月刊マイコン別冊

EXPRESSION OF SENSIBILITY & COMMUNICATION

Super MZ 活用研究

ハード&ソフトマスター編

高橋雄一／多部田俊雄共著



ゲームプログラムから内部解析までスーパー情報満載!!

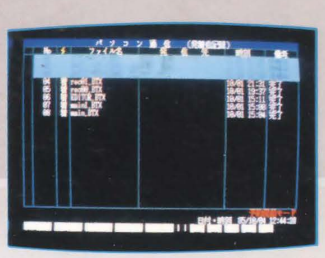
SHARP



●「テレホンソフト」のメインメニュー。時刻表示とともに6つの処理内容が表示され、ディファイナブルファンクションキーにより仕事を選択します。



●データベースで作成した電話帳の表示・検索を指定した画面。TABキーによって検索項目が変更できます。



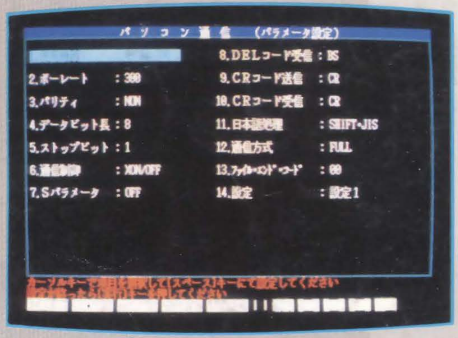
●パソコン通信の発着信記録画面。着発信の順番や区別、ファイル名、発信先、時刻等が一覧表示されます。



●電話帳検索で指定されたメンバーの照会。入力データのすべてが表示されます。



●データ設計のメニュー。基本設計、項目設計、グループ作成、レイアウト作成の4つに分かれています。



●パソコン通信の通信パラメータ設定。設定は全部で13項目。最終設定が自動的にディスクに記録されます。



●発信ファイルの選択画面。ウインドウにはファイル一覧が表示されます。

ぼくたちのネットワークが拡がった。

パソコン通信時代がやってきた。友だちへのメッセージを電子メールで送る。BBS(電子掲示板)を通じて、仲間同士いろんな情報を交換しあう。はたまた、ボイスメールで気になるあのコに声でラブコール。スーパーMZは、その日からパソコン通信が体験できる「テレホンソフト」を標準装備。今日から、僕の部屋は通信ステーションに変貌する。

|新|発|売|

Super MZ

8ビットパーソナルコンピュータMZ-2500シリーズ

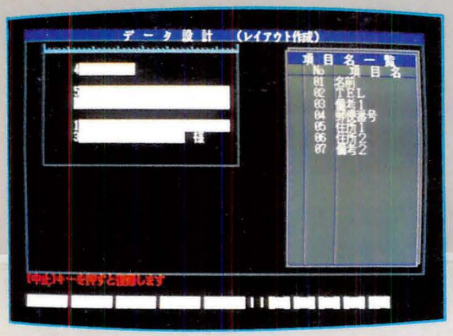
Model 20(MZ-2511・640KB3.5″FD1基付)標準価格168,000円
Model 30(MZ-2521・640KB3.5″FD2基付)標準価格198,000円

●640×400ドット、256色同時表示、圧倒的なスピードと色彩感覚で迫る高密度グラフィック●アナログ音声対応のボイスレコーダ搭載●SSG(8オクターブ3重和音)に加えFM音源(8オクターブ3重和音)を採用した多彩なサウンド機能●JIS第1/第2水準漢字ROM標準装備●最大256KBの大容量メモリ(標準128KB)●強力な漢字BASIC-M25/S25採用●スイッチひとつで切替えできるMZ-2000/2200モード、MZ-80Bモードを装備して、多くの資産を継承。

〈主なオプション〉●14型カラーディスプレイMZ-1D22標準価格108,000円●モデムホンMZ-1X19標準価格98,000円●ボイスコミュニケーションインターフェイスMZ-1E26標準価格24,800円●80桁漢字ドットプリンタMZ-1P18標準価格188,000円●マウスMZ-1X10標準価格19,800円●カラーパレットボードMZ-1M10標準価格14,500円●PERSONAL CP/M™※ MZ-6Z001標準価格16,800円※CP/Mは米国デジタルリサーチ社の登録商標です。

シャープ株式会社 本社 〒545 大阪市阿倍野区長池町22番22号☎(06)621-1221(大代表)●お問い合わせは…本社内国内情報システム営業本部まで

各画面は「テレホンソフト」の使用例です。



●作成したデータをプリンタに印刷するためのレイアウトも作成できます。



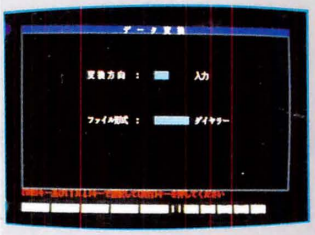
●グループ作成により、作成したデータを目的ごとのグループに分け、効率よく利用できます。



●データ設計のうちの基本設計メニュー。データディスクの作成やデータドライブの選択を行います。

通信機能を搭載し、ニューメディアに対応。





●データ変換により、「テレホンソフト」で作成したデータはBASICやダイヤラーでも活用できます。もちろんその逆もOK。

■主な国内ネットワーク

名　称	問い合わせ先
TeleStar	㈱テレスター (03)375-3800
アスキーネットワーク	㈱アスキー (03)486-7111
J&P HOTLINE	上新電機㈱ (06)644-5550
日本マイコンクラブ マイコンコミュニケーション	日本マイコンクラブ (03)438-1869
JAL旅行情報システム	日本航空㈱文化事業センター (03)284-2766
POPCOM NET	㈱新企画社 (03)263-6940
CANS	千代田常磐マイコンクラブ (0473)42-0584

「テレホンソフト」の主要機能

- オートダイヤル
- リダイヤル
- データ通信
- ターミナル
- データベース
- データ設計
- データ変換
- ダイレクトダイヤル
- 留守番電話*
- ボイスメール*

●テレホンソフトの機能を活用するためには別売のモデムホンが必要です。また※印の機能については、さらに別売のボイスコミュニケーションインターフェイスが必要です。



●モデムホン

写真はModel 30です。写真の14型カラーディスプレイ、モデムホンは別売です。また本体に装着されているカセットテープは撮影用で、付属品・市販品ではありません。画面はハメコミ合成です。

MZ-2500
資料請求券
活用研究



Super MZ 活用研究

ハード&ソフト マスター編

CONTENTS

第1章 MZ-80Kから スーパーMZ誕生まで

／高橋雄一

■シャープパソコンのルーツ「MZ-80K」……8
■MZ-80Kのマイナーチェンジ機「MZ-80C」～「MZ-1200」……11
■えっ！シャープのPCシリーズ……13
■64Kバイト！オールRAM「MZ-80B」……14
■640×200ドット，カラー対応「MZ-2000」……15
■家庭用テレビが使える「MZ-700」……16
■世界初パソコンテレビX1……17
■オールインワンタイプからコンポタイプへ……18
■シャープ初の16ビットパソコン登場……19
■淳美のパソコンMZ-1500！！……20
■高漢度パソコンX1turbo……20
■そしていよいよスーパーMZ登場……21

第2章 ハードウエア解剖

／高橋雄一

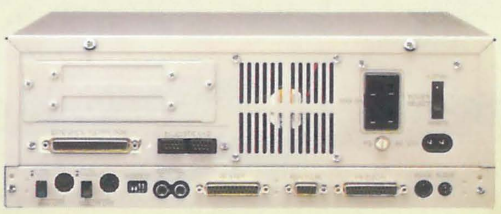
■スーパーMZの外観……23
■高速度CPU Z80Bを採用……27
■ダントツのグラフィック機能……27
■日本語対応のテキスト(文字)画面……32
■ワープロ顔負け？の漢字機能……33
■PCGや外字も標準装備……36
■盛りだくさんの画面制御……37
ハードウエアでビューポートをサポート……38
プライオリティ機能……38
リードモディファイトライトモードとサーチモード……40
■大容量128K RAM実装……41
■FM音源も標準装備……44
■年まで刻めるバッテリバックアップカレンダクロック採用……46
■1/10秒まで刻めるストップウォッチ機能……47
■キーボード……48
■3.5インチ2DDマイクロフロッピーディスクを採用……54
■マウス端子も標準装備……56
■ジョイスティック端子も標準装備……57
■カセットデッキも標準装備……57
■RS-232Cは2チャンネル装備……59
■音声合成もサポート(オプション)……59
■電話機もサポート(オプション)……60
■多種のプリンタに対応……61
■あらゆるRGBディスプレイに対応……63
■拡張I/Oポート……64
■スーパーMZ大解剖……66

第3章 ソフトウエア解剖

／高橋雄一

■史上最強のBASIC……71
すべての演算が16桁で可能……71
ラベルを使用可能……72
漢字変数名使用可能……72
構造化プログラミングをさらに徹底……73
ユーザー座標系も当然装備……74
タイリングペイントやラインスタイルもOK(PAINT,LINE)……75
ペンスタイルが設定可能(PEN)……75
グラフィック画面の保存が可能(GSAVE, GLOAD)……76
シンボル機能(SYMBOL)……76
部分ハードコピーも可能(HCOPY)……76
色の置換え機能(COLOR REPLACE)……77
部分移動機能(MOVE)……77
多角形……77
指定パターンの直接表示機能(PATTERN)……78
PUT@KANJIとROLLも装備……78
各種割込みをサポート……78
&O, &H以外に&B, &J, &Kをサポート……79
連続POKE, 連続CHR$, HEXCHR$, ASCCHR$……79
ユニークな関数，いろいろ……80
ON～RESTORE, ON～RETURN, ON～RESUME……81
省略形による入力が可能……81





CONTENTS

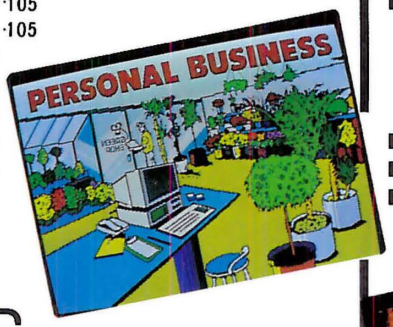


エディットモード……81
サーチ機能……82
ブレークポイントとシングルステップデバッグモード……82
変数保護機能……82
ガーベージコレクションを解決……83
OPTION LIST機能……83
OPTION ANGLE機能……84
OPTION SCREEN機能……84
階層ディレクトリをサポート……85
疑似可変長レコード……85
その他……85
BASIC-M25のまとめ……87
■従来互換のBASICも搭載……91
■ファイルコンバータ……92
■プリンタ選択ユーティリティ……94
■PCGエディタ……94
■スーパーヘルプ機能……95
電卓機能……95
カラーシミュレーション機能……96
オートダイヤラー機能……97
■ロードテスト……97

第4章 テレホンソフトのすべて

／高橋雄一

■通信を行うのに必要なもの……105
■メインメニュー……105
1.電話(電話帳)……108
2.パソコン通信……110
3.データベース……113
4.データ設計……115
5.データ変換……116
6.DATE／TIME……116
■＋α……117
■まとめ……117



第5章 BASICプログラミング入門

／多部田俊雄

■プログラミング0「プログラムの基本」……119
プログラミングの覚え方……119
ゲームの自作……119
基本命令……120
■プログラミング1「画面構成」……120
KMODE命令……121
KEY LIST(KLIST)命令……121
CLS命令……122
ゲームの画面構成……122
■プログラミング2「キー入力」……122
INKEY$関数……122
REPEAT命令……123
CLICK命令……123
ビーム砲移動……123
■プログラミング3「乱数発生」……124
RND関数……124
INT関数……124
UFO出現……125
■プログラミング4「ゲーム完成」……126
SCRN$関数……126
DCGコントロール命令……126
OPNコントロール命令……126
フィニッシュ……126

第6章 マシン語プログラミング入門

／高橋雄一

■MON↵で広がるマシン語の世界……130
■16進数と2進数……132
■CPUの動きはリセットから……135
■基本はレジスタ……136
■レジスタの種類，いろいろ……137
■単独で動くプログラムを組んでみる……149
■マシン語の意味，解明！……151
■マシン語とアセンブリ言語の不思議な関係……152
■ハンドアセンブルにチャレンジ！……154
■ディスアセンブラ……156
■ディスアセンブラの使い方……161
■IOCSサブルーチンを利用したプログラムを組む……166
■フラグの働きをマスターしよう！……168
■まとめ……172

第7章 ハードウエア内部解析

／高橋雄一

■ハードウエア解析表の見方……173
■I/Oポート……180
ポートアドレス(A0H～FFH)
拡張ポートアドレス
入出力方向
内 容
■テキストV-RAM1……206
■アトリビュートV-RAM……207
■テキストV-RAM2……208



CONTENTS

第8章 ソフトウエア内部解析

／高橋雄一

■中間コード……209
■数値(浮動小数点)の内部表現……218
　単精度実数……218
　倍精度実数……219
　変数の内部表現補足……220
■メモリの使われ方……221
■RST 18HとRST 28Hによる……223
　ファンクションコールについて
■カーネル部主要サブルーチン・ワークエリア……241
■BASIC-M25内部解析……243
■ディスクの内部形式……249
　3.5"2DDディスクの仕様……251
　ファイルフォーマット……252
　レコード0000H(IPL用データ)……253
　レコード0001H～0003H(SysLoader用データ)……254
　000FH(ビットマップ)……255
　0010H～0017H(ディレクトリ)……257
　0018H～0??? 1(00EFH)(SysLoader等)……259

第9章 ゲーム&グラフィックプログラム集

／多部田俊雄

9-1 グラフィックエディタ「RERU」 261

■RERUとは……262
■グラフィックモード……263
■使用方法……263
■エディタ……263
■データメイカ……266
■フィニッシュ……267
■マシン語について……268
■VS・テキスト画面……270

9-2 PCGデータメイカ「MIKA」 270

■使用方法……271
■プライオリティ……272

9-3 やまたのおろち 275

■ストーリー……275
■ルール……278
■スコア……279
■1面クリア&ゲームオーバー……279
■プログラム……279

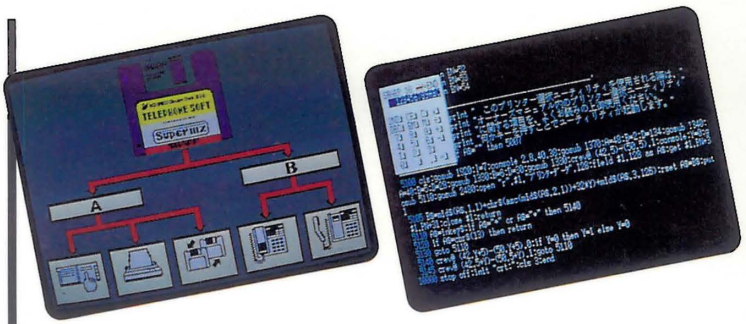
9-4 サルカニウォーズ 282

■ストーリー……282
■ルール……284
■スコア……285
■1面クリア&ゲームオーバー……285
■プログラム……286

9-5 金門橋エイリアン 288

■ストーリー……288
■ルール……290
■スコア……291
■1面クリア&ゲームオーバー……292
■プログラム……292

表紙模型協力：東急ハンズ渋谷店B5 ㈱三ツ尾



付録

Ⅰ スーパーMZ全回路図……295

①CPU・メモリコントロール部……296
②IPL・BIOS・拡張ROM・メインRAM部……297
③グラフィック部……298
④テキストV-RAM・キャラクタジェネレータ部……299
⑤CRTコントロール部……300
⑥I/O・割込みコントロール部……301
⑦RS-232C・マウス・キーボード制御部……302
⑧カセット及び音声合成インターフェース・サウンド部……303
⑨アナログ・デジタルディスプレイ切換え部……304
⑩カレンダクロック・リモート・各種スロット部……305
⑪キーボード・マウス端子(前面部)……306
⑫FDC(フロッピーディスクコントローラ)部……307
⑬ジョイスティックインターフェース部……308
⑭拡張RAMボード(MZ-1R26)……309
⑮拡張ROMボード(MZ-1R28)……310
⑯パレットボード(MZ-1M10)パレットレジスタ部……311
⑰パレットボード(MZ-1M10)D/A変換部……312
⑱拡張グラフィックRAMボード(MZ-1R27)……313
⑲カセットデッキワンチップマイコン部……314
⑳カセットデッキアンプ部……315
㉑電源部……316
㉒ボイスボード(MZ-1M08)……317
㉓キーボード部……318

Ⅱ スーパーMZマシン語活用表……319

BASIC-M25の隠し命令

／高橋雄一

●CLR……70
●FRE……70
●KEY LLIST……70
●SWAP ファイル名……70
●CGPAT$(　　)=……70
●PRINT[　　]……70
●ファイルディスクリプタの"EMM:","ROM:","USR:"……177
●LIST #n……206
●SEARCH #n……206
●FILES #n……206
●KEY LIST #n……206
●KLIST #n……206
●$……206
●PRIT文等での"～"中にコントロールシンボルキャラクタが使える……260
●KILL……260
●IPLの隠し機能……260
●倍速スムーススクロール……294
●ラインスクロールモードで起動する……294

EXPRESSION OF SENSIBILITY & COMMUNICATION

Super mz 活用研究

ハード&ソフトマスター編



ゲームプログラムから内部解析までスーパー情報満載!!

高橋雄一/多部田俊雄 共著



第1章 MZ-80Kからスーパー MZ誕生まで

シャープパソコンのルーツ「MZ-80K」



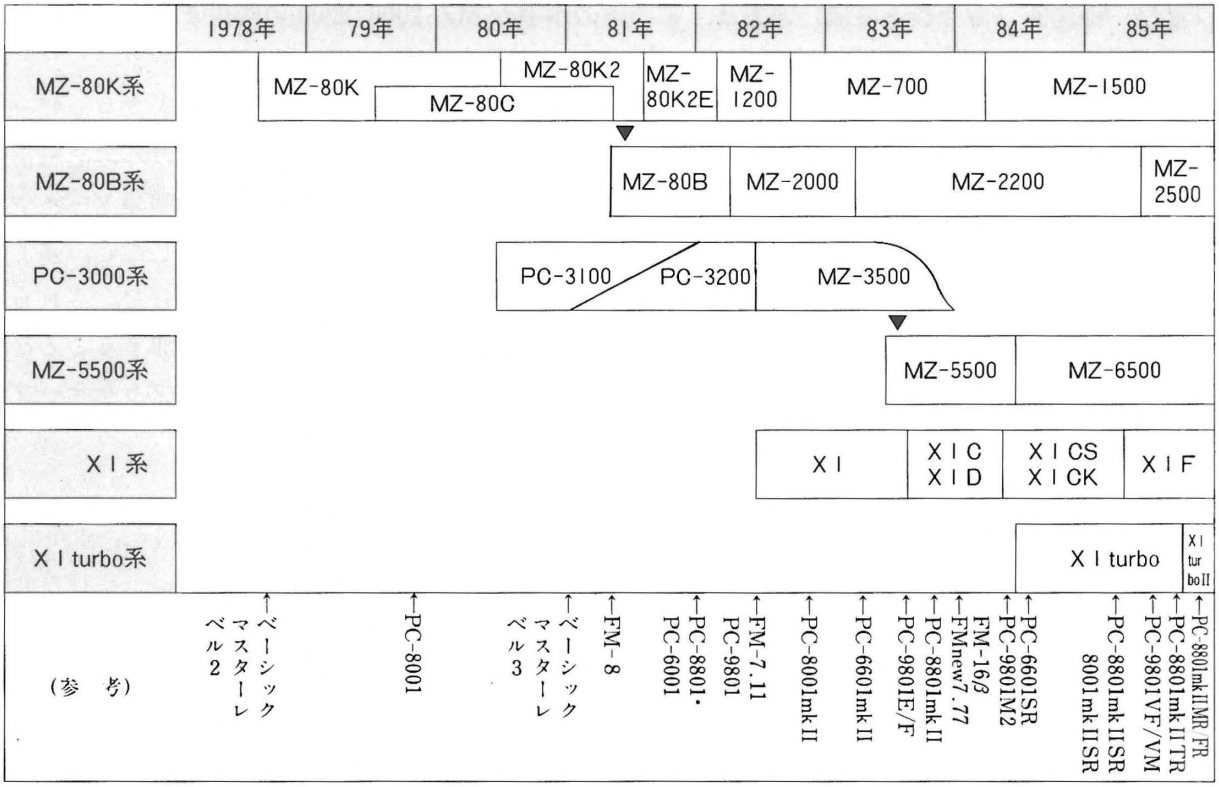
▲セミキットで登場したMZ-80K（￥198,000）

シャープでは，1978年（昭和53年）以来，一貫した設計思想のもとに，安易な妥協を許さない，独自路線の「MZ」シリーズを世に送り出して来ましたが，その原点となったのが，この**「MZ-80K」**でした。

当時，米国ではアップル社の「APPLE II」，コモドール社の「PET-2001」，タンディラジオシャック社の「TRS-80」の3大パソコンが普及していましたが，国内では日立のMB-6880（ベーシックマスターレベル2）が同年の前半に発表されていただけですので，MZ シリーズは日本のパーソナルコンピュータとしては，2番目に長い歴史を持つことになります。ちなみにNECのPC-8001は1979年，富士通のFM-8は1981年の発表です。

MZ-80Kは，**セミキット**という形を採り，ハンダ付けされたCPUボードと，電源などがセット販売され，ユーザーはこれを組み立てて完成させました。つまり，作る楽しみを味わうことができるパソコンだったわけです。このマシンは本体部の他に，キーボード，モノクロディスプレイ，そしてプログラムやデータを保存するためのカセットデッキを同一キャビネットに収めた，**オールインワン（一体型）スタイル**でした。

CPUには，**「Z80」**をいち早く採用し，注目を集めました。「Z80」や，その高速バージョンの「Z



《第1-1図》シャープパソコンの流れ

80A」,「Z80B」は，現在ではパソコン用としては最も普及している８ビットCPUですし，スーパーMZでも採用している素晴しいマイクロプロセッサですが，当時としてはパソコンの前身となったワンボードマイコンまで含めても，国産のマシンではこれを採用したものはありませんでした。逆に考えると，MZ-80Kは大変先見の明のあったパソコンだったと言うことができるでしょう。

メモリは，ROM４Kバイト，RAM20Kバイト（＋V-RAM１Kバイト）を標準装備し，**RAMは最大48Kバイト**まで拡張できました。

当時は，まだバンク切換えは行われていませんでしたので，８ビットCPUのメモリ空間，64Kバイトのうちの$\frac{3}{4}$ものエリアをRAMとしてユーザーに開放していることになります。４KバイトのROMには，キーボードからの入力や画面への出力（表示）など，どんなソフトにも必要不可欠なサブルーチンが収められていました。

表示能力は，文字は**40桁×25行**と，米国製のAPPLE IIやPET-2001とほぼ同じでした。グラフィックは，その後発売になったNECのPC-8001でも採用された文字画面を流用する方式で，横80×縦50ドットで，カラーはできませんでした。CPUのクロックは**２MHz**でしたが，**WAIT***や**DMA***などの邪魔が入らないため，そのままのスピードで動きました。また，これを４MHzにしてすべての速度にする，**倍速ボード***も数種類開発され，販売されていました。

BASICは，スタイルも良く似たPET-2001のものに良く似たものがシャープ自身によって開発され，採用されました。今ではBASICの自社開発は珍しくなってしまいましたが，当時としては当たり前でした。逆に言うと，数多くの「自社製BASIC」が処理速度の競争に敗れ去った中，SHARP-BASICだけは生き残ったと考えても良いでしょう。

ただ，機能面（命令数その他）の方は，その頃の水準がMZ-2000/2200の時代まで続いていた部分が多く，不満も聞かれました。

ところで，MZ-80Kでは，BASICなどの言語は，外部記憶装置（カセットテープやフロッピーディスク）からRAM上に読込んで使うようになっていました。そのため，BASIC以外の高級言語もそれからロードして使うことができました。シャープからは本格的なマシン語開発ツールや，BASICを自社開発した技術をもとに，実数型のパスカルインタプリタが発表されましたし，多くのソフトハウスからFORTRAN系の言語や，整数型のコンパイラなどが発売されました。これらの言語ソフト等はほとんどがカセットテープベースで動きましたので，ハードを追加することなく，様々な言語を利用することができました。4KバイトのROMには，これらのソフトをテープからロードして実行する機能IPL* も含まれていました。

WAIT（ウェイト）

RAMやROMなどのメモリの応答速度が遅く，CPUに追いついていけない場合，CPUを一時的に待たせること。スーパーMZでもクロック6MHzの時のみ，M_1サイクル（命令を読取る時）にWAITを一つだけかけている。クロック4MHzの時にはWAITはかからない。

M_1サイクルに1WAITかけると，4クロックの命令は5クロック，19クロックの命令は20クロックになり，スピードが5～20%低下する。

昔のパソコン等には，クロック4MHzでも追いつかない安価なメモリを使ったものもあり，M_1サイクル以外に，リードサイクル（メモリから何か読出す時）やライトサイクル（書込む時）にまでWAITをかけるものもあった。また，M_1サイクルにWAITを二つもかけるものもあった。

DMA（ディーエムエー）

Direct（ダイレクト） Memory（メモリ） Access（アクセス）。CPUを介さずに，メモリと周辺デバイスとのデータ転送を行うこと。

一般に画面表示等では良く用いられる手法だが，ここでは，CPUを一時的に停止させて転送を行う場合を指す。例えばCPUも周辺装置もテキストV-RAMの内容を読出したい，といった場合にCPUを停止させる（この場合はWAITを使うことが多い）のは，止むを得ないとしても，昔のパソコンや現在発売されているパソコンの一部には，今だにCPUはテキストV-RAMのデータを必要としないのに，DMAによって常に，断続的な停止状態に追い込まれているものがある。このDMAは画面が出ている間じゅう（一般に電源を入れてから電源を切るまでの間ずっと）行われることになるので，当然CPUの速度は遅くなる。

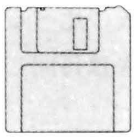
中にはWAITとDMAの両方共かかっているパソコンもあり，正規のクロックの$\frac{1}{3}$以下のスピードでしか動作しない，というものもあった。ただし，現在は$\frac{1}{3}$というほどひどいものはなくなっている。

倍速ボード

WAITやDMA等の条件が変わらず，CPUの種類が同じであれば，CPUのクロックを上げると，実行速度は正比例して上がる。つまりクロックが2MHzから4MHzに上がると，実行速度は2倍になるというもの。ただし，クロックを上げるとメモリが追い付かなくなることがあるため，もともと4MHzのものを8MHzにする等はできない場合がほとんどである。

IPL(アイピーエル)

Initial(イニシャル) Program(プログラム) Loader(ローダ)。ディスクやカセットから目的のプログラムを読込むためのプログラム。MZ-80K～1200は4KバイトのROMに，他のサブルーチンと共に入っているが，MZ-80B・2000ではIPL部だけでROMとなっている。



MZ-80Kのマイナーチェンジ機「MZ-80C」～「MZ-1200」



▲MZ-80C→MZ-1200は，￥268,000→￥148,000と価格が徐々に下がっていった

MZ-80Kに続き，RAM等のコストが下がったことを反映させて価格を下げたり，キーボードやブラウン管を高級化したり，RAMの標準装備容量を増やし，コストパフォーマンスを向上させたコンパチブルモデル，**MZ-80C，80K2，80K2E，MZ-1200**が次々と発表されてきました（このころになると，ハードウエアまで手がける人が減ったので，セミキットでなく完成品になっている）。

この後のシリーズでも見られることですが，シャープのパソコンの場合，ハードウエアの機能を少しだけ追加・変更して，新製品として発売する，と言ったことは原則として行わず，それまでオプションだったものを標準装備するとか，デザインを手直しするとか，キーボードを上質のものにするとか，価格を実売価格に近づけて再設定するとか，ディスプレイを**グリーンディスプレイ***にするとか，いずれも機能的には変わらない様なものをマイナーチェンジ機として発表していきました。そのため，MZ-80Kのユーザーも，MZ-80Cが出たからと言って，特に買い替える必要はありませんでした。ソフトももちろん，共通に使えました。

しかし，ハードウエアはそのままでも，パソコンの性質はソフト仕第でいくらでも変わります。特にMZシリーズはPASCALやFORTRAN系の言語を走らせることができるのと同じように，全く新たなBASICを使うことも可能でした。1981年早春に発表された**HuBASIC***は，当時の各社のBASICと比べても最高水準の性能とスピードを誇っていました。

また，フロッピーディスクを本格的に利用できる開発ツール，**H-DOS***（ハドソン），**F-DOS***（シャープ），**BASICコンパイラ**など，今，考えても注目に値するようなソフトが数多く発表されました。

そういったソフトがなぜ，実現したかと言うと，前にも触れたとおり，8ビットCPUの全メモリ空間64Kバイトの大半（48Kバイト）を，目的を自由に決められるRAMエリアとして，ユーザー・ソフトハウスに開放しているからです。もし，BASICなど，特定の言語をROMに固定していたなら，その言語を多少強化するようなソフトならできても，全く別の言語やシステムの開発は困難を極めることでしょう。なぜなら，それらのソフトを配置するためのRAMは，残されたほんのわずかなエリアにしか割り当てられなくなってしまうからです。

シャープでは，MZのように，IPLをROMとして持つだけで，残りのメモリの大半のエリアをRAMとしたシステムを**「クリーンコンピュータ」**と呼んでいますが，これはMZが最初というわけではなく，ミニコンや大型コンピュータでは至極当然のことになっています。ただ，それらのコンピュータとは違い，MZ-80シリーズの場合，フロッピーディスクを持っている人は少なかったため，カセットテープからBASIC等をロードするのが普通でした。せいぜい1～2分しかかからないのですが，BASICしか使わない人にとってはスイッチONでBASICが動く他社機に比べると，使いづらいと言うこともあったようです。なお，余談ですが，こうしてカセットデッキは毎回必ず使うことになりますので，ボーレイトは**1200ボー**と当時の最高速でしたし，信頼性も抜群でした。

グリーンディスプレイ

モノクロディスプレイの一種。普通のモノクロディスプレイの表示色は，昔の白黒テレビのような水色やカラーディスプレイで表示した白のような色であるが，グリーンディスプレイは目にやさしい，濃い緑色で表示される（ブラウン管は割高である）。他に16ビット機用の一部には，イエローディスプレイや，オレンジディスプレイというのもある。いずれも，その蛍光体の色と，光の出ない「黒」で表示される「モノクローム」なディスプレイである。

HuBASIC（ヒューベーシック）

ハドソンソフトが開発したBASIC。カセットテープベースでは初めて倍精度関数が使え，また，BASIC言語としては初めてラベルの使用が可能となった。

現在のX1・X1 turboのBASICは，これを拡張したものであるほか，MZ-700・1500，そしてスーパーMZのBASICにも，多大な影響を与えている。

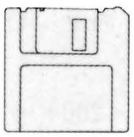
H-DOS(エッチドス)

ハドソンソフトが開発したDOS(Disk Operating System)。シャープの Disk BASICとファイルの互換性があり（注．F-DOSは同じシャープ製ながら互換性はない），プログラムやデータの転送用といった簡単な応用から，リロケータブルなオブジェクトコードを出力するZ80アセンブラを使ったマシン語開発用に使えた。発売はHuBASICやF-DOSよりずっと前で，一説によるとHuBASICはH-DOSで開発されたと言われる。

F-DOS(エフドス)

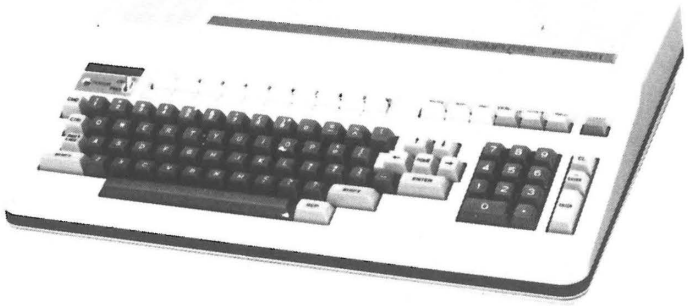
シャープが開発したDOS。リロケータブルなオブジェクトコードを出力するZ80アセンブラを含む。オプションでBASICコンパイラが使えた。

MZ-80B・2000用も発売され，それらでは，BASICコンパイラは標準装備となり，アセンブラはマクロアセンブラとなった。その後，MZ-700のS-BASICと融合し，同機のDisk BASIC，MZ-1500のBASICのOS部を構成するに至る。スーパーMZのIOCS部には，さらにそれを拡張したものが含まれている。



えっ！シャープのPCシリーズ

MZ-80C～MZ-80K 2が発売されていた頃，同じシャープから，全く系統の異なるパソコンが発表されました。名前はMZではなく，**PC-3100**と言いました。

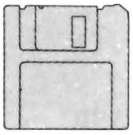


▲シャープのPCシリーズPC-3100（￥250,000）

シャープでは今から10年位前からPCシリーズというものがあり，最初はPC-1200等のProgrammable Calculator（プログラム電卓），次にPC-1500等のPocket Computer（ポケコン），そしてPC-3100のPersonal Computer（パソコン）と続いているわけです。その後，PC-5000等のPortable Computer（ポータブルコンピュータ）も発売になっています。

PC-3100はそういった背景があるため，電卓やポケコンの流れを汲んだパソコンになっていました。例えば[2][*][3][ENTER]とすると6と答えが返って来ましたし，パソコンとしては初めて10進演算（BCD演算）を採用し，また，関数電卓だけにある関数（例えばASN…$\sin^{-1}$を求める）を持っていたりしました。また，あまり効用はなかったようですが，キーボード用に初めて**サブCPU**を使用した，**デュアルCPU構造**を採っていたことも印象的でした。

PC-3100は，すぐ後に発表になった上位機，**PC-3200**と共に，BASIC（自社開発であるが，MZ系とは全く異なる）をROMとして持っていました。また，デザインもオールインワンタイプではなく，ディスプレイが分離し，カセットデッキはオプションとなっていました。



64Kバイト！オールRAM「MZ-80B」

MZシリーズでは，PC-3200とほぼ同時期，1981年の4月に，大幅にバージョンアップを行ったMZ-80Bを発表しました（前出のMZ-80K 2EとMZ-1200は，このMZ-80Bより後の発売です）。

MZ-80BはCPUのクロックを**4MHz**にし，究極のクリーンコンピュータとして**メモリ空間64Kバイトをすべて普通のRAMで埋めつくし，文字画面を80桁×25行**に，グラフィックを**320×200ドット**（オプション）に，カセットデッキをパソコンのCPUから制御可能な**フルロジカルコントロールタイプ**に変えたものという感じでした。テキストV-RAMやグラフィックRAM等は，バンク切換えの手法を取り入れ，別バンクに持たせていましたから，64KバイトのRAMは，完全に自由に利用することができました。

ただ，当時の他メーカー機同様，このような場合の互換性についての配慮はなされておらず，MZ-80K等のソフトは使えませんでした。それでも，BASICで書いたプログラムだけは利用できるよう，**テキストコンバータ***というソフトが初めて開発され，ほぼ同時に発売になりました。BASIC自社開発の特典は，こんなところにも活かされています。

MZ-80B用にも，ずい分多くのソフトが発表されました。ただ，はじめのうちは，MZ-80K用のものを単に移植したような感じのものが多く，グラフィックを活かしたようなものは多くなかったように思います。また，H-DOSがサポートされなかった代わり，**CP/M　V2.2***は簡単に利用できるようになりました（MZ-80K用はハードウエアの追加が必要だった）。

周辺装置については，結構いろいろなものが出て来ました。例えば，MZ-80Bは320×200ドット×モノクロを2ページ持つことができましたが，I/Oデータ機器からは，2ページ分の価格でカラーグラフィックが楽しめる，グラフィックボードが発売されました。同時には4色表示でしたが，パソコンとしては初めてパレット機能を備え，8色中から4色を選ぶことができました。また，前出のHuBASICは，積極的にこれをサポートし，パソコンのBASICとしては初めてユーザー座標系を導入したバージョン（**HuGBASIC**）を発表しています。



▶究極のクリーンコンピュータMZ-80B（￥278,000）

テキストコンバータ

BASICのプログラムは，メモリ上では中間コードと呼ばれる状態に圧縮されて格納されている。ディスクやテープにロード／セーブする際にも通常このままの状態でいるのだが，中間コードはBASICの種類が違っていたり，構造を変えているような場合には，互換性がなくなる。テキストコンバータは，中間コード等を，目的のBASIC用に変換するものである。

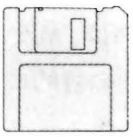
ただし，BASICの文法やグラフィックのドット数その他が違っているような場合には，コンバート後その部分の手直し作業が必要になるなど，元々完璧なソフトウエアではない。

CP/M(シーピーエム)

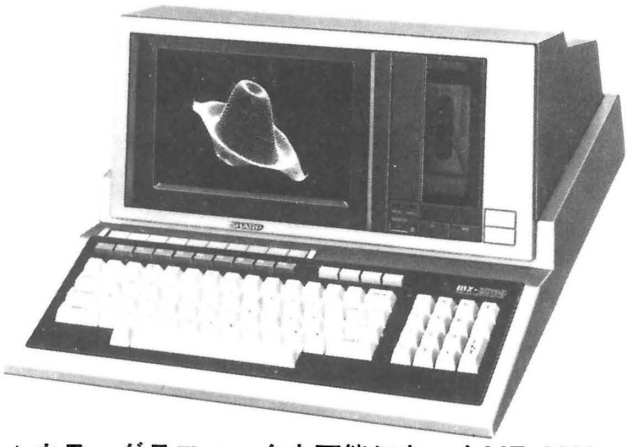
米国デジタルリサーチ社の開発したDOS。MZ-80B/2000用は国内代理店のMSA(マイクロソフトウエアアソシエイツ社）より販売された。

DOS本体には古い時代の使いづらさが見られ，単体ではリロケータブルでない8080のアセンブラしか使えないが，一般の８ビット機用のDOSとしてはシェアが極めて高いため，この上では多くの言語ソフトや開発ツールが動作する。

なお，スーパーMZでは，CP/M　V2.2の使いづらさを改善したP-CP/M(パーソナルCP/M）がオプションで発売される。



640×200ドット，カラー対応「MZ-2000」



▲カラーグラフィックも可能になったMZ-2000

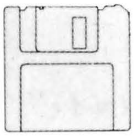
MZ-80Bから１年後，MZシリーズとPC-3000シリーズ（3100と3200）の統合が決定されました。これはシャープ社内の問題でしかないのですが，両事業部が合併したことによるものです。これ以後，新事業部の開発するパソコンは，すべてMZ-××××（３～４桁の数字）という名称になっています。

そして，合併記念というわけではないと思いますが，ちょうどそのころ，**MZ-1200**と**MZ-2000**とが発表されました。MZ-1200の方は前出のとおり，MZ-80K～K２Eとコンパチブル（価格が安い）ですが，MZ-2000の方は，MZ-80Bと良く似たパソコンでした。機能的にはグラフィックが**640×200ドット**になり（ただしオプションのまま），**純正品としては初めてカラーが使えるようになりました。**しかも，**各点毎に８色**と，それまで遅れを取っていた感の強いカラーグラフィックが，遜色なく利用できるようになりまし

た。ただ，実際にはグリーンディスプレイが本体に付いていたため，グラフィックRAMはモノクロ1ページ分(カラーには3ページ分必要)だけを購入して使っていた人が多かったようです。

MZ-2000のハードの追加・変更はグラフィックだけだったのですが，カセットデッキ，グラフィックRAM，キーボードのランプ等，細かな部分の変更が結構あったため，IPLから起動させるソフトは使用できませんでした。BASICで書いたプログラムの方は，コンバータ等を介さず，そのまま動くものが多かったようですが，マシン語サブルーチンでグラフィックを操作するようなものは，動作しませんでした。

なお，MZ-2000は，MZシリーズとしては(純正としては)初めて外部に**RGBディスプレイ**を接続できるようになりました。



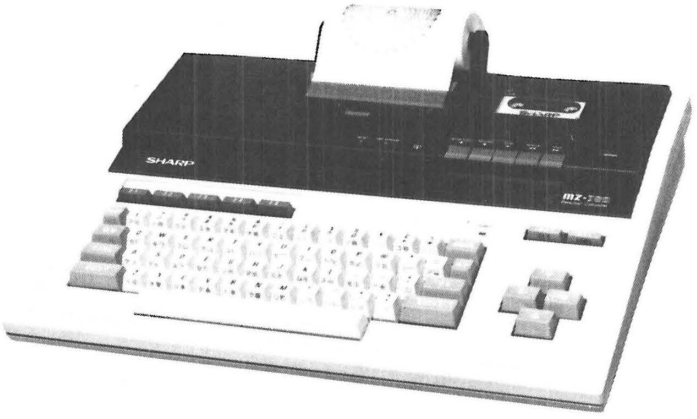
家庭用テレビが使える「MZ-700」

MZ-1200の半年後，つまり，1982年の秋には，**MZ-700**が発表されました。MZ-700は，MZ-80K～1200とかなり充分な互換性を持ちつつ，MZ-80B的な拡張を一部施されています。クロックが**3.579545MHz**（カラーバースト信号と同じ）になり，64Kすべてを**RAM**にできるようになったのです。また，オールインワンスタイルをやめ，ディスプレイを分離しました。これは，文字にカラーをつけられるようになったためです。

MZ-700では，MZ-2000同様のRGBディスプレイの他，**ビデオ信号（NTSC信号*），RF信号（テレビの1チャンネルないし2チャンネル）**も出力できましたので，**家庭用のテレビにつないで利用することもできました。**

このため，MZ-700ではカラー版のソフトがずい分多く発表されました。グラフィックはMZ-80Kの頃のドット数ですが，4ドット単位で色が付けられるようになったのと，元々その目的に作られたキャラクタが多くあったので，ゲームも結構楽しいものが出ました。ジョイスティックやプロッタプリンタに対応している点も見逃せませんでした。

BASICは，MZ-80K～1200シリーズと上位コンパチブルな**S-BASIC***と，カラー関係の命令等を追加し，64KRAMモードで動くようにした新バージョンの**HuBASIC**の合計2本が添付されていました。



▶カセットデッキとプロッタプリンタの組合わせで，MZ-711（￥79,800），MZ-721（￥89,800），MZ-731（￥128,000）の3モデル

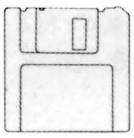
NTSC信号（エヌティエスシーしんごう）

日本や米国で採用されている家庭用カラーテレビの伝送形式。画面を縦方向525分割し，横に細長い1本を約63.5μ秒（$\frac{1}{15750}$秒）で左から右に走査し，それを上から下に1本置きに繰り返し（2回で完全な1枚の絵が送れる），色信号はカラーバースト信号に乗せて送るという規格だが，これに類似した信号なら問題なく映し出すことができる。

NTSC信号は，とても簡単な回路でRF信号に変換することができる。

S-BASIC（エスベーシック）

元来，MZ-80KのBASICがPET-2001（マイクロソフト社製BASICを搭載している）とほぼ同じ仕様だったので，昔からMZシリーズのBASICはマイクロソフト系であった筈なのだが，その後両者とも独自の拡張を続けた結果，この頃には文法等の異なる部分が多くを占めるようになってしまっていた。また，他社のパソコンのほとんどがマイクロソフトBASICを採用し互換性の問題が生じていた。そこで，マイクロソフトBASICに準じた仕様のHuBASICを標準装備して，この問題を乗り越えようとしたが，従来のものと互換性が取れなくなると困るので，それまでのMZのBASICの流れを汲むBASICも合わせて標準装備した。S-BASICというのはその時にHuBASICと区別するために付けられた名前である。なお，S-BASICは，実際にはHuBASICの機能の$\frac{1}{4}$くらいをカットし，文法解析部を一部手直したような構造になっている。



世界初「パソコンテレビX1」

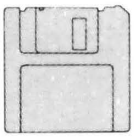


▲パソコンとテレビがドッキングしたX1（￥155,000。X1D～X1Fは￥198,000～￥89,800の間）

MZ-700の発表と時を同じくして，さきほどの統合した事業部とはさらに異なる事業部から，**パソコンテレビX1**という全く新しいパソコンが発表されています。X1は機能的にはMZ-2000に文字画面のカラー・拡大・反転・ブリンク，グラフィックのパレット，グラフィックと文字のプライオリティ，256個のユーザー定義文字（PCG），3重和音サウンドジェネレータ（PSG），バッテリーバックアップカレンダクロック，スーパーインポーズ等を追加したような構造を持っていましたが，MZ-80K～1200と80Bとの間のように（MZ-2000と）ハードウエアの互換性はありませんでした。ただ，やはりBASICレベルでは，テキストコンバータやシステムソフトウエアコンバータと言ったソフトが発表されたこともあり，互換性を持つこともできました。BASICは，MZ-700のそれより（グラフィック等の）

命令が強化されたHuBASICのみが添付されました。

X1は，その後，MZ-80Kと同様にマイナーチェンジを繰り返し，**X1D，X1C，X1Cs，X1Ck，X1F**が発表されています。ソフトもかなり充実しており，シャープパーソナルコンピュータシリーズにとっても大変重要な位置を占めています。



オールインワンタイプからコンポタイプへ

1982年の秋には，**MZ-3500**というパソコンも発表されています（ちなみに，この頃は**NECのPC-9801，富士通のFM-7/11，ソニーSMC-70，カシオ計算機のFP-1000/1100，松下通信工業のJR-200**などと新製品ラッシュが続きました）。

MZ-3500は一見全く新しいパソコンのようなネーミングですが，実は，以前の PC-3100/3200をベースに，大幅な拡張を行ったものでした。

Z80Aを2個搭載し，サブ側にはグラフィック操作をまかせると言った，富士通のFMシリーズに似た配分になっており，グラフィックは**640×400ドット×フルカラー（8色）**がオプションで扱えました。

あまり注目されることのなかったマシンなので，相当詳しい人でも本機の存在を知らないことも多いかも知れませんが，漢字V-RAMを搭載するなど，技術的に考えるとスーパーMZのある意味での前身となっているような部分もあります。



▲5″ミニフロッピーディスク内蔵のMZ-3500（ディスク1基/2基：￥320,000/￥410,000の2モデル）



▲システムアップが自由なコンポタイプのMZ-2200（￥128,000）

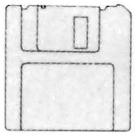
BASICはPC-3100／3200系のものが載りましたが，MZというネーミングをした以上，クリーンコンピュータ構造をとっていました。

また，MZシリーズとしては初めて**フロッピーディスクを1～2基装備**していました。このドライブは2Dタイプのもので，容量は1基あたり**320Kバイト**でした。

続いて，1983年の春には，MZ-2000を分割・再編集したような**MZ-2200**が発表されました。これは，MZ-2000のオールインワンスタイルをやめ，ディスプレイとカセットデッキをオプションにし，それまでオプションだったグ

ラフィックRAM等を標準装備にしたものです。ソフト的にもハード的にも完全に互換性があります。

このMZ-2200をもって，シャープのパソコンはすべて**コンポスタイル**となりました。



シャープ初の16ビットパソコン登場

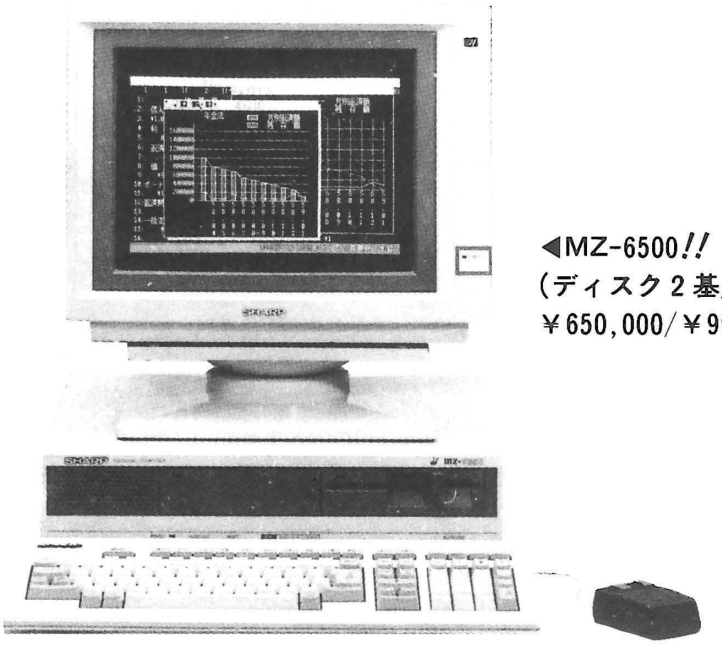
MZ-2200の登場から半年後，つまり，1983年秋には**シャープ初の16ビットパソコン　MZ-5500**が発表されました。他社に比べると，例えばNECのPC-9801に対しては約1年の遅れをとっています（NECからは，MZ-5500と同時期にPC-9801E/Fが発表されている）。

ハードウエア的にはMZ-3500を16ビット機化したような感じですが，漢字V-RAMは省略され，代わりにグラフィック画面に**スムーズスクロールやマルチウインドウ機能**が加わっています。

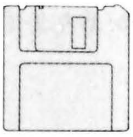
なお，さらに約1年後にはCPU(i8086)のクロックを5 MHz→8 MHzにスピードアップし，MZ-5500でオプションだったいくつかの周辺装置類やソフトを標準装備したMZ-6500も発表されています。



▶シャープの16ビットパソコン第1弾　MZ-5500
(ディスクなし/1基/2基：￥218,000/￥288,000/￥388,000の3モデル)



◀MZ-6500!!　ライバルはもちろんPC-9800シリーズ
(ディスク2基/1基＋ハードディスク1基：￥650,000/￥998,000の2モデル)



淳美のパソコン「MZ-1500」

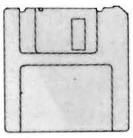
1984年の春には，**MZ-1500**が発表されました。MZ-1500はMZ-700に**QD（クイックディスク）**と，**320×200×フルカラー（8色）**のグラフィック兼1024個のRGB独立型**PCG**と，6重和音の**PSG**等を追加したようなパソコンです。

PCGはX1シリーズには既に搭載されていましたが，MZシリーズとしては初めてです。また，MZ-1500用には**音声合成カード**等も発売になるなど，さらに楽しいホームコンピュータとなりました。ソフトもクイックディスクで待たされないスピーディなものが続々と登場しました。



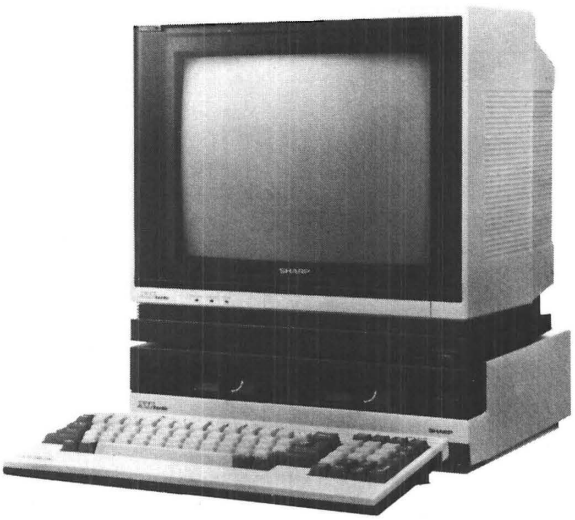
▲QD（クイックディスク）搭載のMZ-1500（￥89,800）

BASICにはMZ-700のS-BASICを大幅に強化したものを搭載しています。なお，MZ-1500は，クイックディスクサポートのため，ROMを6Kバイト増設し，合計10Kバイトとしています。

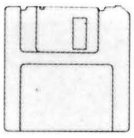


高漢度パソコン「X1 turbo」

MZ-1500の半年後，初代のX1から数えて2年後である1984年秋には，X1に漢字V-RAM，640×400×8色のグラフィック，マウス，RS-232Cポート，テロッパ（スーパーインポーズ後の映像をVTRに録画できる）を追加し，5″2HDやハードディスクに対応した，**X1 turbo**が発表になりました。また，IPL-ROMを大幅に拡張し，各種サブルーチンを加えて32Kとしています。BASICも，漢字処理等を加えています。市販ソフトも高速漢字表示をサポートしたものがあっという間に出揃いました。ホームユースからスモールビジネスまで，応用範囲のとても広いパソコンです。



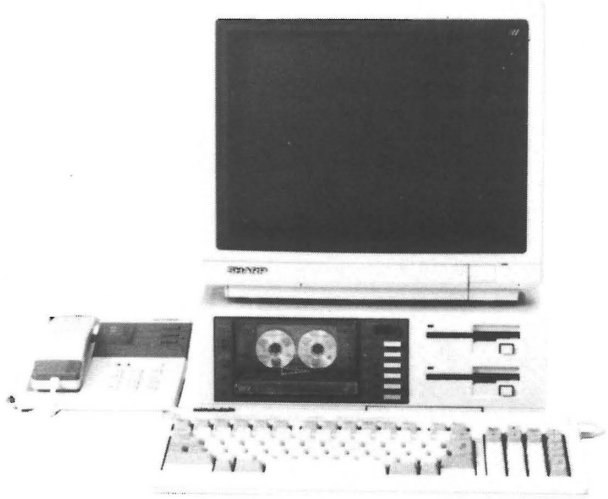
▲漢字BASIC採用のX1 turbo（ディスクなし￥168,000～ディスク2基￥278,000の4モデル）



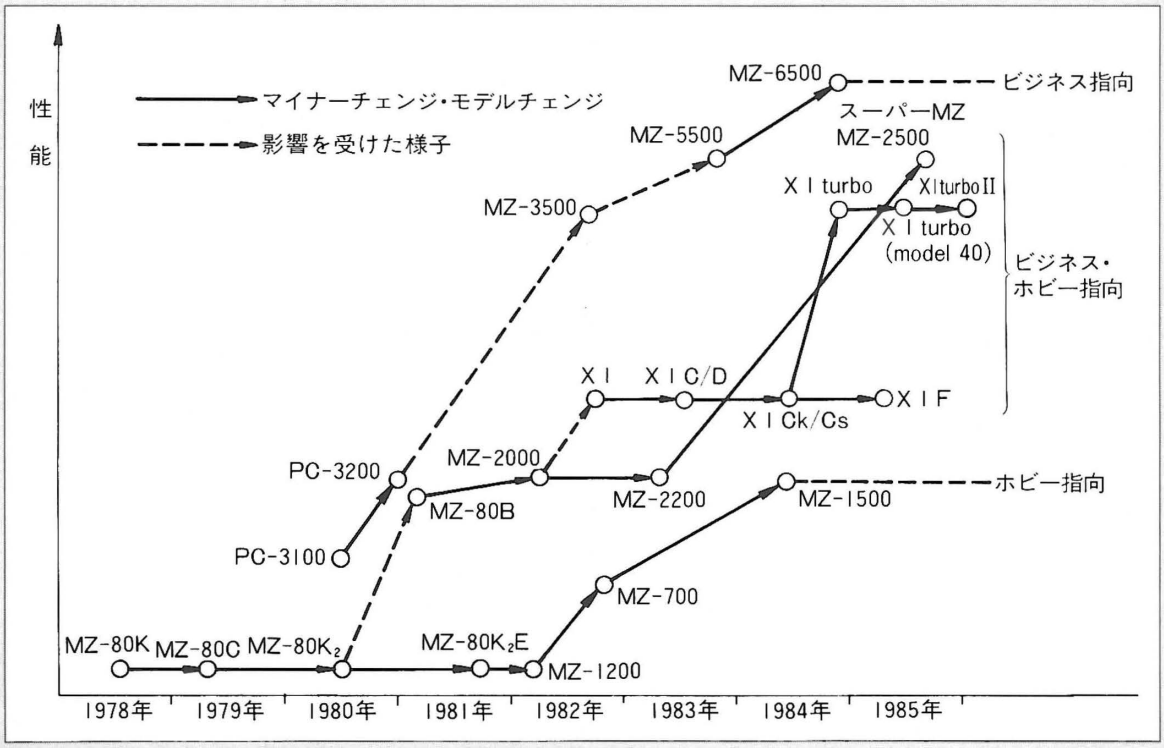
最強パソコン「スーパーMZ」誕生!!

そして，いよいよ**スーパー MZ (MZ-2500)**が1985年の秋に誕生しました。

スーパー MZ は，グラフィックについては1982年春のMZ-2000以来３年半ぶり，それ以外の部分は実に1981年春の MZ-80B以来，４年半ぶりに大幅な見直しを図った，MZ シリーズの後継機です。もちろん今まで長年に渡って蓄積して来た技術と，X1そのものやX1→X 1 turbo への拡張部分，そしてスーパー MZ ならではの新たなアイデアを盛り込んだ，集大成マシンと言うこともできます。次章ではいよいよスーパー MZ の魅力を大々的に探ってみたいと思います。



▲時代は今「スーパーMZ」!!



《第1-2図》性能を中心に見たシャープパソコンの流れ

MZシリーズに限らず，現在のパソコンは，大まかに考えて

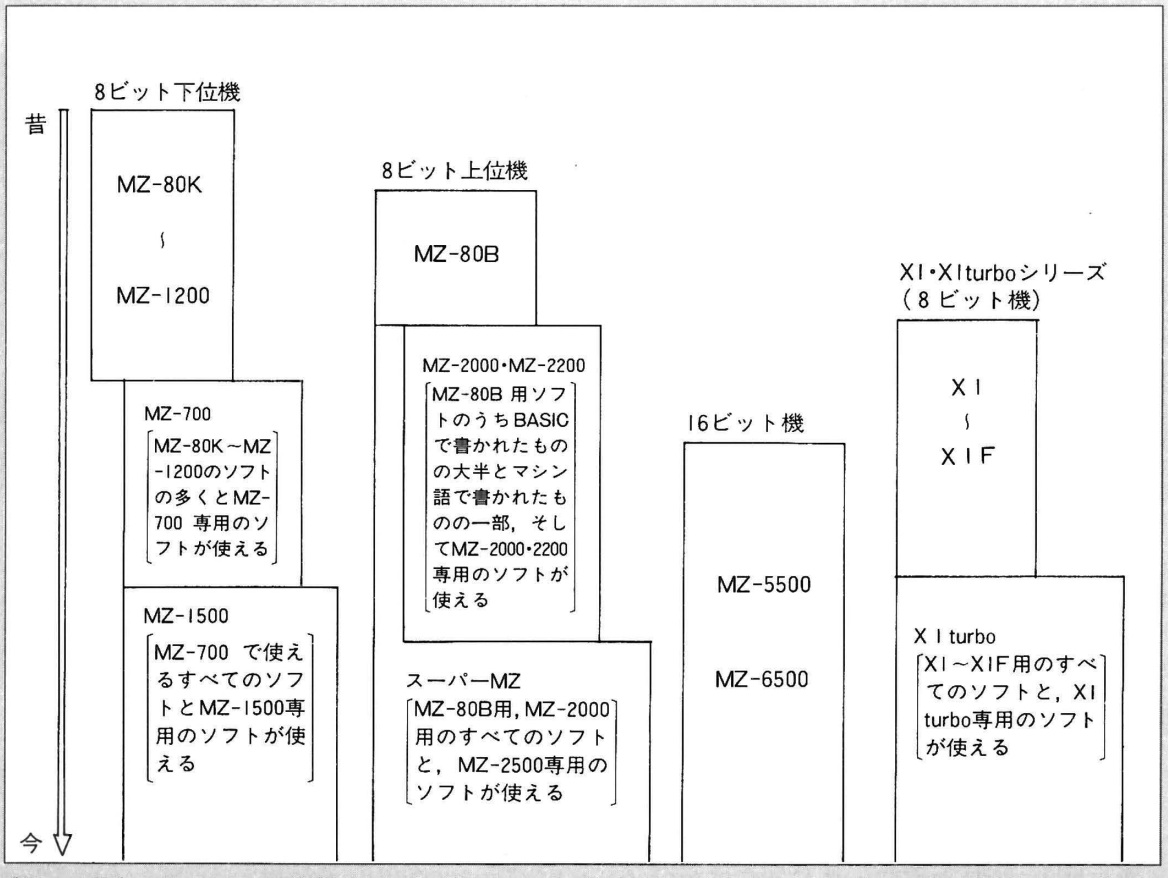
①ビジネス向け16ビット機（ディスク２基付きで約30万円以上）

②ホビー・スモールビジネス向け８ビット上位機（ディスク２基付きで15万円以上～30万円未満）

③ホビー(特にゲーム中心)向け８ビット下位機（ディスクなしで約10万円以下）

の三つに分けることができます。今回のスーパーMZは，このうち②に位置しています。この価格帯のパソコンは，家庭にあってはBASIC等の入門，ゲーム，ワープロ等の実用，その他のホビー用に，オフィスにあっては１人１台の手軽さで使えるワープロ，データベース，その他実務応用にと，幅広い活用ができるのが特長で，16ビット機にない，いくつかの機能を備えていたり，ソフトも個人で購入することを考えて，比較的低く抑えられているのが普通です。

シャープでこれまでに発売して来たパソコンも価格帯は違ってもみな①～③のような分け方ができます。**第1-2図**を見て下さい。→でつないであるのはマイナーチェンジやモデルチェンジを行ったマシンであることを示しています。この図から，スーパーMZは，従来のMZ-2000やMZ-2200の後継機であり，性能を大幅に向上させたものであることがわかります。また，詳しくは後述していますが，**第1-3図**のように，MZ-80Bの上位コンパチブルでもあります。



《第1-3図》ソフトの互換性を中心に見たシャープパソコンの流れ

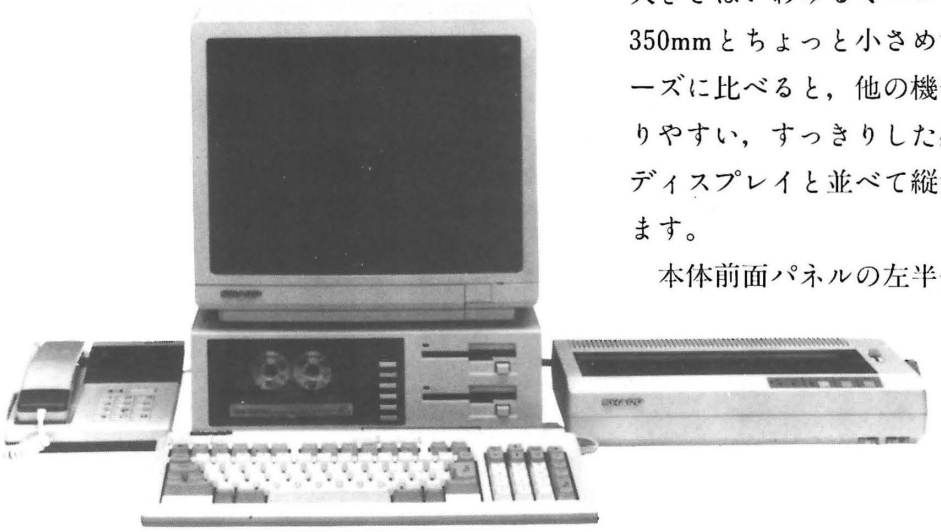


第2章 ハードウエア解剖

第2章・第3章では，既にスーパーMZをお持ちの方には本機の優れた特徴を改めて確認してもらえるよう，そしてこれからユーザーになろうとしている皆さんには他のパソコンとの違いを探る購入ガイドとして利用して頂けるよう，ハードウエア編とソフトウエア編に分けて，本機の性能を深く堀り下げてみたいと思います。

スーパーMZの外観

先進の機能を満載しているスーパーMZは，その能力からは想像し難い，スリムで斬新なデザインにつつまれており，一見オーディオカセットデッキを思わせます。色は白に近いクリーム色，大きさはいわゆるミニコンポサイズで，横幅は350mmとちょっと小さめです。以前のMZシリーズに比べると，他の機器や家具との調和がとりやすい，すっきりした感じがします。また，ディスプレイと並べて縦置きで使うこともできます。



《写真2-1》スーパーMZ(MZ-2500)フルシステム

本体前面パネルの左半分にはフルロジカルコントロールのカセットデッキが，右半分には3.5インチマイクロフロッピーディスクが，縦に2基並んでいます。

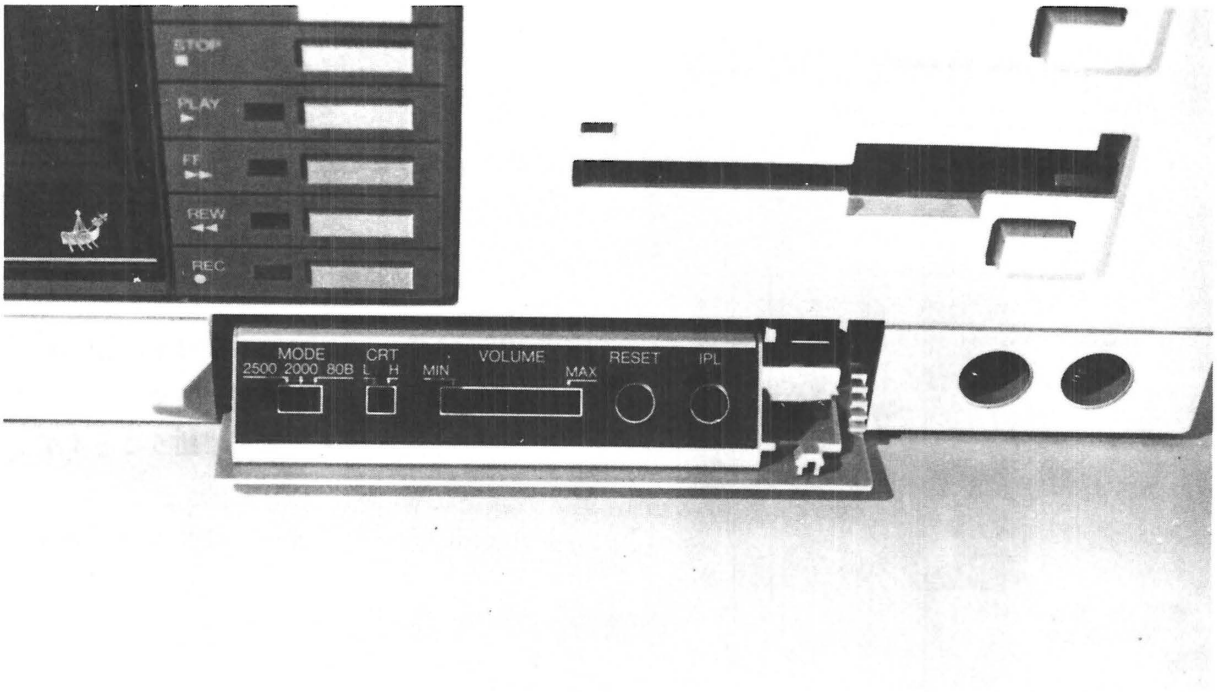
ところで，スーパーＭＺでは，他の多くのパソコンと同じように，複数のモデルが存在しており，ディスク２基を搭載しているのがmodel30(正式型番「MZ-2521」￥198,000)，１基のタイプがmodel20(正式型番「MZ-2511」￥168,000)です。**写真2-1**はmodel30というわけです。この点を除けば他の所は共通ですので，model20でもオプションのディスクドライブ(「MZ-1F21」￥35,000)を追加購入することで，model30と全く同じにすることができます。ただ，スーパーＭＺにはディスクドライブなしのmodel10は存在しません。これは，スーパーＭＺの優れた機能をフルに生かすには，ディスクが必要不可欠だからでしょう。

話を戻しましょう。スーパーＭＺ正面のカセットデッキ操作スイッチの下部にカンガルーポケットがあり，ここを開けると使用頻度の高い，いくつかのスイッチが出て来ます(**写真2-2**)。

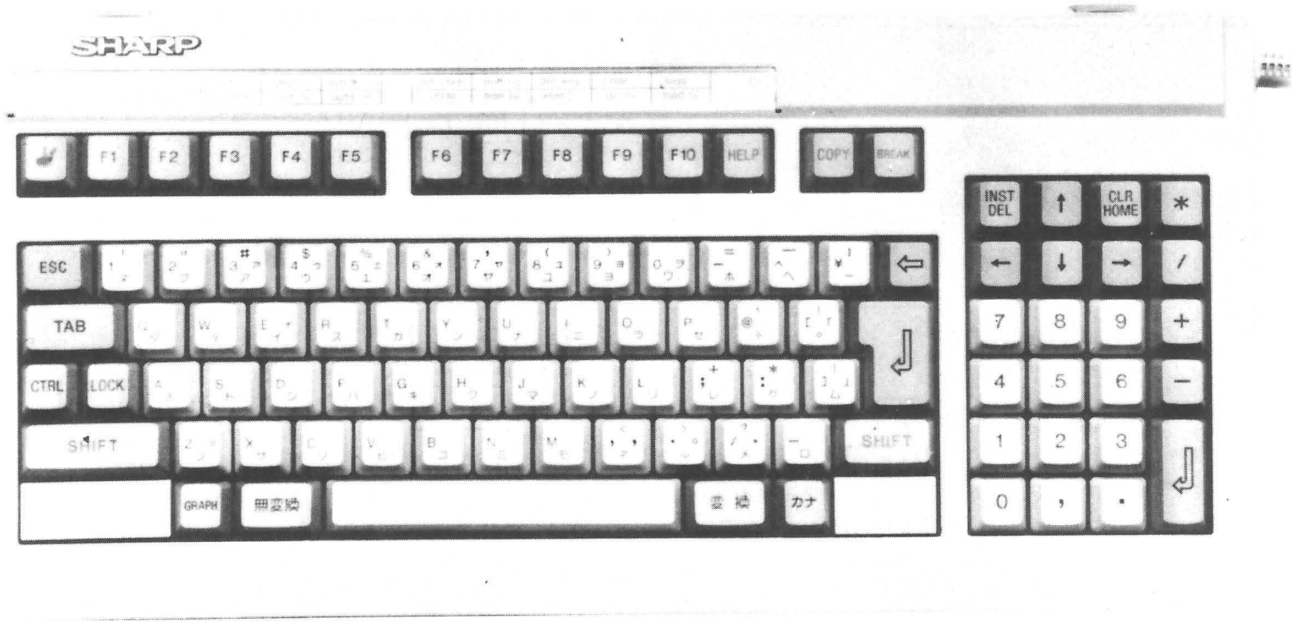
○モード切換えスイッチ(スーパーＭＺモード，MZ-2000モード，MZ-80Bモード)
○ディスプレイ切換えスイッチ(400ライン系，200ライン系)
○スライドボリューム(サウンド音量調節)
○IPL起動スイッチ
○リセットスイッチ

です。

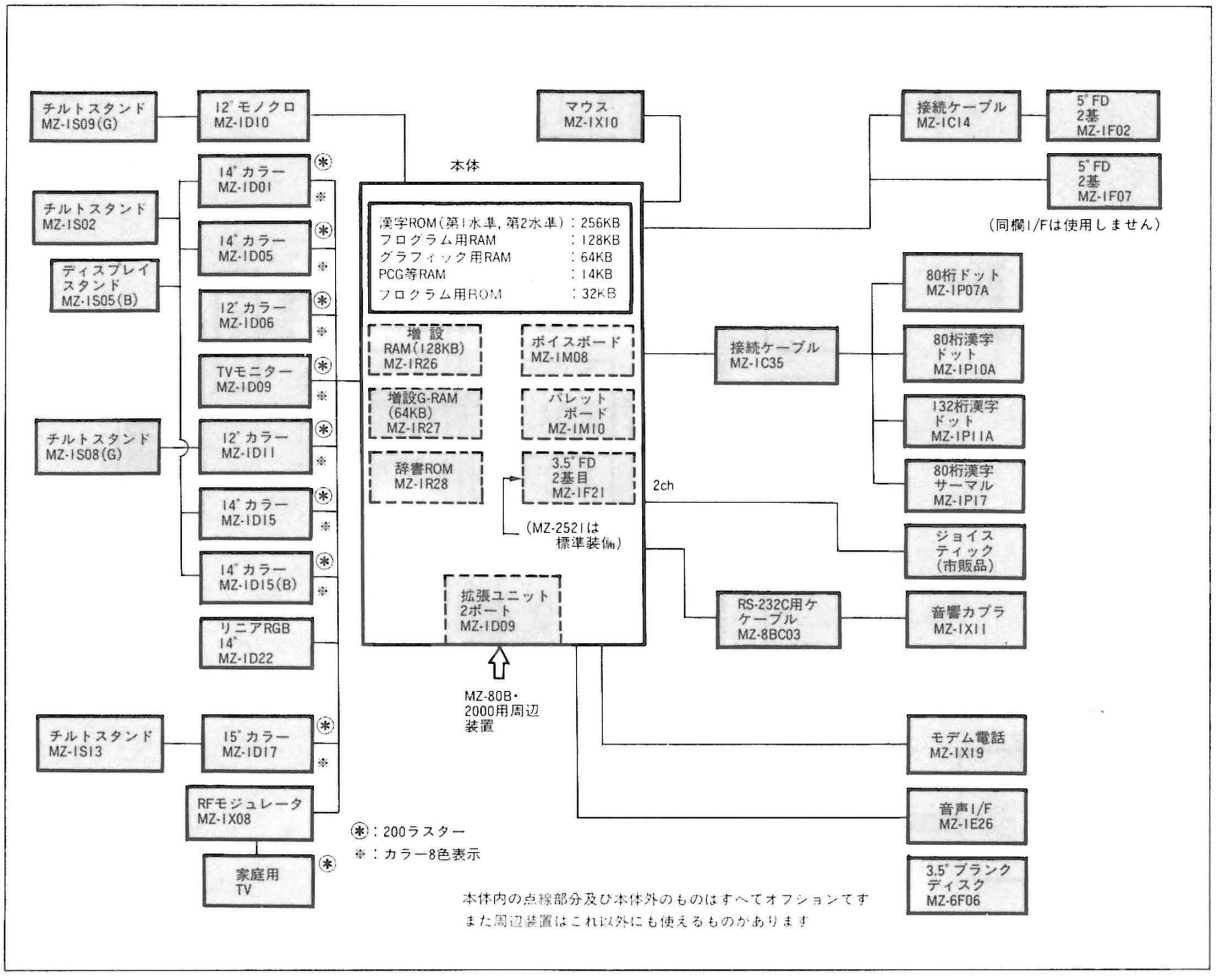
キーボードと本体は細いカールコードで結ばれ，データはマルチプレクス転送されます。従来機との互換性を保つため，本体側には８ビットパラレルのキーマトリックスに変換する専用カスタムLSIを持っています。また，キーボードには 変換 無変換 など，面白そうなキーがいくつかあります(**写真2-3**)。これらについては後ほど詳しく見てみたいと思います。



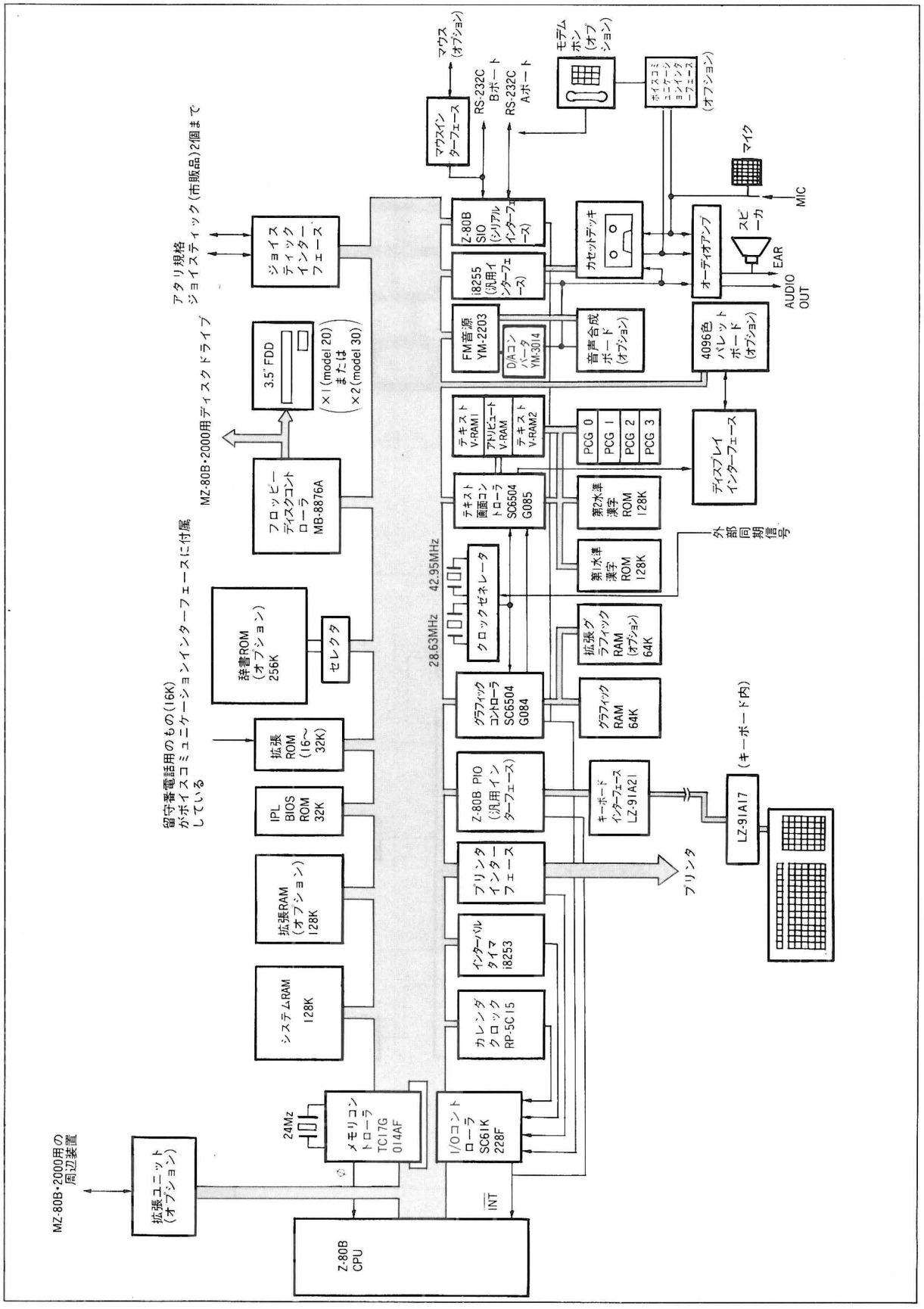
《写真2-2》使用頻度の高いスイッチ類が入っているカンガルーポケット



《写真2-3》アルゴキーを備えたキーボード部

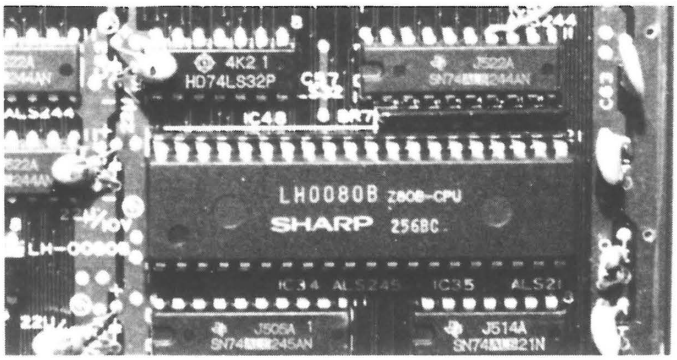


《第2-1図》スーパーMZのシステム構成図



《第2-2図》本体内部を中心に見たブロックダイヤグラム

高速度CPU Z80Bを採用



《写真2-4》スーパーMZの頭脳部「Z80B」CPU

これまでの8ビットパソコンの多くは，CPUにZ80の4 MHzバージョン「Z80A」を採用していました。以前は正規のクロックの半分以下でしか作動しないパソコンもあったようですが，最近はMZ-80B～MZ-2200，X1～X1 turboと同じように，4 MHzきっかりで動くものが多くなりました。

この場合，8088のような8ビットバスの16ビットCPU，特に5 MHz程度のクロックで動かすものとの比較ならともかく，8086を8 MHzで動かす16ビットパソコンと比べると，処理速度の違いは明らかでした。

もちろん画面の表示速度については非常に効果的な高速化技術を取り入れたものもありますし，シャープのパソコンのBASICの場合，Z80のライセンスメーカーとしての技術を生かし，Z80に向いたインタプリタにして高速化を図ってはいるものの，壁を乗り越えるまでには至らなかったようです。

しかしながらスーパーMZでは，CPUにZ80Aよりさらに高速な「**Z80B**」を採用し，高速タイプのメモリをふんだんに使用することで，クロック何と6 MHzで動作させることに成功したのです！　もちろんクロックが速くなればその分実行速度も上がります。これで，ひさしぶりに「4 MHz」の壁を打ち破ることができたと同時に，すべての8ビットパソコンの頂点に立つスピードを誇ることができるようになりました。しかも，それに加えて，MZ-2000以来3年間の沈黙に見合うだけの，その他のハードウエア，ソフトウエア両面での数々の工夫も見られ，これによってBASICの処理速度は8 MHzの8086マシンと比べても，ほとんどひけをとらなくなりました。

なお，MZ-2000・80Bモードでは，従来のソフトの実行速度が変わってしまう(例えばカセットのボーレイトまで速くなってしまう)ため，これらのモードでは自動的に4 MHzで動きます。互換性を重視しているわけですね。

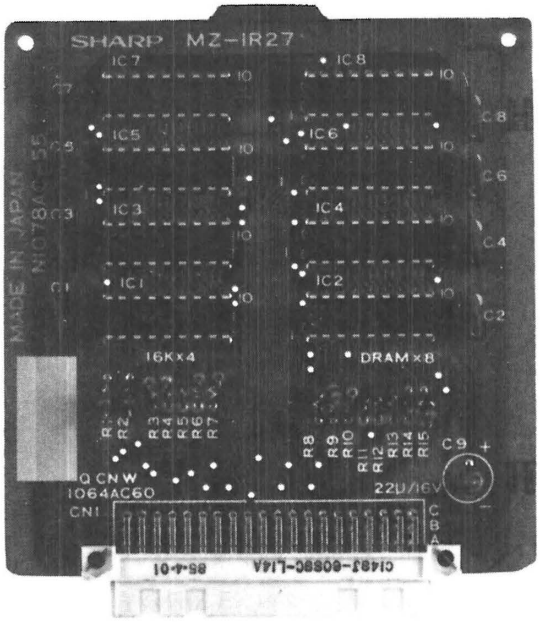
ダントツのグラフィック機能

スーパーMZはグラフィック機能も非常に強力です。

まず，そのままで，横640ドット×縦400ドットのグラフィックができます。また各点毎には4色のカラーが出せます。「各点毎に8色」というのには及びませんが，8ビット機の場合，このドット数ではモノクロしかできないパソコンが多いばかりか，このモードの存在しないパソコンも多いのですから充分，高水準であると言えます。

また，スーパーＭＺでは，画面モードを切換えることで，640ドット×200ドットにすることもできます。この時は各点毎に16色表示が可能で，これまた標準的な640ドット×200ドット×８色よりも上回っています。

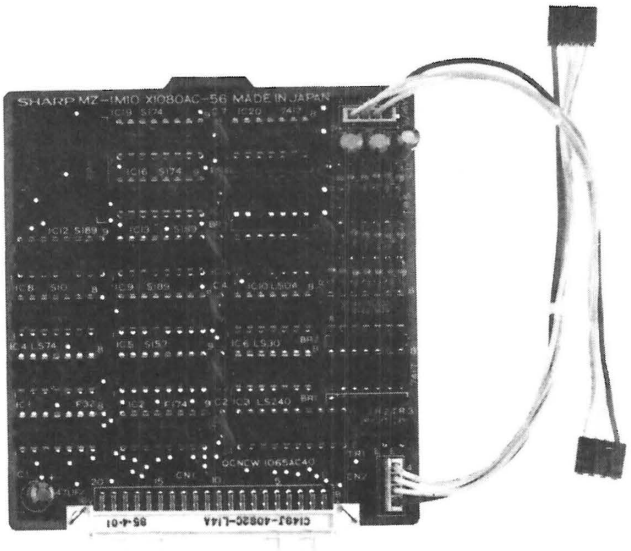
そして，320ドット×200ドットのモードもあります。このモードではページングが可能となり，16色の画面が２枚持てます。これらは別々に描画したり別々に表示したり，一方を表示中にもう一方を描画したりできるほか，両ページの合成も可能です。アニメーション(動画)作成にも最適でしょう。これまで，ごく一部のパソコンではグラフィックの重ね合わせができましたが,「各点毎に16色」でできるというのは，スーパーＭＺが初めてではないかと思います。



《写真2-5》増設グラフィックRAM（MZ-1R27，これをつけると640×400×16色も可能）￥20,000

さらに，これぞ極めつけというモードもあります。320ドット×200ドットの多色モードがこれで，何と，各点毎に256色もの色を付けることができるのです！　当然，８ビット機から16ビット機まで全部含めても，これを越えるパソコンはありません。少し前に発表されたMSX$_2$は256×212×256色なので，これをも２割近く上回わっています。しかも１台のパソコンでこのような多くの色が出せる上，640×400ドットのグラフィックまでできるのは，スーパーＭＺだけです。本機がグラフィックの点においても，これまでに類を見ない存在であることがお分かり頂けたでしょうか？

ところで，まだまだ話は続くのです。スーパーＭＺでは，さらに高度なグラフィックを使いたい人のために，オプションで増設グラフィック RAM カードが用意されています。このカードを使いますと，



●640×400ドット　各点毎に16色

という８ビット機中最高，それに16ビット機でも最新マシンにしか見られないモードや

●640×200ドット　各点毎に16色
　×２ページ(重ね合わせ表示も可)

という普通の８ビット機２台分以上(？)のグラフィック，

そして，

●320×200ドット　各点毎に16色
　×４ページ(1ページ目と２ページ目，そして３ページ目と４ページ目は重ね合わせ表示も可)

《写真2-6》4096色パレットカード(MZ-1M10)￥14,500

さらに，

●320×200ドット　各点毎に256色
　×2ページ

という表示まードまで使えるようになります。スーパーMZがさらにスーパーになるわけですね。

さらに，別のオプションとして，**「4096色パレットカード」**も用意されています。このカードを使わなくとも，256色のモード以外では，16色中16色，または16色中4色のパレットが，利用できるのですが，このカードがあると，4096色中16色※，または4096色中4色※で使えるわけです。本機の「16色」にない，オレンジ色やクリーム色，茶色，スカイブルー，ピンク，はだ色その他の色を，640×200または640×400のモードで見てみたい場合や，例えば，ファッション関係やインテリア関係など「色」を仕事にしていて標準装備の256色でもまだ物足りないような場合に使用するなど，専門的な応用は多く考えられるでしょう。ただ，パレットに依存する形で表示色を増やすことについては「月刊マイコン」等でもたびたび指摘されているようですが，この場合にもあくまでも同時には16色ないし4色ですので，いくら4096色と言っても，万能なわけではありません。普通に絵を描く能力としては「320ドット×200ドット×256色」の方が優れていると考えて良いでしょう。**第2-1表**にスーパーMZのグラフィックモードをまとめておきたいと思います。

ここでちょっと内部に興味のある方に隠し機能をお話ししてみたいと思います。

256色のモードでは，実際には512色中の256色を選択できるようになっています。ただ，この選択は，16色モードの時のパレットのように自由自在にできるというわけではなく，ブロック単位で抜き出した256色ということになります。

BASICではCBLOCK文がこれをサポートしており，256色の選び方は3通りありますが，ハードウエア的には9通り(I_0とI_1の組み合わせ方を変えると18通り)から，選ぶことができるようになっています。CBLOCK文ではG・R・Bのいずれかの最下位ビットを0にしてその光源を4階調にする方法が3通りあるわけですが，この他に最下位ビットを1にして4階調にする方法と，最下位ビットを残りの2原色のどちらかと共用する方法(3原色のいずれかだけが独立するから

《第2-1表》スーパーMZの多彩なグラフィックモード(表中「16色中」とあるところは，パレットカード使用時は「4096色中」となる)

ドット数(横×縦)	標準装備の状態での 使用可能色数，ページ数	増設グラフィックRAM装備時の 使用可能色数，ページ数
640×400	各点毎に16色中4色 1ページ	各点毎に16色中16色1ページ (BASICでは専用の命令はないが，パレットを使って 各点毎に16色中4色×2ページとしても利用可能)
640×200	各点毎に16色中16色 1ページ	各点毎に16色中16色 2ページ
320×200	各点毎に16色中16色 2ページ(重ね合わせ可)	各点毎に16色中16色 4ページ(1ページ目と2ページ目，3ページ目と4 ページ目は重ね合わせ可
	各点毎に256色 1ページ	各点毎に256色 2ページ

3通り)があり，合計9通りということです。ちなみに，さきほど触れた，MSX$_2$の256色モードは，スーパーMZで言うCBLOCK 0の状態，つまり青が4階調の状態に固定されています。

また，本機能には副産物があり，制御ビットの設定の方法により，128色が19通り，64色が8通りできます。128色の場合には，I_0とI_1とを入れ替えると38通りになります。つまり，I_0をG・R・Bのいずれか一つ以上に反映させ，残った光源があれば1か0を割り当てる方法が19通り，I_1で同様に行うのが19通りということです。また，64色はI_0とI_1をどの光源にも割り当てず，G・R・Bの最下位ビットには常に1か0のどちらかを割り当てる状態となります。これらは表にまとめておきましょう(**第2-2表**)。

なお，これを実験したい場合には，何か256色のグラフィックを描いた後で

OUT & HF4, & HA：OUT & HF5, & H21

出力したい値

のようにすればOKです。ただ，この設定はハードウエアに対してのみ有効で，BASICが行っている描画色の512色指定→256色変換等はCBLOCKで指定したもののままです。また，この設定はCBLOCK文かINIT文(CTRL+Dを含む)を実行するまで有効です。

ところで，話のついでに256色のモードの隠し機能をもう一つ紹介しておきましょう。

それは**パレット機能**です。マニュアルその他では256色モードではパレットは使えないことになっていますし，事実，COLOR＝文を使ってみても何の変化もありません。しかし，ハードウエア的には，アナログRGBに変換したり，その際の最下位ビットをどうする云々の前の部分に，パレットレジスタが存在しており，256色モードの時のプレーン並び，

I_1 G_1 R_1 B_1 I_0 G_0 R_0 B_0

この部分

の，上位4ビットだけは，パレットによる色変換が可能です。特にG_1・R_1・B_1については，アナログ信号に変換する際，最も重みのあるビットですので，色を細かく変えることはできなくとも，大域的に変化させることはできるため，工夫次第でなかなか面白そうな応用もできそうです(前出のI_1・I_0の置換えも一緒に使うのも良い)。

BASICで実験するには，COLOR＝は使えないので，OUT文を利用します。

OUT & HBC, &H81：OUT & HBD, &H5

& H80～& H8Fでパレット0～15　& H0～& HF

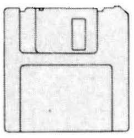
のようにすればOKです。なお，これらのポートの詳しい説明は，内部解析の章を御覧下さい。

※ パレットが作用するのはうち一つした色のみ(パレット0は常にカラー0になる。ただし，カラー0の表示色は変化可能)

《第2-2表》256色モードでの色信号の割り当て

CRTCレジスタ0AHへの出力データ		色成分量			色数	備考
2進表現	16進表現	緑(G)	赤(R)	青(B)		
000000	00	$G_1G_{0l_0}$	$R_1R_{0l_0}$	$B_1B_{0l_0}$	128	
000001	01	$G_1G_{0l_0}$	$R_1R_{0l_0}$	$B_1B_{0l_1}$	256	
000010	02	$G_1G_{0l_0}$	$R_1R_{0l_0}$	B_1B_01	128	
000011	03	$G_1G_{0l_0}$	$R_1R_{0l_0}$	B_1B_00	128	
000100	04	$G_1G_{0l_0}$	$R_1R_{0l_1}$	$B_1B_{0l_0}$	256	
000101	05	$G_1G_{0l_0}$	$R_1R_{0l_1}$	$B_1B_{0l_1}$	256	
000110	06	$G_1G_{0l_0}$	$R_1R_{0l_1}$	B_1B_01	256	
000111	07	$G_1G_{0l_0}$	$R_1R_{0l_1}$	B_1B_00	256	
001000	08	$G_1G_{0l_0}$	R_1R_01	$B_1B_{0l_0}$	128	
001001	09	$G_1G_{0l_0}$	R_1R_01	$B_1B_{0l_1}$	256	
001010	0A	$G_1G_{0l_0}$	R_1R_01	B_1B_01	128	
001011	0B	$G_1G_{0l_0}$	R_1R_01	B_1B_00	128	
001100	0C	$G_1G_{0l_0}$	R_1R_00	$B_1B_{0l_0}$	128	
001101	0D	$G_1G_{0l_0}$	R_1R_00	$B_1B_{0l_1}$	256	
001110	0E	$G_1G_{0l_0}$	R_1R_00	B_1B_01	128	
001111	0F	$G_1G_{0l_0}$	R_1R_00	B_1B_00	128	
010000	10	$G_1G_{0l_1}$	$R_1R_{0l_0}$	$B_1B_{0l_0}$	256	05_Hと同様
010001	11	$G_1G_{0l_1}$	$R_1R_{0l_0}$	$B_1B_{0l_1}$	256	04_Hと同様
010010	12	$G_1G_{0l_1}$	$R_1R_{0l_0}$	B_1B_01	256	06_Hと同様
010011	13	$G_1G_{0l_1}$	$R_1R_{0l_0}$	B_1B_00	256	07_Hと同様 CBLOCK 0
010100	14	$G_1G_{0l_1}$	$R_1R_{0l_1}$	$B_1B_{0l_0}$	256	01_Hと同様
010101	15	$G_1G_{0l_1}$	$R_1R_{0l_1}$	$B_1B_{0l_1}$	128	00_Hと同様
010110	16	$G_1G_{0l_1}$	$R_1R_{0l_1}$	B_1B_01	128	02_Hと同様
010111	17	$G_1G_{0l_1}$	$R_1R_{0l_1}$	B_1B_00	128	03_Hと同様
011000	18	$G_1G_{0l_1}$	R_1R_01	$B_1B_{0l_0}$	256	09_Hと同様
011001	19	$G_1G_{0l_1}$	R_1R_01	$B_1B_{0l_1}$	128	08_Hと同様
011010	1A	$G_1G_{0l_1}$	R_1R_01	B_1B_01	128	$0A_H$と同様
011011	1B	$G_1G_{0l_1}$	R_1R_01	B_1B_00	128	$0B_H$と同様
011100	1C	$G_1G_{0l_1}$	R_1R_00	$B_1B_{0l_0}$	256	$0D_H$と同様 CBLOCK 1
011101	1D	$G_1G_{0l_1}$	R_1R_00	$B_1B_{0l_1}$	128	$0C_H$と同様
011110	1E	$G_1G_{0l_1}$	R_1R_00	B_1B_01	128	$0E_H$と同様
011111	1F	$G_1G_{0l_1}$	R_1R_00	B_1B_00	128	$0F_H$と同様
100000	20	G_1G_01	$R_1R_{0l_0}$	$B_1B_{0l_0}$	128	
100001	21	G_1G_01	$R_1R_{0l_0}$	$B_1B_{0l_1}$	256	
100010	22	G_1G_01	$R_1R_{0l_0}$	B_1B_01	128	
100011	23	G_1G_01	$R_1R_{0l_0}$	B_1B_00	128	
100100	24	G_1G_01	$R_1R_{0l_1}$	$B_1B_{0l_0}$	256	21_Hと同様
100101	25	G_1G_01	$R_1R_{0l_1}$	$B_1B_{0l_1}$	128	20_Hと同様
100100	26	G_1G_01	$R_1R_{0l_1}$	B_1B_01	128	22_Hと同様
100101	27	G_1G_01	$R_1R_{0l_1}$	B_1B_00	128	23_Hと同様
101010	28	G_1G_01	R_1R_01	$B_1B_{0l_0}$	128	
101011	29	G_1G_01	R_1R_01	$B_1B_{0l_1}$	128	29_Hと同様
101000	2A	G_1G_01	R_1R_01	B_1B_01	64	
101001	2B	G_1G_01	R_1R_01	B_1B_00	64	
101110	2C	G_1G_01	R_1R_00	$B_1B_{0l_0}$	128	
101111	2D	G_1G_01	R_1R_00	$B_1B_{0l_1}$	128	$2C_H$と同様
101100	2E	G_1G_01	R_1R_00	B_1B_01	64	
101101	2F	G_1G_01	R_1R_00	B_1B_00	64	
110010	30	G_1G_00	$R_1R_{0l_0}$	$B_1B_{0l_0}$	128	
110011	31	G_1G_00	$R_1R_{0l_0}$	$B_1B_{0l_1}$	256	
110000	32	G_1G_00	$R_1R_{0l_0}$	B_1B_01	128	
110001	33	G_1G_00	$R_1R_{0l_0}$	B_1B_00	128	

CRTCレジスタ0AHへの出力データ		色成分量			色数	備考
2進表現	16進表現	緑(G)	赤(R)	青(B)		
110100	34	$G_1G_0$0	$R_1R_0$1	$B_1B_0$0	256	31Hと同様 CBLOCK 2
110101	35	$G_1G_0$0	$R_1R_0$1	$B_1B_0$1	128	30Hと同様
110110	36	$G_1G_0$0	$R_1R_0$1	$B_1B_0$1	128	32Hと同様
110111	37	$G_1G_0$0	$R_1R_0$1	$B_1B_0$0	128	33Hと同様
111000	38	$G_1G_0$0	$R_1R_0$1	$B_1B_0$0	128	
111001	39	$G_1G_0$0	$R_1R_0$1	$B_1B_0$1	128	38Hと同様
111010	3A	$G_1G_0$0	$R_1R_0$1	$B_1B_0$1	64	
111011	3B	$G_1G_0$0	$R_1R_0$1	$B_1B_0$0	64	
111100	3C	$G_1G_0$0	$R_1R_0$0	$B_1B_0$0	128	
111101	3D	$G_1G_0$0	$R_1R_0$0	$B_1B_0$1	128	3CHと同様
111110	3E	$G_1G_0$0	$R_1R_0$0	$B_1B_0$1	64	
111111	3F	$G_1G_0$0	$R_1R_0$0	$B_1B_0$0	64	



日本語対応のテキスト(文字)画面

長い間8ビットパソコンと16ビットパソコンとをイメージ分けするためのダシに使われて来た感の強い「日本語表示機能」も，1984年秋に発表されたX1turboのおかげで，8ビット機でも漢字V-RAMを搭載して「漢字を一般文字と同じように高速表示することが可能である」と，実証されたようですが，このスーパーMZでも，やはり，漢字V-RAMを採用し，漢字やひらがなを格段に使いやすくしています。

もちろん，BASICでも，

PRINT　“今や常識！　漢字V-RAM”

のように普段私たちが使っているままの文章を，ごく普通の PRINT 文を使って画面に表示することができます。当然，16×16ドットの漢字フォントをわざわざ読出し，グラフィックに点を256個打つのではなく，文字コードを単に文字画面に書込むだけですから，非常に高速なわけです。

ここまで説明すれば，もう言うまでもないかも知れませんが，スーパーMZは最大横に40桁，縦に25行，合計1000文字の日本語(漢字)表示が可能です。一般文字も最大横80桁×縦25行の合計2000文字であり，X1turboや16ビット機と同じです。もちろん80桁×20行，40桁×25行，40桁×20行等のモードもあります。

また，16ビット機には見られない機能としては，例えば80桁×12行や40桁×12行のモードがあります。このモードでは，離れた場所からでもはっきり見える大きな文字を表示させたり，200 ライン系のディスプレイを使った場合にも漢字をきちんと表示させることもできます。

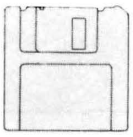
そして，横40桁のモードでは，テキスト画面を2ページ持たせることができるほか，グラフィックの時と同じようにこれら2画面の合成表示も可能です。一枚は固定表示，一枚はキー入力用スクロール表示のように使い分けても良いでしょう。私の知る限りでは，このテキスト画面どおしの重ね合わせ機能は，スーパーMZで初めて実現されたものだと思います。また，グラフィックが256色のモードでは，常に文字画面は40桁(×25行～12行)となり，各文字毎に64色のカラーが使えます。

なお，スーパーＭＺは一般文字１文字分に対し１バイトのアトリビュートを用意していますので，１文字毎に色を変えたり，リバース表示と言って白地に補色で表示したり，点滅表示をさせることもできます。余裕を持って設計されているのがわかります。

第2-3表にスーパーＭＺのテキスト画面モードをまとめておきます。

《第2-3表》スーパーMZのテキスト画面モード

一般文字横桁数（半角文字）	漢字横桁数（全角文字）	縦行数（一般文字，漢字共通）	表示色	ページ数	備　　考
80	40	25	8	1	
		20	8	1	
		12	8	1	200 ライン系ディスプレイでもテキスト画面に書いた漢字がきちんと表示される
40	20	25	8	2	２ページの合成も可
		20	8	2	２ページの合成も可
		12	8	2	200 ライン系ディスプレイでもテキスト画面に書いた漢字がきちんと表示される
40	20	25	64	1	グラフィック256色のモードと対応
		20	64	1	グラフィック256色のモードと対応
		12	64	1	グラフィック256色のモードと対応 また，200ライン系ディスプレイでもテキスト画面に書いた漢字がきちんと表示される

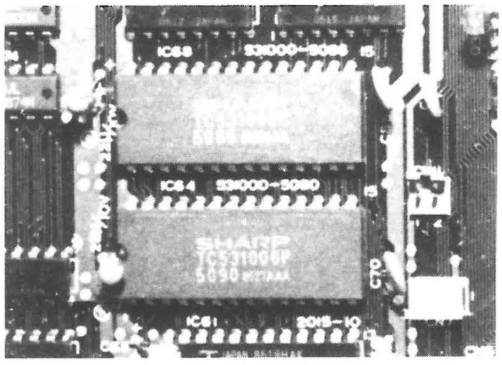


ワープロ顔負け？の漢字機能

前項ではテキスト画面に漢字を表示できる点に触れましたが，この項目では，漢字処理だけについてもっと深く見てみたいと思います。

まず，スーパーＭＺは，他の多くのゲーム用以外のパソコンと同じように，JIS 第１水準の非漢字・漢字を標準装備しています。これは今や常識でしょう。

そして，８ビットパソコンとしては初めて，JIS第２水準までも，標準装備しています。最近の16ビット機では増えて来たとは言え，大半の８ビット機では，たとえオプションででも第２水準は利用できなかったのですから，これはたいへんな進歩です。



《写真2-7》漢字ROM（大容量1Mビットタイプを使用。上はJIS第１水準，下は第２水準）

JIS第２水準には，第１水準に含まれない人名漢字や，固有名詞に多く使われている旧字なども多く含まれています。例えば，

井　亞　佛　曖　曰　漱　藐　獨　珈　琲　發　栞　眞

筺　箘　翔　肛　薇　誂　赳　尭　堯　槇　遙　瑤

などの漢字は，すべて第2水準です。よく，第1水準にないのに問題にされなかったものだと思うようなものが，結構あるでしょう。また，医療関係のように，特定の分野でしか利用されないものはやはり第2水準にまわされています。第2水準があれば，ほとんどの人名，地名，団体名もカナを使わずに表わせるわけですから，本格的な活用には，これが必要不可欠なのではないでしょうか。

