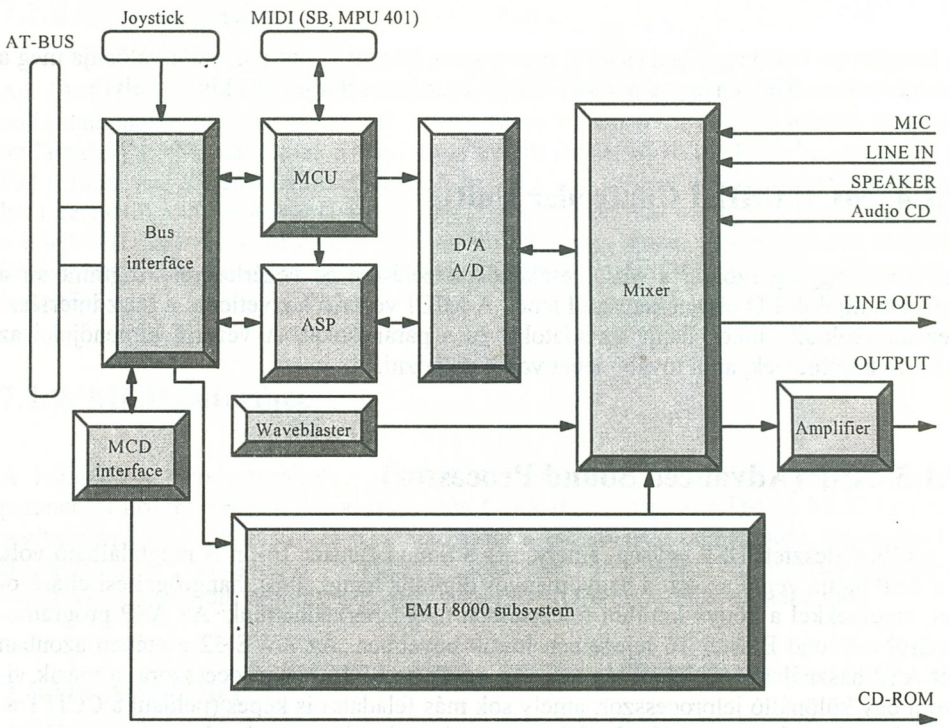
- megszakítási vonal: IRQ 5;
- DMA csatornák: DMA 5 (16 bites) és DMA 1 (8-bites);
- kimeneti eszközök: minden eszköz (*CD, Line In, Microphone*);
- bemeneti eszközök: minden eszköz (*CD, Line In, Microphone, MIDI*);
- bemeneti erősítés: normál egyszeres mindkét oldalon;
- kimeneti erősítés: normál egyszeres mindkét oldalon;
- mikrofon AGC: kikapcsolva.

Ha más paraméterekre van szükség az induláskor, akkor ezeket természetesen tetszés szerint módosíthatjuk. Azért állítottuk be így a regisztereket, mert a kártyának is ez az alapbeállítása az installálás után. A DSP-t kezelő eljárásoknál vigyázzunk arra, hogy ezek még nem elégségesek az átvitel korrekt elindításához, hiszen sok egyéb beállítást is el kell még végezni (megszakítási rutin, DMA vezérlő). Ezt bővebben a Sound Blaster kártyánál írtuk le. ;~)

7. Sound Blaster AWE 32

A Sound Blaster AWE 32 (*Advanced Wave Effects*) az a tagja a Sound Blaster családnak, amelyet már igen jó elektronikus szintetizátornak nevezhetünk. A kártya nem más, mint egy Sound Blaster 16-os, amelyre ráültettek még egy EMU 8000 típusú szintetizátort is. A szintetizátor chip — paraméterei és teljesítménye alapján — megfelel azoknak az áramköröknek, amelyeket a valódi billentyűs szintetizátorokba építenek be. A kártya programozásával részletesen nem fogunk megismerkedni, mert ez jóval túllép egy ilyen könyv lehetőségein, másrészt a gyártó (*Creative Labs*) egyáltalán nem ad meg semmilyen (programozáshoz is használható) hardverinformációt. Ezért most csupán a kártya lehetőségeit és felépítését ismertetjük röviden. :~(

7.1. Belső felépítés



45. ábra. A Sound Blaster AWE 32 belső szervezése

Az ábrán az AWE 32 blokksémája látható. Nézzük meg most külön-külön az egyes egységeket és azok főbb feladatait!

7.1.1. Joystick port

A joystick csatlakoztatására szolgáló bemenet. A legtöbb hangkártyán megtalálható ez a bemenet, így az AWE 32-ről sem maradhatott le.

7.1.2. MIDI port

Más analóg eszközök csatlakoztatására szolgál a MIDI port. Ezt egyébként ugyanazon a csatlakozón vezetik ki, mint a joysticket, ezért a kettőt egyszerre használni csak különleges kábellel lehet.

7.1.3. Bus Interface

A hangkártya belső egységei és a PC processzora közötti kommunikációt valósítja meg a busz interfész. Minden egyes parancs és adat ezen keresztül jut el a kívánt helyre.

7.1.4. MCU (MIDI Controller Unit)

Az MCU egység szolgál a MIDI eszközök kezelésére és vezérlésére. Programozni a számára kijelölt I/O címen keresztül lehet. A MIDI vezérlő közvetlenül a busz interfészhez kapcsolódik, innen kapja az adatokat és a parancsokat. A vezérlő kimenőjelei az ASP-re kerülhetnek, ahol tovább lehet velük dolgozni.

7.1.5. ASP (Advanced Sound Processor)

A továbbfejlesztett DSP egység, amely már a Sound Blaster 16-on is megtalálható volt. Az ASP hajtja végre azokat a hagyományos digitális hangkeltési, hangrögzítési eljárásokat, amelyekkel a könyv korábbi fejezeteiben megismerkedhettünk. Az ASP programozásáról a Sound Blaster 16 fejezetben írtunk bővebben. Az AWE 32 esetében azonban két ASP használatára is lehetőség van. Az egyik az SB16 hangprocesszora, a másik viszont egy különálló jelprocesszor, amely sok más feladatra is képes (például a CCITT a-Law és μ-Law kompressziós és dekompressziós algoritmusok hardveres megvalósítására).

7.1.6. D/A

A jelek átalakítását végzi az A/D (analóg-digitál) és a D/A (digitál-analóg) átalakító. Az átalakító nagy sebességű konverzióra képes mindkét irányban 8 és 16 biten, előjeles vagy előjel nélküli adatformátumban. Erről szintén a Sound Blaster 16-nál írtunk bővebben.

7.1.7. Mixer

Az AWE 32 keverője néhány apróbb módosítástól eltekintve ugyanaz, mint a Sound Blaster 16-é. A Mixer több analóg forrásból tudja összekeverni a kimenőjelet, amelyet szabványos vonali kimeneten (*Line Out*) és erősített formában (*OUTPUT*) is képes kiadni.

7.1.8. Waveblaster

Az AWE 32 előre definiált hullámformákat használ az egyes hangszerekhez. Ezeket hullámtábláknak hívjuk. Az AWE 32-n a memóriában lehet tárolni sokféle hangszer hullámalakját. (Mostanára ez a táblázatos hullámforma-előállítás háttérbe szorította az FM szintézissel történő szintetizálást.) A hangkártyán csupán egy csatlakozó van, amelyen keresztül a külső waveblaster egység kommunikálni tud a hangkártyával. Az MPU-n lehet MIDI parancsokat elküldeni a waveblaster egységnek, amely ezeknek megfelelően fog működni. A waveblasternek saját ROM memóriája és saját (EMU) szintetizátor áramköre van.

7.1.9. MCD Interface

A hangkártya képes vezérelni a CD-ROM olvasót. A CD-ROM néhány egyszerű paranccsal programozható (olvasás, analóg CD-k lejátszása). A CD-t az MCD interfész segítségével lehet elérni. A CD programozásával nekünk nem kell törődni, azt elvégzik a BIOS CD-vel kapcsolatos bővítései. Ezeket a bővítéseket általában egy meghajtó(*driver*)-program tartalmazza, amelyet a CD-ROM olvasóhoz mellékelnek.

7.1.10. Amplifier

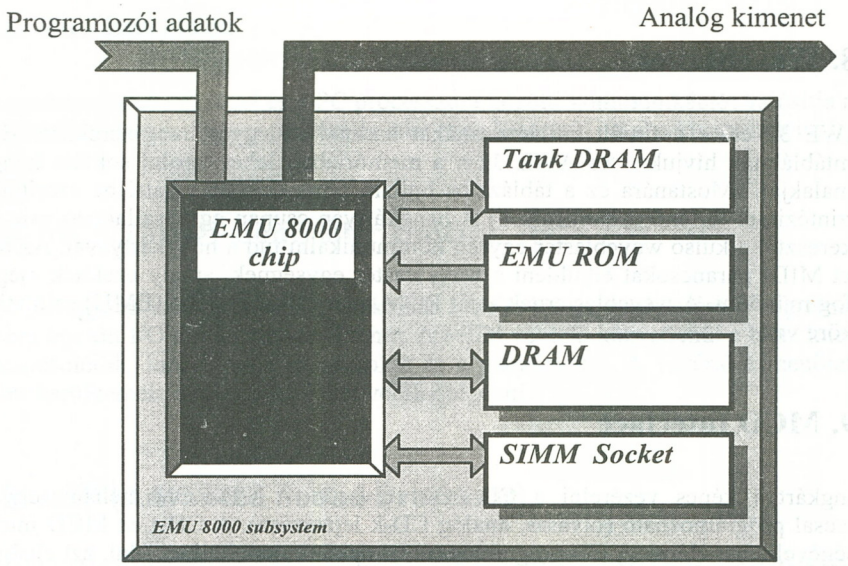
Erősítő. A hangkártya kimeneti jelét két 4 ohm/4 wattos hangszóróra lehet rávezetni, így a hang külső erősítő nélkül is hallgatható. Bármennyire is jó egy hangkártya, az erősítő

rész sohasem lehet olyan, mint egy valódi nagyobb teljesítményű külső hifi erősítő. Már az a tény is, hogy az erősítő a számítógépben van, szinte kizárja — vagy legalábbis nagyon megnehezíti — a hifi hangminőség előállítását. A belső erősítő játékprogramokhoz és kisebb alkalmazások hangjaihoz még éppen megfelel, de komolyabb munkához mindenképpen külső erősítőt és a vonali kimenet hangját kell használni.

7.1.11. EMU8000 subsystem

Az AWE 32 lelke az EMU 8000 jelű szintetizátor rendszer. Ez az az egység, amely jelentősen megnöveli a kártya teljesítményét (és az árát is!) a Sound Blaster 16-hoz képest. A szintetizátor nagyon sok szolgáltatást nyújt, de emiatt meglehetősen bonyolult a belső felépítése.

A szintetizátoregység belső felépítését a következő ábra mutatja:



46. ábra. Az EMU 8000 szintetizátoregység felépítése

A hangokat az EMU 8000 chip állítja elő, de ehhez memóriára van szüksége. A PC saját memóriája ilyen célra bizony nagyon kevés, ezért a kártyának saját memóriája van.

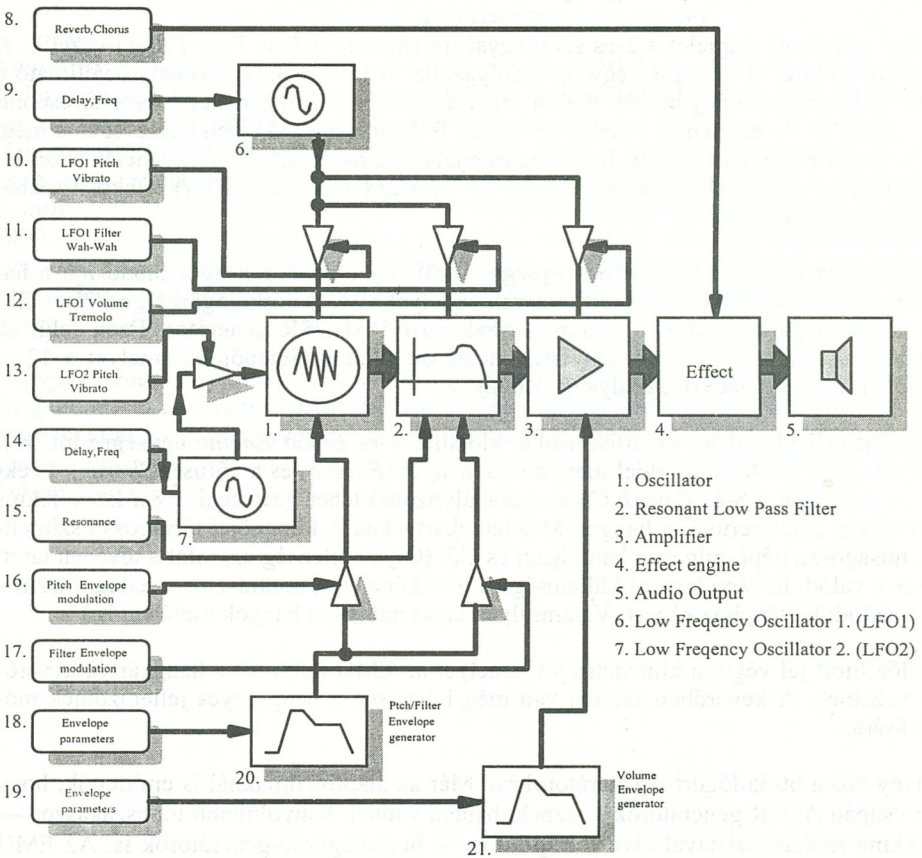
Az EMU ROM memóriája egy 1 Mbájtos terület, amelyben a kártya a legelterjedtebb, legáltalánosabb MIDI hangszerek hullámformáit tárolja. Bár az 1 Mbájt soknak tűnik — amint azt a későbbiekben majd látni fogjuk —, korántsem elegendő. Ezért a gyártó már eleve úgy tervezte a kártyát, hogy annak memóriáját később még lehessen bővíteni.

A ROM nagy hátránya, hogy nem lehet módosítani a tartalmát. Egy hangzás előállításánál viszont ez alapkövetelmény. Az EMU ROM mellett ezért található egy 512 Kbájtos DRAM memória is, amelyben a leggyakrabban használt hangmintákat lehet tárolni.

A gyártók gondoltak a további memóriabővítés lehetőségére is, ezért találhatunk a kártyán SIMM modulokhoz foglalatokat. A SIMM modulokat a világon sok helyen gyártják, és sokféle lehet a kapacitásuk. A beépített 512K memória így SIMM modulokkal egészen 28 Mbájtig bővíthető. (A valódi memóriahossz ugyan 32 Mbájt két 16 Mbájtos modul esetében, de a kártya belső szervezése miatt ebből csak 28 Mbájt használható fel.)

Az EMU 8000-nek harminckét — egymástól teljesen függetlenül használható — szintetizátor csatornája van, amelyek mindegyike hullámtábla alapján dolgozik.

Egyetlen csatorna belső felépítését az alábbi ábrán mutatjuk be:



47. ábra. Egy EMU 8000 hangcsatorna felépítése

Az ábra bal oldalán látható paraméterek (8–19) szoftverből állíthatók, ezek határozzák meg a csatorna kimenő jelalakját. Amint látható, meglehetősen sok adatot kell megadni. Hogyan is működik egy ilyen csatorna? Nos, elég bonyolultan. Három oszcillátor található a rajzon: az 1-es oszcillátor, amely a hang fő hullámalakját adja, valamint a 6. és a 7. alacsonyfrekvenciás oszcillátorok, amelyek modulálni tudják az 1-es oszcillátort. Az 1-es oszcillátor hullámtábla (*wavetable*) alapján dolgozik, tehát a memóriában tárolt hangminta a kimeneti jel alapharmonikusa. Ezt a jelet több más egység is befolyásolja. Egyrészt egy hagyományostól eltérő hatszakaszos (*Delay-Attact-Hold-Decay-Sustain-Release*) burkológörbe-generátor — a 20-as egység — a hangmagasságot modulálja, tehát nem a hangerőt, mint a normál ADSR generátorok, másrészt a két alacsonyfrekvenciás oszcillátor (LFO1 és LFO2, 6 és 7) is modulálja a hangot. Mindkét alacsonyfrekvenciás oszcillátoron be lehet állítani a frekvenciát (*Freq*) és a késleltetést (fáziseltolás, *Delay*). Ezek az egységek állítják elő a vibrátó (FM moduláció) effektust.

Az így előállított alapjelet a 2-es szűrőegységre (*Resonant Low Pass Filter*) vezetik. A szűrő működését is több más egység befolyásolja. A törésponti frekvencia beállítható a 15-ös rezonanciaszabályozóval. Külön lehet szabályozni a rezonanciát és a törésponti frekvenciát is. A rezonancia a szűrő törésponti frekvencián való viselkedését szabja meg. :–) A szűrőt modulálja a 20-as burkológörbe jele és a 6-os alacsonyfrekvenciás oszcillátor kimenete, amelynek az erősségét a 11-es vau-vau (*Wah-wah effect*) effektust szabályozó egység adja meg.

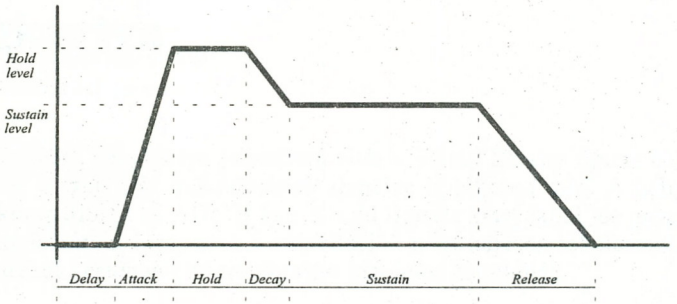
A szűrő kimenőjele a 3-as erősítőegységre kerül. Az erősítő a hangot módosítja a hagyományos amplitúdó burkológörbe szerint, amelyet a 21-es burkológörbe-generátor állít elő. Ez nem normál ADSR, hanem hatszakaszos DAHDSR generátor. Ezen felül az erősítőt modulálja a 6-os alacsonyfrekvenciás oszcillátor kimenőjele, amelyet a 12-es tremoló (AM moduláció) szabályozó szabályoz.

Az erősítőből kijövő jel az effektusokat előállító 4-es *Effect Engine* egységre jut. Két alapvető effektust lehet itt előállítani: a visszhangot (*Reverb*) és a kórust (*Chorus*). Ezeknek az erősségét a 8-as *Reverb,Chorus* szabályozóval lehet beállítani. Ez a két effektus teszi igazán „életszerűvé" a hangot. Más hangkártyáknál a kimenőjel túlságosan szintetikus, túlságosan gépi, míg ez a hang igazi és élő. (Olyan jelenség ez, mint a tévében látott kép és a valódi látvány közötti különbség. A tévé képe éles, kontrasztos, esetleg túlszínezett a valódi látványhoz képest. Valami ilyen tapasztalható a hangok esetében is.)

Az előállított jel végül a kimenetre jut, amelyet az EMU 8000-ről a hangkártya keverőjére vezetnek. A keverőben szintén van még lehetőség a hang egyes jellemzőinek módosítására.

Néhány szó a burkológörbe-generátorokról. Már az alapfogalmaknál is említettük, hogy nem csupán ADSR generátorok léteznek, hanem vannak bonyolultabb többszakaszos — a szakma szóhasználatával élve **többpontos** — burkológörbe-generátorok is. Az EMU

8000 generátorai hat szakaszra bontják a burkológörbét, ahogyan azt a 48. ábra is mutatja.



48. ábra. Hatszakaszos DAHDSR burkológörbe

Amint láthatjuk, a két új szakasz neve *Delay*, vagyis késleltetés és *Hold*, azaz tartás. A *Delay* szakasz a burkológörbe legelején található, a hangkiadás kezdetét lehet vele beállítani. A *Delay* után következik a már ismert *Attack* felfutási szakasz. A felfutás és a lecsengés közé iktatták be a *Hold* tartási szakaszt. Ezzel a legnagyobb amplitúdó értékén lehet tartani a hangot. A *Hold* után a *Decay* szakasz következik, majd a már ismert *Sustain* és *Release*. A másik nagy eltérés a normál hanggenerátorokhoz képest az EMU 8000 esetében az, hogy nem csak az amplitúdó változását lehet egy burkológörbével megadni, hanem a szűrő töréspontí frekvenciáját is burkológörbe szerint lehet változtatni. Ezért a hang sokkal jobban közelít a valóságoshoz, mint az egyszerű ADSR esetében!

Nagyjából ennyi egy csatorna működése, persze jócskán leegyszerűsítve. A részletesebb programozásra és a regiszterek leírására itt nem vállalkozunk. Ha az olvasó többet szeretne megtudni a kártya programozásáról, akkor meg kell szereznie a *Creative Labs* által kiadott információkat és segédprogramokat (*Sound Blaster AWE 32 Developer's Information Pack, 1994*).

8. A GRAVIS ULTRASOUND hangkártya

A Gravis Ultrasound hangkártya jelentősen eltér a Sound Blaster típusú hangkártyáktól, ami nem csoda, hiszen már működésének alapelve is teljesen más. A hangkártya alapesetben nem kompatíbilis az ADLIB és a Sound Blaster kártyákkal, de programmal ezek a kártyák szimulálhatók. Igaz, ezek a szimulációk nem teljesen tökéletesek, ezért egyes programok (játékok) estében előfordulhatnak bizonyos hanghibák.

8.1. A Gravis Ultrasound lehetőségei

- Vonali és erősített sztereó kimenet
- CD-ROM audio bemenet
- Sztereó vonali bemenet más audio eszközöktől (Line In)
- Sztereó mikrofonbemenet (Mic)
- 256 Kbájt — 1 Mbájtig bővíthető — saját memória hullámformák tárolására
- Joystick bemenet
- 32 digitális csatorna
- Hullámforma táblákkal megvalósított FM hangkeltés
- 6850 kompatíbilis MIDI UART (MIDI interfész)
- Minden csatornán külön állítható hangerő és balansz
- Egyidejű felvétel és lejátszás
- Állítható mintavételi frekvencia 44.1 kHz-ig
- 8-bites sztereó vagy mono felvétel
- 8- vagy 16 bites sztereó vagy mono lejátszás
- Szoftverből kiválasztható IRQ szint és DMA csatorna
- Jumperrel állítható I/O báziscím

A kártya 32 digitális csatornája egymástól teljesen függetlenül használható. Minden csatorna képes 8- vagy 16 bites digitális hangminták lejátszására, amelyeket a kártya saját memóriájában (*GUS DRAM*) lehet tárolni, így a hangok nem foglalnak helyet a PC memóriájában. A GUS DRAM tartalmazhat egyszerű digitális hangmintát, egész hangfájlokat (*sound track*) vagy egyszerű hangjegyeket valamely „digitális" (máshol definiált) hangszerhez. A hangkártya minden egysége elérhető a processzorral, ugyanakkor képes függetlenül is dolgozni, ami nagymértékben megkönnyíti a programozó munkáját. Mind a hangprocesszor (GF1), mind a kártya memóriája (GUS DRAM) egyszerű I/O utasításokkal elérhető és programozható. Mindezek miatt a GUS — bár sokkal

összetettebb és bonyolultabb, mint a kisebb hangkártyák — igen könnyen használható, és egészen különleges hangfeldolgozási feladatokra is alkalmas. Nézzük meg néhány előnyös tulajdonságát:

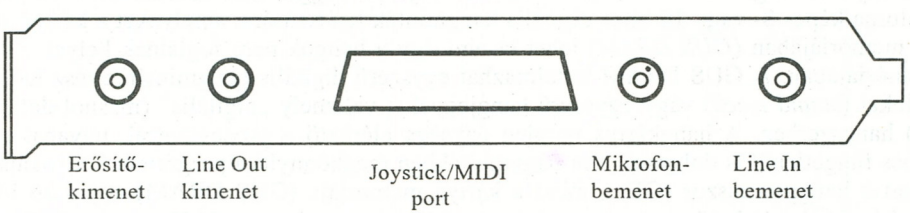
– A kártyának saját memóriája van. A hangszerek mintáit (ezek a *patch*-ek) elegendő egyszer betölteni a memóriába, ott tetszés szerint lehet módosítani vagy megszüntetni őket, ha már feleslegesek.
– A burkológörbe előállítása hardver úton történik. Az egyes ciklusok előállításánál a processzor minimális közreműködésére van csupán szükség, ráadásul nem csak egyszerű ADSR, hanem többpontos burkológörbe is előállítható. Ugyanígy hardver úton lehet elvégezni az amplitúdómodulációt is (tremoló).
– Az FM hangkeltés hullámforma táblák (*wavetable*) alapján történik. A burkológörbés hangok esetében a hullámalakot nem megkötött formákból kell kiválasztani, hanem egyénileg lehet definiálni azokat. Ezt a módszert alkalmazzák a profi szintetizátorokban is.
– A hangminták letöltése miatt a PC memóriája szabadon felhasználható más célokra. A GUS DRAM-ban tárolt hangok lejátszása bármikor leállítható és újraindítható.

8.2. Hardver áttekintés

8.2.1. Csatlakozók

A Sound Blaster 16-hoz hasonlóan a Gravisnek sincs hangerő-szabályozó potenciométere. A gyártók valószínűleg azt vallják, hogy egy ilyen hangkártyához egyrészt használjunk külső erősítőt, másrészt szoftver úton a hangerő szabályozható. (Sajnos ezt néhány játékprogram nem veszi figyelembe, hanem egyszerűen maximális hangerőt állít be. Legalábbis a hangkártya első kipróbálásakor a hangszórók olykor — a teljes hangerő miatt — egyszerűen leperegnek a számítógép dobozáról.)

A csatlakozók már ismertek, de az elrendezésük kissé talán szokatlan:



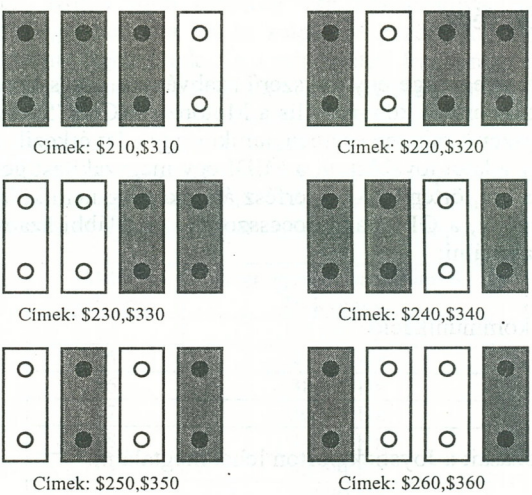
49. ábra. A Gravis Ultrasound csatlakozói

8.2.2. I/O címek

A Gravis Ultrasound három különálló I/O címtartományt használ, de ezeket csak együttesen lehet módosítani. A beállítást négy *jumperrel* kell elvégezni. A három címtartomány a következő:

I/O címtartomány	Funkció
$200 ($2x0)	a $200-as tartományban elérhető regiszterek
$300 ($3x0)	a $300-as tartományban elérhető regiszterek
$388	az ADLIB regiszterek (mindig állandóak)

A négy jumper állapotától függően az első két tartományt meg lehet változtatni.



50. ábra. A Gravis Ultrasound I/O beállításai

A fenti beállításokból a $220,$320 a gyárilag beállított érték. Amint látható, az I/O címnek csupán a középső számjegyét lehet változtatni, ezért a továbbiakban a következő módon jelöljük a címeket: $2x0, $3x0

A beállított I/O címhez mindig tartozik egy tartomány, amely a Gravisnél egy 16 bájt hosszú terület. Ha például a $220,$320 az aktuális I/O bázis, akkor a két tartomány:

I/O bázis	I/O tartomány
$220	$220 ... $22F
$320	$320 ... $32F

Az ADLIB kompatíbilis címeket nem lehet átállítani, hiszen erre az ADLIB-nél sincs lehetőség. Ezért az ADLIB kompatíbilis programozás során (ha a megfelelő emulációs szoftver installálva van) mindig a $388,$389 címeket kell használni!

8.2.3. A GUS DRAM memória

A hangkártya saját memóriája a verziószámtól függően más-más méretű lehet, de általában csupán 256 Kbájt memória van egy új kártyán. Ezt normál dinamikus RAM-okkal lehet bővíteni 1 Mbájtig. A RAM-oknak legalább 80 ns-os vagy ennél gyorsabb elérési idejűeknek kell lenniük. A kártyán megtalálhatjuk az üres RAM foglalatokat, a kézikönyvben pedig a bővítéshez használható típusokat és a bővítés módját. Természetesen annál jobb a kártya teljesítménye, minél nagyobb a memóriája, hiszen a nagyobb memóriában több hangminta tárolható.

8.2.4. MIDI interfész

A GUS MIDI interfész egysége egy egyszerű szabványos soros kommunikációra alkalmas UART chip. Az interfész kompatíbilis a Motorola MC68C50 UART áramkörrel. A MIDI működése egyszerű: minden esetben, amikor egy adat érkezik a MIDI interfészről, vagy az interfész egy adatot továbbított, a MIDI egy megszakítást generál. Ez egyébként más áramköröknél is így történik. Az interfész áramkörileg nem különül el a hangkártya fő végrehajtó egységétől, a GF1 hangprocesszortól. Az alábbi szabványos értékeket és beállításokat lehet használni:

- 31.25 kHz;
- aszinkron kommunikáció;
- 1 startbit;
- 8 adatbit;
- 1 stopbit;

Az interfész csatlakozásait a Joystick porton lehet megtalálni.

8.2.5. Joystick

A Gravis joystick csatlakozója semmivel sem rosszabb a többi hangkártya joystick csatlakozójánál. Használható egy és két joystick csatlakoztatására, és a MIDI interfész kivezetései is itt találhatók meg. Nagy előnye más kártyák joystick portjaihoz képest, hogy ezt a joystick portot hardversebesség-kompenzáló egységgel látták el. :–) Ez főleg akkor lehet hasznos, ha olyan régen írt szoftvert használunk, amely nem oldja meg az időzítéseket, így az újabb gépeken nagyon gyors, kezelhetetlen a joystick. A kártyán a *Game E/D jumper* segítségével lehet letiltani a joystick portokat, ha más kártyán (például IDE+) is rajta vannak már. A legújabb verziójú kártyáknál (3.4 felett) ez már szoftverből történik.

8.2.6. A GF1 hangprocesszor (Voice Sound Synthetizer)

Amint már említettük, a GUS alapvetően RAM bázisra épülő hangkártya. A hangprocesszor többféle műveletet is tud végezni a betöltött hangmintákkal. Minden csatorna függetlenül programozható, a kimeneten megjelenő hang a csatornák kimeneteinek összekevert jele. Amplitúdómoduláció, ADSR-definiálás, teljes hangerő-szabályozás és egyéb speciális effektusok, például az alacsonyfrekvenciás AM, valamint frekvenciaeltolással különböző hangok kiadása ugyanazon hangszerrel; mindezek nem jelentenek gondot a GUS-nak.

A GF1 ***pipeline*** „csővezeték" rendszerű feldolgozó egység. Ez azt jelenti, hogy a GF1 sorban elvégzi minden egyes csatornán az éppen következő műveletet, majd ezt az egészet ciklikusan kezdi elölről. Minden csatornának kell valamekkora kiszolgálási idő, ami alatt a GF1 elvégzi a csatornára előírt feladatot. Tipikusan 1.6 µs idő szükséges egy csatorna kiszolgálásához. Minél több csatorna van bekapcsolva, annál tovább tart a kiszolgálási idő. Ezért egy csatorna két kiszolgálása közötti idő egyre jobban megnő, ahogy egyre több csatornát használunk. Ez a mintavételi frekvencia csökkenését okozza, ami viszont a hangminőség romlását eredményezi. Ez persze csak egy bizonyos csatornaszám felett jelentkezik. Tizennégy aktív csatorna esetében a kártya még mindig képes a 44.1 kHz-es mintavételre, ez után viszont egy megadott értékkel a mintavételi frekvencia lineárisan csökken egészen 19 kHz-ig (persze még ez sem túl rossz, ha figyelembe vesszük, hogy a csatornák száma 32). Az alábbi táblázatban a mintavételi frekvencia csökkenése követhető nyomon a csatornák számától függően:

Aktív csatorna	Legnagyobb mintavételi frekvencia
14	44100 Hz
15	41160 Hz
16	38587 Hz
17	36317 Hz
18	34300 Hz
19	32494 Hz
20	30870 Hz
21	29400 Hz
22	28063 Hz
23	26843 Hz
24	25725 Hz
25	24696 Hz
26	23746 Hz
27	22866 Hz
28	22050 Hz
29	21289 Hz
30	20580 Hz
31	19916 Hz
32	19293 Hz

8.2.7. Különböző verziójú kártyák

Ahogyan a Sound Blaster esetében is különböző verziószámú kártyák kerültek forgalomba, úgy a Gravis Ultrasoundnak is többféle változatát lehet kapni. Jelenleg három típus és ezek más-más verziói vannak forgalomban:

- Gravis Ultrasound;
- Advanced Gravis Ultrasound;
- Gravis UltraMAX.

Az újabb változatok mindig valamivel többet tudnak elődeiknél, de természetesen többe is kerülnek. A regiszterek leírásánál megpróbáljuk majd kiemelni az egyes verziók közötti különbségeket.

8.3. A Gravis Ultrasound regiszterei

A Gravis Ultrasoundnak meglehetősen sok regisztere van, ezért ezeket majd kisebb, logikailag összetartozó csoportokban fogjuk bemutatni. A teljes áttekinthetőség kedvéért nézzük meg az összes I/O címet egy táblázatban:

I/O cím	Írás,olvasás	Funkció
$2x0	–	Gravis Ultrasound bázis I/O cím (*Base*)
$3x0	*write*	MIDI interfész, vezérlő (*MIDI interface control*)
$3x0	*read*	MIDI interfész, állapot (*MIDI interface status*)
$3x1	*write*	MIDI interfész, adás (*MIDI interface transmit data*)
$3x1	*read*	MIDI interfész, vétel (*MIDI interface receive data*)
$201	*write*	joystick, időzítőindítás (*JOYSTICK trigger timer*)
$201	*read*	joystick, adatbeolvasás (*JOYSTICK read data*)
$3x2	*read/write*	GF1, lapregiszter (*GF1 page register*)
$3x3	*read/write*	GF1, regiszterkiválasztó (*GF1 global register select*)
$3x4	*read/write*	GF1, adat alsó bájt (*GF1 data low byte*)
$3x5	*read/write*	GF1, adat felső bájt (*GF1 data high byte*)
$2x6	*read*	megszakításállapot (*IRQ status register*)
$2x8	*read/write*	időzítővezérlő regiszter (*TIMER control register*)
$2x9	*write*	időzítő adatregiszter (*TIMER data register*)
$3x7	*read/write*	GUS DRAM írás/olvasás (*DRAM read/write*)
$2x0	*write*	keverővezérlő regiszter (*MIXER control*)
$2xB	*write*	megszakításvezérlő (*IRQ control register*)
$2xB	*write*	DMA vezérlő (*DMA control register*)
$2xF	*read/write*	regisztervezérlő (*register control, 3.4+*)
$7x6	*read*	hangkártya verzió (*board version, 3.7+*)
$7x6	*write*	keverő vezérlő (*MIXER control register*)
$3x6	*write*	keverő adatregiszter (*MIXER data register*)
$3xC...$3xF	*read/write*	UltraMAX codec regiszterek (*MAX codec registers*)
$3x6	*write*	UltraMAX vezérlő (*MAX control register*)

Bizonyára feltűnt, hogy egyes I/O címek többször is előfordulnak, vagy valamely funkció más-más címen is megtalálható. Ez szándékosan van így, a későbbiekben erre részletesen kitérünk.

Az egyes verziókon megtalálható még az ún. kódoló-dekódoló áramkör, a codec, amely még néhány további I/O címet igényel. Ezeket az alábbi táblázatban tüntettük fel:

I/O cím	Írás, olvasás	Funkció
$530...$533	*read/write*	újabb bővítőkártya terület, UltraMAX (*Daughter card*)
$604...$607	*read/write*	újabb bővítőkártya terület, UltraMAX (*Daughter card*)
$E80...$E83	*read/write*	újabb bővítőkártya terület, UltraMAX (*Daughter card*)
$F40...$F43	*read/write*	újabb bővítőkártya terület, UltraMAX (*Daughter card*)
Base+$00	*read/write*	codec, címkiválasztás (*CODEC address select*)
Base+$01	*read/write*	codec, adat (*CODEC data*)
Base+$02	*read/write*	codec, állapot (*CODEC status*)
Base+$03	*read/write*	codec, PIO (*CODEC PIO*)

Ha esetleg most még valami nem teljesen világos, nem kell megijedni, a regiszterek részletes ismertetésénél minden a helyére fog kerülni!

8.4. Regiszterek

Amint azt már eddig is láthattuk, a Gravis Ultrasound meglehetősen eltér a Sound Blaster család egyes hangkártyáitól. A regiszterkészlet szintén elég sajátos, ezért eltérünk a regiszterleírás eddigi formájától, és egy kicsit másképp mutatjuk be a regisztereket. A következőkben táblázatosan, bitenként fogunk majd bemutatni minden regisztert (persze csak ahol a bitenkénti kiosztás fontos):

Regiszter regisztercím(elérés): Név (*angol név*)

Bitszám	Angol elnevezés
0. bit	*a 7. bit angol neve*
1. bit	*a 6. bit angol neve*
2. bit	*az 5. bit angol neve*
3. bit	*a 4. bit angol neve*
4. bit	*a 3. bit angol neve*
5. bit	*a 2. bit angol neve*
6. bit	*az 1. bit angol neve*
7. bit	*a 0. bit angol neve*

A táblázat után a regiszter részletes leírása következik. Itt leírjuk az egyes bitek funkcióját, az esetleges összefüggéseket és más különleges tudnivalókat, ha a használat során valamire ügyelni kell. (Megítélésünk szerint az angol nevek sokkal tömörebbek, és jobban kifejezik a bitek jelentését, mint a magyar elnevezések. Ezért talán nem haszontalan, ha az olvasó a bitek eredeti nevét is megjegyzi, nem csak a jelentésüket.)

8.4.1. MIDI regiszterek

Regiszter $3x0(w): MIDI vezérlő (*MIDI control port*)

Bitszám	Angol elnevezés
0. bit	*Master Reset (1)*
1. bit	*Master Reset (1)*
2. bit	*Reserved*
3. bit	*Reserved*
4. bit	*Reserved*
5. bit	*Xmit IRQ enabled (1)*
6. bit	*Xmit IRQ enabled (0)*
7. bit	*Receive IRQ enabled (1)*

Bit 0,1: A 0. és az 1. bit inicializálja a MIDI interfészt. Az inicializáláshoz először mindkét bitet 1-be, majd egy kis várakozás után 0-ba kell állítani. Normál használatkor mindkét bit legyen 0.
Bit 5,6: Az 5. és a 6. bit 10 állapota esetén adás közben engedélyezett a megszakításkérés, máskor viszont nem. A 7. bit a vétel megszakításkérését engedélyezi, ha 1. Amikor a megszakítások engedélyezve vannak, a MIDI interfész megszakítást kér minden alkalommal, ha egy adatot elküldött (adásnál), vagy egy adatot vett (vételnél).
Bit 2,3,4: A regiszter 2. 3. és 4. bitje későbbi fejlesztésre van fenntartva.

Regiszter $3x0(r): MIDI állapot (*MIDI status port*)

Bitszám	Angol elnevezés
0. bit	*Receive register full*
1. bit	*Transmit register empty*
2. bit	*Reserved*
3. bit	*Reserved*
4. bit	*Framing error*
5. bit	*Overrun error*
6. bit	*Reserved*
7. bit	*Interrupt pending*

Bit 0,1: A 0. bit 1-es állapota a vevőregiszter betelt állapotát jelzi, azaz megtörtént egy adat vétele. Az 1. bit az adóregiszter üres állapotát jelzi, tehát a MIDI elküldött egy adatot.
Bit 4: A 4. bit a kerethibát jelzi, azaz az átvitel során nem volt megfelelő a startbit vagy a stopbit.
Bit 5: Az 5. bit a túlfutási hibát mutatja.
Bit 7: A függőben lévő megszakítás jelenlétét jelzi. Ha ez a bit 1-es, akkor az átvitel után megszakítás történt, tehát kezelni kell.
Bit 2,3,6: A regiszter többi bitje későbbi fejlesztésre van fenntartva.

 **Regiszter $3x1(r/w):** MIDI adat (*MIDI data port*)

Ebben a regiszterben kell elhelyezni az elküldendő adatot adáskor, illetve innen tudjuk kiolvasni a vett adatot vételkor.

8.4.2. GF1 globális regiszterek

 **Regiszter $3x2(r/w):** GF1 lapregiszter (*GF1 page register*)

A lapregiszter feladata az aktuális — éppen programozás alatt álló — digitális csatorna kiválasztása. A csatornaadatokat beállító regiszterek mindig ennek a regiszternek az értékét tekintik aktív csatornának. A regiszterbe írt érték 5-bites lehet, azaz 0 és 31 közé kell esnie. Alapesetben csak 0 és 13 közé eshet, mert az inicializálás után a hangkártya csak 14 csatornát állít aktív állapotba. Ha ezt a regisztert megszakítás alatt módosítjuk (ami természetes, hiszen a megszakításnak az a feladata, hogy egyes csatornák paramétereit megváltoztassa), akkor a megszakítás végén vissza kell állítani a regiszter eredeti állapotát, máskülönben nagy lesz a baj!

 **Regiszter $3x3(r/w):** GF1 belsőregiszter-kiválasztó (*GF1 register select*)

Ezzel a regiszterrel kell kiválasztani, hogy melyik belső GF1 regisztert akarjuk használni. A GF1 belső regiszterkészlete alapvetően két részre osztható:

- globális regiszterek (*Global registers*);
- csatornaparaméter regiszterek (*Voice-channel registers*).

A globális regiszterekkel általános paramétereket lehet beállítani, míg a csatornaparaméter regiszterek egy-egy csatorna jellemzőit definiálják. Először nézzük meg a globális regisztereket:

Cím	Írás, olvasás	Hossz	A belső regiszter funkciója
$41	*read/write*	8 bit	DMA vezérlő (*DMA control*)
$42	*write*	16 bit	DMA kezdőcím (*DMA start address*)
$43	*write*	16 bit	DRAM cím alsó rész (*DRAM address low*)
$44	*write*	8 bit	DRAM cím felső rész (*DRAM address high*)
$45	*read/write*	8 bit	időzítő vezérlő (*Timer control*)
$46	*write*	8 bit	időzítő #1 adat (*Timer #1 data*)
$47	*write*	8 bit	időzítő #2 adat (*Timer #2 data*)
$48	*write*	8 bit	mintavételi frekvencia (*Sampling frequency*)
$49	*read/write*	8 bit	mintavételezési vezérlő (*Sampling Control*)
$4B	*write*	8 bit	joystick DAC (*JOYSTICK trim DAC*)
$4C	*read/write*	8 bit	inicializálás (*RESET*)

A táblázat cím oszlopa a belső címeket tartalmazza, amelyet a regiszterkiválasztó $3x3 regiszterbe kell beírni, ha az adott regisztert el akarjuk érni. Amint látható, a regiszterek között előfordulnak 16 bites regiszterek is. Ezeket 16 bites I/O utasításokkal lehet megcímezni (`out dx,ax`).

