

MSX マガジン編集部監修

MSX turbo R Technical Hand Book

MSX turbo R テクニカル・ハンドブック

アスキー出版局



-MSX turbo R-

Technical
Hand Book

アスキー出版



MSX turbo R Technical Hand Book

本書は、那ージと氣体と一緒に因版を除いてすべて、アスキー社「日本語 MSX」によっ
て編集処理を行いました。msdos.sty の作者である iohi@cts.dnp.co.jp さんと、出版技術
のみなさんに感謝します。また、那ージのイラストを快く引き受けてくれた、めるへん
かーさん、どうもありがとうございます。
なお、書籍が膨大なものとなってしまうため、今回は「日本語 MSX-DOS2」と、メモ
リーカードに関する記述は書いてあります。これらに関しては、近日発売予定の「日本語
MSX-DOS2 テクニカル・ハンドブック(仮題)」で、解説する予定です。

- ・テキストの一部で、MSX-DOS1 と MSX-DOS2 の二種類の OS を混在して記述されています。
・テキストの一部で、MS-DOS 1.3 と MS-DOS 2.0 の二種類の OS を混在して記述されています。
・OS が 8-Bit と 16-Bit の二種類で記述されています。
・テキストの一部で、Windows 95/98/ME/XP と Macintosh の二種類の OS を混在して記述されています。
・テキストの一部で、Windows 95/98/ME/XP と Macintosh の二種類の OS を混在して記述されています。

アスキー出版局

-MSX Turbo R-

Technical

Hand Book

- MSX、MSX-DOS は、アスキーの商標です。
- MS-DOS は、米国マイクロソフト社の商標です。
- OS-9 は、米国マイクロウエア・システムズ社と米国モトローラ社の商標です。
- TeX は、American Mathematical Society の商標です。
- MicroTeX は、米国 Addison-Wesley Publishing 社の商標です。
- その他本書で使用する CPU 名、システム名、製品名等は、一般に各開発メーカーの商標です。なお、本文中では TM、®マークは明記していません。

MSX turbo R の世界は、これまでの MSX と比べて、CPU の処理速度が 10 倍以上速いなど、多くの機能が追加された高性能なパソコンです。しかし、その一方で、MSX との互換性を保つために、MSX 用のソフトウェアやデータをそのまま利用するには、多くの手間が必要になります。

はじめに

本書は、MSX turbo R の内部構造や、その特徴的な機能について解説するための技術書です。

MSX turbo R の世界によくこそ。本書は、高速 CPU と大容量メモリーを得て、見逃せるほどパワフルになった MSX パーソナルコンピューターを、極限まで使いこなすために必要な下記のような内部情報を詳しく解説したものです。

1. 内部を 16 ビット化し、これまでの MSX と比較して 10 倍以上の処理速度を発揮する高速 CPU、R800 の性能をぎりぎりまで引き出すテクニック。

本書は、扉ページと奥付と一部の図版を除いてすべて、アスキー製『日本語 TeX』によって組版処理を行ないました。msdos.sty の作者である ishii@cts.dnp.co.jp さんと、出版技術部のみなさんに感謝します。また、扉ページのイラストを快く引き受けてくれた、めるへんめーかーさん、どうもありがとうございます。

なお、書籍が膨大なものとなってしまうため、今回は『日本語 MSX-DOS2』と、メモリーマッパーに関する記述は省いてあります。これらに関しては、近日発売予定の『日本語 MSX-DOS2 テクニカル・ハンドブック(仮題)』で、解説する予定です。

2. 画面表示でテクニックを発揮するために、VDP を使いこなし方。

MSX turbo R は、従来機のアーキテクチャーを大きく変えることなく、CPU を 16 ビット化して飛躍的な高性能を実現した、はじめてのパーソナルコンピューターです。

ほかの機種では、8 ビットから 16 ビットに移行するときにアーキテクチャーをまったく変更してしまったため、8 ビットのマシンで多くの人々によって開発されたソフトウェアやノウハウは、すべて捨て去られる結果となってしまいました。

私たちは、MSX の性能を上げるために CPU を 16 ビットとすることは必要だが、そのために MSX のために開発されたソフトウェアやハードウェアの資産、またユーザーのノウハウを捨て去るようなことは、してはならないと考えました。このことを実現するためには、新しい MSX のために Z80 に上位互換な CPU が必要と考え、R800 を開発しました。そして、これまでの MSX との完全な互換性を実現するためには、従来の Z80 も新開発の R800 と共に搭載した、MSX turbo R を開発しました。

MSX turbo R では、このように従来の MSX との上位互換性が理想的に保たれています。したがってユーザーは、今までに積み上げられたソフトウェアの資産を

・本誌は「XeTeX」翻本日¹ 読一チスで、アヘナリバ頃を題材の第一と特典ページへ載。お書本
講義選出。さあち qj.oo.qub.stc@lindai るおう皆君の yui.soham。次にまじか書き版既選出ア
入るやう。次にアセテートを用いたライスモトのセーブ。次に。すまじ機器が入り余るの時
、そなれりあきらめ、ある一枚一枚
テキスト「XeTeX」翻本日¹ 読回今、改めてアセテート大顔映前書。ほか
翻本日¹ の宝文館新刊日記、おアドバイス付。すまじあくび書き風流るす開拓ハヤマード
。すか宝手するす銀紙、クレジット(脚注) ベビーフルカバ・バタニヤテ
MSX-DOS3 です。

- ・ MSX、MSX-DOS は、アスキーの商標です。
- ・ MS-DOS は、米国マイクロソフト社の商標です。
- ・ OS-9 は、米国マイクロウェア・システムズ社と米国モトローラ社の商標です。
- ・ TeX は、American Mathematical Society の商標です。
- ・ MicroType は、米国 Addison-Wesley Publishing 社の商標です。
- ・ その他本書で使用する CPU 名、システム名、製品名等は、一般に各開発メーカーの商
標です。なお、本文中では TM、®マークは明記していません。

まごむが人形を多土向の誰封のよ御神。今やれるす言実ア且 edint XRM まもやう
のきのすまけこ。よ羅歌を愛海コロスるす發開きてエライヤー。アモウアは
るす根叶うやハヤヘの子落とす羅歌ウ書本。ハモモウアはるこるす用部まもやうち
す明鏡ウズベマードウハイスヒト姓チ報。」出をほき強封のくモタコモ。ウシコ
トシテ。」。さくさくするよ羅歌をさるす言実アムスベ

はじめに

翻見不山 番體辭林・研詩詩品錄・草體業事ムマスベ

MSX turbo R の世界にようこと。本書は、高速 CPU と大容量メモリーを得て、見違えるほどパワフルになった MSX パーソナルコンピューターを、極限まで使いこなすために必要な下記のような内部情報を詳しく解説したものです。

1. 内部を 16 ビット化し、これまでの MSX と比較して 10 倍以上の処理速度を発揮する高速 CPU、R800 の性能をぎりぎりまで引き出すテクニック。
2. MSX turbo R に標準搭載された PCM 音源と、FM 音源を使いこなすための情報とテクニック。
3. MSX を使いこなすために必須の SLOT 機構のしくみと、取扱方法。
4. 日本語を取り扱うソフトウェアの開発に必要な、漢字 BASIC の仕組み。
5. 画面表示でテクニックを発揮するための、VDP の使いこなし方法。

MSX turbo R は、従来機のアーキテクチャーを大きく変えることなく、CPU を 16 ビット化して飛躍的な高性能を実現した、はじめてのパーソナルコンピューターです。

ほかの機種では、8 ビットから 16 ビットに移行するときにアーキテクチャーをまったく変更してしまったため、8 ビットのマシンで多くの人々によって開発されたソフトウェアやノウハウは、すべて捨て去られる結果となってしまいました。

私たちは、MSX の性能を上げるために CPU を 16 ビットとすることは必要だが、そのために MSX のために開発されたソフトウェアやハードウェアの資産、またユーザーのノウハウを捨て去るようなことは、してはならないと考えました。このことを実現するためには、新しい MSX のために Z80 に上位互換な CPU が必要と考え、R800 を開発しました。そして、これまでの MSX との完全な互換性を実現するために、従来の Z80 も新開発の R800 と共に搭載した、MSX turbo R を開発しました。

MSX turbo R では、このように従来の MSX との上位互換性が理想的に保たれています。したがってユーザーは、今までに積み上げられたソフトウェアの資産を

そのまま MSX turbo R で実行するだけで、何倍もの性能の向上を手に入れることができます¹。また、ソフトウェアを開発するために必要な知識も、これまでのものをそのまま活用することができますが、本書で解説する若干のノウハウを利用することで、さらにマシンの性能を引き出し、群を抜くコストパフォーマンスを発揮するシステムを実現することが可能となるでしょう。

システム事業部第1製品統括部・統括部長 山下良藏

MSX turbo R の最大の特徴は、CPU 計算能力の向上と、メモリ容量の拡張である。MSX turbo R では、Z80 CPU の計算能力を約 10 倍以上向上させた上で、MSX 本体のメモリ容量を 16 MB から 32 MB へと大幅に増加した。また、MSX turbo R の音楽機能では、PCM フィルター機能を搭載し、音質を向上させた。さらに、MSX turbo R の映像機能では、高解像度のビデオ出力機能を搭載し、映像品質を向上させた。これらの機能により、MSX turbo R は、MSX 本体よりも多くの機能を備えた高性能な家庭用ゲーム機として、多くのユーザーに支持されるようになった。

¹ MSX 用の市販ソフトウェアは、R800 で実行すると速度が速くなり過ぎて互換性がとれなくなるので、自動的に Z80 が動作するため高速にならない場合があります。

81.4.4	V9958 ハードウェア構成(変更部分)とおもに MSX CPU の仕様	90
81.4.5	V9958 と MSX2+ の異なる部分と内部構成の比較	91
02	4.5.1. スクリーンドットロード機能による初期状態の表示	91
02	4.5.2. VDP のレスベイクル機能と MSX2+ の X8M モード	92
52	V9958 の読み書きオペレータ機能による初期状態の X8M モード	95

目 次

1	MSX turbo R	15
1.1	MSX turbo R のハードウェア	16
1.1.1	MSX turbo R の特徴はこれだ!	16
1.1.2	MSX turbo R のシステム構成	16
1.1.3	エレガントな CPU の切り替え	18
1.1.4	何でも詰め込む MSX turbo R の ROM 構成	18
1.1.5	速さを調節するシステムタイマー	19
1.1.6	MSX turbo R の I/O ポート	20
1.1.7	速さを生かすための DRAM モード	22
1.1.8	R800 の特徴はこれだ!	23
1.1.9	R800 のすべて	23
1.2	MSX turbo R 活用法	27
1.2.1	R800 の速さを生かすプログラミング	27
1.2.2	R800 を使う上での注意事項と問題点	27
1.2.3	追加された BIOS とその機能説明	28
1.2.4	変更および削除された BIOS について	31
1.2.5	アプリケーション開発に関する注意点	32
1.2.6	CPU を切り替えるプログラムの例	33
1.3	PCM 限界ギリギリ活用法	37
1.3.1	基礎編…… BASIC での使い方	37
1.3.2	PCM 関係の BASIC 命令	38
1.3.3	BEEP 音を PCM で鳴らすのだ!	39
1.3.4	上級編……マシン語で PCM を!	41
2	SLOT	47
2.1	スロットって何だ?	48

2.1.1	CPU とメモリーはどうつながってるの	48
2.1.2	8 ビット CPU Z80 の内部を探る	48
2.1.3	メモリーの種類は働きによってイロイロ	50
2.1.4	MSX のスロットってどんなものなの?	50
2.1.5	MSX の拡張性の秘密はスロットにあった	52
2.1.6	こう変わった MSX2+ のスロット	53
2.1.7	スロットを拡張しちゃえ	55
2.2	スロット切り替えに挑戦	57
2.2.1	スロットを切り替えるには	57
2.2.2	スロット番号の指定方法	57
2.2.3	スロットを操作する BIOS の機能	58
2.2.4	スロット構成を知る方法	60
2.2.5	システムワークエリアを探ってみる	61
2.2.6	MSX2+ のハードウェア仕様	64
2.2.7	衝突を防ぐデバイスイネーブル	65
2.3	MSX turbo R のスロット構成	67
2.3.1	ついにスロット構成が統一されたぞ	67
3	漢字 BASIC	71
3.1	漢字 BASIC を解析	72
3.1.1	漢字 BASIC に必要なハードウェア	72
3.1.2	MSX-JE 対応のソフトウェアとは	73
3.1.3	漢字ドライバーの動作原理を解説する	73
3.1.4	JE 対応ハード&ソフト	75
3.1.5	漢字 BASIC で使える画面モードいろいろ	76
3.1.6	漢字テキストと漢字グラフィック	77
3.1.7	漢字ドライバーの正しい使い方なのだ	78
4	V9958 VDP	81
4.1	V9958 レジスター一覧	83
4.2	V9958 の新機能	85
4.2.1	水平スクロール	85
4.2.2	ウェイト	87
4.2.3	コマンド	87
4.2.4	YJK 方式の表示	87
4.3	V9958 の廃止機能	89

4.4	V9958 ハードウェア仕様(変更部分)	90
4.5	V9958 と MSX2+	91
4.5.1	スクリーンモードは全部で 12 種類	91
4.5.2	VDP のレジスターをコントロールする	92
4.5.3	V9958 のレジスター	95
4.5.4	VDP による横スクロール	95
4.5.5	何があっても裏技は使ってはいけないぞ	98
4.6	YJK 方式を解剖する	99
4.6.1	テレビ放送と YJK 方式	99
4.6.2	RGB 方式と YJK 方式のデータ構造	99
4.6.3	色見本のプログラム	101
4.6.4	必殺のロジカルオペレーションなのだ	103
4.6.5	いわゆる色化け	105
4.6.6	SCREEN 10 と 11 は何がどう違うのか	105
4.6.7	SCREEN 11 でもテロップを使うには	106
4.6.8	SCREEN 12 で文字表示をするための裏技だ	108
4.6.9	YJK 方式と VDP のレジスター	108
4.7	走査線割り込みを研究する	110
4.7.1	モニター画面を表示する仕組みは?	110
4.7.2	インターレース方式によるテレビ放送	111
4.7.3	MSX2 におけるインターレース画面	112
4.7.4	走査線割り込みの原理を探る	113
4.7.5	走査線割り込みの実例を紹介する	114
4.7.6	いよいよ実践編はりきっていこう!	116
4.7.7	走査線割り込みに使う VDP レジスター	116
4.7.8	アセンブルの方法と BASIC 部分の動作	119
4.7.9	アセンブラー部分の動作原理だ	121
4.7.10	走査線割り込みのマシン語ルーチンだ	128
5	MSX-MUSIC	129
5.1	FM 音源ってどんなもの	130
5.1.1	FM 音源へと至る電子楽器の歴史	130
5.1.2	楽器の音を分析してみよう	132
5.1.3	音程が平均律とは限らない	134
5.1.4	MSX-MUSIC を分析してみる	135
5.1.5	FM 音源を使ってリズム音に挑戦	137

5.2	FM 音源をコントロール	139
5.2.1	マシン語プログラムで音を出してみる	139
5.2.2	ライブラリーの概要を説明する	141
5.2.3	MSX-C でコンパイルしよう	149
5.3	FM 音源のデータ構造だ	150
5.3.1	FM 音源のデータを作ってみよう	150
5.3.2	打楽器音のデータを指定するには	152
5.3.3	楽器音のデータを指定してみよう	154
5.3.4	OPLL ドライバーでできないこと	156
5.3.5	音色データを追加してみよう	156
5.3.6	サンプルデータを解説する	158
5.4	FM 音源にまつわるアレコレ	160
5.4.1	パワフル活用法の内容訂正	160
5.4.2	MSX-MUSIC の音色データ一覧	162
A	R800 インストラクション表	165
A.1	インストラクション表はこうして使おう	166
A.2	8 ビット移動命令	168
A.3	16 ビット移動命令	169
A.4	交換命令	171
A.5	スタック操作命令	171
A.6	ブロック転送命令	172
A.7	ブロックサーチ命令	172
A.8	乗算命令	172
A.9	加算命令	173
A.10	減算命令	175
A.11	比較命令	176
A.12	論理演算命令	177
A.13	ビット操作命令	178
A.14	ローテイト命令	179
A.15	シフト命令	181
A.16	分岐命令	182
A.17	コール命令	183
A.18	入出力命令	185
A.19	CPU 制御命令	186

図 目 次

III	前半のみ及び階級查找	113
IV	一文字で、RGBより主張を読み取り階級查找	93
V	一文字で、VDPより出力され読み取り階級查找	93
VI	一文字で、VDPより映像書き込み画面	93
VII	み監査のポートやスレーブ等でデータ	93
81	る緊き監査の器楽千葉の瞭解ト	13
82	るみアト音楽を育むさう本基	23
83	マーロングエマくそう器楽	23
84	マーマル音器樂	23
1.1	MSX turbo R のシステム構成	17
1.2	MSX turbo R での ROM 構成の変化	19
1.3	R800 内部のブロック図	25
1.4	Z80 と R800 のメモリーアクセス方式の違い	26
2.1	Z80 CPU のメモリー	49
2.2	MSX のスロット構成 (その 1)	51
2.3	MSX のスロット構成 (その 2)	52
2.4	MSX2+のスロット構成の例 (スロット 3 のみを拡張する場合)	54
2.5	MSX2+のスロット構成の例 (スロット 0 と 3 を拡張する場合)	55
2.6	スロット番号の指定方法	58
2.7	デバイスイネーブル	65
2.8	MSX turbo R のスロット構成	68
3.1	漢字ドライバーの動作原理	75
3.2	画面モードの切り替え	78
4.1	水平スクロール (SP2=0 の場合)	85
4.2	水平スクロール (SP2=1 の場合)	86
4.3	V9958 に追加されたコントロールレジスターの機能一覧	96
4.4	2 種類の横スクロールの仕組み	97
4.5	RGB 方式画面のデータ構造	99
4.6	YJK 方式画面のデータ構造	101
4.7	混在方式画面のデータ構造	101
4.8	テレビ画面上の走査線のようす	110
4.9	インターレースモードではこうなるぞ	112
4.10	走査線割り込みの原理なのだ	114

4.11 走査線割り込みの手順	115
4.12 走査線割り込みを発生する VDP レジスター	116
4.13 走査線割り込みを検出する VDP レジスター	117
4.14 画面切り替えを制御する VDP レジスター	117
4.15 ハードウェア縦スクロールの仕組み	118
5. 楽器音データ	
5.1 4種類の電子楽器の構造を探る	131
5.2 基本となる音を分析してみる	132
5.3 楽器とシンセのエンベロープ	134
5.4 打楽器音のデータ	153
5.5 音色データ	157
5.6 OPLL のレジスター一覧	161
6. MSX CPU の音色データ	
6.1 6400 インストラクション表	160
6.2 6400 インストラクション表 (1の子) 音階イヤロスの X2M	165
6.3 6400 インストラクション表 (2の子) 音階イヤロスの XEM	166
6.4 音ビート移動命令	168
6.5 音ビート合体命令 (音階イヤロス) 間の音階イヤロスの +SXZM	169
6.6 音ビート合体命令 (音階イヤロス) 間の音階イヤロスの +SXZM	169
6.7 音階音符	171
6.8 ブラッキング操作命令	171
6.9 ブラッキング操作命令	171
6.10 ブラッキング操作命令	172
6.11 ブラッキング操作命令	172
6.12 加算命令	173
6.13 加算命令	173
6.14 減算命令	175
6.15 減算命令	175
6.16 正規命令	176
6.17 正規命令	176
6.18 正規命令	177
6.19 正規命令	177
6.20 正規命令	178
6.21 正規命令	178
6.22 正規命令	179
6.23 正規命令	179
6.24 正規命令	181
6.25 正規命令	182
6.26 正規命令	183
6.27 正規命令	185
6.28 正規命令	186
6.29 正規命令	186
6.30 正規命令	187

表 目 次

1.1	MSX turbo R の I/O マップ	21
1.2	Z80 と R800 の動作速度を比較	24
1.3	MSX turbo R で変更のあった BIOS と BASIC の一覧	32
1.4	PCM 用の I/O ポート	45
2.1	スロットに関するシステムワークエリア	61
2.2	MSX2+の I/O ポート	64
3.1	MSX-JE 内蔵ハードウェア一覧	73
3.2	漢字 BASIC の画面モード	77
3.3	漢字ドライバーが使うフック	79
4.1	VDP のモードと BASIC の画面モード	82
4.2	モードレジスター	83
4.3	コマンドレジスター	84
4.4	ステータスレジスター	84
4.5	V9958 の端子の変更	90
4.6	V9958 の直流特性	90
4.7	MSX2+の画面モード	91
4.8	VDP の I/O ポート	92
4.9	コントロールレジスターの保存場所	93
4.10	そのほかの便利なシステムワークエリア	94
4.11	MSX2+に追加、変更されたシステムワークエリア	94
4.12	0FAFCH 番地 (MODE) の詳細	94
4.13	ロジカルオペレーション	104
5.1	電子楽器の性能を比較する	131
5.2	音階と周波数の関係	134

第1章

MSX turbo R



この章は、MSX マガジン 1990 年 11 月号、1990 年 12 月号の “MSX turbo R テクニカル・アナリシス” と、“PCM 限界ギリギリ活用法” の記事を再編集したものである。

135
150
151
152

1.1 MSX turbo R のハードウェア

新開発の 16 ビット CPU “R800” を搭載したり、256 キロバイトのメイン RAM や、階層化ディレクトリーをサポートした MSX-DOS2 の標準装備など、何かと話題の多い MSX turbo R。この注目のマシンのシステム構成はどうなっているのか、その概要を紹介する。

1.1.1 MSX turbo R の特徴はこれだ！

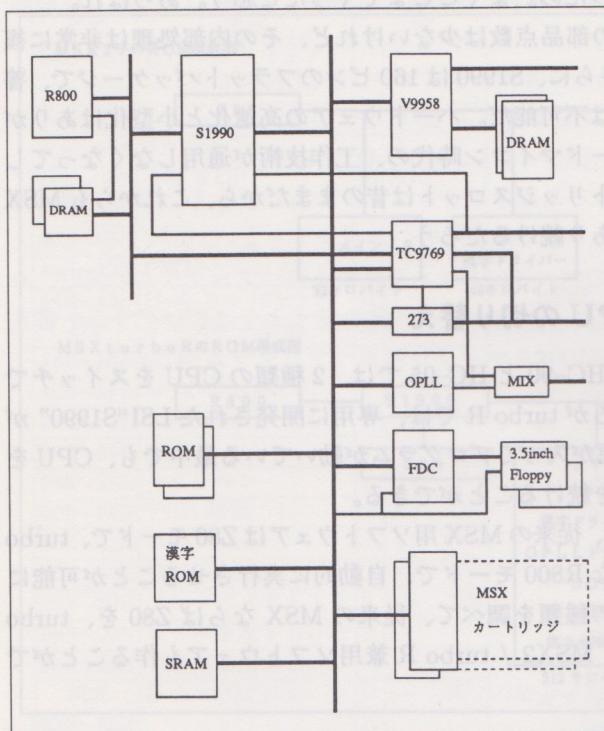
- Z80 に加え上位互換の高速 CPU “R800” を搭載することで、平均 4~5 倍、最大で 10 倍ほどのスピードを実現(対 MSX2+比)。
- MSX-DOS1 とともに、日本語 MSX-DOS2 と漢字ドライバーを搭載。MS-DOS コンパチブルな階層化ディレクトリーや、環境変数をサポート。
- メモリーマッパーに対応した、256 キロバイトのメイン RAM を標準で搭載。さらにスロット構成も標準化された。
- PCM の録音/再生機能を標準搭載。従来はオプション装備となっていた MSX-MUSIC も、標準装備されることになった。

1.1.2 MSX turbo R のシステム構成

MSX turbo R(以下 turbo R) のハードウェア構成は図 1.1 のとおり。従来の MSX と同じ “Z80” 互換 CPU と、新しく開発された “R800” CPU が含まれている。業界内の、“次の MSX にはザイログ社の Z280 か、日立の HD64180(どちらも Z80 互換の高速 CPU) が載るらしい” という噂に反して、何とアスキーが CPU を作ってしまったのだ。

これらのハードウェアの性能は、少し前の 16 ビット機に匹敵し、CPU の速さは V30(NEC が開発した 16 ビット CPU) なみだ。また漢字変換辞書を ROM に入れ、RAM とディスク容量を節約することは、MSX の伝統的な設計方針。最近のノート型パソコンの一部でも採用されている。turbo R のハードウェアを一言で評価すると、“みんなこれを目指してきた” といえるだろう。

図 1.1: MSX turbo R のシステム構成



turbo R のハードウェア構成を、細かい制御信号線を省略し、簡単に表わす。V9958 の出力はビデオ信号、TC9769(Z80)につながる線はキーボードとジョイスティック、273 の出力はプリンターポート、MIX の出力はオーディオ信号だ。

ハードウェア構成をもう少し詳しく説明していくと、まず、図中の“TC9769”は、型番から推定すると東芝製の CMOS-LSI(低消費電力のデジタル LSI の一種)。一般に“MSX-Engine”と呼ばれる、Z80 互換 CPU と、PSG 音源などをふくむ LSI らしい。以下、この本の中で“Z80”という表記が出てきた場合は、このチップを意味する。

その下の“273”は、プリンターを制御するためのバスバッファー、“OPLL”は FM 音源、“FDC”はフロッピーディスクコントローラーのことだ。また“SRAM”というのは、漢字辞書の学習結果を電源を切っても保存するためのメモリーだけれど、この SRAM と連文節変換辞書については、メーカーオプション機能となっている。

ところで、R800 とメイン RAM は、S1990 をとおして、バス(図では長い縦線)につながっている。たとえば、R800 が VDP を操作するときなどは、S1990 が信号を中継し、さらに必要に応じて R800 にウェイト信号を送って信号のタイミングを Z80 の信号のタイミングに合わせる操作を行なう。逆に、Z80 がメイン RAM を使うときは、S1990 と R800 が信号を中継し、メモリーマッピングを処理するわけだ。

turbo R が、こうした複雑な構成になった理由は、すべて従来のハードウェアやソフトウェアとの互換性を保つため。よくここまでやったと思う。あっぱれ。

さてこのように、turbo R の部品点数は少ないけれど、その内部処理は非常に複雑なものへと変化を遂げた。さらに、S1990 は 160 ピンのフラットパッケージで、普通の手作業によるハンダ付けは不可能だ。ハードウェアの高速化と小型化はありがたいけれど、古き良きワンボードマイコン時代の、工作技術が通用しなくなってしまった。しかし、MSX のカートリッジスロットは昔のままだから、これからも MSX はハードウェア入門の教材であり続けるだろう。

1.1.3 エレガントな CPU の切り替え

ビクターの MSX2 マシン、HC-90 と HC-95 では、2 種類の CPU をスイッチで切り替えて使っていた。ところが turbo R では、専用に開発された LSI “S1990” がシステムを管理するので、電源が入ってプログラムが動いている最中でも、CPU を切り替えてプログラムの実行を続けることができる。

このハードウェアのおかげで、従来の MSX 用ソフトウェアは Z80 モードで、turbo R 専用のソフトウェアは高速な R800 モードで、自動的に実行させることができた。また、ハードウェアの種類を調べて、従来の MSX ならば Z80 を、turbo R ならば R800 を選ぶような、MSX2 / turbo R 兼用ソフトウェアも作ることができる。

1.1.4 何でも詰め込む MSX turbo R の ROM 構成

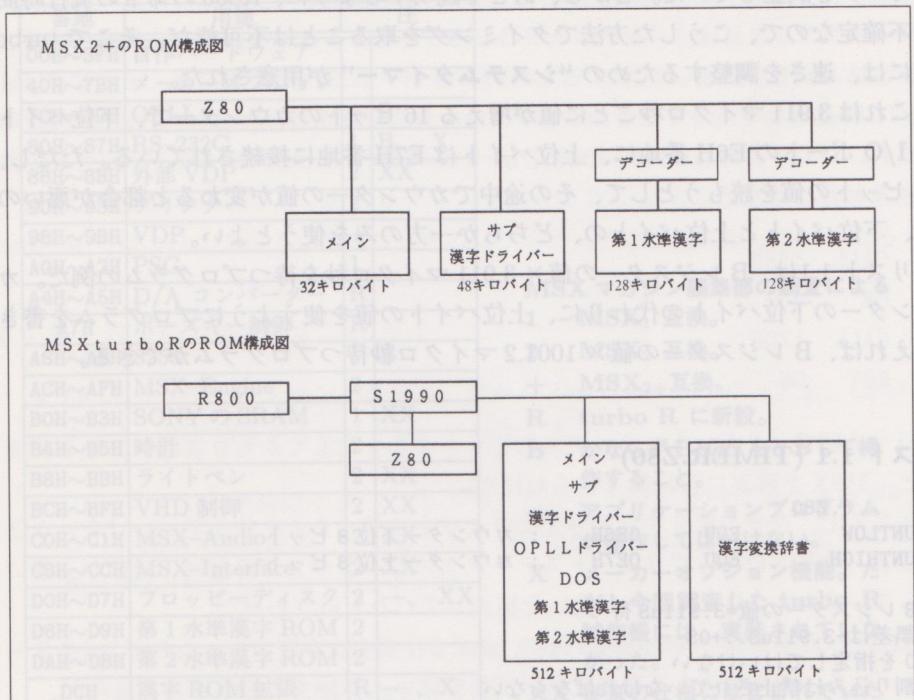
turbo R には多くの ROM が内蔵されているはずだけれど、ふたを開けてみると ROM の数が意外に少ない。その理由が、S1990 が持つメガ ROM 制御機能だ。

MSX2+には、図 1.2 の上側のような ROM が内蔵されている。メイン ROM とサブ ROM はべつべつのスロットに接続され、漢字 ROM は I/O ポートに接続されるので、合計の容量にかかわらず、べつべつの ROM である必要があるわけだ。しかし、32 キロバイトの ROM を 4 個使うよりも、128 キロバイトの ROM を 1 個使う方が、価格も安いし、基板の面積も消費電力も小さくできる。

そこで、turbo R では、図 1.2 の下側のように、1 個の 512 キロバイトの ROM にメイン、サブ、OPLL ドライバー、DOS、第 1 水準漢字、第 2 水準漢字のすべてを詰め込んでしまった。しかし CPU と ROM の間に入っている、S1990 のメガ ROM 制御機能により、ソフトウェアからは、たとえば第 1 水準漢字 ROM は、I/O ポートの D8H 番地と D9H 番地に接続されているように見えるわけだ。

また、合計 64 キロバイトの DOS の ROM (MSX-DOS が 16 キロバイト、MSX-DOS2 が 48 キロバイト) は、スロット 3-2 の 16 キロバイトの空間に、4 バンク切り

図 1.2: MSX turbo R での ROM 構成の変化



替え方式で接続されている。

1.1.5 速さを調節するシステムタイマー

R800 が V9958(画面表示を制御する LSI) を 8 マイクロ秒以内の間隔で使おうとすると、S1990 に内蔵された VDP インターフェース回路が、自動的に R800 にウェイトをかける。これにより、CPU の処理が速すぎるために、V9958 が誤動作する心配はない。

しかし、ほかの周辺 LSI には自動的なウェイト機能がないので、ソフトウェア自身がタイミングを調整する必要がある。従来のソフトウェアの多くは、

EX (SP),HL

EX (SP),HL

または、

PUSH HL

POP HL

のような、時間がかかるけれども副作用がない命令をプログラムに埋め込んで、タイミングを調整していた。しかし、あとで説明するように、R800の命令の実行時間は不確定なので、こうした方法でタイミングを取ることは不可能だ。そこでturbo Rには、速さを調整するための“システムタイマー”が用意された。

これは3.911マイクロ秒ごとに値が増える16ビットのカウンターで、下位バイトはI/OポートのE6H番地に、上位バイトはE7H番地に接続されている。ただし、16ビットの値を読もうとして、その途中でカウンターの値が変わると都合が悪いので、下位バイトと上位バイトの、どちらか一方のみを使うとよい。

リスト1.1は、Bレジスターの値×3.911マイクロ秒を待つプログラムの例だ。カウンターの下位バイトの代わりに、上位バイトの値を使うようにプログラムを書き替えれば、Bレジスターの値×1001.2マイクロ秒待つプログラムができる。

リスト1.1 (TIMER.Z80)

```
.Z80
COUNTLOW      EQU      0E6H      ; カウンタ下位8ビット
COUNTHIGH     EQU      0E7H      ; カウンタ上位8ビット
;
; Bレジスターの値*3.911uS待つ
; 誤差は-3.911uS...+0S
; 0を指定してはいけない
; 割り込みは禁止されていなければならない
; C、A、Fは破壊される
;
WAIT:
    IN      A,(COUNTLOW)   ; カウンターの現在値を得る
    LD      C,A            ; それを保存する
WAIT_LOOP:
    IN      A,(COUNTLOW)   ; カウンターの現在値を得る
    SUB    C                ; 経過時間を算出する
    CP      B                ; 指定された時間経過したか？
    JR      C,WAIT_LOOP    ; 経過していなければループする
    RET
```

1.1.6 MSX turbo R の I/O ポート

turbo Rの記者発表資料にはI/Oマップが含まれていなかったので、取材とハードウェアの解析によって得られた情報をMSX2+のI/Oマップに追加して、編集部が表1.1のI/Oマップを作った。“R”という注が付く項目が、turbo Rに新しく追加されたI/Oポートだ。