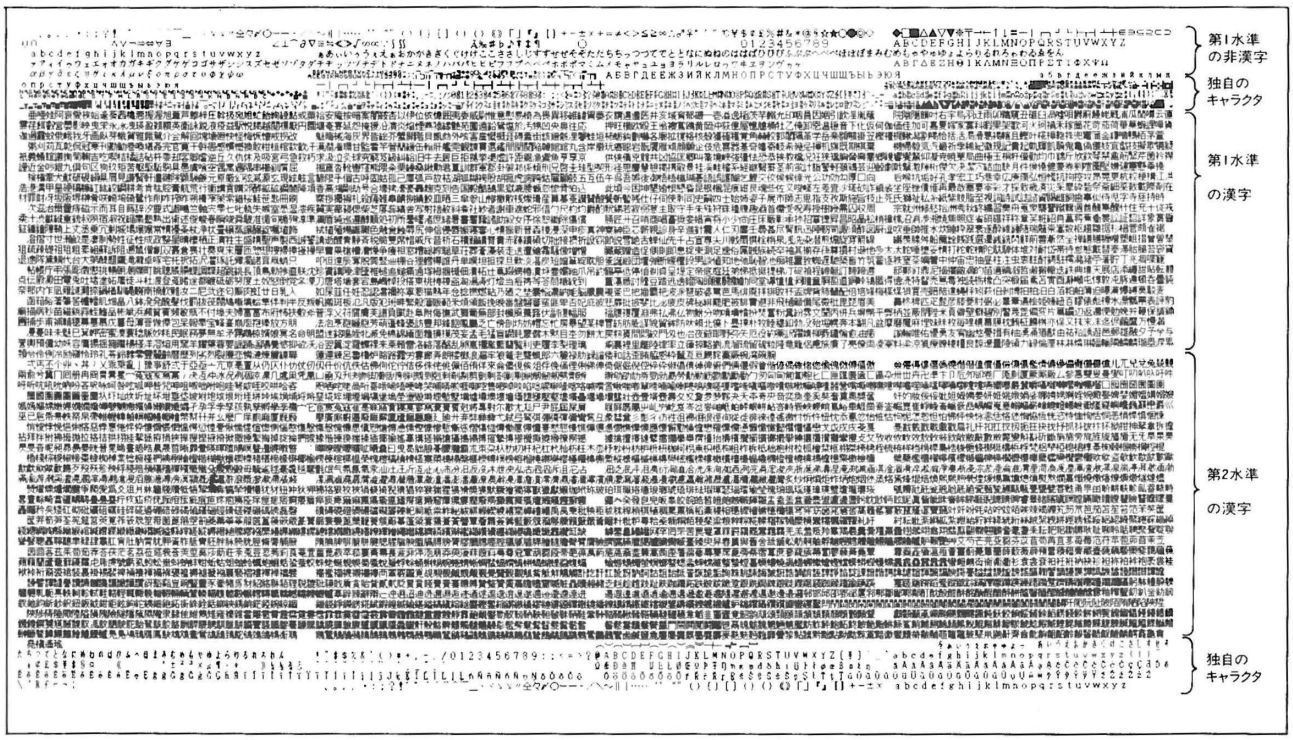
なお，スーパーMZの漢字ROMには，MZ-2500モードの半角キャラクタフォントやMZ-80B・2000モード用フォントも含まれています。80B等のモードでは，起動時にその内容が後述のPCGに転送されて動きます。それと，マニュアルには書かれていませんが，MZ-700・1500の特殊文字フォントも含まれています。**第2-4表**に漢字ROMの全内容を示します。

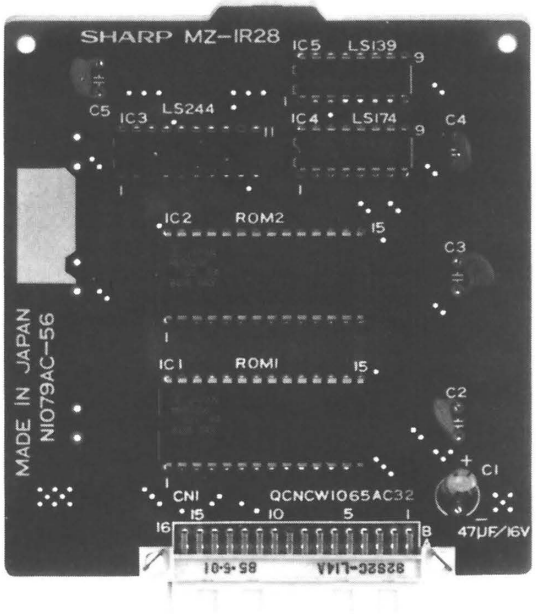
さて，スーパーMZはたくさんの漢字が使えるほかにも素晴しい漢字処理機能を持っています。

それは，オプションで利用できる，**「辞書ROM」**です。これは，9万語もの熟語・単語を，品詞情報とともに持つ，ワープロ顔負けの辞書です。この辞書を使うと，BASICでも9万語もの語を何と文節変換で利用できるようになるのです。

文節変換というのはその語の品詞と，その語が活用する場合には活用語尾まで自動的に判断して変換効率を上げる，たいへん高度なものです。例えば「きる」と入力すると「着る」と「切る」が候補になりますが，「きない」とすると「着ない」に，「きらない」なら「切らない」に，候補を限定してくれます。これは，着るはカ行上一段活用なのに対し，切るはラ行四段活用であり，それを判断してくれたためです。この辞書は，9万語を四段活用，上一段活用，下一段活用，カ行変格活用，サ行変格活用の動詞，形容詞，形容動詞，名詞，副詞，連体詞，接頭詞，接尾詞その他にこと細かに分けて記憶しているのです。

《第2-4表》スーパーMZの持つ全キャラクタ・記号・漢字





《写真2-8》9万語の熟語・単語を登録した辞書
ROMカード(MZ-1R28)￥22,000

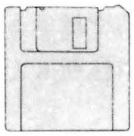
また，この辞書には，よくある人名や地名も記憶されています。みなさんの名前や住所もたぶん変換してくれると思います。

このように，スーパーMZは，扱える漢字の数も，熟語の数や，その変換速度までも，既存のパソコンとは比べものになりません。9万語もの語を集めたのも，またそれを圧縮に圧縮を重ねた結果，256Kバイトにおさめることに成功したのも，長い間ワープロを手がけて来たシャープの技術力のおかげではないでしょうか。

第2-5表にスーパーMZの変換モードをまとめておきます。

《第2-5表》スーパーMZの変換モード

モード	内容
一字変換	辞書ROMは不要。 JIS漢字コードで採用されている読みで変換する。一般に代表的な音読みだけが登録されている。
コード変換	辞書ROMは不要。 JIS漢字コードと，スーパーMZで採用されているシフトJISコードを16進数で入力する。あるいは区点コードを10進入で入力する。予めコード番号がわかっている漢字を入力する時に使う。
半角入力	辞書ROMは不要。 半角のカタカナを入力する時に使うと便利で，アルファベット→カタカナ変換もできる。
文節変換	辞書ROMが必要。 文章を文節ごとに区切って変換する。 例えば，この説明文の場合には，アンダーラインの区切りの所で変換してゆく。
人名・地名変換	辞書ROMが必要 人の氏名(苗字や名前)や都道府県名，都市名，よくある町名が変換できる。



PCGや外字も標準装備

これまでのパソコンの多くは，文字のデザイン(フォント)は固定になっていて，それ以外の文字をテキスト画面に表示することはできませんでした。

それに対してシャープのパソコンでは，以前からフォントを自由に定義できるキャラクタジェネレータ，「PCG」(Programmable Character Generator)を搭載しており，斜体の文字やひらがな等を定義して使うことができました。また，ゲームのキャラクタを高速移動させることができました。

そしてもちろん，スーパーMZでも，PCGが使えるようになっています。

スーパーMZのPCGは，2Kバイト1組のものを，4組(PCG0～PCG3)持っています。2Kバイトというのは，文字のフォントを8ドット×8ドットとした場合，ちょうど256文字全部を定義することができる容量です。

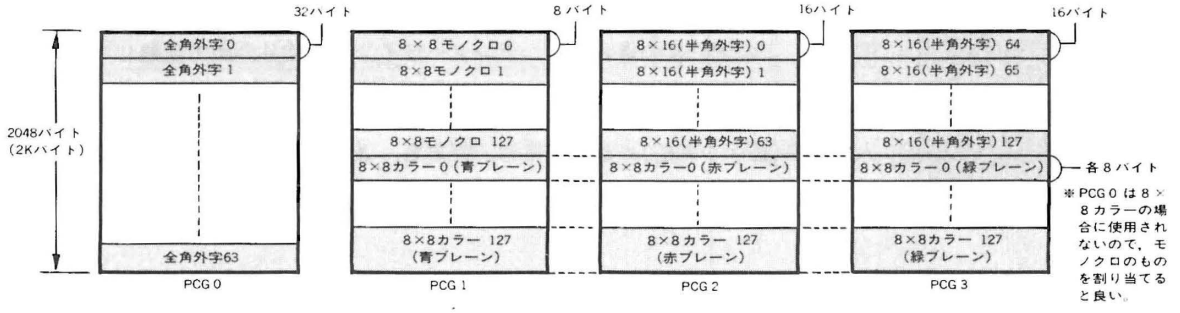
その4組のPCGの使い方は何通りかあり，目的に応じて多用途に使えるようになっています。また，いずれの場合でも，“～”の中に置いて，PRINT文で普通に表示させることができます。

まず一つは，普通のユーザー定義文字として使う方法です。これですと，256×4で1024種もの文字が，書体の違うアルファベットやひらがな，音符や発音記号，ギリシャ文字やロシア文字※などを表現するために使うことができます。

次に，400ライン系のディスプレイを使った場合には一般文字は横8×縦16ドットのきめ細かさで表示されるのですが，これと同じきめ細かさでユーザー定義文字として使う方法があります。あるいは，半角文字の外字と言って良いかも知れません。この場合は，1組につき128文字，全部で512種類の定義が可能になります。

そして，全角(16ドット×16ドット)文字の外字として利用する方法もあります。この場合は，1組について64文字，全部で256文字の外字が使えます。これまでのパソコンで外字が使えるものでも，せいぜい63～64文字程度でしたので，これらを上回わる数となります。

さらに，もう一つの使い方もあります。これはX1シリーズコンパチのもので，8×8ドットのフォントを，各点毎に別々の色(8色)が付けられるものです。スピーディなゲームには，今や必要不可欠になった感じがしますが，この場合，8×8の大きさなら256個のキャラクタの定義が可能です。このモードではPCG1～3を並行して使います。



《第2-3図》PCGの割り当て例(8×8モノクロ128個，8×16モノクロ(半角外字)128個，16×16モノクロ(全角外字)64個，8×8カラー(8色)128個の場合)

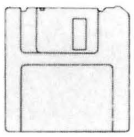
《第２-６表》スーパーMZのPCG機能

モ　ー　ド	使用バンク	使用番号(８×８モノクロの時の通し番号)	使用容量	用　　途
８×８ドットモノクロ	PCG0～PCG3	0～255の任意	８バイト	書体の違うアルファベット等
８×16ドットモノクロ	PCG0～PCG3	0～255の偶数番目と＋１	16バイト	半角の外字
16×16ドットモノクロ	PCG0～PCG3	0～255の４で割り切れる所から四つ	32バイト	全角の外字
８×８ドット各点毎に８色	PCG1～PCG3	0～255の任意だが，PCG1～３の同じ番号のものを，同時に三つ使用する	24バイト	ゲームのキャラクタ

以上，スーパーＭＺには４通りのＰＣＧの活用法がありますが，これらを同時に組合わせて使用することもできます。例えば，８×８モノクロ128個，８×16モノクロ128個，全角外字64個，８×８カラー128個，のような使い方も可能です。

第２-３図と第２-６表に，これらをまとめておきましょう。

※ 普通のアルファベットやカタカナと同じ大きさで(半角サイズで)定義できる。



盛りだくさんの画面制御

スーパーＭＺの画面制御のために，シャープでは専用のLSI，テキスト画面コントローラと，グラフィックコントローラを開発しています。既存のLSIでは機能的にも，スピード的にも満足できるものがなかったためと思われますが，そのおかげで，いくつかの独自機能を持たせることに成功しています。そのうちの主なものを紹介しておきましょう。

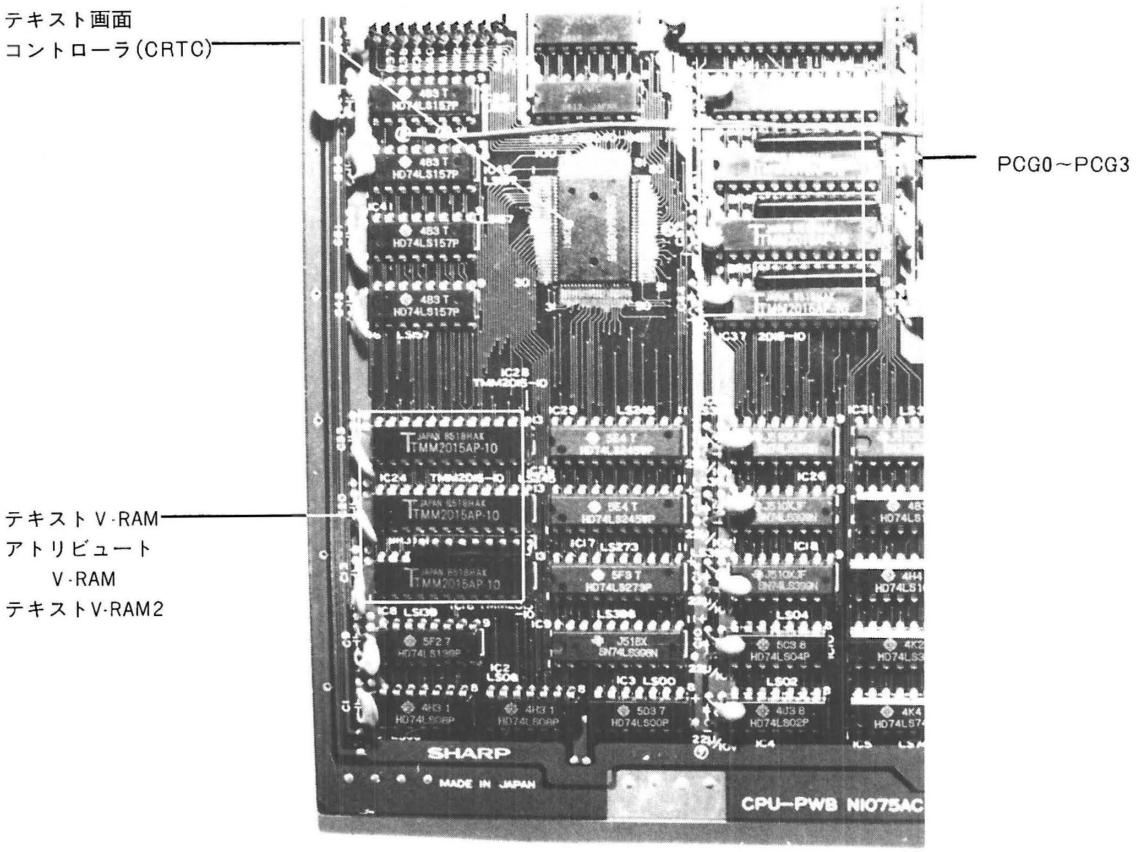
●スムーズスクロール機能

例えば画面ににたくさん文字を表示してゆき，いっぱいになった時には，画面が上にせり上がります。つまり，スクロールアップするわけですが，従来のパソコンでは１文字分，つまり８～16ドット分ずつしか移動できませんでしたので，LISTを表示させたような場合，目がチラついて，本来なら追いつくくらいのスクロールスピードでも，良く読取れないことが多かったと思います。

スーパーＭＺでは，これを改善するため，80桁モードと，40桁モードの中の64色モード，すなわち**第２-３表**のページ数が１のモードの時に限り，１～２ドット分ずつのスムーズスクロールを可能にしました。ちょうど，テレビ番組の最後に出て来るスタッフロール(構成○×△□～制作著作××テレビという，あれです)のように，流れるように出て来ますので，目も疲れませんし，リストを追いかけるのも容易です。ただ，画面全体がスクロールしますので，ファンクションキーの表示や漢字入力はできません。しかし，それでも有難い機能だと思います。

そして，グラフィック画面についても同様のことができます。こちらの方は画面モードの制限はなく，いつでも１ドット単位でスクロールさせることができます(スクロールダウンも可)。

さらに，縦方向に限らず，左右方向にもスムーズスクロールができます。BASICでのサポートはされていませんが，マシン語で操作することは可能です。



《写真2-9》CRTCとPCG0～PCG3，テキスト関係V-RAM

●ハードウエアでビューポートをサポート

これまた高度な処理です。

今までのパソコンでは，BASICのステートメントでソフトウエア的にグラフィック画面にビューポートをもうけ，描画範囲を限定することができるものもありました(スーパーMZにもあります)が，今，紹介しているものはハードウエア的に，表示範囲を限定する機能です。

これまでのパソコンでは，現在表示している画面の一部だけを見せて，他を見せたくたいようなな場合，隠したい部分は黒でぬりつぶすなどして，本当に消さなければなりませんでした。その上，もう一度見せたい場合には，再び描く必要がありました。つまり二度手間となっていたわけです。

スーパーMZでは，この点を改善するため，画面の一部分だけを実際に表示し，他の部分はV-RAMに何が書かれていてもそれを表示しない，ハードウエアビューポートをサポートしたわけです。

なお，さきほどのスムーズスクロール同様，この機能もグラフィック画面とテキスト画面の両方に対し，別々に機能させることができます。例えば，画面の左上は文字とグラフィックの表示，右上は文字だけの表示，左下はグラフィックだけの表示，右下は何も表示しない，というようなことも可能です。この様子は**第2-4図**に示しましょう。

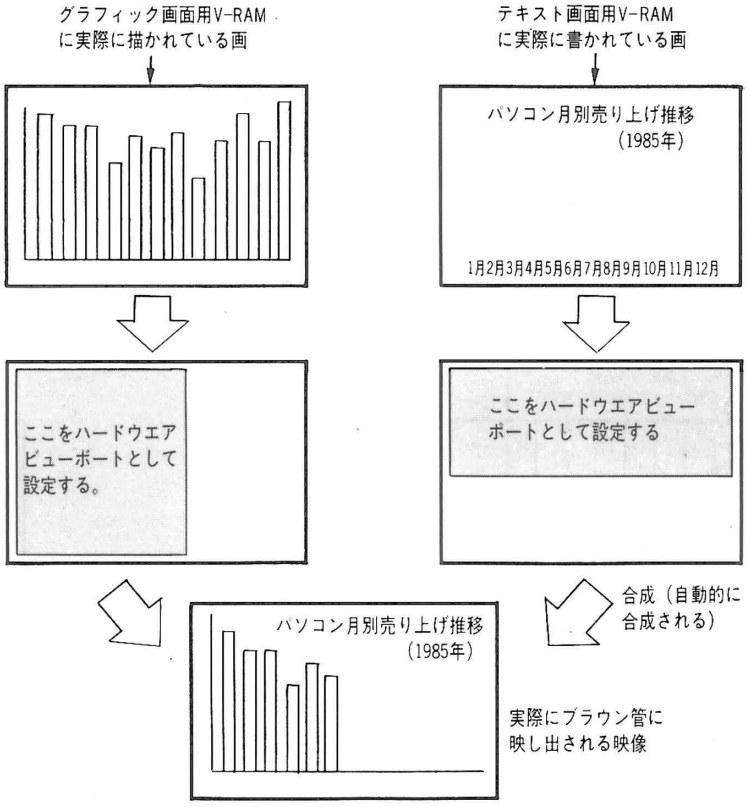
●プライオリティ機能（4色・16色モード時）

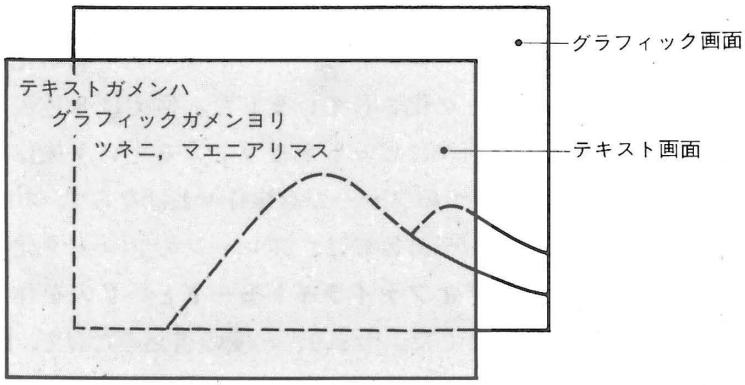
スーパーMZは，グラフィック画面と，テキスト画面を持っていますが，ディスプレイ装置(テ

レビ）に対しては，これらを合成して，出力しています。

その際，これまでのパソコンの多くでは，テキスト画面をグラフィック画面の前に置き，文字の出ていない，黒の所だけ，うしろのグラフィック画面が透けて見えるという**第2-5図**のような方法を採っていました。

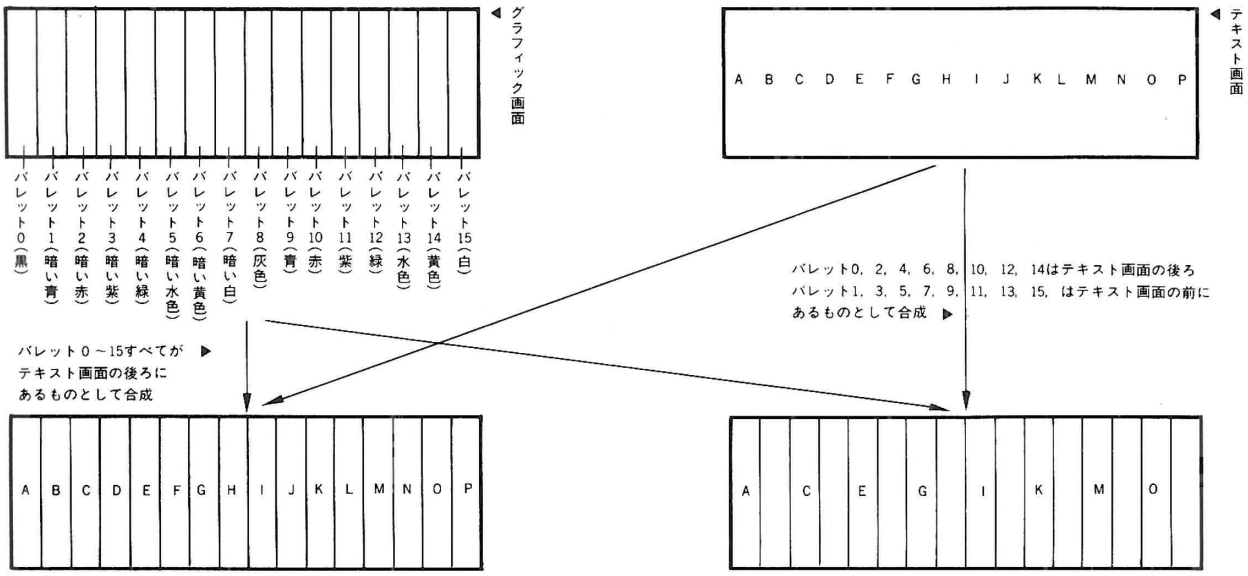
それに対し，スーパーMZでは，グラフィック画面に描かれている画の色(正しくはパレットコード)別に，仮想的にグラフィック画面がいくつものモノクロの画面を重ね合わせた構造になっているものと考え，各々についてテキスト画面の前に来るのか，後ろに行くのかを決められる，という方式になっています。電源投入時はすべてのパレットの画面がテキストより後ろになっていますが，例えば偶数番目のパレットの画面をテキストより後ろに，奇数番目を前にすると，パレット0，2，4，6…と重なったところでは文字が見えますが，パレット1，3，5，7…と重なったところでは文字は隠れてしまって見えません。**第2-6図**のようになるわけです。これをプライオリティ(優先順位)機能というのです。プライオリティ機能の用途ですが，普通にはちょっと思いつきません。しかし，前述のPCGと合わせて使うと，極めて効果的な利用ができます。





《第2-5図》これまでの多くのパソコンのグラフィック・テキスト重ね合わせ方法

それは立体的な動画です。例えばPCGには，それをいくつか使って人間の姿を定義しておき，グラフィックで描いた風景の中を歩かせるのです。すると，灰色の電柱よりは手前だが，赤の車よりはむこう，つまり人間が電柱と車の間を歩いていると言った画面が作れるわけです。もちろん，パレットコードが違うなら，実際に同じ色で表示していても，異なるプライオリティが付け



《第2-6図》スーパーMZのプライオリティ機能の概念図

られますので，2台の赤い車の間を歩かせることもできるのです。

なお，ここまでのプライオリティとはちょっと違うのですが，グラフィック画面同士を2ページ重ね合わせる際にも，優先順位が決められます。この場合，前面に割り当てられたページのパレットコード0の部分だけ，後ろの画面が透けて見えます。

●リードモディファイライトモードとサーチモード

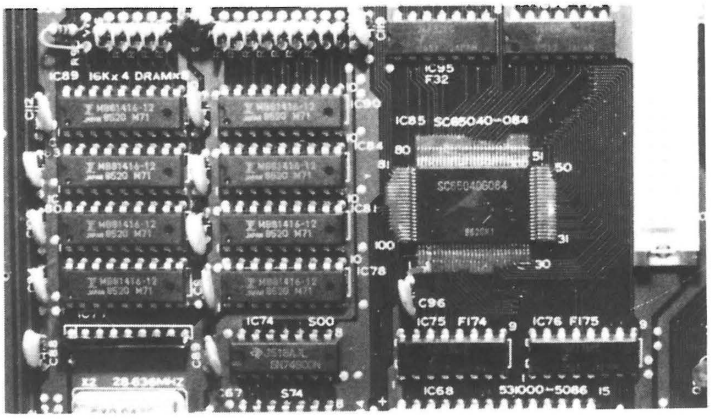
これらは，極めて内面的な機能です。BASICでプログラミングする際には全く意識する必要はありません。グラフィックの処理速度を上げるために用意された，特別なものだからです。

これまでの多くのパソコンでは，ハードウエアを簡略化するために，グラフィックRAMは，各プレーン毎にブロック化されていました。例えば8色表示のパソコンでは，一つの点を打つにも，三つのプレーン別々にビットをセットするという操作が行われていたわけです。また，表示する色が16色なら，四つのプレーンを操作せねばならず，スピードの低下は避けられませんでした。

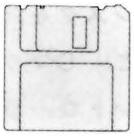
それに対してスーパーMZでは，プレーンのブロック化はそのままですが，グラフィックコントローラに**リードモディファイライトモード**というのを作り，一度の操作だけで全プレーンに書込むことを可能にしました。つまり，一度の書込みだけで，LSIが自動的に各プレーンからデータをとり，OR・ANDを行った後で各プレーンにデータを戻すという処理を自動的に行ってくれるわけです。これで，モノクロの時と同じスピードで，多色描画ができるようになりました。ドットのセットが速くなれば，これを使った他の描画機能も，すべて速くなります。LINEやCIRCLE等も超高速なのです。

次に**サーチモード**というのは，ペイントの速度を上げるためのものです。要するにリードモディファイライトが同時書込み用なら，このサーチモードは同時読出し用の機能です。ペイントというのは，ある境界色までをぬりつぶすものですから，実際の処理はある点が境界色かどうかの判断と，ドットのセットを繰返すことになります。ドットのセットはさきほどのリードモディファイライトで速くなりましたが，境界色の判断はあい変わらずです。サーチモードというのは，この判断を速くするための機能なのです。

グラフィックコントローラにはこの他にも，**リプレース機能**(AND・ORしない)，**タイリングペイント用同時書込み機能，高速画面クリア機能**(最大0.032秒で消せる)等も備えています。大変強力なLSIなのです。



《写真2-10》G-CRTCと標準装備分のグラフィックRAM(64Kバイト)



大容量128K RAM実装

スーパーMZは，何と128KバイトものRAMを標準装備しています。しかも，オプションの追加で，256Kバイトまで拡張することもできます。

従来からシャープでは，パソコンだけを特別扱いせず，ミニコンや大型コンピュータと同じような，メモリのほとんどをRAMとしてユーザーに開放するシステム，「**クリーンコンピュータ**」システムを採用して来ました。

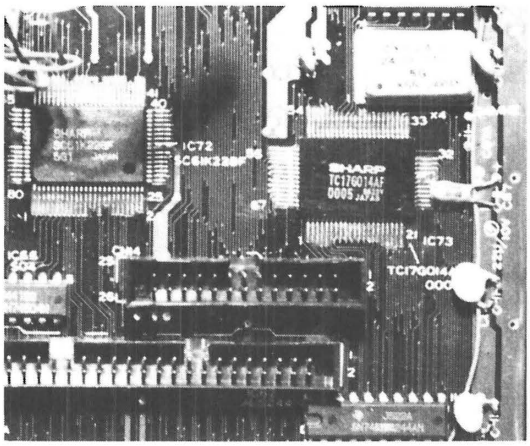
スーパーMZの場合も，キャラクタジェネレータ以外のROMは，フロッピーディスク等からBASICやCP／M等，目的のシステムソフトウエアをロードして実行する**IPL**(イニシャルプログラムローダ)と，キーボードからの入力や画面への表示などハードウエアに密着した基本的な機能をサポートする**BIOS**(Basic Input Output System)のための32Kバイトだけで，他のメモリはすべてRAMです。

ところで，後で詳しく触れますが，スーパーMZは，ハードウエアだけでなく，ソフトウエアも超強力なので，BASICもとても大きくなっています。ROM中のBIOS部を含めなくとも，56K～120Kバイトにも上ってしまうのです。このため，RAMが128Kバイトあっても，実際にユーザーが使えるのは，最大でも約70Kバイトになってしまいます。もちろんディスク使用時のユーザーエリアがこれほどあるZ80マシンは他にX1turbo(80Kバイト)くらいしかありませんし，16ビット機のMS-DOS上で動くBASICの大半は58K～62Kバイト程度しか使えませんので，ほとんどの人はこれでも充分すぎるくらいかも知れません。もし，これで不足ならRAMを256Kバイトに増設すれば良いのです。すると増設した128Kバイト分は，そっくりそのままユーザーエリアに加算され，何と184Kバイトまで使用できるようになります。標準装備のRAMでさえBASICですべてを使えずに，ただの発熱器と化しているパソコンさえある中，信じられない大きさではないでしょうか。

さて，スーパーMZが大容量のメモリを持てることはわかったわけですが，これだけのメモリを，いったいどうやってZ80(B)に接続するのでしょうか？御存知のように，Z80(ファミリ)を

初めとする８ビットCPUのメモリ空間は64Kバイトです。普通につないだだけでは，残りの64Kバイト～192Kバイト分は，それこそ発熱器になってしまいます。

スーパーMZでは，このために，専用のメモリ管理用カスタムLSIを開発しています。そして，単なるバンク切換え方式より，はるかに汎用性の高い，**マッピング方式**を実現しています。このマッピング方式は有名なところでは日立のＳ１が採用してますが，スーパーMZのCPU，Z80(B)ではメモリ空間にでなくI／O空間にマッピングレジスタを置けるため，さらにフレキシブルなシステムに仕上っています。



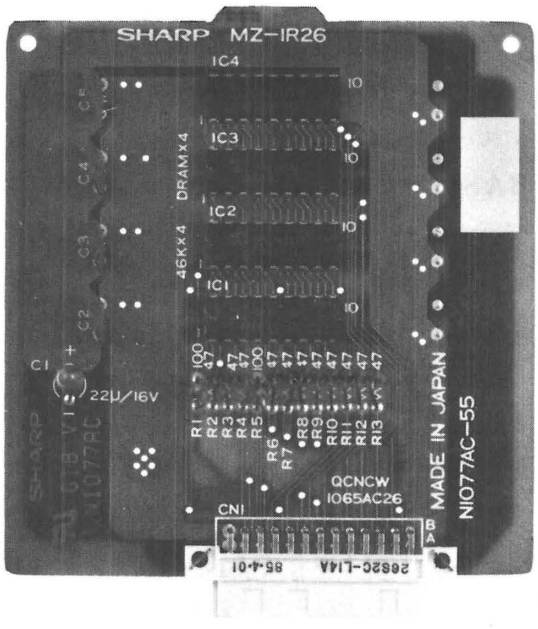
《写真2-11》メモリマッピングレジスタを内蔵するメモリコントローラ(右)とI／Oマップ及び割込みを制御するI／Oコントローラ(左)。右上に見れるのは24MHzのX'tal(水晶)発振子

第２-７図を見て下さい。これはスーパーMZのメモリマップです。このように，Z80(B)メモリを８Kバイトずつ，八つに分割し，各々を一度マッピングレジスタに通すことで，64ブロック中の任意の一つを自由にアクセスすることができるのです。また，二つ以上のZ80側のブロックから共通の実メモリを参照することもできます。パレットレジスタに良く似た構造になっているわけです。

この方式を，単なるバンク切換えの方式と比べると，

・クリーンコンピュータのメリットを最大限に引き出せる。
・サブルーチンパッケージなどがメモリのロケーションを気にせずに作れる。

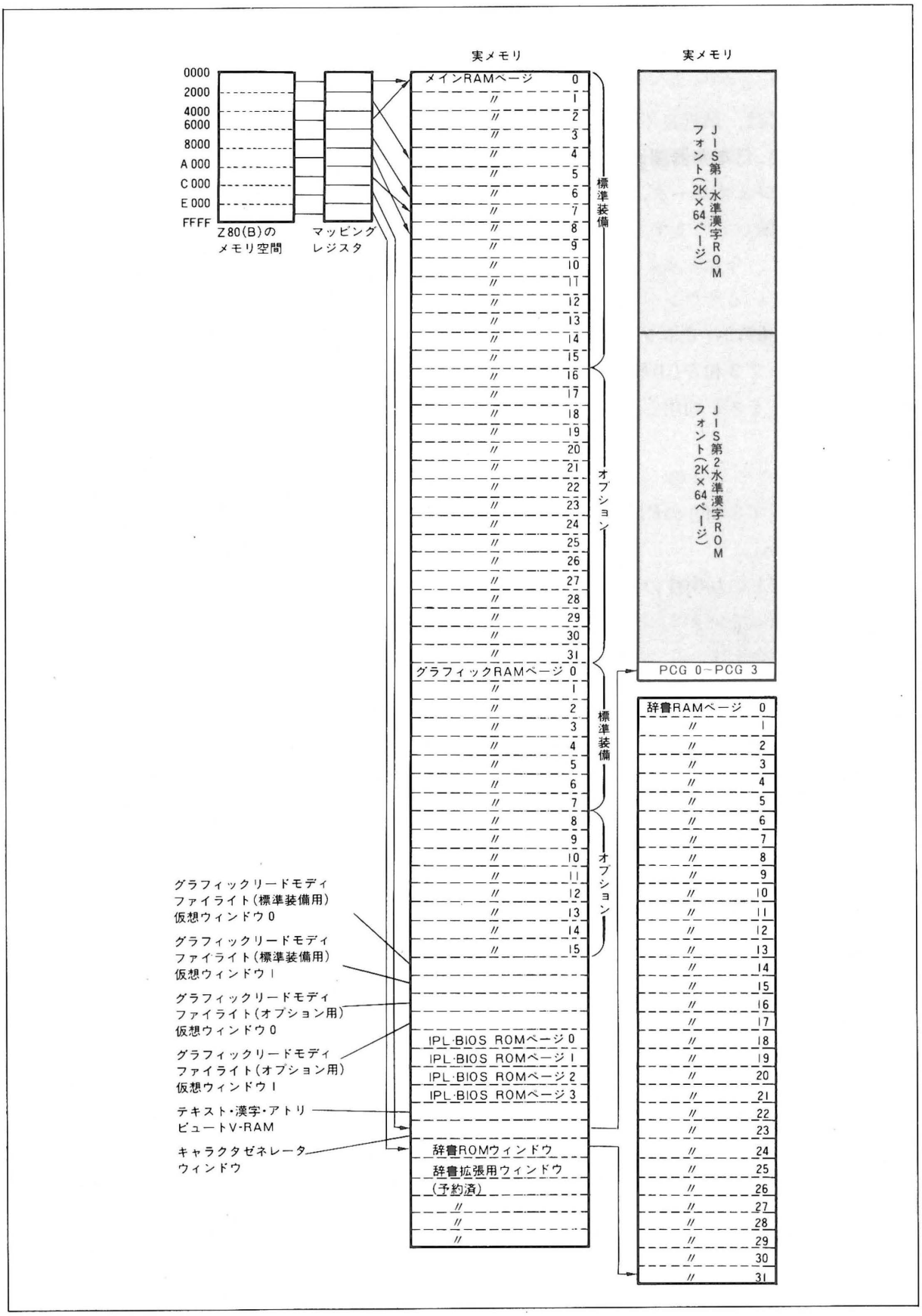
などのメリットがあります。



《写真2-12》128K増設RAM(MZ-１R26，これをつけると８ビット最高の256Ｋ実装になる)￥35,000

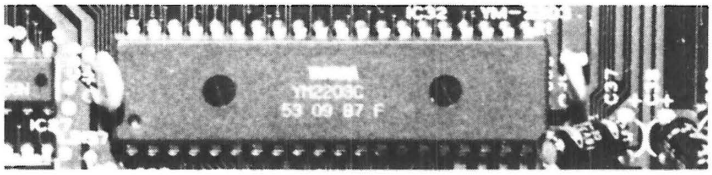
例えば，グラフィックRAMはC000H～FFFFHの部分をバンク切換えでアクセスする，と決めた場合，バンク切換えによって隠れてしまうこのエリアにはグラフィックを扱うプログラム（サブルーチン）は置けません。汎用性が低下してしまうわけです。その点メモリマッピングシステムなら心配無用です。

また，このシステムでは，例えば，２000H～３FFFHのメモリで実行できるプログラムを20本用意しておき，これを順次切換えて実行したり，必要なものをダイレクトに選択して実行することもできます。これが，サブルーチンのパッケージ化です。



《第2-7図》スーパーMZのメモリマップ(マッピング例)

FM音源も標準装備



《写真2-13》FM音源用LSI「YM-2203」

スーパーMZは，最近良く使われるようになった，**日本楽器製造(ヤマハ)のサウンドジェネレータ，「YM-2203」**を標準装備しています。

これによって，今なお多くのパソコンで使われているサウンドジェネレータ**「AY-3-8910」**(ゼネラルインストウルメンツ社製)とコンパチの

・8オクターブ3和音(矩形波)

・ホワイトノイズを利用した1効果音

と，

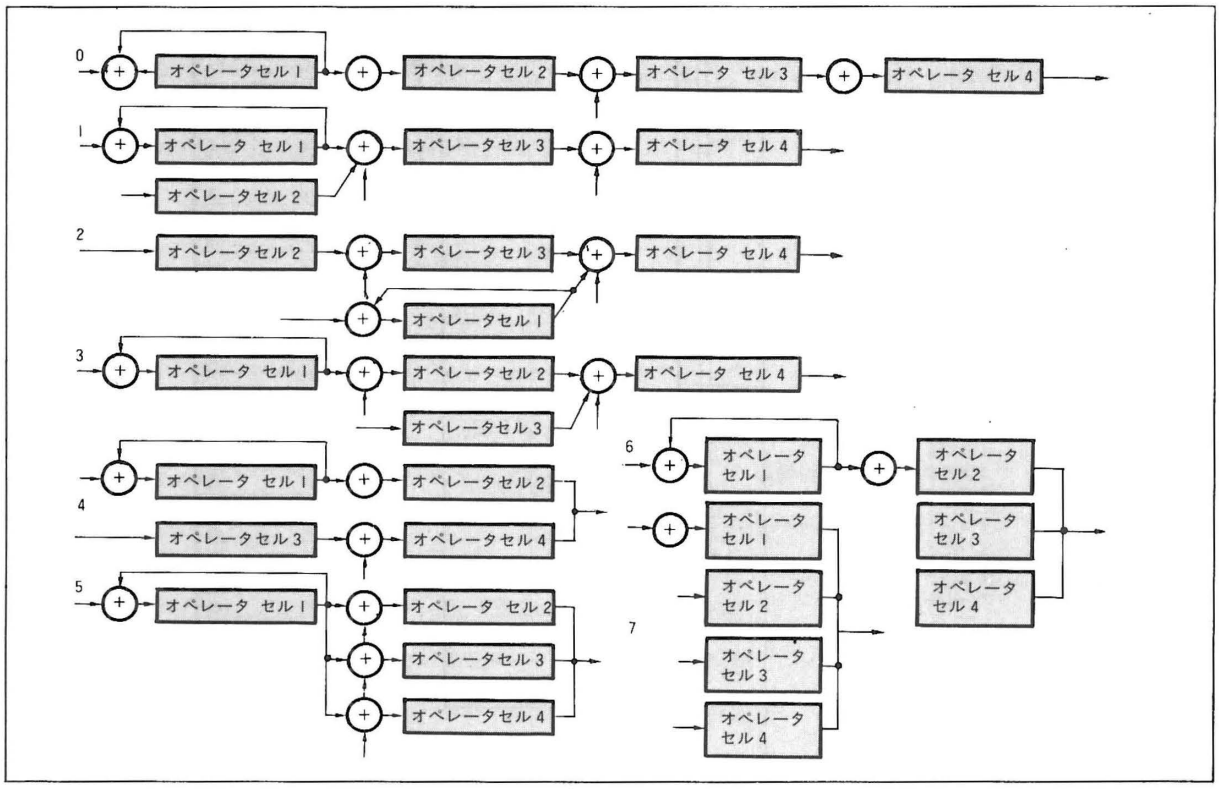
YM－2203オリジナルの

・8オクターブ3和音のFM音源

が利用可能です。

また，従来機との互換性のため，CPUが直接信号をハイレベル・ローレベルにして音を発生できるポートも持っています。MZ－2500のモードでも，エラー等で聞こえる「ピッ」という音はこれで発生させています。

FM音源は，周波数変調を何重にもかけて，本物の楽器にたいへん良く似た音が発生できる方式



《第2-8図》オペレータセルの組合わせ

で，本物の電子楽器(キーボード)でも，使われている本格的なものです。昔からあった，正弦波と，その2倍，4倍，8倍…を合成してゆく方式とは違い，容易にあたたかみのある音を作れるのが特長です。

音楽に詳しい方のために，FM音源の内部構造に触れておきます。

まず，FM音源は，三つのモードに設定できます。

・各音毎にオペレータセルを4個持ち，これを8通りに組合わせて，各々を別の音色で演奏できる。音程のきちんとついた楽器の音を出すのに良い。

・2音は前項と同じだが，もう1音は4個のオペレータセルの周波数パラメータを変えて，効果音に適した音を出せる。

・やはり2音は前々項と同じだが，もう1音は複合正弦波合成によって音声を出力できる。

通常は，1番初めのモードで使います。また，オペレータセルというのは正弦波を発生させるもので，入力x，出力をyとすると，

$$y=\sin(x)$$

のような関係になります。出力レベルを変えることもできます。$y=a\sin(x)$のようにもできるわけです。また8通りの組合わせというのは，**第2-8図**のようなものです。発生させたい音色の種類（例えば弦楽器系，管楽器系など）によって，最適なものが選べます。YM－2203のレジスタ構成を**第2-9図**に示します。

00H	チャンネルA分周比（下位8ビット）							
01H					チャンネルA分周比(上位4ビット)			
02H	チャンネルB分周比（下位8ビット）							
03H					チャンネルB分周比(上位8ビット)			
04H	チャンネルC分周比（下位8ビット）							
05H					チャンネルC分周比(上位4ビット)			
06H				ノイズ（効果音）分周比				
07H	ポートB 方向	ポートA 方向	チャンネルC ノイズ部	チャンネルB ノイズ部	チャンネルA ノイズ部	チャンネルC トーン部	チャンネルB トーン部	チャンネルA トーン部
08H					チャンネルA エンベロープ	チャンネルA音量		
09H					チャンネルB エンベロープ	チャンネルB音量		
0AH					チャンネルC エンベロープ	チャンネルC音量		
0BH	エンベロープ（下位8ビット）							
0CH	エンベロープ（上位8ビット）							
0DH					エンベロープ形状			
0EH	ポートA（スーパーMZでは出力）							
0FH	ポートB（スーパーMZでは入力）							
21H	テストデータ							
24H	タイマーA（上位8ビット）							
25H								タイマーA(下位2ビット)
26H	タイマーB							
27H	モード			タイマーBリセット	タイマーAリセット	タイマーBイネーブル	タイマーAイネーブル	タイマーBロード	タイマーAロード
28H	キーON OFF スロット						チャンネル	
2DH	FM音源部を$\frac{1}{6}$または$\frac{1}{3}$に，SSG部を$\frac{1}{4}$または$\frac{1}{2}$の分周比にする							
2EH	FM音源部を$\frac{1}{3}$に，SSG部を$\frac{1}{2}$の分周比にする（2DH出力後）							
2FH	FM音源部を$\frac{1}{2}$に，SSG部を$\frac{1}{1}$の分周比にする							
30H～3EH		デチューン				マルチプル		
40H～4EH		トータルレベル						
50H～5EH	キースケール		アタックレート					
60H～6EH			ディケイレート					
70H～7EH			サステインレート					
80H～8EH	サステインレベル				リリースレット			
00H～9EH					SSGタイプ エンベロープ制御			
A0H	チャンネル1 F・ナンバー1							
A1H	チャンネル2 F・ナンバー1							
A2H	チャンネル3 F・ナンバー1							
A4H		チャンネル1 ブロック			チャンネル1 F・ナンバー2			
A5H		チャンネル2 ブロック			チャンネル2 F・ナンバー2			
A6H		チャンネル3 ブロック			チャンネル3 F・ナンバー2			
A8H	チャンネル1 F・ナンバー1（効果音モード・音声合成モード用）							
A9H	チャンネル2 F・ナンバー1（〃）							
AAH	チャンネル3 F・ナンバー1（〃）							
ACH		チャンネル1 ブロック(〃)			チャンネル1 F・ナンバー2 (〃)			
ADH		チャンネル2 ブロック(〃)			チャンネル2 F・ナンバー2 (〃)			
AEH		チャンネル3 ブロック(〃)			チャンネル3 F・ナンバー2 (〃)			
B0H		チャンネル1 セルフィードバックク			チャンネル1 コネクション			
B1H		チャンネル2 セルフフィードバック			チャンネル2 コネクション			
B2H		チャンネル3 セルフフィードバック			チャンネル3 コネクション			

SSG部（AY-3-8910コンパチ）

OPN（FM音源）部（YM-2203オリジナル）

30H～9EHの詳細：（スロット＝オペレータセル）（×3H, ×7H, ×BHは未使用）

×0H	チャンネル1	スロット1	×8H	チャンネル1	スロット2
×1H	チャンネル2	スロット1	×9H	チャンネル2	スロット2
×2H	チャンネル3	スロット1	×AH	チャンネル3	スロット2
×4H	チャンネル1	スロット3	×CH	チャンネル1	スロット4
×5H	チャンネル2	スロット3	×DH	チャンネル2	スロット4
×6H	チャンネル3	スロット3	×EH	チャンネル3	スロット4

《第2-9図》YM-2203のレジスタ

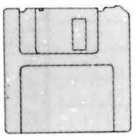
なお，BASICでは初めから**第2-7表**のように30種類の音色が定義されていますので，このように面倒なことを知っている必要はありません。手軽にピアノやフルート，オルガン，ギター，ハープシコード，ブラス，トランペット，ストリング等の音色が楽しめます。もちろん，音楽を演奏しながらグラフィックを描くなどの処理も同時に行えます。

■この項の参考・引用文献

日本楽器製造　ＹＭ－2203
アプリケーションマニュアル

《第2-7表》BASICでサポートされている音色

番号	音色	番号	音色
0	ハープシコード	15	パイプオルガン２
1	ブラス１	16	フルート
2	ブラス２	17	ピッコロ
3	トランペット	18	オーボエ
4	ストリング１	19	クラリネット
5	ストリング２	20	グロッケン
6	エレクトリックピアノ１	21	ビブラホン
7	エレクトリックピアノ２	22	シロホン
8	エレクトリックピアノ３	23	琴
9	ギター	24	チター
10	エレクトリックベース１	25	クラビネット
11	エレクトリックベース２	26	ベル
12	エレクトリックオルガン１	27	ハープ
13	エレクトリックオルガン２	28	ハーモニカ
14	パイプオルガン１	29	ティンパニー

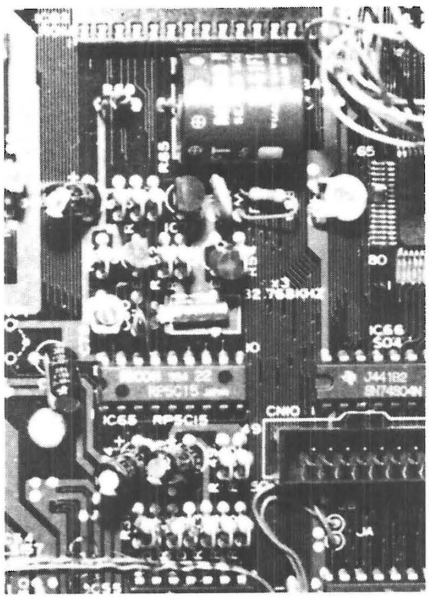


年まで刻めるバッテリバックアップカレンダクロック採用

スーパーＭＺは，カレンダクロックとして**リコーの「RP-5C15」**を採用しており，ニッカド電池でバックアップしていますので，本体側の電源を切っても動き続けます。

ＲＰ－5C15は，年の桁までカウントできるのが特長で，年が変わっても前の年のまま，ということもありません。また，うるう年の自動判別もやってのけたり，アラーム時刻を設定して，その時刻になったらCPUに割込みをかけてくれたりもする，高性能なＩＣです。

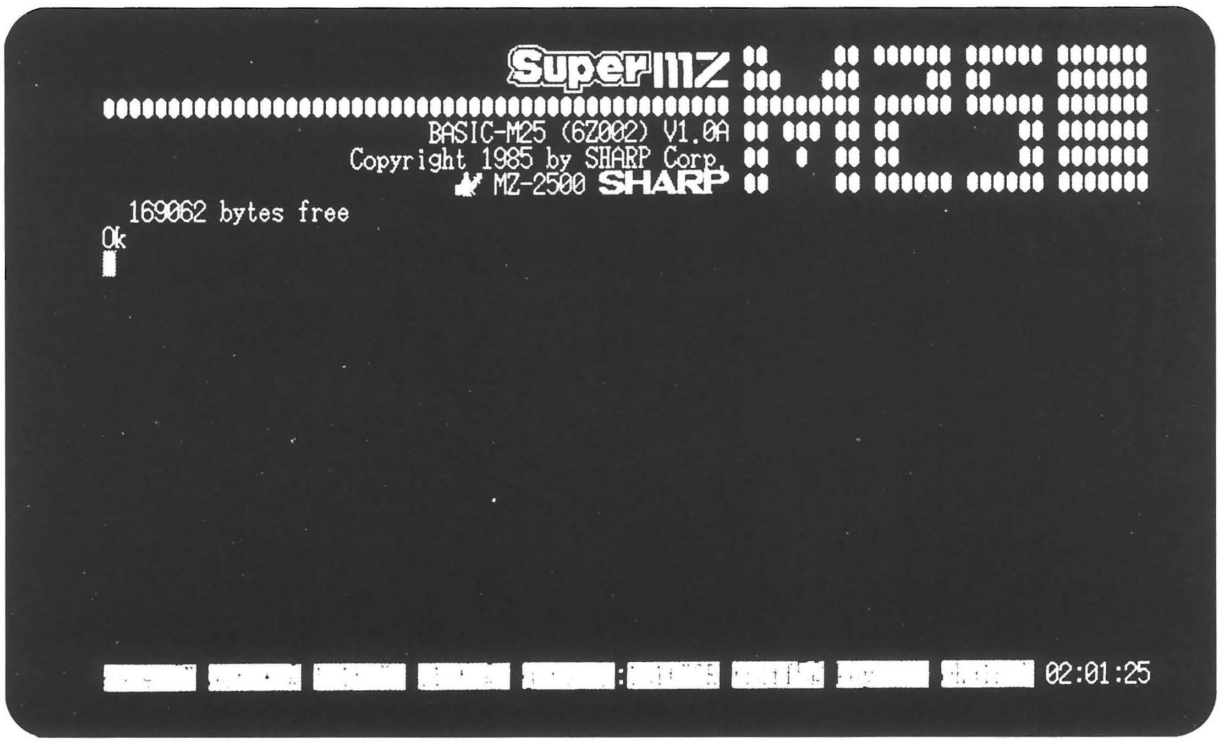
また，BASICでは，普通にセットしたり，読出したりして使うほか，Ｘ１シリーズや16ビット機のMS－DOSと同じように，プログラムをセーブするなどファイルを作成し



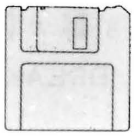
《写真2-14》カレンダクロック「RP-5C15」（真上にある小さな円筒形のものは32.768KHzの水晶発振子。さらに上にある大きな円筒形のものはニッカド電池）

た時の日付けや時刻を，自動的に登録する機能を持っています。プログラムを作ったり，デバッグしていて，どれが最新版？と迷うこともありません。たいへん親切な設計です。

それに，スーパーＭＺでは，画面のファンクションキーリスト表示の10番の所に，常に現在時刻を表示しておけるモードもあります(**画面2-1**。消すこともできます)。

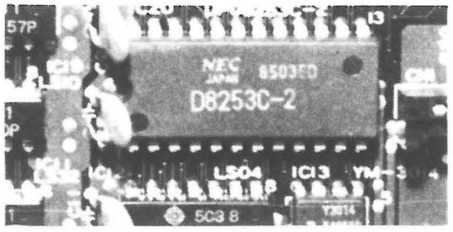


《画面2-1》ファンクションキーの10番の所に現在時刻を表示させておくこともできる



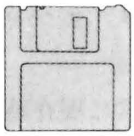
1/10秒まで刻めるストップウォッチ機能

スーパーMZでは，カレンダクロックの他に，汎用カウンタとして有名なｉ8253を搭載し，ハードウエアでストップウォッチを装備しています。私の知る限りでは，カレンダクロックとストップウォッチの両方をハードウエアでサポートしているのは，これが初めてです。ただ実際にはこのLSIはインターバルタイマとして動作し，ストップウォッチはソフトで構成しているのですが，マシン語を使うとハード的に行わせることも可能です。

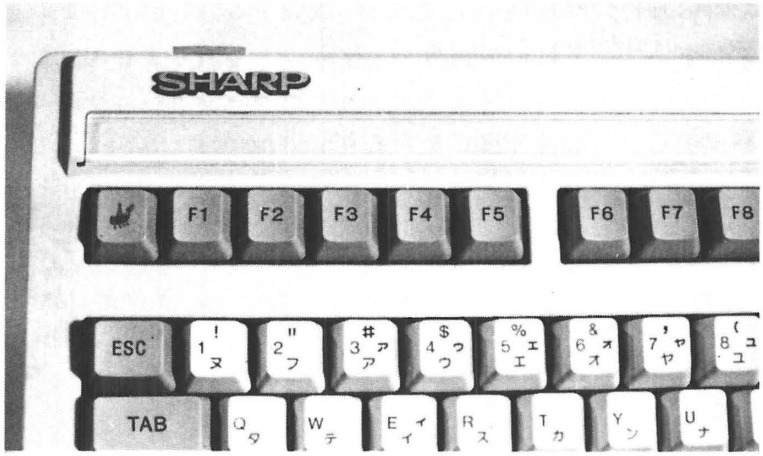


《写真2-15》通常はインターバルタイマとして使用している i8253(セカンドソース使用)

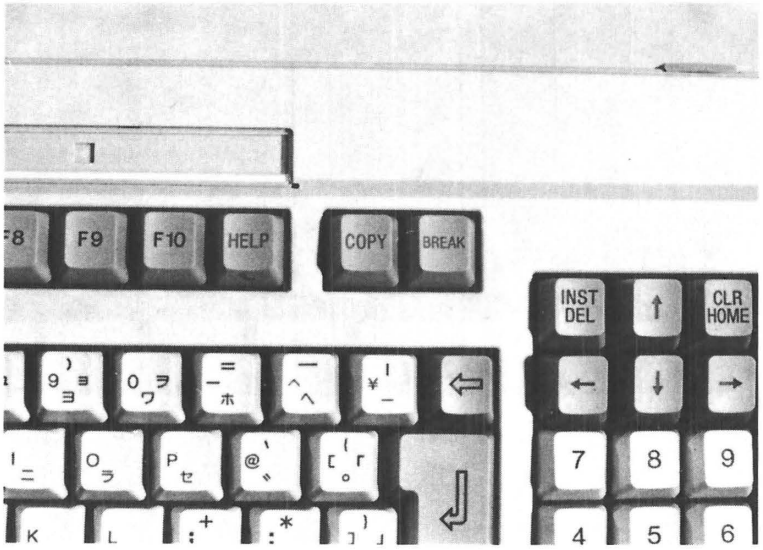
ところで，ストップウォッチ機能は，その名の通り，リアルタイム型ゲーム等で時間を計ったりするものですが，その際，カレンダクロックを破壊せずに済むので便利です。つまり，バッテリバックアップの意義が出て来るわけです。それに，これを行う「TIME」システム変数は，時計の「TIME＄」とは異なり，数値型で扱えますから，たいへん便利です。



キーボード



《写真2-16》キーボード左上すみがアルゴキー



《写真2-17》フルキー部右上にあるHELP，COPY，BREAKキー。HELPキーは(電話)キーも兼ねている

スーパーMZのキーボードは，他の8ビット機と同じ，ちょっと小さめのものですが，98個ものキーを無理なく配置しており，使い勝手は大変良好です。もちろんシリンドリカルステップスカルプチャータイプで，キータッチも良く，操作性にも優れています。

まず，目につくのは，**アルゴキー**(🐓)と10個のファンクションキーです。アルゴキーは，ソフトウエア編で紹介する**スーパーヘルプ機能**を呼び出すために使うもので10進／16進電卓機能，カラーシミュレーション機能，オートダイヤル機能などが利用できます。

次に，10個のファンクションキーは16ビット機では普通ですが，8ビット機ではあまり見かけないものです。BASICではこれを SHIFT キーとの併用で20個として使えるほか，ワープロソフト等の操作性が上がることも予想されます。SHIFT なしで10個あると，頻度の高い機能，例えば，削除，改行，改ページ，再変換，半角，倍角，右よせ，中央印字，均等，移動などが，1キーで実現できることになるからです。

ファンクションキーの右には，HELP キー，COPY キー，BREAK キー等があります。COPY キーは間違って押してしまっても大丈夫です。SHIFT キーや CTRL キーとの併用で，初めて，**ハードコピー**になるからです。また，このキー単独では，画面に既に表示されている文字を1文字単位でコピーすることができます。ワープロで言う「**複写**」ですが，タイプの苦手な人や，漢字を写したい時には大変重宝です。また，BREAK キーも単独では一時停止にしかなりません。SHIFT キーと一緒に押して，初めてプログラムの実行を中断できます。つまり，知らない人が間違って押してしまっても，安全というわけです。

次に，フルキー部で目につくのは 変換 キーと 無変換 キーです。これらは漢字入力モードで使うもので，入力した「読み」等を漢字に変換したり，漢字に変換せずに(カナの)半角・全角文



《写真2-18》少し背の高い←↑↓→キーはたいへん便利

字にしたりするキーです。

なお，このモードに入るには SHIFT ＋ 変換 で良く，右手だけでOKです。

テンキー部では，

↑

← ↓ →

というカーソル移動キーが目につきます。私はかねがね，カーソル位動キーはフルキーとテンキーの間に，と主張して来ましたが，残念なことに，スーパーMZでも実現されませんでした。しかし，それ以外のカーソル移動キーの配置方法としては，やはりこの並べ方がベストでしょう。それに，**写真2-18**のように，この四つのキーは，他の周りのキーに比べ，少し背が高いのです。これなら，目がディスプレイを見つめていても，指は手さぐりでこのキーにたどりつけます。→キーを押そうとして／キーを押してしまった，というミスも防げるわけです。また↑や↓はBASICのエディットモードでは，他のパソコンの ROLL UP ROLL DOWN の働きもします。

そして HOME と CLR の関係が，単独では左上に移動するだけの HOME，SHIFT との併用で画面を消す CLR となる点も感心できます。間違って押してしまっても，安心です。それに，これは常識ですが，5 の上のポッチもきちんと付いています。

このように，なかなかの親切設計のようですが，テンキー側に＝キーがない点が少しだけ気になります。＝はプログラム中でも以外に多く使われる文字だからです。しかし，これも慣れれば何も問題ないことかも知れません。

一方，ソフトウエアの対応も充実しています。

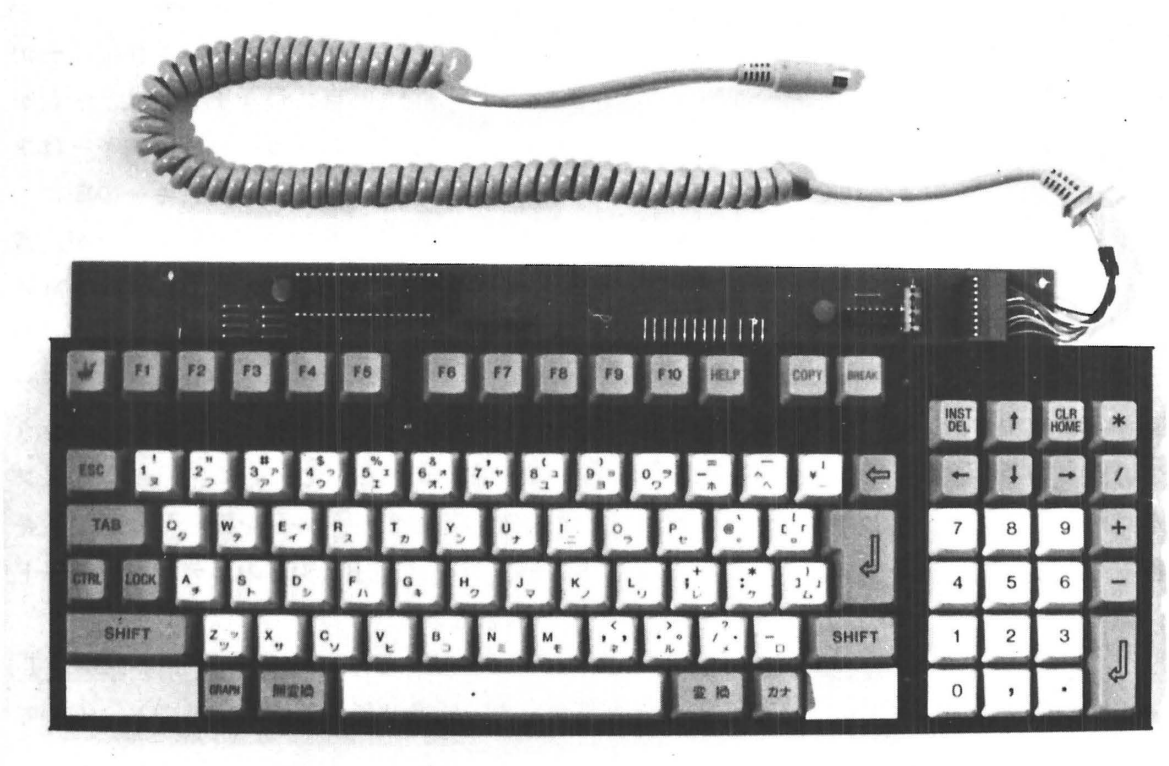
まず，キー入力のシミュレートができます。先行入力バッファにデータをセットすることで，INPUT文での疑似入力や，デモ，自己増殖プログラム等に応用したり，また，〝〟をセットすることで先行入力バッファのクリアにも使えます。

また，キーボードを押した時に，「ポツッ」というクリック音を発生させることができます。キーボードだけを見ながら操作していても，キーが受けつけられたことが，「耳」で確認できるわけです。画面も見ている場合にも先行入力時にはやはり便利です。もちろん，これが見障りな人は，音を出さないようにすることもできます。

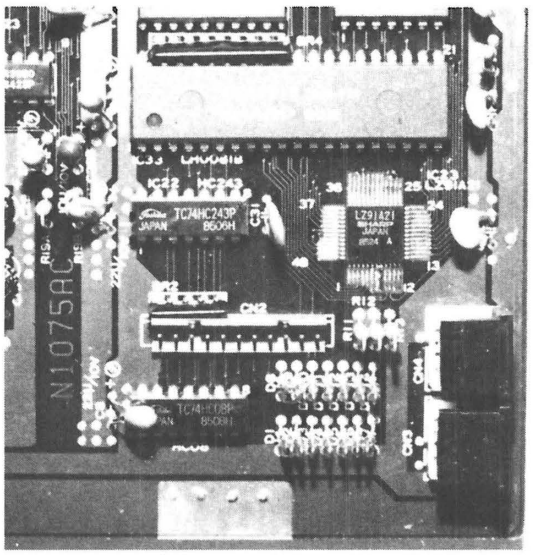
次に，リピートする際，その速度を4段階に変えられます。それにリピートさせないこともできます。慣れない人でキーボードを確実に押すことに専念してしまい，思わずリピートしてしまった(?)という光景はたいへん良く見かけるのですが，そのような時はリピートを禁止するのが良いでしょう(REPEAT OFF ↵とする)。

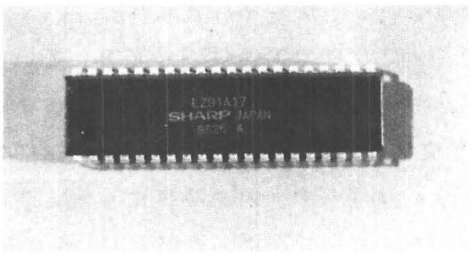
そして，最後にカナ配列です。

スーパーMZでは，もちろんJIS配列を採用していますが，もし，カタカナを習いたての小学校低学年の子や，どうしてもJISに馴じめない人のために，これをBASICの命令で，MSXのような50音配列に変えるこのもできるようになっています。キーボードにはJISの「ヌ フ ア ウ エ オ…」が印刷されていますが，「ア イ ウ エ オ…」と書かれた付属のシールを貼ると，見やすくなるでしょう。



《写真2-19》キーボードを分解した所





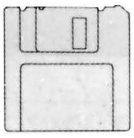
《写真2-20》左は本体側でキーボードを主に扱っているZ80B PIOとLZ91A21。右はキーボード側に入っているLZ91A17

《第2-8表》スーパーMZのコントロールキー

キー	コード	働き
CTRL + A CTRL + SHIFT + INST	01	インサートモードに入ります。インサートモードでは，カーソルの点滅が速くなり，キーが押される度にカーソルから右が一つずつ右にずれ，カーソルも一つ右に動きます。そして元のカーソルの位置には押したキーの文字が表示されます。
CTRL + B SHIFT + ←	02	一つ前の語にカーソルを移動させます。語というのはスペースや記号で区切られた英数字のことです。
CTRL + D	04	コマンド待ちの時にこのキーを押すと，BASICのINIT文を実行したのと似た働きをします。キーボードのリピートの速度を速くしすぎたりCONSOLE文でテキスト(文字)画面のソフトウエアウィンドウを小さくしすぎてINIT↵さえも入力できなくなってしまったり，その他困った時にこれらをまとめて元に戻してくれる「お助け」キーです。TONE文やSOUND文でサウンドLSIを変に操作して音が止まらなくなってしまったりパレットやPCGで画面が見えなくなってしまったりユーザー座標系などを使ったために思い通りのグラフィックを描けなかったような場合も利用できる，大変便利なキーです。
CTRL + E CTRL + DEL	05	カーソル以下その論理行の末尾まで消します。
CTRL + F SHIFT + →	06	一つ後の語にカーソルを移動させます。
CTRL + G	07	ピッというベルを鳴らします。単なるBEEP文と同じです。
CTRL + H ⇦	08	カーソルの左の文字を1字抹消します。つまり，カーソル以降を一つずつ左に移動させカーソルも一つ左に移動します。新しいカーソルの位置に以前書いてあった文字は消えます。⇦は↵の上のキーです。← ではありません。
CTRL + I TAB	09	次のタブ位置までカーソルを移動させます。
CTRL + J	0A	行を分割します。カーソル以降は次の行の先頭に移動します。それと同時に↵キーを押したのと同じになります。つまり分割した行の前半は，そのまま入力されることになります。
CTRL + K HOME SHIFT + ↑ CTRL + ↑	0B	カーソルを画面(正しくはテキスト画面のウィンドウ)の左上隅に移動します。下の二つはエディットモードでは$\frac{1}{2}$ページ分前の行を表示させます。
CTRL + L SHIFT + CLR	0C	画面を消してカーソルを左上隅に移動させます(正確には、テキストウィンドウ内を消してカーソルをその左上隅に移動させます)。
CTRL + M ↵	0D	入力の終わりの合図として押します。カーソルは次の行の先頭に移ります。プリントした時は単に改行します。

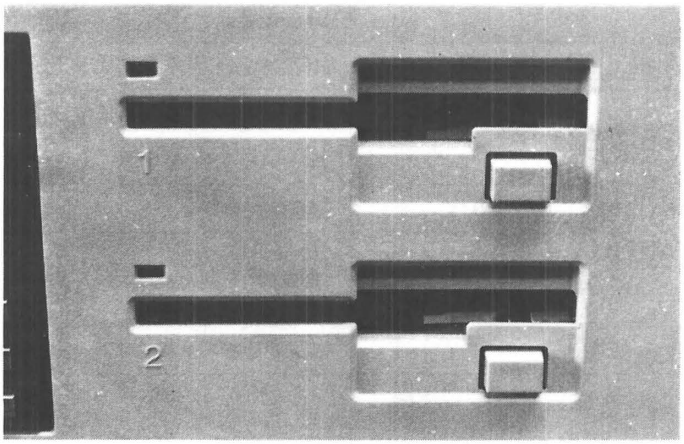
キー	コード	働き
CTRL + N	0E	カーソルのある行以上をスクロールアップさせます。カーソルの位置とカーソルより下の行は変化しません。
CTRL + O	0F	カーソルのある行以下をスクロールダウンさせます。カーソルの位置とカーソルより上の行は変化しません。
CTRL + R INST	12	'I文字，スペースを挿入します。つまりカーソル以降を一つずつ右に移動させカーソル位置にスペースを置きます。カーソルは移動しません。
CTRL + T	14	カーソル位置をタブとして登録します。
CTRL + V	16	先行入力バッファをクリアします。BASICのKEY 0,""と同じ働きをします。
CTRL + W	17	カーソルのある行と次の行とを論理的につながったものとします。通常は，エディット等により1行が長くなってしまってもカーソルのある行から上だけがスクロールアップして極力次の行を保護しますが逆に故意に次の行とつなぎたい時にはこのキーにより論理的に接続します。
CTRL + X	18	カーソルのある論理行全体を消しカーソルをその先頭に移動させます。今までの入力をすべて取り止め初めからやり直したい時などに利用します。
CTRL + Z	1A	カーソル以降画面(正しくはテキストウィンドウ)の右下隅までを消します。カーソルは移動しません。
CTRL + ¥ →	1C	カーソルを一つ右に移動させます。
CTRL +] ←	1D	カーソルを一つ左に移動させます。
CTRL + ^ ↑	1E	カーソルを一つ上に移動させます。
CTRL + _ ↓	1F	カーソルを一つ下に移動させます。
CTRL + →		カーソルをその行のスペース以外の最後の文字の次の位置に移動させます。:でBASICのステートメントを続けたり，最後の方から⇦キーでステートメントを削除するのに便利です。
CTRL + ←		カーソルをその行の先頭に移動させます。
CTRL + ↓ SHIFT + ↓		カーソルを画面(正しくはテキストウィンドウ)の左下隅に移動させます。エディットモードでは$\frac{1}{2}$ページ分後の行を表示させます。
DEL		カーソル位置の文字を抹消します。つまりカーソルの一つ右の文字以降を一つずつ左に移動させます。カーソルは移動しません。カーソルの位置には以前カーソルの右隣にあった文字が来ます。いわゆる吸い取り型のデリートでインサートモード中で使うととても便利です。
CTRL + TAB		次のタブ位置までスペースを置きます。カーソルも移動させます。
CTRL + ⇦		カーソル以降の文字を1行の先頭から始まるように，左に移動させます。⇦キーをたくさん押したのと同じです。
BREAK		プログラムや命令の実行を一時停止します。

キー	コード	働き
SHIFT + BREAK		プログラムの実行を中断します。
☞		スーパーヘルプ機能(電卓・カラーシミュレーション・オートダイヤラ)を呼び出します。
HELP		HELP割り込みを発生させます(INPUT文などで)。
COPY		コピーモードに入ります。コピーモードではカーソル位置に画面に既に表示されている文字を複写することができます。
CTRL + COPY		画面(グラフィック)のハードコピーを取ります。
CTRL + SHIFT + COPY		画面(テキスト+グラフィック)のハードコピーを取ります。
SHIFT + COPY		画面(テキストのみ)のハードコピーを取ります。
LOCK		アルファベットキーの大文字・小文字の状態を切換えます(SHIFT LOCKではなくCAPS LOCKです)。
GRAPH		セミグラフィックキャラクタやコントロールコードシンボルキャラクタを入力するモードに入ったり出たりします。
カナ		カタカナを入力するモードに入ったり出たりします。
SHIFT + 変換		漢字(日本語)入力モードに入ったり出たりします。
漢字入力モード: 変換		入力した文字を漢字に変換します。
漢字入力モード: 無変換		入力したカナ等を漢字に変換せずそのまま取込みます。
漢字入力モード: F 1		ローマ字入力・カタカナ入力のモードを切換えます。ローマ字時にもカタカナ入力は受け付けるので実質的には英字の扱いを決めることになります。
漢字入力モード: F 2		これから入力する文字をひらがなにするかカタカナにするかを切換えます。一字変換・文節変換等はひらがなのみを受け付けるのでカタカナは全角のカタカナ入力時にのみ使うことになります。
漢字入力モード: F 3		コード変換の際, JIS・シフトJISで入力するか, 区点コードで入力するかを切換えます。
漢字入力モード: F 5		すでにバッファに入力されている「読み」(キーボードから打ち込んだ文字)をカタカナにしたりひらがなにしたります。
漢字入力モード: F 6		変換モードを切換えます。変換モードは, 一字変換・コード変換・半角入力・文節変換・人名地名変換の順で切換わり再び一字変換から始まります。ただし辞書ROMが入っていない場合は文節変換・人名地名変換は飛ばされます。
漢字入力モード: →		一字変換時, 次の候補を選びます。
漢字入力モード: ←		一字変換時, 前の候補を選びます。
漢字入力モード: ↑		前の候補に戻ります(一字変換時は前の7文字分に戻る)。
漢字入力モード: ↓		次の候補を出します(一字変換時は次の7文字分を出す)。
漢字入力モード: ↵		選んだ漢字をカーソルの位置に取込みます。

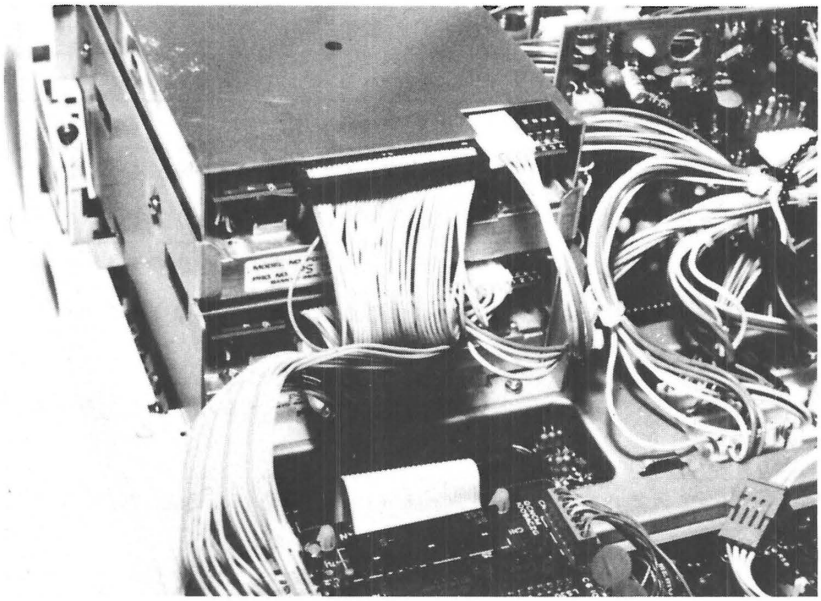


3.5インチ2DD マイクロフロッピーディスクを採用

スーパーMZは,フロッピーディスクに,**3.5インチマイクロフロッピーディスクの2DDタイプ**を採用しています。容量は1基あたり640Kバイトで，８インチや５インチ２HDの約１Mバイト(約1000Kバイト)には及ばないものの，多くの８ビット機が標準装備している，５インチ・3.5インチ２Dは320Kバイトですので，これらよりは倍の大きさになっています。model 20ではこれを１基，model 30では２基，標準装備していることは既に触れた通りです。



《写真2-21》3.5インチマイクロフロッピーディスク部(model 20で１基，model 30で２基)



《写真2-22》本体のフタを開けて3.5インチFDDを背面から見た所

3.5インチのディスケットは,約9cm角(厚さ約3mm)と，５インチの約13cm角(厚さ1.5～2mm)よりかなり小さく，また，堅いジャケットに入っていて，記録窓も金属で保護されていますので，取扱いも比較的容易です。ワイシャツのポケットに入れたり定形郵便物として郵送することも可能です。

ところで，3.5インチと言うと，これまで，方式がとてもたくさんあり，互換性がとりづらいとする向きもあったようですが，最近の動きとしては，スーパーMZでも採用している，

●シャッター	手動	オート
●回転数(転送速度)	600回転/秒 (16μS/1ビット)	300回転/秒 (32μS/1ビット)
●トラック密度	135TPI	67.5TPI
●面	片　面	両　面

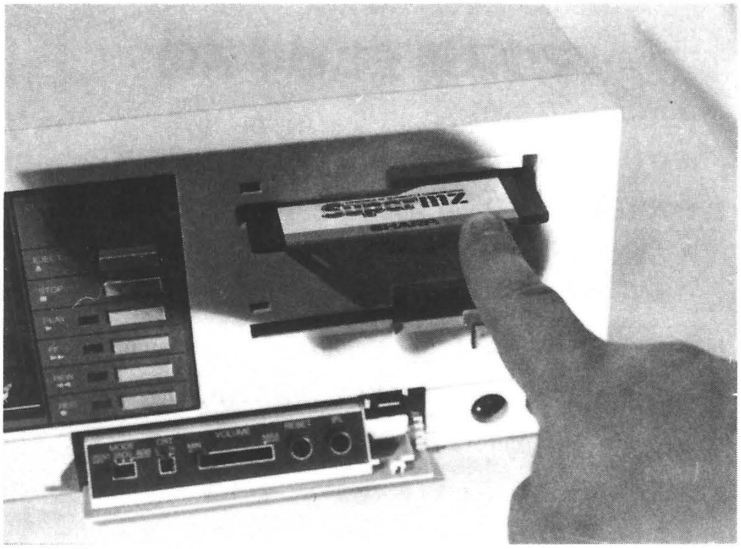
●片面あたり	35トラック		40トラック
	70トラック		80トラック
●アンフォーマット時容量(キロバイト)	218.75	250	437.5
	500	875	1000

という5インチ2DDと置換え可能なタイプにまとまりつつあるようです。このタイプは，今までには日本IBMのJX，日本電気のPC-9801U2などが採用しています。

なおマシン語を使うとこれらのパソコンとのデータの交換が可能になります。また，面を片面にした「1DD」はMSXや日本電気ホームエレクトロニクスのPC-6601SR(320Kバイト)に採用されていますが，これらとのデータ交換も可能です。ただし，これは，片面を使った場合だけで，もう片面はそれらのパソコンではアクセスできません。また，TPIを67.5にした「2D」(320Kバイト)は，富士通のFM-77の各モデルに採用されており，これらで書かれたディスクをスーパーMZで読出すことができます。ただ，その逆に，スーパーMZで書いたディスクを2Dのパソコンで読ませることはできません。2D→2DDだけの一方通行です。この辺が互換性が乏しいと言われて来た所以のようですが，スーパーMZにとっては，要するにデータが受け取れれば良いわけですから，問題ではないでしょう。そういう意味でも，つまり容量だけでなく，互換性についてでも2DDタイプを選択したのは正解だったと言えるでしょう。



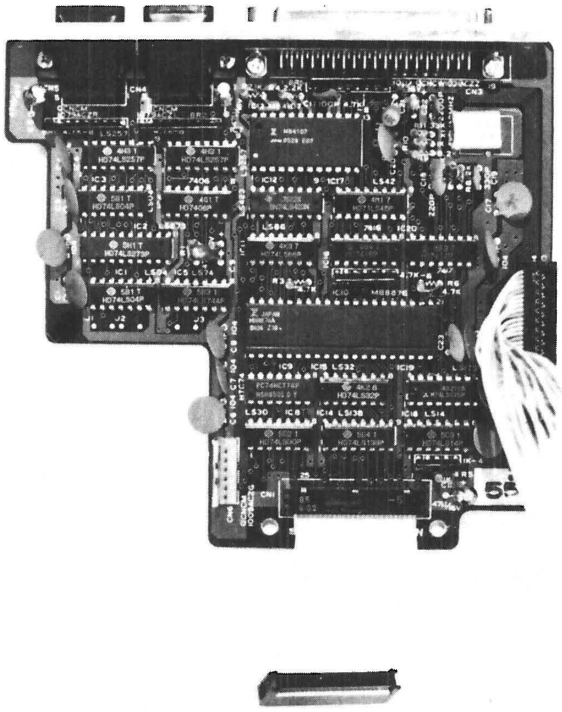
《写真2-23》5インチフロッピーディスクと比べてみた所



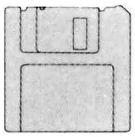
《写真2-24》3.5インチは操作もカンタン！

なお，スーパーMZでは，3.5インチ外にも，従来のMZシリーズ用の5インチ2Dのディスクドライブを3基目・4基目として使用することもできます。また，この場合，MZ-2000と80Bのモードでは，外部の5インチの方を1基目・2基目に，内蔵の3.5インチを3基目・4基目として使うこともできます。この機能はハードウエアでサポートしていますので，物理的なドライブ1か

ら起動しなければならない5インチのソフトも全く問題なく利用できるようになります。コンパチビリティをたいへん重視していることがわかります。

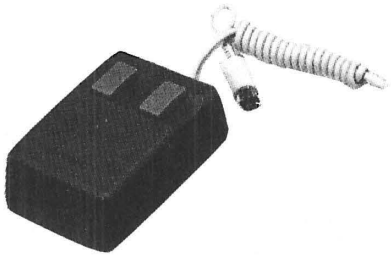


《写真2-25》フロッピーディスクインターフェース・ジョイスティックインターフェース部。ほぼ中央にあるのがFDC（フロッピーディスクコントローラ）のMB-8876A。機能はMB-8877と同じです。

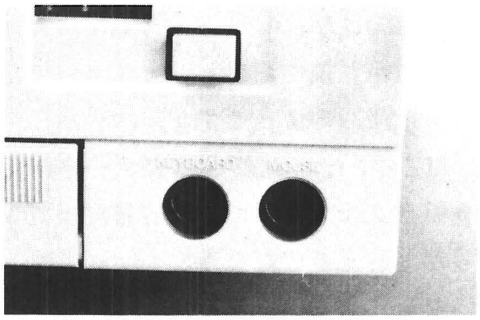


マウス端子も標準装備

腕やひじに無理な力が加わらずに，長時間座標の入力や表示選択ジョブ等に利用できるという実用的な座標入力装置，「**マウス**」は，16ビット機ではすっかり定着した感じですが，スーパーMZでもこれが利用できます。対応しているマウスは，MZ-5500/6500とX1 turbo用のものです。また，スーパーMZではBASICの方でマウスカーソルを自動的に表示・移動してくれますので，動きもスムースです。動きが純い！とイライラすることもないでしょう。256色のグラフィックも，スーパーMZのマウスを使った描画ソフトなら，誰にでも簡単に活用することができるでしょう。

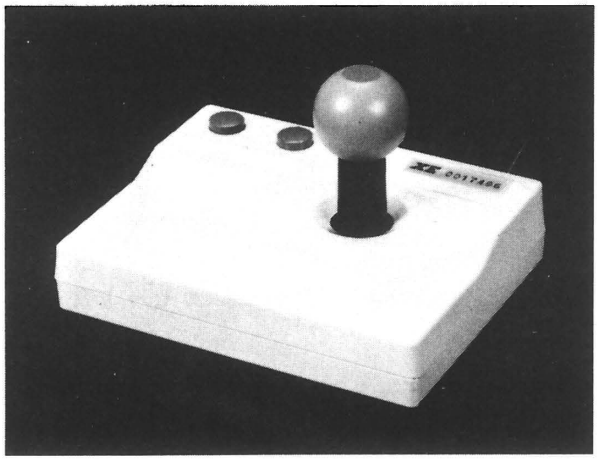


《写真2-26》マウス（MZ-1X10）
￥19,800



《写真2-27》本体前面のキーボード端子（左）とマウス端子（右），側面にもあり

ジョイステック端子も標準装備



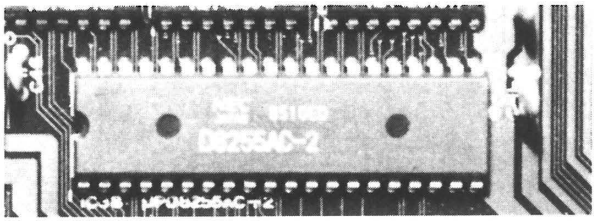
《写真2-28》市販のジョイスティック

スーパーMZは，ジョイスティックポートも持っています。もちろん，世界規格のアタリ社仕様で，２トリガタイプにも対応したものを，2個装備しています。X1用やMSX用，PC-6001系用のジョイスティックをそのまま使うことができます。

ジョイスティックを使えばゲームの迫力も違いますし，大切なキーボードをたたきすぎて壊してしまうこともありません。思いっ切り熱中することができます。

カセットデッキも標準装備

スーパーMZでは，カセットデッキを標準装備しています。これまで，シャープが発売して来たパソコンのほとんどは，フロッピーディスクかカセットデッキのどちらか一方を内蔵させて来ました。しかし，両方共装備しているのは，これが初めてです。では，フロッピーディスクがあるのに，なぜ，カセットデッキまで内蔵しているのでしょうか？



《写真2-29》主にカセットデッキを制御するのに使われる i8255(セカンドソース)



《写真2-30》モデム電話との接続により強力な力を発揮するカセットデッキ

まず，一つには，スーパーMZの前身である，「MZ-2000」や「MZ-80B」が，カセットデッキを標準装備していたからと考えられます。それらのパソコンでは，市販されていたソフトの大半がカセットで供給されていて，ディスク版のソフトはあまり多くありませんでした。そのため，スーパーMZのMZ-2000や80Bのモードを生かすには，カセットデッキが不可欠だったのだと思います。

しかし，この点は新たにスーパーMZのユーザーになろうとする人にはあまり関係のない話です。**本当の理由はもっと別にありそうです。**その前に，ちょっとクイズを出してみましょう。

まず，一般にカセットテープよりもフロッピーディスクの方が，すべての点で勝っていると考えられがちですが，これは本当でしょうか？

また，フロッピーディスクがあれば，カセットテープは必要ないと考えられがちですが，これは本当でしょうか？

答えはどちらもNo.です。今までのパソコンだけを見ている限りでは，確かにYesだったかも知れませんが，原点に返って考えれば明らかにNo.です。

理由を申し上げましょう。それは，

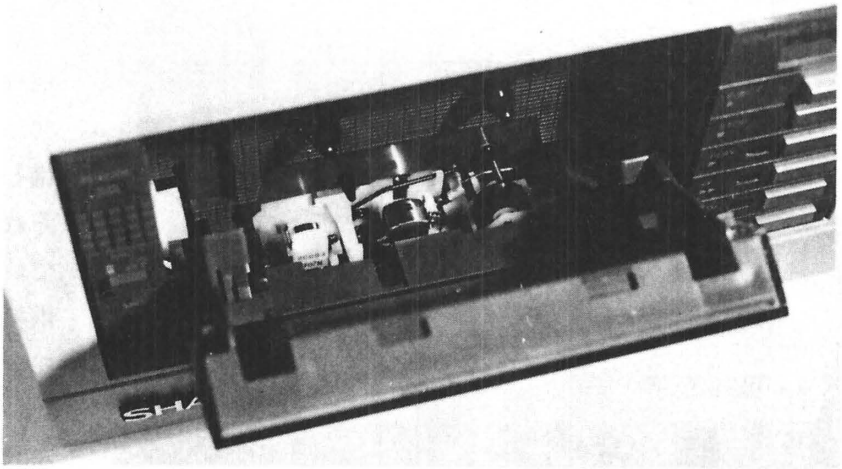
- フロッピーディスクはパソコン(コンピュータ)の信号しか記録できないが，カセットテープは音声や音楽も記録できる　そして，
- フロッピーディスクはパソコンでしか利用できないが，カセットテープはラジカセやオーディオカセットデッキにセットして使うことができる

の2点です。早い話がスーパーMZのカセットデッキは，コンピュータの信号以外に，**音声や音楽**も記録・再生できるようになっているということなのです。

スーパーMZではこのために，エレクトリックコンデンサマイクやライン入力端子，イヤホン端子を備え，ヘッドもLL用の消去ヘッドと，ステレオ(2チャンネル)の録音・再生ヘッドを持っています。そして，カセットテープを常に2チャンネルで使い，1チャンネル(左)はコンピュータ専用，もう1チャンネル(右)はアナログ音声専用になっています。

このため，コンピュータ側，アナログ側，それぞれを一方だけ録音状態にしたり，両方共録音状態にしたり，それを指定時間だけ行ったりすることができます。マイクとラインの切換えは，パソコン側から制御できます。また，コンピュータ側に頭出し信号を記録しておき，その信号をもとに，APSSを行ってランダムに再生することもできます。例えば，人間の声とコンピュータ画面による，「BASICレッスン」などというソフトが出て来ることも考えられます。フロッピーディスクだけでは絶対できない，優れた応用方法の一つだと思います。

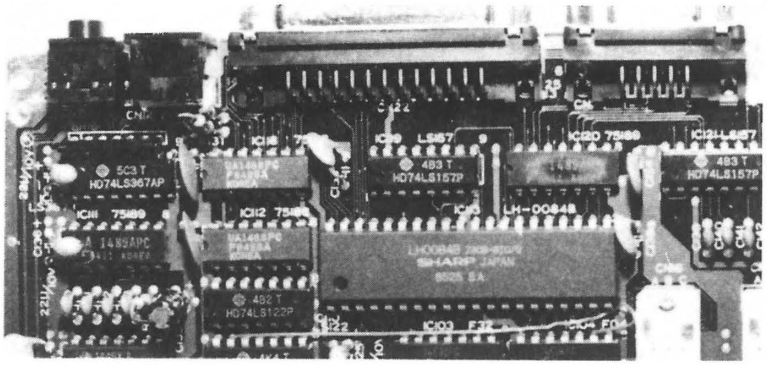
カセットデッキ自体，すべての機能がコンピュータから操作可能な，フルロジカルコントロールタイプなので，ほかにも応用はいろいろ考えられるでしょう。



《写真2-31》ステレオ録再ヘッドとLL用消去ヘッドを使用し，アナログ部／デジタル部が独立して制御可能

RS-232Cは2チャンネル装備

ちょうどスーパーMZが発表された少し前から,全国各地で電話回線を利用した通信システムが始まり，パソコンによるデータ通信がたいへん注目されるようになって久しいですが，スーパーMZでは，これに使う「RS-232C」ポートを２個，標準装備しています。これまでのパソコンでは，全く装備していなかったり，それどころかオプションでも使えなかったり，あるいは１ポートのみ装備しているものが多くありましたが,スーパーMZでは先進の２ポート装備です(ただし，２ポート目はLSIのポートをマウスと共用していますので，２ポート目とマウスを同時に利用することはできません)。



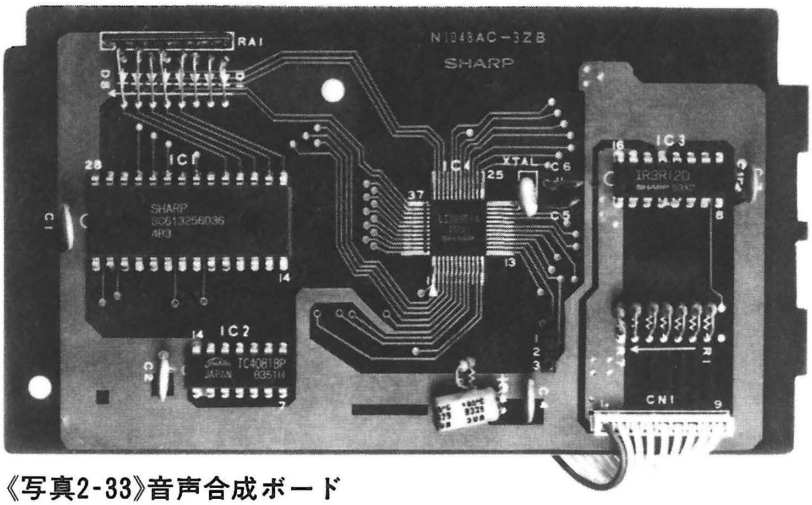
《写真2-32》高性能シリアルインターフェース Z80B SIO は RS-232C コネクタ(中央がAポート，右がBポート)のすぐそばに位置している

ボーレイトは75ボー～19200ボーまで対応しており，高速転送も可能です。また，コントローラにi8251ではなく,高性能なZ80B SIO/0を使っており，一般に使われている８ビット長の時にもパリティが付けられるなど，信頼性にも重点を置いています。単に2個分のRS-232Cがあるのではないです。

音声合成もサポート(オプション)

今までMZ-1500クラスの低価格のパソコンでこれが利用できることはありましたが，上級８ビット機や16ビット機では，オプションでも使うことができませんでした。

しかし,スーパーMZでは(オプションですが)，MZ-1500用の**音声合成ボード「MZ-1M08」**が利用できます。「ア」～「ン」の50音や，「エイ」～「ゼット」のアルファベット，「ゼロ」～「キュウ」の数字の棒読み,「ニセンゴヒャクニ

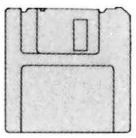


《写真2-33》音声合成ボード「MZ-1M08」 ￥10,000

ジュウイチ」等の数値読み,「YカNノキーヲオシテクダサイ」や「セイカイデス」などの34個の固定音,その他を発声させることができます(**第2-9表**)。

《第2-9表》オプションの音声合成ボードで予め登録されている固定音(これらの他にア～ンの50音等も発声可能)

番号	発声内容	番号	発声内容
0	まちがいです	17	文法上の誤りです
1	正解です	18	私はMZです
2	OK	19	点
3	もう1回チャレンジしますか	20	たす
4	よくできました	21	ひく
5	笑い声	22	イテテ
6	ゲーム・スタート	23	イテッ!
7	ゲーム・オーバー	24	自動車(発車・走行・停止)
8	アウト	25	自動車(発車)
9	GO	26	自動車(走行)
10	命中	27	自動車(停止)
11	お好きなキーを押して下さい	28	メロディ(葬送行進曲)
12	YかN	29	メロディ(デキシーランド)
13	最高得点	30	メロディ(蛍の光)
14	ボーナス点	31	メロディ(おおスザンナ)
15	あなたの番です	32	メロディ(故郷の人々)
16	私の勝です	33	メロディ(村のかじ屋)



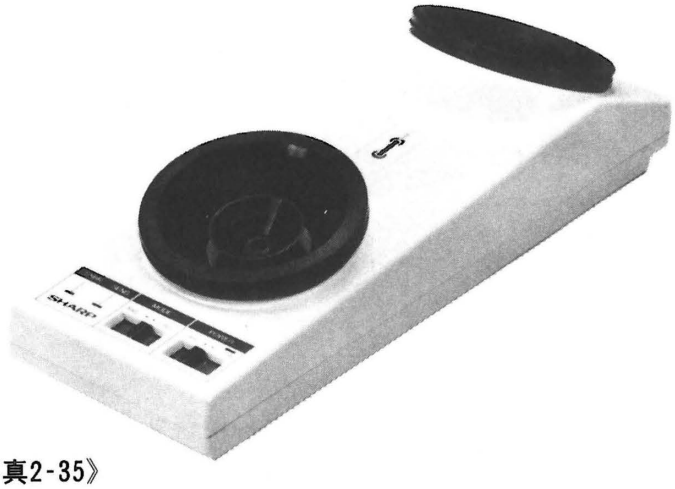
電話機もサポート(オプション)

今までの説明では,スーパーMZの,ごく普通のパソコンとしての性能を見て来たわけですが,この項目だけは,スーパーMZの,普通でない,非常にユニークなところです。ちょうどX1シリーズのスーパーインポーズのように,後から発表される他社機にまで,大きな影響を与えることになるかも知れません。

写真2-34は,スーパーMZと同時に発表された**電話機(モデムホン)「MZ-1X19」**です。このモデムホン



《写真2-34》
通信機能に抜群の能力をもつモデムホン「MZ-1X19」
￥98,000



《写真2-35》
モデムホンまではちょっと……という方にお勧めしたい
音響カプラ「MZ-1X11」￥44,800

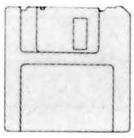
とスーパーMZは,前述のRS-232Cのうちの1ポートで接続し，このシステムの核になります。まず，単純にこれだけで，電話回線を使った，データ通信ができます。パソコン同士をつないでも良いですし，大型コンピュータのターミナルとして使っても良いでしょう。音響カプラと普通の電話機を使った場合は転送速度300ボーでしか使えませんでしたが，スーパーMZ＋MZ-1X19では,1200ボーまで可能です。4倍のスピードで通信できるのです。

次に，やはりスーパーMZとMZ-1X19で，**オートダイヤル機能**が使えます。オートダイヤルというのは，**パソコンが電話機のダイヤルを回してくれる機能**のことで，パソコンから電話したい相手，または通信を行いたい相手を指示するだけで，そこに電話をかけてくれる機能です。これについては第4章で詳しく触れたいと思います。

そして，オプションの**ボイスコミュニケーションインターフェース**というものを使うと，内蔵のカセットデッキを利用した**留守番電話**や，その逆に相手に声のメッセージを送る。**ボイスメール機能**も利用できるようになります。この場合，電話がかかると，パソコンの電源が入るようにできますから電気代を気にする必要はありません。

また，このインターフェースを付けた状態では，カセットデッキのアナログ音声以外に音声合成ボードで作った声，サウンドLSI(FM音源)で作った音等も相手に送ることができます。

RS-232Cを2ポート用意したのも，電話機とつないだままでも，もう1ポートが自由に使えるようにということかも知れませんし，カセットデッキを標準装備しているのも，音声合成をできるようにしたのも，みな，この電話によるコミュニケーション機能を意識してのことだと思います。それだけに，今後の動向に注目して，見守っていきたいと思う機能です。



多種のプリンタに対応

スーパーMZでは，セントロニクス準拠プリンタインターフェースを持っていますが，従来の非セントロニクスのMZシリーズのプリンタや,他社のいくつかの代表的なプリンタも使用できるユーティリティが付属しており，これを選択するだけで，それらのプリンタでリストをとることはもちろん，画面のハードコピーや，漢字の印字まで利用することができるようになっています。しかも，漢字プリンタでなくとも，漢字の印字が可能です。そしてさらに，それ以外のプリンタでも，ビットイメージ印字ができるもの(ボールペン型のプロッタプリンタ以外はほとんどすべて)

なら，同ユーティリティのユーザー定義モードを使って，コントロールコードがどのような仕様になっているかを質問に答えるだけで，そのプリンタでリストをとったり画面のハードコピーや漢字の印字もできるようになります。こちらの場合も，漢字プリンタでないならスーパーMZ本体内に入っている漢字ROMのフォントが生かされて，漢字が印字できるようになっています。もちろん漢字ROM内蔵のプリンタではコードのまま送られますから，24ドット日本語プリンタなら，その美しいフオントで印字が可能です。

第2-10表に，スーパーMZが初めから対応しているプリンタ，つまり単に選択するだけで使えるようになるプリンタの一覧表をまとめておきます。これらのプリンタとコンパチブルというプリンタも，各々のモードで利用可能です。他社のパソコンには見られない，なかなかの親切設計ぶりがうかがえます。

なお，スーパーMZは，プリンタからのデータ要求によって，割込みを発生させることができるハードウエアになっています。BASICでは，選択によって，これを利用したプリンタスプーラを実現することができます。



《写真2-36》
静かな印字でパーソナルユースには持ってこいの80桁24ドットサーマル・熱転写漢字プリンタ「MZ 1 P17」￥79,800



《写真2-37》
高級ビジネス向き 132 桁24ドットドットインパクト漢字プリンタ「MZ- 1 P11A」￥345,000

《第2-10表》スーパーMZで選択するだけで使用することのできるプリンタ(CZ-515E と CZ- 8 PP 2 はプロッタプリンタなのでハードコピーはできないのと，これらに漢字 ROM が内蔵されていない場合は本体から漢字フォントを送って印字させることはできないので注意して下さい)

MZ-80P4B	MZ-1P11/A	CZ-8PP2
MZ-80BP5	MZ-1P17	PC-PR101
MZ-80P6	CE-515P	PC-PR201
MZ-1P03	CZ-800P	PC-PR406
MZ-1P06	CZ-80PK	SP-80(ESS/P)
MZ-1P07/A	CZ-8PD2	RP-80F/T2K
MZ-1P10/A	CZ-8PK2	UP-130K

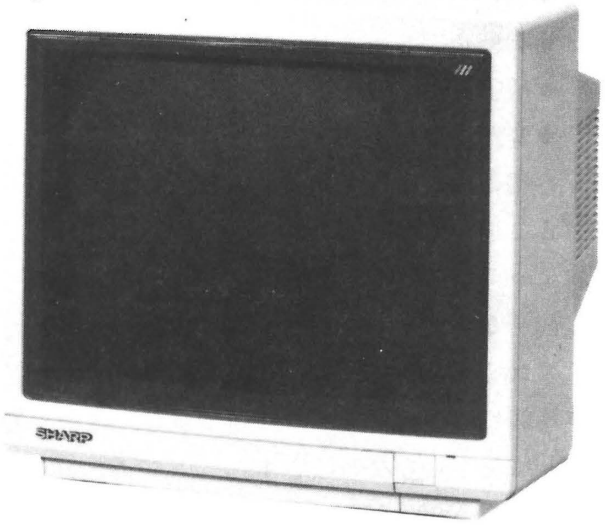


《第2-10図》MZ-1P17の印字サンプル

あらゆるRGBディスプレイに対応

ゲーム用以外のパソコンのディスプレイ装置としては，現在，RGBタイプがそのほとんどを占めていますが，スーパーMZでは，これらへの対応も万全です。

まず，400ライン系（水平同期周波数24.5KHz）と200ライン系（水平同期周波数15.75KHz）とがありますが，両方に対応しています。もちろん他のパソコンと同じように200ライン系では縦400ドットのグラフィックや縦25行・20行の漢字表字はできませんが，前にも触れたように，12行モードを使えば漢字V-RAMでの高速漢字表示はできます。なお，MZ-2000/80Bの

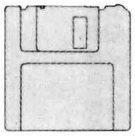


《写真2-38》400ラインのリニアRGBディスプレイ「MZ-1D22」￥108,000

モードでは，400ライン・200ラインのどちらでも，本物と同じ表示ができます。これらのパソコンには，縦400ドット等はないからです。

そして，同じ400ラインまたは200ラインでも，普通に8色しか表示できないデジタルRGBタイプや，それに輝度信号を加えて16色表示のできる(デジタル)RGBIタイプ，そして無制限の色が出せるリニア(アナログ)RGBタイプなどが普及していますが，スーパーMZでは，これらのいずれにも対応しており，ディスプレイの持っている性能を最大限に生かすことができるようになっています。デジタルRGBなら8色，RGBIなら16色，リニア(アナログ)RGBならそのままでは256色，パレットボードを使えば4096色まで利用することができます。もちろん，ニューメディア対応テレビの21ピンアナログRGB端子にも，ケーブルだけで接続することができます。また，外部同期機能を備えていますから，将来的にはスーパーインポーズも楽しめるようになると思います。