GF1 Regiszter $41(r/w): DMA vezérlő (*DMA control register*)

Bitszám	Angol elnevezés
0. bit	*Enable DMA transfer*
1. bit	*DMA direction*
2. bit	*DMA channel type*
3. bit	*DMA rate divider*
4. bit	*DMA rate divider*
5. bit	*DMA interrupt enable*
6. bit	*DMA IRQ pending (r), Data size (w)*
7. bit	*Invert MSB (w)*

A DMA-vezérlő regiszter a kártya és a PC-memória közötti adatforgalom lehetséges módjait határozza meg.
Bit 0: A DMA átvitel elindítása. Ha ezt a bitet 1-be állítjuk, akkor a DMA átvitel megkezdődik a GF1 és a PC memóriája között. Ez persze csak akkor lehetséges, ha a DMA vezérlőt ez előtt megfelelő módon beállítottuk! (Erre nagyon kell figyelni, máskülönben a DMA kérések kiszolgálatlanok maradnak, és ez nem jó!)
Bit 1: A DMA átvitel iránya. Ezzel a bittel lehet megadni, hogy felvétel vagy lejátszás következik-e. Ez persze még nem elegendő, de az átvitelt ez a bit vezérli. A 0 jelenti az írást (lejátszást), az 1 pedig az olvasást (felvételt).
Bit 2: Az aktív DMA csatorna típusa:

0 – 8-bites csatorna
1 – 16 bites csatorna

Ennek a bitnek egyeznie kell az adatszélességgel !
Bit 3,4: A DMA időzítés osztási állandója. Az átvitel legnagyobb sebessége 650 kHz, ezt lehet osztani a két bit állásától függően:

00 – osztás eggyel (*divide by 1*)
01 – osztás kettővel (*divide by 2*)
10 – osztás hárommal (*divide by 3*)
11 – osztás néggyel (*divide by 4*)

A hangprocesszor a megadott osztási állandótól függően csökkenti a DMA kérések számát, ha erre van szükség valamilyen okból.
Bit 5: DMA megszakítás engedélyezés. Ha ez a bit 1, akkor a GF1 a DMA átvitel végén megszakítást kér a beállított IRQ vonalon.
Bit 6: Megszakításérzékelés és adathossz beállítás. Ennek a bitnek más-más szerepe van íráskor és olvasáskor.

Íráskor: DMA adathosszúság, 0 esetén 8 bit, 1 esetén 16 bit.
Olvasáskor: DMA megszakítás érkezett, ha az értéke 1.

Emlékezzünk rá, hogy az adathosszúságnak és a csatorna típusának mindig egyeznie kell!

Bit 7: Kettes komplemensű adatábrázolás. Ezzel a bittel lehet kiválasztani a használt számábrázolást:

0 – normál, előjel nélküli 8- és 16 bites adatok
1 – előjeles, kettes komplemens ábrázolású 8- és 16 bites adatok

 **GF1 Regiszter $42(w):** DMA kezdőcím (*DMA start address*)

A DMA átvitel GUS DRAM-beli kezdőcímét definiálja ez a regiszter. A GUS DRAM 1 Mbájt hosszú lehet, ezért a címeket 20 biten kell megadni. Ez a regiszter viszont csak 16 bites, tehát négy bit még hiányzik. A Gravis tervezői úgy gondolták, hogy négy bit miatt nem készítenek még egy regisztert, ezért a DMA átvitel címét csupán a felső 16 bittel (19...4) adhatjuk meg, az alsó négy bit pedig mindig 0. Az alsó bitek hiánya miatt a DMA átvitelnek megkötései vannak, nevezetesen minden 8-bites átvitel csak 16 bájtos határon, míg a 16 bites átvitel csak 32 bájtos határon kezdődhet.

 **GF1 Regiszter $43(w):** DRAM cím alsó rész (*DRAM address low*)

 **GF1 Regiszter $44(w):** DRAM cím felső rész (*DRAM address high*)

Ezzel a két regiszterrel tudjuk lineárisan megcímezni a hangkártya memóriáját. A $43-as regiszter 16 bites, így ez adja a fizikai cím alsó 16 bitjét (0...15), a $44-es regiszter pedig csupán 4-bites, ez a fizikai cím felső négy bitjét (16...19) adja. A címregiszterek beállítása után a *$3x7 DRAM írás/olvasás* regiszterből lehet kiolvasni az adott bájt értékét, vagy beállítani egy új értéket.

 **GF1 Regiszter $45(w):** Időzítővezérlő (*Timer control*)

Bitszám	Angol elnevezés
0. bit	*Reserved (set to 0)*
1. bit	*Reserved (set to 0)*
2. bit	*Enable Timer #1 Interrupt*
3. bit	*Enable Timer #2 Interrupt*
4. bit	*Reserved (set to 0)*
5. bit	*Reserved (set to 0)*
6. bit	*Reserved (set to 0)*
7. bit	*Reserved (set to 0)*

A két beépített időzítőegység képes megszakítást kérni, ha lejárt. Ez csak akkor történik meg, ha az időzítővezérlő regiszterben a megfelelő bittel engedélyezzük az időzítő megszakítását.

Bit 2: Engedélyezi az első időzítő megszakítási kérelmeit.

Bit 3: Engedélyezi a második időzítő megszakítási kérelmeit.
A regiszter többi bitjét fejlesztésre tartották fenn, mindig 0-ba kell őket állítani.

 **GF1 Regiszter $46(w):** Első időzítő adat (*Timer #1 data*)

 **GF1 Regiszter $47(w):** Második időzítő adat (*Timer #2 data*)

Ezek az időzítők számláló regiszterei. A két regiszter az ADLIB időzítőihez hasonlóan működik. A regiszterbe írt értékek a megadott időállandók szerinti időközökben növekednek. Ha az időzítő túlcsordul, egy megszakítást generál, feltéve, hogy az időzítő-vezérlőben a megfelelő bit engedi a megszakítást. Az első időzítő 80 µs, a második időzítő pedig 320 µs leteltével növeli értékét.

 **GF1 Regiszter $48(w):** Mintavételi (felvételi) frekvencia (*Sampling frequency*)

Ez a regiszter a mintavételi frekvencia megadására szolgál. A különböző frekvenciákhoz egy-egy 8-bites konstans érték tartozik, amelyet a következő képlet alapján tudunk kiszámítani:

SR=9878400/(16*(Freq+2))

 **GF1 Regiszter $49(w):** Mintavételezési (felvételi) vezérlő (*Sampling control*)

Bitszám	Angol elnevezés
0. bit	*Start sampling*
1. bit	*Mono/Stereo mode select*
2. bit	*DMA width (8 or 16 bit)*
3. bit	*Reserved (set to 0)*
4. bit	*Reserved (set to 0)*
5. bit	*DMA Interrupt enable*
6. bit	*DMA Interrupt pending (r)*
7. bit	*Invert MSB*

Bit 0: A felvétel megkezdése. Ha a DMA vezérlőt megfelelően beállítottuk, akkor ezzel a bittel lehet elindítani a mintavételezést.
Bit 1: A hangminta típusát definiálja:
0 – mono minták
1 – sztereó minták
Sztereó módban a Sound Blaster Próhoz hasonlóan a két oldal hangmintái felváltva követik egymást.
Bit 2: A DMA csatorna szélességét definiálja:
0 – 8-bites csatorna
1 – 16 bites csatorna

Bit 5: Engedélyezi a DMA átvitel megszakításkérését, ha az átvitel befejeződik.
Bit 6: Ezt a bitet csak olvasni lehet. Ha értéke 1, akkor egy DMA megszakítás történt, amelyet kezelni kell.
Bit 7: Kettes komplemensű adatábrázolás. Ezzel a bittel tudjuk kiválasztani a használt számábrázolást:

0 – előjeles, kettes komplemens ábrázolású 8- és 16 bites adatok
1 – normál, előjel nélküli 8- és 16 bites adatok

GF1 Regiszter $4B(w): Joystick DAC (*JOYSTICK trim DAC*)

A regiszter a joystick pozitív tápfeszültségének értékét szabja meg. Az alapbeállítás 29, ekkor 4.3 V a feszültség. Ezt a regisztert akkor használhatjuk, amikor a gép sebessége miatt a joystick beolvasásának sebességét meg kell változtatni.

GF1 Regiszter $4C(w): Inicializálás (*RESET*)

Bitszám	Angol elnevezés
0. bit	*Master Reset*
1. bit	*DAC output enable*
2. bit	*GF1 chip Master Interrupt enable*
3. bit	*Reserved (set to 0)*
4. bit	*Reserved (set to 0)*
5. bit	*Reserved (set to 0)*
6. bit	*Reserved (set to 0)*
7. bit	*Reserved (set to 0)*

A regiszter a GF1 hangprocesszor inicializálására szolgál.
Bit 0: GF1 inicializálás.

0 – inicializálás (*reset method*)
1 – normál működési állapot (*running mode*)

Ha a bitet hosszabb ideig 0-ban tartjuk, a hangkártya az inicializálási folyamatot befejezi ugyan, de a GF1 nem használható!
Bit 1: A DAC kimenet engedélyezése. A kimenet csak akkor hallható, ha ez a bit 1.
Bit 2: A GF1 fő megszakítás engedélyezése. Bármilyen megszakítás csak akkor tud generálódni, ha ez a bit 1.
A regiszternek normál működés esetén $07-et kell tartalmaznia.

8.4.3. GF1 csatornaparaméter regiszterek

A globális regiszterek mellett a GF1 regiszterek másik nagy csoportját a csatornaparamétereket definiáló regiszterek képezik. Ezeket a regisztereket olvasni és írni is lehet, de a belső szervezés miatt más címeket kell használni. Amikor írni akarunk egy regisztert, akkor ezt mindig a $00...$0F tartományban lehet megtenni, olvasni pedig mindig a $80...$8F tartományból tudunk.

Ezek a regiszterek tartalmazhatnak önmódosító biteket. A GF1 hangprocesszor ciklikusan eléri ezt a regisztertartományt, és a belső állapotoknak megfelelően állítja be az egyes flagbiteket. A kártyát állító szoftvernek összhangban kell lennie ezekkel a belső állapotváltozásokkal, hiszen előfordulhat, hogy a GF1 megváltoztat egy értéket éppen azután, hogy beállítottuk. Ez azért lehetséges, mert a GF1 — amint azt már említettük — egy ***pipeline*** típusú processzor, amely olvasás–módosítás–írás (*read-modify-write*) operációval dolgozik. Ha tehát éppen akkor módosítunk egy regisztert, miután a GF1 már beolvasta az előző értékét, de még nem írta ki a következőt (azaz a módosítási ciklusban), akkor ez hibát fog okozni, vagy a GF1 egyszerűen figyelmen kívül hagyja a beírt értéket. Ezt a problémát úgy lehet kikerülni, hogy kétszer írunk az adott regiszterbe. A két írás között kell egy kis várakozás. Ez általában háromszorosa a GF1 műveleti idejének (3*1.6 µs). A továbbiakban az ilyen önmódosító biteket a « jellel jelezzük majd.

Írás	Olvasás	Hossz	Funkció
$00	$80	8	hangvezérlő (*Voice control*)
$01	$81	16	frekvenciavezérlő (*Frequency control*)
$02	$82	16	kezdőcím felső rész (*Start address high*)
$03	$83	16	kezdőcím alsó rész (*Start address low*)
$04	$84	16	végcím felső rész (*End address high*)
$05	$85	16	végcím alsó rész (*End address low*)
$06	$86	8	hangerő-változtatási mérték (*Volume ramp rate*)
$07	$87	8	hangerő kezdőérték (*Volume ramp start*)
$08	$88	8	hangerő végérték (*Volume ramp end*)
$09	$89	16	aktuális hangerő (*Current volume*)
$0A	$8A	16	aktuális cím felső rész (*Current address high*)
$0B	$8B	16	aktuális cím alsó rész (*Current address low*)
$0C	$8C	8	balanszpozíció (*Pan position*)
$0D	$8D	8	hangerővezérlő (*Volume control*)
$0E	$8E	8	aktív csatornák (csatornafüggetlen) (*Active voices*)
--	$8F	8	megszakításállapot (*Interrupt status, read only*)

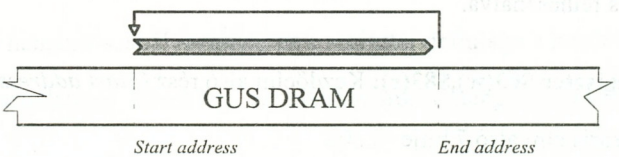
GF1 Regiszter $00(w),$80(r): Hangvezérlő (*Voice control*)

Bitszám	Angol elnevezés
0. bit	*Voice stopped*
1. bit	*Stop voice*
2. bit	*Data size (0=8 bit, 1=16 bit)*
3. bit	*Loop enable/disable*
4. bit	*Bi-directional loop enable/disable*
5. bit	*Wavetable Interrupt enable/disable*
6. bit	*Direction of movement*
7. bit	*Wavetable Interrupt pending*

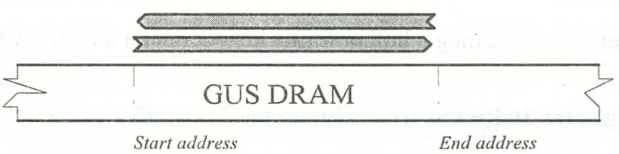
Bit 0: « Leállás. Ez a bit jelzi, ha a csatorna leállt. A bitet a végcím elérésekor állítja be a kártya (ha nem volt engedélyezve a ciklikus hangkiadás), vagy a regiszter 1. bitjének beállítása hatására áll 1-be.

Bit 1: A hangkiadás leállítása kézi úton. Ha ezt a bitet beállítjuk, akkor a lejátszás a csatorna minden más paraméterétől függetlenül azonnal leáll.
Bit 2: Az adatszélesség kiválasztása:
0 – 8-bites adatok
1 – 16 bites adatok
Bit 3: A ciklikus lejátszás engedélyezése. A bit 1-es értékénél a lejátszás a végcím elérésekor nem áll le, hanem újból kezdődik a kezdőcímtől.
Bit 4: Kétirányú ciklikus lejátszás. Ezzel a bittel a kétféle ciklikus lejátszás közül választhatunk:
0 – normál ciklikus lejátszás (ha a 2. bit 1)
1 – kétirányú ciklikus lejátszás (ha a 2. bit 1)

A két ciklikus üzemmódot mutatják a következő ábrák:



51. ábra. Normál ciklikus lejátszás



52. ábra. Kétirányú ciklikus lejátszás

Bit 5: A hullámtábla megszakításának engedélyezése. A csatorna egy megszakítást generál a végcím elérésekor, ha ez a bit 1.
Bit 6: « A lejátszás mozgásának aktuális iránya.
0 – növekvő cím szerint (inkrementális)
1 – csökkenő cím szerint (dekrementális)
Ha a kétirányú ciklikus lejátszás engedélyezve van, akkor a GF1 automatikusan állítja ezt a bitet az irányok megváltoztatásakor.
Bit 7: « Hullámtábla-megszakítás történt. Ha a megszakítást engedélyeztük (6. bit), a ciklikus végrehajtást pedig tiltottuk, akkor a megszakítások állandóan generálódnak, amíg nem állítjuk le a lejátszást.


GF1 Regiszter $01(w),$81(r): Frekvenciavezérlő (*Frequency control*)

A 16 bites regiszter a következő módon van felosztva:
Bit 15...10: egész rész.

Bit 9...1: törtrész.
Bit 0: nem használt.

Ez a regiszter azt határozza meg, hogy mekkora legyen az az érték, amelyet hozzá kell adni (vagy le kell vonni) a hang aktuális pozíciójához, hogy megkaphassuk a következő adat pozícióját. Ha a regiszter értéke 0, akkor a GF1 interpolálja az értéket a két aktuális adatpont között. Ezért ez az érték legyen mindig nagyobb, mint 1. Ezzel engedélyezni lehet az egyes adatbájtok átugrását. Az aktuális lejátszási frekvencia mindig közvetlen kapcsolatban áll az aktív csatornák számával ($0E,$8E regiszterek)!

 **GF1 Regiszter $02(w),$82(r):** Kezdőcím felső rész (*Start address high*)

Bit 12...0: a lineáris cím felső 13 bitje (19...7).
Bit 15..13: nincs felhasználva.

 **GF1 Regiszter $03(w),$83(r):** Kezdőcím alsó rész (*Start address low*)

Bit 15...9: a lineáris cím alsó 7 bitje (6...0).
Bit 8...5: a kezdőpont törtrésze.
Bit 4...0: nincs felhasználva.

Ez a két regiszter határozza meg a hullámforma kezdőcímét a hangkártya memóriájában.

 **GF1 Regiszter $04(w),$84(r):** Végcím felső rész (*End address high*)

Bit 12...0: a lineáris cím felső 13 bitje (19...7).
Bit 15..13: nincs felhasználva.

 **GF1 Regiszter $05(w),$85(r):** Végcím alsó rész (*End address high*)

Bit 15...9: a lineáris cím alsó 7 bitje (6...0).
Bit 8...5: a végpont törtrésze.
Bit 4...0: nincs felhasználva.

Ez a két regiszter határozza meg a hullámforma végcímét a hangkártya memóriájában.

Figyeljünk arra, hogy a végcím önkényes megváltoztatása hanghibát okozhat. Ha például a lejátszás folyamatban van, és a végcímet kisebbre állítjuk, mind az aktuális cím, akkor a hangprocesszor egy megszakítást generál, persze csak akkor, ha ez engedélyezve volt. Ilyenkor korrekten el lehet végezni a szükséges változtatásokat. Ha viszont a megszakítás nincs engedélyezve, akkor a GF1 hibás hangot fog kiadni, és egy kezeletlen megszakítás érkezik!

 **GF1 Regiszter $06(w),$86(r):** Hangerő-változtatási mérték (*Volume ramp rate*)

Bit 5...0: Ez az a 6-bites érték, amelyet az automatikus hangerő-szabályozásnál hozzá kell adni az aktuális hangerőhöz.
Bit 7...6: A változtatási mértéket definiálja ez a két bit.

A regiszter használatáról bővebben az „Automatikus hangerő-szabályozás" pontban írunk majd.

 **GF1 Regiszter $07(w),$87(r):** Hangerő kezdőérték (*Volume ramp start*)

Bit 7...4: exponens rész.
Bit 3...0: mantissza rész.

Az automatikus hangerő-szabályozáshoz ez a regiszter definiálja a kezdeti hangerőt.

A regiszter használatáról bővebben az „Automatikus hangerő-szabályozás" pontban írunk majd.

 **GF1 Regiszter $08(w),$88(r):** Hangerő végérték (*Volume ramp end*)

Bit 7...4: exponens rész.
Bit 3...0: mantissza rész.

Az automatikus hangerő-szabályozáshoz ez a regiszter definiálja a végső hangerőt.

A regiszter használatáról bővebben az „Automatikus hangerő-szabályozás" pontban írunk majd.

 **GF1 Regiszter $09(w),$89(r):** Aktuális hangerő (*Current volume*)

Bit 15...12: « exponens rész.
Bit 11...4: « mantissza rész.
Bit 3...0: fenntartva későbbi fejlesztésre (állítsuk 0-ba!).

Az automatikus hangerő-szabályozásnál ez a regiszter tartalmazza az aktuális hangerőt. A mantissza rész bővítve szerepel, ezért itt finombeállítást lehet végezni.

A regiszter használatáról bővebben az „Automatikus hangerő-szabályozás" pontban írunk majd.

GF1 Regiszter $0A(w),$8A(r): Aktuális cím felső rész (*Current address high*)

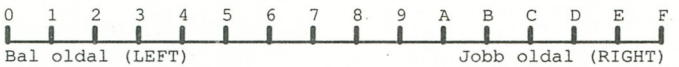
Bit 15..13: fenntartva későbbi fejlesztésre (állítsuk 0-ba!).
Bit 12..0: az aktuális lineáris cím felső 13 bitje (19...7).

GF1 Regiszter $0B(w),$8B(r): Aktuális cím alsó rész (*Current address low*)

Bit 15..9: az aktuális lineáris cím alsó 7 bitje (6...0).
Bit 8..0: az aktuális lejátszási pont törtrésze.

GF1 Regiszter $0C(w),$8C(r): Balanszpozíció (*Pan position*)

Bit 7...4: fenntartva későbbi fejlesztésre (állítsuk 0-ba!)
Bit 3...0: balanszpozíció. Egy csatorna pozícióját 16 lépésben lehet állítani, $00 esetén az aktuális pozíció a bal oldal, $0F esetén pedig a jobb oldal.



53. ábra. Balanszszabályozás 16 lépésben

GF1 Regiszter $0D(w),$8D(r): Hangerővezérlő (*Volume ramp control*)

Bitszám	Angol elnevezés
0. bit	*Ramp stopped*
1. bit	*Stop ramp*
2. bit	*Rollover condition*
3. bit	*Loop enable/disable*
4. bit	*Bi-directional loop enable/disable*
5. bit	*Volume ramp Interrupt enable/disable*
6. bit	*Direction*
7. bit	*Volume ramp Interrupt pending*

Bit 0: « A szabályozás leállt. Ez a bit mutatja, ha a szabályozás valamilyen okból már nem folytatódik tovább.
Bit 1: A szabályozás közvetlen leállítása. Ha ezt a bitet beállítjuk, akkor a hangerő-szabályozás minden más paramétertől függetlenül azonnal leáll (a 0. bit bebillen).
Bit 2: Túlfutási feltétel. (Megszakítás a lejátszás folytatásával.)
Bit 3: A ciklikus hangerő-szabályozás engedélyezése. Ha ez a bit 1, akkor a 4. bit szerinti ciklikus hangerő-szabályozási üzemmód az aktív.
Bit 4: A ciklikus hangerő-szabályozás típusa:
0 – normál ciklikus szabályozás
1 – kétirányú szabályozás.

Bit 5: Megszakításkérés engedélyezése. Ha ez a bit 1, akkor a szabályozás végén egy megszakítás generálódik.
Bit 6: « A szabályozás iránya:
0 – növekvő
1 – csökkenő
Bit 7: « Megszakítás érkezett. Ez a bit jelzi, ha a GF1 hangerő-szabályozás miatt megszakítást kért.

A regiszter használatáról bővebben az „Automatikus hangerő-szabályozás" pontban írunk majd.

GF1 Regiszter $0E(w),$8E(r): Aktív csatornák (*Active voices*)

Bit 7-6: fenntartott, mindig 1-re kell állítani.
Bit 5..0: az aktív csatornák száma (–1).

A kártya alapesetben mindig 14 csatornát tekint aktívnak. Ha ennél többre van szükségünk, akkor ebben a regiszterben kell beállítani a megfelelő csatornaszámot. A kártya figyelmen kívül hagyja a 14-nél kisebb értékeket. Vigyázzunk arra, hogy a számozás 0-val kezdődik, ezért a beírt érték +1 lesz az aktív csatornák száma!

GF1 Regiszter $8F(r): Megszakításállapot (*Interrupt status*)

Bitszám	Angol elnevezés
0. bit	*Interrupt Voice Source*
1. bit	
2. bit	
3. bit	
4. bit	
5. bit	*Reserved (always 1)*
6. bit	*Volume ramp interrupt pending*
7. bit	*Wavetable interrupt pending*

Bit 0..4: a megszakítást kérő csatorna száma (00...31).
Bit 5: fenntartott, mindig 1.
Bit 6: hangerő-szabályozás miatt keletkezett a megszakítás. A bit invertált, azaz a 0 jelzi a megszakítást!
Bit 7: hullámtábla lejátszása miatt keletkezett a megszakítás. A bit invertált, azaz a 0 jelzi a megszakítást!

Ez a regiszter globálisnak számít, mert független a lapregiszter értékétől, és csak olvasható. Előfordulhat, hogy több csatorna látszólag ugyanakkor kér megszakítást. Ha ilyen eset áll elő, akkor ez a regiszter FIFO — elsőként be, elsőként ki (*First In First Out*) — elven működik. Ilyenkor addig kell folyamatosan kiolvasni a regisztert (és persze tárolni a kiolvasott adatokat), amíg a két felső bit egyszerre 1 lesz.

8.4.4. Általános regiszterek


Regiszter $3x4(r/w): GF1 adat alsó bájt (*GF1 data low byte*)


Regiszter $3x5(r/w): GF1 adat felső bájt (*GF1 data high byte*)

Ezt a két regisztert kell használni az adatforgalom lebonyolítására. Ha egy belső regiszter 16 bites, akkor a normál 16 bites I/O utasításokkal a $3x4 címet lehet használni. Ha 8-bites regisztert kell elérni, akkor a $3x5 címet kell használni. Ezt a programozás során majd részletesebben is bemutatjuk.


Regiszter $2x6(r): Általános megszakításállapot (*General interrupt status*)

Bitszám	Angol elnevezés
0. bit	*MIDI transmit interrupt*
1. bit	*MIDI receive interrupt*
2. bit	*Timer #1 interrupt*
3. bit	*Timer #2 interrupt*
4. bit	*Reserved (0)*
5. bit	*Wavetable interrupt (any voice)*
6. bit	*Volume ramp interrupt (any voice)*
7. bit	*DMA transfer interrupt (DRAM or Sample)*

Bit 0: A megszakítást a MIDI adás generálta.
Bit 1: A megszakítást a MIDI vétel generálta.
Bit 2: A megszakítást az első időzítő generálta.
Bit 3: A megszakítást a második időzítő generálta.
Bit 4: Fenntartva későbbi fejlesztésre.
Bit 5: A megszakítást a hullámforma lejátszása generálta (bármelyik csatornán).
Bit 6: A megszakítást a hangerő-szabályozás generálta (bármelyik csatornán).
Bit 7: A megszakítást DMA átvitel váltotta ki (DRAM művelet vagy felvétel).


Regiszter $2x8(r/w): Időzítő állapot (*Timer status*)

Ez a regiszter megfelel az ADLIB státusbájtjának. A 6. bit az első időzítő lejártát, az 5. bit pedig a második időzítő lejártát jelzi.


Regiszter $2x9(w): Időzítővezérlő (*Timer control*)

Bitszám	Angol elnevezés
0. bit	*Timer #1 start*
1. bit	*Timer #2 start*
2. bit	*Reserved (set to 0)*
3. bit	*Reserved (set to 0)*
4. bit	*Reserved (set to 0)*
5. bit	*Timer #2 mask*
6. bit	*Timer #1 mask*
7. bit	*Reset Timer interrupt*

Bit 0: Az első időzítő indítása.
Bit 1: A második időzítő indítása.
Bit 5: A második időzítő maszkolása.
Bit 6: Az első időzítő maszkolása.
Bit 7: Az időzítőmegszakítás törlése.
A vezérlőregiszter bitkiosztása megfelel az ADLIB kártya időzítővezérlőjének. A beépített időzítők a Gravis Ultrasoundnál ugyanúgy működnek, ahogy az ADLIB-nél.

Regiszter $3x7(w): GUS DRAM írás/olvasás (*DRAM read/write*)

A kártya memóriájának közvetlen elérésére szolgál ez a regiszter. A címregiszterek ($43,$44) beállítása után innen lehet kiolvasni az adott DRAM bájt értékét, és itt lehet beírni az új értéket is.

Regiszter $2x0(w): Keverővezérlő (*Mixer control*)

Bitszám	Angol elnevezés
0. bit	*Enable/disable Line In*
1. bit	*Enable/disable Line Out*
2. bit	*Enable/disable Microphone*
3. bit	*Enable latches*
4. bit	*Combine channel #1 IRQ with channel #2*
5. bit	*Enable MIDI loopback TxD to RxD*
6. bit	*Control register select*
7. bit	*Reserved (set to 0)*

Bit 0: Vezérli a Line In bemenetet.
0 – Line In bekapcsolt
1 – Line In kikapcsolt
Bit 1: Vezérli a Line Out kimenetet.
0 – Line Out bekapcsolt
1 – Line Out kikapcsolt
Bit 2: Vezérli a mikrofonbemenetet.
0 – a mikrofon kikapcsolt
1 – a mikrofon bekapcsolt

Bit 3: Engedélyezi az átmeneti tárolók használatát. A DMA és az IRQ átmeneti tárolókat ezzel a bittel lehet bekapcsolni. Fontos, hogy ha egyszer már bekapcsoltuk a tárolókat, akkor nem szabad őket kikapcsolni. Ezt a hardvernek kell megtennie!
Bit 4: A megszakítási csatornák kombinálása: GF1 és MIDI.
Bit 5: A MIDI hurok engedélyezése. Ha ez a bit 1, akkor az elküldött adat visszaérkezik a vevő oldalon.
Bit 6: Vezérlőregiszterek kiválasztása. Ha ez a bit 1-es, akkor a következő I/O művelet, amely a 2xB portot írja, a megszakításvezérlőt választja ki, egyébként pedig a DMA vezérlőt.

Regiszter $2xB(w): Megszakításvezérlő (*Interrupt control*)

Ezzel a regiszterrel a megszakítási szinteket lehet kiválasztani.

Bitszám	Angol elnevezés
0. bit	*Channel #1 (GF1) interrupt level select*
1. bit	
2. bit	
3. bit	*Channel #2 (MIDI) interrupt level select*
4. bit	
5. bit	
6. bit	*Combine both device using channel #1*
7. bit	*Reserved (set to 0)*

Bit 2...0: A GF1 megszakítási vonal kiválasztása:
000 – fenntartott, soha ne használjuk!
001 – IRQ 2
010 – IRQ 5
011 – IRQ 3
100 – IRQ 7
101 – IRQ 11
110 – IRQ 12
111 – IRQ 15
Bit 5...3: A MIDI megszakítási vonal kiválasztása:
000 – nincs megszakítás
001 – IRQ 2
010 – IRQ 5
011 – IRQ 3
100 – IRQ 7
101 – IRQ 11
110 – IRQ 12
111 – IRQ 15
Bit 6: A megszakítási vonalak kombinálása, mindkét egység az első beállítást használja, ha ez a bit 1.
Bit 7: fenntartott.

Ezt a regisztert csak akkor tudjuk elérni, ha a $2x0 regiszter 6. bitje 1!

Regiszter $2xB(w): DMA vezérlő (*DMA control*)

Ezzel a regiszterrel a DMA csatornákat lehet kiválasztani.

Bitszám	Angol elnevezés
0. bit	*DMA select channel #1*
1. bit	
2. bit	
3. bit	*DMA select channel #2*
4. bit	
5. bit	
6. bit	*Combine both device using channel #1*
7. bit	*Reserved (set to 0)*

Bit 2...0: Az első DMA csatorna kiválasztása:
000 – nincs DMA átvitel
001 – DMA 1
010 – DMA 3
011 – DMA 5
100 – DMA 6
101 – DMA 7
110 – nincs jelentése
111 – nincs jelentése
Bit 5...3: A második DMA csatorna kiválasztása
000 – nincs DMA átvitel
001 – DMA 1
010 – DMA 3
011 – DMA 5
100 – DMA 6
101 – DMA 7
110 – nincs jelentése
111 – nincs jelentése
Bit 6: Csak egyetlen DMA csatorna használata. Ha ez a bit 1, akkor a felvétel és a lejátszás ugyanazt a DMA csatornát használja.
Bit 7: fenntartott.

Ezt a regisztert csak akkor tudjuk elérni, ha a $2x0 regiszter 6. bitje 0!

Regiszter $2xF(r/w): Regisztervezérlő (*Register control*)

Bitszám	Angol elnevezés
0. bit	*Reserved (set to 0)*
1. bit	*Enable MIDI port address decode*
2. bit	*Enable JOYSTICK port address decode*
3. bit	*Reserved (set to 0)*
4. bit	*Reserved (set to 0)*
5. bit	*Reserved (set to 0)*
6. bit	*Reserved (set to 0)*
7. bit	*Reserved (set to 0)*

Ez a regiszter a kiegészítő egységek — a MIDI és a joystick — engedélyezésére szolgál. Ha az adott eszközt használni akarjuk, akkor a megfelelő bitet 1-be kell állítani.

Bit 1: a MIDI engedélyezése.
Bit 2: a joystick engedélyezése.
Bit 0,3...7: fenntartva fejlesztésre.

Regiszter $7x6(w): Keverővezérlő (*Mixer control*)

Ez a regiszter a keverőegység vezérlőregisztere.

Regiszter $3x6(w): Keverőadat (*Mixer data*)

Ez a regiszter a keverőegység adatregisztere.

Regiszter $7x6(r): Hangkártya verziószám (*Board version*)

A 3.7-es vagy ennél nagyobb verziószámú kártyáknál a verziószám szoftverből lekérdezhető ezzel a regiszterrel. A regiszter egy jellemző kódot ad vissza minden verzióhoz az alábbi táblázat szerint:

Verziószám-azonosító	Jelentés
$FF	3.7-nél kisebb verzió (nincs ICS keverő)
5	3.7-es verzió, ICS keverővel
6...9	3.7 vagy annál nagyobb verzió ICS keverővel
$0A...??	UltraMAX, CS4231 CODEC

A CS4231 az UltraMAX kártyákon található digitális jelprocesszor, amely magában foglalja a keverőegységet is.

Regiszter $3x6(w): UltraMAX vezérlő (*UltraMAX control*)

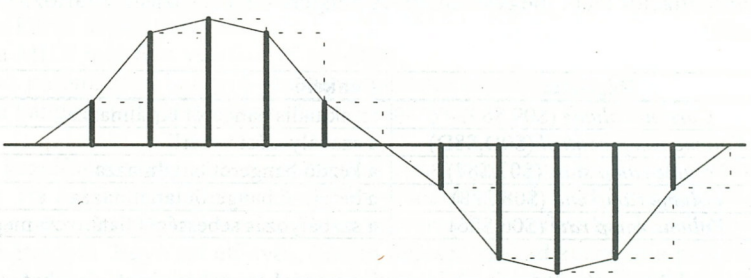
Bitszám	Angol elnevezés
0. bit	*Codec address decode bit 4*
1. bit	*Codec address decode bit 5*
2. bit	*Codec address decode bit 6*
3. bit	*Codec address decode bit 7*
4. bit	*Record channel type*
5. bit	*Play channel type*
6. bit	*Codec enable/disable*
7. bit	*Reserved (set to 0)*

Ez a regiszter csak az UltraMAX kártyákon található meg!

Bit 0...3: A codec címének középső négy bitjét definiálja, azaz az *X* értéket a $3*X*C portcímben.
Bit 4: A felvételhez használt DMA csatorna típusát adja meg:
0 – 8-bites DMA csatorna
1 – 16 bites DMA csatorna
Bit 5: A lejátszáshoz használható DMA csatorna típusát adja meg:
0 – 8-bites DMA csatorna
1 – 16 bites DMA csatorna
Bit 6: a codec chipet engedélyezi.
Bit 7: fenntartva.

8.5. Törtszámok

Amint a regiszterkészlet leírásában láthattuk, több helyen is előfordul, hogy bizonyos értékeket fixpontos számábrázolással kell megadni. De mi értelme van törtszámokat használni például a címek megadásánál, amikor a hangminták csakis pontosan egész számú címekre eshetnek? Első látásra talán tényleg nem látszik túl ésszerűnek a megoldás, pedig ez a GF1-nek egyik igen kényelmes szolgáltatása. Az egyes hangminták közötti helyet a hangprocesszor ugyanis nem vízszintes egyenes jelszakaszokkal köti össze, hanem interpolálja (azaz a két végpont alapján közelítéssel meghatározza a közbenső értékeket is). Ez akkor lehet hasznos, ha egy jelből csak ritkán veszünk mintát. Nézzük meg a következő, szinuszhoz hasonló jelet!



54. ábra. Lineáris interpoláció a minták között

Az ábrán a vastag függőleges jelek az ismert mintavételi pontok. A pontozott vonal lenne a jel alakja, ha nem lenne interpoláció. Ez egy csúnya szögletes jel. A vékony vonal azt a jelalakot mutatja, amikor ismerjük a két mintavételi pontot, és közöttük egy egyenessel helyettesítjük a jelet. Ugyanezt teszi a hangprocesszor is. Az interpolált jel kevesebb felharmonikust tartalmaz, mint a másik, és természetesen sokkal jobban hasonlít is az eredeti jel alakjához. Amint láthatjuk, értelme van például annak, hogy két diszkrét mintavételi pont között egy törtszámmal megadjuk, hogy honnan kellene kezdeni a jel kiadását.

Ugyanez a kényelmes lehetőség megvan a mintavételi frekvencia megadásánál is. Ott is lehetséges olyan mintavétel, ahol két meglévő minta közül több olyan mintára is szükség lenne, amelyek nem is léteznek.

Felmerülhet még egy fontos kérdés: honnan tudjuk, hogy a következő minta kisebb vagy nagyobb lesz-e, mint az előző. A hang rögzítésénél ezt nem lehet pontosan tudni, csupán statisztikai módszerekkel megjósolhatjuk a jel változásait. A lejátszásnál viszont egyszerű a helyzet, hiszen ilyenkor a hangminta már a GUS DRAM-ban van. A GF1 tehát megteheti azt, hogy mindig egy lépéssel késlelteti a lejátszást, azaz mindig az utolsó és az utolsó előtti hangminta alapján interpolál, és ez jelenik meg a kimeneten. Még nagyon alacsony mintavételi frekvencián is — mondjuk 5 kHz-en — ez csupán 0.2 ms-os késleltetést eredményez a jelben. Ezt pedig nehéz füllel érzékelni!

8.6. Automatikus hangerő-szabályozás

A burkológörbék rugalmas előállításához találták ki az automatikus hangerő-szabályozás módszerét. A szabályozáshoz meg lehet adni egy kezdőpontot, ahonnan a hangerő indul, és egy végpontot, ahová a hangerőnek el kell jutnia. A szabályozás sebessége szintén regiszterrel definiálható. Kissé kényelmetlen megoldás, hogy a hangerő regiszterek nem egész számokkal, hanem lebegőpontos számábrázolással dolgoznak. Ráadásul a végpontokat definiáló regiszterek kisebb pontossággal dolgoznak, mint az aktuális hangerőt tartalmazó regiszter. A szabályozásnál lehetőség van ciklikus szabályozásra is, ezzel amplitúdómodulációt lehet megvalósítani. A hangerő-szabályozáshoz tartozó regiszterek a következők:

Regiszter	Funkció
Current volume ($09,$89)	az aktuális hangerőt tartalmazza
Volume ramp control ($0D,$8D)	a szabályozást vezérli
Volume ramp start ($07,$87)	a kezdő hangerőt tartalmazza
Volume ramp end ($08,$88)	a befejező hangerőt tartalmazza
Volume ramp rate ($06,$86)	a szabályozás sebességét határozza meg

A törtszámos alakban tárolt hangerő a programozáskor nehézséget okozhat. (A később ismertetésre kerülő GUS unitban van egy táblázat, amely lineáris szabályozást tesz lehetővé.) Mindenesetre meg kell adni a kezdő és a véghangerőt, amelyek között szabályozni akarunk. A kezdő hangerőnek mindig kisebbnek kell lenni a véghangerőnél. Ha csökkenteni akarjuk a hangerőt, akkor csökkenő irányt kell megadnunk a vezérlő-regiszter megfelelő bitjével. A szabályozás sebességét a *Volume ramp rate* regiszter adja meg. Ez két mezőt tartalmaz. A felső két bit egy osztóértéket határoz meg, az alsó hat pedig a növelés/csökkentés mértékét. Az aktív csatornák számától függően az idő-intervallumok különböznek.

Osztó mező	Egység	14 csatorna	32 csatorna
00	63	1.4 ms	3.3 ms
00	1	91.7 ms	209.7 ms
01	63	11.5 ms	26.2 ms
01	1	733.8 ms	1.7 ms
10	63	91.8 ms	209.7 ms
10	1	5.9 ms	13.4 ms
11	63	734.0 ms	1.7 ms
11	1	47.0 ms	107.3 ms

Ha a vezérlőregiszter megfelelő bitjét beállítjuk, akkor a szabályozás végén — amikor az aktuális hangerő elérte a véghangerőt — a GF1 egy megszakítást generál. A megszakítás beprogramozza az újabb szabályozást. Ezzel a módszerrel tetszőleges szakaszból álló burkológörbét tudunk létrehozni.

8.7. „Megszakadok . . . ”

Ha a GF1 hangprocesszornak volna véleménye, és ezt el is mondhatná, akkor valami ilyesmit hallanánk: „ha megszakadok, akkor sem bírok többet ennél...”. :~) Amint azt bizonyára mindenki észrevette, a Gravis Ultrasound hangkártyák szinte minden fontos feladatot a megszakításokra hárítanak át. Mely esemény generálhat megszakítást?

– A DMA átvitel a PC és a Gravis memóriája között;
– a DMA átvitel felvételkor;
– a MIDI interfész vételkor és adáskor;
– az automatikus hangerő-szabályozás elemi eseményei;
– a hangminta lejátszásának elemi eseményei.

Mindezek kezelésére csupán két megszakítási vonal van, egy a MIDI interfésznek és egy a GF1-nek. Ha összetett feladatot akarunk megoldani, akkor a GF1 megszakítási rutinjának nagyon sok mindent kell tennie. Ez az univerzális „szoftverből kezelhetőség” meglehetősen kétoldalú. Egyrészt előnyös, hiszen teljesen szabad kezet ad a programozónak, másrészt viszont hátrányos, mert nagyon bonyolult, hosszú megszakítási rutint kell készíteni.

Különösen időkritikus a védett módú programokban a megszakítás. A 80386/80486 védett (*protected*) módjában a megszakításokat ún. megszakítási kapukon (*interrupt-gate*) vagy taszk kapukon (*task-gate*) keresztül kell meghívni. A kapu sok védelmi feladatot láthat el (ezért védett mód), de ez időt vesz igénybe. Ezért azt javasoljuk, hogy a megszakítási rutint próbálja mindenki nagyon gyorsra készíteni, nem számít, ha egy kicsivel több helyet igényel!

8.8. A GUS unit

A kártya regisztereinek ismerete nem elégséges a programozáshoz, mert sok olyan szabály van, amelyet be kell tartani, illetve ezek miatt különleges módszereket kell alkalmazni a regiszterek eléréséhez. Ezért — szokásunkhoz híven — a Gravishoz is készítettünk egy unitot, amely tartalmazza az alaprutinokat. A forrás egy kicsit terjedelmesebb lesz, mint az előző kártyáknál, mégis javasoljuk a részletes áttanulmányozását, különben a kártya még sok meglepetést fog okozni!

```
{$G+} { Legalább 80286-os processzor szükséges! }
unit gus; { gus.pas }

{ Használjuk a DOS és a CRT unitokat! }
interface uses DOS,CRT;

{ Definiálunk egy memória és egy memóriamutató típust }
type
   memptr = ^memory;
   memory = array[ 0..65534] of byte;

const
   Hdg : array[ 0..15] of char='0123456789ABCDEF';

   { Hangkártya hardverbeállítások  }
   Base     : word = $220;      { Gravis I/O báziscím          }
   GF1irq   : byte = 11;        { GF1 megszakítási vonal       }
   MIDIirq  : byte =  7;        { MIDI megszakítási vonal      }
   DMA_1    : byte =  3;        { DRAM DMA csatorna            }
   DMA_2    : byte =  3;        { Felvétel DMA csatorna        }

   { Frekvenciatáblázat 5 oktáv számára                        }

   FreqTAB : array[ 0..59] of word=(
      $0038,$003B,$003E,$0042,$0046,$004A,      { 1. oktáv  }
      $004F,$0053,$0058,$005E,$0063,$0069,
      $0070,$0076,$007D,$0085,$008D,$0095,      { 2. oktáv  }
      $009E,$00A7,$00B1,$00BC,$00C7,$00D3,
      $00E0,$00ED,$00FB,$010A,$011A,$012B,      { 3. oktáv  }
      $013D,$014F,$0163,$0179,$018F,$01A7,
      $01C0,$01DB,$01F7,$0214,$0234,$0256,      { 4. oktáv  }
      $027A,$029F,$02C7,$02F2,$031E,$034E,
      $0380,$03B6,$03EE,$0429,$0469,$04AD,      { 5. oktáv  }
      $04F4,$053F,$058F,$05E4,$063D,$069C);

   { Hangerőtáblázat a 128 lépéses lineáris szabályozáshoz  }
```

```
VolTAB: array[ 0..127] of word=(
     $0000,$8ff0,$9ff0,$a800,$aff0,$b400,$b800,$bc00,
     $bff0,$c200,$c400,$c600,$c800,$ca00,$cc00,$ce00,
     $cff0,$d100,$d200,$d300,$d400,$d500,$d600,$d700,
     $d800,$d900,$da00,$db00,$dc00,$dd00,$de00,$df00,
     $dff0,$e080,$e100,$e170,$e200,$e280,$e300,$e370,
     $e400,$e480,$e500,$e580,$e600,$e680,$e700,$e780,
     $e800,$e880,$e900,$e990,$ea00,$ea80,$eb00,$eb80,
     $ec00,$ec80,$ed00,$ed90,$ee00,$ee80,$ef00,$ef90,
     $eff0,$f030,$f070,$f0c0,$f100,$f140,$f170,$f1c0,
     $f200,$f240,$f280,$f2b0,$f300,$f340,$f380,$f3b0,
     $f400,$f440,$f470,$f4c0,$f500,$f540,$f590,$f5c0,
     $f600,$f650,$f690,$f6c0,$f700,$f740,$f790,$f7c0,
     $f800,$f840,$f880,$f8b0,$f900,$f940,$f980,$f9b0,
     $fa00,$fa50,$fa80,$fac0,$fb00,$fb40,$fb80,$fbc0,
     $fc00,$fc40,$fc80,$fcd0,$fd00,$fd40,$fd80,$fdb0,
     $fe00,$fe40,$fe80,$fec0,$ff00,$ff40,$ff80,$ffd0);
```

Állandók

A unitban sok nélkülözhetetlen, kényelmi szempontokat szolgáló állandó található.