まず“D/Aコンバーター”というのは、PCMの録音再生を、BIOSをとおさずに操作するためのI/Oポート。あとで詳細を紹介しよう。“ポーズキー制御”は、ポーズキーによるプログラムの停止を禁止、許可するためのI/Oポート。ディスクの入

表 1.1: MSX turbo R の I/O マップ

番地	用途	注
00H~3FH	自作ハードウェア	
40H~7BH	メーカーオプション	
7CH~7DH	OPLL	+ B
80H~87H	RS-232C	1 B、 X
88H~8BH	外部 VDP	2 XX
90H~93H	プリンター	2
98H~9BH	VDP	+
A0H~A2H	PSG	1
A4H~A5H	D/A コンバーター	R
A7H	ポーズキー制御	R
A8H~ABH	8255	1 B
ACH~AFH	MSX-Engine	2
BOH~B3H	SONY の SRAM	1 XX
B4H~B5H	時計	2
B8H~BBH	ライトペン	2 XX
BCH~BFH	VHD 制御	2 XX
COH~C1H	MSX-Audio	2 XX
C8H~CCH	MSX-Interface	2 XX
DOH~D7H	フロッピーディスク	2 —、 XX
D8H~D9H	第 1 水準漢字 ROM	2
DAH~DBH	第 2 水準漢字 ROM	2
DCH	漢字 ROM 拡張	R —、 X
E3H~E5H	?	R ?
E6H~E7H	システムタイマー	R
F4H	リセットステータス	+ B
F5H	デバイスイネーブル	2
F6H~F7H	AV 制御	2 X
FCH~FFH	メモリーマッパー	2 B

MSX マガジン編集部の調査による

- 1 MSX₁ 互換。
- 2 MSX₂ 互換。
- + MSX₂₊ 互換。
- R turbo R に新設。
- B かならず BIOS をとおして操作すること。
- アプリケーションプログラムが操作してはいけない。
- X メーカーオプション機能。ただし今回調査した turbo R 試作機には、実装されていなかった。
- XX 従来の仕様には含まれていたけれど、turbo R の仕様から削除された。
- ? 何かが接続されているが、仕様書には書かれていない。どうやら、ハードウェア検査用のレジスターがあるようだ。

出力中にプログラムが中断され、ディスクが破壊されるような事態を防ぐために、用意されたようだ。

“漢字 ROM 拡張”は、24 ドットの漢字 ROM や、将来作られるかもしれない JIS 第 3 水準漢字 ROM に備えての、予約機能らしい。その下の“?”は、どの資料にも書かれていないので、I/O ポートを読み書きすると S1990 の内部で何らかのハードウェアが動作するようだ。turbo R のハードウェアを、工場で検査するための I/O ポートではないだろうか。そして“システムタイマー”は、既に説明したおりのものだ。

なお、これは表には書いてないのだけれど、turbo R の速さに対応するためにも、PSG、ジョイスティック、マウス、プリンター、キーボード、時計（バッテリーバッ

クアップされたクロック IC) の操作には、BIOS を使うべきだ。

次に、MSX2+と共に機能なのだけれど、補足説明しておきたいのが“リセットステータス”。これは、ハードウェアのリセットと、メイン ROM の 0 番地へのジャンプによる再起動とを、区別するための I/O ポートだ。具体的には、メイン ROM の 17AH 番地をコールすると、このリセットステータスの値が A レジスターに読み出され、17DH 番地をコールすると、A レジスターの値がリセットステータスに書き込まれる。たとえば、

```
CALL 17AH
OR 80H
CALL 17DH
RST 0H
```

という手順で、リセットステータスのビット 7 を 1 にしてから 0 番地にジャンプすると、MSX を確実に再起動できるわけだ。

ところで、なぜ BIOS をとおさずにリセットステータスを使ってはいけないかというと、マシンによってリセットステータスのハードウェアの信号の論理が、逆になっているから。BIOS がその違いを補正しているというわけだ。

DOS2 が標準装備された turbo R で、ますます重要な存在になったのが“メモリーマッパー”。やや複雑な手順で拡張 BIOS を使い、操作する必要があるものだ。

最後は余談になるけれど、表 1.1 には、かつて実用化または試作されたが、最近の MSX には搭載されていない機能もふくまれている。最近の MSX は当たり前のコンピューターになってしまい、カワリモノの周辺機器が少なすぎると筆者は思うのだが、どうだろうか。

1.1.7 速さを生かすための DRAM モード

メモリーにはそれぞれ、“アクセスタイム”と呼ばれる読み書きの最小時間間隔の制限がある。もしも CPU のスピードが速すぎた場合には、“ウェイト (待ち時間)”を入れて CPU の速さをメモリーに合わせる必要があるわけだ。このアクセスタイムは品種によって異なり、高速に使えるメモリーほど高価になる。また一般的に、ROM よりも RAM のアクセスタイムが短い。

さて R800 の速さを活かすには、プログラムが ROM より RAM に入っているほうがいい。そこで BIOS、BASIC、サブ ROM、漢字ドライバーの各 ROM の内容を、DRAM(メイン RAM) に転送して使う、“DRAM モード”が用意された。

これは、メイン RAM の最後の 64 キロバイトをメモリーマッパーから切り離し、ROM の内容を転送してから書き込み禁止にし、CPU に接続するというもの。CPU

からは、普通の ROM が高速の ROM に差し替えられたように見える。BASIC で書かれたプログラムを実行させる場合など、BIOS と BASIC インタープリターが入った ROM がひんぱんに使われる所以だ。

しかし、マシン語のプログラム、とくに DOS のプログラムを実行させる場合は、ROM が使われる時間が比較的短い。そのため DRAM モードを使うより、余ったメモリーを RAM ディスクなどに活用するほうが有利かもしれない。

また、ROM カートリッジのプログラムも RAM に転送すると高速に動くけど、turbo R ではこれまで以上にディスク版のソフトウェアが主流になっていくだろう。

1.1.8 R800 の特徴はこれだ！

- Z80 とオブジェクトコンパチブル。だから Z80 用のソフトウェアも、CPU のタイミングに依存する部分を除いて動作する。
- CPU のクロック数は 7.16 メガヘルツ。しかも Z80 に比べて命令あたりのクロック数が大幅に減少しているため、Z80 に換算した場合は 29 メガヘルツに相当する（ただし、ノーウェイト時）。
- 16 ビット×16 ビット→32 ビットの精度を持つ乗算命令をサポート。これにより、演算処理速度の大幅な向上が可能になった。
- Z80 では未定義だった、IX / IY レジスターの、上位/下位 8 ビットごとのアクセスを、正式に保証した。

1.1.9 R800 のすべて

turbo R の CPU として採用された R800 は、従来の Z80 にソフトウェア上位互換の高速 CPU だ。つまり、CPU が速すぎて困らない限り、Z80 用に開発されたソフトウェアを、そのまま R800 で高速に実行することができる。

Z80 に追加された機能としては、16 ビットの乗算命令と、Z80 では“裏技”とされていた、IX/IY レジスターのバイトアクセスの命令。詳細については、本書の付録に R800 のインストラクション表を掲載するので、そちらを参考にしてほしい。

従来の MSX のクロック周波数は 3.58 メガヘルツで、turbo R のクロック周波数は 7.16 メガヘルツ。これだけ見ると、速度が 2 倍になっただけのように思えるけど、実際はそうじゃない。R800 ではひとつの命令を実行するのに必要なクロック数が減り、さらに RAM をアクセスするのに M1 サイクルのウェイトが発生しないので、プログラムの実行速度はさらに速くなる。従来の Z80 で、R800 と同じ処理速度を

達成するには、約 29 メガヘルツのクロック周波数になるというから、かなりのスピードアップがはかられたわけだ。

表 1.2: Z80 と R800 の動作速度を比較

命令		MSX2+ (単位μs)	turbo R (単位μs)	倍率
LD	r,s	1.40	0.14	x10.0
LD	r,(HL)	2.23	0.42	x 5.3
LD	r,(IX+n)	5.87	0.70	x 8.4
PUSH	qq	3.35	0.56	x 6.0
LDIR	(BC ≠ 0)	6.43	0.98	x 6.6
ADD	A,r	1.40	0.14	x10.0
INC	r	1.40	0.14	x10.0
ADD	HL,ss	3.35	0.14	x24.0
INC	ss	1.96	0.14	x14.0
JP		3.07	0.42	x 7.3
JR		3.63	0.42	x 8.7
DJNZ	(B ≠ 0)	3.91	0.42	x 9.3
CALL		5.03	0.84	x 6.0
RET		3.07	0.56	x 5.5
MULTU	A,r	—	1.96	—
MULTUW	HL,rr	—	5.03	—

さて、命令の種類ごとに、Z80 と R800 の速さを比較してみた結果が表 1.2。レジスター間のデータ転送 (LD 命令) と、加算の速さが 10 倍になることは、注目に値する。ただし、この表の値は、R800 がノーウェイトで動く場合の速さを測ったもの。実際にはウェイトによって速さが落ちる可能性もあるので、注意しよう。なお、ウェイトが発生する条件とその回避方法を、あとで詳しく説明する。

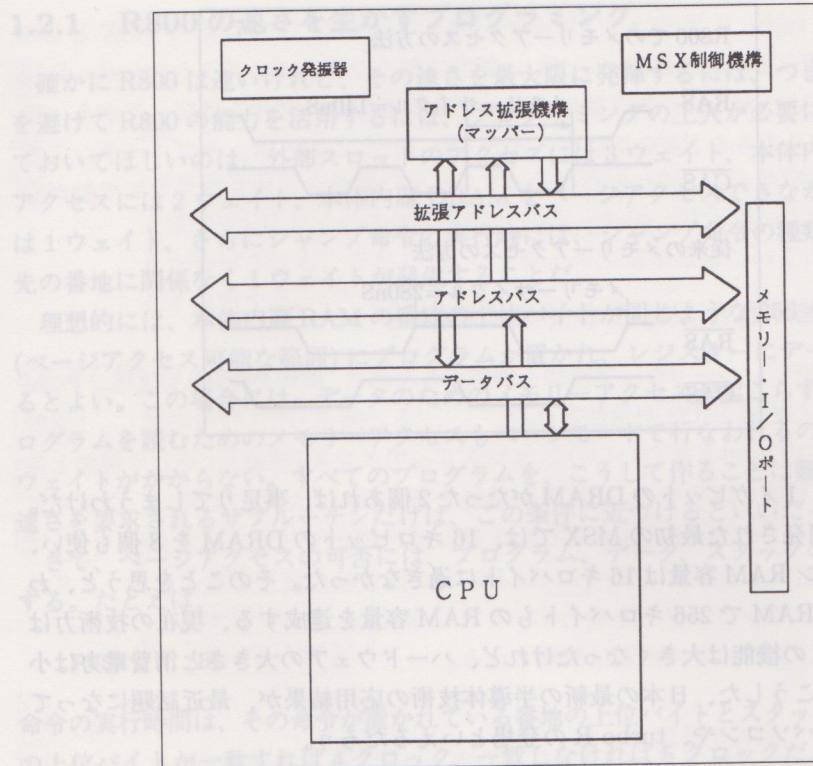
R800 の内部構造は、図 1.3 のようになっている。R800 では、外部データバスは 8 ビットなのだけれど、CPU 内部のデータバスは 16 ビット。だから 16 ビットの加算命令などは、1 サイクルで処理されるわけだ。

このハードウェア構成を見てみると、R800 は 8 ビット CPU の Z80 よりも、外部データバスが 8 ビットの 16 ビット CPU、たとえばインテル社の “8088” やモトローラ社の “MC68008” に近いといえそうだ。

なお、図 1.3 の上のはうに、“アドレス拡張機構 (マッパー)” というものがあるけれど、これは R800 を MSX 以外に使うために用意されたものらしい。turbo R で使う場合は、R800 ではなく S1990 に組み込まれたスロット制御機構と、メモリーマッパーがシステムを制御することになる。

それでは次に、“DRAM のページアクセス” を詳しく説明しよう。まず、これ

図 1.3: R800 内部のブロック図

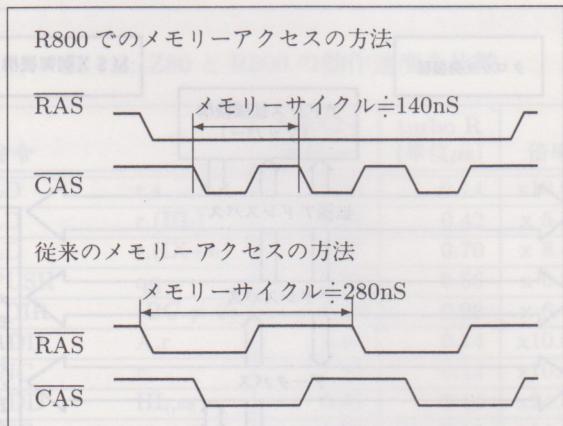


までの Z80 を使ったメモリーアクセスの方法を示したのが、図 1.4 の下側。アドレスの上位バイト (row address) を DRAM へ送り、RAS(row address strobe) 信号を LOW にし、アドレスの下位バイト (column address) を DRAM へと送ったあとで、CAS(column address strobe) 信号を LOW にする。これで、メモリーのアドレスが指定されるわけだ。

一方、R800 での DRAM のページアクセスを示したのが、図 1.4 の上側。アドレスの上位バイトと RAS 信号を固定したまま、アドレスの下位バイトと CAS 信号のみを変化させ、従来の方法の 2 倍の速さで、DRAM を使っている。このように、R800 ではアドレスの上位バイトが変わらずに、連続して DRAM が使われるとき、自動的にページアクセスが行なわれる。

さて、R800 に接続して使用するのが容易な DRAM の種類としては、256 キロビット (32 キロバイト)、1 メガビット (128 キロバイト)、4 メガビット (512 キロバイト) などがあげられる。メイン RAM 容量の最低値が、256 キロバイトと定められた

図 1.4: Z80 と R800 のメモリーアクセス方式の違い



turbo R でも、1 メガビットの DRAM がたった 2 個あれば、事足りてしまうわけだ。

1983 年に開発された最初の MSX では、16 キロビットの DRAM を 8 個も使い、それでもメイン RAM 容量は 16 キロバイトに過ぎなかった。そのことを思うと、わずか 2 個の DRAM で 256 キロバイトもの RAM 容量を達成する、現在の技術力はすごい。MSX の機能は大きくなっただけれど、ハードウェアの大きさと消費電力は小さくなつた。こうした、日本の最新の半導体技術の応用結果が、最近話題になっているノート型パソコンや、turbo R の登場といえるだろう。

ジスター間のデータ転送（レジスタトーレジスタ）が 10 時間になることは、往々に経る。ただし、この動作は、CPU が 1 ワードで動く場合の速さを割ったものである。開拓の上、他の方法で起動するための起動周波数を 1.7MHz とするための起動周波数を 1.7MHz (起動周波数)、その他の外部バス周波数を 1MHz (外部バス周波数) とする。このように、WQI 機器群 (advertisable unit群) は 1MHz に設定される。1MHz で動作されるわけだが、1.7MHz で動作する場合、主な原因は、1.7MHz で動作するための DRAM (DDR SDRAM) が 1MHz の外周周波数で動作するためである。これは DDR SDRAM の特徴である。DDR SDRAM は、DDR SDRAM の場合は、1.7MHz の外周周波数で動作するため、2.7MHz の内部周波数で動作する。DDR SDRAM の場合は、1.7MHz の外周周波数で動作するため、2.7MHz の内部周波数で動作する。DDR SDRAM の場合は、1.7MHz の外周周波数で動作するため、2.7MHz の内部周波数で動作する。

1.2 MSX turbo R 活用法

1.2.1 R800 の速さを生かすプログラミング

確かに R800 は速いけれど、その速さを最大限に発揮するには、つまりウェイトを避けて R800 の能力を活用するには、プログラミングの工夫が必要になる。覚えておいてほしいのは、外部スロットのアクセスには 3 ウェイト、本体内蔵 ROM のアクセスには 2 ウェイト、本体内蔵 DRAM をページアクセスできなかったときには 1 ウェイト、さらにジャンプ命令の実行時には、ジャンプ命令の種類とジャンプ先の番地に関係なく 1 ウェイトが発生することだ。

理想的には、本体内蔵 RAM の番地の上位バイトが同じような 256 バイトの範囲(ページアクセス可能な範囲)にプログラムが置かれ、レジスターにデータが置かれるとよい。この場合には、データのためのメモリーアクセスが起ららず、CPU がプログラムを読むためのメモリーアクセスもページモードで行なわれる所以、CPU にウェイトがかからない。すべてのプログラムを、こうして作ることは難しいけれど、速さを要求されるサブルーチンだけは、この条件に近づけるといいだろう。

さて、ページアクセスの可否には、プログラム、データ、スタックの番地が関係する。たとえば、

```
PUSH      HL
```

命令の実行時間は、その命令が置かれている番地の上位バイトとスタックポインターの上位バイトが一致すれば 4 クロック。一致しなければ 5 クロックだ。ここまで考えながらプログラムを作る必要は少ないだろうけれど、状況に応じて命令の実行時間が異なることは重要なことで、覚えておこう。

1.2.2 R800 を使う上で注意事項と問題点

Z80 では、ひとつの命令を実行するたびに DRAM をリフレッシュしていた。ところが R800 では、31 マイクロ秒ごとに 280 ナノ秒かけて、DRAM をリフレッシュする。注意してほしいのは、このリフレッシュに要する時間と、先ほど説明した DRAM のページアクセス可否の条件のため、R800 のプログラムの実行時間を正確に予測することができないことだ。

そこでプログラムの速さを調節するために、“システムタイマー”というものを使うことになる。あとで、このシステムタイマーの使い方と、CPU と VDP の間の速さの調整について説明するので、待っていてほしい。

また、これはどの新型 CPU でもいえることなのだけど、R800 の問題点として考えられるのは、開発機材が不足していること。とくに、ソフトウェアを開発すると

きに威力を発揮する“ICE(インサーキットエミュレーター)”を、デバッグに使えないことが不便だ。

そのため、turbo R 用のソフトウェアを作るためには、まず従来の MSX と Z80 用の ICE を使って徹底的にデバッグし、確実に動くはずのプログラムを turbo R 用に直す方法がいいだろう。Z80 兼用のプログラムを作って動作を確認してから、掛け算を使う部分のみを R800 用に書き替えるわけだ。このとき、サブルーチンごとにわけて、動作をチェックするのもいい。そして、最後に全体を組み立てて動かなければ……ソースリストを見て考えるしかない。

1.2.3 追加された BIOS とその機能説明

turbo R の新しいハードウェア機能を制御するために、CPU の切り替えと、PCM の録音再生のための BIOS が追加された。

ここでは、BIOS の名称(ラベル)、エントリーアドレス(番地)、そして機能と各レジスターの順番で説明する。BIOS の機能を書き表わすための記号は、以下のとおりだ。まず、[E] とは BIOS を呼び出す前に値を設定すべきレジスター。[R] は BIOS が値を返すレジスターで、[M] は BIOS が無意味な値を書き込む、つまり元の内容が壊されるレジスターを表わす。また IYH とは、IY レジスターの上位バイトを表わし、下位バイトの内容は無視される。

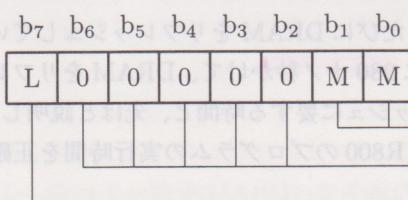
CHGCPU 0180H 番地

機能

CPU を切り替える。

E

A レジスターのビット 1 と 0 で、次のようにモードを設定する。このうち“R800 DRAM”というのは、BIOS の ROM の内容を DRAM に転送して使うモードのことだ。



IY	モード
00	Z80
01	R800 ROM
10	R800 DRAM

また、A レジスターのビット 7 が 1 ならば、どちらの CPU が動いているかを表わす LED が変化する。逆に A レジスターのビット 7 が 0 なら、CPU が切り替えられるが、LED は変化しない。

R

なし

M

AF

注

CPU を切り替える前のレジスターの内容は、AF と R を除いて、切り替え後の CPU にそのまま引き継がれる。また、切り替えたあとは割り込みが許可される。なお CPU 切り替えの注意事項については、あとで詳しく説明する。

GETCPU 0183H 番地

機能

動作中の CPU を調べる。

E

なし

R

動作中の CPU に応じて、A レジスターに次のような値が返される。

0	Z80
1	R800 ROM
2	R800 DRAM

M

F

注

あとで説明する方法でハードウェアが turbo R であることを確かめてから、この BIOS を呼び出す必要がある。

PCMPLY 0186H 番地

機能

PCM の音を再生する。

E

A

b₇ b₆ b₅ b₄ b₃ b₂ b₁ b₀

version

R	0	0	0	0	0	F	F
---	---	---	---	---	---	---	---

周波数

かならず 0 を書き込む

VRAM / MRAM

EHL (データの番地)

DBC (データの長さ)

A レジスターのビット 7 が 1 ならばビデオ RAM に、0 ならばメイン RAM に PCM の音源データが置かれる。なおビデオ RAM にデータがある場合にのみ、D レジスターと E レジスターの値が意味を持つ。

A レジスターのビット 1 とビット 0 で、サンプリング周波数を設定する。ただし 15.75 キロヘルツは、turbo R が R800 の DRAM モードで動いている場合だけ指定可能だ。

00	15.75 キロヘルツ
01	7.875 キロヘルツ
10	5.25 キロヘルツ
11	3.9375 キロヘルツ

R

キャリーフラグ

0 正常終了

1 異常終了

A (異常の原因)

1 周波数指定誤り

2 STOP キーによる中断

EHL (中断番地)

M

すべて

PCMREC

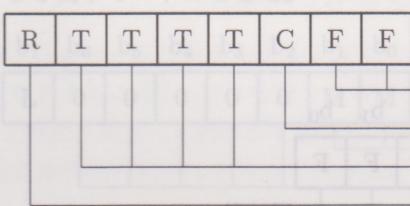
0189H 番地

機能

PCM の音を記録する。

E

A

b₇ b₆ b₅ b₄ b₃ b₂ b₁ b₀

EHL (データの番地)

DBC (データの長さ)

A レジスターのビット 7、1、0 の設定方法は、PCMPY で説明したものと同じ。A レジスターのビット 6 からビット 3 は “トリガーレベル” と

いい、録音をはじめるきっかけとなる音の大きさを指定する。この値が 0 ならば、ただちに録音が開始される。

また、A レジスターのビット 2 が 1 ならば、録音データが圧縮される。0 ならば圧縮されない。

[R]

キャリーフラグ

0 正常終了

1 異常終了

A (異常の原因)

1 周波数指定誤り

2 STOP キーによる中断

EHL (中断番地)

[M]

すべて

1.2.4 変更および削除された BIOS について

turbo R で変更または削除された BIOS は、表 1.3 に示したとおり。それについて、簡単に説明していく。

まず、turbo R ではカセットテープインターフェースがなくなったので、従来の BIOS にあった“TAPION”、“TAPIN”、“TAPIOF”、“TAPOON”、“TAPOUT”、“TAPOOF”をコールすると、キャリーフラグがセットされ、エラーとしてリターンする。また、“STMOTR”もなくなり、コールしても何もしないでリターンする。

また、メイン ROM の容量を変えずに新しい機能を追加するために、パドルとライトペンの BIOS が削除された。BIOS の“GTPDL”をコールすると、A レジスターにかならず 0 が入ってリターンする。同様に、“GTPAD”または“NEWPAD”で、A レジスターにライトペンを指定する 8~11 の値を入れてコールしても、A レジスターにはかならず 0 が入ってリターンする。

変更された BIOS としては、使用中の MSX のバージョンを知るための、“ROM version ID”。これはメイン ROM の 002DH 番地の内容でわかり、turbo R の場合は 03H に変更された。turbo R 用にプログラムを開発するなら、まずこの番地の値が 03H 以上であることを確かめ、そうでなければ、MSX2 用のプログラムとして動作させるか、あるいはエラーメッセージを表示して中断させるようにしよう。

なお、002DH 番地の内容が 03H の場合のみに動くようなプログラムは、将来 MSX がバージョンアップしたときに動かなくなってしまうので、かならず 03H 以上ならば動くように作る必要がある。一般的に、ハードウェアや OS のバージョンについ

表 1.3: MSX turbo R で変更のあった BIOS と BASIC の一覧

追加された BIOS エントリー		追加されたステートメント
CHGCPU	0180H	CALL PCMREC
GETCPU	0183H	CALL PCMLPLAY
PCMPLY	0186H	CALL PAUSE
PCMREC	0189H	
変更された BIOS エントリー		変更されたステートメント
ROM version ID	002DH	COPY
削除された BIOS エントリー		削除されたステートメント
GTPDL	00DEH	CLOAD
TAPION	00E1H	CSAVE
TAPIN	00E4H	MOTOR
TAPIOF	00E7H	
TAPOON	00EAH	
TAPOUT	00EDH	
TAPOOF	00FOH	
STMOTR	00F3H	
GTPAD	00DBH	
NEWPAD	SUB 01ADH	

ては、自分が必要とするバージョン番号以上の値を得られれば、ソフトウェアが動作するようにプログラムしておこう。

これは過去において実際にあったことなのだけれど、MSX のバージョンのチェックを誤ったために、MSX2+では動かない MSX2 用プログラムや、学習機能付きの MSX-JE と組み合わせると動かないアプリケーションなどが、できてしまう。それを避ける意味でも、“03H 以上なら動くようにする”ということを、忘れないでほしい。

また、BIOS と同様に、turbo R になっての BASIC の機能にも追加や変更、削除があった。それらについては、表 1.3 や、マシン付属の BASIC マニュアルを参照してほしい。

1.2.5 アプリケーション開発に関する注意点

MSX turbo R では、R800 は常にノーウェイトで動作しているわけではない。外部スロットをアクセスするときに 3 ウェイト、内部 ROM をアクセスするのに 2 ウェイト、そして内部 DRAM がページブレークを起こしたときに 1 ウェイトかかるのだ。そこで、プログラムの高速化をはかるには、こうしたウェイトをできる限り減らすことを考えながら、作業しなくてはいけない。そのための注意点を 3 つほどまとめてみたので、覚えておこう。

まずひとつ目は、プログラム自体を RAM に転送してから実行させること。フロッピーディスクで供給されるソフトウェアは、必然的に RAM で動作するので問題ないのだけれど、注意したいのはスロット上に ROM カートリッジで供給されるプログラム。必要な部分だけを RAM に転送してから実行させることで、かなりの高速化が可能になる。

ページブレークを起こさないようにコーディングすることも大切だ。R800 では、DRAM のページアクセスをサポートする専用のバスを持っているので、この機能を最大限に活用しよう。具体的には、アドレスの下位 8 ビットだけが変化するような連続したメモリー、つまり ??00H~??FFHまでの 256 バイトの範囲で、メモリーアクセスが行なわれるようプログラムするのが効果的だ。

ちなみに、ページブレークを起こした状態というのは、この範囲を越えてメモリーアクセスが行なわれた場合、つまりアドレスの上位 8 ビットが変化した場合のことを呼んでいる。

前にもちらっと書いたのだけれど、turbo R では MSX2+などとは違い、プログラムのコーディング段階で命令の実行時間が正確にわかるわけではない。その理由としてあげられるのが、いつ発生するか予測のつかない DRAM のページブレークと、Z80 などとは違って、命令の実行とは非同期に行なわれる DRAM のリフレッシュがあるからだ。

また、turbo R と MSX2+のどちらでも動作するようなプログラムを作るのに、ソフトウェアループによってタイミングをとることは勧められない。そこで turbo R には、3.911 マイクロ秒ごとにカウントアップするシステムタイマーが、新たに搭載された。これからは、このシステムタイマーを利用して、タイミングをとるようにしよう。

1.2.6 CPU を切り替えるプログラムの例

リスト 1.2 は、CPU を切り替える “CHGCPU.COM” のソースリストだ。turbo R で DOS2 が動いているときに、

CHGCPU 0

で Z80 モードが、

CHGCPU 1

で R800 の ROM モードが、

CHGCPU 2

でR800のDRAMモードが、それぞれ選択される。プログラムの内容を解説すると、DOSのワークエリア(正確にはdefault FCB area)の5DH番地からコマンドの第1パラメーターの先頭の文字を得て、それに応じてAレジスターの値を設定。そして、メインROMの180H番地のBIOS、“CHGCPU”を呼び出すというものだ。

また、プログラムにより実用性を持たせるため、DOSのバージョン番号をチェックする処理も加えてある。具体的には、まずメインROMの2DH番地の内容が03H以上である、つまりturbo Rであることを確かめ、DOSのシステムコールの6FHを使って、DOSカーネルのバージョン番号が2以上であることを確かめている。

リスト 1.2 (CHGCPU.Z80)

```
.Z80
RDSLت EQU 0000CH ; inter slot read
CALSLت EQU 0001CH ; inter slot call
EXPTBL EQU OFCC1H ; slot # of main ROM
;
ld a,(EXPTBL)
ld hl,2dh ; address to read
call RDSLت ; read version
cp 3 ;
jr nc,TURBOR ;
ld de,MSG_NOTR
ld c,9 ; _STROUT
call 5
rst 0 ; return to DOS
TURBOR:
ld c,6fh ; _DOSVER
call 5
ld a,b ; version of DOS kernel
cp 2 ;
jr c,NOTDOS2
ld a,d ; version of MSXDOS.SYS
cp 2
jr c,NOTDOS2
;
ld a,(005ch+1) ; command parameter
sub '0' ; 0:Z80, 1:R800ROM, 2:R800RAM
ret c ; abort if parameter < '0'
cp 3 ;
ret nc ; abort if '3' <= parameter
or 80h ; set change-LED flag
ld ix,180h ; address of CHGCPU
ld iy,(EXPTBL-1) ; slot of main ROM
call CALSLت ; inter-slot call
rst 0 ; return to DOS
;
NOTDOS2:
ld de,MSG_NOTDOS2
ld c,9 ; _STROUT
```

```

1.3   call    5      ; DOSへ戻る
       rst    0      ; return to DOS
;
MSG_NOTR:
DB     'not MSX turbo R', Odh, Oah, '$'
MSG_NOTDOS2:
DB     'not MSX-DOS 2', Odh, Oah, '$'
END

```

同様に、次のリスト 1.3 は、MSX2 用のプログラムを R800 モードでだまして動かす、“GAMEBOOT.COM”のソースリストだ。このプログラムは、DOS2 が起動され R800 が選択されている状態で、ほかのディスクに入っているプログラムを起動するためのもの。つまり、DOS2 のシステムが含まれていないプログラム（ゲームなど）を、強引に R800 モードで動かすためのものだ。

簡単にプログラムを解説していくと、まず画面にメッセージを表示して、ディスクが交換されるのを待つ。次に、交換されたディスクのブートセクターを読み込んで、それを実行させる。そのときの環境は、普通の方法でブートセクターが 2 回目にコールされるときと同じで、ページ 1 は DOS の ROM、そのほかのページは RAM、キャリーフラグはセットされている。

さらに、エラー処理プログラムへのポインターの、ポインターを記憶するための DOS のワークエリア (F323H 番地) を HL レジスターに、またページ 1 を RAM から DOS の ROM へ切り替えるプログラムの番地 (F368H 番地) を DE レジスターに、それぞれ設定している。

PCM データ用に使うときは、

CLEAR 200,&HC000

のように書いておこう。リスト 1.3 に書いた “CLEAR” のアドレスを記憶しておいたので、それを入力して遊んでみるといいだろう。

もちろん、ビデオ RAM を PCM データ用に使う場合は、任意のビデオメモリを置くことができるので、開始番地で 0x00000000 などとすればいい。ただしビデオ RAM の場合は、PCM データを自分で確認する必要がある。最初に

SCREEN 8

のように、スクリーンモードを設定してから PCM 読音すれば、画面にデータがズラズラっと表示されて、おもしろいかもしれない。

基本的に PCM の録音、再生の方法は、以上のことを注意すれば大丈夫。さらに再生サンプリングレートを変化させれば、4 段階のスピードで再生することもできる。ただ、問題となるのは、PCM を再生しているとき、turbo R がそれにかかりづ

で ROM & RAM モードがそれぞれ選択される。プログラムの内蔵上部にあるリスト 1.3 (GAMEBOOT.Z80) は、ROM の 0DH 番地からスタートの

```
.z80
; This program boots a game disk from drive A:.
; It reads the boot sector of the game disk and executes it.

._conin    equ     01h
._strout   equ     09h
._settda   equ     1ah
._rdabs    equ     2fh

dos        equ     0005h
enaslt    equ     0024h

notfirst  equ     0f340h
master    equ     0f348h

ld         sp,(6)
ld         de,prompt      ; print prompt message
ld         c,_strout
call      dos
ld         c,_conin      ; wait for key in
call      dos
ld         de,0c000h       ; read boot sector at 0c000h
ld         c,_settda
call      dos
ld         de,0            ; logical sector 0
ld         l,0              ; drive A:
ld         h,1              ; read 1 sector
ld         c,_rdabs
call      dos
ld         h,40h
ld         a,(master)
call      enaslt
ld         hl,0f323h
ld         de,0f368h
xor      a
ld         (notfirst),a
scf
jp         0c01eh

prompt:
db         'Insert game disk in drive A:, ,0dh,0ah
db         'and press any key $'

end
ret
nc
or         80h
ld         l,180h
ld         h,(EXPTBL-1)
call      CALLT
ret
0

NOTDOS2:
ld         de,MSG_NOTDOS2
ld         c,8
ret
```