なお，スーパーMZは，RGB端子以外にも，モノクロディスプレイ用のコンポジット出力端子も持っています。



拡張I/Oポート

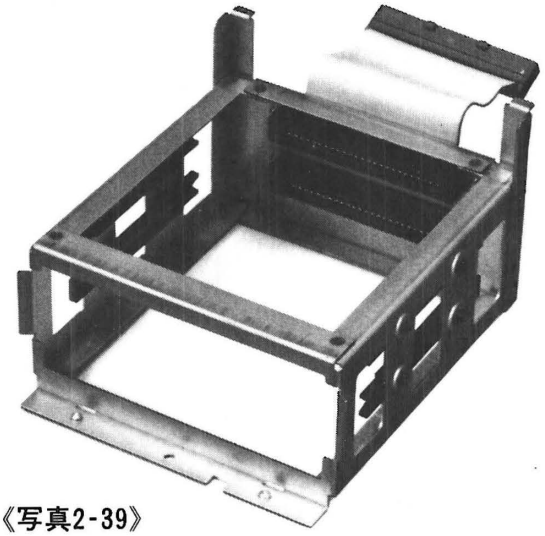
スーパーMZは，汎用の拡張ポートはオプションになっています。**写真2-39**は，本体内に取り付けられる拡張I/Oポート「**MZ-1U09**」ですが，これはMZ-80B・2000等用の周辺装置を使うためのものと考えた方が良く，MZ-2500モードでは利用する機会はほとんどないと思います。また，将来的にもこれを必要とする周辺装置はあまり出ないと思われますので，本体のコストを下げるためにもあえてオプションにしているのでしょう。

それというのも，「MZ-1U09」がなくとも，これまでに触れて来た，ほとんどのオプション群，

- 128Kバイト増設RAM(メインメモリ用)
- 64Kバイト増設RAM(グラフィック用)
- 4096色パレットカード
- 辞書ROM
- 音声合成ボード

は，専用のスロットを持っており，そのまま取り付けることができるからです。ですから，逆に考え方によってはスーパーMZは5スロット設計であるとも言えるのです。しかも，電話機の所で触れたボイスコミュニケーションインターフェースも，本体外部に取り付けるタイプですから，拡張I/Oポートは全くいらないのです。

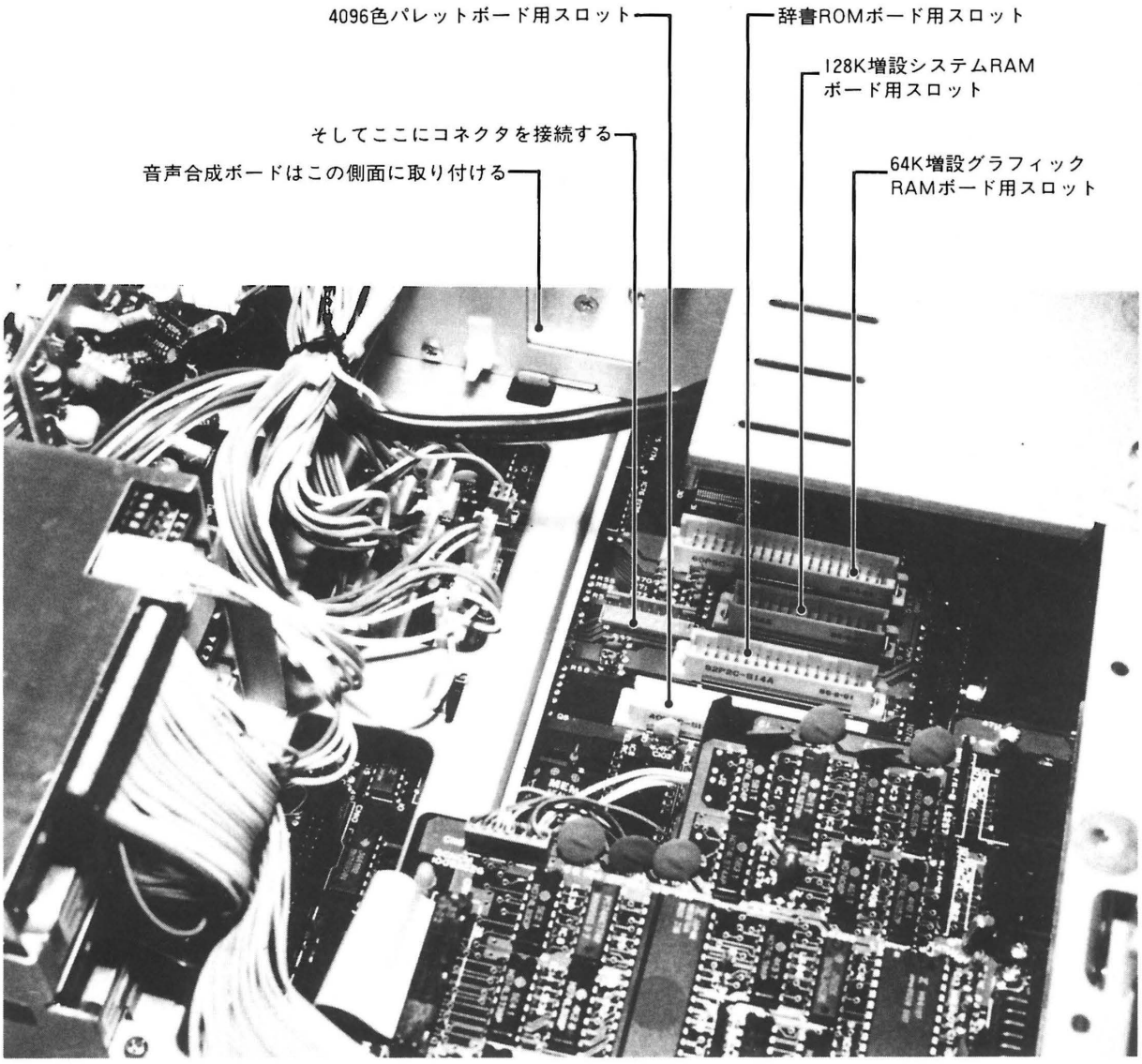
一方MZ-2000と80Bのモードでは，それぞれのために発売されて来たカードが使えますが，プリンタ，フロッピーディスク，RS-232Cの各インター



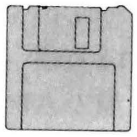
《写真2-39》
拡張I／Oポート「MZ-1U09」￥9,000

フェースについてはコンパチブルなものを内蔵していますので，不要です。しかしMZ-2000用の漢字ROMは使用できます。スーパーMZが内蔵している漢字ROMは，テキストV-RAMと直接つながっているなどの理由でMZ-2000用のものとアクセス方法が違うからです。あとは，MZ-2000用クイックディスクインターフェースも付けられますが，スーパーMZは3.5インチマイクロフロッピーディスクを内蔵しているのですから，新たに購入する必要はないでしょう。ただ，MZ-80B・MZ-2000用のGPIBカードだけはスーパーMZが代わるものを持っているわけではないので，利用する価値がある人はいるのではないかと思います。

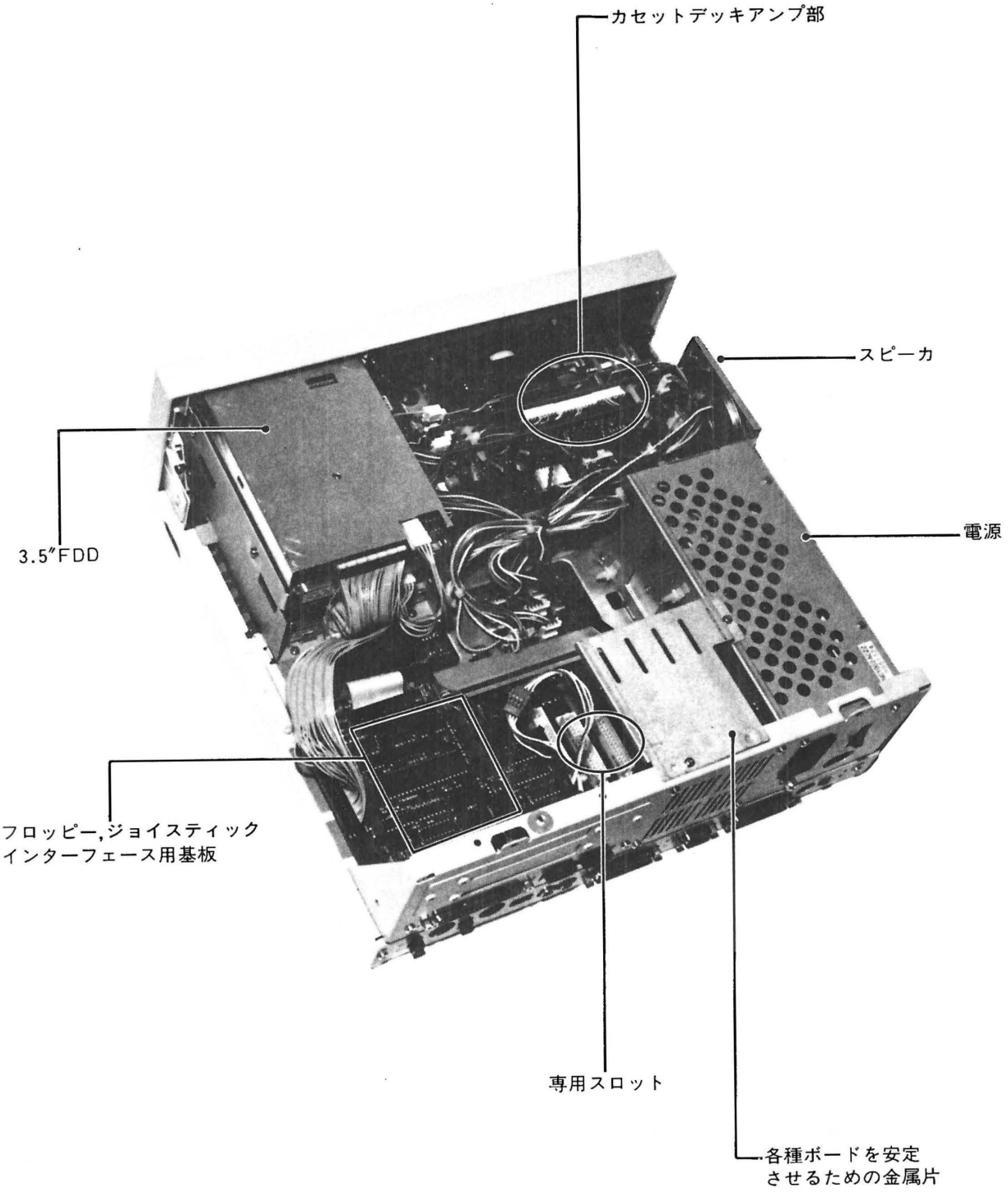
なお，MZ-2000・80Bのモードでも，I/Oポートなしで取り付けられたグラフィックRAM等は，スーパーMZには付けられません。しかし，MZ-2000モードではグラフィックRAM 3 ページ分，MZ-80Bのモードでも 2 ページ分(つまり全部)がつながれた状態として動作しますので，付けられないというより，付ける必要がないと言った方が良いでしょう。



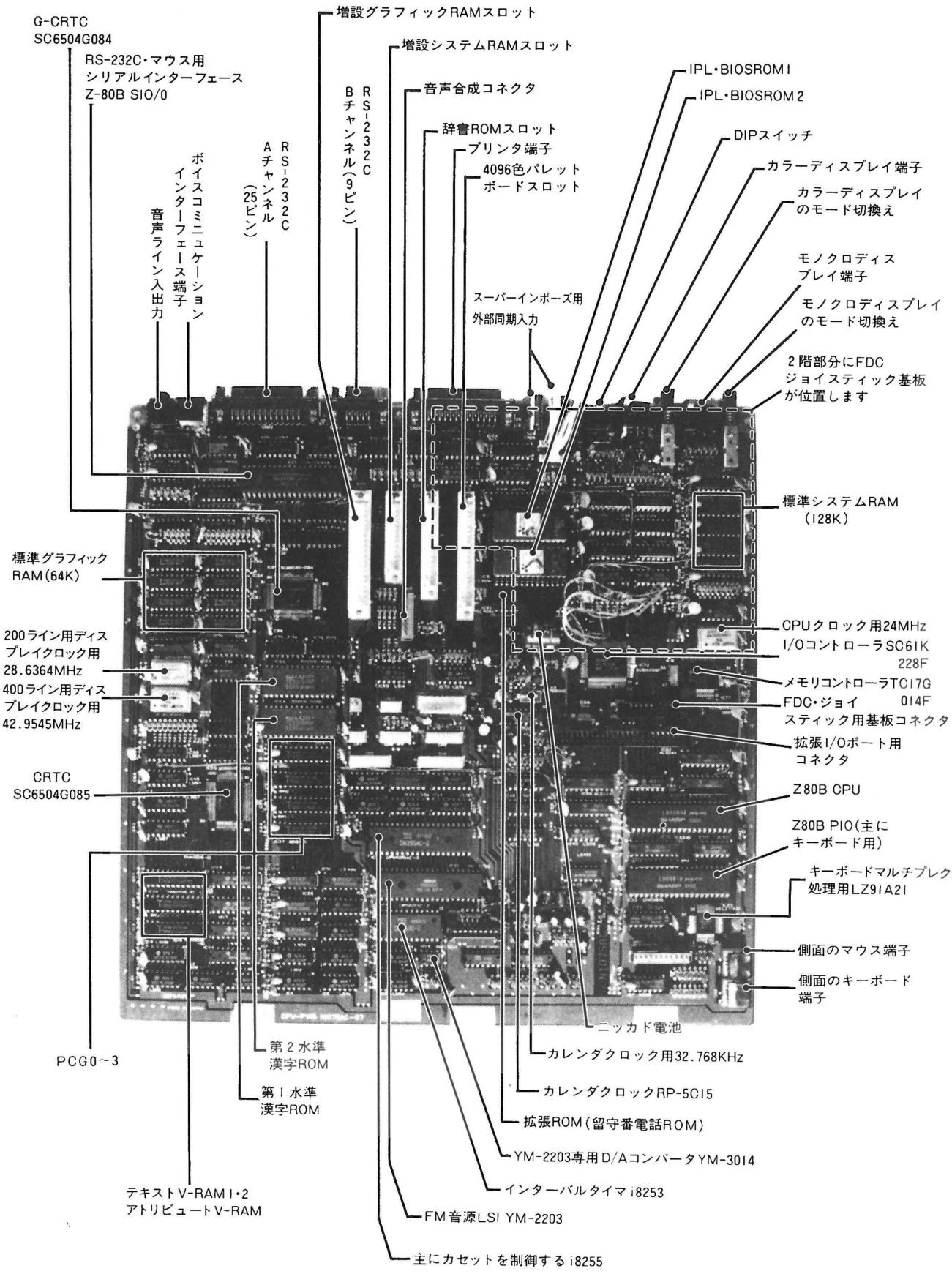
《写真2-40》オプションボード専用スロット



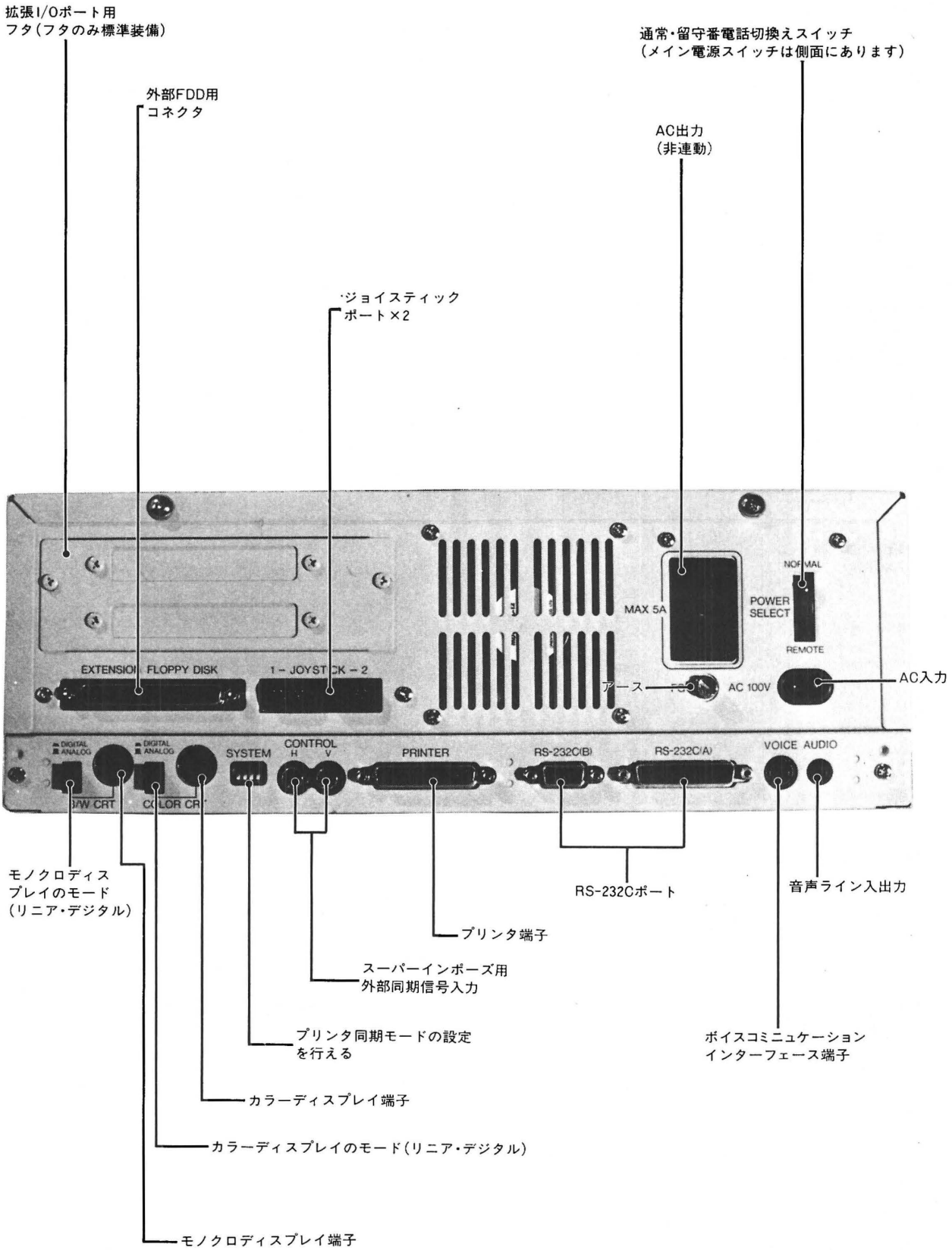
スーパーMZ大解剖



《写真2-41》スーパーMZの本体上ブタを外した所



《写真2-42》メイン基板の様子



《写真2-43》背面のコネクタ部

《第2－11表》スーパーMZの基本仕様

CPU		Z80B(クロック6MHz・4MHz切換え)
ROM	IPL・BIOS	32Kバイト
	キャラクタジェネレータ	JIS第一水準　漢字フォント　128Kバイト(一般文字・従来MZシリーズフォント等含む) JIS第二水準　漢字フォント　128Kバイト
RAM	メイン	128Kバイト(256Kバイトまで拡張可能)
	文字画面	1バイト系文字・2バイト系文字前半 2Kバイト　アトリビュート　2Kバイト 2バイト系文字(漢字)後半　2Kバイト
	PCG・外字	PCB-0(モノクロ)　2Kバイト　PCG-2(赤プレーン)　2Kバイト PCG-1(青プレーン)　2Kバイト　PCG-3(緑プレーン)　2Kバイト
	グラフィック	64Kバイト(128Kバイトまで拡張可能)
表示能力	文字	80×25　80×20　40×25　40×20　80×12　40×12　(40桁モードは2ページ利用・合成可)
	漢字	40×25　40×20　20×25　20×20　40×12　20×12 (文字の欄のそれぞれに対応。縦20行以上は400ラインディスプレイ専用)
	グラフィック	(グラフィックRAM64Kバイト時)　(グラフィックRAM128Kバイト時) 640×400×4色×1ページ　640×400×16色×1ページ 640×200×16色×1ページ　640×200×16色×2ページ 320×200×16色×2ページ　320×200×16色×4ページ 320×200×256色×1ページ　320×200×256色×2ページ (縦400ドットは400ラインディスプレイの時)
パレット機能		16色中4色または15色を選択。　オプションで4096色中3色または15色を選択
文字・グラフィック合成		可能。プライオリティ(優先順位)設定可
プライオリティ機能		文字1面とグラフィック16色間で設定可能　また，グラフィック2ページ間で設定可能
PCG・外字		最大モノクロ1024種類。　またはモノクロ256種類＋カラー256種類等 最大256種類　} 兼用
フロッピーディスクインターフェース		3.5インチ2DD(両面倍密度倍トラック)640K　5インチ2D(両面倍密度)280K～320K なお，インターフェースの能力としては1S(片面単密度)・2S(両面単密度)にも対応。 MB-8876A(MB-8877Aの負論理)使用
フロッピーディスク		3.5インチ両面倍密度(640Kバイト)ドライブモデル20は1基，モデル30は2基装備
カセットデッキ		フルロジカルコントロール(電磁メカ)タイプ内蔵。2トラック独立ヘッド。アナログ記録・再生可能。PWM方式(1バイト＝9ビット)。2000ボー
プリンタインターフェース		セントロニクス社準拠。旧MZタイプにも対応
ジョイスティックコネクタ		2個内蔵。アタリ社仕様準拠。2トリガ
RS-232Cポート		2個(25ピンコネクタ，9ピンコネクタ各1個)内蔵　(Z80A SIO使用)
マウスインターフェース		1回路内蔵(5ピンコネクタ。コネクタは前部側部合わせて2個(並列接続))
サウンド出力		8オクターブ・6和音(BASICと並列処理可能)と，1効果音。うち，3和音はFM音源。 出力コネクタ付き(YM-2203使用)
時計		年・月・日・曜日・時・分・秒。停電等でもバッテリバックアップ可能(RP-5C15使用)
計時		0.0秒～6553.5秒まで0.1秒単位(i8253使用)
映像出力		RGB・RGBI・リニア(アナログ)RGB・コンポジット(リニア・デジタル)
電源		AC 100V 50Hz/60Hz 75W
外形寸法		本体…350(幅)×340(奥)×125(高)mm。　キーボード…410(幅)×195(奥)×33(高)mm
重量		本体…モデル20は8.3kg，モデル30は9.0kg。　キーボード…1.4kg

BASIC-M25の隠し命令 ❶

BASIC-M25を何げなく使っていて気がついたのですが,マニュアルには書いてないのに実際には使用できる命令がいくつか存在していることがわかりました。文脈からは少し外れますが，以下現在までにわかっているものを紹介させて頂きます。

●CLR

変数をクリアするステートメントです。ただのCLEAR(アドレスを指定しないCLEAR)と同じ働きをします。

●FRE

マニュアルでは引き数がついたFRE(0)～FRE(4)だけが述べられていますが,引き数を省略することもできます。単にフリーエリアの総量を知りたいのなら，

PRINT FRE(0)

としなくても，

PRINT FRE　　(または　?FRE)

だけでOKです。

●KEY LLIST

ファンクションキーの定義内容を，プリンタに出力します。KEY LISTのプリンタ版です。

●SWAP ファイル名

変数値を交換するSWAPではなく，プログラム単位でのGOSUBに相当するようなステートメントです。BASIC-M25にはプログラム単位でのGOTOに相当するCHAIN文がありますが，まずメインメニューがあって，そこからサブメニューに移って…といった形式のプログラムでは，むしろCHAINよりもこのSWAP文の方がはるかに便利です。全体が10個のジョブからできていたとすると，一つのプログラムあたりの大きさは平均して$\frac{1}{10}$になりますので，非常に大規模なプログラム(例えば，100K，200K…)も難なく実行できることになるでしょう。

SWAP文はプログラム中でのみ使い，例えば100 SWAP "SUBMENU"

とすると，100行ではまず，このプログラム本体をディスクの専用エリアにSAVEし(ファイル名はつきません)，次に"SUBMENU"というプログラムをロード・実行します。実行の際には変数の値等は変化しません。そして，必要な処理を行った後，END文が来るか，プログラムの最後まで実行し終わると，"SUBMENU"の実行を終了し，専用エリアから元のプログラムを呼び戻し，SWAP文の次のステートメントから実行を続けるというわけです。

なお，SWAP文を使う時は，ディスクを書込み禁止状態にしてはいけません。それと，SWAPのネスティングレベルは1までです。つまり，SWAPしたプログラム中で再びSWAPを使うことはできませんから注意して下さい(メインプログラムに戻った後なら，もう一度SWAP文を使用しても構いません)。

●CGPAT$( )=

本来CGPAT$というのは，キャラクタジェネレータのフォントを返す関数ですが，このように逆に代入するような形で使用すると，PCGへのフォント定義命令として利用できるようになります。ただし，この目的には元々DEF CHR$がありますので，システム変数として統一された形でアクセスしたい場合にのみ利用すると良いでしょう。なおパラメータその他は，DEF CHR$と全く同じです。

●PRINT [ ]

メッセージか何かを表示する際，その一部分だけを別の色で表示したいということは良くあります。その場合，COLOR文とPRINT文とが交互に現われるようなプログラムとなり，煩雑になります。このステートメントは，それを改善するものなのです。

例えば，COLOR 7の状態(通常の白の状態)で

PRINT [6] "ABC"

としますと，ABCと，黄色(=COLOR 6)で表示されます。そして次の[ ]のないPRINT文等からは，元の色(白)での表示に戻ります。

また，一つのPRINT文中でも，次々と色を変えて表示を行うこともできます。また,途中から[ ]指定を行うこともできます。例えば，COLOR 7の状態から

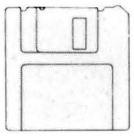
PRINT "ABC";[1]"DEF";[2]"GHI"

とすると，ABCは白で，DEFは青で，GHIは赤で表示できます。また，[ ]の中の色指定には，変数や式を使うこともできます。かなりスマートなプログラムが書けるようになるでしょう。



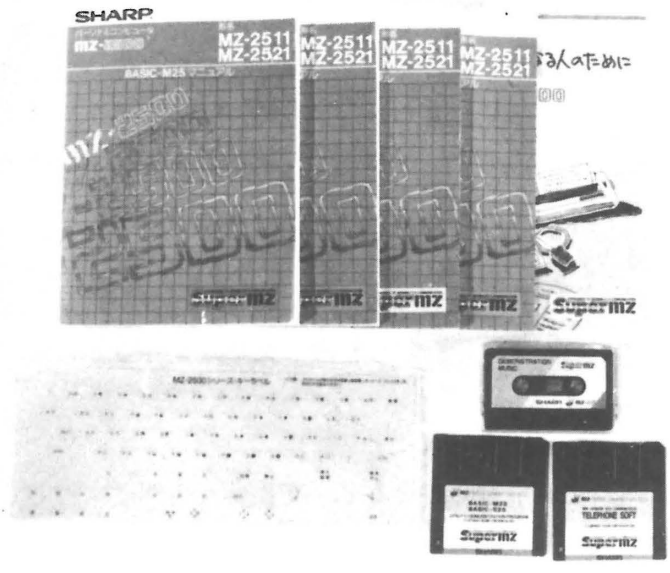
第3章 ソフトウエア解剖

いくら優れたハードウエアでも，ソフトがなければただの箱！そこでこの章では，スーパーMZに添付している「BASIC」を中心に，本機のソフトウエア面の魅力を探ってみたいと思います。



史上最強のBASIC

スーパーMZは，ハードウエアが強力なだけではありません。ソフトウエアもたいへん強力です。まずは，史上最強と言われる，BASIC，『BASIC -M25』から見ていきましょう。このBASICは，シャープ自社開発ですが，命令体系はマイクロソフトBASIC に準拠しています。



《写真３－１》スーパーＭＺの付属品

●すべての演算が16桁で可能

パソコンも一応コンピュータ，つまり計算機ですから，最も基本となるのは，やはり計算です。スーパーＭＺでは，整数型，単精度実数型，倍精度実数型の３タイプのデータがあり，整数4.5桁(－32768～＋32767)，単精度実数9.6桁(±１Ｅ±38程度)，倍精度実数16.8桁(±１Ｄ±38程度)の精度で計算し，整数…そのまま，単精度実数…８桁，倍精度実数…16

桁で表示します。単精度は普通，7.2桁で計算し，6桁表示とするBASICが多いのですが，スーパーMZの場合，従来のMZのBASICが単精度・倍精度の区別がなくこの精度のため，上位コンパチとしたようです。

スーパーMZでは，四則演算で単精度、倍精度とも計算できるほか，数値計算用の関数(sin,cos等）も単精度，倍精度の両方で計算できます。つまり，こういった関数まで，16桁の精度で計算できるわけです。しかし今や，スーパーMZと競合するようなパソコン，特に80系のCPUを採用したものでは常識になっていますので，スピードが抜群に速いことを除けば，スーパーMZの方が機能的に優れているというわけではありません。まずはコテ調べです。

●ラベルを使用可能

プログラムが一画面に収まり切れない長さになると，どうしても見通しが効かなくなります。現在のパソコンは1画面24～25行程度ですので，ちょっとしたプログラフでもすぐに一覧できなくなってしまいます。

すると，GOSUB　で呼び出している行番号が，いったいどのような働きを持ったサブルーチンなのかを知るには，実際にその行をリストしてみるしかなくなります。単なる数字である行番号と，そのサブルーチンの働きとは，関連づけがないため，これを結びつけて覚えるのは，大変なことです。しかも，例え覚えることができても，リナンバーですぐに行番号は変わってしまいますし，プログラム毎に覚え直すのも厄介です。

そこで登場するのが「**ラベル**」です。ラベルは行番号と必要に応じて置換えて

```
GOSUB 1000
```

の代わりに

```
GOSUB ＊GAMENHYOJI
```

のようにできるようにしたものです。

しかし，これも競合マシンの中では常識に近いでしょう。もう何年も前からできるパソコンがいくつもあります。しかし，スーパーMZの場合には，ちょっと次元が違います。単なるラベルだけではないのです。これまで，＊の後に書けるのはアルファベッドだけでしたが，スーパーMZでは何と漢字が使えるのです。

つまり，

```
GOSUB　＊画面表示
```

とできるのです。これなら格段に見やすくなるでしょう。何と言っても日本人には日本語が一番なのです！

●漢字変数名使用可能

ラベルが漢字対応となると，変数名も当然漢字対応です。

```
INPUT NAMAE$
```

とするより，

```
INPUT　名前$
```

の方が，

```
KINGAKU＝TANKA＊KOSUU
```

とするより

金額＝単価＊個数

の方がはるかに見やすく，ミスも少なくなります。漢字ラベルや，ハード編で説明したPRINT〝漢字〟ができる点も含め，スーパーMZのBASICは，完全日本語対応なのです。恐らく，パソコン上で動くBASIC言語としては，これが初めてではないでしょうか。*1

なお，スーパーMZのBASICは，ARUNやRUNAのように予約語を途中に含んでいても，予約語そのものでない語なら，変数名やラベルとして使えます。変数名は200文字以内ならそのすべてを識別します。

●構造化プログラミングをさらに徹底

長いプログラムをいかに見やすくするか，という点は，長年の課題のようなものでもありますが，以前から構造化プログラミングを意識したBASICは多くありました。

```
10  FOR  A＝0  TO  100
20    PRINT  A
30  NEXT
```

20行のように，字下げを行ったり，プログラムを見づらくするGOTO文の使用を抑えるため，WHILE～WENDループを導入したりして来ました。REPEAT～UNTILループを持ったBASICも，いくつかありました。

スーパーMZでも類にもれず，これらの機能をすべて導入しています。また字下げは，それによってプログラムのサイズが増えたり，実行速度が遅くなることのないよう，工夫されています。スペースの数だけを1バイトで保持していますので，10文字分字下げを行っても，20文字分行っても，メモリの消費量は全く変わりません。

ところで，スーパーMZでは，既存のこれらの命令に加えて，極めて画期的な構造化プログラミング機能を導入しました。大型コンピュータのFORTRAN-77でも採用されている，**ブロックIF文**です。簡単な例を示します。

```
100   ＩＦ   A＞＝1000   THEN
110     B＝4
120     GOSUB    ＊印字
130   ELSE  ＩＦ   A＞＝100    THEN
140     B＝3
150     GOSUB    ＊印字
160   ELSE  ＩＦ   A＞＝10   THEN
170     B＝2
180     GOSUB    ＊印字
190   ELSE
200     B＝1
210     GOSUB    ＊印字
220   END  IF
```

※1　X 1 turboはPRINT〝漢字〟, GOSUB〝画面表示〟, INPUT［名前］$のようにできるので，これに迫る能力を持つが，変数名は［　］で囲まないと漢字が使えないので，わずかに劣る。

あまり分かりやすい例ではありませんが，要するに

ＩＦ文の作用が何行にも渡って書ける

ということです。今までのBASICではIF文はその行限りの作用でしたので，その後の処理が長くなる場合にはどうしてもGOTO文を使う必要がありました。これならどんなに長くなっても大丈夫ですから，その気になればGOTO文全くなしのプログラムだって書けるかも知れません。

●ユーザー座標系も当然装備（WINDOW,VIEW）

グラフィック画面のドット数は、パソコンや画面モードによってまちまちですが，たとえドット数が違っていても，同じパラメータでグラフィックが描けると便利です。また，同一画面上でも，原点やスケールを変えるだけで，同じ絵をいくつもの場所に，違った大きさで描くことができると便利です。ユーザー座標系の存在理由はざっとこんなものですが，やはり，今や常識にな

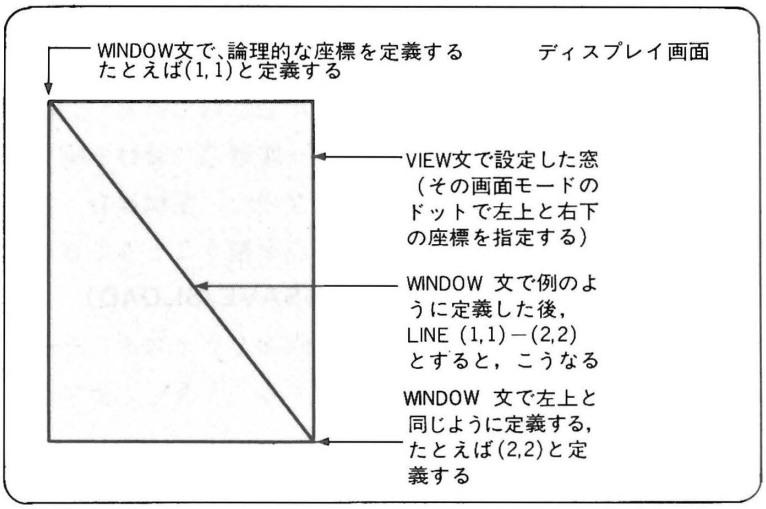
《サンプル３－１》ブロックＩＦ文を使用したプログラム例

```
 10        'ディスクダンプ・エディットユーティリティ
 20        dim A$(1)
 30        init "CRT:,,,0":view@ (1,1)-(0,0):LPT=0
 40        console 0,24:print [5] "   +++ Disk edit utility for MZ-2500 +++"
 50        console 1,23:DV$="1:":REC=-1:key 0,"r"+DV$+chr$(5)
 60 *LOOP:input "",A$
 70        if left$(A$,1)="r" then                          '読み出し・表示
 80          A$=mid$(A$,2):A=instr(A$,":")
 90          if A then DV$=left$(A$,A):A$=mid$(A$,A+1)      'デバイス名登録
100          key 0,"r"+DV$                                  'キーバッファにセット
110          if left$(A$,1)="R" or left$(A$,1)="r" then key 0,chr$(5,4):end
120          if len(A$) then REC=val(A$) else REC=REC+1     'ただのリターンキー
130          devi$ DV$,REC,A$(0),A$(1)
140          for A=0 to 255 step 16
150            X=A¥128:Y=(A and &H7F)+1                     '前半・後半の選択
160            A$=right$("000"+hex$(REC),4)+hex$(A¥16)+"0=":C$=" /"
170            print A$;:if LPT then lprint A$;
180            for B=0 to 15
190              C=asc(mid$(A$(X),Y+B,1)):A$=right$("0"+hex$(C),2)+" "
200              print A$;:if LPT then lprint A$;
210              if C<32 then C=32
220              C$=C$+chr$(C)
230            next
240            print C$:if LPT then lprint C$
250          next
260          print "w";DV$;mid$(str$(REC),2)
270        else if left$(A$,1)="w" then       'w1:0 等でリターンを押した時=書き込み
280          A=instr(A$,":"):key 0,"r"+DV$
290          if A then devo$ mid$(A$,2,A-1),val(mid$(A$,A+1)),A$(0),A$(1)
300        else if left$(A$,1)="0" and mid$(A$,7,1)="=" then       'エディット
310          A$=mid$(A$,5):A=val("&H"+left$(A$,2))-1
320 *EDIT1: A=A+1:A$=mid$(A$,4)
330          if A>&HFF or mid$(A$,3,1)<>" " then *EXIT '範囲外・エデット終了
340          if left$(A$,1)=";" then B=asc(mid$(A$,2,1)):goto *OK  '「;」+文字
350          B=val("&H"+left$(A$,2))
360          if B<&H10 then if left$(A$,1)<>"0" then *EXIT '2文字目が無効
370          if B else if left$(A$,2)<>"00" then *EXIT      '2文字共無効
380 *OK:     mid$(A$(A¥128),(A and &H7F)+1,1)=chr$(B):goto *EDIT1
390 *EXIT:
400        else
410          key 0,"r"+DV$
420        end if
430        goto *LOOP
```

ってしまった感が強いでしょう。

ユーザー座標系は，グラフィック画面の中に，実際に描画を行う窓「**ビューポート**」を設け，ここに，プログラム中で指定したい座標，すなわちワールド座標を割り当てることで行います。また，一部のBASICのユーザー座標系にはない，X軸・Y軸の正負方向変更機能もあります。パソコンのグラフィック座標は，Y軸が下に行くほど大きな値となり，我々の感覚と合いませんがこれを上に行くほど大きな値とすることもできます。

ところで，スーパーMZでは，ビューポートをテキスト画面に対しても設定することができます。これまでの多くのBASICでは，CONSOLE 0，10 のようにして，縦方向だけに窓を作ることができましたが，スーパーMZでは横方向についても可能になっています。

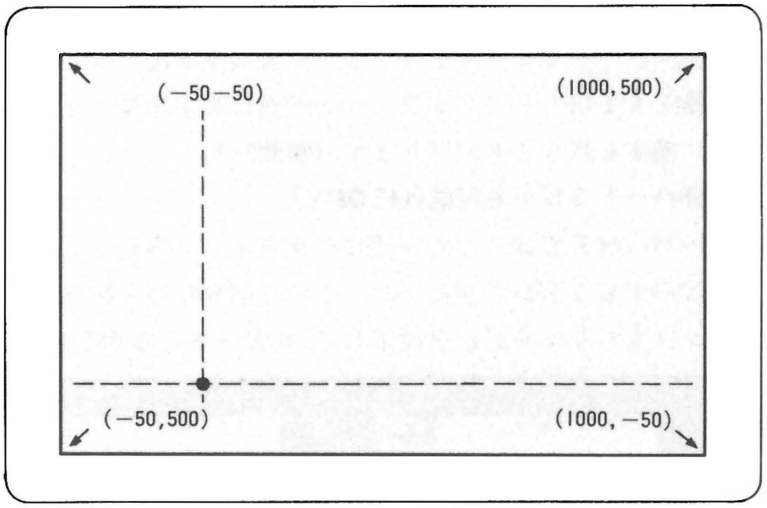


《第３－１図》ユーザー座標系

●タイリングペイントやラインスタイルもOK（PAINT，LINE）

これも今や常識でしょう。ユーザー座標系にしてもラベルにしても，「売れてるパソコン」のほとんどが採用しています。

タイリングペイントは、画面をぬりつぶすペイント機能の一種ですが，その際、単色でぬるのではなく，シマ模様や一松模様，そのほか好きな模様を使うことができる機能です。従来，中間色をシミュレートするのに使われることも多かったのですが，スーパーMZではハードウエアで256色できますので，これについてはあまり使用する必要はないかも知れません。



《第３－２図》Ｙ軸方向を変更し他のユーザー座標系の機能と組合わせてみる

次に，ラインスタイルは，線を引く際，点線や破線，一点鎖線などで描ける機能です。カラーはモノクロのハードコピーをとるとわからなくなってしまいますが，点線等はきちんと残りますので，折れ線グラフを描くのにも便利だと思います。

●ペンスタイルが設定可能（PEN）

これはスーパーMZ独自の機能です。

例えばグラフィック命令のLINEは，点を打つPSETを線状に繰返すもの，CIRCLEは円状に繰返すものですが，これらを描くのに使われる，言ってみれば「**ペン**」にあたるものは，これまでのパソコンのBASICの場合，そのパソコンまたは画面モードの持つ最小のドットに固定されていました。この場合，ちょっと太い線を引きたいとか，大きな点を打ちたいと思っても，なかなか思うようには行かなかったようです。

これに対してスーパーMZのBASICでは，この「ペン」の形状を8×8ドットの範囲内で，自由に変えることができるようになりました。形状は，最小ドット(1ドット)以外にBASICの方が予め5通り用意しており，これを使うこともできますし，ユーザーが定義しなおして使うこともできます。**第3－3図**に，予め用意されているペンの形状を示します。

なお，このペンを使うと，ユーザー座標系における唯一の盲点，すなわち，ドット数の違うパソコン等のシミュレートを行うような場合，座標は合ってもドットは小さいものになってしまって，必ずしも同じ結果が得られない点を補うこともできます。

●グラフィック画面の保存が可能(GSAVE,GLOAD)

スーパーMZでは，グラフィック画面をディスクにセーブすることができます。

また，画面の一部分だけをセーブすることもできますし，それを、オフセットをかけて，好きな位置にロードすることもできます。

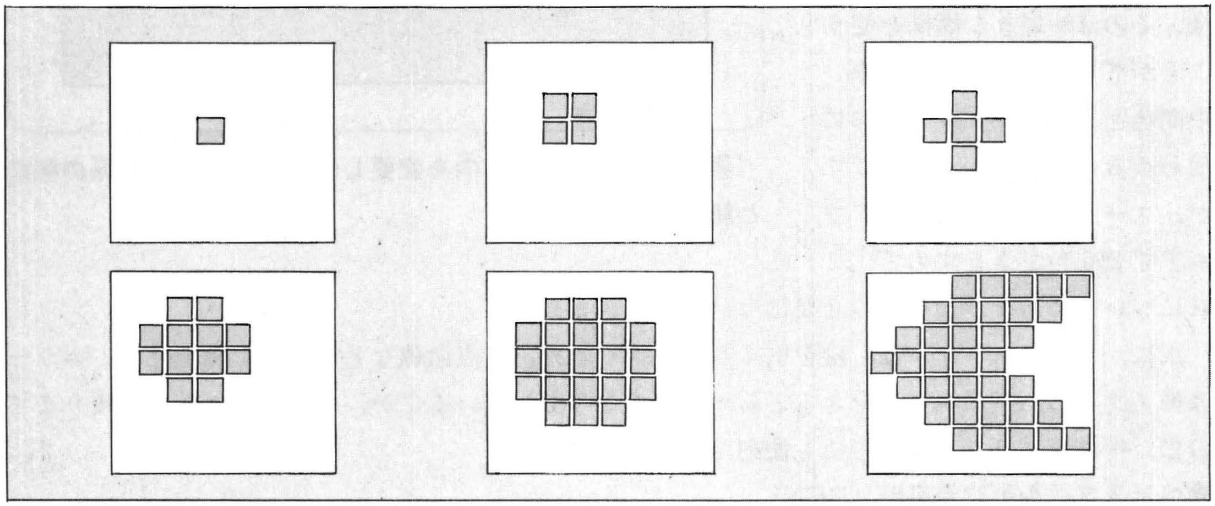
この機能はスーパーMZが最初というわけではありませんが，なかなか便利に使えると思います。

●シンボル機能(SYMBOL)

スーパーMZでは、グラフィック画面に拡大文字を表示することができます。またそれを90°単位で回転させることもできます。拡大倍率は縦・横とも1以上の整数なら何倍でも良く，単に縦・横とも1倍としてグラフィック画面に文字を描かせることもできます。また，ANKだけでなく，漢字も扱うことができます（**画面3-1**）。

●部分ハードコピーも可能(HCOPY)

スーパーMZでは，ごく普通にグラフィック画面やテキスト画面の内容全体をプリンタにハードコピーすることができるほか，その一部分だけを抜き出してコピーすることもできます。また，プリンタ上で左マージンを設定して，コピーされる画を右にずらして印字させることもできます。



《第3－3図》予め定義されているペンの形状(1～5はユーザー定義可能)

●色の置換え機能(COLOR REPLACE)

これまで，画面に表示されているグラフィックの色を変えるには，もっぱらパレットが使われていましたが，本当に，グラフィックＲＡＭの内容を書換えて，色を変えたいこともありました。

そこでスーパーＭＺでは，**カラーリプレース機能を**持たせ，パレットの変更なしで色(正しくはパレットコード)を書換えることができるようにしました。また，これも画面全体のほか，一部分だけを行うことができます。

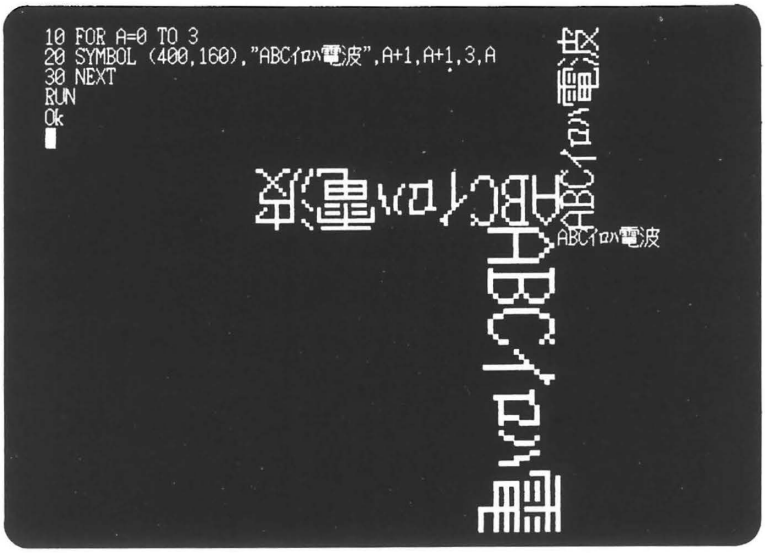
●部分移動機能(MOVE)

これまで，グラフィック画面の一部分を別の場合に移動させたい場合，ＧＥＴ@，ＰＵＴ@を使って，一度メインメモリの配列変数に読込んだ後，再び別の場所に出力する，という手法がとられていましたが，スーパーＭＺでは，これらに加えて，直接グラフィック画面上での移動もできます。また，その時，元の場所を消す本当の移動と，元の場所は消さずに残す，複写のいずれかを選択することができます。

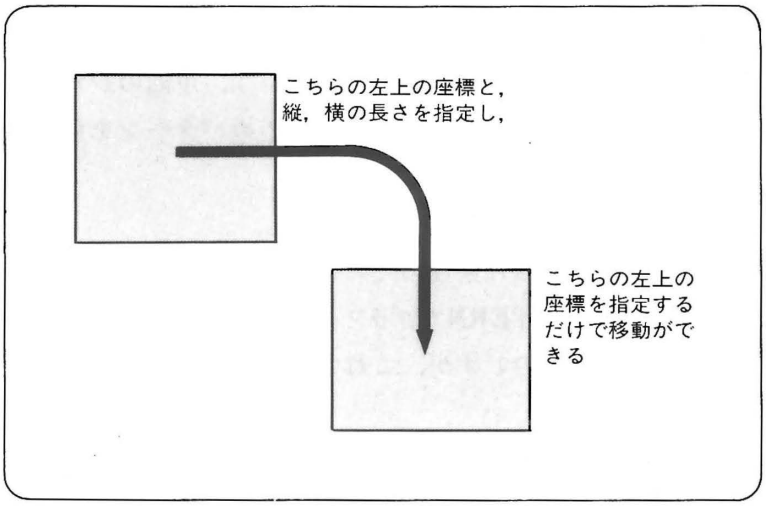
配列変数に読切れないような大きなパターン，例えば画面全体を上下左右に動かすこともできます。

●多角形

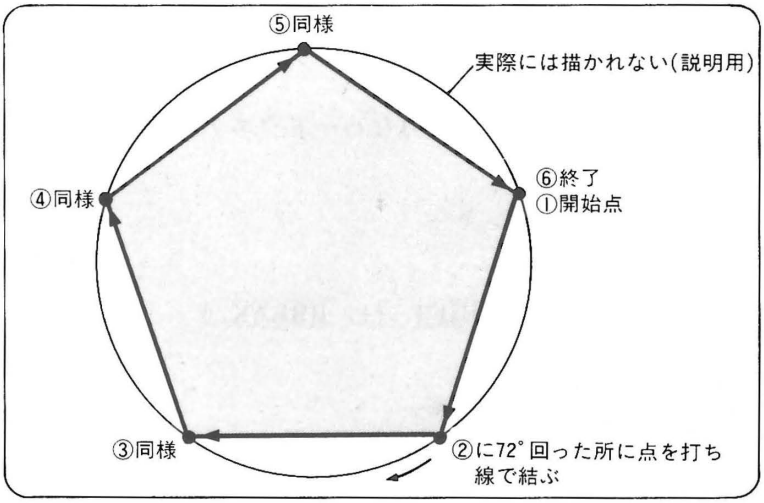
スーパーＭＺでは，五角形や



《画面３－１》SYMBOL文で1×1,2×2,3×3,4×4倍の文字を回転させてグラフィック画面に表示させてみた



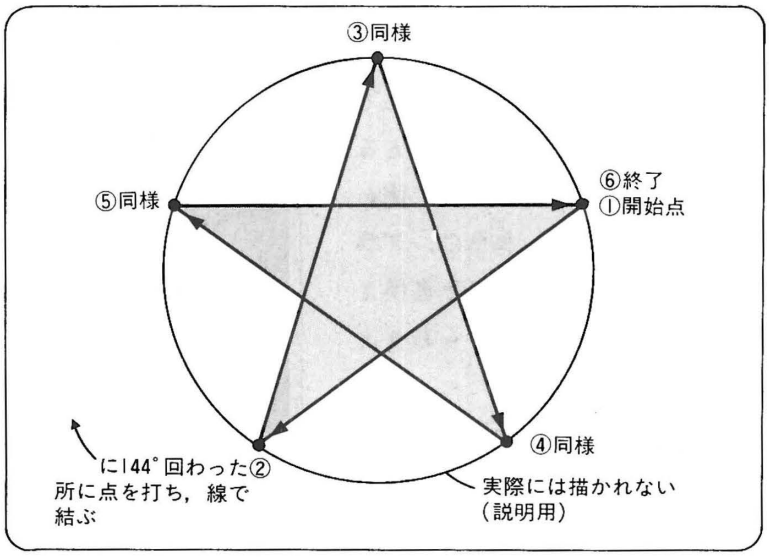
《第３－４図》ムーブ機能



《第３－５図》五角形を描く

八角形，星形などを，複雑な計算なしで，簡単に描ける命令を備えています。

第3－5図と**第3－6図**に一例を挙げておきますが，このように円周を移動させるようにして，多角形が描けるわけです。



《第3－6図》星形を描く

●指定パターンの直接表示機能（PATTERN）

グラフィック画面に表示されている，小さなキャラクタか何かを移動させるには，前述の通りＧＥＴ@，ＰＵＴ@を使っていましたが，予め表示したいパターンがわかっていた場合は，逆に一度点を打つなどして，画面に表示させた後，ＧＥＴ@で再び読込み，ＰＵＴ@で再表示するという，手間のかかることをやっていました。

それに対してスーパーＭＺでは，予めパターンを用意した場合は，専用の命令でダイレクトに表示できるようにしています。

●PUT@　KANJIとROLLも装備

これらの命令は互換性のために用意されています。漢字はテキスト画面に表示できますし，SYMBOLやPATTERNでグラフィック画面に表示させることもできますので，PUT@ KANJIは特に必要ではないのですが，これでしか漢字を表示できないパソコンとの互換性のために使えるというわけです。

また，ROLLはグラフィック画面全体の上下への移動ですが，これを含むMOVEがありますので，やはり互換性のためだと思います。

ただ，ROLL@というハードウエアスクロールをサポートする機能もありますが，こちらはオリジナルです。

●各種割込みをサポート

スーパーＭＺでは，強力なハードウエアをサポートする一手段として，各種の割込みをサポートしています。

- ファンクションキー
- HELPキー
- STOPキー（SHIFT＋BREAK）
- 時刻
- インターバルタイマー
- RS-232C
- マウス
- 音楽演奏の終了

特に最後の音楽演奏の終了というのはユニークです。これまでならせっかくバックグラウンド演奏ができるにもかかわらず，次々と音楽を演奏したいような場合には，演奏が終わるまで何もせずにただ待っているということが多くありましたが，スーパーＭＺなら次のデータが必要になったら割込みがかかりますので，その間グラフィックを表示させるなど，好きなことができます。

●＆Ｏ，＆Ｈ以外に＆Ｂ，＆Ｊ，＆Ｋをサポート

グラフィックの操作や，Ｉ／Ｏポートの操作を行うような場合，コンピュータ本来の姿である２進数を必要とすることが多くあります。これまでの多くのBASICでは，この場合8進数<＆Ｏ>または16進数<＆Ｈ>を使うことになっていたようですが，スーパーＭＺには２進数<＆Ｂ>もありますので便利です。もちろん，ＯＣＴ＄，ＨＥＸ＄同様のＢＩＮ＄もあります。

一方，**＆Ｊと＆Ｋは，ＪＩＳ漢字コードと区点コードを，シフトＪＩＳコードに直す働きを持つ**ものです。スーパーＭＺでは漢字コード体系として，16ビット機のＭＳ-ＤＯＳや漢字ＣＰ／Ｍ-86等で採用され，主流になったシフトＪＩＳを採用[*2]して操作性を向上させていますが，プログラム上で既にわかっているＪＩＳコード・区点コードの漢字を扱う時に＆Ｊや＆Ｋを使うのです。

例えば

```
PRINT ″高″
PRINT CHR＄(＆J3962)
PRINT CHR＄(＆K2566)
PRINT CHR ＄(＆H8D82)
PRINT CHR＄(＆H8D)＋CHR＄(＆H82)
```

などは，すべて同じ働きをします。「高」のＪＩＳコードは3962，区点コードは2566，シフトＪＩＳコードは８Ｄ82だからです。

●連続POKE，連続CHR$，HEXCHR＄，ASCCHR＄

これは説明するより見た方が早いでしょう。

```
POKE ＆HF000，＆H3E，＆H1E，＆HC3，＆H20，＆H0
```

これは

```
POKE ＆HF000，＆H3E：POKE ＆HF001，＆H1E：POKE ＆HF002，＆HC3：POKE ＆HF003，＆H20：POKE ＆HF004，＆H0
```

と同じ働きをします。もちろん後のように書いても良いのですが，連続POKEを使った方がミスが小なくなりますし，すっきりとして見やすくなります。

ＣＨＲ＄も同様です。さきほどの

```
PRINT CHR＄(＆H8D)＋CHR＄(＆H82)はPRINT CHR＄(＆H8D，＆H82)
```

とすることもできますし，ＣＨＲ＄(1,2,3,4)のように，いくつも書くこともできます。

さらに，これを一歩進めたのが**ＨＥＸＣＨＲ＄**です。

```
PRINT HEXCHR＄(″8D828BB4975988EA″)はPRINT CHR＄(＆H8D，＆H82，
＆H8B，＆HB4，＆H97，＆H59，＆H88，＆HEA)
```

※２　キャラクタコード80_H～$9F_H$，$E0_H$～FC_Hの部分を漢字の１バイト目として扱う方式。なお，他の多くの８ビット機同様このエリアをセミグラフィックとして使うモードもある。これらを画面に混在させることもできる。

または

PRINT CHR$(&H8D)+CHR$(&H82)+CHR$(&H8B)+CHR$(&HB4)+CHR$(&H97)+CHR$(&H59)+CHR&(&H88)+CHR$(&HEA)

と同じです。

HEXCHR$はまた，グラフィックでも重宝です。タイリングパターンの指定やペン形状，PCGの定義やPATTERN文のパターンまで，たいへんスマートに書くことができるからです。

**ASCCHR$**というのは，HEXCHR$の逆関数です。

A$の内容がCHR$(&H0C)+CHR$(&H41)だった時，

PRINT A$

とすると画面が消えてから「A」と表示されますが，このようにコントロールコードを含む文字列に本当はどのような文字が入っているかを知るには，今までなら

FOR A=1 TO LEN(A$):PRINT HEX$(ASC(MID$(A$, A, I)));:NEXT

のようにする必要がありましたが，スーパーMZでは

PRINT ASCCHR$(A$)

とするだけで，

0C41

と表示されますので，たいへん便利です。デバッグにも役立つと思います。

●ユニークな関数，いろいろ

スーパーMZでは，このほかにも，ユニークな関数がいくつかあります。

まず，**FRAC**。これは小数部のみを取出すもので，FIXのちょうど逆にあたるものです。

次に**LOG10**。ちょっと変わった名前ですが，常用対数を求めるものです。パソコンのBASICのLOGは，関数電卓で言うLNですので，BASIC-M25のLOG10は，関数電卓のLOGにあたるものです。

次に**FAC**。これはX！つまり階乗を求める関数です。FAC(5)は120になります。

また，**PAI, RAD, DEG**は，π＊nを求める関数，度→ラジアンへの変換関数，ラジアン→度への変換関数です。

SUMというのもあります。

この関数はスーパーMZオリジナルのもので，

SUM(A, 1, 10, A)

は55(1＋2＋…＋10＝55)，

SUM(A, 1, 10, A＊A)

は385($1^2+2^2+3^2+\cdots 10^2=385$)，

SUM(A, &H41, &H5A, CHR$(A))

は〝ABCD…Z〟

を表わします（変数Aの値はこの関数実行後変化します）。また，

SUM(A, 1, 10, X(A))

としてX(1)～X(10)の総和を計算することもできます。FOR～NEXTでも行えるかも知れませんが，

SUM (A, 1, 10, SUM (B, 1, 100, X (A, B))
のようにネストさせて書くこともできますので，このような場合には，すっきりと見やすく，スピーディに計算でき，たいへん便利だと思います。

この他にも，文字列全体をリピートさせるSTRING$…(STRING$(3, "ABC")は "ABCABCABC"になる)。曜日を得るDAY$…(日曜日ならDAY$は"SUN"になる)，画面上に表示されている文字を取ってくるSCRN$，HEXCHR$やPATTERNと組合わせると面白そうな鏡像のパターンを得るMIRROR$や，BASICの名称とバージョンを返す関数SYSIDS$もあります。これを利用すると，今動かしているBASICによって処理を変えるようなアプリケーションが作れるため，いくつものパソコンまたはBASICで共通のプログラムを利用することもできると思います。また，スーパーMZ単独で考えても，BASIC-M25と後で触れるBASIC-S25の両方で動くプログラムも作れるでしょう。

そして，日本語処理関数でかるAKCNV$，KACNN$，KLEN，KPOS等もあります。半角文字の "ABC" を全角の "ＡＢＣ" に変えたり，その逆を行うことも簡単にできます。

●ON～RESTORE, ON～RETURN, ON～RESUME

今まで，

ON～GOTO

ON～GOSUB

で，式の値によってGOTOする行やGOSUBする行を変えることができましたが，スーパーMZでは，式の値によってRESTOREする行を変えたり，RETURNする行を変えたり，RESUMEする行を変えることもできます。

特にRESTOREは，これまで，ON～GOSUBで一度飛ばしてから，その先でRESTOREし，RETURNするという，まるでゲームのようなことをやっていた場合もありますので，連続POKE～HEXCHR$と同じように，すっきりと見やすいプログラムが書け，ミスも極力防げるのではないかと思います。

●省略形による入力が可能

例えば

LIST　100

としなくてはならない所を，

L. 100

で済ませることができる機能です。省略形はプログラム中で使うこともできますし，リストをとるときちんと正しい語に直っています。しかも，コマンドに限らず予約語のほとんどで使うことができます。デバッグを行う時にはひらめきが肝心ですので，考えたことが一刻でも早く実行できると，それだけ，効率も上がることでしょう。また，タイプの苦手な人にも便利ですし，LIST等の後にはスペースが必要なのに対し，省略形では．の後にスペースが(行番号等がすぐ書ける)ですので，せっかちな人(？)にも朗報でしょう。

●エディットモード

これは比較的古くからある機能ですが，スーパーMZでも備えています。しかし，第2章のキ

ーボードの所でも触れたように，１行前・後の行をリストするだけでなく，半ページロールダウン，半ページロールアップを行うこともできるよう拡張されたものになっています。専用のフルスクリーンエディタの感覚で，エディットできることでしょう。

●サーチ機能

例えば，あるサブルーチンを呼び出している行を捜したいとか，ある変数名が代入されているところ，または参照されているところを抜き出して見てみたいと思うことは，いくらでもあります。

これまでの多くのBASICの場合は，プリンタに一度打ち出してから，人間が目で捜すという，まったく非能率なことをやっていました。これしか方法はないのですから，仕方なかったのです。

しかし，スーパーＭＺには，これらが使われている行をBASICの方で捜し出し表示・印字してくれる機能があります。その上，捜したい文字の部分は，画面上では紫色で表示してくれるので，非常に便利です。また捜し出す範囲を例えば，1000行～1999行に限定するといったこともできます。エディットモードonlyの，手抜きエディタではないのです。

●ブレークポイントとシングルステップデバッグモード

BASICとは関係ないのですが，マシン語のソフトを専門に開発するプロのプログラマたちは，そのデバッグ用にインサーキットエミュレータ(ＩＣＥ)という高価な機械を使っています。プログラムの流れがある所に来たら止めるとか，１ステップずつ実行してみる，と言った便利なことができるものですが，スーパーＭＺでは，これと似たことをBASICで実現しています。

まず，**STOP　行番号**とダイレクトモードで行うと，その行を実行する直前で止まってくれます。STOP文そのものをプログラム中に置く場合と決定的に違うのは，そのSTOP文を置くためにプログラムをエディットしてしまうと続けてCONTできないのに対し，STOP　行番号では，それを入力してもプログラムを変更したわけではないので，CONTで実行を再開できるということです。つまり，少しずつ，実行を確認しながら，デバッグを行えるということです。

次に，**STEP命令**です。このSTEPというのは元々はFOR文中で使う予約語ですが，単独で使うと全く別の働きになります。

STEPは，STOP　行番号を自動的に続けて行うもので，一行おきに止めてみることができます。つまり，１行毎に実行が確認できるということです。さらに、極めつけなのは，インサーキットエミュレータそっくりの**「何行分実行したら止める」**という操作ができることです。これですと，プログラムの流れがはっきりとわからない場合にも使えます。TRONと違って，画面がみだれてしまうこともありません。

なお，このTRONは，行番号の表示を行う範囲を限定できます。プリンタに対して行うこともできます。**LTRON**という命令がそれです。

●変数保護機能

これは特に真新しいものではありません。それどころか，シャープでは，**ＭＺ－80Ｋ**のころからこれを持っています。しかし，現在のパソコンでこれを持っているものがあまり多くないので，取り上げてみました。

プログラムがエラーで止まったような場合，そこにはバグがあるわけですが，それを直す，つまりプログラムを修正する(エディット)と，他の多くBASICでは，変数や配列，文字列その他が

すべて消えてしまいました。これですと，プログラムは何もしていない状態と同じになってしまいますから，再びＲＵＮ命令によって，プログラムの先頭から実行し直すしかありません。

それに対して，スーパーＭＺでは，プログラムを直しても変数が消えてしまうことはありません。ですから，CONTは使えないものの，GOTOでプログラムの実行を再開することは可能です。長いループでわざわざたくさんのデータをキーボードから入力した後でエラーが起きたような場合にも，多くのBASICは最初からやりなおしですが，スーパーＭＺでは，GOTO で続きを実行することができるのです。

●ガベージコレクションを解決

これも前項同様，シャープのBASICでは７年前から実現していることですが，同じ理由で取り上げてみました。

ガベージコレクションについては，「マイコン」誌でも，その他の書籍でも何度となく問題にされており，コトの重大さがわかりますが，文字列を扱っている時，突然，数分間に渡って，プログラムの実行が止まってしまい，STOPキーも受け付けられなくなってしまう現象です。これがありますと，そのBASICでは，文字配列変数を多用した，実用的なプログラムを作るのが困難になります。何も知らない人が見たら，機械が故障したのかと思ってしまうかも知れませんし，作った本人でさえ，何度となく繰り返すガベージコレクションには付き合いきれないのではないでしょうか。

次に示すガベージコレクションテスト用プログラムは，ディスクやキーボードから多数の文字列を読込み，その後で何度も一般の文字列の操作を行うことを想定したもので，その処理時間を計るものです。

```
10 DIM A$(2000)
20 FOR A=1 TO 2000
30 A$(A)=CHR$(255)
40 NEXT
50 TIME$="00:00:00"
60 FOR A=1 TO 2000
70 A$="":A$=SPACE$(255)
80 PRINT"・";
90 NEXT
100 PRINT TIME$
```

スーパーＭＺの場合，このプログラムを実行すると，16秒で終わりますが，中には数時間もかかるパソコンもあります(30行がエラーになったらCLEAR700としてRUNする)。

●OPTION LIST機能

スーパーＭＺでは，リストを取る際，予約語を**小文字**に変換して出力する機能を持っています。持っています，というより，起動時はこの状態になっています。もちろん他のパソコンのBASICのように，大文字で出力することもできます。

つまり，起動時の状態では

```
10 dim A$(2000)
```

```
20 for  A＝1  to  2000
```

のように，OPTION LIST UCASEとすると

```
10 DIM A$(2000)
20 FOR A＝1  TO 2000
```

のようにできるわけです。また，OPTION LIST LCASEとすると元に(小文字に)戻ります。なお，UCASEというのはUPPER CASE(大文字)，LCASEというのはLOWER CASE (小文字)という意味です。

予約語が小文字で出て来ると，予約語と変数とが一目で見分けられますから，変数名だと思ったら予約語だったり，予約語だと思ったら打ち込みミスをしたりして変数名の扱いになっていた，ということが極力防げますので，初心者の方でもsyntax errorの回数がかなり減るのではないでしょうか。

●OPTION ANGLE機能

これまでのほとんどのBASICでは，sin，cos，tan，atn，の関数で使用される角度単位には，ラジアンを採用していましたが，スーパーMZでは，「**度**」にすることもできます。例えば普通の状態ではatn(1)は0.785398…つまり$\frac{\pi}{4}$になりますが，「度」に変更すると，45が返されます。

なお，この指定は，CIRCLE文等における，開始角と終了角とにも適用されます。

●OPTION SCREEN機能

これはいわゆる**RAMディスク**です。

スーパーMZではメインメモリは265Kバイト，グラフィックRAMは128Kバイトまで拡張できますが，必ずしもこの全部をフルに使用するとは限りません。そこで，余ったメモリを有効に使用するため，それをディスク装置のように使うことができるようになっています。RAMディスクはフロッピーディスクより高速ですし，スーパーMZではファイルディスクリプタを完備しており(**第3－1表**)すべての周辺機器(RAMディスクも)を同一形式でアクセス可能ですので，それを生かした応用もいろいろ考えられると思います。

また，スーパーMZでは，RAMディスクと同じ方法で，プリンタスプーラを使用する

《第3－1表》スーパーMZのファイルディスクリプタ

装置名(デバイス名)	装置	プログラム入力(バイナリロード)	プログラム出力(バイナリセーブ)	シーケンシャル入力	シーケンシャル出力	ランダム入出力
1：～4：(またはFD1：～FD4：)	フロッピーディスク	○	○	○	○	○
CMT：(CMT1：)	カセットテープ	○	(ユーティリティで対応)	○	(ユーティリティで対応)	
MEM：(MEM1：)	RAMディスク	○	○	○	○	○
CRT：(CRT1：)	テキスト画面				○	
LPT：(LPT1：)	プリンタ				○	
KB：(KB1：)	キーボード			○		
COM：(COM1：)	RS-232C ポートA			○	○	
COM2：	RS-232C ポートB			○	○	
USR：(USR1：)	ユーザーデバイス			○	○	
ROM：(ROM1：)	640K ROMボード	○		○		△(入力のみ)
EMM：(EMM1：)	640K大容量RAMディスク	○	○	○	○	○

この他にもデバイス名としてグラフィックRAMを表わすCRT2：とモノクロディスプレイを表わすCRT3：があり，INIT文でのみ使用可能

こともできます。よく，プリンタバッファというのが市販されていますが，あれと同じ効果が得られるものです。つまり，プリンタの印字速度は遅いため，パソコンが次の仕事をしたくても，それが終わるのを待っていなければなりません。そこで，BASICではプリンタの都合は考えないことにして，一度バッファにたくわえておき，プリンタが印字を終わり，次のデータを必要とした時に初めて，プリンタから割込みをかけてもらい，データを送るのです。すると，待ち時間がなくなり，総合的なスループットが上がるというわけです。

なお，ＲＡＭディスク，スプーラは，８Ｋバイト単位で各々最大128Ｋバイト，64Ｋバイトまで使用可能です。

●階層ディレクトリをサポート

スーパーＭＺでは，将来性を考え，**階層ディレクトリ**をサポートしています。

ディスク装置の容量が増え，一度にたくさんのファイルを収めることができるようになると，その管理がたいへんです。例えば20メガバイトのハードディスクには，10Ｋバイトのファイルは2000個置くことができますが，それらに全く重ならないファイル名を付けたり，必要なファイルを捜すのは，とても面倒です。

そこで，階層ディレクトリというものが考えられたわけですが，スーパーＭＺでは，将来ハードディスクが使えるようになった場合(ハードウエア的には可能になっています)のことを考えて，これを導入したわけです。もちろん内蔵の3.5インチフロッピーディスクに対しても，可能になっています。

第３－７図に階層ディレクトリの概念を示します。このように，まず大まかに種類を決め，だんだん下に行くにつれ，細分化していくことができます。

ルートディレクトリの下には，常通，ファイルは64個作れます。BASICのプログラムやデータ，マシン語プログラムを自由に置くことができます。

ルートディレクトリの下には，また，サブディレクトリも置けます。ファイルと合わせて64個まで可能です。そのサブディレクトリの下には，やはり64個のファイルやサブディレクトリが置けます……。

つまり，サブディレクトリというのは，言ってみれば，三つ又，いや64又コンセントのようなものなのです。スーパーＭＺでは一応10層までに制限していますが，原理上はこれを何回繰返しても良く，こうすることで，目的のファイルを効率良く見つけたり，作成することもできるようになるわけです。フロッピーディスクでも，なかなか便利に使えそうです。

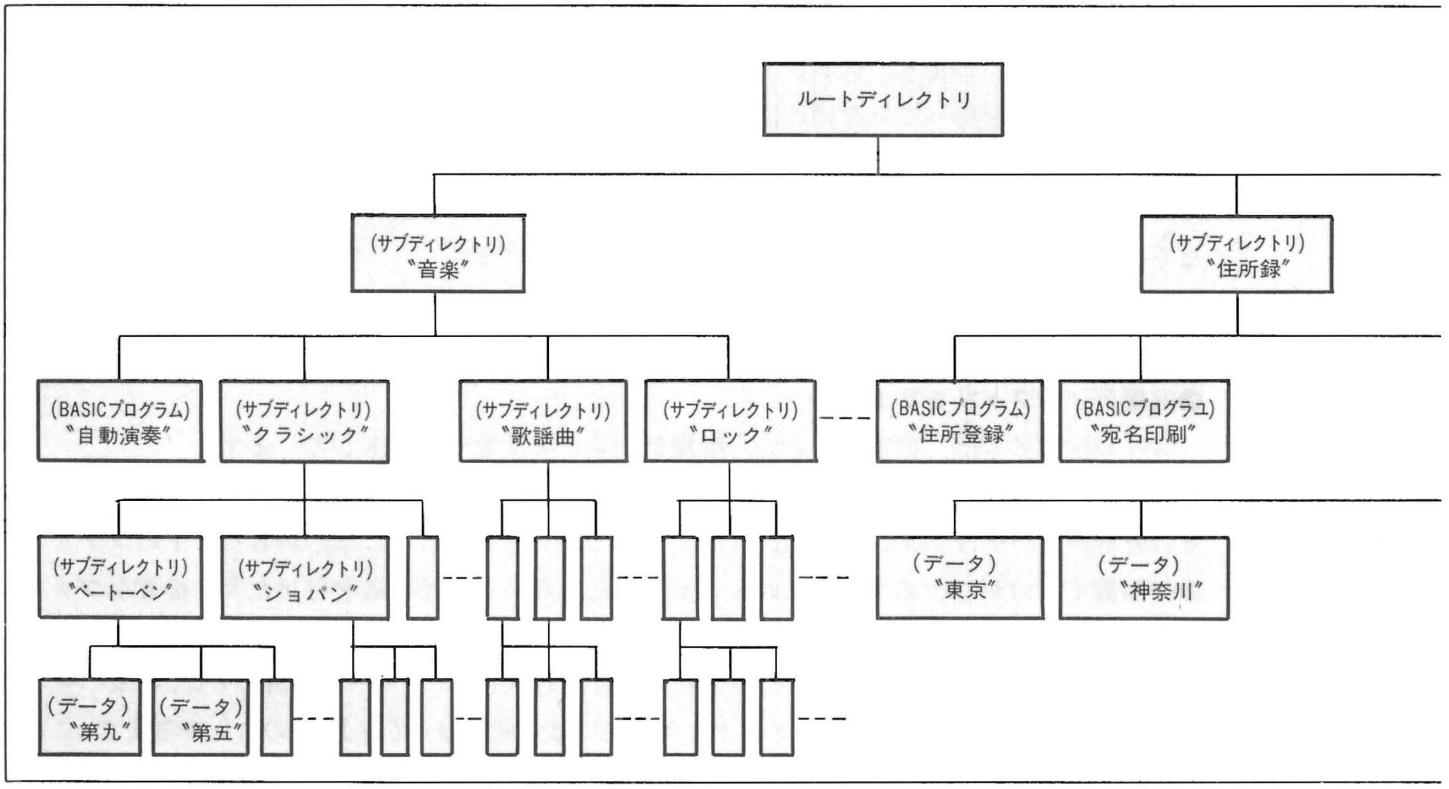
●疑似可変長レコード

これまで，MS-DOS等の下で動くBASICを除き，一般にランダムファイルの１レコードの長さは，フロッピーディスクの１セクタの長さの都合で256バイトに固定されていましたが，１レコードに置くデータがさほど多くない場合，これでは非能率でした。例えば30バイトしか必要ないなら，残りの226バイト分は無駄になっていたからです。ディスクの容量は有限ですから，あまりデータを置かないうちに，いっぱいになってしまうことも多くあったでしょう。

そこで，スーパーＭＺでは，スピードの低下が起きない範囲で，１レコードの長さを変えられるようにしています。32バイト，64バイト，128バイト，256バイトの中から選択できるのです。これらは256の約数になっていますので，効率良く，アクセスできるのです。

●その他

その他，気付いたことを書いておきます。



《第3－7図》階層ディレクトリの概念

LIST＊，LLIST＊という命令があり，'（アポストロフィ）から始まる行の，中身だけを出力することができます。簡易ワープロとしても使えます。また，これに便利な，'の出て来るAUTO＊もあります。

プログラムの一部だけをセーブすることができます。例えば1000行から2000行までをセーブすることができます。

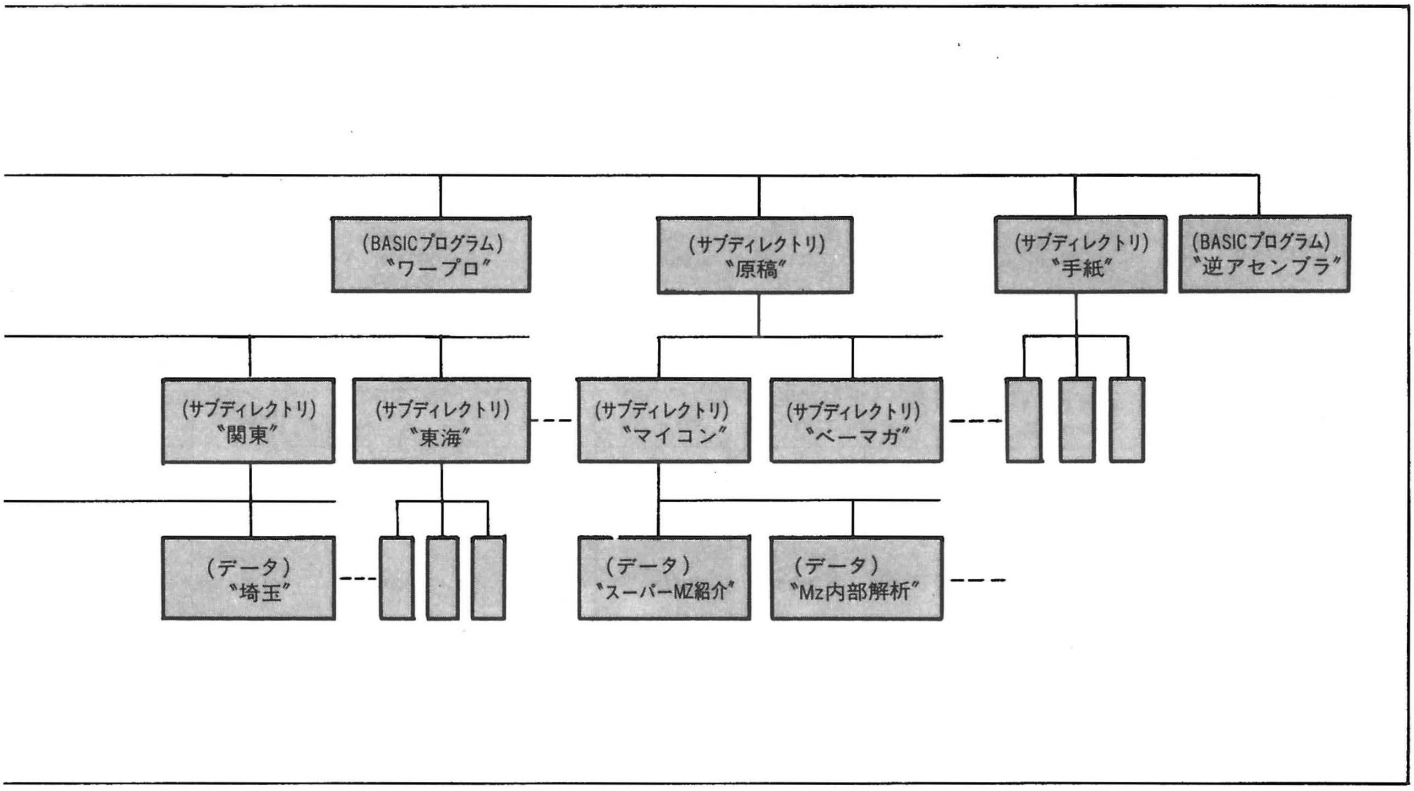
RENUMで，特定行以降全部でなく，ある行からある行までを指定して，リナンバーできます。例えば100行～139行までを100，102，104…と付け換えることもできます。

FILESでワイルドカードの指定ができます。例えば，Aから始まるファイルネームや3文字目がiのファイルネームだけを表示させることもできます。

バイナリセーブされたプログラムでもMERGEできます。

各種デバイスを様々なモードに設定する命令を，「**INIT**」として統一しています。画面モードの設定やJIS／50音カナ配列，RAMディスク・プリンタスプーラなどの容量，RS-232Cパラメータ等はすべて，**INIT〝デバイス名：～〟**で決められます。また，単なるINITでは，グラフィックのユーザー座標系や，テキスト画面のスクロール範囲，パレット，キーボードのリピート速度，FM音源のレジスタ等を，BASIC起動時の状態に戻してくれます。プログラムをいくつか続けて実行するような場合，先頭にこれを置いておくと，以前の状態を気にせずにプログラムが書けます。

これまでの多くのBASICでは，プリンタに送るデータを加工して送っている場合がありました。例えばCHR＄(9)はTABコードとして処理し，複数のスペース(CHR＄(&H20))に置換えているものがありましたがこの場合，ビットイメージ印字等で09Hを送りたいと思っても，LPRINT



ではできませんでした。スーパーＭＺでもシフトJISをJISに変えたり，あるいはフォントに変えたりして送っていますので，事情は同じです。このため，**LPOUT**と言って，データをそっくりそのまま送る命令が入っています。

本来，階層ディレクトリ用の命令である**CHDIR**で，ログインディスクドライブが切換えられます。例えばCHDIR〝２：〟とすると，以後，だたのFILESや，LOAD〝ABC〟とした場合，ドライブ２が対象となります。

BASIC（厳密にはOS部）の一部やグラフィックRAMを段階的にカットし，ユーザープログラムや配列変数等用に転用することができます(プログラム０でも使えます)。NEW ONやOPTION SCREENという命令がそれです。

●BASIC M-25のまとめ

以上，このBASICの言語として見てみた場合の，特徴的な所を抜き出してみました。ただし，ハードウエア編で説明した機能をサポートする命令は，あって当然ですので割愛しました。**第３－３表**にこのBASICの全命令語を示します。

しかし，よくもあま，こんなに機能を詰め込んだものです。部分的にはこれを上回るBASICもありますが，総合的にはこちらの方が上でしょう。また，特にNECのPC-8801mkⅡSRやPC-9801シリーズ用のN88BASICを意識しているようで，ラベルやユーザー座標系の命令等も，同じシャープのＸ１turboのものとは少し異なり，N88に合わせています。また本文中でも触れましたが，スーパーＭＺにとっては必要のないPUT@　KANJI等も持っており，移植性についてはかなり本気で考えているようです。もちろん，マイクロソフト系の命令系であることには違いありませんから，富士通ＦＭやＸ１・Ｘ１turbo等からの移植も容易です。

なお，一般の（Ｘ１turbo以外の）８ビット機からの移植の際には，KMODE　０という命令を使

《第3－2表》NEW ON・OPTION SCREEN・アルゴ機能とフリーエリア(BASIC-M25,BASIC-S25共通)

		システムRAM128Kバイト時				
		通常時 G-RAM 64K時 OPTION SCREEN1・2 GRAM 128K時 OPTION SCREE 1	G-RAM 64K時 OPTION SCREEN3 G-RAM 128K時 該当なし	G-RAM 64K時 OPTION SCREEN4 G-RAM 128K時 OPTION SCREEN2	G-RAM 64K時 該当なし G-RAM 128K時 OPTION SCREEN3	G-RAM 64K時 該当なし G-RAM 128K時 OPTION SCREEN4
NEW ON 非実行時	電卓・カラーシミュレーション・オートダイヤラー使用時					
	電卓・カラーシミュレーション使用時	22217	54985	87753	120521	153289
	アルゴ機能非使用時	30409	63177	95945	128713	161481
NEW ON 1実行時 (音声合成・マウスをカットする)	電卓・カラーシミュレーション・オートダイヤラー使用時					
	電卓・カラーシミュレーション使用時	30409	63177	95945	128713	161481
	アルゴ機能非使用時	38601	71369	104137	136905	169673
NEW ON 2実行時 (NEW ON 1でカットされるものと辞書 ROM を使った文節変換・人名地名変換をカットする)	電卓・カラーシミュレーション・オートダイヤラー使用時					
	電卓・カラーシミュレーション使用時	38601	71369	104137	136905	169673
	アルゴ機能非使用時	46793	79561	112329	145097	177865
NEW ON 3実行時 (NEW ON 2でカットされるものと残りのカナ漢字変換・ターミナルモードをカットする)	電卓・カラーシミュレーション・オートダイヤラー使用時					
	電卓・カラーシミュレーション使用時	46793	79561	112329	145097	177865
	アルゴ機能非使用時	54985	87753	120521	153289	186057
NEW ON 4実行時 (NEW ON 3でカットされるものとグラフィック機能をカットする)	電卓・カラーシミュレーション・オートダイヤラー使用時					
	電卓・カラーシミュレーション使用時	54985	87753	120521	153289	186057
	アルゴ機能非使用時	63177	95945	128713	161481	194249
NEW ON 5実行時 (NEW ON 4でカットされるものとプリンタ機能をカットする)	電卓・カラーシミュレーション・オートダイヤラー使用時					
	電卓・カラーシミュレーション使用時	63177	95945	128713	161481	194249
	アルゴ機能非使用時	71369	104137	136905	169673	202441

- □はごく普通にBASICを起動させた時のフリーエリアです。
- システムRAM128K時はオートダイヤラーは使用できません。これはNEW ONを全く実行していないBASIC起動時に一緒にロードされるためです。
- 電卓・カラーシミュレーションを使用せずにオートダイヤラーを使用してもフリーエリアは三つ同時に使用した時と変わりません。
- アルゴ機能は，BASIC使用時に後からカットすることができます（オートダイヤラーだけをカットする，といったことはできず，全部一斉にカットされます)。NEW ON 0という命令がそれで，これを使うと「アルゴ機能非使用時」の状態のフリーエリアになります。
- 一度NEW ONでカットした機能は，BASICを起動し直さないと元に戻りません。例えばNEW ON 4を実行した後にNEW ON 5を実行することはできますが，NEW ON 3～NEW ON 1は実行することができません。なおNEW ON 0はいつでも実行できますが，やはりその場で再びアルゴ機能を復活させることはできません。また，auto-run.m25やauto-run.S25中では，NEW ON 2を実行していますので文節変換や音声合成を使用したい時は，そのプログラム(BASICで書かれている)を書換えて、BASICを起動し直し利用します。
- グラフィックRAM64Kバイト時のOPTION SCREEN 3と，グラフィックRAM128Kバイト時のOPTION SCREEN 2，OPTION SCREEN 3を実行したことで確保されたエリア(「通常時」との差)は，プリンタスプーラかRAMディスクにしか使用できません。しかし逆にプリンタスプーラ・RAMディスクは一般のメモリを使用することもできますので，例えばOPTION SCREEN 1でスプーラを利用することも可能です。なお，OPTION SCREENを実行すると，フリーエリアが増えた分だけグラフィック用に使用できる正味の量は減少します。
- OPTION SCREEN 4を実行した場合，「通常時」との差は全部，何に使っても構わないようになります。つまり，配列等に使えるエリアは，この段階になって初めて64Kまたは128Kバイト，増えることになるのです。しかし，一度これを実行すると，NEW ON同様，その場では元に戻りません(OPTION SCREEN 3等は，OPTION SCREEN 1に戻すことが自由にできます)。つまり，グラフィックRAMが，本来のグラフィック用に戻ることを放棄した場合のみ，プリンタスプーラ・RAMディスク以外の目的にも利用することができるようになるのです。

うと，漢字の代りにセミグラフィックキャラクタが現われますので，そうすると良いでしょう。これはハードウエアの説明で触れるべきでしたが，スーパーMZのモードの時のセミグラフィックキャラクタは，マイクロソフト社のBASICの載ったテンキーの付いているパソコンとコンパチブルになっています。PC-8001系，8001系，9801系，FM系等と同じというわけです（**第3－4表参照**）。

	システムRAM256Kバイト時(MZ-IR26取り付け時)				
	通常時 G-RAM 64K時 OPTION SCREENI・2 GRAM 128K時 OPTION SCREEN I	G-RAM 64K時 OPTIOTN SCREEN3 G-RAM 128K時 該当なし	G-RAM 64K時 OPTION SCREEN4 G-RAM 128K時 OPTION SCREEN2	G-RAM 64K時 該当なし G-RAM 128K時 OPTION SCREEN3 G-RAM	G-RAM 64K時 該当なし G-RAM 128K時 OPTION SCREEN4
	136905	169673	202441	235209	267977
	153289	186057	218825	251593	284361
	161481	194249	227017	259785	292553
	145097	177865	210633	243401	276169
	161481	194249	227017	259785	292553
	169673	202441	235209	267977	300745
	153289	186057	218825	251593	284361
	169673	202441	235209	267977	300745
	177865	210633	243401	276169	308937
	161481	194249	227017	259785	292553
	177865	210633	243401	276169	308937
	186057	218825	251593	284361	317129
	169673	202441	235209	267977	300745
	186057	218825	251593	284361	317129
	194249	227017	259785	292553	325321
	177865	210633	243401	276169	308937
	194249	227017	259785	292553	325321
	202441	235209	267977	300745	333513

●NEW ONと，OPTION SCREENは，BASICプログラム中でも使用することができます。MZ-1500やX 1 turboのNEW ONは、BASICプログラムのNEWを伴いますが，スーパーMZではフリーエリアが連続して確保される必要が必ずしもないため，ごく一般的な命令のように使用することができるのです。

●フリーエリアのうち，常に6 Kバイトは一般変数用に，約8 KバイトはBASICプログラム用に，8 Kバイトは文字変数・文字配列変数・文字列格納（ストリングバッファ）用に確保されており，他に転用することはできません。その他の分(フリーエリアから22217バイト引いた残り)は，8Kバイト（8192バイト）単位で「一般変数」以外の目的に使用することができますが，BASICのプログラム用…48 K，文字変数・文字配列・文字列格納(ストリングバッファ)用…48 K，CLEARまたはLIMIT文によるユーザーマシン語プログラム用…48 K，プリンタスプーラ…64 K，RAMディスク…128 Kまでに限定されます。ただし数値配列変数は原理上384Kバイト分まで可能になってきますので，事実上使用目的による制限は受けないことになります（システムRAM256K，グラフィックRAM128KでNEW ON 5，NEW ON 0，OPTION SCREEN 4を実行しても汎用のフリーエリアは304Kしかありませんから）。

※1 Kバイト＝1024バイトです（詳しくは第6章を参照）

《第3－3表》BASIC-M25の全命令語

コマンド

RUN CONT STEP STOP 行番号 NEW NEW␣ON TRON TROFF
TERM MON BOOT LIST LLIST LISTファイルディルスクリプタ LIST＊ LLIST＊
AUTO AUTO＊EDIT SEARCH LSEARCH DELETE RENUM
OPTION␣LIST

一般ステートメント・基本入出力・実行制御命令

LET DEF␣INT DEF␣SNG DEF␣DBL DEF␣STR DEF␣FN DIM
OPTION␣BASE ERASE OPTION␣ANGLE REM RANDOMIZE
CLEAR SWP COMMON PRINT LPRINT
PRINT␣USING LPRINT␣USING WRITE INPUT LINPUT
INPUT␣WAIT LINE␣INPUT␣WAIT READ DATA RESTORE
GOTO GOSUB RETURN ON～GOSUB ON～RESTORE ON～RESTORE ON～RETURN
IF～THEN～ELSE IF～THEN…ELSE␣IF…ELSE…END␣IF FOR～TO～STEP…
FOR～TO～STEP…NEXT REPEAT…UNTIL WHILE…WEND STOP END
MID$＝ERASE PAUSE WAIT SWAP CLR

表示設定命令

OPTION␣SCREEN INIT "CRT 1 :" INIT "CRT 2 :" INIT "CRT 3" SCREEN
COLOR CLS

文字表示命令

WIDSH CONSOLE CONSOLE@ COLOR@ LOCATE CFLASH
CFLASH@ CREV CREV@ CGEN DEF␣CHR$ GET% PUT%
KEY␣LIST0／1 KLIST␣0␣1

グラフィック命令

CBLOCK COLOR＝ PRIORITY VIEW VIEW@ WINDOW PEN
PSET PRESET LINE CONNECT CIRCLE POLY PAINT
PATTERN SYMBOL COLOR␣REPLACE MOLL POINT
GET@ PUT@ GLOAD GSAVE ROLL@

ファイル処理

INIT "1 :" ～ "4 :" INIT "MEM :" INIT "COM :" FILES LFILES
CHDIR MKDIR RMDIR RUN "～" LOAD SAVE BLOAD BSAVE
MERGE KILL NAME SET VERIFY␣ON VERIFY␣OFF CHAIN
OPEN CLOSE PRINT # PRINT＃ USING WRITE＃ INPUT＃
LINE␣INPUT＃ FIELD PUT＃ GET＃ LSET RSET DEVO$ DEVI$

例外処理

ON␣ERROROR␣GOTO REUME ON～RESUME ERROR ON␣KEY␣GOSUB
KEY␣ON／OFF／STOP ON␣STOP␣GOSUB STOP␣ON／OFF／STOP
ON␣HELP␣GOSUB HELP␣ON／OFF／STOP ON␣COM␣GOSUB
COM␣ON／OFF／STOP ON␣MOUSE␣GOSUB MOUSE␣ON／OFF／STOP
ON␣TIMES␣GOSUB TIMES␣ON／OFF／STOP INTERVAL
PLAY␣ON／OFF／STOP

日本語処理命令

KMODE

マウス処理

MOUSE

キーボード処理

INIT "KB :" CLICK␣ON CLICK␣OFF REPEAT␣ON REPEAT␣OFF
KEY KEY␣LIST KEY␣0 KLIST KEY LLTIST

プリンタ制御

HCOPY LPOUT WIDTH␣LPRINT
INIT "LPT :"

サウンド処理

BEEP PLAY TONE TONE␣COPY TONE␣LFO SOUNE VOICE@

カセットデッキ制御

INIT "CMT :" CMT＝APSS

マシン語関連処理

DEF␣USR CALL OUT POKE POKE@

数値関数

ABS SGN INT FIX FRAC SIN COS TAN ATN SOR EXP
LOG LOG10 RAD PAI DEG FAC SUM RND CINT CSNG CDBL

文字列関数

LEFT$ RIGHT$ MID$ ASC CHR$ VAL STR$ LEN MIRROR$
SPACE$ STRING$ INSTR HEX$ COT$ BIN$ HEXCHR$
ASCCHR$ MKI$ MKS$ MKD$ CVI CVS CVD

日本語処理関数

AKCNV$ KACNV$ JIS$ KTEN KPOS

その他の関数

SEARCH POS SCRNS CGPAT$ MAP POINT ATTR$ EOF FPOS
LOF LOC DEVF INPUT$ MOUSE LPOS CMT PEEK USR INP
FRE STICK STRIG PEEK@

システム変数(代入可)

DATE$ DAY$ TIM$ TIME

システム変数(代入不可)

DTL CSRLIN PWD$ ERR ERR INKEY$

システム定数

SYSID$

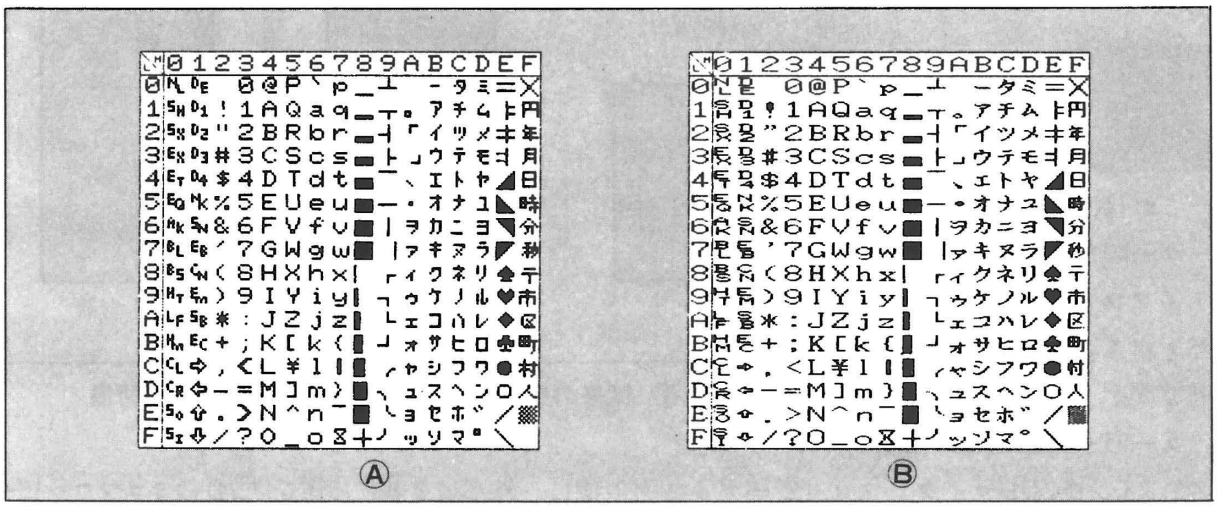
演算子・比較子

＋ － ＊ AND OR XOR EOV IMP ¥ MOD ∧ ＝ ＝＞ ＞＝ ＜＝
＝＜ ＞＜ ＜＞ NOT

その他

&B &H &O &J &K $ SPC TAB

《第３－４表》スーパーＭＺのキャラクタコード（80H～9FH，E0H～FCHは，漢字モードではシフトJISコードの１バイト目になります。KMODE　0を実行すると，これらセミグラフィックキャラクタになります）。Ⓐは8×8ドット、Ⓑは8×16ドットの時のフォント



従来互換のBASICも搭載

スーパーＭＺは，前項のように，マイクロソフト系の言語体系を持ったBASIC，**BASIC-M25**を装備していますが，この他に，従来のＭＺシリーズのBASICの体系を持った，**BASIC-S25**も装備しています。

BASIC-S25では，「CLEAR」は「CLR」，「CLEAR アドレス」は「LIMIT␣アドレス」，「FILFS」は「DIR」のように，予約語の名前が違うほか，「KEY1, "RUN" +CHR$(13)」が「DEF KEY(1)="RUN"+CHR$(13)」となるなど，文法的にも従来のＭＺのものに合わせる工夫がなされています。

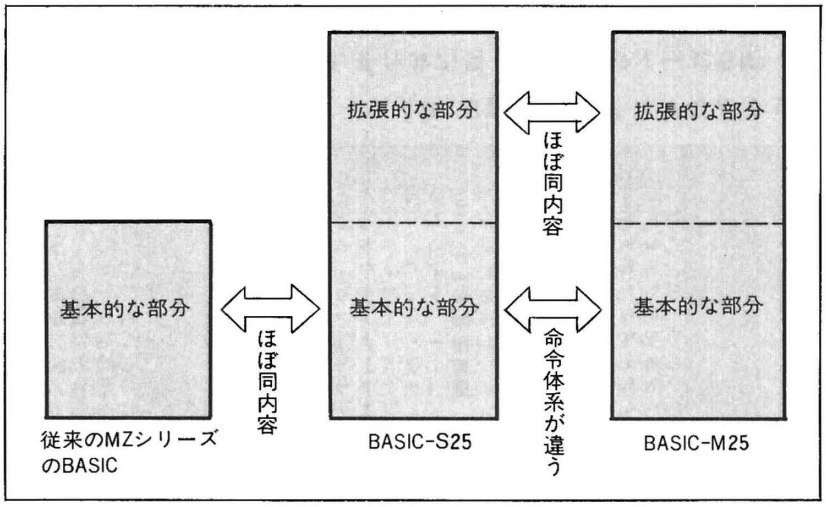
また，BASIC-M25の持つ，ほとんどすべての機能を持っており，全体的に見て，ほぼ遜色のないように仕上っています。また，従来のMZのBASICになくて，S25の方で拡張された部分は，M25の方と互換性があります。これらの関係を図で表わすと，**第３－８図**のようになります。



《写真３－２》スーパーＭＺに付属のBASIC。1枚のディスクにM25とS25の両方が入っている

ところで，BASIC-S25は，やはり，これまでMZ-2000や80ＢのBASICに慣れ親しんで来た人のためのBASICと言えるでしょう。スーパーＭＺが初めて，という人や，他のマイクロソフト系BASICを使ったことがある人，そして従来のＭＺでもハドソンのHuBASICを主に利用して来た人は，BASIC-M25を使うのが良いでしょう。今後発表されるBASICで書かれたソフトも，Ｍ

25の方で供給されることになると思います（本書もM25で統一しています）。

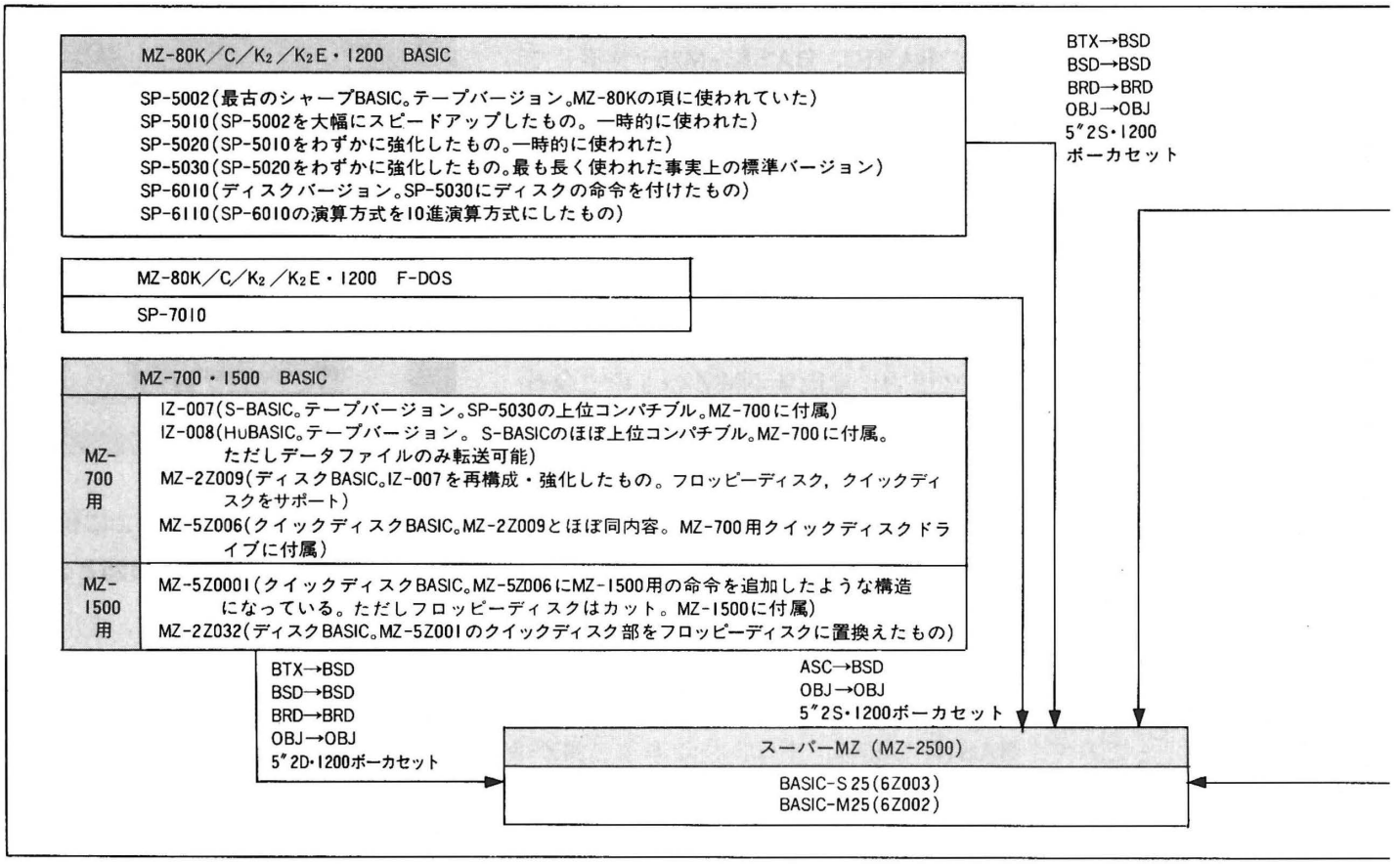


《第3－8図》従来のBASICとBASIC-M25・S25との関係

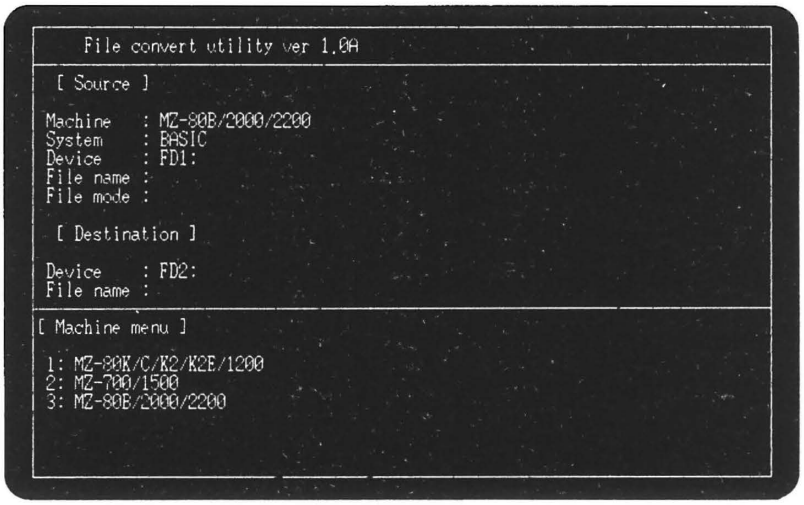
ファイルコンバータ

これは，従来のＭＺのユーザーのためのユーティリティソフトです。BASIC-S25と組み合わせて使うと便利です。

スーパーＭＺでは，内蔵の3.5インチ2DDマイクロフロッピーディスクの他に，外部に２基，５インチ2Dのミニフロッピーディスクを接続できることは，既に触れた通りですが，この状態ではフロッピーディスクとカセットテープから，また，スーパーＭＺ単体ではカセットテープだけから，従来のプログラムやデータをロードし，3.5インチ2DDの内蔵ディスクに転送するのが，ファイルコンバータの働きです(**画面３－２**)。



