

第5章

MSX-MUSIC



この章は、MSX マガジン 1990 年 7 月号から 1990 年 10 月号までの “MSX2+テクニカル探検隊” の記事を再編集したものである。

5.1 FM 音源ってどんなもの

MSX-MUSIC という名称で仕様が定まったFM 音源。ゲームの効果音を迫力あるものにしてくれるのは知っているけど、どんな仕組みになっているのか？ このページではその謎に迫ってみる。

5.1.1 FM 音源へと至る電子楽器の歴史

FM 音源を解説する前に、電子楽器の歴史を振り返ってみよう。

ボグ・ムーグ博士は、電圧で音階を制御できる発振器と、電圧で音色を制御できるフィルターを組み合わせて、“ムーグ式シンセサイザー” という楽器を作った。1968 年にはこれを使用した最初のレコードが発表され、1970 年代になると多くの音楽家がシンセサイザーを使うようになった。ただこのシンセサイザーは、最近主流となっている “デジタルシンセサイザー” とは違っていて、トランジスタなどのアナログ回路の組み合わせで作られたもの。デジタルに対して “アナログシンセサイザー” とも呼ばれている。

でも、このアナログシンセサイザーには、いくつかの欠点があった。それが、温度変化に弱い、高価である、雑音が入りやすい、ということ。筆者も 1970 年代に秋葉原で IC を買って、シンセサイザーを自作したけど、調整が難しかったことが印象に残っている。

さて、そんな欠点を克服するために開発されたのが、デジタル回路による電子楽器。もっとも単純なデジタル音源は、“プログラマブル・サウンド・ジェネレーター”、略して “PSG” だ。これは、4 個程度のデジタル発振器の出力を、デジタル・アナログ (D/A) コンバーターで、オーディオ信号に変えて出力する LSI。価格が安く、使いやすいこともあって、MSX などの多くのパソコンに組み込まれている。

PSG よりも複雑な音を作る方法のひとつに、“サンプリング音源” がある。これは、ほかの楽器の音をマイクで受け取って、A/D(アナログ・デジタル) コンバーターでデジタル信号に変え、メモリーに記憶し、D/A コンバーターでアナログ信号に戻して再生するもの。これを応用した楽器が、“サンプリングシンセサイザー” というわけだ。音を作る自由度は高いけど、大量のメモリーを必要とするなど、ハードウェアが高価になることが欠点といえる。

さて、PSG の安さと、サンプリング音源の自在さを合わせ持った音源として注目されるのが、FM 音源だ。FM とは、FM 放送やモデムの FM 信号と同じ、“周波数

表 5.1: 電子楽器の性能を比較する

	アナログシンセ	PSG	サンプリングシンセ	FM 音源
ハードウェア	複雑	LSI	LSI+大容量メモリー	LSI
安定性	温度変化に弱い	安定	安定	安定
音色	多彩	貧弱	万能	多彩
データ量		小さい	莫大	小さい
価格	高い	安い	高い	並

変調”という意味。図 5.1 のように、1 個のデジタル発振器の出力が、もう 1 個のデジタル発振器の周波数を変調して、PSG よりも複雑な音を作り出す。1 個のデジタル発振器を“オペレーター”ともいい、図 5.1 のように 2 個の発振器を含む FM 音源を、“2 オペレーター式 FM 音源”という。ちなみに“MSX-MUSIC”は、正式名称を“OPLL YM2413”といい、9 組の 2 オペレーター式 FM 音源を内蔵する LSI だ。

図 5.1: 4 種類の電子楽器の構造を探る

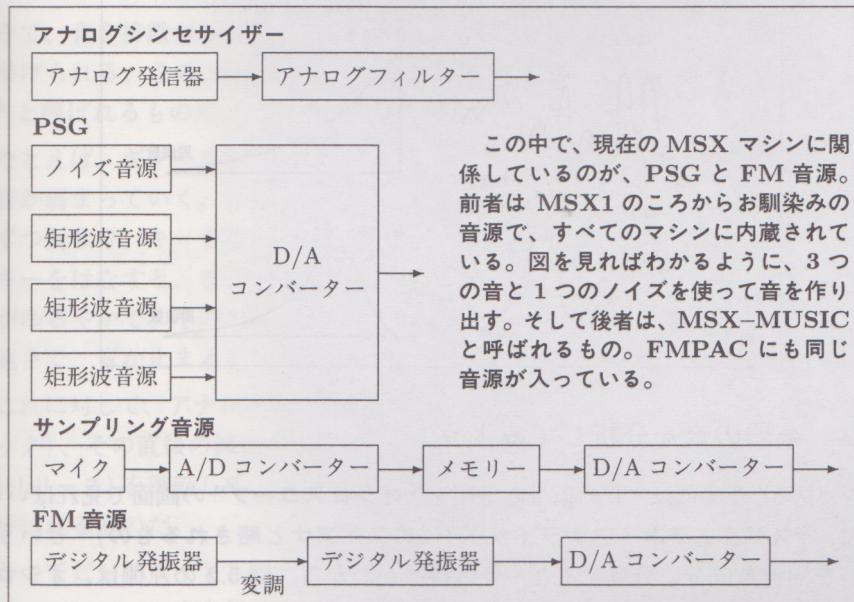
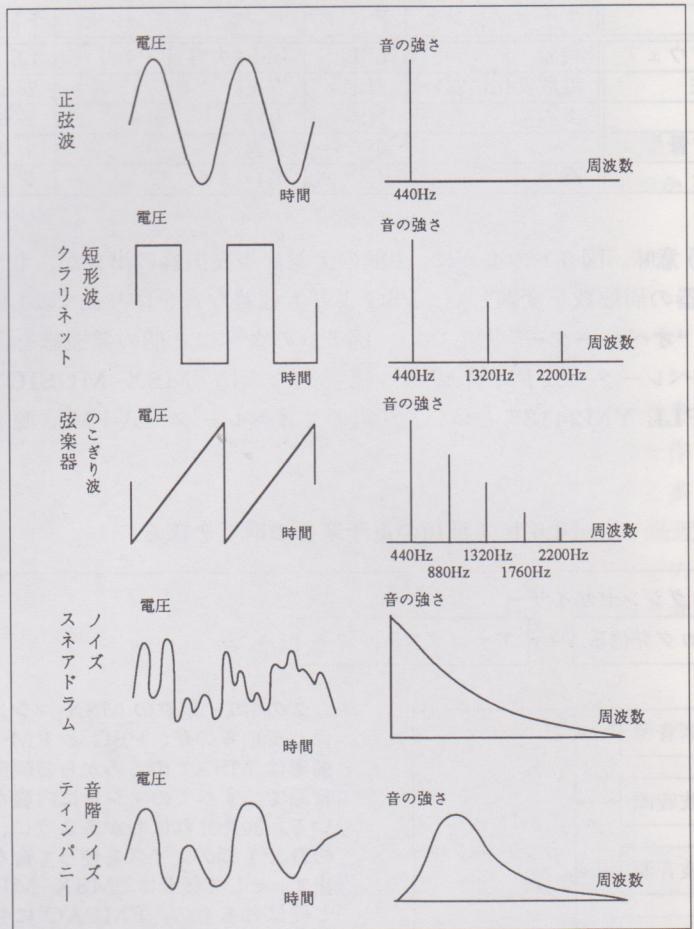


図 5.2: 基本となる音を分析してみる



5.1.2 楽器の音を分析してみよう

音を“見る”ためには、音の信号の電圧を“オシロスコープ”の画面で見ればいい。さらに、“スペクトラム・アナライザー（俗にスペアナと略されるもの）”という装置で音を周波数成分に分解すると、音の特徴がわかる。図 5.2 の左側は、オシロスコープで見た楽器音の波形の特徴を誇張した図で、右側がスペクトラム・アナライザーで測った周波数成分だ。

もっとも基本的な音は、波形が三角関数の \sin で表わされる“正弦波”と呼ばれるもの。これはひとつの周波数の音のみを含む。次は波形が四角い“矩形（くけい）波”で、440Hz、1320Hz、2200Hz ……のように、基本周波数とその奇数倍の周波

数の音を含むものだ。実際の楽器では、クラリネットの音がこの矩形波に近い。次に基本的なのは“のこぎり波”。これは、基本周波数とその倍数の周波数の音を含んでいて、弦楽器の音の性質に似たものだ。アナログシンセサイザーは、のこぎり波を加工して、実際の楽器に似た音を作っている。さて、打楽器、とくにスネアドラムの音は、ほかの楽器とは大きく違っている。規則性がなく、どちらかといえば“ノイズ(雑音)”に近いものだ。スペクトラム・アナライザーで見ると、広い範囲の周波数の音を含んでいる。

ティンパニーの音は、弦楽器と打楽器の中間の性質で、基本周波数とその近くの周波数の音を含んだもの。“音階ノイズ”とも呼ばれている。また、この音階ノイズを加工することで、風、波、口笛などの音も合成できる。体育の授業などで先生が吹く笛や、素人が吹く管楽器の音も、音階ノイズだ。

これらの楽器の音、つまり図5.2のような波形の代わりに、FM音源は変調によって正弦波を歪ませて、基本周波数とその倍数の周波数を含む複雑な波形を作り出す。これは難解な技術で、試行錯誤で数値を調整して楽器の音を真似るしかない。そこで、FM音源にはいくつかの楽器音を合成するためのプログラムが内蔵され、これらの音から選んで使うことが、一般的になっている。

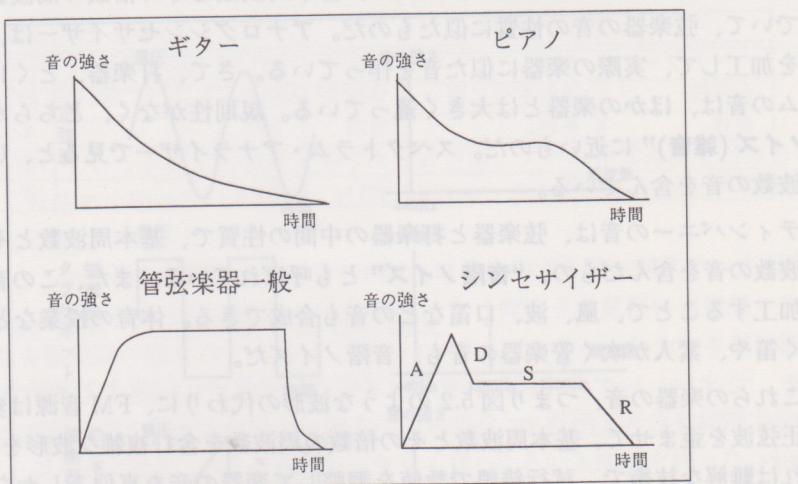
さて、音を特徴づける要素としては、基本となる波形のほかに、音の強弱の変化も挙げられる。この“強弱”は、基本の波形を“包む”という意味で、“エンベロープ”と呼ばれるものだ。

たとえば、ギターや打楽器を鳴らすと、その瞬間に強い音が出て、あとは少しづつ音が弱まっていく。ピアノのキーを押すと、はじめは大きな音が出て、それが少しづつ弱まり、キーを押している間はそのままほぼ一定の大きさの音が続く。そしてキーをはなすと、音が弱くなっていく。また一般の管弦楽器では、音の立ち上がりがゆるやかで、そのあと同じ大きさの音が続き、最後は立ち上がりと同じくらいの速さで、音が止まるといった具合だ(図5.3参照)。

これに対して、アナログシンセサイザーとFM音源では、立ち上がりの速さ(A:アタック)、その直後の減衰(D:ディケイ)、持続の強さ(S:サスティン)、消える速さ(R:リリース)を調節して、エンベロープを合成する。そのための装置が、“ADSR”と呼ばれるものだ。

かつてピンフロイドというロックグループは、“吹けよ風、呼べよ嵐”という曲の中で、シンバルの音を録音したテープを逆に回すことで、少しづつ大きくなつて急に止まる効果音を作り出した。でもシンセサイザーを使えば、アタックを遅く、ディケイを速く、サスティンを0にするだけで、このような音を合成できる。また、エンベロープの合成は、FM音源の波形の合成よりも簡単なので、自分でADSRを調整して効果音を作り出すのも、おもしろいだろう。

図 5.3: 楽器とシンセのエンベロープ



5.1.3 音程が平均律とは限らない

突然だけど、ここでちょっとクラシック音楽の話をする。まずは、音階と周波数の対応関係から整理すると、表 5.2 のようになる。

表 5.2: 音階と周波数の関係

A	440.0Hz
A#	466.2Hz
B	493.9Hz
C	523.3Hz
C#	554.4Hz
D	587.3Hz
D#	622.3Hz
E	659.3Hz
F	698.5Hz
F#	740.0Hz
G	784.0Hz
G#	830.6Hz
a	880.0Hz

このなかで、A の音の周波数は 440Hz、その 1 オクターブ上の a の音の周波数は 880Hz。つまり 1 オクターブ離れた音の周波数は、2 倍になっているわけだ。また、半音離れた音の周波数の比は約 1.0595 倍で、12 半音(1 オクターブ)離れた音の周波数の比は、 1.0595^{12} 、つまり 2 になる。

このように、すべての音階の周波数が等比数列で表わされる規則を、“完全平均律”という。この音律は、バロック時代から古典派時代にかけて成立したものらしい。筆者が高校生のころには、“バッハが平均律を作った”と教えられたけど、最近の研究では、バッハよりもあとに平均律ができたという説が有力だそうだ。現代の音楽の大部分は、この平均律に基づいて演奏されている。

さて、この完全平均律ができる以前には、周波数の比が有理数になるような音律が使われていた。これは平均律と異なり、半音の周波数比が場所によって違うもの。たとえば C と C# の比と、B と C の比が違うという特徴があるわけだ。平均律よりも規則が複雑なこともあって、表 5.3 のような各種の音律が設定されている。ただし平均律以外の音律では、和音の響きは美しいけれど、曲の移調が難しいという問題

表 5.3: MSX-Music で設定できる音律一覧

番号	設定される音律	番号	設定される音律
0	ピタゴラス	11	純正律 cis メジャー (bマイナー)
1	ミーントーン	12	純正律 d メジャー (hマイナー)
2	ヴェルクマイスター	13	純正律 es メジャー (cマイナー)
3	ヴェルクマイスター (修正 1)	14	純正律 e メジャー (cisマイナー)
4	ヴェルクマイスター (修正 2)	15	純正律 f メジャー (dマイナー)
5	キルンベルガー	16	純正律 fis メジャー (esマイナー)
6	キルンベルガー (修正)	17	純正律 g メジャー (eマイナー)
7	ヴァロッティ・ヤング	18	純正律 gis メジャー (fマイナー)
8	ラモー	19	純正律 a メジャー (fisマイナー)
9	完全平均律 (初期設定)	20	純正律 b メジャー (gマイナー)
10	純正律 c メジャー (aマイナー)	21	純正律 h メジャー (gisマイナー)

もあるんだ。

バイオリンのように、連続して周波数を変えられる楽器では、どのような音律にも対応できる。でも、ピアノの音律を変えるには、全部の弦をチューニングし直す必要があり、実用的には音律を変えられない。ところが MSX に搭載された FM 音源では、表 5.3 のように音律を選ぶことができる。これにより、純正律のギターのような、簡単には実現できない楽器の音も作れるわけだ。この特徴を利用し、FM 音源のレジスターを操作して音律を微調整すれば、雅楽や琉球音楽なども、精密に演奏することができるかもしれない。

なお、FM 音源の周波数は音律が問題になるほど正確だけど、PSG の周波数はそれほどでもない。だから、PSG を楽器の調律や発声練習の基準に使うのは危険だ。とはいっても、筆者は音感がニブイので、音律の違いがよくわからない。FM 音源を使いこなすには、数学、電気、音楽理論の知識に加え、正確な音感とセンスが必要になるけど、そんな人はめったにいない。ゲーム音楽を作るにも、作曲家と音色デザイナーとプログラマーが組んで働くように、役割分担が必要だろう。

5.1.4 MSX-MUSIC を分析してみる

MSX-MUSIC には、あらかじめ 63 種類の音色が用意されている。このうちの 15 種類は FM 音源の LSI に内蔵された音色で、残りの 48 種類は ROM に記録された音色だ。ROM に記録された音色のデータは、

CALL VOICE COPY

という命令で呼び出すことができる。リスト 5.1 に掲載したのが、このデータを表示するためのプログラムだ。なお、FM 音源に内蔵された音色番号を指定するとエ

ラーが起き、リスト 5.1 のプログラムの場合は、

Voice No. * has no data.