1.3 PCM 限界ギリギリ活用法

turbo R に加えられた新しい機能が PCM。せっかく用意された機能だから、その性能をギリギリまで引き出したいと思うのが人情だ。そこで、BASIC からマシン語、水平走査線割り込みを利用した特殊な使い方まで、PCM の活用法を紹介する。

1.3.1 基礎編…… BASIC での使い方

まずは BASIC を使った基本的なものから紹介しよう。

そもそも PCM は、マイクなどから入力した音声をデジタルに変換してメモリーに記憶させ、任意にそれを再生させるものだ。

turbo R の場合、PCM データを記憶するのは、メイン RAM かビデオ RAM。サンプリングレートは、15.75 キロヘルツ、7.875 キロヘルツ、5.25 キロヘルツ、3.9375 キロヘルツの 4 種類から選択することになる。この値が大きいほど、より質の高いサンプリングができるというわけだ。

BASIC から PCM を使う場合は、ふたつの命令を覚えておけばいい。使い方はあとにまとめておいたので、参考にしてね。基本的には、これらの命令を実行するだけで、PCM の録音や再生は可能になる。ただし、データを記憶する開始番地と終了番地の設定には、十分に注意する必要があるぞ。

まず最初に、BASIC の “CLEAR” 命令で PCM データ用のメモリー領域を確保しておかないと、間違いなく暴走してしまう。たとえば C000H～D000H 番地までを PCM データ用に使うときは、

```
CLEAR 200,&HC000
```

のようにしておこう。とりあえず、リスト 1.4 に簡単なサンプルプログラムを載せておいたので、これを入力して遊んでみるといいだろう。

もちろん、ビデオ RAM を PCM データ用に使う場合は、任意のどの番地にもデータを置くことができるので、開始番地や終了番地を気にする必要はない。それにビデオ RAM の場合は、PCM データを目で確認することができる。最初に

```
SCREEN 8
```

のように、スクリーンモードを設定してから PCM 録音すれば、画面にデータがズラズラっと表示されて、おもしろいかもしない。

基本的な PCM の録音、再生の方法は、以上のこと들을注意すれば大丈夫。さらに再生サンプリングレートを変化させれば、4 段階のスピードで再生することもできる。ただ、問題となるのは、PCM を再生しているとき。turbo R がそれにかかりつ

きりになってしまふので、PCMを再生しながら何かをするなんてことは、残念ながらできないのだ。

リスト 1.4 (PCM1.BAS)

```
10 CLEAR100,&H9000
20 PRINT "イマカラ ロクオン シマス。";
30 A$=INPUT$(1):PRINT
40 _PCMREC (@&H9000,&HCFFF,0)
50 PRINT "サイセイ シマス。";
60 A$=INPUT$(1):PRINT
70 _PCMPLAY (@&H9000,&HCFFF,0)
80 GOTO 20
```

1.3.2 PCM 関係の BASIC 命令

CALL PCMREC 書式

- メイン RAM またはビデオ RAM への録音。
CALL PCMREC(@開始番地, 終了番地, サンプリングレート [, [トリガーレベル], 圧縮スイッチ][, S])
- 配列変数への録音。
CALL PCMREC(配列変数名, [長さ], サンプリングレート [, [トリガーレベル], 圧縮スイッチ])

サンプリングレートの設定	
指定値	サンプリングレート
0	15.7500KHz
1	7.8750KHz
2	5.2500KHz
3	3.9375KHz

トリガーレベルでは、録音が開始されるときの入力レベルを設定する。値は 0~127 まで。この値以上の入力レベルになると録音が開始され、0 または省略した場合には、すぐに録音がはじまる。圧縮スイッチの設定は、1 で無音部分を圧縮し、0 または省略すると圧縮しない。

CALL PCMPLAY 書式

- メイン RAM またはビデオ RAM からの再生
CALL PCMPLAY(@開始番地, 終了番地, サンプリングレート [, S])
- 配列変数からの再生
CALL PCPLAY(配列変数名, [長さ], サンプリングレート)

PCMREC、PCMPLAY とともに、高速モードでないときは、一時的に高速モードにしてから実行し、終了するとの状態に戻ってくる。また、R800 の ROM モードで 15.75KHz が指定された場合は、エラーになる。

録音、再生中に **STOP** キーが押されると、プログラムの実行は中断される。PCM データの形式は、1~255 までが通常のデータで、0 は特殊なもの。あとに続く 1 バイトで指定された回数分だけ、0 レベル (127) を出力する。

1.3.3 BEEP 音を PCM で鳴らすのだ！

BASIC のプログラム実行しているときに、**CTRL** と **STOP** キーを同時に押してプログラムを中断させると、“ピッ”と BEEP 音が鳴るのは知ってるよね。“LIST”命令でリストを表示させ、**CTRL** + **STOP** で止めたときも、同じように“ピッ”と音がする。BASIC の“SET BEEP”命令を使えば、音を変えることもできるけど、4 種類用意されているどの音も、いまひとつインパクトに欠けるのだ。

そこで、この BEEP 音を PCM で鳴らすとどうなるか。変なセリフを設定しておくと、ことあるごとに MSX がしゃべるので、けっこううるさくて楽しいかもしれない。

というわけで、リスト 1.5 の掲載したプログラムを実行すると、BEEP 音を PCM で鳴らせるようになる。もちろん turbo R 専用だ。それはほど長いものでもないので、頑張って入力してほしい。

なお、このプログラムは、メイン RAM のページ 1(4000H~60FFH 番地まで)に置かれるので、プログラムを実行したあと、“CLEAR”命令でユーザーエリアの上限を B000H 番地以上にしてもかまわない。ただし、メモリーディスク関係は使えないでの、うっかり“CALL MEMINI”なんてやらないように。それから、BASIC の“BEEP”命令を使うときは、

PRINT CHR\$(7)

を代わりに使わないと、BEEP 音が PCM にならない。注意しよう。

プログラムの使い方を説明する。

1 PCM BEEP セット

以降 BEEP 音が PCM になる。1 回実行しておけば、電源を切るまで設定は有効。また、CLEAR 文の設定を変更してもかまわない。

2 PCM BEEP リセット

BEEP 音を、もとの状態に戻す。“CALL SYSTEM”で DOS や DOS2 にするときは、かならずこのコマンドを実行すること。

3 PCM データ再生

現在設定されている PCM データを再生する。確認用に使おう。

4 PCM データ録音

PCM データを 15.75 キロヘルツで録音する。録音時は B000H～CFFFH 番地までのメモリーを使用。

5 PCM データ LOAD

拡張子が “.PCM” の、 BSAVE 形式でセーブされた PCM データを読み出す。

6 PCM データ SAVE

“PCM データ録音” で録音した PCM データを、ディスクに記録する。

0 END

プログラムを終了する。もちろん、 [CTRL] + [STOP] でもかまわない。

なお、簡単なメッセージが画面に表示されるので、参考にしよう。

リスト 1.5 (PCM2.BAS)

```

10 SCREEN0:WIDTH40:DEFINT A-Z
20 CLEAR100,&HB000
30 DEFUSR=&HD800:DEFUSR1=&HD806:DEFUSR2=&HD803
40 FOR I=&HD800 TO &HD87F
50 READ A$:POKE I,VAL("&H"+A$):NEXT
100 PRINT
110 PRINT"1) PCM BEEP セット"
120 PRINT"2) PCM BEEP リセット"
130 PRINT"3) PCM DATA サイセイ"
140 PRINT"4) PCM DATA ロクオン"
150 PRINT"5) PCM DATA LOAD"
160 PRINT"6) PCM DATA SAVE"
170 PRINT"0) END"
180 PRINT"....HIT 0-6 KEY=";
190 A$=INPUT$(1):I=ASC(A$)-ASC("0")+1
200 IF I>0 AND I<8 THEN ELSE190
210 ON I GOTO 230,240,310,220,340,390,430
220 PRINTCHR$(7);:GOTO 190
230 GOSUB 470:END
240 GOSUB 470:I=USR1(0):I=USR(0)
250 PRINT"PCM BEEP ヲ ツカウコトガ テキマス。"
260 PRINT"DOS マタハ DOS2 ヲ ツカウトキハ カナラズ PCM BEEP ヲ リ
セット シナオシテ クダサイ。"
270 PRINT"PCM BEEP ノ データ ハ ページ1(4100H カラ 60FFH) ニ アリ
マス。"
280 PRINT"CLEAR メイレイ デ B000H イショウ ニ セッティ シテモ カ
マイマセンガ、CALL MEMINI ナドノ メモリーディスク カンケイ ノ メ
イレイ ハ ツカウ コトガ テキマセン。"

```

```

290 PRINT"ナオ、BEEP メイレイ ヲ ショウ スルトキ ハ PRINT CHR$(7
) ヲ ツカッテ クダサイ。"
300 END
310 GOSUB 470:POKE &HFDA4,&HC9
320 PRINT"PCM BEEP ヲ リセット シマシタ。"
330 GOTO 100
340 GOSUB 470
350 PRINT"PCM ロクオン ヲ ハジメマス。 (HIT ANY KEY!)";
360 A$=INPUT$(1):PRINT:_PCMREC(&HB000,&HCFFF,0):I=USR(0)
370 PRINT"ロクオン シュウリョウ デス。"
380 GOTO 100
390 GOSUB 470
400 PRINT"PCM データ LOAD"
410 INPUT" FILE NAME(8 モジ)=";A$
420 BLOAD A$+" .PCM":I=USR(0):GOTO 100
430 GOSUB 470
440 PRINT"PCM データ SAVE"
450 INPUT" FILE NAME(8 モジ)=";A$
460 I=USR2(0):BSAVE A$+" .PCM",&HB000,&HCFFF:GOTO 100
470 PRINT CHR$(I+47):PRINT:PRINT:RETURN
480 DATA C3,4D,D8,C3,5F,D8,CD,6D
490 DATA D8,3A,42,F3,32,2A,D8,21
500 DATA 2E,D8,11,00,40,01,00,01
510 DATA ED,B0,CD,76,D8,21,29,D8
520 DATA 11,A4,FD,01,05,00,ED,B0
530 DATA C9,F7,00,00,40,C9,FE,07
540 DATA C0,01,00,20,21,00,41,3E
550 DATA 03,D3,A5,F3,DB,A4,D6,01
560 DATA 38,FA,7E,D3,A4,23,0B,79
570 DATA B0,20,F1,FB,C9,CD,6D,D8
580 DATA 21,00,B0,11,00,41,01,00
590 DATA 20,ED,B0,CD,76,D8,C9,CD
600 DATA 6D,D8,21,00,41,11,00,B0
610 DATA 01,00,20,18,EC,3A,42,F3
620 DATA 21,00,40,C3,24,00,3A,C1
630 DATA FC,21,00,40,C3,24,00,00

```

1.3.4 上級編……マシン語で PCM を！

マシン語で PCM を使う場合、手っとり早いのが BIOS を使う方法。サンプリングレートや、トリガーレベルなどの設定は、BASIC のものとほぼ同じなので問題はないだろう。

ここでは上級編ということなので、この BIOS を使わずに、PCM を録音したり再生したりできるプログラムをふたつ紹介する。

BIOS を使う場合、サンプリングレートは 15.75 キロヘルツ、7.875 キロヘルツ、5.25 キロヘルツ、3.9375 キロヘルツの 4 種類しか選べない。63.5 マイクロ秒ごとに値が変わるカウンターを使っているため、4 種類以上のサンプリングレートを設定できないのだ。このカウンターを、3.911 マイクロ秒ごとに値が変更される、シス

ムタイマーで肩代りしたのがここで紹介するプログラムだ。

まずは、録音プログラムの使い方から説明しよう。HL レジスターには PCM データを格納するメモリーの先頭アドレスを、BC レジスターには録音するデータの大きさを、それぞれ設定しておく。そして E レジスターには、システムタイマーで何カウント分のウェイトを入れるのか、を設定する。16 で、だいたい 15.75 キロヘルツに相当するかな。

再生プログラムのほうも同じ。HL レジスターに再生する PCM データの先頭アドレス、BC レジスターにはデータの大きさ、そして E レジスターにウェイトのカウント数を設定すればいい。

PCM 録音プログラムのリストの途中に、

```
OEDH,70H
```

というヘンなものがあるけど、これは

```
IN (HL),(C)
```

という命令のこと。C レジスターのポートから値を読み、それをフラグだけに反映させる、R800 独自の命令だ。

プログラムの原理はともあれ、とにかく使ってみよう。E レジスターの値を変えることで、音がいろいろに変化して楽しめるはずだ。

リスト 1.6 (PCMREC.MAC)

```
PMDAC EQU 0A4H
PMCNT EQU 0A4H
PMCTL EQU 0A5H
PMSTAT EQU 0A5H
SYSTML EQU 0E6H ; system timer port

REC:
LD A,00001100B
OUT (PMCTL),A ; A/D MODE
DI
XOR A
OUT (SYSTML),A ; reset timer

REC1:
IN A,(SYSTML)
CP E
JR C,REC1 ; wait
XOR A
OUT (SYSTML),A ; reset timer

PUSH BC
```

```

LD    A,00011100B
OUT   (PMCNTL),A      ; DATA HOLD
LD    A,80H
LD    C,PMSTAT

OUT   (PMDAC),A      ; BIT CONVERT
DEFB  OEDH,70H
JP    M,RECAD0
AND   0111111B

RECAD0:
OR    0100000B
0A5F
0A5F OUT   (PMDAC),A
0A4F DEFB  OEDH,70H
0A4F JP    M,RECAD1
AND   1011111B

RECAD1:
OR    0010000B
0A5F
0A5F OUT   (PMDAC),A
0A4F DEFB  OEDH,70H
0A4F JP    M,RECAD2
AND   1101111B

RECAD2:
OR    0001000B
0A5F
0A5F OUT   (PMDAC),A
0A4F DEFB  OEDH,70H
0A4F JP    M,RECAD3
AND   1110111B

RECAD3:
OR    00001000B
0A5F
0A5F OUT   (PMDAC),A
0A4F DEFB  OEDH,70H
0A4F JP    M,RECAD4
AND   1111011B

RECAD4:
OR    00000100B
0A5F
0A5F OUT   (PMDAC),A
0A4F DEFB  OEDH,70H
0A4F JP    M,RECAD5
AND   1111101B

RECAD5:
OR    00000010B
0A5F
0A5F OUT   (PMDAC),A
0A4F DEFB  OEDH,70H
0A4F JP    M,RECAD6
AND   11111101B

RECAD6:
OR    00000001B
0A5F
0A5F OUT   (PMDAC),A

```

```

DEFB OEDH,70H
JP M,RECAD7
AND 11111110B
RECAD7:
OR 0000000B
LD (HL),A
LD A,00001100B
OUT (PMCTL),A
POP BC
INC HL
DEC BC
LD A,C
OR B
JR NZ,REC1
; end of data ?
; next data
LD A,00000011B
OUT (PMCTL),A ;D/A MODE
EI
RET
END

```

リスト 1.7 (PCMPLAY.MAC)

```

PMDAC EQU 0A4H
PMCNT EQU 0A4H
PMCTL EQU 0A5H
PMSTAT EQU 0A5H
SYSTML EQU 0E6H ; system timer port

PLAY:
LD A,00000011B
OUT (PMCTL),A ; D/A MODE
DI
XOR A
OUT (SYSTML),A ; reset timer
REC1:
PLAY1:
IN A,(SYSTML)
CP E
JR C,PLAY1 ; wait
XOR A
OUT (SYSTML),A ; reset timer
LD A,(HL)
OUT (PMDAC),A ; play 1 byte
INC HL
DEC BC
LD A,C
OR B
JR NZ,PLAY1
; end of data ?
; next data

```

EI
RET

END

SLOT

表 1.4: PCM 用の I/O ポート

番地	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0A5H Write	0	0	0	SMPL	SEL	FILT	MUTE	ADDA
0A5H Read	COMP	0	0	SMPL	SEL	FILT	MUTE	BUFF
0A4H Write	DA7	DA6	DA5	DA4	DA3	DA2	DA1	DA0
0A4H Read	0	0	0	0	0	CT1	CT0	

- ADDA (BUFF) : バッファーモード D/A コンバーターの出力を指定する。D/A 時は 0(ダブルバッファー)、A/D 時は 1(シングルバッファー)にしよう。なお、リセット時はダブルバッファーの状態になっている。
- MUTE : ミューティング制御 システム全体の音声出力をオンにしたり、オフにしたりする。
0 : 音声出力オフ (リセット時)
1 : 音声出力オン
- FILT : サンプルホールド回路入力信号の選択 A/D 時にサンプルホールド回路に入力する信号を、フィルターの出力信号にするか、基準信号にするかを選択する。0 で基準信号、1 でフィルター出力信号音になる。リセット時は 0。
- SEL : フィルター入力信号の選択 ローパスフィルターに入力する信号を、D/A コンバーターの出力信号にするか、マイクアンプの出力信号にするかを選択する。0 で D/A コンバーター出力信号、1 でマイクアンプ出力信号音。
- SMPL : サンプルホールド信号 入力信号をサンプルするか、ホールドするかを選択する。
0 : サンプル (リセット時)
1 : ホールド
- COMP : コンパレーターの出力信号 サンプルホールドの出力信号と D/A コンバーターの出力信号とを比較する。
0 : D/A 出力 > サンプルホールド出力
1 : D/A 出力 < サンプルホールド出力
- DA7~DA0 : D/A 出力データ PCM データを再生するときに、用意されたデータをここに出力することで、PCM 音を再生することができる。データの形式はアブソリュートバイナリーで、127 が 0 レベルに相当する。
- CT1、CT0 : カウンターデータ 63.5 マイクロ秒ごとにカウントアップされる。D/A 時にはカウントアップに同期し、0A4H 番地に書かれたデータが繰り返し出力される。また 0A4H にデータを書き込むとカウンターはクリアされる。

```

        DEFB 0EDH,70H
        JP    H,REGAD7
        AND  11111110B
REGAD7:
        DB  00000000B

```

LD {HL},A
LD A,(REGAD7)
OUT (REGAD7)

	REGAD7	REGAD8	REGAD9	REGAD10	REGAD11	REGAD12	REGAD13	REGAD14	REGAD15	REGAD16	REGAD17	REGAD18	REGAD19	REGAD20	REGAD21	REGAD22	REGAD23	REGAD24	REGAD25	REGAD26	REGAD27	REGAD28	REGAD29	REGAD30	REGAD31
0A4H Mdata	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0A5H Regd	COWB	DAB	DAB	DAB	DAB	DAB	DAB	DAB	DAB	DAB	DAB	DAB	DAB	DAB	DAB	DAB	DAB	DAB	DAB	DAB	DAB	DAB	DAB	DAB	DAB
0A4H Mdata	DA2	DA2	DA2	DA2	DA2	DA2	DA2	DA2	DA2	DA2	DA2	DA2	DA2	DA2	DA2	DA2	DA2	DA2	DA2	DA2	DA2	DA2	DA2	DA2	DA2
0A7H Regd	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

音符再生ホールドビット : REGAD1
ホールドビットを出力するにはサウンドデータ
DA1～DA10を0に設定する。
DA1～DA10(音イマジナリ)を0に設定する。
REGAD1～REGAD10を0に設定する。

音符データモードへ戻すには : C07H
A\DA1～DA10を音符データへ戻す。
音符データモードへ戻すには音符データ出力モードへ戻す。
DA1～DA10(音イマジナリ)を0に設定する。

音符データ出力モード : DAB～DA0
音符データモードを出力モードへ戻す。
音符データモードへ戻すには音符データ出力モードへ戻す。
DA1～DA10(音イマジナリ)を0に設定する。

モード切替 : C01F, C01D
モード切替で音符データモードへ戻す。
モード切替で音符データモードへ戻す。
DA1～DA10(音イマジナリ)を0に設定する。
PLATI～PLAT0を0に設定する。

IN A,(SYSTBL),A ; a1

CP E

JR C,PLAY1 ; WAIT

TOR A

OUT (SYSTBL),A ; recent timer

LD A,(HL)

OUT (PHDAC),A ; play 1 byte

INC HL

DEC BC

LD A,C

OR B

JR NZ,PLAY1 ; end of data ?

JR NZ,PLAT1 ; next data

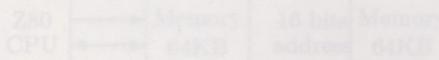
* ADDY (BULE) : バーチャルメモリを実現するためのアドレス空間。
DA1～DA10(音イマジナリ)を0に設定する。
DA1～DA10(音イマジナリ)を0に設定する。
DA1～DA10(音イマジナリ)を0に設定する。
DA1～DA10(音イマジナリ)を0に設定する。

音符データ出力モードへ戻すには : C07H
A\DA1～DA10(音イマジナリ)を0に設定する。
DA1～DA10(音イマジナリ)を0に設定する。

音符入力モードへ戻すには : E01H
外部の音
入力端子を0に設定する。
音符入力モードへ戻すには音符入力モードへ戻す。
音符入力モードへ戻すには音符入力モードへ戻す。

* REG : 音符データモードへ戻す。
音符データモードへ戻すには音符データモードへ戻す。
音符データモードへ戻すには音符データモードへ戻す。

第2章 SLOT



スより。“4端口式セレクタ”七穴の方式による多機能性をもつた。またデータバスを複数の端子に接続するため、各端子間のデータ交換が可能である。



この章は、MSX マガジン 1989 年 2 月号、1989 年 3 月号の“MSX2+テクニカル探検隊”と、1990 年 11 月号の“テクニカル・アナリシス”的記事を再編集したものである。

2.1 スロットって何だ

MSX にカートリッジをセットするための穴は“カートリッジスロット”。でもスロットが意味するのは、MSX のメモリーを管理する機能もある。この章では、もっとも重要で難解なスロットを説明するぞ。

2.1.1 CPU とメモリーはどうつながってるの

コンピューターを構成するもっとも重要な部品といったら、“CPU”とメモリー。CPU とは“中央処理装置 (Central Processing Unit)”の略称で、コンピューター全体を管理し計算を行なう装置のこと。一方メモリーとは、CPU が扱う情報を覚えるメモ帳のような装置を指す。

コンピューターが扱う情報は、数字の 0 と 1 を組み合わせた“2 進数”で表わされることは知ってると思う。この 2 進数の 1 桁を“ビット (bit)”, 8 桁を“バイト (byte)”という。また、プログラムリストの中などで、2 進数をそのまま表記すると桁数が多くなってしまうので、4 ビットの 2 進数を 0~9 と A~F の文字で表わす“16 進数”もよく使われる。

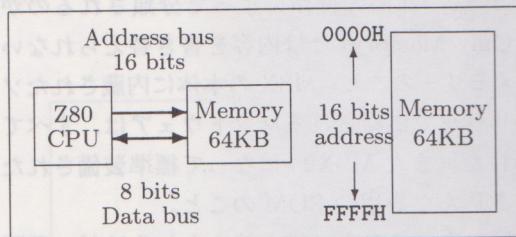
これは間違えないでほしいのだけど、コンピューターの世界では、“キロ (K)”という単位が、1000 倍ではなく 1024 倍を意味する。たとえば、64 キロバイトのメモリーとは、 $64 \times 1024 = 65536$ バイトのメモリーのこと。これは、 $65536 \times 8 = 524288$ ビットでもあるわけだ。

これらのメモリーを管理するために、多くのマイクロコンピューターでは、1 バイトごとにメモリーに番号が付けられている。それが“番地”や“アドレス”と呼ばれるもの。よくマシン語のプログラムなどで、“実行開始番地は 8000H”などと書かれているのがそうだ。

2.1.2 8 ビット CPU Z80 の内部を探る

CPU とメモリーは、図 2.1 のように“アドレスバス”と“データバス”で接続されている。アドレスバスとは、CPU が読み書きしたいメモリーの番地を指定する信号を、CPU からメモリーへ送るための電線。データバスとは、メモリーの内容を通信するための電線のこと。前者が CPU からメモリーへの一方通行であるのに対し、後者は双方向になっている点に注意しよう。

図 2.1: Z80 CPU のメモリー



MSX に使われている Z80 CPU には、基本的に 64 キロバイトのメモリーを接続できる。16 ビットのアドレスバスの信号で、0000H ~ FFFFH までのメモリーの中の 1 バイトを指定できるわけだ。またデータをやり取りするためのデータバスは、8 ビットになっている。

turbo R 以前の MSX の CPU である “Z80” は、8 ビット（物理的には 8 本の電線）のデータバスと、16 ビットのアドレスバスを持っている。これにより、64 キロバイトのメモリーを 1 バイトずつ読み書きできるわけだ。このような CPU を “8 ビット CPU”、8 ビット CPU が組み込まれたコンピューターを “8 ビットコンピューター” と呼ぶ。だから MSX は 8 ビットコンピューターというわけ。

さて、アドレスバスに関して具体的に説明すると、16 ビットのアドレスバスで指定できるメモリーの番地は、2 進数の

0000000000000000B

から (B は 2 進数を表わす記号)

1111111111111111B

まで。これを 10 進数で表わすと 0~65535 まで、16 進数で表わすと 0000H~FFFFH 番地までということになる (H は 16 進数を表わす記号)。各番地の内容は 8 ビット (=1 バイト) で、10 進数では 0~255 までの値を表わす。また、バイト単位で表わされたこれらのメモリーを、キロバイト単位に直すと 64。ゆえに 8 ビットコンピューターには、64 キロバイトのメモリーを接続できるというわけだ。

MSX は 8 ビットコンピューターだと前に書いたけど、最近は 16 ビットコンピューター (16 ビット CPU を搭載したコンピューター) も普及してきている。この場合はデータバスが 16 ビットなので、8 ビットコンピューターに比べ 2 倍の情報を 1 度に読み書きできる。アドレスバスも多くなり、それだけ多くのメモリーを接続できるなどの利点も多い。けれど配線が複雑になることなどから、比較的高価になってしまうのが現状だ。また、大型コンピューターの多くは、32 ビットや 64 ビットのデータバスとアドレスバスを持っている。

なお、MSX turbo R には 16 ビット CPU の R800 が搭載されたが、従来のカートリッジを接続できるように、データバスは 8 ビットのままだ。

2.1.3 メモリーの種類は働きによってイロイロ

メモリーには多くの種類がある。まず、部品の種類によって分類されるのが“ROM”と“RAM”。ROM(Read Only Memory)とは内容を書き替えられないかわりに、電源を切っても内容が残るメモリーのこと。MSXの本体に内蔵されたソフトウェア(BASICなど)や、カートリッジで供給されるソフトウェアは、すべてこのROMの中に書き込まれているわけだ。またMSX2+になって標準装備された漢字ROMとは、漢字の文字の形を書き込んだ専用のROMのこと。

RAM(Random Access Memory)とは、内容を自由に書き替えられるけど、電源を切るとその内容が消えてしまうメモリー。プログラム中で計算結果を一時的に記憶させたり、フロッピーディスクからプログラムを読み込んで実行させたりするのに利用する。たとえば、Mマガに載ったショートプログラムを打ち込んでゲームをするなんて場合も、このRAMに記憶されるわけだ。

なお、“SRAM”とは、消費電力が小さいRAMで、電池で動くノートパソコンやポータブルワープロ、そして、MSXやファミコンのバッテリーバックアップ付きゲームカートリッジにも使われているものだ。

このほかにも、メモリーを使い方によって分類することも可能。図2.1のように、CPUに直結しているメモリーは“主記憶”または“メインメモリー”。つまり、MSX本体の漢字ROM以外のROMと、64キロバイトのメインRAMは、MSXのメインメモリーというわけだ。

またMSXには、このほかに“ビデオRAM(VRAM)”というメモリーもある。ビデオRAMとは、テレビ画面に表示する図形や文字を記憶するためのRAM。コンピューターの機種によっては、ビデオRAMがCPUに直結しているものもあるけど、MSXではVDP(ビデオ・ディスプレー・プロセッサーの略)という専用の部品を経由して、CPUとビデオRAMが接続されている。

ここまで基本的な話は、MSXに限らず、コンピューターについてのもっとも基本的な知識。MSXに付属してくるBASICの入門書などにも、詳しく書かれていると思うから参考にしよう。

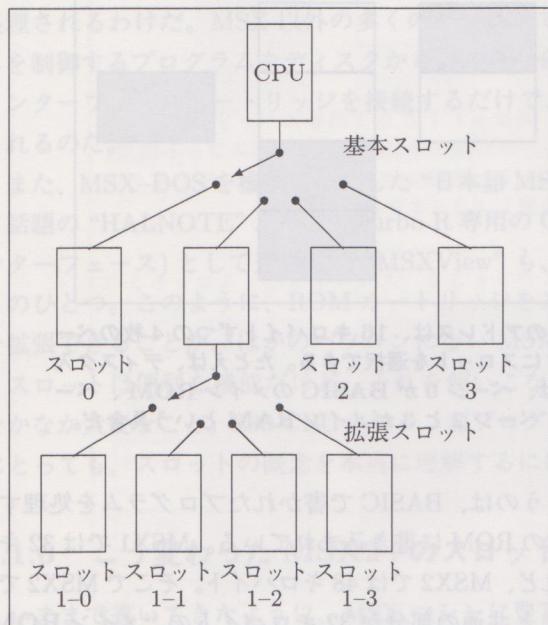
2.1.4 MSXのスロットってどんなものなの?