《第３－９図》ファイルコンバータでコンバート可能なファイル



《画面3－2》ファイルコンバータ使用中

転送元のシステムは

- MZ-2000・80Bの(ディスク)BASICで書いたBASICプログラム(BTX), データ(BSD, BRD), マシン語プログラム(OBJ)
- MZ-2000・80BのF-DOSで書いたプログラム・データ(ASC), マシン語プログラム(OBJ)
- MZ-700(S-BASIC) ／1500の(ディスク)BASICプログラム(BTX), データ(BSD, BRD), マシン語(OBJ)
- MZ-80K／C／K2／K2E／1200の(ディスク)BASICで書いたBASICプログラム(BTX), データ(BSD, BRD), マシン語(OBJ)

をサポートしています。8ビットのMZはすべて網羅しています。

これらから転送されたファイルは, BTX→BSDとなる, つまり普通に中間コードでバイナリセ

MZ-80B・2000・2200 BASIC	
MZ-80B用	SB-5520(標準的なテープバージョン。MZ-80Bに付属) SB-5620(SB-5520とほぼ同等だが, 演算方式が10進演算になっている) SB-6520(標準的なディスクバージョン。SB-6520にディスクの命令を付けたもの) SB-6620(SB-6520とほぼ同等だが, 演算方式が10進演算になっている) SB-6521(SB-6520にRS-232C, GPIB命令を付けたもの。MZ-2Z001の前身とも言える)
MZ-2000／2200用	MZ-1Z001(標準的なテープバージョン。MZ-2000／2200に付属) MZ-1Z002(MZ-1Z001にカラグラフィック機能を付けたもの。MZ-2200には付属) MZ-1Z003(MZ-1Z002とほぼ同等だが, 演算方式が10進演算になっている) MZ-1Z010(MZ-1Z001にRS-232C, GPIB, ディスク命令を付けたもの。MZ-2Z001と同内容だがテープで共給されている) IZ-012 (MZ-1Z002の8088バージョン。8088ボードに付属していた) MZ-2Z001(標準的なディスクバージョン。MZ-1Z001にディスク命令, RS-232C, GPIB命令を付けたもの) MZ-2Z002(MZ-2Z001にカラーグラフィック機能を付けたもの) MZ-2Z003(MZ-2Z002とほぼ同等だが, 演算方式が10進演算になっている MZ-2Z021(MZ-2Z002にグラフィック画面に漢字を描く機能を付けたもの) MZ-5Z007(MZ-2Z021とほぼ同等だが, フロッピーディスクの代りにクイックディスクをサポートしている。MZ-2000／2200用クイックディスクドライブに付属)

BTX→BSD
BSD→BSD
BRD→BRD
OBJ→OBJ
5″2D・2000ボーカセット

MZ-80B・2000・2200 F-DOS	
MZ-80B用	SB-7020
MZ-2000／2000用	MZ-2Z004

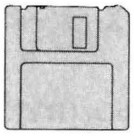
ASC→BSD
OBJ→OBJ
5″2D・2000ボーカセット

ーブしたプログラムファイルが，アスキーセーブした形に変化することを除けば，ファイルの内容が変わることはありません。**第3－9図**にこれらの関係をまとめておきます。

そして，これらのファイルは，データファイルはそのまま，BASICで書いたプログラムは，PEEK，POKEやハードウエアに密着した命令，文法の多少変わった命令をBASIC-S25用に手直しすることで利用できるようになります。ただし，マシン語プログラムは，よほど簡単なものでない限り全面的な書換えが必要でしょう。

ですから，特にMZ-80B・2000用のソフトを使いたいというのなら，それらのモードで，それら用のBASICを利用した方が良いと思います(その場合はそっくりそのまま動きます)。

なお，MZ-2000・80B・700・1500(ディスク)BASIC，MZ-2000・80BのF-DOSのカセットデータファイルとのファイルの読書きは，このユーティリティを使わなくとも，そのまま可能です。これは，ファイルの管理方法等が上位コンパチブルになっているからです。

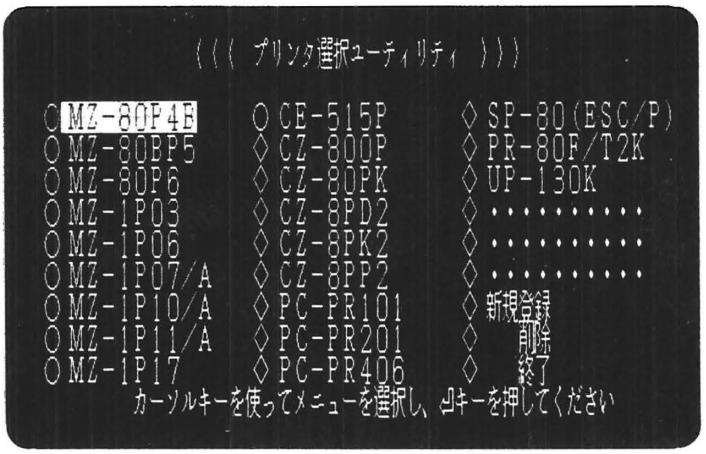


プリンタ選択ユーティリティ

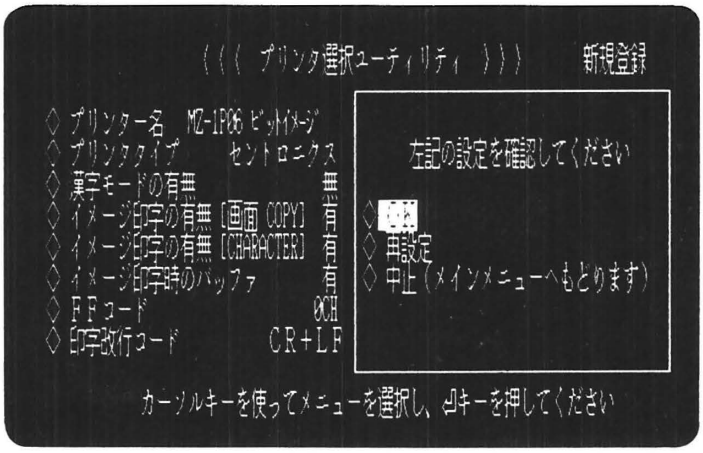
第2章のプリンタの所で説明したプリンタを選択・定義するユーティリティです。

既に入力されているプリンタを使う時は**画面3－3**のような状態からカーソルを動かして↵を押すだけで選択することができます。

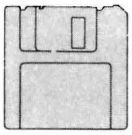
また，**画面3－4～3－7**は新たなデータを入力する例です。ここでは既に定義されている漢字プリンタ，MZ-1P06を，全く別の状態で使う(本体側のフォントで印字する)ための設定を行っています。新しいプリンタ以外にも，このような目的で利用することもできるのです。



《画面3－3》プリンタ選択ユーティリティ使用中

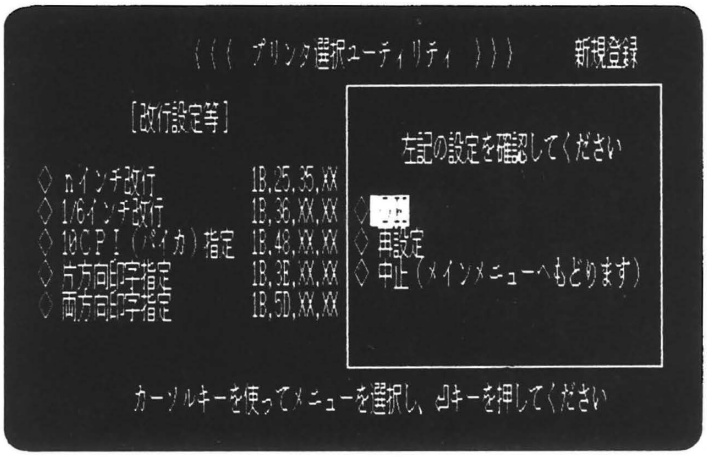


《画面3－4》画面3－3の状態から新規登録を選んで基本データを入力した所

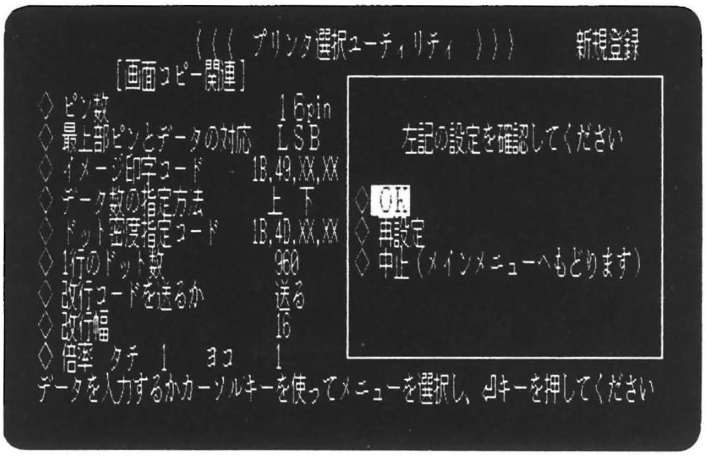


PCGエディタ

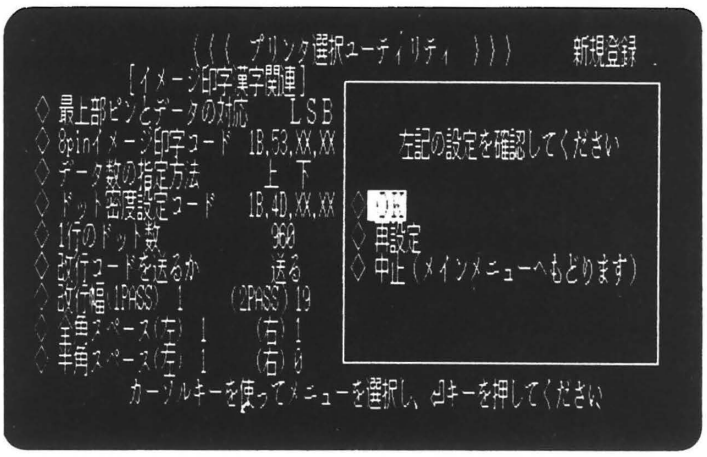
これは，スーパーMZの強力な機能の一つである。**PCG（プログラマブルキャラクタジェネレータ）**のフォントを効率良く作成するための，



《画面３－５》次に改行コード等を入力する



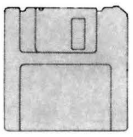
《画面３－６》次に画面コピーのデータを入力する



《画面３－７》最後に漢字のデータを入力する

便利なユーティリティです。

ドット毎の上下左右への移動やカラーのセット，ＰＣＧデータの表示その他たくさんの機能があり，２進数や16進数について良く知らない人にも，簡単にデータを作ることができます。



スーパーヘルプ機能

この機能は，BASICやデータベースから，いつでも呼び出すことのできるもので，キーボードの所で触れた「アルゴキー」を使用するものです。

●電卓機能

一度アルゴキーを押した後，電卓を選択すると，画面左上に電卓の絵が出て来ます。そして，ここで普通にテンキーの０～９や＋－＊／↵等を使って計算を行うことができます。BASICでもPRINT文を使えば良いのだから，そのくらいはカンタンじゃないか！と言われる方もあるかと思いますが，INPUT文でデータを入力している時には，それはできません。それにプログラム作成中にも思わず「PRINT」を付け忘れて，プログラムの一部を消してしまったという体験をお持ちの方も多いのではないでしょうか。

この電卓は12桁，メモリ付きで，しかも，16進↔10進の相互変換・混合計算も可能です。もちろん16進数同士の＋－＊／もできます。また，パソコンの16進数は$FFFF_H$のように４桁までしか扱えませんが，ここでは16進数も12桁，扱うことができ

ますので，たいへん便利です。

それに，計算結果をそのままINPUT 文での入力値として入力することもにできます。

私の知人で，パソコンに電卓をテープで止めておき，いつも一諸に持ち運んでいる人がいるのですが，スーパーMZなら，これも不要でしょう。

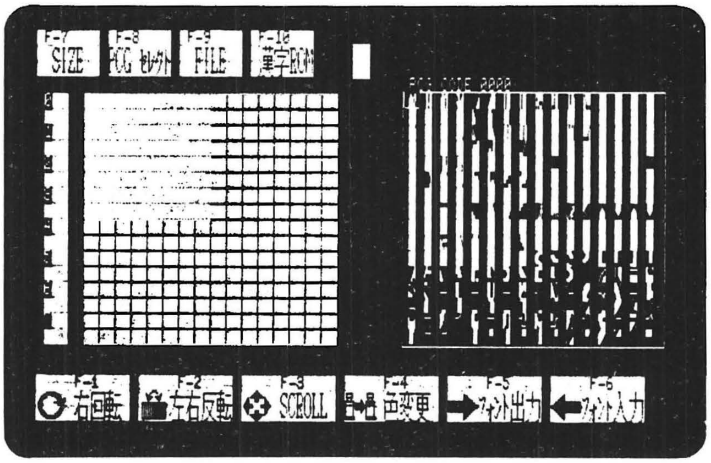
●カラーシミュレーション機能

スーパーMZは，第2章でも触れたように16色・256色，4096色(オプション) もの色を扱うことができます。

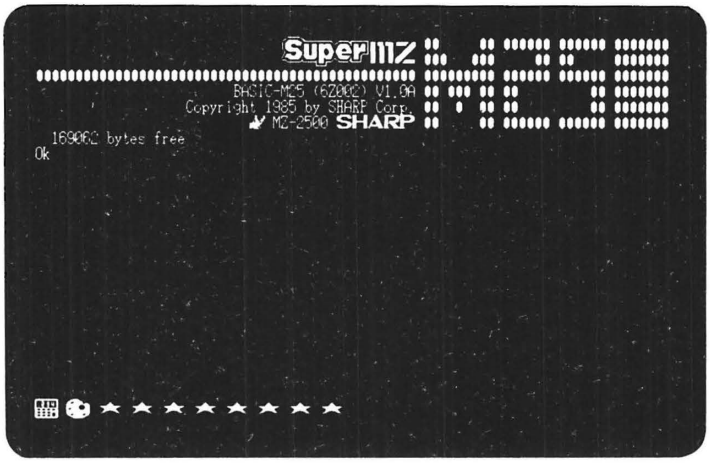
そこで，プログラムをわざわざ作ったりしなくとも，これらの色が見られるように考えられたのがこの機能です。まず，通常の16色のモードは，**画面3－11**のように16個の長方形が出て来ます。それぞれの中はパレットコード0～15の色でぬりつぶされており，←→キーでそれらを選択し↑↓キーを押すと，そのパレットの表示色が次々と変わります。既に描いたカラーの絵も，グラフィックのカラーコードを修正してRUNし直す，といったことをしなくても，その場で簡単に色調を変えるみることができます。

次に，256色モードでは長方形は三つです(**画面3－12**)。この3という数字は特に意味はなく，単にスペースの都合でそうなったのでしょう。

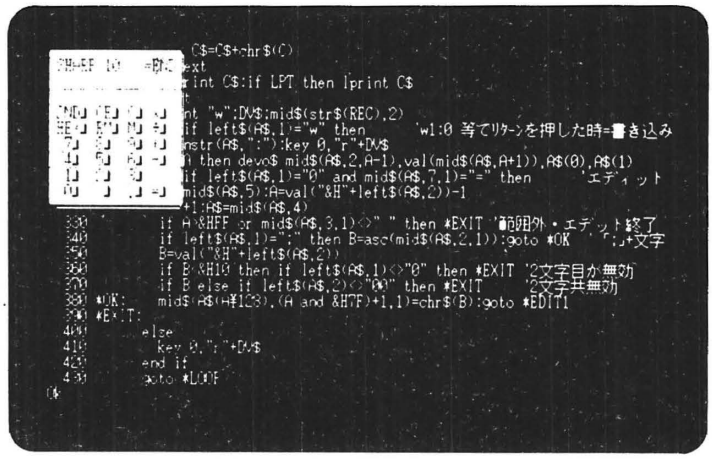
色が三原色でできているのは御存知だと思いますが，このモードでは，その原色，つまり青，赤，緑の混合比を少し($\frac{1}{8}$)ずつ変えて，色サンプル，を表示することができるのです。混



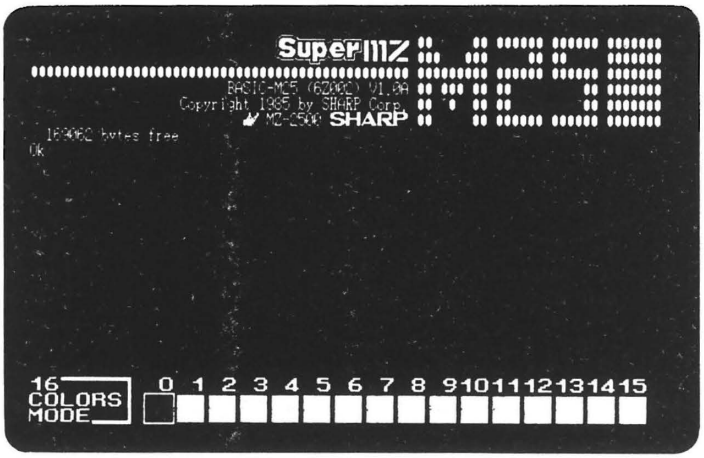
《画面3－8》PCGエディタ使用中



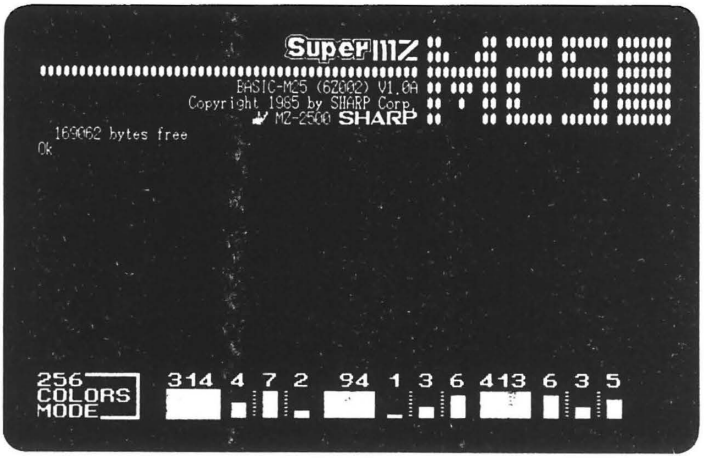
《画面3－9》アルゴキーを押した所（左は電卓，右はパレットです）



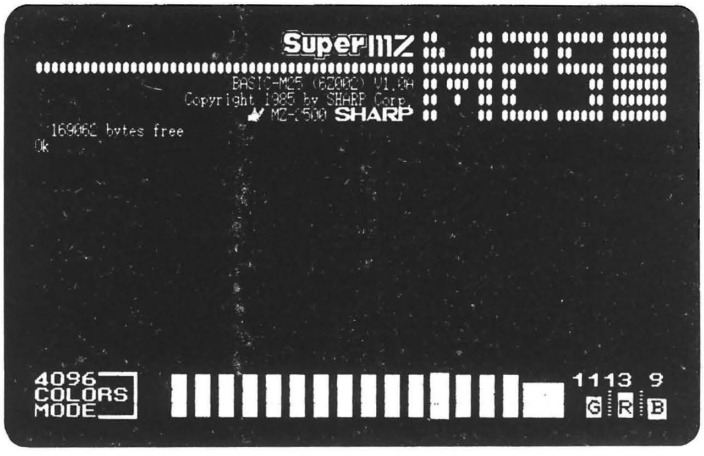
《画面3－10》プログラム作成中に電卓を呼び出してみた所



《画面３－11》16色モードでのカラーシミュレーション



《画面３－12》256色モードでのカラーシミュレーション



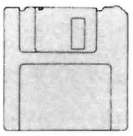
《画面３－13》4096色モードでのカラーシミュレーション

合比は画面のように棒グラフ状に出て来ますので感覚もつかみやすく，スピーディに目的の色を得ることができるでしょう。

そして，4096色時は**画面３－13**のようになり，4996色のパレットが適用される15色分全部が表示され，256色の時と同じように，混合比を少し(こちらは$\frac{1}{16}$)ずつ変えて見ることができます。ただし，「棒グラフ」は各々についてでなく，現在変更中のパレットの分だけになります。

●オートダイヤラー機能

これは，スーパーＭＺの大きな特長の一つである，モデムホンを使ったデータ通信機能をサポートするもので，こちらが選んだ相手に自動的に電話をかけてくれるものです。これについては次章でも触れています。



ロードテスト

第３章ソフトウエア編の締めくくりとして，BASIC-M25の実行速度を他のパソコンと比べてみることにしましょう。

第３－５表を見て下さい。これは「月刊マイコン」で度々使われているプログラムでテストしたものです。また，参考として16ビット機のＰＣ－9801Ｍ２のデータを1985年７月号付録Ｐ10～Ｐ12の記事から，引用しています。元々のプログラムはN88 BASIC用なので，他のパソコンでは，文法等が異なるものがあるため，テストの主旨を変えない範囲で，ごく

```
Directory of FD1:*  68 KB free.
 OBJ* "BASIC-M25(6Z002)"'85/09/03 00:00  BASIC-M25本体
 OBJ* "BASIC-S25(6Z003)"'85/09/03 00:00  BASIC-S25本体
 OBJ* "algo manager"     '85/09/03 10:00  アルゴ機能の中心部
 OBJ* "auto dialer"      '85/09/03 00:00  アルゴ機能のオートダイヤラー
 BTX  "auto-run.m25"     '85/09/04 10:30  BASIC-M25実行時に自動的に実行されるプログラム
 BTX  "auto-run.s25"     '85/09/04 10:16  BASIC-S25実行時に自動的に実行されるプログラム
 OBJ* "color simulation"'85/09/02 00:00  アルゴ機能のカラーシミュレーション
 BSD  "dial"             '85/09/02 00:00  オートダイヤラー用のデータ
 BTX* "pcg ｴﾃﾞｨﾀ.m25"    '85/09/03 13:30  BASIC-M25用  PCGエディタ
 BTX* "pcg ｴﾃﾞｨﾀ.s25"    '85/09/03 13:44  BASIC-S25用  PCGエディタ
 OBJ* "電卓(calculator)"'85/09/02 00:00  アルゴ機能の電卓
 DIR  "ﾃﾞﾓﾌﾟﾛｸﾞﾗﾑ.dir"  '85/09/02 00:00  デモプログラム用サブディレクトリ（第3-11図参照）
 BTX* "ﾃﾞﾓﾌﾟﾛｸﾞﾗﾑ.main" '85/09/02 00:00  デモプログラム本体
 OBJ* "ﾌｧｲﾙ･ｺﾝﾊﾞｰﾀ.obj" '85/09/02 00:00  ファイルコンバータサブルーチン(マシン語)
 BTX* "ﾌｧｲﾙ･ｺﾝﾊﾞｰﾀ.s25" '85/09/02 00:00  ファイルコンバータ本体(BASIC-S25専用)
 BTX* "ﾌｫｰﾏｯﾄ&ｺﾋﾟｰ.m25" '85/09/04 11:44
 BTX* "ﾌｫｰﾏｯﾄ&ｺﾋﾟｰ.msub"'85/09/04 11:38  BASIC M25で利用するディスクの初期化コピーユーティリティ
 OBJ* "ﾌｫｰﾏｯﾄ&ｺﾋﾟｰ.obj" '85/09/04 11:39
 BTX* "ﾌｫｰﾏｯﾄ&ｺﾋﾟｰ.s25" '85/09/04 11:40  BASIC-S25
 BTX* "ﾌｫｰﾏｯﾄ&ｺﾋﾟｰ.ssub"'85/09/04 11:40  マシン語サブルーチン(M25用・S25用から共用される)
 BTX* "ﾌﾟﾘﾝﾀ 選択.m25"  '85/09/03 14:15
 BTX* "ﾌﾟﾘﾝﾀ 選択.msub" '85/09/03 14:29  BASIC-M25で利用するプリンタユーティリティ
 BTX* "ﾌﾟﾘﾝﾀ 選択.s25"  '85/09/03 14:15
 BTX* "ﾌﾟﾘﾝﾀ 選択.ssub" '85/09/03 14:31  BASIC-S25
 BRD  "ﾌﾟﾘﾝﾀ･ﾃﾞｰﾀ"      '85/09/04 13:28  プリンタ制御コード等が入っている(M25用・S25用から共用される)
```

《第3－10図》BASICのディスクに入っているファイル

一部に手を加えたものもあります（これによってその機種が有利にならないように気を付けています)。

スーパーMZで使用したプログラムは**リスト3－1**に示します。それと，元々のプログラムのグラフィックのテスト部は640×400ドット×8色(以上)のグラフィックができるものを対象にしていますので，640×200×8色(以上)のテスト用に27'～29'を設け，比較してみました。PC－9801M2についても，この部分だけ追加テストしています。

さて，表を見る限りでは，スーパーMZのスピードはやはり速いの一言に尽きるようです。また**第3－6表**でハード・ソフトの機能を比較してみましたが，機能面・速度面共に頂点に立つ8ビット機と考えて間違いないようです。今後もスーパーMZを追いかけて，多くのマシンがデビューすることでしょうが，スーパーMZの優位性は，ここ当分，揺らぎそ

```
Directory of ﾃﾞﾓﾌﾟﾛｸﾞﾗﾑ.dir/*  68 KB free.
 DIR* "."                '85/09/02 00:00
 DIR* ".."               '85/09/02 00:00
 OBJ* "ANIME.PAC"        '85/09/02 00:00
 OBJ* "BALL.PAC1"        '85/09/02 00:00
 OBJ* "BALL.PAC2"        '85/09/02 00:00
 OBJ* "BALL.PCG"         '85/09/02 00:00
 BTX* "DANCE"            '85/09/02 00:00
 OBJ* "DANCE.PAC"        '85/09/02 00:00
 BRD* "DANCE.PIC"        '85/09/02 00:00
 BTX* "DEMONSTRATION1.0" '85/09/02 00:00
 BRD* "EXMAN.PIC"        '85/09/02 00:00
 BRD* "EXMOU.PIC"        '85/09/02 00:00
 BTX* "GAME.BAS"         '85/09/02 00:00
 OBJ  "GAME.DAT"         '85/09/02 00:00
 OBJ* "GAME.MAP"         '85/09/02 00:00
 OBJ* "GAME.PCG"         '85/09/02 00:00
 BRD* "LASVEGAS1.PIC"    '85/09/02 00:00
 BRD* "LASVEGAS2.PIC"    '85/09/02 00:00
 BRD* "LASVEGAS3.PIC"    '85/09/02 00:00
 BTX* "PROG.a"           '85/09/02 00:00
 BTX* "PROG.b"           '85/09/02 00:00
 BTX* "PROG.c"           '85/09/02 00:00
 BTX* "PROG.e"           '85/09/02 00:00
 BRD* "SHARP.PIC"        '85/09/02 00:00
 BTX* "SHOW TIME"        '85/09/02 00:00
 BRD* "SIGN1.PIC"        '85/09/02 00:00
 BRD* "SIGN2.PIC"        '85/09/02 00:00
 BTX* "SLOT.BAS"         '85/09/02 00:00
 OBJ* "SLOT.PCG"         '85/09/02 00:00
 BRD* "SLOT1.PIC"        '85/09/02 00:00
 BRD* "SLOT2.PIC"        '85/09/02 00:00
 BRD* "SLOT3.PIC"        '85/09/02 00:00
 OBJ* "STAGE MAP"        '85/09/02 00:00
 BRD* "SUPER MZ.PIC"     '85/09/02 00:00
```

《第3－11図》第3－10図の「デモプログラム.dir」というサブディレクトリ内に含まれるファイル

うにありません。なお，スーパーＭＺの高速処理のヒミツはグラフィックでは専用ＬＳＩを開発したためBASICではその画期的な中間コードの構造にあるようです。中間コードについては第８章「ソフトウエア内部解析」で詳細に紹介していますので興味のある方はぜひ，御覧になって下さい。

《第３－５表》ロードテスト結果(秒)

		スーパーMZ (MZ-2500)	Xlturbo	PC-8801 mkIISR (V2モード)	FM-77L2	PC-9801 M2(参考)
1	FOR～NEXT 10万回	66	74	162	91	53
2	２重ループによる10万回ループ	23	78	166	101	52
3	WHILE～END３万回ループ	49(41)	125	150	100	61
4	整数型の場合のWHILE～ENDループ	47(40)	114	140	95	58
5	GOTOによる分岐テスト１万回	9(5)	11	28	21	11
6	GOSUBによる分岐テスト１万回	10(6)	14	29	16	11
7	足し算１万回	18(13)	37	46	38	16
8	引き算１万回	17(13)	37	46	38	16
9	掛け算１万回	25(21)	47	57	39	16
10	割り算１万回	24(20)	48	69	47	21
11	べき乗１万回	33(28)	66	181	145	48
12	SQR１万回	49(45)	287	361	153	74
13	LOG１万回	94(89)	167	188	67	43
14	SIN １万回	78(74)	183	159	88	62
15	COS１万回	96(92)	181	180	91	80
16	TAN１万回	168(163)	349	332	146	137
17	ATN１万回	39(35)	74	241	102	54
18	EXP１万回	16(12)	164	176	93	41
19	READ文による文字列読取り１万回	15(11)	162	532	76	18
20	READ文による数値読取り１万回	20(16)	532	565	78	21
21	配列への文字格納１万回	21(14)	57	48	41	17
22	配列への数値格納１万回	24(17)	53	51	39	17
23	FDDを使った文字列の読書き１万回	14	17	70	16	7
24	FDDを使った数値の読書き１万回	19	23	43	20	10
25	画面への文字表示速度	64	156	164	134	65
26	PSET文の速度	37	126	166	65	45
27	LINE文の速度　(400ライン)	13	105	—	—	11
28	CIRCLE文の速度　(400ライン)	20	84	—	—	25
29	PAINT文の速度　(400ライン)	12	74	—	—	14
27′	LINE文の速度　(200ライン)	7	51	11	21	7
28′	CIRCLE文の速度　(200ライン)	21	64	52	38	20
29′	PAINT文の速度　(200ライン)	6	38	4	4	7

※スーパーＭＺの(　)内は1200行～4360行の中にある！を取って実行したものです。元々のプログラムはある程度マイクロソフトBASICに有利なように作成されており，！を付けた方が速くなりますが，スーパーＭＺでは逆に！を取った方が速くなります。なお，1030行は元々のプログラムでは！が付いていますが，これを付けて入力してもスーパーＮＺではLIST時にカットされて出力されます。また，！を付けずに入力しても実行結果は変わりません。

※テスト23はフォーマット直後のデータディスク（何もファイルが入っていないディスク）で行っています。これは，ファイルが既に入っている場合には，その量によって実行速度が違って来るためです。

※１～24は起動時のスムーズスクロールモードのままＲＵＮさせて計ったものですが，25以降はラインスクロールモードにしないとCONSOELE文がエラーになるため，5015行にリストにある様なステートメントを入れていますから，プログラムの5015行以降を実行した後（テストが終了した後など）に再び１～24のテストを行う場合にはINIT "",,1" を実行してスムーズスクロールモードに戻して再実行するか，BASICを起動しなおして実行して下さい。なお，テスト23以降を実行した場合も，KILL "TEMP" として下さい。

《第3－6表》ハード・ソフトの主な機能の比較

	機　種　名　（メーカー）	MZ-2500 model 30(シャープ)	
ハードウエア面	CPU・クロック	Z-80B (6MHz)	
	メインRAM容量(標準時)	128K	
	メインRAM容量(拡張時。ただしBASICで利用できない分は除く)	256K	
	グラフィックRAM容量(標準時)	64K	
	グラフィックRAM容量(拡張時)	128K	
	キャラクタV-RAM容量	2K	
	アトリビュートV-RAM容量	2K	
	漢字V-RAM容量	2K	
	外字・PCG RAM容量	8K	
	グラフィック640×400ドットモードG-RAM標準実装時カラーページ／拡張時カラーページ	4色×1ページ／16色×1ページ	
	グラフィック640×200ドットモードG-RAM標準実装時カラーページ／拡張時カラーページ	16色×1ページ／16色×2ページ	
	グラフィック320×200ドットモードG-RAM標準実装時カラーページ／拡張時カラーページ	256色×1ページ、16色×2ページ／256色×2ページ、16色×4ページ	
	漢字表示能力(グラフィック使用時　横×縦)	40桁×25行	
	漢字表示能力(漢字V-RAM使用時　横×縦)	40桁×25行	
	全角外字個数(16×16ドット)	256個	
	カラー(ドット単位着色)PCG個数(8×8ドット換算)	256個	
	パレット機能	16色中16色(オプションで4096色中15色)	
	プログラム用ROM容量	32K	
	キャラクタジェネレータROM容量(標準装備の漢字ROM含む)	128K	
	JIS第1水準の漢字	○	
	JIS第2水準の漢字	○	
	フロッピーディスク	3.5"2DD(640K)×2	
	サウンド機能	6和音(うち3和音はFM音源)	
	ジョイスティックポート	2ポート	
	RS-232Cインターフェース	2ポート(うち1ポートはマウスと共用)	
	マウスインターフェース	○	
	バッテリバックアップカレンダクロック	○	
	拡張I／Oポート	オプション(2ポート)	
	その他	アナログ音声対応カセットデッキ内蔵 モデムホン対応	
ソフトウエア面	四則演算桁数(最大)	16	
	関数桁数(最大)	16	
	変数名識別長	200	
	漢字変数名	○	
	ラベル	○	
	漢字ラベル	○	
	"～"の中での漢字使用	○	
	ブロックIF	○	
	エディット時の変数保護	○	
	カベージコレクション対策	○	
	ユーザー座標系	○	
	タイリングペイント・ラインスタイル	○	
	ペンスタイル	○	
	シンボル機能	○	
	漢字シンボル機能	○	
	エディットモード	○	
	サーチ機能	○	
	ブレークポイント・シングルステップデバッグ機能	○	
	階層ディレクトリ	○	
	漢字入力方式	1字変換・文節変換(オプション)	

※オプションは，BASICでサポートしているもののみ取り上げています。

	X1turbo model 30(シャープ)	PC-8801mkⅡSRmodel 30(日本電気)	FM-77L2（富士通）
	Z-80A (4MHz)	Z-80Aコンパチ(4MHz)	MB-6809(8MHz)
	64K	61K（相当）	64K（うち32Kのみ使用）
	96K	48K	48K
	2K	2K	なし(仮想的に2K使用)
	2K	1K	なし(仮想的に2K使用)
	2K	なし	なし(仮想的にもなし)
	6K	なし	なし
	8色×1ページ	モノクロ(2色)	(このモードは存在しない)
	8色×2ページ	8色×1ページ	8色×1ページ
	8色×4ページ	(このモードは存在しない)	(このモードは存在しない)
	40桁×25行	40桁×25行	40桁×12行
	40桁×25行	不可	不可
	64個	なし	なし
	256色	なし	なし
	8色中8色	8色中8色	8色中8色
	32K	96K（N-BASIC,N88BASIC×2モード分）	40K
	134K（実質的には128K程度）	130K（実質的には128K程度）	130K（実質的には128K程度）
	○	○	○
	オプション		
	5"2D (320K)×2	5"2D (320K)×2	3.5"2D (320K)×2
	3和音	6和音(うち3和音はFM音源)	3和音(オプションでFM音源)
	2ポート		オプション
	1	1	オプション
	○		
	○	○	
	2ポート	3	2ポート
	スーパーインポーズ・テロッパ内蔵 ハードディスク対応(BASICで)		
	16	16	16
	16	16	8
	無制限(250)	40	16
	△		
	○	○	
	○		
	○		
	○		
	○		○
	○	○	
	○	○	
	○		○
	○		
	○	○	
	○		
	○		
	1字変換・音読み訓読変換・熟語変換		

《リスト3－1》ロードテストに使用したテストプログラム(スーパーMZ用)

```
1000 '/*    TEST1    */
1010 '
1020 time$="00:00:00"
1030 for I=0 to 100000
1040 next I
1050 print time$
1060 '
1070 '/*    TEST2    */
1080 '
1090 time$="00:00:00"
1100 for I=0 to 1000
1110 for J=0 to 100
1120 next J
1130 next I
1140 print time$
1150 '
1160 '/*    TEST3    */
1170 '
1180 time$="00:00:00"
1190 I=0
1200 while I<30000!
1210 I=I+1
1220 wend
1230 print time$
1240 '
1250 '/*    TEST4    */
1260 '
1270 time$="00:00:00"
1280 I%=0
1290 while I%<30000!
1300 I%=I%+1
1310 wend
1320 print time$
2000 '/*    TEST5    */
2010 '
2020 time$="00:00:00"
2030 for I=0 to 10000!
2040 goto 2045
2045 goto 2050
2050 next I
2060 print time$
2070 goto 2100
2090 '
2100 '/*    TEST6    */
2110 '
2120 time$="00:00:00"
2130 for I=0 to 10000!
2140 gosub 2180
2150 next I
2160 print time$
2170 goto 2200
2180 return
2190 '
2200 '/*    TEST7    */
2210 '
2220 time$="00:00:00"
2230 A=.999999:S=0
2240 for I=0 to 10000!
2250 S=S+A
2260 next I
2270 print time$,
2280 print S
2290 '
2300 '/*    TEST8    */
2310 '
2320 time$="00:00:00"
2330 A=.999999:S=0
2340 for I=0 to 10000!
2350 S=S-A
2360 next I
2370 print time$,
2380 print S
2390 '
2400 '/*    TEST9    */
2410 '
2420 time$="00:00:00"
2430 A=.999999:S=1
2440 for I=0 to 10000!
2450 S=S*A
2460 next I
2470 print time$,
2480 print S
2490 '
2500 '/*    TEST10   */
2510 '
2520 time$="00:00:00"
2530 A=.999999:S=1
2540 for I=0 to 10000!
2550 S=S/A
2560 next I
2570 print time$,
2580 print S
2590 '
2600 '/*    TEST11   */
2610 '
2620 time$="00:00:00"
2630 A=.999999:S=1
2640 for I=0 to 10000!
2650 S=S^A
2660 next I
2670 print time$,
2680 print S
2690 '
3000 '/*    TEST12   */
3010 '
3020 time$="00:00:00"
3030 A=10000
3040 for I=0 to 10000!
3050 B=sqr(A)
3060 next I
3070 print time$,
3080 print B
3090 '
3100 '/*    TEST13   */
3110 '
3120 time$="00:00:00"
3130 A=10000
3140 for I=0 to 10000!
```

```
3150 B=log(A)
3160 next I
3170 print time$,
3180 print B
3190 '
3200 '/*   TEST14  */
3210 '
3220 time$="00:00:00"
3230 A=1
3240 for I=0 to 10000!
3250 B=sin(A)
3260 next I
3270 print time$,
3280 print B
3290 '
3300 '/*   TEST15  */
3310 '
3320 time$="00:00:00"
3330 A=1
3340 for I=0 to 10000!
3350 B=cos(A)
3360 next I
3370 print time$,
3380 print B
3390 '
3400 '/*   TEST16  */
3410 '
3420 time$="00:00:00"
3430 A=1
3440 for I=0 to 10000!
3450 B=tan(A)
3460 next I
3470 print time$,
3480 print B
3490 '
3500 '/*   TEST17  */
3510 '
3520 time$="00:00:00"
3530 A=1
3540 for I=0 to 10000!
3550 B=atn(A)
3560 next I
3570 print time$,
3580 print B
3590 '
3600 '/*   TEST18  */
3610 '
3620 time$="00:00:00"
3630 A=1
3640 for I=0 to 10000!
3650 B=exp(A)
3660 next I
3670 print time$,
3680 print B
3690 '
4000 '/*   TEST19  */
4010 '
4020 time$="00:00:00"
4030 for I=0 to 10000!
4040 restore 4080
4050 read A$
4060 next I
4070 print time$
4080 data "ﾏｲｺﾝ"
4090 '
4100 '/*   TEST20  */
4110 '
4120 time$="00:00:00"
4130 for I=0 to 10000!
4140 restore 4180
4150 read A
4160 next I
4170 print time$
4180 data 1234
4190 '
4200 '/*   TEST21  */
4210 '
4220 dim A(100)
4230 time$="00:00:00"
4240 for I=0 to 100!
4250 for J=0 to 100!
4260 A(J)=J
4270 next J
4280 next I
4290 print time$
4300 '
4310 '/*   TEST22  */
4320 '
4330 dim A$(100)
4340 time$="00:00:00"
4350 for I=0 to 100!
4360 for J=0 to 100!
4370 A$(J)="A"
4380 next J
4390 next I
4400 print time$
4410 '
4420 '/*   TEST23  */
4430 '
4440 time$="00:00:00"
4450 A$="ﾏｲｺﾝ"
4460 open "O",#1,"TEMP"
4480 for I=0 to 1000
4490 print #1,A$
4500 next I
4505 close:open "I",#2,"TEMP"
4510 for I=0 to 1000
4520 if eof(2) then 4550
4530 input #2,A$
4540 next I
4550 print time$:close:kill "TEMP"
4560 '
4570 '/*   TEST24  */
4580 '
4590 time$="00:00:00"
4600 A=1234
4610 open "O",#1,"TEMP"
4630 for I=0 to 1000
4640 print #1,A
4650 next I
```

```
4655 close:open "I",#2,"TEMP"
4660 for I=0 to 1000
4670 if eof(2) then 4700
4680 input #2,A
4690 next I
4700 print time$:close
4710 '
5000 '/*   TEST25  */
5010 '
5015 klist 0:init "CRT1:,,,0"
5020 width 80,25:console 0,25
5030 time$="00:00:00"
5040 for I=0 to 10
5050 cls
5060 for J=0 to 24
5070 for K=0 to 79
5080 locate K,J:print "*";
5090 next K
5100 next J
5110 next I
5120 print time$
5130 '
5140 '/*   TEST26  */
5150 '
5155 init "CRT2:640,400,16"
5160 width 80,25:console 0,25
5170 cls 3
5180 time$="00:00:00"
5190 for I=0 to 99
5200 for J=0 to 199
5210 pset (J,I),2
5220 next J
5230 next I
5240 print time$
5250 '
5260 '/*   TEST27  */
5270 '
5280 width 80,25:console 0,25
5290 cls 3
5300 time$="00:00:00"
5310 for I=0 to 399
5320 line (0,I)-(639,I),1
5330 next I
5340 for I=0 to 639
5350 line (I,0)-(I,399),2
5360 next I
5370 print time$
5380 '
5390 '/*   TEST28  */
5400 '
5410 width 80,25:console 0,25
5420 cls 3
5430 time$="00:00:00"
5440 for I=0 to 199
5450 circle (320,200),I,5
5460 next I
5470 print time$
5480 '
5490 '/*   TEST29  */
5500 '
5510 width 80,25:console 0,25
5520 cls 3
5530 time$="00:00:00"
5540 for I=0 to 10
5550 line (0,0)-(639,399),5,B
5560 paint (320,200),2,5
5570 cls 3
5580 next I
5590 print time$
```

※　このプログラムを実行すると，時計が変わります。実行終了後，TIME＄＝で現在の時刻をセットしなおして下さい。

※　27'～29'は　5155行　の　400→200に
5310行　の　399→199に
5350行　の　399→199に
5450行　の　200→100に
5550行　の　399→199に
5560行　の　200→100に
それぞれ変えたものです。

※　27～29はグラフィックRAMを増設している時にのみ実行できます。ただし，
5155行　の　16→4 に
5450行・5550行・5560行の 5 → 1 または 3
に変えますと，標準実装時にも実行できます。この場合の実行時間は
16色モードの時と同じになります



第4章 テレホンソフトのすべて

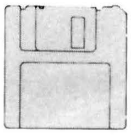
スーパーＭＺを単なるパソコンとしてとらえた場合の性能は，すでに第２章・第３章でお伝えしてきた通りです。しかし，スーパーＭＺのもう一つの顔は，「とにかく通信に強い」ということです。本機は，電話による通信を，設計段階から盛り込んだ日本で唯一のパソコンであると言っても良いでしょう。

そして，これはハードウエアに限ったことではなく，ソフトの方にも言えることです。スーパーＭＺには大変強力な「テレホンソフト」が付いていて，初めての人でもその日からパソコン通信を行うことができるのです。

本章では，この「テレホンソフト」にスポットを当ててみることで，スーパーＭＺのパソコン通信の能力を探ってみたいと思います。



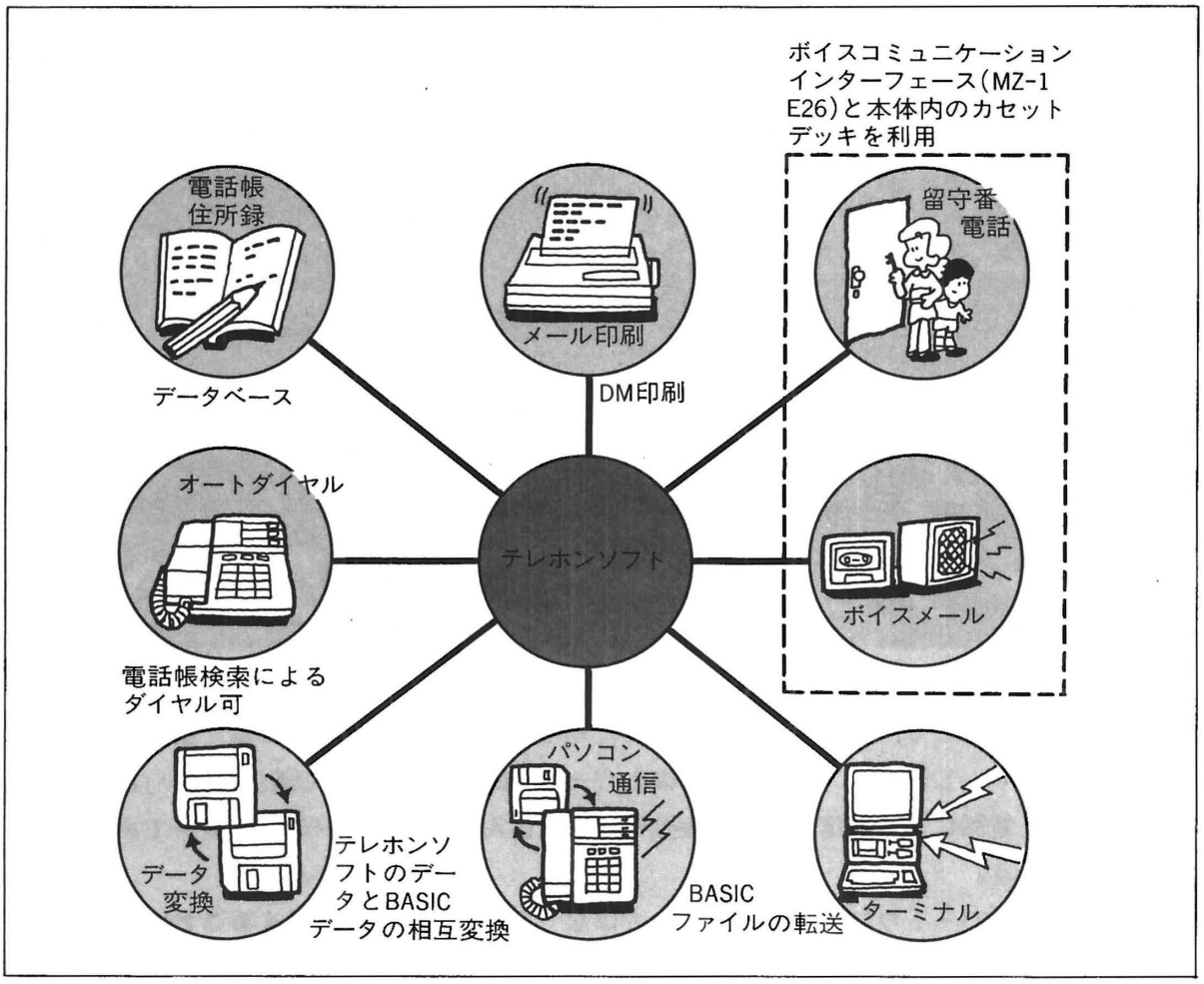
▲スーパーMZの特徴の一つ「テレホンソフト」



通信を行うのに必要なもの

スーパーＭＺを使ってパソコン通信を行う場合，必要となるのはオプションの**モデムホンMZ-IX19**だけです。これをスーパーMZのRS-232CのＡポートに接続すれば，すぐ通信が可能となります。

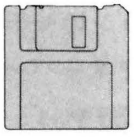
また，データベースをアクセスするような場合，ボーレイト，パリティ，ビット長，ストップビット，日本語処理方式その他の相異点があり、これらをきちんと設定する必要がありますが，「テレホンソフト」を使



《第4-1図》テレホンソフトの概要

えば，このようなことはいとも簡単に行えます。それに「テレホンソフト」には，電話を中心として考え得る機能はすべてと言っても良いくらい備えており，誰でも手軽に扱えるようになっています。

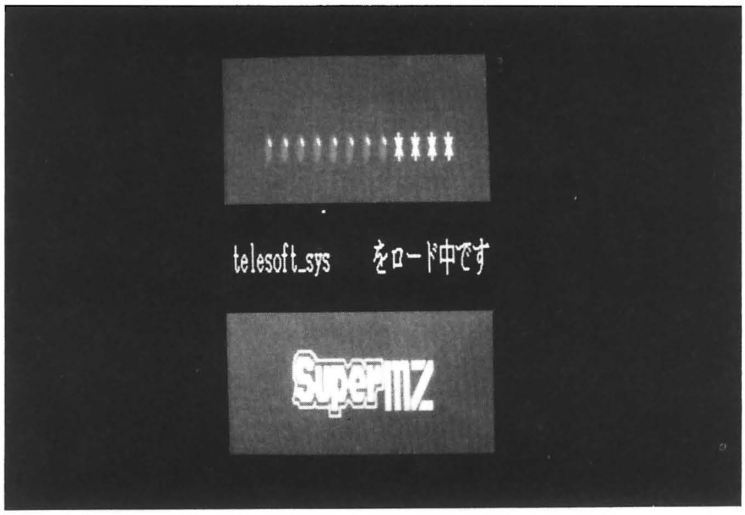
なお，「テレホンソフト」は，一部機能を除き，純正モデムホンだけでなく，音響カプラでも使用できるようになっています。



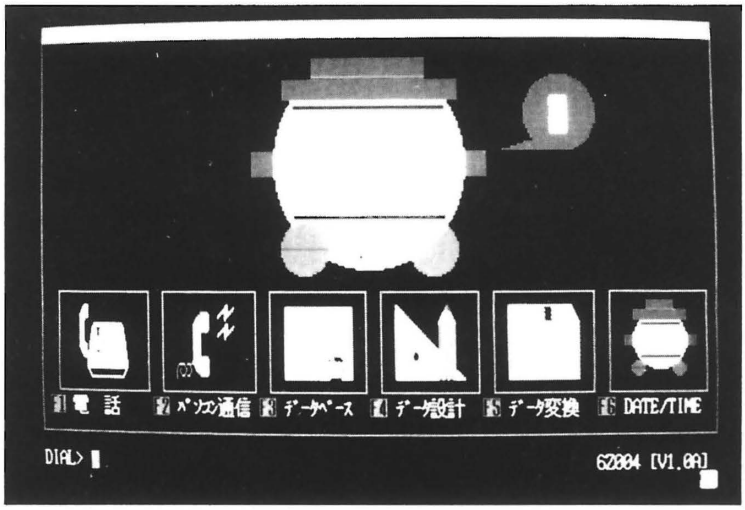
メインメニュー

テレホンソフトはオールマシン語でできており，合計すると約200 Kバイトを越えるシステムになっています。スーパーＭＺのディスクは２ＤＤですので，ディスクにはまだ400 Kバイト以上の空きエリアが存在しますが，そこにはユーザーのデータを自由に登録することができます。これでも足りないなら２ドライブ目も使えますが，普通は必要ないかと思います。

さて，「テレホンソフト」を起動させるには，面倒な操作はいりません。「テレホンソフト」のディスクをドライブ１に入れ，電源を入れるだけです。すると画面が**画面４－１**のように変化し，



《画面4-1》テレホンソフトをロード中



《画面4-2》テレホンソフトのメインメニュー

約20秒間で「テレホンソフト」が起動します(**画面４－２**)。メインメニューの状態となるわけです。

まず，この状態では，６個の仕事が選べます。「テレホンソフト」も例にもれず，メニューからサブメニュー，サブメニューからさらに細かなメニュー，と言った具合いに目的別に枝分かれしてゆく形式を採っているので，ここではまず，大まかな選択を行うのです。

６個の仕事とは，

①電話(電話帳)
②パソコン通信
③データベース
④データ設計
⑤データ変換
⑥DATE／TIME

というものです。

また，画面の中央に見えるのは時計（デジタル時計）です。メインメニューと④のサブメニューの状態では，いつでも現在時刻を表示しているのです。

それから，画面左下には

DIAL>■

と出ていますが，これは，ダイレクトに電話をかけられる状態であることを示しています。例えば

03－445－6111

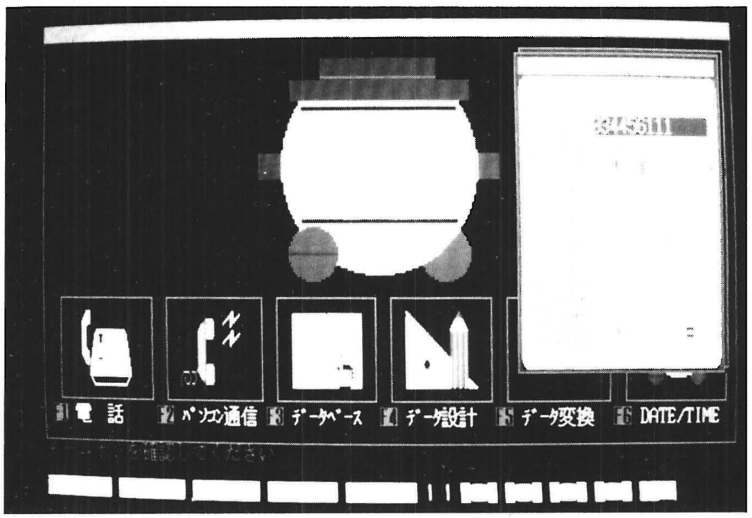
とか，

０３４４５６１１１

と入力し，↵ではなく☎（電話キーで，HELP と兼用になっています）を押します。電話番号には－やスペースなどを適当に含めても構いませんが，テレホンソフトの方では数字だけを抜き出してダイヤルしてくれます。すると画面は**画面４－３**のようになり，呼び出し中であることを知らせてくれます。

相手が出ると，画面に「ターミナル通信したい場合，通信キーを押して下さい」と出て来ます。ここで通信キー(F1)を押すとターミナルモードに入ります。そして後は自由にパソコン通信が楽しめます。そしてまた，F1 キーを押さずに，モデムホンの通信／通話ボタンを押せば，普通の電話として使えます。受話機を取って，話ができるわけです。

相手が話し中とかで，出なかった場合には，F1キーがリダイヤルキーとなり，これを押すだけで自動的に同じ相手にかけることができます。この種のソフトでは全く珍しくないと思いますが，それだけ便利なものだということでしょう（ちなみに私の使っているＮＴＴのプッシュホンにも付いているのですが，一般通話でも大変重宝です）。



《画面4-3》メインメニューからダイレクトにダイヤする

では、①～⑥の機能について，細かく見て行くことにしましょう。

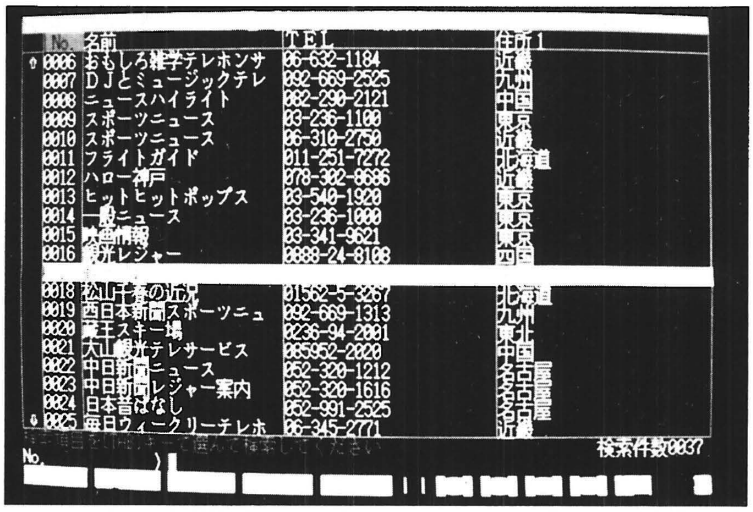
1．電話（電話帳）

メインメニューからF1キーを押すと，「電話」（電話帳）のモードに入り，**画面４－４**のように表示されます。

画面にあるのは元々入力されているサンプルデータですが，団体の電話番号第は本物です。No.0026以降は架空の人物のデータが入っています。

ここで，実行できることは，四つあります。

まず一つは，カーソルキー（↑↓）を使って相手を選び，📞キーで電話をかけることです。**画面４－４**は適当にカーソルを動かしてみな所ですが，📞を押すと**画面４－３**と同じように電話の絵が現われ，同じようにデータ通信等ができます。なお写真からは分からないと思いますが，電話の絵等はすべてスーパーＭＺの1024個のＰＣＧを活用したものですので，スピードは文字表示と同じくらい超高速です。また，F10やESCで中断した時など，電話の絵等の表示が必要なくなった時には，以前その部分に表示されていた内容が自動的に復帰します。**「マルチウィンドウ」**をソフトで実現しているのです。



《画面4-4》「電話」モードに入る

次に，「電話」モードでは，検索を行うことができます。手帳の「住所録」程度の個数のデータしか登録していない時ならカーソルを持って行って📞で良いのですが，自分の知っている会社を全部入力したりして，データが数百件に上るような場合には使い勝手が大変悪くなります。そこで，

・登録番号
・名前
・電話番号
・住所（住所の先頭11文字）

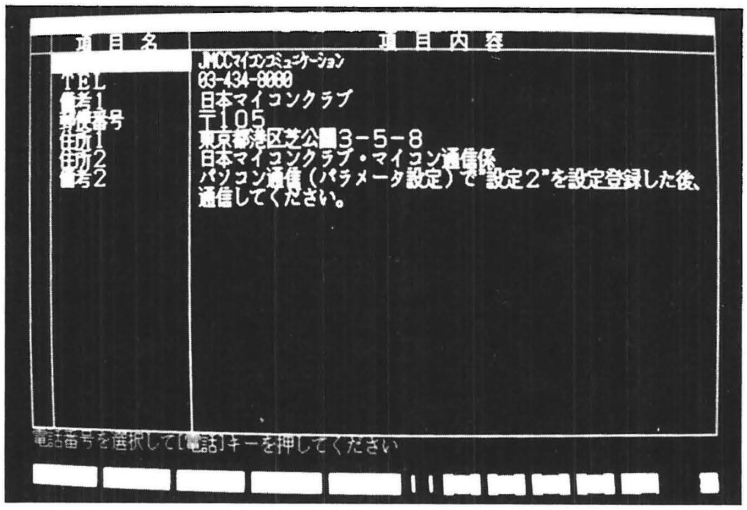
によって，目的となる相手を限定しようというわけです。登録番号の場合には，その番号のデータが中央に来るように言わば「ジャンプ」を行います。100番付近のデータを見たい，という場合にも↓キーを100回押したりしなくとも，100↵で良いのです。その後は↑↓を使い，さきほどと同じように☎でダイヤルできます(これ以降も電話のモードではいつでも可能です)。また，名前・電和番号・住所1で検索する場合には，その全部を覚えていなくとも，あるいはその全部を打ち込むのが面倒でも，先頭の何文字分かを入力すれば、その文字で始まる相手だけが抜き出されて表示されます。名前なら「高橋雄一」とか「高橋で検索するだけでなく，「高」で見つけ出すこともできるのです。この場合は「高井」「高田」「高野」「高橋」「高松」さんたちも一緒に出て来ます。電話番号も「03」，「07435」，「02874」，「02」と言った市外局番やその一部だけのもの「03-445」と言った市外局番入りのものも可能です。最後の1桁だけが思い出せない，という場合にも使えるでしょう。住所も「東京都」「東京都港区」など，自由に捜せます。なお，登録番号・名前・電話番号・住所1のどれで検索するかはTABキーで選択できます。すると，上から2行目の項目名のいずれかが反転表示になり，画面左下にもそれが表示されます。

次に，グループ呼び出しの機能を使うことができます。例えば**画面4－4**で，ニュースならニュースだけを抜き出したい、という場合や、「会社関係の人」「旧友」「親類」を捜したいということもあります。そういった場合にはF1キーを押すと，**画面4－5**のように表示され，ＢＢＳデータベース（ＢＢＳ）だけを表示させる，ということができるのです。なお，**画面4－5**では，グループ名はテレホンソフトが使用している「SYSTEM」を除き，「住所録」「ＢＢＳ」「ニュース」「レジャー」「ミュージック」「案内」「その他」の7種類あり、それぞれ，12件，4件，6件，6件，3件，3件，3件のデータが登録されていることがわかりますが，1件のデータに「ニュース」と「ミュージック」の両方の属性を付けることもできます(3種類以上に所属させることもできる)。また，グループ名は14種類まで持たせることができます。多方面のユーザーが使えるよう，半分をユーザーに開放しているのです。なお，グループ呼び出しを行った後，さらに名前や住所等で検索を行うこともできます。「データベース（ＢＢＳ）の中で，大阪にあるものを捜したい」といったことも可能です。



《画面4-5》「電話」中の「グループ」をテスト

「電話」モードの残る一つの機能は，「照会」を行うことです。F2キーを押すと，**画面4－6**のように表示され、現在選択している相手，つまり，キーを押すだけで電話がかけられる状態の相手の詳細なデータ――名前・電話番号・備考1・郵便番号・住所1・住所2・備考3――がわかります。うろ覚えの相手を確めたり，データベースの場合の注意事項を見たりもできます。



《画面4-6》「電話」の中の「照会」をテスト

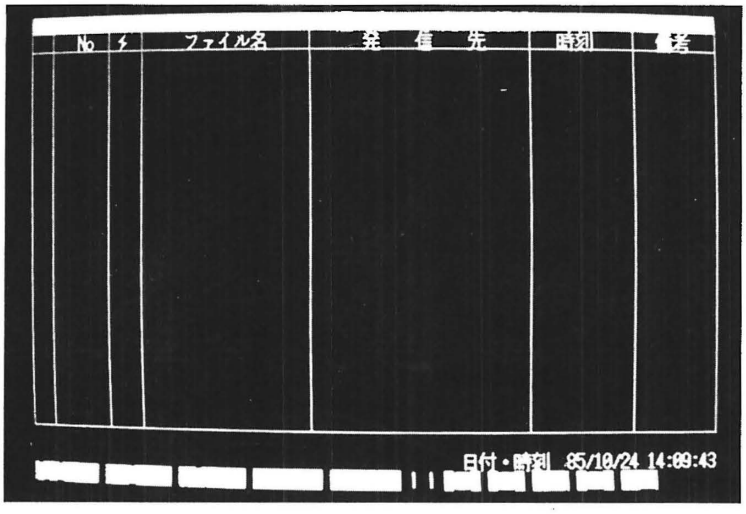
なお，**画面4－5**や**画面4－6**のの状態から何もせずに抜け出したい，とか，電話のモードからメインメニューに戻りたい，という場合には，F10キーかESCキーでＯＫです。テレホンソフトは常にF1～F6で何かを選択・実行，F10かESCで中断・1段前に戻る，↑↓で上下選択，TABで横選択，という操作方法になっています。また，SHIFT+BREAKとすると一番初めのメインメニューに戻れます。これさえ把握していれば，例えマニュアルなしでも使えるよう，工夫されています。

2.パソコン通信

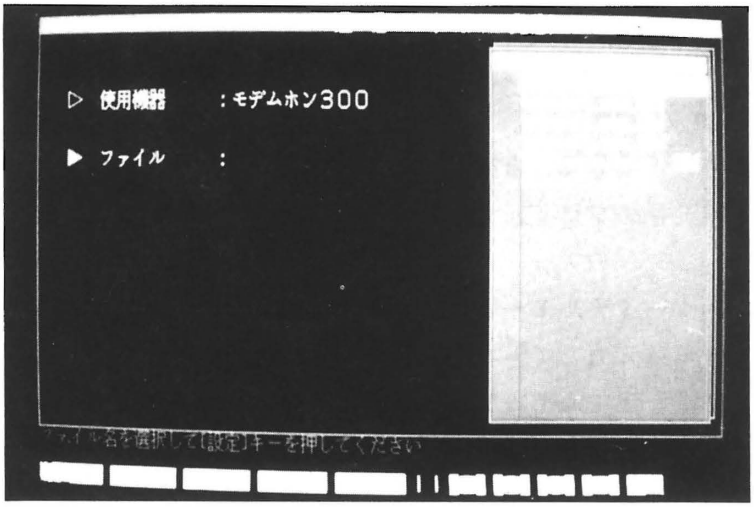
メインメニューからF2キーを押すと，「**パソコン通信**」のモードに入ります。**画面4－7**のようになります。

このモードでは五つのことが実行できます。

まず，発信です。F1（発信）キーを押すと，**画面4－8**のように表示され，ここで送信したいファイルを，右の一覧表の中から↓↑キーで選びます。通常はドライブに入っているファイルが出て来ますが，F2（ＦＤでドライブ2等にすることもできます。また，スーパーＭＺモデル10（ディスク1基）の場合には，ドライブ1，つまりテレホンソフトの入ったディスクと入れ換え，F3（ＤＩＲ）を押しても構いません。モデル20でも，F2で設定したドライブに目的のファイルが見つからなかったら，ディスクを入れ換え，F3を押して，一覧表を再表示させることもできます。そして，送信したいファイルを選択したら，F1（設定）を押します。すると，次に相手の電話番号



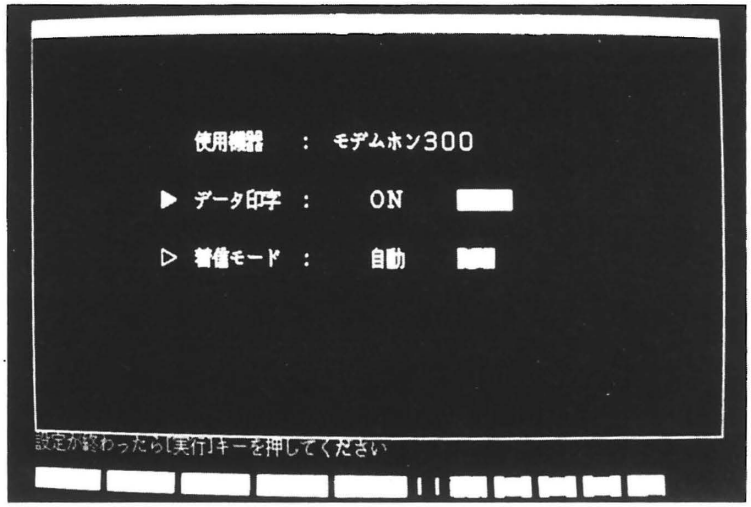
《画面4-7》「パソコン通信」モードに入る



《画面4-8》「パソコン通信」の発信を選択する

を聞いて来ます。テンキーから直接入力してF1（設定）を押しても良いですし，F2（検索）を押して，メインメニューからF1を押した時のように「電話」(電話帳)を呼び出すこともできます。その場合は☎キーで，この電話番号の所に戻って来ます（電話番号は自動的に入力されます)。また，電話番号を入力しようとして，ファイル名のミスに気付いた時は，F3(復帰)を押して，ファイル名の入力の所に戻ることもできます。もちろんF10を押して**画面４－７**の状態に戻ることもできます。電話番号の入力が済んだら，次は予約発信をするかどうか聞いて来ます。Yと入力すると発信したい日時（分の位まで）を聞いて来ます。それを答えると**画面４－７**の状態に戻り，送信ファイル名・相手・時刻等が表中に登録されます。あとはその時刻が来たら，スーパーＭＺが勝手に送信を行ってくれます。ただ、その時もし，相手が話し中とかで送信がうまくできなかった場合には，時刻が５分後に再設定され，５分たったら再び自動的に送信を試みてくれます。これは３回まで（合計４回）行われますので，大変有難いところでしょう。また，予約中・送信成功・送信失敗といったことは表中の表示に残ります。外出中・就寝中の予約送信にも安心でしょう。なお，予約発信の所でNと答えた場合には即時発信となります。F5キーを押すと，すぐに送信が行われます。音響カプラ使用時は、もっぱらこれを利用することになります。

次に，パソコン通信のモードでは，着信が行えます。F2（着信）キーを押すと，**画面４－９**のように表示されます。ここで入力することは二つあり，プリンタ印字をするしかないか，即時着信とするか自動着信とするかです。音響カプラでは，即時のみになります。プリンタ印字が行われるのは，着信時刻・あれば、ＩＤ情報（ファイル名）・アスキーファイル（文字のファイルならその内容・エラーがあった時はその内容についてです。なお，着信したデータは，階層ディレクトリ「／recfile／」中に，ファイル名がＩＤに含まれていた時はそのファイル名（16字まで）で，入ってなかった時は「rec××」(××は００～９９の数字で，空いているものが順に分り当てられます）という

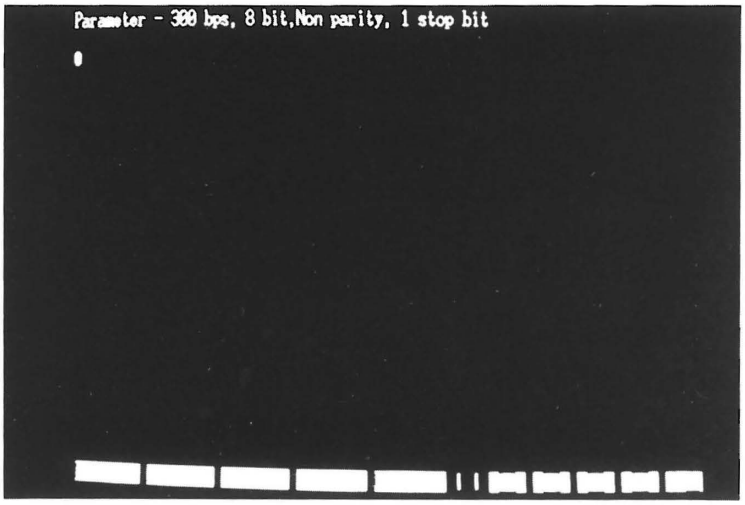


《画面4-9》「パソコン」の「着信」を選択する

ファイル名で，ディスクに登録されます。なお，スーパーＭＺはMS-DOSのようにファイル作成日時が自動的に登録されますので「rec００」のようなファイルが多数存在することになってもそれで判別できると思います。

次に，[F3]（解除）キーを押すことで，**予約発信と自動着信を解除**することができます。さらに[F1]を押すと，番号を聞いて来て，未発信のファイルを１個単位で解除できます。[F2]なら，すぐ、自動着信を止めます。

次に，[F4]（ターミナル）を押すと，**ターミナルモード**に入ります（**画面４－10**)。これまでの説明で，メインメニューから電話番号を直接入力して[☎]キーを押した時と，「電話」のモードで相手を探し[☎]キーを押した時には，電話の絵が出て来てパソコンがダイヤルを行ってくれ，相手が出たらターミナルモードに入ることができるとお話しして来ましたが，今説明しているのは，パソコンから制御可能なモデムホンを使った場合ではなく，音響カプラや直接ケーブル（変換器を介して光ファイバー等を使用しても良い）を使って他のコンピュータと接続し，スーパーＭＺをそれらのターミナルとして使う場合のことです。ただ，今まで話して来たものとはターミナルモードに入るまでの手順が少し異なるだけで，入った後の機能は変わりません。そのため，ターミナルモードの詳しい所までは説明せずに来ましたが，ここでまとめて見たいと思います。まず，ターミナルモードでは，普通にキーボードを打って，それを相手に送ることができます。また，[SHIFT]+[変換]で日本語入力モードに入って，漢字等全角文字を入力・送信することもできます。この時，オプションの辞書ROM(MZ-1R28) を付けていれば，９万語の文節変換・人名地名変換を利用することもできます（「電話」で名前を検索・入力する場合等にも可能)。また，[F1]キーを押すと，キーボードから入力・相手から受信したコントロールコード（キャラクターコード20Ｈ未満のコード）を，普通に実行するか，BSや⇨のようなシンボルキャラクタで表示するかを変更できます。同様に[F2]キーでは全２重モード・半２重モードを選択できます。全２重では相手から来たデータはそのまま画面に表示されますが，こちらの押したキーは１文字単位でRS-232Cから送信するだけで，そのまま画面に表示することはありません。しかし相手が自動的にそれを送り返してくれるので，結果的には自分の打った文字も自分の画面に出て来ます。このため，データが確実に行われているのが見てわかります。一方半２重では，こちらが入力した文字は一度画面に表示されますが，BASICのプログラム入力と同じように[↵]キーが押されるまで他のことはしません。そして[↵]が押されて初めてRS-232Cからそのデータを送信し，今後は受信モードに入ります。そして１行分のデータを受信したならそれを画面に表示し，再び送信モードに入ります。全２重では送信と受信を自動的にその都度行うのに対し，半２重では改行コードによってバドミントン的に送受信を切り換えるということです。なお，モデムホンを1200ボーで利用時は半２重専用となります。次に，[F3]キーを押すと，交信（送信・受信）の内容をプリンタで記録しておくことができます。データベースをアクセスした時には，せっかくですからこれを使って紙の形にしてとっておくと良いでしょう。これは次に[F3]キーを押すまで続きます。次に，[F4]キーを押すと，交信の内容をディスク上のファイルとして保存しておくことができます。「ダウンロードをするファイル名を入力して下さい」と表示され，それを入力すると，次に再び[F4]キーを押すまでの間，BSD形式のファイル（いわゆるアスキーファイル）として，書込まれます。次に，[F5]キーを押すと，今度は逆にディスク上のＢＳＤファイルを相手に送信し，同時にこちら側の



《画面4-10》「パソコン通信」の「ターミナルモード」



《画面4-11》「パソコン通信」の「パラメータ」変更中

ディスプレイにもそれを表示することができます。「アップロードするファイル名を入力して下さい」と表示されますので，それに答えれば、再びF5キーを押すかファイルの内容をすべて出力し終わるまで，送信が行われます。次に，F10キーを押すと，ターミナルモードから抜けられます。他のモードではESCでもF10と同じ動きでしたが，ターミナルモードだけはエスケープコードを送れるように，というわけで，F10だけとなっています。

五つあるパソコン通信のモードのうち，最後の一つは**パラメータの設定**です。F5（パラメータ）キーを押すと，**画面4－11**のように表示され，**第4－1表**に示した14種類の設定ができます。特に最後の「設定」などは，利用者の使い勝手を配慮した，心憎い設計だとは思いませんか？

3.データベース

メインメニューからF3キーを押すと，**「データベース」**のモードに入ります。なお，ここで言うデータベースとは，電話を使ったサービス，つまりBBSのことではなくて，パソコンデータベースとか，リレーショナルデータベースに近い，パソコン単独で行うデータ管理システムのことです。

このモードは，1の電話（電話帳）のモードと大変似ており，画面その他はそっくりです。ただ，

- 電話モードでは☎キーで電話をかけることができるが，データベースのモードではこれはできない
- 電話モードではデータを見るだけだが，データベースのモードでは，データを書替えたり，加えたり，削除したりできる

《第4-1表》通信パラメーター一覧表

使用機器	モデムホンを300ボーで使用，1200ボーで使用，モデムホン以外を使用するかを指定します
ボーレイト	75～19200ボーが可能ですが，モデムホン使用時は300ボーか1200ボー，音響カプラ使用時は300ボーのみです。他は，直接コンピュータ同士をつないだ時のみです
パリティ	なし・奇数・偶数が可。Z80BSIOを使用しているのでデータ8ビット時にもパリティ設定可
データビット長	7・8ビットが可
ストップビット	1・1.5・2ビットが可
通信制御	バッファがいっぱいになりそうな時の制御で，XON／XOFF，RS／CS，使用しない，が可
Sパラメータ	データ7ビット送信時，SI(0FH)／SO(0EH)でカナを送信するかしないか
DELコード受信	DELコード(7FH)受信時，何もしないか・BS(08H)扱いにするかDEL(7FH)扱いにするか
CRコード送信	↵キーをCR(0DH。カーソルを左はしに移動させる)として送信させるか，CR+LF(0AH。カーソルを1行下に下げる)として送信するか
CRコード受信	CRコードを受け取った時，単にCRとして処理するか，CR+LFとみなすか
日本語処理	シフトJIS(X 1 turbo，MS-DOS，CP／M-86コンパチ)，K1=1BH+4BH・KO=1BH+48H(PC-9801のN88BASICコンパチ)，K1=1BH+09H・KO=1BH+01H(MZ-5500／6500BASICコンパチ)，K1=1BH+24H+42H・KO=H・KO=1BH+28H+4AH(日本マイコンクラブコンパチ)，K1=1BH+24H・40H・KO=1BH+28H+48Hの5種類が選択可能
通信方式	全2重・半2重が選択可
ファイルエンドコード	アップロード・ダウンロード時の自動停止コードで，00（なし）・01H～1FHが可能設定可
設定	使用機器～ファイルエンドコードについて，良く利用する5通りの設定が保存しておけます。また，テレホンソフトを電源ONで起動させた時のデフォルトをその5通りの中から選べます

の2点が異なります。

電話のモードでは[F2]キーは「照会」になっていますが，このモードでは「呼出」になっています。しかし表示される内容は**画面4-6**とほぼ同じで，「項目内容」の所にカーソルが出て来るだけです。この状態では[SHIFT]+[変換]を利用して日本語で入力することも含め，名前や電話番号等を自由に変更できます。また，[F1](変更登録)を押すと，その時までに入力したものを，データの変更とみなし，元のデータが修正されます。また，その際，グループの所属を変更することもできます。一方，[F2]（追加登録）を押すと，その時までに入力したものを，新しい「相手」データとみなし，さきほどこの画面を呼び出した時の「相手」のすぐ下に追加登録されます。例えば，新しいBBSが始まった時には，[↓]でNo.4の所に移動してこのモードに入り，[F2]で抜ければ，No.5の所に追加できます（旧0004と旧0005の間に入る)。また，この時にもグループを設定できます。それから，[F3]（データ消去）を押すと，名前その他が消去されます。追加登録を行う時は，この画面に入った直後にこれを押し，新たに入力しても良いでしょう（特に住所等が全く異なるような場合)。

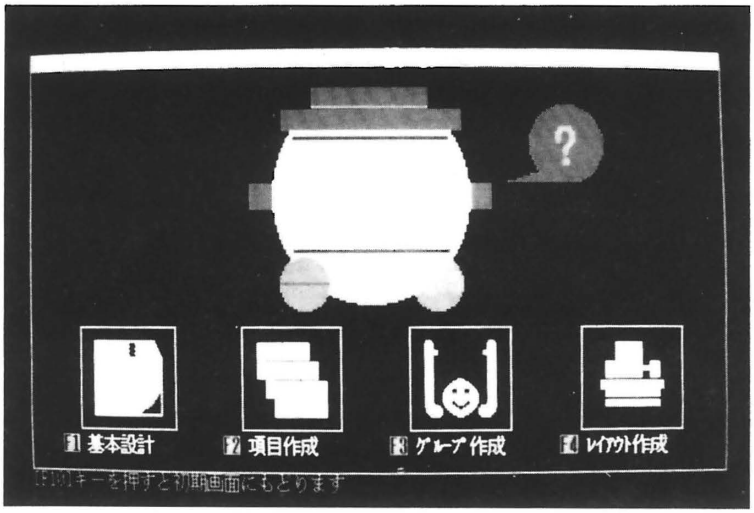
データベースのモードでは，電話モードにない，[F3]・[F4]・[F8]キーも意味を持ちます。

[F3]（一括登録）は，データの最初から最後まで全部の「相手」に，一斉にグループの属性を付けることができます。新しいグループを作って，現在入力されているすべて，または一つか二つを除いた残り全部の「相手」が適合する場合に利用できるでしょう。一つや二つの例外は，まずこれで一斉に設定を行った後，それらを個別に選んで（[F2]で呼び出しを行う）元に戻すと良いでしょう。

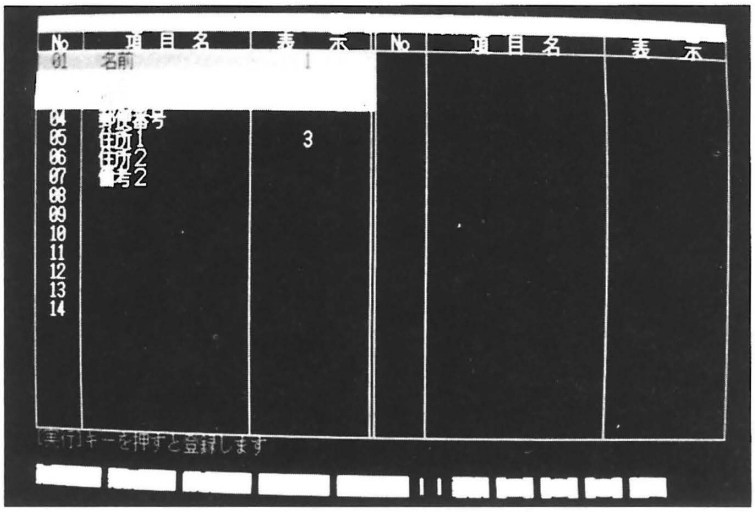
F 4 （削除）は，現在反転表示されている「相手」を削除するのに使います。

F 8 （印刷）は，データベースで作成したデータをプリンタで印字したい時に使えます。印字の方法は２種類あり，一覧印刷と内容印刷とが選べます。一覧印刷というのは、画面に出ているデータ，つまり登録番号・名前・電話番号・住所１を全員分，一覧表形式で印刷するものです。内容印刷というのは，主にＤＭ（ダイレクトメール）を打ったりするようなために，様式を指定して，印字を行うものです。ＤＭ用の用紙に合ったフォーマット３種（横１列・２列・３列）と一覧印刷よりさらに詳しく，入力されているすべてのデータを印刷するモード，案内状として入力した文字だけを打ち出す列（ワープロ的）の計５種が登録されていますが　次項のデータ設計を使うと，40種類まで様式を設定することができます。また，F 2 （呼出）の状態からF 8 を押すと，１人分（１社分）だけの印刷を行えます。

4.データ設計



《画面4-12》「データ設計」モードに入る



《画面4-13》「データ設計」の中の「項目作成」を選択する

テレホンソフトは１～３の使い方だけでも，充分すぎるくらいの実用性を持っていると思いますが，この項と次項を利用すると，非常に高度な活用も可能です（**画面４－12**）。

例えば，今までの説明では，「電話」の所で出て来るのは登録番号と名前，電話番号，住所１と決まっているような言い方をしてきましたが，これを変えることもできます（**画面４－13**）。

また，**画面４－13**では下の方にすき間がたくさんありますが，１件分のデータは最大1000字までの「項目内容」を持つことができます。（通常は200字程度で使用しています）。個人の場合には生年月日や「手持ち機種」「勤務先（学校名）」の項目を追加したり，会社の場合には設立年月・資本金・売上げ・代表者（社長）名等を追加することもできます。

さらに，住所や電話番号などはすべて取ってしまって，全く新た

なデータベースとして利用することもできます。「書籍管理」などと言ったこともできるわけです。

また，既に触れましたが，グループ名を追加・削除したり（**画面4－14**），プリンタのレイアウトを追加・変更することもできます（**画面4－15**）。

それと，データの入ったドライブを変更したり，データディスクを作成したり，システムのコピー（テレホンソフトのコピー）を行う，と言った基本的なこともできます（**画面4－16**）。

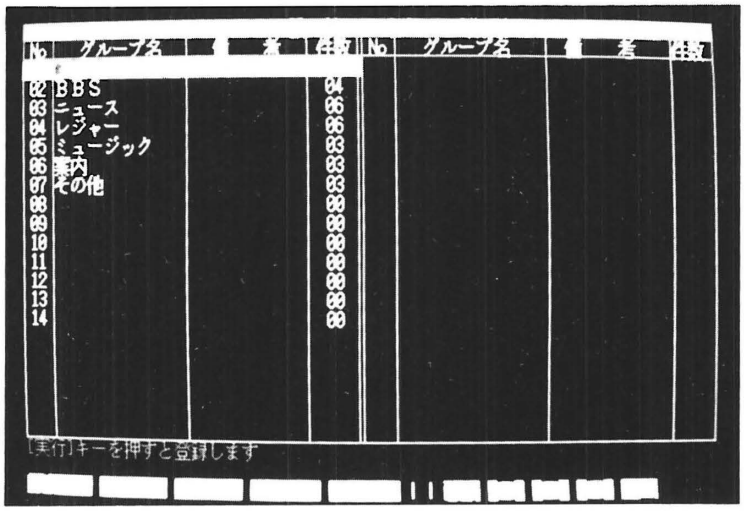
5.データ変換

テレホンソフトがこれだけ機能を持つようになると，これで作成したデータをBASICで使いたいとか，逆にBASICで作成したデータをテレホンソフトで利用したい場合もあります。

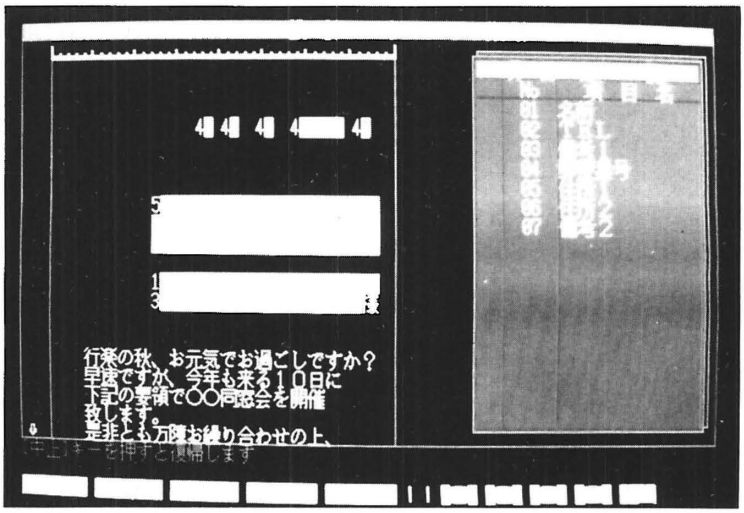
そう言った場合にこの機能を使えば，テレホンソフトのデータの構造をわざわざ解析してプログラムを作らなくとも，テレホンソフトの方で，こちらの希望通りの順序に合わせ、こちらの希望するものだけを転送してくれます（**画面4－17，4－18**）。プログラム自作派にとってもうれしいところでしょう。

6.DATE／TIME

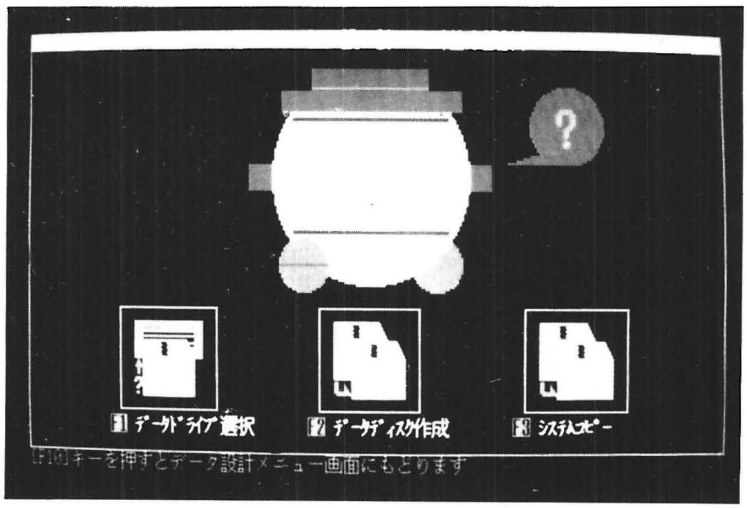
現時刻・日付けの修正が行えます（**画面4－19**）。



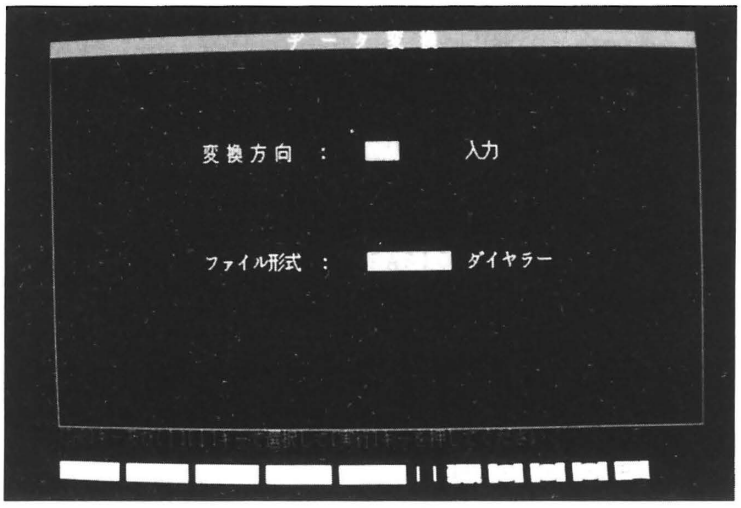
《画面4-14》「データ設計」中の「グループ作成」を選択する



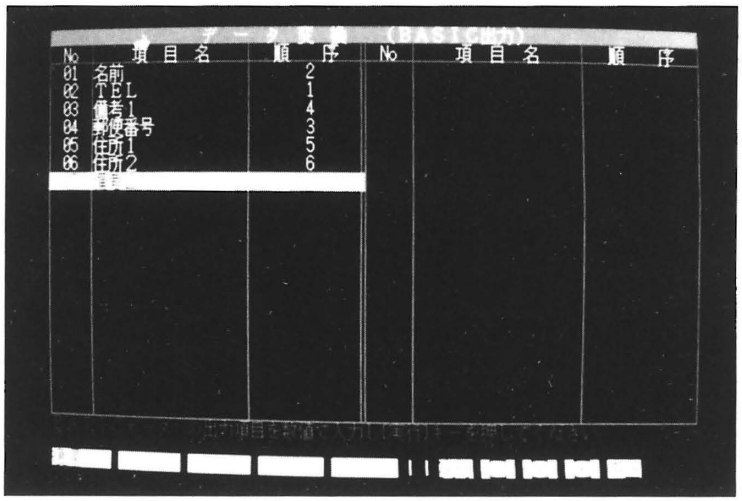
《画面4-15》「データ設計」中の「様式呼出」で詳しい様式を変更可



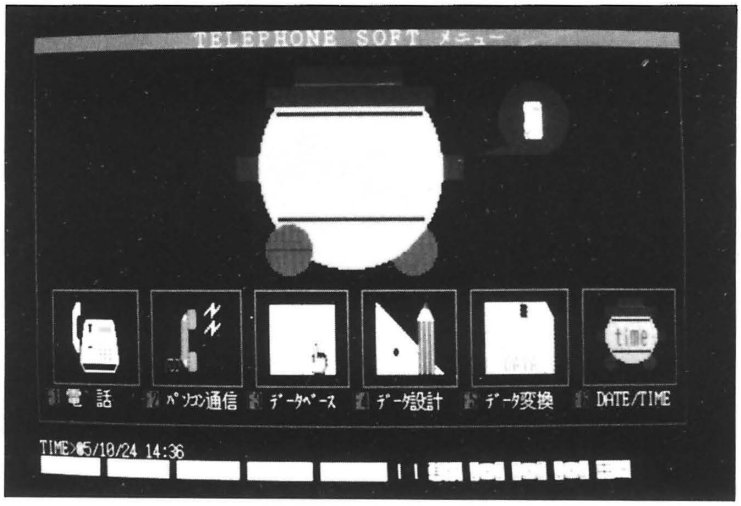
《画面4-16》「データ設計」中の「基本設計」を選択する



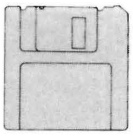
《画面4-17》「データ変換」モードに入る



《画面4-18》「データ変換」で並べる順序を指定



《画面4-19》「DATA／TIME」で時刻，日付けを設定する

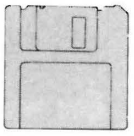


＋α

さて，以上がテレホンソフトの概要ですが，注目すべきことが一つあります。それは，

BASIからオートダイヤル機能が使えるということです。

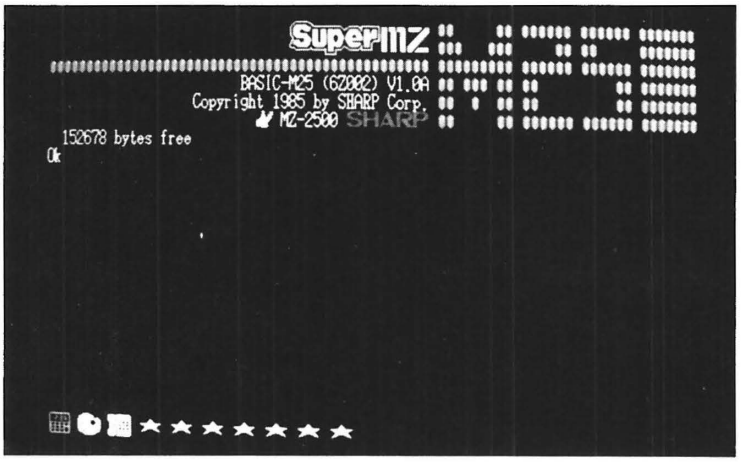
画面４－20はBASICのコマンド待ちの状態からアルゴキーを押した所ですが，ここで電話を選択すると，すぐさま**画面４－21**のような表示となり，テレホンソフトの「電話」モードのように，通信を行うことができるのです。この辺は開発当初から念頭においてＢＡＳＩＣを作成している強みと言えるでしょう。



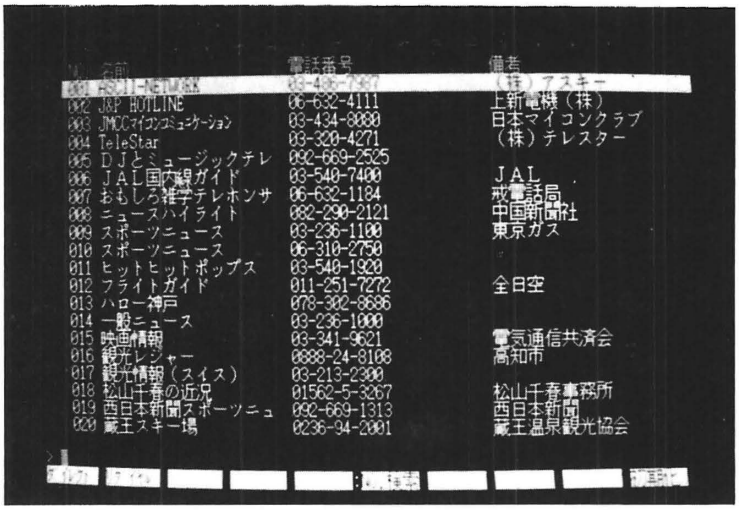
まとめ

テレホンソフトは，単に通信機能を行うためのソフトウエアというより，パソコン通信に対応した，データベースソフトと言った方が良いかも知れません。

スーパーＭＺの場合，これが標準装備となっていることから，まず大多数の人はテレホンソフトを利用することになると思いますが，それだけに，これだけ強力なサポートには大変心強いものを感じます。また，オプションのボイスコミニュケーションインターフェースボードを購入すれば，内蔵カセットデッキによる留守番電話やボイスメール機能も利用できます。



《画面4-20》BASICからアルゴキーを押したところ



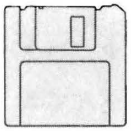
《画面4-21》さらに「電話」を選択したところ





第5章 BASIC プログラミング入門

せっかくいいコンピュータを買ったんだから，自分でプログラミングできるようにならないともったいない！そこで第5章では，BASIC—M25によるプログラミング法をお教えいたします。



プログラミング0「プログラムの基本」

★プログラミングの覚え方

さて，プログラミングを覚えるのにはどうしたらいいのでしょう？手段は色々と考えられます。

①パソコンスクールに通う

②TVのパソコン番組を見る

③BASIC入門を買う

以上，三つぐらいはすぐに思いつくでしょう。でもこれ等には

①はお金がかかる。

②はあれよあれよと言っている間にブラウン管のむこうの賢い少年達と司会者のみが納得して終ってしまう。

③はBASICのリファレンスマニュアル程度のことしか書いていないので無意味。

と言った具合に大きな欠点があります。

では，一体どうするのがベストなのか？それは「簡単なゲームを自作する」これしかないのです。

★ゲームの自作

「さあ，プログラミングを覚えよう」と意気込んでBASICのリファレンスマニュアルを開くと，

まず命令の多さに驚くでしょう。昔は40個ばかりの命令しかなかったBASICが，今やコマンドと一般命令合わせて100個以上の命令があるなんてのがザラだからです。このスーパーMZのBASIC－M25の命令も200個以上あり，マニュアルの最初から最後まで読み通すには人並み外れた気力がいることでしょう。大底の人は意気消沈してしまい，そのうちあきらめてしまいます。しかし「BASICでゲームを作る」となると話が違ってきます。何百何十ある命令のほんの数十個さえ理解していれば完成させることができるのです。おまけに作りながら遊べるので飽きるなんてことも少なくて済みます。コンピュータになじむにはもってこいの方法なのです。

★基本命令

たとえば以下の様なゲームを作るとします。

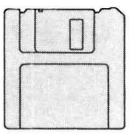
「左右に移動するビーム砲を操り，　ランダムの高さで出現するUFOを撃つ。UFOを一つ撃つとプラス10点。左右どちらかに逃がしてしまうとミス1回。10回ミスするとゲームオーバー」

このようなゲームを作るとなると，おおよそ以下の様な命令が基本となります。

IF～THEN～ELSE
LOCATE
PRINT
FOR～NEXT
GOTO

また，これ等の命令はゲームばかりではなく全ての**BASICプログラムの基本**となります。ここでは特に説明はしないので**BASIC**のマニュアルとよくにらめっこをして頭の内にたたき込んでおいて下さい。

さて，それではいよいよゲームを作りましょう。モノは先に書いた**「UFO撃墜ゲーム」**です。



プログラミング1「画面構成」

ゲームにしても他のアプリケーションにしても，プログラミングするにあたって真っ先に考えなくてはいけない事が**「画面構成」**でしょう。「どこに，何を表示させるか」が決まらなくては漠然としていて手のつけようがないのです。

ここではスーパーMZの文字画面の種類と初期設定の方法を簡単に説明しましょう。

★INIT "CRT 1 :"命令

スーパーMZの文字画面には多くのモードがあり，およそ以下の四つの組み合わせで決まります（注：ハードウエアの制限上，実行できない組み合わせもあります）。

- **桁数（ヨコの表示文字数）　40，80**
- **行数（タテの表示文字数）　12，20，25**
- **フォント（1文字を何ドットで表示するか）　8×8，8×16**
- **スムーススクロール　　する，しない**

そして，これ等を設定する為の命令が　INIT“CRT１：”なのです。書式は次の様になります。

INIT“CRT1：桁数,行数,フォント，スムーススクロール”

桁数，行数にはそのまま数字を入れ，フォントの部分には８×８ドットで使いたい時は０を，８×16ドットで使いたい時は１を入れます。またスムーススクロールの部分には，スムーススクロールさせたい時は１を，通常のスクロールをさせたい時は０を入れます。

★KMODE命令

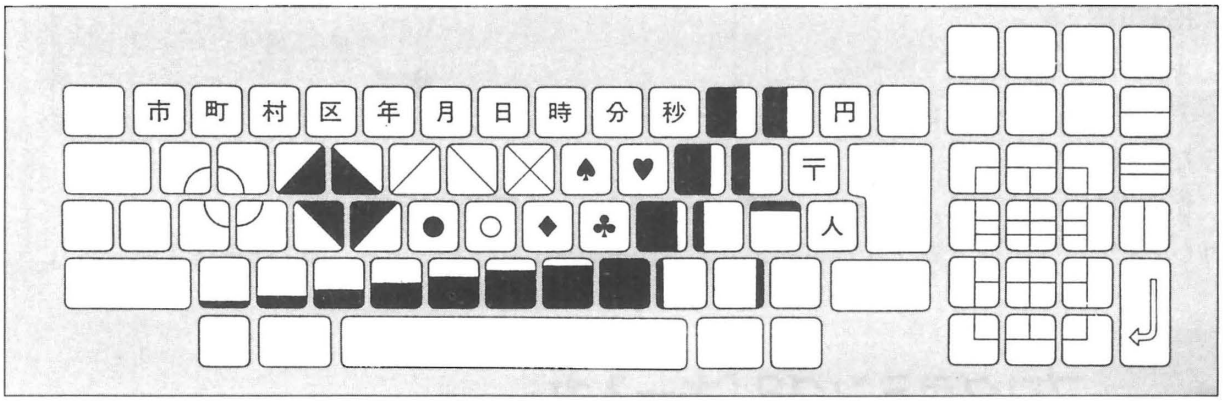
スーパーＭＺはＰＣ－8801のように♥や●などのグラフィックキャラクタを持っていて，ＧＲＡＰＨキーを押してから各キャラクタに対応しているキーを押すことによって表示させることができます（第5—1図）。

ゲームなどのキャラクタを作る時，このグラフィックキャラクタを使うと簡単にできるのですが，表示モードが漢字モードになっているとグラフィックを表示させることができなくなります。このモードを切換えるのがKMODE命令で以下の様に使います。

グラフィックキャラクタを出力したい時　KMODE　0

漢字を出力したい時　　KMODE　1

BASIC起動時では，KMODE　１になっているのでグラフィックキャラクタを使いたい時は，KMODE　０を実行してからキーを押します。



《第５—１図》グラフィックモードキーの配列

★KEY LIST(KLIST)命令

スーパーＭＺでは最下行にファンクションキーの定義状態や，時計を表示させることができます（注：スムーススクロールを設定した場合は表示できません)。それを設定するための命令が，KEY　LIST命令(KLISTでも同)で，以下の様に使います。

KEY　LIST　0　ファンクションキーや時計の表示をやめます

KEY　LIST　1　ファンクションキーを表示します

KEY　LIST　2　ファンクションキーと時計を表示します

ゲームなどで遊ぶ時，ファンクションキーの表示は邪魔なので，KEY　LIST　０で消しておきます。

★CLS命令

画面表示の消去を行う命令で，以下の様に使います。

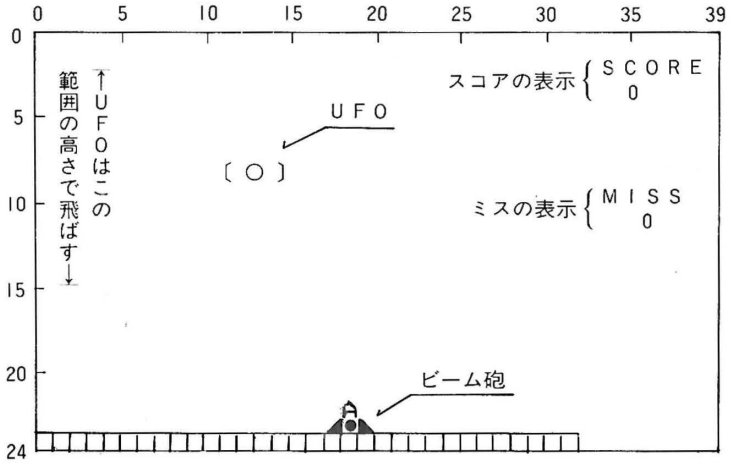
CLSまたはCLS　1　　文字画面のみを消去
CLS　2　　グラフィック画面のみを消去
CLS　3　　文字グラフィック画面を消去