といったメッセージが表示されるようになっている。

リスト 5.1 (READFM.BAS)

```

100 ' read VOICE DATA of MSX-MUSIC
110 ' by nao-i on 20. Apr. 1990
120 '
130 CALL MUSIC : DEFINT A-Z
140 DIM VI(15),VD(31),VO(3)
150 PRINT "Voice Number (0,...,63; 64 for all; 65 for end) ";
160 INPUT ME
170 IF 0 <= ME AND ME <= 63 THEN VN=ME : GOSUB 200 : GOTO 150
180 IF ME = 64 THEN FOR VN=0 TO 63 : GOSUB 200 : NEXT VN : GOTO 150
190 END
200 '
210 ON ERROR GOTO 460
220 CALL VOICE COPY(@VN,VI)
230 ON ERROR GOTO 0
240 FOR I=0 TO 15
250   VD(I*2)=VI(I) AND 255
260   VD(I*2+1)=(VI(I) / 256) AND 255
270 NEXT I
280 NA$=""
290 FOR I=0 TO 8
300   IF VD(I) THEN NA$=NA$+CHR$(VD(I))
310 NEXT I
320 PRINT : PRINT "Voice No.";VN;" : " NA$
330 PRINT "Transpose=";VI(4);
340 PRINT " Feedback=";(VD(10) AND 14) / 2
350 FOR I=0 TO 3: VO(I)=VD(I+16): NEXT I
360 PRINT "Operator 0" : GOSUB 490
370 FOR I=0 TO 3: VO(I)=VD(I+24): NEXT I
380 PRINT "Operator 1" : GOSUB 490
390 CALL BGM(0)
400 CALL VOICE(@VN,@VN,@VN)
410 PLAY #2,"CED<G>CR","V6EGF<B>ER","V4GBADGR"
420 CALL VOICE(@O,@O,@O)
430 CALL BGM(1)
440 ON ERROR GOTO 0
450 RETURN
460 '*** error
470 PRINT "Voice No.";VN;" has no datum."
480 RESUME 440
490 '*** print data of an operator
500 PRINT " AM =";(VO(0) ¥ 128) AND 1;
510 PRINT " PM =";(VO(0) ¥ 64) AND 1;
520 PRINT " EG =";(VO(0) ¥ 32) AND 1;
530 PRINT " KSR =";(VO(0) ¥ 16) AND 1;

```

```

540 PRINT " MULT=";VO(0) AND 15;
550 PRINT " LKS=";(VO(1) ¥ 64) AND 3;
560 PRINT " TL=";VO(1) AND 63
570 PRINT " ADSR=";(VO(2) ¥ 16) AND 15;" ,"; VO(2) AND 15;" ,";
580 PRINT (VO(3) ¥ 16) AND 15;" ,"; VO(3) AND 15
590 RETURN
600 ON ERROR GOTO 0

```

プログラム中で操作しているオペレーターは2種類。オペレーター1は、基本の波形を作るための“キャリアー・オペレーター”で、オペレーター2が、1を変調するための“モジュレーター・オペレーター”だ。それぞれ設定している数値の意味を解説すると、それだけで1冊の本になってしまふので、ここでは省略。参考書などを使って、各自で調べてほしい。そうそう、リスト5.1の“PRINT”という部分を“LPRINT”に変更して、音色データの表を印字しておけば、自分で音色を設計するための参考資料として便利かもしれない。

5.1.5 FM 音源を使ってリズム音に挑戦

MSX-MUSICに限らず、FM音源が苦手とする音は打楽器音だ。実際の打楽器や、アナログシンセサイザーが作り出す打楽器音は不規則なノイズだけど、FM音源の打楽器音は規則性がありすぎることが災いして、“安っぽい”あるいは“機械的な”音となってしまう。

そこでMSX-MUSICには、63種類の楽器音とはべつに、“リズム音”を発生するための機能が用意されている。そもそも63種類の楽器音の中には、打楽器の音色も含まれているのだけれど、これらは楽器音と同じ方法で合成される音色。ここでいうリズム音とは、あくまでも別物だ。

マニュアルなどにも書かれているように、MSX-MUSICにはチャンネル1から9までの、9組の2オペレーター式FM音源が内蔵されている。その全部を楽器音として使えば、9声の演奏が可能なわけだ。

ところが、リズム音を作り出すための方法として、チャンネル1から6までを普通の楽器音に割り当て、チャンネル7から9までの3組分の6オペレーターを、リズム音として使うことも可能になっている。そのため、MSX-MUSICの機能を表わすのに、“9楽器音または6楽器音+1打楽器音”という表現が使われるわけだ。

BASICからこれらの機能を利用するには、“CALL MUSIC”命令のパラメーターを変更すればいい。たとえば、

```
CALL MUSIC(0,0,1,1,1,1,1,1,1,1)
```

で9楽器音が。

```
CALL MUSIC(1,0,1,1,1,1,1,1,1)
```

で6 楽器音+1 打楽器音が選択される。

参考までに次に掲載したプログラムは、MSX-MUSICの音色データと、自分で作り出したリズム音の違いを聞きわけるためにもの。はじめに、楽器音の音色31番の2オペレーターによる“Bass Drum”を4回鳴らしたあと、リズム音の6オペレーターによるバスドラム音を4回鳴らす。リズム音のほうが本物のドラムに似ていることを、実際に打ち込んで、自分の耳で確認してみよう。

リスト 5.2 (BASSDRUM.BAS)

```

10 CALL MUSIC (1,0,1,1,1,3)
20 CALL BGM(0)
30 CALL VOICE(@31)
40 PLAY #2,"V15CCCC","","","","","RRRRB!4B!4B!4B!4"
50 CALL VOICE(@0)

```

また、MSXではFM音源とPSGを同時に鳴らせるので、FM音源で楽器音を出し、PSGで打楽器音と効果音を出すことも可能だ。しかし、MSX本体の機種によって、FM音源の音の大きさとPSGの音の大きさのバランスが違っているので、それぞれのマシンに応じて音量を調整するためのプログラムが、必要になってくる。

```

270 NEXT 1
280 '出前手 "音△入り" にておもな音器楽の順序
290 '各音器楽にて、おもな音器楽の順序
300 'IP VOL=100, VCO1=100, VCO2=100
310 '音序二。音音るけち複合ウタ表示同様音器樂台されこ。音音り出でるべく付を含
320 PRINT : PRINT "Voice No.":VN;"":N&H"
330 PRINT "Transpose":VI(4);
340 GOSUB 400 BASIC-KBM400を見るバブル表示を並べてエニマ
350 END 100, VOL=100, VCO1=100, VCO2=100
360 '音器樂手完全のナーフ。おもな音器樂 MT方一マークで示す
370 FOR I=0 TO 3: VO(I)=0(I+24): NEXT I: 12716を掛けて表す
380 PRINT "Operator": I: GOSUB 400
390 '音器樂手完全のナーフ。おもな音器樂手出でる音多音入り一組ごと
400 GOSUB 400 BASIC-KBM400を見るバブル表示を並べてエニマ
410 'AY 0 TO 100, VOL=100, VCO1=100, VCO2=100
420 '音器樂手完全のナーフ。おもな音器樂手出でる音多音入り一組ごと
430 CALL MUSIC(0) '音器樂手完全のナーフ。おもな音器樂手出でる音多音入り一組ごと
440 OF BGM(0)
450 'BASICで音器樂手完全のナーフ。CALL MUSIC(0)を用いて音器樂手完全のナーフ。
460 '*** ERROR
470 PRINT "Voice No.":VN;" has no datum."
480 RESUME 440
490 '*** print data of an operator (1,1,1,1,1,1,1,1,1,0,0)
500 PRINT " AM =":(VO(0) % 128) AND 1;
510 PRINT " PM =":(VO(0) % 64) AND 1;
520 PRINT " BG =":(VO(0) % 32) AND 1;
530 PRINT " RR =":(VO(0) % 16) AND 1;
540 GOSUB MUSIC(1,0,1,1,1,1,1,1,1,0,0)

```

5.2 FM 音源をコントロール

いまやゲームのBGMや効果音には欠かせない存在となったFM音源。このページでは、マシン語プログラムからFM音源をコントロールすることに、挑戦してみよう。

5.2.1 マシン語プログラムで音を出してみる

前のページでも紹介したように、拡張 BASIC を使えば、簡単に FM 音源を操作することができた。でも、ゲームの効果音や BGM として応用するには、マシン語プログラムが直接“FM-BIOS”を呼び出して、FM 音源を操作する必要がある。

ここからは、MSX-C で作られたプログラムが FM 音源を操作するための、ライブラリーとプログラム例を紹介しようと思う。当然のことながら、このプログラムをコンパイルして実行可能なマシン語ファイルにするためには、“MSX-DOS TOOLS (または DOS2 TOOLS)”と、“MSX-C ver.1.1 (または ver.1.2)”が、それぞれ必要になってくる。

さて、リスト 5.3 は、MSX-C で作られたテストプログラムの“TESTFM.C”。 “fmdata”という配列がテストデータで、バスドラムとスネアドラムを叩きながら、“ドレミファソラシド”を 4 回演奏させるものだ。このデータの作り方は、アスキーネット MSX にある、msx.spec のボードで公開された資料にも書かれている。

リスト 5.3 (TESTFM.C)

```
/*
 *      testfm.c
 *      by nao-i on 29. May. 1990
 *      (C) Isikawa 1990
 *      free to use and copy, but no guarantee or support
 */
#include <stdio.h>
#include "fmlib.h"
#pragma nonrec

#define TESTLENGTH 20
#define TESTTIMES 4

static char fmdata[] = { /* test data */ 
    14, 0, /* 0: offset to rythme */
    33, 0, /* 2: offset to ch 1 */
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 4: offset to ch 2...6*/
    FM_RVOL + 31, 8, /* 14: rhythm VOL = 8 */
    0x30, TESTLENGTH, /* 16: B drum */
    0x28, TESTLENGTH, /* 18: S drum */
};
```

```

0x28, TESTLENGTH,          /* 20: S drum */
0x28, TESTLENGTH,          /* 22: S drum */
0x30, TESTLENGTH,          /* 24: B drum */
0x28, TESTLENGTH,          /* 26: S drum */
0x28, TESTLENGTH,          /* 28: S drum */
0x28, TESTLENGTH,          /* 30: S drum */
FM_END,                   /* 32: end of rhythm */

FM_VOL + 8,                /* 33: Ch. 1 VOL = 8 */
FM_INST + 3,                /* 34: Guiter */

FM_SUSON,
FM_LEGOFF,
FM_Q, 6,
FM_04 + FM_C, TESTLENGTH,
FM_04 + FM_D, TESTLENGTH,
FM_04 + FM_E, TESTLENGTH,
FM_04 + FM_F, TESTLENGTH,
FM_04 + FM_G, TESTLENGTH,
FM_04 + FM_A, TESTLENGTH,
FM_04 + FM_B, TESTLENGTH,
FM_05 + FM_C, TESTLENGTH,
FM_END                   /* end of Ch. 1 */
};

VOID main(argc, argv)
int argc;
char **argv;
{
    auto char fmwork[FMWORK]; /* address must be >= 8000H */
    auto char fmbuf[256];      /* address must be >= 8000H */
    char fmstat;

    if ((fmstat = fmopen(fmwork)) == 1) {
        puts("No FM-BIOS.");
        exit(1);
    } else if (fmstat == 2) {
        puts("Bad address.");
        exit(1);
    }
    printf("fmopen : address of work area is %04X\n",
           (unsigned)fmwork);
    memcp(fmbuf, fmdata, sizeof(fmdata));
    fmstart(fmbuf, (char)TESTTIMES);
    do {
        fputs("Playing.\$015", stdout);
    } while (fmtest());
    fputs("\$nEnd of play.\$n", stdout);
    fmstop();
    fmclose();
    fputs("fmclose : complete\$n", stdout);
    exit(0);
}

```

それでは、プログラムを簡単に説明していこう。まずは、大きさが“FMWORK”バイトの auto 配列である“fmwork”を用意する。そして、その番地をわたして、ライブラリーの“fmopen”を呼び出す。この配列は 8000H 以上の番地に置かれる必要があるので、static ではなく auto と宣言しよう。

以上の手続きにより、FM 音源の準備に成功すれば 0 が、FM 音源がなければ 1 が、“fmwork”の番地が 7FFFH 以下ならば 2 が、それぞれ“fmopen”から返されるはずだ。

このとき注意しなければいけないのは、FM 音源を使う前にかならず“fmopen”を呼び出し、プログラムが終了する前に“fmclose”を呼び出す必要があるということ。もしも“fmclose”が呼ばれる前に、プログラムが終了してしまうと都合が悪いので、このライブラリーでは [CTRL]+[C] キーや、 $\text{[CTRL}+\text{[STOP]}$ キーが押されても、無視するようになっている。

もしも FM 音源を使ったプログラムを自作しようなんてときは、 $\text{[CTRL}+\text{[C]}$ キーや、ディスクエラーに対する処理をきちんとすることが大切だ。どんな場合でも、“fmclose”を呼び出してから、終了させるように注意しよう。

さて、データが入った番地と、演奏回数のパラメーターをわたして“fmstart”を呼び出すると、すぐに FM-BIOS が演奏をはじめる。この FM-BIOS は、タイマー割り込みで動くようになっているので、演奏を続けながらもプログラムを先に進めることも可能だ。このプログラムでは、演奏しながら画面に“Playing.”と表示させるようにしてみた。

“fmtest”は、演奏中ならば 1 を、演奏が終わっていれば 0 を返す。また“fmstop”は、演奏を終了させて、FM-BIOS を初期化するためのもの。

5.2.2 ライブラリーの概要を説明する

リスト 5.4 に掲載したのは、FM 音源ライブラリーの関数と定数を定義するための、ヘッダーファイル“FMLIB.H”。

リスト 5.4 (FMLIB.H)

```
/*
 *      fmlib.h : header file for fmlib
 *      by nao-i on 31. May. 1990
 * by nao-i on 24. Feb. 1991  FM_01 changed from 0 to 1
 *      (C) Isikawa 1990
 *      free to use and copy, but no guarantee or support
 */
extern char   fmopen();           /* please call this first      */
extern VOID   fmclose();          /* please call this last       */
```

```

extern VOID fmwrite();      /* write to OPLL register */
extern VOID fmotir();      /* write to OPLL register 0...7 */
extern VOID fmstart();     /* back ground music */
extern VOID fmstop();      /* stop back ground music */
extern char *fmread();     /* read data from ROM */
extern char fmtest();      /* now playing ? */
#define FMWORK (0x00a0+32)  /* size of work area */

#define FM_VOL        0x0060 /* volume 60H...6FH */
#define FM_INST       0x0070 /* instulment 70H...7FH */
#define FM_SUSOFF    0x0080 /* sustain off */
#define FM_SUSON     0x0081 /* sustain on */
#define FM_EXPINST   0x0082 /* expandet instulment */
#define FM_USRINST   0x0083 /* user-defined instulment */
#define FM_LEGOFF    0x0084 /* legato off */
#define FM_LEGON     0x0085 /* legato on */
#define FM_Q          0x0086 /* Q */
#define FM_END        0x00ff /* end of data */
#define FM_RVOL      0x00a0 /* volume of rhythm */

/* pitch */
#define FM_C          0      /* C      */
#define FM_CS         1      /* C#    */
#define FM_D          2      /* D      */
#define FM_DS         3      /* D#    */
#define FM_E          4      /* E      */
#define FM_F          5      /* F      */
#define FM_FS         5      /* F#    */
#define FM_G          7      /* G      */
#define FM_GS         8      /* G#    */
#define FM_A          9      /* A      */
#define FM_AS         10     /* A#    */
#define FM_B          11     /* B      */
/* octave */
#define FM_01         1      /* FM_01+FM_C means C of octove 0 */
#define FM_02         13     /*          */
#define FM_03         25     /*          */
#define FM_04         37     /*          */
#define FM_05         49     /*          */
#define FM_06         61     /*          */
#define FM_07         73     /*          */
#define FM_08         85     /*          */