はじめにも書いたように、8ビットCPUに接続できるメインメモリーは64キロバイト。しかし、これで話が済んでいたのは、初期の8ビットコンピューターだけ。最近はいろいろな方法を使って、64キロバイトを超えるメモリーを接続できるようになっている。

MSXの場合は“スロット切り替え”という方法。図2.2のように、64キロバイトのメモリーを4組用意し、それぞれを切り替えて使うことで、最大256キロバイト

のメモリーを扱うことが可能になる。これらのメモリーは“**基本スロット**”と呼ばれ、マシンに用意されたカートリッジスロットなどにも割り当てられている。

図 2.2: MSX のスロット構成 (その 1)



MSX では、64 キロバイトを越えるメモリーを扱うために、“**スロット切り替え**”という方法を使う。4 組の 64 キロバイトのメモリーを切り替えて、最大で 256 キロバイトのメモリーを扱うわけだ。この 4 組のメモリーを“**基本スロット**”、そこから拡張されるそれぞれ 4 個のスロットを、“**拡張スロット**”と呼ぶ。

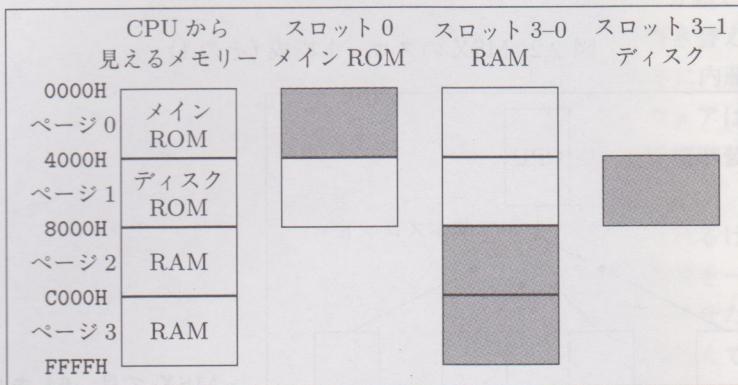
さらに、1 個の基本スロットの代わりに 4 組の“**拡張スロット**”を切り替えて使う方法もある。この場合には、64 キロバイトのメモリーが全部で 16 組。つまり最大で 1 メガ (1024 キロ) バイトのメモリーを接続できるというわけ。ただし、拡張スロットをさらに拡張することはできない。

さて、スロットを切り替えることで、64 キロバイトを越えるメモリーを扱えるのはいいのだけど、その際にメモリー全体が同時に切り替わってしまうのは不便だ。そこで MSX では、メモリーを“**ページ**”という単位に分割して、扱うように考えられている。

メモリーの 0000H~3FFFH 番地までの 16 キロバイトをページ 0、4000H~8000H 番地までの 16 キロバイトをページ 1。同様にして、8000H~BFFFH 番地までをページ 2、C000H~FFFFH 番地までをページ 3 という具合。それぞれ 16 キロバイトを 1 ブロックとしたページごとに、べつべつのスロットを選択できるわけだ。

たとえば、BASIC のディスク入出力関係の命令が処理されているときには、ページ 0 が BASIC インタープリターのメイン ROM、ページ 1 がディスクインターフェースの ROM、ページ 2 と 3 がメイン RAM に切り替えられる (図 2.3 参照)。

図 2.3: MSX のスロット構成 (その 2)



64 キロバイトのメモリーのアドレスは、16 キロバイトずつの 4 枚のページに分割され、各ページごとにスロットを選択できる。たとえば、ディスク入出力の命令を処理する場合は、ページ 0 が BASIC のメイン ROM、ページ 1 がディスクの ROM、ページ 2 と 3 がメイン RAM という具合だ。

BASIC インタープリターというのは、BASIC で書かれたプログラムを処理するプログラムのこと、MSX 本体の ROM に書き込まれている。MSX1 では 32 キロバイトの ROM 入っていたけれど、MSX2 では 48 キロバイト。そこで MSX2 では ROM が 2 個に分けられ、MSX1 と共に部分が 32 キロバイトの “メイン ROM” に、MSX2 で拡張された機能が 16 キロバイトの “サブ ROM” に書き込まれている。

また、ディスク内蔵型の MSX や、外部ドライブのインターフェースカートリッジには、16 キロバイトの ROM が内蔵。BASIC のディスク入出力を処理するためのプログラム (DISK-BASIC) が、書き込まれているというわけ。だから、ディスク入出力中には、図 2.3 のような状態になるわけだ。

2.1.5 MSX の拡張性の秘密はスロットにあった

MSX のスロットは、メモリーを増設するだけでなく、MSX の機能を拡張するためにも使われる。ゲームカートリッジを接続するのも、モデムカートリッジを接続するのもスロットというわけ。

いま書いたように、MSX にディスクインターフェースカートリッジを接続すると、カートリッジ内の 16 キロバイトの ROM がスロットに接続される。そしてディスク入出力が行なわれるときには、自動的にメモリーがディスクインターフェース ROM のスロットに切り替えられるわけだ。このため、ディスクインターフェース自体が本体に内蔵されていても、カートリッジとして接続されていても、プログラ

ムの動作には支障がない。

また、ディスクインターフェースを接続すると、“CALL FORMAT”という命令が、通信カートリッジを接続すると、“CALL TELCOM”という拡張 BASIC の命令が、使えるようになる。これらの拡張命令は、カートリッジ内の ROM によって処理されるわけだ。MSX 以外の多くのパソコンでは、周辺機器を使うときにそれらを制御するプログラムをディスクから読み込む必要がある。ところが MSX では、インターフェースカートリッジを接続するだけで、自動的に BASIC の命令が拡張されるのだ。

また、MSX-DOS を機能アップした“日本語 MSX-DOS2”や、統合化ソフトとして話題の“HALNOTE”、そして turbo R 専用の GUI(グラフィカル・ユーザー・インターフェース)として登場した“MSXView”も、カートリッジで供給されるソフトのひとつ。このように、ROM カートリッジをスロットに接続すると簡単に機能を拡張できることが、ほかのパソコンにない MSX の長所なのだ。

スロットは便利な機能だけど、それを使いこなしたプログラムを開発するのは、なかなか大変なこと。Z80 CPU のマシン語プログラムを自在に書けるプログラマーにとっても、スロットの概念を本当に理解するには 1 年以上かかるかもしれない。

2.1.6 こう変わった MSX2+のスロット

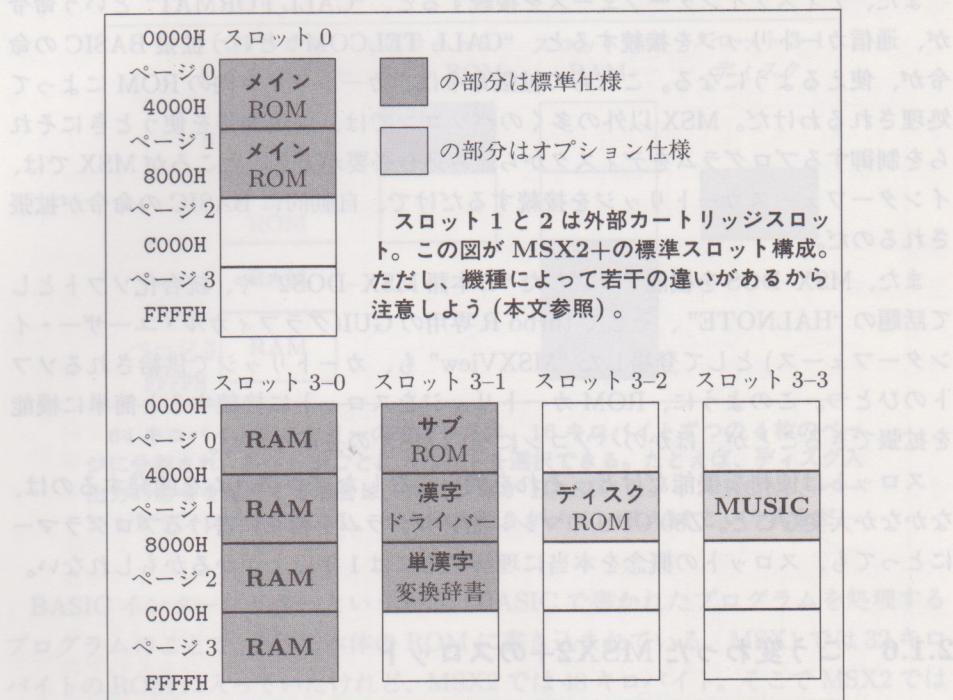
今まで書いてきたように、MSX マシンに豊富な拡張性を持たせ、特徴づけてくれたのがスロットというもの。けれども、このスロットはまた、MSX の弱点でもあった。それが、“従来の MSX では機種によってスロット構成が異なるために、ソフトウェアの互換性の問題が起こりやすい”ということだ。

たとえば、スロット 1 と 3 がカートリッジスロットに割り当てられている機種で、ある特定のソフトが動かない。また、スロット 3 の拡張スロットに RAM が置かれていると、DOS からサブ ROM の機能を使えなかったりするなど。慎重にプログラムを作って、すべての MSX マシンについて動作を確認すれば、こうした問題は避けられるはず。けれども、あらゆるスロット構成のマシンに対応させると、プログラムが長くなったり、実行速度が遅くなったりという弊害も出てくる。一筋縄ではいかないのがスロットというわけだ。

これが MSX2+になって、やっとスロット構成に関するある程度の基準が決められた。図 2.4 と図 2.5 に示したのが、その MSX2+のスロット構成の例。本体に内蔵するソフトウェアの数によって、スロット 3 のみを拡張する場合と、スロット 0 とスロット 3 を拡張する場合とに分けられている。

図 2.4 に掲載したのが、スロット 3 のみを拡張する場合。基本スロット 0 に、BASIC のメイン ROM を置き、スロット 1 とスロット 2 を外部カートリッジスロットとする。

図 2.4: MSX2+のスロット構成の例(スロット3のみを拡張する場合)



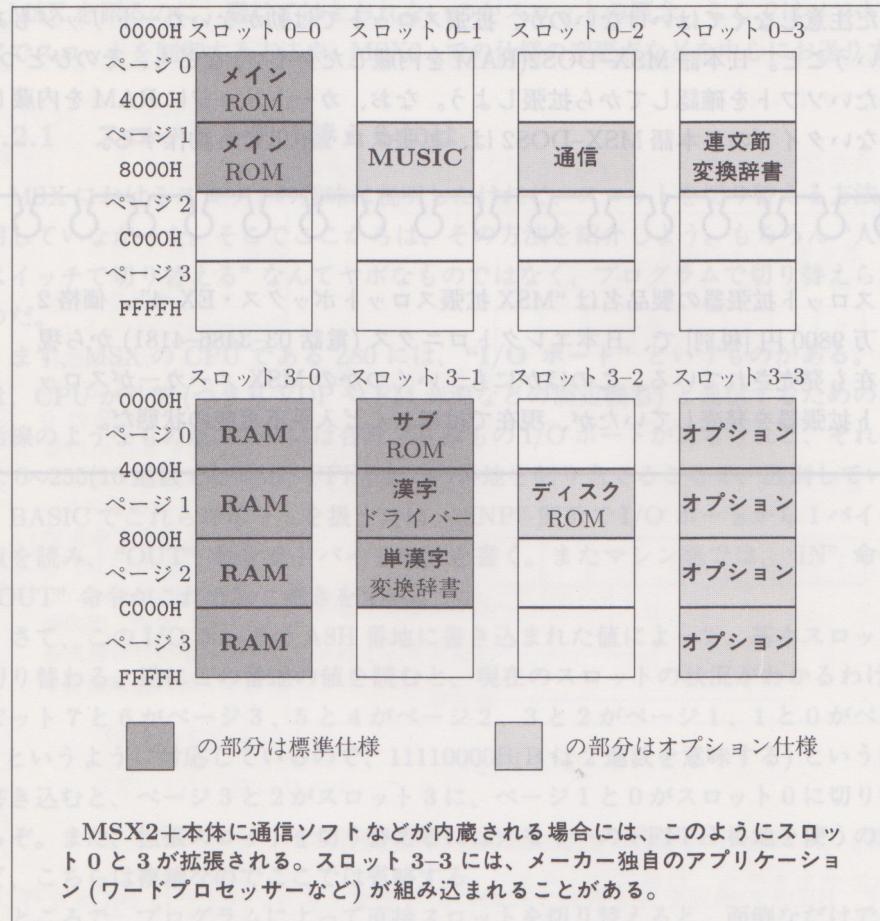
そして、スロット3の拡張スロットのどれかひとつに、64キロバイトのRAM(メインRAM)を、0~3ページまでかならず同じスロットにRAMがくるように置く。同様に、サブROM、漢字ドライバー、単漢字変換辞書の合計48キロバイトのROMを、スロット3の拡張スロットのどれかひとつに置くという具合。図ではスロット3の0(基本スロット3の0番目の拡張スロット)にRAMが、3の1にROMが置かれているけど、これは機種によって異なるわけだ。

これに対し図2.5は、スロット0とスロット3の両方を拡張する場合。基本スロット0の拡張スロット0に、メインROMが置かれている。スロット3の構成は、図2.4の場合とはほぼ同じ。ディスクインターフェースを内蔵する場合には、スロット0の拡張ではなく、かならずスロット3の拡張にROMが置かれることに注意しよう。

なお、図2.4と図2.5の薄い灰色の部分。つまり、ディスクインターフェース、MSX-MUSIC(FM音源)、通信、連文節変換辞書の各ROMに関しては、MSX2+のオプション仕様となっている。そのため、本体に内蔵されずに、外部カートリッジとして接続されることもある。

さて、図2.5と同じだけの内蔵ソフトウェアを持つMSX2+には、理論上36通り

図 2.5: MSX2+のスロット構成の例(スロット0と3を拡張する場合)



のスロット構成が考えられる。「うわー、そんなにあるのか」なんて声も聞こえてきそうだけど、それでも MSX2 のスロット構成よりは、組み合わせの数が減っているのだ。また、サブ ROM と漢字ドライバーと単漢字変換辞書がかならず同じスロットに置かれることで、MSX2+の漢字入出力が予想よりも速くなったはずだ。

2.1.7 スロットを拡張しちゃえ

MSX マシンにひとつかふたつ用意された外部カートリッジスロット(普段ゲームカートリッジを差し込むところ)は、どれも基本スロット。そこで、“スロット拡張器”を接続して、4 個の拡張スロットに拡張することができる。たとえば本体の 2 個

のスロットの両方に、スロット拡張器を接続すれば、合計8個ものスロットが誕生するわけだ。

ただ注意しなくてはいけないのが、拡張スロットでは動かないカートリッジもあるということ。日本語 MSX-DOS2(RAM を内蔵したタイプ)なども、そのひとつ。使いたいソフトを確認してから拡張しよう。なお、カートリッジに RAM を内蔵していないタイプの日本語 MSX-DOS2 は、拡張スロット上でも動作する。

H0008

スロット拡張器の製品名は“MSX 拡張スロットボックス・EX-4”。価格 2 万 9800 円 [税別] で、日本エレクトロニクス（電話 03-3486-4181）から現在も発売されている。このほかにも、いくつかの MSX メーカーがスロット拡張器を発売していたが、現在ではほとんど入手不可能の状態だ。

拡張スロット3の構成 拡張スロット3の構成は、スロット3と ROM の接続状況である。
 そして、スロット3の拡張スロット4でそれもひとつに、64キロバイトのRAM（ノンRAM や RAMコドも）を組み込んだ形で内蔵するROMをスロットにRAMがくるように置く。同じく、拡張スロット3の拡張スロット4で、他のスロットにROMがくるように置く。
 同様に、拡張スロット3と拡張スロット4で、他のスロットにROMがくるように置く。
 ROMを、拡張スロット3の拡張スロット4で、他のスロットにRAMがくるように置く。
 拡張スロット3の拡張スロット4で、他のスロットにROMがくるように置く。
 これが複数によって異なるわけだ。

2.2 スロット切り替えに挑戦

MSXを語るのに、避けてはとおれないのがスロットの概念。ここではソフトウェアでスロットを制御する方法や、MSX2+での仕様の変更点などを中心にお送りする。

2.2.1 スロットを切り替えるには

MSXにおけるスロットの意味は説明したけれど、スロットを切り替える方法は説明していなかった。そこでここからは、その方法を紹介しよう。もちろん“人間がスイッチで切り替える”なんてヤボなものではなく、プログラムで切り替えられるのだ。

まず、MSXのCPUであるZ80には、“I/Oポート”というものがある。これは、CPUが外部(つまりVDPやFM音源などの周辺機器)と通信するための、電話線のようなものだ。Z80には合計256本ものI/Oポートがあるけれど、それぞれに0~255(16進数では00H~FFH)までの番地を割り当てることで、区別している。

BASICでこれらのポートを扱うには、“INP”関数でI/Oポートから1バイトの値を読み、“OUT”命令で1バイトの値を書く。またマシン語では、“IN”命令と“OUT”命令がこれと同じ働きをする。

さて、このI/OポートのA8H番地に書き込まれた値によって、基本スロットが切り替わる。逆にこの番地の値を読むと、現在のスロットの状況がわかるわけだ。ビット7と6がページ3、5と4がページ2、3と2がページ1、1と0がページ0というように対応しているので、11110000B(Bは2進数を意味する)という値を書き込むと、ページ3と2がスロット3に、ページ1と0がスロット0に切り替わるぞ。また、拡張スロットを切り替えるにはメモリーのFFFFH番地を使うのだけれど、こちらは複雑なのでここでは省略する。

ところで、プログラムによって直接スロットを切り替えると、面倒なだけではなく、機種によってはプログラムが動かないといった、互換性の問題が起こりやすい。そこで実際には、“BIOS”によってスロットを切り替える。BIOSとは、“Basic Input Output System”という意味。あとでくわしく説明するけど、ハードウェアを制御するための、マシン語サブルーチンの集まりだ。これにはスロットを切り替える以外にも、多くの機能があるのでチェックしよう。

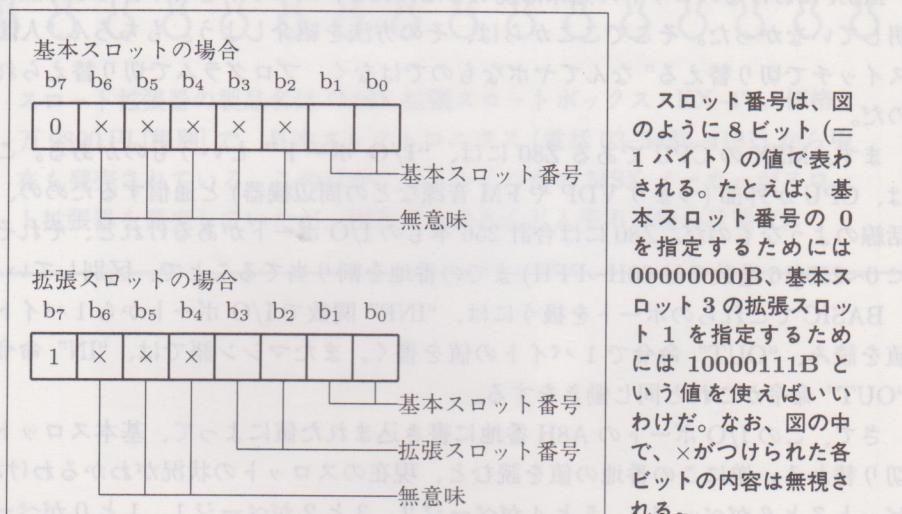
2.2.2 スロット番号の指定方法

BIOSを使ってスロットを切り替える場合には、基本スロット番号と拡張スロット番号を、それぞれべつに指定すればいい。でもそのためには、2個のレジスター(CPU内のデータの一時記憶場所)が必要になってしまい不経済(?)だ。そこで、図

2.6 のように 8 ビット (1 バイト) の各ビットをうまく使って、基本スロットと拡張スロットをまとめて指定する方法が取られている。

たとえば基本スロット 0 を指定するには、00000000B(16 進数では 00H) という値を、基本スロット 3 の拡張スロット 1 を指定するには、10000111B(87H) という値を指定すればいい。

図 2.6: スロット番号の指定方法



2.2.3 スロットを操作する BIOS の機能

まず、BIOS の機能を書き表わすための記号を覚えよう。**E** とは BIOS を呼び出す前に値を設定すべきレジスター。**R** は BIOS が値を返すレジスター。**M** は BIOS が無意味な値を書き込む、つまり元の内容が壊されるレジスターを表わす。また **IYH** とは、IY レジスターの上位バイトを表わし、下位バイトの内容は無視される。またはじめに書かれた番地は、その BIOS を呼び出すためのエントリーアドレスだ。

RDSLTH 000CH 番地

機能

A レジスターで指定されたスロットの、HL レジスターで指定された番地の内容を読む。

E

A スロット番号

HL 番地

<input type="checkbox"/> R	A 読んだ値
<input type="checkbox"/> M	AF、BC、DE
<input type="checkbox"/> 注	割り込みが禁止される。

WRSLT 0014H 番地

<input type="checkbox"/> 機能	A レジスターで指定されたスロットの、HL レジスターで指定された番地に、E レジスターの内容を書き込む。
<input type="checkbox"/> E	A スロット番号
	HL 番地
	E 書き込む内容
<input type="checkbox"/> R	なし
<input type="checkbox"/> M	AF、BC、D
<input type="checkbox"/> 注	割り込みが禁止される。

CALSLT 001CH 番地

<input type="checkbox"/> 機能	ほかのスロットにあるサブルーチンを呼び出す。
<input type="checkbox"/> E	IX 呼び出す番地
	IYH スロット番号
<input type="checkbox"/> R	呼び出す相手による
<input type="checkbox"/> M	IX、IY、裏レジスター
<input type="checkbox"/> 注	現在のスロットの状態をスタックに保存し、目的のサブルーチンをコールする。AF、BC、DE、HL レジスターの内容は、そのままサブルーチンに渡され、サブルーチンがRET命令を実行すると、元のプログラムに戻る。このときも、AF、BC、DE、HL レジスターの値はサブルーチンから渡される。何バイトのスタックが使われるかどうかは、スロット構成によって異なる。

ENASLT 0024H 番地

<input type="checkbox"/> 機能	スロットを切り替える。
<input type="checkbox"/> E	A スロット番号
	H ページ(上位 2 ビット)
<input type="checkbox"/> R	なし
<input type="checkbox"/> M	AF、BC、DE、HL

注 たとえば、2ページを切り替えるためには、Hレジスターに80H～BFHの値を設定すればよい。割り込みが禁止される。

CALLF 0030H 番地

機能 ほかのスロットにあるサブルーチンを呼び出す。

E 以下のプログラムのように、“RST 30H”命令に続けてスロット番号と番地をプログラムに書き込んでおく。

RST	30H
DB	スロット番号
DW	番地

R 呼び出す相手による

M IX、IY、裏レジスター

注 スロットと番地の指定方法以外は、CALS LTと同じ。特別な目的(フック)に使う。

EXTROM 015CH 番地

機能 サブ ROM を呼び出す。

E IX呼び出す番地

R 呼び出す相手による

M IX、IY、裏レジスター

注 自動的にサブ ROM のスロットが選択される以外は、CALS LTと同じ働きをする。

と、以上紹介してきた BIOS には若干の制限がある。どれもページ 3 に対しては使えない。ページ 0 に対しては DOS からメイン ROM を呼び出す場合にのみ使える。ページ 2 と 3 に対しては問題なく使える、ということだ。“使えない”といっても、スロット構成によっては使えることもあるから、“自分の MSX だけで動くプログラム”を作らないように注意しよう。とくに DOS からサブ ROM を呼び出そうとして CALS LT を使うと、スロット構成とディスクインターフェースの種類によって、動いたり動かなかったりするぞ。

2.2.4 スロット構成を知る方法

前にも説明したように、MSX のスロット構成は機種によって異なる。ディスクインターフェースのようなオプション仕様もあるため、マシンの数だけスロット構成

があるといつても過言ではない。そこで、自分のマシンのスロット構成と、オプション機器の有無を調べる方法を紹介しよう。

メモリーのF380H～FFFEH番地までを“システムワークエリア”といい、ここにはBIOSなどにとって重要な情報が記憶されている。ディスクインターフェースが接続されれば、システムワークエリアより少し番地が小さい場所に“ディスクワークエリア”が用意されるわけだ。またスロットに関する情報は、表2.1のようにシステムワークエリアとディスクワークエリアに記憶されている。

メインRAMがどのスロットにあるかという問題は重要だけど、表2.1の“RAMADO”などはディスクワークエリア内にある。そのため、ディスクがないとこれらの情報はわからないという問題もある。

リスト2.1は、これらのシステムワークエリアから調べたスロット構成を、わかりやすく表示するプログラムだ。機種によって構成が違ってくるから、自分のMSXでも試してみよう。

表2.1に掲載した以外にも、プログラムの役に立つシステムワークエリアがあるけど、詳細は“MSX2テクニカルハンドブック”などを見てほしい。それから、これらのシステムワークエリアは、とくに指示される場合を除いて、アプリケーションプログラムが書き替えてはいけない。メモリーが不足して苦しまぎれにシステムワークエリアを使うプログラムがあるけど、互換性をなくすもとので注意しよう。

表2.1: スロットに関するシステムワークエリア

名称	番地	意味
RAMADO	F341H	ページ0のRAMのスロット番号(1)
RAMAD1	F342H	ページ1のRAMのスロット番号(1)
RAMAD2	F343H	ページ2のRAMのスロット番号(1)
RAMAD3	F344H	ページ3のRAMのスロット番号(1)
MASTER	F348H	ドライブAのインターフェースのスロット番号(1)
EXBRSA	FAF8H	サブROMのスロット番号(MSX1では0)
EXPTBL	FCC1H	メインROMのスロット番号
	FCC2H	スロット1が拡張されているかどうか(2)
	FCC3H	スロット2が拡張されているかどうか(2)
	FCC4H	スロット3が拡張されているかどうか(2)

(1) ディスクがある場合のみ有効。
 (2) 拡張されていれば80H、そうでなければ0。

2.2.5 システムワークエリアを探ってみる

MSX2用のゲームソフトの中に、MSX2+で動かせばSCREEN12のタイトル画面を表示するようなものがある。また、モデムカートリッジがないのに通信しよう

とすると、親切にエラーメッセージを表示するプログラムもある。こんなソフトを作るために、プログラムがハードの種類や構成を調べる方法を紹介する。

まず“ディスクがあるかどうか”を調べるために、FFA7H 番地の内容を読む。もし C9H であればディスクがなく、そのほかの値であればディスクがある。

“MSX の種類”を調べるために、メイン ROM の 2DH 番地を読む。0 ならば MSX1、1 ならば MSX2、2 ならば MSX2+、そして 3 ならば turbo R というわけ。

一般に、2BH 番地と 2CH 番地の内容は 0 だけど、“海外への輸出用に作られた MSX”では、キーボードや通貨記号の種類を表わす番号が入っている。輸出用ソフトウェアを作る場合だけ気にすればいいので、番号の一覧は省略する。

これは余談になるけど、MSX パソコンはヨーロッパをはじめ、ソビエトや、中近東のクウェートなどにも相当数が輸出されている。また、お隣の韓国では、学校に多数導入され、授業に役立てられているとか。なんともインターナショナルなマシンなのだ。

さて、“ビデオ RAM 容量”を調べるには、FAFCH 番地を読む。ビット 2 とビット 1 の値が、00 ならば 16 キロバイト、01 ならば 64 キロバイト、10 ならば 128 キロバイトだ。これ以外のビットはべつの目的に使われているようなので無視しよう。次のページのリストのように“AND 6”でビット 2 とビット 1 の値を取り出し、それを 2 で割ればいい。

このリストでは、おまけとして“拡張 BIOS”的有無も調べている。これは、通信モ뎀、FM 音源、漢字辞書といった、オプションハードウェアを制御するための機能だ。FB20H 番地のビット 0 の内容が 1 で、FFCAH 番地の内容が C9H でなければ、何らかの拡張 BIOS 機能があることになる。それが何であるか調べるには、複雑なマシン語のプログラムが必要になるので、今回はパス。それから、これは仕様書には書かれていないのだけど、FFCBH 番地の内容は、拡張 BIOS 機能を持っているプログラムのスロット番号らしい。

なお、意味が決められていないビット、たとえばスロット番号を表わす値の、ビット 6 からビット 4 などには、何が書き込まれているかわからない。そこで、“AND”を使って、その内容を無視していることがわかるかな。

何度も書くようだけど、たとえ同じメーカーのマシンであっても、機種によってスロット構成が異なることがある。だから、自分の持っているマシンで試したあとは、友だちのマシンでも試してみよう。多くのマシンでテストして、その結果を表にしてみるとおもしろいぞ。

2.2.4 スロット構成を知る方法^④ 紹介マニュアルモード

この章では、MSX の構成情報を読み取る方法を紹介する。MSX の構成情報を読み取る方法は、MSX の構成情報を読み取る方法である。

リスト 2.1 (WHO_AM_I.BAS)

```

100 ' Analizing slot structure of MSX
110 ' by nao-i on 9. Jan. 1989
120 CLEAR : DEFINT A-Z : CLS
130 VE = PEEK(&H2D) : ' Version No. of BASIC
140 IF VE=0 THEN PRINT "I am MSX1"
150 IF VE=1 THEN PRINT "I am MSX2"
160 IF VE=2 THEN PRINT "I am MSX2+"
165 IF VE=3 THEN PRINT "I am MSX turbo R"
170 IF VE>3 THEN PRINT "Who am I ?"
180 VR = (PEEK(&HFAFC) AND 6) ¥ 2 : ' size of VRAM
190 IF VR=0 THEN PRINT "VRAM 16KB"
200 IF VR=1 THEN PRINT "VRAM 64KB"
210 IF VR>1 THEN PRINT "VRAM 128KB"
220 MR = PEEK(&HFC49) : ' size of main RAM
230 IF MR >= &HE0 THEN PRINT "RAM 8KB"
240 IF MR < &HE0 AND MR >= &HC0 THEN PRINT "RAM 16KB"
250 IF MR < &HC0 THEN PRINT "RAM >= 32KB"
260 FOR SS = 0 TO 3 : 'EXPTBL
270   PRINT USING "Slot # is "; SS;
280   FF = PEEK(&HFCC1 + SS) AND 128
290   IF FF THEN PRINT "expanded slot" ELSE PRINT "primary slot"
300 NEXT SS
310 PRINT
320 SS = PEEK(&HFCC1) : PRINT "Main ROM is in "; : GOSUB 530
330 SS = PEEK(&HFAF8) : PRINT "Sub ROM is in "; : GOSUB 530
340 IF PEEK(&HFFA7) <> &HC9 THEN 360
350 PRINT "I have no disk." : GOTO 450
360 PRINT "I have disk(s)."
370 SS = PEEK(&HF348) : PRINT "FDC ROM is in "; : GOSUB 530
380 SS = PEEK(&HF341) : PRINT "P0 RAM is in "; : GOSUB 530
390 SS = PEEK(&HF342) : PRINT "P1 RAM is in "; : GOSUB 530
400 SS = PEEK(&HF343) : PRINT "P2 RAM is in "; : GOSUB 530
410 SS = PEEK(&HF344) : PRINT "P3 RAM is in "; : GOSUB 530
420 PRINT "Bottom address of disk work area is ";
430 PRINT RIGHT$("00"+HEX$(PEEK(&HFC4B)),2);
440 PRINT RIGHT$("00"+HEX$(PEEK(&HFC4A)),2)
450 ' detectiong extended BIOS
460 IF (PEEK(&HFB20) AND 1) = 0 THEN GOTO 520
470 IF PEEK(&HFFCB) = &HC9 THEN GOTO 520
480 PRINT : PRINT "I have extended BIOS."
490 SS = PEEK(&HFFCB)
500 PRINT "ROM of the extended BIOS may be in "
510 GOSUB 530
520 END
530 ' displaying slot number
540 PRINT USING "primary slot #"; SS AND 3;
550 IF (SS AND 128) = 0 THEN 570
560 PRINT USING " extended slot #"; (SS AND 12) ¥ 4;
570 PRINT : RETURN

```

2.2.6 MSX2+のハードウェア仕様

MSX2+には、ハードウェアの細かい改良点が加えられた。表2.2にまとめたのが、新しく仕様が定義または追加されたI/Oポートだ。

表2.2: MSX2+のI/Oポート

I/O番地	用途
7CH	本体内蔵FM音源
7DH	本体内蔵FM音源
DAH	第2水準漢字ROM
DBH	第2水準漢字ROM
F4H	初期化の制御
F5H	デバイスイネーブル

これが新しく仕様が定義または追加されたもの。ただし、実際のプログラムでは、I/Oポートを直接使わずにBIOSを使うほうがよい。

7CH番地と7DH番地は、本体に内蔵されたFM音源を操作するためのI/Oポート。これとは違い、カートリッジで供給されるタイプのFM音源は(今後もし発売されるなら)、パナソニックの“FM-PAC”と同じI/Oポートを使う。

FM音源が本体に内蔵されているかどうかを調べるために、各スロットについて、4018H番地から401FH番地までを読む。その内容が“APRLOPLL”という文字列と一致すれば、そのスロットにFM音源を制御するプログラムのROMがあり、本体にFM音源が内蔵されているというわけだ。

一方、FM音源カートリッジの場合には、4018H番地からの内容が“PAC2OPPLL”的ように、製品の種類を表わす4文字と“OPPLL”いう文字になるようだ。

“MSX-Write”や一部のモデムカートリッジには、最初のメニューで“BASIC”を選ぶと、リセットされたようにMSXのタイトル画面が表われるものがある。これは、ソフトウェアの準備の都合で、メインROMの0番地へジャンプして、リセットと同じような処理をさせているからだ。

以前は、0番地へのジャンプと本当のリセットを確實に区別する方法がなかったので、ソフトウェアの誤動作が起きたことがあった。それがMSX2+からは、I/OポートのF4H番地にリセットの状態を調べるためのハードウェアが追加された。ただし、実際には、次のようにMSX2+のメインROMに追加されたBIOSを使う。

```

CALL 17AH
OR 80H
CALL 17DH
JP 0

```

初期化時に呼び出されたROMカートリッジのプログラムは、

2.2.6 CALL 17AH

を行なう。そして、A レジスターのビット 7 が 0 ならば本当のリセット。1 ならばジャンプ 0 で、自分が呼び出されたことがわかるというわけだ。

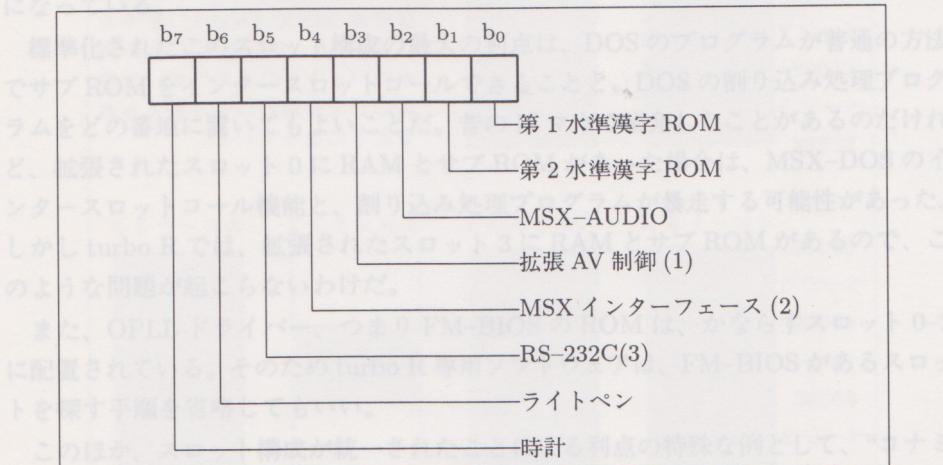
2.2.7 衝突を防ぐデバイスイネーブル

漢字 ROM を内蔵している MSX に漢字 ROM カートリッジを接続すると、漢字が正しく表示されないだけでなく、ハードウェアが衝突して故障する危険がある。これを防ぐありがたい機能が、I/O ポートの F5H 番地で制御される“デバイスイネーブル”というものだ。

図 2.7 に書かれているハードウェアは、リセット時にバスから切り離されている。そして、I/O ポートの F5H 番地の 1 バイト (8 ビット) の値を書き込むことで、1 になったビットに対応する内蔵ハードウェアが、バスに接続されるわけだ。これらの処理は、リセットまたはジャンプ 0 (ソフトウェアによって、メイン ROM の 0 番地にプログラムの実行が移ること) のあとで自動的に行なわれる。

MSX2 では、I/O ポートの F5H 番地に 0 を書き込んだ場合、既に接続されているハードウェアを切り離すかどうかの、規定がされていなかった。このため “MSX-

図 2.7: デバイスイネーブル



I/O ポートの F5H 番地により、本体内蔵のハードウェアを有効にするか無効にするかを選択する (1 を書き込まれたビットに対応するハードウェアが有効)。

(1) I/O ポートの F7H 番地で制御されるスーパーインポーズ機能など。

(2) 仕様書にあるが実用化されていない。

(3) モデムには関係ない。

かつ確実に実現できるような環境が整ったわけだ。

Write”などが、ROMの0番地へジャンプして BIOS を再度初期化しようとすると、混乱が起ることもあったわけだ。

それが MSX2+からは、0を書き込めば、内蔵ハードウェアがバスから切り離されるように統一された。これにより、I/O ポートの F4H 番地と F5H 番地を活用して MSX2+用の基本ソフトウェアを作ることで、互換性と信頼性が、今まで以上によくなるわけだ。

2.3 MSX turbo R のスロット構成

ここでは、新しく発表された、MSX turbo R のスロット構成について解説する。特筆すべきは、ここにきて、やっとのことでの、スロット構成が統一されたことだ。これは、なんとも意義深いことなのだ。