★ゲームの画面構成

UFO撃墜ゲームでは以上のような命令を使って次のような画面設定を行うことにしました。

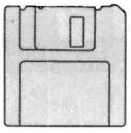
・ファンクションキーの表示はしない　KLIST　0
・文字画面は桁数40，行数25，フォントは8×8ドット，スムーススクロールなし
INIT "CRT 1 : 40, 25, 0, 0"
・グラフィックキャラクタ使用
KMODE　0

また，この画面設定でゲームをデザインし，**第5－2図**のような画面のゲームにしました。



《第5－2図》ゲーム画面デザイン

文字画面の設定方法，お分かりいただけたでしょうか？次はキー入力の方法を説明し，実際にビーム砲を動かしてみましょう。



プログラミング2「キー入力」

キー入力には大きく分けてリアルタイムの入力方法（**INKEY$命令**）とカーソルを点滅させて入力を待つ方法(**INPUT命令**)とがあります。データベースやワープロなどのアプリケーションソフトのキー入力は後者が主となり，アクションゲームなどでは前者が主となっています。

今回はとりあえず前者,「**リアルタイムのキー入力**」を説明しましょう。

★INKEY$関数

リアルタイムのキー入力には普通INKEY$ 関数を使います。例えば

```
A$=INKEY$
```

を実行すると，その時押していたキーの文字データがA$ に入ります。

```
10  A$ =INKEY$
20  PRINT  A$
```

```
30  GOTO  10
```

以上の様にプログラミングして色々なキーを押して確かめてみて下さい。

★REPEAT命令

先のプログラムを実行してみて何か気づかれたでしょうか？ よく調べてみるとキーを押し続けているにもかかわらず，所々にNull（空文字）が入っているのです。キーのリピート速度が遅いため，押しているキーの文字データを常に読めないのです。このままでは完全なリアルタイム入力とは言えません。

そこでゲームなどを作るには，**REPEAT命令**でキーリピートの速度を上げてやらなければいけません。以下の様に使います。

REPEAT ON, 0	**リピートさせない**
REPEAT ON, 1	**低速リピート**
REPEAT ON, 2	**中速，通常はこのモード**
REPEAT ON, 3	**高速リピート**
REPEAT ON, 4	**最高速リピート**
REPEAT OFF	**リピートさせない**

つまりREPEAT ON, 4を実行すれば，スルドイ入力ができるわけです。先のプログラムの5行にでも入れて確かめてみて下さい。

注：リピート速度を上げてから次の作業に移る時は[CTRL]+[D]キーでリピート速度を通常に戻します。

★CLICK命令

スーパーMZでは，キーを押す度に「ポッ」というクリック音が出ます。このクリック音を出す，出さないをコントロールするのが**CLICK命令**で，以下の様に使います。

CLICK ON　クリック音を出す
CLICK OFF　クリック音を出さない

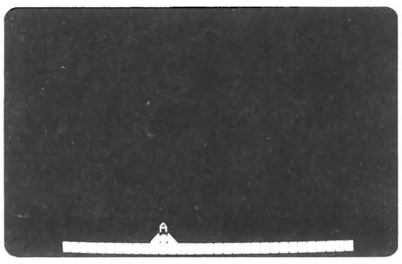
これによりゲーム最中に何となく耳障りなクリック音を消すことができます。

★ビーム砲移動

ビーム砲移動は，以上の命令を使って次の様なキー入力設定を行うことにしました。

・クリック音は出さない　　CLICK OFF
・リピートは最高速　　REPEAT ON, 4
・4と6のキーで左右に動かし，スペースで弾を発射

そして今まで説明した命令を組み合わせてプログラミングしたのが**リスト5—1「ビーム砲移動」**です。変数のCXにはビーム砲のX座標を，GE$には押されたキーの文字コードを入れています。また150～160行のFOR～NEXT命令は，単なる時間がせぎです。これが無いと速く動きすぎてしまうので



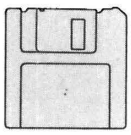
《写真5－1》ビーム砲が移動

す。

リアルタイムのキー入力方法，おわかりいただけたでしょうか？次はゲームの要，「乱数」について説明し，UFOを飛ばしてみましょう。

《リスト5－1》ビーム砲移動プログラム

```
 10 klist 0
 20 init "crt1:40,25,0,0"
 30 kmode 0
 40 click off
 50 repeat on ,4
 60 cls
 70 locate 0,24
 80 print "■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■";
 90 CX=15
100 GE$=inkey$
110 if GE$="4" then CX=CX-1:if CX<1 then CX=1
120 if GE$="6" then CX=CX+1:if CX>28 then CX=28
130 locate CX,22:print " A "
140 locate CX-1,23:print " ◢■◣ ";
150 for D=0 to 200
160 next D
170 goto 100
```



プログラミング3「乱数発生」

乱数――ランダムはゲームの要です。どんなゲームにも必ずといっていいほど乱数は使われています。ここではUFOをランダムの高さで出現させて，左右に飛ばしてみましょう。

★RND関数

乱数発生には**RND関数**を使います。これは0から0.99999……の数をランダムに発生させるもので，だいたい以下の様に使います。

RND(-1)　乱数を初期化する

RND(0)　直前に発生させた乱数と同じ値になる

RND(1)　次々に新しい乱数を発生させる

RND(-1)の「乱数の初期化」にはRANDOMIZEという命令が関わってくるのですが，普通に乱数を作るのに支障はないので今回はふれていません。

★INT関数

INT関数は，以下の様に使って数式の値を越えない最大の整数を作ることができます。

INT(数式)

つまりINT(3.5)は3になり，INT(−4.2)は-5になります。

これを乱数発生にどう使うのかというと，例えば0から14までの整数の乱数を得たい時など，以下のようになります。

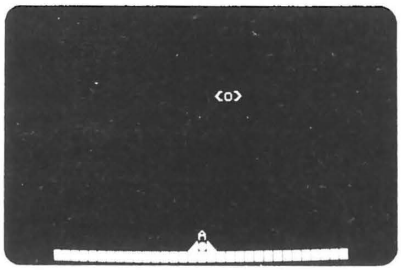
```
10  A＝RND(1)＊15
20  B＝INT(A)
30  PRINT B
```

つまりAには0から14.999……までの数が入り，それをINT関数に入れることにより，0から4までの整数を作るのです。

★UFO出現

リスト5－2は，**リスト5－1**に95行と170行～220行を加えたものです。180行と190行でUFOが左右どちらかの端まで来た時に，乱数で高さを変えています。

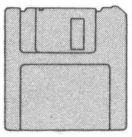
次はいよいよゲームの完成。ゲームに必要な他の命令を説明し，まとめにしましょう。



《写真5－2》UFOが乱数で動く

《リスト5－2》UFO出現プログラム

```
 10 klist 0
 20 init "crt1:40,25,0,0"
 30 kmode 0
 40 click off
 50 repeat on ,4
 60 cls
 70 locate 0,24
 80 print "■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■";
 90 CX=15
 95 UX=0:UY=12:URUN=1
100 GE$=inkey$
110 if GE$="4" then CX=CX-1:if CX<1 then CX=1
120 if GE$="6" then CX=CX+1:if CX>28 then CX=28
130 locate CX,22:print " A "
140 locate CX-1,23:print " ◢0◣ ";
150 for D=0 to 200
160 next D
170 XX=UX:YY=UY:UX=UX+URUN
180 if UX=0 then URUN=1:UY=2+int(rnd(1)*15)
190 if UX=29 then URUN=-1:UY=2+int(rnd(1)*15)
200 locate XX,YY:print "   "
210 locate UX,UY:print "<O>"
220 goto 100
```



プログラミング4「ゲーム完成」

弾を飛ばし，得点やミスをカウントさせれば，いよいよこのゲームも完成。ついでにゲーム作りに必要な他の命令も挙げてみました。

★SCRN$関数

このゲームでは使用しませんが，例えば迷路の中を上下左右に動くゲームでは，どこが壁なのか道なのかを画面を読んで判断させる必要があります。**SCRN$**とは画面上の座標を指定することによって，その座標にある文字を取り入れてくれる関数なのです。書式は次の様になっています。

SCRN$(X座標，Y座標，文字数)

文字数で「その座標から横に何文字取り入れるか？」を指定でき，省略すると1になります。

★DCGコントロール命令

スーパーMZにはDCGが内蔵されていて，これでキャラクタを再定義することによって更にゲームっぽいキャラクタを作ることができます。これをコントロールするための命令が

CGEN
DEF CHR$
CGPAT$

ですが，ここでは特にふれていません。第9章の「PCGデータメイカ」を参考にして下さい。

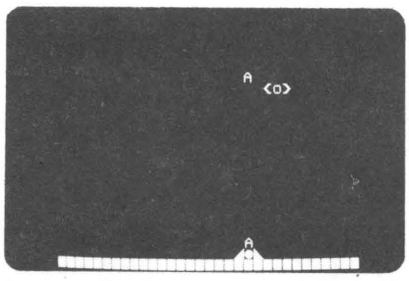
★OPNコントロール命令

スーパーMZには音楽用のLSI，**「OPN」**が内蔵されていて色々なサウンドが楽しめます。普通の音楽演奏に用いるPLAY命令，BGMを流すのに便利なON PLAY GOSUB命令など豊富なコントロール命令がありますが，今回はPLAY文だけを使って「ビッ」と鳴らす程度にしておきます。

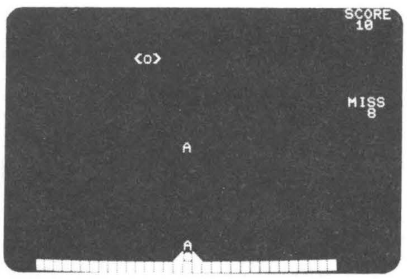
★フィニッシュ

リスト5－3は**リスト5－2**の95行にBOM=0を加え，145，147行と230から330行を追加したものです。これでスペースで弾を発射，UFOの撃墜ができるようになります。

また**リスト5－4**は**リスト5－3**の180行と190行を



《写真5－3》弾を発射できる



《写真5－4》点数表示を入れゲーム完成

変更し，85から87行，285行，340から360行を追加したものです。これでスコアやミスが表示され，ゲームオーバーにもなります(0キーで再スタートします)。

変数表(**第5－1表**)を用意しましたので，一つ一つを理解しながら順序よく遊んで下さい。

《リスト5－3》ビーム砲の弾発射プログラム

```
 10 klist 0
 20 init "crt1:40,25,0,0"
 30 kmode 0
 40 click off
 50 repeat on ,4
 60 cls
 70 locate 0,24
 80 print "■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■";
 90 CX=15
 95 UX=0:UY=12:URUN=1:BOMB=0
100 GE$=inkey$
110 if GE$="4" then CX=CX-1:if CX<1 then CX=1
120 if GE$="6" then CX=CX+1:if CX>28 then CX=28
130 locate CX,22:print " A "
140 locate CX-1,23:print " ◢■◣ ";
145 if GE$=" " and BOMB=0 then BOMB=21:BX=CX
147 if BOMB<>0 then goto 230
150 for D=0 to 200
160 next D
170 XX=UX:YY=UY:UX=UX+URUN
180 if UX=0 then URUN=1:UY=2+int(rnd(1)*15)
190 if UX=29 then URUN=-1:UY=2+int(rnd(1)*15)
200 locate XX,YY:print "   "
210 locate UX,UY:print "<O>"
220 goto 100
230 locate BX+1,BOMB:print "A"
240 locate BX+1,BOMB+1:print " "
250 if UY<>BOMB then goto 320
260 if UX>BX+1 then goto 320
270 if UX<BX-1 then goto 320
280 play "g32":BOMB=0
290 locate UX,UY:print "   ":if URUN=1 then goto 310
300 URUN=1:UX=0:UY=2+int(rnd(1)*15):goto 170
310 URUN=-1:UX=29:UY=2+int(rnd(1)*15):goto 170
320 BOMB=BOMB-1:if BOMB>0 then goto 170
330 locate BX+1,1:print " ":goto 170
```

《リスト5－4》UFO撃墜ゲーム完成版

```
 10 klist 0
 20 init "crt1:40,25,0,0"
 30 kmode 0
 40 click off
 50 repeat on ,4
 60 cls
 70 locate 0,24
 80 print "■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■";
 85 TEN=0:MIS=0
 86 locate 33,1:print "SCORE":locate 33,2:print TEN
 87 locate 33,9:print "MISS":locate 34,10:print MIS
 90 CX=15
 95 UX=0:UY=12:URUN=1:BOMB=0
100 GE$=inkey$
110 if GE$="4" then CX=CX-1:if CX<1 then CX=1
120 if GE$="6" then CX=CX+1:if CX>28 then CX=28
130 locate CX,22:print " A "
140 locate CX-1,23:print " ◢■◣ ";
145 if GE$=" " and BOMB=0 then BOMB=21:BX=CX
147 if BOMB<>0 then goto 230
150 for D=0 to 200
160 next D
170 XX=UX:YY=UY:UX=UX+URUN
180 if UX=0 then URUN=1:UY=2+int(rnd(1)*15):MIS=MIS+1:locate 34,10:print MIS:if MIS=10 then goto 340
190 if UX=29 then URUN=-1:UY=2+int(rnd(1)*15):MIS=MIS+1:locate 34,10:print MIS:if MIS=10 then goto 340
200 locate XX,YY:print "   "
210 locate UX,UY:print "<O>"
220 goto 100
230 locate BX+1,BOMB:print "A"
240 locate BX+1,BOMB+1:print " "
250 if UY<>BOMB then goto 320
260 if UX>BX+1 then goto 320
270 if UX<BX-1 then goto 320
280 play "g32":BOMB=0
285 TEN=TEN+10:locate 33,2:print TEN
290 locate UX,UY:print "   ":if URUN=1 then goto 310
300 URUN=1:UX=0:UY=2+int(rnd(1)*15):goto 170
310 URUN=-1:UX=29:UY=2+int(rnd(1)*15):goto 170
320 BOMB=BOMB-1:if BOMB>0 then goto 170
330 locate BX+1,1:print " ":goto 170
340 locate 13,6:print "G A M E":locate 13,8:print "O V E R"
350 if inkey$="0" then goto 60
 60 goto 350
```

《第5－1表》変数表

変数名	内容
TEN	得点
MIS	ミスの回数。イコール10でゲームオーバー
CX	ビーム砲のX座標。1～28の間
UX	UFOのX座標。0と29の時にUYを設定する
UY	UFOのY座標
URUN	1の時UFOを右に動かし，－1で左に動かす。UXに加算する
BOMB	0の時弾の発射が可能となる。発射後は弾のY座標となる
BX	弾のX座標
GE$	押されたキーの文字が入る
D	ループ用
XX	UXを変化させる前にUXの値を代入する
YY	UYを変化させる前に，UYの値を代入する。この後，XXとYYの座標のUFOをクリアし，UXとUFOを書込む



第6章 マシン語プログラミング入門

マシン語。または**機械語**。

それは，ＣＰＵ（Z80）にとって，唯一理解できる言語です。

私たちが日頃，スーパーＭＺをＢＡＳＩＣでしか利用していなくとも，ＣＰＵはマシン語でしか動いていません。ＣＰＵは，シャープ技術陣の作成した**「ＢＡＳＩＣ－Ｍ25（Ｓ25）インタプリタ」**という長大なプログラムを実行しているだけで，我々の入力したＢＡＳＩＣのプログラムは単なる記号でしかありません。

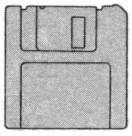
スーパーＭＺのＢＡＳＩＣは8ビット機中最高速ですから，例えば1万回ループは1.84秒で終わりますが，マシン語では何と約0.34秒です。32ＫバイトのメモリフィルはＢＡＳＩＣでは約31.7秒ですが，マシン語では約0.128秒しかかかりません。マシン語では我々の意志がＣＰＵに直接伝わることになり，さらに速く動かすことができるのです。

また，マシン語には，ビット操作などＢＡＳＩＣではない，または実用にならないような大変キメの細かな処理ができます。半面，ＢＡＳＩＣのタイリングペイントのような複雑な処理も，結局はインタプリタすなわちマシン語プログラムで処理しているのですから，「マシン語には不可能はありません」。

また，ＢＡＳＩＣのプログラムは長大なインタプリタがないと動きませんが，マシン語はそれ単独で動きます。1万回ループやメモリフィルは，わずか10数バイトで作れます。

もちろん，ＢＡＳＩＣは素晴らしい言語です。しかし，マシン語にはＢＡＳＩＣにない醍醐味があります。マシン語もまた，たいへん魅力的な言語です。

この章では，ＢＡＳＩＣもひととおり覚え，そろそろマシン語に興味を持って来た皆さんのために，**マシン語とは何か**について，話を進めて行きたいと思います。



MON↵　で広がるマシン語の世界

マシン語プログラムを見るのは，極めて簡単です。

ＢＡＳＩＣから，

ＭＯＮ↵　　〈アンダーラインの所を入力します〉

と入力すると，そこはもう，マシン語の世界です。画面には

＊■　　(←カーソル)

と，＊（アスタリスク）が出てモニタに入り，あなたのコマンドを持っていることを示しています。

続けて，

＊Ｄ↵

とやってみて下さい。

```
:００００＝Ｃ３　Ｂ８　００　００　００　００　００　００　／テサ・・・・・・
:０００８＝Ｃ３　３Ｂ　００　００　００　００　００　００　／テ；・・・・・・
:００１０＝Ｃ３　３Ｂ　００　００　００　００　００　００　／テ；・・・・・・
:００１８＝Ｃ３　８５　０１　００　００　００　００　００　／テ※・・・・・
:００２０＝Ｃ３　３Ｂ　００　Ｃ３　２６　０１　００　００　／テ；・テ＆・・・
:００２８＝Ｃ３　Ａ０　０１　００　００　００　００　００　／テ　・・・・・・
                        :
                        :
                        :
```

と言った具合に，何やら次々と画面に表示されます。実は，このＣ３　Ｂ８　００…という部分がマシン語なのです。

もう少し詳しく説明しましょう。

まず，０００００や０００８の部分は，Ｚ80のアドレスを示しています。アドレスというのは直訳すると**「番地」**という意味ですが，実際にはメモリ（ＲＡＭ）が配置されていますので，ここでは単に箱のようなものがたくさん並んでいると考えて良いでしょう。アドレスはこのモニタのように16進数で表現するのが普通で，００００Ｈ～ＦＦＦＦＨまであります（００００とＦＦＦＦの後に付いている**「Ｈ」**というのは Hexadecimal つまり16進数の意味で，10進数との区別を明確にするために付けます。

つまり，箱は６５５３６（64Ｋ）あるわけです。

次に，Ｃ３　Ｂ８　００　００ …というのは，各々のアドレス，つまり，メモリに格納されている値を２桁の16進数で示しています。２桁ですから，必ず００Ｈ～ＦＦＨまでのいずれかになり，何も入っていないということはあり得ません。ただ，何も意味のある値は入っていないということはあり得ます。しかしその場合でも，００ＨとかＦＦＨと言った，何かしらの値が入っています。この例の場合は，

アドレス００００Ｈのメモリには３Ｃ３Ｈという値が，
アドレス００００１Ｈのメモリには８Ｂ８Ｈという値が，
アドレス００００２Ｈのメモリには００Ｈという値が，
：
：
：

入っていることを示しています。アドレスと入っている値とを1対1で並べて表示しても良いのですが，それではかえって見づらくなりますので，アドレスは8バイト置きにしか表示しないのです。

ところで，各メモリに格納されている値は，マシン語プログラムである場合のほか，そのプログラムで使用するデータである場合があります。プログラムとデータの区別は必ずしも明確ではなく，適当に混在しているのが普通ですが，そのプログラムの作成者は，データの部分を誤まってプログラムとして実行することのないよう注意しています。例えば，サブルーチンとの区切り目にデータを置くようようにしているのです。

最後に　テサ……　という部分は，メモリに表示用か何かのための**「文字列」**のデータが入っているものと考えた場合，その文字列が一目瞭然で分かるように設けられた，**キャラクタダンプ（アスキーダンプ）**です。キャラクタコード20Ｈ未満のコントロールコードに当る部分は文字列として表示できないので，「．」に置換えて表示しています。スペースにしても良いのですが，その数を容易に数えられるよう，ピリオドを使っているのです。

００００Ｈ～００００７Ｈの「テサ……」は，見て分かるように，文字列データではありませんが，例えば

＊Ｄ２４Ｆ８↵

として，アドレス２４Ｆ８Ｈ番地以降を見てみると，　ｉｎ　とか　＊Ｂｒｅａｋ　と言った文字列が見られます。

さて，これで皆さんは，モニタの**「Ｄコマンド」**の意味が分かったと思います。モニタにはまだまだコマンドがありますが，とりあえずは　Ｄ↵　を繰り返し，メモリのあちこちを眺めてみて下さい。特にモニタは初めてという方は，少なくとも10時間くらいは，これをやってみて下さい。

また，この項ではしきりに**16進数**という言葉を使って来ましたが，その意味についてまだ良く分からない方は，次項を良く読んで下さい。

Z80の歩みはMZの歩み？

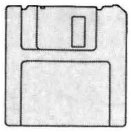
Ｚ80は，1975年に米国Ｚｉｌｏｇ（ザイログ）社によって開発された8ビットＣＰＵです。日本では，シャープ（株）がＺｉｌｏｇ社と技術提携を行い，Ｚ80ＣＰＵやファミリＬＳＩの実施権を保有しています。

Ｚ80開発当時，マイクロプロセッサとしては，米国Ｉｎｔｅｌ（インテル）社の開発した「ｉ8080」が多大な支持を集めていましたが，Ｚ80はその「ｉ8080」の機能をすべて持った上で，さらに独自の拡張を加えたＣＰＵとして設計されました。

Z80は，ハードウエア的にも優れたアイデアを持っており、現在では極めて一般的なＲＡＭの一種である，Ｄ－ＲＡＭ（ダイナミック ラム）との接続が容易にできるよう，リフレッシュコントロール回路を備えていました。そのため，16Kバイト以上のＲＡＭを使用するマイクロコンピュータ応用機器では大変な勢いで普及し，シェアを著しく伸ばしました。その結果，現在のパーソナルコンピュータの多く， 特に8ビット機では大半のものに採用されるに到っています。

Z80はクロック2.5ＭＨｚまで利用できましたが，その後現在までに，実行速度の上限を上げたZ80Ａ（4ＭＨｚ）やZ80Ｂ（6ＭＨｚ）が開発されています。これはそのままシャープＭＺシリーズの歩みであるとも言えます。Ｚ80はＭＺ－80Ｋ～ＭＺ－1200，Z80ＡはＭＺ－80Ｂ～ＭＺ－2200，Z80ＢはＭＺ－2500と，Z80の進化に合わせて，ＭＺシリーズも発展してきたわけです。

Z80ＡやZ80Ｂは，高速処理が可能となるような工夫がなされていますが，内部の基本的な構造や命令については初代のZ80と全く変わりません。つまり，ソフトウエア的には完全コンパチブルだということです。よって，本文中の説明では，Z80・Z80Ａ・Z80Ｂ等を特に区別することなく，単にZ80と呼んでいます。



16進数と2進数

スーパーＭＺのＢＡＳＩＣには，**＆Ｈ（＄）**や**ＨＥＸ＄**といった16進数処理のための機能がありますので，16進数については既に御存知の方も多いと思いますが，まだの皆さんのために，ここで簡単に説明しておきましょう。また，2進数についても同様に触れておきたいと思います。

10進数が**0～9までの数字**を使い，9の次は桁上がりして10となるのと同じように，**16進数では 0～Ｆまでの数字**を使い，Ｆの次は桁上がりして10となります。また，**2進数では0～1までの数字**を使い，1の次は桁上がりして10となります。10進数と違うのは，単に，これだけです。

16進数で0～Ｆを使うのは変な感じもしますが，人間社会では昔から10進数を使って来たため数字は0～9しかありませんので，便宜的にＡ～Ｆのアルファベットを10進数で言う10～15の値を表わす数字（Ａ＝10，Ｂ＝11，Ｃ＝12，Ｄ＝13，Ｅ＝14，Ｆ＝15）に見たて，使用するわけです。16進数の10Ｈは，10進数の16になります。16で桁上りするから16進数と言うのです。

ところで，10進数の１２３４が

$10^3*1+10^2*2+10^1*3+10^0*4$

を示すように，16進数の4Ｄ3ＡＨは，

$10H^3*4H+10H^2*DH+10H^1*3H+10H^0*AH$

を示します。言い換えると，

4096＊4＋256＊ １３＋１６＊３＋１＊10

となり，10進数の１９７７０を意味することになります。

2桁の16進数（例えばC3H）の場合は，上2桁が００の16進数であると考え，

$10H^3*0H+10H^2*0H+10H^1*CH+10H^0*3H$

つまり　４０９６＊０＋２５６＊０＋１６＊１２＋１＊３（＝１９５）と計算しても良いのですが，単に，

$10H^1*CH+10H^0*3H$

と考え，１６＊１２＋１＊３＝１９５としても結構です。大差ありません。

2進数の計算方法は16進数と同様の考え方でできます。０１００１１０１００１１１０１０Ｂ(2進数の後には「B」を付けることがあります）は，

$10B^{15}*0B+10B^{14}*1B+10B^{13}*0B+10B^{12}*0B$
$+10B^{11}*1B+10B^{10}*1B+10B^9*0B+10B^8*1B$
$+10B^7*0B+10B^6*0B+10B^5*1B+10B^4*1B$
$+10B^3*1B+10B^2*0B+10B^1*1B+10B^0*0B$

を示します。言い換えると，

３２７６８＊０＋１６３８４＊１＋８１９２＊０＋４０９６＊０
＋　２０４８＊１＋　１０２４＊１＋　５１２＊０＋　２５６＊１
＋　　１２８＊０＋　　６４＊０＋　　３２＊１＋　　１６＊１
＋　　　８＊１＋　　　４＊０＋　　　２＊１＋　　　１＊０

となり，10進数の１９７７０を意味します。

ところで，アドレスは，ハードウエア上では0Vと5Vの二つの状態を示す電気信号線16本で表現しています。また，データ（入っている値）も同様のもの8本で表現しています。従ってこれをソフトウエア的に表現するには，16桁（16ビット）や８桁（８ビット）の２進数を使うのが適切です。しかし，それではあまりにも長くなってしまい，大変見づらいので，４桁（４ビット）毎に１桁の16進数に変換して，結局４桁または２桁の16進数として表現する方法が考案されました。

つまり，

２進数		16進数	10進数	２進数		16進数	10進数
００００	…	0	0	１０００	…	8	8
０００１	…	1	1	１００１	…	9	9
００１０	…	2	2	１０１０	…	A	１０
００１１	…	3	3	１０１１	…	B	１１
０１００	…	4	4	１１００	…	C	１２
０１０１	…	5	5	１１０１	…	D	１３
０１１０	…	6	6	１１１０	…	E	１４
０１１１	…	7	7	１１１１	…	F	１５

という規則に従って変換するわけです。

例えば，０１００１１０１００１１１０１０Ｂという2進数のアドレスは，

４　D　３　A

４Ｄ３ＡＨとなるわけです。また，１１００００１１Ｂという2進数の値は，

C　3

Ｃ３Ｈとなるわけです。もちろん，この逆に16進数から2進数へ変換することもできます。ただ，2進数4桁と16進数1桁を直接変換しても良いものか？と疑問に思う方は，さきほどの，

```
  ３２７６８＊０＋１６３８４＊１＋８１９２＊０＋４０９６＊０
＋　２０４８＊１＋　１０２４＊１＋　５１２＊０＋　２５６＊１
＋　　１２８＊０＋　　６４＊０＋　　３２＊１＋　　１６＊１
＋　　　８＊１＋　　　４＊０＋　　　２＊１＋　　　１＊０
```

をさらに書換えると，

```
  ４０９６＊（８＊０＋４＊１＋２＊０＋１＊０）
＋　２５６＊（８＊１＋４＊１＋２＊０＋１＊１）
＋　　１６＊（８＊０＋４＊０＋２＊１＋１＊１）
＋　　　１＊（８＊１＋４＊０＋２＊１＋１＊０）
```

に，そしてさらに書換えると，

```
  ４０９６＊４＋２５６＊１３＋１６＊３＋１＊１０
```

となる点に注目すれば，問題ないことが実感としてつかめると思います。

なお，マシン語には16進数は必要不可欠ですので，スーパーＭＺにあるスーパーヘルプ機能の10進・16進電卓を活用して，充分に親しんでおいて下さい。それと，例えば「4D3AH」は「ヨンデーサンエー（エッチ）」と発声すれば良いのですが，全桁が0～Ｆのうちのたまたま0～9の部分しか使わないような時，例えば「2000H」を「ニセン」と呼んではいけません。「ニーゼロゼロゼロ（エッチ）」または「ニーマルマルマル（エッチ）」または「ニーレイレイレイ（エッチ）」などと言うように，必ず棒読みして下さい。そうでないと，10進数と区別が付かなくなり，大変分かりづらくなります。

1K＝1024？？？

例えば，距離が10km，という場合，それは10000ｍを示しています。

また，重さが48kg，という場合，それは48000ｇを示しています。

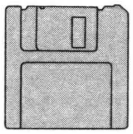
しかし，メモリの容量が64Kバイト（Ｂｙｔｅ）という場合，それは64000バイトを示すのではなく，65536バイトを言うのです。

本来，小文字の「ｋ」という記号は「キロ」と読み，1000倍であることを表わすものですが，コンピュータの世界では，大文字の「K」という記号を「ケー」と読み，1024倍であることを表わすものとして利用しています（ただ昔は，65536バイトを65ｋ（キロ）と言っていたメーカーもあったようです）。

これは，1024というたまたま1000に極めて近い値（差は2.4％）が，たまたま2^{10}に当たり，コンピュータにとって区切りが良いため，ｋ（キロ）と同じような使い方をしているのです。ＲＡＭやＲＯＭと言ったメモリや，ＣＰＵのアドレス線が10本あると$2^{10}=1024$で1Kバイト，11本あると$2^{11}=2048$で2Kバイト，12本で4Kバイト，13本で8Kバイト…という具合に容量は1000で割り切れることがなくとも，（アドレス線が10以上あれば）1024

では必ず割り切れるということなのです。また，フロッピーディスク等もこれに合わせ，１Ｋバイト＝1024バイトとしています。

なお，アドレス線が20本になると2^{20}で1048576バイト，つまり1024Kバイトになるのですが，これを1M（メガ）バイトと言います。さらにアドレス線30本では2^{30}で1073741824バイト，つまり1048576Kバイト，つまり1024Mバイトになるのですが，これを1G（ギガ）バイトと言います。また，G（ギガ）の上はT（テラ）になるでしょう。オーディオ用のコンパクトディスク（ＣＤ）は外部記憶装置として使うと500Mバイト以上，ＶＨＤビデオディスクをデジタルオーディオ用に使うＡＨＤシステムでは（ＶＨＤビデオディスク）にG（ギガ）バイトクラスのデータが入ります。光ディスクや光磁気ディスクも同様です。将来，これらをパソコンで利用できるようになると，これらの単位にもお目にかかるようになるかも知れません。



CPUの動きはリセットから

この項と次の項では，ちょっと難しい話が続きます。抽象的な表現も増えますが，じっくりと気分を落ち着けて読んで下さい。マシン語の感覚がつかめるようになるでしょう。

ＣＰＵは，電源が投入された後，リセット信号の入力によって，初めて意味のある動作を始めます。

リセットがかかると，プログラムカウンタ（メモリのどこを実行しているかを示す）に００００Ｈが入り，すぐさまプログラムカウンタの指すメモリからデータを読取ります。つまり，ＣＰＵは００００Ｈ番地からデータを読取ることになります。Ｃ３Ｈとか言った値が返って来ることでしょう。

このデータは命令として解釈され，その種類によっては，続くアドレスにある1バイト～3バイトのデータを，命令の一部または命令のパラメータとして次々に読込み，その命令の実行に移ります。プログラムカウンタは順に1ずつ増えて行きます。

一つの命令の実行が終わると，原則としてさらに次のアドレスにあるデータを命令として読込み，同様の処理を行います。例外はプログラムカウンタを操作する命令て，これを実行しますと，次の命令を読取るアドレスが変更されます。ＢＡＳＩＣのＧＯＴＯやＧＯＳＵＢに相当する命令（ＪＰ，ＣＡＬＬ等）が，それです。

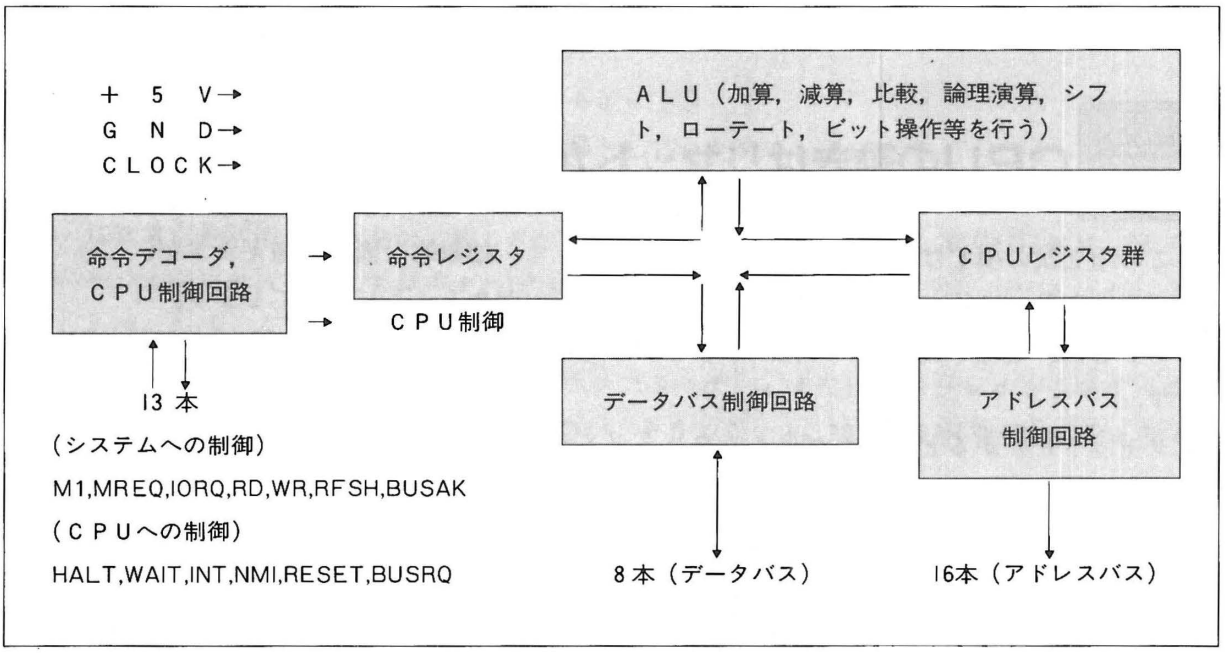
これらの動作は電源を切るまで続けられ，ＣＰＵは終始，命令の取込みと実行とを繰返します。

実際，ＣＰＵが行う処理というのは，その種類を問わず，だいたいこのようなものです。リセット後００００Ｈから実行を行う，という点だけは違うものもありますが，基本的な考え方は８ビットＣＰＵでも16ビットＣＰＵでも，あるいはパソコン以外のコンピュータでも全く変わりません。

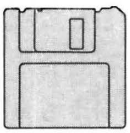
ＣＰＵの実行できる命令は，極端な言い方をすると，**「データ転送（代入）命令」と「演算命令」**の2種類に分けることができます。ＣＰＵ内部には**「レジスタ」**と呼ばれるＢＡＳＩＣにと

っての変数のようなものが，外部には，64Kバイトの**「メモリ」**がありますが，ＣＰＵのすべての命令は，この**「レジスタ」**と**「レジスタ」**の間，または**「レジスタ」**と**「メモリ」**の間，またはプログラムで直接指定された値と**「レジスタ」**や**「メモリ」**との値でデータの転送や演算を行うものばかりです。文字通り単なる計算機でしかありません。

パソコンで採用されているＣＰＵの場合，「演算」の種類はＢＡＳＩＣと比べ浮動小数点（実数）を扱う高度なものがない代り，ビット単位の演算が強化されたようなものになっています。また，Ｚ80やそれより前の時代のＣＰＵでは，四則演算のうち乗算や除算も省略または簡略化されています。乗算，除算，実数の四則演算，それにｓｉｎやｌｏｇと言った数値関数は，すべて，整数の値同士の加算や減算，またはビット単位の演算命令を巧みに組合わせることで，実現させています（スーパーＭＺの場合，そのプログラムはＩＰＬ・ＢＩＯＳ　ＲＯＭに格納されています）。



《第6-1図》Ｚ80の内部構造



基本はレジスタ

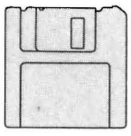
Ｚ80の命令は，細かく数えると158種類ほどあります。また，命令名が同じでもオペランド(対象となるレジスタやメモリ）が異なるものを分けて数えると，700近くに上りますが，ＢＡＳＩＣの命令や関数同様，その全部を覚える必要はありません。また，全部を使う必要もありません。せいぜい4分の1も知っていれば充分です。それに，今も言ったように，158個の命令はいずれも，単純な働きをするものばかりで，マシン語の基本的な考え方さえつかんでしまえば，後の命令は次々と理解することができます。マシン語は一般に難しいと言う人が多いのですが，一つ一つの命令を見てみれば，ＢＡＳＩＣなどよりはるかに簡単です。ただ，ある目標があって，それを実現するようなプログラムをＢＡＳＩＣかマシン語で作るような場合，単純な命令をたくさん組合

わせて作るマシン語より，高度（複雑）な命令数個で作れるＢＡＳＩＣの方が楽である，という印象は受けるかも知れません。ですから，最初からマシン語だけの長大なプログラムを組もうとはせずに，ほんの数バイトで各々の命令を確認してみたり，ＢＡＳＩＣのどうしても時間のかかるような所をマシン語サブルーチンに置換えて，ＢＡＳＩＣ＋マシン語の組合わせのプログラムを作りながら，徐々に理解してゆくのが良いと思います。

ところで，「レジスタ」はＢＡＳＩＣの変数のようと言いましたが，扱える値や，（レジスタの）個数は厳しく制限されています。あるいは，初めからある範囲の値を扱うレジスタが，決められた数しかないと言っても良いでしょう。

また，Ｚ80やｉ8086（16ビット）と言ったいわゆる**「80系」**のマイクロプロセッサでは，同じ範囲を扱えるレジスタでも，そのレジスタによって向き・不向きの命令があり，全部のレジスタが同等に扱えるわけではありません。つまり，レジスタ毎に，**「オペランド」**になり得る命令と，「オペランド」になり得ない命令とが，かなりバラバラに分かれているということです。ですから，マシン語のプログラムを作れるようなるには，各命令の働きとともに，レジスタの使い方をも覚える必要もあるのです。ただ，「働き」は理解する必要があるのに対し，こちらはまさに「習うより慣れろ」という感じで，いつの間にか自然に身に付いてゆくのが普通です。しかし，それでも，レジスタの名前くらいは最初に覚えておくのが得策だと思います。

一方，レジスタが限られた数しかない点については，もし，ＣＰＵ内のレジスタが足りなくなった場合には，その時に必要でないレジスタの内容を一時的にメモリの一部に確保した「スタック」という特別なエリアに退避させ，そのレジスタを利用する，という手法で解決させます。そして，計算等が終わったなら退避していた値をレジスタに復帰させれば良いわけです。しかし，それでもレジスタが足りない時や，大量のデータを一度に扱うような場合には，一般の「メモリ」を直接利用することになります。



レジスタの種類、いろいろ

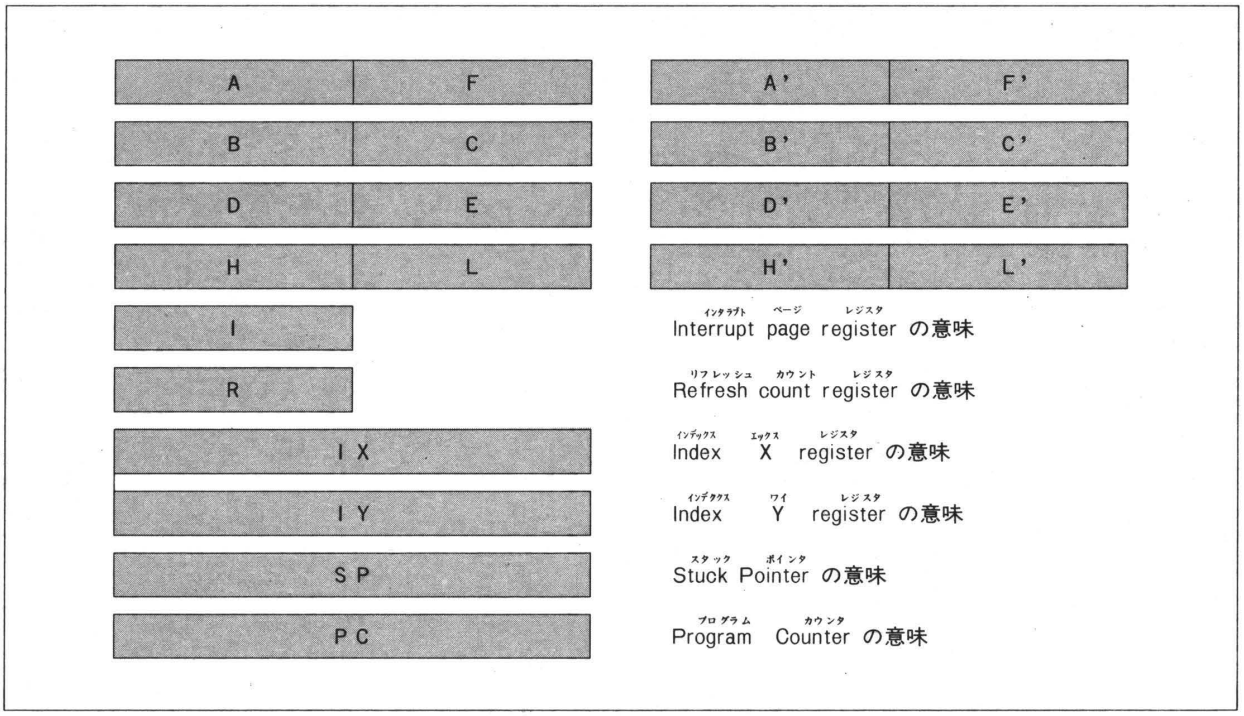
いかがでしたでしょうか。初めてモニタを操作したばかりの皆さんには，ちょっと難しかったかも知れません。しかし，最初にぜひとも覚えてほしいのは，

ＣＰＵの中には「レジスタ」という大変重要な働きをするものがある

という点です。あとの部分はいずれ分かるようになると思います。では，その重要な働きをする，レジスタから見てゆきましょう。

Ｚ80には，**第6―2図**のようなレジスタがあります。

レジスタの大きさは，実際の長さに比例して書いていますから，Ａレジスタやレジスタはメモリ1バイトの長さ・大きさと同じ8ビット，ＩＸレジスタやＩＹレジスタは倍の16ビットです。つまり，Ｚ80には，8ビットのレジスタは全部で18個，16ビットのレジスタは全部で4個あるわけです。8ビットレジスタには16進数で００Ｈ～ＦＦＨ，10進数で考えると０～２５５（－１２８～＋１２７としても扱える）の値を，16ビットレジスタには16進数で０００００Ｈ～ＦＦＦＦＨ，10進数で考えると０～６５５３５（－３２７６８～＋３２７６７としても扱える）の値を入れてお



《第6-2図》Z80のレジスタ

くことができます。前にも触れたように，ＢＡＳＩＣでいう単精度または倍精度実数を直接扱えるレジスタはありません。

枠をつないで並べたレジスタ同士，例えば，Ｈレジスタとレジスタとは，連結して一つの16ビットのレジスタとして使うこともできます。この時，連結してできたレジスタのことは，「ＨＬレジスタ」と呼びます。ＨＬレジスタには０００００Ｈ～ＦＦＦＦＨの値を入れておくことができるわけです。また，ＤレジスタとＥレジスタなら，ＤＥレジスタ，ＢレジスタとＣレジスタならＢＣレジスタと呼びます。またこのように，連結して利用することを，**ペアレジスタ**として使う，と言うこともあります。8ビットのレジスタ二つとして使うか，16ビットのペアレジスタ一つとして使うかはプログラマの自由ですが，それらの区別は必ずしも明確でなく，一つ一つの命令ごとに,8ビットの扱いにしたり，16ビットの扱いにしたりを，自由に決められるのです。ただ，詳しくは後述していますが，Ｈレジスタ・Ｌレジスタはペアレジスタ（ＨＬレジスタ）として利用する機会が多く，Ｂレジスタ・Ｃレジスタは逆に単独で使うことが多くなります。Ｄレジスタ・Ｅレジスタはその中間になります。これは，ＨＬレジスタ用には，16ビットの演算命令などが多く用意されているからです。

さて，これからのレジスタは，大きく2種類に分けられます。

まず一つは，初めから半ば強制的に用途が決められており，他の用途にはなかなか利用し難い，または通常使用しないものです。Ｉ，Ｒ，ＳＰ，ＰＣがそれで，専用レジスタと呼ぶこともあります。また，Ｆレジスタも，これに準じていると言えます。

もう一つは，プログラムによって，あるいは，サブルーチン毎に，または，その場に応じて自由に使うことができるレジスタです。残りのレジスタは皆これで，汎用レジスタと呼びます。ただ，さきほどの話の通り，汎用レジスタの中でも命令によってオペランドになれる・なれないが

あり，向き・不向きがでて来ます。

しかし，初めのうちは，汎用レジスタのうち，

A　B　C　D　E　H　L

と専用レジスタの

F（フラグ）の中の**Ｃｙフラグと Ｚフラグ**

だけを覚えれば良いでしょう。

第６－１表に全部のレジスタの働きをまとめておきました。また，マシン語に関する様々な話も，最も関連の強いレジスタの所に入れておきましたから，いずれ，プログラムを組んでいて何か分からないことが出て来たなら，その都度参照すると良いと思います。もし，この表の内容を完全に理解できるようになったなら，その頃には様々なプログラムを自由に組めるようになっていることでしょう。

《第6－1表》Z80のレジスタの利用方法

A	他のレジスタや直接指定した値との加算，減算など，数多くの算術演算や論理演算等が可能です。そして，データの受け渡しも最も自由に広範囲にできるため，転送の中継ぎを行ったり，パラメータが8ビット長のもの一つで良いようなサブルーチンとのデータの受け渡し(例えば画面への1文字表示やキーボードからの1文字入力)にも使われ，文字通り何にでも利用されます。 なお，Aレジスタは，アキュームレータと呼ぶこともあります（特に他のＣＰＵを扱ったことのある人の場合)。
F	このレジスタは，8ビットあるうち，実際には6ビットだけが，使われています。 各ビットは，プログラマが意識して変えなくとも，演算などの命令を実行するだけで，勝手に（実は規則的に)，しかも，6ビットがバラバラに変化してしまいます。しかしこれらは，プログラマにとって必要な況状を，的確に知らせてくれる，とても重要な働きを持っています。 これらフラグのビット構成は，次の通りです。各フラグは，常に0か1の値を持ちます。また，0に変化させることを「リセットする」，1に変化させることを「セットする」と言うことがあります。

7	6	5	4	3	2	1	0
S	Z		H		P/V	N	Cy

S……サイン（Sign）フラグ，または，Ｓフラグと呼びます。
8ビットの演算命令の大半と，その他のいくつかの命令を実行すると，変化します。ただし，あまり利用する機会はありませんので，

初めのうちは働きを覚える必要はないでしょう。

Sフラグは，8ビットのデータなら－128～127を，16ビットのデータなら－32768～32767を表わしていると考えた場合，演算等の結果が負になったならセットされ，正になったならリセットされます。

単純に言えば，演算等の結果の値の最上位ビットが，そのままコピーされるということです。

このフラグの内容は，ＪＰ　Ｍ，ｘｘｘｘ・ＪＰ　Ｐ，ｘｘｘｘ・ＲＥＴ　Ｐ・ＲＥＴ　Ｍ　の各命令によって，簡単に調べることができます。

なお，このフラグは，実際には特定のメモリ・レジスタ・Ｉ／Ｏポートの内容の最上位ビットを，ＯＲ　Ａ命令，または，ＡＮＤ　Ａ命令を使って調べる時に多く利用します。8ビットを－128～127として扱うことは，まず，ないからです。

Z……ゼロ（Ｚｅｒｏ）フラグ，または，Ｚフラグと呼びます。

8ビットの演算命令の大半と，との他のいくつかの命令を実行すると，変化します。Ｓフラグが変化する命令では，一緒にＺフラグも変化します。Ｚフラグは，Ｃｙフラグとともに参照される機会のとても多いフラグです。

Ｚフラグは，データが全ビットとも全部0，すなわち，演算等の結果が0になったならセットされ，それ以外の値になったならリセットされます。比較命令では，値が等しい時つまり引いた結果が0になった時にだけセットされると考えることもできます。また加算命令では，結果が0でなくとも，転送された値が0なら，セットされます。例えば，8ビット命令で60Ｈ＋90Ｈを実行すると結果は100Ｈですが，Ａレジスタに格納されるのは00Ｈなので，Ｚフラグはセットされます。

また，ＢＩＴ命令で，特定のレジスタ・メモリの，指定したビットが0か1かを示します。0ならセット，1ならリセットされるわけです。

このフラグの内容は，ＪＰ　Ｚ，ｘｘｘｘ・ＪＰ　ＮＺ，ｘｘｘｘ・ＪＲ　Ｚ，ｘｘｘｘ・ＪＲ　ＮＺ，ｘｘｘｘ・ＲＥＴ　Ｚ・ＲＥＴ　ＮＺ　の各命令によって，簡単に調べることができます。

H……ハーフキャリー（Ｈａｌｆ　ｃａｒｒｙ）フラグ，または，Ｈフラグと呼びます。

このフラグの内容は，ＤＡＡ命令によって参照されるだけで，プログラマが利用することは原則としてできません。ですから，このフラグは，あることだけを心の片隅にでも留めておくだけで，充分です。

このフラグは8ビット16ビットの加算・減算によって値が変わりま

す。加算では8ビットの場合は3ビット目から4ビット目に対する，16ビットの場合は11ビット目から12ビット目に対する桁上がり（キャリー）の有無がコピーされます。

減算では8ビットの場合は4ビット目から3ビット目に対する，16ビットの場合は12ビット目から11ビット目に対するボローの有無がコピーされます。キャリー・ボローが生じた場合は1に，そうでない場合は，0になります。

P／V……パリティ（Ｐａｒｉｔｙ）フラグ，または，オーバーフロー（Ｏｖｅｒｆｌｏｗ）フラグ，または，P／V（ピーブイ）フラグと呼びます。Sフラグより，さらに利用する頻度の少ないフラグです。

このフラグはPフラグとして使うことと，Vフラグとして使うことがあります。これらは，特に区別されることはなく，各命令ごとにPフラグとしての結果が入ったり，Vフラグとしての結果が入ったりします。

Pフラグとして使われるのは論理演算命令の際で，8ビットのデータのうち，セットされている（1になっている）ビットが，奇数個（1，3，5，7個）あれば0に，偶数個（0，2，4，6，8個）あれば1になります。結局，フラグを含めた全9ビットのデータの中には，必ず奇数個の1のビットが存在することになります（これを奇数パリティと呼びます）。

Vフラグとして使われるのは加減算の際で，値が2の補数を表わしている(8ビットでは－128～＋127，16ビットでは－32768～＋32767として使用すること）と考えた場合，演算が正しく行われたかどうかを持ちます。一般の命令は，数値は正の整数として扱われ，計算されたり，Ｃｙフラグもそれに従って変化するわけですが，このフラグは2に補数として計算したものとみなして，エラーの有無を知らせるものです。ですから，もし，正の値として計算したのならＣｙフラグを，2の補数として計算したのならVフラグを見れば，正しい結果が得られたかどうかが分かります。

Vフラグとして，このフラグがセットされる（演算がオーバーフローしたと知らせて来る）のは，以下の場合です。

① 同符号同士の加算で，結果が異符号になった場合。

(例) 41H＋42H→83H

2の補数では，

41H＋42H→－7DHを意味するからです。

(例) 7FH＋01H→80H

2の補数では，

7FH ＋01H→－80Hを意味するからです。

② 異付号の値を引いた結果，符号が減算と同じになった場合。

(例) C0H－70H→50H

2の補数では，

－40H－40H→50Hを意味するからです。

(例) 70H－C0H→B0H

2の補数では，

70H－（－40H）→－50Hを意味するからです。

(例) 80H－01H→7FH

2の補数では，

－80H－01H→7FHを意味するからです。

以下の場合はリセット（0になる）され，2の補数としての計算が正しく行えたことを知らせます。

① 異符号の数値による加算。

(例) C0H＋70H→30H

2の補数では，

－40H＋70H→30Hを意味するからです。

② 同符号同士の加算で，結果が同符号になった場合。

(例) 21H＋22H→43H （2の補数でも同意）

(例) F0H＋E0H→D0H

2の補数では，

－10H＋（－20H）→－30Hを意味するからです。

③ 同符号同士の減算。

(例) 5AH－41H→19H （2の補数でも同意）

④ 異符号の値を引いた結果，符号が，被減数と同じになった場合。

(例) 60H－F0H→70H

2の補数では，

60H－（－10H）→70Hを意味するからです。

また，このフラグの特殊な使われ方としては，LDI・LDD・CPI・CPDの各命令で，残りバイト数の有無を調べたり，LD A，R・LD A・Iの各命令で，IFF2の内容がコピーされたりもします。なお，IFF2というのは，DI命令とEI命令のうち，どちらが最も近くに行われたかを持つ，特殊フラグで，リセット直後とDI命令実行後には0，EI命令実行後には1になります。ただ，通常は参照の必要がないため，Fレジスタの中には入っていません。

このフラグの内容はJP PE，xxxx・JP PO， xxx

x・RET　PE・RET　PO　の各命令によって，簡単に調べることができます。

N……ネガティブ(Ｎｅｇａｔｉｖｅ)フラグ，または，Nフラグと呼びます。8ビットの演算等によって変わります。ＤＡＡ命令のアルゴリズムが加算用と減算用とでは異なるため，ＣＰＵはその区別のために，加算と減算のどちらが最近行われたかをこのフラグに残します。ＡＤＤ・ＡＤＣ等の命令（加算）の後では0に，ＳＵＢ・ＳＢＣ等の命令（減算）の後では1になります。

このフラグの内容は，Ｈフラグ同様，ＤＡＡ命令によって参照されるだけでプログラマが利用することは原則としてできません。ですから，このフラグは，あることだけを心の片隅にでも留めておくだけで充分です。

Ｃｙ……キャリー（Ｃａｒｒｙ）フラグ，または，Ｃｙ（シーワイ）フラグと呼びます。Ｃレジスタとの混乱を避けるため，Ｃフラグと呼ぶことはありません。

8ビットの演算命令の大半（Ｓフラグ・Ｚフラグよりは少ない）によって値が変わります。しかし，利用する機会のとても多いフラグです。本文中でもう一度説明しています。

加算の後では，最上位ビットからの桁上がり（キャリー）があった場合にセットされ，なかった場合にはリセットされます。

減算の後では，最上位ビットのボロー（算数で言うところの「左どなりの桁から借りてくる」ことを，こう言います）があった場合にセットされ，なかった場合にはリセットされます。

論理演算の後では，リセットされます（0になります）。

ほかの演算では，レジスタの特定のビットの状態によって変化したり，逆に，Ｃｙフラグの内容がレジスタの特定のビットにコピーされたりします。このため，Ｃｙフラグの内容を強制的に変えたいことがありますが，セットするにはＳＣＦ命令があります。リセットするには専用の命令はありませんが，ＯＲ　Ａまたは，ＡＮＤ　Ａで代用するか，他のフラグ（特にＺフラグ）を変化させたくない場合には，一旦ＳＣＦ命令でセットしてからＣＣＦ命令で反転させます。

このフラグの内容は，ＪＰ　Ｃ，ｘｘｘｘ・ＪＰ　ＮＣ，ｘｘｘｘ・ＪＲ　Ｃ，ｘｘｘｘ・ＪＲ　ＮＣ，ｘｘｘｘ・ＲＥＴ　Ｃ・ＲＥＴ　ＮＣ　の命令によって，簡単に調べることができます。

Ｆレジスタの，この他のビット（空白になっている2ビット）は，特に使われることはありませんし，内容を調べることも原則としてできません。

また，ＰＵＳＨ　ＡＦやＰＯＰ　ＡＦ，またＥＸ　ＡＦ，ＡＦ’の各命令を除

	いて，フラグが一まとまりとなって，Fレジスタを構成していることは，考えないで良いでしょう。
B	Aレジスタを使用中に，別の計算を行いたい場合，一時的にデータの退避をする際に，利用します。 ループのカウンタとして，DJNZ命令で，使用します。
C	データの退避を行う際，Bレジスタがすでに使われている時に利用します。 I／Oポートとのデータ交換を行う場合（IN命令，OUT命令）に，使用することができます。
D	データの退避や，パラメータの引き渡しに時々使われますが，どちらかというと，DEレジスタとして利用されることが多いため，単独の8ビットレジスタとして使うことは，あまり多くありません。
E	Dレジスタと同じです。
H	データの退避や，パラメータの引き渡しに使えますが，DEレジスタにも増して，ペアレジスタとして（HLレジスタとして）使うことが多いため，単独の8ビットレジスタとして利用することは，まず，ありません。
L	Hレジスタと同じです。
AF	このペアレジスタは，Aレジスタ，または，Fレジスタの内容を一時的にスタックに退避させたい場合に，ペアとなります。よって，これによってメモリの特定アドレスをポイントするようなことはありません。 また，EX　AF，AF’命令を利用すると，AレジスタとA’レジスタが入れ換わり，同時にFレジスタとF’レジスタも入れ換わります。これも，一種のペアレジスタとしての使われ方と考えることができます。
BC	257回以上のループを行う時，Bレジスタが上位8ビット，Cレジスタが下位8ビットを表わしたカウンタとして利用します。 BCレジスタの指すメモリと，Aレジスタとの間ではデータの転送が可能ですので，この用途に使われることも，まれにあります。

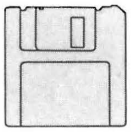
DE	BCレジスタ同様，特定のメモリをポイントして，Aレジスタとのデータ転送ができます。 逆に，DEレジスタをカウンタとして使うことはほとんどありません。 なぜ，BCレジスタとDEレジスタの役割分担がこのように決まっているかというと，EX DE，HLという命令はあるが，EX BC，HLという命令はない，すなわち，DEレジスタはポインタとして非常に操作性に優れたHLレジスタとのデータ交換命令があり，（HLレジスタの）退避用に使うことができるからです。これは，Z80の前身であるi8080で慣例となった上に，Z80ではBレジスタにループ用のDJNZ命令，CレジスタにINレジスタ,(C)・OUT （C），レジスタの各命令が拡張されたため，より，はっきりしています。 また，画面に対して文字列を出力する場合の文字列エリアや，キーボードから文字列を入力する際のバッファ（転送するエリア）や，LDIR・LDDR命令等で,(HL)側からのデータ転送の宛先をポイントする用途にも使われます。
HL	ペアレジスタの中では，最も，ペアとして使われる頻度の高いレジスタです。 まず，HLのポイントするメモリ———アセンブリ言語では，（HL）と書きます———は，B，C，D，E，H，Lの各レジスタに共通して使える命令のほとんどが利用できます。Aレジスタとのデータ転送はもとより，Aレジスタにおける演算命令のオペランドとして，など，（HL）は擬似レジスタとしての性格を持つようになります。 また，Aレジスタが8ビットの演算用に使われたように，HLレジスタは16ビットの演算として，BCレジスタ・DEレジスタ等との加算・減算に利用します。
AF'	A' レジスタとF' レジスタのペアレジスタは，A' F' レジスタとは呼ばずに，AF' レジスタと言います。 このレジスタは，前述のようにEX AF，AF' 命令で表と交換できるだけで，いつもは裏に隠れています。よって，AF' やBC' 等は，裏レジスタとか，補助レジスタと呼びます。
BC'	EXX命令によって，BCレジスタと交換することができます。
DE'	EXX命令によって，DEレジスタと交換することができます。

HL'	ＥＸＸ命令によって，ＨＬレジスタと交換することができます。 なお，ＥＸＸ命令では，ＢＣとＢＣ'，ＤＥとＤＥ'，ＨＬとＨＬ'が，全部一度に交換されてしまいます。 これら，ＡＦ'，ＢＣ'，ＤＥ'，ＨＬ'の各レジスタは，ｉ8080にはなく，Ｚ80で拡張されたものということもあって，全く，全く使用していないソフトウエアも珍らしくありません。よって，これらは最初から使いこなそうとする必要はありません。 しかし，これら裏レジスタを利用すると，メモリの節約，スピードアップ等を図ることができます。
I	このレジスタは，ＩＭ　2命令によって，強力なＺ80特有のモード2割込みを行なう場合のみ，使用します。スーパーＭＺはモード2割込みを利用してはいるものの，ＩＰＬ起動プログラム以外では，特にこのレジスタについて知っている必要はありません。ＬＤ　Ｉ，Ａ命令を使って，このレジスタの内容を変化させたりしなければ良いだけです。 モード2割込みでは，割込みの発生源は最大128個持つことができますが，それらの処理アドレスは2バイトずつ，一般の命令における2バイトデータ同様，あるアドレスに下位バイトを，＋1のアドレスに上位バイトの順に置くテーブルにして，まとめて，最下位ビット＝0の偶数アドレスからに配置します。そして，テーブルの上位バイト（上位8ビット）を，このＩレジスタに入れておきます。 インタラプト発生源には，下位8ビット（最下位ビットは0に決まっているから，実質的には第7ビット～第1ビットの7ビットのみ有効）を通知します。インタラプト発生後，ＣＰＵがそれを受付けた場合には，ＣＰＵは発生源のすべてに受付けたことを知らせます。すると，その中から，実際にインタラプトを発生させたデバイスが，あらかじめ通知されていた下位8ビットの値をＣＰＵに返答します。一方，ＣＰＵは，そのデータをもとに，Ｉレジスタにある上位8ビットと合わせ，テーブルのアドレスを計算します。すると，そこには実際の処理ルーチンのアドレスが書かれていますので，ここでようやく割込み処理ができるようになります（同時に2個以上のデバイスからインタラプトが発生しても，ハードウエア上の割込みコントローラが優先順位を決めるので，プログラマはこの点について考慮する必要はない）。 なお，上位アドレスはＩレジスタに入っていることから，テーブルは，異なる2ページ（ｘｘ００Ｈ～ｘｘＦＦＨのアドレスを，1ページと呼びます。Ｚ80のアドレス空間は256ページということになります）にまたがって置いてはなりません。

R	プログラムではあまり利用することのないレジスタです。 Z80はD－RAM（ダイナミックRAM）との接続が容易だと言いましたが，このレジスタはそのためのものです。D－RAMというのは，S－RAM（スタティックRAM）と違って，コンデンサへの充電によって0か1かを記憶します。しかし，コンデンサの容量は非常に小さく，リフレッシュという作業をしませんと，記憶が失なわれてしまいます。リフレッシュのための回路のすべてはD－RAM内部に持っていますが，そのタイミングをうまく取らないと，CPUからのアクセス（読出し・書込み）と重なってしまい，支障をきたしますので，CPUがD－RAMに対して，リフレッシュのタイミングを知らせなければなりません。リフレッシュは，2ms（ミリセカンド）のうちにアドレスの下位7ビット（b6～b0）をひと通り選択するだけで良く，Z80は命令バイトの読み出し直後，つまりM1サイクルの最後（通常は1回。拡張命令では2バイト続くから計2回）に，これを行う機能を持っていますがこのための7ビットのアドレスをCPU側に記憶しておく必要があります。「ひと通り」選択するには，前回の状態をもとに，次のアドレスを，というように，もれなく行う必要があるからです。そして，そのための記憶場所が，このRレジスタです。Rレジスタは8ビット長ですが，実際に使われているのは，b7を除いた7ビットです。よって，このレジスタは，ソフトウエア上で利用する機会は，まずありません。特に，やたらなデータを連続して書込むと，リフレッシュが正常に行うことができなくなることがあり，RAM上の記憶内容（プログラムがRAM上にあるならそれ自身も）が破壊されてしまい，プログラムが暴走してしまったり，大切なデータが失なわれてしまうことが，あり得るからです。よって，このレジスタへの書込み命令であるLD　R，A命令は使用しないのが無難です（だいたい，それを行う必要などない）。 ただし，Rレジスタの内容を参照することについては，全く問題ありません。それればかりか，Rレジスタは，まず予測できない値をとります（実際は＋1ずつしていくのだが，完全に周期的に読出さぬ限り，例えばキーボードからの待ちループを少しでも行うなどによって，次にアクセスする際には規則的でない値が得られる）ので，ランダムな値，すなわち乱数としてゲーム等に応用することが可能となります。
IX	i8080でのHLレジスタの，つまり，Hレジスタ，Lレジスタとしてではなく，ペアレジスタとしての働きのほとんどすべて――EX　DE，HLだけは除く――を備えており，しかもインデックスアドレッシングを可能にしているレジスタです。ただし，Z80で拡張されたHLレジスタの働きはあまり持っていません。インデックスアドレッシングというのは，(HL）による擬似レジスタを，前後の－128～＋127のアドレスに拡張できるというもので，(IX＋70H)

	や，（ＩＸ—20Ｈ）のように指定することができます。 なお，このレジスタは，裏レジスタ同様，全く利用していないプログラムもあり，取りあえずは覚える必要はないと言えるでしょう。
ＩＹ	このレジスタは，完全にＩＸレジスタと同じことができます（Ｚ80には，このような関係のレジスタは，ほかにはありません）。 よって，利用方法等も，完全に同じです。
ＳＰ	単独で動作させるようなプログラムまたは，とても大規模なプログラムを除き，普段はあまり意識する必要のないレジスタです。 サブルーチンを呼び出す際には，行った先から戻るための情報，すなわち，帰って来るべきアドレスを保存しておかねばなりませんが，そのためのメモリをスタックと言います。スタックはまた，ＰＵＳＨ命令によるデータの退避用にも利用します。 スタックというのは先入れ後出しのバッファになっていて，プログラマは，メモリとして意識せずに利用することができる空間です。このためのエリアは特にスタックエリアと言い，プログラムやワークエリアとの境界さえ守れば，自由に使って構いません。逆に，プログラムの先頭では，原則としてこのようにできるよう，設定する必要があります。しかし，ＢＡＳＩＣのサブルーチンとしてマシン語プログラムを使う程度であれば，ＢＡＳＩＣが設定したスタックがそのまま流用できることが多いので，特に意識しなくとも良いというわけです。 スタックエリアは，プログラムの種類によっても異なりますが，おおよそ256バイト（16進数で0100Ｈバイト）もあれば充分です。ただし，スタックはワークエリアと同じように，必ずＲＡＭ上になくてはなりません。 また，スタックをメモリのどこに配置するかは自由ですが，（スタックは）高位アドレスから消費（使用）しますので，一般にはＲＡＭの後ろや，スタックを利用するプログラムの直後に置くのが普通です。 スタックは，国語辞典などのあき箱のようなものを入口が見えるようにして横に倒し，それを縦に何段も重ねたようなものだと，考えてよいでしょう。そして，ＣＰＵは下（メモリの後）から，順序よく使っていくのです。初めは，最も下にある箱の下，つまり，それらを置いてある机などの表面を指さしています。 例えば，サブルーチンに行く前には，指を少し上に持って行き，そのケースの中に，次のアドレス（戻るべきアドレス）を紙に書いたものを入れます。 サブルーチンから帰る時には，指でポイントしているケースから紙を取り出して，指は一つ下に戻します。後は，書いてあったアドレスにリターンするだけ

	です。データを退避させる時も，これと同じ手法がとられます。 なぜ，箱を何段も重ねるかというと，行った先でさらにサブルーチンを呼んだり，データを退避させたりする時，また同じ箱を使ってしまっては困るからです。ポイントする理由は，どこまで使っているかを明確に示すためです。そして，ＣＰＵで，この指に相当するのが，ＳＰレジスタ（スタックポインタと呼ぶことのほうが多い）なのです。 ただし，メモリは8ビット，アドレスは16ビットですから，一つのメモリには，アドレスは入りません。よって，2バイトのメモリに分けて入れます。ですから，このレジスタは，一度に2ずつ，＋，－します。
ＰＣ	このレジスタは，プログラムのどこを実行中であるかを指すレジスタです。ＣＰＵがリセットされた時には，0000Ｈになり，また，ＪＰ命令では，このレジスタに指定のデータ（アドレス）が入ります。ＣＡＬＬ命令でも，スタックに次のアドレス（ＰＣ＋3）を格納した後には，ＰＣレジスタ（プログラムカウンタと呼ぶことのほうが多い）に指定のアドレスが入ります。ＲＥＴ命令は，スタックからプログラムカウンタに対するデータの復帰です（ＰＯＰ　ＰＣとでも言うべき働きをする）。



単独で動くプログラムを組んでみる

何はともあれ，簡単なプログラムを作ってみましょう。

ＢＡＳＩＣから，

ＣＬＥＡＲ　＆ＨＥ０００↵

ＭＯＮ↵

でモニタに入って下さい。Ｅ０００Ｈ～ＦＦＦＦＨをマシン語プログラム用に確保してから，モニタに入るわけです。もし，今モニタの状態でしたら，一度

＊Ｒ↵

でＢＡＳＩＣに戻って，これを実行して下さい。

いつものように，

＊■

と，モニタのコマンド待ちになると思いますが，ここで，

＊ＤＥ０００↵

と入力します。すると，また，いつものように，

：Ｅ０００＝００　００　００　００　００　００　００　００　／・・・・・・・・

：Ｅ００８＝００　００　００　００　００　００　００　００　／・・・・・・・・

```
:E010=00  00  00  00  00  00  00  00  /........
:E018=00  00  00  00  00  00  00  00  /........
                                      ⋮
                                      ⋮
                                      ⋮
```

と出て来ると思います。このように各アドレス００Ｈという値が入っていることが多いと思いますが，別にそうでなくとも構いません。

では，ＢＡＳＩＣのスクリーンエディットの要領で，「００」等の部分を書換えて下さい。つまり，次のプログラムを入力するわけです（これ以後，／以降のキャラクタダンプ部は省略させて頂きます)。一行分，例えばアドレスＥ０００Ｈ～Ｅ００７Ｈの部分が入力し終わったら，ＢＡＳＩＣ同様，↵キーを忘れずに押して下さい。

```
:E000=3A  11  E0  47  3A  10  E0  80
:E008=32  12  E0  C9  00  00  00  00
:E010=14  07  00  00  00  00  00  00
                                      ⋮
                                      ⋮
                                      ⋮
```

さて，1分もあれば，入力し終わるでしょう。念のため，

＊ＤＥ０００↵

として，正しく入力されているかどうかを確認して下さい。マシン語はちょっとしたミスがあっても，エラーで知らせて来ないで，すぐ暴走したりします。電源を入れ直すかＩＰＬスイッチを押すまで，うんともすんとも言わなくなったりしますので，注意して下さい。ただ，ハードウエア（ＭＺ－２５００）を壊すことはありませんので，ＢＡＳＩＣでどんどんエラーを出したように，マシン語でもどんどん暴走させて，大いに勉強して下さい（もちろん正しく動くに越したことはありませんが)。

では，動かしてみましょう。

＊ＧＥ０００↵

として下さい。

すぐに，特に何も起きずに，

＊■

と出て来るでしょう。そして，

＊ＤＥ０００↵

として下さい。

```
:E000=3A  11  E0  47  3A  10  E0  90
:E008=32  12  E0  C9  00  00  00  00
:E010=14  07  1B  00  00  00  00  00
                                      ⋮
```

⋮
⋮

のように出て来るでしょう。アドレスＥ０１２Ｈが１ＢＨに変わっているのがわかります。
では次に，Ｅ００７Ｈの８０Ｈを９０Ｈに変えて実行してみましょう。そして，同じようにしてメモリを見てみましょう。

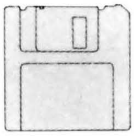
：Ｅ０００＝３Ａ　１１　Ｅ０　４７　３Ａ　１０　Ｅ０　９０
：Ｅ００８＝３２　１２　Ｅ０　Ｃ９　００　００　００　００
：Ｅ０１０＝１４　０７　０Ｄ　００　００　００　００　００

⋮
⋮
⋮

のようになっているでしょう。つまり，Ｅ０１２Ｈが０ＤＨとなったわけです。

同じ要領でＥ００７ＨをＡ０ＨやＢ０ＨやＡ８Ｈに変えてみて下さい。結果は０４Ｈ，１７Ｈ，１３Ｈになると思います。

また，Ｅ０１０Ｈの１４Ｈや，Ｅ０１１Ｈの０７Ｈも，いろいろ変えてみて下さい。どちらも００Ｈ～ＦＦＨのどんな値でも結構です。そしてこれらの値と結果との関係をいろいろ研究してみて下さい……。



マシン語の意味，解明！

さて，そろそろタネ明かしをしましょう。最初のプログラムを見ながら，説明してゆきましょう。

まず，E000Hからの3バイト，つまり，**３Ａ　１１　Ｅ０**（いずれも16進数ですが，ここではそれが明白なので，Hは省略します）の部分は，**ＣＰＵ内の「Ａレジスタ」に，アドレスＥ０１１Ｈのメモリに格納されている値を入れろ**（ＢＡＳＩＣ風に言うと代入せよ）という意味です。E011H番地には07Hという値が入っていますので，Aレジスタにも07Hが入ります。なお，ＢＡＳＩＣ同様，転送元の内容は変わりません。つまり，E011Hの内容（07H）はそのままであるということです（以下同様）。

次のアドレスE003Hからの1バイト，つまり　**４７**　の部分は，**「Ｂレジスタ」に，「Ａレジスタ」の値を入れろ**という意味です。さきほどの命令と合わせて考えると，**Ｅ０１１Ｈのメモリの内容を，Ｂレジスタに入れる**ということになります。これは，あるメモリの内容を直接Ｂレジスタに入れる命令がないので，一度Ａレジスタを仲介させているのです。

次の3バイト，つまり　**３Ａ　１０　Ｅ０**　の部分は，**「Ａレジスタ」に，アドレスＥ０１０Ｈのメモリに格納されている値を入れろ**，という意味です。E010の値は14Hですので，Aレジスタには14Hが入ることになります。

次の1バイト，つまり　**８０**　の部分は，**「Ａレジスタ」の値と「Ｂレジスタ」の値とを加算（＋）し，結果を「Ａレジスタ」に入れろ**，という意味です。ＢＡＳＩＣ風に言うと，Ａ＝Ａ＋

Bという感じです。Aレジスタには直前の命令で14Hが入っており，Bレジスタにはその前の命令で07Hが入っていますので，結果は1BHになります。つまり，Aレジスタには1BHという値が入ることになります。

次の3バイト，つまり　３２　１２　Ｅ０　の部分は，**「Aレジスタ」の値をアドレスＥ０１２Hのメモリに入れろ**，という意味です。

次の1バイト，つまり　Ｃ９　の部分は，**サブルーチンから帰る**,という意味です。ＢＡＳＩＣのＲＥＴＵＲＮと同じわけですが，ここでは，**モニタに帰る**，という働きをします。モニタのGコマンドは，（ユーザーの作った）マシン語プログラムを，サブルーチンとして呼び出す，という働きなので，このようにサブルーチンからリターンする命令を使うのです。

さて，その後の4バイト，つまりE00CH～E00FHは00Hが続きますが，これは特に意味はありません（未使用です）。便宜的に開けています。

そして，　E010からは，データエリアです。プログラムではありません。

いかがですか？　簡単に言うと，　**Ｅ０１０Hに入っている値とＥ０１１Hに入っている値とを足し算し，結果（答）をＥ０１２Hに入れる**，というプログラムだったわけです。マシン語もそれほど難しいものではないでしょう。

また，E007Hを，80H以外の「90H」や「Ａ０Hや「Ｂ０Hや「Ａ８H」に変えた場合の意味は，

９０H　…　「Aレジスタ」の値から「Bレジスタ」の値を減算（－）し，結果を「Aレジスタ」に入れる。ＢＡＳＩＣ風に表現すると　Ａ＝Ａ－Ｂ。

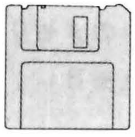
Ａ０H　…　「Aレジスタ」の値と「Bレジスタ」の値の論理積を求め，結果を「Aレジスタ」に入れる。ＢＡＳＩＣ風に表現すると　Ａ＝Ａ　ＡＮＤ　Ｂ。

Ｂ０H　…　「Aレジスタ」の値と「Bレジスタ」の値の論理和を求め，結果を「Aレジスタ」に入れる。ＢＡＳＩＣ風に表現すると　Ａ＝Ａ　ＯＲ　Ｂ。

Ａ８H　…　「Aレジスタ」の値と「Bレジスタ」の値の排他的論理和を求め，結果を「Aレジスタ」に入れる。ＢＡＳＩＣ風に表現すると　Ａ＝Ａ　ＸＯＲ　Ｂ。

になります。

では，意味の分かった所で，もう一度，プログラムを実行してみて下さい。特に，「Ａ０H」と「70H」のような値を加算したらどうなるか，とか，「12H」から「34H」を引いたらどうなるか，と言った実験も行ってみて下さい。



マシン語とアセンブリ言語の不思議な関係？

さて，今，見て来たプログラムには，いくつかの命令が使われていました。

しかし，いずれも16進数であり，例えば「80H」という値がどんな働きをするのか，すぐに連想することはできないでしょう（暗記大好き！という人は除きますが）。

そこで，このマシン語を少しは人間的なものに１対１で置換えた，**「アセンブリ言語」**が考案されました。つまり，プログラマは通常「アセンブリ言語」でプログラムを組み，それを「マシ

ン語に変換するプログラム」にかけて，目的の（マシン語）プログラムを得る，というわけです。これはZ80に限らず，他のマイクロプロセッサや大型コンピュータのＣＰＵでも極めて一般的なことで，昔から使われている方法でもあります。

このマシン語に変換するプログラム」のことを，普通，**「アセンブラ」**と言います。逆に考えると，「アセンブラ」で使われる言語だから「アセンブリ言語」と呼ばれるのでしょう。

さきほどのマシン語プログラムを，アセンブリ言語で見てみると，次のようになります。

```
        ＬＤ      Ａ，（Ｅ０１１Ｈ）
        ＬＤ      Ｂ，Ａ
        ＬＤ      Ａ，（Ｅ０１０Ｈ）
        ＡＤＤ    Ａ，Ｂ
        ＬＤ      （Ｅ０１２Ｈ），Ａ
        ＲＥＴ
```

つまり

```
３Ａ  １１  Ｅ０  は  ＬＤ    Ａ，（Ｅ０１１Ｈ）  を
４７            は  ＬＤ    Ｂ，Ａ              を
３Ａ  １０  Ｅ０  は  ＬＤ    Ａ，（Ｅ０１０Ｈ）  を
８０            は  ＡＤＤ   Ａ，Ｂ              を
３２  １２  Ｅ０  は  ＬＤ    （Ｅ０１２Ｈ），Ａ  を
Ｃ９            は  ＲＥＴ                      を
```

を表わしているのです。なお，

ＬＤとは，ＬｏａＤという意味です。英和辞典には「荷を積む」と書いてありますが，何となく雰囲気は分かるでしょう（プログラムのロードと同じ語源です）。

ＡＤＤは加算。ずばりそのものです。

ＲＥＴはＲＥＴｕｒｎという意味で，これまたずばりそのものでしょう。

いかがですか？　分かり易いですか？　少なくとも，16進数よりは親しみ易いでしょう。

ところで，「アセンブラ」というのは，いったいどこにあるのでしょうか？もちろん，スーパーＭＺには付属していません。

実は，「Ｐ－ＣＰ／Ｍ」を購入すると，市販の（ＣＰ／Ｍ用に開発された）Ｚ80アセンブラが利用できるようになるのです。ただ，他のパソコンの例でみると，これには，ＣＰ／Ｍ＋アセンブラで，熱転写24ドット漢字プリンタが買えるくらいの出費となります。

また，別の方法としては，ＭＺ－2000／2200用またはＭＺ－80Ｂ用に発売されている，単独で動く（ＣＰ／Ｍがいらない）アセンブラ等を利用する，という手もあります。もちろん，それらのパソコンのモードで動かすわけですが，マシン語は同じですから，Ｚ80の各々の命令の働きを学習するには充分です。これですと，数千円からせいぜい２万円くらいで購入できます。特にこの２万円のもの（シャープ「システムプログラム」ＭＺ－１Ｚ００５），40Ｋバイト以上のマシン語まで作れる性能の良いアセンブラを含む，総合的なマシン語開発ツールです。ただ，それらのパソコンではフロッピーディスクがあまり普及していなかったので，その価格のものはみなテープ版です。ディスク版にはシャープの「Ｆ－ＤＯＳ・ＭＺ－２Ｚ００４」（５万円）があり

ますが，5インチ版ですから，旧MZのディスクを持っていないと使えません。

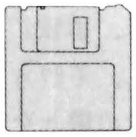
そして，方法はもう一つあります。出費0円でOKという，まことに有り難い（？）話があるのです。それはそれは，あなた自身が，アセンブラになってしまうのです。**付録IIのマシン語活用表**を見て下さい。ここには，Z80の全部の命令が入っています。これを見ながら，あなたがアセンブリ言語→マシン語に変換し，入力すれば良いのです。数10バイトからせいぜい256バイト（0100H）程度のプログラムは，この方法で充分ですし，大変勉強になります。そこでほとんどの人は，初めはこの方法をとるようです。そして，それ以上の大きなプログラムを作りたい時に，または作れるようになった時に，市販のアセンブラを購入すれば良いでしょう。そのころには，P－CP／M用にも安価なアセンブラが発売になっているかも知れません。

アセンブラはアセンブリ言語で書いた？

最初の「アセンブラ」はいったい何で書いたのでしょうか？「アセンブラ」がない以上，「アセンブリ言語」で書いても，それを（アセンブラで）マシン語に変換することはできません。

しかし，それでも「アセンブリ言語」で書くのだと思います。「アセンブリ言語」の仕様を紙上で決定した後で，それに基づいた「アセンブラ」を作成するわけですから，その時，わざわざ分かりづらいマシン語で直接書いたりはせず，たとえ紙上でも，アセブリ言語で書き，それをハンドアセンブルによってマシン語に置換えていくはずです。

ただ，すべてのCPUがこのような方法をとるのではなく，「新CPU用のアセンブリ言語を，新CPUのマシン語に変換するプログラム」を，既に動いている別のCPUのコンピュータ上で，作成して，それを使用することも多くいります（というより，現在ではこれが普通）。このプログラムは単なる「アセンブラ」とは言わず，**「クロスアセンブラ」**と言います。また，新CPUが相当普及した後でも，クロスアセンブラが使われることはあり，例えばCP／M（P－CP／M）用のソフトには，i8086や6809と言ったCPU用のクロスアセンブラが販売されていますし，ミニコン（数千万円クラスの汎用コンピュータで，大型コンピュータよりは小規模）用にはZ80等のクロスアセンブラが供給されているのが普通です。



ハンドアセンブルにチャレンジ！

では，ハンドアセンブル（人間がアセンブラの代りをすることを，普通こう言います）を行ったり，**付録IIのマシン語活用表**を見る上での注意事項と，いくつか参考になりそうなことを申し上げておきましょう。

まず，例えば　LD　A，（E011H）という命令は，表中では　LD　A，（nnmm）としてあり，マシン語の所では　3A　mm　nn　と書いてあります。つまり，LD　A，

（E011H）なら　３Ａ　１１　Ｅ０　となるわけですが，この時，下位バイトの「11H」が先に来て，上位バイトの「Ｅ０Ｈ」が後に来ますので，充分注意して下さい。また，これ以外のあらゆる２バイト（16ビット）のデータを扱う命令と，メモリ上に置く２バイトのデータも，すべて，同様となります。この法則は，ｉ8080やZ80，ｉ8086等の「80系」ＣＰＵと，6502（第一章で触れたＡＰＰＬＥ　ⅡやＰＥＴ－2001のＣＰＵ。任天堂のファミリーコンピュータもこれの上位コンパチＣＰＵを使っている）等の一部のＣＰＵに共通するもので，こうすると実行速度の面で有利なのだそうです。つまり，下位バイトが下位アドレスに，上位バイトが上位アドレスに入る，という方が，ハードウエア的に容易なのでしょう。

次に，アセンブリ言語の（　　）で囲った部分は，（　）内に書いたレジスタか，（　）内に直接書いた値が指すメモリを意味します。ＢＡＳＩＣのように，式の計算順序を変える働きをするのではありません。ＬＤ　Ａ，（E011H）　ではE011H番地のメモリの内容をＡレジスタに入れますが，ＬＤ　Ａ，20HではＡレジスタには 20Hそのものが入ります（Ａレジスタは８ビットなので，ＬＤ　Ａ，1234Hのような命令はない）。また，ＬＤ　ＨＬ，2000Hという命令（マシン語では　21　00　20）はＨＬ（ペア）レジスタに2000Hが入る，もう少し詳しく言うと，Ｈレジスタに20Hが，Ｌレジスタに00Hが入るわけですが，その後で，ＬＤ　Ａ，（ＨＬ）という命令を実行すると，ＨＬレジスタの指すメモリ，つまり，アドレス2000Hのメモリに入っている値が，HLレジスタに入ります。また，逆にＬＤ　（ＨＬ），Ａを実行すると，Ａレジスタの値が，ＨＬレジスタが指すメモリに入ります。つまり，ＨＬレジスタに2000Hが入っているなら，ＬＤ（2000H），Ａと同じ働きになるわけです。ちょっとややこしい気もしますが，慣れれば何でもなくなります。なお，「働き」の欄でも，この法則に従って書いています。

次に，働きは似ているのに，

ＡＤＤ　Ａ，Ｂ　　ＡＤＣ　Ａ，Ｂ　　ＳＢＣ　Ａ，Ｂ

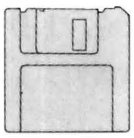
のように，「Ａ，」を付ける命令と，

ＡＮＤ　Ｂ　　ＯＲ　Ｂ　　ＸＯＲ　Ｂ　　ＳＵＢ　Ｂ

のように，「Ａ，」を付けない命令とがある点です。これらはいずれもＡレジスタとＢレジスタまたは他のレジスタまたは直接指定した値またはＨＬやＩＸ・ＩＹで指したメモリとの間で何か演算を行い，結果をＡレジスタに入れる命令ですが，「Ａ，」が付くのと付かないのとがあるでしょう。これは，HLレジスタを使って16ビットの計算を行う命令と混乱しそうな命令では「Ａ，」を付け，その心配がない命令では付けていないのです。つまり，ＡＤＤ　ＨＬ，ＢＣ（ＨＬ＝ＨＬ＋ＢＣのような働き）はあってもＡＮＤ　ＨＬ，ＢＣのような命令はないので，こうしているわけです。

最後に，アセンブリ言語（マシン語）には，常套手段というものがあります。これは，ある程度プログラムを組めるようになった時に参考にして下さい。例えば，Z80の場合，Ａレジスタの内容が０であるかどうかを調べるには，比較を行う**「ＣＰ」命令**を使う（ＣＰ　００Ｈとする）ことが考えられますが，実際には「ＡＮＤ　Ａ」または「ＯＲ　Ａ」を使うのが普通です。これは，ＣＰ　00Hが２バイトの命令（ＦＥ　00となる）なのに対し，ＡＮＤ　Ａ（Ａ７）やＯＲ　Ａ（Ｂ７）は１バイトで済むからです。特にマシン語（アセンブリ言語）のプログラムは，速く動いてメモリ消費量が小さいものほど優れているので，本来意味の違う命令でも，同じような働きをするなら使うことが多いのです。これ以外にも，

ＬＤ　Ａ，Ｂ
ＯＲ　Ｃ
（ＢＣレジスタが００００Ｈであるかどうか調べる。ＤＥ・ＨＬでも同様に可）
ＸＯＲ　Ａ　または　ＳＵＢ　Ａ
（Ａレジスタに０を入れる。ＬＤ　Ａ，００Ｈの代りに使うことが多い）
ＡＮＤ　Ａ　または　ＯＲ　Ａ
（Ｃｙフラグをリセットさせる）
ＩＮＣ　Ａ
ＤＥＣ　Ａ
（Ｃｙフラグを変化させずにＡレジスタが０かどうかを調べる（結果はＺフラグ））など，いろいろあります。ここに挙げたのは，２命令以下の，比較的良く使われるものですが，他にも数多くのテクニックがあります。他人の，特にプロの作ったプログラムを解析してみると，そういったことが良く分かりますし，命令一つ一つの働きも，実感として理解できるようになります。また，思ったことが，プログラムですらすら書けるようになってゆきます。大変良い勉強になりますので，例え１Ｋバイト程度でも，身近にあるマシン語プログラム（例えばＢＡＳＩＣインタプリタ）を解析してみると良いでしょう。



ディスアセンブラ

さきほど，「アセンブラ」については「アセンブリ言語」を「マシン語」に変換するもの，つまり，

ＬＤ　　Ａ，（Ｅ０１１Ｈ）　などを，　３Ａ　１１　Ｅ０

に変換するものと言って来ましたが，**「ディスアセンブラ」または「逆アセンブラ」**というのは，その反対に，

３Ａ　１１　Ｅ０　などを，　ＬＤ　Ａ，（Ｅ０１１Ｈ）　に変換するものです。

例えば，他人の作ったマシン語プログラム（16進数のら列）を解析する際，ハンドディスアセンブル（？）で，つまり手作業でアセンブリ言語に変換しても良いのですが，１Ｋバイトのプログラムなら数100個もの命令になってしまい，それだけで大変な作業になります。まして，ＢＡＳＩＣ　—M25やＩＰＬ・ＢＩＯＳ—ＲＯＭ，そしてスーパーヘルプ機能まで全部解析しようとすると，140Ｋバイトにも上るため，命令数（ステップ数）は10万近くになってしまいます。

そこで，マシン語（アセンブリ言語）入門者の皆さんには「アセンブラ」より意義のある，「ディスアセンブラ」をお届けしましょう**リスト6－1**を見て下さい。これがあると，ハンドアセンブルが正しくできたかどうかを確認することもできます。

このディスアセンブラは，ＢＡＳＩＣで書いており，約10Ｋバイトの大きさです。ＢＡＳＩＣ—M25で使用します。このプログラムを使用しなくとも，ＲＡＭ未拡張の状態（128Ｋ）で，ＣＬＥＡＲ　＆Ｈ8000のように多量のマシン語エリアを確保しようとすると，メモリが足りなくなってしまいますが，ＮＥＷ　ＯＮ　４（プリンタを使用する時）・ＮＥＷ　ＯＮ　５（プリンタ

を使用しない時）でＢＡＳＩＣの機能を一部カットしたりして，フリーエリアを広げて下さい。その上でＣＬＥＡＲを実行したり，このプログラムを実行したりして下さい。ただし，ＣＬＥＡＲ　＆ＨＥ000程度では，ＢＡＳＩＣ起動時に auto-run. m25中で「ＮＥＷ　ＯＮ　2」が実行されていますから，全く問題ありません。

さて，では，**リスト6-1**を打ち込んで下さい。ところどころ，漢字や全角文字（ひらがな）が使われていますが，入力が面倒な時は，アルファベットや半角のカタカナに変えてしまっても結構です。使うのは皆さんですから，皆さんの使い易いようにして良いのです。また，ＰＯＫＥ文とＰＥＥＫ関数の所は，入力ミスのないように気を付けて下さい。マシン語のプログラムを間違えたのと同じように，暴走の危険があります。

入力が終わったら，一応ディスクにセーブしておいて下さい。ファイルネームは「ディスアセンブラ」でも「Ｄｉｓａｓｓｅｍｂｌｅｒ」でも結構ですから，適当に考えて下さい。

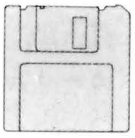
《リスト6-1》Z80ディスアセンブラ・プログラムリスト

```
100 '*********************************
110 '*                               *
120 '* Z-80 ﾃﾞｨｽｱｾﾝﾌﾞﾗ for MZ-2500 *
130 '*                               *
140 '*Copyright(C) 1985 by 高橋雄一 *
150 '*                               *
160 '*********************************
170 'CLEAR &HE000
180 init "CRT1:80,25,1,0"
190 click off
200 color 5:print "Z-80 Disassembler for MZ-2500"
210 color 7:klist 0
220 PAGE=0:DEV=0
230 def FNW$(A)=right$("0"+hex$(peek(A+1)),2)+right$("0"+hex$(peek(A)),2)+"H"
240 def FNH$(A)=right$("0"+hex$(peek(A)),2)+"H"
250 def FNH4$(A)=right$("000"+hex$(A),4)
260 data A,BC,DE,HL,SP,B,C,D,E,H,L,(HL),A,NZ,Z,NC,C,PO,PE,P,M
270 dim RP$(4),R8$(7),JK$(7)
280 for I=0 to 4:read RP$(I):next
290 for I=0 to 7:read R8$(I):next
300 for I=0 to 7:read JK$(I):next
310 gosub 560 '1' Command
320 INK$="":if csrlin>12 then print
330 gosub 2460 'Command list
340 key 0,"":A$="":INK=0
350 MSG$="Command="
360 gosub 2620:A$=left$(A$,1):if A$="!" then run
370 if A$="E" or A$="e" then 2740 else if A$="M" or A$="m" then 840
380 if A$="D" or A$="d" or A$="" then SK=1:DEV=0:goto 910
390 if A$="P" or A$="p" else 540
400 SK=2:DEV=1:if PAGE then 440
410 MSG$="ﾌﾟﾘﾝﾀの準備は良いですか？[Y/N] ":A$="":gosub 2620
420 if A$="Y" or A$="y" then 440 else 330
430 beep 'TITLE length Error
440 TIT=len(TITLE$)-2:if TIT+2 else 460
450 TITLE$=mid$(TITLE$,2,TIT):TITLE$=TITLE$+string$(TIT,29)
460 MSG$="タイトルは？(15字まで。不要ならそのままリターンを押して下さい)"
470 gosub 2670
480 key 0,TITLE$:input "",TITLE$ '(ﾒﾓﾘが余っているならLINE INPUTの方が良い)
490 gosub 2640
500 if len(TITLE$)>15 then TITLE$=left$(TITLE$,15):goto 430
510 if len(TITLE$) then TITLE$="<"+TITLE$+">"
520 if GYOU=0 then GYOU=65:gosub 2470
530 goto 910
540 on val(A$)+1 gosub 550,560,580,630,650,700,720,760,780,800:goto 340
550 beep:return 'Command error
560 '1' = 全画面
570 console 1,23,0,80:locate 1,23:ADD=22:goto 830
580 '2' = 上半分
```

```
 590 console 1,11,0,80:locate 1,11
 600 ADD=10:gosub 830
 610 for C=0 to 79:poke@ &H38,&H3C0+C,&H54:poke@ &H38,&HBC0+C,1
 620 poke@ &H38,&H13C0+C,&H8E:next:return
 630 '3' = 下半分
 640 console 13,11,0,80:locate 1,23:goto 600
 650 '4' = 左半分
 660 console 1,23,0,39:locate 1,23
 670 ADD=22:gosub 820
 680 for C=1 to 23:poke@ &H38,&H27+80*C,&H58:poke@ &H38,&H827+80*C,1
 690 poke@ &H38,&H1027+C*80,&H8E:next:return
 700 '5' = 右半分
 710 console 1,23,40,39:locate 41,23:goto 670
 720 '6' = 左上
 730 console 1,11,0,39:locate 1,11
 740 gosub 610:gosub 680:poke@ &H38,&H3E7,&H3C
 750 ADD=10:goto 820
 760 '7' = 右上
 770 console 1,11,40,39:locate 41,11:goto 740
 780 '8' = 左下
 790 console 13,11,0,39:locate 1,23:goto 740
 800 '9' = 右下
 810 console 13,11,40,39:locate 41,23:goto 740
 820 T15=15:T21=21:T34=35:WID84=38:COME=60:return '40桁表示の時
 830 T15=28:T21=35:T34=74:WID84=78:COME=55:return '80桁表示の時
 840 MSG$="ＣＰＵのブロック番号は？ (0～7) ":A$="":gosub 2620
 850 if A$="" or A$="*" then 340
 860 BLK=val(A$):if BLK<0 or BLK>7 then beep:goto 840
 870 MSG$="メモリのブロック番号は？ (00～3F) &H":A$="":gosub 2620
 880 if A$="" or A$="*" then 340
 890 BL=val("&H"+A$):if BL and &HFFC0 then beep:goto 840
 900 poke@ 0,&H580+BLK,BL:goto 340
 910 MSG$="START adrs.=&H":A$="":gosub 2620
 920 if A$="-" then 940 else if A$="*" then 330
 930 S=val("&H"+A$):if S<0 then S=S+65536
 940 if right$(A$,1)="-" then E=S+ADD:goto 980
 950 MSG$="END   adrs.=&H":MSG1$="":MSG2$="":A$="":gosub 2620
 960 if A$="" then A$="FFFF" else if A$="*" then 330
 970 E=val("&H"+A$)
 980 if E<0 then E=E+65536 else if E>65535 then E=65535
 990 OFSET=0:OP$=""
1000 '
1010 '
1020 gosub 2550
1030 A=S:gosub 1300
1040 CH$=""
1050 if OP$<>" " then 1070 else if asc(INK$)=27 then 320 else gosub 2470
1060 goto 1030
1070 AD$=FNH4$(S+OFSET)+" "
1080 C=peek(S):AD$=AD$+right$("0"+hex$(C),2)
1090 if &H80<C and C<&HDF then C=C and &H7F
1100 if C<32 then C=32
1110 CH$=CH$+chr$(C):S=S+1:if S<>A then 1080
1120 if DEV then 1180 else PRT$=string$(WID84,32):mid$(PRT$,1)=AD$
1130 mid$(PRT$,T15)=OP$:mid$(PRT$,T21)=OL$:mid$(PRT$,T34)=CH$
1140 if COMMENT$<>"" then mid$(PRT$,COME)=";"+COMMENT$
1150 print
1160 print PRT$;
1170 goto 1210
1180 PRT$=string$(79,32):mid$(PRT$,1)=AD$:mid$(PRT$,25)=OP$:mid$(PRT$,32)=OL$
1190 mid$(PRT$,75)=CH$:if COMMENT$<>"" then mid$(PRT$,60)=";"+COMMENT$
1200 lprint PRT$
1210 gosub 2480
1220 if asc(INK$)=27 then 320
1230 if inkey$=chr$(&H1B) or S>E then 320 else 1030
1240 DH=DT¥&O100
1250 DM=(DT and &O70)¥&O10
1260 DL=DT and &O7
1270 D4=DM and 1
1280 DV=DM¥2
1290 A=A+1:return
1300 if SP then SP=SP-1:OP$=" ":OL$="":return 'RET,JP,JR,HALT
```

```
1310 if OP$="RST" then if OL$="18H" or OL$="28H" then goto 2100 '不要なら取る
1320 DF=0 'not DD,FD
1330 OP$="LD":OL$="":COMMENT$="" 'DD,FD Kenyou
1340 DT=peek(A):gosub 1240
1350 if (DT and &HCF)=&HCD then on (DT and &H30)¥&H10 goto 2350,2060,2360
1360 on DH goto 1630,1660,1720
1370 rem 00H-3FH
1380 on DL goto 1470,1490,1530,1550,1560,1580,1610
1390 if DM=0 then OP$="NOP":return
1400 if DM=1 then OP$="EX":OL$="AF,AF'":return
1410 '00,08 other
1420 OP$="JR":if DM=2 then OP$="DJNZ":goto 1450
1430 if DM=3 then SP=SK else OL$=mid$("ZZCC",DM-3,1)+","
1440 if D4 else OL$="N"+OL$
1450 DT=peek(A):A=A+1:if DT and &H80 then DT=DT+&HFF00
1460 OL$=OL$+FNH4$(A+DT+OFSET)+"H":return '10,18,20,28,30,38 JR
1470 if D4 then OP$="ADD":OL$="HL,"+RP$(DV+1):return '09,19,29,39 DAD
1480 OL$=RP$(DV+1)+","+FNW$(A):A=A+2:return '01,11,21,31 LXI
1490 if DM>3 then OL$=FNW$(A):A=A+2 else OL$=RP$(DV+1)
1500 OL$="("+OL$+")":if DV=2 then A$="HL" else A$="A"
1510 if D4 then OL$=A$+","+OL$ else OL$=OL$+","+A$
1520 return '02,0A,12,1A,22,2A,32,3A LDAX,STAX,LHLD,SHLD,LDA,STA
1530 if D4 then OP$="DEC" else OP$="INC"
1540 OL$=RP$(DV+1):return '03,0B,13,1B,23,2B,33,3B INX,DCX
1550 OP$="INC":goto 1570
1560 OP$="DEC"
1570 OL$=R8$(DM):return'04,1C,14,1C,24,2C,34,3C,05,1D,15,1D,25,2D,35,3DINR,DCR
1580 if DF and DT=&H36 then B=1 else B=0
1590 OL$=R8$(DM)+","+FNHS$(A+B):gosub 2600
1600 A=A+1:return '06,0E,16,1E,26,2E,36,3E MVI
1610 OP$=mid$("RLCARRCARLA RRA DAA CPL SCF CCF ",DM*4+1,4):return
1620 '07,0F,17,1F,27,2F,37,3F Other 1 byte
1630 rem 40H-7FH
1640 if DT=&H76 then OP$="HALT":SP=SK:return '76 HLT
1650 OL$=R8$(DM)+","+R8$(DL):return '40-75,77-7F MOV
1660 rem 8bit Culc.
1670 gosub 1690
1680 OL$=OL$+R8$(DL):return '80-BF ANA,ORA,....
1690 if DM<2 or DM=3 then OL$="A,"
1700 OP$=mid$("ADDADCSUBSBCANDXORORCP",DM*3+1,3):if DM=6 then OP$="OR"
1710 return
1720 rem C0-FF JP,CALL,RET,etc.
1730 on DL goto 1750,1790,1820,1910,1920,1940,1970
1740 OP$="RET":OL$=JK$(DM):return 'C0,C8,D0,D8,E0,E8,F0,F8
1750 if DM=1 then OP$="RET":SP=SK:return
1760 if DM=3 then OP$="EXX":return
1770 if DM=5 then OP$="JP":OL$="(HL)":SP=SK:return
1780 if DM=7 then OL$="SP,HL":return else OP$="POP":goto 1930
1790 OP$="JP"
1800 OL$=JK$(DM)+","+FNW$(A):A=A+2:return
1810 'C2,CA,D2,DA,E2,EA,F2,FA,C4,CC,D4,DC,E4,EC,F4,FC
1820 if DM=0 then OP$="JP":OL$=FNW$(A):A=A+2:SP=SK:return
1830 if DM=1 then 1990
1840 if DM and 4 then 1870 else OL$="("+FNHS$(A)+")":A=A+1
1850 if D4 then OP$="IN":OL$="A,"+OL$ else OP$="OUT":OL$=OL$+",A"
1860 return 'D3,DB IN,OUT
1870 if DM and 2 then 1900
1880 OP$="EX":if D4 then OL$="DE,HL" else OL$="(SP),HL"
1890 return 'E3,EB XTHL,XCHG
1900 OP$=chr$(&H44+D4)+"I":return 'F3,FB DI,EI
1910 OP$="CALL":goto 1800
1920 if DM=1 then OP$="CALL":OL$=FNW$(A):A=A+2:return else OP$="PUSH"
1930 OL$=mid$("BCDEHLAF",DM+1,2):return 'C5,D5,E5,F5,C1,D1,E1,F1 PUSH,POP
1940 gosub 1690
1950 OL$=OL$+FNHS$(A):B=0:gosub 2600
1960 A=A+1:return 'C6,CE,D6,DE,E6,EE,F6,FE ADI,ACI,...
1970 OP$="RST":OL$=right$("0"+hex$(DM*8),2)+"H":return
1980 'C7,CF,D7,DF,E7,EF,F7,FF RST
1990 rem CB BIT,SET,RES
2000 DT=peek(A+DF):gosub 1240
2010 if DH=0 and DM=6 then 2100
2020 OP$=mid$("   BITRESSET",DH*3+1,3)
```

```
2030 if DH then OL$=chr$(48+DM)+","+R8$(DL):return '(DD,FD) CB (XX) 40-FF
2040 OP$=mid$("RLCRRCRL RR SLASRA   SRL",DM*3+1,3+(DM=2 or DM=3))
2050 OL$=R8$(DL):return '(DD,FD) CB (XX) 00-2F,38-3F
2060 'ED ext.
2070 if DF then 2100 'DD+ED,FD+ED
2080 DT=peek(A):gosub 1240
2090 if DH+1 and 2 then 2110
2100 OP$="DEFB":OL$=FNHS$(S):A=S+1:return 'ED 00-3F,CB 30,37
2110 if DT>&H9F then 2300 else if DT>&H7B then 2100
2120 on DL goto 2150,2170,2190,2220,2230,2250,2270
2130 if DM=6 then 2100 else OP$="IN":OL$=R8$(DM)+",(C)":return
2140 'ED 40,48,50,58,60,68
2150 if DM=6 then 2100 else OP$="OUT":OL$="(C),"+R8$(DM):return
2160 'ED 41,49,51,59,61,69,79
2170 if D4 then OP$="ADC" else OP$="SBC"
2180 OL$="HL,"+RP$(DV+1):return 'ED 42,4A,52,5A,62,6A,72,7A
2190 OL$="("+FNW$(A)+")":A=A+2
2200 if D4 then OL$=RP$(DV+1)+","+OL$ else OL$=OL$+","+RP$(DV+1)
2210 return 'ED 43,4B,53,5B,63,6B,73,7B
2220 if DM then 2100 else OP$="NEG":return 'ED 44
2230 if DV then 2100 else SP=SK:if D4 then OP$="RETI" else OP$="RETN"
2240 return 'ED 45,4D RETN,RETI
2250 OP$="IM":if DM then if DV=1 then OL$=chr$(&H31+D4):return else 2100
2260 OL$="0":return 'ED 46,56,5E IM 0,IM 1,IM 2
2270 if DM<4 then OL$=mid$("I,AR,AA,IA,R",DM*3+1,3):return
2280 if DM>5 then 2100 else if D4 then OP$="RLD":return else OP$="RRD":return
2290 'ED 47,4F,57,5F,67,6F LD I,A·R,A·A,I·A,R
2300 if DL>3 or DM<4 then 2100
2310 OP$=mid$("LDICPIINIOTILDDCPDINDOTD",DL*3+1,3)+chr$(32-(DV=3)*50)
2320 if DV=2 and DL=3 then OP$=mid$("OUTIOUTD",D4*4+1,4):return
2330 if D4 then mid$(OP$,3)="D"
2340 return 'ED A0-AB LDI,CPI,INI,OUTI,LDD,CPD,IND,OUTD
2350 IZ$="IX":goto 2370
2360 IZ$="IY"
2370 if DF then 2100 else DF=1:AX=A:gosub 1330
2380 C=instr(OL$,"HL"):if C=0 or left$(OL$,2)="DE" then A=AX:goto 2100
2390 if instr(OL$,"(HL)") then 2420 else mid$(OL$,C)=IZ$
2400 if mid$(OL$,4,2)="HL" then mid$(OL$,4)=IZ$
2410 return '16 bit reg.
2420 if OP$="JP" then OL$="("+IZ$+")":return
2430 A=A+1:PM=peek(AX+1):if PM and &H80 then PM$="-":PM=256-PM else PM$="+"
2440 OL$=left$(OL$,C-1)+IZ$+PM$+right$(FNH4$(PM),2)+"H"+mid$(OL$,C+2):return
2450 'Index adrs.
2460 MSG$="Command= D , P , 1 - 9 , M , E , !":goto 2560
2470 if DEV then lprint else print
2480 INK$=inkey$:key 0,"":if inp(&HEA) and 4 else pause 5
2490 if inp(&HEA) and 2 else 2490
2500 if DEV=0 then return else GYOU=GYOU+1:if GYOU=<60 then return
2510 PAGE=PAGE+1:lprint chr$(12)
2520 lprint "** Z-80 Disassembler for MZ-2500 ";
2530 lprint TITLE$ " PAGE " PAGE tab(68) date$:lprint
2540 GYOU=1:return
2550 MSG$="Key in= [ESC],[BREAK],[space],[CR]"
2560 gosub 2710:poke@ 0,&H5E2,0,&H18,0,&H18,&H4F,&H18
2570 color 6:print MSG$;:color 7
2580 goto 2660
2590 return
2600 C=peek(A+B):if &H20=<C and C=<&H5F then COMMENT$="'"+chr$(C)+"'"
2610 return
2620 gosub 2670
2630 key 0,chr$(5):input "",A$
2640 locate 0,23:kmode 0:print SCR$;:kmode 1
2650 poke@ &H38,&HF57,ATR
2660 poke@ &H0,&H5E2,XC,YC,YS,XS,YE,XE:return
2670 gosub 2710:SCR$=scrn$(0,23,79):ATR=peek@(&H38,&HF57)
2680 poke@ 0,&H5E2,0,&H17,0,&H17,&H4F,&H17
2690 print MSG$;
2700 return
2710 XC=peek@(0,&H5E2):YC=peek@(0,&H5E3):YS=peek@(0,&H5E4):XS=peek@(0,&H5E5)
2720 YE=peek@(0,&H5E6):XE=peek@(0,&H5E7)
2730 return
2740 klist 2:console 0,24,0,80:init:poke@ 0,&H580,0
```



ディスアセンブラの使い方

このディスアセンブラは　RUN↵　で実行開始です。

すると，

COMMAND＝■

と聞いて来るので，コマンドを答えます。コマンドは，画面下部の表示を見ると分かると思いますが，

D　P　1　2　3　4　5　6　7　8　9　M　E　!