```

そして、次の長大なリスト 5.5 が、FM 音源ライブラリーだ。リストのはじめから順番に、BIOS などの番地の定義、FM-BIOS を呼び出すマクロの定義、ライブラリーが使用するワークエリアの定義、そしてライブラリーのプログラム本体が書かれている。

リスト 5.5 (FMLIB.Z80)

```

;
;      fmlib.z80 : library for MSX-C
;      by nao-i on 29. May. 1990
;      (C) ASCII 1988 for 'search', (C) Isikawa 1990
;      free to use and copy, but no guarantee or support
;
;      .Z80
;
;      address of BIOS and system work area
;
rdslt    equ     000ch
calslt   equ     001ch
enaslt   equ     0024h
breakv   equ     0f325h      ; ^C break vector
ramad0   equ     0f341h      ; slot # of RAM
ramad1   equ     0f342h
ramad2   equ     0f343h
ramad3   equ     0f344h
exptbl   equ     0fcc1h
h.timi   equ     0fd9fh      ; timer interrupt hook
;
;      address of FM-BIOS jump table
;
idstrg   equ     4018h+4
_wrtopl  equ     4110h
_iniopl  equ     4113h
_mstart  equ     4116h
_mstop   equ     4119h
_rddata  equ     411ch
_opldrv  equ     411fh
_tstbgm  equ     4122h
;
;      MACROs to call FM-BIOS
;
CALLFM  MACRO  ADDRESS
    ld      ix,address
    ld      iy,(biosslot-1)
    call   calslt
    ENDM
;
JUMPFM  MACRO  ADDRESS
    ld      ix,address
    ld      iy,(biosslot-1)
    jp      calslt
    ENDM
;
dseg
biosslot: ds   1          ; slot of FM-BIOS
breaks:   ds   2          ; saving ^C vector
p.ontime: ds   2          ; address of interrupt handler
p.oldhook:ds  2          ; address of saved hook
;
```

```

cseg
;
ontime:
    push    af
    ld      ix,_opldrv
    ld      iy,0          ; will be modified
ontime.slot equ $-1
    call    calslt
    pop    af
oldhook:
    nop
    nop
    nop
    nop
    nop
    ret
sizeof_ontime equ $-ontime
IF      sizeof_ontime GT 31
    .PRINTX "ontime routine too big"
ENDIF
;
hooktbl:
    rst    30h
    db     0          ; will be modified
    dw     0          ; will be modified
toret:
    ret
vvv:
    dw     toret        ; to ignore ^C
;
;      char   fmopen(address)
;      char   *address; /* address of work area */
;
;      0 : successful
;      1 : no FM-BIOS
;      2 : bad address of work area
;
fmopen@:
    ld     a,2
    bit   7,h
    ret   z          ; address of work area < 8000H
;
    ld     (p.ontime),hl
    push  hl
    ex    de,hl
    ld    hl,ontime
    ld    bc,sizeof_ontime
    ldir   ; trasfer ontine routine
    pop   hl
    push  hl
    ld    de,oldhook-ontime
    add   hl,de
    ld    (p.oldhook),hl
    pop   hl
    ld    de,32

```

```

search add    hl,de
res   0,1          ; make sure address is even
search
push  hl
call  search
ld    a,(biosslot)
ld    hl,(p.ontime)
ld    de,ontime.slot-ontime
add   hl,de
ld    (hl),a        ; modify LD IY,??00
or    a
ld    a,1
pop   hl          ; address of work area
ret   z            ; no FM-BIOS
CALLFM _iniopl
ld    h,40h
ld    a,(ramad1)    ; because FM-BIOS
call  enaslt      ; does not restore slot1
;
ld    hl,h.timi
ld    de,(p.oldhook)
ld    bc,5
ldir  ; save h.timi
ld    hl,hooktbl
ld    de,h.timi
ld    bc,5
di
ldir  ; set h.timi
ld    a,(ramad2)
ld    (h.timi+1),a
ld    hl,(p.ontime)
ld    (h.timi+2),hl
;
ld    hl,(breakv)
ld    (breaks),hl    ; save break vector
ld    hl,yyy
ld    (breakv),hl    ; set break vector
ei
xor  a
ret
;
;     VOID     fmclose()
;
fmclose@:::
di
ld    hl,(breaks)
ld    (breakv),hl    ; restore break vector
ld    hl,breaks
ld    hl,(p.oldhook)
ld    de,h.timi
ld    bc,5
ldir  ; restore h.timi
ei
ret
;

```

```

;      VOID    fmwrite(RegNum, Datum)          sh, fd   bbs
;      char   RegNum;           /* OPLL register number    set */
;      char   Datum;            /* datum to write          */
;
fmwrite@::             Id      daaq
                      JUMPFM _wrtopl
;
;      void    fmotir(aData)          sh, fd   bf
;      char  aData[8];              sh, fd   bbs
;
fmotir@::             a      zo
ld      b,8             l,a   bf
xor     a               fd   bqq
di
fmotir_loop:           di      xer
ld      e,(hl)          fd,gd   bf
inc     hl               fd,gd   bf
CALLFM _wrtopl
inc     a               fd,gd   bf
djnz   fmotir_loop
ret
;
;      void    fmstart(pDatum, Times)
;      char  *pData;           /* pointer to Music data bi */
;      char  Times;           /* times to play          bi */
;
fmstart@::             bi
ld      a,e             iihl
inc     a               bi
ret    z                make sure that Times != 255 bi
dec     a               bi
bit    7,h              bi
ret    z                make sure that pData >= 8000H bi
JUMPFM _mstart
;
;      VOID    fmstop()
;
fmstop@::              id      ioh
JUMPFM _mstop
;
;      char   *fmread(ptr, num)        8000H
;      char   *ptr;                  :00000000  VOID
;      char   num;                 :00000000
;
fmread@::              ib
ld      a,e             ib
JUMPFM _rddata
;
;      char   fmtest()           fd,fd   bf
;
fmtest@::              fd,fd   bf
JUMPFM _tstbgm
;
; Search FM-BIOS
;

```

```

search:
    ld      b,4
search_id:
    push   bc          ;save counter
    ld     a,4
    sub   b           ;make primary slot number
    ld     c,a        ;save it
    ld     hl,exptbl  ;point expand table
    ld     e,a
    ld     d,0
    add   hl,de
    ld     a,(hl)     ;get the slot is expanded or not
    add   a,a         ;expanded ?
    jr    nc,no_expanded ;no..
    ld     b,4         ;number of expanded slots
search_exp:
    push   bc          ;save it
    ld     a,24h       ;[a]=00100100b
    sub   b           ;make secondary slot # A=001000ss
    rrlca
    rrlca
    or    c           ;make slot address A=1000sspp
    call  chkids      ;check id string
    pop   bc          ;restore counter
    jr    z,search_found ;exit this loop if found
    djnz search_exp
not_found:
    xor   a
    ld    (biosslot),a
    pop   bc
    djnz search_id
    ret
no_expanded:
    ld    a,c          ;get slot address
    call chkids        ;check id string
    jr    nz,not_found ;exit this loop is found
search_found:
    pop   bc
    ret
;
id_string:
    db    'OPLL'
id_string_len equ    $-id_string
;
; Check ID string
; Entry : [A]=slot address to check
; Return: Zero flag is set if ID is found
; Modify: [AF],[DE],[HL]
;
chkids:
    ld    (biosslot),a
    push  bc          ;save environment
    ld    hl,idstrg
    ld    de,id_string

```

```

ld      b,id_string_len
chkids_loop:
push    af          ; save slot address
push    bc          ; save counter
push    de          ; save pointer to string
call    rdslt      ; read a byte
ei
pop     de          ; leave critical
pop     bc          ; restore pointer
pop     bc          ; restore counter
ld      c,a        ; save data
ld      a,(de)     ; get character
cp      c           ; same ?
jr      nz,differ ; no..
pop     af          ; restore slot address
inc    de          ; point next
inc    hl
djnz   chkids_loop
pop     bc          ; restore environment
xor    a           ; found set zero flag
ret
;
differ:
pop    af          ; restore slot address
pop    bc          ; restore environment
xor    a           ; clear zero flag
inc    a
ret
;
end

```

プログラムのポイントを解説すると、まず“fmopen@”は、“ontime”からのタイマー割り込み処理プログラムをべつの番地に転送し、FM-BIOS が置かれているスロットを探し、初期化し、タイマー割り込みフックを設定するためのもの。割り込み処理プログラムと、それが参照するデータは、8000H 番地以上に置かれる必要があるので、プログラムを転送して、“ld iy, 0”という部分を、“ld iy, FM-BIOS のスロット番号*256”に書き替えている。

FM-BIOS が置かれているスロットを探すプログラム自体は、アスキーネット MSX に公開されている、FM-BIOS の仕様書から引用した。全部のスロットについて、401CH 番地に“OPLL”という文字列があるかを調べることで、FM-BIOS を探している。

さて、“fmopen@”にわたされた 192 バイトのワークエリアは、割り込み処理プログラム、“h.tim”の元の内容の保存場所、FM-BIOS のワークエリア(160 バイト)に使われる。このとき FM-BIOS のワークエリアは、偶数番地からはじまる必要があるので、余分にワークエリアを用意し開始番地を切り上げている。

これは仕様書に書かれてなかったのだけど、“CALLFM _iniopl”でFM-BIOSを初期化すると、ページ1がべつのスロットに切り替えられたまま戻ってくることがあった。かならず、ページ1を元のスロットに戻す必要がある。

前にも書いたように、タイマー割り込みフックを書き替えたままプログラムが終了すると困るので、MSX-DOSのワークエリアのF325H番地を書き替えて、**[CTRL]+[C]**キーを無視するようにした。もしプログラムがディスクを使うならば、ディスクエラーを処理するプログラムを追加する必要がある。そして“fmclose@”は、タイマー割り込みフックと**[CTRL]+[C]**キーの処理を元に戻すためのものだ。

ライブラリーの残りの部分については、レジスターに必要な値を入れて、FM-BIOSを呼び出すだけで使用することができる。各自でいろいろ試してみよう。

5.2.3 MSX-C でコンパイルしよう

MSX-DOSのMEDやKIDなどのエディターでリストを打ち込んだら、リスト5.3を“TESTFM.C”、リスト5.4を“FMLIB.H”、そしてリスト5.5を“FMLIB.Z80”というファイル名でセーブしよう。次の手順でコンパイルすると“TESTFM.COM”ができるはずだ。プログラムを実行するには、MSX-DOSのコマンドラインから“TESTFM **[←]**”と入力すればいい。

リスト 5.6 (FMLIB.BAT)

```
m80 ,=fmlib.z80/r/m/z
cf testfm
cg testfm
m80 ,=testfm.mac/r/m/z
180 testfm,fmlib,ck,clib/s,crun/s,cend,testfm/n/y/e:xmain
```

5.3 FM 音源のデータ構造だ

このページでは、FM 音源のデータ構造と、実際に音源データを指定する方法を説明する。前に説明したマシン語で FM 音源を操作するプログラムと合わせて利用しよう。

5.3.1 FM 音源のデータを作ってみよう

前のページでは、FM-BIOS をマシン語から呼び出して、音を出すためのプログラム例を紹介した。ここでは、そのプログラムを使って演奏するため、FM 音源のデータの作り方を説明しよう。

FM-BIOS のデータ構造を要約するなら、大きな配列ということになる。配列の中にあるそれぞれのデータ位置は、配列の先頭からのバイト数で数えられ、“オフセット”と呼ばれている。ただし、配列の先頭のオフセットは 0 だ。また、配列の先頭は“0 バイト目”、その次は“1 バイト目”というように、呼ばれることもある。

表 5.4 に掲載したのは、6 楽器音+1 打楽器音のデータ構造と、実際のデータ例を示したもの。これを見てもらえばわかるように、データ本体は配列の末尾の方に並べられ、配列の先頭の 14 バイトには、それぞれのデータが置かれるオフセットが書き込まれている。

表 5.4: 6 楽器音+1 打楽器音のデータ構造

オフセット	内容
0	打楽器音データのオフセット (注 1)
2	楽器音 1 データのオフセット (注 2)
4	楽器音 2 データのオフセット (注 2)
6	楽器音 3 データのオフセット (注 2)
8	楽器音 4 データのオフセット (注 2)
10	楽器音 5 データのオフセット (注 2)
12	楽器音 6 データのオフセット (注 2)
14	打楽器音データ
(注 3)	楽器音 1 データ
(注 3)	楽器音 6 データ

(注 1) からず 0EH、00H の 2 バイトを書き込む。

(注 2) 楽器音データの開始位置の、この表のデータの先頭からのバイト単位のオフセットを、それぞれ下位バイト、上位バイトの順に指定する。また、使わないチャンネルに対しては 0 を指定する。

(注 3) オフセットで指定された場所に楽器音データが置かれる。

順番にデータ構造を説明していくと、まず 0 バイト目と 1 バイト目 (オフセット 0 と 1) は、打楽器音データが置かれているオフセットで、からず 14 が書き込まれる。この値を CPU の Z80 が理解できるように、下位バイト、上位バイトの順に 2 バイトで表わすと、それぞれ 0EH、00H となる。

続く 2 バイト目から 13 バイト目 (オフセット 2~13) には、楽器音のチャンネル 1~6 までのデータのオフセットが置かれる。もしもこのときに、オフセット 0 が指

表 5.5: 6 楽器音+1 打楽器音のデータ構造の例

オフセット	内容
0	0EH 00H
2	21H 00H
4	00H 00H
6	00H 00H
8	00H 00H
10	00H 00H
12	00H 00H
14~	打楽器音データ
33~	楽器音 1 データ

これは 6 楽器音+1 打楽器音のデータ構造の例。14~32 バイト目までに打楽器音の音源データが、32 バイト目からは楽器音の第 1 チャンネルの音源データが置かれ、楽器音の第 2~6 チャンネルは使われていない。

定されれば、そのチャンネルは使われないというわけだ。

たとえば表 5.5 のデータ構造例では、2 バイト目と 3 バイト目 (オフセット 2 と 3) に、21H と 00H が書かれている。これが意味するのは、楽器音のチャンネル 1 の音源データが、オフセット 33 以降に置かれているということだ。そして 4 バイト目から 13 バイト目 (オフセット 4~13) には、00H が書き込まれているので、楽器音のチャンネル 2~6 は使われないことがわかる。

前に紹介した FM 音源をコントロールするプログラムでは、C 言語のプログラムの中にテストデータを埋め込んだので、データの長さを数えてオフセットを指定する必要があった。しかし、よく考えると、以下のようなアセンブラーのプログラムで、音源データを作るほうが簡単だったかもしれない。参考までに書いておくと、

FMDATA:

```
DW 14
DW CH1-FMDATA
DW CH2-FMDATA
DW CH3-FMDATA
DW CH4-FMDATA
DW CH5-FMDATA
DW CH6-FMDATA
DB 打楽器音データ...
```

CH1:

```
DB チャンネル 1 データ...
```

CH2:

```
DB チャンネル 2 データ...
```

(以下、CH6 まで同様)

ということになる。このプログラムなら、ソースリストをアセンブルするときに、MSX-DOS のアセンブラー (M80) がオフセットを自動的に計算してくれるからだ。

もっとも、このプログラムをそのまま実用として使うことはできない。BASIC 言語における PLAY 文の役割を果たすような、ミュージック・マクロ・ランゲージ

(MML) を、FM-BIOS の音源データに変換するためのプログラムが、必要になるだろう。

さて、打楽器音なしで 9 楽器音を演奏する場合は、表 5.6 に掲載したようなデータ構造になる。基本的には、6 楽器音+1 打楽器音のデータ構造と同じで、まず 0 バイト目から 17 バイト目 (オフセット 0~17) までに、楽器音のチャンネル 1~9 の音源データが書き込まれたオフセットを指定する。もしも 00H が指定された場合は、該当するチャンネルは使われないということだ。