2.3.1 ついにスロット構成が統一されたぞ

図 2.8 が、turbo R のスロット構成だ。CPU の高速化に対応し、アプリケーションプログラムの開発やデバッグを容易にするために、スロット構成が統一された。

この図では、スロット 3-0 に 64 キロバイトの RAM があるよう見えるけど、実際にはメモリーマッパーをとおして、256 キロバイトのメイン RAM が接続されている。このうち 64 キロバイトを越える部分は、日本語 MSX-DOS2 のワークエリアや RAM ディスク、べつの章で説明する “DRAM モード” などに、通常は使われる。でも、アプリケーションプログラムが拡張 BIOS を使ってマッパーを切り替え、これらの RAM を使うことも可能だ。

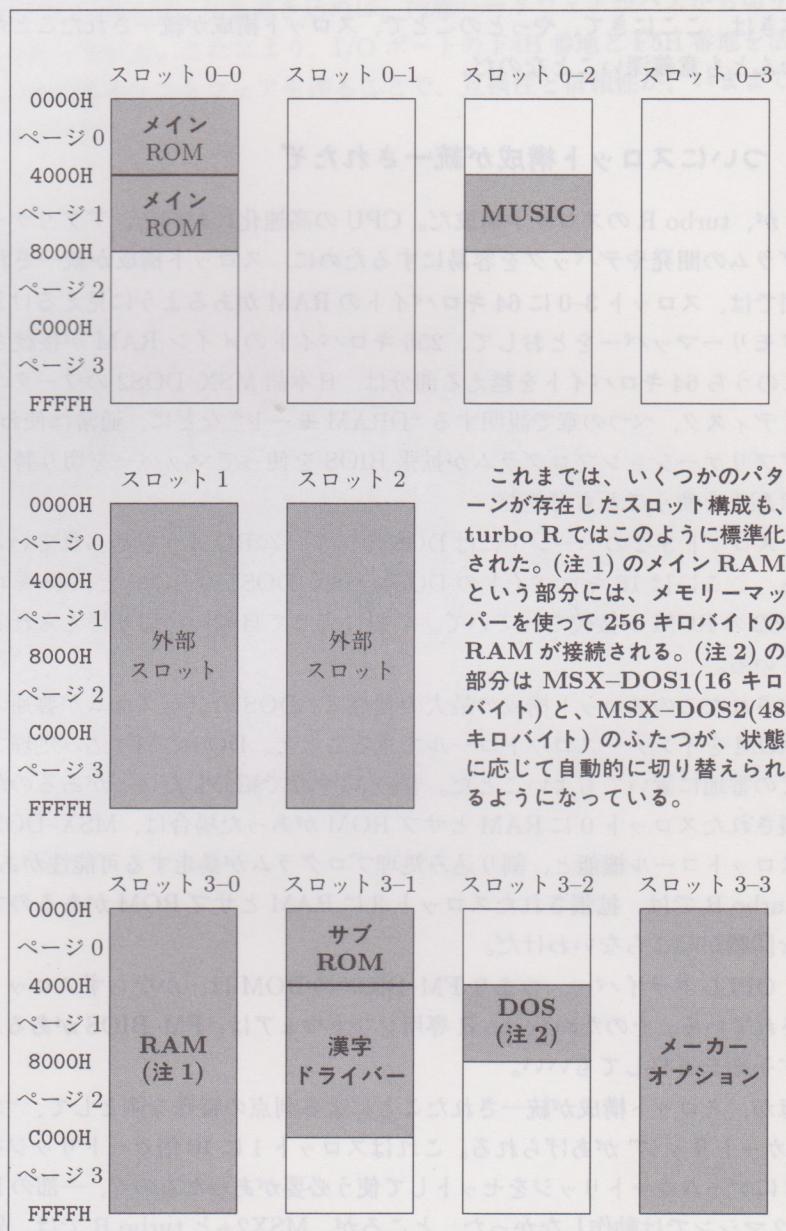
また、スロット 3-2 のページ 1 には DOS のシステム ROM が収められている。といっても、ここには 16 キロバイトの DOS1(MSX-DOS) の ROM と、48 キロバイトの DOS2 の ROM が接続されていて、必要に応じて自動的に切り替えられるようになっている。

標準化されたこのスロット構成の最大の利点は、DOS のプログラムが普通の方法でサブ ROM をインタースロットコールできることと、DOS の割り込み処理プログラムをどの番地に置いてもよいことだ。昔の M マガで紹介したことがあるのだけれど、拡張されたスロット 0 に RAM とサブ ROM があった場合は、MSX-DOS のインターフェースコール機能と、割り込み処理プログラムが暴走する可能性があった。しかし turbo R では、拡張されたスロット 3 に RAM とサブ ROM があるので、このような問題が起こらないわけだ。

また、OPLL ドライバー、つまり FM-BIOS の ROM は、かならずスロット 0-2 に配置されている。そのため turbo R 専用ソフトウェアは、FM-BIOS があるスロットを探す手順を省略してもいい。

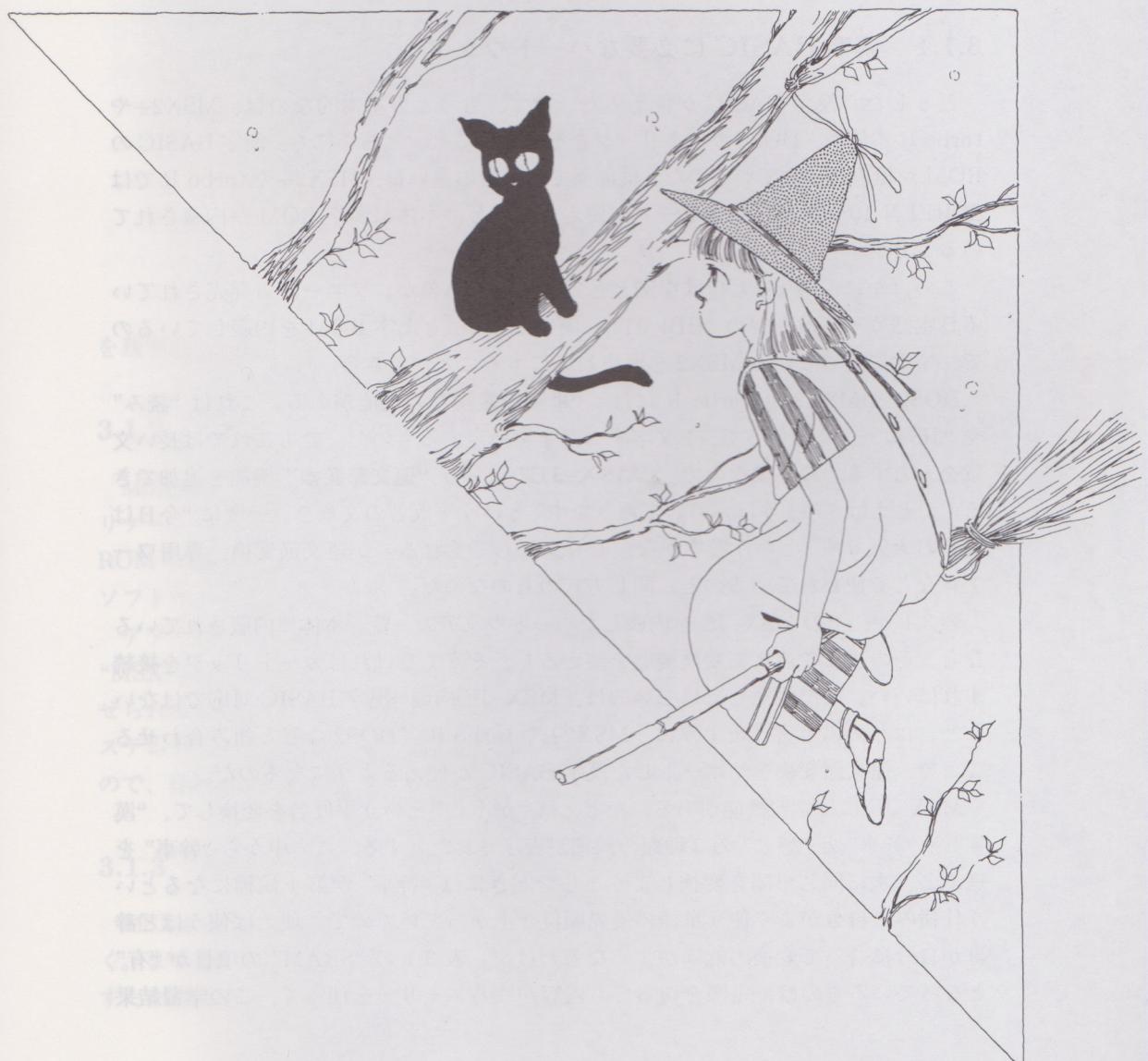
このほか、スロット構成が統一されたことによる利点の特殊な例として、“コナミの 10 倍カートリッジ” があげられる。これはスロット 1 に 10 倍カートリッジを、スロット 2 にゲームカートリッジをセットして使う必要があったもので、一部の MSX1 と MSX2 マシンでは動作しなかった。ところが、MSX2+ と turbo R では、外部スロットがスロット 1 と 2 に決められたので、こうした特殊なプログラムも、簡単にかつ確実に実現できるような環境が整ったわけだ。

図 2.8: MSX turbo R のスロット構成



またソフトウェアハウスにとっては、スロット構成が統一されたために、特定のスロット構成で発生するバグに悩まされることが減るのが、最大の利点といえる。ソフトウェアを作る立場からすると、CPU の高速化や RAM 容量が増設されたことよりも、スロット構成の統一のほうがはるかにウレシイのだ。turbo R 万歳！

第3章 漢字BASIC



この章は、MSX マガジン 1989 年 4 月号の“MSX2+テクニカル探検隊”の記事を再編集したものである。

3.1 漢字 BASIC を解析

MSX2+以降のマシンや DOS2 の特長のひとつは、漢字が使いやすくなつたこと。BASIC のプログラムの文字列やファイル名にも、漢字を使うことができる。この章では、その漢字 BASIC の機能をレポートする。

3.1.1 漢字 BASIC に必要なハードウェア

どうしたら漢字 BASIC が使えるか。まず、もっとも基本的なのは、MSX2+や turbo R 本体か、DOS2 カートリッジをそろえること。どちらにも、漢字 BASIC の ROM が組み込まれているのだ。機能面での両者の違いは、MSX2+や turbo R では SCREEN 10~12 の自然画モードを使えることと、本体に漢字 ROM が内蔵されていること。

このほかに、ちょっと特殊な例として注目したいのが、ソニーから発売されている日本語カートリッジの“HBI-J1”。漢字 BASIC と漢字 ROM を内蔵しているので、すでに持っている MSX2 を漢字対応にすることができる。

DOS2 と MSX2+、turbo R には、“単漢字変換”の機能がある。これは“読み”や“JIS コード”を使って、1 文字ずつ漢字を指定するものだ。でもこれでは長い文章を入力するのに不便なので、“MSX-JE”という“連文節変換”機能を追加できる。たとえば“きょうはいいお天んきです”という平仮名の文章を、一度に“今日は良いお天気です”という漢字かな混じり文にしてくれるのが連文節変換。専用ワープロなどで使われているのと同じ方式のものなのだ。

表 3.1 が、その MSX-JE を内蔵したハードウェアの一覧。本体に内蔵されているなら、そのままで連文節変換機能を使えるし、そうでなければカートリッジを接続すればいい。ただ注意してほしいのは、MSX-JE 内蔵=漢字 BASIC 対応ではないこと。はじめにも書いたように、MSX2+や turbo R、DOS2 などと組み合わせることで、連文節変換をサポートした漢字 BASIC が使えるようになるのだ。

MSX-JE には学習機能がある。たとえば“かんじ”という平仮名を変換して、“漢字”と“幹事”と“感じ”的 3 種類の候補が表示されたとする。この中から“幹事”を選ぶと、次に同じ単語を変換しようとしたときには“幹事”が第 1 候補になるという仕組み。自分がよく使う単語の優先順位が上がっていくので、使えば使うほど辞書が自分に合って変換の能率がよくなるわけだ。表 3.1 の“SRAM”の項目が“有”となっているものは、電源を切っても内容が残るメモリーを使って、この学習結果

表 3.1: MSX-JE 内蔵ハードウェア一覧

メーカー	製品名	形態	漢字 ROM	SRAM
パナソニック	FS-A1ST	MSX turbo R	1、2	有
パナソニック	FS-A1WSX	MSX2+	1、2	有
パナソニック	FS-A1WX	MSX2+	1、2	有
パナソニック	FS-SR021	カートリッジ(1)	1、2	有
パナソニック	FS-4600F	MSX2	1	有
パナソニック	FS-PW1	プリンター(2)	1	有
ソニー	HB-F1XV	MSX2+	1、2	有
ソニー	HB-F1XDJ	MSX2+	1、2	有
ソニー	HBI-J1	カートリッジ	1、2	有
サンヨー	PHC-77	MSX2	1	無
HAL 研究所	HALNOTE	カートリッジ(3)	1、2	有
アスキー	MSX-Write	カートリッジ	1	無
アスキー	MSX-Write II	カートリッジ	1、2	有

漢字 ROM の項目は、第 1、第 2 水準漢字 ROM の有無。SRAM“有”は、電源を切っても学習が残るもの。(1) A1WX のワープロをカートリッジ化。(2) カートリッジと専用プリンターのセット。(3) カートリッジとディスクによる専用 OS。

を残すようになっている。

3.1.2 MSX-JE 対応のソフトウェアとは

MSX-JE はただのワードプロセッサーや拡張 BASIC ではなく、いろいろなアプリケーションと組み合わせて、連文節変換の機能を使えるようになっている。漢字 ROM や連文節変換辞書の ROM などは比較的高価なので、MSX-JE 自体を多くのソフトウェアが共有することは、とっても経済的なのだ。

アプリケーションが MSX-JE を利用する方法は仕様書で定められているので、“MSX-JE 対応”と表示されているソフトウェアは、どの MSX-JE とでも組み合わせられるようになっている。ただし、表 3.1 の HALNOTE は、カートリッジとシステムディスクの組み合わせで “HALOS” という専用 OS を使うようになっているので、HALOS 用でないソフトウェアとの組み合わせには制限がある。

3.1.3 漢字ドライバーの動作原理を解説する

MSX に限らず、コンピューターの内部で漢字を表わす方法を、簡単に説明しておく。まず英字とカタカナは 1 バイト (8 ビット) の値で表わせるけれど、漢字を表わすためには 2 バイトの符号が必要になる。JIS(日本工業規格) では、次のように漢字

を2バイトの符号で表わしている。

亜……3021H(第1水準)

腕……4F53H(第1水準)

式……5021H(第2水準)

龠……737EH(第2水準)

機種	音品算	一々一々
MSX turbo R	E-S-A121	セヤニシナハ
MSX2+ / MSX-JE	E-S-A121	セヤニシナハ
MSX2+ / MSX-JE	E-S-A121	セヤニシナハ
MSX2+ / MSX-JE	E-S-A121	セヤニシナハ

しかし、この“JIS コード”には、英字と漢字が混ざると処理が面倒になるという問題がある。そこで、JIS コードをコンピューターが処理しやすいように変換した“シフト JIS コード”が、MSX を含む多くのパーソナルコンピューターで使われている。一部では“マイクロソフト漢字コード”とも呼ばれているけど、これはまあ興味のある人だけ覚えておこう。

いずれにせよシフト JIS コードは、英文字用に作られたソフトウェアを、そのまま、あるいはわずかな修正で、日本語でもだまして使えるように設計された便利な漢字コード。16 ビットパソコンでもっとも普及しているオペレーティングシステムの MS-DOS や、OS-9 といった最近話題のオペレーティングシステムにも、シフト JIS 漢字コードが使われている。だから、異なるコンピューター間で文書ファイルを交換することも、できるというわけだ。

ところで、漢字コード表を見ていると、数字や英字が混じっていることに気づく。混乱を避けるために、2バイトの漢字コードで表わされる英字や漢字などを総称して“全角文字”。これに対し、1バイトの文字コードで表わされる文字を“半角文字”と呼び、両者を区別することにする。たとえば、半角文字の“ABCD”と全角文字の“ABC D”は、まったくべつの文字として扱われるから注意しよう。

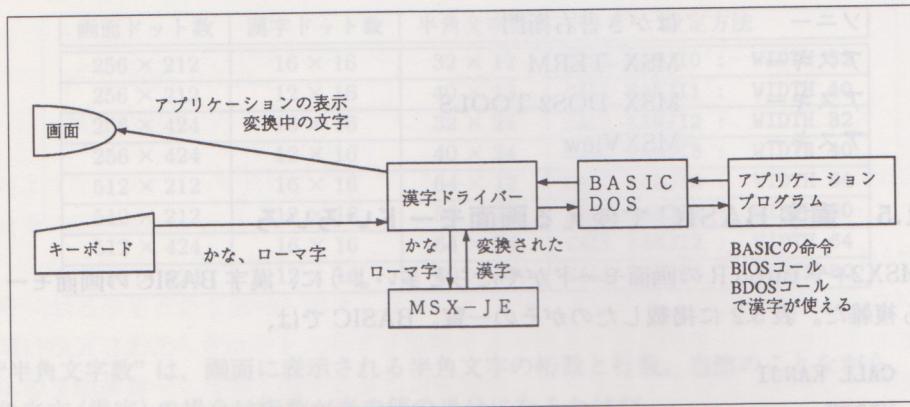
それでは、いよいよ本題に入るぞ。MSX2+と turbo R、DOS2 の漢字入出力機能は“漢字ドライバー”と呼ばれている。これは、BASIC や DOS に限らず、多くのアプリケーションプログラムが利用できるようになっているものだ。

図3.1 にあるのが、漢字ドライバーの動作原理。順を追って説明すると、まずキーボードから入力された全角の“かな”または“ローマ字”を読んで(判断して)、画面に表示する。次に MSX-JE を呼び出し、ひらがなを漢字(全角文字)に変換。最後に、変換された漢字のコードが、BASIC または DOS をとおして、アプリケーションプログラムに送られるというわけ。

これとは逆に、アプリケーションプログラムが画面に文字を表示する場合は、漢字コードを漢字ドライバーに送ればいいわけだ。

この漢字ドライバーは、アプリケーションプログラムにとって BASIC の命令や BIOS コール、BDOS コール(DOS のプログラムが入出力を行なうための、BIOS に似た機能)などに、漢字入出力機能が追加されたように見えるだけ。とくに、アプ

図 3.1: 漢字ドライバーの動作原理



リケーションプログラム自身が MSX-JE の辞書を操作しなくてよいことが、漢字ドライバーの大きな利点だ。

3.1.4 JE 対応ハード&ソフト

ここでは参考までに、これまでに発売された MSX-JE 対応のハードとソフトをまとめてみた。

● モデムカートリッジ

パナソニック	FS-CM1
パナソニック	FS-CM820
ソニー	HBI-1200
キヤノン	VM-300
明星電気	V-3

● モデム内蔵 MSX

パナソニック	FS-A1FM
ソニー	HB-T7
ソニー	HB-T600
三菱	ML-TS2H

- ソフトウェア

ソニー	文書作左衛門
ソニー	はがき書右衛門
アスキー	MSX-TERM
アスキー	MSX-DOS2 TOOLS
アスキー	MSXView

3.1.5 漢字 BASIC で使える画面モードいろいろ

MSX2+や turbo R の画面モードがやたらと多いように、漢字 BASIC の画面モードも複雑だ。表 3.2 に掲載したのがその一覧。BASIC では、

CALL KANJI

WIDTH

命令で。DOS2 では

KMODE

MODE

命令で、それぞれ画面モードを指定するわけだ。たとえば、

CALL KANJI : WIDTH 32

という命令を実行すると、画面は 32 文字 × 12 行表示になる。同じことを DOS2 でやるには、

KMODE 0

MODE 32

とすればいい。ただし、英語版のシステムを立ち上げた場合は、漢字ドライバーが読み込まれていないから、一度 BASIC にもどって漢字モードを呼び出そう。

それでは、表 3.2 の意味を詳しく説明する。まず“画面ドット数”とは、画面に表示される点(ドット)の数。漢字はこの点の組み合わせで表わされる。

縦が 424 ドットの画面モードでは、“インターレース”という表示方法が使われている。これは縦方向に半ドットずらした 2 枚の画面を交互に表示し、結果として画面のドット数を増やす方法。ただ、画面がちらつくため、目が疲れやすいのが欠点だ。

“漢字ドット数”とは、1 個の漢字を表わすための点の数。普通は漢字を 16 × 16 ドットで表わすけれど、横 12 × 縦 16 ドットに圧縮して、横 512 ドットの画面に 40 文字の漢字を表示することも可能だ。また、パナソニックのモデムカートリッジを接続すれば、内蔵されている 12 × 12 ドットの漢字 ROM が、自動的に選択される。

表 3.2: 漢字 BASIC の画面モード

画面ドット数	漢字ドット数	半角文字数	設定方法
256 × 212	16 × 16	32 × 12	CALL KANJI0 : WIDTH 32
256 × 212	12 × 16	40 × 12	CALL KANJI1 : WIDTH 40
256 × 424	16 × 16	32 × 24	CALL KANJI2 : WIDTH 32
256 × 424	12 × 16	40 × 24	CALL KANJI3 : WIDTH 40
512 × 212	16 × 16	64 × 12	CALL KANJI0 : WIDTH 64
512 × 212	12 × 16	80 × 12	CALL KANJI1 : WIDTH 80
512 × 424	16 × 16	64 × 24	CALL KANJI2 : WIDTH 64
512 × 424	12 × 16	80 × 24	CALL KANJI3 : WIDTH 80

“半角文字数”は、画面に表示される半角文字の桁数と行数。当然のことながら、全角文字(漢字)の場合は桁数が表の値の半分になるわけだ。

ここで、ひとつ注意しておいて欲しいのは、表にある行数のすべてが BASIC などで使えるわけではないこと。ファンクションキーの表示や、漢字変換のためのエリアで、画面下の 1~2 行は使われてしまう。

3.1.6 漢字テキストと漢字グラフィック

さて、画面モードには、じつはもっと複雑な問題がある。BASIC で、

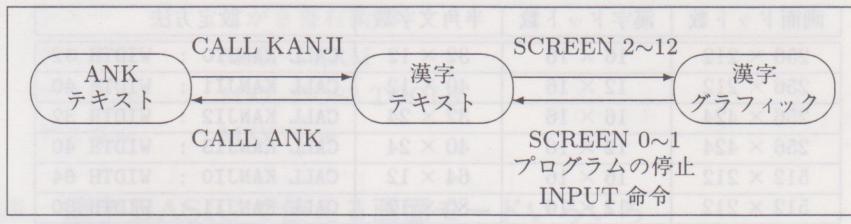
```
CALL KANJI1
WIDTH 40
SCREEN 0
```

という命令を実行すると、表 3.2 の上から 2 番目の画面モードになるのはわかるかな。ここで注目して欲しいのが、“SCREEN 0”を指定しているにもかかわらず、VDP は 256 × 212 ドットの “SCREEN 5” の状態になっていること。このような状態を、“漢字テキストモード”といふ。

このモードでは、BASIC プログラムの入力や修正、INPUT 命令による入力、PRINT 文による出力などは問題なく行なえる。しかし、LINE や PAINT などのグラフィック機能は使えない。

漢字モードでグラフィックを操作するには、“SCREEN 5”などの命令で、画面を“漢字グラフィックモード”に切り替える必要がある。ただ、覚えておいて欲しいのは、このモードではグラフィック機能と漢字の出力は使えるけど、原則として漢字の入力はできないこと。間違えないように。以上の画面モードの切り替えを図 3.2 にまとめておいた。

図 3.2: 画面モードの切り替え



プログラムが動きはじめるとき、終了するとき、エラーが起きたときなどは、予定どおりに画面モードが設定されているか確認しよう。うっかり“CALL KANJI”を忘れていると、プログラムを入力した直後には正しく動くけど、一度電源を切ってからプログラムをロードし直すと動かない、なんて事態が発生する。

また、漢字グラフィックモードで、“INPUT”命令と“LINE INPUT”命令を使うと、自動的に漢字テキストモードに戻ってしまう。これを避けるには、“INPUT\$”や“INKEY\$”関数でキーボードを読めばいい。

3.1.7 漢字ドライバーの正しい使い方なのだ

漢字ドライバーは、MSXの漢字機能は実用にならないという過去の常識をくつがえした、“天晴(あっぱれ)”なソフトウェア。だけど、BASICに“あとづけ”されたために、意外な落とし穴がある。いま説明したばかりの、漢字テキストと漢字グラフィックの違いも、そんな秘孔のひとつだ。ここでは、そんな漢字ドライバーを使う上での注意点を列挙する。

MSXをリセットしてから最初に“CALL KANJI”命令が実行されるときに、漢字ドライバーとMSX-JEを利用するためのワークエリアが用意される。そのときにBASICの変数の内容が消滅し、“FOR～NEXT”命令の繰り返し回数や、“RETURN”命令で戻る行番号を記録するための“ソフトウェアスタック”というワークエリアが初期化されてしまう。たとえば、

```
10 A=1
```

```
20 CALL KANJI
```

```
30 PRINT A
```

というプログラムを実行した場合、1回目には“0”が表示される(つまり変数Aの値が0になっている)けど、2回目以降は正しく“1”が表示されるという具合。また、

```

10 GOSUB 80
:
70 END
80 CALL KANJI0
90 RETURN

```

のようなプログラムを実行すると、“CALL KANJI0”が実行されたとき“RETURN”命令で戻る場所が忘れられ、プログラムの動作がおかしくなってしまうんだ。

このほかにも、漢字ドライバーがメモリー上にワークエリアを確保すると、BASIC のワークエリアがそのぶん減ってしまうという問題もある。メモリーの大きさいっぽいのプログラムや、マシン語サブルーチンを使うプログラムでは、メモリーが不足して動かなくなることもあるかもしれない。

漢字ドライバーは、プログラム内部のシフト JIS コードを、JIS コードに変換して漢字プリンターに印字する機能を持っている。この機能は、BASIC の“LPRINT”命令と、BIOS の“LPTOUT”に対して働くものだ。しかし、プリンターを 1 ドットずつ制御して、グラフィックを印字する“ビットイメージ印字”では副作用が出る。そこで、ビットイメージ印字の前には、システムワークエリアの中の F418H 番地 (RAWPRT と呼ぶ) に 0 でない値を書き込んで、漢字コードの変換を禁止する必要がある。

さて、ここからの話は、上級プログラマー向けのもの。まず、表 3.3 に掲載したの

表 3.3: 漢字ドライバーが使うフック

番地	名前	機能
FD9FH	H.TIMI	タイマー割り込み
FDA4H	H.CHPU	画面に 1 文字を表示する
FDA9H	H.DSPC	カーソルを表示する
FDAEH	H.ERAC	カーソルを消去する
FDB3H	H.DSPF	ファンクションキーを表示する
FDB8H	H.ERAF	ファンクションキーの表示を消去する
FDBDH	H.TOTE	画面をテキストモードに切り替える
FDC2H	H.CHGE	キーボードから 1 文字を読む
FDBBH	H.PINL	BASIC のエディターが 1 行を読む
FDE5H	H.INLI	1 行を読む
FF84H	H.WIDT	BASIC の WIDTH 命令の処理
FFB6H	H.LPTO	プリンターに 1 文字を書く
FFCOH	H.SCRE	BASIC の SCREEN 命令の処理
FFCAH	FCALL	拡張 BIOS

これらのフックをアプリケーションプログラムが使うと、漢字ドライバーが正しく動かない可能性がある。

4.1

第 4 章

V9958VDP

R#01
R#02
R#1
R#2
R#3
R#4
R#5
R#6
R#7
R#8
R#9
R#10
R#11
R#12
R#13
R#14
R#15
R#16
R#17
R#18
R#19
R#20
R#21
R#22
R#23
R#24
R#25
R#26
R#27

T.B.A.
この表で
* (断片)
はいけな
V9958
V9959
V9958 M



本章の第1節から第4節は、“V9938 MSX-VIDEO テクニカルデータブック”と“V9958 仕様書”を編集部が再編集したものである。V9938とV9958の共通の機能については省略するので、“MSX-Datapack”などを参照してほしい。

本章の第5節から第7節は、“MSX マガジン”1988年12月号、1989年1月号、11月号、12月号、1990年1月号の“MSX2+テクニカル探検隊”的記事を再編集したものである。

“V9958 仕様書”などのハードウェア資料では、“VDP のモード”が説明に使われているが、本書では、MSX マガジンの記事に合わせて、BASIC の画面モードを使う。VDP のモードと BASIC の画面モードは、表 4.1 のように対応する。

表 4.1: VDP のモードと BASIC の画面モード

VDP のモード	BASIC の画面モード
TEXT 1	SCREEN 0 : WIDTH 40
TEXT 2	SCREEN 0 : WIDTH 80
MULTI COLOR	SCREEN 3
GRAPHIC 1	SCREEN 1
GRAPHIC 2	SCREEN 2
GRAPHIC 3	SCREEN 4
GRAPHIC 4	SCREEN 5
GRAPHIC 5	SCREEN 6
GRAPHIC 6	SCREEN 7
GRAPHIC 7	SCREEN 8 (SCREEN 10~12)

4.1 V9958 レジスター一覧

表 4.2: モードレジスター

	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀	
R#0	0	DG	IE ₂ †	IE ₁	M ₅	M ₄	M ₃	0	Mode 0
R#1	0	BL	IE ₀	M ₁	M ₂	0	SI	MAG	Mode 1
R#2	0	A ₁₆	A ₁₅	A ₁₄	A ₁₃	A ₁₂	A ₁₁	A ₁₀	Pattern name T.B.A.
R#3	A ₁₃	A ₁₂	A ₁₁	A ₁₀	A ₉	A ₈	A ₇	A ₆	Color T.B.A. (Low)
R#4	0	0	A ₁₆	A ₁₅	A ₁₄	A ₁₃	A ₁₂	A ₁₁	Pattern gen. T.B.A.
R#5	A ₁₄	A ₁₃	A ₁₂	A ₁₁	A ₁₀	A ₉	A ₈	A ₇	Sprite attr. T.B.A. (Low)
R#6	0	0	A ₁₆	A ₁₅	A ₁₄	A ₁₃	A ₁₂	A ₁₁	Sprite pat. gen. T.B.A.
R#7	TC ₃	TC ₂	TC ₁	TC ₀	BD ₃	BD ₂	BD ₁	BD ₀	Text / Back drop color
R#8	MS†	LP†	TP	CB*	VR*	0	SPD	BW*	Mode 2
R#9	LN	0	S ₁ *	S ₀ *	IL	EO	NT*	DC*	Mode 3
R#10	0	0	0	0	0	A ₁₆	A ₁₅	A ₁₄	Color T.B.A. (High)
R#11	0	0	0	0	0	0	A ₁₆	A ₁₅	Sprite attr. T.B.A. (High)
R#12	T2 ₃	T2 ₂	T2 ₁	T2 ₀	BC ₃	BC ₂	BC ₁	BC ₀	Text / Back color
R#13	ON ₃	ON ₂	ON ₁	ON ₀	OF ₃	OF ₂	OF ₁	OF ₀	Blinking period
R#14	0	0	0	0	0	A ₁₆	A ₁₅	A ₁₄	VRAM access base addr.
R#15	0	0	0	0	S ₃	S ₂	S ₁	S ₀	Status reg. pointer
R#16	0	0	0	0	C ₃	C ₂	C ₁	C ₀	Color palette addr.
R#17	AII	0	RS ₅	RS ₄	RS ₃	RS ₂	RS ₁	RS ₀	Control reg. pointer
R#18	V ₃	V ₂	V ₁	V ₀	H ₃	H ₂	H ₁	H ₀	Display adjust
R#19	IL ₇	IL ₆	IL ₅	IL ₄	IL ₃	IL ₂	IL ₁	IL ₀	Interrupt line
R#20*	0	0	0	0	0	0	0	0	Color burst 1
R#21*	0	0	1	1	1	0	1	1	Color burst 2
R#22*	0	0	0	0	0	1	0	1	Color burst 3
R#23	DO ₇	DO ₆	DO ₅	DO ₄	DO ₃	DO ₂	DO ₁	DO ₀	Display offset
R#25‡	0	CMD	VDS*	YAE	YJK	WTE	MSK	SP2	Horizontal scroll (High)
R#26‡	0	0	HO ₈	HO ₇	HO ₆	HO ₅	HO ₄	HO ₃	Horizontal scroll (Low)
R#27‡	0	0	0	0	0	HO ₂	HO ₁	HO ₀	