Hdg: A hexadecimális kiíráshoz használjuk ezt a tömböt. A unitban van két eljárás (*Hex* és *HexW*), amelyekkel egy bájt, illetve egy word értékét lehet átalakítani hexadecimális szöveges formába. A hangkártyához erre nincs szükség, de egyéb esetekben hasznos lehet, amint azt a későbbiekben látni fogjuk.

Base, GF1irq, MIDIirq, DMA_1, DMA_2: A kártya alapvető hardverbeállításait tartalmazzák ezek a változók. A kifejezés egyébként egy kicsit sántít, mert az összes jellemzőt — a báziscím kivételével — szoftverből lehet beállítani. A DMA csatornákat nem fogjuk használni, a GF1 megszakítási vonalát pedig mindig 11-re állítjuk. Hogy ez miért jó, arra majd később fény derül.

FreqTAB: A GUS DRAM-ban tárolt hangminták visszajátszásához definiáltuk ezt a tömböt, amely öt oktáv számára tartalmazza a megfelelő frekvenciakonstansokat. Mindez egy kicsit öncélú, hiszen egy tárolt hangminta visszajátszása eléggé szubjektív. (Elöljáróban csak annyit, hogy a MOD fájlok lejátszásánál lehet ezt jól felhasználni.)

VolTAB: A lebegőpontos hangerő nem szép. :-(A lineáris hangerő-szabályozáshoz készítettük ezt a táblázatot, amely — egy logaritmikus algoritmus szerint számított — 128 lépéses lineáris szabályozást tesz lehetővé.

```
{ Eljárások és függvények }
function  GetByte(Address:longint):byte;
procedure PutByte(Address:longint; value:byte);
procedure MoveSample(Adr:longint; var Src; count:word);
procedure GravisWait;
procedure GravisRESET;
function  GravisTEST:boolean;
procedure SetGF1Byte(Ch,Reg,Value:byte);
procedure SetGF1Word(Ch,Reg:byte; Value:word);
function  GetGF1Byte(Ch,Reg:byte):byte;
function  GetGF1Word(Ch,Reg:byte):word;
function  GravisFindBase:word;
function  GravisMemory:word;
procedure GravisVolume(ch:byte; volume:word);
procedure GravisBalance(ch:byte; balance:byte);
procedure GravisFreq(ch:byte; freq:word);
procedure PlayVoice(ch,mode:byte; b,s,e:longint);
procedure StopVoice(ch:byte);
function  Hex(b:byte):string;
function  HexW(w:word):string;
procedure Note(ch,n:byte);
procedure Volume(ch,n:byte);
procedure SetIRQservice(Irqr:pointer);
procedure ResetIRQservice;

implementation
```

Eljárások, függvények

```
function  GetByte(Address:longint):byte;        assembler;
asm
  mov   dx,Base
  add   dx,$103                 { Globálisregiszter-kiválasztás  }
  mov   al,$43
  out   dx,al                   { A DRAM alsó szó regiszter      }
  inc   dx
  mov   ax,Address.word
  out   dx,ax                   { Kiírjuk a cím alsó 16 bitjét   }
  dec   dx
  mov   al,$44                  { A DRAM felső bájt regiszter    }
  out   dx,al
  add   dx,$0002
  mov   al,[Address+2].byte
  and   al,00001111b
  out   dx,al                   { Kiírjuk a cím felső 4 bitjét   }
  add   dx,$0002                { DX: $3x7, DRAM read/write      }
  in    al,dx                   { Beolvassuk az adatot az AL-be  }
end;
```

GetByte: A függvénnyel a GUS DRAM memória tetszőleges bájtjának értékét lehet lekérdezni. Ha olyan bájtot kérdezünk le, amely fizikailag nincs meg a kártyán, akkor a függvény természetesen hibás eredményt fog visszaadni. (Ezt a tulajdonságot egyébként később felhasználjuk a báziscím detektálására.)

```
procedure PutByte(Address:longint; value:byte); assembler;
asm
  mov   dx,Base
  add   dx,$103                 { Globálisregiszter-kiválasztás  }
  mov   al,$43
  out   dx,al                   { A DRAM alsó szó regiszter      }
  inc   dx
  mov   ax,Address.word
  out   dx,ax                   { Kiírjuk a cím alsó 16 bitjét   }
  dec   dx
  mov   al,$44                  { A DRAM felső bájt regiszter    }
  out   dx,al
  add   dx,$0002
  mov   al,[Address+2].byte
  and   al,00001111b
  out   dx,al                   { Kiírjuk a cím felső 4 bitjét   }
  add   dx,$0002                { DX: $3x7 DRAM read/write       }
  mov   al,value                { AL: a bájt új értéke           }
  out   dx,al                   { Beírjuk az értéket a DRAM-ba   }
end;
```

PutByte: Egy GUS DRAM bájt beírása. Ezzel az eljárással a kártya memóriájának egy bájtját tudjuk módosítani. Ha olyan helyre írunk, amely fizikailag nem létezik, akkor természetesen nem történik módosítás (vagy ami a rosszabb eset, egy jó adat módosítására kerül sor).

```
procedure MoveSample(Adr:longint; var Src; count:word); assembler;
asm
  mov   di,Adr.word;
  mov   bl,[Adr+2].byte
  and   bl,00001111b            { BL:DI = DRAM cím (első bájt)   }
  cld
  les   si,Src
  mov   ax,ES:[si]
  mov   ES,ES:[si+2]
  mov   si,ax                   { ES:SI a forrásadatra mutat     }
  mov   cx,count                { CX a bájtok számát tartalmazza }
@putbytes:
  { A kiírás a PutByte eljáráshoz hasonlóan történik             }
  mov   dx,Base
  add   dx,$103
  mov   al,$43
  out   dx,al
  inc   dx
```

```
  mov   ax,di
  out   dx,ax
  dec   dx
  mov   al,$44
  out   dx,al
  add   dx,$0002
  mov   al,bl
  out   dx,al
  add   dx,$0002
  segES lodsb                  { Beolvasás a forrásadatokból     }
  out   dx,al                  { Kiírás a célhelyre (GUS DRAM)   }
  add   di,$01
  adc   bl,$00                 { Növeljük a DRAM címet (BL:DI)   }
  loop  @putbytes              { CX darab bájtot írunk ki        }
end;
```

MoveSample: A *PutByte* eljáráshoz hasonlóan ez is a GUS DRAM közvetlen írását teszi lehetővé, de ezzel nem egy bájtot, hanem egy (legfeljebb 64K hosszú) memóriaterületet lehet betölteni a DRAM-ba. Ez a módszer valamivel gyorsabb, mintha ciklussal a *PutByte*-ot használnánk.

```
procedure GravisWait;                                  assembler;
asm
  push  dx                     { Tároljuk a DX,AX regisztereket  }
  push  ax
  mov   dx,$0000               { Egy >veszélytelen< I/O cím      }
  in    al,dx
  in    al,dx
  in    al,dx
  in    al,dx
  in    al,dx
  in    al,dx                  { Várakozási ciklusok             }
  pop   ax
  pop   dx                     { AX,DX vissza!                   }
end;
```

GravisWait: Várakozás. Ez az eljárás bizonyos buszciklus ideig várakozik. Erre azért van szükség, mert a GF1 *pipeline* rendszere miatt néhol bizonyos időre van szükség. A felhasznált I/O cím „veszélytelen", azaz a kiolvasása nem jár vészes következményekkel.

```
procedure GravisRESET;                                 assembler;
asm
  mov   dx,Base
  add   dx,$103                { Globálisregiszter-kiválasztás   }
  mov   al,$4C
  out   dx,al                  { $4C: RESET regiszter            }
  add   dx,$0002
  mov   al,$00                 { A GF1 inicializálása            }
  out   dx,al
```

```
  call   GravisWait              { Várakozás                          }
  sub    dx,$0002
  mov    al,$4C
  out    dx,al                   { Ismét a RESET regiszter kell       }
  add    dx,$0002
  mov    al,$07                  { Visszaállítjuk az eredeti          }
  out    dx,al                   { normál működési állapotot          }
  sub    dx,$0002
  mov    al,$0E
  out    dx,al                   { 14 aktív csatorna                  }
  mov    al,$CE
  add    dx,$0002
  out    dx,al
  mov    dx,Base
  mov    al,$0C
  out    dx,al                   { Engedélyezzük a bemeneteket        }
end;
```

GravisReset: A hangkártya inicializálása. Ezzel az eljárással alapállapotba lehet hozni a kártyát. A rutin végrehajtja a RESET metódust, az aktív csatornák számát 14-re állítja be, és engedélyezi a bemeneteket (*Line In, MIC*).

```
function  GravisTEST:boolean;
begin
  asm
    mov    dx,Base
    add    dx,$103
    mov    al,$4C                  { Megpróbáljuk inicializálni a       }
    out    dx,al                   { hangkártyát                        }
    add    dx,$0002
    xor    al,al
    out    dx,al
    call   GravisWait
    call   GravisWait
    sub    dx,$0002
    mov    al,$4C
    out    dx,al
    add    dx,$0002
    mov    al,$01
    out    dx,al
  end;
 PutByte(256,$55);               { Kiírunk néhány bájtot a DRAM-ba    }
 PutByte(512,$aa);
 PutByte(000,$0f);
 { Ha ugyanazt olvassuk vissza, mint amit kiírtunk, akkor a           }
 { DRAM jól működik, tehát van GRAVIS a gépben                        }
 GravisTest:=(GetByte(256)=$55)and(GetByte(512)=$aa);
end;
```

GravisTest: A hangkártya tesztelése. Ez a függvény igaz értékkel tér vissza akkor, ha az adott báziscímen érzékelte a Gravis hangkártyát. A tesztelés nagyon egyszerű. Egy RESET metódus után a függvény adatokat ír a *PutByte* eljárással a DRAM különböző helyeire. Ha a *GetByte* ugyanazokat az értékeket olvassa vissza, mint amelyet a *PutByte* kiírt, akkor a DRAM jól működik, azaz van hangkártya. Ezt a függvényt a későbbi *GravisFindBase* függvény felhasználja a báziscím detektálására.

```
procedure SetGF1Byte(Ch,Reg,Value:byte);                    assembler;
asm
  mov   dx,Base
  add   dx,$102                 { Csatorna lapregiszter ($3x2)   }
  mov   al,&Ch                  { Kiírjuk a csatornaszámot       }
  out   dx,al                   { többször is ...                }
  out   dx,al
  out   dx,al
  inc   dx                      { Regiszterkiválasztás ($3x3)    }
  mov   al,Reg
  out   dx,al                   { Kiírjuk a regiszter számát     }
  add   dx,$0002                { Regiszter adat I/O cím ($3x5)  }
  mov   al,Value
  out   dx,al                   { Kiírjuk a regiszter új értékét }
end;

procedure SetGF1Word(Ch,Reg:byte; Value:word);              assembler;
asm
  mov   dx,Base
  add   dx,$102                 { Csatorna lapregiszter ($3x2)   }
  mov   al,&Ch                  { Kiírjuk a csatornaszámot       }
  out   dx,al                   { többször is ...                }
  out   dx,al
  out   dx,al
  inc   dx                      { Regiszterkiválasztás ($3x3)    }
  mov   al,Reg
  out   dx,al                   { Kiírjuk a regiszter számát     }
  inc   dx                      { Regiszter word adat cím ($3x4) }
  mov   ax,Value
  out   dx,ax                   { Kiírjuk a word adatot          }
end;
```

SetGF1byte, SetGF1word: A GF1 regiszterek közvetlen módosítása. Ezekkel az eljárásokkal a GF1 globális és csatornaspecifikus regisztereit tudjuk elérni. Az eljárásokban látható a *pipeline* rendszer kellemetlen hatása. A lapregiszter beírását többször is meg kell ismételni, mert a GF1 folyamatosan dolgozik, és a biztonságos módosításhoz ez a többszörös írás szükséges. Ez bizony nem szép megoldás, de legalább működik.

```
function  GetGF1Byte(Ch,Reg:byte):byte;                     assembler;
asm
  mov   dx,Base
  add   dx,$102                 { Csatorna lapregiszter ($3x2)   }
```

```
  mov   al,&Ch                  { Kiírjuk a csatornaszámot        }
  out   dx,al                   { többször is ...                 }
  out   dx,al
  out   dx,al
  inc   dx                      { Regiszterkiválasztás ($3x3)     }
  mov   al,Reg
  out   dx,al                   { Kiírjuk a regiszter számát      }
  add   dx,$0002                { Regiszter adat I/O cím ($3x5)   }
  in    al,dx                   { Beolvassuk a regiszter értékét  }
end;

function  GetGF1Word(Ch,Reg:byte):word;                    assembler;
asm
  mov   dx,Base                 { Ugyanúgy, mint az előzőeknél... }
  add   dx,$102
  mov   al,&Ch
  out   dx,al
  out   dx,al
  out   dx,al
  inc   dx
  mov   al,Reg
  out   dx,al
  inc   dx
  in    ax,dx                   { Beolvassuk a regiszter értékét  }
end;
```

GetGF1byte, GetGF1word: A GF1 regisztereinek kiolvasása. Ez a két függvény az előző két eljárás fordítottja. A globális és a csatornaspecifikus GF1 regisztereket lehet lekérdezni velük (persze csak azokat, amelyek olvashatók).

```
function  GravisFindBase:word;
var i:byte;
    w:word;
begin
 w:=0;
 for i:=1 to 6 do
  begin
   Base:=$200 or i*16;
   if GravisTEST then w:=Base;
  end;
 GravisFindBase:=w;
 Base:=w;
end;
```

GravisFindBase: A hangkártya báziscímének meghatározása. Ez a függvény sorra beállítja a lehetséges I/O bázisértékeket, és a *GravisTest* függvénnyel ellenőrzi, hogy az aktuális beállítás helyes-e. Ha igen, akkor a *Base* változót beállítja, és ez lesz a függvény visszatérési értéke is. Ha $0000, akkor nem talált Gravis regisztereket semelyik címen (azaz valószínűleg nincs is Gravis hangkártya a gépben).

```
function  GravisMemory:word;
var b:word;
begin
 PutByte($3ffff,$01); { 256*1024-1 }
 if GetByte($3ffff)=$01 then b:=256;
 PutByte($7ffff,$02); { 512*1024-1 }
 if GetByte($7ffff)=$02 then b:=512;
 PutByte($bffff,$04); { 768*1024-1 }
 if GetByte($bffff)=$04 then b:=768;
 PutByte($fffff,$08); { 1024*1024-1 }
 if GetByte($fffff)=$08 then b:=1024;
 GravisMemory:=b;
end;
```

GravisMemory: A kártya memóriájának meghatározása. Ez a függvény megállapítja, hogy mennyi memória van a hangkártyán. A visszaadott érték egy word, amelynek a következő kódjai vannak:

Visszaadott érték	GUS DRAM memória
256	256 K
512	512 K
768	768 K
1024	1 M
bármi más	nincs memória

A lekérdezés igen egyszerű: megpróbálunk minden 256K memóriaszelet utolsó bájtjába írni, és ha ez sikerül, akkor az adott szelet biztosan megvan,

```
procedure GravisVolume(ch:byte; volume:word);
begin
 SetGF1Word(ch,$09,volume);
end;
```

GravisVolume: A hangerő direkt módosítása. Az eljárással a *ch* csatorna hangerejét lehet beállítani a *volume* értékre. A *volume* paraméter a regiszterkészletnél ismertetett módon egy törtszámot tartalmaz. Ha lineárisan akarjuk beállítani a hangerőt, akkor használjuk a *Volume* eljárást!

```
procedure GravisBalance(ch:byte; balance:byte);
begin
 SetGF1Byte(ch,$0C,balance);
end;
```

GravisBalance: Egy csatorna balanszértékének beállítása. Az eljárással a *ch* számú csatorna jobb/bal eltolását lehet megadni a *balance* paraméterrel. A szabályozás 16 lépésben történhet, a $00 a teljes bal a $0F a teljes jobb oldalt jelenti.

```
procedure GravisFreq(ch:byte; freq:word);
begin
 SetGF1Word(ch,$01,freq);
end;
```

GravisFreq: A lejátszási frekvencia beállítása. A *ch* számú csatorna lejátszási mintavételezési frekvenciáját adhatjuk meg a *freq* paraméterrel. Ezzel az eljárással közvetlenül, a regiszterkészletnél ismertetett módon lehet megadni a frekvenciát. Ha szabályos — oktávon belüli — hangokat akarunk kiadni, akkor használjuk a *Note* eljárást.

```
procedure PlayVoice(ch,mode:byte; b,s,e:longint);
begin
 SetGF1Word(Ch,$03,(s and $7f) shl 9);
 SetGF1Word(Ch,$02,(s shr 7));
 SetGF1Word(Ch,$05,(e and $7f) shl 9);
 SetGF1Word(Ch,$04,(e shr 7));
 SetGF1Word(Ch,$0B,(b and $7f) shl 9);
 SetGF1Word(Ch,$0A,(b shr 7));
 SetGF1Byte(Ch,$00,mode);
end;
```

PlayVoice: Egy GUS DRAM-ba töltött hangminta lejátszása. Az eljárás paraméterei a következők:

ch: a lejátszáshoz felhasznált csatorna száma;
mode: a lejátszás módja;
b: a hangminta aktuális címe;
s: a hangminta kezdőcíme;
e: a hangminta végcíme.

A *mode* határozza meg a lejátszás módját, ez az érték közvetlenül a *Voice control* regiszterbe kerül (ciklikus lejátszás, megszakítás...). A *b* a lejátszás kezdőpontját adja, az *s* és az *e* pedig a hangminta kezdő- és végcímét definiálják. A *b* és az *s* nem szükség-szerűen egyenlőek, hiszen nem fontos a hangot a legelején kezdeni. Ugyanígy eltérés lehet a ciklikus lejátszásnál is. Ez az eljárás önmagában még nem elegendő a hang kiadásához, hiszen először be kell tölteni a hangmintát, és be kell állítani az egyéb paramétereket is (frekvencia, hangerő...).

```
procedure StopVoice(ch:byte);                    assembler;
asm
  push  &ch.word
  push  $0080                  { Lekérdezzük a Voice control      }
  call  GetGF1Byte             { regiszter értékét                }
  and   al,$DF
  or    al,$03                 { Voice Stop bit=1                 }
  xor   ah,ah
  push  ax
  push  &ch.word
  push  $0000
```

```
  push  ax
  call  SetGF1Byte              { Beállítjuk a Voice controlt    }
  call  GravisWait
  pop   ax
  push  &ch.word
  push  $0000
  push  ax
  call  SetGF1Byte
end;
```

StopVoice: A hangminta lejátszásának leállítása. Ez az eljárás leállítja a *ch* csatornán a *PlayVoice*-szal elindított lejátszást.

```
function  Hex(b:byte):string;
begin
 Hex:=Hdg[ b shr 4] +Hdg[ b and 15] ;
end;

function  HexW(w:word):string;
begin
 HexW:=Hdg[ w shr 12] +Hdg[ (w shr 8) and 15] +
       Hdg[ (w shr 4) and 15] +Hdg[ w and 15] ;
end;
```

Hex,Hexw: Bájt és word hexadecimális értékének meghatározása. A két függvénnyel egy szám hexadecimális alakját lehet meghatározni.

```
procedure Note(ch,n:byte);
begin
 GravisFreq(ch,FreqTAB[ n] );
end;
```

Note: Egy adott hangjegyhez tartozó frekvencia beállítása. Az eljárás a *FreqTAB* táblázat alapján beállítja a *ch* számú csatorna frekvenciáját. Az *n* a hangjegy száma, ennek mindig 0...59 közé kell esnie, mert a táblázat csupán öt oktáv hangjait tartalmazza (5*12=60)! Emlékezzünk rá, hogy ez az eljárás csak a frekvenciát állítja be, semmi mást nem tesz!

```
procedure Volume(ch,n:byte);
begin
 GravisVolume(ch,VolTAB[ n and 127] );
end;
```

Volume: Egy csatorna hangerejének lineáris beállítása. Az eljárással 0...127 között lineárisan lehet állítani a hangerőt. A 127 a legnagyobb hangerőt jelenti.

```
{ Nyugtázó megszakítás, FIFO-kiürítés }
procedure FirstIRQ;                                    interrupt;
```

```
var a:byte;
begin
 a:=port[ Base+$0006] ;
 a:=$00;
 while a and $c0<>$c0 do a:=getgf1byte($00,$8f);
 port[ $20] :=$20;
 port[ $a0] :=$20;
end;
```

FirstIRQ: A unit belső megszakítási rutinja, kívülről nem érhető el.

```
procedure SetIRQservice(Irqr:pointer);
var i:byte;
begin
 SetGF1Byte($00,$4C,$03);        { Minden GF1 IRQ-t tiltunk          }
 SetIntVec($73,@FirstIRQ);       { Beállítjuk az init IRQ-t          }
 port[ Base] :=$4c;              { Kiválasztjuk az IRQ regisztert    }
 port[ Base+$000b] :=$05;        { IRQ11 , int 73h  00001000         }
 port[ $a1] :=port[ $a1] and $F7; { Engedélyezzük a vonalat          }
 for i:=0 to 31 do
  begin                          { Minden csatorna letiltva          }
   setgf1byte(i,$0d,$02);
   setgf1byte(i,$00,$02);
  end;
 SetGF1byte($00,$41,$00);        { DRAM DMA IRQ tiltva               }
 SetGF1byte($00,$49,$00);        { Felvétel IRQ tiltva               }
 SetGF1Byte($00,$4C,$07);        { GF1 IRQ engedélyezve              }
 delay(400);                     { Várakozás                         }
 SetIntVec($73,IRQR);            { Az új megszakítási rutin          }
end;
```

SetIRQservice: A megszakítás inicializálása és egy megszakítási rutin címének beállítása.

```
procedure ResetIRQservice;
begin
 port[ $a1] :=port[ $a1] or $08;  { Tiltjuk a vonalat               }
 SetGF1Byte($00,$4C,$03);         { GF1 IRQ letiltva                }
end;
```

ResetIRQservice: A GF1 megszakításainak letiltása.

```
END. of unit
```

A unit három utolsó eljárása a megszakításkezeléshez szükséges. A megszakítási vonalat minden esetben 11-re állítjuk (függetlenül az eredeti beállítástól), mert ez egy szabad vonal, ezért legtöbbször konfliktus nélkül lehet használni. A *SetIRQservice* egy mutatót vár, amelynek a saját megszakítási rutinunkra kell mutatnia. Az átkapcsolás után esetleg

lehet egy-két megszakítási kérelem, amelyet még nem szolgáltunk ki. Ezért a *FirstIRQ* rutin ezeket nyugtázza. Ha ez megtörtént (a várakozás után), akkor beállítjuk az általunk megadott címet. A *ResetIRQservice* letiltja a megszakítást.

Talán van egy-két eljárás, amelynek működése (vagy célja) még nem világos. Nézzünk meg ezért néhány gyakorlati példát a unit és a hangkártya használatára!

8.8.1. A Gravis Ultrasound tesztelése

Az első példa a hangkártya lekérdezését mutatja be. Ehhez a *GravisFindBase* és a *GravisMemory* függvényeket használjuk. A program lekérdezi és kiírja a beállított értékeket.

```
program TestGravis;

uses Crt,GUS;

BEGIN
 clrscr;
 writeln('Gravis I/O báziscím: $',HexW(GravisFindBase));
 writeln('Gravis memória:      ',GravisMemory,' K');
 GravisReset;
END. of program
```

8.8.2. Egyszerű szinuszos hang előállítása

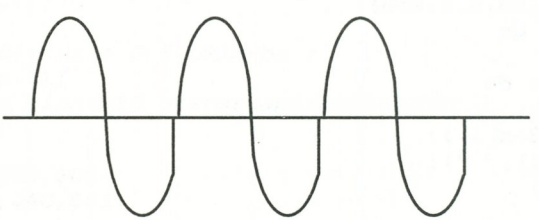
A második példában előállítunk egy szinuszhullámot és betöltjük a GUS DRAM-ba. A hang így egyetlen szinuszperiódust fog tartalmazni, amelyet 257 mintával adunk meg. A Sound Blaster család tagjaitól eltérően a Gravis valódi kettes komplemens számként értelmezi a DRAM tartalmát, ezért itt a kiszámított értékeket egyszerűen csak ki kell írni, nem szükséges semmilyen módosítás.

```
program sine_wave;

uses Crt,GUS;

var
   i : integer;

BEGIN
 clrscr;
```

```
 writeln('Gravis I/O báziscím: $',HexW(GravisFindBase));
 writeln('Gravis memória:       ',GravisMemory,' K');
 GravisReset;
 { A hullámforma: egyszerű szinusz (257 minta!) }
 for i:=0 to 256 do putbyte(i,round(100*sin(i*pi/32)));

 Volume($00,64);
 Note($00,36);
 GravisBalance($00,$08);
 SetGF1byte($00,$4C,$03);

 writeln('Start... Stop:ENTER');
 PlayVoice($00,$08,0,0,256);

 readln;
 StopVoice($00);
END. of program
```

A program elején egy egyszerű *for* ciklussal előállítjuk a hangmintát, és feltöltjük vele a GUS DRAM első 257 bájtját. Ez után a hang legfontosabb paramétereit állítjuk be, majd a *PlayVoice* eljárással megszólaltatjuk a 0. csatornát. A lejátszás módja $08. Ha visszalapozunk a regiszterek leírásához, akkor látható, hogy ez egy egyszerű, megszakítás nélküli ciklikus lejátszás.

A ciklikus lejátszás miatt kis helyen is tudunk tárolni egyszerű hangmintákat, de egyvalamire nagyon kell vigyázni: a ciklus kezdő és záró bájtja ugyanaz az érték legyen! Ha ez nem teljesül, akkor a hangmintában szakadás keletkezik:



55. ábra. Hibás hangminta

Az ilyen különbségek miatt a jelben felharmonikusok jelennek meg, amelyek zavaró kattogó hangot adnak. A jelenséget bárki észlelheti, ha az előbbi programban a *PlayVoice* utolsó paraméterét 256-ról 255-re változtatja. Ekkor a ciklus eleje és vége között nagy ugrás lesz, és ez a kimeneten jól hallható.

8.8.3. Oktávok

Az előző programhoz teljesen hasonló módon most is előállítunk egy szinuszhullámot, de most a *Note* eljárással az összes hangot megszólaltatjuk.

```
program notes;

uses Crt,GUS;

const
   n   : array[ 0..7] of byte=(0,2,4,5,7,9,11,12); { Egész hangok }
   txt : array[ 0..7] of char='CDEFGAHC';          { Hangjegynevek }
var
   i,j : integer;

BEGIN
 clrscr;
 writeln('Gravis I/O báziscím: $',HexW(GravisFindBase));
 writeln('Gravis memória:      ',GravisMemory,' K');
 GravisReset;

 { Szinusz }
 for i:=0 to 256 do PutByte(i,round(100*sin(i*pi/16)));
 { Csatornaparaméterek }
 Volume($00,32);
 GravisBalance($00,$08);
 SetGF1Byte($00,$4C,$03);

 writeln('Start...');
 PlayVoice($00,$8,0,0,256);
 for j:=0 to 4 do
  begin
  for i:=0 to 6 do
   begin
    Note(0,j*12+n[ i] );
    write(txt[ i] ,' ');
    delay(500);
   end;
   writeln;
  end;

 StopVoice($00);
END. of program
```

A program előállítja a hullámalakot, beállítja a 0. csatorna paramétereit, majd a *for* ciklusok kiadják a hangokat. A kiadott hangot az *n* tömb definiálja, a hangok neveit pedig a *txt* tömb tartalmazza. A *j* változó számolja az oktávokat (0...4), az *i* pedig az oktáv egész hangjait (0...6).

8.8.4. Hangminta lejátszása

Tegyük fel, hogy van egy Gravis formájú (kettes komplemens) 8-bites hangmintánk. Ezt betöltjük a GUS DRAM-ba, és egyszerűen lejátsszuk. Ugyanennek a feladatnak a megoldása a Sound Blaster esetében vagy közvetlen mintavételezést, vagy DMA átvitelt igényelt. Nézzük meg, hogy a Gravis kártyával ezt hogyan kell megtenni!

```
program PlaySample;

{ Egy hangminta lejátszása }
uses Crt,GUS;

var
   l : longint;
   p : memptr;
   f : file;

BEGIN
 clrscr;
 writeln('Gravis I/O báziscím: $',HexW(GravisFindBase));
 writeln('Gravis memória:      ',GravisMemory,' K');
 GravisReset;
 { Beolvassuk a hangmintát }
 assign(f,'sample.00');
 reset(f,1);
 l:=filesize(f);
 getmem(p,l);
 blockread(f,p^,l);
 close(f);
 { A mintát áttöltjük a GUS DRAM-ba }
 MoveSample($0,p,l);
 { Beállítjuk az alapvető csatornaparamétereket }
 Volume($00,64);
 Note($00,28);
 GravisBalance($00,$08);
 SetGF1byte($00,$4C,$03);

 write('Start...[ENTER]'); readln;
 PlayVoice($00,$00,0,0,l-1);

 readln;
 StopVoice($00);
 freemem(p,l);
END. of program
```

A hangminta neve *sample.00*, és az egyszerűség kedvéért feltételezzük, hogy kisebb, mint 64K. A program elején beolvassuk a heapbe (*blockread*), majd átmásoljuk onnan a

GUS DRAM-ba (*MoveSample*). A letöltés és a csatornaparaméterek beállítása után egyszerűen lejátsszuk a *PlayVoice* eljárással.
Ez a módszer feltételezi, hogy a hangminta normál kettes komplemens alakú. Ha viszont olyan hangot akarunk lejátszani, amelyet egy Sound Blaster mintavételezett, akkor az összes mintát korrigálni kell a kettes komplemens aritmetika szabályai szerint. A legegyszerűbb ilyen átalakítás például a következőképpen festene:

```
for i:=0 to size do Sample[ i] :=Sample[ i] -128;
```

vagy

```
for i:=0 to size do Sample[ i] :=Sample[ i] xor 127;
```

(A későbbiekben, a MOD fájlok ismertetésénél van egy program, amely egy MOD fájl hangmintáit menti normál kettes komplemens formában.)

8.8.5. Megszakítások

Eddig nem sok szó esett a megszakításokról, pedig ez nagyon fontos a Gravis használatakor. A GUS unitban megtalálható a *SetIRQservice* eljárás, amely beállítja a megszakítási rutin címét, amelyet mi adunk meg. Nézzük most ennek a használatát!

```
{$G+}
program GgravisIRQ;

uses Crt,Dos,GUS;

var
     i : integer;
     w : word;

{ A saját megszakításkezelő eljárásunk (IRQ 11 -> INT $73) }
procedure IRQ;                                          interrupt;
var i,j:byte;
begin
 { Kiolvassuk a globális megszakításforrás regisztert ($2x6) }
 i:=port[ Base+$0006] ;
 j:=0;
 { Kiolvassuk a megszakításforrás FIFO regisztert ($0f,$8f) }
 while j and $c0<>$c0 do j:=getgf1byte($00,$8f);
 { A képen megjelenik egy sárga színű # (hashmark) jel }
 memw[ $b800:w] :=$0e00+ord('#');
 asm
   add  w,$0002
   mov  al,$20
```

```
    out   $20,al            { Nyugtázzuk a megszakítást       }
    out   $a0,al            { mindkét megszakításvezérlőn!    }
  end;
end;

BEGIN
  clrscr;
  writeln('Gravis I/O báziscím: $',HexW(GravisFindBase));
  writeln('Gravis memória:      ',GravisMemory,' K');
  GravisReset;
  { Hullámalak: 8193 minta, szinusz }
  writeln('Hullámalak generálás !');
  for i:=0 to 8192 do putbyte(i,round(100*sin(i*pi/16)));
  { Csatornaparaméterek }
  Volume($00,32);
  Note($00,36);
  GravisBalance($00,$08);
  { Beállítjuk a megszakítási rutin címét és a GF1 megszakításait }
  SetIRQservice(@IRQ);

  write('Start...'); readln;
  w:=0;
  { A mode paraméter értéke $28, azaz 01001000 }
  { 3. bit: loop enable }
  { 6. bit: IRQ  enable }
  PlayVoice($00,$28,0,0,8192);

  readln;
  StopVoice($00);
  { Tiltjuk a GF1 megszakításait }
  ResetIRQservice;
END. of program
```

A program első része hasonló az eddigiekhez: előállítunk egy szinuszjelet a GUS DRAM-ban (most 8193 bájt hosszú), és beállítjuk a csatornaparamétereket. Ez után installáljuk a saját megszakítási rutinunkat, az *IRQ*-t. A *SetIRQservice* beállítja a GF1 megfelelő regisztereit, kiválasztja a 11-es megszakítási vonalat, nyugtázza az esetleges előző megszakítási kéréseket a *FirstIRQ* eljárással, végül pedig beállítja a megadott című rutint. Természetesen az eljárás nem ellenőrzi, hogy az átadott pointer egyáltalán eljárásra mutat-e, mindig magunknak kell gondoskodni róla, hogy az átadott cím egy korrekt megszakítási rutin címe legyen!

A *PlayVoice* eljárás elindítja a 8K méretű hangminta lejátszását, de a lejátszás módját ezúttal úgy állítjuk be, hogy a minta végén — amikor a ciklikus lejátszás miatt elölről kezdődik a mintavételezés — a GF1 egy megszakítást adjon. A processzor ilyenkor hívja meg az *IRQ* rutint.

Mi történik a megszakítás alatt? Elsőként kiolvassuk a globális megszakítási állapotregisztert, bár a beállítások miatt most biztosan csak a hullámtábla megszakítási bitje lehet beállítva benne (hiszen más nem okozhat megszakítást, mivel nem is történik más, csak egyszerű lejátszás). A globális állapotregiszter mellett a csatornaforrás FIFO-t is ki kell olvasni. Ebben most csupán a 0. csatorna hullámtáblakérése szerepelhet. Hogy a megszakítás végrehajtását a képernyőn követni tudjuk, ezért minden egyes megszakítás kitesz egy # jelet. Befejezésül nyugtázzuk a megszakításvezérlő áramköröket, jelezve ezzel, hogy vége a megszakításnak. Mivel a 11-es megszakítási vonalat használtuk, amelyet a két vezérlő kaszkádosításával oldottak meg, ezért mindkét vezérlőt nyugtázni kell!

De miért kell éppen a 11-es megszakítási vonalat használni? Az AT-k megszakítási vonalainak kiosztása a következő:

Az első megszakításvezérlő kiosztása		
IRQ x	**INT $xx**	**A megszakítást kérő eszköz**
IRQ 0	INT $08	időzítő
IRQ 1	INT $09	billentyűzet
IRQ 2	INT $0A	a második megszakításvezérlő
IRQ 3	INT $0B	a második aszinkron soros vonal
IRQ 4	INT $0C	az első aszinkron soros vonal
IRQ 5	INT $0D	a második párhuzamos port (szabad)
IRQ 6	INT $0E	floppymeghajtó
IRQ 7	INT $0F	az első párhuzamos port

A második megszakításvezérlő kiosztása		
IRQ x	**INT $xx**	**A megszakítást kérő eszköz**
IRQ 8	INT $70	valós idejű óra
IRQ 9	INT $71	szabad
IRQ 10	INT $72	szabad
IRQ 11	INT $73	szabad
IRQ 12	INT $74	szabad
IRQ 13	INT $75	80x87 koprocesszor
IRQ 14	INT $76	merevlemez-illesztő
IRQ 15	INT $77	szabad

Az első vezérlőn általában az 5-ös vonal szabad, mert ritka (persze lehetséges) egyszerre a két párhuzamos port. A második vezérlőn már jóval több szabad vonal található. A hangkártya ezek közül a 11, 12 és 15-ös vonalat tudja használni. Ezek közül bármelyiket kiválaszthattuk volna, de a 11-es az alapbeállítás, a gyári programok is ezt használják.

9. Zenefájl formátumok

A zenét, a hangokat valamilyen formában tárolnunk kell. (Tulajdonképpen a kotta is a zene egyik jól átgondolt, papíron rögzített formája.) Ehhez szükségünk van valamilyen megegyezés szerinti adatszerkezetre — fájlformátumra —, amely egységesen és logikusan tudja tárolni a zeneadatokat, és amely elég jól van megkonstruálva ahhoz, hogy a zenét valós időben vissza lehessen játszani. Mára már kialakult jó néhány ilyen zenefájl formátum, amelyeket a hangkártyákhoz mellékelt szoftverek általában kezelni tudnak. Ismerkedjünk meg most a leggyakrabban használt zenefájlok szerkezetével!

9.1. Az SBI fájlok formátuma

Az SBI (*Sound Blaster Instrument*) fájlokat az FM hanggenerálással előállítható hangszerek tárolására használhatjuk. A fájl tartalmazza az összes paramétert az FM chip megfelelő programozásához (igaz, hogy csak a kétoperátoros hanggeneráláshoz). Az adatszerkezet éppen ezért nagymértékben hasonlít az FM regiszterek hardver által definiált felépítéséhez. Minden SBI fájl 51 (esetleg 52) bájt hosszú.

Eltolás	Adattípus	A mező jelentése
$00...$03	szöveg	SBI azonosító (*„SBI"+$1A*)
$04...$23	szöveg	a hangszer neve (*Instrument name*)
$24	bájt	modulátor paraméterek (*Modulator sound properties*)
$25	bájt	carrier paraméterek (*Carrier sound properties*)
$26	bájt	modulátor hangerő (*Modulator volume*)
$27	bájt	carrier hangerő (*Carrier volume*)
$28	bájt	modulátor burkológörbe AD összetevő (*Attack/Decay*)
$29	bájt	carrier burkológörbe AD összetevő (*Attack/Decay*)
$2A	bájt	modulátor burkológörbe SR összetevő (*Sustain/Release*)
$2B	bájt	carrier burkológörbe SR összetevő (*Sustain/Release*)
$2C	bájt	modulátor hullámforma (*Modulator wave select*)
$2D	bájt	carrier hullámforma (*Carrier wave select*)
$2E	bájt	visszacsatolás/összekapcsolás (*Feedback/Connection*)
$2F...$33	bájt	fejlesztésre fenntartott terület (*Reserved*)

Fájlazonosító: Bájt $00...$03 (*File ID*)

Ebben a mezőben található a fájl azonosítására szolgáló „SBI" szöveg, amelyet egy $1A bájt zár le. Ez alapján lehet meghatározni, hogy a fájl SBI formátumú-e.

Hangszernév: Bájt $04...$23 (*Instrument name*)

Itt található a hangszer neve, amely legfeljebb 32 karakterből állhat. A szöveget a C nyelv szabályai szerint egy $00 értékkel zárják le. Ha a saját hangszerünknek nem akarunk nevet adni, akkor ezt a területet $00-val kell kitölteni.

Modulátor paraméterek: Bájt $24 (*Modulator sound properties*)

Ez a bájt a hangszermodulátor (1. operátor) jellemzőit tartalmazza. A bitek kiosztása megfelel a *$20 Effektusok* regiszternek (lásd az ADLIB-nél):

- bit 7: amplitúdómoduláció;
- bit 6: frekvenciamoduláció;
- bit 5: a GATE bit;
- bit 4: a KSR (skálatényező) bit;
- bit 3–0: frekvenciaszorzó-érték.

Carrier paraméterek: Bájt $25 (*Carrier sound properties*)

Ez a bájt a carrier (2. operátor) jellemzőit tartalmazza. A bitek kiosztása ugyanaz, mint a $24-es bájtnál.

Modulátor hangerő: Bájt $26 (*Modulator volume*)

A modulátor hangerejét definiálja ez a bájt, amely gyakorlatilag a *$40 Kimeneti jelszint* regiszter értékét tartalmazza.

Carrier hangerő: Bájt $27 (*Carrier volume*)

A carrier hangereje. Ugyanaz a feladata, mint a $26-os bájtnak, csak ez a 2. operátor hangerejét adja.

Modulátor ADSR ciklus AD összetevő: Bájt $28 (*Modulator attack/decay*)

A modulátor burkológörbéjének *attack* és *decay* összetevőit tartalmazza.

- bit 7–4: *attack* időtartam;
- bit 3–0: *decay* időtartam.

Carrier ADSR ciklus AD összetevő: Bájt $29 (*Carrier attack/decay*)

A carrier burkológörbéjének *attack* és *decay* összetevőit tartalmazza. A bitek kiosztása megegyezik a $28-as bájt bitjeinek kiosztásával.

Modulátor ADSR ciklus SR összetevő: Bájt $2A (*Modulator sustain/release*)

A modulátor burkológörbéjének *sustain* és *release* összetevőit tartalmazza.

bit 7–4: *sustain* szint;
bit 3–0: *release* időtartam.

Carrier ADSR ciklus SR összetevő: Bájt $2B (*Carrier sustain/release*)

A carrier burkológörbéjének *sustain* és *release* összetevőit tartalmazza. A bitek ugyanúgy vannak definiálva, mint a $2A bájtnál.

Modulátor hullámalak: Bájt $2C (*Modulator waveform*)

A modulátor hullámalakját definiálja.

Érték	Hullámforma
$00	egyszerű szinusz
$01	pozitív szinusz
$02	szinusz abszolút érték
$03	50%-os fázishasított szinusz

Carrier ADSR hullámalak: Bájt $2D (*Carrier waveform*)

A carrier hullámalakját definiálja ugyanúgy, mint a $2C bájt.

Szintézis mód és fáziseltolás: Bájt $2E (*Synthesis mode and phase shifting*)

Ez a bájt adja meg az összekapcsolás típusát és a visszacsatolás erősségét. A bitek kiosztása megfelel a *$C0 visszacsatolási szint* regiszter bitkiosztásának.

bit 3–1: a visszacsatolás erőssége;
bit 0: az összekapcsolás módja (0: FM szintézis, 1: additív szintézis).