また、チャンネル 1 の音源データは、かならず 18 バイト目 (オフセット 18) からはじまることになるので、チャンネル 1 のデータのオフセットには、12H、00H (10 進数で 18) の値がいつでも書き込まれている。

表 5.6: 9 楽器音のデータ構造

オフセット	内容
0	楽器音 1 データのオフセット (注 1)
2	楽器音 2 データのオフセット (注 2)
:	
16	楽器音 9 データのオフセット (注 2)
18	楽器音 1 データ
:	
(注 3)	楽器音 9 データ

(注 1) かならず 12H、00H の 2 バイトを書き込む。

(注 2) 楽器音データの開始位置の、この表のデータの先頭からのバイト単位のオフセットを、それぞれ下位バイト、上位バイトの順に指定する。また、使わないチャンネルに対しては 0 を指定する。

(注 3) オフセットで指定された場所に楽器音データが置かれる。

なお、これは余談になるけど、FM-BIOS は、音源データの先頭が 0EH であるか 12H であるかによって、打楽器音の有無 (つまり 6 楽器音+1 打楽器音か、9 楽器音のみか) を決めている。だから、打楽器音のデータはかならず 14 バイト目 (オフセット 14) から、打楽器音がない場合にはチャンネル 1 の楽器音データがかならず 18 バイト目 (オフセット 18) から、はじまる必要があるようだ。

5.3.2 打楽器音のデータを指定するには

図 5.4 に掲載したのが、打楽器音データの詳細と、実際のデータ列だ。ここでは、5 種類の打楽器を “BSTCH” のアルファベットで、音のデータを 2 進数でそれぞれ表わしている。

まずは次のような 2 バイトのデータで、打楽器ごとの音量を指定してみよう。

101BSTCH
0000VVVV

図 5.4: 打楽器音のデータ

b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀		B	バスドラム
1	0	1	B	S	T	C	H		S	スネアドラム
0	0	0	0						T	タム
0	0	1	B	S	T	C	H		C	シンバル
									H	ハイハット
										音長 (1~255)

10110000	バスドラム
00000000	音量 0
10101000	スネアドラム
00000001	音量 1
00110000	バスドラム
00010100	音長 20
00101000	スネアドラム
00010100	音長 20
00101000	スネアドラム
00010100	音長 20
11111111	データ終了

打楽器を選択するには、BSTCHと書かれた 5 ビットについて、それぞれ 1 か 0 を指定する。音量は最大音量に対する減衰量を、音長は音を出してから次の音を出すまでの間隔を表わしている。

このとき、“BSTCH”の5ビットの列には、音量を指定したい打楽器を1、そうでないものを0で指定する。また、“VVVV”的部分には、音量を指定する0~15の値(実際には2進数の値)が入る。ただし、このときの“音量”は、最大音量に対する減衰量を指定するので、0なら最大の音が、15なら最小の音が出るので注意しよう。

たとえば、バスドラムの音量を0、スネアドラムとシンバルの音量を1に設定するには、

```
10110000
00000000
10101010
00000001
```

と指定すればいい。

音量の指定が終わったら、次に打楽器ごとの音長を指定しよう。ただし、打楽器音自体の長さは常に一定なので、この場合の“音長”とは、音を出してから次の音を出すまでの間隔を指している。そして、

```
001BSTCH
```

によって、打楽器の種類が指定され、次の1バイトで音長(255までの値)を指定する。255以上の音長を指定するには、まず255を書き、その次に実際の音長から

255 を引いた値を指定する。この値が 255 以上ならば、同様の操作を繰り返せばいい。たとえば、

```
00110000
11111111
00000000
```

はバスドラム、音長 255 を表わし、

```
00110010
11111111
11111111
11111111
11101011
```

は、バスドラムとシンバル、音長 1000 を表わしている。

FM 音源の仕様書には“音長”的単位が書かれていなかったけど、テストデータで実測した結果、音長の単位はタイマー割り込みの周期と同じ、60 分の 1 秒だった。

5.3.3 楽器音のデータを指定してみよう

表 5.7 に掲載したのが、楽器音のデータの詳細だ。このデータの中には、表の 1 バイトの値だけで意味を持つものと、続く 1 バイトまたは 2 バイトの値との組み合わせで、意味を持つものがある。

実際に楽器音データを指定する順番は、音量、音色、サスティン、レガート、Q の順だ。

音量の指定は打楽器のデータと同じで、最大音量からの減衰量で表わす。つまり、0 で最大の音が、15 で最小の音が出るという具合。

サスティンは、楽器音の減衰を調整するためのものだ。楽器音のエンベロープは前にも説明したように、“ADSR”という値で決定される。繰り返すなら、A はアタック(立ち上がりの速さ)、D はディケイ(減衰)、S がサスティン(持続の強さ)、R がリリース(消える速さ)の略だ。

OPPL 内蔵音色の“ADSR”は、音色ごとに固定されている。でも、サスティンをオンにすると、リリースが遅くなって、音が伸びる。さらに、サスティンはチャンネルべつに指定可能なので、チャンネル 1 と 2 の両方にギター音を割り当て、チャンネル 1 だけのサスティンをオンにするというような、細かい工夫も可能だ。

レガートをオンにすると、ひとつの音符と次の音符との音がつながる。ただし、レガートを使いすぎると曲のメリハリがなくなるので、一部分のチャンネルのみのレガートをオンにするような工夫が必要だろう。

表 5.7: 楽器音のデータ

値	意味
00H	休符、続く 1 バイトが音長
01H	オクターブ 1 の C、続く 1 バイトが音長
:	:
5FH	オクターブ 7 の A#、続く 1 バイトが音長
60H	音量 0 (最大音量)
:	:
6FH	音量 15 (最小音量)
70H	音色 0 (ROM 内蔵音色またはユーザー指定音色)
71H	音色 1 (バイオリン)
:	:
7FH	音色 15 (エレキベース)
80H	サスティン・オフ
81H	サスティン・オン
82H	続く 1 バイト (00H~3FH) が ROM 内蔵音色番号
83H	続く 2 バイト (下位、上位の順) が音色データの番地
84H	レガート・オフ
85H	レガート・オン
86H	続く 1 バイト (01H~08H) が Q
FFH	データ終了

Q に指定できる値は 1~8 で、音符の長さと実際に音を出す長さの比を表わしている。たとえば、Q = 6 で音符の長さが 80 ならば、 $80 \times 6 \div 8 = 60$ の長さの音が出て、 $80 \times (8 - 6) \div 8 = 20$ の休みが入る。

以上の、レガート、サスティン、Q に指定する値の組み合わせで、音符のつながり方、つまり曲の滑らかさが決まるわけだ。

表 5.8: 楽器音のデータの例

68H	音量 8
73H	音色 3(ギター)
81H	サスティン・オン
84H	レガート・オフ
86H, 06H	Q = 6
25H, 14H	オクターブ 4 の C、音長 20
27H, 14H	オクターブ 4 の D、音長 20
29H, FFH, FFH, FFH, EBH	オクターブ 4 の E、音長 1000
FFH	データ終了

次に、音符ごとの音階と音長を指定する。00H～5FHまでの1バイトの値が音階を表わし、その次の1バイトが音長を表わしている。たとえば、

25H, 14H

の2バイトのデータは、それぞれオクターブ4のCと、音長20を表わしている。また、255以上の音長を表わす方法は打楽器の場合と同様で、たとえば、

29H, FFH, FFH, FFH, EBH

の5バイトのデータは、オクターブ4のEと、音長1000を表わしているわけだ。

5.3.4 OPLL ドライバーでできること

FM-BIOS が持つ機能の中で、タイマー割り込みにより呼び出されて与えられたデータを、自動的に演奏するものを、“OPLL ドライバー”と呼ぶ。ここでは、このドライバーを利用して、BASIC の

```
CALL PITCH
CALL TEMPER
CALL TRANSPOSE
```

と同じ機能を実現する方法を、解説するつもりでいた。ところが、FM-BIOS の開発元に問い合わせてみたところが、“自分でやってください”とのこと。つまり、自分でOPLLのレジスターを書き替えないと、できないことが判明してしまったのだ。

結局、OPLL ドライバーを使った場合は、12平均率でAが440ヘルツの標準的な音律でのみ、演奏が可能だということ。BASIC がサポートするMMLには、音量を細かく設定する機能と、音源チップのレジスターに値を書き込む機能があるけど、FM-BIOS のドライバーで同じことをするのは不可能だ。また、ゲームのバックグラウンドに音楽を鳴らしながら、効果音を出すことも困難だ。

こうして考えてみると、FM 音源を使いやすいものにするには、いま述べたような機能を追加したドライバーと、MML をそのドライバーのデータに変換するプログラムが必要になりそうだ。ここまで記述と、あとで紹介する FM 音源に関する参考書があれば、必要な情報はそろははず。プログラムに自信のある人は、ぜひとも、挑戦してみよう。

5.3.5 音色データを追加してみよう

前にも書いたように、FM 音源の LSI (OPLL) には、15種類の、それをコントロールする FM-BIOS の ROM には、48種類の音色データが用意されている。けれ

図 5.5: 音色データ

	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀	
0	AM	VIB	EGT	KSR		Multiple			Op.0
1	AM	VIB	EGT	KSR		Multiple			Op.1
2	KSL (Op.0)		Total LEVEL	MODELATER					
3	KSL (Op.1)	空き	DC	DM	Feedback				
4		Attack (Op.0)			Decay (Op.0)				
5		Attack (Op.1)			Decay (Op.1)				
6		Sustain (Op.0)			Release (Op.0)				
7		Sustain (Op.1)			Release (Op.1)				

Op.0 はモジュレーター・オペレーター、Op.1 はキャリアー・オペレーターを表わす。この 8 バイトが、OPLL のレジスター 0~7 に書き込まれる。詳細は“MSX2+パワフル活用法（アスキー刊）”を参照。

ども、図 5.5 に掲載したようなデータ構造で、さらに音色を追加することもできるようになっている。

この 8 バイトの音色データは、FM 音源の LSI (OPLL) のレジスター 0~7 に、そのまま書き込まれる。BASIC の拡張コマンドである、

```
CALL VOICE
CALL VOICE COPY
```

に使われる 32 バイトのデータと、形式が異なることに注意しよう。

OPLL の内部では、各チャンネルごとの音色が、0~15 の値で指定されている。このとき、音色 0 は OPLL のレジスター 0~7 までで設定される、オリジナル音色を表わしている。つまり、自作の音色データまたは、FM-BIOS の ROM に内蔵された音色データは、同時に 1 種類のみ使えるわけだ。

このことは、音色の数の制限により発生したこと。チャンネル数の制限ではない。だから、たとえばチャンネル 1 と 2 に自作の音色を割り当て、チャンネル 2~4 に OPLL に内蔵された音色を、割り当てることも可能なのだ。

音色データの内容を解説していると、それだけで 1 冊の本が書けてしまうので、今回は割愛。そのかわり、参考書を紹介する。

“MSX2+ パワフル活用法” 杉谷成一著・アスキー出版局刊

ただし、この本には若干の間違いがあるようだ。あとのページに内容訂正を掲載しておくので、各自で修正しておこう。

このほか、パソコン通信をしているなら一度アクセスしてほしいのが、アスキーネット MSX の “msx.spec” というボード。ここには、マシン語のプログラムが MSX-

MUSIC を使うための、“FM-BIOS” の仕様書が公開されている。また、それだけではなく、MSX に関するさまざまな情報が掲載されているのだ。

5.3.6 サンプルデータを解説する

それでは最後にまとめとして、実際にデータを指定した例を紹介しよう。リスト 5.7 は、前にも掲載したプログラムリストの一部分だ。

まず一番上の部分では、チャンネルごとのデータのオフセットを指定している。打楽器音のデータが 14 バイト目から、楽器音チャンネル 1 のデータが 33 バイト目からはじめり、チャンネル 2~5 は使われないことを表わしている。

リストの中央の部分は、打楽器音のデータだ。まず、全部の打楽器の音量を 8 に設定している。“FM_RVOL”的値は 0xa0 で、これに 31 を加えると 0xbff、つまり全打楽器の音量指定になるわけだ。引き続きバスドラムを音長 20 で、スネアドラムも音長 20 でというように、順番に音を指定していく。“FM_END”的値は 0xff で、データの終わりを表わしている。

リストの下半分は、楽器音チャンネル 1 のデータだ。まず音量 8、音色 3 (OPLL 内蔵のギター)、サステイン・オン、レガート・オフ、Q=6 を指定する。そして、“ドレミファソラシド”を各音長 20 で鳴らし、“FM_END”まできたら終了。このとき、音階を直接 0~95 までの数値で指定すると不便なので、12 音階とオクターブの値をわけている。たとえば、

```
FM_O4 + FM_D
```

によって、オクターブ 4 の D を指定するわけだ。“FM_O4”的値は 37 で、“FM_D”的値は 2 だから、これらを足すとオクターブ 4 の D を表わす 39 になる。

なお、アセンブラー (M80) でテストデータを作るためには、次のように定数を定義すると便利だろう。

```
FM_C EQU 1
FM_CS EQU 2
:
FM_VOL EQU 60H
:
```

そして音量を、

```
DB FM_VOL + 8
```

のように。同じく音階を、

DB FM_04 + FM_C

のように指定すればいい。

リスト 5.7 (楽器音のサンプルデータ)

```
#define TESTLENGTH      20
#define TESTTTIMES       4

static char    fmdata[] = {
/* ここでは、各チャンネルごとのオフセットを指定している。 */
    14, 0,
    33, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
/* これは打楽器音のデータ。 */
/* はじめに全体の音量を設定したあとで、 */
/* それぞれの楽器の音長指定していく。 */
    FM_RVOL + 31, 8,
    0x30, TESTLENGTH,
    0x28, TESTLENGTH,
    0x28, TESTLENGTH,
    0x28, TESTLENGTH,
    0x30, TESTLENGTH,
    0x28, TESTLENGTH,
    0x28, TESTLENGTH,
    0x28, TESTLENGTH,
    FM_END,
/* これは、楽器音チャンネル 1 のデータ。 */
/* このリストでは、チャンネル 2～5 は使われていない。 */
    FM_VOL + 8,
    FM_INST + 3,
    FM_SUSON,
    FM_LEGOFF,
    FM_Q, 6,
    FM_04 + FM_C, TESTLENGTH,
    FM_04 + FM_D, TESTLENGTH,
    FM_04 + FM_E, TESTLENGTH,
    FM_04 + FM_F, TESTLENGTH,
    FM_04 + FM_G, TESTLENGTH,
    FM_04 + FM_A, TESTLENGTH,
    FM_04 + FM_B, TESTLENGTH,
    FM_05 + FM_C, TESTLENGTH,
    FM_END
};
```

$$88\% = 1 - \frac{1}{2} \times \left(\frac{0.00008}{0.00008 + 0.00008} \times 0.04 \right) = \text{volume}\% - 4$$

5.4 FM 音源にまつわるアレコレ

ここまで説明で、FM 音源に関する説明はすべて終わったと思っていたら、やはり残していたことが出てきてしまった。もうしばらく、おつきあいください。

5.4.1 パワフル活用法の内容訂正

MSX2+マシンの参考書として紹介してきた“MSX2+パワフル活用法”(アスキー刊、価格 1240 円 [税込])に、いくつかの誤りが見つかった。本に書いてあるとおりにプログラムしても、予定どおりに動かないで頭をかかえている人も多いはず。このページでは、現在わかっている範囲での誤りを、訂正していくことにする。もしも、ここに記載した以外にも誤りを見つけた人がいたら、M マガ編集部あてに、ぜひ知らせください。

それでは、まず、ひとつ目の訂正から。147 ページに掲載されている、“表 4.4 音色ライブラリー一覧表”を見てみよう。この中の音色番号 10、音色名が“Guitar”となっている項目の“OPLL VOICE”の欄に、“2 ギター”を追加する。