の14個あります（1－9というのは1から9という意味です）。

まず，「D」は，ディスアセンブル結果を画面に表示するものです。このコマンドは使用頻度が最も高いと思いますので，単に↵を押すだけでもこの意味になるようにしています。

D↵　または，ただの　↵

と入力すると，次に，

START　adrs.＝&H■

と聞いて来ます。ディスアセンブルを開始したいアドレスを入力します。さきほどの簡単なプログラムを見るなら　E000↵　のように，また，BASIC等を解析したいなら　0↵　（0000↵でも同じ）のように答えて下さい。何も入力せず，単に↵を押した時は，0の意味になります。

すると，次に，

END　　adrs.＝&H■

と聞いて来ます。ディスアセンブルを終了するアドレスを入力して下さい。E00B↵とか，A28F↵　と入力するわけです。

すると，ディスアセンブル結果が次々と画面に出て来ます。ディスアセンブルを途中で終了したい時はESCキーを押して下さい。COMMAND＝の所に戻ります。またBREAKを押すと何かキーを押すまで，（スペース）を押すとそれを離すまでの間，出力を停止します。また，↵キーを押すと，押し続けている間中，出力速度が遅くなります。ゆっくり見たい時に利用すると良いでしょう。

次に，**Pコマンド**について，説明します。このコマンドは，プリンタにディスアセンブル結果を出力するものです。まず，

P↵

と入力すると，RUNした後，初めてこのコマンドを使った時は，

プリンタの準備は良いですか？〔Y／N〕■

と聞いて来ます。プリンタの電源，SELECTまたはON－LINEスイッチ，プリンタ用紙のフォーム位置（ヘッドの位置と用紙のミシン目の位置合わせ），本体との接続等を確認したら，Y↵と答えて下さい。何かを既に印字した後では，これは聞いて来ません。

次に，

タイトルは？（15字まで。不要ならそのままリターンを押して下さい）■

と聞いて来ます。このディスアセンブラではプリンタ用紙の上部にタイトルやページ，打ち出した日付等が印字できるので，もし，タイトルを付けたい時は，それを入力して下さい。カーソル以降に何も文字が表示されていない状態で↵を押すと，タイトルは付きません。

これ以降は，Dコマンドと同様です。アドレスを聞いて来るので，それに答えれば，プリンタに打ち出しが始まります。

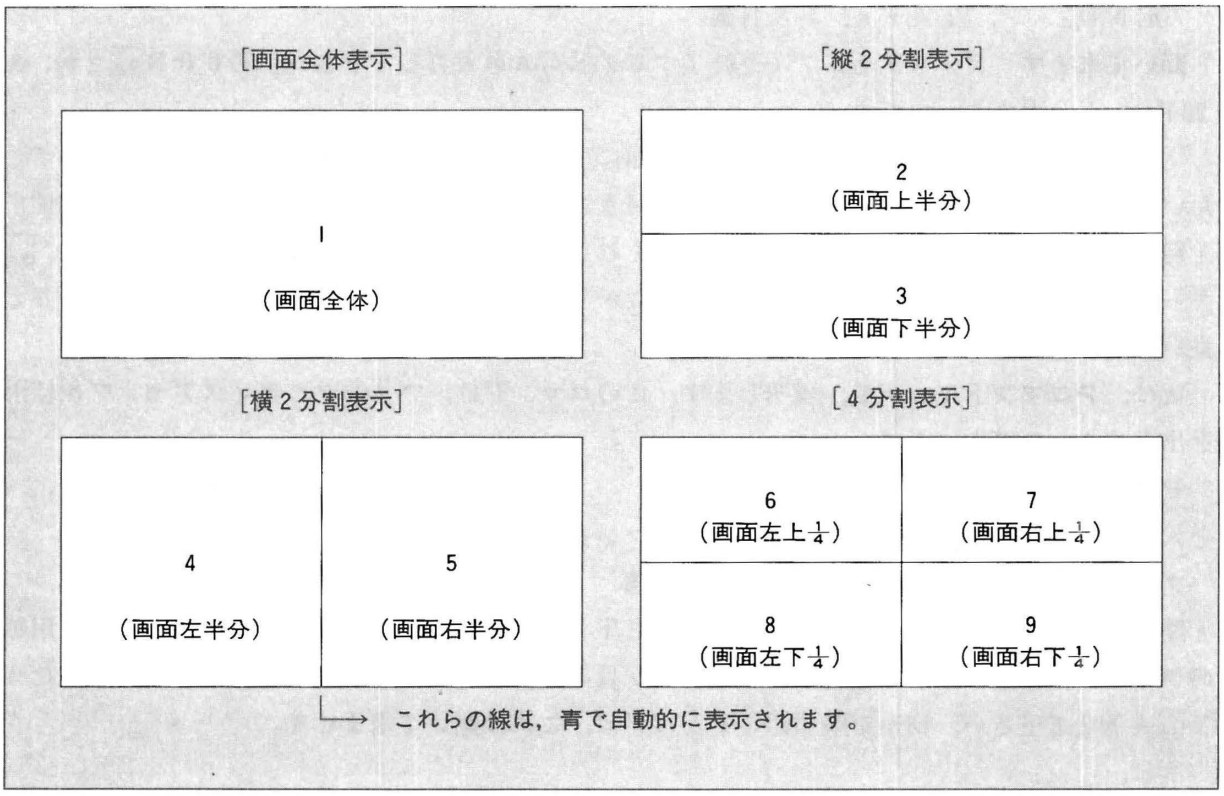
次に，**1～9のコマンド**について，説明します。

これらは，このディスアセンブラの特長である**「画面分割機能」**をサポートするものです。**第6－3図**を見て下さい。これは，分割の様子を示したものです。1では画面全体を使い（最初はこの状態），2では上半分，3は下半分，4は左半分，5は右半分，6～9は4分割した上で左上・右上・左下・右下を使います。

BASICでGOSUBの飛び先を次々と追いかけて行くと，しまいに元々どこを見ていたのか忘れてしまうことがあるように，画面上でマシン語のプログラムを解析していても，同様のことが考えられます（この場合GOSUBではなくてCALL命令）。そこで，それを少しでも改善しようと思い，この機能を付けたわけです。

次に，**Mコマンド**は，メモリマッピング用に作った命令です。

ここまでの説明では，モニタでもマシン語でも，アドレス空間64Kバイトとして話を進めて来ましたが，第2章でも触れているように，スーパーMZではメモリマッピングシステムというのを採用しているので，Z80側の論理メモリ空間64Kバイトを8分割し，その各々のブロック毎に，実メモリ空間512Kバイトのうちの任意の8Kバイトを選択させるようになっています。このディスアセンブラでは，通常，8ブロックを，モニタの各コマンドやBASICのPOKE・PE



《第6－3図》画面分割の様子

ＥＫでアクセスしている実メモリにマッピングしていますが，このコマンドを実行すると，それを変えることができます。

まず，

Ｍ↵

と入力しますと，

ＣＰＵのブロック番号は？（０～７）■

と聞いて来ます。Ｚ80やモニタやＰＯＫＥ・ＰＥＥＫやこのディスアセンブラで0000Ｈ～１ＦＦＦＨをアクセスした時，実際にどのメモリ（等）を使用するのかを設定したいのなら０を同様に2000Ｈ～３ＦＦＦＨについて設定したいのなら１を……Ｅ000Ｈ～ＦＦＦＦＨについて設定したいのなら７を入力します。

次に，

メモリのブロック番号は？（00～３Ｆ）＆Ｈ■

と聞いて来ます。ここで，実際にどのメモリ（等）を使用するのかを答えるわけです。例えば，

００～０Ｆ…標準装備のＲＡＭ（128Ｋバイト中の８Ｋバイト）

１０～１Ｆ…オプションのＲＡＭ（128Ｋバイト中の８Ｋバイト）

３４～３７…ＩＰＬ・ＢＩＯＳ　ＲＯＭ（32Ｋバイト中の８Ｋバイト）

のように対応します。この値は，ＰＯＫＥ@やＰＥＥＫ@で指定するブロック番号と同じものですが，これ以外の部分（例えばグラフィックＲＡＭ）にマシン語プログラムが置かれることは通常考えられませんので，普通は上の範囲の値を答えます。

なお，例えば，0000Ｈ～７ＦＦＦＨを一斉に変えたい，といったような場合には，Ｍコマンドを何度も実行して下さい。

次に，**Ｅコマンド**は，ディスアセンブラから抜けます。つまり，ＢＡＳＩＣに戻るわけです。例えば，ディスアセンブル中にもＳＨＩＦＴ＋ＢＲＥＡＫで強制的に実行を中断させたりせず，一度ＥＳＣで　ＣＯＭＭＡＮＤ＝　に戻り，さらにこのコマンドでＢＡＳＩＣに戻るようにして下さい。特に，Ｍコマンドでメモリの0000Ｈ～１ＦＦＦＨの部分をブロック00Ｈ以外（例えば34ＨのＩＰＬ・ＢＩＯＳ－ＲＯＭ）に割り当てていた場合には，この後モニタに入ってＤコマンド等を実行すると，暴走することがあります。また，0000Ｈ～１ＦＦＦＨのマッピングを変えていない場合でも，画面の状態が元に戻りませんので，このコマンドを利用して下さい。もし，いつものクセ（？）でＳＨＩＦＴ＋ＢＲＥＡＫしてしまった，という場合には，再びＲＵＮ（Ｆ５キーが便利）して，Ｅ↵とするのが手っ取り早くて良いでしょう。なお，0000Ｈ～１ＦＦＦＨ以外のマッピングは，ＥコマンドでＢＡＳＩＣに戻った後も，ＣＬＥＡＲ　アドレス　または　ＣＬＥＡＲ　ＭＡＸ　を実行するまで，有効です。

最後に，**！コマンド**は，Ｐコマンドにおけるページ数・タイトル・フォーム位置を初期化します。Ｅコマンド実行後，再びＲＵＮしたのと同じです。

ところで，ディスアセンブラは，メモリに入っている値を，ＣＰＵがマシン語として実行した場合，どのような意味になるかを我々に知らせてくれるものですが，初めにも触れたように，メモリ上には，ＣＰＵが実行しない「データ」も含まれています。しかし，ディスアセンブラにはどの部分がデータエリアなのかは分かりません。「Ｂｒｅａｋ」や「Ｏｋ」などという，人間に

は一目瞭然のデータも，ディスアセンブラには判別できないのです（どんなディスアセンブラでもそうです）。

そこで，データエリアは皆さんに発見してもらうしかないのですが，このディスアセンブラでは，少しでもそれが容易にできるよう，いくつか，工夫をしています。

まず，モニタと同じように，キャラクタダンプを付けています。メッセージ等用の文字列はすぐ，発見できるでしょう。

次に，サブルーチンの区切れ目，例えば無条件のＲＥＴ・ＪＰ・ＪＲ等の後には，空行を１行（プリンタでは何か書込んで解析に役立てられるよう，２行）開けています。データは，そのような場所にあることが多いからです。

データはまた，プログラマの趣味（？）によっては，ＣＡＬＬ命令やその一種であるＲＳＴ命令（ＣＡＬＬ　0000H・ＣＡＬＬ　0008H・ＣＡＬＬ　0010H…ＣＡＬＬ　0038Hを１バイトで実行できる）直後に置かれることがあります。こうなると，そこに文字列以外が置かれている場合には，お手上げ状態に近くなるのですが，ＭＺ－2500モードでは，ＲＳＴ　18H（ＣＡＬＬ　0018Hと同意）とＲＳＴ　28Hの後には必ず１バイトのデータが来ることになっていますので，それだけはディスアセンブラがデータエリアとして判別するようにしています。なお，これは1310行を削除するか，「ｉｆ」の前に「’」を付けて注釈にすれば，止めることができます。

```
Z-80 Disassembler for MZ-2500
480A E1          POP     HL                    ※
480B D1          POP     DE                    Q
480C DDE5        PUSH    IX                    ]※
480E CD1D48      CALL    481DH                 M H
4811 DDE1        POP     IX                    ]※
4813 E1          POP     HL                    ※
4814 CDB721      CALL    21B7H                 M7!
4817 C8          RET     Z                     H
4818 CD7233      CALL    3372H                 Mr3
481B 18D9        JR      47F6H                  Y

481D 1A          LD      A,(DE)
481E FE03        CP      03H                   ■
4820 CA8753      JP      Z,5387H               J S
4823 E5          PUSH    HL                    ※
4824 48          LD      C,B                   H
4825 0600        LD      B,00H
4827 09          ADD     HL,BC
4828 3600        LD      (HL),00H              6
482A E1          POP     HL                    ※
482B 47          LD      B,A                   G
482C D5          PUSH    DE                    U
Command=
Command= D , P , 1 - 9 , M , E , !
```

《第６－４図》ディスアセンブラ実行画面（分割しない時）

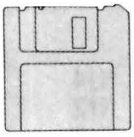
```
Z-80 Disassembler for MZ-2500
                                                  20A7 CD5D34   CALL  345DH        M]4
003E DDE9      JP    (IX)          ]※             20AA CD2125   CALL  2521H        M!%
                                                  20AD CD9F23   CALL  239FH        M #
0040 ED4D      RETI                ▒▒             20B0 AF       XOR   A            /
                                                  20B1 329520   LD    (2095H),A    2
0042 DDE3      EX    (SP),IX       ]※             20B4 CD6B26   CALL  266BH        Mk&
0044 E5        PUSH  HL            ※              20B7 DF       RST   18H          °
0045 C5        PUSH  BC            E              20B8 02       DEFB  02H
0046 D5        PUSH  DE            U              20B9 111325   LD    DE,2513H      %
0047 F5        PUSH  AF            ※              20BC DF       RST   18H          °
0048 CD3E00    CALL  003EH         M>             20BD 05       DEFB  05H
-------------------------------------------------------------------------------------
6001 5D        LD    E,L           ]              E000 3A11E0   LD    A,(E011H)    : ※
6002 C9        RET                 I              E003 47       LD    B,A          G
                                                  E004 3A10E0   LD    A,(E010H)    : ※
6003 CDE45B    CALL  5BE4H         M膽            E007 80       ADD   A,B
6006 CD1A5E    CALL  5E1AH         M ^            E008 3212E0   LD    (E012H),A    2 ※
6009 AF        XOR   A             /              E00B C9       RET                I
600A 32CA0C    LD    (0CCAH),A     2J
600D DD210110  LD    IX,1001H      ]!             E00C 00       NOP
6011 CD7F33    CALL  337FH         M⌧3            E00D 00       NOP
6014 F8        RET   M             ※              E00E 00       NOP
Command=
Command= D , P , 1 - 9 , M , E , !
```

《第６－５図》ディスアセンブラ実行画面（４分割の時）

```
** Z-80 Disassembler for MZ-2500 <TEST> PAGE  1                          85/09/23

20A4 213B9E          LD     HL,9E3BH                                      !;
20A7 CD5D34          CALL   345DH                                         M]4
20AA CD2125          CALL   2521H                                         M!%
20AD CD9F23          CALL   239FH                                         M #
20B0 AF              XOR    A                                             /
20B1 329520          LD     (2095H),A                                     2
20B4 CD6B26          CALL   266BH                                         Mk&
20B7 DF              RST    18H                                           °
20B8 02              DEFB   02H
20B9 111325          LD     DE,2513H                                       %
20BC DF              RST    18H                                           °
20BD 05              DEFB   05H
20BE DF              RST    18H                                           °
20BF 02              DEFB   02H
20C0 CD5D34          CALL   345DH                                         M]4
20C3 CD6B26          CALL   266BH                                         Mk&
20C6 3A1525          LD     A,(2515H)                                     : %
20C9 B7              OR     A                                             7
20CA 200B            JR     NZ,20D7H
20CC ED7B9205        LD     SP,(0592H)
20D0 3EFF            LD     A,FFH                                         >
20D2 F5              PUSH   AF                                            ※
20D3 ED738B20        LD     (208BH),SP
20D7 210000          LD     HL,0000H                                      !
20DA 226820          LD     (2068H),HL                                    "h
```

《第６－６図》ディスアセンブラ印字サンプル



IOCSサブルーチンを利用したプログラムを組む

さて，さきほど説明したマシン語プログラムは，それ単独で動くものでした。つまり，外部とのコミュニケーションを行わないプログラムだったわけです。そういったプログラムは，Z80を採用している他のパソコン（例えばX1）でも全く同じように動作させることができます（モニタの仕様が違うと，最後のRETだけは変えなければならないこともある）。

しかし，マシン語プログラムを使って画面に何かを表示させたい，といったような場合には，単独で動く，「メモリやI／Oポートに接続されたLSIを制御するプログラム」をわざわざ新たに作るのではなく，すでにスーパーMZのメモリ上に置かれているプログラムを呼び出して，使うのが便利です。

幸い，スーパーMZの場合，その方法がきちんと確立されており，これを利用するとBASICでやっていることの多くは，マシン語でもできるようになります。

次のようなプログラムを入力して下さい。CLEAR &HE000としてからMONを実行するのをお忘れなく。

：E000＝3E 41 DF 03 C9 00 00 00

00の部分は他の値でも構いません。

では，実行してみます。

＊GE000↵

とします。すると， A と表示されるでしょう。

このプログラムは，アセンブリ言語で書くと，

```
LD      A，41H
RST     18H
DEFB    03H
RET
```

になります。Aレジスタに，41Hを入れ，（ ）がないから，アドレス41Hに入っているデータではなく，41Hそのものです），RST 18Hつまり，0018H番地から始まるサブルーチンを呼び出します（CALL 0018Hでも良い）。次のDEFB 03Hは，データを置くためのアセンブラに対する擬似命令です。そして，RETでモニタに戻ります。

さて，「0018Hから始まるサブルーチン」というのは，第7章でも触れていますが，スーパーMZの外部とのコミュニケーションを行うもので，RST命令の次に置かれた1バイトのデータにより，働きが変わります。**03Hというのは，「Aレジスタの値」をキャラクタコードとみなし，そのコードの文字を画面に表示せよ，**という意味です。BASIC風に書くと，PRINT CHR$（A）；とでも言う働きをするのです。

もう一つ，サンプルをお見せしましょう。

：E000＝DF 0C DF 05 C9 00 00 00

これは，

RST 18H

```
DEFB  0CH
RST   18H
DEFB  05H
RET
```

という意味です。実行させてみて下さい。

0CHは「キーボードから1行入力し，結果を所定のバッファに入れる。そのバッファのアドレスは，DEレジスタに入れて帰る」，05Hは「画面に文字列を表示する。ただし，表示させたい文字列は，あらかじめメモリ上に確保し，そのアドレスをDEレジスタで指しておく」という働きです。

今度は05H（文字列表示）だけのテストをしてみます。

```
:E000=11  10  E0  DF  05  C9  00  00
:E008=00  00  00  00  00  00  00  00
:E010=4D  5A  2D  32  35  30  30  8A
:E018=88  97  70  8C  A4  8B  86  82
:E020=F0  82  A8  94  83  82  A2  8F
:E028=E3  82  B0  92  B8  82  AB  82
:E030=DC  82  B5  82  C4  90  BD  82
:E038=C9  97  4C  82  E8  93  EF  82
:E040=A4  82  B2  82  B4  82  A2  82
:E048=DC  82  B5  82  BD  81  42  0D
:E050=92  98  8E  D2  91  E3  95  5C
:E058=20  8D  82  8B  B4  97  59  88
:E060=EA  00  00  00  00  00  00  00
```

００は他の値になっていても結構ですが，Ｅ０６１Ｈのものだけはきちんと00にして下さい。(表示したい文字列の末尾には00Hを置きます)。

そして，同様に実行させてみて下さい。E000H～E005Hまではマシン語ですが，E010H～E061Hまではデータです。マシン語の部分は，皆さんで，ディスアセンブルして確かめてみて下さい（DEレジスタで指す，の意味が分かりましたか？)。

Z80の隠し命令

Z80の命令はマシン語レベルで細かく分けると700種類近く（正しくは698種類)，それを大まかに分類する（ＡＤＤ　Ａ，ＡとＡＤＤ　Ａ，Ｂは一つと数えるがＡＤＤ　Ａ，ＡとＡＤＤ　ＨＬ，ＢＣとは分けて数える）と158種類あると言いましたが，実はZ80には隠し命令があります。

マシン語の構造を良く見ると分かるのですが，マシン語は無造作に決められているのではなく，ある程度規則的に並べられています。それが特に顕著なのは，ＩＸ・ＩＹレジスタの扱いです。これが使われる命令ではすべて，命令の１バイト目がＤＤＨ（ＩＹ

の場合）またはＦＤＨ（ＩＹの場合）になっています。しかも，ＩＸ・ＩＹが単なるレジスタとして使われる時はＨＬレジスタに対する命令と同じものが２バイト目以降に続き，ポインタ（(ＩＸ＋ｎｎＨ)等）として使われる時は３バイト目にオフセット値が，２バイト目（と，必要なら４バイト目）にＨＬレジスタに対する命令と同じものが入ります。つまり，ＣＰＵ内部では，ＤＤＨやＦＤＨが現われた時は，以下のＨＬレジスタに対する命令をＩＸ・ＩＹに対して行うようになっているわけです。

ところで，

ＤＤ　６７

というマシン語を実行する（正式な命令ではないから，アセンブリ言語は定められていない）と，Ａレジスタの内容がＩＸレジスタの上位８ビットに転送されます。また，

ＤＤ　７Ｃ

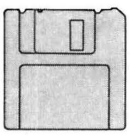
なら，その逆にＩＸレジスタの上位８ビットがＡレジスタに入ります。

他にも，ＨレジスタまたはＬレジスタが単独で，つまりペアレジスタとしてでなく，８ビットのレジスタとして使用される命令では，前にＤＤＨまたはＦＤＨを付けるだけでＩＸ・ＩＹの上位（Ｈレジスタに対する命令の時）または下位（Ｌレジスタに対する命令の時）に対する命令になります。ちなみに67ＨはＬＤ　Ｈ，Ａ，７ＣＨはＬＤ　Ａ，Ｈという命令です。

本来なら，こうした命令はメーカーが保証しているわけではないので，何台かのパソコンで動作が確認されたからといって，あるいは何千台ものパソコンで確認されたとしても，全部のＺ80で通用するとは限りません。個人で楽しむ分には大変面白い実験ですが，市販するようなプログラムでは使うべきではありません。

と，従来言われて来たのですが，スーパーＭＺでは，Ｚ80の正式なセカンドソースメーカーであるシャープ自身が，ＢＡＳＩＣ及びＩＯＣＳ（ＲＳＴ　18Ｈ等の実際の処理を行う部分）の中にこれらの命令を使用していることが分かりました。言わば，これらの隠し命令に対し，メーカーの「お墨付き」が出たと言っても良いでしょう。

なお，スーパーＭＺでは使われていないようですが，他にも　ＥＤ　７０（Ｃレジスタで指すポートからデータを入力し値は捨てられる。つまりフラグのみ変化する。これはアセンブリ言語の表記こそ決まっていないが，ザイログ・シャープの資料には文書でこの命令が認められている）や，ＣＢ　30 ～ ＣＢ　37（ローテイト＝回転命令の一種）等が知られています。



フラグの働きをマスターしよう！

マシン語プログラムでキーポイントとなるものの一つに，**「フラグ変化」**があります。これをマスターしますと，結構使えるプログラムを組めるようになります。本章の締めくくりとして，**ＣＶフラグ**と**Ｚフラグ**について，見てみることにしましょう。

さきほど，８ビットの値と８ビットの値との加算を行うごく簡単なプログラムを作ってみましたが，今度は16ビット＋16ビットをやってみましょう。

```
ＬＤ　　Ａ，Ｌ
ＡＤＤ　Ａ，Ｃ
ＬＤ　　Ｌ，Ａ
ＬＤ　　Ａ，Ｈ
ＡＤＣ　Ａ，Ｂ
ＬＤ　　Ｈ，Ａ
```

上のプログラム（の破片？）は，ＨＬレジスタの値と，ＢＣレジスタの値を加算し，結果をＨＬレジスタに入れるものです。例えばＨＬレジスタには6282Ｈ，ＢＣレジスタには398DHが入っているものとして，動きを追ってみましょう。

まず，ＬＤ　Ａ，ＬでＬレジスタの値をＡレジスタに持って来た上で，ＡＤＤ　Ａ，ＣでＣレジスタの値を加えます。結果は10ＦＨになる筈ですが，Ａレジスタには０ＦＨしか入りません。なぜなら，Ａレジスタは８ビットだからです。結果がどうであろうと，８ビット分のデータしか入らないのです。その代り，このように繰り上がりがあった時は，Ｃｙフラグ(キャリーフラグ)がセットされます。フラグは言わば，１ビットの記憶素子であり，常に０か１かのどちらかの値を保持しています。なお，フラグが１へ変化する，または変化させることを「セットする」，０へ変化する，または変化させることを「リセットする」と言います。話を戻すと，Ｃｙフラグは繰り上がりがあった時は１に，なかった時は０に変化するわけです。そして，加算の結果，Ａレジスタに入った値，つまり０ＦＨは，次のＬＤ　Ｌ，ＡでＬレジスタに転送されます。この時はフラグは変化しません。このＬＤ命令は，フラグ変化をしない命令だからです。

次に，ＬＤ　Ａ，Ｈで，Ｈレジスタの値をＡレジスタに持って来ます。この時もフラグは変化しません。そして，次に　ＡＤＣ　Ａ，Ｂ　とすると，そこにＢレジスタの値とＣｙフラグの値を加えます。例の場合ですと，62Ｈと39Ｈと，さらに１（Ｃｙの値）を加算し，結果９ＣＨがＡレジスタに入ります。そして，ＬＤ　Ｈ，Ａで，Ｈレジスタにその値が転送されます。なお，ＡＤＣ　Ａ，Ｂの後には，このＡＤＣ命令の結果，繰り上がりがあったかなかったのかが，新たにＣｙフラグに入ります。この例では，62Ｈ＋39Ｈ＋１＝９ＣＨですから(19ＣＨではないですから)，Ｃｙフラグはリセットされます。

まとめると，ＨＬ＋ＢＣ→ＨＬ，つまり，6282Ｈ＋398ＤＨ→９Ｃ０ＦＨ が計算されたわけです。

また，さきほどの命令に続けて

```
ＬＤ　　Ａ，Ｄ
ＡＤＣ　Ａ，Ｅ
ＬＤ　　Ｄ，Ａ
```

を実行すると，ＤＨＬ＋ＥＢＣ→ＤＨＬつまり24ビットの加算を行うことができます。

次に，多桁の減算を行う例です。

```
ＬＤ　　Ａ，Ｌ
ＳＵＢ　Ｃ
```

```
LD    L，A
LD    A，H
SBC   A，B
LD    H，A
```

やはりＨＬ＝6282Ｈ，ＢＣ＝398ＤＨで追ってみましょう。

まず，ＬＤ　Ａ，ＬでＡレジスタに82Ｈが入り，ＳＵＢ　Ｂでそこから８ＤＨを引こうとします。しかし，82Ｈ－８ＤＨは計算できませんので，Ｃｙフラグをセットした上で，182Ｈ－８ＤＨとして計算するのです。結果はＦ５Ｈとなり，Ａレジスタにこの値が入ります。そしてＬＤ　Ｌ，Ａで，これ（Ｆ５Ｈ）がＬレジスタに転送されます。

次に，ＬＤ　Ａ，Ｈで，Ａレジスタには62Ｈが入ります。そしてＳＢＣ　Ａ，Ｂでここから39ＨとＣｙフラグの値が引かれます。62Ｈ－39Ｈ－１が計算されるわけです。Ｃｙフラグを引くのは，さきほどの「借り」を返してもらうためです。結果は28Ｈとなり，ＬＤ　Ｈ，ＡでＨレジスタにこれが転送されます。

これで6282Ｈ－398ＤＨ→28Ｆ５Ｈが計算できたわけです。また，続けて

```
LD    A，D
SBC   A，E
LD    D，A
```

を行うと，ＤＨＬ－ＥＢＣ→ＤＨＬという，24ビットの減算ができます。

そして，フラグは，比較を行う時にも利用します。この場合には，Ｃｙフラグともに，Ｚフラグも活躍します。フラグを利用する頻度としては，これが最も高いと思います。

次のプログラムを実行してみて下さい。

```
：Ｅ０００＝３Ａ　１１　Ｅ０　４７　３Ａ　１０　Ｅ０　Ｂ８
：Ｅ００８＝３０　０１　７８　３２　１２　Ｅ０　Ｃ９　００
：Ｅ０１０＝２０　１０　００　００　００　００　００　００
＊ＧＥ０００↵
```

で良いわけです。

このプログラムは，

```
Ｅ０００    LD    A，（Ｅ０１１Ｈ）
Ｅ００３    LD    B，A
Ｅ００４  ア LD    A，（Ｅ０１０Ｈ）
Ｅ００７  ド CP    B
Ｅ００８  レ JR    NC，  Ｅ００ＢＨ
Ｅ００Ａ  ス LD    A，B              Ｃｙ＝０ならここにジャンプ
Ｅ００Ｂ      LD    （Ｅ０１２Ｈ），A
Ｅ００Ｅ      RET
```

という意味です。まず，ＬＤ　Ａ，（Ｅ０１１Ｈ）とＬＤ　Ｂ，Ａで，Ｂレジスタにアドレス０１１Ｈのメモリの値が入ります。そして，ＬＤ　Ａ，（Ｅ０１０Ｈ）でＡレジスタにＥ０１０Ｈのメモリの値が入ります。ここまでは何度となく見て来た通りです。

次に，ＣＰ　Ｂという命令が出て来ます。この命令は比較を行うものですが，厳密に言うとＡレジスタの値からＢレジスタの値を減じ，結果をどこにも入れない，という働きをします。つまり，ＣＰ命令は減算の際に変化するフラグだけが必要な時に使うのですが，実際はそう固く考えることはなく，ＳＵＢやＳＢＣとは特に関係ない命令として覚えるのが良いでしょう（ただ，フラグ変化はＣＰ命令とＳＵＢ命令は全く同じですので，考えが行きづまった時にはこれを思い出すと良いでしょう）。

例の場合では，Ａレジスタには20Ｈが，Ｂレジスタには10Ｈが入っていますので，20Ｈと10Ｈを比較することになります。ＣｙフラグはＡレジスタより比較した相手（ここではＢレジスタ）の方が大きいならセット，それ以外ではリセットされますので，ここでは20Ｈ≧10Ｈにより，Ｃｙフラグはリセットされます。また，このプログラムでは無視していますが，20Ｈ≠10Ｈ（20Ｈ＜＞10Ｈ）ですので，Ｚフラグもリセットされます（Ｚフラグは値が等しい時のみセットされる）。また，ＡレジスタやＢレジスタの値は変化しません。

次のＪＲ　ＮＣ，という命令は，Ｃｙフラグがリセットされていたら分岐する，という命令です。マシン語は30　01となっていますが，ＪＲ　ＮＣ，命令の次の命令の始まるアドレス，つまりＥ00ＡＨを原点として，＋01Ｈのアドレス（Ｅ00ＢＨ）にジャンプするということです。「ＪＲ」で始まる全部の命令と，「ＤＪＮＺ」命令では，（条件付きならその条件を満足したなら）その２バイト目が00Ｈ～７ＦＨの時は＋方向に，80Ｈ～ＦＦＨでは－方向（－80Ｈ～－01Ｈ）にジャンプせよ，という意味なので，ＪＲまたはＤＪＮＺの命令のあるアドレスから数えると，－７ＥＨ～＋81Ｈ（－126～＋129）の範囲にジャンプできるわけです。もし，この範囲外にジャンプしたいなら，３バイト命令のＪＰを使用します。ＪＰはＪＲのように相対アドレス指定ではなく，ジャンプしたいアドレスを直接指定しますので，ある意味では分かり易いかも知れません。ただ，命令長が長くなるので，近いアドレスへの分岐には，２バイトのＪＲを使うのが一般的です。なお，難しい話になりますが，ＪＲ命令の次の命令を起点とするのは，まず，分岐先を示す，ＪＲ等の命令の２バイト目（01Ｈ等）を取込み，プログラムカウンタが次の命令のアドレスを指した後で，条件が成立するかどうかを判断して，プログラムカウンタへの加算を行うためです。

話を戻しましょう。この例の値の場合，ＪＲ　ＮＣ，では，分岐する条件は成立しています。Ｃｙフラグはリセットされている（Ｎｏ　Ｃａｒｒｙ）からです。

すると，次の　ＬＤ　Ａ，Ｂは実行せず，いきなりＥ00ＡＨのＬＤ（Ｅ01Ｈ），Ａを実行します。そして次のＲＥＴでモニタに帰って来ます。

この結果，例の場合では，Ｅ012Ｈには20Ｈが入ることになります。

次に，同じプログラムで，Ｅ010ＨをＦＥＨに，Ｅ011ＨをＦＦＨにして，実験してみて下さい。

ＣＰ　Ｂの所では，ＡレジスタにはＦＥＨが，ＢレジスタにはＦＦＨが入っていますので，今度は，Ｃｙフラグはセットされます。

すると，ＪＲ　ＮＣ，Ｅ00ＡＨは条件が不成立ですので，特に何もせず，次のＬＤ　Ａ，Ｂを実行します。そして，ＬＤ（Ｅ012Ｈ），ＡとＲＥＴは同様に実行します。

その結果，Ｅ012Ｈには元のＢレジスタの値が転送されます。つまり，ＦＦＨが入るわけです。

つまり，このサンプルプログラムは，**「Ｅ０１０Ｈの値とＥ０１１Ｈの値を比較して，大きい方をＥ０１２Ｈに入れる」**という働きをするプログラムだったわけです。Ｅ010ＨとＥ011Ｈのデー

タをいろいろ入れ替え，E012Hの結果を確かめてみて下さい。

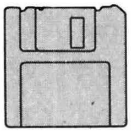
また，E008Hからの2バイトをＪＲ　Ｃ，E00AHに変える（38　01にする）と，小さい値をE012Hに入れるプログラムになります。

また，ＪＲ　ＮＣ，E00AHでなく，ＪＲ　ＮＣ，E00CH（30　03）にしてみて下さい。すると，E011Hに入っている値の方が大きい場合だけ，E012Hにそれが入ります。予めE012 Hに関係なさそうな値（例ば80H）を入れた上で実験してみて下さい。

では，ここで，ＣＰ命令における，ＣｙフラグとＺフラグの変化をまとめておきましょう。

条　　件	例		フ　ラ　グ　変　化	
	Aレジスタの値	Bレジスタ等の値	Ｃｙフラグ（キャリーフラグ）	Ｚフラグ（ゼロフラグ）
Aレジスタ≧比較する値	３５Ｈ	３２Ｈ	リセット（０）	リセット（０）
Aレジスタ＝比較する値	３０Ｈ	３０Ｈ	リセット（０）	セット（１）
Aレジスタ＜比較する値	４ＤＨ	５ＡＨ	セット（１）	リセット（０）

比較する値：Ｂレジスタや他の８ビットレジスタ，直接指定した値，ＨＬやＩＸ・ＩＹでポイントしたメモリで，ＣＰ　命令のオペランドになっているものです。



まとめ

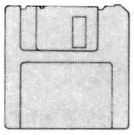
誌面の都合もあり，命令個々の説明はできませんでしたが，もともとマシン語の命令は単純なものばかりなので，ある程度マシン語の感覚をつかんでもらえば，後は解析と実践に限ると思います。もし，今後，市販のＺ80の解説書を使用するのでしたら，一つ一つの命令を詳細に説明したものを選ばれるのが良いと思います。本章は，そういった書籍とはうまく補完するように，「ころも」の部分にも気を使って話を進めて来たつもりです。

一人でも多くの皆さんが，スーパーＭＺのマシン語の世界を楽しんでいただければと思います。



第7章 ハードウエア内部解析

この章では，スーパーＭＺのＩ／Ｏポートのアドレス，入出力等のハードウエアの内部解析を紹介します。初めにお断わりしておきますが，これらの内容は，一部メーカー側が発表しているもの（マニュアルに記載されているもの）を除き，筆者が独自に解析したものです。よって，これらについてメーカーに問い合わせることは御遠慮下さい。



ハードウエア解析表の見方

本章では，

●各Ｉ／Ｏポートのアドレス

●入出力の方向（入力または出力または両方可）

●入力の場合の意味（入力できる場合のみ）

●出力の場合の意味（出力できる場合のみ）

について，まとめてあります。一つのポートアドレスで入力・出力するビットはビット７（ｂ７）～ビット０（ｂ０）までの８ビットですが，できる限り，各ビット毎に，しかもそのビットが０の場合と１の場合に分けて，詳しく説明したつもりです。ただし，

●あるものの大きさや強さを連続的に示すもの

●あるものの位置を連続的に示すもの

などについては，単に「００００～１１１１（それを４ビットで表わす場合）で，××を示す」のように簡略化しているものもあります。

また，いくつものビットが合わさって複数の意味を持つものがあります（特に出力の場合）。このようなものでは，全ビットについて0と1に場合分けすると，かえって見づらくなりますので，

０００　…　黒

００１……　暗い青

のように，数ビットまとめて示しています。

一方，一つのポートアドレスが入力も出力も可能な場合がありますが，これは次の二つのうちのどちらかです。

●本来，出力を行うためのポートであるが，現在出力しているデータを入力することでモニタできる場合。8255やＺ80ＰＩＯなど汎用ＬＳＩに多い。または，時計のように読出すことも書換えることもできる場合（要するに入力と出力とが同じ意味を持つもの）。

●入力した場合にアクセスされる回路と，出力した場合にアクセスされる回路とが異なるもの。これはＩ／Ｏポート空間節約のため，単なる出力専用ポートと，単なる入力専用ポートとを，たまたま同じポートアドレスに割り当てたものと考えて良い。ランダムロジック（小規模，ＩＣで作った回路）かそれをゲートアレイに置換えたものに多い。

このうち，後者は入力の場合と出力の場合とに分けて説明していますが，前者の方では入力と出力とをまとめて書いています。

また，各ポートは原則としてマシン語（アセンブリ言語）の

ＩＮ　　A, (nnH)
ＯＵＴ　　(nnH), A　　} アドレス直接指定

ＩＮ　　レジスタ名, (C)
ＯＵＴ　　(C), レジスタ名　　} Ｃレジスタによるアドレス間接指定

などと言った命令を使ってアクセスできますが，

●4096色パレットボード上のパレットレジスタ

●カレンダクロック（ＲＰ－５Ｃ１５）

の二つについては，原則としてＣレジスタを使った入出力命令を使わなければなりません。

そして，同時にＢレジスタには，表中に示している拡張ポートアドレスを入れておく必要があります。

これは，そのままでは16個ものアドレス空間を占有することになるものを，見かけ上１個で済ませるための工夫なのですが，プログラミングの際には気を付ける必要があります。ただ，入力については

```
ＬＤ　　Ａ，００Ｈ
ＩＮ　　Ａ，（ＣＣＨ）
```

のようにすることで，

```
ＬＤ　　ＢＣ，００ＣＣＨ
ＩＮ　　Ａ，（Ｃ）
```

と同じ効果を得ることができます（しかもＢＣレジスタを破壊せずに済む！）。かなり専門的な話になりますが，理由を説明しておきましょう。

Ｚ80の場合，Ｉ／Ｏ空間はメモリ空間と同じ，64Ｋバイトあります。アドレスを伝達する信号線「アドレスバス」はA_{15}～A_0までの16本あるのです。しかし，Ｚ80の前身である「８０８０」というＣＰＵでは，Ｉ／Ｏ空間はA_7～A_0の256バイト分しかありませんでした。Ｚ80は8080のすべての命令を持っていて，拡張命令も多くあり，そのＺ80特有の命令では64Ｋバイトサポートし

ているのですが，8080コンパチの命令ではそれが不完全なのです。

つまり，Z80オリジナルの

ＩＮ　　レジスタ名，（Ｃ）

と

ＯＵＴ　（Ｃ），レジスタ名

では，Ｃレジスタの内容がアドレスの下位８ビット，A_7～A_0 に出力されるとともに，Ｂレジスタの内容が，上位８ビット，A_{15}～A_8に出力されるため，64Ｋバイトの空間をアクセスできます。そして，ＩＮではデータを伝達するデータバスにはポートの内容が現われ，レジスタに代入されますし，ＯＵＴではデータバスにレジスタの値が出力されます（データバスは双方向です）。

しかし，8080コンパチの

ＩＮ　　Ａ，（nnH）

と

ＯＵＴ　（nnH），Ａ

では指定したnnHというポートアドレスがA_7～A_0 に出力されるとともに，Ａレジスタの内容がA_{15}～A_8に出力されます。そしてＩＮではデータバスに現われた値をＡレジスタに代入しますし，ＯＵＴではＡレジスタの値がデータバスに出力されます。一見問題なさそうですが，ＯＵＴではＡレジスタの内容がA_{15}～A_8とデータバスの両方に出力されてしまう点に注目して下さい。つまり，

```
ＬＤ　　Ａ，００Ｈ
ＬＤ　　ＢＣ，００ＣＣＨ
ＯＵＴ　（Ｃ），Ａ
```

や

```
ＬＤ　　Ａ，０１Ｈ
ＬＤ　　ＢＣ，０１ＣＣＨ
ＯＵＴ　（Ｃ），Ａ
```

を

```
ＬＤ　　Ａ，００Ｈ
ＯＵＴ　（ＣＣＨ），Ａ
```

や

```
ＬＤ　　Ａ，０１Ｈ
ＯＵＴ　（ＣＣＨ），Ａ
```

に置換えることができても

```
ＬＤ　　Ａ，０１Ｈ
ＬＤ　　ＢＣ，００ＣＣＨ
ＯＵＴ　（Ｃ），Ａ　　を
ＯＵＴ　（nnH），Ａには置換えることはできないのです。もっと簡単に言うと，
００ＣＣＨに００Ｈを
０１ＣＣＨに０１Ｈを
```

０２ＣＣＨに０２Ｈを

出力することができても，それ以外の値は出力することはできないということです。これが不完全と言った理由ですが，例えこれが可能な場合（確率的には$\frac{1}{256}$です）でも，ＯＵＴ　（ＣＣＨ），Ａを使うのはお勧めできません。デバッグその他で，違う値を出力したくなってもマシン語モニタでは修正が困難だからです。やはり，4096色のパレットレジスタとカレンダクロックへの出力には，ＯＵＴ　（Ｃ），レジスタ名の方を使って下さい。

なお，間接指定の入出力命令には，他にＯＵＴＩ・ＯＴＩＲ・ＯＵＴＤ・ＯＴＤＲ・ＩＮＩ・ＩＮＩＲ・ＩＮＤ・ＩＮＤＲがあり，同様に利用できますが，このうち，ＯＵＴＩ・ＯＴＩＲ・ＯＵＴＤ・ＯＴＤＲでは，カウンタとなるＢレジスタの値は，実際にはデータが出力される前に－１されますので，充分注意して下さい。

パレットレジスタとカレンダクロック以外にも，アクセス方法の複雑なＬＳＩがいくつかあります。

ＬＳＩの機能が増え，複雑化してゆくと，ＣＰＵからの命令はポート一つや二つでは足りなくなって来ます。その場合，カレンダクロックのような方法を採ると特にＺ80以外のＣＰＵに接続した時にＩ／Ｏ空間を浪費し，汎用性が下がってしまうのと，ＬＳＩのピン数が必要以上に増えて部品のスペースが大きくなってしまったりコストアップにつながることがあるため，ＣＰＵからＬＳＩ内部のレジスタ番号を選択した上で，別のポートにそのレジスタへのデータを書込む（または読出す）という，２段階アクセスを行う方式が考案されました。スーパーＭＺが使用しているＬＳＩでは，

●Ｚ８０（Ｂ）　ＳＩＯ
●メモリマッピングレジスタ（メモリコントローラ内）
●グラフィックコントローラ（Ｇ－ＣＲＴＣ）
●割込みコントローラ（ベクタ設定）
●ＹＭ－２２０３
●テキスト画面コントローラ（ＣＲＴＣ）

が，この方法を採っています。もちろん，この場合にも，各レジスタのビット毎の説明は詳細に書いておきましたが，その意味はレジスタを選択するポートに書込まれた値によって変化することになります。

この他に、知っていると便利なことを簡単に触れておきます。

表中には，ポート番号の抜けているものがありますが，それらは将来的使われる可能性がありますので，ボードを自作する人は，使用してはいけません。80_H～$9F_H$も同様です。ユーザーには，00_H～$7F_H$が開放されています（A_7～A_0がその範囲ならA_{15}～A_8は自由に使って良い）。

表中では，ＭＺ－2500用以外のオプション群，つまり，ＭＺ－80ＢやＭＺ－2000用の周辺装置でそれらのモードでのみ使用できるものについては，簡略化または省略しています。

ＣＲＴＣ等はＭＺ－2500・2000・80Ｂの各モードでは働きが違います。(ＭＺ－2500）とか，(ＭＺ－2000）とあるものは，それぞれのモードでの意味を示しています。

あまり一般的ではありませんが，これらのＩ／Ｏポートは，ＢＡＳＩＣでアクセスすることも可能です。A_{15}～A_8を使用しているものも，ＯＵＴ　＆Ｈ０１ＣＣ，０　のように，拡張ポートの方を上位に置いて使えます。ＦＯＲ～ＮＥＸＴでレジスタの内容を次々と書込み・読出しを行う時は，

```
１０　ＦＯＲ　Ａ＝０　ＴＯ　１５
２０　ＰＲＩＮＴ　ＨＥＸ＄（ＩＮＰ（Ａ＊２５６＋＆ＨＣＣ））
３０　ＮＥＸＴ
```

のように＊ 256を付けるのをお忘れなく。また，メモリマッピングレジスタやキーボードのポートのように，ＢＡＳＩＣプログラム実行中に，インタプリタが書換えを行っているものがありますので，このような場合はマシン語でアクセスするしかありません。

最後に，画面の制御が，とても複雑ですので，カスタムＬＳＩ内部まで立ち入った概念図（16色モード時）を示しておきます（**第７－１図**）。本当は回路図のようにＡＮＤ・ＯＲ・ＮＯＴの記号を併用するともっと見やすくなるのですが，ハードウエアの知識が必要となりますから，独自に考案した方法で書いてみました。もちろん，あくまでもソフトウエアから制御を行うのに必要な程度に簡略化し，キャラクタジェネレータの選択ではその働きをする部分をわかり易いように移動させたりもしていますが，プログラムを作る上では充分役に立つと思います。

BASIC-M25の隠し命令 ❷

●ファイルディスクリプタの〝EMM：〟,〝ROM：〟,〝USR：〟

まずEMM：は640KバイトのRAMディスクです。このRAMディスクは１枚のボードになっており，オプションの拡張I/Oポートを購入した後でないと利用できません。また，シャープからではなく，関連するサードパーティから発売されることになりますが，本体内のグラフィックRAM等を使うRAM：に比べ，容量がかなり大きくなっています。ただ，OPTION SCREEN 4などとして，一般のRAMに転用したりすることはできません。

次にROM：は，このEMM：のRAM部をROM化したようなもので，IPL起動用のROMカード「MZ-1R12」とは異なるものです。ROM：はEMM：とは違い一般ユーザーにあまり恩恵がないかも知れませんが，固定データを大量に使うような場合，予めフロッピーディスクで開発しておき，そっくりそのままROM化して利用すると言ったことも可能です。また，ROM：とEMM：を組み合わせ，EMM：をワーク用に使用するようにすれば，前代未聞の超高速システムとして利用することもできるでしょう。

そして最後にUSR：ですが，これは特別なハードウエアを必要とせず，スーパーMZ本体だけで利用できます。そして，使い方によっては非常に便利なものです。

例えば，

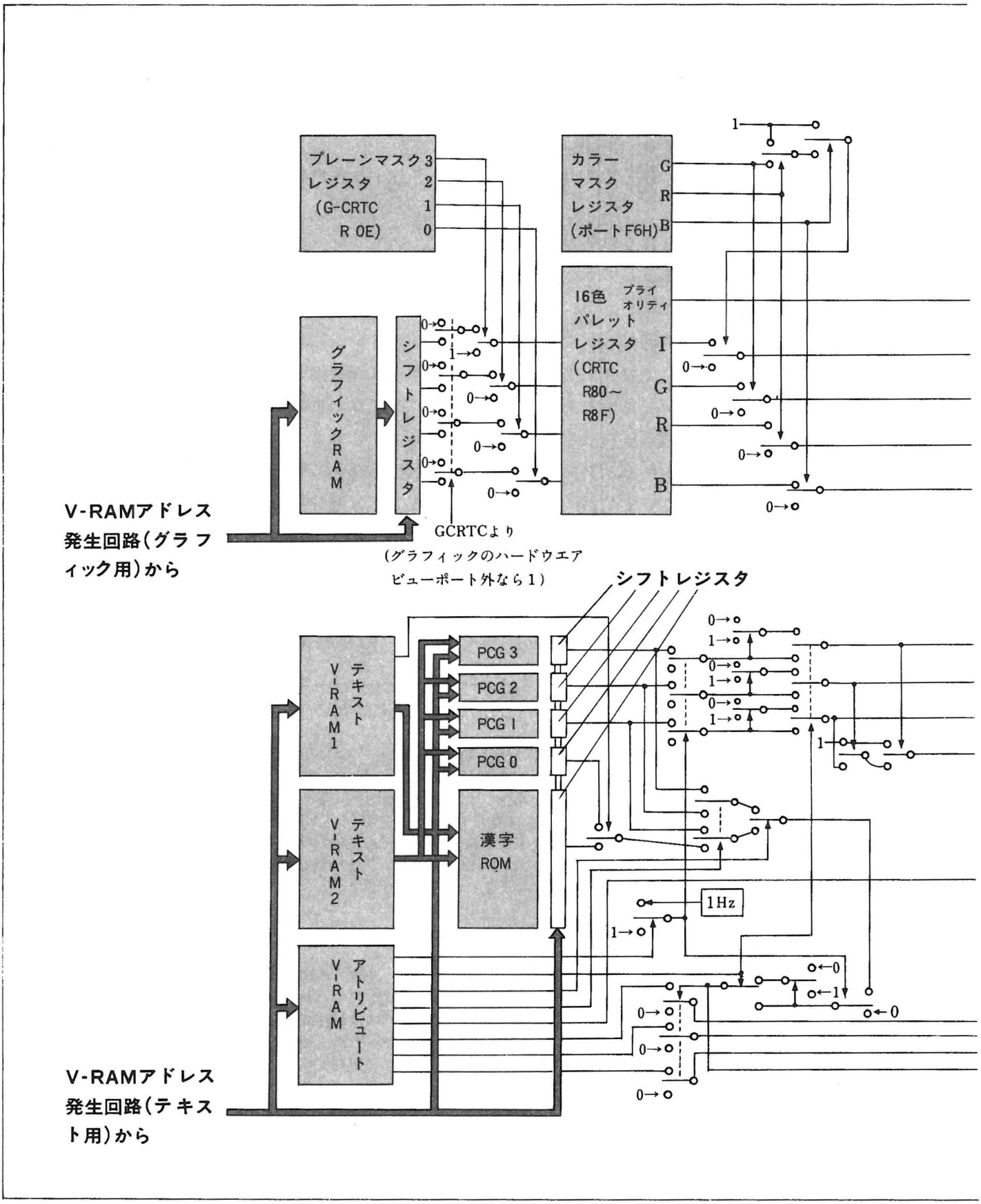
```
OPEN 〝０〟, ＃１, 〝USR：＆ＨＥ０００〟
```

または，

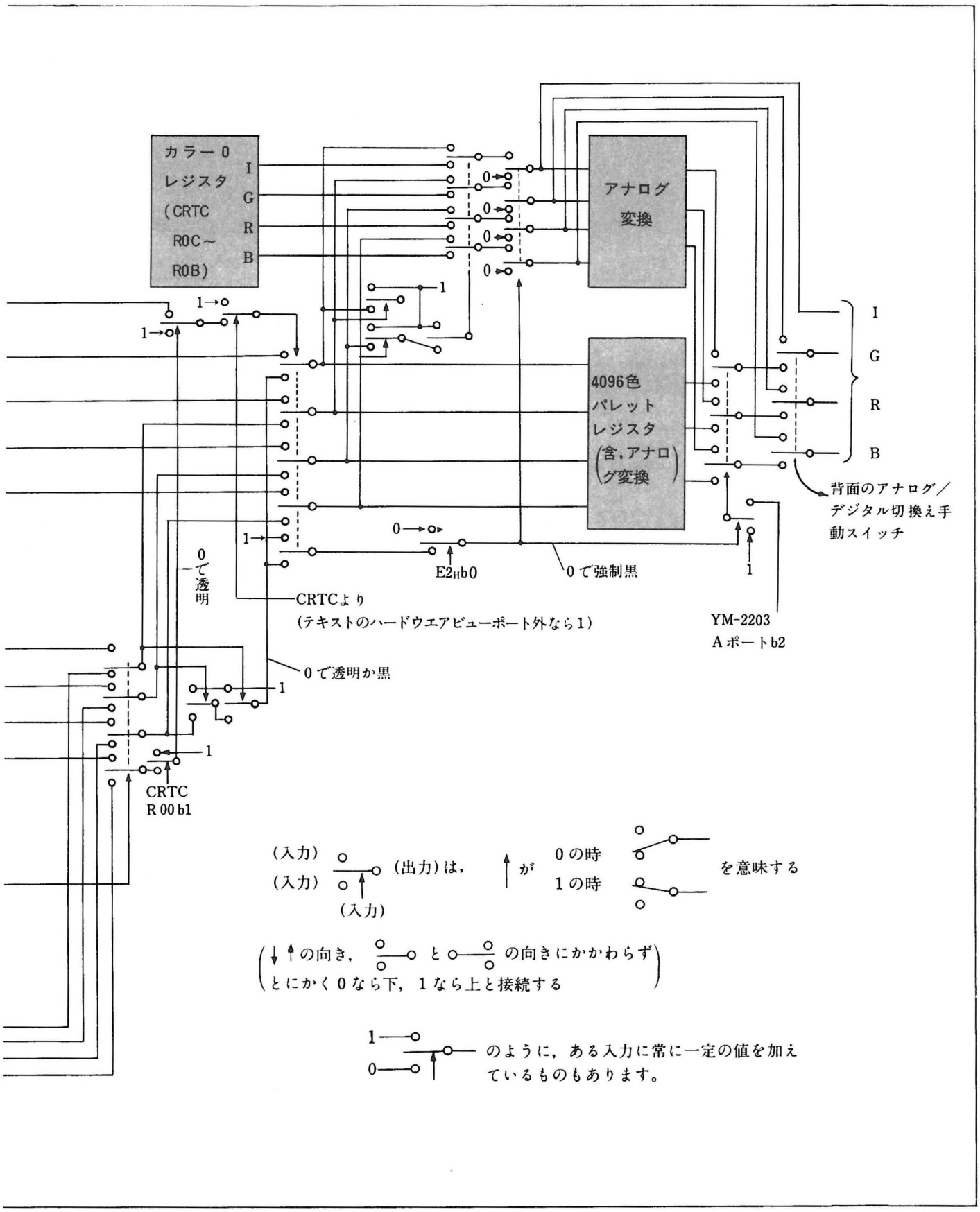
```
OPEN 〝１〟, ＃１, 〝USR：＆ＨＥ０００〟
```

としますと，以後，PRINT ＃１やINPUT ＃１等の際に，E000H番地がCALLされます。データの受け渡しは１文字ずつ行われ，Aレジスタが使われます。また，１行の末尾は$0D_H$で表わします。要するにユーザーのデバイスが作成できるというわけです。もちろん，使用するマシン語プログラムは予めCLEARで確保したエリアに入れておく必要がありますが，USR関数やCALL文を使った場合に比べると，文字例を扱うのに便利ですし，何かに出力するデータを，その時だけ画面に対して行うことでデバッグ効率を上げることもできます。

また，PRINT ＃１，CHR＄(A)；やPRINT ＃１，Ａ＄；　そして　A＝ASC(INPUT＄(1, ＃1))やＡ＄＝INPUT＄(1, ＃1)のようにすると，バイナリデータの受け渡しもできます(0DHもデータとして入力できる)。ただし，OPENするモードは〝O〟か〝I〟に限られます。なお，USR：は，CALLする番地さえ違うなら，いくつも同時にOPENすることができます。



第7-1図　ビデオ信号ブロックダイヤグラム論理図(16色モード・カラーディスプレイ接続時)


カラー0
レジスタ
(CRTC
R0C～
R0B)
I
G
R
B
アナログ
変換
4096色
パレット
レジスタ
(含,アナログ変換)
I
G
R
B
背面のアナログ／
デジタル切換え手
動スイッチ
E2Hb0
0で強制黒
0で透明
CRTCより
(テキストのハードウエアビューポート外なら1)
YM-2203
Aポートb2
0で透明か黒
CRTC
R00b1
(入力)
(入力)
(出力)は，
が
0の時
1の時
を意味する
(↓↑の向き，
と
の向きにかかわらず
とにかく0なら下，1なら上と接続する)
のように，ある入力に常に一定の値を加え
ているものもあります。

《第7－1表》I／Oポート

ポートアドレス（マシン語のOUT命令による直接指定・Cレジスタによる間接指定）	拡張ポートアドレス（Bレジスタによる間接指定ただし空欄は任意）	入出力方向	内容
A0H		入出	Z80B SIOのチャンネルA（RS-232Cの9ピン側ポート）の送受信データ入出力ポート （ポート CDHのb7によって、ポートアドレスをB0Hに変更可能）
A1H		入 読出し専用レジスタ	Z80B SIOのチャンネルAのレジスタ選択・コマンド出力・ステータス入力ポート （ポート CDHのb7によって，ポートアドレスをB1Hに変更可能） 読出しレジスタ0と書込みレジスタ0は1度の書込み・読出し操作でアクセスできるが，それ以外のレジスタは，1度書込みレジスタ0のb2～b0にレジスタ番号を書込んだ後，続けてこのポートを読出し・書込みすれば良い

レジスタ番号		b7	b6	b5	b4	b3	b2	b1	b0	
00H	0	ブレークは検出されない	送信アンダーランが起きていない	このレジスタのb7～b3が最後に変化した時の$\overline{CTS}$端子の反転状態	非同期モードではb5同様の状態の時の$\overline{SYNC}$端子の状態	b5同様の状態の時の$\overline{DCD}$端子の状態	送信バッファは空でない	割込み発生条件は成立していない	受信バッファは空である	b1はチャンネルAにのみあり，チャンネルBでは常に0。このレジスタの割込み発生条件は書込みレジスタ1のb0＝1で，b7・b6・b1、それ以外はモードによって違う。 b4の非同期以外の状態…外部同期変化時も割込み発生。モノシンク・バイシング＝同期キャラクタ検出時1・SDLC時フレーム開始フラグ検出時0・そして割込発生
	1	受信中にブレークを検出した	通常または送信アンダーランが発生				送信バッファは空である	割込み発生条件が成立している	受信バッファは空でない）読出可）	
01H	0	終了フラグ・CRC・端数が異常	フレミング・CRCエラー発生せず	オーバーランエラーしていない	パリティエラーが発生していない	SDLCモードでの端数コード 次のxxx＝A・Bでは，Aは1バイト前、Bは2バイト前の1フィールドビット。 000＝2・8　001＝0・6 010＝0・4　011＝0・8 100＝0・3　101＝0・7 110＝0・5　111＝1・8			送信バッファに何か残っている	b7＝SDLCモード専用 b6＝フレミングエラー…非同期時・CRCエラー…SDLC専用 b0＝非同期モード専用
	1	終了フラグ・CRC・端数Cが皆正常	フレミング・CRCエラー発生した	受信バッファがオーバーフローした	パリティエラーが発生した				送信バッファは完全に空	
02H		割込みベクタ（書込みレジスタ1のb2には影響せず設定したまま読出される）								チャンネルBにのみ存在する

00H	00…何もしない 01…受信CRCチェッカをリセット 10…送信CRCゼネレータをリセット 11…送信アンダーラン・EOMをリセット	000…何もしない 001…アポートを送出する 010…外部・ステータス割込みをリセットする 011…チャンネルリセット 100…受信割込み許可 101…送信割込みの保留をリセット 110…パリティ・オーバーランエラーをリセット 111…割込みからの復帰（チャンネルAに出力）	000…レジスタ0への単独出力 001…レジスタ1をアクセス 010…レジスタ2をアクセス 011…レジスタ3をアクセス 100…レジスタ4をアクセス 101…レジスタ5をアクセス 110…レジスタ6をアクセス 111…レジスタ7をアクセス 001～111では次のこのポートへの書込み・読出しでそれぞれのレジスタをアクセス			通常、書込みレジスタ0のb7～b3を設定したい時は，b2～b0は000に，逆にb2～b0を設定したい（他のレジスタを読出し・書込みしたい）時は，b7～b3は00000にするのが普通（b2～b0＝000では，単独出力というより次もレジスタ0をアクセスする，という意味と考えて良い）
01H	〔$\overline{W/RDY}$出力ピンの働き〕 000・001…フロート状態 010・011…Hレベル 100…送信$\overline{WAIT}$ 101…受信$\overline{WAIT}$	00…受信割込みを禁止する 01…最初のキャラクタのみ受信割込み発生	割込みベクタはそのまま使用	送信割込みを禁止する	外部・ステータス割込みを禁止	b4・b3＝10ではパリティエラーは特別受信割込み条件となり，11ではならない。 b2はチャンネルBでのみ設定する

		出	書込み専用レジスタ			110…送信READY 111…受信READY	10,11 すべてのキャラクタで受信割込み発生	原因に応じ割込みベクタを変化	送信割込みを許可する	外部・ステータス割込みを許可	
				02H		割込みベクタの設定。ただしレジスタのｂ１＝１ではベクタのｂ３～ｂ１を無視し、右のように原因毎に違ったベクタに変化させて発生する	xxxx000y…チャンネルBの送信要求割込み xxxx001y…チャンネルBの外部・ステータス割込み xxxx010y…チャンネルBの有効データ受信割込み xxxx011y…チャンネルBの特別条件(エラー)割込み xxxx100y…チャンネルAの送信要求割込み xxxx101y…チャンネルAの外部・ステータス割込み xxxx110y…チャンネルAの有効データ受信割込み xxxx111y…チャンネルAの特別条件(エラー)割込み				チャンネルBにのみ存在する xは任意（このレジスタに設定した値），yは通常０（設定した通りになるが，普通０を設定する）
				03H	0	受信データ長 00…５ビット 01…７ビット 10…６ビット 11…８ビット	DCD・CTSは単なる入力	ノーマル	受信CRC計算をしない	アドレスが一致しなくとも受信	同期キャラクタも受信する	受信動作を許可する	ｂ４は同期モード・ＳＤＬＣモード専用 ｂ２はＳＤＬＣモード専用（アドレスはレジスタ６で設定）
					1		DCD・CTSで送受信を許可する	ハント(検索)フェーズに入る	受信CRC計算を行う	アドレスが一致しないと受信しない	同期キャラクタも受信しない	受信動作を禁止する	
				04H	0	ボーレイト 00…×１倍速 01…×16倍速 10…×32倍速 11…×64倍速	同期モードの選択 00…８ビット同期 01…16ビット同期 10…ＳＤＬＣ 11…外部同期	ストップビット 00…同期モード 01…１ビット 10…1.5ビット 11…２ビット		パリティは奇数	パリティなし	ｂ７・ｂ６＝00は同期モード
					1					パリティは偶数	パリティあり	
				05H	0	DTRにHを出力	送信ビット長 00…5ビット以下 01…7ビット 10…6ビット 11…8ビット	TxDをスペース状態に	送信を禁止・中止する	CRCはCCITTを使用	RTSにHを出力	送信CRC計算をしない	
					1	DTRにLを出力		TxDをマーキング状態に	送信を許可・再開する	CRCは16多項式を使用	DTRにLを出力	送信CRC計算を行う	
				06H		モノシンクモード…送信同期キャラクタ 外部同期モード　…送信同期キャラクタ バイシンクモード…同期キャラクタ前バイト ＳＤＬＣモード　…チェックアドレス							
				07H		モノシンクモード…受信同期キャラクタ 外部同期モード　…使用しない バイシンクモード…同期キャラクタ後バイト ＳＤＬＣモード　…７ＥＨ（ＳＤＬＣフラブ）							
A2H		入出	Z80B ＳＩＯ のチャンネルＢ(RS-232Cの25パン側ポートまたは，YM-2203ポートＡのｂ３＝１ではマウス） の受信信データ入出力ポート （ポート　ＣＤＨのｂ７によって，ポートアドレスをＢ２Ｈに変更可能）								
A3H		入出	Z80B ＳＩＯ のチャンネルＢのレジスタ選択・コマンド出力・ステータス入力ポート （ポート　ＣＤＧのｂ７によって，ポートアドレスをＢ３Ｈに変更可能）(詳細はポートＡ１Ｈの所を参照)								
A8H	ページ	出	MZ-2500専用ROMファイルアドレス(上位4ビット→Bレジスタで指定・中位8ビット→データとして出力)								
A9H	アドレス	入	MZ-2500専用ROMファイルからの読出しデータ(アドレス下位8ビット→Bレジスタで指定)								
ACH	ページ	出	MZ-2500専用EMMアドレス(上位4ビット→Bレジスタで指定・中位8ビット→データとして出力)								
ADH	アドレス	入出	MZ-2500専用EMM読書きデータ(アドレス下位8ビット→Bレジスタで指定)								
			4096色パレットボード（ＭＺ－１Ｍ10）のパレットレジスタ（拡張ポート00Ｈ～01Ｈは実際に黒で表示されている部分のパレットレジスタとしてボード上には実在するが，実際にはこれらに何を書込んでも何の変化もない。黒はあくまでも黒で表示される）								

			b7	b6	b5	b4	b3	b2	b1	b0
	02H	出	4096色パレットボード(MZ-1M10)使用時，通常「暗い青」で出力される信号の赤成分のレベル(0000～1111)				4096色パレットボード(MZ-1M10)使用時，通常「暗い青」で出力される信号の青成分のレベル(0000～1111)			
AEH	03H	出	未使用				4096色パレットボード(MZ-1M10)使用時，通常「暗い青」で使用される信号の緑成分のレベル(0000～1111)			
	04H	出	4096色パレットボード(MZ-1M10)使用時，通常「暗い赤」で出力される信号の赤成分のレベル(0000～1111)				4096色パレットボード(MZ-1M10)使用時，通常「暗い赤」で出力する信号の青成分のレベル(0000～1111)			
	05H	出	未使用				4096色パレットボード(MZ-1M10)使用時，通常「暗い赤」で使用される信号の緑成分のレベル(0000～1111)			
	06H	出	4096色パレットボード(MZ-1M10)使用時，通常「暗い紫」で出力される信号の赤成分のレベル(0000～1111)				4096色パレットボード(MZ-1M10)使用時，通常「暗い紫」で出力される信号の青成分のレベル(0000～1111)			
	07H	出	未使用				4096色パレットボード(MZ-1M10)使用時，通常「暗い紫」で使用される信号の緑成分のレベル(0000～1111)			
	08H	出	4096色パレットボード(MZ-1M10)使用時，通常「暗い緑」で出力される信号の赤成分のレベル(0000～1111)				4096色パレットボード(MZ-1M10)使用時，通常「暗い緑」で出力される信号の青成分のレベル(0000～1111)			
	09H	出	未使用				4096色パレットボード(MZ-1M10)使用時，通常「暗い緑」で使用される信号の緑成分のレベル(0000～1111)			
	0AH	出	4096色パレットボード(MZ-1M10)使用時，通常「暗い水色」で出力される信号の赤成分のレベル(0000～1111)				4096色パレットボード(MZ-1M10)使用時，通常「暗い水色」で出力される信号の青成分のレベル(0000～1111)			
	0BH	出	未使用				4096色パレットボード(MZ-1M10)使用時，通常「暗い水色」で使用される信号の緑成分のレベル(0000～1111)			
	0CH	出	4096色パレットボード(MZ-1M10)使用時，通常「暗い黄色」で出力される信号の赤成分のレベル(0000～1111)				4096色パレットボード(MZ-1M10)使用時，通常「暗い黄色」で出力される信号の青成分のレベル(0000～1111)			
	0DH	出	未使用				4096色パレッドボード(MZ-1M10)使用時，通常「暗い黄色」で使用される信号の緑成分のレベル(0000～1111)			
	0EH	出	4096色パレットボード(MZ-1M10)使用時，通常「暗い白」で出力される信号の赤成分のレベル(0000～1111)				4096色パレットボード(MZ-1M10)使用時，通常「暗い白」で出力される信号の青成分のレベル(0000～1111)			
	0FH	出	未使用				4096色パレットボード(MZ-1M10)使用時，通常「暗い白」で使用される信号の緑成分のレベル(0000～1111)			
	10H	出	4096色パレッドボード(MZ-1M10)使用時，通常「灰色」で出力される信号の赤成分のレベル(0000～1111)				4096色パレットボード(MZ-1M10)使用時，通常「灰色」で出力される信号の青成分のレベル(0000～1111)			
	11H	出	未使用				4096色パレットボード(MZ-1M10)使用時，通常「灰色」で使用される信号の緑成分のレベル(0000～1111			
	12H	出	4096色パレットボード(MZ-1M10)使用時，通常「青」で出力される信号の赤成分のレベル(0000～1111)				4096色パレットボード(MZ-1M10)使用時，通常「青」で出力される信号の青成分のレベル(0000～1111)			
	13H	出	未使用				4096色パレットボード(MZ-1M10)使用時，通常「青」で使用される信号の緑成分のレベル(0000～1111)			
	14H	出	4096色パレットボード(MZ-1M10)使用時，通常「赤」で出力される信号の赤成分のレベル(0000～1111)				4096色パレットボード(MZ-1M10)使用時，通常「赤」で出力される信号の青成分のレベル(0000～1111)			

	15H	出	未使用	4096色パレットボード(MZ-1M10)使用時，通常「赤」で使用される信号の緑成分のレベル(0000～1111)
	16H	出	4096色パレットボード(MZ-1M10)使用時，通常「紫」で出力される信号の赤成分のレベル(0000～1111)	4096色パレットボード(MZ-1M10)使用時，通常「紫」で出力される信号の青成分のレベル(0000～1111)
	17H	出	未使用	4096色パレットボード(MZ-1M10)使用時，通常「紫」で使用される信号の緑成分のレベル(0000～1111)
	18H	出	4096色パレットボード(MZ-1M10)使用時，通常「緑」で出力される赤成分のレベル(0000～1111)	4096色パレットボード(MZ-1M10)使用時，通常「緑」で出力される信号の青成分のレベル(0000～1111)
	19H	出	未使用	4096色パレットボード(MZ-1M10)使用時，通常「緑」で使用される信号の緑成分のレベル(0000～1111)
	1AH	出	4096色パレットボード(MZ-1M10)使用時，通常「水色」で出力される信号の赤成分のレベル(0000～1111)	4096色パレットボード(MZ-1M10)使用時，通常「水色」で出力される信号の青成分のレベル(0000～1111)
	1BH	出	未使用	4096色パレットボード(MZ-1M10)使用時，通常「水色」で使用される信号の緑成分のレベル(0000～1111)
	1DH	出	4096色パレットボード(MZ-1M10)使用時，通常「黄」で出力される信号の赤成分のレベル(0000～1111)	4096色パレットボード(MZ-1M10)使用時，通常「黄」で出力される信号の青成分のレベル(0000～1111)
	1DH	出	未使用	4096色パレットボード(MZ-1M10)使用時，通常「黄」で使用される信号の緑成分のレベル(0000～1111)
	1EH	出	4096色パレットボード(MZ-1M10)使用時，通常「白」で出力される信号の赤成分のレベル(0000～1111)	4096色パレットボード(MZ-1M10)使用時，通常「白」で出力される信号の青成分のレベル(0000～1111)
	1FH	出	未使用	4096色パレットボード(MZ-1M10)使用時，通常「白」で使用される信号の緑成分のレベル(0000～1111)
B0H		入出	Z80B　SIO のチャンネルA（RS-232Cの9ピン側ポート）の送受信データ入出ポート　(詳細はポートA1Hの所を参照) (MZ-80B／2000用のものをこのポートで使用できるし，ポートCDHのb7でポートA0Hのものをそのポートに変更して利用することもできる)	
B1H		入出	Z80B　SIO のチャンネルAのレジスタ選択・コマンド出力・ステータス入力ポート (MZ-80B／2000用のものをこのポートで使用できるし，ポートCDHのb7でポートA1Hのものをこのポートに変更して利用することもできる)	
B2H		入出	Z80B　SIO のチャンネルB（MZ-2500内蔵では25ピン側。MZ-80-B等用はチャンネルA・Bともに9ピン)の送受信データ入出力ポート (MZ-80B・2000用のものをこのポートで使用できるし，ポートCDHのb7でポートA2Hのものをこのポートに変更して利用することもできる)	
B3H		入出	Z80B　SIO のチャンネルBのレジスタ選択・コマンド出力・ステータス入力ポート (詳細はポートA1Hの所を参照) (MZ-80B・2000用のものをこのポートで使用できるし，ポートCDHのb7でポートA3Hのものをこのポートに変更して利用することもできる)	
B4H		出	ポートB4Hでアクセスするメモリマッピングレジスタの番号（ポートB5Hを連続してアクセスすると，このポートでレジスタ番号を指定しなくとも自動的にレジスタ番号はインクリメント（＋1）してゆく）	

b7	b6	b5	b4	b3	b2	b1	b0
未使用					メモリマッピングレジスタの番号(000～111)		

B5H

入出

ポートB4Hで指定したメモリマッピングレジスタへの書込み・読出しデータ

b7	b6	b5	b4	b3	b2	b1	b0
未使用		実メモリブロックの番号					

レジスタ番号とZ80Bのメモリ空間の対応は以下の通り

レジスタ番号	書込み・読出しデータ
00H	Z80B CPUのアドレス空間 0000H～1FFFH でアクセスするMZ-2500の実メモリブロック番号(00H～3FH)
01H	Z80B CPUのアドレス空間 2000H～3FFFH でアクセスするMZ-2500の実メモリブロック番号(00H～3FH)
02H	Z80B CPUのアドレス空間 4000H～5FFFH でアクセスするMZ-2500の実メモリブロック番号(00H～3FH)
03H	Z80B CPUのアドレス空間 6000H～7FFFH でアクセスするMZ-2500の実メモリブロック番号(00H～3FH)
04H	Z80B CPUのアドレス空間 8000H～9FFFH でアクセスするMZ-2500の実メモリブロック番号(00H～3FH)
05H	Z80B CPUのアドレス空間 A000H～BFFFHでアクセスするMZ-2500の実メモリブロック番号(00H～3FH)
06H	Z80B CPUのアドレス空間 C000H～DFFFHでアクセスするMZ-2500の実メモリブロック番号(00H～3FH)
07H	Z80B CPUのアドレス空間 E000H～FFFFHでアクセスするMZ-2500の実メモリブロック番号(00H～3FH)

これらのレジスタへ出力するデータ(メモリブロック番号) と実メモリとの対応は以下の通り

(注・このレジスタを読出しても，b7・b6ビットの内容は不定)

レジスタへの出力データ	アクセスされる実アドレス空間	内容
00H	00000H～01FFFH	標準実装のメインRAMのページ00H (メインRAM通し番号00H)
01H	02000H～03FFFH	標準実装のメインRAMのページ01H (メインRAM通し番号01H)
02H	04000H～05FFFH	標準実装のメインRAMのページ02H (メインRAM通し番号02H)
03H	06000H～07FFFH	標準実装のメインRAMのページ03H (メインRAM通し番号03H)
04H	08000H～09FFFH	標準実装のメインRAMのページ04H (メインRAM通し番号04H)
05H	0A000H～0BFFFH	標準実装のメインRAMのページ05H (メインRAM通し番号05H)
06H	0C000H～0DFFFH	標準実装のメインRAMのページ06H (メインRAM通し番号06H)
07H	0E000H～0FFFFH	標準実装のメインRAMのページ07H (メインRAM通し番号07H)
08H	10000H～11FFFH	標準実装のメインRAMのページ08H (メインRAM通し番号08H)
09H	12000H～13FFFH	標準実装のメインRAMのページ09H (メインRAM通し番号09H)
0AH	14000H～15FFFH	標準実装のメインRAMのページ0AH (メインRAM通し番号0AH)
0BH	16000H～17FFFH	標準実装のメインRAMのページ0BH (メインRAM通し番号0BH)
0CH	18000H～19FFFH	標準実装のメインRAMのページ0CH (メインRAM通し番号0CH)
0DH	1A000H～1BFFFH	標準実装のメインRAMのページ0DH (メインRAM通し番号0DH)
0EH	1C000 H～1DFFFH	標準実装のメインRAMのページ0EH (メインRAM通し番号0EH)
0FH	1E000H～1FFFFH	標準実装のメインRAMのページ0FH (メインRAM通し番号0FH)
10H	20000H～21FFFH	オプションのメインRAMのページ00H (メインRAM通し番号10H)
11H	22000H～23FFFH	オプションのメインRAMのページ01H (メインRAM通し番号11H)
12H	24000H～25FFFH	オプションのメインRAMのページ02H (メインRAM通し番号12H)
13H	26000H～27FFFH	コプションのメインRAMのページ03H (メインRAM通し番号13H)

B5H		入出	14H	28000H～29FFFH	オプションのメインRAMのページ04H（メインRAM通し番号14H）
			15H	2A000H～2BFFFH	オプションのメインRAMのページ05H（メインRAM通し番号15H）
			16H	2C000H～2DFFF	オプションのメインRAMのページ06H（メインRAM通し番号16H）
			17H	2E000H～2FFFFH	オプションのメインRAMのページ07H（メインRAM通し番号17H）
			18H	30000H～31FFFH	オプションのメインRAMのページ08H（メインRAM通し番号18H）
			19H	32000H～33FFFH	オプションのメインRAMのページ09H（メインRAM通し番号19H）
			1AH	34000H～35FFFH	オプションのメインRAMのページ0AH（メインRAM通し番号1AH）
			1BH	36000H～37FFFH	オプションのメインRAMのページ0BH（メインRAM通し番号1BH）
			1CH	38000H～39FFFH	オプションのメインRAMのページ0CH（メインRAM通し番号1CH）
			1DH	3A000H～3BFFFH	オプションのメインRAMのページ0DH（メインRAM通し番号1DH）
			1EH	3C000H～3DFFFH	オプションのメインRAMのページ0EH（メインRAM通し番号1EH）
			1FH	3E000H～3FFFFH	オプションのメインRAMのページ0FH（メインRAM通し番号1FH）
			20H	40000H～41FFFH	標準実装のグラフィックRAMブロック0 （ページ0・プレーン0（起動時青）の前半）
			21H	42000H～43FFFH	標準実装のグラフィックRAMブロック1 （ページ0・プレーン0（起動時青）の後半）
			22H	44000H～45FFFH	標準実装のグラフィックRAMブロック2 （ページ0・プレーン1（起動時赤）の前半）
			23H	46000H～47FFFH	標準実装のグラフィックRAMブロック3 （ページ0・プレーン1（起動時赤）の後半）
			24H	48000H～49FFFH	標準実装のグラフィックRAMブロック4 （ページ0・プレーン2（起動時緑）の前半）
			25H	4A000H～4BFFFH	標準実装のグラフィックRAMブロック5 （ページ0・プレーン2（起動時緑）の後半）
			26H	4C000H～4DFFFH	標準実装のグラフィックRAMブロック6 （ページ0・プレーン3（起動時輝度の前前半）
			27H	4E000H～4FFFFH	標準実装のグラフィックRAMブロック7 （ページ0・プレーン3（起動時輝度の後前半）
			28H	50000H～51FFFH	オプションのグラフィックRAMブロック0 （ページ1・プレーン0（起動時青）の前半）
			29H	52000H～53FFFH	オプションのグラフィックRAMブロック1 （ページ1・プレーン0（起動時青）の後半）
			2AH	54000H～55FFFH	オプションのグラフィックRAMブロック2 （ページ1・プレーン1（起動時赤）の前半）
			2BH	56000H～57FFFH	オプションのグラフィックRAMブロック3 （ページ1・プレーン1（起動時赤）の後半）
			2CH	58000H～59FFFH	オプションのグラフィックRMAブロック4 （ページ1・プレーン2（起動時緑）の前半）
			2DH	5A000H～5BFFFH	オプションのグラフィックRAMブロック5 （ページ1・プレーン2（起動時緑）の後半）
			2EH	5C000H～5DFFFH	オプションのグラフィックRAMブロック6 （ページ1・プレーン3（起動時輝度）の前半）
			2FH	5E000H～5FFFFH	オプションのグラフィックRAMブロック7 （ページ1・プレーン3（起動時輝度）の後半）
			30H	60000H～61FFFH	標準実装のグラフィックRAMのリードモディファイライト用仮想ウィンドウ（前半）

B5H			31H	62000H～63FFFH	標準実装のグラフィックRAMのリードモディファイライト用仮想ウィンドウ（後半）
			32H	64000H～65FFFH	オプションのグラフィックRAMのリードモディファイライト用仮想ウィンドウ（前半）
			33H	66000H～67FFFH	オプションのグラフィックRAMのリードモディファイライト用仮想ウィンドウ（後半）
			34H	68000H～69FFFH	IPL・BIOS ROM ページ0 （ロケーション0000H～0697H・4698H～5FFFH用）
			35H	6A000H～6BFFFH	IPL・BIOS ROM ページ1 （ロケーション6000H～7FFFH用）
			36H	6C000H～6DFFFH	IPL・BIOS ROM ページ2 （ロケーション4000H～5FFFH用）
			37H	6E000H～6FFFFH	IPL・BIOS ROM ページ3 （ロケーション6000H～7FFFH用）
			38H	70000H～707FFH 70800H～70FFFH 71000H～717FFH	テキスト V－RAM 1 アトリビュート V－RAM テキスト V－RAM 2
			39H	72000H～727FFH 72800H～72FFFH 73000H～737FFH 73800H～73FFFH	ポートCFHのb7が0の時PCG 0のフォント，1の時漢字ROMフォントのウィンドウ PCG 1のフォント PCG 2のフォント PCG 3のフォント
			3AH	74000H～75FFFH	辞書ROMのウィンドウ（ポートCEHで選んだブロックが読出し可能）
			3BH	76000H～77FFFH	予約領域（大容量ROMのウィンドウ用？）
			3CH～3FH	78000H～7FFFFH	予約領域（オプションの通信ROM等用）

B7H		出	未使用 （メモリコントローラに対するモード設定）	00・01…MZ-2500モード 10 …MZ-2000モード 11 …MZ-80Bモード
B8H		入	漢字ROM（MZ-1R13つまり従来のMZシリーズ用のもの）フォントの左側のデータ	
		出	漢字ROMのアドレス（下位）	
B9H		入	漢字ROMフォントの右側のデータ	
		出	漢字ROMのアドレス（上位）	
BAH		出	漢字ROMのチップセレクト（通常 00H，読出し直前01H，読出し後00Hを出力）	
BCH		入	G－CRTCのレジスタ07Hのb4＝0の時…グラフィックRAMプレーン0（初期値＝青のプレーン）の直接読み出しデータ G－CRTCのレジスタ07Hのb4＝1の時…G－CRTCのレジスタ07Hのb1～b0で指定したプレーン（0～3のどれか）の読出しデータ	

出

	b7	b6	b5	b4	b3	b2	b1	b0
0	レジスタ番号をオートインクリメントしない	未使用		ポートBDHで書込むG－CRTC（グラフィック制御用LSI）のレジスタ番号（b7＝0ではBDHのデータを何度アクセスしてもレジスタ番号は不定だが，b7＝1では00H→01H→02H→03H→01H→02H…のように，レジスタ番号の下位2ビットが自動的にインクリメント（＋1）してゆく）				
1	レジスタ番号をオートインクリメントする							

BDH

入

G-CRTCのレジスタ07Hのb4＝0の時…グラフィックRAMプレーン1(初期値＝赤のプレーン）の直接読出しデータ
G-CRTCのレジスタ07Hのb4＝1の時…b7＝負論理の垂直同期信号を縮めたもの（約60Hzの ┐└ ）
b6～b1＝未使用
b0＝画面クリア中フラグ（0＝クリア終了・1＝クリア中)

出

グラフィックＬＳＩのレジスタへの書込みデータ。レジスタ番号に＊の付いたものは垂直ブランキング期間中に書込みを行う必要がある

レジスタ番号	b7	b6	b5	b4	b3	b2	b1	b0	
00H	プレーン0用タイリングペイントパターン								通常のペイント時はＦＦＨにしておく
01H	プレーン1用タイリングペイントパターン								
02H	プレーン2用タイリングペイントパターン								
03H	プレーン3用タイリングペイントパターン								
04H	未使用				描画したい色（パレット番号）				
05H 0	00…リプレース 01…ＰＳＥＴ 10…画面クリア 11…未使用		未使用		プレーン3をアクセスせず	プレーン2をアクセスせず	プレーン1をアクセスせず	プレーン0をアクセスせず	アクセスするプレーンとモードを設定する。b3～b0が0のプレーンは、レジスタ04Hで指定されていても変化しない
05H 1					プレーン3をアクセスする	プレーン2をアクセスする	プレーン1をアクセスする	プレーン0をアクセスする	
06H	ビットマクスデータ（リプレースモードで特定のビットだけ変化させたくない時使用)								通常は無視される。画面クリアではＦＦＨにする
07H	未使用			0	未使用		0～3のプレーン番号		複数の境界色のペイント用 / リードモディファイのグラフィックＲＡＭ読出し時の修飾情報を設定。またその時の生データはＢＣＨ～ＢＦＨまたはＢＣＨにﾗｯﾁされる
				1	読出したい（サーチしたい）色のパレットコード（その色のビットだけが1に）				単独の境界色のペイント用
08H＊	ハードウエアビューポート縦方向開始位置（下位バイト）								実際に表示されるハードウェアビューポートの設定。設定値は開始位置・終了位置ともに0～200または0～400(00H～C8H・00H～190H)
09H＊	ハードウエアビューポート縦方向開始位置（上位バイトだが，実際はｂ0のみ有効)								
0AH＊	ハードウエアビューポート縦方向終了位置＋1（下位バイト）								
0BH＊	ハードウエアビューポート縦方向終了位置＋1（上位バイトだが実際はｂ0のみ有効)								
0CH＊	ハードウエアビューポート横方向開始位置（0～50Hで指定。ｂ7は無効)								横320ドットモードでは4で，640ドットモードでは8で割った値を書込む
0DH＊	ハードウエアビューポート横方向終了位置＋1（0～50Hで指定。ｂ7は無効)								
0EH	0	0	0	1	0	1	0	1	320×200×16色　0＞1で0＋1ページ目
	0	0	0	1	0	1	0	0	320×200×16色　1＞0で0＋1ページ目
	0	0	0	1	1	1	0	1	320×200×256色　0ページ目

				0	0	0	1	0	1	1	1	640×200×16色　0ページ	
				0	0	0	0	0	0	1	1	640×400×4色　0ページ	
				1	0	0	1	0	1	0	1	320×200×16色　2＞3で2＋3ページ目	
				1	0	0	1	0	1	0	0	320×200×16色　3＞2で2＋3ページ目	
				1	0	0	1	1	1	0	1	320×200×256色　1ページ目	
				1	0	0	1	0	1	1	1	640×200×16色　1ページ目	
				1	0	0	1	0	0	1	1	640×400×16色　0ページ目	
			0FH ＊	未使用					横スクロール量				
			10H ＊	画面開始(左上)V-RAMアドレス(下位バイト)								レジスタ12～17は固定しておいてこのアドレスを変えるだけで、スクロールエリアがスクロールする	
			11H ＊	画面開始(左上)V-RAMアドレス(上位バイトだがb7は無効)									
			12H ＊	スクロールエリア終了メモリアドレス(下位バイト)								表示アドレスがこのアドレスを越えたら，自動的に0から再びスキャンしてくれる	
			13H ＊	スクロールエリア終了メモリアドレス(上位バイトだが，b7は無効)									
			14H ＊	固定表示エリア開始(左上)V-RAMアドレス(下位バイト)									
			15H ＊	固定表示エリア開始(左上)V-RAMアドレス(上位バイトだが，b7は無効)									
			16H ＊	スクロールエリアの縦のラスタ数(下位バイト)								スクロールエリアをこのラスタ数だけ表示したら，次はレジスタ14・15のアドレスから表示を始める	
			17H ＊	スクールエリアの縦ラスタ数(上位バイトだが、b0のみ有効)									
			18H ＊	I1	G1	R1	B1	I0	G0	R0	B0	320×200×256色モード	表示したいページまたは面（プレーン）のビットを1にする
				0	0	0	0	0	0	1ページ	0ページ	640×400×4色モード	
				0	0	0	0	面3	面2	面1	面0	640×200×16色モード	
				面3	面2	面1	面0	面3	面2	面1	面0	320×200×16色モード	
				0	0	0	0	0	1	1	1	MZ-2000シミュレートモード	
				0	0	1	1	0	0	0	0	MZ-80Bシミュレートモード	
BEH		入	グラフィックプレーン2(初期値緑)の直接読出しデータまたは未使用										
BFH		入	グラフィックプレーン3(初期値輝度)の直接読出しデータまたは未使用										
C0H		入	Z80A PIOデータポート(MZ-8BI04つまりGPIBインターフェース)（ハンドシェーク入力）										
C1H		入出	Z80A PIOデータポート(b7・b4～b0＝出力／b6・b5＝入力)										
C2H		出	Z80A PIOコントロールポート										

C3H		出	Z80A PIO コントロールポート
C4H		入出	GPIB 入出力データポート
C5H		出	GPIB 制御出力

C6H　出

割込みコントローラに対する設定

	b7	b6	b5	b4	b3	b2	b1	b0
0	0000…何もしない 1000…C7Hへの出力データを垂直同期割込みベクタとする 0100…C7Hへの出力データをインターバルタイマ割込みベクタとする 0010…C7Hへの出力データをプリンタ送信要求割込みベクタとする 0001…C7Hへの出力データをカレンダクロック割込みベクタとする (これ以外のデータのセットは禁止)				垂直同期割込み（約60Hz）を禁止・中止する	インターバルタイマ割込みを禁止・中止する	プリンタ送信要求割込みを禁止・中止する	カレンダクロック割込みを禁止・中止する
1					垂直同期割込み（約60Hz）を許可・再開する	インターバルタイマ割込みを許可・再開する	プリンタ送信要求割込みを許可・再開する	カレンダクロック割込みを許可・再開する

C7H		出	垂直同期・インターバルタイマ・プリンタ送信要求・カレンダクロックの割込みベクタ(ポートC6Hでいずれかを指定する)のセット
C8H		出	サウンドLSI(YM-2203)のレジスタ選択

C9H

ポートC8Hで選択したYM-2203のレジスタへのデータ書込み・読出しデータ
(レジスタ00H～0FHまでは書込んだ値等を読出すことができるが，21H～B2Hは書込み専用で，読出しても無意味な値しか得られない)

レジスタ番号	b7	b6	b5	b4	b3	b2	b1	b0	
00H	チャンネルAトーン分周比　下位8ビット								レジスタに書込む分周比と発生周波数との関係は，トーンA～C・ノイズ共通で 周波数＝250KHz/n/分周比 ただしnはレジスタ2DH～2FHで指定したSSG部用プリスケール値で，1・2・4のいずれか (BASIC使用時は通常2)
01H	未使用				チャンネルAトーン分周比 上位4ビット				
02H	チャンネルBトーン分周比　下位8ビット								
03H	未使用				チャンネルBトーン分周比 上位4ビット				
04H	チャンネルCトーン分周比　下位8ビット								
05H	未使用				チャンネルCトーン分周比 上位4ビット				
06H	未使用			ノイズ分周比(チャンネルA～C共通)					発生したい中心周波数の分周比を指定
07H 0	ポートBは入力ポートとして使用	ポートAは入力ポートとして使用	チャンネルCにノイズを出力する	チャンネルBにノイズを出力する	チャンネルAにノイズを出力する	チャンネルCにトーンを出力する	チャンネルBにトーンを出力する	チャンネルAにトーンを出力する	汎用入出力ポートAは出力，ポートBは入力と指定すること (b7＝0・b6＝1とする)
07H 1	ポートBは出力ポートとして使用	ポートAは出力ポートとして使用	チャンネルCにノイズを出力しない	チャンネルBにノイズを出力しない	チャンネルAにノイズを出力しない	チャンネルCにトーンを出力しない	チャンネルBにトーンを出力しない	チャンネルAにトーンを出力しない	
08H	未使用			0	チャンネルA音量				音量は0000が最小，1111が最大
				1	未使用（音量はエンベロープで決まる）				
09H	未使用			0	チャンネルB音量				
				1	未使用（音量はエンベロープで決まる）				

SSG音源部

C9H

0AH	未使用				0	チャンネルC音量				
					1	未使用（音量はエンベロープで決まる）				
0BH	エンベロープ周期(分周比)　下位8ビット									エンベロープ周期(Hz)＝15625 Hz/n/分周比nはレジスタ0～6と同じ
0CH	エンベロープ周期(分周比)　上位8ビット									
0DH	未使用					エンベロープ周期 0000～0011 1001 ＼ 0100～0111 1111 ／ 1000 … ＼＼＼＼ 1010 … ＼／＼／ 1011 … ＼＿ 1100 … ／／／／ 1101 … ／＿ 1110 … ／＼／＼				＼や／が1サイクルで，1秒間にこれを何回繰り返すかをレジスタ0BH～0CHで指定する（左図はいずれも8サイクル分です）
0EH（出）	0	ノーマル	音声合成LSI内部ROMのデータを使用	Z80Bと音声合成LSIとのハンシェーク信号（Z80B→音声合成LSI）	音声合成LSIへの転送データ	SIOチャンネルBはRS-232Cに使用	4096色パレットカードを使用	外部ディスクは3・4基目とする	RS-232CチャンネルAの$\overline{\text{READY}}$端子に出力するデータ	汎用入出力ポートA（入力で使用している）
	1	音声合成ボードをリセットする	音声合成LSI外付けROMのデータ使用			SIOチャンネルBはマウスに使用	4096色パレットカードを無視	外部ディスクは1・2基目とする		
0FH（入）	0	音声合成LSIからのアクノリッジ信号	CRTスイッチはH	01…MZ-80Bモード 10…MZ-2000モード 11…MZ-2500モード		カレンダクロック$\overline{\text{アラーム}}$(0で合致)・パルス出力	RS-232CチャンネルBからの$\overline{\text{CD}}$信号入力	RS-232CチャンネルAからの$\overline{\text{CI}}$信号入力	RS-232CチャンネルAからの$\overline{\text{CD}}$信号入力	汎用入出力ポートB（出力で使用している）b6～b4は，本体前面パネル下部にあるカンガルーポケット内スイッチの状態
	1		CRTスイッチはL							
21H	テストポート(b＝1にすると，b4・b3によって4通りのテスト音が出る。b7～b5・b2・b0は未使用。通常は00Hを入れておく)									YM-2203のメーカー（日本楽器製造㈱）の動作確認用
24H	タイマーAの下位8ビット									0.006×mミリ秒毎にカウントするタイマー（mはOPN部用プリスケール値）
25H	未使用							タイマーAの上位2ビット		
26H	タイマーB									0.096×mミリ秒毎のカウンタ
27H	0	チャンネル3モード 00…通常 01…効果音モード 10…音声合成モード 11…未使用		Bフラグをリセットしない	Aフラグをリセットしない	タイマーBフラグ使用せず	タイマーAフラグ使用せず	タイマーBを停止する	タイマーAを停止する	ここで言うフラグとは，タイマーをカウントアップ(＋1)していって1023(タイマーA)または255(タイマーB)を越えた時にセットされるもの
	1			Bフラグをリセットする	Aフラグをリセットする	タイマーBフラグ使用する	タイマーAフラグ使用する	タイマーBを始動する	タイマーAを始動する	

（AY-3-8910コンパチ）

C9H　出

28H	0	4スロットKEY－OFF	3スロットKEY－OFF	2スロットKEY－OFF	1スロットKEY－OFF	未使用	b7～b4の対象 00…チャンネル1 01…チャンネル2 10…チャンネル3 11…未使用	
	1	4スロットKEY－ON	3スロットKEY－ON	2スロットKEY－ON	1スロットKEY－ON			
2DH	プリスケール値をSSG部＝4・OPN部＝6に設定する							出力データはダミーであり，意味を持たない。2FHのモード→2EHのモードへの変更は，一度2DHのモードを通して行う。つまり2FH→2EHとせず2FH→2DH→2EHとする
2EH	プリスケール値をSSG部＝3・OPN部＝2に設定する							
2FH	プリスケール値をSSG部＝1・OPN部＝2に設定する(ただしこのモードはMZ-2500の場合YM-2203の仕様違反となり，正式には動作保証されない)							

30H	未使用	DETUNE(デチューン) (F-Numberに対するわずかな周波数のズレで，これを設定すると，音に「うねり」がかかった感じになり，厚みのある音にすることができる)	MULIPLE(マルチプル) (F-NumberとDETUNEで設定した周波数の倍率を設定する) 0000…0.5倍　1000…8倍 0001…1倍　1001…9倍 0010…2倍　1010…10倍 0011…3倍　1011…11倍 0100…4倍　1100…12倍 0101…5倍　1101…13倍 0110…6倍　1110…14倍 0111…7倍　1111…15倍	チャンネル1・スロット1への設定
31H				チャンネル2・スロット1への設定
32H				チャンネル3・スロット1への設定
34H				チャンネル1・スロット3への設定
35H				チャンネル2・スロット3への設定
36H				チャンネル3・スロット3への設定
38H				チャンネル1・スロット2への設定
39H				チャンネル2・スロット2への設定
3AH				チャンネル3・スロット2への設定
3CH				チャンネル1・スロット4への設定
3DH				チャンネル2・スロット4への設定
3EH				チャンネル3・スロット4への設定
40H	未使用	Total Level (トータルレベル) (エンベロープジェネレータの出力を減少させる) b6＝1…48dB減衰 b5＝1…24dB減衰 b4＝1…12dB減衰 b3＝1…6dB減衰 b2＝1…3dB減衰 b1＝1…1.5dB減衰 b0＝1…0.75dB減衰 0000000…0dB減衰 0000001…0.75dB減衰 0000010…1.5dB減衰 0000011…2.25dB減衰 0000100…3dB減衰 0000101…3.75dB減衰 (中略) 1111111…92.25dB減衰		チャンネル1・スロット1への設定
41H				チャンネル2・スロット1への設定
42H				チャンネル3・スロット1への設定
44H				チャンネル1・スロット3への設定
45H				チャンネル2・スロット3への設定
46H				チャンネル3・スロット3への設定
48H				チャンネル1・スロット2への設定
49H				チャンネル2・スロット2への設定
4AH				チャンネル3・スロット2への設定
4CH				チャンネル1・スロット4への設定
4DH				チャンネル2・スロット4への設定
4EH				チャンネル3・スロット4への設定
50H	Key Scale (キースケール)		Attack Rate (アタックレイト)	チャンネル1・スロット1への設定
51H				チャンネル2・スロット1への設定
52H				チャンネル3・スロット1への設定
54H				チャンネル1・スロット3への設定