T.B.A. : table base address

この表で“0”と書かれているビットには、かならず0を書き込む必要がある。

*(編集部注) ハードウェア制御用なので、普通のアプリケーションプログラムが書き替えてはいけない。

†V9938 にはあるがV9958 はないフラグなので、かならず0を書き込む必要がある。

‡V9958 に新しく追加されたレジスターである。これらの初期値は0で、V9958 の機能がV9938 同等になる。なお、レジスター 24 は欠番である。

表 4.3: コマンドレジスター

	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀	
R#32	SX ₇	SX ₆	SX ₅	SX ₄	SX ₃	SX ₂	SX ₁	SX ₀	Source X (Low)
R#33	0	0	0	0	0	0	0	SX ₈	Source X (High)
R#34	SY ₇	SY ₆	SY ₅	SY ₄	SY ₃	SY ₂	SY ₁	SY ₀	Source Y (Low)
R#35	0	0	0	0	0	0	SY ₉	SY ₈	Source Y (High)
R#36	DX ₇	DX ₆	DX ₅	DX ₄	DX ₃	DX ₂	DX ₁	DX ₀	Destination X (Low)
R#37	0	0	0	0	0	0	0	DX ₈	Destination X (High)
R#38	DY ₇	DY ₆	DY ₅	DY ₄	DY ₃	DY ₂	DY ₁	DY ₀	Destination Y (Low)
R#39	0	0	0	0	0	0	DY ₉	DY ₈	Destination Y (High)
R#40	NX ₇	NX ₆	NX ₅	NX ₄	NX ₃	NX ₂	NX ₁	NX ₀	Number of dot X (Low)
R#41	0	0	0	0	0	0	0	NX ₈	Number of dot X (High)
R#42	NY ₇	NY ₆	NY ₅	NY ₄	NY ₃	NY ₂	NY ₁	NY ₀	Number of dot Y (Low)
R#43	0	0	0	0	0	0	NY ₉	NY ₈	Number of dot Y (High)
R#44	CH ₃	CH ₂	CH ₁	CH ₀	CL ₃	CL ₂	CL ₁	CL ₀	Color
R#45	0	MXC	MXD	MXS	DIY	DIX	EQ	MAJ	Argument
R#46	CM ₃	CM ₂	CM ₁	CM ₀	LO ₃	LO ₂	LO ₁	LO ₀	Command

表 4.4: ステータスレジスター

	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀	
S#0	F	5SF	C	5S ₄	5S ₃	5S ₂	5S ₁	5S ₀	Status 0
S#1	FL†	LPS†	ID ₄	ID ₃	ID ₂	ID ₁	ID ₀	FH	Status 1
S#2	TR	VR	HR	BD	1	1	EO	CE	Status 2
S#3	X ₇	X ₆	X ₅	X ₄	X ₃	X ₂	X ₁	X ₀	Column (Low)
S#4	1	1	1	1	1	1	1	X ₈	Column (High)
S#5	Y ₇	Y ₆	Y ₅	Y ₄	Y ₃	Y ₂	Y ₁	Y ₀	Row (Low)
S#6	1	1	1	1	1	1	EO	Y ₈	Row (High)
S#7	C ₇	C ₆	C ₅	C ₄	C ₃	C ₂	C ₁	C ₀	Color
S#8	BX ₇	BX ₆	BX ₅	BX ₄	BX ₃	BX ₂	BX ₁	BX ₀	Border X (Low)
S#9	1	1	1	1	1	1	1	BX ₈	Border X (High)

†V9938 には存在するが V9958 には存在しない機能に関するビットなので、V9958 でこれらの値は無意味である。

V9958 の ID は 00010B である。

4.2 V9958 の新機能

4.2.1 水平スクロール

	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
R#25	0	CMD	VDS	YAE	YJK	WTE	MSK	SP2
R#26	0	0	HO ₈	HO ₇	HO ₆	HO ₅	HO ₄	HO ₃
R#27	0	0	0	0	0	HO ₂	HO ₁	HO ₀

HO₈ ~ HO₀ は、画面の水平スクロール量を、SCREEN 6 と 7 では 2 ドット単位で、その他の画面モードでは 1 ドット単位で、設定する。

SP2 = 0 (初期値) ならば、水平方向画面サイズが 1 ページとなる。

SP2 = 1 ならば、水平方向画面サイズが 2 ページとなる。

MSK = 0 (初期値) ならば、画面の左端がマスクされない。

MSK = 1 ならば、SCREEN 6 と 7 では画面の左端 16 ドットが、その他の画面モードでは画面の左端 8 ドットが、マスクされ、ボーダーカラーが表示される。

HO₈ ~ HO₀ に対して、表示画面は設定値だけ左向きに、8 ドット単位 (SCREEN 6 と 7 では 16 ドット単位) でシフトする。

図 4.1: 水平スクロール (SP2=0 の場合)

表示画面								
HO 7-3	8dot							
0	0	1			30	31	1 line	
1	1	2			31	0		
•	•	•			•	•		
31	31	0			29	30		

図 4.2: 水平スクロール (SP2=1 の場合)

表示画面								
HO 8-3 8dot								
0	0	1		31	32		62	63
1	1	2		32	33		63	0
31	31	32		62	63		29	30
32	32	33		63	0		30	31
63	63	0		30	31		61	62

SP2 = 0 のとき、1画面分のデータが水平スクロールで表示される。HO₈ は無視される。

SP2 = 1 のとき、2画面分のデータが水平スクロールで表示される。パターンネームテーブルのベースアドレスの A₁₅には 1 を設定する。パターンネームテーブルのベースアドレスは、0~31 は、設定値の A₁₅=0 とした値、32~63 は、設定値の A₁₅=1 とした値になる。パターンジェネレーターテーブルとカラーテーブルのベースアドレスは、設定値そのままで、スクロールによって変化しない。

HO₂ ~ HO₀ に対して、表示画面は設定値だけ右向きに、1 ドット単位 (SCREEN 6 と 7 では 2 ドット単位) でシフトする。

4.2.2 ウェイト

	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀
R#25	0	CMD	VDS	YAE	YJK	WTE	MSK	SP2

WTE = 0 (初期値) ならば、ウェイト機能が無効になる。

WTE = 1 ならば、ウェイト機能が有効になる。CPU が VRAM をアクセスした際に、それによる V9958 の VRAM アクセスが完了するまで、すべての V9958 ポートへのアクセスに対してウェイトがかかる。レジスターとカラーパレットへのアクセス未完および、コマンドのデータレディーによるウェイト機能はない。

4.2.3 コマンド

	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀
R#25	0	CMD	VDS	YAE	YJK	WTE	MSK	SP2

CMD = 0 (初期値) ならば、SCREEN 5~12 の画面モードでのみコマンド機能が有効になる。

CMD = 1 ならば、全画面モードにおいてコマンド機能が有効になる。

SCREEN 5 ~ 12 以外の画面モードでは、SCREEN 8 として動作する。従ってパラメーターは、SCREEN 8 の X-Y 座標系で設定する。

4.2.4 YJK 方式の表示

レジスターの設定

	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀
R#25	0	CMD	VDS	YAE	YJK	WTE	MSK	SP2

YJK = 0 (初期値) ならば、VRAM 上のデータを RGB 方式 (各 3、3、2 ビット) として扱う。スプライトの表示色は従来どおり。

YJK = 1 ならば、VRAM 上のデータを YJK 方式とみなし、これを RGB 信号 (各 5 ビット) に変換し、RGB 端子よりアナログ出力する。スプライトの表示色にはパレットが有効になる。

YAE は、YJK 方式のデータフォーマットを選択する。

YAE = 0 の場合

アトリビュートがない。データフォーマットを次に示す。連続した 4 ドットをグループピングして表わす。

C ₇	C ₆	C ₅	C ₄	C ₃	C ₂	C ₁	C ₀	アトリビュート	YJK
				Y				K _L	
				Y				K _H	
				Y	ALY	AYA	BYA	J _L	
				Y				J _H	

YAE = 1 の場合

1 ドットごとにアトリビュートがある。データフォーマットを次に示す。連続した 4 ドットをグルーピングして表わす。

C ₇	C ₆	C ₅	C ₄	C ₃	C ₂	C ₁	C ₀	アトリビュート	YJK
				Y	A			K _L	
				Y	A			K _H	
				Y	A			J _L	
				Y	A			J _H	

A = 0 (初期値) ならば、Y、J、K は、すべて YJK 方式のデータとなる。

A = 1 ならば、Y データはカラーコードとなりカラーパレットをとおして RGB 出力される。J と K は、YJK 方式のデータとなる。

YJK 方式と RGB 方式の変換式 (参考)

$$\begin{aligned}
 R &= Y + J \\
 G &= Y + K \\
 B &= \frac{5}{4}Y - \frac{1}{2}J - \frac{1}{4}K \\
 Y &= \frac{1}{2}B + \frac{1}{4}R + \frac{1}{8}G \\
 J &= R - Y \\
 K &= G - Y
 \end{aligned}$$

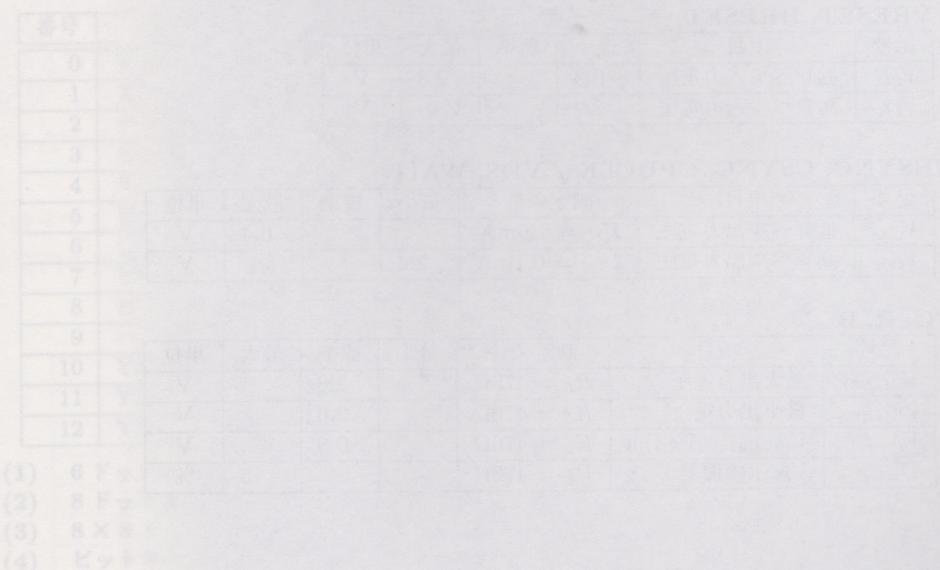
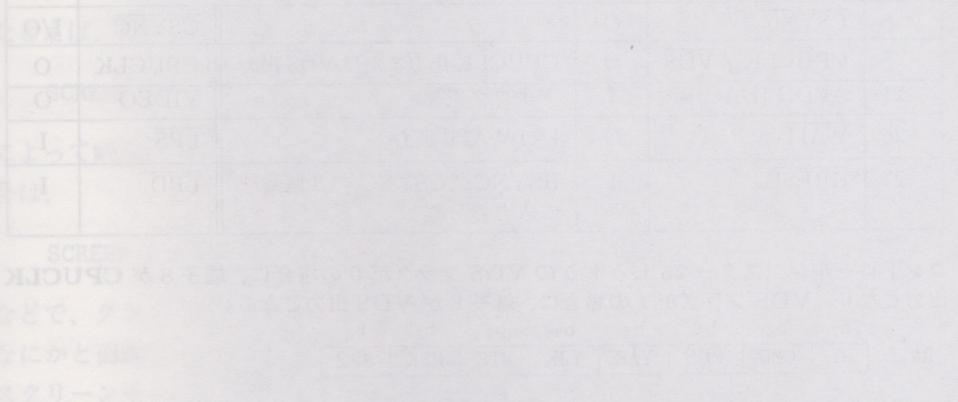
(編集部注) Y の値は、アトリビュートがない場合には 0 ~ 31 の整数、アトリビュートがある場合には 0 ~ 30 の偶数である。J と K の値は、-32 ~ 31 の整数である。YJK から RGB への変換結果は、0 ~ 31 にクリッピングされる。

4.3 V9958 の廃止機能

V9938 に存在した次の機能は廃止された。

- コンポジットビデオ出力
- マウス/ライトペンインターフェース

(編集部注) MSX のマウスは、V9938 のマウスインターフェース機能を使っていないので、この機能の削除は影響ない。



4.4 V9958 ハードウェア仕様 (変更部分)

表 4.5: V9958 の端子の変更

番号	V9958			V9938	
	名称	I/O	説明	名称	I/O
4	VRESET	I	HSYNC / CSYNC の 3 値論理の入力の分離	VDS	O
5	HSYNC	O	HSYNC 出力またはバーストフラグ出力	HSYNC	I/O
6	CSYNC	O		CSYNC	I/O
8	CPUCLK / VDS	O	CPUCLK 出力または VDS 出力	CPUCLK	O
21	AVDD (DAC)	I	アナログ電源	VIDEO	O
26	WAIT	O	I/O WAIT 出力	LPS	I
27	HRESET	I	HSYNC / CSYNC の 3 値論理の入力の分離	LPD	I

コントロールレジスター 25 ビット 5 の VDS フラグが 0 の場合に、端子 8 が CPUCLK 出力となり、VDS フラグが 1 の場合に、端子 8 が VDS 出力となる。

b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀	
R#25	0	CMD	VDS	YAE	YJK	WTE	MSK	SP2

表 4.6: V9958 の直流特性

VRESET, HRESET

記号	項目	最小	標準	最大	単位
V_{IL}	低レベル入力電圧	-0.3		0.8	V
V_{IH}	高レベル入力電圧	2.2		V_{CC}	V

HSYNC, CSYNC, CPUCLK / VDS, WAIT

記号	項目	測定条件	最小	標準	最大	単位
V_{OL}	低レベル出力電圧	$I_{OL} = 1.6\text{mA}$			0.4	V
V_{OH}	高レベル出力電圧	$I_{OH} = 0.1\text{mA}$	2.4			V

G, R, B

記号	項目	測定条件	最小	標準	最大	単位
V_{RGB31}	最大出力電圧	$R_L = 470\Omega$		2.8		V
V_{RGB0}	最小出力電圧	$R_L = 470\Omega$		2.0		V
V_{P-P}	$V_{RGB31} - V_{RGB0}$	$R_L = 470\Omega$		0.8		V
D_{RGB}	V_{P-P} の偏差	$R_L = 470\Omega$			5	%

4.5 V9958 と MSX2+

MSX の画面表示を制御する部品を “Video Display Processor”、略して VDP という。MSX2 の VDP は “V9938” というものだったけど、MSX2+以降のマシンでは “V9958” に機能アップされた。ここでは、その V9958 に追加された機能を紹介する。

4.5.1 スクリーンモードは全部で 12 種類

スクリーンモードとは、BASIC の SCREEN 命令で切り替えられる画面の状態。たとえば、横 40 枠表示で BASIC のプログラムを書きたい場合は、

```
SCREEN 0 : WIDTH 40
```

によって画面をテキスト（文字表示）モードに切り替え、グラフィックを描きたい場合は、

```
SCREEN 8
```

などで、グラフィック（図形表示）モードに切り替える。スクリーンモードが多いとにかく面倒なので、少なくてすむならそれにこしたことはない。でも MSX2+ のスクリーンモードが多いことには、それなりの理由がある。

表 4.7: MSX2+の画面モード

番号	表示方法	解像度	色数
0	文字 (1)	80 × 24 文字	文字の色と背景の色を指定する
1	文字 (2)	32 × 24 文字	文字の色と背景の色を指定する
2	テーブル (3)	256 × 192 ドット	16 色 横 8 ドットごとに 2 色
3	ビットマップ (4)	64 × 48 ドット	16 色
4	テーブル	256 × 192 ドット	16 色 横 8 ドットごとに 2 色
5	ビットマップ	256 × 212 ドット	16 色
6	ビットマップ	512 × 212 ドット	4 色
7	ビットマップ	512 × 212 ドット	16 色
8	ビットマップ	256 × 212 ドット	256 色
9	ハングル文字表示用で、日本のMSXにはない。		
10	YJK / RGB	256 × 212 ドット	12,499 色 (本文参照)
11	YJK / RGB	256 × 212 ドット	12,499 色 (本文参照)
12	YJK	256 × 212 ドット	19,268 色 (本文参照)

- (1) 6 ドット × 8 ドットで 1 文字を表わし、英字とカタカナを表示する。
- (2) 8 ドット × 8 ドットで 1 文字を表わし、英字とカタカナとひらがなを表示する。
- (3) 8 × 8 ドットのパターンの組み合わせで画面を作る。
- (4) ビットマップ表示では、ドットの色を隣の色に関係なく表示可能である。

第1番目が速さの問題。グラフィックモードの画面に文字を出力すると、表示に時間がかかるてしまう。だからプログラムを入力するようなときは、図形や漢字を使えない代わりに表示が速い、テキストモードが便利だ。

第2の理由は、機能が高い(表示できるドット数や色が多い)画面ほど、たくさんのデータを必要とすること。たとえばSCREEN 8の画面データは1枚で54,272バイトもあり、これでは1枚のディスク(2DD)に12枚の絵しか記録できない。そのためROMカートリッジのゲームでは、色数に制限があるかわりにデータが少なくて動作が速い、SCREEN 2を使ってメモリーを節約することが多くなっている。

第3に、互換性を保ちながら機能を追加したために、多くのスクリーンモードが必要になったこと。最初のMSXでは、SCREEN 0から3までの4種類のスクリーンモードしかなかった。それがMSX2になり、高解像度グラフィック表示のためSCREEN 4から8が追加。さらにMSX2+では、色数を増やすためにSCREEN 10から12が追加されたというわけ。

これらのスクリーンモードをまとめたのが表4.7。でも実際にはスクリーンモードの問題はこれで終わりではなく、縦方向の解像度を2倍に増やすインターレースモードや、MSX2+で新しく追加された漢字モードなどもある。

4.5.2 VDPのレジスターをコントロールする

MSXの画面表示を制御するVDPの中には、マシンの心臓部であるCPUと同様に“レジスター”というものがある。そしてこのVDPのレジスターは、I/OポートとおすることでCPUが操作できるものだ。中でも、CPUがVDPを制御するために使うレジスターを“コントロールレジスター”、CPUがVDPの状態を知るために使うレジスターを“ステータスレジスター”と呼んでいる。このほかにも、VDPに高度な命令を実行させるためにCPUが操作する“コマンドレジスター”があるけど、本書のプログラムでは使用しないので、説明は省略する。

CPUとVDPを接続するI/Oポートの番地は、普通は98Hから9BHまで使っている。でも正確には表4.8に掲載したように、ROMの6番地と7番地の内容で

表4.8: VDPのI/Oポート

ポート名	R/W	I/O番地	用途
ポート0	READ	ROMの6番地の内容	VRAM読み出し
ポート1	READ	ROMの6番地の内容+1	ステータスレジスター
ポート0	WRITE	ROMの7番地の内容	VRAM書き込み
ポート1	WRITE	ROMの7番地の内容+1	コントロールレジスター
ポート2	WRITE	ROMの7番地の内容+2	パレットレジスター
ポート3	WRITE	ROMの7番地の内容+3	間接指定レジスター

決まつてくる。これは、MSX 本体の外に VDP を増設できるようにとの配慮から。なお、同じポート 0 でも、書き込む場合と読み出す場合とでは、I/O ポートの番地が異なることがあるので注意しよう。

さて、VDP のコントロールレジスターの値を設定するためには、まず、表 4.8 のポート 1 に設定したいデータを書き込み、続けて同じポート 1 に“レジスター番号 +128”を書き込む。この“続けて”という条件は意外と重要で、2 回の書き込みの間に割り込みが起こると VDP が混乱してしまうんだ。だからこの場合は、まず“DI”命令で割り込みを禁止しておいてから、ふたつのデータを続けて書き込む方法が一般的に使われている。

ところで、コントロールレジスターというのは書き込み専用のもの。一度設定された値は、読み出すことができない。だから、レジスターの特定のビットだけを書き替えたい、なんて場合は都合が悪くなってしまう。そこで通常は、コントロールレジスターに書き込む値を、表 4.9 のようなシステムワークエリア (RAM) にも書き込んでおくという方法をとる。たとえば、コントロールレジスター 1 のビット 4 を 1 に変えたい場合には、RAM の F3E0H 番地の内容を読んでビット 4 を 1 に変え、その値をコントロールレジスター 1 と RAM の F3E0H 番地に書き込むという具合だ。あとで紹介するリスト 4.9 の走査線割り込みのサンプルプログラムの中では、“WRTVDP”というサブルーチンがこれと同じ動作を行なっている。

次に、ステータスレジスターの値を読むための方法を説明する。これは、コントロールレジスター 15 に読みたいステータスレジスターの番号を設定し、VDP のポート 1 の値を読み、コントロールレジスター 15 を 0 に戻すという手続きを、割り込みを禁止したままで行なうというもの。リスト 4.9 のサンプルプログラムの中では、“_VDPSTA”というサブルーチンがこれにあたる。

表 4.9: コントロールレジスターの保存場所

レジスター番号	VDP 関数番号	保存番地	ラベル
0	0	F3DFH	RGOSAV
⋮	⋮	⋮	⋮
7	7	F3E6H	RG7SAV
8	9	FFE7H	RG8SAV
⋮	⋮	⋮	⋮
23	24	FFF6H	RG23SA
25	26	FFF8H	RG25SA
26	27	FFF9H	RG26SA
27	28	FFFCCH	RG27SA

表 4.10: そのほかの便利なシステムワークエリア

番地	ラベル	意味
F341H	RAMADO	ページ 0 の RAM のスロット番号*
F342H	RAMAD1	ページ 1 の RAM のスロット番号*
F343H	RAMAD2	ページ 2 の RAM のスロット番号*
F344H	RAMAD3	ページ 3 の RAM のスロット番号*
FAF5H	DPPAGE	ディスプレーページ番号
FAF6H	ACPAGE	アクティブページ番号
FD9AH	H.KEYI	割り込みフック
FD9FH	H.TIMI	タイマー割り込みフック

* ディスクがある場合のみ有効。

表 4.11: MSX2+に追加、変更されたシステムワークエリア

番地	ラベル	意味
OFAFCH	MODE	次表参照
OFAFDH	NORUSE	漢字ドライバーのワークエリア
OFDOAH	SLTWRK+1	漢字ドライバーのワークエリア
:	:	
OFDOFH	SLTWRK+6	
OFFFAH	RG25SA	VDP レジスターのセーブ
OFFFBH	RG26SA	
OFFFCB	RG27SA	

表 4.12: OFAFCH 番地 (MODE) の詳細

ビット	意味
b ₇	1 ならばカタカナ、0 ならばひらがな
b ₆	1 ならば第 2 水準漢字 ROM あり
b ₅	1 ならば SCREEN 11、0 ならば SCREEN 10
b ₄	内部で使用
b ₃	1 ならば SCREEN 0~3 で VRAM 番地に 3FFFH をマスク
b ₂	VRAM 容量
b ₁	00 : 16KB、01 : 64KB、10 : 128KB
b ₀	1 ならばローマ字カナ変換

もっとも、MSX2 や 2+ の ROM にはこれらのサブルーチンと同じ機能の BIOS があるので、普通は自分でサブルーチンを作らずに BIOS を使えばいい。でも、これらの BIOS はサブ ROM にあったり、処理中にサブ ROM を呼び出したりするので、

多少時間がかかるという難点がある。そのため、走査線割り込みのような処理には都合が悪いので、あえて BIOS を使わないわけだ。

表 4.10 ~ 4.12 に、そのほかのシステムワークエリアをまとめておいたので、参考にしてほしい。

4.5.3 V9958 のレジスター

図 4.3 に掲載したのは、V9958 に追加された 3 個のコントロールレジスター。これらのレジスターは書き込み専用のもので、書き込まれた値を表 4.9 のシステムワークエリアに記録する。コントロールレジスター 24がないことや、レジスター番号と BASIC の VDP 関数の番号が異なることに注意しよう。

V9958 で追加された機能の大部分は、コントロールレジスター 25 で制御される。有名な YJK(自然画) 表示と横スクロールは後回しにして、残りの機能から紹介するぞ。

レジスター 25 のビット 7 には、かならず 0 を書き込もう。ビット 5 は “VDS” と呼ばれ、VDP の端子 8 の機能を制御する。けれども、普通のプログラムがこのビットを書き替えることは禁止されている。

またビット 6 に 0 を書き込めば、V9938 と同様に SCREEN 5 から 8 の画面に対してのみ “VDP コマンド” が使えるようになる。これとは逆に 1 を書き込めば、全画面モードで VDP コマンドが使えるわけだ。この VDP コマンドとは、BASIC の COPY 命令や LINE 命令のような仕事を、VDP にさせる機能。細かい話になるけど、SCREEN 5 から 8 以外の画面に対する VDP コマンドでは、128 キロバイトのビデオ RAM の中の場所を、SCREEN 8 のように X 座標が 0 から 255、Y 座標が 0 から 511 の値で指定する。

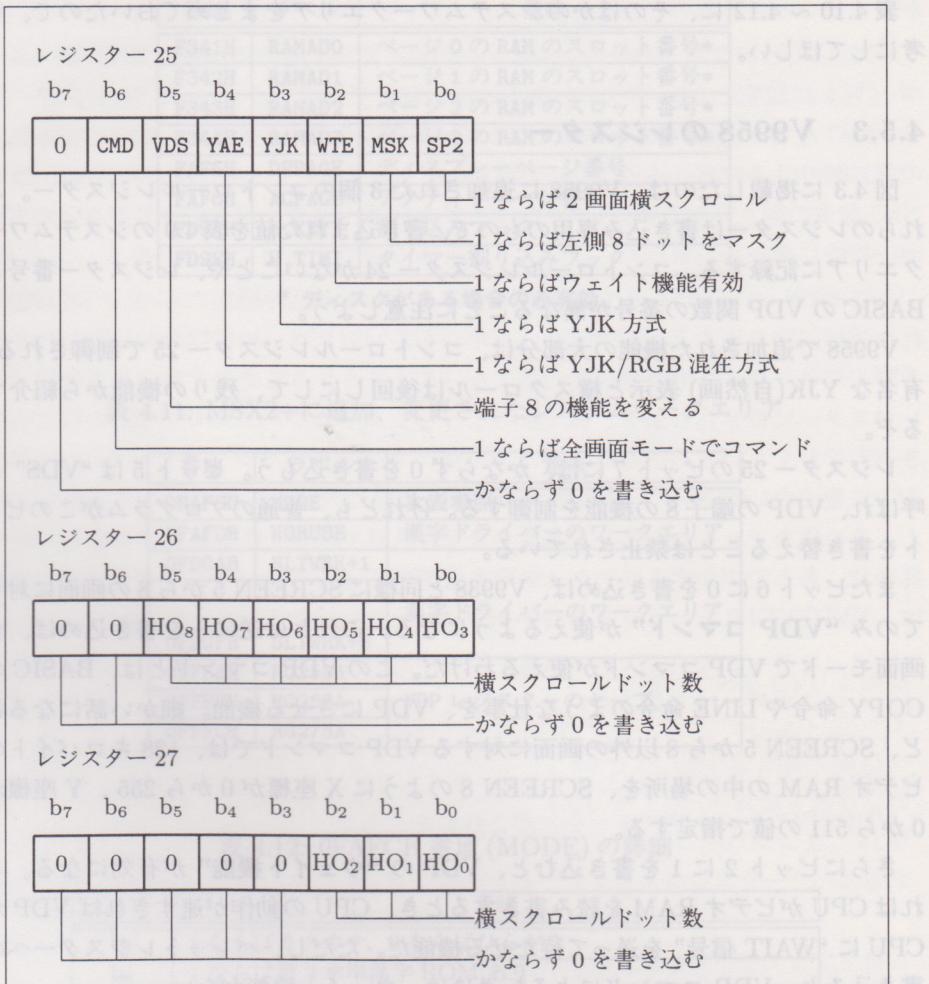
さらにビット 2 に 1 を書き込むと、VDP の “ウェイト機能” が有効になる。これは CPU がビデオ RAM を読み書きするとき、CPU の動作が速すぎれば VDP が CPU に “WAIT 信号” を送って待たせる機能だ。ただし、パレットレジスターへの書き込みと、VDP コマンドによる転送には、ウェイト機能がない。

4.5.4 VDP による横スクロール

横スクロールには、1 画面分の画像データを使う方法と、2 画面分の画像データを使う方法がある。それぞれの場合について、順番に説明していこう。

まず、コントロールレジスター 25 のビット 0 “SP2” が 0 の場合には、図 4.4 の上のように 1 画面分のデータによる横スクロールが起こる。スクロールのドット数は、レジスター 26 と 27 で指定することができる。レジスター 26 に 0 から 63 の値を書き込むと、その値 × 8 ドット単位で画面が左にスクロールし、レジスター 27 に

図 4.3: V9958 に追加されたコントロールレジスターの機能一覧

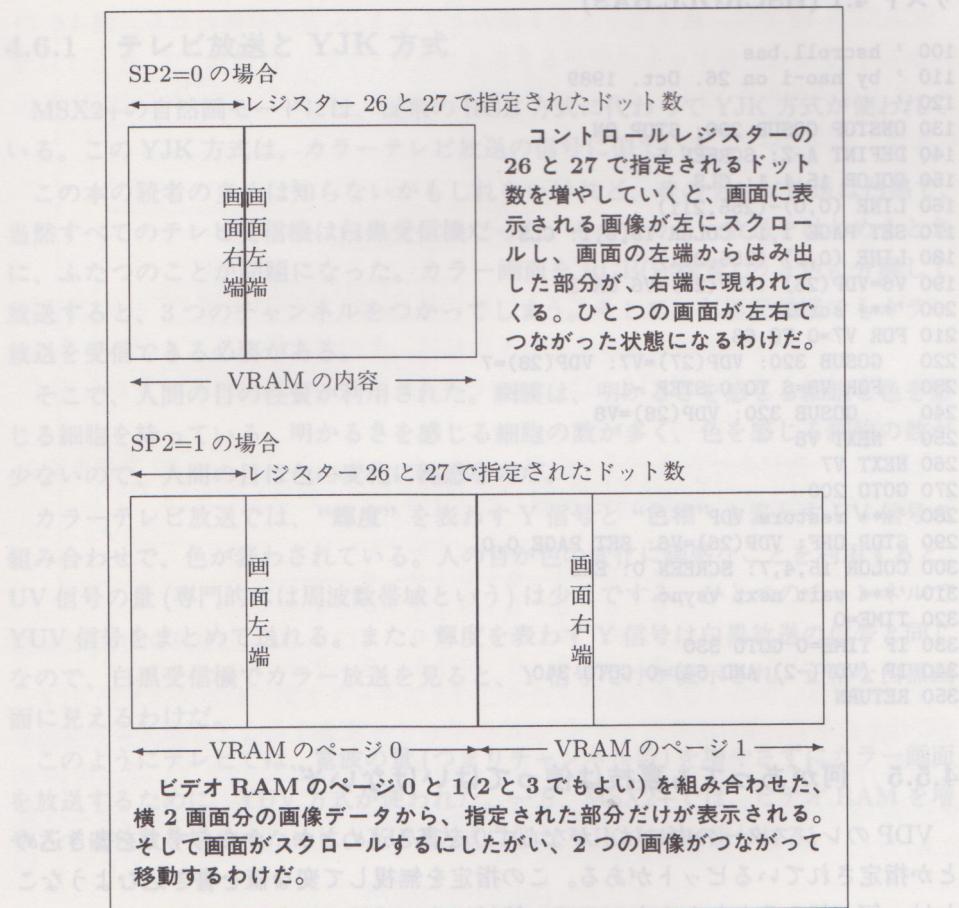


0 から 7 の値を書くとその数だけ右に移動する。ただし SCREEN 6 と 7 では、指定した数の 2 倍のドット数のスクロールが起こるから注意しよう。ドット数を 0 から 255 まで順番に増やしていくと、画面が左へスクロールし、はみだした部分が右端から現われてくる。

一方レジスター 25 のビット 0 が 1 の場合には、図 4.4 の下のように、2 画面分の画像データから指定された部分が表示され、横スクロールがはじまる。

このときの画像データは、ビデオ RAM のページ 0 と 1 または 2 と 3 に記憶され

図 4.4: 2種類の横スクロールの仕組み



ている。ページが 0 と 1 の場合はディスプレーページを 1 に、2 と 3 の場合にはディスプレーページを 3 に設定しよう。横スクロールのドット数は 0 から 511(つまり 1 画面スクロールの 2 倍)になる。

また、レジスター 25 のビット 1 を 1 にすると、画面左端の 8 ドット (SCREEN6 と 7 では 16 ドット) が表示されず、代わりにその場所に画面の周辺色が表示される。これは横スクロール、とくに 1 画面分のデータによる横スクロールをする場合、画面からはみ出した部分がすぐに反対側から現われるのを隠すのに便利だ。リスト 4.1 に横スクロールのプログラム例を掲載しておくので、参考にしよう。

リスト 4.1 (HSCROLL.BAS)

```

100 ' hscroll.bas
110 ' by nao-i on 26. Oct. 1989
120 '
130 ONSTOP GOSUB 290: STOP ON 1
140 DEFINT A-Z: SCREEN 5
150 COLOR 15,1,1: CLS
160 LINE (0,0)-(255,211)
170 SET PAGE 1,1: COLOR 15,8,1: CLS
180 LINE (0,0)-(255,211)
190 V6=VDP(26) :VDP(26)=V6 OR 3
200 '*** scroll
210 FOR V7=0 TO 63
220   GOSUB 320: VDP(27)=V7: VDP(28)=7
230   FOR V8=6 TO 0 STEP -1
240     GOSUB 320: VDP(28)=V8
250   NEXT V8
260 NEXT V7
270 GOTO 200
280 '*** restore VDP
290 STOP OFF: VDP(26)=V6: SET PAGE 0,0
300 COLOR 15,4,7: SCREEN 0: END
310 '*** wait next vsync
320 TIME=0
330 IF TIME=0 GOTO 330
340 IF (VDP(-2) AND 64)=0 GOTO 340
350 RETURN

```

4.5.5 何があっても裏技は使ってはいけないぞ

VDP のレジスターの中には、かならず 0 を書き込めとか、かならず 1 を書き込めとか指定されているビットがある。この指定を無視して変な値を書き込むようなことは、何が起こるかわからないので、絶対にやってはいけない。