また、この表に記載されている“略号”にも、いくつか誤りがあった。これについては、この本の 136 ページに掲載したプログラム (READFM.BAS) を実行すると、正しい略号を表示するようになっている。それぞれ実際に音色を演奏させながら、略号を確認していってほしい。

続いて、148 ページの“VOICE COPY”に関する説明の部分。文章の真ん中あたりに、“ソース (パラメーター 1) に指定できる音色番号は 0~63 のうち OPLL VOICE の欄に指定がある音色の番号です”となっているけど、正しくは“指定がない音色の番号です”ということになる。

これと同様に、その少しあとにある“ソースに OPLL VOICE 欄に指定のない音色の番号を指定すると “Illegal function call” となります”という記述も逆。正しくは“指定のある音色の番号を指定すると……”となるわけだ。

また、151 ページから 158 ページにかけて掲載されていた、OPLL のレジスターの表にも、いくつかの誤りがあった。それらを正したものを作成しておいたので、参考にしてほしい。これをもとに、手元にある MSX2+パワフル活用法の内容を修正しておくと、便利だと思う。

レジスターの説明に関連して、155 ページにある、目的の周波数から F-Number と BLOCK を求める式にも、誤りがあった。一番下に掲載されている、

$$F\text{-Number} = (440 \times 2^{18} \div 50000) \div 2^4 - 1 = 288$$

という式は、正しくは、

$$F\text{-Number} = (440 \times 2^{18} \div 50000) \div 2^{(4 - 1)} = 288$$

ということになる。

最後に、これは MSX2+ パワフル活用法に限ったことではないのだけど、FM 音源に関する楽器音データの指定方法に、一般に間違った説明がされているようなので、訂正しておく。

音符ごとの音階を指定するのに、00H~5FH までが、それぞれオクターブ 1 の C~オクターブ 7 の B までに対応している、と一般にはいわれているけど、これは間違

図 5.6: OPLL のレジスター一覧

	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀					
00H	AM(M)	VIB(M)	EGT(M)	KSR(M)				Multiple(M)					
01H	AM(C)	VIB(C)	EGT(C)	KSR(C)				Multiple(C)					
02H	KSL(M) Total Level Modelater												
03H	KSL(C)	空き	DC	DM				Feed Back					
04H	Attack(M)				Decay(M)								
05H	Attack(C)				Decay(C)								
06H	Sustain(M)				Release(M)								
07H	Sustain(C)				Release(C)								
0EH	空き	R	BD	SD	TOM	T-CT		HH					
0FH	検査用レジスター												
10H													
:	F-number												
18H													
20H													
:	空き	Sus.	Key	Block			F-number						
28H													
30H													
:	Inst.				Vol.								
38H													

リズムモードの場合

	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀
36H	空き				Bass Drum			
37H	Hi Hat				Snare Drum			
38H	Tom Tom				Top Cymbal			

表中で (M) となっている部分は、モジュレーターとして働くオペレーター 0 を、(C) となっているものは、キャリアーとして働くオペレーター 1 を示している。詳細については、“MSX2+パワフル活用法”または、ヤマハの技術資料を参照のこと。

い。正しくは 00H は休符となり、続く 01H~5FH までが、オクターブ 1 の C~オクターブ 7 の A#に対応している。

5.4.2 MSX-MUSIC の音色データ一覧

プログラマーのみなさまのご要望にお答えして、MSX-MUSIC の ROM に内蔵された音色データの、ダンプリストを掲載しよう。

表 5.9 の左側に掲載したのは、BASIC の “CALL VOICE COPY” ステートメントで得られる 32 バイトの音色データから、OPLL のレジスター 0~7 に書き込まれる 8 バイトのデータを抜き出したもの。音色データには、“ボイス移調”と呼ばれる音の高さを制御する 2 バイトのデータも含まれているけど、OPLL レジスターに直接書き込まれるデータではないので、掲載を省略した。

音色 60 と音色 61 は、この表ではまったく同じに見えるけど、ボイス移調の値が異なるので、実際には違う音色になっている。また、表中で “using data of OPLL” と書かれている音色番号については、OPLL に内蔵された音色が使われる所以、ROM には音色データが含まれていない。

さて、FM-BIOS で得られる 63 種類の ROM 内蔵音色データは、この BASIC の ROM 内蔵音色データと共に、みんなが信じて疑わなかった。ところが、実際には異なっているという事実がいまになって判明した。というわけで、表 5.9 の右側は、FM-BIOS の “RDDATA” 機能を使って得られた、各音色につき 8 バイトの音色データだ。

表 5.9 の左右を比べるとわかるように、拡張 BASIC と FM-BIOS について音色番号と名称の対応は共通だけど、音色データは微妙に違っている。そのため、BASIC の MML を使って曲を試作して、そのデータを FM-BIOS 用に変換するような場合に、音色の違いが問題になるかもしれない。

なお、FM-BIOS の大部分の音色データについて、レジスター 3 に書き込まれる値が 20H となっていることを、不思議に感じる人もいるかもしれない。でも、レジスター 3 のビット 5 は “空き” になっているので、レジスター 3 に書き込まれる値が 20H であっても、そうでなくとも、実際に演奏される音色は同じになる。

表 5.9: 音色データ一覧

番号	音色名	拡張 BASIC の音色データ	FM-BIOS の音色データ
0	Piano 1	using data of OPLL(3)	31 11 0E 20 D9 B2 11 F4
1	Piano 2	30 10 0F 04 D9 B2 10 F4	30 10 0F 20 D9 B2 10 F3
2	Violin	using data of OPLL(1)	61 61 12 20 B4 56 14 17
3	Flute 1	using data of OPLL(4)	61 31 20 20 6C 43 18 26
4	Clarinet	using data of OPLL(5)	A2 30 A0 20 88 54 14 06
5	Oboe	using data of OPLL(6)	31 34 20 20 72 56 0A 1C
6	Trumpet	using data of OPLL(7)	31 71 16 20 51 52 26 24
7	Pipe Organ 1	34 30 37 06 50 30 76 06	34 30 37 20 50 30 76 06
8	Xylophone	17 52 18 05 88 D9 66 24	17 52 18 20 88 D9 66 24
9	Organ	using data of OPLL(8)	E1 63 0A 20 FC F8 28 29
10	Guitar	using data of OPLL(2)	02 41 15 20 A3 A3 75 05
11	Santool 1	19 53 0C 06 C7 F5 11 03	19 53 0C 20 C7 F5 11 03
12	Electric Piano 1	using data of OPLL(15)	23 43 09 20 DD BF 4A 05
13	Clavicode 1	03 09 11 06 D2 B4 F5 F6	03 09 11 20 D2 B4 F4 F5
14	Harpsicode 1	using data of OPLL(11)	01 00 06 20 A3 E2 F4 F4
15	Harpsicode 2	01 01 11 06 C0 B4 01 F7	01 01 11 20 C0 B4 01 F6
16	Vibraphone	using data of OPLL(12)	F9 F1 24 20 95 D1 E5 F2
17	Koto 1	13 11 0C 06 FC D2 33 84	13 11 0C 20 FC D2 33 83
18	Taiko	01 10 0E 07 CA E6 44 24	01 10 0E 20 CA E6 44 24
19	Engine 1	E0 F4 1B 87 11 F0 04 08	E0 F4 1B 20 11 F0 04 08
20	UFO	FF 70 19 07 50 1F 05 01	FF 70 19 20 50 1F 05 01
21	Synthesizer Bell	13 11 11 07 FA F2 21 F5	13 11 11 20 FA F2 21 F4
22	Chime	A6 42 10 05 FB B9 11 02	A6 42 10 20 FB B9 11 02
23	Synthesizer Bass	using data of OPLL(13)	40 31 89 20 C7 F9 14 04
24	Synthesizer	using data of OPLL(10)	42 44 0B 20 94 B0 33 F6
25	Synthesizer Percussion	01 03 0B 07 BA D9 25 06	01 03 0B 20 BA D9 25 06
26	Synthesizer Rhythm	40 00 00 07 FA D9 37 04	40 00 00 20 FA D9 37 04
27	Harm Drum	02 03 09 07 CB FF 39 06	02 03 09 20 CB FF 39 06
28	Cowbell	18 11 09 05 F8 F5 26 26	18 11 09 20 F8 F5 26 26
29	Close Hi-hat	0B 04 09 07 F0 F5 01 27	0B 04 09 20 F0 F5 01 27
30	Snare Drum	40 40 07 07 D0 D6 01 27	40 40 07 20 D0 D6 01 27
31	Bass Drum	00 01 07 06 CB E3 36 25	00 01 07 20 CB E3 36 25

番号	音色名	拡張 BASIC の音色データ					FM-BIOS の音色データ				
32	Piano 3	11	11	08	04	FA B2 20 F5	11	11	08	20	FA B2 20 F4
33	Electric Piano 2	using data of OPLL(14)					11	11	11	20	CO B2 01 F4
34	Santool 2	19	53	15	07	E7 95 21 03	19	53	15	20	E7 95 21 03
35	Brass	30	70	19	07	42 62 26 24	30	70	19	20	42 62 26 24
36	Flute 2	62	71	25	07	64 43 12 26	62	71	25	20	64 43 12 26
37	Clavicode 2	21	03	0B	05	90 D4 02 F6	21	03	0B	20	90 D4 02 F5
38	Clavicode 3	01	03	0A	05	90 A4 03 F6	01	03	0A	20	90 A4 03 F5
39	Koto 2	43	53	0E	85	B5 E9 85 04	43	53	0E	20	B5 E9 84 04
40	Pipe Organ 2	34	30	26	06	50 30 76 06	34	30	26	20	50 30 76 06
41	PohdsPLA	73	33	5A	06	99 F5 14 15	73	33	5A	20	99 F5 14 15
42	RohdsPRA	73	13	16	05	F9 F5 33 03	73	13	16	20	F9 F5 33 03
43	Orch L	61	21	15	07	76 54 23 06	61	21	15	20	76 54 23 06
44	Orch R	63	70	1B	07	75 4B 45 15	63	70	1B	20	75 4B 45 15
45	Synthesizer Violin	61	A1	0A	05	76 54 12 07	61	A1	0A	20	76 54 12 07
46	Synthesizer Organ	61	78	0D	05	85 F2 14 03	61	78	0D	20	85 F2 14 03
47	Synthesizer Brass	31	71	15	07	B6 F9 03 26	31	71	15	20	B6 F9 03 26
48	Tube	using data of OPLL(9)					61	71	0D	20	75 F2 18 03
49	Shamisen	03	0C	14	06	A7 FC 13 15	03	0C	14	20	A7 FC 13 15
50	Magical	13	32	81	03	20 85 03 B0	13	32	80	20	20 85 03 AF
51	Huwawa	F1	31	17	05	23 40 14 09	F1	31	17	20	23 40 14 09
52	Wander Flat	F0	74	17	47	5A 43 06 FD	F0	74	17	20	5A 43 06 FC
53	Hardrock	20	71	0D	06	C1 D5 56 06	20	71	0D	20	C1 D5 56 06
54	Machine	30	32	06	06	40 40 04 74	30	32	06	20	40 40 04 74
55	Machine V	30	32	03	03	40 40 04 74	30	32	03	20	40 40 04 74
56	Comic	01	08	0D	07	78 F8 7F FA	01	08	0D	20	78 F8 7F F9
57	SE-Comic	C8	CO	0B	05	76 F7 11 FA	C8	CO	0B	20	76 F7 11 F9
58	SE-Laser	49	40	0B	07	B4 F9 00 05	49	40	0B	20	B4 F9 FF 05
59	SE-Noise	CD	42	0C	06	A2 F0 00 01	CD	42	0C	20	A2 F0 00 01
60	SE-Star 1	51	42	13	07	13 10 42 01	51	42	13	20	13 10 42 01
61	SE-Star 2	51	42	13	07	13 10 42 01	51	42	13	20	13 10 42 01
62	Engine 2	30	34	12	06	23 70 26 02	30	34	12	20	23 70 26 02
63	Silence	00	00	00	00	00 00 00 00	00	00	FF	20	00 00 FF FF

APPENDIX



R800 インストラクション表

1991年1月24日

株式会社アスキー

システム事業部、MSX マガジン編集部

マシン語レベルのプログラミングに燃える人なら、ぜひとも挑戦してほしいのが、R800 でのプログラム開発。ニーモニックや命令動作、マシン語コードを記した、インストラクション表を掲載したので活用してほしい。さあ、R800 の速度を活かしたプログラムはできるかな？

A.1 インストラクション表はこうして使おう

この表は、命令の種類ごとに分類して、R800 のインストラクションをまとめたもの。表中の“ニーモニック”は各命令の名前を現わし、“命令動作”でその動作内容を簡潔に示している。

命令動作の欄で“←”とあるのは、右側の内容を左側に代入することを、カッコでくくられたものは、くくられたレジスターなどで示されるメモリーの内容を、それぞれ意味している。たとえば、

`r ← [.hl]`

とあるのは、.hl レジスターで示されるアドレスのメモリーの内容を、8 ビットレジスターに代入するということだ。ただし入出力命令の [n] と [.c] は、対応する入出力ポートの番号を意味している。

“フラグ”の欄は各フラグの動作を、“オペコード”はそれぞれの命令に対するマシン語コードを、2 進数と 16 進数で記したもの。その右側の“B”と“C”は、各命令の長さ(バイト数)と、命令を実行するのに要するクロック数を、それぞれ現わしている。

このほか、インストラクション表に出てくる略号に関して、次の凡例にまとめておいたので参考にしてほしい。また、表に記載されたニーモニックが Z80 と違っている理由は、それがザイログ社の著作物だから。といっても、R800 で追加された乗算命令や、Z80 で正式に動作が保証されていなかった命令以外は、ニーモニックの違いがあるにせよ、命令動作はすべて同じになっている。Z80 のインストラクション表と見比べながら、プログラムしていってほしい。

.a{7}	レジスター.a の最上位ビット
.a{4..7}	レジスター.a のビット 4-7
;	動作の区切り
.de:.hl	上位 16 ビットが.de、下位 16 ビットが.hl に入る、32 ビット整数
[.ix+d]	.ix に 8 ビットの符号つき変位を足した値が示すアドレス
C	キャリーフラグ
Z	ゼロフラグ
P _N	パリティー・オーバーフローフラグ
S	サインフラグ
N	減算フラグ
H	ハーフキャリーフラグ
•	フラグは変化しない
↑	フラグは実行結果により変化する
0	フラグは 0
1	フラグは 1
?	不定になる
V	オーバーフローフラグとして使われる
P	パリティーフラグとして使われる
FF	割り込みフリップフロップの値が入る
r,r'	8 ビットレジスター、.a,.b,.c,.d,.e,.h,.l
u,u'	8 ビットレジスター、.a,.b,.c,.d,.e,.ixh,.ixl
v,v'	8 ビットレジスター、.a,.b,.c,.d,.e,.iyh,.iyL
p	8 ビットレジスター、.ixh,.ixl
q	8 ビットレジスター、.iyh,.iyL
ss	16 ビットレジスター、.bc,.de,.hl,.sp
pp	16 ビットレジスター、.bc,.de,.ix,.sp
rr	16 ビットレジスター、.bc,.de,.iy,.sp
qq	16 ビットレジスター、.bc,.de,.hl,.af
e	short br 系の命令の飛び先アドレスへの差分、8 ビットの符号つき即値 (+127~-128)
k	brk 命令の飛び先アドレス、00h,08h,10h,18h,20h,28h,30h,38h
nn	16 ビットの即値、もしくは絶対アドレス
n	8 ビットの即値
b	ビット演算命令の第何ビットかを示す値
NOT	ビットを反転する
∨	ビットの OR をとる
⊻	ビットの XOR をとる
∧	ビットの AND をとる
tmp	一時的に値を待避する
B	命令のバイト数
C	命令の実行に必要な最小クロック数