Nem definiált terület: Bájt $2F...$33 (*Bytes for future use*)

Ezek a bájtok már lényegtelenek a hangkeltés szempontjából. Az esetleges jövőbeni bővítésekhez tartották fenn ezt a területet (pl. négyoperátoros hanggenerálás).

A következő egyszerű program szöveges formára konvertál egy SBI fájlt. A szövegben benne lesz minden fontos paraméter, amely a hangszert jellemzi. A program csupán egy egyszerű Pascal „ujjgyakorlat", ezért a működését most nem ismertetjük részletesen.
:-(

```
program sbi2txt; {sbi2txt.pas }

uses crt;

type
   Pmem = ^Tmem;
   Tmem = array[ 0..65520] of byte;

const
   { Karakterek a hexadecimális kiíráshoz }
   Hd : array[ 0..15] of char='0123456789ABCDEF';
   { Bitmaszkok }
   Bt : array[ 0..7] of byte=($80,$40,$20,$10,$08,$04,$02,$01);

   { Az operátorok összekapcsolásai }
   Conn : array[ 0..1] of string[ 20]=(
          'FM szintézis',
          'Additív szintézis');

var
   Name : string[ 100];          { A forrás SBI fájl neve      }
   Tx   : string[ 100];          { A cél    TXT fájl neve      }
   Buff : array[ 0..51] of byte; { Az SBI fájl buffere         }
   i    : integer;
   l    : longint;
   f    : file;                  { Forrásfájl                   }
   t    : text;                  { Célfájl                      }
   s    : string;
   ID   : string[ 4];            { SBI azonosító                }

{ Egy $00-val végződő sztring beolvasása (hangszernév)        }
function GetString(const Src:Pmem):string;
var s : string;
    i : byte;
    p : pmem;
begin
 s:=''; i:=0;
 while Src^[ i]<>$00 do
  begin
   s:=s+Chr(Src^[ i]); inc(i);
  end;
 if s='' then s:='NONAME';
 GetString:=s;
```

```
end;

{ Egy bájt hexadecimális alakja                                    }
function Hex(x:byte):string;
begin
 Hex:=Hd[ x shr 4] +Hd[ x and $0f] ;
end;

{ Egy bájt bináris alakja                                          }
function Bin(x:byte):string;
var i:byte;
    s:string[ 8] ;
begin
 s:='00000000';
 for i:=0 to 7 do
  if X and Bt[ i] =Bt[ i]  then s[ i+1] :='1';
 Bin:=s;
end;

{ Egy bájt adott bitjének értéke                                   }
function Bit(x,n:byte):boolean;
begin
 Bit:=x and Bt[ 7-n] =Bt[ 7-n] ;
end;

BEGIN
 clrscr;
 { Előállítjuk vagy bekérjük a forrásfájl nevét }
 if paramcount=0 then
  begin
   write('Az SBI fájl neve (kiterjesztés nélkül) :');
   readln(Name);
  end
 else Name:=paramstr(1);
 for i:=1 to length(name) do name[ i] :=upcase(name[ i] );
 Tx:=Name+'.TXT';
 Name:=Name+'.SBI';
 assign(t,Tx);
 rewrite(t);
 { Beolvassuk a forrásfájlt }
 assign(f,Name);
 reset(f,1);
 l:=filesize(f);
 if l>52 then l:=52;
 blockread(f,Buff,l);
 close(f);
 ID[ 0] :=#4;
 move(Buff,ID[ 1] ,4);
 { Ellenőrizzük, hogy SBI fájl-e, és kilépünk, ha nem az }
```

```
 if ID<>'SBI'#$1A then
  begin
   writeln('Ez nem egy SBI hangszer fájl !');
   HALT;
  end;

  clrscr;
  { Kiírjuk az SBI paramétereket }
  writeln(t,'A fájl neve:      '+Name);
  writeln(t,'A hangszer neve: '+GetString(@Buff[ 4] ));
  writeln(t,'---------------------------------');
  writeln(t,'Modulátor:            Carrier:');
  writeln(t,'- karakterisztika:');
  writeln(t,bin(Buff[ $24] )+'($'+hex(Buff[ $24] )+')          '+
          bin(Buff[ $25] )+'($'+hex(Buff[ $25] )+')');
  writeln(t,'- alaphangerő:');
  writeln(t,bin(Buff[ $26] )+'($'+hex(Buff[ $26] )+')          '+
          bin(Buff[ $27] )+'($'+hex(Buff[ $27] )+')');
  writeln(t,'- burkológörbe:');
  write   (t,'Attack:   ',Buff[ $28] shr 4:2,'          ');
  writeln(t,'Attack:   ',Buff[ $29] shr 4:2);
  write   (t,'Decay:    ',Buff[ $28] and 15:2,'          ');
  writeln(t,'Decay:    ',Buff[ $29] and 15:2);
  write   (t,'Sustain:  ',Buff[ $2a] shr 4:2,'          ');
  writeln(t,'Sustain:  ',Buff[ $2b] shr 4:2);
  write   (t,'Release:  ',Buff[ $2a] and 15:2,'          ');
  writeln(t,'Release:  ',Buff[ $2b] and 15:2);
  writeln(t,'- hullámalak:');
  writeln(t,bin(Buff[ $2c] )+'($'+hex(Buff[ $2c] )+')          '+
          bin(Buff[ $2d] )+'($'+hex(Buff[ $2d] )+')');
  writeln(t,'---------------------------------');
  writeln(t,'- visszacsatolás: ',Buff[ $2e] shr 1);
  writeln(t,'- összekapcsolás: ',Conn[ Buff[ $2e] and 1] );
 close(t);
END. of program
```

9.1.1. Az SBI unit

A kiíratásnál érdekesebb feladat a hangszer megszólaltatása. Ez nem túl nehéz feladat, hiszen az SBI fájl szerkezete illeszkedik az FM regiszterek kiosztásához. Egyszerűen csak be kell tölteni a megfelelő regiszterbe a megfelelő értéket, és máris készen vagyunk. A most következő egységben egy egyszerű dinamikus objektumot :-) definiáltunk, amely az ADLIB unitot használva megszólaltatja a hangot. (Természetesen ADLIB kompatíbilis hangkártyát használunk!)

```
unit SBI; { sbi.pas }

interface

type
    Pidata = ^Tidata;
    Tidata = array[ 0..10] of byte;

    Pinstrument = ^Tinstrument;
    Tinstrument = object
     Chan : byte;    { Csatornaszám     }
     Data : Tidata; { Hangszeradatok  }
     constructor Init(var Src; Ch:byte);
     procedure   Double(Reg,Idx:byte);
     procedure   KeyOn(Octave,Note:byte);          virtual;
     procedure   KeyOff;                           virtual;
     procedure   ChangeChan(NewChan:byte);
     procedure   ChangeData(var NewData);
     destructor  Done;
    private
     procedure   InitChan;
    end;
```

Pidata: Mutató a *Tidata* típusra.

Tidata: Ez a típus tárolja a hangszer adatait. Amint azt az SBI fájl leírásánál láthattuk, a hang kiadása szempontjából csak az SBI fájl $24...$2E bájtjainak van jelentősége. Ezt a 11 bájtot tároljuk az ilyen típusú változókban.

Pinstrument: Mutató a *Tinstrument* típusra.

Tinstrument: A hangszert tároló objektum. Ez az objektum tárolja az adatokat, és képes megszólaltatni a hangszert. Nézzük meg az objektum mezőit és metódusait !

Chan (mező): A hangszerhez rendelt fizikai ADLIB csatorna száma. Mivel az ADLIB 9 csatornát tartalmaz, ennek 0...8 közé kell esnie. Az *Init* és a *ChangeChan* metódusokkal lehet módosítani.

Data (mező): A hangszer adatai. Ez a mező *Tidata* típusú, ez tárolja a regiszterek értékeit. Az *Init* és a *ChangeData* metódusokkal lehet módosítani.

Init (metódus, konstruktor): Az objektum konstruktora. Paraméterként a hangszeradatok címét (*Src*) és a fizikai csatorna számát (*Ch*) várja. Az ADLIB regisztereket beállítja, és a megadott csatornát kikapcsolja.

Double (metódus): A kétoperátoros regisztereket állítja be ez a metódus. Az első operátorba a modulátor, a második operátorba a carrier értékeket tölti. Leginkább belső használatra készítettük, kívülről nincs sok értelme meghívni.

KeyOn (metódus): A hang megszólaltatása. A metódus az *octave* oktáv *note* hangját szólaltatja meg a hangszerrel. A metódust a későbbi módosításokhoz virtuálisnak deklaráltuk.

KeyOff (metódus): A hang kikapcsolása. A metódus kikapcsolja a fizikai csatornát. A metódust a későbbi módosításokhoz virtuálisnak deklaráltuk.

ChangeChan (metódus): A fizikai csatorna megváltoztatása. Ez a metódus a *NewChan* paraméterrel megadott értékre állítja a *Chan* mezőt, és a megfelelő ADLIB csatornát is beállítja. A beállítás után a régi csatorna elhallgat, de a paraméterek változatlanok maradnak.

ChangeData (metódus): A hangszer adatainak megváltoztatása. A metódus a *NewData* által meghatározott új adatokkal tölti fel a *Data* mezőt. A hangot kikapcsolja.

Done (metódus, destruktor): Az objektum destruktora.

InitChan (privát metódus): A *Data* mező értékeinek átmásolása az ADLIB regiszterekbe. Csak belső használatra!

A megvalósítás az ADLIB unittal nem okozhat gondot:

```
implementation uses ADLIB;

constructor Tinstrument.Init(var Src; Ch:byte);
begin
 Chan:=Ch;                      { Beállítjuk a csatornaszámot }
 if Chan>8 then Chan:=8;
 move(Src,Data,11);             { Átmásoljuk az adatokat a    }
                                { Data mezőbe és onnan az     }
 InitChan;                      { ADLIB regiszterekbe         }
 KeyOff;                        { Kikapcsoljuk a csatornát    }
end;

procedure Tinstrument.InitChan;
begin
 { Az ADLIB regiszterek beállítása }
 Double($20,$00);               { Effektusok      }
 Double($40,$02);               { Hangerő         }
 Double($60,$04);               { Attack/Decay    }
 Double($80,$06);               { Sustain/Release }
 Double($e0,$08);               { Hullámforma     }
```

```
 WriteReg($c0+Chan,Data[ 10] ); { Visszacsatolás  }
end;

procedure Tinstrument.Double(Reg,Idx:byte);
begin
 { A kétoperátoros regiszterek beállítása }
 WriteReg(Reg+Cnum[ Chan]   ,Data[ Idx]   ); { Modulátor }
 WriteReg(Reg+Cnum[ Chan] +3,Data[ Idx+1] ); { Carrier   }
end;

procedure Tinstrument.KeyOn(Octave,Note:byte);
begin
 { A hangszert az ADLIB unit Sound eljárásával szólaltatjuk meg }
 Sound(Chan,Octave,Note);
end;

procedure Tinstrument.KeyOff;
begin
 { A csatornát az ADLIB unit Off eljárásával kapcsoljuk ki }
 Off(Chan);
end;

procedure Tinstrument.ChangeChan(NewChan:byte);
begin
 KeyOff;                        { Csend legyen!          }
 Chan:=NewChan;                 { Az új csatornaszám     }
 if Chan>8 then Chan:=8;
 InitChan;                      { Regiszterbeállítás     }
end;

procedure Tinstrument.ChangeData(var NewData);
begin
 KeyOff;                        { Csend legyen!          }
 move(NewData,Data,11);         { Az új adatok           }
 InitChan;                      { Regiszterbeállítás     }
end;

destructor Tinstrument.Done;
begin
 KeyOff;                        { Csend legyen!          }
end;

END. of unit SBI
```

A unit használatára nézzünk egy rövid példát! Tegyük fel, hogy van egy 'PIANO.SBI' nevű hangszerünk. (Ha nincs, akkor a függelékekben megtalálhatók hozzá a legfontosabb paraméterek.) Olvassuk be ezt a hangszert, hozzunk létre a heapben egy hangszer-objektumot, és szólaltassuk meg!

```
program SBItest; { instr.pas }

uses Crt,ADLIB,SBI;

var
    f : file;
    b : array[ 0..51] of byte;
    X : Pinstrument;          { A hangszer mutatója }
    i : byte;

const
                              {C,D,E,F,G,A,H }
    n : array[ 0..6] of byte=(0,2,4,5,7,9,11);

BEGIN
 { Beolvassuk a fájlt a b bufferbe }
 assign(f,'PIANO.SBI');
 reset(f,1);
 blockread(f,b,52);
 close(f);
 { Inicializáljuk a hangkártyát }
 ResetADLIB;
 { Létrehozzuk a hangszerobjektumot a NEW-val }
 new(X,init(b[ $24],$00));

 { Megszólaltatjuk egy oktáv egész hangjait }
 for i:=0 to 6 do
  begin
   X^.KeyOn(4,n[ i]);
   delay(300);
  end;
 X^.KeyOn(5,0);
 delay(1500);

 { Megszüntetjük az objektumot a DISPOSE-zal }
 dispose(X,done);
END. of program
```

9.2. A CMF fájlok formátuma

A CMF (*Creative Music File*) fájlokat a Sound Blaster család gyártója, a *Creative Labs* dolgozta ki az FM hangszerekkel lejátszott zene tárolására. A zenefájlt három fontos részre lehet bontani:

- fejléc (*Header block*);
- hangszerek (*Instrument block*);
- zene (*Music block*).

A fejléc a fájl főbb adatait tartalmazza, a hangszerblokk a felhasznált FM hangszerek leírását (hasonló módon, mint az SBI fájlban), a zeneblokk pedig a lejátszandó zenét leíró MIDI eseményeket. Nézzük meg sorban ezeket a részeket!

9.2.1. CMF fejléc

Eltolás	Adattípus	A mező jelentése
$00...$03	szöveg	a fájl azonosítója (*„CTMF"*)
$04,$05	word	fájl verziószám alsó, felső (*File version number*)
$06,$07	word	a hangszerek eltolása (*Instrument offset*)
$08,$09	word	a zeneadatok eltolása (*Music offset*)
$0A,$0B	word	negyedhangok üteme (*Ticks per quarter note*)
$0C,$0D	word	időzítő ütem másodpercenként (*Clock ticks per second*)
$0E,$0F	word	szöveges fejléc eltolása (*Title offset*)
$10,$11	word	a szerző nevének eltolása (*Author name offset*)
$12,$13	word	megjegyzések eltolása (*Remarks offset*)
$14...$23	bájt	csatornahasználati táblázat (*Channel table*)
$24,$25	word	a használt hangszerek száma (*Number of instruments*)
$26,$27	word	tempó (*Tempo*)
$28...??	szöveg	szövegek (*Text bytes*)

Fájlazonosító: Bájt $00...$03 (*File ID*)

Ebben a mezőben található a fájl azonosítására szolgáló „CTMF" szöveg. Ez alapján lehet meghatározni, hogy a fájl CMF formátumú-e.

Fájl verziószám: Bájt $04,$05 (*Version number*)

Ez a két bájt tartalmazza a verziószámot (alverzió és főverzió).

Hangszerleírás eltolása: Bájt $06,$07 (*Instrument offset*)

Ez az adat egy 16 bites (word) eltolás, amely megadja, hogy a fájlban hol kezdődik az FM hangszerek leírása. Az eltolást természetesen a fájl kezdőbájtjához kell viszonyítani. A hangszerek leírása logikusan kapcsolódik az FM chip regiszterkiosztásához. (Mi a különbség a fájlszerkezet leírásában definiált eltolás és a felhasználói eltolás között? A leírásban definiált eltolás a fájl kezdőbájtjától számított relatív távolságot jelenti, és az adott fájlkomponens abszolút pozíciójának meghatározására használható fel, a felhasználói eltolás pedig az, amikor a begépelés — vagy az algoritmus kidolgozása — során bekövetkezett hiba miatt a gépet a RESET gomb segítségével újraindítjuk. :-) Mindezt

csak azért írjuk le így, hogy az *eltolás* szót senki ne a köznapi negatív jelentéssel ruházza fel, hanem mint programozási alapszót értse!)

A zeneadatok eltolása: Bájt $08,$09 (*Music offset*)

A hangszerek eltolásához hasonló módon ez a két bájt (egy word) a zeneadatok eltolását adja meg a fájlban. Az eltolást a fájl kezdőbájtjához kell viszonyítani.

Negyedhangok üteme: Bájt $0A,$0B (*Ticks per quarter*)

Ez az adat meghatározza, hogy hány timer ütem jelent egy negyedhangot a zene egy részében.

Időzítő ütem másodpercenként: Bájt $0C,$0D (*Clock ticks per second*)

Ez a két bájt a másodpercenkénti időzítőmegszakítások számát adja meg. Az alaplap időzítő egysége az IRQ 0 vonalon megszakításokat kér, alapesetben 18.2-t. A kérés minden esetben generál egy INT $08 megszakítást. A CMF fájlok esetében az alapbeállítás $60, azaz 96. Tehát az FM egység számára minden másodpercben 96 megszakítás szükséges.

Szöveges fejléc eltolása: Bájt $0E,$0F (*CMF title*)

A szöveges fejléc eltolása a fájlon belül. Ha értéke $0000, akkor nincs szöveges fejléc.

A szerző nevének eltolása: Bájt $10,$11 (*Author name*)

A szöveges fejléchez hasonlóan a szerző nevét is lehet tárolni a CMF fájlban. Az eltolás a névre mutat a fájlon belül. Ha értéke $0000, akkor nincs tárolva a szerző neve.

Megjegyzések eltolása: Bájt $12,$13 (*CMF remarks*)

A CMF fájlban lehetőség van bizonyos szöveges megjegyzések elhelyezésére. Ilyenkor mindig egy $00 bájttal kell lezárni a szöveges megjegyzéseket. Ez a két bájt a szöveg eltolását mutatja a fájlon belül. Ha értékük $0000, akkor a CMF fájlban nincs megjegyzés.

Csatornatáblázat: Bájt $14...$23 (*Channel table*)

A CMF fájl zenei adatai MIDI formátumúak. A MIDI 16 csatornát használ. Ez a 16 bájtos terület mutatja, hogy a zene mely csatornákat használja. Ha az adott bájt értéke 1, akkor a fájl használja a csatornát, ha 0, akkor nem.

A felhasznált hangszerek száma: Bájt $24,$25 (*Number of instruments*)

A használt hangszerek száma. Egy CMF fájlban legfeljebb 128 hangszert lehet használni.

Tempó: Bájt $26,$27 (*Tempo*)

A zene tempója.

Szövegek: Bájt $28...?? (*Text bytes*)

A fájl szövegeit helyezhetjük el ezen a területen (szöveges fejléc, szerző, megjegyzések). Minden szöveget egy $00 bájttal kell lezárni.

A szövegek után következnek a hangszerek adatai. A hangszerek kezdőcímét a fájl $06,$07-es bájtjain találjuk. Minden hangszer 16 bájtot igényel, ezek a bájtok megegyeznek az SBI fájl $24-ik bájtjától kezdődő résszel. Ezek az adatok az FM chip belső szerkezetéhez illeszkednek, csupán a megfelelő regisztereket kell feltölteni velük.

Általában a hangszerek adatai után következik a zene (bár ez nem törvényszerű, hiszen mind a hangszereket, mind a zenét eltolással kell megadni, tehát a fordított sorrend sem okozhat gondot). A zene megfelel a szabványos MIDI formátumnak, erről bővebben a MIDI-ről szóló fejezetben írunk.

A CMF formátum is definiál néhány MIDI-vezérlő eseményt, ezek a következők:

Eseménykód	Eseményadat	MIDI esemény
$66	1...127	megjegyzett pontok a zenében (*markers*)
$67	0,1	normál FM mód, ritmus mód
$68	0...127	minden következő hang emelése (1/128-ad félhanggal)
$69	0...127	minden következő hang süllyesztése (1/128-ad félhanggal)

Ha a zene ritmus módban szól, akkor a MIDI csatornák közül az utolsó öt le van foglalva ritmushangszerek számára!

9.2.2. A CMF fájlok lejátszása

A Sound Blaster hangkártyákhoz a gyári lemezmellékleten található egy SBFMDRV.COM nevű fájl, amely kifejezetten a CMF formátumú fájlok lejátszásához készült. Ez a program egy memóriarezidens meghajtóprogram (*driver*), amellyel az FM hangkeltés könnyen és egyszerűen megvalósítható. A program az elindításakor lefoglal egy megszakítást (általában az INT $80-at, bár ez nem törvényszerű), majd ez után úgy tudjuk használni a függvényeit, mint a BIOS szubrutinjait. A paraméterek átadása a processzor regiszterein keresztül történik, és az esetlegesen visszaadott értékek is itt jelennek meg. Az egyes funkciók a E függelékben találhatók. A *driver* használatához készítettünk egy rövid unitot és egy egyszerű példaprogramot, amellyel CMF fájlokat lehet lejátszani.

```
unit FMDRV; { FMDRV.PAS }

interface

type

 { CMF fájl fejléc rekord }
 CmfHead = record
  ID           : array [ 0..3] of char;
  Ver          : word;
  Instruments  : word;
  Music        : word;
  TickPerBeat  : word;
  Clock        : word;
  FileTitleOfs : word;
  ComposerOfs  : word;
  MusicRemOfs  : word;
  Channels     : array [ 0..15] of char;
  InstrNumber  : word;
  BasicTempo   : word;
 end;

const
  { Karakterek a hexadecimális kiíráshoz }
  Hd : array[ 0..15] of char='0123456789ABCDEF';

  { Rendszer alapfrekvencia }
  DefaultFreq = 1193180;

var
  IntNum : byte;        { A megszakítás száma      }
  Status : byte;        { Az állapotbájt           }
```

```
  Driver : procedure;   { Az FMDRV belépési pontja  }

function  Hex(x:byte):string;
function  InitDriver:boolean;
procedure StatusAddress(var Status:byte);
function  ResetDriver:boolean;
function  GetDriverVersion:word;
function  Instruments(var CMFdata):boolean;
function  PlayCMFfile(var CMFdata):boolean;
function  StopCMFfile:boolean;
function  PausePlay:boolean;
function  ContinuePlay:boolean;

procedure SetDriverClock(CLK:word);
procedure SetSystemClock(CLK:word);
procedure Transpose(Delta:integer);

implementation uses DOS;

function Hex(x:byte):string;  { DEC->HEX átalakítás }
begin
 Hex:=Hd[ x shr 4] +Hd[ x and 15] ;
end;

procedure StatusAddress(var Status:byte);                    assembler;
asm
  mov  bx,$0001          { A funkciókód $0001                        }
  les  ax,Status
  mov  dx,ES             { DX:AX az állapotbájt címét tartalmazza }
  pushf
  call Driver            { Meghívjuk a megszakítást                  }
end;
```

StatusAddress: Az állapotbájt címének beállítása. A *driver* egy állapotbájtban jelzi vissza a műveletek végrehajtásának állapotát és az esetleges hibákat. Ennek a bájtnak a címét nekünk kell definiálni. Ezt lehet megtenni ezzel az eljárással.

```
function InitDriver:boolean;
var i : word;            { Ciklusváltozó          }
  P,Q : Pointer;         { Mutatók                }
   ID : string[ 5] ;     { Azonosító szöveg       }
    D : procedure;       { Eljárás a kereséshez   }
begin
 @D:=nil;                { Kezdéskor a D még üres }
 ID[ 0] :=#5;            { Az azonosító sztring ötkarakteres }
 for i:=$80 to $BF do    { A lehetséges megszakítások: $80...$BF }
  begin
   GetIntVec(i,P);       { Lekérdezzük a megszakítás címét }
```

```
   Q:=Ptr(Seg(P^),$103); { Előállítjuk az azonosító sztring  }
                         { címét a megszakítás címéből       }
   move(Q^,ID[1],5);     { Átmásoljuk a karaktereket         }
   if ID='FMDRV' then
    begin
     @D:=P;              { Ha megtaláltuk, akkor kilépés a   }
     break;              { for ciklusból                     }
    end;
  end;
 { Ha egyik helyen sem volt, akkor a meghajtó valószínűleg   }
 { nincs a memóriában                                        }
 if @D=nil then InitDriver:=false
  else
   begin                       { Sikeres volt a keresés:    }
    Driver:=D;                 { A rutin beállítása         }
    IntNum:=i;                 { A megszakítás száma        }
    InitDriver:=True;          { A függvény 'igaz' lesz     }
    StatusAddress(Status);     { Az állapotbájtot beállítjuk }
    ResetDriver;               { Inicializálás              }
   end;
end;
```

InitDriver: A *driver* lekérdezése és inicializálása. Ez a függvény megállapítja, hogy az SBFMDRV.COM program benne van-e a memóriában. A lekérdezés meglehetősen sajátos módon történik. A program a $80...$BF megszakításokat használhatja. A *driver* mindig egy 64K-nál rövidebb program (COM fájl), ezért mindig elfér egyetlen kódszegmensben. A verziószámtól függetlenül ezen kódszegmens $103-as bájtján egy azonosító szövegnek („FMDRV") kell lennie. A lekérdezésnél ezt használjuk ki. Sorra lekérdezzük a lehetséges megszakítások vektorait (a *P* pointerben), és ebből előállítjuk az azonosító szöveg címét (a *Q* pointerben). Ha az itt található öt bájt az „FMDRV" szöveget tartalmazza, akkor megvan a *driver* megszakítása. Ezt a címet tároljuk a *Driver* eljárásváltozóban, és a továbbiakban minden egyes meghíváskor ezt használjuk. Mivel ez egy megszakítás, ezért minden hívás előtt a flagregiszter tartalmát is ki kell tenni a verembe, hogy a visszatéréskor ne legyen hiba. A meghívás és a visszatérés ezért mindig a következő:

```
      { A rutinok meghívása }
      pushf              { Tároljuk a flagregisztert   }
      call   Driver      { Meghívjuk a megszakítást    }

      { A driver visszatérése }
      ...
      iret
```

Tehát minden további esetben ezt a módszert kell alkalmazni.

```
function GetDriverVersion:word;                    assembler;
asm
```

```
  xor  bx,bx
  pushf
  call Driver
end;
```

GetDriverVersion: A program verziószámának lekérdezése. Ez a függvény visszaadja a *driver* verziószámát egy word adatban. A felső bájt a főverziószámot, míg az alsó bájt az alverziószámot tartalmazza.

```
function Instruments(var CMFdata):boolean;
var Pinst : pointer;     { A hangszerek mutatója                }
    Ioffs : word;        { A hangszerek eltolása a fájlon belül }
    Num   : word;        { A hangszerek száma                   }
begin
  { Megállapítjuk a hangszerek számát }
  Num:=CmfHead(CMFdata).InstrNumber;
  if Num<129 then
   begin
    { Lekérdezzük a hangszerek eltolását }
    Ioffs:=CmfHead(CMFdata).Instruments;
    asm
     les  dx,CMFdata          { ES:DX a CMF adatokra mutat  }
     mov  Pinst.word,dx
     mov  [Pinst+2].word,ES   { Feltöltjük a Pinst mutatót  }
     mov  bx,$0002
     mov  cx,Num
     les  ax,Pinst
     mov  dx,ES
     add  ax,Ioffs            { Hozzáadjuk a címhez az eltolást }
     pushf
     call Driver              { Beállítjuk a hangszereket       }
     xor  al,$01
     mov  @result,al
    end;
   end
  else Instruments:=false;
end;
```

Instruments: A hangszerek betöltése az FM chip regisztereibe. Ez a rutin a CMF fájl hangszereinek adatait (lásd az SBI fájlok leírásánál) tölti be a megfelelő FM regiszterekbe. Bemenőparaméterként a CMF adatok címét kell megadni egy típus nélküli változóban, amelyet a függvény majd a megfelelő módon kiolvas. A fájlban legfeljebb 128 hangszer lehet. A függvény igaz (*true*) értékkel tér vissza, ha a betöltés sikeres volt.

```
function PlayCMFfile(var CMFdata):boolean;
var
   Clock : word;        { Az óraütem          }
   Offs  : word;        { A zeneadatok eltolása }
   Music : pointer;     { A zene kezdőcíme      }
```

```
begin
 PlayCMFfile:=false;
 { Megállapítjuk a lejátszás óraütemét }
 Clock:=DefaultFreq div CmfHead(CMFdata).Clock;
 { Lekérdezzük a zeneadatok eltolását }
 Offs:=CmfHead(CMFdata).Music;
 asm
   les  dx,CMFdata
   mov  Music.word,dx
   mov  [Music+2].word,ES
   mov  bx,$0004
   mov  ax,Clock
   pushf
   call Driver          { Beállítjuk az időzítést }
 end;
 { Betöltjük a hangszereket }
 if Instruments(CMFdata) then
  begin
   asm
    mov   bx,$0006
    les   ax,Music
    mov   dx,ES
    add   ax,Offs
    pushf
    call  Driver        { Elindítjuk a lejátszást }
    xor   al,$01
    mov   @result,al
   end;
  end;
end;
```

PlayCMFfile: A zene lejátszása. Ezzel a függvénnyel lehet lejátszani a memóriába töltött CMF fájlt. A függvény elvégzi az alapbeállításokat (óraütem, hangszerek), és elindítja a zene visszajátszását. Az egész folyamat az időzítő megszakításai alatt zajlik, ezért a lejátszás alatt bármi mást tehetünk. A függvény bemenőparaméterként a CMF adatok címét várja. Ha a lejátszás bármilyen ok miatt nem indult el (hangszerhiba, adathiba, már lejátszás alatt van stb.), akkor a függvény hamis (*false*) értékkel fog visszatérni.

```
function StopCMFfile:boolean;                          assembler;
asm
  mov  bx,$0007
  pushf
  call Driver
  xor  al,$01
end;
```

StopCMFfile: A lejátszás leállítása. Ezzel a függvénnyel a *PlayCMFfile*-lal elindított lejátszást lehet leállítani. Ha a leállítás sikertelen volt, akkor a függvény hamis (*false*) értékkel tér vissza.

```
function ResetDriver:boolean;                                    assembler;
asm
  mov  bx,$0008
  pushf
  call Driver
  xor  al,$01
end;
```

ResetDriver: A *driver* inicializálása. A függvény alapállapotba állítja a meghajtó programot. Ha az inicializálás nem volt sikeres, akkor a függvény hamis (*false*) értékkel tér vissza.

```
function  PausePlay:boolean;                                     assembler;
asm
  mov  bx,$0009
  pushf
  call Driver
  xor  al,$01
end;
```

PausePlay: Az elindított lejátszás felfüggesztése. Ezzel a függvénnyel a *StopCMFfile*-hoz hasonlóan le tudjuk állítani a zenét, de a *driver* ilyenkor megjegyzi az aktuális pozíciót, és innen egy későbbi időpontban tovább lehet folytatni a lejátszást. Ha a leállítás valamilyen okból sikertelen volt, akkor a függvény hamis (*false*) értékkel tér vissza.

```
function  ContinuePlay:boolean;                                  assembler;
asm
  mov  bx,$000a
  pushf
  call Driver
  xor  al,$01
end;
```

ContinuePlay: A *PausePlay* függvénnyel leállított zene folytatása. Ha a lejátszást nem lehet folytatni, akkor a függvény hamis (*false*) értékkel tér vissza.

```
procedure SetDriverClock(CLK:word);
var C:word;
begin
 C:=DefaultFreq div CLK;
 asm
   mov  bx,$0004
   mov  ax,C
   pushf
   call Driver
 end;
end;
```

SetDriverClock: A *driver* időzítésének beállítása. Ezzel az eljárással a lejátszás sebességét tudjuk módosítani. A *PlayCMFfile* végrehajtása alatt automatikusan meghívódik, így a zene a lejátszás kezdetekor mindig a fájlban tárolt időzítő ütemmel indul.

```
procedure SetSystemClock(CLK:word);
var C:word;
begin
 C:=DefaultFreq div CLK;
 asm
   mov  bx,$0003
   mov  ax,C
   pushf
   call Driver
 end;
end;
```

SetSystemClock: A rendszer időzítésének beállítása. Ez a függvény átprogramozza az időzítő egységet, így a megszakítások számát növeli vagy csökkenti.

```
procedure Transpose(Delta:integer);                    assembler;
asm
  mov  bx,$0005
  mov  ax,Delta
  pushf
  call Driver
end;
```

Transpose: Emelés és süllyesztés. A lejátszás hangmagasságát lehet szabályozni ezzel az eljárással. A negatív értékek az egész hangskála süllyesztését, a pozitív értékek pedig az emelését eredményezik. A változtatás legkisebb egysége egy félhang.

```
BEGIN
 Driver:=nil;
 IntNum:=$00;
END. of unit
```

A unit használatát a következő rövid program mutatja be. (A gyári lemezmellékleten található CMF fájlokkal lehet kipróbálni.) A program egy tetszőleges — 64K-nál kisebb — CMF fájlt betölt a memóriába, és lejátssza azt. Lejátszás közben „Norton Commander-szerűen" folyamatosan változtatja a képernyő tartalmát.

```
program PlayCMF;   { PLAYCMF.PAS }

uses Crt,Fmdrv;

type
   ver = record          { A verziószám tárolása }
```

```
    sub,main:byte;
  end;

var
  f    : file;          { A forrásfájl azonosítója    }
  cmf  : pointer;       { A CMF adatok mutatója       }
  w    : word;
  i    : integer;
  l    : longint;
  Name : string[80];    { A fájl neve                 }
  ID   : string[4];     { A fájl azonosítója ('CTMF') }
  v    : ver;

const
  { Karakter- és színkódok a képernyő módosításához }
  Ch   : array[0..3] of byte=($20,$fa,$2a,$f9);
  Cl   : array[0..7] of byte=($0a,$0f,$0e,$09,$04,$0b,$0c,$0d);

BEGIN
 textattr:=$07;
 clrscr;
 { Megpróbáljuk inicializálni a drivert }
 if not InitDriver then
  begin
   { Kilépés, ha a driver nincs a memóriában }
   writeln('Az SBFMDRV nincs a memóriában');
   writeln('Indítsuk el az SBFMDRV.COM programot!');
   HALT;
  end;

 { Kiírjuk a megszakítás számát és a verziót }
 writeln('Az SBFMDRV megszakítása: INT $',Hex(IntNum));
 w:=GetDriverVersion; move(w,v,2);
 writeln('A driver verziószáma:',v.main,'.',v.sub);

 { A fájl neve }
 if paramcount=0 then Name:='starfm.cmf'
                 else Name:=paramstr(1);

 { Beolvassuk a fájlt a heapbe }
 assign(f,Name);
 reset(f,1);
 l:=filesize(f);
 getmem(cmf,l);
 blockread(f,cmf^,l);
 close(f);

 { Ellenőrizzük, hogy valóban CMF fájlról van-e szó }
 ID[0]:=#4;
 move(cmf^,ID[1],4);
```

```
if ID<>'CTMF' then
 begin
  { Kilépés, ha nem CMF fájl }
  writeln('Ez nem CMF fájl !');
  HALT;
 end;

{ Lejátszás }
writeln('CMF lejátszás indul ... ',Name,' [ENTER]');
readln;

if not PlayCMFfile(CMF^) then
 begin
  { Kilépés, ha a lejátszás indítása sikertelen volt }
  writeln('A fájlt nem lehet lejátszani !');
  HALT;
 end;

clrscr;

{ A képernyő folyamatos változtatása a zene alatt }
while (not keypressed)and(status<>0) do
 begin
  memw[ $b800:random(2000) shl 1] :=
   Cl[ random(8)]*256+Ch[ random(4)];
  for i:=0 to 15 do
   memw[ $b800:random(2000) shl 1] :=$0720;

  mem[ $b800:$0001] :=$00;

  delay(100);
 end; { while }

{ Kiürítjük a billentyűzetbuffert }
while keypressed do readkey;

textattr:=$07;
clrscr;
writeln('Bye !');
{ Leállítjuk a lejátszást }
StopCMFfile;
freemem(cmf,l);
{ Felszabadítjuk a memóriát, kilépés }
END. of program
```

9.3. A VOC fájlok formátuma

A *Creative Labs* — mivel a Sound Blaster kártya a digitálisan mintavételezett hangokat is képes lejátszani — definiálta a VOC fájlokat, amelyek alkalmasak a digitális hangminták tárolására. A digitálisan tárolt hangok sokkal több helyet igényelnek, mint az FM chip által előállított zene, ezért az ilyen hangokat tároló fájlok mérete is általában nagyobb. A tárolás elve is alapvetően más, ezért a VOC fájl szerkezete eltér az előzőekben ismertetett fájlokétól.

A fájl elején található egy rövid fejléc, amely tartalmazza a verziószámot és a formátumazonosítót. Ezek után különálló blokkok következnek, amelyek más-más célt szolgálnak, de a fejrészük ezeknek is azonos.

Eltolás	Adattípus	A mező jelentése
$00...$13	szöveg	a fájl azonosítója („*Creative Voice File*"+*$1A*)
$14,$15	word	a hangminta eltolása a fájlban (általában $001A)
$16,$17	word	a fájl verziószáma (alsó, felső)
$18,$19	word	ellenőrző verziószám (alsó, felső)
$1A...	bájt blokk	VOC blokk

Fájlazonosító: Bájt $00...$13 (*File ID*)

A fájl elején egy szöveg található („Creative Voice File") , majd egy $1A bájt. Ezen mező alapján lehet azonosítani, hogy az adott file VOC formátumú-e, vagy sem.

A hangminta eltolása: Bájt $14,$15 (*Sample offset*)

Az első hangmintát tartalmazó blokk eltolása. Ez általában $001A, mert ennyi a fájl fejlécének hossza, de más érték is elképzelhető. (A VOC fájlokat ugyanis eredetileg a Sound Blaster kártyához találták ki, amelynek még csupán egyetlen 8-bites mono digitális csatornája volt. A később kiadott kártyák viszont túlléptek ezen, így a fájl formátumát is módosítani kellett.)

A VOC fájl verziószáma: Bájt $16,$17 (*Version number*)

A fájl verziószáma. Az első bájt az alverzió száma, a felső pedig a fő verziószám. Az egyes verziók alapján a lejátszók meg tudják különböztetni a tárolt adatok típusát (sztereó/mono, 8/16 bit).

A VOC fájl ellenőrző verziószáma: Bájt $18,$19 (*Control version number*)

Ezt a mezőt az eredeti verziószámból képezik, mégpedig úgy, hogy veszik a verziószám kettes komplemensét és hozzáadnak $1234-et. A *Creative Labs* szerint ezzel még egyszer ellenőrizni lehet, hogy a fájl valóban VOC fájl-e, és a verziószám érvényességét is jelzi.

Adatblokkok: Bájt $1A... (*Data blocks*)

A fejléc után következnek a hangminták és az egyéb vezérlési információkat tartalmazó adatblokkok. Ezekből az adatblokkokból eredetileg csak 8 volt, de a későbbi hangkártyák szükségessé tették újabb blokkok definiálását. A Sound Blaster Pro kiadásáig definiált blokkok a következők:

A blokk típusa	A blokk funkciója
$00 lezáró blokk (*Voice End block*)	a hangkiadás befejezése
$01 hangminta (*Normal sound data*)	a hangkiadás elkezdése
$02 hangminta folytatás (*Subsequent sound*)	a hangkiadás folytatása
$03 üres (néma) blokk (*Silence block*)	a hangkiadás szüneteltetése
$04 megjegyzett cím (*Marker block*)	cím megjegyzése
$05 szöveges megjegyzés (*Message block*)	szöveges információ
$06 ismétlés kezdése (*Repeat block*)	a hang ismétlése, kezdőpont
$07 ismétlés vége (*Repeat end block*)	a hang ismétlése, végpont
$08 bővítés (*Extended block*)	bővítés

Minden blokk első két adata azonos:

1. a blokk típusa (*block type*);
2. a blokk hossza (*block length*).

A típus egyetlen bájt, a hossz pedig egy hárombájtos (24 bites) szó. Ezzel tehát legfeljebb 16 Mbájtos hosszúság adható meg (persze ez bőven elég). A blokk további adatai a blokk típusától függően már mások lehetnek. Vegyük most sorra részletesen ezeket a blokkokat!

Lezáró blokk: $00 (*Voice end block*)

A lezáró blokkot a hangminta végén szokták elhelyezni. A lejátszó programnak minden egyes blokkot fel kell dolgoznia a blokk típusától függően, majd keresnie kell az újabb blokkot. Az utolsó blokk a lezáró blokk, a hang elhallgat, és a lejátszás véget ér.

A „lezáró" blokk szerkezete	
Blokktípus (*block type*)	1 bájt = $00
Blokkhossz (*block length*)	nincs
Adatbájtok (*data bytes*)	nincs

Hangminta: $01 (*Normal sound data block*)

A hangkiadását a „hangminta blokk" kezdi el. A blokk tartalmazza a hangmintákat, a hosszúságot és a mintavételi frekvenciát is. Ezt a blokkot használjuk a leggyakrabban a VOC fájlokban.

A „hangminta" blokk szerkezete	
Blokktípus (*block type*)	1 bájt = $01
Blokkhossz (*block length*)	3 bájt = ???
Mintavételezési állandó (*SR byte*)	1 bájt = ???
Tömörítési típus (*pack byte*)	1 bájt = 0,1,2,3
Adatbájtok (*data bytes*)	? bájt = ???

A blokk hossza tartalmazza a mintavételezési állandót és a tömörítési típust definiáló bájtot is, így a hangminta hossza mindig kettővel kevesebb, mint a blokk hossza.

A visszajátszáshoz elengedhetetlen a felvételkor használt mintavételezési frekvencia ismerete. Erről tájékoztat a mintavételezési állandó, amely 256 lehetséges frekvenciát tud definiálni. Az átszámításhoz a következő egyszerű képletet használhatjuk:

```
Frekvencia = -1000000 div (SR - 256)
```

A mintavételezési állandót a tömörítési típust definiáló bájt követi. A tömörítések természetesen illeszkednek a Sound Blaster hardverdekompressziós eljárásaihoz, így négy különböző tömörítési arány adható meg:

Tárolás	Arány	Tömörítés
8-bites	—	nincs tömörítés
4-bites	2:1	4-bites ADPCM
2.6 bites	3:1	2.6 bites ADPCM
2-bites	4:1	2-bites ADPCM

A tömörítésről tudni kell, hogy a hangkártyák csak a kicsomagolást (dekompresszió) támogatják hardverből, és felvételkor a felvételt készítő programnak kell elvégeznie a becsomagolást!

A tömörítési típus után a hangminták következnek. Egyetlen blokkban nagyjából 16 Mbájt információ tárolható.

Hangminta folytatás: $02 (*Subsequent sound block*)

A hangminták folytatása. Ezt a blokkot akkor kell használni, ha az 1-es blokk túl kevés a hang tárolásához. A VOC fájlt szerkesztő programok akkor használják, amikor a

memóriában nem fér el egyben a hangminta. Az ilyen blokk előtt lennie kell valahol egy 1-es blokknak, hiszen azt kell folytatni. Ezért ebben a blokkban nem adják már meg sem a mintavételi állandót, sem a tömörítési típust (ezeket az információkat az 1-es blokk már tartalmazza).

A „hangminta folytatás" blokk szerkezete	
Blokktípus (*block type*)	1 bájt = $02
Blokkhossz (*block length*)	3 bájt = ???
Adatbájtok (*data bytes*)	? bájt = ???

Üres blokk: $03 (*Silence block*)

A 3-as blokk a hangok közötti szünetek beiktatására szolgál.