また、YJK=0 と YAE=1 の組み合わせのように、VDP の動作が仕様書で決められていない設定があるけど、こうした仕様書に書かれてない“裏技”も、使ってはいけない。たとえ自分が持っているマシンで問題なく動いたとしても、それはたまたま動いただけのこと。ほかの MSX マシンで正常に動作するという保障はどこにもない。また、現在の V9958 に対しては有効な裏技であっても、今後 VDP が改良されたり、ヤマハ以外のメーカーが V9958 互換の VDP を作ったりすれば（いまのところ V9958 はすべてヤマハ製）、同じ裏技が有効とは限らない。

これと同じように、CPU の裏技（正式には“未定義命令”という）も一切使ってはいけない。過去の例でも、Z80 の裏技を使ったために、ピクターの HC-95 のターボモード（Z80 の上位互換 CPU である HD64180 を使っている）で暴走したソフトウェアがあった。

4.6 YJK 方式を解剖する

4.6.1 テレビ放送と YJK 方式

MSX2+の自然画モードには、従来の RGB 方式に代わって YJK 方式が使われている。この YJK 方式は、カラーテレビ放送の信号に似ている。

この本の読者の多くは知らないかもしれないけれど、昔のテレビ放送は白黒で、当然すべてのテレビ受信機は白黒受信機だった。その後カラー放送をはじめると、ふたつのが問題になった。カラー画面を RGB(赤緑青) の 3 色に分解して放送すると、3 つのチャンネルをつかってしまう。そして、白黒受信機でもカラー放送を受信できる必要がある。

そこで、人間の目の性質が利用された。網膜は、明かるさを感じる細胞と色を感じる細胞を持っている。明かるさを感じる細胞の数が多く、色を感じる細胞の数が少ないので、人間の目は色の変化に鈍感なのだ。

カラーテレビ放送では、“輝度”を表わす Y 信号と “色相” を表わす UV 信号の組み合わせで、色が表わされている。人の目が色の変化に鈍感なことを利用すると、UV 信号の量(専門的には周波数帯域という)は少しですみ、ひとつのチャンネルで YUV 信号をまとめて送れる。また、輝度を表わす Y 信号は白黒放送の信号と同じなので、白黒受信機でカラー放送を見ると、Y 信号だけが表示され、正常な白黒画面に見えるわけだ。

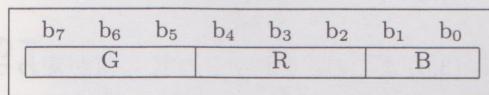
このようにテレビでは、電波の量(つまりチャンネル数)を増やすずにカラー画面を放送するために、YUV 方式が使われた。一方、MSX2+では、ビデオ RAM を増やすずに色数を増やすために、YUV 方式に似た YJK 方式が使われる。

4.6.2 RGB 方式と YJK 方式のデータ構造

RGB 方式 (SCREEN 8)

SCREEN 8 では、R、G、B の明かるさをそれぞれ 3、3、2 ピットで表わし、1 ドットごとに 256 色中の任意の色を表示できる。

図 4.5: RGB 方式画面のデータ構造



YJK 方式 (SCREEN 12)

SCREEN 12 では、横 4 ドットを 1 組みとする YJK 方式が使われる。図 4.6 の Y_0 から Y_3 までは、各ドットの明かるさを 5 ビット (つまり 32 段階) で、K と J は 4 ドット全体の色相を 12 ビット (4096 色) で表わす。YJK 方式のデータを RGB 方式に変換する方法は次の通りだ。

$$\begin{aligned} R &= Y + J \\ G &= Y + K \\ B &= 1.25Y - 0.5J - 0.25K \end{aligned}$$

ただし、Y の値は 0 から 31 まで、J と K の値は -32 から 31 までだ。上の式で計算した R、G、B の値が 0 よりも小さくなれば、その値の代わりに 0 が出力され、同様に 31 よりも大きくなれば、31 が出力される。このような処理を、“0 から 31 にクリッピングする”という。

たとえば、次にあげた 4 バイトのデータは、 $Y = 0$ 、 $J = K = -32$ で黒を表わすことがわかるかな。電卓片手に計算してみよう。

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0

これとは逆に、RGB のデータを YJK のデータへ変換することを考えてみよう。つまり、RGB への変換式を 3 元連立 1 次方程式とみなし、これを Y、J、K についてそれぞれ解けばいいわけだ。高校の“代数・幾何”レベルの問題だぞ。答えは次にあげた 3 つの式。ちゃんとできたかな。

$$\begin{aligned} Y &= (2R + G + 4B)/8 \\ J &= (6R - G - 4B)/8 \\ K &= (-2R + 7G - 4B)/8 \end{aligned}$$

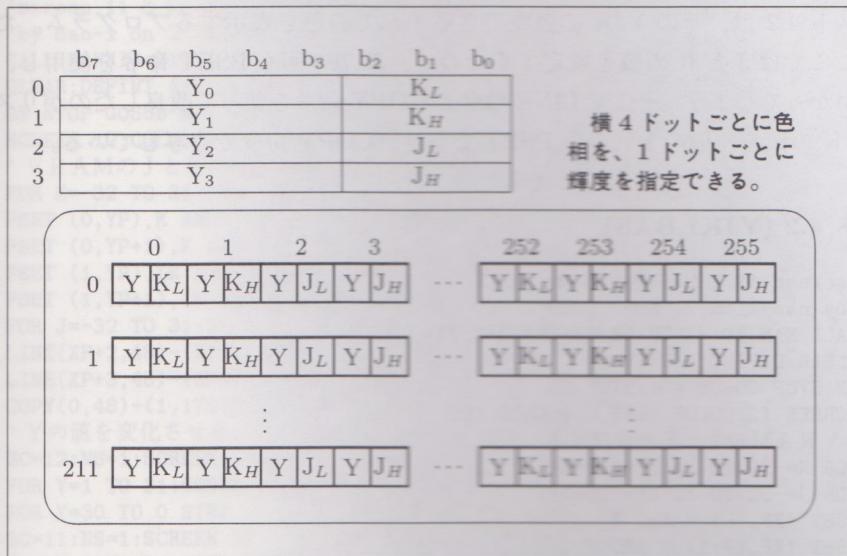
YJK 方式の画像を表示するときにも、VDP から出力される信号は前に説明した式で従来と同じアナログ RGB 信号と、コンポジット（ビデオ）信号に変換されているので、MSX2+に特別なモニターテレビは必要ない。

YJK 方式の画面では、SCREEN 8 に似て、基本的には 1 バイトが 1 ドットに対応する。しかし、J と K の値は横 4 ドットごとに指定されるので、たとえば、

PSET (0,0),0

を行なうと、(0,0) から (3,0) までの、4 ドットの K の値が変わってしまう。そのため SCREEN 12 で LINE 命令などを使うと、“色化け”が起きるわけだ。これに関しては、あとでもう一度説明する。

図 4.6: YJK 方式画面のデータ構造



混在方式 (SCREEN 10、11)

YJK 方式の欠点は、横 4 ドットごとにしか色を指定できないので、自然画に文字や線画を重ね書きしにくいことだ。そこで、RGB 方式の SCREEN 5 と YJK 方式の SCREEN 12 の長所を合わせ持つ SCREEN 10 と 11 が用意された。これらの画面モードでは、自然画に文字を重ねて表示することが容易だが、色数は SCREEN 12 よりも少なくなる。

図 4.7: 混在方式画面のデータ構造

	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀	1 ドットごとに、YJK 方式と RGB 方式を選択できる。VRAM のビット 3 が 0 ならば、YJK 方式で表示される。ビット 3 が 1 ならば、ビット 4 からビット 4 で指定されるパレットの色が表示される。
0	Y ₀				A ₀	K _L			
1	Y ₁				A ₁	K _H			
2	Y ₂				A ₂	J _L			
3	Y ₃				A ₃	J _H			

4.6.3 色見本のプログラム

MSX2+の最大のウリは、19,268 色表示の SCREEN 12。ビデオ RAM を増やすずに (MSX2 と同じ 128 キロバイト) 色数を増やすため、YJK という表示方式が取

られている。これまで説明してきたように、輝度(明かるさ)を表わすYの値と、色相を表わすJとKの組み合わせで、色を指定する方式だ。

リスト4.2は、そのYJKで表現できるすべての色を表示するプログラム。ただし、ここではJとKの値を設定するために、32,768回もPSET命令を使用し、時間がかかるてしまう。そこでLINE命令とCOPY命令を使って改良したのがリスト4.3。Kの値を左側の1列のみPSETで書き、COPY命令で複写している。

リスト4.2 (YJK1.BAS)

```

100 'screen 11と12の色見本 by PSET
110 'by nao-i on 2. Nov. 1988
120 CALL KANJI0:WIDTH 64:DEFINT A-Z:YY=0
130 CLEAR:DEFINT A-Z:YY=0
140 ON STOP GOSUB 400:STOP ON
150 SCREEN 12:COLOR &HF8,0,0:CALL CLS
160 'VRAMのJとKを設定する
170 FOR K=-32 TO 31:YP=112+K+K
180 FOR J=-32 TO 31:XP=128+J*4
190 PSET (XP,YP),K AND 7
200 PSET (XP,YP+1),K AND 7
210 PSET (XP+1,YP),(K AND 56)¥8
220 PSET (XP+1,YP+1),(K AND 56)¥8
230 PSET (XP+2,YP),J AND 7
240 PSET (XP+2,YP+1),J AND 7
250 PSET (XP+3,YP),(J AND 56)¥8
260 PSET (XP+3,YP+1),(J AND 56)¥8
270 NEXT:NEXT
280 'Yの値を変化させる
290 SC=12:NS=1:SCREEN 12
300 FOR Y=1 TO 31:GOSUB 360:NEXT
310 FOR Y=30 TO 0 STEP -1:GOSUB 360:NEXT
320 SC=11:NS=1:SCREEN 11
330 FOR Y=2 TO 30 STEP 2:GOSUB 360:NEXT
340 FOR Y=28 TO 0 STEP -2:GOSUB 360:NEXT
350 GOTO 280
360 'VRAMのYを書き替えるサブルーチン
370 IF NS THEN LOCATE 1,0:PRINT USING"SCREE
N ##";SC:NS=0
380 LOCATE 1,1:PRINT USING"Yの値(輝度)は
##です。";Y
390 LINE(0,48)-(255,175),(Y XOR YY)*8,BF,XO
R:YY=Y:RETURN
400 'on stopで呼び出される
410 SCREEN 0:COLOR 15,4,7:END

```

リスト 4.3 (YJK2.BAS)

```

100 'screen 11 と 12 の色見本
110 'by nao-i on 2. Nov. 1988
120 CALL KANJI:WIDTH 64:DEFINT A-Z:YY=0
130 CLEAR:DEFINT A-Z:YY=0
140 ON STOP GOSUB 380:STOP ON
150 SCREEN 12:COLOR &HF8,0,0:CALL CLS
160 ' VRAM の J と K を設定する
170 FOR K=-32 TO 31:YP=112+K+K
180 PSET (0,YP),K AND 7
190 PSET (0,YP+1),K AND 7
200 PSET (1,YP),(K AND 56)¥8
210 PSET (1,YP+1),(K AND 56)¥8:NEXT
220 FOR J=-32 TO 31:XP=128+J*4
230 LINE(XP+2,48)-(XP+2,175),J AND 7
240 LINE(XP+3,48)-(XP+3,175),(J AND 56)¥8
250 COPY(0,48)-(1,175)TO(XP,YO):NEXT
260 ' Y の値を変化させる
270 SC=12:NS=1:SCREEN 12
280 FOR Y=1 TO 31:GOSUB 340:NEXT
290 FOR Y=30 TO 0 STEP -1:GOSUB 340:NEXT
300 SC=11:NS=1:SCREEN 11
310 FOR Y=2 TO 30 STEP 2:GOSUB 340:NEXT
320 FOR Y=28 TO 0 STEP -2:GOSUB 340:NEXT
330 GOTO 260
340 ' VRAM の Y を書き替えるサブルーチン
350 IF NS THEN LOCATE 1,0:PRINT USING"SCREE
N ##";SC:NS=0
360 LOCATE 1,1:PRINT USING"Y の値 (輝度) は
##です。";Y
370 LINE(0,48)-(255,175),(Y XOR YY)*8,BF,X0
R:YY=Y:RETURN
380 'on stop で呼び出される
390 SCREEN 0:COLOR 15,4,7:END

```

4.6.4 必殺のロジカルオペレーションなのだ

ロジカルオペレーションとは日本語で論理演算のこと。2進数の1ビットごとに
行なわれる計算を意味する。これにはAND、OR、XOR、NOTの4種類がある。

AND 演算とは、2つの値の両方が1になっているビットを1にする演算。たと
えば、

```

X%=&B00000011
Y%=&B00000101

```

に対してAND演算を行なうと、変数X%と変数Y%のビット0(一番右側のビット)
のみが1なので、

X% AND Y%=&B00000001 たように、論理積(明かる色)と論理和(明るい色)

となるわけだ。同様にして、OR 演算では 2 つの値の両方もしくは一方が 1 になっているビットを 1 に、XOR 演算では 2 つの値の一方だけ 1 になっているビットを 1 にする。また NOT とは、1 つの値の各ビットを反転させる演算で、画像データを反転して書き込むことは PRESET 演算とも呼ばれる。言葉で説明するとわかりにくないので、表 4.13 に内容をまとめてみた。

表 4.13: ロジカルオペレーション

記号	意味	例
PSET	指定された色をそのまま書き込む	
AND	元の色との論理積を書き込む	0011 AND 0101 → 0001
OR	元の色との論理和を書き込む	0011 OR 0101 → 0111
XOR	元の色との排他的論理和を書き込む	0011 XOR 0101 → 0110
PRESET	指定された色のビット反転を書き込む	NOT 0101 → 1010

さて、MSX2 や MSX2+ でビデオ RAM を扱うには、こうしたロジカルオペレーションの考えが必要になる。例題として、前に掲載したリスト 4.3 のプログラムで、ビデオ RAM の Y の値を 1 (2 進数で 00001) から 2 (00010) へ増やすことを考えてみよう。つまり、

b₇ b₆ b₅ b₄ b₃ b₂ b₁ b₀
0 0 0 0 1 ? ? ?

というビデオ RAM の内容を、

b₇ b₆ b₅ b₄ b₃ b₂ b₁ b₀
0 0 0 1 0 ? ? ?

に変えるわけだ。ビット 2 から 0 には J または K の値が入っているので、ここを変えてはいけない。

筆者が考えたのは、元の Y の値の 1 と目的の値の 2 の XOR である 3 を 8 倍し、

LINE(0,48)-(255,175), 24, XOR

を行なうこと。1 回の書き替えでの Y 値が 1 から 2 に変わるので、これがリスト 4.3 の 370 行、

LINE(0,Y0)-(255,Y0+127), (Y XOR YY)*8, BF, XOR

の意味。Y は目的の Y の値、YY は元の Y の値を表わしている。

LINE 命令でロジカルオペレーション(論理演算)を指定すると、書き込もうとする色と元の色との間で演算を行なった結果が、ビデオ RAM に書き込まれる。たとえば SCREEN 12 の画面に対して、

```
LINE (0,0)-(255,211), &B11111000, BF, AND
```

を行なうと、ビデオ RAM のビット 7 からビット 3 まではそのままで、ビット 2 からビット 1 までが 0 になり、画面全体が白黒の濃淡で表示される。

4.6.5 いわゆる色化け

YJK 画面のデータ構造は、前に図 4.6 にまとめたとおり。SCREEN 8 と同じように、基本的には画面の 1 ドットがビデオ RAM の 1 バイトに対応。横 256 × 縦 212 ドットの画面を、54,272 バイトのビデオ RAM で表示している。でも、色相を指定する J と K の値は横 4 ドットごとに指定されるので、たとえば SCREEN 12 で、

```
PSET (3,0),3
```

を実行すると、(3,0) の 1 ドットだけでなく (0,0) から (3,0) までの 4 ドットの J の値が変わり、この部分が赤くなってしまう。SCREEN 12 の画面を “sample.s12” というファイルにセーブしてから、リスト 4.4 を実行すると、いわゆる “色化け” が起こってしまうはず。実際に試してみよう。原則として SCREEN 12 では文字や線を表示できないのだ。

リスト 4.4 (S12.BAS)

```
10 ' 色化けの例
20 CALL KANJI
30 SCREEN 12:COLOR &HF9,2
40 BLOAD "sample.s12",S
50 LINE (0,0)-(211,211),3
60 LOCATE 4,6:COLOR &HF9,2
70 PRINT "SCREEN 12 では色が化ける。"
80 GOTO 80
```

4.6.6 SCREEN 10 と 11 は何がどう違うのか

SCREEN 10 と 11 のデータ構造は、前の図 4.7 にまとめたとおり。どちらも機能的にはまったく同じだ。BLOAD された画面を表示する場合にも、これらのスクリーンモードの違いは表われない。それでは何が違うのか。BASIC の命令などで、画面に図形や文字を書き込むとき、問題になってくる。

リスト 4.5 が、その違いを見せるプログラム例。YJK 方式で記録された画面データを BLOAD し、

```
CIRCLE (128,106),100,6
```

を行なってみよう。SCREEN 10 では、ビデオ RAM のビット 3 が 1 になり、ビット 7 からビット 4 に 6 (サークル命令で指定した赤の色番号。2 進数では 0110) が書き込まれる。そのため背景の YJK 画面を壊さずに、赤い円を描くことができる。しかし SCREEN 11 では、ビデオ RAM に直接 6 (つまり 8 ビットに渡って 00000110) が書かれるので、“色化け”が起こってしまう。

この例でもわかるように、YJK 画面に文字や図形を重ねて描くには SCREEN 10 を。YJK の画面データを加工するには SCREEN 11 を使う必要がある。

リスト 4.5 (S10.BAS)

```

100 ' 色化けを回避するには
110 SCREEN 12
120 BLOAD "sample.s12",S
130 SCREEN 10
140 CIRCLE (128,106),100,6
150 FOR I%=1 TO 50:BEEP:NEXT
160 SCREEN 12
170 BLOAD "sample.s12",S
180 SCREEN 11
190 CIRCLE (128,106),100,6
200 GOTO 200

```

4.6.7 SCREEN 11 でもテロップを使うには

リスト 4.6 は、SCREEN 11 の画面に文字を書き込むプログラム。文字を表示するには SCREEN 10 を使うと書いたばかりだけど、PRINT 命令の関係で SCREEN 11 の方が都合がいいこともある。

たとえば、SCREEN 10 で LINE や PUT KANJI 命令を使うと、YJK 画面には RGB 方式で線や文字が書き込まれる。しかし PRINT 命令を使うと背景に正方形の枠が出現し、その中に文字が書かれてしまう。これではどうも、テロップとして使うには適さない。

そこで登場するのが SCREEN 11。SET PAGE 命令でビデオ RAM をページ 1 に切り替え、そこに背景の画面を BLOAD する。そしてページを 0 に戻してから、前景色を 7、背景色を 0 に切り替え、PRINT 命令で文字を表示。さらにこの文字を、COPY 命令で “TAND” を指定して、ページ 1 の画像データの目的の部分に複写するというわけ。

TAND とは、色番号が 0 (透明) の部分は複写せず、ほかの色の部分だけを AND 演算しながら複写する機能。その結果、色番号が 0 で書かれた文字の枠は無視され、7 (水色) で書かれた文字だけが複写される。文字を表示したい部分のビデオ RAM のビット 7 からビット 3 が 0 になるわけだ。

次に、同じ文字を ($C \times 16+8$) で指定された色でページ 0 に書く。この C の値は、プログラムのはじめの方で入力した色番号。

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
C	1	0	0	0			

というように、ビット 7 からビット 4 が目的の色番号で、RGB 表示を指定するビット 3 が 1、残りのビット 2 からビット 0 が 0 となる。この文字を、今度は TOR を指定して複写すると、文字の部分のビデオ RAM の内容は、

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
C	1	元の値					

となり、背景の YJK 画面を壊さずに、RGB 方式で文字を表示できるというわけだ。このように、べつの場所に書いた文字を COPY 命令で複写すると、PRINT 命令にロジカルオペレーション機能がないという欠点を補える。

MSX2+の YJK 方式は難しいけど、うまい使い方をすれば、すばらしい効果を得られるモード。みんな頑張って研究してみよう。

リスト 4.6 (S11.BAS)

```

100 ' SCREEN 11 のテロップ
110 ' by nao-i on 4. Nov. 1988
120 SCREEN 0:WIDTH 32:CALL KANJI
130 DEFINT A-Z:ON STOP GOSUB300:STOP ON
140 FILES:PRINT:INPUT "file name";FF$
150 OPEN FF$ FOR INPUT AS #1:CLOSE #1
160 INPUT "string";SS$
170 INPUT "color(1...15)";C
180 INPUT "X(0...240)";X
190 INPUT "Y(0...196)";Y
200 SCREEN 12:SET PAGE 1,1:BLOAD FF$,S
210 SCREEN 11
220 SET PAGE 0,0:COLOR 7,0:CALL CLS
230 L=LEN(SS$)*8-1
240 LOCATE 0,0:PRINT SS$
250 COPY (0,0)-(L,15),0 TO (X,Y),1,TAND
260 LOCATE 0,0:COLOR C*16+8,0:PRINT SS$
270 COPY (0,0)-(L,15),0 TO (X,Y),1,TOR
280 SET PAGE 1,1
290 GOTO 290
300 '*** called by STOP ***
310 SET PAGE 0,0:COLOR 15,4,7

```

4.6.8 SCREEN 12 で文字表示をするための裏技だ

完全 YJK 方式の SCREEN 12 には、原則として文字や線を表示することはできない。けれどもロジカルオペレーションを活用して、白い文字を表示する裏技があるので紹介しよう。それがリスト 4.7 のプログラム。プログラムの構造自体は、さきほどのリスト 4.6 とほとんど同じ。でも注意してほしいのが、

```
COLOR &HF8, 0
```

と、

```
COPY ... , TOR
```

の 2ヶ所の部分だ。これで、文字を表示する部分の Y の値を最大の 31 に設定でき、背景より白っぽい色で文字を表示できるようになる。逆に文字の色を 3 に、ロジカルオペレーションを TAND にすれば、黒っぽい文字が表示される。

プログラムを実行すると、まずディスクのファイル一覧が表示されるので、背景に使いたい画像ファイルを指定しよう。続いて画面に表示するテロップを書き、位置を座標で入力する。これで SCREEN 12 の画面に、テロップが表示されるはずだ。プログラムを修正して、テロップをいっぱい出すのもおもしろいかも。

リスト 4.7 (S12T.BAS)

```

100 ' SCREEN 12 のテロップ
110 ' by nao-i on 4. Nov. 1988
120 SCREEN 0:WIDTH 32:CALL KANJI
130 DEFINT A-Z:ON STOP GOSUB260:STOP ON
140 FILES:PRINT:INPUT "file name";FF$
150 OPEN FF$ FOR INPUT AS #1:CLOSE #1
160 INPUT "string";SS$
170 INPUT "X(0...240)";X
180 INPUT "Y(0...196)";Y
190 SCREEN 12:SET PAGE 1,1:BLOAD FF$,S
200 SET PAGE 0,0:COLOR &HF8,0:CALL CLS
210 L=LEN(SS$)*8-1
220 LOCATE 0,0:PRINT SS$
230 COPY (0,0)-(L,15),0 TO (X,Y),1,TOR
240 SET PAGE 1,1
250 GOTO 250
260 *** called by STOP ***
270 SET PAGE 0,0:COLOR 15,4,7

```

4.6.9 YJK 方式と VDP のレジスター

YJK 方式による表示を、BASIC の SCREEN 10~12 の代わりに、マシン語プログラムが VDP のレジスターを操作して行なう方法を紹介しよう。コントロールレ

ジスター 25 のビット 3 “YJK” とビット 4 “YAE” で、RGB 方式による画面表示と YJK 方式による表示を選択できる。

まずレジスター 25 以外のレジスターを、SCREEN 8 の場合と同様に設定しよう。BASIC 言語では、スクリーンモードの 10 から 12 を指定することで、YJK 方式を選択するわけだ。でも VDP の機能としては、SCREEN 8 の画面が RGB 方式と YJK 方式に切り替えられる、と考えたほうがわかりやすい。ビット 3 の YJK が 0 で、ビット 4 の YAE も 0 なら、SCREEN 8 そのものが設定されるわけだ。そして、ほかのレジスターはそのままで、ビット 3 の YJK を 1 に切り替えると、SCREEN 12 と同じ YJK 方式の表示が設定される。

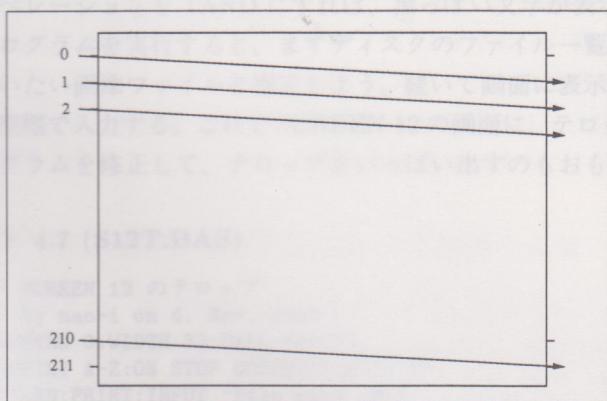
SCREEN 10 や 11 の、YJK と RGB の混在方式での画面を表示するためには、YJK に 1、YAE にも 1 を設定すればいい。なお、YJK が 0 の場合には、スプライトの色がパレットの影響を受けることはないけど、YJK が 1 だとパレットで変更することができる。

4.7 走査線割り込みを研究する

4.7.1 モニター画面を表示する仕組みは？

MSX2 の SCREEN5 の画面が、横 256 × 縦 212 ドットの点（ピクセル）の集まりで表わされることは知ってるよね。これを実際にモニターに表示する仕組みが、図 4.8 に示したもの。画面の左上から右下に向かって、1 ラインずつ順番に各ピクセルのデータを送ることで、画面を表示するというわけだ。このとき、横に並んだ 256 個のピクセルの集まりが、“走査線”と呼ばれるもの。つまり、横に 256 個のピクセルが集まって走査線ができ、縦に 212 本の走査線が集まって画面ができる、という仕組みになっている。

図 4.8: テレビ画面上の走査線のようす



ノン・インターレース画面の表示方法。走査線の隙間がちゃんと見える。

コンピューターの画面だけでなく、普通のテレビ放送なども、この走査線を使って表わされている。日本やアメリカの“NTSC”方式というテレビ放送では、525 本の走査線で画面が表示されているぞ。ただし実際に画面に映る数は約 490 本。残りの走査線には、“同期信号”と呼ばれるテレビを制御するための信号や、文字多重放送のための信号が含まれている。1 秒間に表示される画面の数は 30 枚、1 画面には 525 本の走査線が含まれるわけだから、 $525 \times 30 = 15750$ 本もの走査線が、たった 1 秒間の画面に表示されているわけだ。テレビ放送や MSX の画面表示の仕様にある、“垂直走査周波数 30Hz、水平走査周波数 15.75kHz” という値は、こうした意味を持っている。

16 ビットコンピューターなどに多く見られる、横 640 × 縦 400 ドットの画面表示機能を持つものでは、多くのドットを表示するために水平走査周波数が 24kHz の、専用モニターが必要になる。また最新のコンピューターではさらに画面表示が細か

くなり、31.5kHz や 34kHz のモニターが使われる。

こうした、さまざまな水平走査周波数の画面にも対応したものが、“マルチ・スキャン・モニター”。15.75kHz から 34kHz までのどの周波数にも対応したものから、15.75kHz と 24kHz の 2 種類を切り替えるものなど、さまざまな機種がある。これから新しくモニターを買うなんて場合は、自分が持っているコンピューターや将来使いたいコンピューターの仕様をよく調べて、多くの水平走査周波数に対応するマルチ・スキャン・モニターを選ぼう。

参考までに書いておくと、西ヨーロッパ諸国では“PAL”、フランスやソ連などでは“SECAM”という方式のモニターが使われ、これらの方では NTSC 方式と走査線の数が異なる。だから、ヨーロッパ製のコンピューターや輸出用の MSX を日本のモニターに接続しても、画面を表示することはできない。ソニー・ビクターなどでは NTSC、PAL、SECAM の各方式を切り替えられるモニターを作っているけど、輸出用コンピューターの検査などといった特殊な用途に使われるものなので、非常に高価だ（最近、パナソニックから、世界各国のビデオを再生できるビデオデッキが発売されたようだ）。

また、これも余談になるけど、VTR やビデオディスクの性能を示す“水平解像度”というものは、走査線の数とは関係ない。コンピューター画面の横方向のドット数に相当する、画面の細かさを表わしたもの。垂直解像度は走査線の数と同じで、どの VTR でも同じ。けれども、“ED β ”や“SVHS”では、従来の VTR よりも水平解像度が良くなっている。

4.7.2 インターレース方式によるテレビ放送

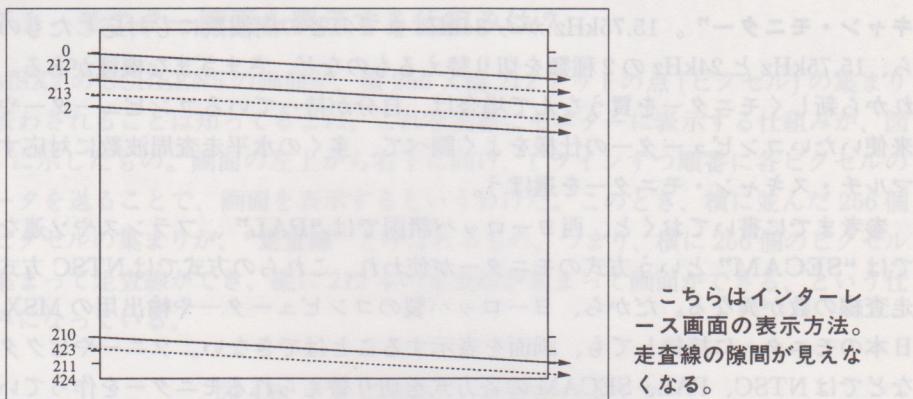
コンピューター画面を表示する仕組みは図 4.8 のようだったけど、これがテレビ放送ということになると、厳密にはちょっと違ってくる。まずは図 4.9 をじっくりと見てほしい。

これは“インターレース”という画面の表示方式を説明したもの。実際のテレビ放送では、この方式が採用されているわけだ。まず画面上端の走査線を 0 番とすると、1 番、2 番、3 番と、図 4.8 で説明したのと同じように走査線を表示していく。そして画面の一番下までいったら、今表示した各走査線の間を埋めるように 212 番、213 番、214 番と、順番に走査線を表示していくわけだ。

インターレースという単語を直訳すると、“織り交ぜる”という意味。図 4.8 で説明した走査線の間を、さらにもう 1 本の走査線で埋めることで、ピクセル間の隙間をなくしているんだ。ちなみに、図 4.8 のように、インターレースによらずに全部の走査線を順番に表示する方式を、“ノン・インターレース”という。

それでは、テレビ放送でなぜこのインターレース方式が採用されたかといえば、

図 4.9: インターレースモードではこうなるぞ



さまざまな原因から発生する雑音(ノイズ)と、それにより起きる画面の乱れを目立たなくするため。たとえば、0番と1番の走査線が雑音によって乱れても、多少の時間をおいてから、その間に表示される212番の走査線が正常ならば、画面の乱れも目立たないというわけだ。電波が空中を飛んでくる間に受ける雑音や、ほかの電化製品の影響などで発生する雑音、電源コンセントから拾う雑音など、テレビ放送がさまざまな雑音から影響を受けやすいだけに、このインターレースというのは有効な方式なんだ。

これとは逆にインターレース方式の欠点は、となりあう走査線の位置がわずかにずれているために、画面がゆれて見えること。テレビ放送のように動きがある画面ではさほど気にはならないけど、コンピューター画面のように細かい文字を静止した状態で表示する場合などには、意外とちらつきが目立ちやすい。このため、MSXも含めて、多くのコンピューターでは、ノン・インターレースでの画面表示が通常は使われている。

4.7.3 MSX2におけるインターレース画面

多くのコンピューターでは、“通常は”ノン・インターレース画面が使われる……と条件付きで書いた理由は、MSX2以降ではインターレース画面も使えるから。

どうして、ちらつきの多いインターレース方式を採用したかという第1の目的は、家庭用のテレビモニター(つまりコンピューター専用でない、いわゆる家庭用テレビと呼ばれるもの)で、縦424ドットの画面表示を行なうためだ。MSX2が開発された1985年当時は、マルチ・スキャン・モニターが一般的でなく、また水平走査周波数24kHzのモニターも10万円を越える高価なものだった。そこで、MSX2と組み

合わせるモニターとしては、一般の家庭用テレビや、水平走査周波数 15.75kHz の低解像度モニターが主流となっていたわけだ。たとえば MSX2+で、

CALL KANJI2

という命令を実行し、画面をインターレースモードに切り替えることで、縦 24 行の漢字表示を可能にしたわけだ。ただし、前にも書いたように、インターレースでの画面表示はちらつきが多く目が疲れやすいので、時々休憩するように心がけよう。