分岐命令、コール命令でクロック数がふたつ書いてあるものは、上が条件が成立しないとき、下が条件が成立したときを意味する。

また、入出力命令でクロック数がふたつ書いてあるものは、上がまだ転送が終わらないとき、下が転送が終わったときをそれぞれ意味している。

ここに記す命令表のクロック数は、SYSCLK 換算で XTAL の発振周波数の 4 分の 1。またノーウェイトで実行したときの値で、DRAM 上で実行したときはページブレークやリフレッシュにより、自動的にウェイトが挿入される。

A.2 8 ビット移動命令

ニーモニック	命令動作	flags S Z H P _N NC	オペコード		BC
			76543210	Hex	
ldr,r'	r←r'	• • • • •	01 r r'		1 1
ldr,n	r←n	• • • • •	00 r 110 ← n →		2 2
ldr,[.hl]	r←[.hl]	• • • • •	01 r 110		1 2
ldr,[.ix+d]	r←[.ix+d]	• • • • •	11011101 01 r 110 ← d →	DD	3 5
ldr,[.iy+d]	r←[.iy+d]	• • • • •	11111101 01 r 110 ← d →	FD	3 5
ld [.hl],r	[.hl]←r	• • • • •	01110 r		1 2
ld [.ix+d],r	[.ix+d]←r	• • • • •	11011101 01110 r ← d →	DD	3 5
ld [.iy+d],r	[.iy+d]←r	• • • • •	11111101 01110 r ← d →	FD	3 5
ld u,u'	u←u'	• • • • •	11011101 01 u u'	DD	2 2
ld v,v'	v←v'	• • • • •	11111101 01 v v'	FD	2 2
ld u,n	u←n	• • • • •	11011101 00 u 110 ← n →	DD	3 3
ld v,n	v←n	• • • • •	11111101 00 v 110 ← n →	FD	3 3
ld [.hl],n	[.hl]←n	• • • • •	00110110 ← n →	36	2 3
ld [.ix+d],n	[.ix+d]←n	• • • • •	11011101 00110110 ← d → ← n →	DD	4 5
ld [.iy+d],n	[.iy+d]←n	• • • • •	11111101 00110110 ← d → ← n →	FD	4 5

ニーモニック	命令動作	flags S Z H P _N N C	オペコード		B	C
			76 543 210	Hex		
ld .a,i	.a ← .i	↑ ↑ 0 FF 0 •	11 101 101 01 010 111	ED 57	2	2
ld .a,r	.a ← .r	↑ ↑ 0 FF 0 •	11 101 101 01 011 111	ED 5F	2	2
ld .i,a	.i ← .a	• • • • •	11 101 101 01 000 111	ED 47	2	2
ld .r,a	.r ← .a	• • • • •	11 101 101 01 001 111	ED 4F	2	2
ld .a,[.bc]	.a ← [.bc]	• • • • •	00 001 010	0A	1	2
ld .a,[.de]	.a ← [.de]	• • • • •	00 011 010	1A	1	2
ld .a,[nn]	.a ← [nn]	• • • • •	00 111 010 ← nn _l → ← nn _h →	3A	3	4
ld [.bc],a	[.bc] ← .a	• • • • •	00 000 010	02	1	2
ld [.de],a	[.de] ← .a	• • • • •	00 010 010	12	1	2
ld [nn],a	[nn] ← .a	• • • • •	00 110 010 ← nn _l → ← nn _h →	32	3	4

000	001	010	011	100	101	110	111
r .b	.c	.d	.e	.h	.l		.a
u .b	.c	.d	.e	.ixh	.ixl		.a
v .b	.c	.d	.e	.iyh	.iyi		.a

A.3 16 ビット移動命令

ニーモニック	命令動作	flags S Z H P _N N C	オペコード		B	C
			76 543 210	Hex		
ld ss,nn	ss ← nn	• • • • •	00 ss 0001 ← nn _l → ← nn _h →		3	3
ld .ix,nn	.ix ← nn	• • • • •	11 011 101 00 100 001 ← nn _l → ← nn _h →	DD 21	4	4
ld .iy,nn	.iy ← nn	• • • • •	11 111 101 00 100 001 ← nn _l → ← nn _h →	FD 21	4	4
ld .sp,.hl	.sp ← .hl	• • • • •	11 111 001	F9	1	1
ld .sp,.ix	.sp ← .ix	• • • • •	11 011 101 11 111 001	DD F9	2	2
ld .sp,.iy	.sp ← .iy	• • • • •	11 111 101 11 111 001	FD F9	2	2

二モニック	命令動作	flags S Z H P _V N C	オペコード 76 543 210 Hex	B	C
ld ss,[nn]	ss _h ←[nn+1] ss _l ←[nn]	• • • • •	11 101 101 01 ss 1 011 ← nn _l → ← nn _h →	ED	4 6
ld .hl,[nn]	.h←[nn+1] .l←[nn]	• • • • •	00 101 010 ← nn _l → ← nn _h →	2A	3 5
ld .ix,[nn]	.ixh←[nn+1] .ixl←[nn]	• • • • •	11 011 101 00 101 010 ← nn _l → ← nn _h →	DD 2A	4 6
ld .iy,[nn]	.iyh←[nn+1] .iyL←[nn]	• • • • •	11 111 101 00 101 010 ← nn _l → ← nn _h →	FD 2A	4 6
ld [nn],ss	[nn+1]←ss _h [nn]←ss _l	• • • • •	11 101 101 01 ss 0 011 ← nn _l → ← nn _h →	ED	4 6
ld [nn],.hl	[nn+1]←.h [nn]←.l	• • • • •	00 100 010 ← nn _l → ← nn _h →	22	3 5
ld [nn],.ix	[nn+1]←.ixh [nn]←.ixl	• • • • •	11 011 101 00 100 010 ← nn _l → ← nn _h →	DD 22	4 6
ld [nn],.iy	[nn+1]←.iyh [nn]←.iyL	• • • • •	11 111 101 00 100 010 ← nn _l → ← nn _h →	FD 22	4 6

00	01	10	11
ss, bc, de, hl, sp			

12	100 001 00		
13	—		
14	—		
15	—		
16	101 011 11	• • • • •	111 101 FD 4
17	100 001 00		
18	—		
19	—		
20	—		
21	100 011 00	• • • • •	111 → Q8
22	110 111 00	• • • • •	111, q8, b8
23	110 111 01	• • • • •	xi, → Q8
24	111 101 01	• • • • •	xi, q8, b8
25	111 101 11	• • • • •	vi, → Q8
26	111 101 11	• • • • •	vi, q8, b8

A.4 交換命令

ニーモニック	命令動作	flags S Z H P _V N C	オペコード		B	C
			76543210	Hex		
xch .de,.hl	.de↔.hl	• • • • •	11101011	EB	1	1
xch .af,.af'	.af↔.af'	↑↑↑↑↑↑	00001000	08	1	1
xch [.sp],.hl	.l↔[.sp];.h↔[.sp+1]	• • • • •	11100011	E3	1	5
xch [.sp],.ix	.ixl↔[.sp] .ixh↔[.sp+1]	• • • • •	11011101 11100011	DD E3	2	6
xch [.sp],.iy	.iyL↔[.sp] .iyH↔[.sp+1]	• • • • •	11111101 11100011	FD E3	2	6
xchx	.bc↔.bc';.de↔.de';.hl↔.hl'	• • • • •	11011001	D9	1	1

A.5 スタック操作命令

ニーモニック	命令動作	flags S Z H P _V N C	オペコード		B	C
			76543210	Hex		
push qq	[.sp-2]←qq _l ; [.sp-1]←qq _h .sp←.sp-2	• • • • •	11qq0101		1	4
push .ix	[.sp-2]←.ixl; [.sp-1]←.ixh .sp←.sp-2	• • • • •	11011101 11100101	DD E5	2	5
push .iy	[.sp-2]←.iyL; [.sp-1]←.iyH .sp←.sp-2	• • • • •	11111101 11100101	FD E5	2	5
pop qq	qq _l ←[.sp]; qq _h ←[.sp+1] .sp←.sp+2	• • • • •	11qq0001		1	3
pop .ix	.ixl←[.sp]; .ixh←[.sp+1] .sp←.sp+2	• • • • •	11011101 11100001	DD E1	2	4
pop .iy	.iyL←[.sp]; .iyH←[.sp+1] .sp←.sp+2	• • • • •	11111101 11100001	FD E1	2	4

00	01	10	11
qq.bc	.de	.hl	.af

pop .af のときは flags はすべて変化する

add	0100
sub	1001
mul	0010
div	1000
neg	0101

A.6 ブロック転送命令

ニーモニック	命令動作	flags S Z H P _N N C	オペコード		B	C
			76543210	Hex		
move [.hl++], [.de++]	[.de] ← [.hl]; .de ← .de + 1 .hl ← .hl + 1; .bc ← .bc - 1	• • 0 ↑ 0 • *1	11101101 10100000	ED A0	2	4
move [.hl--], [.de--]	[.de] ← [.hl]; .de ← .de - 1 .hl ← .hl - 1; .bc ← .bc - 1	• • 0 ↑ 0 • *1	11101101 10101000	ED A8	2	4
movem [.hl++], [.de++]	repeat; [.de] ← [.hl]; .de ← .de + 1 .hl ← .hl + 1; .bc ← .bc - 1; until .bc = 0	• • 0 0 0 • 10110000	11101101 10110000	ED B0	2	4
movem [.hl--], [.de--]	repeat; [.de] ← [.hl]; .de ← .de - 1 .hl ← .hl - 1; .bc ← .bc - 1; until .bc = 0	• • 0 0 0 • 10111000	11101101 10111000	ED B8	2	4

*1. bc - 1 = 0 のとき 0、その他 1

A.7 ブロックサーチ命令

ニーモニック	命令動作	flags S Z H P _N N C	オペコード		B	C
			76543210	Hex		
cmp .a,[.hl++]	.a - [.hl]; .hl ← .hl + 1 .bc ← .bc - 1	↑ ↓ ↓ ↓ 1 • *2 *1	11101101 10100001	ED A1	2	4
cmp .a,[.hl--]	.a - [.hl]; .hl ← .hl - 1 .bc ← .bc - 1	↑ ↓ ↓ ↓ 1 • *2 *1	11101101 10101001	ED A9	2	4
cmpm .a,[.hl++]	repeat;.a - [.hl]; .hl ← .hl + 1 .bc ← .bc - 1; until .bc = 0 or .a = [.hl]	↑ ↓ ↓ ↓ 1 • *2 *1	11101101 10110001	ED B1	2	5
cmpm .a,[.hl--]	repeat;.a - [.hl]; .hl ← .hl - 1 .bc ← .bc - 1; until .bc = 0 or .a = [.hl]	↑ ↓ ↓ ↓ 1 • *2 *1	11101101 10111001	ED B9	2	5

*1. bc - 1 = 0 のとき 0、その他 1

*2.a=[.hl] のとき 1、その他 0

A.8 乗算命令

ニーモニック	命令動作	flags S Z H P _N N C	オペコード		B	C
			76543210	Hex		
mulub .a,r	.hl ← .a * r	0 ↓ • 0 • ↓ 11 r 001	11101101 11 r 001	ED E1	2	14
muluw .hl,ss	.de: .hl ← .hl * ss	0 ↓ • 0 • ↓ 11 ss 0011	11101101 11 ss 0011	ED E9	2	36

mulub では r が.b,.c,.d,.e のとき以外は動作が保証されない

muluw では ss が.bc,.sp のとき以外は動作が保証されない

A.9 加算命令

ニーモニック	命令動作	flags S Z H P _N N C	オペコード		B	C
			76543210	[Hex]		
add .a,r	.a←.a+r	↑↓↓V0↓	10000	r		1 1
add .a,p	.a←.a+p	↑↓↓V0↓	11011101	DD	2	2
			10000	p		
add .a,q	.a←.a+q	↑↓↓V0↓	11111101	FD	2	2
			10000	q		
add .a,[.hl]	.a←.a+[.hl]	↑↓↓V0↓	10000110	86	1	2
add .a,[.ix+d]	.a←.a+[.ix+d]	↑↓↓V0↓	11011101	DD	3	5
			10000110	86		
			← d →			
add .a,[.iy+d]	.a←.a+[.iy+d]	↑↓↓V0↓	11111101	FD	3	5
			10000110	86		
			← d →			
add .a,n	.a←.a+n	↑↓↓V0↓	11000110	C6	2	2
			← n →			
addc .a,r	.a←.a+r+c	↑↓↓V0↓	10001	r		1 1
addc .a,p	.a←.a+p+c	↑↓↓V0↓	11011101	DD	2	2
			10001	p		
addc .a,q	.a←.a+q+c	↑↓↓V0↓	11111101	FD	2	2
			10001	q		
addc .a,[.hl]	.a←.a+[.hl]+c	↑↓↓V0↓	10001110	8E	1	2
addc .a,[.ix+d]	.a←.a+[.ix+d]+c	↑↓↓V0↓	11011101	DD	3	5
			10001110	8E		
			← d →			
addc .a,[.iy+d]	.a←.a+[.iy+d]+c	↑↓↓V0↓	11111101	FD	3	5
			10001110	8E		
			← d →			
addc .a,n	.a←.a+n+c	↑↓↓V0↓	11001110	CE	2	2
			← n →			
addc .hl,ss	.hl←.hl+ss+c	↑↓?V0↓	11101101	ED	2	2
			01 ss 1010			
add .hl,ss	.hl←.hl+ss	••?•0↓	00 ss 1001		1	1
add .ix,pp	.ix←.ix+pp	••?•0↓	11011101	DD	2	2
			00 pp 1001			
add .iy,rr	.iy←.iy+rr	••?•0↓	11111101	FD	2	2
			00 rr 1001			

ニーモニック	命令動作	flags S Z H P V N C	オペコード		B	C
			76	543210	Hex	
inc r	r ← r + 1	↑ ↓ ↓ V 0 •	00	r 100	1	1
inc p	p ← p + 1	↑ ↓ ↓ V 0 •	11 011101	DD	2	2
			00 p 100			
inc q	q ← q + 1	↑ ↓ ↓ V 0 •	11 111101	FD	2	2
			00 q 100			
inc [.hl]	[.hl] ← [.hl] + 1	↑ ↓ ↓ V 0 •	00 110100	34	1	4
inc [.ix+d]	[.ix+d] ← [.ix+d] + 1	↑ ↓ ↓ V 0 •	11 011101	DD	3	7
			00 110100	34		
			← d →			
inc [.iy+d]	[.iy+d] ← [.iy+d] + 1	↑ ↓ ↓ V 0 •	11 111101	FD	3	7
			00 110100	34		
			← d →			
inc ss	ss ← ss + 1	• • • • •	00 ss 0011		1	1
inc .ix	.ix ← .ix + 1	• • • • •	11 011101	DD	2	2
			00 100011	23		
inc .iy	.iy ← .iy + 1	• • • • •	11 111101	FD	2	2
			00 100011	23		