C9H	出	55H	（エンベロープレイトを音程によって変化させる）	未使用	（音の立ち上がり時間）	チャンネル2・スロット3への設定	FM音源部（YM-2203オリジナル）
		56H				チャンネル3・スロット3への設定	
		58H				チャンネル1・スロット2への設定	
		59H				チャンネル2・スロット2への設定	
		5AH				チャンネル3・スロット2への設定	
		5CH				チャンネル1・スロット4への設定	
		5DH				チャンネル2・スロット4への設定	
		5EH				チャンネル3・スロット4への設定	
		60H	未使用		Decay Rate（ディケイアドレス） （Decay時の減衰時間）	チャンネル1・スロット1への設定	
		61H				チャンネル2・スロット1への設定	
		62H				チャンネル3・スロット1への設定	
		64H				チャンネル1・スロット3への設定	
		65H				チャンネル2・スロット3への設定	
		66H				チャンネル3・スロット3への設定	
		68H				チャンネル1・スロット2への設定	
		69H				チャンネル2・スロット2への設定	
		6AH				チャンネル3・スロット2への設定	
		6CH				チャンネル1・スロット4への設定	
		6DH				チャンネル2・スロット4への設定	
		6EH				チャンネル3・スロット4への設定	
		70H	未使用		Sustain Rate（サステインレイト） （Sustain時の減衰時間）	チャンネル1・スロット1への設定	
		71H				チャンネル2・スロット1への設定	
		72H				チャンネル3・スロット1への設定	
		74H				チャンネル1・スロット3への設定	
		75H				チャンネル2・スロット3への設定	
		76H				チャンネル3・スロット3への設定	
		78H				チャンネル1・スロット2への設定	
		79H				チャンネル2・スロット2への設定	
		7AH				チャンネル3・スロット2への設定	
		7CH				チャンネル1・スロット4への設定	
		7DH				チャンネル2・スロット4への設定	
		7EH				チャンネル3・スロット4への設定	
		80H	Sustain Level サステインレベル（サステイン減衰量） 0000… 0 dB　1000…24dB 0001… 3 dB　1001…27dB		Release Rate （リリースレイト）	チャンネル1・スロット1への設定	
		81H				チャンネル2・スロット1への設定	
		82H				チャンネル3・スロット1への設定	
		84H				チャンネル1・スロット3への設定	
		85H				チャンネル2・スロット3への設定	

C9H	出	86H	0010…6dB 1010…30dB 0011…9dB 1011…33dB 0100…12dB 1100…36dB 0101…15dB 1101…39dB 0110…18dB 1110…42dB 0111…21dB 1111…93dB		(リリース時の減衰時間)	チャンネル3・スロット3への設定
		88H				チャンネル1・スロット2への設定
		89H				チャンネル2・スロット2への設定
		8AH				チャンネル3・スロット2への設定
		8CH				チャンネル1・スロット4への設定
		8DH				チャンネル2・スロット4への設定
		8EH				チャンネル3・スロット4への設定
		90H	未使用		SSG-Type Envelop Control (SSG部とコンパチブルなエンベロープ制御) 1000… ⋀⋀⋀⋀ 1001… ╲___ 1010… ⋀╲⋀╲ 1011… ╲⋀___ 1100… ⋀⋀⋀⋀ 1101… ╱___ 1110… ⋀╱⋀╱ 1111… ╱___	チャンネル1・スロット1への設定
		91H				チャンネル2・スロット1への設定
		92H				チャンネル3・スロット1への設定
		94H				チャンネル1・スロット3への設定
		95H				チャンネル2・スロット3への設定
		96H				チャンネル3・スロット3への設定
		98H				チャンネル1・スロット2への設定
		99H				チャンネル2・スロット2への設定
		9AH				チャンネル3・スロット2への設定
		9CH				チャンネル1・スロット4への設定
		9DH				チャンネル2・スロット4への設定
		9EH				チャンネル3・スロット4への設定
		A0H	レジスタA2H～A6Hで指定したブロックへのF-Number 1 (下位8ビット)			チャンネル1への設定
		A1H				チャンネル2への設定
		A2H				チャンネル3への設定，効果音・音声合成モード時のチャンネル3・スロット4への設定
		A4H	未使用	データを書込みたいブロックの番号	F-Number 2 (音程上位2ビット)	チャンネル1への設定
		A5H				チャンネル2への設定
		A6H				チャンネル3への設定・効果音・音声合成モード時のチャンネル3・スロット4への設定
		A8H	レジスタACH～AEHで指定したブロックへのF-Number 1 (下位8ビット) (チャンネル3を効果音・音声合成モードで使用したい時に使う)			チャンネル3・スロット3への設定
		A9H				チャンネル3・スロット1への設定
		AAH				チャンネル3・スロット2への設定
		ACH	未使用	データを書込みたいブロックの番号	F-Number 2 (音程上位2ビット)	チャンネル3・スロット3への設定
		ADH				チャンネル3・スロット1への設定
		AEH				チャンネル3・スロット2への設定
		B0H	未使用	000…変調なし 001～111…2^値/32×πの変調をかける	オペレータセルの結合方法 (別図参照)	チャンネル1(スロット1)への設定
		B1H				チャンネル2(スロット1)への設定
		B2H				チャンネル3(スロット1)への設定

CAH			ボイスコミュニケーションインターフェスボード(MZ-1E28)のレジスタ(データ入力・スタンバイモード制御)							
			b7	b6	b5	b4	b3	b2	b1	b0
		入 0	普通に電源が入った	未使用	音声回線接続中	b3～b0は有効	モデムホンのコマンダーからのデータ（負論理）			
		入 1	電話の着信により電源が入った		音声回線非接続中	b3～b0は無効				
		出 0	未使用			正論理パルス(⊓)で電源が切れる	未使用			
		出 1								
CCH			カレンダクロック(RP−5C15)のレジスタ。バンクビットに0を書込むとバンク0が，1を書込むと，バンク1がアクセスできる。バンクビットは拡張ポート0DHのb0にある。							
				b7	b6	b5	b4	b3	b2	b1	b0
	00H	入 出	バンク＝0	未使用				1秒(0000～1001)			
			バンク＝1	未使用				クロック端子への出力選択 000…ハイインピーダンス 001…16384Hz　101…1Hz 010…1024Hz　110…1／60Hz 011…128Hz　111…常にL(0) 100…16Hz			
	01H		バンク＝0	未使用				10秒(000～101)			
			バンク＝1 0	未使用							何もしない
			バンク＝1 1								秒を0クリア。ただし30～59秒は分を＋1
	02H		バンク＝0	未使用				1分(0000～1001)			
			バンク＝1	未使用				アラーム1分(0000～1001)			
	03H		バンク＝0	未使用				10分(000～101)			
			バンク＝1	未使用				アラーム10分(000～101)			
	04H		バンク＝0	未使用				1時(0000～1001)			
			バンク＝1	未使用				アラーム1時(0000～1001)			
	05H		バンク＝0	未使用　(12時間制ではb1＝0…午前・1…午後，b0＝10時の桁(0～1)となる)						10時(00～10)	
			バンク＝1	未使用　(12時間制ではb1＝0…午前・1…午後，b0＝アラーム10時の桁(0～1)となる)						アラーム10時(00～10)	
	06H		バンク＝0	未使用				曜日(0＝日，1＝月，2＝火，3＝水…6＝土)			
			バンク＝1	未使用				アラーム曜日(000～110)			
	07H		バンク＝0	未使用				1日(0000～1001)			
			バンク＝1	未使用				アラーム1日(0000～1001)			
	08H		バンク＝0	未使用						10日(00～11)	
			バンク＝1	未使用						アラーム10日(00～11)	
	09H		バンク＝0	未使用				1月(0000～1001)			
			バンク＝1	未使用							
	0AH		バンク＝0	未使用							10月(0～1)
			バンク＝1	未使用							12時間制 24時間制

	0BH		バンク＝0	未使用		1年（0000～1001）			
			バンク＝1	未使用				閏年カウンタ（00＝閏年）	
	0CH		バンク＝0	未使用		10年（0000～1001）			
			バンク＝0	未使用					
	0DH		バンク共通 0	未使用		時計動作禁止	アラーム禁止	未使用	バンク0を選択
			バンク共通 1			時計動作許可	アラーム許可		バンク1を選択
	0EH		バンク共通 0	未使用		1年の桁には10月からの繰り上がりデータを追加する	1日・曜日の桁には10時からの繰り上がりデータを追加する	1分の桁には10秒からの繰り上がりデータを追加する	1秒の桁には1Hzを追加する（通常b3～b0は0で使用）
			バンク共通 1			1年の桁に16384Hzを追加し，テストを行う	1日・曜日の桁に16384Hzを追加し，テストを行う	1分の桁に16384Hzを追加し，テストを行う	1秒の桁に16384Hzを追加し，テストを行う
	0FH		バンク共通 0	未使用（b3＝b2＝0では1Hzと16HzのANDが出力される。つまり0.5秒「0」，次の0.5秒は16Hzのパルスが8個）		アラーム非一致時，アラーム端子から1Hzのパルスを出力	アラーム非一致時，アラーム端子から16Hzのパルスを出力	通常動作	通常動作
			バンク共通 1			通常動作	通常動作	秒未満の内部カウンタをクリア	アラームをすべてクリアする

CDH　出

内蔵Z80B　SIO制御

b7	b6	b5	b4	b3	b2	b1	b0

0	内蔵Z80BSIOはA0H～B3Hに置く	未使用	チャンネルBのボーレイト 000…19200ボー　100…1200ボー 001… 9600ボー　101… 600ボー 010… 4800ボー　110… 300ボー 011… 2400ボー　111… 150ボー チャンネルAのボーレイト 000…19200ボー　100…1200ボー 001… 9600ボー　101… 600ボー 010… 4800ボー　110… 300ボ 011… 2400ボー　111… 150ボー
1	内蔵Z80BSIOはB0H～B3Hに置く		

CEH　出

辞書ROMバンク切り換え

b7	b6	b5	b4	b3	b2	b1	b0
未使用		辞書ROMバンク（マップ3AHで参照するROMの上位6ビット（A18～A13））。ただし，MZ－1R28は256Kバイトなのでa18（このポートのb5）＝0とすること（将来性を考え空間は512K分確保されている）					

CFH　出

マップ39Hの0000H～07FFHで参照するキャラクタジェネレータの選択（0800H～1FFFHは常にPCG　1～3）（b7＝0の時はb6～b0は無効）

b7	b6	b5	b4	b3	b2	b1	b0

0	PCG　0	第一水準	漢字フォントROMバンク（上位6ビット（A16～A11）））
1	漢字ROM	第二水準	

FDC（MB－8876A）のコマンド／ステータスレジスタ（負論理）。以下のデータはすべて筆者が予め論理を反転（全ビットの0と1を入れ換える）していますので，負論理であることを意識せずに，そのまま使用できます。

以下の欄で出て来る①～⑦は最後に出力したコマンドによって意味が異なるものを示す。①…タイプ1コマンド全部，②…リードデータ，③…ライトデータ，④…リードアドレス，⑤…リードトラック，⑥…ライトトラック，⑦…フォースインタラプト（何かのコマンドを実行中だった場合のみ）
なお何もコマンドを実行していなかった場合にフォースインタラプトを実行した場合はb7～b1は変化せず，b0＝1となる。

b7	b6	b5	b4	b3	b2	b1	b0

1	ディスクドライブが動作可能状態である	ライトプロテクトされていない。または	①⑦…ヘッドはメディアに接触せず②…	IDを照合した場合，それが合致してい	CRCエラーは発生していない（IDま	①⑦…ヘッドはトラック0にない（トラ	①⑦…インデックスホールの空いていな	FDCは何もしていない（実行中のコ

D8H

入	1	（メディアが正常に回転しインデックスホールが検出できている）	ライトプロテクトエラーが発生していない	DM検出 ③⑥…ライトフォルト不発生（本機は常にこの状態）	た。または，IDを照合しないコマンドを実行した	たは1セクタの内容をエラーせずに読出せた）	ック0は物理的に検出できる）その他…ロストデータエラー不発生	い場所にいるその他…データの読書きを要求していない	マンドがない） ⑦は常にこの状態（実行を中断させた後だから）
		常に1…	②④⑤（不検出）	②⑥	⑤⑥⑦（不検出）	⑤⑥⑦（不検出）			⑦
	0	ディスクドライブが動作可能状態ではない（メディアが入っていない，ドライブの電源が入ってない等）	ライトプロテクトされている状態（①は単なる検出，③⑥⑦はライトプロテクトエラーが発生した）	①⑦…ヘッドはメディアに接触中 ②…DDM検出 ③⑥…ライトフォルト発生（8インチのみ発生）	IDが合致しない（①…指定トラックが見つからない ②～④…指定のセクタが見つからない等）	CRCエラーが発生した（CRC…チェックサムのようなもので，データの誤りを発見する）	①⑦…ヘッドはトラック0にある その他…ロストデータエラー発生（データ転送速度が追いつけない）	①⑦…たった今，インデックスホールの位置にいる その他…データの読書きをすぐに行う必要がある	FDCは最後に与えたコマンドを実行中である（次のコマンドを送ってはならない）

出

	b7	b6	b5	b4	b3	b2	b1	b0	
タイプ1コマンド	1	1	1	1	1…ヘッド移動時にヘッドをメディアに接触させない 0…接触させる	1…トラックレジスタとメディア上のIDに記録されているトラック番号とを照合する 0…照合しない。	ヘッド移動時間 11…6mS 10…12mS 01…20mS 00…30mS		トラック0にヘッドを移動する（リストア）
	1	1	1	0					データレジスタで指定したトラックに移動する。トラックレジスタは更新される（シーク）
	1	1	0	1…トラックレジスタの値を更新する 0…更新しない					ヘッドを1トラック内・外に移動させる。方向は前回の移動と同じ（ステップ）
	1	0	1						ヘッドを1トラック内（トラック番号の大きい方）に移動させる（ステップイン）
	1	0	0						ヘッドを1トラック外（トラック番号の小さい方）に移動させる（ステップアウト）
タイプ2コマンド	0	1	1	1…単独のセクタ（セクタレジスタで指定）をアクセス 0…複数のセクタ（1ずつ自動的に増加）をアクセス	b1＝1の時は無効 b1＝0の時は，1…0面 0…1面とIDに記録されていることを確認する	1…HLD端子をHレベルにした（ヘッドをメディアに接触させた）後，すぐにHLT信号（HLDの応答信号）を見に行く 0…HLD端子をHレベルにした後15mS待ってからHLTを見に行く	1…面番号の照合を行わない 0…面番号の照合を行う（面番号はb3で指定する）	1	現在ヘッドのあるトラックから，セクタレジスタで指定されたセクタを捜し出し，そこに記録されているデータを読出す（リードデータ）
	0	1	0					1…DMを書込む（通常動作） 0…DDMを書込む	現在ヘッドのあるトラックから，セクタレジスタで指定されたセクタを捜し出し，そこにデータを記録する。DDMは他のFDCでは普通に読出せない（DDM＝デリーテッドデータマーク・DM＝データマーク）（ライトデータ）
タイプ3コマンド	0	0	1	1	1		1	1	このコマンドを送った後に最初に見つけたID（トラック・セクタ・面番号・セクタ長等が記録されている）を読出す（リードアドレス）
	0	0	0	1	1		1	1	インデックスホールの位置から，1トラック分の全データをそっくり（セクタの区切り等も考えない）読出す（リードトラック）

		出		0	0	0	0	1		1	1	インデックスホールの位置から，1トラック分の全データをそっくり書込む（ディスクのフォーマットに使う）（ライトトラック）	
			タイプ4コマンド	0	0	1	0	任意	任意	任意	0	READY端子の入力がLレベル→Hレベルに変化した時に割込みを発生	フォースインタラプト（組み合せ可）
				0	0	1	0	任意	任意	0	任意	READY端子の入力がHレベル→Lレベルに変化した時に割込みを発生	
				0	0	1	0	任意	0	任意	任意	インデックスホールの位置で割込みを発生	
				0	0	1	0	0	任意	任意	任意	無条件に割込みを発生（割込みは実行中のコマンドを中断する働きがある）	

D9H

入出

FDCのトラックレジスタ（負論理。つまり全ビット反転している。そのため，このレジスタへのデータ書込み直前・データ読み直後にCPLを行う必要がある。これはセクタレジスタでも同様）

DAH

入出

FDCのセクタレジスタ（負論理。つまり全ビット反転している）

DBH

入出

FDCのデータレジスタ（負論理。つまり全ビット反転しているのだが，純正ソフト（IPL−ROM，BASIC等）では反転せずに正論理として利用している。そのため自己読書きでは書き込み時に反転し読出し時にも反転するため結局元のデータに戻ることになるが，他のパソコンのデータは反転して読出されてしまう（書込み時も同様）。またポートDDHのサイド（面）は本来正論理の筈だが，MZ−80Bの時代に何らかの事情で負論理用のプログラムを作成してしまったことのなごりで，結局これも他のパソコンと面が逆になっている）

DCH

出

ディスクドライブの選択信号とモータ制御

	b7	b6	b5	b4	b3	b2	b1	b0
0	ディスクドライブのモータを止める	未使用				b1～b0の選択信号を無効にする（どのドライブも選択しない）	00…ドライブ1を選択 01…ドライブ2を選択 10…ドライブ3を選択 11…ドライブ4を選択	
1	ディスクドライブのモータを回わす					b1～b0の選択信号を有効にする（ドライブ1～4のどれかを選択）		

DDH

出

ディスクのサイド（面）選択

	b7	b6	b5	b4	b3	b2	b1	b0
0	未使用							物理的0面を選択（純正ソフトでは1面として使用）
1								物理的1面を選択（純正ソフトでは0として使用）

DEH

ディスクの記録方式の選択

	b7	b6	b5	b4	b3	b2	b1	b0
出 0	未使用							倍密度（MFM）記録方式を選択
出 1								単密度（FM）記録方式を選択

E0H

カセットデッキ等制御ポート（MZ-2500・2000・80Bの各モードで働きの違うビットがありますので，注意して下さい）

	b7	b6	b5	b4	b3	b2	b1	b0
出 0	(MZ-2500・2000時) 早送り・巻戻しでAPSSする。 (MZ-80B時) 無効	(MZ-2500・2000時) 巻戻し終了時自動的に再生モードにする。 (MZ-80B時) 無効	(MZ-2500・2000時) 再生・録音終了時自動的に巻戻す。 (MZ-80B時) 無効	モノクロディスプレイは画面全体を白黒反転表示して表示（カラーディスプレイは不可）	(MZ-2500・2000時) 負論理のパルス（⊓⊔）出力でカセットが停止する (MZ-80B時) 正論理のパルス（⊔⊓）出力でカセットが停止する	(MZ-2500・2000時 負論理のパルス（⊓⊔）出力でカセットが再生・録音状態になる (MZ-80B時) 正論理のパルス（⊔⊓）出力でカセットが再生・録音状態になる	(MZ-2500・2000時) 負論理のパルス（⊓⊔）出力でカセットが早送り状態になる (MZ-80B時) 0…b0＝1で巻戻しする 1…b0＝1で早送りする	(MZ-2500・2000時) 負論理のパルス（⊓⊔）出力でカセットが巻戻し状態になる (MZ-80B時) 正論理のパルス（⊔⊓）出力で早送りまたは巻戻し状態になる
出 1	(MZ-2500・2000時) 早送り・巻戻しでAPSSしない。 (MZ-80B時) 無効	(MZ-2500・2000時) 巻戻し終了時は普通に停止する。 (MZ-80B時) 無効	(MZ-2500・2000時) 再生・録音終了時普通に停止する。 (MZ-80B時) 無効	モノクロディスプレイは，画面を反転させずに，黒地に白で表示する。				

E1H

カセットデッキのセンサー等の入力・カセットデッキアナログ部制御（出力は8255からの出力ではない）

	b7	b6	b5	b4	b3	b2	b1	b0
入 0	カセット動作中に BREAK が押された（正しくはポートEAHb7のコピー）	カセットからの読出しデータ	カセットデッキにテープが入っている	カセットデッキにテープが入っていて，しかも，ツメ（消去防止用）が折れていない	カセットデッキは動作中である（モータが回転してテープが走行している）	未使用		現在垂直ブランキング期間中である
入 1	カセット動作中に BREAK が押されてない（正しくはポートEAHb7のコピー）		カセットデッキにテープが入っていない	カセットデッキにテープが入っていないまたは，ツメ（消去防止用）が折れている	カセットデッキは動作していない（モータが回転せずテープも停止している）			垂直ブランキング期間中ではない（表示期間か水平ブランキング期間である）
出 0	カセットデッキのアナログ側録音時はラインから入力	カセットデッキのアナログ側アンプを録音状態にする	未使用					
出 1	カセットデッキのアナログ側録音時はマイクから入力	カセットデッキのアナログ側アンプを再生状態にする						

カセットデッキ・システム等制御

E2H　出

	b7	b6	b5	b4	b3	b2	b1	b0
0	カセットへの書込みデータ	カセットデッキのデジタル側アンプを録音状態にする	(MZ-2500・2000時) 0…ノーマル 1…デッキ横のREW・FF・STOPキーを無効にする (MZ-80B時) 正論理のパルス（⎍⎍）でE0Hのb1を登録する	負論理のパルス（⎍⎍）でカセットデッキをEJECT状態にする（ドアを開ける）	パワーONリセットを強制発生する（IPLスイッチを押したようにIPLが起動する）	ダイレクトサウンド出力 （CPUが直接パルスを発生）	メモリを0番地からRAMの状態にして強制リセット（RESETスイッチと同等）	画面を普通に表示させる
1		カセットデッキのデジタル側アンプを再生状態にする			（ノーマル状態）		（ノーマル状態）	画面の表示をすべて禁止する（画面全体を黒にする）

E3H　出

E0H～E2Hのコントロールポート(8255のコントロールポート)(b7＝0ではビット制御, b7＝1ではA～Cポートのモード設定となる)
モード設定では10000010B (82H) を出力している (Aポート・Cポート＝出力・Bポート＝入力)

		b7	b6	b5	b4	b3	b2	b1	b0
b7＝0	0	未使用				b0でセット・リセットしたい，Cポートつまりポートe2Hの中の，ビットの位置（000～111）。ただし，そのビットが出力モードに設定されている時のみ有効。			リセットする（ビットを0にする）
	1								セットする（ビットを1にする）
b7＝1	0	00…グループA(AポートつまりポートE0HとCポート上位つまりE2Hのb7～b4）はモード0（基本的な入出力ポート） 01…グループAはモード1（単方向ハンドシェーク） 10・11…グループAはモード2（双方向ハンドシェーク)(b5～b3は未使用となる) 本機ではグループA・Bともモード0		Aポートは出力ポートとして使用（本機ではこの状態で使用）	Cポート上位4ビットは出力ポートとして使用（本機ではこの状態で使用）	グループB（Bポートつまりポートe1HとCポート下位（b3～b0））はモード0で使用	Bポートは出力ポートとして使用	Cポート下位4ビットは出力ポートとして使用（本機ではこの状態で使用）
	1			Aポートは入力ポートとして使用	Cポート上位4ビットは入力ポートとして使用	グループBはモード1で使用（グループBにはモード2はない）	Bポートは入力ポートとして使用（本機ではこの状態で使用）	Cポート下位4ビットは入力ポートとして使用	

E4H　入出　インターバルタイマ(8253)のカウンタ0（カウンタ0の出力はカウンタ1の入力と割込みコントローラへ接続されている）

E5H　入出　インターバルタイマ(8253)のカウンタ1（カウンタ1の出力はカウンタ2の入力へ接続されている）

E6H　入出　インターバルタイマ(8253)のカウンタ2（カウンタ2の出力はどこにも接続されない）

E7H　出

E4H～E6Hのコントロールポート（8253のコントロールポート）

	b7	b6	b5	b4	b3	b2	b1	b0
0	00…b5～b0の設定はカウンタ0に対して行う 01…b5～b0の設定をカウンタ1に対して行う 10…b5～b0の設定はカウンタ2に対して行う 11…指定禁止（動作保証され		00…カウンタを一時停止する 01…カウンタの上位バイトの設定を0に固定し，下位バイトのみをポートE4H～E6Hで読書きする 10…カウンタの下位バイトの設定を0に固定し，上位		000…モード0（カウント終了時に割込みを発生させる） 001…モード1（プログラマブルワンショットと言い，指定長のパルスを発生させる） 010・110…モード2（レートジェネレータと言い，プログラマブル分周器とし			カウンタはバイナリ（16桁の2進数。あるいは4桁の16進数と考えても良い）で使用する）

				ない)		バイトのみをポートE4H～E6Hで読書きする 11…カウンタの内容をポートE4H～E6Hで下位，上位バイトのの順で読書きする		使用する) 011・111…モード3（方形波ジェネレータと言い，分周比が偶数の時デューティ比は50%） 100…モード4（ソフトウエアトリガストローブと言いカウント出力時1サイクルのパルスを出力する） 101…モード5（ハードウェアトリガストローブと言いGATEの立ち上がりでモード4のように動作）			
			1								カウンタはBCD（4桁の10進数。4ビット毎に0000～1001の値を指定する）で使用する
E8H		出		旧モードでのメモリ制御・キーボード制御等							
				b7	b6	b5	b4	b3	b2	b1	b0
			0	(MZ-2500時)　無効 (MZ-2000時) 00・01…通常状態 10…メモリのC000H～FFFFHをグラフィックRAMに 11…メモリのD000H～DFFFHをテキストRAMに (MZ-80B時) 00・01…通常状態 10…メモリのD000H～DFFFHをテキスト，E000H～FFFFHをグラフィックRAMに 11…メモリの5000H～5FFFHをテキスト，6000H～7FFFHをグラフィックRAMに		テキスト画面を80桁モードに設定する	キーストローブ信号（b3～b0）を無効にする	ポートE9Hで読出すキーボードのロウ(ROW)アドレス (キースキャン用のストローブデータ)			
			1			テキスト画面を40桁モードに設定する	キーストローブ信号（b3～b0）を有効にする				
E9H		出		E8H（Z80B）PIOのAポート）の制御ポート（制御は原則としてAポート・Bポート別々に設定できる）							
				b7	b6	b5	b4	b3	b2	b1	b0	
				割込みベクタ							0	割込みベクタの設定を行う
			0	割込み動作を行わない	指定ビットのどれかがb5と同じになった時に割り込みを発生する	指定ビットのいずれかまたは全部が0になった時に割込み発生する	ノーマル	0	1	1	1	割込み制御を行う。 動作モードが，モード3，の時に b4＝1 を指定すると，次に割込みマスクワードをこのポートに続けて書込む。割込みマスクワードは00H～FFHの任意の値で，ビットが0のビットの入力は割込み発生条件として判定の対象から外される。1が指定されたビットは入力ポートの対応するビットのいずれかまたは全部が0または1になった時（条件はb6・b5で指定）に，b7＝1が設定されていたなら割込みを発生する。
			1	割込み動作を行う	指定ビットのすべてがb5と同じになった時に割込みを発生する	指定ビットのいずれかまたは全部が1になった時に割み発生する	割込み要求を解除，モード3時は割込みマスクワードが続く					

		0 1	00…モード0（全ビットとも単純な出力） 01…モード1（全ビットとも単純な入力） 10…モード2（双方向モードと言いハンドシェークを行う） 11…モード3（特定のビットを入力に，他のビットを出力にできる）	未使用	1	1	1	1	モード2には制限がある。すなわち，Aポートをモード2で使用する時は，Bポートはモード3で使用する必要がある。この逆も同様。 モード3を設定した場合（b7・b6＝11を指定した時）は続いて入出力選択ワードをこのポートに続けて出力する。そのデータが0の場合は出力となる

EAH 入

キーボードスキャンデータ入力（負論理なので，押されているキーが0，離されているキーが1として読出される）

E8Hで出力したロウアドレス	b7	b6	b5	b4	b3	b2	b1	b0
0000 (0)	F8	F7	F6	F5	F4	F3	F2	F1
0001 (1)	−	+	・	,	9	8	F10	F9
0010 (2)	7	6	5	4	3	2	1	0
0011 (3)	BREAK	→	←	↓	↑	⏎	(スペース)	TAB
0100 (4)	G キ	F ハ	E ィ イ	D シ	C ソ	B コ	A チ	? ⊙ ／ メ
0101 (5)	O ラ	N ミ	M モ	L リ	K ノ	J マ	I ニ	H ク
0110 (6)	W テ	V ヒ	U ナ	T カ	S ト	R ス	Q タ	P セ
0111 (7)	＜ ，、 ネ	＞ ・。 ル	− ロ	¥ | ー	＾ − ヘ	Z ッ ツ	Y ン	X サ
1000 (8)	， 7 ャ ヤ	& 7 ォ オ	% 6 ェ エ	$ 4 ゥ ウ	# 3 ァ ア	“ 2 フ	! 1 ヌ	0 ヲ ワ
1001 (9)		｛ [「 ム	− @ “	= − ホ	+ ; レ	* : ケ	) 9 ョ ヨ	(8 ュ ユ
1010 (A)	／	*	ESC	⇦	INST DEL	CLR HOME	COPY	｝] 」 ム
1011 (B)				CTRL	カ ナ	SHIFT	LOCK	GRAPH
1100 (C)							変換	無変換
1101 (D)							HELP	

EBH 出　EAH（Z-80B　PIOのBポート）の制御ポート（詳細はE9Hの所を参照）

ECH 入出　マークカードリーダー（MZ-80MCB）のデータポート

EDH 入出　マークカードリーダー（MZ-80MCB）のコントロールポート

ジョイスティック制御・入力データ（出力はx？001111Bつまり0FH（8FHでも可）でジョイスティック1，4FH（CFHでも可）でジョイスティック2を選択とするのが普通）

b7	b6	b5	b4	b3	b2	b1	b0

<table>
<tr><td rowspan="4">EFH</td><td rowspan="2">入</td><td>0</td><td colspan="2" rowspan="2">未使用</td><td>ジョイスティックのトリガ1が押されている</td><td>ジョイスティックのトリガ2が押されている</td><td>ジョイスティックの前方向スイッチが押されている</td><td>ジョイスティックの後方向スイッチが押されている</td><td>ジョイスティックの左方向スイッチが押されている</td><td>ジョイスティックの右方向スイッチが押されている</td></tr>
<tr><td>1</td><td>ジョイスティックのトリガ1が押されていない</td><td>ジョイスティックのトリガ2が押されていない</td><td>ジョイスティックの前方向スイッチが押されていない</td><td>ジョイスティックの後方向スイッチが押されていない</td><td>ジョイスティックの左方向スイッチが押されていない</td><td>ジョイスティックの右方向スイッチが押されていない</td></tr>
<tr><td rowspan="2">出</td><td>0</td><td rowspan="2">未使用</td><td>ジョイスティック1からデータを入力する</td><td>ジョイスティック2の4方向スイッチ使用を許可する</td><td>ジョイスティック1の4方向スイッチ使用を許可する</td><td>ジョイスティック2トリガ2を常に押していると判断</td><td>ジョイスティック2トリガ1を常に押していると判断</td><td>ジョイスティック1トリガ2を常に押していると判断</td><td>ジョイスティック1トリガ1を常に押していると判断</td></tr>
<tr><td>1</td><td>ジョイスティック2からデータを入力する</td><td>ジョイスティック2の4方向スイッチ使用を禁止する</td><td>ジョイスティック1の4方向スイッチ使用を禁止する</td><td>ジョイスティック2トリガ2は普通に入力する</td><td>ジョイスティック2トリガ1は普通に入力する</td><td>ジョイスティック1トリガ2は普通に入力する</td><td>ジョイスティック1トリガ1は普通に入力する</td></tr>
<tr><td>F0H
～
F3H</td><td></td><td>出</td><td colspan="8">インターバルタイマ（ポートE4H～E7Hの8253）のカウンタ0・1にGATE信号を与える（H→L→Hつまり┐└ というパルスを加える）。出力データはダミーであり，意味を持たない。また，ポートF0H～F3Hはいずれも同じ働きとなる（F1H～F3Hはイメージである）。GATE信号の働きは8253のモードによって異なる。
モード0・4……一瞬（端子がLレベルの間）カウントを停止する（特に使用する意味はない）
モード1・5…カウントを開始する。カウント中の時は初期値からやり直す（このモードでは8253のレジスタへの書込みではカウント開始しない）
モード2・3…初期値から，カウントをやり直す。</td></tr>
<tr><td rowspan="4">F4H
～
F7H
(MZ-2500)</td><td rowspan="4"></td><td rowspan="4">入</td><td colspan="8">CRT－Cの読出しポート。ブランキング期間のモニタ</td></tr>
<tr><td>b7</td><td>b6</td><td>b5</td><td>b4</td><td>b3</td><td>b2</td><td>b1</td><td>b0</td></tr>
<tr><td>0</td><td colspan="5" rowspan="2">未使用</td><td colspan="2">テキスト画面は水平ブランキング期間である</td><td>テキスト画面は垂直ブランキング期間である</td></tr>
<tr><td>1</td><td colspan="2">テキスト画面は水平ブランキング期間でない</td><td>テキスト画面は垂直ブランキング期間でない</td></tr>
<tr><td>F4H
(MZ-2500)</td><td></td><td>出</td><td colspan="8">テキスト・画面コントローラのレジスタ選択</td></tr>
<tr><td rowspan="3">F4H
(MZ-2000)</td><td rowspan="3"></td><td rowspan="3">出</td><td colspan="8">文字の背景色</td></tr>
<tr><td>b7</td><td>b6</td><td>b5</td><td>b4</td><td>b3</td><td>b2</td><td>b1</td><td>b0</td></tr>
<tr><td colspan="5">未使用</td><td colspan="3">カラーコード
000…黒　100…緑
001…青　101…水色
010…赤　110…黄
011…紫　111…白</td></tr>
<tr><td rowspan="4">F4H～
F7H
(MZ-80B)</td><td rowspan="4"></td><td rowspan="4">出</td><td colspan="8">グラフィック画面のページングの制御</td></tr>
<tr><td>b7</td><td>b6</td><td>b5</td><td>b4</td><td>b3</td><td>b2</td><td>b1</td><td>b0</td></tr>
<tr><td>0</td><td colspan="5" rowspan="2">未使用</td><td>グラフィックページ1を表示しない</td><td>グラフィックページ0を表示しない</td><td>バンク切換え時ページ0アクセス</td></tr>
<tr><td>1</td><td>グラフィックページ1を表示する</td><td>グラフィックページ0を表示する</td><td>バンク切換え時ページ1アクセス</td></tr>
</table>

F5H (MZ-2500)　出

グラフィックＬＳＩのレジスタへの書込みデータ

レジスタ番号		b7	b6	b5	b4	b3	b2	b1	b0	
00H	0	未使用			縦25行表示	00…40桁64色 01…40桁0ページ表示または80桁表示		ノーマル	グラフィック256色・文字64色	12行モードではｂ４＝０とするとともにポートＦ７Ｈも変える。ｂ１＝０では文字の背景部を「透明」として扱い，テキストより優先順位の低いグラフィックは表示されるが，ｂ１＝１では文字の背景部を「黒」として扱い，優先順位の低いグラフィックは表示されない
	1				縦20行表示	10…40桁1ページ表示 11…40桁0・1ページ合成		バックのグラフィック消す	グラフィック16色・文字8色	
01H		表示開始Ｖ－ＲＡＭアドレス（下位８ビット）								このレジスタを書換えるだけで，画面をスクロールアップ・ダウンさせることができる
02H		未使用					表示開始Ｖ－ＲＡＭアドレス（上位３ビット）			
03H		200ライン時…文字表示期間縦開始位置　(38～238) 400ライン時…文字表示期間縦開始位置(34～434)／2(17～217)								レジスタ３＜レジスタ５では画面の上部と下部にブランクができ，中程に表示部分ができる。その逆では画面の中程にブランクができ，上部と下部に，表示部分ができる。レジスタ３＝レジスタ５とでは画面全体がブランクになる。ブランク部分はレジスタ00Hのb1の状態にかかわらず，透明の扱いとなる
05H		200ライン時…文字非表示期間縦開始位置　(38～238) 400ライン時…文字非表示期間縦開始位置(34～434)/2(17～217)								
07H		未使用	40桁200ライン時…文字表示期間横開始位置×2　(10～90) 40桁400ライン時…文字表示期間横開始位置×2　(8～88) 80桁200ライン時…文字表示期間横開始位置　(11～91) 80桁400ライン時…文字表示期間横開始位置　(9～89)							ブランクのでき方はレジスタ３・５の場合と同じ。ブランク部はパレット０として扱われる。レジスタに指定外の値を書込んでも無効となり，表示期間が増えるようなことはない（レジスタ３・５でも同様）。なお，200ライン・400ラインとは，接続しているディスプレイやカンガルーポケット内のＣＲＴスイッチの状態を指し，グラフィックの200ドット・400ドットのモードのことではない（レジスタ３～９すべてに共通）
08H		未使用	40桁200ライン時…文字非表示期間横開始位置×2(10～90) 40桁400ライン時…文字非表示期間横開始位置×2(8～88) 80桁200ライン時…文字非表示期間横開始位置　(11～91) 80桁400ライン時…文字非表示期間横開始位置　(9～89)							
09H		未使用				200ライン時…最上行の各文字の表示開始ラスタ 400ライン時…最上行の各文字の表示開始ラスタ×2				スムーズスクロール用。25行モードでは0000Ｂ～0111Ｂ(０～７)，20行モードでは0000Ｂ～1010Ｂ(０～９)を設定
0AH		未使用		緑の最下位ビットは 00…Ｉ０（グラフィックページ０プレーン３）を採用 01…Ｉ１（グラフィックページ１プレーン３）を採用 10…常に１ 11…常に０		赤の最下位ビットは 00…Ｉ０（グラフィックページ０プレーン３）を採用 01…Ｉ１（グラフィックページ１プレーン３）を採用 10…常に１ 11…常に０		青の最下位ビットは 00…Ｉ０（グラフィックページ０プレーン３）を採用 01－Ｉ１（グラフィックページ１プレーン３）を採用 10…常に１ 11…常に０		グラフィック256色モードでは，青・赤・緑の各成分を３ビット（８階調）でデジタル→アナログ変換を行っているが，それでは全部で９ビット必要になり，１ビット足りなくなる。そこで各成分の最上位ビットはグラフィック１ページ目の通常その原色を担当しているプレーン（青はプレーン０，赤はプレーン１，緑はプレーン２）に，中位ビットは０ページの各プレーンに対応させ最下位ビットをこのレジスタで選択する。10または11を選択し，常に０または１とした成分は４階調となる

			0BH	G1	G0	R2 (R)	R1	R0	B2 (B)	B1	B0 (I)	グラフィックのパレットでカラー0（通常黒）を割当てられた部分の実表示色。256色モードではG2～B0を使って（青ならB2が最上位）512色中から，16色モードでは（ ）内を使って16色中から選択する
			0CH	未使用							G2 (G)	
			0FH	未使用			0…400ライン 1…200ライン		00…2500 01…2000固定 10…80B固定 11…2500固定			CRTCのモード設定。b1・b0は一度00以外に設定すると，IPLを再起動しない限り，他のモードに切換えることはできない
			80H	未使用		0…そのパレットよりテキストの方が優先順位が高い 1…テキストよりそのパレットの方が優先順位が高い		0000…そのパレットは黒で表示				パレット0に対する設定
			81H					0001…そのパレットは暗い青で表示				パレット1に対する設定
			82H					0010…そのパレットは暗い赤で表示				パレット2に対する設定
			83H					0011…そのパレットは暗い紫で表示				パレット3に対する設定
			84H					0100…そのパレットは暗い緑で表示				パレット4に対する設定
			85H					0101…そのパレットは暗い水色で表示				パレット5に対する設定
			86H					0110…そのパレットは暗い黄色で表示				パレット6に対する設定
			87H					0111…そのパレットは暗い白で表示				パレット7に対する設定
			88H					1000…そのパレットは灰色で表示				パレット8に対する設定
			89H					1001…そのパレットは青で表示				パレット9に対する設定
			8AH					1010…そのパレットは赤で表示				パレット10に対する設定
			8BH					1011…そのパレットは紫で表示				パレット11に対する設定
			8CH					1100…そのパレットは緑で表示				パレット12に対する設定
			8DH					1101…そのパレットは水色で表示				パレット13に対する設定
			8EH					1110…そのパレットは黄色で表示				パレット14に対する設定
			8FH					1111…そのパレットは白で表示する				パレット15に対する設定

F5H

(MZ-2000)

文字表示色・プライオリティ（優先順位）の設定

出	b7	b6	b5	b4	b3	b2	b1	b0
0	未使用				すべての文字がグラフィックに優先	文字表示色		
1					グラフィックがすべての文字に優先	000～111（0～7）で画面全体の文字が変わる。		

F6H

(MZ-2500)

(MZ-2000)

グラフィックマスク等制御（マスク回路は16色パレットレジスタの後，4096色パレットレジスタの前に存在する）

出	b7	b6	b5	b4	b3	b2	b1	b0
0	未使用				モノクロディスプレイにグラフィックも表示する	ディスプレイに緑成分の信号を出力しない	ディスプレイに赤成分の信号を出力しない	ディスプレイに青成分の信号を出力しない
1					モノクロディスプレイにはグラフィックは表示しない	ディスプレイに緑成分の信号を出力する	ディスプレイに赤成分の信号を出力する	ディスプレイに青成分の信号を出力する

<table>
<tr><td rowspan="4">F7H
(MZ-2500)</td><td rowspan="4"></td><td rowspan="4">出</td><td colspan="8">縦12行モードの制御（ｂ０＝１では各文字は縦方向半分の速度でスキャンし，縦２倍の大きさとなる。ＷＩＤＴＨ，12の時はこの状態で使用する。また，アトリビュートＶ－ＲＡＭの「上半分」「下半分」の指定が有効となる。一方，ｂ０＝０の設定を200ラインディスプレイ使用時に行うと，偶数ラインのみの表示となるため16ライン型キャラクタフォントは利用できない）</td></tr>
<tr><td>ｂ７</td><td>ｂ６</td><td>ｂ５</td><td>ｂ４</td><td>ｂ３</td><td>ｂ２</td><td>ｂ１</td><td>ｂ０</td></tr>
<tr><td>0</td><td colspan="7" rowspan="2">未使用</td><td>すべての文字を16ラインで表示</td></tr>
<tr><td>1</td><td>すべての文字を８ラインで表示</td></tr>
<tr><td rowspan="4">F7H
(MZ-2000)</td><td rowspan="4"></td><td></td><td colspan="8">グラフィックバンク切換えの制御</td></tr>
<tr><td></td><td>ｂ７</td><td>ｂ６</td><td>ｂ５</td><td>ｂ４</td><td>ｂ３</td><td>ｂ２</td><td>ｂ１</td><td>ｂ０</td></tr>
<tr><td>出</td><td colspan="6">未使用</td><td colspan="2">グラフィックのバンク切換え時，
01…青プレーン
10…赤プレーン
11…緑プレーン
をアクセスする。
これ以外はどこもアクセスしない</td></tr>
<tr><td></td><td></td><td></td><td colspan="8"></td></tr>
<tr><td>F8H</td><td></td><td>入</td><td colspan="8">外部ＲＯＭ（ＭＺ－１Ｒ12）からのデータ読出し</td></tr>
<tr><td></td><td></td><td>出</td><td colspan="8">外部ＲＯＭのアドレス上位バイト</td></tr>
<tr><td>F9H</td><td></td><td>出</td><td colspan="8">外部ＲＯＭのアドレス下位バイト</td></tr>
<tr><td>FCH</td><td></td><td>入
出</td><td colspan="8">紙テープパンチャ・リーダ（ＭＺ－８Ｉ01を使用して作成）データ入出力ポート
(パンチャへの出力・リーダからの入力)</td></tr>
<tr><td>FDH</td><td></td><td>入
出</td><td colspan="8">紙テープパンチャ・リーダからの制御信号出力・ステータス入力</td></tr>
<tr><td rowspan="5">FEH</td><td rowspan="5"></td><td rowspan="4">入</td><td colspan="8">プリンタからのステータス入力・制御信号出力</td></tr>
<tr><td></td><td>ｂ７</td><td>ｂ６</td><td>ｂ５</td><td>ｂ４</td><td>ｂ３</td><td>ｂ２</td><td>ｂ１</td><td>ｂ０</td></tr>
<tr><td>0</td><td colspan="4" rowspan="2">未使用</td><td>常にこの状態</td><td>常にこの状態</td><td>プリンタはオフライン状態（電源ＯＦＦを含む）</td><td>プリンタはビジーでない（オンラインなら転送可）</td></tr>
<tr><td>1</td><td>あり得ない</td><td>あり得ない</td><td>プリンタはオンライン（セレクト）状態</td><td>プリンタはビジー状態（データ転送不可）</td></tr>
<tr><td>出</td><td colspan="2">プリンタへのデータストローブ（⌐¬とする）</td><td>プリンタの強制リセット（⌐¬とする）</td><td colspan="5">未使用</td></tr>
<tr><td>FFH</td><td></td><td>出</td><td colspan="8">プリンタへのデータ出力ポート</td></tr>
</table>

《第7－2表》テキストV-RAM 1

		b7	b6	b5	b4	b3	b2	b1	b0
テキスト(文字) V－RAM 1 (マップ38Hの 0000H～07FFH)	0	キャラクタジェネレータのアドレス（漢字ROMまたはPCGのA10～A5） ※MZ－2500では，一般文字つまりアルファベット・数字・カタカナ・記号（KMODE 0の時…全部の文字，KMODE 1の時…キャラクタコード 00H～7FH・A0H～DFHの文字）は，漢字ROMの通常空いているエリアに割り当てて表示しています。 〈漢字ROMアドレス計算方法〉 〔JIS漢字コード XXYYの漢字の場合〕 ●XXが21H～28Hの時：（（XX－21H）＊60H＋YY－20H）＊20H（JIS第一水準非漢字） ●XXが30H～4FHの時：（（XX－30H）＊60H＋YY－20H）＊20H＋8000H（JIS第一水準漢字） ●XXが50H～74Hの時：（（XX－50H）＊60H＋YY－20H）＊20H（JIS第二水準漢字） ●XXが78H～7AHの時：（（（XX－78H）＊5EH＋YY－21H）¥3）＊20H ただし，PCGのバンク（PCG1～3のどれを割り当てるか）は （（XX－78H）※5EH＋YY－21H）MOD 3＋1 としている（BASICの利用時）						PCG（カラー・モノクロを問わない）を選択している場合にはA4＝0を意味する。漢字ROMでは，その左半分を表示する	縦12行モード（ポートF7Hのb0＝1）の時のみ有効で，漢字ROMでは上半分を表示する。PCGではA3＝0を意味する。縦25行・20行モードでは無視される
	1	〔キャラクターコードXXの一般文字（いわゆるANK）の場合〕 ●XX×10H＋6000H（JIS第一水準内） ただし，8×16ドットのフォント(400ラインディスプレイに接続時は全部の一般文字，200ラインディスプレイ接続時は縦12行モードの文字）ではその左半分を，8×8ドットのフォント(200ラインディスプレイの25・20行モード）では左上4分の1を使用する（b1・b0で指定する）						PCG（カラー・モノクロを問わない）を選択している場合にはA4＝0を意味する。漢字ROMでは，その右半分を表示する	縦12行モード（ポートF7Hのb0＝1）の時のみ有効で，漢字ROMでは下半分を表示する。PCGではA3＝1を意味する。縦25行・20行モードでは無視される

BASIC-M25の隠し命令 ❸

●LIST #n
●SEARCH #n
●FILES #n
●KEY LIST #n
●KLIST #n

これらは本来，「何かを画面に表示するもの」ですが，#nを付けるとその出力データを画面でなくOPEN "O"～で開いたファイルに対して行うことができます。また，#nの後に「,」を付けると，本来の命令用のパラメータを続けることもできます。
例えば
LIST #1，＊100－200
といった指定も可能です。

●$

16進数を表わす記号は通常&Hですが，$も全く同じように使用することができます。例えば
PRINT $FF
は255になります。

《第７－３表》アトリビュートV-RAM

		b７	b６	b５	b４	b３	b２	b１	b０
アトリビュートV−RAM	0	ブリンク（点滅表示）しない	リバース（反転）表示しない。漢字・一般文字・モノクロＰＣＧ…ドットのある部分はｂ２～ｂ０の色で、ドットのない背景の部分は透明で表示。カラーＰＣＧ…ドット毎の指定色で表示(000…透明・001…青・010…赤・011…紫・100…緑・101…水色・110…黄111…白・※)	モノクロキャラクタジェネレータの選択（ｂ３＝１の時に有効。ｂ３＝０の時は無効) 00…漢字ＲＯＭ　または　ＰＣＧ　０を表示		モノクロのキャラクタジェネレータ（漢字ＲＯＭまたはＰＣＧ　０～３のうちのいずれか）で表示する。そのうちのどれを選択するかは，ａ５・ｂ４で選択する。また，ドットのある部分の色付けのために，ｂ２～ｂ０が有効となる	ｂ６＝０の時には，ドットのある部分の色， ｂ６＝１の時には，ドットのない部分の色 ただし，ｂ３＝１の時は無効 000…黒（透明ではない） 001…青 010…赤 011…紫 100…緑 101…水色 110…黄 111…白 なお、ｂ３＝１（カラーＰＣＧ）の場合のＰＣＧ　０・ＰＣＧ　１・ＰＣＧ　２がすべて０のビットや，ｂ３＝０（モノクロＰＣＧ）の場合のｂ６＝０の部分の背景部，ｂ６＝１の部分の文字の所は透明になるが，ＣＲＴ−Ｃ（ポートＦ４Ｈ・Ｆ５Ｈ）のレジスタ00Hのｂ１＝１の場合には，透明でなく，黒の扱いになる。黒と透明は，グラフィックを使わない場合にはどちらも黒として表示されるが，テキスト画面より優先順位の低いグラフィックに何か表示している場合には，黒ではそのグラフィックは見えなくなってしまうが，透明では見ることができる		
(マップ38Hの0800H～０ＦＦＦＨ)	1	ブリンク（点滅）表示する。つまり通常の状態と，その文字のエリア全部をドットのない背景部とした状態(ｂ６の状態によって漢字・一般文字・モノクロＰＣＧでは全部透明または全部ｂ２～ｂ０の色で，カラーＰＣＧでは全部透明または全部白(透明の補色)で)とを約1Hzで繰り返す	リバース（反転)表示する。漢字・一般文字・モノクロＰＣＧ…ドットのある部分は透明で，ドットのない背景の部分はｂ２～ｂ０の色で，表示。カラーＰＣＧ…ドット毎の補色で表示(000…白・001…黄・010…水色・011…緑・100…紫・101…赤・110…青・111…透明※)	（テキストＶ－ＲＡＭ　2のｂ７で選択） 01…ＰＣＧ　１を表示 10…ＰＣＧ　２を表示 11…ＰＣＧ　３を表示		ＰＣＧ　１～３を同時に使用して，ドット毎のカラーで表示する。この状態ではｂ５・ｂ４・ｂ２～ｂ０，そして，テキストV-RAMのｂ７は，すべて無効になる			

※このアトリビュートＶ−RAMのｂ２～ｂ０ではなく，ＰＣＧ　３・２・１のビットの様子

《第7-4表》テキストV-RAM 2

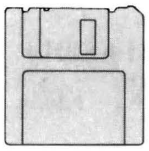
		b7	b6	b5	b4	b3	b2	b1	b0
テキストV−RAM 2（マップ38Hの1000H～17FFH）	0	漢字ROMで表示する。ただし，アトリビュートV−RAMのb3＝0によってモノクロ系のキャラクタジェネレータが選択され，同RAMのb5・b4＝00によって漢字またはPCG 0が選択されている場合のみ有効（それ以外の場合には無視される）	JIS第一水準を選択する。ただし，アトリビュートV−RAMのb3＝0かつb5・b4＝00かつテキストV−RAM 2のb7＝0によってキャラクタジェネレータに漢字ROMが選択されている場合のみ有効（それ以外の場合には無視される）	キャラクタジェネレータのアドレス （A16～A11） なお各種，キャラクタジェネレータは次の論理によって決定される （Xは任意の状態で良いことを示す）					
	1	PCG 0で表示する。ただし，アトリビュートV−RAMのb3＝0によってモノクロ系のキャラクタジェネレータが選択され，同RAMのb5・b4＝00によって漢字またはPCG 0が選択されている場合のみ有効（それ以外の場合には無視される）	JIS第二水準を選択する。ただし，アトリビュートV−RAMのb3＝0かつb5・b4＝00かつテキストV−RAM 2のb7＝0によってキャラクタジェネレータに漢字ROMが選択されていた場合のみ有効（それ以外の場合には無視される）						

アトリビュート V−RAM			テキスト V−RAM 2		選択されるキャラクタジェネレータ
b3	b5	b4	b7	b6	
0	0	0	0	X	PCG 0（モノクロ）
0	0	0	1	0	漢字ROM JIS第一水準(一般文字を含む)
0	0	0	1	1	漢字ROM JIS第二水準
0	0	1	X	X	PCG 1（モノクロ）
0	1	0	X	X	PCG 2（モノクロ）
0	1	1	X	X	PCG 3（モノクロ）
1	X	X	X	X	PCG 1＋PCG 2＋PCG 3 （ドット単位のカラー）



第8章 ソフトウエア内部解析

この章では，BASIC-M25・S25の中間コード，M25の内部解析からRST　18H・28Hによるファンクションコール，ディスクフォーマットまでソフトウエアの内部解析を紹介しています。前章同様にこれらの内容は，筆者が独自に解析したものですので，メーカーに問い合わせることは御遠慮下さい。



中間コード

BASICで作成したプログラムは，キーボードから入力したままの形でメモリ上に置かれることはありません。例えば，

```
10　FOR　A＝1　TO　10000
20　PRINT　A
30　NEXT
```

というプログラムを打ち込んでも，31H，30H，20H，46H，4FH，52H，20H，41H，3DH，31H，20H，54H，4FH，20H，31H，30H，30H，30H，30H,……と言った具合いに格納されているわけではないのです。

ではどうなっているかと言うと，FORやTOなどと言った**「予約語」**は1バイトないし2バイトの**「中間コード」**に圧縮されているのです。「FOR」ならキャラクタコードの52H，4FH，52Hではなく，8DHの1バイトだけになっています。また，TOはE0Hです。

このように「予約語」を「中間コード」に変換してメモリ上に蓄え，リストをとる時には元のものに逆変換するようにすると，メモリが節約できるだけでなく，実行時のスピードを上げることもできます。つまり，52H，4FH，52Hと続いたらFOR文であると認識するより，8DHだけでFORと判別できるため，インタプリタが速く動くのです。このため，「中間コード」はパソコンのBASIC

インタプリタシステムでは，７年前からほぼすべてのもので導入されている常套手段です。ただ，コードの割り当てはそのBASICによって独自に決められており，例えば「FOR」ならMZ-80K～1200のBASICでは87H，MZ-80B～2200・5500～6500のBASICでは80H＋8AH（２バイト），MZ-700，1500，2500，Ｘ１/turboでは8DH，68系と昔の80系のマイクロソフト社のBASICは81H，今の80系ものは82H(PC98は９EH)…などとなっています。

また，「FOR」のようにすべてのBASICにあるような超基本的なものはこの程度ですが，グラフィック命令，例えば「PAINT」などはMZ-2000～2200・5500～6500(それより前にはなかった)のＢASICでは80Ｈ＋Ｅ5Ｈ，MZ-1500・2500ではFEH＋99H，Ｘ１/turboではFEH＋86H，マイクロソフトBASICではD1H(PC88)，B3H(FM)，BFH(MSX)，FEH＋8FH(IBM・東芝・日立の16ビット機・PC-100)，Ｃ５Ｈ(PC98)などバラバラ…となっています。さらにはスーパーMZ内でもＭ25とＳ25とでは違いが見られます。例えばＤ５ＨはM25ではFILES，S25ではDIRです。

さて，そろそろ中間コードの全貌をお見せしましょう。**第8-1表**と**第8-2表**を見て下さい。**第8-1表**はBASIC-M25，**第8-2表**はBASIC-S25のものです。全部の予約語がこの中に含まれています。

また，数値等も中間コードに変換されることがわかります。順に説明して行きましょう。

まず，00Hは「１行終了」ですが，これは１行の最後に必ず付くものです。詳しくは後でもう１度触れます。

次に01H～0AHは，０～９を表わします。ただ，10以上では，これらの組合わせとなるのではありません。あくまでも Ａ＝0 などのように１桁で使われた時だけ，01H～0AHのコードになります。また，USR0～USR9でもこれが使われ，例えば「USR５」ならFFH＋CFH＋06H（3バイト）になります。

次に，0BHは行番号を表わすのに使われます。例えば「GOTO 1000」なら，80H＋0BH＋E8H＋03H（４バイト)になります。0BHの次には，行番号を16進数に変換したものが，2バイト，下位・上位の順で来るわけです。なお，10進数の1000は，16進数では03Ｅ８Ｈです。

次に，0CHは0BHの変形です。例えばGOTO 1000は80H＋0BH＋E8H＋03Hだと言いましたが，実際にこれが使われる(プログラムがRUNしてここを通過し1000行にジャンプした時)と，その場で0BH＋E8H＋03Hの部分が0CH＋××Ｈ＋△△Hに変換されるのです。△△××Hは，行番号1000がスーパーMZ のメモリの何番地から格納されているかを示すアドレスで，やはり下位・上位の順に入ります。つまり，GOTOの飛び先等を１度捜したら，２度目からは捜し直さなくても良いようになっているのです。例えば

```
10  FOR A=1 TO 10000
20  GOSUB 50
30  NEXT
40  END
50  PRINT A
60  RETURN
```

というプログラムでは，20行のGOSUBで，10000回のうちの最初の１回だけは50行を検索するものの，後の9999回は，ダイレクトに50行を呼び出すことができ，スピードアップに大きく貢献しているのです。MZ-80K～1200，80B～2200，5500～6500のBASICではこれを行わず，行番号はキ

《第8-1表》BASIC-M25　中間コード表

	0x	1x	2x	3x	4x	5x	6x	7x	8x	9x	Ax	Bx	Cx	Dx	Ex
x0	1行終了	16進数(&H)	(スペース)		@		型無変数アドレス	型無変数	GOTO	WRITE	POKE		ERASE		TO
x1	0	16進数($)							GOSUB	INPUT	OFF		ERROR		STEP
x2	1	10進整数	"				整数変数アドレス	整数変数				KEY	ELSE	OPEN	THEN
x3	2	JISコード(&J)	#				文字変数アドレス	文字変数	RUN	IF			CALL		USING
x4	3	区点コード(&K)							RETURN	DATA		LOAD	MON		
x5	4	単精度実数					単精度アドレス	単精度変数	RESTORE	READ		SAVE	GO TO	FILES	ALL
x6	5								RESUME	DIM		MERGE	GO SUB	MKDIR	TAB
x7	6		'						LIST	REM		CHAIN	DEF	RMDIR	SPC
x8	7	倍精度実数	(				倍精度アドレス	倍精度変数		END		CONSOLE		NAME	EQV
x9	8		)						DELETE	STOP	TRON	.SEARCH	WHILE	KILL	IMP
xA	9		ラベル	:					RENUM	CONT	TROFF	OUT	WEND		XOR
xB	行番号			;		[			AUTO	CLS			REPEAT		OR
xC	行番号アドレス		,						EDIT		CHDIR		UNTIL	INIT	AND
xD	ラベルアドレス					]			FOR	ON	GET			AS	NOT
xE	2進数(&B)		.						NEXT	LET		WAIT		MAX	><
xF	8進数(&O)	ブロック情報						プリンタへ	PRINT	NEW		SWAP	CLOSE		<>

	Fx	FE+8x	FE+9x	FE+Ax	FE+Bx	FE+Cx	FE+Dx	FF+8x	FF+9x	FF+Ax	FF+Bx	FF+Cx	FF+Dx	FF+Ex
x0	=<		COLOR	GLOAD	CBLOCK	.COM	RADIANS	INT	FAC	CHR$	CVD	CMT	COM	
x1	<=		PRIORITY	GSAVE	CGEN	.HELP	DEGREES	ABS	DEG	STR$	MIRROR$	SCRN$	HELP	
x2	=>		PSET	.PLAY	.CGPAT$	.MOUSE	BLOAD	SIN	FIX	HEX$	HEXCHR$	INPUT$	MOUSE	
x3	>=		PRESET				BSAVE	COS	CINT	OCT$	ERR	STRING$	PLAY	ANGLE
x4	=	PUT	LINE	LOCATE		.INTERVAL	LFO	TAN	CSNG	BIN$	ERL		INTERVAL	BASE
x5	>	SET		VERIFY		.TIME$	COPY	LOG	CDBL	MKI$	FRE	POINT	TIME$	SNG
x6	<	OPTION		CLR	MOVE	.TIME	UCASE	EXP		MKS$	POS	EOF	TIME	DBL
x7	+	RANDOMIZE	CIRCLE	CLEAR	PEN	.DATE$	LCASE	SQR		MKD$	CSRLIN	FN.	DATE$	STR
x8	−	KMODE		.STICK	POLY	.DAY$	SCR	RND	AKCNV$	SPACE$		DTL	DAY$	KANJI
x9	¥	KLIST	PAINT	CLICK	ROLL	.MID$.	FPSET	PEEK	KACNV$	INP		FPOS		
xA	MOD	SOUND	.POINT	PAUSE	SCREEN		.CMT	ATN	JIS$	ASCCHR$	LEFT$	LOC	ATTR$	
xB	/	TONE	PATTERN	WIDTH		FIELD	APSS	SGN	KTN$	ASC	RIGHT$	LOF	DEVF	
xC	*		HCOPY	COMMON	SYMBOL	LSET	TERM	LOG10	KLEN	LEN	MID$	PWD$	CGPAT$	
xD	^	BEEP		LPOUT	WINDOW	RSET	TEL	FRAC	LPOS	VAL		SYSID$	MAP	
xE	拡張命令	VOICE		BOOT	VIEW	DEVI$		PAI	STICK	CVI	INSTR	KPOS	SEARCH	
xF	関数	CFLASH	CREV	CONNECT	REPLACE	DEVO$		RAD	STRIG	CVS	INKEY$	USR.	SUM	

0BH～14H・60H～68Hは2バイト，15H・1FHは5バイト，18Hは8バイトのバイナリデータが後に続きます。
2AH，70H～78Hの後には長さが1バイトと，ラベル・変数名を表す英数字・漢字等がそのバイト数だけ続きます。
7FHはLIST・PRINT等のコードの後に付きLLIST・LPRINT等を表します。
表中には，""中の文字列，変数名でのみ現れるものは含めていません(それらはキャラクタコードのまま入ります)。
「.」で始まる語は関数にも同一の語が存在する命令です。
「.」で語わる語はリスト時その後にスペースを空けません。
なお，これらの詳細については，本文を参照して下さい。

ャラクタコードのまま(1000なら31H，30H，30H，30H)ですので，毎回「文字列→バイナリ変換・プログラムの先頭から検索し直し」を繰返すことになり，長いプログラムの後ろの方のサブルーチンをループ中で何度も呼び出すような場合には特にスピードダウンを招いていましたが，スーパーMZではこの心配は全くありません。なお，0BH→0CH変換は，GOTO・GOSUB以外にも，行番号を扱うすべてのステートメントで行われます。RESTOREやRESUME等でも適用されるわけです。

《第8-2表》BASIC-S25　中間コード表

	0x	1x	2x	3x	4x	5x	6x	7x	8x	9x	Ax	Bx	Cx	Dx	Ex
x0	1行終了	16進数(&H)	(スペース)		@		型無変数アドレス	型無変数	GOTO		POKE		ERASE	ROPEN	TO
x1	0	16進数(S)							GOSUB	INPUT	OFF		ERROR	XOPEN	STEP
x2	1	10進整数	"				整数変数アドレス	整数変数				KEY	ELSE		THEN
x3	2	JISコード(&J)	#				文字変数アドレス	文字変数	RUN	IF			CALL		USING
x4	3	区点コード(&K)							RETURN	DATA		LOAD	MON		
x5	4	単精度実数					単精度アドレス	単精度変数	RESTORE	READ		SAVE	GO TO	DIR	ALL
x6	5								RESUME	DIM		MERGE	GO SUB	MKDIR	TAB
x7	6		'						LIST	REM		CHAIN	DEF	RMDIR	SPC
x8	7	倍精度実数	(				倍精度アドレス	倍精度変数		END		CONSOLE		RENAME	EQV
x9	8		)						DELETE	STOP	TRON	.SEARCH	WHILE	KILL	IMP
xA	9		ラベル	:					RENUM	CONT	TROFF	OUT@	WEND	LOCK	XOR
xB	行番号			;					AUTO	CLS	INP@		REPEAT	UNLOCK	OR
xC	行番号アドレス		,						EDIT	FILL	CHDIR		UNTIL	INIT	AND
xD	ラベルアドレス								FOR	ON	GET		AOPEN	AS	NOT
xE	2進数(&B)		.						NEXT	LET		WAIT	WOPEN	MAX	><
xF	8進数(&O)	ブロック情報						フリンタへ	PRINT	NEW		SWAP	CLOSE		<>

	Fx	FE+8x	FE+9x	FE+Ax	FE+Bx	FE+Cx	FE+Dx	FF+8x	FF+9x	FF+Ax	FF+Bx	FF+Cx	FF+Dx	FF+Ex
x0	=<		COLOR	GLOAD	CBLOCK	.COM	RADIANS	INT	FAC	CHR$	CVD	CMT	COM	
x1	<=		PRIORITY	GSAVE	CGEN	.HELP	DEGREES	ABS	DEG	STR$	MIRROR$	CHARACTER$	HELP	
x2	=>		SET	.MUSIC	.CGPAT$	.MOUSE		SIN	FIX	HEX$	HEXCHR$	INPUT$	MOUSE	
x3	>=	CCOLOR	RESET	TEMPO				COS	CINT	OCT$	ERR	STRING$	MUSIC	ANGLE
x4	=	PUT	LINE	CURSOR		.INTERVAL	LFO	TAN	CSNG	BIN$	ERL		INTERVAL	BASE
x5	>		BLINE	VERIFY		.TI$	COPY	LN	CDBL	MKI$	FRE	POINT	TI$	SNG
x6	<	OPTION		CLR	MOVE	.TIME	UCASE	EXP		MKS$	CSRH	EOF	TIME	DBL
x7	+	RANDOMIZE	CIRCLE	LIMIT	PEN	.DATE$	LCASE	SQR		MKD$	CSRV	FN.	DATE$	STR
x8	−	KMODE	BOX	.STICK	POLY	.DAY$	SCR	RND	AKCNV$	SPACE$	POSH	DTL	DAY$	KANJI
x9	¥	KLIST	PAINT	CLICK	ROLL	.MID$.	FPSET	PEEK	KACNV$		POSV	FPOS		
xA	MOD	SOUND	POSITION		SCREEN		.CMT	ATN	JIS$	ASCCHR$	LEFT$	LOC	ATTR$	
xB	/	TONE	PATTERN	WIDTH		FIELD	APSS	SGN	KTN$	ASC	RIGHT$	LOF	DEVF	
xC	*		HCOPY	COMMON	SYMBOL	LSET	TERM	LOG	KLEN	LEN	MID$	PWD$	CGPAT$	
xD	^	BEEP		LPOUT	WINDOW	RSET	TEL	FRAC	LPOS	VAL		SYSID$	MAP	
xE	拡張命令	VOICE		BOOT	VIEW	DEVI$		PAI	STICK	CVI	INSTR	KPOS	SEARCH	
xF	関数	CFLASH	CREV		REPLACE	DEVO$		RAD	STRIG	CVS	INKEY$	USR.	SUM	

0BH～14H・60H～68Hは2バイト，15H・1FHは5バイト，18Hは8バイトのバイナリデータが後に続きます。
2AH，70H～78Hの後には長さが1バイトと，ラベル・変数名を表す英数字・漢字等がそのバイト数だけ続きます。
7FHはLIST・PRINT等のコードの後に付きLIST/P・PRINT/P等を表します。
表中には，""中の文字列，変数名でのみ現れるものは含めていません(それらはキャラクタコードのまま入ります)。
「.」で始まる語は関数にも同一の語が存在する命令です。
「.」で終わる語はリスト時その後にスペースを空けません。
なお，これらの詳細については，本文を参照して下さい。

次に，0DHはラベルを同様にアドレスに変換したものです。行番号をアドレスに書換える手法は以前から存在しており，80系CPU(Z80，8086等)のBASICでは今や常識になっていますが，ラベルまでアドレスに変換してしまうのは，ラベル使用可能BASIC多しと言えども，これが初めてです。今までのBASICでは，ラベルは使いたいがスピードが落ちてしまうから…ということも多かったのですが，このBASIC-M25・S25ではこの心配も不要でしょう。

なお，0CH・0DHはプログラムの追加・変更・削除を行った時・SAVEを行った時等に元に

戻ります。

次に，０EHは&Bから始まる２進数で使われます。例えば，A＝&B11100101100010なら，&B～の部分は０EH＋62H＋39Hになります。２進数の「11100101100010」は，16進数の3962Hだからです。入力した時点でこのように変換されるのです。

次に，０FH，10Hも同様に，８進数・16進数を表わすものです。&O33066は０FH＋36H＋36Hに，&H4D3Aは10H＋３AH＋４DHになります。

次に，11Hは16進数を表わす「＄」です。マニュアルには説明がないようですが，M25，S25共に，この機能があります。働きは「&H」と全く同じなのですが，リストをとった時に１文字ですっきりと出て来ますし，これまでMZを使って来た人にはお馴じみのことと思います(MZ-2000等は＄で16進数を表わしていた)ので，利用してみるのも良いでしょう。なお，＄306Cは11H＋6CH＋30Hになります。

次に，12Hは10進数の整数を表わします。0～9は01H～0AHになりますが，10～32767ではこれになります。例えばA＝1000なら，1000の部分は12H＋E8H＋03Hになります。なお，32768以上の値は実数扱いになります。また，負の定数では，「－」のコードが先に来てその後に正の定数が来ます。－１ならF8H＋02H，－1000ならF8H＋12H＋E8H＋03Hになるわけです。

次に，13Hと14HはJISコード・区点コードを表わすものです。しかし，次に続く２バイトは，これらをシフトJISコードに予め変換したものです。CHR＄の引き数やASCの結果等がシフトJISであるため，実行スピードを上げるためにこうなっているわけです。例えば&J4545は13H＋64H＋93Hに，&K3940は14H＋67H＋94Hになります。JISコード4545Hはシフト JISでは9364H，区点コード3940はシフトJISでは9467Hだからです。

なお，０EH～14Hは，入力時・リスト出力時にははっきりと区別されますが，実行時にはいずれも後続の２バイトの値を符号付き整数と判断しています。逆に言うと，こうしてスピードアップできるよう工夫されているわけです。

次に，15Hは単精度実数(５バイト)が後に続くことを示します。32768以上の整数と，小数部を含む９桁以内の値は，特に指定しない限りこの扱いになります。また，これらの条件を満たさなくとも，値の後に「！」が付いているか，最後に「＃」が付いていなくて「E表現」を使っている場合には，これになります。例えばA＝100000またはA＝1E5またはA＝100000！と入力すると，値の部分はいずれも15H＋91H＋43H＋50H＋00H＋00Hになります(リスト出力時は100000だけになる)。また，A＝１！やA＝1.0，A＝1E0と入力すると，値の部分はいずれも15H＋81H＋00H＋00H＋00H＋00Hになります(リスト出力時は１！となる)。なお，単精度実数を示す５バイトの内容については，数値の内部表現を見て下さい。

次に，18Hは倍精度実数(８バイト)が後に続くことを示します。有効数字10桁以上の値は，特に指定しない限り，この扱いになります。また，９桁以下の値でも，最後に「＃」が付いているか，最後に「！」が付いていなくて「D表現」を使っている場合には，これになります。

例えば，A＝10000000000またはA＝1D10等と入力すると，値の部分はいずれも18H＋A2H＋15H＋02H＋F9H＋00H＋00H＋00H＋00Hになります。また，A＝1＃やA＝1.0＃，A＝1D0と入力すると，値の部分はいずれも18H＋81H＋00H＋00H＋00H＋00H＋00H＋00H＋00Hになります(リスト時は１＃となる)。A＝１！やA＝1＃等は，値の部分がA＝１の場合よりメモ

リを食い，一見無駄のように見えますが，例えばA＝SQR(2.!)のような場合には有効です。つまり，A＝SQR(2)では整数の2を実数に一度変換してからSQRを計算するのに対し，A＝SQR(2.!)なら変換作業なしで計算に移れるというわけです。また，四則演算の片方の値が実数または結果が実数になる時は，演算は実数で行われますので，高速処理を特に望むなら，A＝1/3+1よりA＝1.!/3.!+1.!の方が有利です。また，変数Aの値が実数であることが多いなら，A＝A＋1よりA＝A＋1.!の方が有利です。

次に，1FHはFOR，WHILE，ただのREPEAT，ブロックIFのTHEN，ELSEの後に続き，さらに5バイトのスペースが置かれることを示しています。インタプリタにはあるフラグがあって，GOTO，GOSUB，RUN等の実行時にそれをチェックし，もしリセットされているとプログラム全体の「1FH」に続く5バイトに，FOR等と対応するNEXT，WEND，UNTILや，ブロックIFでは対応するELSE IF，またはELSE，またはEND IFを次々とチェインさせる形で位置を記録して行きます。そしてそのフラグはセットされます。また，プログラムの追加・修正・削除等によってリセットされます。5バイトの内容は

・1行の中でNEXT等のある位置(1バイト)
・NEXT等のある行の始まるアドレス(2バイト)
・NEXT等のある行の行番号(2バイト)

になります。ただ，対応するNEXT等がない時にはその場でエラーにはせず，そこを実行する時に，初めてエラーになります。プログラムの最後の方など取りあえずは実行しなくて良い所にミスがあったためプログラムが全く動かなくなってしまう，と言ったコンパイラ言語のような使いづらさを招くことはないのです。なお，このFOR～NEXT等の実アドレス変換は，ラベル同様，初めてのものです。

次に，20H，22H，23H，27H，28H，29H，2CH，2EH，3AH，3BH，40H，5BH，5DHは，キャラクタコードと一致しており，原則として，これらが現われた時には，特に他の中間コードに変換せずにそのまま入ります。ただし，23H(#)はPRINT#等，ファイル番号を表わす時だけ使われます。また，27H(')は，単独では出て来ないで，3AH，27Hと続いて使われます。'はREMの省略形と言われていますが，実際には「:」の働きを含んでおり，「:REM」の省略形です。このため，中間コード上でも3AHが前に来るのです。また，3AHは，C2H(ELSE)の前にも自動的に入ることがあります。普通のIF文中のELSEは3AH＋C2Hとなるのです(ブロックIFのELSEはC2Hのみ)。「'」も含め，一見無駄にも感じられますが，ステートメントの区切りは「1行が終わる」または「:が来る」とだけ判断した方がスピードの点で有利になるため，他のBASICでもこのようにするのが普通になっています。また，2EH(.)は，LISTやEDIT等で，インタプリタが着目している行(最後に入力・修正・実行した行)を表わすのにのみ使われ，小数点を表わすためには使われません。それと，スペースは「:」の前後等に入るだけでその他の部分には存在しません。

次に，2AHはラベルが続くことを表わします。例えば，＊TESTなら，2AH＋04H＋54H＋53H＋54Hになります。04Hというのは，ラベルが4文字であることを示しています。また，これがアドレスに変更される時は0DH＋××H＋△△Hになると言いましたが，ラベルが2文字以上の時には，2文字目以降が余ってしまいます。＊TESTでも，45H以降が不要となります。そこで，余った部分にはスペースが自動的に入れられます。なお，ラベルはアルファベット・数字・アン

ダーライン以外に，カタカナや全角の漢字も利用でき，それらはキャラクタコードのまま入りますが，全角のひらがな等はエラーになってしまいます。しかし，BASIC-M25の場合は7E35H，S25の場合は7C11Hの88Hという値を82Hに変更すると，全角のアルファベット（大文字・小文字共），ひらがな，カタカナ，ギリシャ文字，ロシア文字等もラベル名，変数名，として使えるように，つまり，入力してもエラーにならないようになります。これはエディタの文字列→中間コード変換ルーチンを変更するものですので，一度この状態で入力してしまったら，LOADしてRUNする時はこの変更は必要ありません。なお，この改造は，POKE@ &HA, &H1E35, &H82(M25)または，POKE@ &HA, &H1C11, &H88(S25)でもOKです。auto-run.m25等にでも入れておくと良いでしょう。

次に，60H，62H，63H，64H，68Hと70H，72H，73H，74H，78Hは0DHと2AHとの関係に大変良く似ており，60H～68Hはアドレス，70H～78Hは変数名が続きます。最初は70H～78Hの変数名ですが，一度そこを通過すると，60H～68Hのアドレスに変わります。余った部分がスペースになるのも同じです。ただ，プログラム入力・変更・削除・SAVE時等に加え，変数がクリアされる時にも元の70H～78H＋長さ＋名前に戻ります。ラベルでもそうでしたが，変数名がアドレスに変換されるのもBASICインタプリタとしては，これが初めてです。ところで，60H～68H・70H～78Hの違いですが，リストを取った時に%が付く整数変数が62H・72H，$が付く文字変数が63H・73H，!が付く単精変度数が65H・75H，#が付く倍精度変数が68H・78Hです。また，何も付かず，DEF INT，DEF STR，DEF SNG，DEF DBLで指定した型になる変数は，60H・70Hになります。型無変数と書いたのは型が無いという意味ではなく，型の指定が無い，という意味なのです。この場合は型テーブルを参照することになりますので，スピードはわずかですが落ちます。72H・73H・75H・78Hの後の変数名の部分には，!や$などの記号は含まれませんので，見づらくなければA＝0とするよりA!＝0の方が良いかも知れません(メモリ消費量は同じでスピードが少し速い)。また，漢字・カタカナの変数は，DEF INT等の指定の対象外ですから，型指定記号がないと無条件に倍精度型になりますので，こちらではむしろ積極的に!や%等を付けるようにすると良いでしょう。

なお，70H～78Hは，A!やA$などの単独変数の他に，配列変数やFNの後に付くユーザー関数名，同仮引数，「SAVE～,A」の「A」，LINE文の「B」または「BF」オプション等でも使われますが，これらは60H～68Hに変更されることはありません。

次に，7FHは，LIST，PRINT，TRON，FILES(DIR)等の後に付き，プリンタへの出力であることを指定するものです。M25では87H＋7FHでLLIST，8FH＋7FHでLPRINTを示しますし，S25では87H＋7FHでLIST/P，8FH＋7FHでPRINT/Pを示します。見かけ上は異なりますが，実行時は同じ働きをするのです。

次に，80H～E7Hは一般ステートメントと，その中で補助的に使われる予約語が割り当てられています。M25とS25は，大まかには同じような語が入っていると考えられますが，FILESとDIRのように，名前が異なり意味が同じか似ているものは同一コードになっています。M25のLOG・LOG10とS25のLN・LOGについてもLOG同士を同じコードにするのではなく，意味の同じ，M25のLOGとLN，LOG10とS25のLOGを同一コードにしています。

次に，E8H～FDHは演算子です。左右の値を元に，一つの結果を得る2項演算子と，右の値を

元に一つの結果を得る( )なし関数とも言うべき単項演算子とが，優先順位の低い方から並んでいます。BASICの中には，中間コードは適当に決めておき，それとは別に優先順位テーブルを持っているものもありますが，それよりも高速です。

次に，FEHとFFHは2バイトコードの1バイト目となります。つまり，予約語の数が100個程度であれば80H～FFHに収めてしまうということも可能なのですが，M25やS25は約300にも上るため，それは不可能です。そこで，FEHとFFHを拡張子としておき，次に続く1バイトによって別の語にしようというわけです。

FEH＋80H～FEH＋DFHは，基本的でないステートメントが入っています。何をもって「基本的でない」とするかは明確ではありませんが，一応グラフィック関係のものと，MZ-1500や2000では存在しなかったようなものが中心になっています。また，SCRのように単独では使われずFE＋??Hのステートメントの補助として使われるものも含まれています。

次に，FFH＋80H～FFH＋DFHは，関数が割り当てられています。そのうち，FFH＋80H～FFH＋95Hは引き数・結果共に数値型であり，浮動小数点パッケージを直接中間コードを元にした番号で呼び出しています。その他は一般ステートメント同様，インタプリタ内部で処理していますが，FFH＋98H～FFH＋9CHとFFH＋AAH～FFH＋B2Hは引き数は文字型，FFH＋9DH～FFH＋A9Hは引き数は数値型，FFH＋B3H～FFH＋DFHは引き数がないか2以上あって，個別に文法チェック等まで行っているものになっています。

そして最後に，FFH＋E3H～FFH＋E8Hは，DEF，OPTION等で補助的に使われるものが並んでいます。

ところで，80H以上の語の中には「.」で始まるものと，「.」で終わるものとが含まれていますが，実際の予約語はここから「.」を取り除いたものになります。

まず，前に「.」が付いている語は80H～FDHとFEH＋80H～FEH＋DFHの間にしかありません。ステートメントの前に付くことがあるわけです。そして，さらに注意深く見てみると，これらの語は，関数にも同じ名前のものが存在することがわかります。これらは，BASICインタプリタとしては恐らく初めての試みだと思いますが，同じ名前の予約語であっても，関数として使われた時と，ステートメントとして使われた時とでは，別の中間コードにするようになっているのです。予約語は80H～FDH，FEH＋80H～FEH＋DFH，FFH＋80H～FFH＋EFHの3段階で，またその中ではコードの低い方から順に合致しているかどうかを検索しており，例えば「SEARCH」が関数として使われているにもかかわらず，B9Hで合致してしまっては困るので，前に「.」を付けているのです。そして，ステートメントとして使われている場合に限り，この「.」を無視するようになっているのです。ステートメントと関数の区別は，

・1行の先頭
・：の後
・THENやELSEの後

にあるのはステートメント，それ以外の所にあるのは関数とみなしています。

一方，後に「.」が付いている語はFEH＋C9HのMID＄(代入を行う方)とFFH＋C7HのFN，FFH＋CFHのUSRだけです。BASIC-M25・S25ではスピードアップのため，スペースの置き方に制限を加えており，1行の先頭と：の前後等だけは自由な数だけスペースを置けますが，その他の部

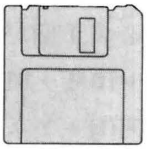
分では，必要最少限に留められています。つまり，ステートメントの後のスペースは必ず１個だけ置かれるものとし，中間コードでは省略しているのです。例えばPRINT Aは８FH＋70H＋01H＋41Hになり，PRINTとAとの間のスペースは入っていません。また中間コードがFFH＋？？Hである関数は，後にスペースは入りません。また，演算子のうち，ORなどアルファベットのものは前後にスペースを置きますが，＋など記号のものはPRINT ＋Aなど前には置かれることも，PRINT A＋Bのように置かないこともあります。また，後ろには無条件で来ません。また，演算子以外でも「,」の前後はスペースなしですし＠は前にはスペースなし，後にはスペースありです。ただ,その場合でも後が「(」の時にはスペースなしになります。このように，普通にリストを見ただけではスペースが(中間コードとしては)入っていないことに気が付かないくらいきめの細かな処理を中間コード→文字列変換ルーチンで行っているのです。ただ，その中にも例外があるため，「.」を後に置いています。リスト出力時には，「.」で終わる語の後には無条件にスペースを置かない，また，文字列→中間コード変換時は「.」の前まで合致したら，そこで比較を打ち切って予約語とみなす，という働きをするのです。後者はちょっと分かりづらいかも知れませんが，要するにLISTAやRUNXはLISTやRUNとみなさないが，FNAやUSR0はFNやUSRとみなすということです。ただ，これはMID＄の方では意味はありません。「＄」等の記号はセパレータとして働くからです。

では，中間コードの締めくくりとして，１行の構造を見てみましょう。１行は，

・１行のバイト数(これ自身を含む)　…１バイト
・RENUM用ワーク(通常は使われない)…１バイト
・行番号(下位，上位の順)　…２バイト
・字下げ数　…１バイト
・中間コード列　…249バイト以内
・終了マーク(00H)　…１バイト

の順で入っており，必ず255バイト以内に収まるようになっています。字下げは１行の先頭に置かれるスペースのことですが，このように常に１バイトでその桁数だけを保持していますので，思い切り字下げを行っても，メモリを浪費したりスピードが遅くなったりはしません。

なお，BASICのプログラムは4000Hから始まり，容量が増える毎に８Kバイトずつ割り当てを追加するようになっていますが，このうちの4000H～410FHはダイレクトコマンド用のバッファとプログラム入力の際の一時的なバッファとして使われ，4110H以降にプログラムが格納されます。「１行のバイト数」が０であるところがプログラム全体の終了位置となります。また，行番号等を検索する場合を除き，実際にはA000H～DFFFHがウィンドウとなり，インタプリタはここを見ながら動きます。そして，１行の先頭がD000H以降になったら８Kバイト毎にウィンドウをスライドさせるようになっています。また，E000H～FFFFHには，変数名・FN等の登録エリアがマッピングされます(うちE800H～FFFFHのみ使用)。中間コード60H～68Hのアドレスは，この範囲を指すことになります。また，中間コード0DH，0EHの飛び先のアドレスは，4000H～FFFFHの範囲になります。



数値(浮動小数点)の内部表現

変数とBASICのプログラム中，また，インタプリタや浮動小数点パッケージで使用している数値の内部表現はいずれも同形式で，次のようになっています。

単精度実数

単精度実数は5バイトで表わされ，指数用が1バイトと仮数用が4バイトの配分になっています。仮数はビットに直すと「32ビット」ありますが，このうち31ビットを使って

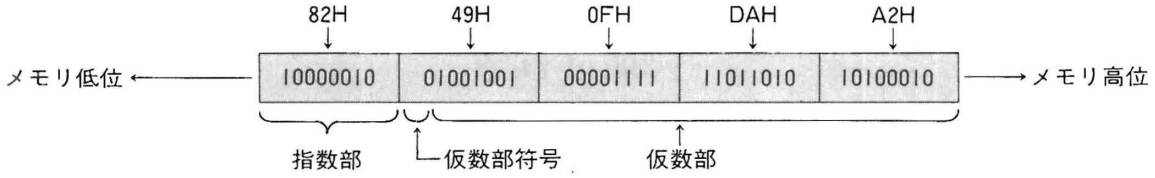
$\frac{2^{16}}{2^{32}}$～$\frac{2^{32}-1}{2^{32}}$を$\frac{1}{2^{32}}$間隔で

(0.5～約0.99999999976717の間を0.0000000002328306437間隔で)表わし，それを指数を使って2^{-127}～2^{127}倍して実際の値を表わすようになっています。また，仮数の余った1ビットを使って，仮数部の符号(その値の符号)を表わしています。また，指数部が255通りしかありませんので，最後の一つを使って「0」を表わしています。この方式で表現できる最大の数は，$\frac{2^{32}-1}{2^{32}}\times 2^{127}$つまり$(2^{32}-1)\times 2^{95}$で，$1.701411834\times 10^{38}$になります。また，0を除いて最も絶対値の小さい正の数は$\frac{2^{16}}{2^{32}}\times 2^{-127}$つまり$2^{-128}$で，$2.938735877\times 10^{-39}$になります。

では，実際の例を見てみたいと思います。

82H，49H，0FH，DAH，A2Hとメモリのアドレスの低い方から並んでいたとします。

まず，これらを2進数に直してみましょう。



このようになります。仮数部は31ビットですが，計算上は最上位ビットを1とした32ビットとして扱います。つまり，ここでは10010010000111111011010100010ですが，11001001000011111101101010100010として扱うわけです。するとこの値は3373259426(10進数)になります。この計算はちょっと面倒ですが，アルゴ機能の10進・16進電卓で，16進数のC90FDAA2(80000000＋490FDA42)を10進数に変換すると，一発でわかります。

そしてこの3373259426を2^{32}(4294967296)で割ります。これも同電卓で計算すると良いでしょう。結果は0.78539816337になると思います。他の値の場合でも，ここでの結果は必ず0.5以上1未満になります。

次に指数部を見てみます。指数は－127～＋127が可能ですが，メモリ上では128(80H)されて入っています。逆に言うと，メモリ上の指数部から80Hを引いたものが$\times 2^{n}$のnであると言えるでしょう。この例の場合は82Hですので，ここから80Hを引いた02H(10進数でも2)がnとなります。つまり，倍率は2^{2}で4となるのです。

そして仮数部と指数部を「かけ算」してみます。0.78539816337×4で，3.14159265347となると

思います(多少誤差が出ています)。

最後に仮数部符号は0ですので,この値の符号は正です(1なら負です)。つまり,82H,49H,0FH,DAH,A2Hは,+3.14159265347を表わすというわけです。

もう一つ,例を見てみましょう。

7DH,CCH,CCH,CCH,CDHで考えてみます。

まず,31ビットの仮数部は,4CCCCCCDHになり,計算上は32ビットのCCCCCCCDHになります。これを10進数に直すと3435973837になります。これを2^{32}で割ると0.80000000005になります。そして仮数は7DH−80Hは−3。2^{-3}は$\frac{1}{2^3}$ですので$\frac{1}{8}$。これらを乗算すると約0.1になります。最後に仮数部の符号は「1」つまり負なので,この値は−0.1を表わすことになります。

なお,指数部が01H〜7FHは−127〜−1を,80H〜FFHは0〜+127を表わすことになりますが,指数部が00Hの時は−128ではなく,その値全体が0と見なされます。つまり,指数部が00Hなら仮数部がどんな値でも0になるということです。

倍精度実数

倍精度実数は,指数1バイト,仮数7バイトとなる点が異なるだけで,基本的な考え方や計算方法は全く変わりありません。

仮数の方だけ$2^{32}\rightarrow 2^{56}$,31→55等と読み替えれば,そのまま通用します。ですから,倍精度の9FH,13H,2CH,05H,A4H,00H,00H,00Hと単精度の9FH,13H,2CH,05H,A4Hとは,同じ値になります。小数以下の有効桁が多いか少ないかだけの違いであると言っても良いでしょう。実際,指数部と仮数部符号の間に仮想小数点があり,仮数部符号の所には常に1が入る,と考えても良いのです。

では,一応例を見てみましょう。

B5H,39H,F8H,20H,34H,E0H,65H,88Hの場合では,仮数部はB9F82034E06588Hが10進の52345678901234568であり,これを2^{56}(72057594037927936)で割ると,結果は0.72644234934741380715になります。ここまでは,BASICで

```
PRINT &HB9/256+&HF8/65536+&H20/1677216+…
```

のようにして計算しても構いません。

次に,指数部はB5H−80H=35H(53)なので,倍率は2^{53}になります。また,39Hのb_7(最上位ビット)は0なので符号は正です。

この結果,答えは0.72644234934741380715×2^{53}で,6543210987654321になります。

以上で浮動小数点内部表現の仕組みがだいたいわかってもらえたかと思いますが,もし,内部解析等をしていて,手っ取り早く値を知りたい時には,BASICのCVS,CVDを使うと便利です。

さきほどの例なら

```
PRINT CVD(HEXCHR$("B539F82034E06588"))
```

とするだけでOKです。単精度なら"〜"の中は10文字にし,CVDの代わりにCVSにします。あるいは最後の3バイト分を00にして,このまま使用しても構いません。いわゆる2進↔10進変換誤差が良くわかります。

また,この逆も簡単にできます。1.5の内部表現なら,

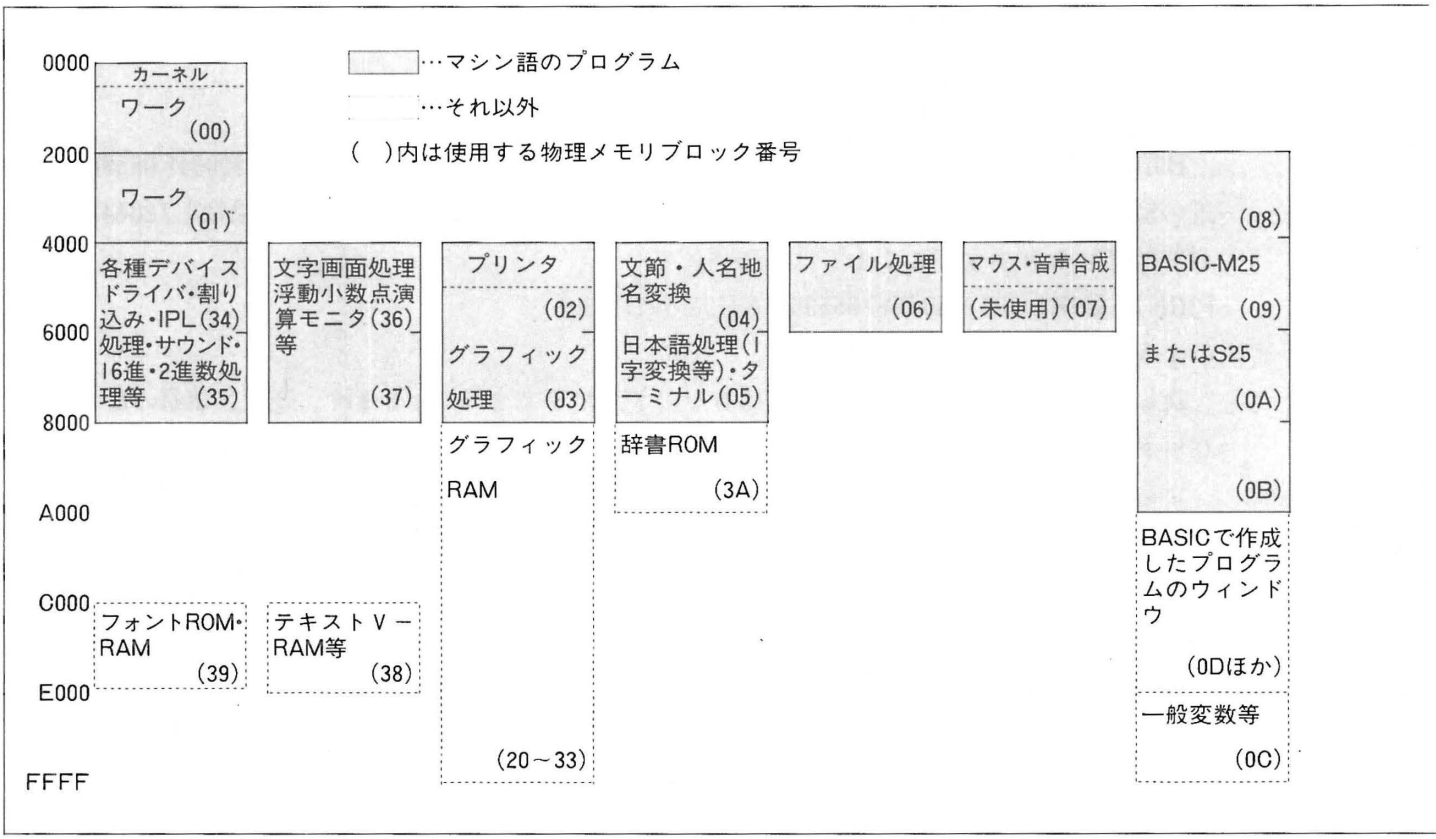
PRINT ASCCHR$ (MKS$(1.5))でOKです。倍精度なら
PRINT ASCCHR$ (MKD$(1.5#))とすれば可能です。

変数の内部表現補足

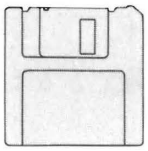
単精度・倍精度実数の内容表現は今の話の通りですが，BASICの単精度変数についてだけは少し変わった表現になることがあります。

それは，値が－32768～＋32767の間の整数の時です。例えばA＝100という場合の100の部分は12H＋64H＋00Hであり，整数の0064Hを示しています。普通ならこれを単精度実数の形式（87H，48H，00H，00H，00H）にわざわざ変換してから代入を行うのですが，BASIC-M25・S25の場合には，これを，整数のまま，代入してしまうのです。このため，変換に要する時間と，その後の四則演算等に要する時間が(整数は速いから)大幅に節約でき，スピードアップにつながるのです。

しかし，代入した値が実数の形式だったか整数の形式だったかを，別の場所に保持しているわけではありません。メモリ効率を落としてまで，スピードアップを図ろうというわけではないのです。ではどうしているかと言うと，５バイトのうちの先頭の１バイトを00Hにして,続く２バイトに下位・上位の順で整数のまま置くのです。つまり,実数の１バイト目が00Hの時には２～５バイト目が未使用(何でも良い)となる点に着目し，うまく利用しているわけです（もし，０そのものをストアしたいなら１～３バイト目を全部00Hにすれば良い)。コロンブスの卵的な発想ですが，これまたBASICでは初めての例です。



《第8-1図》BASIC使用時のメモリ状態



メモリの使われ方

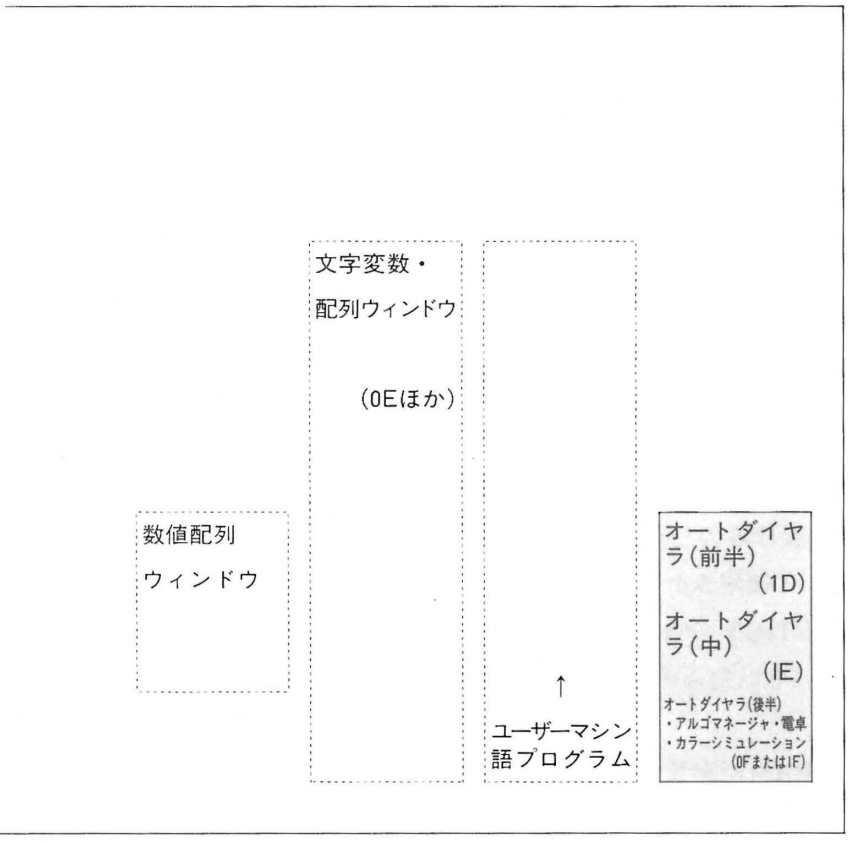
スーパーMZの内面的なハードウエアで，最も特徴的なのはメモリのマッピングシステムでしょう。

各種メモリ等の空間が512Kバイト分あり，それを８Kバイトずつ区切ってZ80のメモリに自由自在にマッピングできることは第２章で触れた通りですが，BASIC利用時に実際にどのように使われているかを見てみたいと思います。

第8-1図を見て下さい。かなり複雑な様子になっているように見えますが，実はかなり効率の良いものになっています。

まず，左上に**「カーネル」**(核)という部分がありますが，これが全体の中心になっています。カーネルは，後で触れるRST 18H，RST 28Hによる，各種処理ルーチンの呼出し（ファンクションコール)と，その前処理・後処理(マッピング)を行うのが役割りで，その他，言わばZ80の命令を補足するような10数バイトクラスの汎用サブルーチン(HL～HL＋B－１のアドレスに０を入れるとか，LD DE，(HL)とも言うべきもの等)をいくつか含んでいます。しかし，カーネルはわずか約１Kバイト強と小さく，しかもその約半分はジャンプテーブルになっています。また，カーネルの後には重要な共有ワークエリアが続きます。

各種ファンクションコールの処理ルーチンは，一般に4000Ｈ～７ＦＦＦＨの間で動くように作られており，ある程度のまとまりを作っています。(　)内が34～37のものはROMですが，それ以外はRAMです。つまり，ROMもRAMも全く同レベルのものとして作られているのです。



また，グラフィックRAM等は，ファンクションコールで飛んだ先で必要に応じてアクセスされるようになっており，無駄がありません。それに，ファンクションコール自体，ある程度機能的にまとまったものにしているため，一旦カーネルを通してマッピングの切換えを行うためにかかる時間をほとんど無視できるようになっています。つまり，0.00002秒かかる簡単な処理を行うのに0.00001秒余分な時間がかかるのは問題だが，0.1秒かかるグラフィックの処理を行うために，余分に0.0001秒かかるだけなら全く問題ないというわけです。その点を考慮して，BASIC本体(自身)は連続したマッピング(2000$_H$～９FFF$_H$)にしているのです。また，BASICのプログラムのウィンドウを

16Kバイトと大きくとっているのも，それによるスピードダウンを皆無にするためです。

ところで，各種ファクションコールは，不要であるなら8K～16Kバイト単位で削除できるようになっています。BASICではNEW ONがこれをサポートしており，

NEW ON 0…アルゴ機能(1D～1Fまたは0F)をカット

NEW ON 1…07(マウス・音声合成の部分)をカット

NEW ON 2…07＋04(文節・人名地名変換の部分)をカット

NEW ON 3…07＋04＋05(日本語処理・ターミナルの部分)をカット

NEW ON 4…07＋04＋05＋03(グラフィックの大部分)をカット

NEW ON 5…07＋04＋05＋03＋02(プリンタとグラフィックの残りの部分)をカット

のようになっています。06つまりファイル処理はNEW ONではカットされません。また，原理上，グラフィックはカットして日本語処理をカットしない，ということもできるのですが，サポートはされていません。ただ，さほど難しくないのでファイトのある人は，マップ34Hの49A7H以降を解析してみると良いでしょう。

ところで，数多く存在する**「共有ワークエリア」**の中で，最も重要で使用頻度の高いものがあります。これもマッピングに関連したものです。

まず，アドレス0540H～056FHの30Hバイトです。ここらは，物理ブロック00H～2FHと1対1で対応しています。

1バイトの内容は次の通りです。

	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	未使用 (使用可能)	プログラムは置けない	データ・グラフィックは置けない	システムでは使用していない	ユーザーで使用可。ただしb_4＝1の時は 0000…OS部(BASIC以外)またはグラフィックRAM 0001…アプリケーション部(BASIC等) 0010…アルゴ機能 0011…プリンタスプーラ・RAMディスク 1001…BASIC等で使用するデータ(BASICプログラムや変数等) として使用している			
1	使用中またはRAM未実装 (未拡張)	プログラムは置ける（システム RAM である）	データ・グラフィックのみ置ける（グラフィックRAMである）	シャープのプログラムで使用している(BASIC等で使用中)				

ただし，実際には

20H…未使用(マシン語プログラムは置けない…OPTION SCREEN 1～3でグラフィックRAMを解放した場合。原理上はCLEARアドレスでのマシン語プログラムを置かない限り，BASICのプログラムでもデータでも何でも置けるのだが，実際にはプリンタスプーラか，RAMディスクかグラフィック用に戻すだけに使用される)