第 2 の目的は、第 1 の目的よりも積極的なもので、MSX2 とテレビ画面を接続することによる“スーパーインポーズ”や“ビデオ・デジタイズ”を可能にするためだ。水平走査周波数 15.75kHz と、インターレース方式を使う MSX2 の画面は、テレビ画面にコンピューターの画面を重ねるスーパーインポーズや、テレビ放送やビデオカメラの画像をコンピューターに取り込む、ビデオ・デジタイズに最適。コンピューター本体にこうした機能をはじめから内蔵したパナソニックの FS-5500 や、オプションのボードを接続することで可能となるビクターの HC-95 などは、ずいぶん前に発売されたマシンながら、いまでも画像データの取り込みや加工といった目的に活躍しているという。

そしてインターレース方式の第 3 の利点は、写真写りがよいこと。ノン・インターレースの画面写真では走査線の隙間が見え、印刷すると“モアレ”という縞模様が現われやすい。でもインターレース画面では走査線の隙間がなく、モアレが出る心配もないわけだ。

MSX2 以降のマシンでは、つぎのように、画面をインターレースモードに切り替えることができる。ただし漢字モードを指定できるのは、MSX2+以降だけだ。

MSX2 マシンの場合 SCREEN ,,,,1

MSX2+マシン以降の場合 CALL KANJI3

4.7.4 走査線割り込みの原理を探る

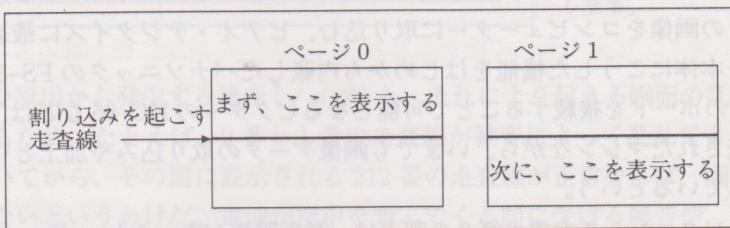
“割り込み”とは、例外的な条件によってプログラムの流れを変えること。たとえばジョイスティックのボタンが押されたら、BASIC のプログラムの流れを変えるための、“ON STRIG GOSUB”という命令も、この割り込みを処理する命令の一種なんだ。

“走査線割り込み”もこれと原理は同じで、指定された番号の走査線の表示が終わると、割り込み処理が行なわれるというもの。でも、走査線割り込みは高速に処理される必要があるので、BASIC でプログラムしていたのでは間に合わない。マシン語プログラムによる割り込み処理が必要になる。

それでは、割り込み処理の原理を簡単に説明する。まず画面表示を制御する VDP は、特定の条件によって、CPU に割り込み信号を送ることができる。この条件には、“ライトペン”、“垂直帰線”、“走査線”という 3 種類があるけど、MSX2 ではライトペンの割り込みは使われない。

まず“垂直帰線割り込み”とは、画面の下端の表示が終わり、次の画面の表示を準備しているときに発生する割り込みで、1/60 秒ごとからならず発生する。これが、ゲーム中の音楽の演奏タイミングの調整などに使われる、MSX の“タイマー割り込み”的正体だ。そして今回問題となるのが、特定の走査線の表示が終わったときに発生する、走査線割り込みだ。この割り込みも、1/60 秒ごとに発生する。

図 4.10: 走査線割り込みの原理なのだ



垂直帰線割り込みと走査線割り込みを組み合わせると、画面の上端と任意の走査線の 2 カ所で、割り込みを発生させることができる。これで、画面を垂直帰線割り込みから走査線割り込みまでの上部分と、走査線割り込みから垂直帰線割り込みまでの下部分に、2 分割できるというわけだ。

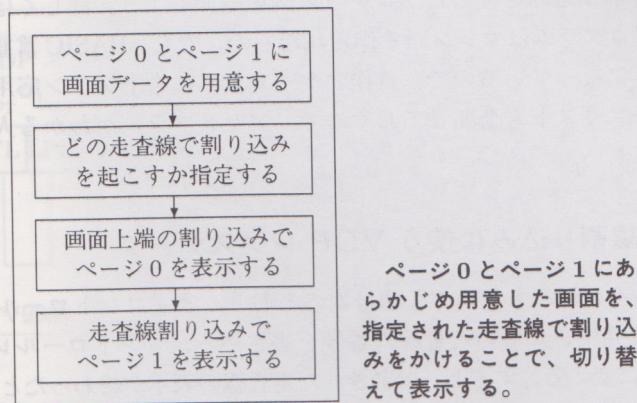
また MSX2 では、“ページ”と呼ばれる複数の画面を持つことができる。ビデオ RAM が 128 キロバイトの MSX2 では、SCREEN 5 または 6 で 4 画面、SCREEN 7 と 8 では 2 画面のページを持てるというわけ。これらを BASIC の、“SET PAGE”命令で切り替えれば、複数の画面を瞬時に表示することができる。

この機能を利用したのが、ゲームソフトなどで活用されている走査線割り込み。マシン語で割り込み処理プログラムを作り、走査線割り込みでページを切り替えることで、画面の上側と下側にべつべつのページを表示する。さらに、VDP のスクロール機能を組み合わせれば、片方の画面だけをスクロールさせることも可能だ。

4.7.5 走査線割り込みの実例を紹介する

筆者は正直にいって、はじめて VDP の仕様書を見たとき、“走査線割り込みなんか何の役に立つのだろう”と疑問を持ったものだった。“MSX2 テクニカルハンドブック”などにも、走査線割り込み機能があるということは紹介されていたものの、

図 4.11: 走査線割り込みの手順



具体的なプログラム例や応用例は掲載されていなかった。それが実際にゲームソフトに応用され、だれもをウーンとうならせたのは、コンパイルが開発しポニーキャニオンから発売された、“ザナック EX”というゲームが登場してからだと思う。

MSX が MSX2 になって拡張された機能のひとつに、“ハードウェア縦スクロール”がある。これはシューティングゲームを作るには非常に便利な機能だったのだけど、そのままで画面全体がスクロールしてしまうため、ゲームには欠かせないスコア表示などを、画面上に固定することができなかった。ところが“ザナック EX”では、画面を高速に縦スクロールさせながら、画面上端の固定位置にスコアを表示させていたのだ。当時の M マガ編集部では、どういうワザを使っているのか話題になつたけど、スコア部分とスクロール部分の境界のちらつきから、走査線割り込みと判明した。と、まあ、一度気付いてしまえば“コロンブスの卵”で、これ以降、走査線割り込みを利用したシューティングゲームが、次々と開発されるようになる。走査線割り込みを使ったゲームソフトを、ポーズキーで停止させると、ふたつの画面のうちどちらかしか表示されない。実際に試してみよう。

さらに MSX2+ では、走査線割り込みと“ハードウェア横スクロール”を組み合わせて、画面の一部分だけを横スクロールさせることも行なわれている。コナミから発売された“F-1スピリット 3D スペシャル”では、ゲーム画面だけを横スクロールさせながら、走査線割り込みによって、画面下部に F1 マシンのコックピットのようすを表示しているようだ。

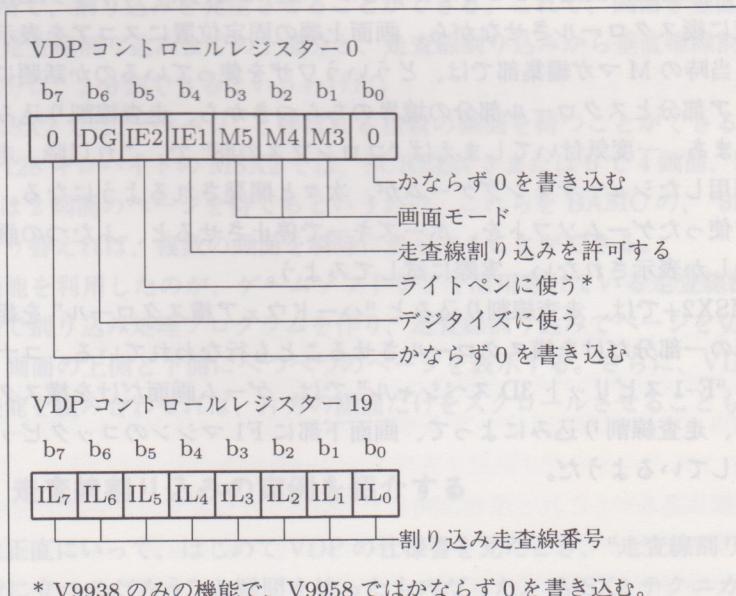
4.7.6 いよいよ実践編はりきっていこう！

実際に走査線割り込みを使ったプログラム例を紹介しよう。詳しくは後で説明するけど、このプログラムはマシン語で作られたもの。でも、BASIC言語から呼び出して使えるようになっているので、自作のゲームなどにもドンドン応用できるはずだ。また、ソースリストも公開しておくので、アセンブラーがわかる人は頑張って解析してみてほしい。

4.7.7 走査線割り込みに使うVDPレジスター

走査線割り込みを起こすための手順は次のとおり。まずコントロールレジスター19に割り込みを発生させたい走査線の番号を書き込み、コントロールレジスター0のビット4を1に変える。すると、指定された走査線の表示が終わったときに、VDPがCPUに対して割り込みをかける。また、割り込みがかかったときに、ステータスレジスター1のビット0が1であれば、割り込みの原因が走査線割り込みであることがわかる。これらのことまとめたのが、図4.12と4.13だ。

図4.12: 走査線割り込みを発生するVDPレジスター





とまあ、以上が走査線割り込みに直接関係するレジスターの説明。それでは、べつのレジスターを使って、画面の切り替えとハードウェア縦スクロールを制御してみよう。

SCREEN 5 または SCREEN 6 で、コントロールレジスター 2 のビット 6 と 5 にページ番号を書き込むと、BASIC の “SET PAGE” 命令と同様に、ディスプレーページ(画面に表示するページ)を切り替えることができる(図 4.14 参照)。ビット 7 には 0 を、ビット 4 から 0 には 1 をそれぞれ書き込むわけだ。これと同じように SCREEN 7 や SCREEN 8 の場合は、ビット 5 でページを指定し、ビット 6 にはかならず 0 を

図 4.14: 画面切り替えを制御する VDP レジスター

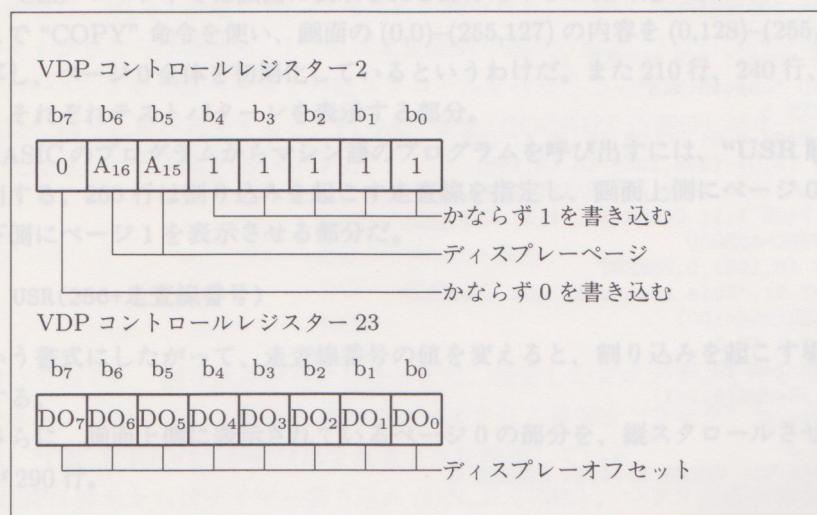
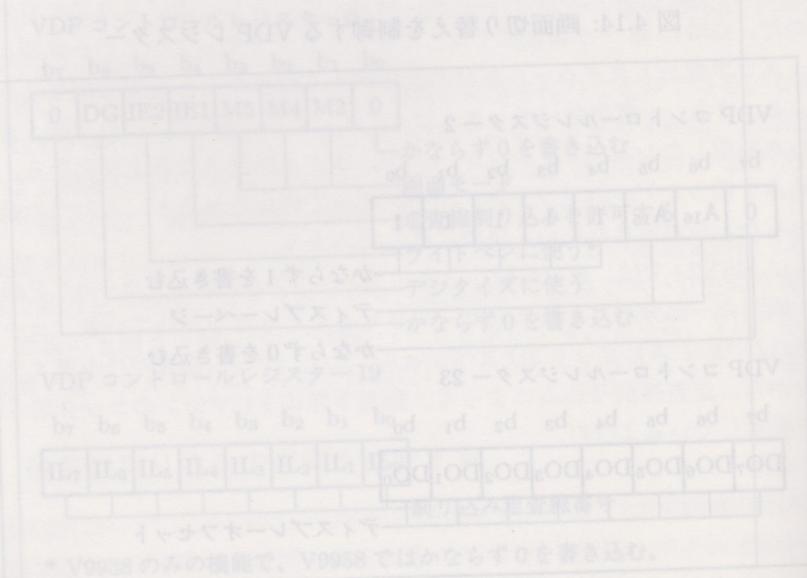


図 4.15: ハードウェア縦スクロールの仕組み

コントロールレジスター 23 が 0 の場合		コントロールレジスター 23 が 128 の場合	
VRAM 番地	表示位置	VRAM 番地	表示位置
0 6A00H 7FFFH	0 212 255	0 2A00H 4000H 7FFFH	128 212 0 128

書き込むことで、ページを切り替えることが可能になる。

また、コントロールレジスター 23 を使うと、ハードウェア縦スクロールを行なうことができる。具体的には図 4.15 のように、レジスターに設定した値によって画面の表示位置とビデオ RAM の番地の対応が変わり、画面が縦スクロールするという仕組みだ。ただし、このハードウェア縦スクロールを使うと、割り込みを起こす走査線の番号もずれてしまうので、走査線番号に縦スクロール量を加えるなどの補正が必要になる。サンプルプログラムでは、“ON_VSYNC” からはじまるサブルーチンで、この補正をしている。



4.7.8 アセンブルの方法と BASIC 部分の動作

走査線割り込みを起こすためのサンプルプログラムは、アセンブラーで作られた部分と BASIC 言語で作られた部分からできている。アセンブラーの部分の動作原理は次に説明するとして、まずはアセンブルの方法と、BASIC 部分の動作原理から紹介しよう。

アセンブラーのプログラム（ソースリスト）自体は、後のページのリスト 4.9 に掲載しておいた。これを MSX-DOS のスクリーンエディターなどで入力し、“ON-SCAN.Z80”というファイル名でセーブしよう。次に“MSX-DOS TOOLS”に入っている、“M80.COM”と“L80.COM”というプログラムを使い、次の手順どおりにアセンブルとリンクをする。“ONSCAN.BIN”というマシン語ファイルができれば完了だ。なお“DEL ONSCAN.BIN”という部分は、アセンブルをやり直すときに古いファイルを消すためのもの。一度目は不要のものだ。

```
M80 ,=ONSCAN.Z80/R/Z  
L80 ONSCAN,ONSCAN/N/E  
DEL ONSCAN.BIN  
REN ONSCAN.COM ONSCAN.BIN
```

それでは次に、マシン語ファイルをコントロールする、BASIC プログラム（リスト 4.8 参照）の動作原理を説明する。

まず 190 行から 200 行の部分は、ビデオ RAM のページ 0 を初期化するためのもの。画面を縦スクロールさせる場合にはページ 0 全体を初期化する必要があるのだけど、“CLS” コマンドでは画面に表示される部分だけしか初期化することができない。そこで“COPY” 命令を使い、画面の (0,0)–(255,127) の内容を (0,128)–(255,255) に複写し、ページ 0 全体を初期化しているというわけだ。また 210 行、240 行、250 行は、それぞれテストパターンを表示する部分。

BASIC のプログラムからマシン語のプログラムを呼び出すには、“USR 関数”を利用する。260 行は割り込みを起こす走査線を指定し、画面上側にページ 0 を、画面下側にページ 1 を表示させる部分だ。

USR(256+走査線番号)

という書式にしたがって、走査線番号の値を変えると、割り込みを起こす場所が変化する。

さらに、画面上側に表示されているページ 0 の部分を、縦スクロールさせているのが 290 行。

USR(512+走査線番号)

という書式で値を指定する。また、今回は使っていないけど、

USR(768 + 走査線番号)

を指定すると、画面下側に表示されるページ1の部分を縦スクロールさせることができる。走査線割り込みを中止するには、

USR(0)

を実行すればいい。

なおこのプログラムでは、アセンブラー部分を簡単にするためUSR関数のパラメーターを整数に限っている。だから、

USR(868.0)

とか

USR(100+A!)

のような、実数のパラメーターは使えない。

リスト 4.8 (ONSCAN.BAS)

```

100 ,
110 ' onscan1.bas : test interrupt on scan
120 ' by nao-i on 24. Sep. 1989
130 '
140 CLEAR 300,&HAFFF
150 DEFINT A-Z
160 OPEN "GRP:" FOR OUTPUT AS #1
170 BLOAD "onscan.bin"
180 SCREEN 5
190 SET PAGE 0,0: COLOR 15, 6,1: CLS
200 COPY (0,0)-(255,127) TO (0,128)
210 CIRCLE (128,106),80: PAINT (128,106)
220 SET PAGE 1,1: COLOR 15, 1,1: CLS
230 DEFUSRO=&HB000
240 PSET (8,192),0,PRESET
250 PRINT #1,"This area will not scroll."
260 JK=USR(256+100)
270 FOR J=1 TO 5
280   FOR I=0 TO 255
290     JK=USR(512+I)
300   NEXT I
310 NEXT J
320 JK=USR(0): COLOR 15,4,4: SCREEN 0
330 END

```

4.7.9 アセンブラー部分の動作原理だ

それではいよいよ、リスト 4.9 の、実際に走査線割り込みを処理するメインプログラムの動作原理を解説する。

まず、“ASEG”からの 3 行は、M80.COM や L80.COM を使って BASIC のサブルーチンなどの特殊なオブジェクトを作るための命令。M80.COM と L80.COM は通常ペアにして使い、アセンブラーで作られたプログラムからマシン語のファイルを作り出すものだ。ただ、このとき作られるファイルの拡張子は “COM”。つまり MSX-DOS 上で実行可能なマシン語ファイルになってしまう。そこで今回のように BASIC で扱えるファイルを作りたい場合には、この ASEG からの命令が必要になるというわけだ。

また、BASIC で BLOAD できる形式のマシン語プログラムの先頭には 7 バイトのヘッダーがあり、

FEH	ロード開始番地		
EXTRN	VDPSTA	013FH	ロード終了番地
VDPSTA	013H	実行開始番地	

といったデータがそれぞれ書き込まれている。これらを指定しているのが、ASEG の次の 4 行だ。

さて、“AD_LOAD:” からうしろの部分が、プログラムの本体になる。最初に行なっているのが、USR 関数のパラメーターの処理。パラメーターが整数ならば A レジスターの内容が 2 になるので、それを確認している。また、入力されたパラメーターの値は、HL+2 番地と HL+3 番地に記録される。このとき、パラメーターの上位バイトが 0 ならば後始末、1 ならば走査線割り込みの設定、2 ならばページ 0 の縦スクロール、3 ならばページ 1 の縦スクロールを行なうというわけ。

実際に割り込みが発生すると、FD9AH 番地 (HOOKDT の部分) がコールされることになる。ここからの 5 バイトに、

RST	30H
DB	スロット番号
DW	番地
RET	

を書いておくと、割り込みが発生したときに、指定したプログラムを呼び出すことができる。このように、何かの条件でコールされる場所を “フック” という。ここでは、“ON_H.KEYI:” が呼び出されるように準備している。

FD9FH 番地からはタイマー割り込み (ON_H.TIMI:)、つまり垂直帰線割り込みを呼び出す部分。ここでは、割り込みが発生したときの VDP ステータスレジスター

0の値がAレジスターに記憶されるので、AFレジスターを書き替えてはいけない。もし、どうしてもフックを書き替えるならば、サンプルプログラムのようにフックの元の値を保存しておき、割り込み処理が終わったときに保存したフックへジャンプするようにしよう。

割り込みの発生で呼び出され、VDPを操作するプログラムを，“ON_VSYNC:”と、“ON_SCAN:”からはじまるサブルーチンにまとめておいた。ここを書き替えれば、走査線割り込みをべつの目的に使えるだろう。

サブルーチン“_VDPSTA”は、VDPのステータスレジスターを読む。サブルーチン“WRTVDP”は、VDPのコントロールレジスターに指定された値を書き込んで、その値をそれぞれの保存場所(表4.9参照)に保存する。もっとも、前にも書いたように、MSX2や2+のROMにはこれらのサブルーチンと同じ機能のBIOSがあるので、普通は自分でサブルーチンを作らずにBIOSを使えばいい。でも、これらのBIOSはサブROMにあったり、処理中にサブROMを呼び出したりするので、多少時間がかかるという難点がある。そのため、今回のような割り込み処理には都合が悪いので、あえてBIOSを使わなかった。

これはBASICのマシン語サブルーチンには関係ないことだけど、DOSのプログラムでは割り込み処理プログラムを4000Hよりも大きい番地に置く必要がある。これ以下では特定のスロット構成のMSXで不都合が起きるからだ。

```

100 REM ***** (ONSCAN.BAS)
110 REM 1. Aを読み出し確認→A=0なら、奥側のスクリーンの左端 H2U, 右側のスクリーンの右端 H3U
120 REM 2. サブROM呼び出し。引数を「(A)」で初期化せず、引数の値をスクリーン内の一文字
130 REM 3. SCREEN 0, COLOR 15, PAINT 128, LINE 3, PSET 0, RST 30H
140 REM 4. DEFUSRO=AEB000
150 REM 5. PSET (8,192),0,PRESET
160 REM 6. PRINT #1,"This area will not scroll"
170 REM 7. JK=USR(268+100)
180 REM 8. FOR J=1 TO 6
190 REM 9. ここで出力範囲(256×192)をオーバーするが、これはアバウト
200 REM 10. JK=USR(512+1)
210 REM 11. ここでNEXT #1 "で"で"を演算するが、これはアバウト
220 REM 12. JK=USR(0): COLOR 15,1,4, SCREEN 0
230 REM 13. ここで出力範囲(256×192)をオーバーするが、これはアバウト
240 REM 14. RST 30H
250 REM 15. END

```

リスト 4.9 (ONSCAN.Z80)

```

;
;       onscan.z80 : test program for interrupt on scan
;       by nao-i on 26. Sep. 1989
;
;       called as USR function from BASIC
;       USR(&H00xx)      restore registers and hooks
;       USR(&H01xx)      set interrupt line
;       USR(&H02xx)      set display offset line of page 0
;       USR(&H03xx)      set display offset line of page 1
;
;       .Z80
START   EQU    0B000H ; address to load and execute
USE_SUB  EQU    0
USE_WRTVDP EQU    0
;
;       IF      USE_WRTVDP
WRTVDP  EQU    0047H
ENDIF
EXTROM  EQU    015FH
VDPSTA  EQU    0131H
SETPAG  EQU    013DH
;
RAMAD2 EQU    0F343H      ; slot of RAM in page 2
RAMAD3 EQU    0F344H      ; slot of RAM in page 3
RGOSAV  EQU    0F3DFH
DPPAGE  EQU    0FAF5H
ACPAGE  EQU    0FAF6H
H.KEYI  EQU    0FD9AH
H.TIMI  EQU    0FD9FH
RG8SAV  EQU    OFFE7H
;
ASEG
ORG    100H           ; to make .COM file
.PHASE START-7
;
DB      OFEH           ; header to BLOAD
DW      AD_LOAD         ; address to load
DW      AD_NEXT-1        ; address of end of file
DW      DO_NOTHING       ; address to execute
;
;       CAL
AD_LOAD:
PUSH   AF
PUSH   HL
PUSH   DE
PUSH   BC
CP     2
JR     NZ,RESET_SCAN   ; parameter is not integer
INC    HL
INC    HL
LD     E,(HL)
INC    HL
LD     D,(HL)          ; DE = parameter of USR()
;
```

```

LD      A,D
OR      A
JR      Z,RESET_SCAN
DEC    A
JR      Z,SET_SCAN
DEC    A
JR      Z,SET_DO
DEC    A
JR      Z,SET_D1
JR      RESET_SCAN
;
SET_SCAN:
LD      A,E
LD      (ILSAV),A
CALL   SET_ILREG      ; set interrupt line
LD      A,(HOOKED)
OR      A
JR      NZ,RET_BASIC  ; hook is already set
;
LD      HL,H.KEYI
LD      DE,HOOKSA
LD      BC,10
LDIR   ; save hooks
LD      HL,HOOKDT
LD      DE,H.KEYI
LD      BC,10
DI
LDIR   ; set hooks
LD      A,(RAMAD2)
LD      (H.KEYI+1),A
LD      (H.TIMI+1),A
LD      A,(RGOSAV)
OR      00010000B
LD      B,A
LD      C,0
CALL   WRTVDP        ; interrupt on
LD      A,1
LD      (HOOKED),A
JR      RET_BASIC
;
RESET_SCAN:
CALL   RESET_SCAN_SUB
JR      RET_BASIC
;
SET_DO:           ; set display offset of page 0
LD      A,E
LD      (DOVAL),A
JR      RET_BASIC
SET_D1:           ; set display offset of page 1
LD      A,E
LD      (D1VAL),A
;
RET_BASIC:
EI
POP   BC

```

```

        POP      DE
        POP      HL
        POP      AF
DO NOTHING:
        RET

;
RESET_SCAN_SUB:
        DI
        LD       A,(RGOSAV)
        AND     11101111B
        LD       B,A
        LD       C,0
        CALL    WRTVDP      ; interrupt off
        LD       BC,23       ; write 0 into reg#23
        CALL    WRTVDP      ; restore display offset
        XOR     A
        LD       (DPPAGE),A
        LD       BC,1F02H     ; write 1FH into reg#2
        CALL    WRTVDP      ; set page 0
        LD       A,(HOOKED)
        OR      A
        RET     Z
        LD       HL,HOOKSA
        LD       DE,H.KEYI
        LD       BC,10
        LDIR    A             ; restore hooks
        XOR     A
        LD       (HOOKED),A
        RET

;
SET_IILREG:
        LD       A,(ILSAV)
        LD       HL,DOVAL
        ADD     A,(HL)       ; interrupt line = (ILSAV) + (DOVAL)
        LD       B,A
        LD       C,19
        JP      WRTVDP      ; write interrupt line # into reg#19

;
ON_H.KEYI:                      ; called from H.KEYI
        LD       A,1
        CALL   _VDPSTA      ; read status reg#1
        AND     1
        CALL   NZ,ON_SCAN
        JR      HOOKSA

;
ON_H.TIMI:                      ; called from H.TIMI
        PUSH   AF
        CALL   ON_VSYNC
        POP    AF           ; do not change AF in H.TIMI
        EI
        JR      HOOKSA+5

;
HOOKDT:
        RST    30H
        DB      0

```

```

DW      ON_H.KEYI
RET
RST    30H
DB      0
DW      ON_H.TIMI
RET
;
;      data area
;
HOOKED: DB      0      ; non-zero if hooks have been set
HOOKSA: DS     10      ; save area for hooks
DOVAL:  DB      0      ; display offset of page 0
D1VAL:  DB      0      ; display offset of page 1
ILSAV:  DB      0      ; interrupt line
;
;      _VDPSTA : read a VDP status register
;      Entry   A      VDP status register #
;      Return  A      value of the status register
;      Modify  AF, BC
;      Note    compatible with ROM-BIOS
;      DI when return
;
_VDPSTA:
IF      USE_SUB
LD      IX,_VDPSTA
JP      EXTROM
ELSE
DI
AND    00001111B
LD      B,A
LD      A,15
LD      C,A
CALL   WRTVDP
LD      BC,(6)
INC    C
IN     A,(C)
PUSH   AF
LD      BC,15
CALL   WRTVDP
POP    AF
RET
ENDIF
;
;      WRTVDP : write a byte into VDP register
;      Entry   B      datum to write
;              C      VDP register #
;      Return  none
;      Modify  AF, BC
;      Note    compatible with ROM-BIOS
;              DI when return
;
IFE    USE_WRTVDP
WRTVDP:
PUSH   HL
PUSH   DE
;
```

```

LD      D,B          ; =datum
LD      A,C          ; =register #
LD      HL,RGOSAV
CP      8
DI
JR      NC,SAVEREG
LD      HL,RG8SAV-8
CP      24
JR      NC,NOSAVE

SAVEREG:
XOR     A
LD      B,A          ; BC=register #
ADD     HL,BC          ; HL=RG?SAV
LD      (HL),D          ; save datum
;                                ; BC=register #        ; HL=RG?SAV
;                                ; save datum
NOSAVE:
LD      A,C          ; =register #
LD      BC,(7)          ; BC=(A+(1)(0)-(1)(1))
INC     C
OUT    (C),D          ; (C)=D
AND     00111111B
OR      10000000B
OUT    (C),A          ; (C)=A
POP     DE
POP     HL
RET
ENDIF
;
;                                ; IFE USE_WRTVDP
;
;                                ; please modify following subroutines as you need
;
ON_VSYNC:
XOR     A
LD      B,(DPPAGE),A
LD      BC,1F02H          ; write 1FH into reg#2
CALL    WRTVDP          ; set page 0
LD      A,(DOVAL)
LD      B,A
LD      C,23
CALL    WRTVDP          ; set display offset
JP      SET_ILREG         ; set interrupt line
;
ON_SCAN:
LD      A,1
LD      B,(DPPAGE),A
LD      BC,3F02H          ; write 3FH into reg#2
CALL    WRTVDP          ; set page 1
LD      A,(D1VAL)
LD      B,A
LD      C,23
JP      WRTVDP          ; set display offset
;
AD_NEXT EQU      $          ; end of program + 1
.DEPHASE
;
END

```

4.7.10 走査線割り込みのマシン語ルーチンだ

最後に、走査線割り込みのためのプログラムのソースリストを打ち込むのが面倒な人や、アセンブラーを持っていない人のために、自動的にマシン語ファイルを作るBASICプログラムを掲載する。以下のプログラムを打ち込み、実行させると、自動的に“ONSCAN.BIN”というファイルを作成してくれる。

リスト 4.10 (MKONSCAN.BAS)

```

10 CLEAR 100,&HCFFF:DIMD(15)
20 PRINT"Making onscan.bin":AD=&HB000:C=0:L=0
30 FOR I=0TO15:READ A$:IF A$="*" GOTO100
40 A=VAL("&h"+A$):C=(C+A) AND 255:D(I)=(D(I)+A) AND 255
45 POKE AD,A:AD=AD+1:NEXT
50 READA$:A=VAL("&h"+A$):L=L+1
55 IF C<>A THEN PRINT "Error in line ";990+10*L:END
60 GOTO 30
100 '
110 PRINT"Saving"
120 BSAVE"onscan.bin",&HB000,&HB14D
130 PRINT"Done.":END
1000 DATA F5,E5,D5,C5,FE,02,20,53,23,23,5E,23,56,7A,B7,28, 5D
1010 DATA 4A,3D,28,08,3D,28,49,3D,28,4C,18,3F,7B,32,D9,B0, 00
1020 DATA CD,A1,B0,3A,CC,B0,B7,20,41,21,9A,FD,11,CD,B0,01, 33
1030 DATA 0A,00,ED,B0,21,C2,B0,11,9A,FD,01,0A,00,F3,ED,B0, B0
1040 DATA 3A,43,F3,32,9B,FD,32,A0,FD,3A,DF,F3,F6,10,47,0E, 20
1050 DATA 00,CD,F4,B0,3E,01,32,CC,B0,18,0F,CD,70,B0,18,0A, B4
1060 DATA 7B,32,D7,B0,18,04,7B,32,D8,B0,FB,C1,D1,E1,F1,C9, 61
1070 DATA F3,3A,DF,F3,E6,EF,47,0E,00,CD,F4,B0,01,17,00,CD, E0
1080 DATA F4,B0,AF,32,F5,FA,01,02,1F,CD,F4,B0,3A,CC,B0,B7, 54
1090 DATA C8,21,CD,B0,11,9A,FD,01,0A,00,ED,B0,AF,32,CC,B0, 67
1100 DATA C9,3A,D9,B0,21,D7,B0,86,47,0E,13,C3,F4,B0,3E,01, 2F
1110 DATA CD,DA,B0,E6,01,C4,32,B1,18,13,F5,CD,1C,B1,F1,FB, BA
1120 DATA 18,10,F7,00,AE,B0,C9,F7,00,BA,B0,C9,00,00,00,00, 2A
1130 DATA 00,00,00,00,00,00,00,00,00,00,00,F3,E6,0F,47,3E,0F, A6
1140 DATA 4F,CD,F4,B0,ED,4B,06,00,0C,ED,78,F5,01,0F,00,CD, E7
1150 DATA F4,B0,F1,C9,E5,D5,50,79,21,DF,F3,FE,08,F3,30,07, EB
1160 DATA 21,DF,FF,FE,18,30,04,AF,47,09,72,79,ED,4B,07,00, 5D
1170 DATA 0C,ED,51,E6,3F,F6,80,ED,79,D1,E1,C9,AF,32,F5,FA, F3
1180 DATA 01,02,1F,CD,F4,B0,3A,D7,B0,47,0E,17,CD,F4,B0,C3, E7
1190 DATA A1,B0,3E,01,32,F5,FA,01,02,3F,CD,F4,B0,3A,D8,B0, 0D
1200 DATA 47,0E,17,C3,F4,B0,* register #
      RETURN none
      Modify AF, BC
      Note compatibility with 8080
      DI when return
      I + memory to bus :
      IPE USE_WRTVDP
      WRTVDP:
      PUSH HL
      PUSH DE

```