	00	01	10	11
ss	.bc	.de	.hl	.sp
pp	.bc	.de	.ix	.sp
rr	.bc	.de	.iy	.sp

	000	001	010	011	100	101	110	111
p				.ixh	.ixl			
q				.iyh	.iyi			

A.10 減算命令

ニーモニック	命令動作	flags S Z H P _N N C	オペコード		BC
			76543210	Hex	
sub .a,r	.a←.a-r	↑ ↓ ↓ V 1 ↓	10010	r	1 1
sub .a,p	.a←.a-p	↑ ↓ ↓ V 1 ↓	11011101	DD	2 2
			10010 p		
sub .a,q	.a←.a-q	↑ ↓ ↓ V 1 ↓	11111101	FD	2 2
			10010 q		
sub .a,[.hl]	.a←.a-[.hl]	↑ ↓ ↓ V 1 ↓	10010110	96	1 2
sub .a,[.ix+d]	.a←.a-[.ix+d]	↑ ↓ ↓ V 1 ↓	11011101	DD	3 5
			10010110	96	
			← d →		
sub .a,[.iy+d]	.a←.a-[.iy+d]	↑ ↓ ↓ V 1 ↓	11111101	FD	3 5
			10010110	96	
			← d →		
sub .a,n	.a←.a-n	↑ ↓ ↓ V 1 ↓	11010110	D6	2 2
			← n →		
subc.a,r	.a←.a-r-C	↑ ↓ ↓ V 1 ↓	10011	r	1 1
subc.a,p	.a←.a-p-C	↑ ↓ ↓ V 1 ↓	11011101	DD	2 2
			10011 p		
subc.a,q	.a←.a-q-C	↑ ↓ ↓ V 1 ↓	11111101	FD	2 2
			10011 q		
subc.a,[.hl]	.a←.a-[.hl]-C	↑ ↓ ↓ V 1 ↓	10011110	9E	1 2
subc.a,[.ix+d]	.a←.a-[.ix+d]-C	↑ ↓ ↓ V 1 ↓	11011101	DD	3 5
			10011110	9E	
			← d →		
subc.a,[.iy+d]	.a←.a-[.iy+d]-C	↑ ↓ ↓ V 1 ↓	11111101	FD	3 5
			10011110	9E	
			← d →		
subc.a,n	.a←.a-n-C	↑ ↓ ↓ V 1 ↓	11011110	DE	2 2
			← n →		
subc.hl,ss	.hl←.hl-ss-C	↑ ↓ ? V 1 ↓	11101101	ED	2 2
			01 ss 0010		

xor .a,.hl	.a←.a⊕hl
xor .a,.hl	.a←.a⊕hl
xor .a,.iy+d	.a←.a⊕iy+d
xor .a,.iy+d	.a←.a⊕iy+d

ニーモニック	命令動作	flags S Z H P V N C	オペコード		B	C
			76	543210	Hex	
decr	r←r-1	↑ ↓ ↓ V 1 •	00 r 101		1	1
dec p	p←p-1	↑ ↓ ↓ V 1 •	11011101 00 p 101	DD	2	2
dec q	q←q-1	↑ ↓ ↓ V 1 •	11111101 00 q 101	FD	2	2
dec [.hl]	[.hl]←[.hl]-1	↑ ↓ ↓ V 1 •	00110101	35	1	4
dec [.ix+d]	[.ix+d]←[.ix+d]-1	↑ ↓ ↓ V 1 •	11011101 00110101 ← d →	DD 35	3	7
dec [.iy+d]	[.iy+d]←[.iy+d]-1	↑ ↓ ↓ V 1 •	11111101 00110101 ← d →	FD 35	3	7
decss	ss←ss-1	• • • • • •	00 ss 1011		1	1
dec .ix	.ix←.ix-1	• • • • • •	11011101 00101011	DD 2B	2	2
dec .iy	.iy←.iy-1	• • • • • •	11111101 00101011	FD 2B	2	2

A.11 比較命令

ニーモニック	命令動作	flags S Z H P V N C	オペコード		B	C
			76	543210	Hex	
cmp .a,r	.a-r	↑ ↓ ↓ V 1 ↑	10111 r		1	1
cmp .a,p	.a-p	↑ ↓ ↓ V 1 ↑	11011101 10111 p	DD	2	2
cmp .a,q	.a-q	↑ ↓ ↓ V 1 ↑	11111101 10111 q	FD	2	2
cmp .a,[.hl]	.a-[.hl]	↑ ↓ ↓ V 1 ↑	10111110	BE	1	2
cmp .a,[.ix+d]	.a-[.ix+d]	↑ ↓ ↓ V 1 ↑	11011101 10111110 ← d →	DD BE	3	5
cmp .a,[.iy+d]	.a-[.iy+d]	↑ ↓ ↓ V 1 ↑	11111101 10111110 ← d →	FD BE	3	5
cmp .a,n	.a-n	↑ ↓ ↓ V 1 ↑	11111110 ← n →	FE	2	2

A.12 論理演算命令

ニーモニック	命令動作	flags S Z H P V N C	オペコード		B	C
			76543210	[Hex]		
and .a,r	.a←.a∧r	↑ ↓ 1 P 0 0	10100	r		1 1
and .a,p	.a←.a∧p	↑ ↓ 1 P 0 0	11011101		DD	2 2
			10100	p		
and .a,q	.a←.a∧q	↑ ↓ 1 P 0 0	11111101		FD	2 2
			10100	q		
and .a,[.hl]	.a←.a∧[.hl]	↑ ↓ 1 P 0 0	10100110		A6	1 2
and .a,[.ix+d]	.a←.a∧[.ix+d]	↑ ↓ 1 P 0 0	11011101		DD	3 5
			10100110		A6	
			← d →			
and .a,[.iy+d]	.a←.a∧[.iy+d]	↑ ↓ 1 P 0 0	11111101		FD	3 5
			10100110		A6	
			← d →			
and .a,n	.a←.a∧n	↑ ↓ 1 P 0 0	11100110		E6	2 2
			← n →			
or .a,r	.a←.a∨r	↑ ↓ 0 P 0 0	10110	r		1 1
or .a,p	.a←.a∨p	↑ ↓ 0 P 0 0	11011101		DD	2 2
			10110	p		
or .a,q	.a←.a∨q	↑ ↓ 0 P 0 0	11111101		FD	2 2
			10110	q		
or .a,[.hl]	.a←.a∨[.hl]	↑ ↓ 0 P 0 0	10110110		B6	1 2
or .a,[.ix+d]	.a←.a∨[.ix+d]	↑ ↓ 0 P 0 0	11011101		DD	3 5
			10110110		B6	
			← d →			
or .a,[.iy+d]	.a←.a∨[.iy+d]	↑ ↓ 0 P 0 0	11111101		FD	3 5
			10110110		B6	
			← d →			
or .a,n	.a←.a∨n	↑ ↓ 0 P 0 0	11110110		F6	2 2
			← n →			
xor .a,r	.a←.a∨r	↑ ↓ 0 P 0 0	10101	r		1 1
xor .a,p	.a←.a∨p	↑ ↓ 0 P 0 0	11011101		DD	2 2
			10101	p		
xor .a,q	.a←.a∨q	↑ ↓ 0 P 0 0	11111101		FD	2 2
			10101	q		
xor .a,[.hl]	.a←.a∨[.hl]	↑ ↓ 0 P 0 0	10101110		AE	1 2
xor .a,[.ix+d]	.a←.a∨[.ix+d]	↑ ↓ 0 P 0 0	11011101		DD	3 5
			10101110		AE	
			← d →			
xor .a,[.iy+d]	.a←.a∨[.iy+d]	↑ ↓ 0 P 0 0	11111101		FD	3 5
			10101110		AE	
			← d →			
xor .a,n	.a←.a∨n	↑ ↓ 0 P 0 0	11101110		EE	2 2
			← n →			

A.13 ビット操作命令

ニーモニック	命令動作	flags S Z H P _N N C	オペコード		B	C
			76543210	Hex		
bit b,r	$z \leftarrow \text{NOT } r_{\{b\}}$? ↓ 1 ? 0 •	11001011 01 b r	CB	2	2
bit b,[.hl]	$z \leftarrow \text{NOT } [.hl]_{\{b\}}$? ↓ 1 ? 0 •	11001011 01 b 110	CB	2	3
bit b,[.ix+d]	$z \leftarrow \text{NOT } [.ix+d]_{\{b\}}$? ↓ 1 ? 0 •	11011101 11001011 ← d → 01 b 110	DD CB	4	5
bit b,[.iy+d]	$z \leftarrow \text{NOT } [.iy+d]_{\{b\}}$? ↓ 1 ? 0 •	11111101 11001011 ← d → 01 b 110	FD CB	4	5
set b,r	$r_{\{b\}} \leftarrow 1$	• • • • •	11001011 11 b r	CB	2	2
set b,[.hl]	$[.hl]_{\{b\}} \leftarrow 1$	• • • • •	11001011 11 b 110	CB	2	5
set b,[.ix+d]	$[.ix+d]_{\{b\}} \leftarrow 1$	• • • • •	11011101 11001011 ← d → 11 b 110	DD CB	4	7
set b,[.iy+d]	$[.iy+d]_{\{b\}} \leftarrow 1$	• • • • •	11111101 11001011 ← d → 11 b 110	FD CB	4	7
clr b,r	$r_{\{b\}} \leftarrow 0$	• • • • •	11001011 10 b r	CB	2	2
clr b,[.hl]	$[.hl]_{\{b\}} \leftarrow 0$	• • • • •	11001011 10 b 110	CB	2	5
clr b,[.ix+d]	$[.ix+d]_{\{b\}} \leftarrow 0$	• • • • •	11011101 11001011 ← d → 10 b 110	DD CB	4	7
clr b,[.iy+d]	$[.iy+d]_{\{b\}} \leftarrow 0$	• • • • •	11111101 11001011 ← d → 10 b 110	FD CB	4	7

A.14 ローテイト命令

ニーモニック	命令動作	flags S Z H P _N N C	オペコード	
			76543210	Hex BC
rola	C←.a{7};.a←.a*2;.a _{0} ←C	• • 0 • 0 ↑	00000111	07 1 1
rora	C←.a _{0} ;.a←.a/2;.a _{7} ←C	• • 0 • 0 ↑	00001111	0F 1 1
rolca	tmp←C;C←.a _{7} ;.a←.a*2;.a _{0} ←tmp	• • 0 • 0 ↑	00010111	17 1 1
rorca	tmp←C;C←.a _{0} ;.a←.a/2;.a _{7} ←tmp	• • 0 • 0 ↑	00011111	1F 1 1
rol r	C←r _{7} r←r*2;r _{0} ←C	↑ ↑ 0 P 0 ↑ 00000 r	11001011 00000	CB 2 2
rol [.hl]	c←[.hl] _{7} [.hl]←[.hl]*2;[.hl] _{0} ←C	↑ ↑ 0 P 0 ↑ 00000110	11001011 06	CB 2 5
rol [.ix+d]	c←[.ix+d] _{7} [.ix+d]←[.ix+d]*2 [.ix+d] _{0} ←C	↑ ↑ 0 P 0 ↑ 11001011 ← d → 00000110	11011101 CB 06	DD 4 7
rol [.iy+d]	c←[.iy+d] _{7} [.iy+d]←[.iy+d]*2 [.iy+d] _{0} ←C	↑ ↑ 0 P 0 ↑ 11001011 ← d → 00000110	11111101 CB 06	FD 4 7
ror r	C←r _{0} r←r/2;r _{7} ←C	↑ ↑ 0 P 0 ↑ 00001 r	11001011 00001	CB 2 2
ror [.hl]	c←[.hl] _{0} [.hl]←[.hl]/2;[.hl] _{7} ←C	↑ ↑ 0 P 0 ↑ 00001110	11001011 0E	CB 2 5
ror [.ix+d]	c←[.ix+d] _{0} [.ix+d]←[.ix+d]/2 [.ix+d] _{7} ←C	↑ ↑ 0 P 0 ↑ 11001011 ← d → 00001110	11011101 CB 0E	DD 4 7
ror [.iy+d]	c←[.iy+d] _{0} [.iy+d]←[.iy+d]/2 [.iy+d] _{7} ←C	↑ ↑ 0 P 0 ↑ 11001011 ← d → 00001110	11111101 CB 0E	FD 4 7

shra [d]

shra [.hl]

shl 命令と shla 命令

ニーモニック	命令動作	flags S Z H P _N NC	オペコード		B	C
			76543210	Hex		
rolc r	tmp←C; C←r{7} r←r*2; r _{0} ←tmp	↓ ↓ 0 P 0 ↑ 00010 r	11001011 00010	CB	2	2
rolc [.hl]	tmp←C; C←[.hl]{7} [.hl]←[.hl]*2; [.hl] _{0} ←tmp	↓ ↓ 0 P 0 ↑ 00010 110	11001011 16	CB	2	5
rolc [.ix+d]	tmp←C C←[.ix+d] _{7} [.ix+d]←[.ix+d]*2 [.ix+d] _{0} ←tmp	↓ ↓ 0 P 0 ↑ 11001011 ← d → 00010 110	11011101 CB 16	DD CB	4	7
rolc [.iy+d]	tmp←C C←[.iy+d] _{7} [.iy+d]←[.iy+d]*2 [.iy+d] _{0} ←tmp	↓ ↓ 0 P 0 ↑ 11001011 ← d → 00010 110	11111101 CB 16	FD CB	4	7
rorc r	tmp←C; C←r _{0} r←r/2; r _{7} ←tmp	↓ ↓ 0 P 0 ↑ 00011 r	11001011 00011	CB	2	2
rorc [.hl]	tmp←C; C←[.hl] _{0} [.hl]←[.hl]/2; [.hl] _{7} ←tmp	↓ ↓ 0 P 0 ↑ 00011 110	11001011 1E	CB	2	5
rorc [.ix+d]	tmp←C C←[.ix+d] _{0} [.ix+d]←[.ix+d]/2 [.ix+d] _{7} ←tmp	↓ ↓ 0 P 0 ↑ 11001011 ← d → 00011 110	11011101 CB 1E	DD CB	4	7
rorc [.iy+d]	tmp←C C←[.iy+d] _{0} [.iy+d]←[.iy+d]/2 [.iy+d] _{7} ←tmp	↓ ↓ 0 P 0 ↑ 00011 110	11111101 CB 1E	FD CB	4	7
rol4 [.hl]	tmp←.a _{0..3} ; .a _{0..3} ←[.hl] _{4..7} [.hl] _{4..7} ←[.hl] _{0..3} ; [.hl] _{0..3} ←tmp	↓ ↓ 0 P 0 • 11101111	11101101 11101111	ED 6F	2	5
ror4 [.hl]	tmp←.a _{0..3} ; .a _{0..3} ←[.hl] _{0..3} [.hl] _{0..3} ←[.hl] _{4..7} ; [.hl] _{4..7} ←tmp	↓ ↓ 0 P 0 • 11100111	11101101 11100111	ED 67	2	5

A.15 シフト命令

ニーモニック	命令動作	flags S Z H P N C	オペコード 76543210 [Hex]	B	C
shl r	$C \leftarrow r_{\{7\}}$	$\uparrow \downarrow 0 P 0 \uparrow$	11001011	CB	2
shla	$r \leftarrow r * 2$	$00100 r$			
shl [.hl]	$C \leftarrow [.hl]_{\{7\}}$	$\uparrow \downarrow 0 P 0 \uparrow$	11001011	CB	2
shla	$[.hl] \leftarrow [.hl] * 2$	00100110	26		
shl [.ix+d]	$C \leftarrow [.ix+d]_{\{7\}}$	$\uparrow \downarrow 0 P 0 \uparrow$	11011101	DD	4
shla	$[.ix+d] \leftarrow [.ix+d] * 2$	11001011 $\leftarrow d \rightarrow$ 00100110	CB 26		
shl [.iy+d]	$C \leftarrow [.iy+d]_{\{7\}}$	$\uparrow \downarrow 0 P 0 \uparrow$	11111101	FD	4
shla	$[.iy+d] \leftarrow [.iy+d] * 2$	11001011 $\leftarrow d \rightarrow$ 00100110	CB 26		
shr r	$C \leftarrow r_{\{0\}}$ $r \leftarrow r / 2$	$\uparrow \downarrow 0 P 0 \uparrow$	11001011	CB	2
shr [.hl]	$C \leftarrow [.hl]_{\{0\}}$ $[.hl] \leftarrow [.hl] / 2$	$00111 r$			
shr [.ix+d]	$C \leftarrow [.ix+d]_{\{0\}}$ $[.ix+d] \leftarrow [.ix+d] / 2$	$\uparrow \downarrow 0 P 0 \uparrow$	11011101	DD	4
		11001011 $\leftarrow d \rightarrow$ 00111110	CB 3E		
shr [.iy+d]	$C \leftarrow [.iy+d]_{\{0\}}$ $[.iy+d] \leftarrow [.iy+d] / 2$	$\uparrow \downarrow 0 P 0 \uparrow$	11111101	FD	4
		11001011 $\leftarrow d \rightarrow$ 00111110	CB 3E		
shra r	$tmp \leftarrow r_{\{7\}}; C \leftarrow r_{\{0\}}$ $r \leftarrow r / 2; r_{\{7\}} \leftarrow tmp$	$\uparrow \downarrow 0 P 0 \uparrow$	11001011	CB	2
shra [.hl]	$tmp \leftarrow [.hl]_{\{7\}}; C \leftarrow [.hl]_{\{0\}}$ $[.hl] \leftarrow [.hl] / 2; [.hl]_{\{7\}} \leftarrow tmp$	$00101 r$			
shra [.ix+d]	$tmp \leftarrow [.ix+d]_{\{7\}}$ $C \leftarrow [.ix+d]_{\{0\}}$ $[.ix+d] \leftarrow [.ix+d] / 2$ $[.ix+d]_{\{7\}} \leftarrow tmp$	$\uparrow \downarrow 0 P 0 \uparrow$	11011101	DD	4
		11001011 $\leftarrow d \rightarrow$ 00101110	CB 2E		
shra [.iy+d]	$tmp \leftarrow [.iy+d]_{\{7\}}$ $C \leftarrow [.iy+d]_{\{0\}}$ $[.iy+d] \leftarrow [.iy+d] / 2$ $[.iy+d]_{\{7\}} \leftarrow tmp$	$\uparrow \downarrow 0 P 0 \uparrow$	11111101	FD	4
		11001011 $\leftarrow d \rightarrow$ 00101110	CB 2E		

shl 命令と shla 命令はまったく同じものなのでオペランドは同一

call m,nn	$[sp-2] \leftarrow pc; [sp-1] \leftarrow pc$
	$sp \leftarrow sp - 2; pc \leftarrow nn$