Az „üres" blokk szerkezete	
Blokktípus (*block type*)	1 bájt = $03
Blokkhossz (*block length*)	3 bájt = $0003
A szünet hossza (*duration*)	2 bájt = ???
Mintavételezési állandó (*SR byte*)	1 bájt = ???

A blokk mérete minden esetben 3 bájt, mert csupán a beiktatott üres rész hosszát és a mintavételezési állandót kell megadni. Egy üres blokk legfeljebb 64K méretű szünetet tud beiktatni. A mintavételezési állandót ugyanúgy kell kiszámítani, mint az 1-es blokk esetében.

Megjegyzett cím: $04 (*Marker block*)

A különböző események szinkronizálására a címet tároló blokk szolgál. Amikor a lejátszó egy ilyen blokkot talál, akkor meg kell jegyeznie az aktuális pozíciót. Ha később azután utalunk erre, akkor a lejátszónak innen kell ismét lejátszania a hangot. A Sound Blaster könyv szerint ezzel szinkronizálni lehet például a hangot a grafikához. Mindennek persze csak akkor van értelme, ha a *Creative Labs* gyári meghajtóprogramját (*ct-voice.drv*) használjuk a hang lejátszására, ami viszont elég ritka. **:)**

A „megjegyzett cím" blokk szerkezete	
Blokktípus (*block type*)	1 bájt = $04
Blokkhossz (*block length*)	3 bájt = $0002
Bejegyzés (*marker*)	2 bájt = ???

Szöveges megjegyzés: $05 (*Message block*)

A VOC fájlban lehetőség van szöveges információ tárolására is. Erre szolgál a szöveges megjegyzéseket tároló blokk. A szöveg hossza tetszőleges lehet, de a végére kötelezően ki kell tenni egy $00 bájtot.

A „szöveges megjegyzés" blokk szerkezete	
Blokktípus (*block type*)	1 bájt = $05
Blokkhossz (*block length*)	3 bájt = ???
A szöveg (*text data*)	? bájt = ???
A szövegvégjel (*end text*)	1 bájt = $00

Ismétlés kezdése: $06 (*Repeat block*)

Nagyon hasznos szolgáltatás a VOC fájlban, hogy bizonyos hangrészeket ismételni lehet. Erre két blokkot kell használni, a 6-os „ismétlés kezdete" és a 7-es „ismétlés vége" blokkokat.

Az „ismétlés kezdete" blokk szerkezete:	
Blokktípus (*block type*)	1 bájt = $06
Blokkhossz (*block length*)	3 bájt = $0002
Ismétlések száma (*counter*)	2 bájt = ???

A 6-os blokk definiálja, hogy az adott szakaszt hányszor kell megismételni. Az ismétlések száma mindig 1-gyel több, mint a fájlban lévő érték (counter+1), ha pedig ez $ffff, akkor az ismétlésnek nincs vége!

Ismétlés vége: $07 (*Repeat end block*)

A megkezdett ismétlést a 7-es blokk zárja le. Figyeljünk arra, hogy egy megkezdett ismétlésnek mindig legyen befejezése, mert különben nincs ismétlés. Még ennél is fontosabb, hogy csak akkor legyen 7-es blokk a fájlban, ha 6-os is van!

Az „ismétlés vége" blokk szerkezete	
Blokktípus (*block type*)	1 bájt = $07
Blokkhossz (*block length*)	3 bájt = $0000

Bővítés: $08 (*Extended block*)

A „bővítés" blokkot a Sound Blaster Pro kiadásakor tették hozzá az előző blokkokhoz, mert a normál VOC fájl nem tudott sztereó hangmintákat tárolni. A 8-as blokk ugyan a hang különböző paramétereit definiálja, még sincs benne hangminta. Ez azért van így, hogy a régebbi verziójú lejátszók is le tudják játszani az újabb fájlokat. Ha egy régi lejátszó számára ismeretlen blokkot talál, akkor azt egyszerűen átlépi. Ezért a sztereó

hangokat nem a 8-as blokk tárolja, hanem ez csak definiálja a következő blokkhoz a mintavételi és az adatmód paramétereket.

A „bővítés" blokk szerkezete	
Blokktípus (*block type*)	1 bájt = $05
Blokkhossz (*block length*)	3 bájt = $0004
Mintavételezési állandó (*SR byte*)	2 bájt = ???
Adatmód (*data mode*)	2 bájt = ???

A 8-as blokk tehát mindig egy 1-es blokkot előz meg, így előre beállítja a paramétereket, amelyeket az 1-es blokkban már nem kell figyelembe venni. A mintavételezési állandó kiszámítása sztereó módban más:

```
Frekvencia = 65536 - (25600000 div (2*SR))
```

Az adatmód bájt a felvétel módját definiálja:

0 – mono felvétel;
1 – sztereó felvétel.

9.4. A WAV fájlok formátuma

A WAV (*waves*) fájlokat a Microsoft definiálta, mint általános digitális hangmintákat tartalmazó szabványos *Windows* fájlformátumot. (A *Windows* fájlokat egyébként eléggé logikus rendszerbe foglalták, bár első ránézésre kissé bonyolultnak tűnhetnek.) A WAV fájlok elég univerzálisak ahhoz, hogy mind 8- mind 16 bites sztereó és mono hangmintákat tudjanak tárolni. A fájl az ún. *RIFF* formátumot használja, amely a *Windows* multimédiás alkalmazásaihoz készült.

Eltolás	Mező	Hossz	Jelentés
$00	*rID*	$04	RIFF azonosító szöveg („*RIFF*")
$04	*rLEN*	$04	az adatmező hossza (*rDATA length*)
$08	*rDATA*	rLEN	adatmező (*Riff Data field*)

RIFF azonosító szöveg: Bájt $00...$03 (*RIFF ID text*)

A fájl első négy bájtja a „RIFF" szöveg karaktereit tárolja, ez alapján azonosítható, hogy egyáltalán RIFF fájlról van-e szó.

Az adatmező hossza: Bájt $04...$07 (*Data field length*)

A $04-es fájlpozíción az adatmező hosszát találjuk egy *longint* típusú változóban.

Az adatmező: Bájt $08...?? (*Data field*)

Az adatmező tartalmazza a WAV típusú mezőt. Ez ugyanúgy, ahogyan a RIFF fejléc, több részmezőből áll. A következőkben tehát az eltolásokat már a WAV adatmező kezdetéhez kell viszonyítani!

Eltolás	Mező	Hossz	Jelentés
$00	*wID*	$04	WAV azonosító szöveg („*WAVE*")
$04	*Format*	$14	WAV adatformátum (*wave format*)
$18	*Data*	??	WAV audio adathalmaz (*audio data*)

WAV azonosító szöveg: Bájt $00...$03 (*WAVE ID text*)

A fájl első négy bájtja a „WAVE" szöveg karaktereit tárolja, ez alapján azonosítható, hogy a RIFF fájl WAV adatokat tárol-e. A RIFF ugyanis egyéb adatok tárolására is képes.

WAV adatformátum leíró mező: Bájt $04...$1B (*WAVE data format*)

A $18 bájtból álló formátumleíró mező az audioadatok tárolásáról, adatábrázolásáról és a mintavételi frekvenciáról ad információt.

Eltolás	Mező	Hossz	Jelentés
$04	*fID*	$04	formátumleíró azonosító („*fmt* ")
$08	*fLEN*	$04	a leírómező hossza
$0C	*wFormatTag*	$02	hullámtárolási mód
$0E	*nChannels*	$02	a csatornák száma
$10	*nSamplePerSec*	$04	mintavételezési frekvencia
$14	*nAvgBytesPerSec*	$04	másodpercenkénti átviteli sebesség
$18	*nBlockAlign*	$02	az adatok blokkhatárai
$1A	*FormatSpecific*	$02	adatformátum-definiáló mező

Formátumleíró mező azonosító szöveg: Bájt $04...$07 (*"fmt "*)

Ez a négy bájt a formátumleíró mezőt azonosítja, ahol az utolsó karakter mindig egy *space*, a szöveg pedig „fmt " (a kisbetű fontos).

A leírómező hossza: Bájt $08...$0B (*Format specifier field length*)

A leírómező hossza az azonosító nélkül mindig $10 bájt, ezért ennek a mezőnek is mindig ennyi az értéke.

Hullámtárolási mód: Bájt $0C,$0D (*Wave format*)

A hangminta tárolásának módja. Ez a leggyakrabban $0001, amely a nem tömörített egyszerű hangminta jele.

A csatornák száma: Bájt $0E,$0F (*Mono/stereo channels*)

Ez a mező a felvétel módját adja meg:
$0001 – mono felvétel;
$0002 – sztereó felvétel.

Mintavételezési frekvencia: Bájt $10...$13 (*Playback frequency*)

A mező a visszajátszáshoz szükséges mintavételi frekvenciát tartalmazza. Tipikus értékek a következők:
$2B11 – 11025 Hz;
$5622 – 22050 Hz.

Másodpercenkénti átviteli sebesség: Bájt $14...$17 (*Number of byte per second*)

Átlagos átviteli sebesség. Ez az érték megmondja, hogy egy másodperc alatt átlagosan mennyi adatot kell átvinni a fájlból (memóriából) a hangkártyára (DAC-ra). Értékét a következő képlet alapján lehet kiszámítani:

*nAvgBytesPerSec = nChannels * nSamplePerSec *(BitsPerSample* **shr** *3)*

A hangadatok blokkhatárai: Bájt $18,$19 (*Block aligment of datas*)

Ez a mező az egyszerre — egy csomagban, egy blokkban — kikerülő bájtok számát adja. Értékét a következő képlettel lehet meghatározni:

*nBlockAlign = nChannels * (BitPerSample* **shr** *3)*

Adatformátum-definiáló mező: Bájt $1A,$1B (*Format specific area*)

Az adatok formátumát adja meg, amely nem más, mint az előbbi két képletben emlegetett *BitPerSample* érték.

8-bites hangmintáknál: *BitPerSample* = $0008
16 bites hangmintáknál: *BitPerSample* = $0010

A formátumleíró mező után az adatmező következik. Az adatmező is tartalmaz egy azonosító szöveget és egy hosszúságot, de egyéb kiegészítő adatra itt nincs szükség, hiszen minden mást ismerünk már az előbbi mezőkből.

Eltolás	Mező	Hossz	Jelentés
$00	*dID*	$04	az adatmező azonosítója („*DATA*")
$04	*dLEN*	$04	az adatok hossza
$08	*Datas*	*dLEN*	a felvétel hangmintái

9.5. A MOD fájlok formátuma

A MOD (*Module*) fájlokat eredetileg a Commodore AMIGA gépekre fejlesztették ki, de mostanára már a PC kártyákhoz is készülnek MOD szerkesztők és lejátszók. Az AMIGA esetében ez a formátum annyira ésszerű és hardverhez illeszkedő, hogy a profi programozók is ezt használják a játékokban és a demoprogramokban. A MOD fájlok szerkezetének megértéséhez ismerni kell az AMIGA hangkeltési lehetőségeit. Nézzük meg ezt egy kicsit részletesebben!

Az AMIGA-nak négy, egymástól teljesen független digitális hangcsatornája van, amelyek mindegyike külön DMA csatornát használ, így az egyidejű megszólaltatás nem okoz gondot. (A legnagyobb mintavételi frekvencia csupán 20 kHz körül van, de számunkra ez most lényegtelen.) A négy csatornából kettő csak a jobb, kettő pedig csak a bal oldalon tud szólni. A csatornák 8-bitesek. Ezek a csatornák igen hasonlítanak a Sound Blaster digitális csatornájához, de másképpen kell őket programozni. Ez azért van így, mert az AMIGA szinte minden fontosabb feladatot céláramkörök segítségével old meg, ezeket pedig DMA átvitellel kell programozni.

A MOD fájlok tehát ehhez a koncepcióhoz igazodnak. A fájlban digitális hangmintákkal tárolják a hangszereket. A hangszerek mintavételezési frekvenciáját változtatni lehet, ezáltal a hangmagasság emelkedik vagy süllyed. A digitális hangszerek miatt a hangzás jobb, mint az egyszerű FM szintézissel előállított hangszereknél. A hangminták mellett a zenét is tárolják. A hardverhez igazodva négy különálló csatorna van, amelyeket az ún. *patternek*, azon belül pedig a csatornákhoz tartozó *note*-ok vezérelnek. A *note*-ban a hang mellett speciális vezérlőparancsok is lehetnek. Az AMIGA DMA alatt, megszakítás segítségével képes lejátszani a MOD fájlt, a processzor elenyésző idejének felhasználásával.

Amilyen könnyű lejátszania egy MOD-ot az AMIGA-nak, legalább annyira nehéz feladat ez a Sound Blaster család tagjai számára. A Sound Blasternek ugyanis csupán egyetlen digitális csatornája van, amely sztereó módban ráadásul összefésülve kezeli a két oldal hangmintáit. A négy különálló digitális mintát tehát össze kell keverni — a parancsoknak megfelelő módon módosítani —, és egyetlen csatornán kiadni. Ez bizony nem könnyű feladat!

Más a helyzet a Gravis Ultrasound esetében, hiszen itt nemhogy négy, de harminckettő digitális csatorna van. Ezért a Gravis kártyára sokkal könnyebb MOD lejátszót készíteni.

Térjünk vissza a fájl szerkezetéhez! Az egyes lejátszók különböző verziói miatt jelentős eltérések vannak. A legelső MOD fájlok — amelyeket a *Soundtracker* használt — csak 15 digitális hangszert tároltak. Az újabb verziókban a *Startracker* programozói a hangszerkészletet kibővítették 31-re. Mindkét formátumban négy csatornát lehetett használni. Ma már olyan fájlok is vannak, amelyek nyolc csatornát tárolnak. (Ezekhez persze megfelelő szerkesztő és lejátszó program is szükséges. A Gravis lemezmellékletein találhatók ilyen lejátszók.) Mi most a legelterjedtebb formátumot nézzük meg: a 31 hangszeres négycsatornás MOD fájlokat.

Modulnév: Bájt $00...$13 (*Module name*)

Az első húsz bájt a fájl (a modul) nevét tartalmazza. Itt legfeljebb egy 19 karakteres nevet lehet megadni, mert a nevet mindenképpen egy $00 bájttal kell lezárni.

Hangszerek leírása: Bájt $14...$3B5 (*Instruments*)

A modul neve után a hangszerek leírása következik. A fájlban 31 hangszer lehet, és minden hangszerhez 30 bájt leíró adatterület tartozik. Egy hangszer leírótáblázata a következő alakú (a táblázat az első hangszer eltolásait mutatja !!):

Eltolás	Adat	Jelentés
$14...$29	*Instrument name*	a hangszer neve
$2A,$2B	*Instrument length*	a hangszer hangmintájának hossza
$2C	*Finetune*	a hangszer finomszabályozója
$2D	*Volume*	a hangszer alaphangereje
$2E,$2F	*Repeat loop*	a hangszer ismétlésének kezdete
$30,$31	*Repeat length*	a hangszer ismétlési hossza

Instrument name: A hangszer neve. Minden hangszernek lehet egy legfeljebb 21 karakterből álló neve, amelyet egy $00 bájt zár le.

Instrument length: A hangszer hangmintájának hossza. Ez a két bájt wordben adja meg a hangminta hosszát, ráadásul a Motorola 68000 processzorcsalád számábrázolási

rendszere szerint. A 68000-es családnál az Intel processzoroktól eltérően egy 16 bites word adat esetében az első bájt mindig a felső (15...8 bitek), a második pedig mindig az alsó (7...0 bitek). Ez a fájl további adataira is vonatkozik! (Ne felejtsük el, hogy egy AMIGA fájlról van szó, amely igazodik a gép processzorához.) A két bájtból a következő képlettel lehet meghatározni a hangszer hangmintájának valódi hosszát:

```
Hossz:= 2*[ 256* ($2A)+($2B)]
```

Ez a képlet megadja a bájtokban mért hosszúságot.

Finetune: A hangszer finomszabályozása. Ez a bájt adja meg a hangmagasság finombeállítását minden egyes hangszerhez. Csak az alsó négy bit értéke számít, a felső négy bit mindig 0! A számot négybites kettes komplemens számként kell értelmezni, tehát –8...+7 közötti értékek találhatók benne.

Volume: A hangszer hangereje a kezdéskor. Ez a bájt a hangerőt adja meg a 00...64 ($00...$40) tartományban.

Repeat loop: Az ismétlés kezdete. A MOD fájl hangszereit ismételni lehet. Ez a két bájt egy eltolást tartalmaz wordben a hangszer adatainak kezdetétől, és azt mondja meg, hogy a digitális minták közül melyiktől kezdve kell ismételni. A számot a hosszúságnál (*Instrument length*) megadottak szerint kell értelmezni!

Repeat length: A megismételt szakasz hossza. A hangminta ismétlését a *Repeat Loop* címtől kell kezdeni, és a *Repeat length* hosszúságú adatokat kell ismételni. A hosszúságot wordben mérjük! Ha a *Repeat loop*=$0000 és a *Repeat length*=$0002, akkor nem kell ismételni.

Minden hangszert ezen a módon adnak meg. A hangszerleíró blokk így 31*30, azaz 930 bájt területet igényel.

A zene hossza: Bájt $3B6 (*Module length*)

A zene hosszát a felhasznált *patternek* számával kell megadni. Ez nem szükségszerűen ugyanannyi, mint a létrehozott *patterneké*. Előfordulhat például, hogy létrehozunk 10 *patternt*, de ebből csak 5-öt használunk fel a zenében. Ekkor ez a bájt 5-öt tartalmaz.

CIA-A sebesség: Bájt $3B7 (*CIA A speed*)

Ez a bájt az AMIGA egyik speciális áramköréhez, a CIA-A chip programozásához szükséges, a PC-n ez lényegtelen.

Patternsorozat: Bájt $3B8...$437 (*Pattern positions*)

A fájlban a lejátszandó *patternek* sorozata. Ez a 128 bájtos terület a *patternek* sorszámait tartalmazza abban a sorrendben, ahogy le kell őket játszani. Minden bájt csak 00...63 értékeket tartalmazhat, mert a MOD fájl egyszerre legfeljebb 64 *patternt* tud tárolni.

Bejegyzés: Bájt $438...$43B (*Marking files*)

A szerző bejegyzése. Ezen a helyen egy négybetűs bejegyzés található, általában „M.K." vagy „FLT4". A szöveg információt ad a hangszerek és a csatornák számáról. A 31 hangszeres, négycsatornás MOD fájloknál ez a két bejegyzés szokott előfordulni. Ha ezeket a karaktereket nem lehet itt megtalálni, akkor nagy valószínűséggel egy régebbi 15 hangszeres MOD fájlról van szó.

Patternek

Ezek után a *patternek* következnek. Minden *pattern* négy csatornához egy négybájtos *note*-ot tartalmaz, és egy pattern 64 *note*-ból áll. Ezért egy pattern statikusan 16*64, azaz 1024 bájt hosszú. A *note*-ok egymás után helyezkednek el, azaz a 0. csatorna *note*-ja az első négy bájtot foglalja le, majd az 1. csatorna *note*-ja a második négy bájtot és így tovább. Egy *note* négy bájtja felfogható egy *longword* típusú adatnak, azaz egy 32 bites előjel nélküli számnak, amely persze a 68000-es processzorok adatábrázolása szerint értendő!

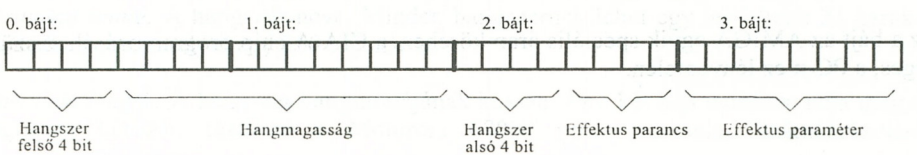
Egy *pattern* egy sora négy *note*-ot tartalmaz:

Sorszám	Eltolás	0. csatorna	1. csatorna	2. csatorna	3. csatorna
$00 (00)	$00	note (4 bájt)	note (4 bájt)	note (4 bájt)	note (4 bájt)
$01 (01)	$10	note (4 bájt)	note (4 bájt)	note (4 bájt)	note (4 bájt)
$02 (02)	$20	note (4 bájt)	note (4 bájt)	note (4 bájt)	note (4 bájt)
...	...				
$3F (63)	$3F0	note (4 bájt)	note (4 bájt)	note (4 bájt)	note (4 bájt)

Noteok

Egy *note* felépítése a következő:

0. bájt	1. bájt	2. bájt	3. bájt
Instrument,Pitch	*Pitch*	*Instrument,Effect*	*Effect argument*



Ez a kiosztás a 31 hangszeres MOD fájlokra vonatkozik. A régebbi 15 hangszeres fájlok esetében a 0..3 bitek nem a hangszert, hanem a modul hangmagasságát tartalmazták, ezért nem lehetett 15 hangszernél többet használni. A 31 hangszeres fájloknál viszont a 0. bájt felső négy bitjén és a 2. bájt felső négy bitjén összevontan 8 biten lehet tárolni a hangszer számát, így nincs akadálya 256-féle hangszernek sem, de a hangszerek elhelyezése miatt itt is csak 31-et lehet használni. A 0. bájt alsó fele és az első bájt a hang magasságát adja. Ezt az ún. periódustábla alapján lehet meghatározni. A periódustábla felépítése a következő:

Oktáv	C	C#	D	D#	E	F	F#	G	G#	A	A#	H
1.	856	808	762	720	678	640	604	570	538	508	480	453
2.	428	404	381	360	339	320	302	285	269	254	240	226
3.	214	202	190	180	170	160	151	143	135	127	120	113

A táblázat értékei az AMIGA hardverének beállításából következnek, ezért PC-n ezeket csupán a hangok azonosítására lehet felhasználni!

9.5.1. Effektus parancsok

A *note* utolsó adata az effektus, amelynek parancskódját a 2. bájt alsó négy bitje adja, paraméterét pedig a 3. bájt. Ezekkel a parancsokkal különleges zenei hatásokat lehet megadni. A megadható parancsok a következők:

Parancskód	Effektus
$00	semmi vagy arpeggio
$01	portamentonövelés
$02	portamentocsökkentés
$03	hangportamento
$04	vibrátó
$05	hangportamento és hangerőcsúsztatás
$06	vibrátó és hangerőcsúsztatás
$07	tremoló
$08	nincs kihasználva
$09	mintaeltolás
$0A	hangerőcsúsztatás
$0B	ugrás új *patternpozícióra*
$0C	hangerőbeállítás
$0D	*patternleállítás*
$0E	extra parancsok
$0F	sebességbeállítás

Parancs $00: Semmi vagy arpeggio (*None/Arpeggio*)

Az effektus paraméterétől függően ez a parancs kétféle feladatot lát el. Ha a paraméter $00, akkor nincs semmiféle effektus, ha pedig nem $00, akkor akkordok szimulálása történik. A paraméterbájt két fele egy-egy relatív távolságot ad meg félhangokban az alaphangtól mérve. Így ezt a három hangot gyorsan egymás után leütve akkordhatás érhető el.

Példa: A C-dúr akkord előállítása. Az akkord hangjai: C, E, G (E=C+4, G=C+7)
A *note* alakja: C–3 $00 $47.

Parancs $01: Portamentonövelés (*Portamento Up*)

A teljes hangzás frekvenciájának megemelése egy egységgel. A parancs hatására a hangok magasabban fognak szólni. Figyeljünk arra, hogy a H–3 (113) hangnál nem lehet magasabbra lépni. A paraméterbájt a növelés sebességét adja meg.

Példa: A hangmagasság megnövelése 1 egységgel, 5-ös sebességgel.
A *note* alakja: C–3 $01 $05.

Parancs $02: Portamentocsökkentés (*Portamento Down*)

A $01-es parancshoz hasonlóan működik a $02-es, de itt lefelé lehet vinni a hangmagasságot. Figyeljünk arra, hogy nem lehet lejjebb menni, mint a C–1 (856) hang. A paraméterbájt a csökkentés sebességét adja meg.

Példa: A hangmagasság csökkentése 1 egységgel, 3-as sebességgel.
A *note* alakja: C–3 $02 $03.

Parancs $03: Hangportamento (*Tone portamento*)

A hangportamento az előző két parancshoz hasonlóan a hangmagasságot változtatja meg, de itt a célhangot kell megadni. A lejátszó az előző hangtól a megadott célhangig elkezdi módosítani a hangmagasságot. A módosítás a paraméterrel megadott sebességgel történik. Azzal nem kell törődni, hogy felfelé, vagy lefelé kell-e lépni, ennek meghatározása a lejátszó feladata.

Példa: Csúsztatás C–3-ról D–3-ra 4-es sebességgel.
A *note* alakja: C–3 $00 $00 (előző *note* !);
D–3 $03 $04.

Parancs $04: Vibrátó (*Vibrato*)

Vibrátósebesség és -mélység megadása. A parancs segítségével FM modulációt lehet elérni. A paraméterbájt felső négy bitje a vibráció sebességét, alsó négy bitje pedig a vibráció mélységét határozza meg.

Példa: Egy 3-as sebességű 2-es mélységű vibráció előállítása.
A *note* alakja: C–3 $04 $32.

Parancs $05: Hangportamento és hangerőcsúsztatás (*Tone portamento+ Volume slide*)

Ezzel a paranccsal a megkezdett hangportamentót ($03) lehet folytatni a hangerő fokozatos változtatásával. A paraméterbájtban a felső négy bit a hangerő növelését, az alsó négy bit pedig a hangerő csökkentését adja meg. Természetesen értelmetlen mindkét értéket egyszerre megadni !

Példa: A portamento folytatása és a hangerő növelése 3 egységgel.
A *note* alakja: C–3 $05 $30.
Példa: A portamento folytatása és a hangerő csökkentése 2 egységgel.
A *note* alakja: C–3 $05 $02.

Parancs $06: Vibrátó és hangerőcsúsztatás (*Vibrato+ Volume slide*)

A $06-os parancs a megkezdett vibráció folytatása közben lehetővé teszi a hangerőszabályozást. A paraméterbájtban a felső négy bit a hangerő növelését, az alsó négy bit pedig a hangerő csökkentését adja meg. Nincs értelme, hogy mindkét értéket egyszerre megadjuk!

Példa: A vibráció folytatása és a hangerő növelése 7 egységgel.
A *note* alakja: C–3 $06 $70.
Példa: A vibráció folytatása és a hangerő csökkentése 4 egységgel.
A *note* alakja: C–3 $06 $04.

Parancs $07: Tremoló (*Tremolo*)

A $07-es paranccsal lehet megadni a tremoló (amplitúdómoduláció) sebességét és mélységét. A paraméterbájt felső négy bitjén a sebességet, az alsó négy biten pedig a mélységet kell definiálni.

Példa: Egy 3-as sebességű, 7-es mélységű tremoló beállítása.
A *note* alakja: C–3 $07 $37.

Parancs $09: Hangmintaeltolás (*Sample offset*)

Ez a parancs lehetővé teszi, hogy egy hangmintát ne az elejétől, hanem más pozíciótól játsszon le a lejátszó program. Egy eltolást kell megadni, amelyet a hangminta elejétől kell számítani. A parancsot leginkább a beszédnél szokták alkalmazni.

Példa: Egy hangminta kezdése a $2500. bájttól.
A *note* alakja: C–3 $09 $25.

Parancs $0A: Hangerőcsúsztatás (*Volume slide*)

A hangerő szabályozása. A parancs segítségével a $05-ös és a $06-os parancs mintájára a hangerőt lehet szabályozni.

Példa: A hangerő növelése 3 egységgel.
A *note* alakja: C–3 $0A $30.
Példa: A hangerő csökkentése 6 egységgel.
A *note* alakja: C–3 $0A $06.

Parancs $0B: Ugrás új *patternpozícióra* (*Position jump*)

Ezzel a paranccsal direkt módon lehet ugrálni az egyes *patternek* között. A paraméterbájtban egy patternszámot kell megadni, ahonnan a zenét folytatni kell. Ha ez a szám $01, akkor természetesen újrakezdődik a lejátszás.

Példa: Ugrás a $03-as *patternre.*
A *note* alakja: C–3 $0B $03.

Parancs $0C: Hangerő-beállítás (*Set volume*)

A hangerő közvetlen beállítása. Ez a parancs az előző hangerő figyelembevétele nélkül a paraméterként megadott hangerőt állítja be. Emlékezzünk rá, hogy a hangerőt csak a $00...$40 tartományban lehet állítani!

Példa: A hangerő beállítása $30-ra.
A *note* alakja: C–3 $0C $30.

Parancs $0D: Patternleállítás (*Pattern break*)

Egy *pattern* lejátszásának azonnali leállítása. A parancs leállítja az aktuális pattern lejátszását, és a paraméterként megadott *patternpozíción* kezdi el játszani a következő *patternt.* Az egyes lejátszók kétféleképpen értelmezhetik ezt a parancsot. A fejlettebbek a fenti módon működnek, a régebbiek viszont egyszerűen — a paraméterbájtot figyelmen kívül hagyva — a legelejétől kezdik el lejátszani a következő *patternt.*

Példa: Ugrás a következő *patternre*.
A *note* alakja: C–3 $0D $00.
Példa: Ugrás a következő *patternre*, lejátszás a $20-as pozíciótól.
A *note* alakja: C–3 $0D $20.

Parancs $0E: Extra parancsok (*Extra commands*)

A bővített parancskészlet használatát teszi lehetővé. Ilyenkor a paraméter bájt egyik fele újabb parancsokat definiál, így csak 4 bit marad a paraméterekre. Később bővebben is szó lesz még erről.

Parancs $0F: Sebességbeállítás (*Set speed*)

A lejátszás sebességének direkt módosítása. A parancs hatására a paraméterbájtban megadott sebesség lesz az új lejátszási sebesség. Ezt az értéket 5 biten lehet megadni, azaz a paraméter csak a $00...$1F tartományba eshet!

Példa: A lejátszás sebességének beállítása $10-re.
A *note* alakja: C–3 $0F $10.

9.5.2. Extra effektus parancsok

Ha a parancs kódja $0E, akkor a paraméterbájt felső négy bitje egy újabb parancskészletből választja ki a megfelelő extra parancsot. Ebben az esetben a paraméterbájtnak csak az alsó négy bitjét használhatjuk paraméterek megadására. Az extra parancskészlet a következő:

Parancskód	Extra effektus
$0E,$00	szűrő be- és kikapcsolás
$0E,$01	csúsztatás felfelé
$0E,$02	csúsztatás lefelé
$0E,$03	glisszandóvezérlés
$0E,$04	vibrációvezérlés
$0E,$05	finombeállítás
$0E,$06	*patternciklus*
$0E,$07	tremolóvezérlés
$0E,$08	nincs kihasználva
$0E,$09	hang újraindítása
$0E,$0A	finom hangerőnövelés
$0E,$0B	finom hangerőcsökkentés
$0E,$0C	hangkivágás
$0E,$0D	hangkésleltetés
$0E,$0E	*patternkésleltetés*
$0E,$0F	ciklusinvertálás

Parancs $0E,$00: Szűrő be- és kikapcsolás (*Filter on/off*)

Az AMIGA szűrőjének be- és kikapcsolása. A régebbi *AMIGA 500* és *AMIGA 2000* gépeken az audioszűrőt lehetett vezérelni ezzel a paranccsal.

Példa: A szűrő bekapcsolása.
A *note* alakja: C–3 $0E $00.
Példa: A szűrő kikapcsolása.
A *note* alakja: C–3 $0E $01.

Parancs $0E,$01: Csúsztatás felfelé (*Fineslide up*)

A *portamento up* parancshoz hasonlóan ez a parancs is megemeli a hang magasságát, de csak egyszer, a hangkiadás legelején.

Példa: Emelés 2 egységgel.
A *note* alakja: C–3 $0E $12.

Parancs $0E,$02: Csúsztatás lefelé (*Fineslide down*)

A *portamento down* parancshoz hasonlóan ez a parancs is csökkenti a hang magasságát, de csak egyszer, a hangkiadás legelején.

Példa: Süllyesztés 1 egységgel.
A *note* alakja: C–3 $0E $21.

Parancs $0E,$03: Glisszandóvezérlés (*Glissando control*)

A glisszandó parancsot a hangportamento paranccsal együtt szokták használni. Ha aktív, akkor a portamento egy fél hanggal csúsztatja a hangot az egyenletes növelés helyett.

Példa: A glisszandó bekapcsolása.
A *note* alakja: C–3 $0E $31.
Példa: A glisszandó kikapcsolása.
A *note* alakja: C–3 $0E $30.

Parancs $0E,$04: Vibrációvezérlés (Vibrato control)

A vibrátor hullámalakjának definiálása. A vibrációhoz négyféle hullámalakot lehet megadni.

Példa: Szinuszos vibráció. (Ez az alapbeállítás is.)
A *note* alakja: C–3 $0E $40.
Példa: Csillapodó vibráció.
A *note* alakja: C–3 $0E $41.
Példa: Négyszög vibráció.
A *note* alakja: C–3 $0E $42.
Példa: Véletlenszerű vibráció.
A *note* alakja: C–3 $0E $43.

Parancs $0E,$05: Finombeállítás (*Fine tune*)

A hangszer *finetune* ($2C) értékének beállítása. A paraméterbájt alsó négy bitje a *finetune* értéket adja kettes komplemens alakban.

Példa: A *finetune* beállítása háromra.
A *note* alakja: C–3 $0E $53.

Parancs $0E,$06: *Patternciklus* (*Pattern loop*)

Ezzel a paranccsal egy pattern belsejében lehet hangismétlő ciklusokat előállítani.

Példa: A belső ciklus kezdőpontjának beállítása.
A *note* alakja: C–3 $0E $60.
Példa: A belső ciklus négyszeri ismétlése.
A *note* alakja: C–3 $0E $64.

Parancs $0E,$07: Tremolóvezérlés (*Tremolo control*)

A tremoló hullámalakjának definiálása. Az AM modulációhoz négyféle hullámalakot lehet megadni.

Példa: Szinuszos amplitúdómoduláció. (Ez az alapbeállítás is.)
A *note* alakja: C–3 $0E $70.
Példa: Csillapodó amplitúdómoduláció.
A *note* alakja: C–3 $0E $71.
Példa: Négyszög amplitúdómoduláció.
A *note* alakja: C–3 $0E $72.
Példa: Véletlenszerű amplitúdómoduláció.
A *note* alakja: C–3 $0E $73.

Parancs $0E,$09: Hang újraindítása (*Note retrigger*)

A parancs a megadott ütem szerint újraindítja a hangot, így az többször szólal meg. Az ütemszám csak kisebb lehet, mint az aktuális sebesség!

Példa: Újraindítás egyszer.
A *note* alakja: C–3 $0E $91.

Parancs $0E,$0A: Finom hangerőnövelés (*Fine volume up*)

A parancs a normál hangerőnöveléshez hasonlóan működik, de ebben az esetben csak egyszer, a hangkiadás legelején változik a hangerő.

Példa: A hangerő növelése 4 egységgel.
A *note* alakja: C–3 $0E $A4.

Parancs $0E,$0B: Finom hangerőcsökkentés (*Fine volume down*)

A parancs a normál hangerőcsökkentéshez hasonlóan működik, de ebben az esetben csak egyszer, a hangkiadás legelején változik a hangerő.

Példa: A hangerő csökkentése 3 egységgel.
A *note* alakja: C–3 $0E $B3.

Parancs $0E,$0C: Hangkivágás (*Note cut*)

Ezzel a paranccsal különlegesen rövid hangokat lehet előállítani. A paraméterként megadott ütemeket a lejátszó kivágja a hangból, így lerövidíti azt. A kivágás csak kisebb lehet, mint az aktuális sebesség (például 3-as sebesség mellett nem lehet 5 ütemet kivágni)!

Példa: Három ütem kivágása.
A *note* alakja: C–3 $0E $C3.

Parancs $0E,$0D: Hangkésleltetés (*Note delay*)

A hang késleltetése a megadott ütemig.

Példa: Késleltetés öt ütemen keresztül.
A *note* alakja: C–3 $0E $D5.

Parancs $0E,$0E: *Patternkésleltetés* (*Pattern delay*)

Egy teljes *pattern* késleltetése. Ebben az esetben hangok számában kell megadni a késleltetést. (Minden más effektus aktív a késleltetés alatt.)

Példa: A *pattern* késleltetése 6 hanggal.
A *note* alakja: C–3 $0E $E6.

Parancs $0E,$0F: Ciklusinvertálás (*Invert loop*)

Ez az effektus rövid ciklusokban alkalmazható. A parancs hatására a lejátszó egyszerűen megfordítja a ciklus bájtjait.

Példa: Ciklus megfordítása négyes sebességgel.
A *note* alakja: C–3 $0E $F4.

9.5.3. Hangminták

A MOD fájlok utolsó komponensei a hangminták, amelyek a *patternek* mögött helyezkednek el. A hangleíró táblában minden egyes hangszernek megvan a hossza, így a címeket egymás után meg lehet határozni, hiszen a patternek száma és hossza ismert. Emlékezzünk rá, hogy a fájl word méretben tárolja a minta hosszát, ezért a valódi hossz ennek kétszerese, és minden hangminta páros számú bájtból áll!

9.5.4. MOD fájlok kilistázása

A MOD fájlok ugyan nagyon jól hangzanak, ha a megfelelő lejátszó programot sikerül beszerezni, de küllemre éppen olyan csúnyák, mint akármelyik bináris fájl. Sokszor pedig hasznos lenne, ha a fájl tartalmát valamilyen „emberi fogyasztásra alkalmas" formában is meg lehetne nézni. Az sem volna rossz, ha a fájlban tárolt digitális hangmintákat valahogyan ki lehetne emelni és máshol felhasználni. Ezeket a feladatokat oldja meg a most következő program. A MOD fájl *patternjeit* egy szöveges fájlba másoljuk, ahol csatornánként és hangonként minden adat szerepel. Ha a hangmintákra van szükségünk, akkor azokat is menteni tudja *'sample.xx'* néven.

```
uses crt; { mod2txt.pas }

const
    PeriodTABLE : array[ 0..35] of word=(
    856,808,762,720,678,640,604,570,538,508,480,453,
    428,404,381,360,339,320,302,285,269,254,240,226,
```

```
214,202,190,180,170,160,151,143,135,127,120,113);
```

Periodtable: Hangjegy-azonosítási táblázat. A fájlban a hangok *period* mezőjében ezek az értékek találhatók meg. Ez alapján lehet azonosítani az oktávot és a hangjegyet.

```
NoteName : array[ 0..11] of string[ 2] =
('C ','C#','D ','D#','E ','F ','F#','G ','G#','A ','A#','H ');
```

NoteName: A hangjegyek nevei.

```
Hd : array[ 0..15] of char='0123456789ABCDEF';
```

Hd: Segédtáblázat a számok hexadecimális alakjának kiírásához.

```
type
  Pmem = ^Tmem;                    { Memóriamutató          }
  Tmem = array[ 0..65534] of byte; { Memória                }

  PpatternData = ^Tpatterndata;          { Patternmutató    }
  TpatternData = array[ 0..3] of byte;   { Pattern           }

  Tinstrument = object                   { Hangszer         }
   Name      : string[ 22];   { A hangminta neve     }
   Length    : longint;       { A hangminta hossza   }
   Finetune  : byte;          { Finombeállítás       }
   Volume    : byte;          { Alaphangerő          }
   Rpoint    : longint;       { Ismétlési kezdőpont  }
   Rlength   : longint;       { Ismétlési hossz      }
   procedure Init(Data:Pmem);     { Beállítás        }
   procedure Show;                { Megjelenítés     }
  end;
```

Tinstrument: Egy digitális hangminta tárolása. A *Tinstrument* egy egyszerű statikus objektum, amely a hangszerek paramétereinek tárolására szolgál. A mezők nevei magukért beszélnek. Az objektumnak csupán két metódusa van. Az *Init* egy forrásadatból kiszámítja a mezők értékeit, a *Show* pedig szövegesen megjeleníti azokat.

```
Tnote = object
 Instrument : byte;        { Hangszer(hangminta)    }
 Note       : byte;        { Hangjegy               }
 Octave     : byte;        { Oktáv                  }
 Effect     : byte;        { Effektus parancs       }
 Parameter  : byte;        { Effektus paraméter     }
 procedure Init(Data:PpatternData);
 function  Show:string;
end;
```

Tnote: Egy hangjegy tárolása. Szintén egy statikus objektum, amely a *pattern* egy hangjegyét tárolja.

```
var
   t : text; { A célfájl azonosítója }

{ Egy bájt hexadecimális alakjának meghatározása }
function Hex(x:byte):string;
begin
 Hex:=Hd[ x shr 4] +Hd[ x and $0f] ;
end;

{ Egy $00-val lezárt sztring kiolvasása a fejlécből }
function GetString(const Src:Pmem):string;
var s : string;
    i : byte;
begin
 s:=''; i:=0;
 while Src^[ i]<>$00 do
  begin
   s:=s+Chr(Src^[ i] ); inc(i);
  end;
 GetString:=s;
end;

procedure GetPeriod(Period:word; var Note,Octave:byte);
var i,j:byte;
begin
 Note:=$ff;
 Octave:=$ff;
 if Period=$0000 then exit;
 for i:=0 to 2 do
  for j:=0 to 11 do
   if Period=PeriodTABLE[ i*12+j] then
    begin
     Note:=j;
     Octave:=i+1;
    end;
end;
```

GetPeriod: A *note period* mezője alapján ez az eljárás meghatározza a hangjegyet és az oktávszámot. Ha a keresett periódusérték nincs a táblázatban — azaz $0000 —, akkor az adott *patternpozícióban* nem kell új hangot megszólaltatni.

```
procedure Tnote.Init(Data:PpatternData);
begin
 Instrument:=(Data^[ 0] and $f0) or (Data^[ 2] shr 4);
 GetPeriod((Data^[ 0] and $0f)*256+Data^[ 1] ,Note,Octave);
```

```
 Effect     :=Data^[ 2] and $0f;
 Parameter  :=Data^[ 3] ;
 if Effect=$0E then
  begin
   Effect:=(Data^[ 2] shl 4) or (Data^[ 3] shr 4);
   Parameter:=Data^[ 3] and $0f;
  end;
end;
```

Tnote.Init: Egy hangjegy paramétereinek meghatározása. A *Tnote* objektum *Init* metódusa az adott *patternadat* alapján előállít egy hangjegyet. Ezt később a *Show* metódussal jelenítjük meg.

```
function  Tnote.Show:string;
begin
 if Note=$ff then Show:='------'+
    '['+Hex(Effect)+','+Hex(Parameter)+']'
 else
 Show:=NoteName[ Note]+'-'+Chr(Octave+$30)+':'+
       Hex(Instrument)+'['+Hex(Effect)+','+
       Hex(Parameter)+']';
end;
```

Tnote.Show: A hangjegy paramétereinek kijelzése. Ez a metódus egy függvény, amely a hangjegy paraméterei alapján egy sztringet ad vissza. A sztring tartalmazza a hangjegyet és az oktávot (ha van), valamint kapcsos zárójelek között megjeleníti az effektus parancsot és hexadecimálisan a paramétert.

```
procedure Tinstrument.Init(Data:Pmem);
begin
 Name      :=GetString(Data);
 Length    :=(Data^[ 22]*256+Data^[ 23]) shl 1;
 Finetune  :=Data^[ 24] ;
 Volume    :=Data^[ 25] ;
 Rpoint    :=(Data^[ 26]*256+Data^[ 27]) shl 1;
 Rlength   :=(Data^[ 28]*256+Data^[ 29]) shl 1;
end;
```

Tinstrument.Init: Egy hangminta paramétereinek meghatározása. A metódus a *Data* alapján meghatározza a hangszer adatait (név, hossz, hangerő stb.). Az adatokat a *Show* metódussal lehet megjeleníteni.

```
procedure   Tinstrument.Show;
begin
 { Ha a hossz $0000, akkor illegális hangszer }
 if Length=0 then
  begin
   writeln(t,'Illegális hangszer !');
```

```
  end
 else
  begin { Kijelezzük a paramétereket }
   writeln(t,'A sample neve:    ',Name);
   writeln(t,'Hossza:           ',Length);
   writeln(t,'Hangerő:          ',Volume);
   writeln(t,'Ismétlési pont:   ',Rpoint);
   writeln(t,'Ismétlési hossz:  ',Rlength);
  end;
 writeln(t,'--------------------------');
end;

var
   MODname : string[100];      { A forrásfájl neve          }
   TXTname : string[100];      { A célfájl neve             }
   f,s     : file;
   ID      : string[4];        { MOD azonosító              }
   i,j     : integer;
   M       : Pmem;
   Header  : array[0..1083] of byte;      { MOD fejléc      }
   Pattern : array[0..1023] of byte;      { Egy pattern     }
   MaxPatt : byte;
   Instr   : array[0..30] of Tinstrument; { Hangszerek      }
   Note    : Tnote;                       { Egy hangjegy    }
   c       : char;

{ Egy teljes pattern kiírása a szövegfájlba }
procedure ShowPattern(pnum:integer);
var i,j:integer;
begin
 writeln(t,'================================='+
           '=================================');
 writeln(t,'A pattern száma:',pnum);
 for j:=0 to 63 do
  begin
   write(t,j:2,'. ');
  for i:=0 to 3 do
   begin
    Note.init(Ppatterndata(@pattern[j*16+i*4]));
    write(t,Note.show,'  ');
   end;
   writeln(t);
  end;
end;
```

ShowPattern: Egy *pattern* kiírása a célfájlba. Ez az eljárás végigveszi a *pattern* összes csatornájának összes hangját, és szöveges formában kiírja a hangjegyeket.

```
BEGIN
 clrscr;
 { Beolvassuk vagy előállítjuk a forrásfájl nevét }
 if paramcount=0 then
  begin
   write('A MOD fájl neve (kiterjesztés nélkül) : ');
   readln(MODname);
  end else MODname:=paramstr(1);

 TXTname:=MODname+'.txt';      { A célfájl neve     }
 MODname:=MODname+'.mod';      { A forrásfájl neve }

 assign(f,MODname);
 assign(t,TXTname);
 rewrite(t);
 { Beolvassuk a fejlécet, ez mindig 1084 bájt       }
 reset(f,1);
 blockread(f,Header,1084);
 { Meghatározzuk a fájl azonosítóját }
 ID:=Chr(Header[1080])+Chr(Header[1081])+
     Chr(Header[1082])+Chr(Header[1083]);

 if not ((ID='M.K.')or(ID='FLT4')) then
  begin
   writeln(' Ez nem egy 31 hangszeres, négycsatornás MOD fájl! ');
   close(f);
   close(t);
   HALT; { Kilépünk, ha nem jó a fájl ! }
  end;

 writeln(t,'A fájl neve:                    ',MODname);
 write   (t,'A modul neve:                  ',
         GetString(Pmem(@Header)));
 writeln(t,',ID: ',ID);
 writeln(t,'A lejátszott patternek száma: ',Header[$3b6]);

 MaxPatt:=$00;
 for i:=$3b8 to $3b7+Header[$3b6] do
   if Header[i]>MaxPatt then MaxPatt:=Header[i];

 writeln(t,'Az összes létező pattern: ',MaxPatt+1);
 writeln(t,'~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~');
 writeln(t,'Patternsorrend: ');
 j:=0;
 for i:=$3b8 to $3b7+Header[$3b6] do
  begin
   write(t,hex(Header[i])+'  ');inc(j);
   if j=16 then begin j:=0;writeln(t);end;
  end;
```

```
writeln(t);

for i:=0 to 30 do
 begin
  writeln(t,i+1,'. hangszer:');
  Instr[ i] .init(Pmem(@header[ 20+i*30] ));
  Instr[ i] .Show;
 end;

writeln(t,'Patternleírás: ');
for i:=0 to MaxPatt do
 begin
  BlockRead(f,Pattern,1024);
  ShowPattern(i);
 end;
close(t);
writeln('A MOD szöveges listája elkészült! ');
writeln('A hangszereket kimentsem? (I/N) ');

c:=upcase(readkey);
if c='I' then
 begin
  getmem(M,65520);
  for i:=0 to 30 do
   if Instr[ i] .length<>0 then
    begin
     assign(s,'sample.'+hex(i));
     rewrite(s,1);
     blockread(f,m^,instr[ i] .length);
     blockwrite(s,m^,instr[ i] .length);
     close(s);
    end;
  freemem(M,65520);
 end;
close(f);

END. of program
```

Hogyan működik a program? A *paramcount* értékéből megállapítjuk, hogy van-e paraméter a parancssorban. Ha van, akkor ez lesz a forrásfájl neve, ha pedig nincs, akkor bekérjük a nevet. Megnyitjuk a fájlt, és beolvassuk a fejléc részt. Ez minden esetben 1084 bájt hosszú. Kiolvassuk az azonosító karaktereket, és csak akkor folytatjuk a feldolgozást, ha az azonosító szöveg helyes. A patternsorrend alapján meg lehet állapítani a legnagyobb létező pattern számát és a felhasznált patternek számát is. Ez után a hangszerek paramétereit olvassuk be a *Tinstrument* objektumokba, és rögtön ki is írjuk őket a célfájlba. A hangszerek paraméterei után következnek a patternek, amelyeket a *ShowPattern* eljárás sorban kiír a szövegbe. Amikor minden patternt kiírtunk, a szöveges fájl is elkészült, így már le lehet zárni. Ekkor a képernyőn megjelenítjük a kérést, hogy

kell-e menteni a hangszerek hangmintáit is. Ha erre igen (I) a válasz, akkor egy egyszerű for ciklussal *'sample.xx'* (*xx* a $00..$1E tartományban lehet) néven külön fájlokba mentjük a hangmintákat.

Az alábbiakra nagyon kell figyelni a program használata során:

– A programban csak minimális ellenőrzés van, ezért figyeljünk arra, hogy a MOD fájl hibátlan legyen;

– A forrásfájl nevénél sohasem szabad kiterjesztést megadni;

– A szövegfájl felülírja az esetlegesen már létrehozott ugyanolyan nevű fájlt;

– A hangminták számozása hexadecimális, és csak a legális hangszereket menti;

– A program csak négycsatornás, 31 hangszeres fájlokat tud kezelni (*'FLT4'*, *'M.K.'*);

– A hangmintáknál feltételezzük, hogy egyik sem nagyobb 64K-nál.

9.6. A MID fájlok formátuma

A MIDI tulajdonképpen egy jól átgondolt és elterjedt szabvány, amely elektronikus hangszerek és a hozzájuk kapcsolható egyéb eszközök összekötését definiálja. Ha a fizikai megvalósítást nézzük, akkor azt mondhatjuk, hogy a MIDI egy egyszerű soros átvitelre alkalmas kábel, amellyel össze lehet kötni a zenei eszközöket. Ha viszont a MIDI létrejöttének alapelveit nézzük, és figyelembe vesszük a szabványt alkalmazó nagy hangszergyártó cégeket, akkor azt kell mondani, hogy a MIDI sokkal több egy egyszerű kábelnél. Eredetileg a MIDI csupán néhány ésszerű ajánlás volt a kommunikációhoz, amelyet a gyártók vagy alkalmaztak, vagy nem. Mivel azonban az egész rendszer ésszerű és könnyen implementálható (megvalósítható) volt, ezért egyre többen alkalmazták, aminek az lett az eredménye, hogy 1992-re mindenki — aki csak számít az elektronikus hangszergyártásban — elfogadta, és ma már el sem képzelhető, hogy például egy szintetizátornak ne legyenek MIDI csatlakozói. Minderről sokat ír a megfelelő magyar irodalom, ezért itt most csupán a MID fájlok formátumát ismertetjük részletesen. Mivel azonban a MID fájlok gyakorlatilag MIDI szekvenciák — azaz MIDI üzenetek sorozatai —, ezért egy ilyen fájl némileg bonyolultabb, mint az előzőekben ismertetett fájltípusok. (Nem véletlenül hagytuk ezt utoljára, hiszen ez az egyik legelterjedtebb és legfontosabb szabvány!)

A MID fájloknak a csatlakozóhoz és az átviteli protokollhoz persze kevés köze van; ez mindössze egy ajánlás a programozóknak (amely ajánlás azért megpróbálja a fájl szerkezetét minél jobban a MIDI üzenetekhez hasonlóvá tenni), hogy a számítógépes zeneszerkesztők egységes formátummal tudjanak dolgozni, azaz fájl szinten kompatíbilisek

legyenek. Egészen elnagyolva azt mondhatjuk, hogy a zene tárolásához nem kell más, mint ismerni az egyes zenei eseményeket és azok bekövetkezésének időpontját. (Emlékezzünk vissza az ADLIB-nél ismertetett lejátszó programra, ott is ilyesmit tettünk, de egyéni adatformátummal dolgoztunk.) Mindez tárolva valahogy így festene:

dTIME	MIDIevent	dTIME	MIDIevent	dTIME	MIDIevent

MIDIevent: Két egymást követő MIDI esemény.
dTime: A két MIDI esemény között eltelt idő.

A MIDI események értelmezéséhez fontos tudni a következőket:

- Az üzeneteknél a kezdőbájtban a 7. bit mindig 1.
- A paramétereknél a 7. bit általában 0.
- Egy MIDI rendszerben 16 logikai csatorna van.
- A rendszer elemei a hangszerek (*instruments*) és a vezérlők (*controllers*).

9.6.1. Az idő tárolása: dinamikus adathossz

A MIDI események közötti idők széles sávban mozoghatnak. Lehetnek egészen rövid, időben egymás után következő események, mint például egy akkord hangjainak egyszeri (majdnem egyszeri) lefogása; és lehetnek időben nagyon távoli események is (szünetek). Az idők tárolása ezért úgy lenne gazdaságos, ha a kis időket kevés biten, a nagyobb időközöket pedig több biten lehetne tárolni. Erre a célra dolgozták ki a **dinamikus adathosszúságot**, illetve az ilyen formában tárolt **dinamikus adatokat**. Az ilyen adat legkevesebb egy, és legfeljebb négy bájtból állhat. Minden bájt 7. bitje azt mutatja, hogy van-e következő adatbájt. Így az adattárolásra csupán 4*7=28 bit marad. Nézzünk erre néhány példát!

Az első esetben tároljunk egy 100-as időegységet ilyen módon! (Hogy az időegység mit takar, arra később még visszatérünk.) Az adat tehát 100, és nincs következő bájt, ezért a bájt értéke:

00100000 $40

Legyen a feladat 255 időegység tárolása! Ez hexadecimálisan a következő: $00FF. Tehát 8 bitre van szükség, de egyetlen bájt csak 7 bitet képes tárolni, ezért kell két bájtot használni:

10000001 01111111 $81 $7F

A leghosszabb tárolható időegység formája:

1xxxxxxx 1xxxxxxx 1xxxxxxx 0xxxxxxx

9.6.2. MIDI események

A MID fájlokban MIDI eseményeket tárolunk, de tudni kell, hogy ezek az események valójában üzenetek két hangszer között. Az adó hangszer kiad valamilyen üzenetet, amelyet a vevőnek fel kell dolgoznia, és végre kell hajtania a megfelelő parancsot. Amikor egy MIDI lejátszó program lejátszik egy MID fájlt, akkor is valami ilyesmi történik, de ilyenkor természetesen nincs két összekötött hangszer. Ezért az események gyakran olyanok, amelyek egy számítógépen belül nem igazán fordulhatnak elő — mint például egy hang leütése vagy egy pedál lenyomása —, de annál inkább előfordulhatnak a valóságos hangszerek esetében. Mindez azért fontos, mert a fájl komponenseit nem úgy kell értelmezni, mint közönséges adatokat, hanem mint egy MIDI csatorna üzeneteit. Leginkább úgy lehet felfogni, hogy a fájl az adó hangszer, míg a lejátszó program a vevő és végrehajtó hangszer szerepét tölti be.

A MIDI események mindig egy státusbájttal kezdődnek, amelyet még egyéb adatbájtok is követhetnek. Pontosabban: minden csatornán van állapottartás, és ha egymás után azonos üzenetek jönnek, akkor a parancskód csak egyszer megy ki, és utána már csak a paraméterek következnek. Ezért az események is változó adathosszúságúak lehetnek. Fontos, hogy az adatbájtok mindig kisebbek 128-nál, azaz a 7. bit 0, mert ez jelzi, hogy nem dinamikus adatról van szó.

9.6.3. Hang csatornaüzenetek

MIDI esemény $8x aa bb: A hang kikapcsolása (*Note off*)

Ez a parancs kikapcsolja a hangot. A csatornaszámot az *x* adja, ez 0...15 lehet. Az *aa* paraméter a hang számát adja, míg a *bb* az elhalkulás (elengedés) idejét definiálja. Az *aa* 128 különböző hangot tud leírni, így több mint 10 oktáv kezelhető. A *bb* nem más, mint a billentyű felengedésének a sebessége a hangszernél. Ha egy adott hangszer nem érzékeli a sebességet, akkor általában $40-et küld.

MIDI esemény $9x aa bb: A hang bekapcsolása (*Note on*)

A parancs hatására az *x*. MIDI csatornán megszólal egy hang. A hang magasságát, sorszámát az *aa* érték adja ugyanúgy, mint a hang kikapcsolásánál. A *bb* érték a billentyű leütésének sebességét definiálja. Ez az érték 1...127 között lehet, mert a $00 speciálisan egy *Note off* parancsnak felel meg. Ez feleslegesnek tűnik, hiszen a hang kikapcsolására ott van a $8x parancs, de mégis értelmeznünk kell a régi hangszerek miatt.

 **MIDI esemény $Ax aa bb:** Billentyű nyomva tartásának erőssége (*Polyphonic pressure*)

A mai korszerű billentyűzetek képesek érzékelni a lenyomás és a nyomva tartás erősségét. Ilyen formán például meg lehet állapítani, hogy egy akkord hangjait milyen súllyal kell megszólaltatni. Az előzőekhez hasonlóan az *x* a csatorna száma, az *aa* a hang kódja. A *bb* a lenyomás erősségét jelzi a 0...127 tartományban.

 **MIDI esemény $Bx aa bb:** Vezérlők beállítása (*Control change*)

A vezérlők olyan eszközök, amelyek nem hangszerek ugyan, mégis hatással vannak a zenére (pedálok, modulátorok). Ezeknek az eszközöknek a beállítására szolgál ez a parancs. Az *x* a csatorna számát adja, az *aa* a vezérlő eszköz azonosító kódja (0...127), a *bb* pedig az eszköz új beállítása.

 **MIDI esemény $Cx aa :** Program beállítása (*Program change*)

Ez az üzenet arra szolgál, hogy a vevő hangszer hangprogramjából egy újat válasszunk ki. A hangprogram lehet előre meghatározott vagy a felhasználó által definiált program is. Az *x* a csatorna számát adja, az *aa* pedig az új programot azonosítja (0...127).

 **MIDI esemény $Dx aa :** Általános lenyomáserősség (*Channel pressure*)

A *$Ax* parancshoz hasonlóan ez a parancs is nyomáserősséget definiál, de itt egy egész MIDI csatornára általánosan lehet megadni a nyomást. Az *aa* paraméter a csatorna általános nyomását határozza meg a 0...127 tartományban.

 **MIDI esemény $Ex aa bb :** Hajlítás (*Pitch bend*)

Az alaphang meghajlítására ez a parancs szolgál. Az *x* a csatorna számát adja, az *aa* és a *bb* pedig egy 14 bites értéket (a 7. bitek töröltek), amely így egy 0...16383 közötti tartományt tud leírni. Ha ez az érték a 0...8191 ($0000...$1FFF) tartományban van, akkor lefelé, ha pedig a 8192...16383 ($2000...$3FFFF) tartományban van, akkor felfelé kell hajlítani. Az alapbeállítás a 8192 ($2000), ez a középállás.

9.6.4. Rendszerüzenetek

Ezeket a parancsokat a MIDI rendszer valamennyi eszköze használhatja. Mivel az ilyen üzenetek nincsenek MIDI csatornákhoz kötve, ezért ilyenkor a csatornaszám nincs az üzenetben. A rendszerüzeneteknek két nagy csoportja van:

– Egyszerű (*normal*) rendszerüzenetek. Ezek nincsenek szorosan időhöz kötve, és több kiegészítő adatbájtot is tartalmazhatnak.

– Valós idejű (*real time*) rendszerüzenetek. Ezek bizonyos szinkronizálási feladatokat látnak el, és mindig egy bájt hosszúak. A járatlan programozók külön örömére a valós idejű rendszerüzenetek bárhol — egy megkezdett más MIDI parancson belül is — elhelyezhetők. Ez azt jelenti, hogy például egy $Ax esemény kellős közepén is szerepelhet egy ilyen bájt. A 7. bit segítségével megállapítható ugyan, hogy ez egy rendszerüzenet, de még így is sok bonyodalmat okoz.

MIDI esemény $F0 aa bb...cc $F7: Exkluzív rendszerüzenet (*System exclusive*)

Ezt a parancsot a MIDI rendszer az eszközspecifikus üzenetek továbbítására használja. Ennek akkor van értelme, ha a rendszerben csupán két kitüntetett eszköz — például két azonos típusú szintetizátor vagy egy szintetizátor és egy számítógép — akar kommunikálni, de a többi eszköz számára ez a kommunikáció lényegtelen. Az $F0 bájt tudatja minden eszközzel, hogy egy exkluzív üzenet fog érkezni. Az *aa* paraméter azonosítja az eszközt. Ez csak az 1...124 tartományban mozoghat. Lehetőség van persze több eszköz azonosítására is, de ez most számunkra nem lényeges. A további paraméterek (*bb..cc*) már az exkluzív üzenetet tartalmazzák. Az üzenet végét a $F7 bájt jelzi. Az üzenet első két bájtja speciális gyártóazonosító, ebből tudják az eszközök, hogy kinek szól az üzenet. Ezt minden eszköznek értelmeznie kell. Az ez után következő részek már gyártóspecifikusak.
Típus: Valós idejű rendszerüzenet.

MIDI esemény $F1 nn : Időkód negyedhang mező (*MTC Quarter-frame*)

Időkódolt átviteli protokollhoz tartozó üzenet az időkód tényleges pozíciójának jelzésére. A MID fájlok szempontjából ez most lényegtelen.
Típus: Normál rendszerüzenet.

MIDI esemény $F2 aa bb : Zenepozíció mutató (*Song position pointer*)

Ez az üzenet a zene lejátszásának kezdetén a kezdőhang pozíciójának megadására szolgál. Gyakran van szükség erre, hogy a MIDI rendszerben esetleg még részt vevő eszközök is tudják a zene pontos kezdetét. A két paraméter (*aa,bb*) egy 14 bites pozíciót ad meg, és a pozíció minden 1/16 időütem elteltével növekszik.
Típus: Normál rendszerüzenet.

MIDI esemény $F3 aa : Zenekiválasztás (*Song select*)

Ezzel a paranccsal az *aa* számú zenét lehet kiválasztani lejátszásra. A lejátszás ekkor még nem indul el.
Típus: Normál rendszerüzenet.

MIDI esemény $F6 : Önhangolás (*Tune request*)

Ennek az üzenetnek a hatására egy MIDI hangszernek el kell kezdenie az önhangolási metódust. Ma már ez nem túl lényeges, mivel a mostani hangszereket elég pontosan hangolják.
Típus: Normál rendszerüzenet.

MIDI esemény $F7 : Exkluzív rendszerüzenet vége (*System exclusive end*)

A $F0 kóddal megkezdett exkluzív üzenetek végét ez a parancs jelzi.
Típus: Valós idejű rendszerüzenet.

MIDI esemény $F8 : Időzítés (*Timing clock*)

Időzítési sorozat. Ezt a parancsot a vezérlést ellátó MIDI eszköz szokta küldeni 24-szer egyetlen negyedhang alatt, hogy a többi eszköz értesüljön a lejátszás menetéről. Az üzenet elsősorban szinkronizációs feladatokat lát el.
Típus: Valós idejű rendszerüzenet.

MIDI esemény $FA : A lejátszás megkezdése (*Start*)

A vezérlőeszköz ezzel az üzenettel jelzi, hogy a lejátszás elkezdődött.
Típus: Valós idejű rendszerüzenet.

MIDI esemény $FB : A leállított lejátszás folytatása (*Continue*)

Ez a parancs egy megszakított lejátszás folytatását teszi lehetővé.
Típus: Valós idejű rendszerüzenet.

MIDI esemény $FC : A lejátszás leállítása (*Stop*)

A lejátszás felfüggesztése. A parancs leállítja a lejátszást, de a vevő eszköznek meg kell őriznie a zenepozíció mutatót, mert egy $FB parancs hatására esetleg folytatnia kell a lejátszást.
Típus: Valós idejű rendszerüzenet.

 **MIDI esemény $FE :** Az összeköttetés tesztelése (*Active sensing*)

Ennek az üzenetnek a célja az összeköttetés ellenőrzése. Ha a kapcsolatban nincs semmilyen átvitelre szánt adat, akkor az adónak ezt a parancsot kell küldenie 300 ms-os időközönként. Ha a vevő ezt nem érzékeli, akkor úgy tekinti, hogy a kapcsolat megszakadt. Az üzenet értelmezése teljesen opcionális, bár egyes billentyűzetek (pl. némely Roland) szorgalmasan küldik. Ha valamilyen eszköz értelmezi, és egy idő után nem észleli a vonalon, akkor általában az ilyen eszköz inicializálja magát, hiszen számára az üzenet elmaradása azt jelenti, hogy nincs kapcsolat.

 **MIDI esemény $FF:** Rendszerinicializálás (*System reset*)

A rendszer alapállapotba hozása.

A MID fájlokban — mivel ezek egyszerű, zenét tartalmazó fájlok — van néhány változtatás az eredeti üzenetekhez képest, ennek ellenére mégis célszerűnek tartottuk a valódi kommunikáció alapelemeit is ismertetni, hiszen ez a kiindulási alap. A MIDI üzenetek után most már valóban következzék a fájl leírása!

9.6.5. MID fejléc

Minden MID fájl egy fejléccel kezdődik, amely tartalmazza a fájlban elhelyezett zene típusát és az időzítéshez szükséges legkisebb időegységet.
A fejléc formája az alábbi:

Eltolás	Mező	Hossz	Jelentés
$00	*mID*	$04	a fejléc azonosító szövege („*MThd*")
$04	*mHeaderLEN*	$04	a fejléc hossza (mindig $0006)
$08	*mTrackType*	$02	a fájl típusa
$0A	*mTrackNum*	$02	a fájlban elhelyezett sávok száma
$0C	*mDtime*	$02	a legkisebb időegység

 **MID fájl azonosító :** Bájt $00..$03 (*MIDI ID*)

A fájl első négy bájtja az „MThd" szöveget tartalmazza, ez jelzi, hogy MID fájlról (pontosabban MID fájl fejlécről) van szó.

 **MID fejléc hossz :** Bájt $04..$07 (*MID header length*)

A fejléc hossza; mindig $0006, mert a fejléc még 6 bájt adatot tartalmaz. A hosszúságot a MOD fájlokhoz hasonlóan itt is a Motorola processzorcsalád adatábrázolási formája szerint kell megadni, tehát ez az érték a fájlban a következő:

$00 $00 $00 $06

A MID fájl típusa : Bájt $08,$09 (*MID track type*)

A fájlban alapvetően háromféle típusú adat helyezhető el:

$0000 – egyszerű egysávos zene (*single track*);
$0001 – többsávos szinkronizált zene (*synchronous multiple tracks*);
$0002 – többsávos nem szinkronizált zene (*asynchronous multiple tracks*) .

(A $0002 hasonló formában tárolja a zenét, mint a MOD fájl patternjei, de általában ritkán használatos.)

A fájlban elhelyezett sávok száma : Bájt $0A,$0B (*Number of tracks*)

A fájlban több különálló zenesáv található, ezeknek a számát adja meg ez a mező. Az egyszerű egysávos ($0000) fájltípusnál természetesen itt mindig $0001 található.

A fájl legkisebb időegysége : Bájt $0C,$0D (*Delta time ticks per quarter note*)

Ez a mező megadja a fájlban az időzítések legkisebb egységét. Minden további időzítő adat ilyen egységekben értendő. Az értelmezés kétféle lehet:

– A legfelső bit 0. Az alsó 15 bit ilyenkor azt mutatja, hogy egy negyedhangot hány elemi időegységre bontottak. Két egymást követő MIDI üzenet idejét tehát ennek egész többszörösében lehet megadni.

– A legfelső bit 1. Ezt az időkód alapú rendszernél használják (kép és hang összehangolására). Ilyenkor nem negyedhang alapján, hanem másodpercből kell kiszámítani az időegységet. A megmaradt 15 bit felső hét bitje az időkódolást mutatja kettes komplemens alakban. Ez négy különböző érték lehet, amihez másik négy állandó időérték tartozik:

A 14...8 bitek értéke	A kód jelentése
–24	1/24 másodperc
–25	1/25 másodperc
–29	1/30 másodperc
–30	1/29.96 másodperc

Az alsó nyolc bit pedig azt tartalmazza, hogy a fenti módon megadott időt hány részre kell osztani. Az így kapott időt kell ezután alapegységnek tekinteni. (Egyszerű! :–))

A MID fejléc után a sávok (*tracks*) következnek, illetve a $0000 típusnál csak egy sáv. A sáv szintén tartalmaz egy azonosító nevet és egy hosszúságot (az eltolásokat a sáv kezdetéhez mérve adtuk meg!) :

Eltolás	Mező	Hossz	Jelentés
$00	*trID*	$04	a sáv azonosító szövege („*MTrk*")
$04	*trLEN*	$04	az adatbájtok száma
$08	*trDATA*	*trLEN*	az adatok (MIDI szekvencia)

MID sáv azonosító : Bájt $00...$03 (*MIDI Track ID*)

Az első négy bájtja az „MTrk" szöveget tartalmazza, ez jelzi, hogy MIDI sáv következik.

Sávhossz : Bájt $04...$07 (*Track length*)

A sáv hossza. Ez az érték megmondja, hogy hány bájtot kell MIDI üzenetként értelmezni.

MIDI szekvencia : Bájt $08...?? (*MIDI datas*)

A hossz után közvetlenül a MIDI sorozat következik, amely a zenét tartalmazza. A szekvencia az előbbiekben ismertetett MIDI üzeneteket tartalmazhatja, de néhány apró eltérés azért van. A következőkben ezekről lesz szó.

9.6.6. Exkluzív üzenetek

A MID fájlban az exkluzív rendszerüzenetek tárolása eltér a vonali üzenet valódi formájától. Az ilyen üzenetek ugyanis változó hosszúságúak lehetnek. Ezért ezek mindig a $F0 státusbájttal kezdődnek, majd egy dinamikus adattal az üzenet hosszát adják meg. A dinamikus adat után az üzenet következik — ez egy egyszerű bájtsorozat —, majd a lezáró $F7 bájt. A hosszúságba a $F7 bájtot is bele kell számolni.

A $F7 bájt felszabadul, mert a hosszban már eleve benne van, hogy figyelembe kell venni, ezért másra is fel lehet használni. A MID fájlokban a $F7 az ún. 'escape' szekvenciák kezdőbájtja. A $F7 után egy dinamikus adatnak kell következnie, amely megadja a szekvencia hosszát. A hossz után pedig egy bájtsorozat következik, amely bármit tartalmazhat.

9.6.7. Metaesemények

Külön feladatot kapott a $FF bájt is, mert ez a fájlban az ún. metaeseményeket (*MIDI meta events*) azonosítja. Az ilyen üzenetek nem MIDI jellegűek, hanem egyéni feladatokat is elláthatnak. Minden metaesemény a $FF bájttal kezdődik, majd a metaesemény kódja következik, utána pedig az esemény esetleges paraméterei. A jelenleg elfogadott metaesemények:

Eseménykód	Az esemény paraméterei	Esemény
$FF $00	azonosító kód	a zene azonosító kódja
$FF $01	hossz, szöveg	tetszőleges szöveges üzenet
$FF $02	hossz, szöveg	szerzői szöveg (*Copyright*)
$FF $03	hossz, szöveg	a sáv vagy a dal neve
$FF $04	hossz, szöveg	hangszernév
$FF $05	hossz, szöveg	szöveg
$FF $06	hossz, szöveg	jelző definiálása
$FF $07	hossz, szöveg	követőpont definiálása
$FF $20	csatornaszám	MIDI csatorna definiálása
$FF $2F	állandó $00	sáv végének jelzése
$FF $51	tempó	tempóbeállítás
$FF $54	időkód	az indítás idejének definiálása
$FF $58	előjegyzés	ütemelőjegyzés
$FF $59	előjegyzés	emelés/süllyesztés előjegyzés
$FF $7F	hossz, adatok	egyéni üzenet

MIDI metaesemény $00 : A zene azonosítója (*Song ID*)

Egy zene (sáv) azonosítása. A kód után a parancs hossza áll két bájtban tárolva (ez mindig $02), majd pedig a zenét azonosító kód, amely szintén két bájt.
Forma: $FF $00 $02 aa bb

MIDI metaesemény $01 : Tetszőleges szöveges üzenet (*General message*)

Ezzel a paranccsal bármilyen szöveget el lehet helyezni a fájlban. A kód után a szöveg hosszát tároljuk (ez két bájt), majd a szöveg karaktereit.
Forma: $FF $01 aa bb t1 t2 t3 ...

MIDI metaesemény $02 : Szerzői szöveg (*Copyright information*)

A $01-es eseményhez hasonló módon ez is szöveget tárol, amelyben a szerzőről és a fájl tulajdonosáról kapunk információkat. A tárolási módszer ugyanaz, mint a $01 esetében.
Forma: $FF $02 aa bb t1 t2 t3 ...

MIDI metaesemény $03 : A sáv vagy a dal neve (*Track name*)

A fájl típusától függően a zene vagy csak az adott sáv nevét lehet tárolni ezzel a parancscsal. Ha a zene egysávos ($0000 típus), akkor ez a zene neve, egyébként csak a sávé. A tárolási módszer ugyanaz, mint a $01 esetében.
Forma: $FF $03 aa bb t1 t2 t3 ...

MIDI metaesemény $04 : A hangszer neve (*Instrument name*)

Ez az esemény megadja a hangszer nevét. A tárolási módszer ugyanaz, mint a $01 esetében.
Forma: $FF $04 aa bb t1 t2 t3 ...

MIDI metaesemény $05 : Szöveg (*Song lyric*)

A zenében bizonyos helyeken azt a szöveget adja meg, amelyet énekelni kell. A tárolási módszer ugyanaz, mint a $01 esetében.
Forma: $FF $05 aa bb t1 t2 t3 ...

MIDI metaesemény $06 : Jelzőpont definiálása (*Marker point*)

Ezzel az eseménnyel egy jól definiált időpontot lehet megadni, amelyre esetleg később még szükség lehet. Az időpontot szövegesen kell megadni, a tárolási módszer megegyezik a $01 esetben alkalmazottal.
Forma: $FF $06 aa bb t1 t2 t3 ...

MIDI metaesemény $07 : Követőpont definiálása (*Cue point*)

A jelzőponthoz hasonló a funkciója, ezzel valamilyen másik esemény (amely nem zenei, hanem például képi) szinkronizálható a zenéhez. Az időpontot szöveggel kell megadni ugyanúgy, mint a $01 esetben.
Forma: $FF $07 aa bb t1 t2 t3 ...

MIDI metaesemény $20 : MIDI csatorna definiálása (*MIDI channel*)

Ezzel a paranccsal kell megadni, hogy az adott sáv melyik MIDI csatornát használja. A kód után közvetlenül egy $01 érték áll, majd a csatorna száma (0..15).
Forma: $FF $20 $01 aa

MIDI metaesemény $2F : Sáv vége (*End of track*)

Ezt az eseményt minden egyes sáv végén kötelező kitenni. A kód után egy $0000 értéknek kell állnia.
Forma: $FF $2F $00

MIDI metaesemény $51 : Tempóbeállítás (*Set tempo*)

A zene tempójának beállítása. A kód után közvetlenül egy $03 érték áll, majd három bájt, amelyek a tempó idejét adják meg µs/negyed mértékegységben.
Forma: $FF $51 $03 aa bb cc

MIDI metaesemény $54 : Az indítás idejének definiálása (*SMPTE offset*)

Az időkódot használó rendszereknél a zene indításának idejét definiálja.
Forma: $FF $54 $05 hh mm ss aa bb

MIDI metaesemény $58 : Ütemelőjegyzés (*Time singature*)

Az ütemelőjegyzést tudjuk megadni ezzel az üzenettel. Sajnos ez egy kissé bonyolult. :–(Az első paraméter (*aa*) az előjegyzés számlálója, a második (*bb*) a nevező kettes alapú logaritmusa, a harmadik (*cc*) pedig a metronóm osztása MIDI órában. Az utolsó (*dd*) paraméter azt adja meg, hogy egy zenei negyedet hány harmincketted részre lehet osztani; ez általában 8.
Forma: $FF $58 $04 aa bb cc dd

MIDI metaesemény $59 : Emelés/süllyesztés előjegyzése (*Key singature*)

Ezzel a paranccsal a hangok előjegyzését lehet megadni. Az első paraméter az emelés/süllyesztés értékét definiálja, a második pedig az akkordot.
Forma: $FF $59 $02 aa bb

MIDI metaesemény $7F : Egyéb üzenet (*Sequencer specific information*)

Ezzel a paranccsal az egyes lejátszók egyéni üzeneteket tudnak elhelyezni a fájlban. Az első paraméter az üzenet hossza bájtokban, majd az üzenet következik.
Forma: $FF $7F aa bb t1 t2 t3 ...

9.6.8. A GENERAL MIDI szabvány

A 80-as évek végére kidolgozták a GENERAL MIDI (**GM**) szabványt, amely az alap MIDI-től eltérően nem a kommunikációhoz, hanem a megvalósított hangszerekhez tartalmaz ajánlásokat. A GM nem egy lezárt fejlesztés, nyitott maradt, hogy a további fejlődés során létrejövő újdonságokat könnyedén be lehessen vezetni. Mindez részletesebben és pontosabban megtalálható az irodalomjegyzékben említett MIDI könyvben, itt most csupán a legfontosabb részekről írunk.

A GM tulajdonképpen a kezdők számára nyújt nagy segítséget a házi MIDI rendszer kialakításához. Nem kell hozzá sok minden, csupán a következők:

- egy MIDI mesterbillentyűzet;
- egy számítógép, amelyben van egy GM kompatíbilis szerkesztőprogram;
- egy hangkártya, amely támogatja a MIDI-t.

A hangkártya legalább 24 független csatornát tudjon kezelni, és a 16 MIDI csatornát egymástól teljesen külön kell választani. Jó, ha legalább 128 hangprogramja van.

A GM egységesíti a hangprogramokat. A hangprogramokhoz külön-külön lehet megadni a hangszereket. A 10-es MIDI csatorna megkötése, hogy ezen csak ütőhangszereket és dobokat lehet megszólaltatni, ráadásul minden billentyűhöz külön hangszer tartozik.

Ezeken kívül még további fontos megkötések is vannak, de ezeket most csupán címszavakban említjük, hiszen nem e könyv hivatott az ismertetésükre:

- Meg kell valósítani a hajlítást (minden csatornán).
- Minden csatornán reagálni kell a billentyűzet leütésének erősségére (*Note On,* $9x).
- Meg kell valósítani a GM vezérlőket (lásd a megfelelő függelékben).
- A közép C hang kódja mindig a $3C legyen.
- A csatornák között dinamikus allokációt kell alkalmazni (minden MIDI csatorna annyi hangkártya fizikai csatornát kapjon, amennyire éppen szüksége van!).

Két fontos rendszerüzenet van, amely a GM-et aktiválja, illetve kikapcsolja:

A GENERAL MIDI bekapcsolása: $F0 $7E $7F $09 $01 $F7

A GENERAL MIDI kikapcsolása: $F0 $7E $7F $09 $02 $F7

9.6.9. A ROLAND GS MIDI szabványa

A GM mellett a másik jelentős előrelépés volt a MIDI fejlesztésében a Roland cég GS szabványa. Ez kompatíbilis a GM-mel, ugyanakkor több újítást is tartalmaz. (Ezeket sem részletezzük, csupán felsoroljuk őket.)

- Legfeljebb 16384 hangprogram kezelése.
- Új vezérlők bevezetése.
- Sokkal több ütőhangszer és dob.
- Prioritás a dinamikus allokációhoz.

10. MOD lejátszó Gravis kártyára

A MOD fájlok leírásánál láthattuk, hogy ezek lejátszása a Gravis Ultrasound kártyán nem túlságosan nehéz feladat. Sajnos a könyvben a hosszabb forrásliták közlésének nincs túl sok értelme — hiszen ezeket begépelni igen hálátlan feladat —, mégis úgy gondoltuk, hogy egy ilyen lejátszó programnak mindenképpen szerepelnie kell. Nem mintha nem készült volna már éppen elég MOD lejátszó a Gravis kártyára (a gyári lemezmellékleteken több is van, és ezek biztosan jobbak annál, mint amelyet most közlünk), de az elv bemutatása és a megvalósítás mindenképpen fontos. A most következő program közel sem tökéletes! (Megpróbáltuk minél rövidebbre írni, de ennek az lett az eredménye, hogy sok mindent ki kellett hagyni.) :-{(

```
program modplayer;

uses crt,dos,GUS;

type
    Pmem = ^Tmem;                          { Memóriamutató          }
    Tmem = array[ 0..65520] of byte;       { Memória                }

    Pheader = ^Theader;                    { MOD fejléc mutató      }
    Theader = array[ 0..1083] of byte;     { MOD fejléc             }

    Tpattern = array[ 0..1023] of byte;    { MOD pattern adat       }

    Pmodule  = ^Tmodule;                   { Teljesmodul-mutató     }
    Tmodule  = array[ 0..62] of Tpattern;  { Teljes modul           }

    Pnote = ^Tnote;                        { Note mutató            }
    Tnote = array[ 0..3] of byte;          { MOD Note adat          }

    Tinst = record                         { Hangszerrekord         }
     S,E   : longint;                      { GUS címek              }
     LS,LE : longint;                      { GUS loop címek         }
     Vol   : byte;                         { Alaphangerő            }
     Flag  : byte;                         { Állapotjelző           }
     { 00 - illegális hangszerszám }
     { 01 - normál hangszer        }
     { 02 - loopolt hangszer       }
    end;
```

Ezek a legfontosabb típusok, amelyek a feldolgozás során előfordulhatnak. A nevek magukért beszélnek, egyedül talán a *Tinst* rekord igényel némi magyarázatot. Ennek az elemeit a fejléc hangszerleírásaiból állítjuk elő, de itt már a konkrét GUS DRAM címeket tároljuk. Az *S* mező a hangminta DRAM fizikai kezdőcímét tárolja, az *E* a végcímet, az *LS*, *LE* pedig a *loop* típusú hangszereknél a hurok kezdő- és végcímét. A *Flag* mező jelzi, hogy milyen hangszerről van szó.

```
const
    { Periódustábla a hangjegyek azonosításához }
    PeriodTABLE : array[ 0..35] of word=(
    856,808,762,720,678,640,604,570,538,508,480,543,
    428,404,381,360,339,320,302,285,269,254,240,226,
    214,202,190,180,170,160,151,143,135,127,120,113);

    { Sebességtáblázat }
    Speed : array[ 0..15] of byte=
       (80,100,110,115,120,130,135,138,
       140,143,144,145,150,155,160,165);
```

PeriodTABLE: A *note*-ok azonosításához használt táblázat. A hangokat a MOD fájl az AMIGA időzítése szerint tárolja, ami nekünk természetesen nem felel meg. Ezért minden egyes *pattern* minden egyes *note*-ját át kell alakítani egy számunkra is feldolgozható formára. Az átalakítást a *ConvertPatterns* eljárás végzi. A hangjegyeket a periódustábla alapján tudjuk meghatározni. A hangok valódi frekvenciáját a *GUS* unit *FreqTAB* táblázata tartalmazza.