A.16 分岐命令

ニーモニック	命令動作	flags S Z H P _N N C	オペコード		B	C
			76543210	Hex		
br nn	.pc←nn	• • • • •	11000011	C3	3	3
			← nn _l →			
			← nn _h →			
bnz nn	if z=0 .pc←nn	• • • • •	11000010	C2	3	3
			← nn _l →			
			← nn _h →			
bz nn	if z=1 .pc←nn	• • • • •	11001010	CA	3	3
			← nn _l →			
			← nn _h →			
bnc nn	if c=0 .pc←nn	• • • • •	11010010	D2	3	3
			← nn _l →			
			← nn _h →			
bc nn	if c=1 .pc←nn	• • • • •	11011010	DA	3	3
			← nn _l →			
			← nn _h →			
bpo nn	if P _N =0 .pc←nn	• • • • •	11100010	E2	3	3
			← nn _l →			
			← nn _h →			
bpe nn	if P _N =1 .pc←nn	• • • • •	11101010	EA	3	3
			← nn _l →			
			← nn _h →			
bp nn	if s=0 .pc←nn	• • • • •	11110010	F2	3	3
			← nn _l →			
			← nn _h →			
bm nn	if s=1 .pc←nn	• • • • •	11111010	FA	3	3
			← nn _l →			
			← nn _h →			
br [hl]	.pc←.hl	• • • • •	11101001	E9	1	1
br [ix]	.pc←.ix	• • • • •	11011101	DD	2	2
			11101001	E9		
br [iy]	.pc←.iy	• • • • •	11111101	FD	2	2
			11101001	E9		

ニーモニック	命令動作	flags S Z H P _V N C	オペコード		B	C
			76543210	[Hex]		
short br e	.pc←.pc+e	• • • • •	00011000 ← e-2 →	18	2	3
short bnz e	if z=0 .pc←.pc+e	• • • • •	00100000 ← e-2 →	20	2	2
short bz e	if z=1 .pc←.pc+e	• • • • •	00101000 ← e-2 →	28	2	2
short bnc e	if c=0 .pc←.pc+e	• • • • •	00110000 ← e-2 →	30	2	2
short bc e	if c=1 .pc←.pc+e	• • • • •	00111000 ← e-2 →	38	2	2
dbnz e	.b←.b-1;if .b≠0 .pc←.pc+e	• • • • •	00010000 ← e-2 →	10	2	2

A.17 コール命令

ニーモニック	命令動作	flags S Z H P _V N C	オペコード		B	C
			76543210	[Hex]		
call nn	[.sp-2]←.pcl;[.sp-1]←.pch .sp←.sp-2;.pc←nn	• • • • •	11001101 ← nn _l → ← nn _h →	CD	3	5
call nz,nn	if z=0 .sp-2]←.pcl;[.sp-1]←.pch .sp←.sp-2;.pc←nn	• • • • •	11000100 ← nn _l → ← nn _h →	C4	3	3
call z,nn	if z=1 .sp-2]←.pcl;[.sp-1]←.pch .sp←.sp-2;.pc←nn	• • • • •	11001100 ← nn _l → ← nn _h →	CC	3	3
call nc,nn	if c=0 .sp-2]←.pcl;[.sp-1]←.pch .sp←.sp-2;.pc←nn	• • • • •	11010100 ← nn _l → ← nn _h →	D4	3	3
call c,nn	if c=1 .sp-2]←.pcl;[.sp-1]←.pch .sp←.sp-2;.pc←nn	• • • • •	11011100 ← nn _l → ← nn _h →	DC	3	3
call po,nn	if P _V =0 .sp-2]←.pcl;[.sp-1]←.pch .sp←.sp-2;.pc←nn	• • • • •	11100100 ← nn _l → ← nn _h →	E4	3	3
call pe,nn	if P _V =1 .sp-2]←.pcl;[.sp-1]←.pch .sp←.sp-2;.pc←nn	• • • • •	11101100 ← nn _l → ← nn _h →	EC	3	3
call p,nn	if s=0 .sp-2]←.pcl;[.sp-1]←.pch .sp←.sp-2;.pc←nn	• • • • •	11110100 ← nn _l → ← nn _h →	F4	3	3
call m,nn	if s=1 .sp-2]←.pcl;[.sp-1]←.pch .sp←.sp-2;.pc←nn	• • • • •	11111100 ← nn _l → ← nn _h →	FC	3	3

ニーモニック	命令動作	flags S Z H P _N N C	オペコード		B	C
			76	543210	Hex	
ret	.pc _l ←[.sp];.pc _h ←[.sp+1];.sp←.sp+2	• • • • •	11	001001	C9	1 3
ret nz	if z=0 .pc _l ←[.sp];.pc _h ←[.sp+1];.sp←.sp+2	• • • • •	11	000000	C0	1 1 3
ret z	if z=1 .pc _l ←[.sp];.pc _h ←[.sp+1];.sp←.sp+2	• • • • •	11	001000	C8	1 1 3
ret nc	if c=0 .pc _l ←[.sp];.pc _h ←[.sp+1];.sp←.sp+2	• • • • •	11	010000	D0	1 1 3
ret c	if c=1 .pc _l ←[.sp];.pc _h ←[.sp+1];.sp←.sp+2	• • • • •	11	011000	D8	1 1 3
ret po	if P _N =0 .pc _l ←[.sp];.pc _h ←[.sp+1];.sp←.sp+2	• • • • •	11	100000	E0	1 1 3
ret pe	if P _N =1 .pc _l ←[.sp];.pc _h ←[.sp+1];.sp←.sp+2	• • • • •	11	101000	E8	1 1 3
ret p	if s=0 .pc _l ←[.sp];.pc _h ←[.sp+1];.sp←.sp+2	• • • • •	11	110000	F0	1 1 3
ret m	if s=1 .pc _l ←[.sp];.pc _h ←[.sp+1];.sp←.sp+2	• • • • •	11	111000	F8	1 1 3
reti	interrupt return	• • • • •	11	101101 01001101	ED 4D	2 5
retn	Non Maskable Interrupt return	• • • • •	11	101101 01000101	ED 45	2 5
brk k	[.sp-2]←.pc _l ;[.sp-1]←.pc _h .sp←.sp-2;.pc _l ←k;.pc _h ←0	• • • • •	11	k/8 111	SD	1 4

A.18 入出力命令

ニーモニック	命令動作	flags S Z H P V N C	オペコード		B	C
			76543210	Hex		
in .a,[n]	.a←[n]	• • • • •	11011011 ← n →	DB	2	3
in r,[.c]	r← [.c]	↑ ↓ 0 P 0 •	11101101 01 r 000	ED	2	3
in .f,[.c]	[.c]	↑ ↓ 0 P 0 •	11101101 01110000	ED 70	2	3
in [.hl++],[.c]	[.hl]← [.c]; b← .b-1 .hl← .hl+1	? ↓ ? ? 1 • *1	11101101 10100010	ED A2	2	4
in [.hl--],[.c]	[.hl]← [.c]; b← .b-1 .hl← .hl-1	? ↓ ? ? 1 • *1	11101101 10101010	ED AA	2	4
inm [.hl++],[.c]	repeat; [.hl]← [.c]; b← .b-1 .hl← .hl+1; until .b=0	? 1 ? ? 1 •	11101101 10110010	ED B2	2	4
inm [.hl--],[.c]	repeat; [.hl]← [.c]; b← .b-1 .hl← .hl-1; until .b=0	? 1 ? ? 1 •	11101101 10111010	ED BA	2	4
out [n],.a	[n]← .a	• • • • •	11010011 ← n →	D3	2	3
out [.c],r	[.c]← r	• • • • •	11101101 01 r 001	ED	2	3
out [.c],[.hl++]	[.c]← [.hl]; b← .b-1 .hl← .hl+1	? ↓ ? ? 1 • *1	11101101 10100011	ED A3	2	4
out [.c],[.hl--]	[.c]← [.hl]; b← .b-1 .hl← .hl-1	? ↓ ? ? 1 • *1	11101101 10101011	ED AB	2	4
outm [.c],[.hl++]	repeat; [.c]← [.hl]; b← .b-1 .hl← .hl+1; until .b=0	? 1 ? ? 1 •	11101101 10110011	ED B3	2	4
outm [.c],[.hl--]	repeat; [.c]← [.hl]; b← .b-1 .hl← .hl-1; until .b=0	? 1 ? ? 1 •	11101101 10111011	ED BB	2	4

*1.b-1=0 のとき 1、他は 0

in .f,[.c] は.c レジスターが示すポートの内容によってフラグを変えるだけで、その内容はどこにも格納されない



A.19 CPU 制御命令

ニーモニック	命令動作	flags S Z H P V N C	オペコード		B	C
			76	543210[Hex]		
adj .a	adjust to decimal	↑↑↑P •↑	00100111	27	1	1
not .a	.a←NOT .a	••1•1•	00101111	2F	1	1
neg .a	.a←NOT .a+1	↑↑↑V1↑	11101101 01000100	ED 44	2	2
notc	c←NOT c	••?•0↑	00111111	3F	1	1
setc	c←1	••0•01	00110111	37	1	1
nop	NO operation	••••••	00000000	00	1	1
halt	HALT	••••••	01110110	76	1	2
di	IFF←0	••••••	11110011	F3	1	2
ei	IFF←1	••••••	11111011	FB	1	1
im 0	interrupt mode 0	••••••	11101101 01000110	ED 46	2	3
im 1	interrupt mode 1	••••••	11101101 01010110	ED 56	2	3
im 2	interrupt mode 2	••••••	11101101 01011110	ED 5E	2	3

索引

ア

I/O ポート	57
アクセスタイム	22
アドレス	48
アドレスバス	48
インターレース	111
ウェイト機能	95
ADSR	133
SECAM	111
SRAM	50
NTSC	110
FM-BIOS	139
MSX-Engine	17
MSX-JE	72
MSX-MUSIC	131
MSX の種類	62
エンベロープ	133
OPLL YM2413	131
OPLL ドライバー	156
オペレーター	131
音階ノイズ	133

カ

海外への輸出用に作られた MSX	62
拡張 BIOS	62
漢字 ROM 拡張	21

漢字グラフィックモード	77
漢字テキストモード	77
漢字ドライバー	74
完全平均律	134
輝度	99
キャリアー・オペレーター	137
キロ (K)	48
矩形(くけい)波	132
コマンドレジスター	92
コントロールレジスター	92

サ

サブ ROM	52
サンプリングシンセサイザー	130
CPU	48
色相	99
JIS コード	74
システムタイマー	20, 27
システムワークエリア	61
シフト JIS コード	74
16 進数	48
主記憶	50
垂直帰線割り込み	114
水平解像度	111
ステータスレジスター	92
スーパーインポーズ	113
スロット拡張器	55
正弦波	132
全角文字	74
ソフトウェアスタック	78

タ

タイマー割り込み	114
単漢字変換	72
TAND	106
D/A コンバーター	20

DRAM のページアクセス	24
DRAM モード	22
TC9769	17
ディスクがあるかどうか	62
ディスクワークエリア	61
データバス	48
同期信号	110

ナ

2進数	48
のこぎり波	133
ノン・インターレース	111

ハ

BIOS	57
バイト (byte)	48
PAL	111
半角文字	74
番地	48
PSG	130
ビット (bit)	48
ビットイメージ印字	79
ビデオ RAM(VRAM)	50
ビデオ RAM 容量	62
ビデオ・デジタイズ	113
VDP コマンド	95
VDP のモード	82
フック	121
プログラマブル・サウンド・ジェネレーター	130
Basic Input Output System	57
ページ	51
ポーズキー制御	20

マ

マイクロソフト漢字コード	74
マルチ・スキャン・モニター	111
未定義命令	98

メイン ROM.....	52
メインメモリー	50
メモリーマッパー	22
モアレ	113
モジュレーター・オペレーター	137

ヤ

USR 関数.....	119
-------------	-----

ラ

RAM.....	50
リズム音	137
リセットステータス.....	22
連文節変換	72
ROM.....	50

列

参考文献

- [1] アスキー・マイクロソフト FE 本部、日本楽器製造株式会社、“V9938 MSX-VIDEO テクニカルデータブック”、アスキー、1985 年(絶版)
- [2] 株式会社アスキー、“V9958 仕様書”、非売品、1988 年
- [3] アスキー・マイクロソフト FE 監修、“MSX2 テクニカル・ハンドブック”、アスキー、1986 年
- [4] 杉谷成一、“MSX2+パワフル活用法”、アスキー、1989 年
- [5] 株式会社アスキー、“MSX-Datapack”、アスキー、1991 年

■著者略歴

いしかわなおた
石川直太

横浜国立大学卒業後、アスキーに入社。MSX の開発に携わる。その後、東京理科大学理学部第二部数学科、同大学院理学研究科修士課程を卒業。現在は慶應義塾大学大学院理工学研究科で、後期博士課程に在学中。MSX マガジンに連載された歴代のテクニカル記事の筆者でもある。第 1 種情報処理技術者、オンライン情報処理技術者。蠍座、B 型。naota@slab.sfc.keio.ac.jp

52
50
22
113
117
119

精文書

MSX turbo R テクニカル・ハンドブック

1991年 7月31日 初版発行

1992年 6月15日 第3刷発行

定価2,500円(本体2,427円)

著 者 石川直太

発行者 藤井章生

編集人 塩崎剛三

発行所 株式会社アスキー

〒107-24 東京都港区南青山6-11-1スリーエフ南青山ビル

振替 東京4-161144

大代表 (03)3486-7111

出版営業部 (03)3486-1977 (ダイヤルイン)

本書は著作権法上の保護を受けています。本書の一部あるいは全部について(ソフトウェア及びプログラムを含む)、株式会社アスキーから文書による承諾を得ずに、いかなる方法においても無断で複写、複製することは禁じられています。

制 作 東京書籍印刷株式会社

印 刷 東京書籍印刷株式会社

編 集 MSX マガジン編集部

ISBN4-7561-0621-8

Printed in Japan







内容一覧

- MSX turbo R のシステム構成
- R800 のプログラミング・テクニック
- BASIC コマンドの追加、削除、変更一覧
- 拡張 BIOS の追加、削除、変更一覧
- PCM 録音、再生機能活用法
- MSX2+とMSX turbo R のスロット
- 漢字 BASIC 概要
- V9958VDP 仕様書
- 自然画表示モード詳細
- 走査線割り込みの実践
- MSX-MUSIC のコントロール
- BASIC と BIOS の音色データ一覧
- R800CPU インストラクションコード表

ISBN4-7561-0621-8 C3055 P2500E

定価2,500円(本体2,427円)

MSX turbo R テクニカルハンドブック