Speed: A lejátszás lehetséges sebességei. Ezt a táblázatot úgyszólván teljesen véletlen értékekkel (hallás alapján) töltöttük fel. A MOD fájl sebességét a lejátszó főciklusba iktatott várakozás szabja meg. A *Speed* ezen várakozásokat tartalmazza, de még egyszer hangsúlyozzuk, hogy ezek nem a szabványos értékek, csak így hangzott a legjobban. :–)

```
var
  j,k      : integer;          { Általánosan használt változók }
  Gpt      : longint;          { GUS DRAM foglaltságmutató     }
  f        : file;             { A forrásfájl azonosítója      }
  ID       : string[ 4] ;      { A MOD fájl azonosítója        }
  H        : Pheader;          { A MOD fájl fejléce            }
  M        : Pmodule;          { A teljes modul mutatója       }
  I        : array[ 0..30] of Tinst;  { Hangszerek             }
  Seq      : array[ 0..127] of byte;  { Patternsorozat         }
  Snu,Max  : byte;
  Spd,b,v  : byte;
  Chan     : array[ 0..3] of byte;    { Csatornahangszerek     }
  FIFO     : array[ 0..31] of byte;   { IRQ FIFO tárolás       }
  Mp,Ml    : word;                    { Mp: Az aktuális pattern }
                                      { Ml: Az aktuális sor    }
  Name     : string[ 80] ;            { A MOD fájl neve        }
```

A változók a lejátszás alapvető paramétereit tartalmazzák. Részletesen majd az aktuális helyen írunk róluk.

```
procedure GravisIRQ;                                    interrupt;
var j,k,Is : byte;
begin
```

```
 k:=0;j:=0;
 while j and $c0<>$c0 do        { Kiolvassuk a FIFO regisztert }
  begin
   j:=getgf1byte($00,$8f);
   k:=j and 31;
   if j and $c0<>$c0 then
    begin                       { Ciklikus hangszerek }
     Is:=Chan[ k] ;
     PlayVoice(k,$08,I[ Is] .LS,I[ Is] .LS,I[ Is] .LE);
    end;
  end;
 asm
  mov  al,$20                   { A megszakítás nyugtázása }
  out  $20,al
  out  $A0,al
 end;
end;
```

GravisIRQ: A hangkártya megszakítása. A program egyetlen esetben generál megszakítást: ha egy *loop*-olt hangszer első lejátszása véget ért. Az ismétlő hangszereket a következőképpen oldottuk meg: az első lejátszásnál a hangminta végig lejátszásra kerül az *S* címtől az *E* címig, de a lejátszás végén a kártya egy megszakítást ad. Ilyenkor a megszakítás beállítja az *LS* és *LE* címeket, valamint a ciklikus lejátszást, de most már nincs szükség további megszakításokra, ezért ezt letiltjuk. A megszakítás a *Chan* tömb alapján azonosítja, hogy a fizikai csatornához éppen melyik hangszer tartozik, és a hangszertömb megfelelő elemei alapján (*I[Is].LS* és *I[Is].LE*) elindítja a ciklikus lejátszást. Természetesen minden csatornát meg kell vizsgálni, hiszen egyszerre több csatornán is szólhat *loop*-olt hangszer.

```
procedure KeyON(Ch,Nt,Is,Cm:byte);
var w:word;
begin
 with I[ Is] do
  begin
   Chan[ Ch] :=Is;          { Tároljuk az aktuális hangszert       }
   StopVoice(Ch);           { Az előző hangot leállítjuk           }
   if Cm<>$0c then Volume(Ch,Vol div v);  { Hangerő-beállítás      }
   Note(Ch,Nt+12);          { Beállítjuk az új frekvenciát         }
   case Flag of             { Normál vagy ciklikus hangszer?       }
    $01: PlayVoice(Ch,$00,S,S,E);   { Normál   }
    $02: PlayVoice(Ch,$28,S,LS,E);  { Ciklikus }
   end;
  end;
end;
```

KeyOn: Egy hang megszólaltatása. Az eljárás a *Ch* (0..3) csatornán kiad egy *Nt* (0...35) hangot az *Is* (0..30) hangszerrel. A *Cm* a *note* effektusparancsát adja. A *Chan* tömbben tároljuk a hangszer kódját, hogy később a megszakítás ez alapján azonosítani tudja a

loop-olt hangszert. A *Cm*-től függően beállítjuk a hangszer alaphangerejét, amelyet a *v* hangerőosztó szabályoz. A lejátszás globális hangerejét a *v* szabja meg, ezért minden hangerő-beállításnál figyelembe kell venni. Ez után a hangjegyhez tartozó frekvenciát is beállítjuk, majd a hang kiadása következik a *PlayVoice* eljárással. Attól függően, hogy a hangszer normál vagy *loop*-olt, a lejátszás módját $00-ra vagy $28-ra kell állítani. Minden csatorna minden hangját a *KeyOn* eljárás szólaltatja meg.

```
procedure Instruments; { A hangszerek beolvasása és tárolása }
var w,l:word;
      x:pmem;
begin
 new(x);                          { Átmeneti adattömb a heapben  }
 for w:=0 to 30 do                { Legfeljebb 31 hangszer lehet }
  with I[ w] do
  begin
   S:=Gpt;                        { Hangminta kezdőcím }
   l:=(H^[ 42+w*30]*256+H^[ 43+w*30]) shl 1; { Hossz }
   if l=0 then Flag:=$00          { Illegális, ha a hossz $0000, }
    else                          { egyébként be kell olvasni    }
     begin
      blockread(f,x^,l);          { Beolvassuk a heapbe          }
      MoveSample(Gpt,x,l);        { Átmásoljuk a GUS DRAM-ba     }
      inc(Gpt,l);                 { Növeljük a DRAM mutatót      }
      E:=Gpt-1;                   { Hangminta végcím             }
      Vol:=H^[ 45+w*30];          { Hangminta alaphangerő        }
      {Loop-olt hangszernél az ismétlés kezdőcíme az LS          }
      LS:=S+((H^[ 46+w*30]*256+H^[ 47+w*30]) shl 1);
      { Az ismétlés hossza }
      l:=(H^[ 48+w*30]*256+H^[ 49+w*30]) shl 1;
      LE:=LS+l;
      { Ha az l=2, akkor a hangszer nem loopolt                  }
      if l>2 then Flag:=$02
             else Flag:=$01;
     end;
   case Flag of
    $01 : writeln(w:2,': Normál hangszer');
    $02 : writeln(w:2,': Loop-olt hangszer');
   end;
  end;
 dispose(x); { Felszabadítjuk az átmeneti adattömböt             }
end;
```

Instruments: A hangszerek beolvasása és tárolása a GUS DRAM-ban. Ez az eljárás tölti fel a hangmintákkal a GUS DRAM-ot. A program elején abból indultunk ki, hogy a GUS DRAM elég az összes minta tárolásához, és hogy egyetlen minta sem nagyobb 64K-nál. Az eljárás nem végez semmiféle ellenőrzést, ezért hiba léphet fel, ha ezek a feltételek nem teljesülnek.

```
procedure ConvertPatterns;
var
   p,q     : word;
   Period  : word;
   Instr   : byte;
   Cmd     : byte;
   Param   : byte;
   Note    : byte;
   Src     : array[ 0..3] of byte;

  { Belső eljárás: egy note átalakítása }
  procedure Convert(var S);
  var X:Tnote;
      j:byte;
  begin
   X:=Tnote(S);
   Period:=((X[ 0] and $0f) shl 8) or X[ 1];
    Instr:=(X[ 0] and $f0) or (X[ 2] shr 4);
    Cmd:=X[ 2] and $0f;
    Param:=X[ 3];
    if Cmd=$0E then
     begin
      Cmd:=$E0 or(X[ 3] shr 4);
      Param:=X[ 3] and $0F;
     end;
    if Period=0 then Note:=$ff
     else
      for j:=0 to 35 do
       if Period=PeriodTABLE[ j] then Note:=j;
    X[ 0] :=Note;
    X[ 1] :=Instr-1;
    X[ 2] :=Cmd;
    X[ 3] :=Param;
    move(X,S,4);
  end;

begin
 for p:=0 to max do
  for q:=0 to 255 do
   Convert(M^[ p,q* 4] );
end;
```

ConvertPatterns: A *patternek* átalakítása. Ez az eljárás nagyon fontos feladatot lát el: az AMIGA formátumú *patterneket* egy egyszerűbb formára alakítja át. A pattern eredeti alakját a fájl formátumának ismertetésénél adtuk meg. Az átalakított adat egy négybájtos mező, amely a következőképpen fest:

Bájt	Jelentés
0. bájt	hangjegy (0...35)
1. bájt	hangszer (0...30)
2. bájt	effektus parancs
3. bájt	effektus paraméter

Hogyan történik az átalakítás? A periódustáblázat alapján a hangjegy száma könnyen meghatározható. Mivel a szabványos MOD fájl három oktávot tárol, ezért a hangjegy csupán 36-féle lehet. Ezt tárolja a 0. bájt. Ha a periódus szerint érvénytelen hangjegy van az adott *patternpozícióban*, akkor a 0. bájt $ff lesz. (Ezt a lejátszás során majd figyelembe kell venni.) A hangjegy meghatározása után logikai műveletekkel és eltolásokkal meghatározzuk a hangszer kódját, az effektus parancsot és annak paraméterét. Ezeket sorra tároljuk egy-egy bájtban. Az eljárás minden egyes pattern minden egyes hangját (egy *patternben* 4*64, azaz 256 *note* található) átalakítja. A létrehozott új adatokat a *PlayPatternLine* eljárás le tudja játszani.

```
procedure PlayPatternLine; { Mp: patternszám, Ml:sorszám }
var
    j  : byte;
    Cn : Tnote;
    Br : boolean;
 Np,Nl : word;
begin
 Br:=false;
 { Egy pattern-sor: minden csatornára egy-egy note }
 for j:=0 to 3 do { 4 csatorna }
  begin
   { Bemásoljuk az aktuális note-ot }
   move(M^[ Seq[ Mp] ,Ml*16+j*4] ,Cn,4);
   { Az effektus parancsok végrehajtása (nem mind!) }
   case cn[ 2] of
    $0b : begin { Position jump }
            Nl:=00;Np:=cn[ 3] and $7f;
            Br:=true;
          end;
    $0c : Volume(j,cn[ 3] div v); { Set volume }
    $0d : begin { Pattern break }
            Np:=Mp+1; Nl:=cn[ 3] and $3f;
            dec(Nl);
            Br:=true;
          end;
    $0f : begin { Set speed }
            Spd:=Speed[ (cn[ 3] shr 1) and $f];
          end;
   end;
   { Kiadjuk a hangot, ha a hangjegy érvényes }
   if Cn[ 0] <>$ff then KeyON(j,cn[ 0] ,cn[ 1] ,cn[ 2] );
  end;
```

```
 if Br then { Pattern break, Position jump }
  begin
   Mp:=Np;Ml:=Nl;
  end;
end;
```

PlayPatternLine: Egy *patternsor* lejátszása. Ez az eljárás felelős egyetlen *patternsor* négy *note*-jának lejátszásáért. A belső *for* ciklus (*j* változó) végigveszi a négy *note*-ot, és a *KeyOn* eljárással kiadja a megfelelő hangot. Az effektus parancsok végrehajtását is itt oldjuk meg. Amint látható, az összes parancsból csupán négyet valósítottunk meg, hogy az eljárás rövid legyen. (Természetesen nincs akadálya a bővítésnek.) Mivel sok fontos parancs kimaradt, ezért előfordulhat, hogy néhány MOD fájl furcsán fog szólni! :–\

```
procedure PlayModule;

var
  ShiftState: byte absolute $40:$17;

 { Belső eljárás: a következő pattern megjelenítése }
 procedure nextpattern;
 begin
  mem[ $b800:(Mp shr 4)*160+(Mp and 15)*6+161] :=$0b;
  mem[ $b800:(Mp shr 4)*160+(Mp and 15)*6+163] :=$0b;
  inc(Mp);
  mem[ $b800:(Mp shr 4)*160+(Mp and 15)*6+161] :=$4f;
  mem[ $b800:(Mp shr 4)*160+(Mp and 15)*6+163] :=$4f;
 end;

 { Belső függvény: a shift billentyűk lekérdezése }
 function Shift:byte; assembler;
 asm
   mov      ES,Seg0040
   mov      al,ES:ShiftState
 end;

begin
 { Kiürítjük a billentyűzetbuffert }
 while keypressed do readkey;
  Mp:=$ffff;
  nextpattern;  { Kijelzés }
  repeat
   Ml:=0;
   repeat
    { Hangerőnövelés }
    if (Shift and 1=1)and(v<40) then inc(v);
    { Hangerőcsökkentés }
    if (Shift and 2=2)and(v>1)  then dec(v);
    { Kijelzés }
    gotoxy(1,10);
```

```
    write('Pattern:',hex(Mp),'[',hex(seq[mp]),'] Sor:',hex(Ml),
          ' Hangerő:',41-v:2);
    { Megszólaltatjuk a csatornákat }
    PlayPatternLine;
    { Várakozás a sebességnek megfelelően }
    delay(Spd);
    inc(Ml); { Növeljük a patternsorszámot }
   until (Ml=64) or keypressed;
   nextpattern; { Növeljük a patternszámot }
  until (Mp=Max+1) or keypressed;
end;
```

PlayModule: A teljes MOD fájl lejátszása. Ez az eljárás a fő lejátszó rutin. A lejátszás mellett néhány alapvető kijelzési feladatot is végez, valamint figyeli a billentyűzet lenyomását. A hangerőt a két *shift* billentyűvel lehet szabályozni, más billentyű lenyomásakor a lejátszás leáll.

```
BEGIN
 clrscr;
 { A hangkártya inicializálása }
 GravisFindBase;
 GravisReset;
 Gpt:=0;
 { Megállapítjuk, vagy bekérjük a fájl nevét }
 if paramcount=0 then
  begin
   write('A MOD fájl neve (kiterjesztés nélkül) : ');
   readln(Name);
  end else Name:=paramstr(1);
 Name:=Name+'.MOD';
 assign(f,Name);
 reset(f,1);
 { Létrehozzuk a fejlécet a heapben }
 new(H);
 { Betöltjük a fájl fejlécét }
 blockread(f,H^,Sizeof(Theader));
 ID[0]:=#4; move(H^[$438],ID[1],4);
 { Megállapítjuk, hogy a fájl valódi MOD fájl-e }
 if not ((ID='M.K.')or(ID='FLT4')) then
  begin
   writeln('Ez nem egy 31 hangszeres négycsatornás MOD fájl! ');
   close(f); HALT;
  end else writeln('Module ID: ',ID);
 Snu:=H^[$3b6];                { Ennyi patternt kell lejátszani }
 Max:=$00;
 { Megkeressük a legnagyobb patternszámot }
 for j:=$3b8 to $3b7+Snu do
  if H^[j]>Max then Max:=H^[j];{ Max: a legnagyobb patternszám   }
 writeln('Patternek száma:',Max+1);
```

```
 move(H^[ $3b8],Seq,Max+1);
 { Beolvassuk a patterneket }
 new(M);
 blockread(f,M^,(Max+1)*1024);  { A patternek beolvasása  }
 Instruments;                   { A hangszerek beolvasása }
 close(f);
 { Minden adat beolvasása korrektül megtörtént! }

 writeln('A patternek átalakítása ...');
 ConvertPatterns; { Átalakítás }
 dispose(H);      { A fejléc most már szükségtelen }

 { A balanszértékek beállítása az első négy csatornára }
 GravisBalance($00,$02);       { Bal  }
 GravisBalance($01,$0D);       { Jobb }
 GravisBalance($02,$02);       { Bal  }
 GravisBalance($03,$0D);       { Jobb }

 { Beállítjuk a megszakítási rutint }
 SetIRQservice(@GravisIRQ);

 clrscr;
 b:=0;
 { Kiírjuk a patternsorozatot (sequence) }
 writeln('Pattern sorozat:');
 for j:=0 to 7 do
  begin
   for k:=0 to 15 do
    begin
     if b<max+1 then textattr:=$0b
                else textattr:=$01;
     write(hex(seq[ b]),' ');
     inc(b);
    end;
   writeln;
  end;

 textattr:=$07;
 Spd:=Speed[ 3];    { Alapbeállítású sebesség     }
 v:=5;              { Hangerőosztó                }
 { Lejátszás indul }
 PlayModule;

 { Visszaállítjuk a megszakítást }
 ResetIRQservice;
 { Tiltjuk a hangkiadást }
 GravisReset;
 { Felszabadítjuk a heapet }
 dispose(M);

END. of program MODplay
```

Hiányosságok

Sok mindent igencsak le kellett egyszerűsíteni, hogy egy ilyen tömör program is képes legyen a lejátszásra. Ezért a programnak igen sok hiányossága van (ezek kiküszöbölését az Olvasóra bízzuk):

– A GUS DRAM méretét nem ellenőrzi, de az összes hangszernek bele kell férnie a kártya memóriájába.

– A MOD fájlnak hibátlannak kell lennie, és csak négycsatornás, 31 hangszeres fájlokat használhatunk.

– A fájlban legfeljebb 63 pattern lehet.

– A fájl nevében a kiterjesztést nem szabad megadni.

– A program csak a legszükségesebb effektusokat ismeri, ezért néhány MOD fájl lejátszása hibás lehet.

11. Összegzés

A könyv befejezéseképpen röviden ismertetjük az egyes hangkártyatípusok műszaki paramétereit és a legfontosabb jellemző adatokat. Távol áll tőlünk, hogy bármelyik kártya mellett állást foglaljunk és a többihez képest valamelyiket is előnyben részesítsük. Nem akarunk olyasféle kijelentéseket tenni, hogy „ez jobb, mint az", vagy „ezt nem is érdemes megvenni"; semmilyen ehhez hasonló reklámozás vagy antireklámozás nem e könyv feladata. Itt csupán felsoroljuk a kártyák tulajdonságait, és a döntés azután legyen az Olvasóé!

ADLIB

FM csatornák száma	9 (vagy 6+ 5) mono
CD minőségű hang	– – –
Digitális csatornák száma	– – –
Digitális csatornák típusa	– – –
Digitális csatornák felbontása lejátszáskor	– – –
Digitális csatornák felbontása felvételkor	– – –
Maximális lejátszási mintavételi frekvencia	– – –
Maximális felvételi mintavételi frekvencia	– – –
MIDI interfész, MIDI támogatás	– – –
CD-ROM interfész	– – –
Joystick-csatlakoztatási lehetőség	– – –
Joysticksebesség-vezérlési lehetőség	– – –

Sound Blaster 1.0

FM csatornák száma	9 (vagy 6+ 5) mono
CD minőségű hang	– – –
Digitális csatornák száma	1
Digitális csatornák típusa	mono
Digitális csatornák felbontása lejátszáskor	8 bit
Digitális csatornák felbontása felvételkor	8 bit
Maximális lejátszási mintavételi frekvencia	4000 – 23000 Hz
Maximális felvételi mintavételi frekvencia	4000 – 23000 Hz
MIDI interfész, MIDI támogatás	van
CD-ROM interfész	– – –
Joystick-csatlakoztatási lehetőség	van
Joysticksebesség-vezérlési lehetőség	– – –

Sound Blaster 1.5

FM csatornák száma	9 (vagy 6+ 5) mono
CD minőségű hang	– – –
Digitális csatornák száma	1
Digitális csatornák típusa	mono
Digitális csatornák felbontása lejátszáskor	8 bit
Digitális csatornák felbontása felvételkor	8 bit
Maximális lejátszási mintavételi frekvencia	4000 – 23000 Hz
Maximális felvételi mintavételi frekvencia	4000 – 23000 Hz
MIDI interfész, MIDI támogatás	van
CD-ROM interfész	– – –
Joystick-csatlakoztatási lehetőség	van
Joysticksebesség-vezérlési lehetőség	– – –

Sound Blaster 2.0

FM csatornák száma	9 (vagy 6+ 5) mono
CD minőségű hang	– – –
Digitális csatornák száma	1
Digitális csatornák típusa	mono
Digitális csatornák felbontása lejátszáskor	8 bit
Digitális csatornák felbontása felvételkor	8 bit
Maximális lejátszási mintavételi frekvencia	4000 – 44100 Hz
Maximális felvételi mintavételi frekvencia	4000 – 23000 Hz
MIDI interfész, MIDI támogatás	van
CD-ROM interfész	– – –
Joystick-csatlakoztatási lehetőség	van
Joysticksebesség-vezérlési lehetőség	– – –

Sound Blaster Pro 1.0

FM csatornák száma	2*9 (vagy 2*6 +2*5) sztereó
CD minőségű hang	– – –
Digitális csatornák száma	1
Digitális csatornák típusa	mono/sztereó
Digitális csatornák felbontása lejátszáskor	8 bit
Digitális csatornák felbontása felvételkor	8 bit
Maximális lejátszási mintavételi frekvencia	4000 – 44100 Hz
Maximális felvételi mintavételi frekvencia	4000 – 22050, 44100 Hz (sztereó/mono)
MIDI interfész, MIDI támogatás	van
CD-ROM interfész	van
Joystick-csatlakoztatási lehetőség	van
Joysticksebesség-vezérlési lehetőség	– – –

Sound Blaster Pro 2.0

FM csatornák száma	2*9 (vagy 2*6 +2*5) OPL–3
CD minőségű hang	– – –
Digitális csatornák száma	1
Digitális csatornák típusa	mono/sztereó
Digitális csatornák felbontása lejátszáskor	8 bit
Digitális csatornák felbontása felvételkor	8 bit
Maximális lejátszási mintavételi frekvencia	4000 – 44100 Hz
Maximális felvételi mintavételi frekvencia	4000 – 22050, 44100 Hz (sztereó/mono)
MIDI interfész, MIDI támogatás	van
CD-ROM interfész	van
Joystick-csatlakoztatási lehetőség	van
Joysticksebesség-vezérlési lehetőség	– – –

Sound Blaster 16

FM csatornák száma	2*9 (vagy 2*6 +2*5) OPL–3
CD minőségű hang	van
Digitális csatornák száma	1
Digitális csatornák típusa	mono/sztereó
Digitális csatornák felbontása lejátszáskor	8 bit, 16 bit (ASP)
Digitális csatornák felbontása felvételkor	8 bit, 16 bit (ASP)
Maximális lejátszási mintavételi frekvencia	5000 – 48000 Hz (ASP)
Maximális felvételi mintavételi frekvencia	5000 – 48000 Hz (ASP)
MIDI interfész, MIDI támogatás	van
CD-ROM interfész	van
Joystick-csatlakoztatási lehetőség	van
Joysticksebesség-vezérlési lehetőség	– – –

Sound Blaster AWE 32

FM csatornák száma	2*9 (vagy 2*6 +2*5) OPL–3
CD minőségű hang	van
Digitális csatornák száma	1+ 32(*EMU 8000*)
Digitális csatornák típusa	mono/sztereó
Digitális csatornák felbontása lejátszáskor	8 bit, 16 bit (SB16 ASP)
Digitális csatornák felbontása felvételkor	8 bit, 16 bit (SB16 ASP)
Maximális lejátszási mintavételi frekvencia	5000 – 48000 Hz (SB16 ASP)
Maximális felvételi mintavételi frekvencia	5000 – 48000 Hz (SB16 ASP)
MIDI interfész, MIDI támogatás	van
CD-ROM interfész	van
Joystick-csatlakoztatási lehetőség	van
Joysticksebesség-vezérlési lehetőség	– – –

Gravis Ultrasound

FM csatornák száma	32 sztereó, emulációval
CD minőségű hang	van
Digitális csatornák száma	32
Digitális csatornák típusa	mono/sztereó
Digitális csatornák felbontása lejátszáskor	8 bit, 16 bit (GF1)
Digitális csatornák felbontása felvételkor	8 bit (GF1)
Maximális lejátszási mintavételi frekvencia	??? – 44100 Hz (14 csatorna) ??? – 19293 Hz (32 csatorna)
Maximális felvételi mintavételi frekvencia	??? – 44100 Hz
MIDI interfész, MIDI támogatás	van
CD-ROM interfész	van
Joystick-csatlakoztatási lehetőség	van
Joysticksebesség-vezérlési lehetőség	van

A függelék

MIDI adattáblázatok

A GENERAL MIDI szabvány hangszerkészlete							
$00	*Acoustic grand piano*	$20	*Acoustic bass*	$40	*Soprano sax*	$60	*RainFX (1st)*
$01	*Bright acoustic piano*	$21	*Electric bass (finger)*	$41	*Alto sax*	$61	*Soundtrack FX (2nd)*
$02	*Electric grand piano*	$22	*Electric bass (pick)*	$42	*Tenor*	$62	*Crystal FX (3rd)*
$03	*Honky-tonk piano*	$23	*Fretless bass*	$43	*Baritone sax*	$63	*Atmosphere FX (4th)*
$04	*Rhodes piano*	$24	*1st step bass*	$44	*Oboe*	$64	*Brightness FX (5th)*
$05	*Chorused piano*	$25	*2nd step bass*	$45	*English horn*	$65	*Goblins FX (6th)*
$06	*Harpsichord*	$26	*1st synth bass*	$46	*Bassoon*	$66	*Echoes FX (7th)*
$07	*Clavinet*	$27	*2nd synth bass*	$47	*Clarinet*	$67	*Sci-fi FX (8th)*
$08	*Celesta*	$28	*Violin*	$48	*Piccolo*	$68	*Sitar*
$09	*Glockenspiel*	$29	*Viola*	$49	*Flute*	$69	*Banjo*
$0A	*Music box*	$2A	*Cello*	$4A	*Recorder*	$6A	*Shamisen*
$0B	*Vibraphone*	$2B	*Contrabass*	$4B	*Pan flute*	$6B	*Koto*
$0C	*Marimba*	$2C	*Tremolo strings*	$4C	*Blown bottle*	$6C	*Kalimba*
$0D	*Xylophone*	$2D	*Pizzicato strings*	$4D	*Shakuhachi*	$6D	*Bag pipe*
$0E	*Tubular bells*	$2E	*Orchestral harp*	$4E	*Whistle*	$6E	*Fiddle*
$0F	*Dulcimer*	$2F	*Timpani*	$4F	*Ocarina*	$6F	*Shanai*
$10	*Hammond organ*	$30	*1st String ensemble*	$50	*Suare lead (1st)*	$70	*Tinkle bell*
$11	*Percussive organ*	$31	*2nd String ensemble*	$51	*Saw lead (2nd)*	$71	*Agogo*
$12	*Rock organ*	$32	*1st Synth strings*	$52	*Calliope lead (3rd)*	$72	*Steel drums*
$13	*Church organ*	$33	*2nd Synth strings*	$53	*Chiff lead (4th)*	$73	*Woodblock*
$14	*Reed organ*	$34	*Choir Aahs*	$54	*Charang lead (5th)*	$74	*Taiko drum*
$15	*Accordion*	$35	*Voice Oohs*	$55	*Voice lead (6th)*	$75	*Melodic tom*
$16	*Harmonica*	$36	*Synth voice*	$56	*Fifths lead (7th)*	$76	*Synth drum*
$17	*Tango accordian*	$37	*Orchestra hit*	$57	*Bass+ lead lead (8th)*	$77	*Reverse cymbal*
$18	*Nylon acoustic guitar*	$38	*Trumpet*	$58	*New age pad (1st)*	$78	*Guitar fret noise*
$19	*Steel acoustic guitar*	$39	*Trombone*	$59	*Warm pad (2nd)*	$79	*Breath noise*
$1A	*Jazz electric guitar*	$3A	*Tuba*	$5A	*Polysynth pad (3rd)*	$7A	*Seashore*
$1B	*Clean electric guitar*	$3B	*Muted trumpet*	$5B	*Choir pad (4th)*	$7B	*Bird tweet*
$1C	*Muted electric guitar*	$3C	*French horn*	$5C	*Bowed pad (5th)*	$7C	*Telephone ring*
$1D	*Overdriven guitar*	$3D	*Brass section*	$5D	*Metallic pad (6th)*	$7D	*Helicopter*
$1E	*Distortion guitar*	$3E	*1st Synth brass*	$5E	*Halo pad (7th)*	$7E	*Applause*
$1F	*Guitar harmonics*	$3F	*2nd Synth brass*	$5F	*Sweep pad (8th)*	$7F	*Gunshot*

A GENERAL MIDI szabvány dobkészlete (*channel #10*)							
35	*Acoustic bass drum*	47	*Low-mid tom*	59	*2nd Ride cymbal*	71	*Short whistle*
36	*1st Bass drum*	48	*Hi-mid tom*	60	*Hi bongo*	72	*Long whistle*
37	*Side stick*	49	*1st Crash cymbal*	61	*Low bongo*	73	*Short guiro*
38	*Acoustic snare*	50	*High tom*	62	*Mute hi conga*	74	*Long guiro*
39	*Hand clap*	51	*1st Ride cymbal*	63	*Open hi conga*	75	*Claves*
40	*Electric snare*	52	*Chinese cymbal*	64	*Low conga*	76	*High wood block*
41	*Low floor tom*	53	*Ride bell*	65	*High timbale*	77	*Low wood block*
42	*Closed hi-hat*	54	*Tambourine*	66	*Low timbale*	78	*Mute cuica*
43	*High floor tom*	55	*Splash cymbal*	67	*High agogo*	79	*Open cuica*
44	*Pedal hi-hat*	56	*Cowbell*	68	*Low agogo*	80	*Mute triangle*
45	*Low tom*	57	*2nd Crash cymbal*	69	*Cabasa*	81	*Open triangle*
46	*Open hi-hat*	58	*Vibraslap*	70	*Maracas*	82	- -

MIDI kontrollerek							
$00	*Bank select MSB*	$20	*Bank select LSB*	$40	*Sustain pedal*	$60	*Data increment*
$01	*Modulation MSB*	$21	*Modulation LSB*	$41	*Portamento off/on*	$61	*Data decrement*
$02	*Breath controller MSB*	$22	*Breath controller LSB*	$42	*Sostenuto off/on*	$62	*NRP LSB*
$03	---	$23	---	$43	*Soft pedal*	$63	*NRP MSB*
$04	*Foot controller MSB*	$24	*Foot controller LSB*	$44	*Legato off/on*	$64	*RP LSB*
$05	*Portamento time MSB*	$25	*Portamento time LSB*	$45	*Hold 2 off/on*	$65	*RP MSB*
$06	*Data entry MSB*	$26	*Data entry LSB*	$46	*Sound Variation*	$66	---
$07	*Volume MSB*	$27	*Volume LSB*	$47	*Harmonic content*	$67	---
$08	*Balance MSB*	$28	*Balance LSB*	$48	*Release time*	$68	---
$09	---	$29	---	$49	*Attack time*	$69	---
$0A	*Pan controller MSB*	$2A	*Pan controller LSB*	$4A	*Brightness*	$6A	---
$0B	*1st effect control MSB*	$2B	*1st effect control LSB*	$4B	---	$6B	---
$0C	*2nd effect control MSB*	$2C	*2nd effect control LSB*	$4C	---	$6C	---
$0D	---	$2D	---	$4D	---	$6D	---
$0E	---	$2E	---	$4E	---	$6E	---
$0F	---	$2F	---	$4F	---	$6F	---
$10	*General pur-c. #1 MSB*	$30	*General pur-c. #1 LSB*	$50	*General pur-c. #5*	$70	---
$11	*General pur-c. #2 MSB*	$31	*General pur-c. #2 LSB*	$51	*General pur-c. #6*	$71	---
$12	*General pur-c. #3 MSB*	$32	*General pur-c. #3 LSB*	$52	*General pur-c. #7*	$72	---
$13	*General pur-c. #4 MSB*	$33	*General pur-c. #4 LSB*	$53	*General pur-c #8*	$73	---
$14	---	$34	---	$54	---	$74	---
$15	---	$35	---	$55	---	$75	---
$16	---	$36	---	$56	---	$76	---
$17	---	$37	---	$57	---	$77	---
$18	---	$38	---	$58	---	$78	*Off All Sound*
$19	---	$39	---	$59	---	$79	*Resel All controllers*
$1A	---	$3A	---	$5A	---	$7A	*Local control off/on*
$1B	---	$3B	---	$5B	*Effect depth #1*	$7B	*All notes off*
$1C	---	$3C	---	$5C	*Effect depth #2*	$7C	*Omni off*
$1D	---	$3D	---	$5D	*Effect depth #3*	$7D	*Omni on*
$1E	---	$3E	---	$5E	*Effect depth #4*	$7E	*Mono on*
$1F	---	$3F	---	$5F	*Effect depth #5*	$7F	*Mono off*

Roland GS nem regisztrált MIDI paraméterek		
MSB	**LSB**	***Function***
$01	$08	*Vibrato speed*
$01	$09	*Vibrato depth*
$01	$0A	*Vibrato delay*
$01	$20	*Filter frequency*
$01	$21	*Filter position*
$01	$63	*Attack time*
$01	$64	*Decay time*
$01	$66	*Release time*
$18	*drum*	*Drum note select*
$1A	*drum*	*Drum volume control*
$1C	*drum*	*Drum pan position*
$1D	*drum*	*Drum verb effect*
$1E	*srum*	*Drum chorus effect*

B függelék

SBI hangszerek paraméterei

A hangszer neve: **ACCORDN.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00100100 ($24)	00110001 ($31)
- alaphangerő:	01001111 ($4F)	00000000 ($00)
- burkológörbe:		
	AD: $F2	AD: $52
	SR: $0B	SR: $0B
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	7	
- összekapcsolás:	FM szintézis	

A hangszer neve: **ALTOVIOL.SBI**

	Modulátor:	Carrier:
- karakterisztika:	01110000 ($70)	01110001 ($71)
- alaphangerő:	11010000 ($D0)	10000000 ($80)
- burkológörbe:		
	AD: $52	AD: $31
	SR: $11	SR: $FE
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	0	
- összekapcsolás:	FM szintézis	

A hangszer neve: **BAGPIPE.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00110000 ($30)	01100011 ($63)
- alaphangerő:	00000000 ($00)	00000000 ($00)
- burkológörbe:		
	AD: $FF	AD: $65
	SR: $A0	SR: $0B
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	0	
- összekapcsolás:	FM szintézis	

A hangszer neve: **BANJO.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00110001 ($31)	00010110 ($16)
- alaphangerő:	10000111 ($87)	10000000 ($80)
- burkológörbe:		
	AD: $A1	AD: $7D
	SR: $11	SR: $43
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	4	
- összekapcsolás:	FM szintézis	

A hangszer neve: **BASS.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00000001 ($01)	00000001 ($01)
- alaphangerő:	00011101 ($1D)	00000000 ($00)
- burkológörbe:		
	AD: $F2	AD: $F5
	SR: $EF	SR: $78
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	5	
- összekapcsolás:	FM szintézis	

A hangszer neve: **BELLS.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00000111 ($07)	00010010 ($12)
- alaphangerő:	01001111 ($4F)	00000000 ($00)
- burkológörbe:		
	AD: $F2	AD: $F2
	SR: $60	SR: $72
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	4	
- összekapcsolás:	FM szintézis	

A hangszer neve: **BELSHORT.SBI**

	Modulátor:	Carrier:
- karakterisztika:	11100000 ($E0)	01110000 ($70)
- alaphangerő:	01100011 ($63)	10000000 ($80)
- burkológörbe:		
	AD: $F8	AD: $F7
	SR: $F3	SR: $F3
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	2	
- összekapcsolás:	FM szintézis	

A hangszer neve: **BNCEBASS.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00100000 ($20)	00100001 ($21)
- alaphangerő:	01001011 ($4B)	00000000 ($00)
- burkológörbe:		
	AD: $7B	AD: $F5
	SR: $04	SR: $72
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	7	
- összekapcsolás:	FM szintézis	

A hangszer neve: **BRASS.SBI**

	Modulátor:	Carrier:
- karakterisztika:	01100001 ($61)	01100000 ($60)
- alaphangerő:	00011100 ($1C)	00000000 ($00)
- burkológörbe:		
	AD: $71	AD: $81
	SR: $AE	SR: $2E
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	7	
- összekapcsolás:	FM szintézis	

A hangszer neve: **CELESTA.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00110011 ($33)	00010100 ($14)
- alaphangerő:	10000111 ($87)	10000000 ($80)
- burkológörbe:		
	AD: $01	AD: $7D
	SR: $10	SR: $33
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	4	
- összekapcsolás:	FM szintézis	

A hangszer neve: **CELLO.SBI**

	Modulátor:	Carrier:
- karakterisztika:	01110000 ($70)	01110000 ($70)
- alaphangerő:	11000101 ($C5)	10000100 ($84)
- burkológörbe:		
	AD: $52	AD: $31
	SR: $11	SR: $FE
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	4	
- összekapcsolás:	FM szintézis	

A hangszer neve: **CHIRP.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00000000 ($00)	00001100 ($0C)
- alaphangerő:	00101110 ($2E)	00000000 ($00)
- burkológörbe:		
	AD: $F0	AD: $5F
	SR: $F0	SR: $F0
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	7	
- összekapcsolás:	FM szintézis	

A hangszer neve: **CLARINET.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00110010 ($32)	10100001 ($A1)
- alaphangerő:	10011010 ($9A)	10000010 ($82)
- burkológörbe:		
	AD: $51	AD: $A2
	SR: $1B	SR: $3B
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	6	
- összekapcsolás:	FM szintézis	

A hangszer neve: **CYMBAL.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00000001 ($01)	00000000 ($00)
- alaphangerő:	00000000 ($00)	00000000 ($00)
- burkológörbe:		
	AD: $F5	AD: $D6
	SR: $B5	SR: $4F
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	0	
- összekapcsolás:	FM szintézis	

A hangszer neve: **ELGUIT1.SBI**

	Modulátor:	Carrier:
- karakterisztika:	11110001 ($F1)	00100001 ($21)
- alaphangerő:	00000001 ($01)	00001101 ($0D)
- burkológörbe:		
	AD: $97	AD: $F1
	SR: $17	SR: $18
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	4	
- összekapcsolás:	FM szintézis	

A hangszer neve: **ELGUIT2.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00010011 ($13)	00010001 ($11)
- alaphangerő:	10010110 ($96)	10000000 ($80)
- burkológörbe:		
	AD: $FF	AD: $FF
	SR: $21	SR: $03
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	5	
- összekapcsolás:	FM szintézis	

A hangszer neve: **ELPIANO1.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00000001 ($01)	00000001 ($01)
- alaphangerő:	01001111 ($4F)	00000100 ($04)
- burkológörbe:		
	AD: $F1	AD: $D2
	SR: $50	SR: $7C
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	3	
- összekapcsolás:	FM szintézis	

A hangszer neve: **ELPIANO2.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00000010 ($02)	00000010 ($02)
- alaphangerő:	00100010 ($22)	00000000 ($00)
- burkológörbe:		
	AD: $F2	AD: $F5
	SR: $13	SR: $43
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	7	
- összekapcsolás:	FM szintézis	

A hangszer neve: **GUITAR.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00000001 ($01)	00000001 ($01)
- alaphangerő:	00010001 ($11)	00000000 ($00)
- burkológörbe:		
	AD: $F2	AD: $F5
	SR: $1F	SR: $88
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	5	
- összekapcsolás:	FM szintézis	

A hangszer neve: **HARP.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00000010 ($02)	00000001 ($01)
- alaphangerő:	01010111 ($57)	10000000 ($80)
- burkológörbe:		
	AD: $F5	AD: $F6
	SR: $56	SR: $54
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	0	
- összekapcsolás:	FM szintézis	

A hangszer neve: **KEYBRD.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00000001 ($01)	11100100 ($E4)
- alaphangerő:	00000000 ($00)	00000011 ($03)
- burkológörbe:		
	AD: $F0	AD: $F3
	SR: $F0	SR: $36
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	0	
- összekapcsolás:	FM szintézis	

A hangszer neve: **KOTO.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00001110 ($0E)	00000010 ($02)
- alaphangerő:	01000000 ($40)	00000000 ($00)
- burkológörbe:		
	AD: $09	AD: $F7
	SR: $53	SR: $94
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	7	
- összekapcsolás:	FM szintézis	

A hangszer neve: **LASER.SBI**

	Modulátor:	Carrier:
- karakterisztika:	11100110 ($E6)	00110000 ($30)
- alaphangerő:	00000000 ($00)	00000000 ($00)
- burkológörbe:		
	AD: $25	AD: $F0
	SR: $B5	SR: $40
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	0	
- összekapcsolás:	FM szintézis	

A hangszer neve: **MERI.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00000001 ($01)	01110100 ($74)
- alaphangerő:	01000000 ($40)	11000001 ($C1)
- burkológörbe:		
	AD: $18	AD: $F6
	SR: $03	SR: $90
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	7	
- összekapcsolás:	FM szintézis	

A hangszer neve: **NOISE.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00001110 ($0E)	00001110 ($0E)
- alaphangerő:	01000000 ($40)	00000000 ($00)
- burkológörbe:		
	AD: $D1	AD: $F2
	SR: $53	SR: $7F
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	7	
- összekapcsolás:	FM szintézis	

A hangszer neve: **OBOE.SBI**

	Modulátor:	Carrier:
- karakterisztika:	01110001 ($71)	00100010 ($22)
- alaphangerő:	11000101 ($C5)	00000101 ($05)
- burkológörbe:		
	AD: $6E	AD: $8B
	SR: $17	SR: $0E
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	1	
- összekapcsolás:	FM szintézis	

A hangszer neve: **ORGAN.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00000011 ($03)	00000001 ($01)
- alaphangerő:	01011011 ($5B)	10000000 ($80)
- burkológörbe:		
	AD: $F0	AD: $F0
	SR: $1F	SR: $1F
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	5	
- összekapcsolás:	FM szintézis	

A hangszer neve: **ORGAN2.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00000011 ($03)	00000001 ($01)
- alaphangerő:	01011011 ($5B)	10001101 ($8D)
- burkológörbe:		
	AD: $F0	AD: $F0
	SR: $1F	SR: $13
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	5	
- összekapcsolás:	FM szintézis	

A hangszer neve: **ORGAN3.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00000011 ($03)	00000001 ($01)
- alaphangerő:	01011011 ($5B)	10010010 ($92)
- burkológörbe:		
	AD: $F0	AD: $F0
	SR: $1F	SR: $12
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	5	
- összekapcsolás:	FM szintézis	

A hangszer neve: **PIANO.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00000011 ($03)	00010111 ($17)
- alaphangerő:	01001111 ($4F)	00000000 ($00)
- burkológörbe:		
	AD: $F1	AD: $F2
	SR: $53	SR: $74
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	3	
- összekapcsolás:	FM szintézis	

***A hangszer neve:* PIANO2.SBI**

	Modulátor:	Carrier:
- karakterisztika:	10000001 ($81)	00010011 ($13)
- alaphangerő:	10011101 ($9D)	00000000 ($00)
- burkológörbe:		
	AD: $F2	AD: $F2
	SR: $51	SR: $F1
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	3	
- összekapcsolás:	FM szintézis	

***A hangszer neve:* PIANO3.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00000001 ($01)	00000001 ($01)
- alaphangerő:	01001111 ($4F)	00000100 ($04)
- burkológörbe:		
	AD: $F1	AD: $D2
	SR: $50	SR: $7C
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	3	
- összekapcsolás:	FM szintézis	

***A hangszer neve:* SCRATCH.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00000111 ($07)	00000000 ($00)
- alaphangerő:	00000000 ($00)	00000000 ($00)
- burkológörbe:		
	AD: $F0	AD: $5C
	SR: $F0	SR: $DC
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	7	
- összekapcsolás:	FM szintézis	

***A hangszer neve:* SITAR.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00000001 ($01)	00001000 ($08)
- alaphangerő:	01000000 ($40)	01000000 ($40)
- burkológörbe:		
	AD: $F1	AD: $F1
	SR: $53	SR: $53
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	0	
- összekapcsolás:	FM szintézis	

***A hangszer neve:* SYNTH.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00000111 ($07)	00000001 ($01)
- alaphangerő:	10000111 ($87)	10000000 ($80)
- burkológörbe:		
	AD: $F0	AD: $F0
	SR: $05	SR: $05
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	2	
- összekapcsolás:	FM szintézis	

***A hangszer neve:* TOM.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00000100 ($04)	00000000 ($00)
- alaphangerő:	00000000 ($00)	00000000 ($00)
- burkológörbe:		
	AD: $F7	AD: $D6
	SR: $B5	SR: $4F
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	0	
- összekapcsolás:	FM szintézis	

***A hangszer neve:* TROMB.SBI**

	Modulátor:	Carrier:
- karakterisztika:	01110001 ($71)	10100001 ($A1)
- alaphangerő:	00011100 ($1C)	10000000 ($80)
- burkológörbe:		
	AD: $41	AD: $92
	SR: $1F	SR: $3B
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	7	
- összekapcsolás:	FM szintézis	

***A hangszer neve:* VIBRA.SBI**

	Modulátor:	Carrier:
- karakterisztika:	01000100 ($44)	01100000 ($60)
- alaphangerő:	01010011 ($53)	10000000 ($80)
- burkológörbe:		
	AD: $F5	AD: $FD
	SR: $33	SR: $25
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	3	
- összekapcsolás:	FM szintézis	

***A hangszer neve:* VIOLIN.SBI**

	Modulátor:	Carrier:
- karakterisztika:	11100001 ($E1)	00100010 ($22)
- alaphangerő:	10001000 ($88)	10000000 ($80)
- burkológörbe:		
	AD: $62	AD: $53
	SR: $29	SR: $2C
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	6	
- összekapcsolás:	FM szintézis	

***A hangszer neve:* WAVE.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00000000 ($00)	00010100 ($14)
- alaphangerő:	00000010 ($02)	10000000 ($80)
- burkológörbe:		
	AD: $00	AD: $1B
	SR: $F0	SR: $A2
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	7	
- összekapcsolás:	FM szintézis	

A hangszer neve: **XYLO1.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00010001 ($11)	00110001 ($31)
- alaphangerő:	00101101 ($2D)	00000000 ($00)
- burkológörbe:		
	AD: $C8	AD: $F5
	SR: $2F	SR: $F5
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	6	
- összekapcsolás:	FM szintézis	

A hangszer neve: **XYLO2.SBI**

	Modulátor:	Carrier:
- karakterisztika:	00101110 ($2E)	00000010 ($02)
- alaphangerő:	00000000 ($00)	00000000 ($00)
- burkológörbe:		
	AD: $FF	AD: $F6
	SR: $0F	SR: $4A
- hullámalak:	00000000 ($00)	00000000 ($00)
- visszacsatolás:	7	
- összekapcsolás:	FM szintézis	

C függelék

Sound Blaster AWE 32 nem regisztrált MIDI paraméterek

A paraméter neve: **NRPN LSB 0** (*Delay before LFO1 starts*)
Típus: normál
Értéktartomány: 0...5900
Egység: 4 ms

A paraméter neve: **NRPN LSB 1** (*LFO1 frequency*)
Típus: valós idejű
Értéktartomány: 0...127
Egység: 0.084 Hz

A paraméter neve: **NRPN LSB 2** (*Delay before LFO2 starts*)
Típus: normál
Értéktartomány: 0...5900
Egység: 4 ms

A paraméter neve: **NRPN LSB 3** (*LFO2 frequency*)
Típus: valós idejű
Értéktartomány: 0...127
Egység: 0.084 Hz

A paraméter neve: **NRPN LSB 4** (*Envelope 1 delay time*)
Típus: normál
Értéktartomány: 0...5900
Egység: 4 ms

A paraméter neve: **NRPN LSB 5** (*Envelope 1 attack time*)
Típus: normál
Értéktartomány: 0...5940
Egység: 1 ms

A paraméter neve: **NRPN LSB 6** (*Envelope 1 hold time*)
Típus: normál
Értéktartomány: 0...8191
Egység: 1 ms

A paraméter neve: **NRPN LSB 7** (*Envelope 1 decay time*)
Típus: normál
Értéktartomány: 0...5940
Egység: 4 ms

A paraméter neve: **NRPN LSB 8** (*Envelope 1 sustain level*)
Típus: normál
Értéktartomány: 0...127
Egység: 0.075 dB

A paraméter neve: **NRPN LSB 9** (*Envelope 1 release time*)
Típus: normál
Értéktartomány: 0...5940
Egység: 4 ms

A paraméter neve: **NRPN LSB 10** (*Envelope 2 delay time*)
Típus: normál
Értéktartomány: 0...5900
Egység: 4 ms

A paraméter neve: **NRPN LSB 11** (*Envelope 2 attack time*)
Típus: normál
Értéktartomány: 0...5940
Egység: 1 ms

A paraméter neve: **NRPN LSB 12** (*Envelope 2 hold time*)
Típus: normál
Értéktartomány: 0...8191
Egység: 1 ms

A paraméter neve: **NRPN LSB 13** (*Envelope 2 decay time*)
Típus: normál
Értéktartomány: 0...5940
Egység: 4 ms

A paraméter neve: **NRPN LSB 14** (*Envelope 2 sustain level*)
Típus: normál
Értéktartomány: 0...127
Egység: 0.075 dB

A paraméter neve: **NRPN LSB 15** (*Envelope 2 release time*)
Típus: normál
Értéktartomány: 0...5940
Egység: 4 ms

A paraméter neve: **NRPN LSB 16** (*Inital pitch*)
Típus: valós idejű
Értéktartomány: –8192...8191
Egység: 1 cent

A paraméter neve: **NRPN LSB 17** (*LFO 1 to pitch*)
Típus: valós idejű
Értéktartomány: –127...127
Egység: 9.375 cent

A paraméter neve: **NRPN LSB 18** (*LFO 2 to pitch*)
Típus: valós idejű
Értéktartomány: –127...127
Egység: 9.375 cent

A paraméter neve: **NRPN LSB 19** (*Envelope 1 to pitch*)
Típus: normál
Értéktartomány: –127...127
Egység: 9.375 cent

A paraméter neve: **NRPN LSB 20** (*LFO 1 to volume*)
Típus: valós idejű
Értéktartomány: 0...127
Egység: 0.1875 dB

A paraméter neve: **NRPN LSB 21** (*Inital filter cutoff*)
Típus: valós idejű
Értéktartomány: 0...127
Egység: 62 Hz

A paraméter neve: **NRPN LSB 22** (*Inital filter resonance coefficient*)
Típus: normál
Értéktartomány: 0...127

A paraméter neve: **NRPN LSB 23** (*LFO 1 to filter cutoff*)
Típus: normál
Értéktartomány: –127...127
Egység: 56.25 cent

A paraméter neve: **NRPN LSB 25** (*Chorus effect send*)
Típus: normál
Értéktartomány: 0...255

A paraméter neve: **NRPN LSB 26** (*Reverb effect send*)
Típus: normál
Értéktartomány: 0...255

D függelék

A CT-VOICE.DRV meghajtóprogram funkciói

Funkció: **$0000** **A meghajtóprogram verziójának lekérdezése** (*Get driver version*)
Bemenőparaméterek: BX = $0000
Visszaadott értékek: AH = a főverziószám (*main version number*)
AL = az alverziószám (*sub-number*)
Megjegyzés: – –

Funkció: **$0001** **A hangkártya báziscímének beállítása** (*Set SB baseport address*)
Bemenőparaméterek: BX = $0001
AX = a port címe (*Base address*)
Visszaadott értékek: – –
Megjegyzés: Ezt a funkciót mindig a $0003-as funkció előtt kell meghívni.

Funkció: **$0002** **A hangkártya megszakításának beállítása** (*Set SB interrupt*)
Bemenőparaméterek: BX = $0002
AX = a megszakítás száma (*interrupt number*)
Visszaadott értékek: – –
Megjegyzés: Ezt a funkciót mindig a $0003-as funkció előtt kell meghívni.

Funkció: **$0003** **A meghajtóprogram inicializálása** (*Initalize driver*)
Bemenőparaméterek: BX = $0003
Visszaadott értékek: AX = $0000 az inicializálás sikeres volt (*successfull*)
AX = $0001 nincs hangkártya a gépben (*SB not found*)
AX = $0002 hibás báziscím (*port address error*)
AX = $0003 megszakítási hiba (*interrupt error*)
Megjegyzés: A funkció hívása előtt a báziscímet és a megszakítás számát be kell állítani a $0001-es és a $0002-es funkciókkal.

Funkció: **$0005** **Az állapotszó címének beállítása** (*Set statusword address*)
Bemenőparaméterek: BX = $0005
ES:DI = az állapotszó címe (*statusword address*)
Visszaadott értékek: – –
Megjegyzés: Ez a funkció az aktuális folyamat (lejátszás/felvétel) állapotáról tájékoztató állapotszó címét állítja be.

Funkció: **$0006** **Egy hangminta lejátszása** (*Sample palyback*)
Bemenőparaméterek: BX = $0006
ES:DI = a hangminta kezdőcíme (*sample address*)
Visszaadott értékek: – –
Megjegyzés: Az ES:DI mutatónak a VOC fájl legelső blokkjára kell mutatnia. Az állapotszó értéke a lejátszástól függően változhat.

Funkció: **$0007** **Egy hangminta felvétele** (*Record sample*)
Bemenőparaméterek: BX = $0007
AX = mintavételi frekvencia (*sampling rate*)
DX:CX = a felvétel hossza (*length*)
ES:DI = a hangminta kezdőcíme (*sample address*)
Visszaadott értékek: – –
Megjegyzés: Az állapotszó értéke a felvétel kezdetekor $0000-nál nagyobb értéket vesz fel, s ha a buffer megtelt, akkor ismét $0000 lesz.

Funkció: **$0008** **A felvétel vagy lejátszás leállítása** (*Abort sample*)
Bemenőparaméterek: BX = $0008
Visszaadott értékek: – –
Megjegyzés: Az állapotszó értéke $0000 lesz.

Funkció: **$0009** **A meghajtóprogram érvénytelenítése** (*De-install driver*)
Bemenőparaméterek: BX = $0009
Visszaadott értékek: – –
Megjegyzés: A kimenet kikapcsolt állapotba kerül, a hangkártya a bekapcsoláskor érvényes állapotot veszi fel.

Funkció: **$000A** **A lejátszás felfüggesztése** (*Pause sample*)
Bemenőparaméterek: BX = $000A
Visszaadott értékek: AX = $0000 sikeres volt (*successful*)
AX = $0001 nem volt sikeres (*not successful*)
Megjegyzés: A lejátszás leáll, de az állapotszó értéke nem áll $0000-ba.

Funkció: **$000B** **A lejátszás folytatása** (*Continue sample*)
Bemenőparaméterek: BX = $000B
Visszaadott értékek: AX = $0000 sikeres volt (*successful*)
AX = $0001 nem volt sikeres (*not successful*)
Megjegyzés: Ez a funkció a felfüggesztett lejátszást folytatja.

Funkció: **$000C** **Megszakítási hurok létrehozása** (*Interrupt loop*)
Bemenőparaméterek: BX = $000C
AX = $0000 csak a lejátszás végén (*at end of loop*)
AX = $0001 közvetlenül mindig (*immediately*)
Visszaadott értékek: AX = $0000 sikeres volt (*successful*)
AX = $0001 nem volt sikeres (*no loop being executed*)
Megjegyzés: A program a VOC fájlok blokkjainak lejátszásakor megszakítást tud adni.

Funkció: **$000D** **Felhasználói funkció installálása** (*User-defined function*)
Bemenőparaméterek: BX = $000D0
DX:AX = a felhasználói rutin címe (*user-defined funcion address*)
Visszaadott értékek: ES:BX = az aktuális blokk címe (*address of current data block*)
Megjegyzés: A felhasználói rutin minden blokk lejátszásának kezdetén automatikusan meghívásra kerül. A felhasználói rutinnak a következő feltételeket kell teljesítenie:
– minden regisztert tárolni kell;
– távoli hívásnak számít, ezért RETF-fel kell visszatérni belőle;
– a Cy jelzőbit vezérli a lejátszást:
Cy=0 lejátszás;
Cy=1 nincs lejátszás.

(A CT-VOICE.DRV programot egyszerűen be kell tölteni a memóriába egy $0000 ofszetrészű címre, és minden alkalommal meg kell hívni ezt a címet; így lehet aktiválni a funkciókat. A funkció számát a BX regiszter tartalmazza. Az egyes rutinok megváltoztathatják az AX és a DX értékét, a többi regiszter érintetlen marad.)

E függelék

Az SBFMDRV.COM meghajtóprogram funkciói (*CMF driver*)

Funkció: **$0000** **A meghajtóprogram verziójának lekérdezése** (*Get driver version*)
Bemenőparaméterek: BX = $0000
Visszaadott értékek: AH = a főverziószám (*main version number*)
AL = az alverziószám (*sub-number*)
Megjegyzés: – –

Funkció: **$0001** **Az állapotbájt címének beállítása** (*Set status byte address*)
Bemenőparaméterek: BX = $0001
DX:AX = az állapotbájt címe (*status byte address*)
Visszaadott értékek: – –
Megjegyzés: Az állapotbájt a lejátszás aktuális állapotáról tájékoztat.

Funkció: **$0002** **A hangszerek beállítása** (*Set instruments*)
Bemenőparaméterek: BX = $0002
CX = a hangszerek száma (*number of instruments*)
DX:AX = a hangszeradatok kezdőcíme (*start address of datas*)
Visszaadott értékek: – –
Megjegyzés: A CMF fájlban tárolt hangszerek letöltése.

Funkció: **$0003** **A rendszer óraütemének beállítása** (*Set system clock*)
Bemenőparaméterek: BX = $0003
AX = 1193189/rendszerütem
Visszaadott értékek: – –
Megjegyzés: Alapesetben az időzítő 18.2 Hz-es.

Funkció: **$0004** **A meghajtó óraütemének beállítása** (*Set driver clock*)
Bemenőparaméterek: BX = $0004
AX = 1193189/lejátszási ütem
Visszaadott értékek: – –
Megjegyzés: Alapesetben az időzítő 96 Hz-es.

Funkció: **$0005** **Az összes hang megváltoztatása** (*Transpose all notes*)
Bemenőparaméterek: BX = $0005
AX = az emelés vagy süllyesztés mértéke félhangokban
Visszaadott értékek: – –
Megjegyzés: A pozitív értékek emelést, a negatív értékek süllyesztést eredményeznek.

Funkció: **$0006** **A lejátszás elindítása** (*Start playback*)
Bemenőparaméterek: BX = $0006
DX:AX = a zeneadatok kezdőcíme (CMF)
Visszaadott értékek: AX = $0000 sikeres volt
AX = $0001 más zene lejátszása van folyamatban
Megjegyzés: – –

Funkció: **$0007** **A lejátszás leállítása** (*Stop playback*)
Bemenőparaméterek: BX = $0007
Visszaadott értékek: AX = $0000 sikeres volt
AX = $0001 nincs zene, amelyet le lehetne állítani
Megjegyzés: – –

Funkció: **$0008** **A meghajtóprogram inicializálása** (*Initalize driver*)
Bemenőparaméterek: BX = $0008
Visszaadott értékek: AX = $0000 sikeres volt
AX = $0001 nem lehet inicializálni, mert még tart a lejátszás
Megjegyzés: – –

Funkció: **$0009** **A lejátszás felfüggesztése** (*Pause playback*)
Bemenőparaméterek: BX = $0009
Visszaadott értékek: AX = $0000 sikeres volt
AX = $0001 nincs zene, amelyet fel lehetne függeszteni
Megjegyzés: – –

Funkció: **$000A** **A lejátszás folytatása** (*Continue playback*)
Bemenőparaméterek: BX = $000A
Visszaadott értékek: AX = $0000 sikeres volt
AX = $0001 nincs zene, amely le lett állítva
Megjegyzés: – –

Funkció: **$000B** **Felhasználói szubrutin installálása** (*Set user-defined function*)
Bemenőparaméterek: BX = $000B
DX:AX = a felhasználói rutin címe
Visszaadott értékek: – –
Megjegyzés: – –

F függelék

CSATLAKOZÓK

A JOYSTICK/MIDI csatlakozók kiosztása:

A kivezetés száma	A kivezetés funkciója
1	+5V tápfeszültség
2	az első JOYSTICK első tűzgombja
3	az első JOYSTICK jobb-bal bemenete
4	GND jelföld
5	GND jelföld
6	az első JOYSTICK fel-le bemenete
7	az első JOYSTICK második tűzgombja
8	+5V tápfeszültség
9	+5V tápfeszültség
10	a második JOYSTICK első tűzgombja
11	a második JOYSTICK jobb-bal bemenete
12	MIDI adatkimenet (TxD, *transmit data*)
13	a második JOYSTICK fel-le bemenete
14	a második JOYSTICK második tűzgombja
16	MIDI adatbemenet (RxD, *receive data*)

A *PANASONIC* CD-ROM csatlakozók kiosztása:

Kivezetés	Jel	Kivezetés	Jel
1	GND	2	–RESET*
3	GND	4	GND
5	GND	6	PO#0
7	GND	8	PO#1
9	GND	10	–IOR
11	GND	12	–IOW
13	GND	14	STAT#0
15	GND	16	
17	GND	18	
19	GND	20	STAT#1
21	GND	22	–SELECT
23	GND	24	STAT#2
25	GND	26	–DATAR
27	GND	28	STAT#3
29	GND	30	GND
31	D7	32	D6
33	GND	34	D5
35	D4	36	D3
37	GND	38	D2
39	D1	40	D0

* A – jellel kezdődő kivezetések alacsony szinten aktívak.

A *SONY* CD-ROM csatlakozók kiosztása:

Kivezetés	Jel	Kivezetés	Jel
1	–RESET*	2	GND
3	D7	4	GND
5	D6	6	GND
7	D5	8	GND
9	D4	10	GND
11	D3	12	GND
13	D2	14	GND
15	D1	16	GND
17	D0	18	GND
19	–IOR	20	GND
21	–IOW	22	GND
23	DACK	24	GND
25	DRQ	26	GND
27	–IRQ	28	GND
29	A1	30	GND
31	A0	32	GND
33	–SELECT	34	GND

* A – jellel kezdődő kivezetések alacsony szinten aktívak.

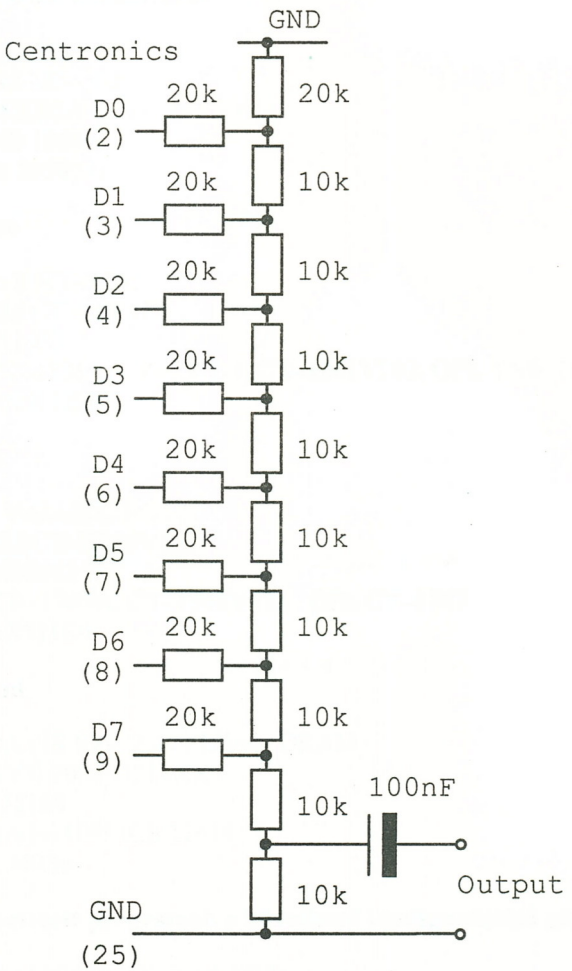
A *MITSUMI* CD-ROM csatlakozók kiosztása:

Kivezetés	Jel	Kivezetés	Jel
1	A0	2	GND
3	A1	4	GND
5		6	GND
7		8	GND
9		10	GND
11		12	GND
13		14	GND
15	DRQ	16	GND
17	DACK	18	GND
19	–IOW*	20	GND
21	–IOR	22	GND
23	–SELECT	24	GND
25	D0	26	GND
27	D1	28	GND
29	D2	30	GND
31	D3	32	GND
33	D4	34	GND
35	D5	36	GND
37	D6	38	GND
39	D7	40	GND

* A – jellel kezdődő kivezetések alacsony szinten aktívak.

G függelék

A PÁRHUZAMOS PORTRA ÉPÍTHETŐ D/A KONVERTER KAPCSOLÁSA



Technikai adatok

A könyv írásakor felhasznált hangkártyák műszaki azonosító adatai:

Sound Blaster/ADLIB kompatíbilis

Media Concept MF–002
FFC ID: J98 MF–002
No.: 152003A
Chip: MS 1006
Card: M 2059781

Sound Blaster Pro

Sound Blaster Pro 2 CT–1600
FFC ID: IBACT–SBP2P5
No.: 811863
Chip: CT–1336A, CT–1345-3, CT–1341V203, OPL YMF 262–M
Card: CN511X0

Sound Blaster 16

Sound Blaster 16 VALUE CT–2770
FFC ID: IBACT–SB16VAL
No.: a492442
Chip: CT– 1745A, CT–1741V413, OPL CT–1747
Card: AZ521E0

Gravis Ultrasound

ADVANCED GRAVIS Rev. 2.4 (1 Mbyte DRAM)
FFC ID: HYN 700–0021–XX
No.: 223169
Chip: Gravis GF1 ICS 11614
Card: K34980

A könyvben ismertetett programok a következő konfigurációjú gépen készültek:

CPU: Intel 80486 DX–2, 66 MHz
Memory: 16 Mbyte (4x4 SIMM)
Harddisks: 546 Mbyte, 270 Mbyte
Floppy disks: 1.2 Mbyte 5 1/4, 1.44 Mbyte 3 1/2
Video: Trident 8900D 1M true-color (*ISA BUS*)
CD-ROM: Panasonic CR–562 B (*double speed, multisession*)

Irodalomjegyzék

Örley Gábor:

Elektronikus szintetizátorok, Műszaki Könyvkiadó, 1979.

Norbert Hesselmann:

Digitális jelfeldolgozás, Műszaki Könyvkiadó, 1985.

Axel Stolz:

The Sound Blaster BOOK, Data Becker, 1992.

Gerényi Gábor — Sík Zoltán:

MIDI alapozás, MIDI protokoll, Pixel Graphics, 1992.

Angster Erzsébet — Kertész László:

Turbo Pascal 6.0, Angster—Kertész, 1992.

Abonyi Zsolt:

PC hardver kézikönyv, ComputerBooks, 1993; bővített kiadás: 1995.

Dr. Kovács Magda:

32 bites mikroprocesszorok 80386/80486 I-II, LSI, 1991,1994.

Ultrasound Lowlevel Toolkit, Advanced Gravis, 1994

Ultrasound user's guide, Advanced Gravis, 1994

Sound Blaster development kit, Creative Technology Ltd, 1991

Sound Blaster AWE 32 Developer's Information Pack, Creative Technology Ltd, 1994

Sound Blaster user' guide, Creative Technology Ltd, 1992

Sound Blaster Pro user's guide, Creative Technology Ltd, 1994

Sound Blaster 16 user's guide, Creative Technology Ltd, 1994

Tárgymutató

A

ADC, 22, 142
additív szintézis, 46, 52, 253
ADLIB, 43, 101, 145, 208, 323
ADLIB unit, 54, 175, 256
AdlibBase, 149, 175
ADPCM, 104, 275
ADSR, 14, 50, 58, 98, 169, 202, 252
Advanced Gravis Ultrasound, 210
Advanced Sound Processor, 178, 198
AGC, 166, 196
akkord, 30, 48
alapharmonikus, 10
aliasing, 22
aluláteresztő szűrő, 24
AM, 12, 48
AM Depth, 53
AMIGA, 281
amplitúdó, 7, 8, 14, 17, 19, 25, 43, 58
amplitúdómoduláció, 12, 206, 287
amplitúdóváltozás, 15
analóg, 7, 18
analóg jel, 18, 19, 20
analóg-digitális átalakítás, 22
arpeggio, 285
asm, 33, 37
ASP, 178, 198
Assembler, 33, 37
Assembler fordító, 33
assembly, 33
assembly blokk, 33
AT, 124
attack, 14, 50, 252
audio CD, 18
auto-init DMA, 183
Automatic Gain Control, 166, 184

Á

állandó, 35
állapotregiszter, 44
árnyéktár, 43, 55
átalakító, 18
átmeneti folyamat, 7
átmeneti jelenségek, 13, 14, 16

B

balansz, 222
BaseAddr, 107
Bass Drum, 52
Bass level, 191
basszus, 29
báziscím, 57, 101, 146, 166, 205
bináris kód, 19, 24
bitcsoport, 55
built-in Assembler, 33
burkológörbe, 14, 15, 43, 48, 50, 61, 202, 206
burkológörbe-generátor, 15, 202

C

Call, 70
carrier, 45
CD-ROM, 146, 155, 166, 199, 349
ChNum, 71, 81
ciklikus lejátszás, 219, 245
cimbalom, 16
címke, 35, 36
cli, 47
CMF, 260, 347
Compact Disk, 18
compiler, 85
const, 38
csatorna, 43, 60
csillapítás, 49

D

DAC, 24, 143, 217, 280
DAHDSR, 202, 203
decay, 14, 17, 50, 252
dekompresszió, 102, 198, 275
determinisztikus, 7
digitális csatorna, 205
digitális hangkeltés, 18
digitális hangprocesszor, 102
digitális hangrögzítés, 18
digitális jel, 18, 19
digitális jelfeldolgozás, 18
digitális-analóg átalakítás, 24
digitalizálás, 20
dinamikus adat, 301
dinamikus adathossz, 301
Direct Memory Access, 125
direktívák, 34
disszonáns, 30
diszkrét, 19, 24, 25
diszkrét értékű jel, 19
Djnz, 71
DMA, 101, 104, 124, 146, 153, 160, 166, 178, 196, 205, 216, 227, 281
DMA vezérlő, 125
DMAchannel, 107
dob, 16
DoubleOperator, 56
DSP, 102, 131, 143, 153, 160, 166
DSP params, 105
DSP unit, 107
DSP.PAS unit, 133
DSPread, 110
DSPreset, 109

DSPwrite, 110
dúr, 30

E

effektus, 48, 51, 315
egységugrás, 15
elemi esemény, 66
eljárás, 35, 37
EMU 8000, 197, 200
end, 33, 37, 70
ErrorExit, 88, 96
esemény, 66
exkluzív rendszerüzenet, 305

É

értéktartomány, 20

F

fázis, 10, 25
Feedback, 148
felfutási idő, 14
félhang, 9
felharmonikus, 10
feszültség, 19, 22
FETCH, 76
finetune, 291
Fixed Gain Control, 166
fizikai címzés, 120
fizikai csatorna, 68, 77, 258
flash A/D converter, 23
FLT4, 284
FM, 13, 59
FM chip, 43, 147, 175, 261
FM szintézis, 46, 52, 253, 281
FM voice, 43, 48
FMDRV, 266
Fourier, 10, 25
Fourier-analízis, 25
frekvencia, 8, 13, 43, 50, 59
frekvenciamoduláció, 13
frekvenciaszorzó, 48, 58
FreqMultiple, 58
furulya, 12
függvény, 35, 37
fűrészfogjel, 11, 174

G

GATE, 48, 66, 74, 82
GENERAL MIDI, 312
getmem, 119, 121
GetMixer, 159, 191
GF1, 205, 208, 209, 213, 218, 226
gitár, 12, 13, 30
glisszandó, 290
globális változó, 34
Gravis UltraMAX, 210
Gravis Ultrasound, 205, 326
GUS DRAM, 205, 208, 230, 313
GUS unit, 232

H

hang, 8, 24, 29, 246, 251
hangerő, 8, 12, 49, 57, 98, 143, 222, 230, 288
hanggenerátor, 45
hangmagasság, 9
hangminta, 104
hangskála, 9
hangsor, 51, 68
hangszer, 30, 252, 267, 299, 315
hangszín, 13
hangszintetizálás, 18
hardvermegszakítás, 130
harmonikus, 26, 48
harmonikus rezgés, 7
háromszögjel, 11
heap, 163, 247
heapkezelés, 119
Hi-Hat, 52
High DMA channel, 178
hullám, 10, 14
hullámalak, 10, 15, 45, 46, 174, 253, 291
hullámforma, 53, 59, 61, 224
hullámtábla, 199, 219

I

időállandó, 142
időegység, 65
időfüggvény, 19
időtartomány, 7
időzítés, 44, 142
időzítő, 47
implicit, 77
indexregiszter, 120
inicializálás, 57
integrátor, 22
Intel, 119, 283
Intel 8237A, 124
Intel 8259A, 130
interpoláció, 229
IRET, 83
IRQ, 125, 205, 226, 250, 262
IRQ vezérlő, 130, 136
IRQnumber, 107

J

jel, 7, 13, 14, 15, 18, 24, 57
jelprocesszor, 198, 228
jelszint, 49
Jmp, 70

K

kettes komplemens aritmetika, 248
keverő, 150, 184
KEY ON, 51, 62, 77, 173

kitartási idő, 14
kitartási szint, 15
kódgenerálás, 33
komment, 33
komparátor, 23
komponens, 28
kompresszió, 198
konszonáns, 30
konverter, 24
kotta, 29, 251
körfrekvencia, 25
közvetlen memória-hozzáférés, 125
KSR, 48
kvantálás, 18, 19
kvantálási hiba, 20
kvantálási zaj, 20

L

laphatár, 129
lapregiszter, 126, 213, 223
lecsengési idő, 14, 15
Line In, 150
logikai csatorna, 75
lokális változó, 34, 37
Low DMA channel, 178
LSB, 24

M

M/S-mode, 180
maxavail, 121
megszakítás, 60, 65, 101, 125, 146, 226
memóriafoglalás, 119
MID fájl, 300, 302, 306
MIDI, 102, 166, 189, 198, 205, 212, 261, 300, 327, 339, 349
MIDI csatornaüzenetek, 302
MIDI esemény, 302
MIDI metaesemény, 309
mikrofon, 150
mintavételezés, 18, 20
mintavételezési frekvencia, 21, 142, 165, 275
mintavételezési tétel, 21
mintavételi frekvencia, 102, 209
Mixer, 150, 163, 166, 184, 199
MOD, 281, 293, 313
moduláció, 58
modulator, 45
moll, 30
mono, 145, 154, 166, 280
Motorola, 282
MPU401, 166
MSB, 24
MTrk, 308

N

négyoperátoros hanggenerálás, 150, 169, 253
négyszögjel, 11, 28, 174
nemdeterminisztikus, 7
new, 119
nextline, 89
nextnum, 89
nextword, 88
normál A, 51
normál A hang, 10
note, 59, 66, 91, 281, 284, 314
nyomáshullám, 8

O

offszet, 40, 120
oktáv, 9, 49, 51, 54, 59, 246
operátor, 37, 45, 54
OPL–3, 43, 148, 166, 168
OPL3 unit, 176
orgona, 12, 17

Ö

összekapcsolási mód, 172
összetett hullám, 13

P

PageMaker, 133
paraméter, 38, 39
Pascal, 33
patch, 206
pattern, 284, 288, 297, 314
Pause, 71
periodikus, 26
periodikus függvény, 10
periodikus időfüggvény, 25
periódus, 8
periódustábla, 285, 318
pipeline, 209, 218
Play, 83
PlayChannel, 78, 83
player, 72, 85
PlayNext, 76, 80, 81
portamento, 286
posztinkrementális, 79
pozitív élvezérelt, 51
pozitív élvezérlés, 63
predekrementális, 79
protected mód, 119, 124

R

real mód, 119
referencia, 22
referenciabájt, 105
regiszter, 33
RegSet, 70, 75, 76, 80, 81
release, 15, 17, 50, 253
repeater, 68, 71, 80, 98
ResetADLIB, 57
ResetMixer, 191
result, 41
Ret, 70

rezgés, 25
rezonancia, 202
rhytm control, 52, 63
RIFF, 278
ritmus, 29, 52
ROLAND GS MIDI, 312
Rpt, 71

S

SB_DMA unit, 130, 140, 161
SB_IRQ, 134
SB16MIX unit, 189
SBI, 251, 329
SBPRO unit, 156
scale, 49
SetAM, 58
SetDriverClock, 270
SetFM, 59
SetMixer, 159, 191
SetSystemClock, 270
SetWave, 59
shadow RAM, 43
Shannon tétele, 21
SIMM modul, 201
single slope converter, 22
single-cycle DMA, 182
skála, 65
skálatényező, 48
snare, 65
Snare Drum, 52
sorba fejtés, 25
Sound, 59
Sound Blaster, 101, 145, 273, 324
Sound Blaster 16, 165, 325
Sound Blaster AWE 32, 197, 325, 339
Sound Blaster Pro, 145, 325
Soundtracker, 282
spaceskip, 88
spektrum, 14, 15, 22, 24, 26
StartPlay, 133
Startracker, 282
StartRec, 133
status register, 44
StereoOutput, 160, 163
sustain, 14, 48, 50, 253
szegmens, 40, 120, 140
szegmensregiszter, 34, 36, 124
szelektor, 124
szimbólum, 35
szintaktika, 33
szinusz, 13
szinuszhullám, 111, 244
szinuszos rezgés, 10
sztereó, 145, 148, 154, 161, 166, 172, 205, 278
szünet, 67
szűrő, 24, 202, 290

T

T-mode, 180
temperált hangskála, 9
tempó, 311
TestADLIB, 57
timer, 47, 75
típus, 35
típus nélküli paraméter, 39
Tom-Tom, 52
Top-Cymbal, 52
töréspont frekvencia, 202
tranziens, 13
Treble level, 191
tremoló, 12, 202, 287
trombita, 15

U

UART chip, 167, 208

V

valószínűségszámítás, 7
váltakozó feszültség, 8
változó, 35
var, 38
védett üzemmód, 119
verem, 75, 79
VGA, 26
VIB, 48
vibráció, 13, 59, 287
vibrátó, 13
Vibrato Depth, 53
vibrátor, 13, 48
violin, 29
visszacsatolás, 52, 61, 173, 253
VOC, 273
VOC blokk, 273
Voice Sound Synthesizer, 209
Volume, 57
Vstc, 153

W

WAV, 278
Waveblaster, 199
waveform, 46, 53
Windows, 278
WriteReg, 55

X

XT, 124

Y

YM-3812, 43, 147, 170

Z

zene, 29, 251
zenefájl, 251
zongora, 12, 17, 30

Megrendelőlap

Megrendelem az alábbi kiadványokat postai utánvéttel. Tudomásul veszem, hogy a postaköltség felszámításra kerül, és a szállítási idő 2–3 hét.

Utánnyomásoknál árváltozás lehetséges.

... *pl Füzi János:* **3 dimenziós grafika és animáció IBM PC-n** – lemezmelléklettel	**1.283.–**
... *pl Jakab Zs. Juhász Gy. Vémi J.;* **Adobe PHOTOSHOP**	**2.480.–**
... *pl Dr.Kovács T.–Ozsváth M.:* **Adatkezelés az MS ACCESS 2.0 alkalmazásával**	**1.890.–**
... *pl Nagy G.:* **Kézikönyv az adattömörítéshez – ARJ. PKZIP & Co.** – lemezmelléklettel	**1.298.–**
... *pl Pintér M.:* **AutoCAD parancsok és változók** – Release 13 – angol & magyar – DOS & WINDOWS & UNIX	**1.176.–**
... *pl Pintér M.:* **AutoCAD R13 szerkesztési újdonságok**	**599.–**
... *pl Pintér M.:* **AutoCAD tankönyv** – DOS & WINDOWS; AutoCAD LT; AutoCAD R12 angol & magyar	**899.–**
... *pl Benkő–Poppe–Benkő*: **Bevezetés a BORLAND C++ programozásába**	**975.–**
... *pl Benkő L.–Benkő T.né–Tóth*: **Programozzunk C nyelven kezdőknek * középhaladóknak** – lemezmelléklettel	**1.371.–**
... *pl Dr.Dedinszky F.:* **CA-VISUAL OBJECTS**	**1.559.–**
... *pl Dr.Dedinszky F.:* **CLIPPER 5 – 5.0, 5.01 és segédprogramjai**	**899.–**
... *pl Nagy Z.–Spányik B.–Weisz T.:* **CorelDRAW! 5**	**895.–**
... *pl Gazsó Z.:* **Adatbáziskezelés dBASE 5.0 for Windows rendszerben** – lemezmelléklettel	ir.ár.: **1.800.–**
... *pl Juhász–Kiss–Kuzmina–Sólétormos–Dr.Tamás–Tóth:* **DELPHI – út a jövőbe** – lemezmelléklettel	**1.999.–**
... *pl Tamás–Kiss–Tóth:* **MS-DOS 6 – 6.2; 6.22 kiegészítéssel**	**1.200.–**
... *pl Kovalcsik G.:* **EXCEL for Windows 5.0 kezdőknek * haladóknak** – magyar és angol változathoz	**1.147.–**
... *pl Dr. Kovácsné C. J. – Ozsváth M.:* **EXCEL 5 függvényei** – magyar változathoz	**990.–**
... *pl Gazsó Z.:* **FoxPRO 2.5, 2.6** – Windows/DOS – lemezmelléklettel	**1.475.–**
... *pl Nádai P.–Rezessy B.–né:* **Visual FoxPro 3.0** (PROXERV Kft kiadványa)	**1.860.–**
... *pl Abonyi Zs.*: **PC hardver kézikönyv** (bővített, átdolgozás kiadás)	**975.–**
... *pl Stolnicki Gyula:* **Hálózatokról kezdő felhasználóknak**	**1.369.–**

A 175-35-91 telefonszámon könyvszolgálatunk tájékoztatja Önt a lakó- vagy munkahelyéhez legközelebb eső szaküzletről, ahol kiadványainkat megvásárolhatja.

Ha mégis a postai utat választja, kérjük levélcímre visszaküldeni

COMPUTERBOOKS Kft – 1253 Bp., Pf.: 71 .

... *pl László J.:* **Hangkártya programozása Pascal és Assembly nyelven** – lemezmelléklettel	**1.568.–**
... *pl Székely V.:* **Áramkőrszimuláció PC–n** – lemezmellékleten a TRANZ–TRAN szimulációs program	**1.990.–**
... *pl Lengyel Veronika:* **Az INTERNET világa**	**1.456.–**
... *pl Székely V.:* **Képkorrekció, hanganalízis, térszámítás PC–n** – lemezmelléklettel	**1.258.–**
... *pl Benkő–L.–Benkő T.né:* **MS WORKS 3.0 – a mindennapi életben** – magyar verzióhoz	**793.–**
... *pl Rudnai P.né:* **Novell NetWare 3.11, 3.12 felhasználóknak és rendszergazdáknak**	**945.–**
... *pl Tóth Dezső:* **OS/2 Warp felhasználói ismeretek**	**1.680.–**
... *pl Benkő–Tóth–Varga*: **Programozzunk TURBO PASCAL nyelven** – lemezmelléklettel (javított, átdolgozott kiadás)	**796.–**
... *pl Benkő T.né–Kiss Z.–Tóth B:.* **Objektum–orientált programozás Turbo Pascal 6.0–ban és a Turbo VISION** – lemezmelléklettel	**979.–**
... *pl Benkő T.né–Kiss Z.–Tamás P.–Tóth B.*: **Programozás Borland Pascal 7.0 rendszerben /DPMI, WINDOWS** – példaprogramok lemezmellékleten	**1.586.–**
... *pl Kovácsné C. J.–Pergelné B. I.–Benkő L.*: **Mindenkinek! a PC–ről**	**699.–**
....*pl Gerő Judit:* **PowerPoint 4**	**1.426.–**
... *pl dr. Kovácsné C. J.–Ozsváth M.:* **QuarkXPress for Windows**	**979.–**
... *pl Borgulya I.*: **Szakértői rendszerek, technikák és alkalmazások**	**1.375.–**
... *pl Stolnicki Gy.:* **SQL Kézikönyv** – bővített, átdolgozott kiadás – lemezmelléklettel	**1.499.–**
... *pl László József:* **VGA kártya programozása** Pascal és Assembly nyelven – lemezmelléklettel	**1.375.–**
... *pl Nagy Gábor:* **Vírusvédelem a PC–n** – lemezmelléklettel	**1.157.–**
... *pl dr.Tamás–Horváth–Kiss–Tóth*: **WINDOWS 3.1 felhasználóknak**	**698.–**
... *pl dr.Kovácsné Cohner Judit:* **Magyar WINDOWS 3.1**	**990.–**
... *pl Benkő T.né–Kuzmina J.–Kiss Z.–dr.Tamás P.–Tóth B.:* **Könnyű a WINDOWS–t programozni!?** – lemezmelléklettel	**1.683.–**
... *pl Tóth B.–Tamás P és trsai:* **WINDOWS 95 & Microsoft Plus felhasználóknak**	**1.995.–**
... *pl Tóth B.–Tamás P és trsai:* **WINDOWS 95 & Microsoft Plus felhasználóknak – magyarnyelvű változathoz**	**1.960.–**
....*pl Dr.Kovácsné C. J.–Ozsváth M.:* **Windows for Workgroups 3.11** – hálózattal vagy anélkül	**1.115.–**
... *pl Gerő Judit–Reich Gábor*: **WORD for WINDOWS 6.0** – magyar & angol nyelvű verzióhoz	**980.–**
... *pl Gerő Judit–Krizsák László:* **WORD for WINDOWS 6.0 kis☺kos**	**498.–**

Megrendelő neve: _____

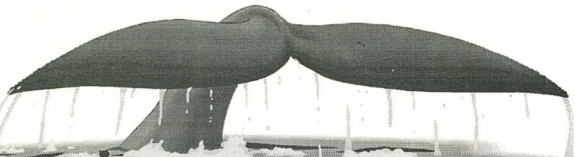
szállítási cím: _____

város: _____

utca: _____

irányítószám: _____

1996/04



Tengernyi SZOFTVER és CD hegyek!

szoftver ABC

10 pack - 5ft vol. 2 cd
1st dessign
act! for windows
adobe acrobat exchange
adobe art sampler cd win-mac
adobe illustrator for windows
adobe pagemaker
adobe premiere for windows
adobe type manager for wp.
adventures cd
autocad lt
bbs mate cd
berlitz think & talk german cd
berlitz think & talk french cd
best of entertainment cd
best of pc/windows sharew. cd
bioforge cd
blinker
boris valejo cd
borland c++
borland knowledge base appl.
brief dos & os/2
btrieve for dos/win./ os/2
ca c++ for os/2
ca clipper
ca dbfast for windows
ca exospace for clipper
ca realizer
ca supercalc for windows
ca superproject for windows
ca tools iii
ca uptodate for windows
cd game pack cd
cd rom workshop
calendar creator
chirstmas for windows cd
close up
co/session host
co/session remote
codebase ++
codeprint pro for vb
conan cd
corel cd power pack
corel gallery
corel network manager
corel professional photos
corel ventura cd
coreldraw
corelflow
dbase 5
dge
director multimedia studio win.
dr. games for windows cd
doom companion cd
dragon's lair cd
education cd-pack for win. 1
education cd-pack for win. 2
enable
enterprise developer
eye of the beholder
ékszer
fastback plus dos
faxpress
flipper
framework iv
goblins 3 cd
graphicon
great naval battles
group wise
gupta sql
harvard graphics for win.
helyes-e?/ms+
helyette/win
hsc digital morph
internet express
it kontír 2000 plusz
it napló 2000 professzionál
it print master for windows
it számla 2000
jungle book: mowgli cd
just grandma and me cd
kai's power tools
lanpress
laplink pro
learn to speak english cd
linux
linux bible
lord of rings cd
lotus 1-2-3 ee
lotus 1-2-3 for win. magyar
lotus ami pro magyar
lotus notes client for win.
lotus notes for netrware start.
mayo clinic: the total heart
micrografx abc flowcharter
ms access
ms beethoven's 9th
ms cinemania for windows
ms creative writer
ms dangerous creatures win.
ms delta
ms dos
ms electronic forms designer
ms encarta for windows
ms excel for windows magyar
ms fine artist
ms flight simulator
ms foxpro for win. prof.
ms macro assembler
ms office for win. prof.
ms strauss
ms stravinsky
ms test win
ms ultimate robot
ms visual basic for win. prof.
ms win. for workgr. add on m.
ms win. for workgr. magyar
ms windows nt
ms word for win. magyar
ms works for windows magyar
netlib
netware 4.1 5 user
netware 4.1 50 user
north american indians cd
norton commander
norton pcanywhere for win.
norton utilities
novell dos
nyelvmester angol kezdő
nyelvmester angol haladó
nyelvmester német kezdő
ocular
our solar system cd
paradox for windows
pic-dic képes szótár francia
pic-dic képes szótár olasz
pic-dic képes szótár angol cd
pic-dic képes szótár német cd
pirates realmspace
pressworks
procomm plus
prof. music producer cd
prof. office for windows
q&a for windows
q&a write for windows
quattro pro dos/win.
r&r report writer for dos/win.
rebel assoult cd
recognita go-cr for win.
renegade cd
return to ring world cd
sea and sky cd
six driver
space and astronomy cd
space quest iv cd
space series: apollo cd
spt-gib hangos szótár cd
spt-gib országh nagyszótár cd
spt-gib műszaki szótár cd
star wars chess cd
stereogram workshop cd
stronghold
syndicate cd
take ten 10 pak cd
time line for windows
tripleplay english cd
tripleplay french cd
tripleplay plus englsh cd
tripleplay plus french cd
turbo c++ for windows
turbo pascal
turbo pascal for win.
ultima i-vi
ultimate shareware cd
unlimited adv.
u.s. aircraft carriers
u.s. military helicopters
u.s. submarine
win platinum cd
windows master iv cd
windows mate platinum cd
winfax pro
wing commander iii cd
wolfstein 3d cd
wordperfect for widows
wordperfect magyar
yoga! a multimedia guide cd

Budapest XIII. ker. Jászai Mari tér 3.

Telefon: 269-4737, 269-4738
Fax: 269-4720, 201-8619
Levél cím: 1391 Budapest Pf: 218
E-mail: 100324.661@ compuserve.com

Ára: 1.568.–Ft Áfával

ISBN 963-618-091-1