40H…未使用(マシン語プログラムは置ける…OPTION SCREEN 4でグラフィックRAMを解放した場合・純粋に未使用の場合・既に使っていたものをNEW ONで解放したもの。何にでも使用可能)

80H…増設用のメインRAMまたはグラフィックRAMが実装されていない

B0H…グラフィックRAMとして使用中

B3H…グラフィックRAMをプリンタスプーラ・RAMディスクとして使用中

D0H…OS部(BASIC以外)で使用中
D1H…アプリケーション部(BASIC等)が存在する
D2H…アルゴ機能のプログラムが入っている
D3H…メインRAMをプリンタスプーラ・RAMディスクとして使用中
D9H…BASICのプログラム・各種変数・CLEARで確保したマシン語エリアとして使用中

等が入っていることが多いようです。ところで，ここで一つ注意してほしいのは，OPTION SCREEN 4の扱いです。マニュアルでは「一度OPTION SCREEN 4を実行すると元に戻すことはできなくなる」という内容が書かれているだけですが，実際にはこの時，グラフィックRAMをメインRAMに変更しています。グラフィックのハードウエアビューポートを操作(VIEW@(1,1)－(0,0))して，G-CRTCがグラフィックRAMを全くアクセスしないようにし，その上で20H(未使用グラフィックRAM)・B0H(グラフィックRAM)→40H(未使用メインRAM)へ変更しています。この結果，OPTION SCREEN 4を行うと，その場でFREで得られるフリーエリアが128Kバイト分増加することになります。

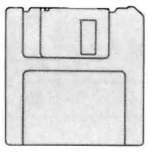
なお，ユーザーは，0540H～056FHの中の希望のブロックのデータのb_7を1にセットするだけで，即，使用可能となります。また，これを0にするだけで未使用に戻すことができます。

次に，アドレス0570H～057FHも大変重要な働きを持っています。

0570H～0577H…現在実行中のアプリケーション部(BASICインタプリタとかテレソフト等)のマッピング情報が入っています。0570Hは一般に00Hですが，0571H～0577Hは，2000H～FFFFHについての対応する物理メモリ番号が入っているわけです。メモリコントローラ(B4H・B5H)へのデフォルトの設定値が入っていると考えても良いでしょう。

0578H～057FH…モニタのD・Gコマンド等でアクセスされるメモリのマッピングです。MMコマンドを使うと，このうちの0579H～057FHを書換えられることになります(;mapping＝00の00は書換えられない)。また，この内容は，モニタに入る時に0580Hを00Hから転送して来ます。

0580H～0587H…BASICのPEEK，POKE，CALL等でアクセスされるメモリのマッピングです。なお，このエリアはCLEAR文を使用した時に書換えられるだけですので，ユーザーは自由に変更可能です。ただ，PEEK・POKEだけなら良いのですが，CALL・USRを使う時には0580Hを00H以外にしてはなりません。これは，目的プログラムへのジャンプのために0000H～1FFFHの空間を使用しているからです。



RST 18HとRST 28Hによるファンクションコールについて

スーパーMZでは，大規模システムを効率良く働かせられるよう，BASICはOS(オペレーティングシステム)の上で動くような形をとっています。次に示す表がその全貌ですが，BASICはインタプリタとしての命令語の解釈だけを行い，実際の処理はこのOSにまかせています。

OSを呼び出すのは浮動小数点を扱うタイプとそうでないタイプの2通りがあります。このように分けたのは，将来的にハードウエアで浮動小数点がサポートできる16または32ビット機との互換性等を考えたためかと思われます。従って，実際には浮動小数点を扱わない方のブロックで，

OSとしてのほとんどの機能を備えていることになります。

これらの呼び出し方は，OSの中心となる方が

RST 18H

DEFB nnH　(nnHは00H～7FHの機能番号)

そして浮動小数点を扱う方が

RST 28H

DEFB nnH　(nnHは00H～37Hの機能番号)

です。マクロアセンブラを使う方はこの二つはマクロ定義しておくと良いかと思います。

パラメータ等も表中に入れてますので，一つ一つ確かめながら，理解してゆくのが良いでしょう。

そして皆さんのプログラム作りにお役立て下さい。

《第8-3表》RST 18HによるIOCS(OS)ファンクションコール

保存レジスタ：裏レジスタとAレジスタを除きます
(裏レジスタは常に破壊されます)

RST 18H DEFB nnH のnnHの値 (リクエスト番号)	名称	機能	入力レジスタ	出力レジスタ	保存レジスタ	処理ルーチン	
						マップ	アドレス
00	MONOP	モニタを呼び出す	なし	なし	なし	36	7C02
01	CR1	改行を行う	なし	なし	すべて	36	4657
02	CR2	カーソルが0文字目にない時には改行を行う	なし	なし	すべて	36	4653
03	CRT1C	画面に1文字表示する(コントロールコードは実行する)	A＝キャラクタコード	なし	すべて	36	4659
04	CRT1X	画面に1文字表示する(コントロールコードはシンボル表示)	A＝キャラクタコード	なし	すべて	36	46A6
05	CRTMS	文字列を画面に表示する(コントロールコードは実行する)	DE＝文字列(00Hがエンドマーク)	なし	すべて	00	0110
06	LPT0T	プリンタに1文字データを送る(コード変換は一切しない)	A＝キャラクタコード	なし	すべて	34	60A4
07	LPT1C	プリンタに1文字データを送る(コード変換は行う)	A＝キャラクタコード	なし	すべて	34	609A
08	DISP	テキストの指定位置に文字を表示する	A＝キャラクタコード， L＝X位置，H＝Y位置	なし	すべて	36	5BD2
09	TERM	ターミナルモードに入る	HL＝文字列(TERM文コンパチ)， A＝0…全二重・A≠0…半二重	なし	なし	04	7783
0A	(未使用)					00	003B
0B	(未使用)					00	003B
0C	GETL	キーボードから1行入力を行う(BASICのLINE INPUT文コンパチ)	なし	DE…入力文字列のあるアドレス Cy…SHIFT＋BREAKされたらセット	HL,BC	36	41CD
0D	INKEY	キーボードから1文字入力する	Aレジスタの b_3＝0…アルゴキーはフラグで返すのみ b_3＝1…アルゴ機能を実行する b_2＝0…カナ漢字変換可能 b_2＝1…カナ漢字変換禁止	Zフラグ＝0…入力データあり Zフラグ＝1…入力データなし C…b_7＝0…Bレジスタは普通のキャラクタコード b_7＝1…Bレジスタは特殊コード	HL,DE	34	59CF

RST 18H DEFB nnH のnnHの値 (リクエスト番号)	名称	機能	入力レジスタ	出力レジスタ	保存レジスタ	処理ルーチン	
						マップ	アドレス
0D	INKEY	キーボードから1文字入力する	b_1=0…ファンクションキーは専用コードで返す b_1=1…ファンクションキーは定義されている文字列で返す b_0=0…キーが押されてなければZ=1でリターン b_0=1…キーが押されていなければ押されるまで待つ	b_6…無変換 b_5…変換 b_4…CTRL b_3…カナモード b_2…SHIFT b_1…LOCKモード b_0…GRAPHモード } 押されているか，そのモードならセットされる B…キャラクタコードまたは特殊コード〈特殊コードの詳細は次の通り〉 81~8A…F1~F10　93…BREAK 8C…→ } CTRL　94…HOME/CLR 8D…← } か　95…ESC 8E…↑ } SHIFT　96…変換 8F…↓ } と併用　97…無変換 90…TAB　98…アルゴ 91…DEL/INST　99…HELP 92…⇦　9A…COPY A…Bと同内容	HL, DE	34	59CF
0E	BREAK	SHIFT+BREAKキーをテスト	なし	Z=0…SHIFT+BREAKされてない Z=1…SHIFT+BREAKされた	すべて	00	0126
0F	HALT	BREAKが押されていたら一時停止	なし	なし (注.一時停止中にSHIFT+BREAKされたらエラー0扱い)	なし	34	59AB
10	DI	割込みを一時禁止する	A=0…1ms~1秒程度で使い，RS-232Cのみ停止 A≠0…1秒以上の時使い，音楽も停止させる なお1msec以内なら単なるDIで良い	なし	すべて	34	4698
11	EI	割込みを許可する	なし	なし	すべて	34	46A7
12	NEWON	IOCSから機能を削除する	A=0…アルゴ機能(DIOCS) A=1…MOUSE, VOICE A=2…JISHO A=3…TERM, KKAN A=4…CLS, POSCK, POSSV, PSET, LINE, PATTR, BOX, PAINT, CIRCLE, POINT, GINIT, VIEW, SCREN, COLOR, SYMBL, HVIEW, CREP, GPUT, GGET, PENSE, MOVE, PRT, CBLOC, CRTI, COCNV, HROLL A=5…LPTRA, HCPY	なし	なし	34	49A7
13	DEASC	文字列→バイナリ変換	HL…文字列をポイント	DE…結果(符号付き16ビット) HL…文字列の次を指す Cy=0…A=型(0EH…&B, 0FH…&O, 10H…&H, 11H…$, 12H…10進数, 13H…&J, 14H…&K) Cy=1…A=2…オーバーフロー A=3…エラー	DE・HL以外	34	47A5

RST 18H DEFB nnH のnnHの値 (リクエスト番号)	名称	機能	入力レジスタ	出力レジスタ	保存レジスタ	処理ルーチン	
						マップ	アドレス
14	DEHEX	16進数文字列→バイナリ変換 (&H, $は不要)	HL…文字列をポイント	DE…結果 HL…文字列の次を指す Cy=0…エラーなし Cy=1…A=2…オーバーフロー	DE・HL以外	34	47E8
15	CKHEX	16進数かどうかテスト	A…キャラクタコード	Cy=0…A=00H～0FH Cy=1…〝0〞～〝9〞・〝A〞～〝F〞・〝a〞～〝f〞でない	すべて	34	4847
16	ASCHL	バイナリ(正の16ビット)→文字列変換	HL…バイナリデータ DE…結果を入れたいアドレス B=0…ゼロサプレスする B≠0…ゼロサプレスしない (結果は常に5文字)	DE…文字列の次を指す	DE以外	34	4765
17	COUNT	文字列をカウントする(DEのアドレスから00Hが来るまでのバイト数を数える)	DE…文字列(00H…エンドマーク)	B…長さ	B以外	00	0102
18	KKAN	カナ漢字変換の表示・制御				04	6C91
19	JISHO	文節変換を行う				04	48D0
1A	DIOCS	アルゴ機能を呼び出す	A=0…メニュー選択を行う A=1以上…A番目のアルゴ機能を直接呼び出す			36	7F90
1B	ERRX	エラーメッセージを表示する	A…エラーコード(80H以上はAND 7FHしてデバイス・ファイル名と共に表示される)	なし	なし	06	5512
1C	KCONV	漢字コードを変換する	HL…コード A=0…シフトJIS →JIS A=1…JIS →シフトJIS A=2…シフトJIS →区点コード A=3…区点コード→シフトJIS A=4…区点コード→JIS A=5…JIS →区点コード	Cy=0…HL=コード Cy=1…HL=〝※〞(エラー)	HL以外	36	7B2F
1D	LPTRA	07H(LPT1C)の実処理ルーチン	(LPT1Cを使用のこと)			02	407C
1E	?PONT	文字V-RAMのアドレスを計算	L…X位置 H…Y位置	HL…アドレス(C000H～C7FFH)		36	4610
1F	KCALL	半角・全角変換を行う	A=0…一字変換用(HLは文字列を指す) A=1…半角→全角(D=ANK 1文字目, E=ANK 2文字目…〝と゜用) A=2…全角→半角(HL=シフトJIS)	HL…結果(シフトJIS, ANK等)		06	5924
20	BELL	ベルを鳴らす	なし	なし	なし	34	497C
21	PLAY	音楽を演奏する。ただし, 直前にA=3のMCTRLと, 直後にA=1のMCTRLを実行する	A…チャンネル番号(FFなら全チャンネル) DE…文字列を指す(BASICコンパチ)	なし	なし	34	5558
22	SOUND	YM-2203を直接制御	B=0…A=0～83の音階 E=0.1秒単位の音長 B=1…A=0～$B3_H$のレジスタ番号 E=出力したいデータ	なし	なし	34	55A4
23	MCTRL	音楽の制御	B=0…PLAY INIT(初期化) B=1…音楽演奏 B=2…音楽ストップ B=3…PLAY WAIT B=4…演奏中かどうか調べる(Z=1なら演奏中)	(左に書いてないものはなし)	なし	34	562B

RST 18H DEFB nnH のnnHの値 (リクエスト番号)	名称	機能	入力レジスタ	出力レジスタ	保存レジスタ	処理ルーチン	
						マップ	アドレス
23	MCTRL	音楽の制御	B＝5…音色データを取り出す(A＝音色番号, DE＝転送アドレス) B＝6…音色データをセット(C＝チャンネル, HL＝音色データアドレス) B＝7…LFOデータのセット・取り出し A＝0…セット, A＝1…取り出し, C＝チャンネル, HL＝アドレス	(左に書いてないものはなし)	なし	34	562B
24	(未使用)					00	003B
25	(未使用)					00	003B
26	PRT1C	ファイルに1文字出力する	A…キャラクタコード	なし	すべて	06	40AB
27	INP1C	ファイルから1文字入力する	なし	A…キャラクタコード Cy＝1…EOF	すべて	06	418C
28	CLRIO	すべてのOPENをキャンセルし, IOCSを初期化する	なし	なし	すべて	06	4463
29	(未使用)					00	003B
2A	PWD$	パスネームを取り出す	なし	DE…文字列の入ったアドレス B…バイト数(文字列の長さ)		06	4FD0
2B	MKDIR	サブディレクトリを作成する	なし(予めDEVFNを呼び出す)	なし	すべて	06	5051
2C	DEVNM	デバイス名とディレクトリ名を解釈する	DE…文字列を指す B…文字列の長さ	HL…文字列の次のアドレス DE…デバイステーブル A…ファイル番号	なし	06	4287
2D	DEVFN	デバイス名, ディレクトリ名, ファイル名を解釈する	DE…文字列を指す B…文字列の長さ	なし	なし	06	42F6
2E	LUCHK	次に使用するファイル番号を選択する	A…ファイル番号	A＝1…入力 A＝2…出力 A＝4…ランダム	すべて	06	4226
2F	LOPEN	OBJ, BTX用のファイルを OPENする	なし(予めDEVFNを呼び出す)	なし	なし	06	43DF
30	LOADF	LOPENで開いたファイルをロードする	HL…ロードアドレス	なし	なし	06	44FC
31	SAVEF	LOPENで開いたファイルをセーブする	(DE＋14H)…長さ(2バイト) (DE＋16H)…ロードアドレス (DE＋18H)…実行アドレス } OBJの時 A＝0…セーブする A≠0…ファイルを作るだけ	なし	なし	06	4509
32	(未使用)					00	003B
33	RWOPEN	BSD, BRD用のファイルを OPENする(予めDEVFNを呼び出す)	D…モード(1＝I, 2＝O, 4＝R, 8＝A) A…ファイル番号 E…ランダムファイル時レコード長/20H－1	ランダムファイル時…Z＝0…新ファイル, Z＝1…以前からあるファイル	すべて	06	431C
34	BGETL	BASICのINPUT文コンパチのGETL	なし	DE…文字列の入ったアドレス Cy＝1…SHIFT＋BREAK	すべて	36	41B4
35	INMSG	ファイルから1行入力する	DE…バッファアドレス	B…文字数 Cy＝1…EOF	HL, DE	06	4153
36	UNPUT	入力ファイルに1文字戻す(1バイトのみ可)	A…データ	なし	すべて	06	417B
37	PRSTR	ファイルに文字列を出力する	DE…文字列の入ったアドレス B…文字列の長さ	なし	すべて	06	4080
38	CLKL	ファイルをCLOSE, KILLする	A＝0…すべてのファイルが対象 A≠0…その番号のファイルが対象 B＝0…KILL(出力ファイルを登録しない)	なし	すべて	06	442C

RST 18H DEFB nnH のnnHの値 (リクエスト番号)	名称	機能	入力レジスタ	出力レジスタ	保存レジスタ	処理ルーチン	
						マップ	アドレス
38	CLKL	ファイルをCLOSE, KILLする	B≠0…CLOSE(出力ファイルを登録する)	なし	すべて	06	442C
39	DIR	ディレクトリをとる(DEVNMの後に呼び出す)。DIRはA=0とA≠0との2度呼び出すこと	A=0…ディレクトリを1000H～に読み出し，ソートとワイルド(DEVNM時に指定)の処理を行う A≠0…Aのファイル番号にディレクトリ文字列を出力する	なし	すべて	06	455C
3A	CHDIR	パスネームを登録する	なし(DEVNMの後に呼び出す)	なし	すべて	34	495D
3B	(未使用)					00	003B
3C	FINIT	BASICのINITと同じ	なし(DEVNMの後に呼び出す)	なし	なし	06	4725
3D	DELET	ファイル(サブディレクトリを含む)を削除する(DEVNMの後に呼び出す)	Cy=0…通常ファイル(KILL) Cy=1…サブディレクトリ(RMDIR)	なし	すべて	06	50C0
3E	RENAM	ファイル名を付け換える(旧ファイル名はDEVNMで登録しておく)	DE…新ファイル名を指す B…新ファイル名の長さ	なし	すべて	06	4F41
3F	LOCK	ファイル属性の制御を行う(DEVNMの後に呼び出す)	A=0…SET～, "″(ロック解除) A=1…SET～, "P″(ロックする) A=FF_H…ATTR$の取り出し	ATTR$の取り出し時は Cy=1…ファイルがない Cy=0, A=0…ロックされていない Cy=0, A=1…ロックされている	すべて	06	4F01
40	BRD	ランダムアクセスを行う	A=0…次にアクセスするレコード番号をDEレジスタで指定したものに変更する(これがない時は前回のレコード番号＋1になる) A=1…1レコード読み出す。結果はバッファに入る。ただしDE≠0とすると，そのアドレスに読み出せる A=2…1レコード書き込む。出力データはバッファに入れておくか，DE≠0とするとそのアドレスからBバイトの文字列を書き込む A=3…バッファからそのデータの一部を取り出す。DE…ストアしたいアドレス，B…バッファの何バイト目から取り出すか。C…何バイト分取り出すか。L…ファイル番号 A=4…バッファにデータをセットする。レジスタはA=3の時と同様 A=5…ファイルから何レコードもまとめてデータを取り出す(GLOAD用)。HL…アドレス，DE…バイト数ただし0100H, 0200H, 0400H, 0800H,	A=1，DE≠0の時…Bレジスタに文字列の長さが入る	なし	06	498E

RST 18H DEFB nnH のnnHの値 (リクエスト番号)	名称	機能	入力レジスタ	出力レジスタ	保存レジスタ	処理ルーチン	
						マップ	アドレス
40	BRD	ランダムアクセスを行う	1000Hのいずれか，BC＝先頭からのバイト数(DEレジスタの倍数になること) A＝6…ファイルに何レコードもまとめてデータを出力する(GSAVE用)。レジスタはA＝4の時と同様	A＝1，DE＋0の時…Bレジスタに文字列の長さが入る	なし	06	498E
41	FFUNC	ファイルの情報を得る	A＝0…LOC　A＝1…LOF A＝2…FPOS A＝3…1行中の位置	BC…結果	DE，HL	06	4014
42	DFUNC	デバイスの情報を得る(DEVNMの後に呼び出す，	A＝0…DEVF A＝1…DEVI A＝2…DEVO HL…アドレス，DE…長さ，BC…ブロック番号	DEVFの時，Cy＝0…DE＝Kバイトで示すフリーエリア，Cy＝1…実行できないデバイス	DEVF…DE以外 それ以外…すべて	06	4C54
43	ERCVR	エラー時のデバイス回復処理を行う	なし	なし	すべて	06	48A6
44	SWAP	SWAPファイルの読み出し・書き込みを行う	A＝0…現在のプログラムのセーブ。DE…アドレス，HL…長さ。この後で新ファイルをロードする A＝FFH…1日プログラムのロードを準備。この後でLOADFを呼び出す。			06	4CA8
45	CLS	グラフィック画面を消す	A…ぬりつぶしたい色(通常0)	なし	なし	02	6275
46	POSCK	座標がビューポート内にあるかどうかチェックする	(HL＋0)・(HL＋1)…X座標 (HL＋2)・(HL＋3)…Y座標	なし(エラー時はBASIC等のエラールーチンへジャンプする)	なし	02	4DAD
47	POSSV	グラフィックポインタをセット・読み出す	(HL＋0)・(HL＋1)…X座標 (HL＋2)・(HL＋3)…Y座標 セット時 Cy＝0…セット(POINT文相当) Cy＝1…読み出し(POINT(機能)相当)	DE…X座標 HL…Y座標 読み出し時	なし	02	4D9A
48	PSET	ドットをセット・リセットする	(HL＋0)…カラー　(HL＋1)…モード (HL＋2)・(HL＋3)…X座標 (HL＋4)・(HL＋5)…Y座標			02	4F24
49	LINE	一本線を描く	(HL＋0)…カラー (HL＋1)…モード (HL＋2)・(HL＋3)…開始X座標 (HL＋4)・(HL＋5)…開始Y座標 (HL＋6)・(HL＋7)…終了X座標 (HL＋8)・(HL＋9)…終了Y座標 (HL＋10)…b0＝0…実数 b0＝1…ラインスタイル b1＝0…新ラインスタイル b1＝1…以前のラインスタイル (HL＋11)・(HL＋12)…ラインスタイル	なし	なし	02	4F3C
4A	PATTR	パターン描画を行う (座標はPOSSVで設定しておく)	(HL＋0)…カラー (HL＋1)…モード (HL＋2)…背景色 (HL＋3)…データの長さ (HL＋4)・(HL＋5)…データのあるアドレス	なし(グラフィックポインタは自動的に移動する)	なし	02	555E

RST 18H DEFB nnH のnnHの値 (リクエスト番号)	名称	機能	入力レジスタ	出力レジスタ	保存レジスタ	処理ルーチン マップ	処理ルーチン アドレス
4A	PATTR	パターン描画を行う (座標はPOSSVで設定しておく)	(HL+6)…方向(0=上, 1=下) (HL+7)…高さ	なし(グラフィックポインタは自動的に移動する)	なし	02	555E
4B	BOX	長方形を描く	(HL+0)…カラー (HL+1)…モード (HL+2)・(HL+3)…開始X座標 (HL+4)・(HL+5)…開始Y座標 (HL+6)・(HL+7)…終了X座標 (HL+8)・(HL+9)…終了Y座標 (HL+10)…ラインモード(0…実線の箱のみ, 1…ラインスタイルの箱, 2…単色をぬりつぶす, 3…パターンでぬりつぶす) (HL+11)・(HL+12)…ラインスタイル(ラインモード=1の時のみ) (HL+11)・(HL+12)…パターンのアドレス (HL+13)…パターンの長 (ラインモード=3の時のみ)	なし	なし	02	53CE
4C	PAINT	ぬりつぶしを行う	(HL+0)・(HL+1)…X座標 (HL+2)・(HL+3)…Y座標 (HL+4)・(HL+5)…境界色テーブルのアドレス (HL+6)…境界色の数。ただしFFHとするとNOTモード (HL+7)…0…単一色でぬりつぶす 1…タイリングペイント (HL+9)…カラー(単一色時) (HL+8)・(HL+9)…パターンのアドレス (HL+10)…パターンの長さ タイリングペイント時	なし	なし	02	68C0
4D	CIRCLE	円を描く	(HL+0)…カラー (HL+1)…モード (HL+2)・(HL+3)…X座標 (HL+4)・(HL+5)…Y座標 (HL+6)・(HL+7)…半径 (HL+8)…b_0=0…開始点と中心の間に半径を描かない b_0=1… 〃 描く b_1=0…終了点と中心の間に半径を描かない b_1=1… 〃 描く (HL+9)~(HL+13)…開始角 (HL+14)~(HL+18)…終了角 (HL+19)~(HL+23)…比率 単精度実数	なし	なし	02	5976
4E	POINT	ドットの状態を調べる	(HL+0)・(HL+1)…X座標 (HL+2)・(HL+3)…Y座標	A=カラー Cy=0…ビューポート内, Cy=1…ビューポート外	なし	02	4EB0
4F	HCPY	ハードコピーをとる	(HL+0)・(HL+1)…開始X座標 (HL+2)・(HL+3)…開始Y座標 (HL+4)・(HL+5)…終了X座標 (HL+6)・(HL+7)…終了Y座標 (HL+8)・(HL+9)…ドットで表す左マージン (HL+10)…機能(0…文字のみ, 1…グラフィックのみ, 2…両方)	なし	なし	02	4183
50	DFKEY	ファンクションキー・キーバッファに文字列を定義する	HL…文字列のアドレス B…文字列のバイト数 A…番号(0=先行入力バッファ, 1~20=ファンクションキー)	なし	なし	34	594F

RST 18H DEFB nnH のnnHの値 (リクエスト番号)	名称	機能	入力レジスタ	出力レジスタ	保存レジスタ	処理ルーチン	
						マップ	アドレス
51	GINIT	グラフィック画面モードの初期化	Aレジスタ b_7=0…320ドット b_7=1…640ドット (X方向) b_6=0…200ドット b_6=1…400ドット (Y方向) b_5・b_4=00…4色モード b_5・b_4=01…16色モード b_5=1…256色モード	なし(G-CRTC, VIEW@, SCREEN, グラフィックポインタ, CBLOCK, PRIORITY, パレットが初期化される)	なし	02	4D32
52	VIEW	ビューポートを設定する	(HL+0)…b_0=0…ビューポート内はそのまま b_0=1…ビューポート内を消す b_1=0…枠を描かない b_1=1…枠を描く (HL+1)…消す時はその色(通常0) (HL+2)…枠を描く時はその色 (HL+3)…枠のモード (HL+4)・(HL+5)…X方向開始座標 (HL+6)・(HL+7)…Y方向開始座標 (HL+8)・(HL+9)…X方向終了座標 (HL+10)・(HL+11)…Y方向終了座標 Cy=0…上記設定が有効 Cy=1…デフォルトのVIEWに初期化する(上記設定は無用)	なし	なし	02	4DB7
53	SCREN	スクリーンモードを設定する	A=0…L=アクティブプレーン A=1…Lレジスタのb_0=0…標準グラフィック出力 b_0=1…拡張グラフィック出力 (320×200×16色のみ有効) b_1=0…0・2面を出力しない b_1=1…0・2面を出力する b_2=0…1・3面を出力しない b_2=1…1・3面を出力する A=2…L=アクティブプレーン A=3…L=第1出力プレーン H=第2出力プレーン A=FFH …スクリーンモードを初期化する(アクティブページ=0, 出力グラフ=標準0面, 全プレーンアクティブ, 第1出力プレーン=全プレーン, 第2出力プレーン=256色時のみ全プレーン)	なし	なし	02	5F5E
54	COLOR	パレットの設定	A=0…Cy=0…(HL+0)…データの組織(16色) (HL+1)パレット (HL+2)…カラー (これが続く) Cy=1…初期化する A=1…Cy=0…(HL+0)…データの組数(4096色) (HL+1)…パレット (HL+2)…緑成分 (HL+4)…赤成分 (これが続く)	なし	なし	02	7B96

RST 18H DEFB nnH のnnHの値 (リクエスト番号)	名称	機能	入力レジスタ	出力レジスタ	保存レジスタ	処理ルーチン	
						マップ	アドレス
54	COLOR	パレットの設定	(HL+4)…青成分 / Cy＝1…初期化する	なし	なし	02	7B96
55	SVMBL	シンボルを表示する(グラフィックに拡大文字を表示する) (座標はPOSSVで設定しておく)	(HL＋0)…カラー (HL＋1)…モード (HL＋2)…文字列の長さ (HL＋3)・(HL＋4)…文字列のアドレス (HL＋5)…Y方向倍率 (HL＋6)…X方向倍率 (HL＋7)…角度コード(0＝ノーマル,1=90°,2=180°,3=270°) (HL＋8)…0＝8×8フォント,1＝8×16フォント)	なし(グラフィックポインタは自動的に移動する)	なし	02	57DF
56	HVIEW	ハードウエアビューポートを設定する	(HL＋0)・(HL＋1)…X方向開始座標 (HL＋2)・(HL＋3)…Y方向開始座標 (HL＋4)・(HL＋5)…X方向終了座標 (HL＋6)・(HL＋7)…Y方向終了座標 Cy＝0…上記設定は有効 Cy＝1…デフォルトのVIEW@に初期化する (上記設定は無用)	なし	なし	02	661F
57	CREP	カラーリプレイス（置き換え）を行う	(HL＋0)・(HL＋1)…開始X座標 (HL＋2)・(HL＋3)…開始Y座標 (HL＋4)・(HL＋5)…終了X座標 (HL＋6)・(HL＋7)…終了Y座標 (HL＋8)・(HL＋9)…データテーブルのアドレス (HL＋10)…データテーブルの組数 ただしデータテーブルは旧カラー（1バイト），新カラー(1バイト）で1組	なし	なし	02	7CF3
58	GPUT	PUT@を行う	(HL＋0)・(HL＋1)…配列マッピングテーブルアドレス (HL＋2)・(HL＋3)・(HL＋4)…配列のアドレス (HL＋5)・(HL＋6)…配列のバイト数 (HL＋7)・(HL＋8)…配列に対するオフセットアドレス (HL＋9)・(HL＋10)…開始X座標 (HL＋11)・(HL＋12)…開始Y座標 (HL＋13)…モード	なし	なし	02	7A37
59	GGET	GET@を行う	(HL＋0)・(HL＋1)…配列マッピングテーブルアドレス (HL＋2)・(HL＋3)・(HL＋4)…配列のアドレス (HL＋5)・(HL＋6)…配列のバイト数 (HL＋7)・(HL＋8)…配列に対するオフセットアドレス (HL＋9)・(HL＋10)…開始X座標 (HL＋11)・(HL＋12)…開始Y座標	なし	なし	02	66CF

RST 18H DEFB nnH のnnHの値 (リクエスト番号)	名称	機能	入力レジスタ	出力レジスタ	保存レジスタ	処理ルーチン	
						マップ	アドレス
59	GGET	GETα行う	(HL＋13)・(HL＋14)…終了X座標 (HL＋15)・(HL＋16)…終了Y座標 (HL＋17)…カラー	なし	なし	02	66CF
5A	PENSE	ペンを設定する	A＝0…1×1ドットで描画 A≠0…指定のドットで描画 (HL＋0)～(HL＋7)…ペンパターン(A≠0のみ)	なし	なし	02	66C4
5B	MOVE	図形を移動する	(HL＋0)…カラー (HL＋1)…モード (HL＋2)・(HL＋3)…転送元X座標始点 (HL＋4)・(HL＋5)…転送元Y座標始点 (HL＋6)・(HL＋7)…X方向長さ (HL＋8)・(HL＋9)…Y方向長さ (HL＋10)・(HL＋11)…転送先X座標始点 (HL＋12)・(HL＋13)…転送先Y座標始点 (HL＋14)…0＝転送元は消す 1＝転送元はそのまま (HL＋15)…消す時のカラー(通常0)	なし	なし	02	71E6
5C	PRI	プライオリティを設定する	Cy＝0…設定 (HL＋0)・(HL＋1)…グラフィックと文字の優先順位 (HL＋2)…グラフィック同士の優先順位 Cy＝1…初期化	なし	なし	02	7CB5
5D	CBLOC	256色モードのカラーブロックを設定する	Cy＝0…設定(A＝データ(BASICコンパチ)) Cy＝1…初期化	なし	なし	02	7EC0
5E	CRTI	グラフィック画面の初期化	HL…文字列(00Hがエンドマーク) 文字列はINIT"CRT2:～"の～とコンパチ	なし	なし	02	684B
5F	COCNV	256色と512色の変換 (CBLOCKの変換)	Cレジスタ＝0…DE(512色系コード)→A(256色系コード)に変換 Cレジスタ＝1…A(256色系コード)→DE(512色系コード)に変換	AまたはDE	AまたはDE以外	02	7EDC
60	HROLL	ハードウエアスクロールを行う	Cy＝0…DE＝画面表示開始Y位置 Cy＝1…初期化	なし	なし	02	7E85
61	(未使用)					00	003B
62	(未使用)					00	003B
63	GRINT	SCREN, VIEW, HVIEW, COLOR (16, 4096共), PRI, CBLOC, HROLL等を初期化	なし	なし	なし	36	553C
64	TINIT	テキスト画面を初期化	Cy＝0…Aレジスタのb_4＝0…通常スクロール b_4＝1…スムーススクロール b_3＝0…8×8フォント b_3＝1…8×16フォント b_2・b_1＝00…25行	なし	なし	36	4532

RST 18H DEFB nnH のnnHの値 (リクエスト番号)	名称	機能	入力レジスタ	出力レジスタ	保存レジスタ	処理ルーチン	
						マップ	アドレス
64	TINIT	テキスト画面を初期化	b_2・b_1＝01…20行 b_2・b_1＝10…12行 b_0＝0…80桁 b_0＝1…40桁 Cy＝1…A＝0…8色モード A＝1…256色モード	なし	なし	36	4532
65	MSGX	文字列を画面に表示する（コントロールコードはシンボル表示）	DE＝文字列(00Hがエンドマーク)	なし	すべて	00	011C
66	TMOD	テキスト画面のモードを設定	C＝0…A＝0…KMODE 0（セミグラフィック） A≠0…KMODE 1（シフトJIS） C＝1…A＝0…KLIST 0(Fキー表示せず) A＝1…KLIST 1(Fキー表示する) A＝2…KLIST 2(Fキーと時計を表示) C＝2…A＝0…常にカーソルを消す A＝1…常にカーソルをつける A＝2…キー入力時のみカーソルをつける C＝3…A＝0…スクロール可能 A≠0…スクロール禁止 C＝4…A＝0…文字の背景色は透明 A≠0…文字の背景色は黒 C＝5…A＝0…モノクロディスプレイを通常表示 A≠0…モノクロディスプレイを反転表示 C＝6…Cy＋Aの9ビットで，カラー0の実表示色を設定 C＝7…A＝0…モノクロディスプレイにグラフィックも表示する A≠0…モノクロディスプレイは文字のみ表示	なし	なし	36	5A07
67	GPCG	文字のフォントを得る	DE…文字コード(BASICのCGPAT$と同じ) B…PCG番号（0～3以外ならカラー）	DE…フォントの入っているアドレス	なし	36	5CCC
68	SPCG	PCGを定義する	DE…文字コード(BASICのDEFCHR$と同じ) B…PCG番号（0～3以外ならカラー） 1000H～にフォント	なし	なし	36	5CAC
69	TGET	画面の指定した位置から文字列を入力。ただしシフトJISの途中ではシフトJISの2バイト目も取り込む。画面の終了にかかると，そこで止める。 255バイトまで入力。	L…X位置 H…Y位置 C…バイト数(希望)	C…バイト数(実際に入力された文字の長さ) DE…文字列の入っているアドレス	なし	36	4501

RST 18H DEFB nnH のnnHの値 (リクエスト番号)	名称	機能	入力レジスタ	出力レジスタ	保存レジスタ	処理ルーチン	
						マップ	アドレス
6A	TSCRN	文字画面のページング制御	Aのb_1・b_0＝00…アクセスページは変更せず b_1・b_0＝01…ページ1をアクセスする b_1・b_0＝10…ページ2をアクセスする b_3・b_2＝00…表示ページは変更しない b_3・b_2＝01…ページ1のみ表示 b_3・b_2＝10…ページ2のみ表示 b_3・b_2＝11…1＞2で合成表示	なし	なし	36	5929
6B	TATR	既に表示されている文字のアトリビュート部のみを変更する (CREV@等に相当)	C＝0…A(64色系)の色に変更 C＝1…A＝0ならブリンク解除, A≠0ならブリンク C＝2…A＝0ならリバース解除, A≠0ならリバース L…X開始位置　H…Y開始位置 E…X長さ　D…Y長さ	なし(注.これから表示しようとする文字の色等は,05F4H, 05F5Hに直接書き込むこと)	なし	36	5640
6C	TWID	表示桁・行数を変更する	Aのb_0＝0…80桁 b_0＝1…40桁 b_2・b_1＝00…25行 b_2・b_1＝01…20行 b_2・b_1＝10…12行	なし	なし	36	586A
6D	TWARE	テキストウィンドウを設定する (CONSOLE相当)	L…開始X位置　H…開始Y位置 E…X方向長さ　D…Y方向長さ	なし	なし	36	56E7
6E	TDARE	ハードウエアテキストビューポートを設定する (CONSOLE@相当)	L…開始X位置 H…開始Y位置(0～199) 400ライン時もこの値で指定する E…X方向長さ D…Y方向長さ(0～200) 400ライン時もこの値で指定する	なし	なし	36	5785
6F	CMOV	カーソルを移動させる	L…X位置 H…Y位置	なし(注.現在のカーソル位置は05E2H, 05E3Hを直接見ること)	なし	36	5C7D
70	TCLR	画面を（部分）クリアする	Aのb_1・b_0＝00…全テキスト画面をクリア b_1・b_0＝01…アクセスページのみクリア b_1・b_0＝10…アクセスページの中の指定した部分だけをクリアする。また,行継続フラグはクリアしない b_7＝0…カーソルをTWAREで指定した左上の座標に移動する b_7＝1…カーソルは移動しない L…開始X位置　H…開始Y位置 E…X方向長さ　D…Y方向長さ (H, L, D, EはAのb_1・b_0＝10の時のみ有効)	なし	なし	36	5C3B

RST 18H DEFB nnH のnnHの値 (リクエスト番号)	名称	機能	入力レジスタ	出力レジスタ	保存レジスタ	処理ルーチン	
						マップ	アドレス
71	TSCRL	画面全体をスクロールする	A＝0…上方向にスクロール A≠0…下方向にスクロール	なし	なし	36	5116
72	CRPUT	PUT%を行う	(HL＋0)・(HL＋1)…配列マッピングテーブルアドレス (HL＋2)・(HL＋3)・(HL＋4)…配列のアドレス (HL＋5)・(HL＋6)…配列のバイト数 (HL＋7)・(HL＋8)…配列に対するオフセットアドレス (HL＋9)…開始X位置 (HL＋10)…開始Y位置 (HL＋11)…終了X位置 (HL＋12)…終了Y位置	なし	なし	36	4CED
73	CRGET	GET%を行う	(HL＋0)・(HL＋1)…配列マッピングテーブルアドレス (HL＋2)・(HL＋3)・(HL＋4)…配列のアドレス (HL＋5)・(HL＋6)…配列のバイト数 (HL＋7)・(HL＋8)…配列に対するオフセットアドレス (HL＋9)…開始X位置 (HL＋10)…開始Y位置 (HL＋11)…終了X位置 (HL＋12)…終了Y位置	なし	なし	36	4C95
74	SETCG	PCG 0 に決められたフォントをセット	A＝0…MZ-2500モードのフォントをセット A≠0…MZ-80B・2200モードのフォントをセット	なし	なし	36	55B5
75	?TEXT	画面の指定位置のANK，漢字等の情報を取り出す	L…X位置 H…Y位置	Cy＝0…ANK(半角)，PCG Cy＝1，z＝0…漢字の左半分 z＝1…漢字の右半分	なし	36	4C0F
76	CURBL	現在のカーソルの状態を制御する	A＝0…カーソルが既に出ている時に使い，インサートモードに対応してタイマを速く見るようにする A＝1…カーソルをつける(LOCATE，0では無視) A＝2…カーソルを消す A＝3…カーソルの状態を元に戻す	なし	なし	36	4054
77	TSUB	雑用処理を行う	C…機能番号， B＝サブ機能番号	なし	なし	36	5ABB
78	RS232	RS-232Cを制御する	C…対象となるチャンネル番号(0…A，1…B) A＝0…(IX＋0)…ボーレイト(0＝19200～8＝75) (IX＋1)…パリティ(0＝なし，1＝奇，2＝偶) (IX＋2)…語長(1＝7，3＝8) (IX＋3)…ストップビット(0＝1，1＝1.5，2＝3) A＝1…Bのb_5＝0…LFは無視しない b_5＝1…LFは無視する b_4＝0…7ビット時SI/SOしない	左にないものはなし	なし	34	697E

RST 18H DEFB nnH のnnHの値 (リクエスト番号)	名称	機能	入力レジスタ	出力レジスタ	保存レジスタ	処理ルーチン	
						マップ	アドレス
78	RS232	RS-232Cを制御する	b_4＝1…7ビット時SI SOする b_3・b_2＝00…送信時WAITしない b_3・b_2＝01…CSによりWAITする b_3・b_2＝10…X-ON OFFによりWAITする b_1＝0…OPEN時ERDRチェック無 b_1＝1…OPEN時ERDRチェック有 b_0＝0…全二重 b_0＝1…半二重 A＝2…OPENする(エラー時はCy＝1でリターンする) A＝3…CLOSEする A＝4…Bレジスタの内容を送信する A＝5…データを受信してBレジスタに入れる(エラー時Cy＝1，バッファが空の時Z＝1，Eのb_6＝フレミングエラー・b_5＝オーバーランエラー，b_4＝パリティエラーの有無(Cy＝1の時有効 1＝エラーあり) A＝6…エラーをリセットする A＝7…半二重時に使い，B＝0…RSをOFF，B＝FFH…RSをONする A＝8…B＝0…ERをOFF B＝1…ERをON A＝9…B＝0…RRをOFF B＝1…RRをOFF A＝10…Bx10msだけBREAKを送出 A＝11…制御信号を読み出す(Bレジスタに結果が入り，CI→b_7，CD→b_6，CS→b_5，DR→b_3。ただしON＝1，OFF＝0) A＝12…受信バッファをクリアする A＝13…バッファにたまった受信データのバイト数をBレジスタに入れて返す	左にないものはなし	なし	34	697E
79	RTC	カレンダクロック(RP-5C15)を制御する	A＝0…時刻を読み出す(DE＝結果を入れたいアドレス) A＝1…時刻をセットする(DE＝データの入っているアドレス) A＝2…日付けを読み出す(DE＝結果を入れたいアドレス) A＝3…日付けをセットする(DE＝データの入っているアドレス) A＝4…アラーム時刻を読み出す(DE＝結果を入れたいアドレス) A＝5…アラーム時刻をセットする(DE＝データの入っているアドレス) A＝6…曜日を読み出す(E＝結果) A＝7…曜日をセットする(E＝データ)	左にないものはなし なお，時刻は，(DE＋0)…1秒の位，(DE＋1)…10秒の位，(DE＋2)…1分の位，(DE＋3)…10分の位，(DE＋4)…1時の位，(DE＋5)…10時の位，日付けは(DE＋0)…1日の位，(DE＋1)…10日の位，(DE＋2)…1月の位，(DE＋3)…10月の位，(DE＋4)…1年の位，(DE＋5)…10年の位で，各桁，00H～09H以内のデータになる。曜日は，0＝日，1＝月，2＝火，3＝水，4＝木，5＝金，6＝土。	すべて(A＝6ではE以外すべて)	34	5744

RST 18H DEFB nnH のnnHの値 (リクエスト番号)	名称	機能	入力レジスタ	出力レジスタ	保存レジスタ	処理ルーチン	
						マップ	アドレス
7A	MOUSE	マウスを制御する	A＝0…マウスの使用を開始する（RS-232CのBチャンネル使用時はCy＝1となり使用できない） A＝1…HL…マウスX座標 DE…マウスY座標 C…カーソルスイッチ（0以外でON） A＝2…HL…マウスカーソル形状を指す A＝3…E…X方向移動比率 A＝4…E…Y方向移動比率 A＝5…HL…マウス移動範囲X開始座標 DE…マウス移動範囲Y開始座標 IX…マウス移動範囲X終了座標 IY…マウス移動範囲Y終了座標 A＝6…マウスカーソルのカラー A＝7…マウスの使用を停止 A＝80H…現在のX座標→HL A＝81H…現在のY座標→HL A＝82H…Eで示すボタン（0か1）の状態→HL（0000H＝OFF FFFFH＝ON） A＝83H＝Eで示すボタンが押された時のX座標→HL A＝84H…Eで示すボタンが押された時の座標→HL A＝85H…Eで示すボタンが離された時のX座標→HL A＝86H…Eで示すボタンが離された時のY座標→HL A＝87H…前回これを呼び出した時からのX方向移動距離→HL A＝88H…前回これを呼び出した時からのY方向移動距離→HL	A＜80H…なし A≧80H…HL＝結果	なし	07	46F7
7B	IOSUB	ファイル関係用の雑用処理	A…機能番号			34	4858
7C	DVCAL	マップ 34H・35H のROMを直接呼び出す	IX…アドレス	（ものによって違う）	（ものによって違う）	34 (00)	003E
7D	FDRES	ディスクドライブを初期化する（モーターOFF等）	なし	なし	なし	34	628A
7E	CMT	カセットデッキを制御する	A＝0…Bのb_0＝0…ライン，1＝マイク b_1＝0…ボタン有効，1＝禁止 b_2＝0…オートプレイする，1＝しない b_3＝0…オートリワインドする，1＝しない b_4＝0…APSS・A＝2の処理中，CPUは待たない b_4＝1…それらが終了するまで待つ A＝1…カセットの状態を得る（結果→A） A＝2…BASICのCMT＝と同様。	（A＝1で呼び出し時） Aのb_0～b_4…A＝0でセットしたライン/マイク等 b_5＝0…カセット動作中，1＝停止中 b_6＝0…ツメがある，1＝ツメがない b_7＝0…テープがセットされている，1＝いない （A＝2時） Cy＝1…機能2～13で，ツメが折られていてエラーとなった （A＝3時） Cy＝1…ツメが折られていてエラーとなった	なし	34	7129

RST 18H DEFB nnH のnnHの値 (リクエスト番号)	名称	機能	入力レジスタ	出力レジスタ	保存レジスタ	処理ルーチン アップ	処理ルーチン アドレス
7E	CMT	カセットデッキを制御する	ただし，B＝0～13の機能番号，HL＝第1パラメータ（0.1秒単位の時間），DE＝第2パラメータ（0.1秒単位の時間） A＝3…デジタル側にAPSS用のマーキングをする A＝4…APSSを行う。B＝－128～127（80H～FFH，00H～7FH）で方向と回数を示す A＝5…第1ブロック（インフォメーションブロック）をロード A＝6…第2ブロック（データブロック）をロード A＝7…A＝5の後のファイル名の合致確認，表示処理を行う	（A＝1で呼び出し時） Aのb_0～b_4…A＝0でセットしたライン/マイク等 b_5＝0…カセット動作中，1＝停止中 b_6＝0…ツメがある，1＝ツメがない b_7＝0…テープがセットされている，1＝いない （A＝2時） Cy＝1…機能2～13で，ツメが折られていてエラーとなった （A＝3時） Cy＝1…ツメが折られていてエラーとなった	なし	34	7129
7F	VOICE	音成合成を行う	A＝0…VOICE文コンパチの発音 B…1～3でスピード E…1（棒読み），2（数値読み） IX…文字列を指す（ここでは0DHがエンドマークなので注意） A＝1…マシン語コンパチの発音 MZ-1X08の「外部ROM」のデータがそのまま使用できる。VOICEの固定データは＋5CHすれば使える。 HL…文字列を指す（0DHがエンドマーク）	なし	なし	07	4000

〈ファイル番号の特例〉

88H…CRT：とKB：
89H…LPT：
） OPENせずに使えます

〈グラフィックのモード〉

0…PSET
1…OR
2…XOR
3…AND
4…FPSET
5…SCR
） PATTRのみ
6…NOT（補色） GPUTのみ

〈グラフィックのカラー〉

4色モード…0～3のパレットコード

16色モード…00H～0FHのパレットコード

256色モード…00H～FFHのカラーコード

〈バッファについて〉

グラフィックでは(HL＋0)～がパラメータになっていますが，その受け渡し用に0CC0H～0CEFHが確保されているので，ここを使うと便利です。

《第8-4表》RST 28Hによる浮動小数点ファンクションコール

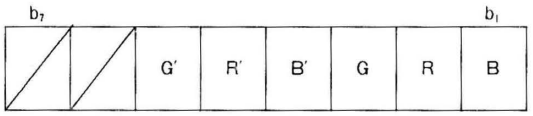
RST 28H DEFB nnH のnnHの値 (リクエスト番号)	名称	機能	パラメータ	処理アドレス (マップはいずれも36H・37Hです)
00	INT	その値を越えない最大の整数を計算する([X])	(HL)＝値・B＝型(2＝整数・5＝単精度実数・倍精度実数)→(HL)＝結果	680A
01	ABS	絶対値を計算する(|X|)	(HL)＝値・B＝型(2＝整数・5＝単精度実数・倍精度実数)→(HL)＝結果	67F6
02	SIN	正弦を計算する(sin X)	(HL)＝値・B＝型(2＝整数・5＝単精度実数・倍精度実数)→(HL)＝結果	6AA0
03	COS	余弦を計算する(cos X)	(HL)＝値・B＝型(2＝整数・5＝単精度実数・倍精度実数)→(HL)＝結果	6A87
04	TAN	正接を計算する(tan X)	(HL)＝値・B＝型(2＝整数・5＝単精度実数・倍精度実数)→(HL)＝結果	6CAF
05	LN	自然対数を計算する(log eX)	(HL)＝値・B＝型(2＝整数・5＝単精度実数・倍精度実数)→(HL)＝結果	6EB4
06	EXP	自然対数のべき乗を計算する(e^X)	(HL)＝値・B＝型(2＝整数・5＝単精度実数・倍精度実数)→(HL)＝結果	6D7A
07	SQR	平方根を計算する($\sqrt{X}$)	(HL)＝値・B＝型(2＝整数・5＝単精度実数・倍精度実数)→(HL)＝結果	693B
08	RND	乱数を発生する	(HL)＝値・B＝型(2＝整数・5＝単精度実数・倍精度実数)→(HL)＝結果	6D22
09		(未使用)		
0A	ATN	逆正接を計算する($\tan^{-1}$ X)	(HL)＝値・B＝型(2＝整数・5＝単精度実数・倍精度実数)→(HL)＝結果	69C9
0B	SGN	符号部だけを取り出す	(HL)＝値・B＝型(2＝整数・5＝単精度実数・倍精度実数)→(HL)＝結果	6CC7
0C	LOG	常用対数を計算する(log 10X)	(HL)＝値・B＝型(2＝整数・5＝単精度実数・倍精度実数)→(HL)＝結果	6EAB
0D	FRAC	小数部のみを取り出す	(HL)＝値・B＝型(2＝整数・5＝単精度実数・倍精度実数)→(HL)＝結果	67D1
0E	PAI	円周率の整数倍を計算する(Xπ)	(HL)＝値・B＝型(2＝整数・5＝単精度実数・倍精度実数)→(HL)＝結果	6D00
0F	RAD	度をラジアンに変換する(Xπ÷180)	(HL)＝値・B＝型(2＝整数・5＝単精度実数・倍精度実数)→(HL)＝結果	6CFB
10	FAC	階乗を計算する(X!)	(HL)＝値・B＝型(2＝整数・5＝単精度実数・倍精度実数)→(HL)＝結果	68E6
11	DEG	ラジアンを度に変換する(180X÷π)	(HL)＝値・B＝型(2＝整数・5＝単精度実数・倍精度実数)→(HL)＝結果	6CF2
12	FIX	整数部のみを取り出す	(HL)＝値・B＝型(2＝整数・5＝単精度実数・倍精度実数)→(HL)＝結果	6798
13	CINT	値を四捨五入して整数化する	(HL)＝値・B＝型→B＝結果の型・(HL)＝結果	5DF4
14	CSNG	値を単精度実数に変換する	(HL)＝値・B＝型→B＝結果の型・(HL)＝結果	5E08
15	CDBL	値を倍精度実数に変換する	(HL)＝値・B＝型→B＝結果の型・(HL)＝結果	5E40
16～1F		(未使用)		
20	ADD	加算を行う(X＋Y→X)	(HL)＝被加数・B＝型, (DE)＝加数・C＝型→(HL)＝結果・B＝型	5E95
21	SUB	減算を行う(X－Y→X)	(HL)＝被減数・B＝型, (DE)＝減数・C＝型→(HL)＝結果・B＝型	5EB6
22	MUL	乗算を行う(X×Y→X)	(HL)＝被乗数・B＝型, (DE)＝乗数・C＝型→(HL)＝結果・B＝型	5EDB
23	DIV	除算を行う(X÷Y→X)	(HL)＝被除数・B＝型, (DE)＝除数・C＝型→(HL)＝結果・B＝型	5F04
24	CMP	比較を行う(X－Y→捨てる)	(HL)＝被減数・B＝型, (DE)＝減数・C＝型→(HL)＝結果・B＝型	5F4C
25	MOD	整数剰余を計算する(X÷Yの余り→X)	(HL)＝被除数・B＝型, (DE)＝除数・C＝型→(HL)＝結果・B＝型	5F2F
26	YEN	整数除算を行う([X÷Y]→X)	(HL)＝被除数・B＝型, (DE)＝除数・C＝型→(HL)＝結果・B＝型	5F3F
27	POWER	べき乗を計算する(X^Y→X)	(HL)＝被べき乗数・B＝型, (DE)＝べき数・C＝型→(HL)＝結果・B＝型	6832
28	NEG	符号を反転させる(－X→X)	(HL)＝値・B＝型→(HL)＝結果・B＝型(－32768の時は変わるため)	5E65
29	RNDMIZ	乱数を初期化する(RANDOMIZE X)	HL＝値→結果なし	6D53
2A	INT0	値を整数化する(－32768～＋32767)	(HL)＝値・B＝型→HL＝結果	7057
2B	INT2	値を整数化する(－32768～＋65535)	(HL)＝値・B＝型→HL＝結果	703D
2C	FLT0	値を実数化する(－32768～＋32767)	HL＝結果をストアしたいアドレス・DE＝値→(HL)＝結果	70A6
2D～2F		(未使用)		
30	C.FLAS	文字列を浮動小数点(バイナリ)に変換する	(DE)＝文字列・HL＝結果をストアしたいアドレス→(HL)＝結果・B＝型	7129

RST 28H DEFB nnH のnnHの値 (リクエスト番号)	名 称	機 能	パ ラ メ ー タ	処理アドレス (マップはいずれも36H・37Hです)
31	C.NMFL	浮動小数点を文字列に変換する (STR$用)	(HL)＝値・B＝型→DE＝文字列のあるアドレス・(DE)＝文字列	7548
32	C.ASFL	浮動小数点を文字列に変換する (LIST用)	(HL)＝値・B＝型→DE＝文字列のあるアドレス・(DE)＝文字列	756A
33	USING	書式を指定して浮動小数点を文字列に変換したりする	(DE)＝値・A＝型(文字可)・BC＝文字列をストアしたいアドレス・(05A2H)＝書式指定文字列(BASICコンパチ)→(BC)～結果の文字列	7745
34	C.FLHX	多桁の16進数文字列を整数・浮動小数点に変換	(DE)＝文字列・HL＝結果をストアしたいアドレス→(HL)＝結果・B＝型	70E0
35	C.HXFL	整数・浮動小数点を多桁の16進数文字列に変換	(HL)＝値, B＝型, マップ01Hの(247EH)＝結果の桁数→DE＝文字列のあるアドレス, (DE)＝文字列	748C
36	C.NBFL	浮動小数点→文字列 (結果の桁数指定可)	(HL)＝値, B＝型, A＝指数表現の有無(0＝なし, 1＝あり), マップ01Hの(247EH)＝結果の桁数→DE＝文字列のあるアドレス, (DE)＝文字列	75AF
37	C.FXAS	結果の型を指定する文字列から浮動小数点への変換	(DE)＝文字列・HL＝結果をストアしたいアドレス・B＝型→(HL)＝結果	7123

単に値または浮動小数点となるものでも，型をBレジスタ等で指定するものは，浮動小数点だけでなく整数もあり得ます。

《第8-5表》カーネル部主要サブルーチン・ワークエリア

アドレス	機 能
0000	リセット動作
0018	IOCS(OS)ファンクションコールを行う
001E 0023	) BREAKキーの高速チェック(SHIFT＋BREAKならCy＝Z＝1)
0028	浮動小数点ファンクションコールを行う
003B	汎用のRET(未使用のコール等はここにジャンプして来る)
003D	JP (HL) (CALL 003DHとすると，CALL (HL)として使える)
003E	JP (IX) (CALL 003EHとすると，CALL (IX)として使える)
0040	RETI
0042	AF・BC・DE・HL・IXの自動PUSH～POP(サブルーチンの最初でCALL 0042Hとすると，その先でこれらのレジスタを壊しても，RETする時にきちんと元の値に元る)
004E	BC・DE・HL・IXの自動PUSH～POP(Aレジスタに結果を入れて帰りたい時に使う)
005C	CALL 005CH・DEFB nnH・DEFB ??H…とすると，??H以降とAレジスタの値とをnnHバイト分まで比較する。一つでも合致したら，元のルーチンのテーブルの次 (CALL 005CHのCALLのあるアドレス＋nnH＋2) にリターンする。 結果はZフラグ
006E	LD DE, (HL) (HLは＋2)
0073	LD DE, (HL) (HLは変化せず)
0078	(HL)～(HL＋B－1) ←0
0079	(HL)～(HL＋B－1) ←Aレジスタの値
007E	(DE)～(DE＋B－1) ←0
007F	(DE)～(DE＋B－1) ←Aレジスタの値
0084	(HL)→(DE)にBバイト転送
008B	(DE)→(HL)にBバイト転送
0092	LD HL, (HL)
0099	ADD HL, A
009E	INC HL後スペースカット
009F	スペースカット ((HL)以降のスペースを無視する)

アドレス	機　　能
00A5	BREAK処理
00A7	BREAK処理
00AA	ファイルネームを表示するエラー処理
00AC	ファイルネールを表示しないエラー処理
	ファンクションコールでエラー等があると，フラグで呼び出したルーチンに知らせるのではなく，ここに飛んで来て，BASICやモニタに戻る
00BB	リセット処理(マップをセットして，059EHにあるベクタにジャンプ)
00DC	0588H～058AHにあるデータで2000H～7FFFHをマッピングする(リセット・割り込み用)
00E9	YM-2203のポートBからデータを入力する
00F4	Aレジスタがシフト JISの1バイト目ならCy＝0
0102	COUNT（RST 18H・DEFB 17Hの処理ルーチン)。DEVFN等，Bレジスタに長さをセットするファンクションコールを行う前にはこれを利用すると便利。
0110	CRTMS(RST 18H・DEFB 05Hの処理ルーチン)。条件RSTはないので，CALL C，0110Hのようにも使える。
011C	MSGX(RST 18H・DEFB 65H)の処理ルーチン)。同様に使える。
0126	BREAKチェック(RST 18H・DEFB 0EHや001E・0023Hから飛んで来る)
0139	HL×DE→HL（符号なし）
0153	HL÷DE→HLあまりDE（符号なし）
0185	RST 18Hの処理ルーチン
01A0	RST 28Hの処理ルーチン
01EB	インターバルタイマ割込み
01F3	プリンタ割込み
01F9	V(垂直同期)割込み
01FF	RTC(RP-5C15，カレンダクロック）アラーム割込み
0205	Z-80B SIO割込み
0254	RTC割込みの処理ルーチン
028A	0C20H～のサブルーチンを，2000H～FFFFHを所定のマッピングに変えた後で呼び出す(メモリのロード，セーブ等で使われている)
02C4	割込みテーブル（02F6H以降は使用済）
0300	IOCSファンクションコールのジャンプテーブル。実行アドレス(2バイト)・マップ(1バイト)の順で80H組並ぶ。なお，マップは指定のものが4000H～5FFFHに，＋1のものが6000H～7FFFHに割当てられ，実行される。
0480	浮動小数点ファンクションコールのジャンプテーブル(マップは常に36H・37Hを使用するので，実行アドレス2バイトが，38H組並ぶ)
04F0	～04FFは未使用なので，浮動小数点ファンクションコールを拡張しても良い
0518	浮動小数点の0
0520	浮動小数点の1
0528	浮動小数点の65536
0530	浮動小数点のπ
0538	～053Fは未使用
0540	MAP00H～2FHの使用情況
0570	システムのマッピング
0578	モニタのマッピング
0580	BASICのPEEK，POKE等のマッピング
	本文参照
0588	(～058Aまで)割込み・リセット時のマッピング
059C	リセット時のジャンプ先(マップは0588H～058AHで指定)。初期値は76FFHでIPL，BASIC時は7D01Hになっている
059E	エラー・BREAK時のジャンプ先(マップは0570H～0577Hで指定)。初期値は7C2CHでIPLの「Loading error」に，モニタでは7C81Hに，BASICでは2545Hになっている。
05E2	カーソルX位置
05E3	カーソルY位置
05E4	テキストウインドウ左上X位置
05E5	テキストウインドウ左上Y位置
05E6	テキストウインドウ右下X位置
05E7	テキストウインドウ右下Y位置
05E8	横桁数－1（39か79）
05E9	縦桁数（スムーススクロール時10，18，23。通常11，19，24）
05EA	インサートモードフラグ（0はノーマル。≠0ならインサートモード）
05EB	テキストアクティブページ（0か1）
05EC	スクロールフラグ（b_0＝0…ライン／1＝スムース，b_1＝0…スクロール可能／禁止）
05F4	文字表示色。64色系で，  となる（8色時はb_3～b_5は無効）

アドレス	機　　能
05F5	文字表示属性 b_7 … b_0 0：ブリンクしない／リバースしない／00…PCG 0 か一般文字 01…PCG 1 10…PCG 2 11…PCG 3（b_3=0時のみ有効）／PCG 0 ～ PCG 3 単独使用か一般文字／一般文字（漢字含む）を使用 1：ブリンクする／リバースする／カラー PCG／PCG 0 を使用 となっている。b_2は，b_5～b_3=000の時のみ有効 （この2バイトは、直接変更して良い）
05F8	ディスプレイモード。TINIT(64H)で指定したものがそのまま入る

《第8-6表》BASIC-M25内部解析

アドレス	機　　能
2000	コールドスタート
2003	プログラム使用ブロック数
2004	プログラムのマッピング（6バイト…48Kまでだから）
2012	文字(配列)変数使用ブロック数
2013	文字(配列)変数のマッピング（6バイト）
2023	マシン語領域の使用ブロック数
2024	マシン語領域のマッピング（6バイト）
202E	数値配列変数の使用ブロック数
202F	数値配列変数のマッピング(30Hバイト)
205F	一般変数のマップ（1ブロックのみなので1バイト）
2068	現在実行中の行番号（2バイト）
206A	現在実行中の行の始まるアドレス（2バイト）
2089	「.」で指すエディタが注目している行番号
20B0	「OK」を表示してコマンド待ちへ
20C0	「OK」と表示せずにコマンド待ちへ
215C	REM
2164	1行実行
21B7	次が：か1行の終了かをテスト
222B	AUTO
2273	AUTO実処理(行番号発生)
22DE	RUN
22FF	END
231B	NEW
232E	CLR，単なるCLEAR
23D2	DELETE
2482	syntax error(エラー1)
｝	｝（3バイトおきにエラー2～エラー26）
24CD	too many nesting error(エラー27)
24D0	not open error
24D3	field error
24D6	open mode error
24D9	device name error
24DC	logical number error
24DF	system id error
24E2	can't execute error
24E5	file name error

（主要ワークエリア）

アドレス	機　　能
24E8	file mode error
24FB	'in', 0
2500	'＊Break', 0
2507	'＊Stop', 0
250D	'＊Step', 0
2513	'OK', 0
253A	ERROR
25EC	STOP
2633	CONT
266B	デフォルトのカラーをセット
2712	INTERVAL
28B4	FOR～NEXT等をアドレス化する
29BC	行番号を得る(DE＝アドレス)…一般用
2A21	行番号を得る(DE＝行番号)…LIST等用
2AE5	RENUM
2C7C	1語読みとばす
2CE8	DATA
3073	LSET
3074	RSET
3357	次に「：」か1行終了となることをチェックする(違えばsyntax error)
3363	次に「＝」が来ることをチェックする(違えばsyntax error)
3368	次に「(」が来ることをチェックする(違えばsyntax error)
336D	次に「)」が来ることをチェックする(違えばsyntax error)
3372	次に「,」が来ることをチェックする(違えばsyntax error)
3379	次に「,」が来るかどうかをテストする
337F	CALL 337FH・DEFB nnHとして，次にnnHが来るかどうかテストする
3387	CALL 3387H・DEFB nnHとして，次にnnHが来ることをチェックする
3460	4000H～FFFFHをBASICのプログラム領域にする
3487	A000H～DFFFHにBASICのプログラムを映し出す
34F9	SEARCH（命令）
3516	LIST
3604	1行分のリストを表示(行番号は空色で表示)
366C	RESTORE
3717	LOAD
373B	RUNファイル名処理
2753	CHAIN
37E1	MERGE
3921	VERIFY
392E	SAVE
3A5C	BLOAD
3BE5	BSAVE
3C2A	CALL
3C68	USR
3CD9	DEF USRで設定されたサブルーチンのテーブル(2×10バイト)
3CFB	EDIT
4004	スタックオーバーフローチェック
4060	GOSUB
4085	RETURN
40BA	GOTO
410D	RESUME
416D	ON ERROR GOTO
419D	IF
41CD	ELSE(従来コンパチ)
4227	ELSE(ブロックIF中)
423C	ELSE IF(ブロックIF中)
4256	ELSE(ブロックIF中で，ELSE IF以外)
4265	END IF

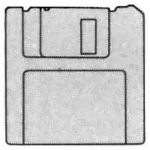
アドレス	機　　能
428A	WEND
42A3	WHILE
42CA	REPEAT
42E3	UNTIL
430B	FOR
43ED	NEXT
440D	NEXTで，整数変数の時の実処理
4412	NEXTで，単精度実数変数で整数を扱っている時の実処理
4472	NEXTで，単精度実数変数で実数を扱っている時の実処理
44BF	PRINT
4589	PRINT USING
4630	WRITE
4661	INPUT
4743	READ
47C4	LINE INPUT
4840	ファイル番号・プリンタへの出力かどうかをテストし，ロジカルユニット番号に変換する
4878	OPEN
48F6	KILL
4902	KILL＃（ファイルを登録せずキャンセル）
4917	すべてのファイルをKILL(ファイルを登録せずキャンセル)
492A	SET
4953	INIT
495D	FILES
4972	KILL ファイルネーム
4973	RMDIR
497C	NAME
4989	CHDIR
49A9	MKDIR
49AF	EOF
49CC	LPOS
49D7	PWDS
49E9	LOC
49EB	LOF
49EE	FPOS
4A25	DEVF
4A51	DEVIS
4A83	DEVOS
4AE8	ATTRS
4B0E	INKEYS
4B1B	INPUTS
4BA1	PUT
4BE0	LPOUT
4BFF	予約語テーブル(中間コード80H～FDH用)
4D78	予約語テーブル(中間コードFEH＋nnH用)
4F15	予約語テーブル(中間コードFFH＋nnH用)
50A6	予約語テーブル(特例入力)
50CD	50A6H～50CCHの語の展開(4バイトずつ)…L？→LPRINT等の変換用
50EB	ジャンプテーブル(中間コード80H～EIH用)
51AF	ジャンプテーブル(中間コードFEH＋nnH用)
526F	ジャンプテーブル(中間コードFFH＋nnH用ただしnnHは98H以上)
531F	LET，LETなし代入文
5390	OUT
539E	INP
53AF	PAUSE
53C7	LOCATE
53FB	GET
5410	ON

アドレス	機　　能
547E	MON
5484	BOOT
548D	BEEP
5492	WAIT
54B9	KEY
54F8	KLIST
550F	KEY LIST
5515	ファンクションキーリストを出力
55EF	'key␣', 0
55F5	', chr$ (', 0
55FB	ERASE
560E	COMMON
578B	SWAP
5807	RANDOMIZE
5818	KMODE
5833	TRON
584B	TROFF
5852	STEP
5873	STOP(単なるSTOP以外)
587E	STOP行番号
5921	STOP, ON, OFFであるかどうかテスト
5928	ON, OFFかどうかテスト
5935	CLICK
5940	REPEAT(ON OFF)
5973	FIELD
59AE	MID$代入
59E6	SEARCH関数
5A39	CLEAR
5A8F	PEEK
5AAA	POKE
5B1C	HELP
5B22	MOUSE文
5B33	COM
5B5E	HELP・MOUSE等の割込みのON, OFF, STOP処理
5B83	ON KEY
5B89	ON STOP
5B8F	ON TIME$
5B9A	ON その他の割込み用語
5E7D	次に「,」が来るかどうかテスト
5EA8	PSET
5EA9	PRESET
5EFC	LINE
6003	CONNECT
60D3	POINT文
60E8	PAINT
6178	CIRCLE
62D5	CLS
6305	PATTERN
63AC	VIEW
6451	WINDOW
65B3	SYMBOL
666A	SCREEN
686C	POLY
6A23	GET@
6B22	PUT@
6BD0	GET%
6BD1	PUT%

アドレス	機　　能
6C3D	MAP
6CC9	POINT
6DBC	PEN
6E7A	～6EAI…ペン形状バッファ（データ）
6EA2	GLOAD
6EA6	GSAVE
729B	MOVE
7338	ROLL@
737F	ROLL
74B8	HCOPY
74D5	CBLOCK
750A	PRIORITY
7545	COLOR
75D7	COLOR=
75FA	COLOR@
7645	COLOR REPLACE
76CA	CFLASH
7749	CREV@
774D	CFLASH@
77B9	CREV
77D9	CGEN
7819	CONSOLE
7832	CONSOLE@
78F9	WIDTH LPRINT
7901	WIDTH
7995	CGPAT$( )=, DEF CHR$( )=
79C0	CGPAT$
7A69	文字列→中間コード変換
7E7D	中間コード→文字列変換
81A4	0～255のパラメータを得る
81ED	汎用の式評価ルーチン
81F9	EQV
8218	IMP
8239	XOR
8256	OR
8273	AND
828F	NOT
82AE	不等号処理
82C3	＜
82DB	＞
82E6	＝（比較）
82EF	＝＞，＞＝
82F8	＝＜，＜＝
82FF	＞＜，＜＞
8311	－
831F	＋
8332	¥
8348	MOD
8366	＊
8374	／
838A	－（負号）
8397	＾
855B	POS
856B	CSRLIN
8570	ERR
8582	ERL
8588	DTL

アドレス	機　　能
8673	DIM
888D	DEF FN
8900	FN
89B2	OPTION
89CD	OPTION BASE
89E4	OPTION LIST
89FE	OPTION ANGLE
8A0F	OPTION SCREEN
8A6E	OPTION SCREEN 4
8ABF	DEF
8B43	DEF USR
8B73	CHR$
8BB2	BIN$
8BB8	OCT$
8BBE	HEX$
8BD0	SPACE$
8BED	STR$
8BF7	SYSID$
8C03	ASCCHR$
8C1F	HEXCHR$
8C63	VAL
8C80	ASC
8CA0	LEN
8CAB	LEFT$
8CBF	RIGHT$
8CD6	MID$
8D22	SUM
8DFD	MKI$
8E02	MKS$
8E07	MKD$
8E10	CVD
8E13	CVS
8E16	CVI
8E29	STRING$
8E6D	MIRROR$
8E89	SCRN$
8F52	TIME$
8F57	DATE$
8F8C	TIME$（代入）
8FA2	DATE$（代入）
8FE1	DAY$
8FF7	DAY$（代入）
903E	TIME
9053	TIME（代入）
907F	FRE
9151	MOUSE
917E	NEW ON
9241	INSTR
92F7	JIS$
92FC	KTN$
931D	AKCNV$
9323	KACNV$
9339	KLEN
9360	KPOS
9439	SOUND
944E	TONE
9513	ただのTONE

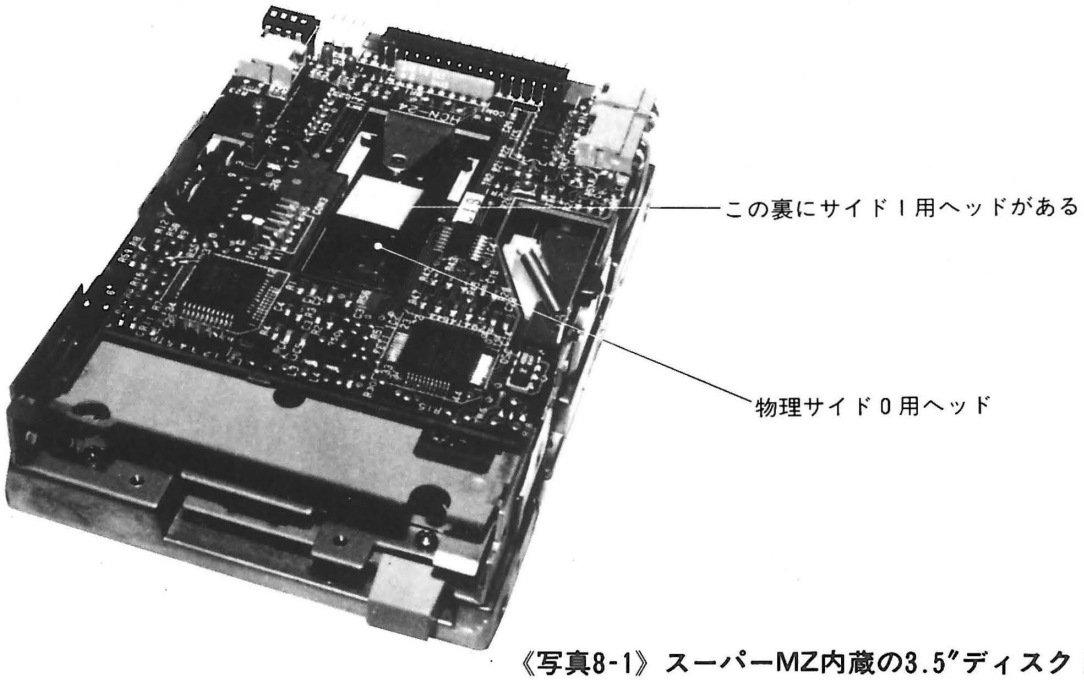
アドレス	機　　能
9600	TONE COPY
9732	PLAY
97B5	PLAY INIT
97C8	PLAY WAIT
9ABA	STICK
9ADA	STRIG
9B02	STICK文
9B0D	MOUSE文（ON, OFF, STOP以外）
9C2E	TERM
9C63	TEL
9CCA	APSS
9CF0	CMT文
9D4B	CMT
9D73	VOICE
9D79	VOICE@
9D97	VOICE（@以外）
9E0B	SYSID$で返される文字列（'6Z002␣V1.0A', 0等）
9E19	'␣bytes␣ free', 0
9E26	起動時先行入力バッファに入る長さとその文字列(13H, 'run "auto-run.m25', 0DH, 0)
9E3B	起動時のメッセージ（———M25 等）を表示
A225	～A284…起動時メッセージ用PCGフォントデータ



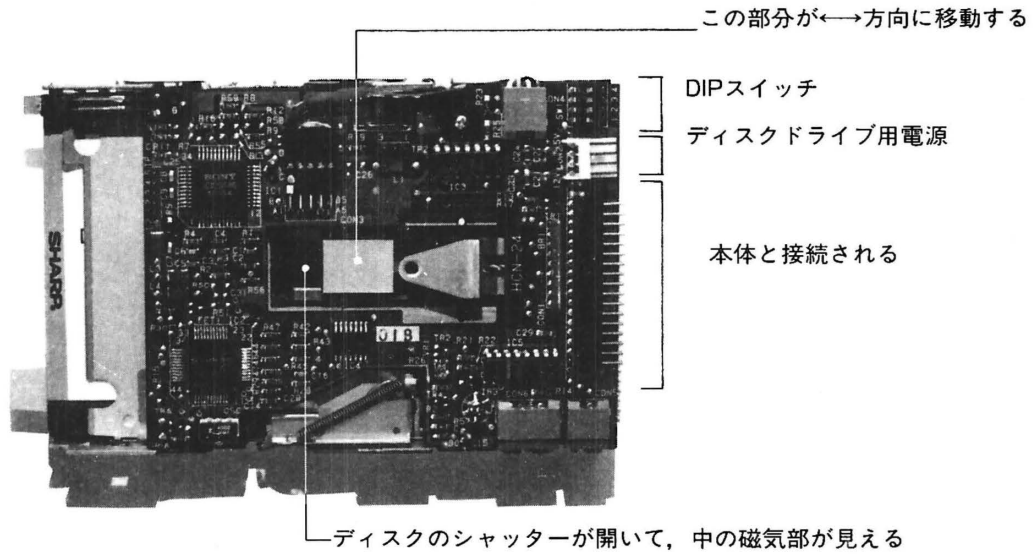
ディスクの内部形式

スーパーMZのディスクフォーマットは，MZ-80B～2200及び700～1500のものと上位互換性を持たせたものになっています。

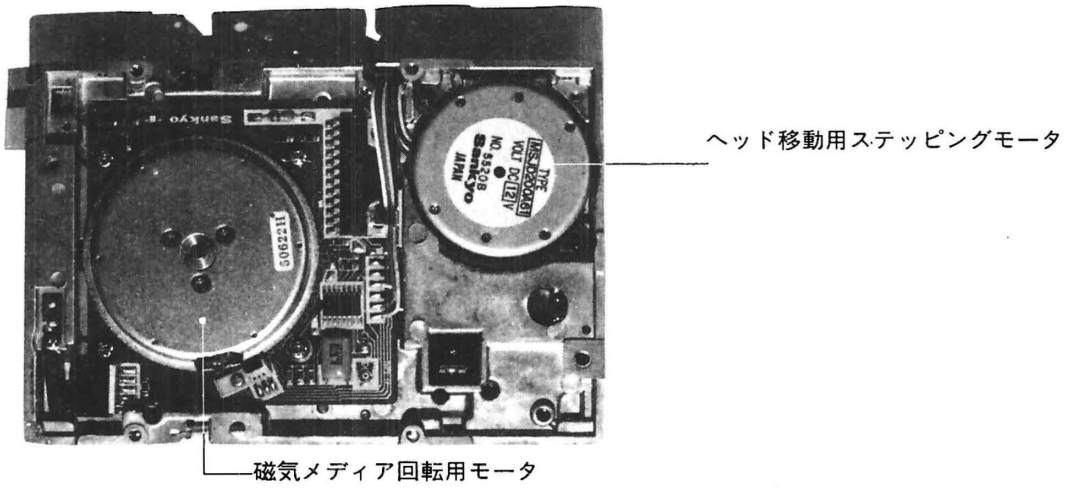
その前に，ディスクの物理的な仕様を見てみたいと思います。



《写真8-1》スーパーMZ内蔵の3.5″ディスクドライブ



《写真8-2》ディスクを入れドライブを真上から見た所



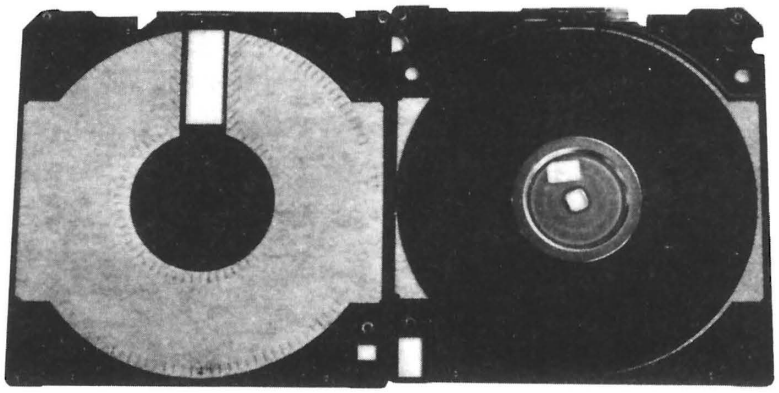
《写真8-3》ドライブを裏側から見た所



《写真8-4》3.5″マイクロフロッピーディスク



《写真8-5》ディスク裏面

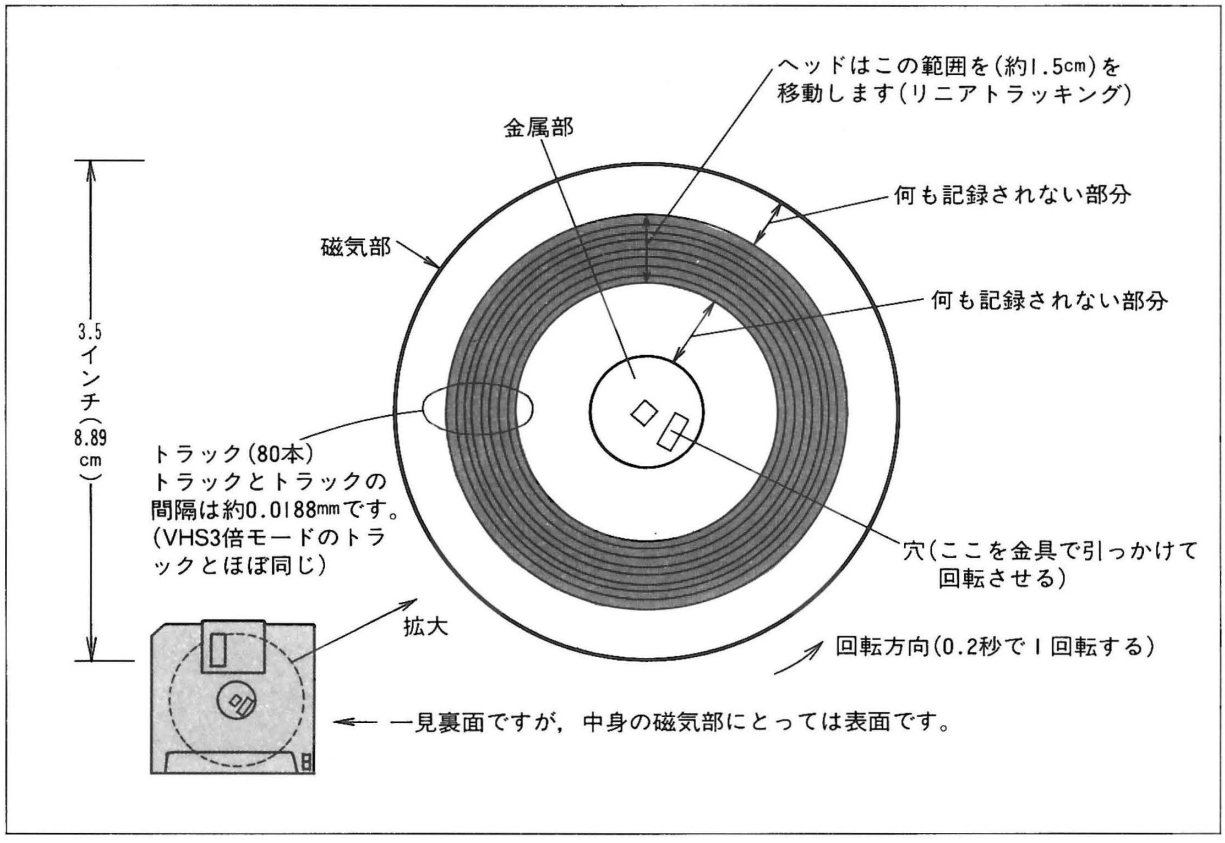


《写真8-6》ディスクの内部

3.5″ 2DDディスクの仕様

3.5″ 2DDディスクには，**第8-2図**のように，片面当たり80本の同心円状のトラックがあります。反対側も同様の構造ですので，合計でトラックは160本あることになります。ちなみに，2DDとは，2 sided(heads) Double density Double track つまり両面倍密度倍トラックと言う意味です。

図中にも書いてありますが，ディスクは0.2秒で１回転します。また，ディスクと本体とは250000ビット／秒で転送を行います。つまりトラック１周には50000ビット（6250バイト）のデータが記録できることになります。2DDではトラック160本ですので，ディスク１枚は1000000バイトの容量を持つことになります（アンフォーマット時１Mバイトと言う)。



《第8-2図》ディスクの内部構造

しかし，実際には，1トラック6250バイト分をそっくりそのままデータの記録に使うことはありません。そのうちの1バイトしか必要でなくとも，全部をそっくり読込まなくてはなりませんし，1バイトしか書込む必要がなくとも，一度全部のデータを読込んだ上で1バイトだけ書替え，再び全部を書込む必要があるからです。当然本体側には1トラック分のバッファメモリが必要ですし，読み書きの時間もかかります。また，トラックは円状になっているため，バラつきを考えると最初と最後の部分には余裕を持たせる必要があります。メモリも時間もそれで良いとしても結局，6250バイトフルに利用することはできないのです。

そこで，1トラックをいくつかのセクタに分割して記録する方法が考案され，採用されています。スーパーMZの場合，1トラックは16に分けて利用しています。

1セクタには，本来のデータの他に，そのセクタが第何トラックにあるのか，また，そのトラックの中で何セクタ目に当たるのか，データの部分は何バイトあるのか，また，それらの CRC (チェックサムの一種で読込んだ内容にエラーがあった時にそれを検出できる)，これらが始まることを示すマーク，データが始まることを示すマーク，データのCRC，消去ヘッドと読書き用ヘッドが離れている・回転数のバラつきを吸収するための余裕等が入ります。トラックのデータは，ヘッドを移動させた時に正しく目的の場所に着いたかどうかを確かめるためにも使われます。

「データ」の部分の長さは一般に128，256，512，1024バイトが可能ですが，スーパーMZの場合は256バイトで使用しています。また，他のほとんどの8ビットパソコンと，16ビット機の一部でも，256バイトを採用しています。

また，「データ」以外の部分の長さは，合計すると1セクタ当たり116バイト(程度)に上ります。かなり多いように感じられますが，「余裕」の部分が相当大きいのです。

すると，1セクタの長さは，372バイトになり，これを16倍すると5952バイトとなります。17セクタにしようとすると6324バイト必要になるため，1トラックは16セクタということになります。余った298バイト(6250－5952)は，1トラックの最後の部分の「余裕」となります。

なお，余談ですが，5インチ2HD・8インチでは$\frac{1}{8}$秒で回転，500000ビット／秒で転送，1トラックの容量は10416バイト，トラック数154となります。(256＋116)×26セクタまたは (1024＋116)×8セクタで利用します。

また5インチ・3.5インチの2DはスーパーMZの2DDに比べ，トラックが半分(片面40トラック)になっているだけです。また3.5インチ1DDは片面80トラックですが，表しか使いませんのでやはり合計トラック数は半分になります。ただ，(512＋116)×9セクタとして利用しているパソコンもあります。スーパーMZ内蔵のドライブでは，他のパソコンの3.5インチディスクの内容を2DD・2D・1DDにかかわらず読出すことができますから，マシン語を利用すればファイルコンバータ等も作成可能です。

ファイルフォーマット

ここまでの説明で，内蔵の2DDドライブは，

- 1基(1枚)当たり160トラックある
- 1トラック当たり16セクタある
- 1セクタ当たり正味256バイトのデータが入る

ということがわかりました。

全体ではセクタは2560個あり，正味655360バイト(640Kバイト)のデータが入ることになりますが，ここから先は，これだけを覚えていれば結構です。また，今後は，セクタを最初のトラック(トラック0)の最初のセクタ(セクタ1。セクタは1から数えます)を0，最後のトラック（トラック159)の最後のセクタ(セクタ16)を2559とした通しのレコード番号で数えたいと思います。

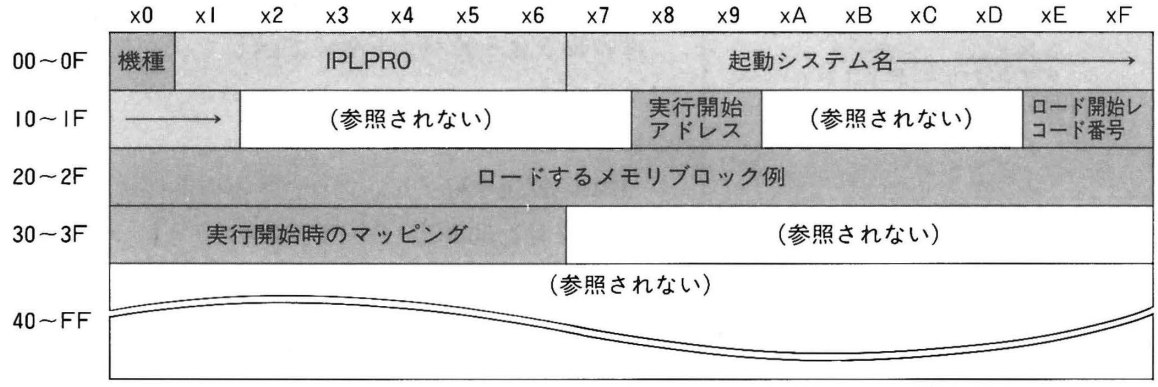
ところで，我々は，BASICのSAVEやLOADコマンドではこのディスクのどの部分を使用せよ，という指定は行いません。BASICの方で，自動的に空いているレコードを探し出してSAVEし，どこを使用しているかを自動的に判断してLOADしてくれます。また，FILES↵とするとどんなプログラムが入っているか等が出て来ます。つまり，2560レコード全部がプログラムやデータファイル等の保存用に使われているわけではなく，その他の情報も含まれているのです。

第8-3図を見て下さい。これは，BASIC-M25・S25，正確にはIPLから起動するOSが採用しているディスクの使用状況です。このOSはテレホンソフトでも使われていますから，やはり同様になります。しかし，シャープ以外からBASIC等が発表になることがあれば，これとは異なるものになる可能性もあります。またP-CP/Mや他のDOSも異なるものになります。ただ，その場合でも，IPLから起動するソフトでは，レコード0だけは同じになります。

では，これらを順に説明してゆきましょう。

レコード0000H(IPL用データ)

ここにはIPLからプログラムを起動させるための情報が入ります。



〈機種〉

01…MZ-80B～2200，2500

02…MZ-80A(MZ-1200の輸出バージョン)

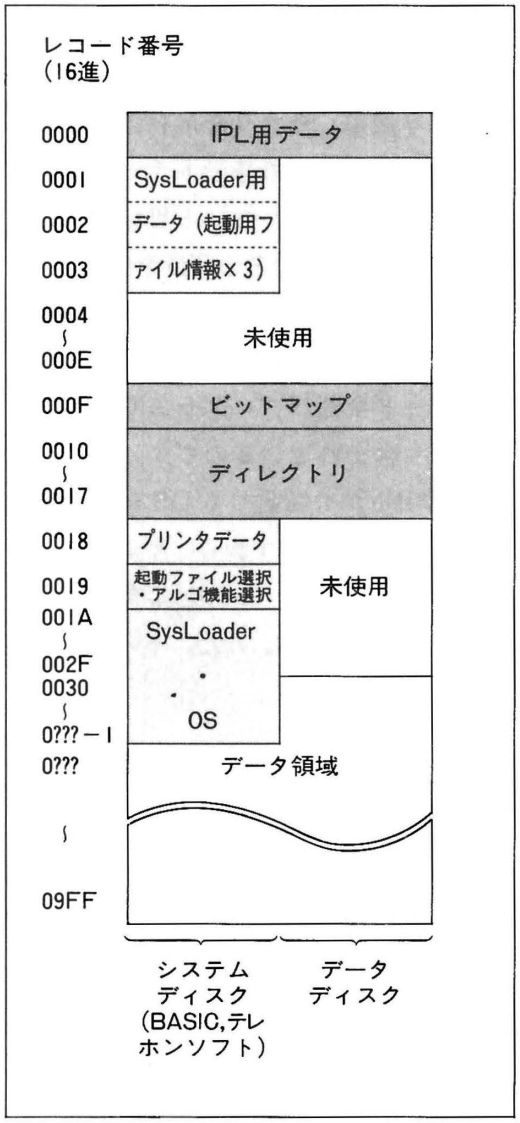
03…MZ-700・1500

ただしIPLはここを参照していません。

〈IPLPRO〉

ここは〝IPLPRO〟という文字列が入り，IPL起動プログラムであることを示しています。逆に，単なるデータディスクでは，00H等のデータが入り，IPLから起動させようとすると「not master」と表示され，エラーになります。

〈起動システム名〉



《第8-3図》ディスクの使用状況

IPL動作中に「IPL is loading〜」で表示させたい文字列を入れます。文字列の最後には少なくとも一つの0DHを入れる必要があります。BASICやテレソフトでは「Sys Loader」になっていますが，これは2段階ブートを行うためのソフトの名前です。

〈実行開始アドレス〉

これはロードしたプログラムを何番地から実行するかを示すものです。

〈ロード開始レコード番号〉

ディスクのどのレコードから起動させたいプログラムが入っているかを示しています。内容は後で触れているOBJ・BTX同様，連続して入ります。

〈ロードするメモリブロック列〉

「開始レコード番号」のレコードからプログラムをロードする際，どのメモリブロックにストアするかを指定するものです。メモリブロックは00H〜3FHのうち，IPLで使用しないRAM領域である02H〜0EH・10H〜1FH(メモリ拡張時)が指定できます。また，FFHが来ると，そこでロードを終了します。つまり，このメモリブロック列はロードするプログラムの長さを8Kバイト単位で示していることにもなるのです。

〈実行開始時のマッピング〉

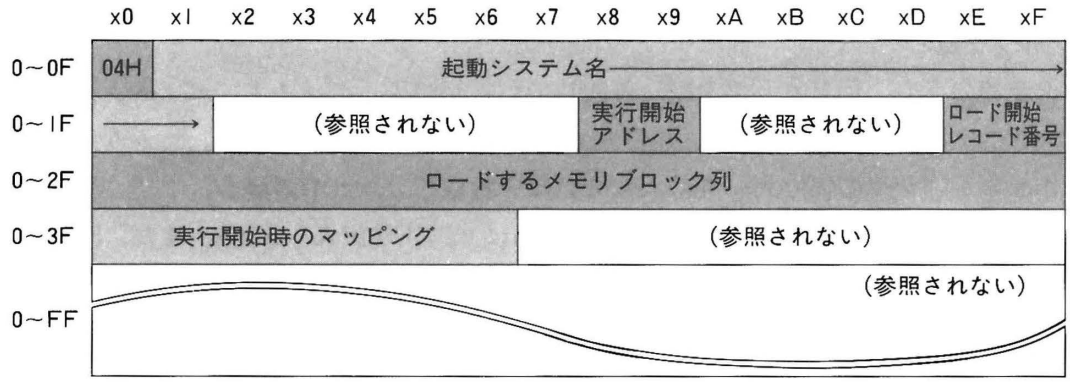
IPLの動作を終了し，プログラムの実行に移る際，CPUから見た0000H〜DFFFHをどのようにマッピングするかを指定するものです。なお，E000H〜FFFFHは常にIPLのワークである0FHが割り当てられますので，必要ならプログラムを起動した後で書き替えれることになります。また，このため，18H・19Hの実行開始アドレスは，事実上0000H〜DFFFHの範囲に限定されることになります。

レコード0001H〜0003H(Sys Loader用データ)

ここには，Sys Loaderを使用して2段階ブートを行う時に使用し，起動するファイルの情報を入れておきます。BASICのマスターディスクでは0001HにM25，0002HにS25の情報が入っており，0003Hは未使用になっています。

〈起動システム名〉

これは，Sys Loaderが起動した後，2段階目のローディングを行う際，「BASIC-M25 (6Z002)をロード中です」のように表示を行いますが，その時のプログラム名を指定するものです。文字列の最後には0DHが必要ですので，名前は16文字以内となります。



〈実行開始アドレス〉

〈ロード開始レコード番号〉

〈ロードするメモリブロック列〉

〈実行開始時のマッピング〉

これらはいずれもレコード0000Hと同じです。

ただし，SysLoaderと，それと一緒にロードされるOSが02H～07Hと0EHに入っており，またワークとして00H～01Hを使用しているため，これらのメモリにロードを行うことはできません。また，マッピングは0000H～DFFFHが指定可能ですが，E000H～FFFFHは0EHが使用されています。

000FH(ビットマップ)

ここには，「データ領域」の使用状況が入ります。SAVEを行ったり，データファイルを作ったりする際，ディスクのどの部分が空いているかをこれで判断しています。また，その後で，このレコードの書換えを行っています。つまり，使用したレコードを「使用済」として登録するわけです。また，KILLでは，その逆に「使用済」を「未使用」に戻しています。なお，LOAD等ではこのビットマップを参照することはありません。

なお，このビットマップでは，「レコード番号」や「レコード数」は用いず，「ブロック番号」や「ブロック数」を使っています。2Dでは1ブロック＝1レコードですが，2DDでは1ブロック＝2レコードになります。つまり，2DDは容量が大きいので，ここだけは2レコード分をまとめて管理しているのです。また，2DDの「倍率」は「2」ですが，将来的には4，8，16…とすることで，16Mバイトまでのハードディスクにも対応できるようになっています。

〈ボリウム番号〉

マスター(システムディスク)では00H，データ専用ディスクでは01H～FFHでボリウム番号を表わすのですが，MZ-2500モードでは使用されず，MZ-2000等のなごりとして残っているだけです。

〈オフセット〉

データ領域がどこから始まるかを示します。

スーパーMZに付属のBASICの入っているディスクでは78Hとなっており，データ領域はレコード00F0Hから始まっていることがわかります。つまり**第8-3図**の「0？？？」は00F0Hになるわけで

す。また，テレホンソフトは68Hで，レコード00D0H以降がデータ領域です。

〈使用済ブロック数〉

• ディスク全体のうち，何ブロックを既に使用しているかが入っています。ディスク全体のうちですから，システム領域(データ領域以外の部分)の分も，無条件で入ります。つまり，データディスクを作成したばかりで何もデータの入っていない状態でも，2DDの場合，ここには0030H÷2で「0018H」が入るのです(さきほどの「オフセット」にも，18Hが入ります)。また，BASICのディスクをバックアップし，全部のファイルを消してデータ領域を空にした場合，使用済ブロック数は00F0Hになります。

〈全体のブロック数〉

2DDでは0500Hが入ります。また，2Dでも原理上は0500Hで良いのですが，MZ-80Bや2000の頃のディスクドライブ「MZ-80BF」の一部では片面当たり35トラックしか使用できないため，その他のものや「MZ-1F07」は40トラックが可能ではあるものの，ディスク全体を70トラック・1120レコードとして使っています。そのため，全体のブロック数は04A0H(＝1120)になっています。

なお，この「全体のブロック数」から「使用済ブロック数」を引き，2DDでは2倍，2Dでは1倍(そのまま)すると，未使用レコード数が得られます。そして，それを256倍すると未使用領域が「バイト」で計算できます。

〈各ブロックの使用情況〉

ここは，データ領域の最初から最後までの各ブロックについて，使用中であるか否かが入る所です。1ブロックが1ビットに対応しており，1バイト中ではビットの低い方が小さい番号のブロックに対応しています。そのビットが1なら使用中(使用済)，0なら未使用を意味しています。

なお，データ領域のブロック数は，システムディスク／データディスク，2DD／2Dによって多少違って来ます。そのため，この「使用状況」もそれに伴なって多少変わります。例えば2DDのデータディスクなら「データ領域」は0500H－0018H＝04E8H(1256)ブロックですので，「使用状況」は06H～A2Hの9DHバイト(157バイト＝1256ビット)になります。しかし，2DDのBASICのシステムディスクでは0500H－0078H＝0488Hブロック→0488Hビット→91Hバイト→06H～96Hを使用となります。なお，**第8-4図**は2DDデータディスクの例で，「使用状況」と未使用の部分を分けています。

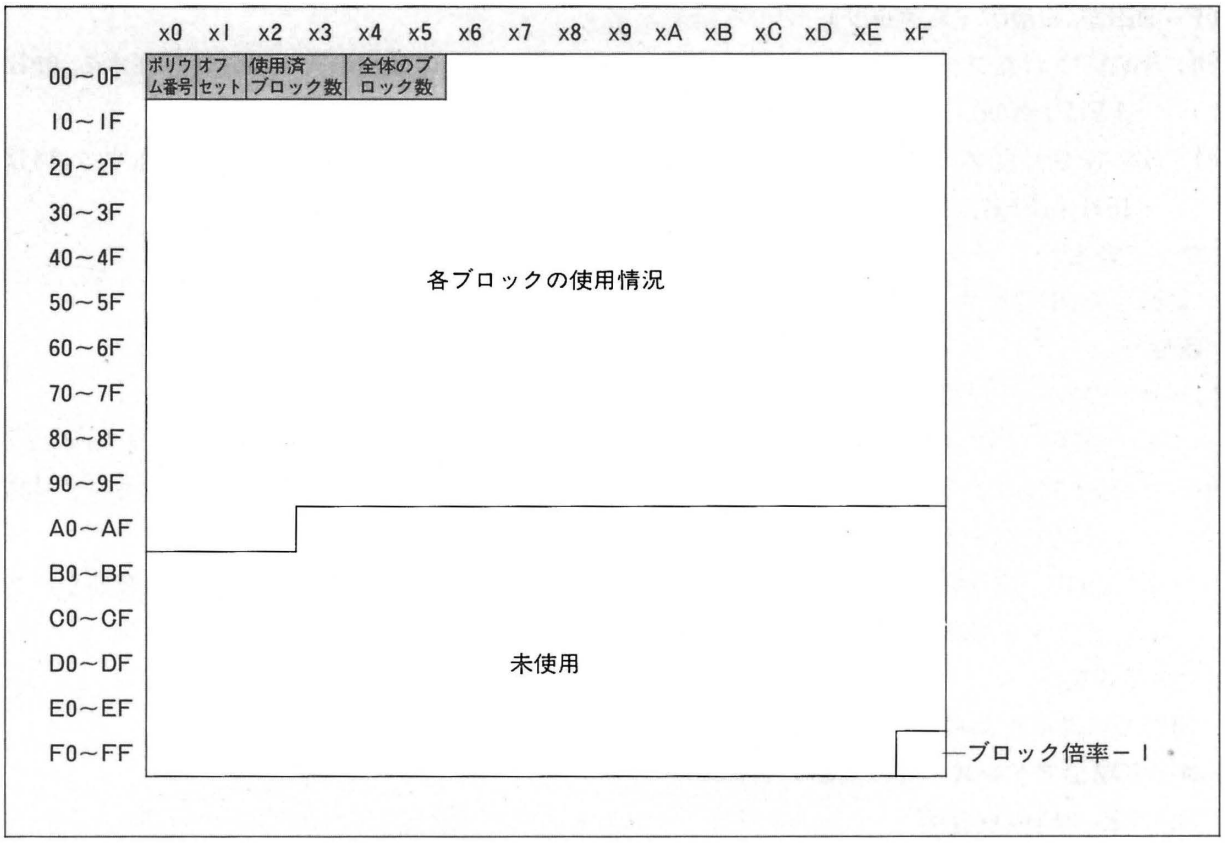
〈ブロック倍率－1〉

これは，何レコードを1ブロックとするかを示すものです。ただし，－1された値が入りますから，2DDでは1，2Dでは0になります。また，将来的に3.5″ないし5″の2HDタイプを使用できるようになった時でも，このバイトは1で済みます。154トラック×26セクタ÷2倍率＝2002ブロックですが，2002ブロック－18H(オフセット)＝1978ブロック，1978ビット÷8＝247.25バイト＝F7Hバイト＋2ビットとなり，データディスクでも「使用状況」は06H～FDHしか使わないからです。

0010H～0017H(ディレクトリ)

このエリアは8レコード(2Kバイト)分あり，それを32バイトずつ64個に区切って使います。

そしてその32バイトに，ファイル名やファイルのモード(OBJ，BTX等)，ファイルの大きさ等

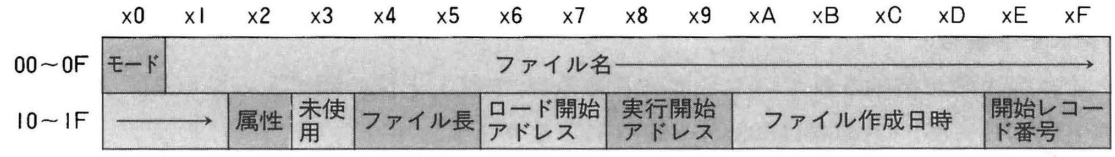


《第8-4図》ビットマップの状態

が入ります。

ただし，64個のうちの最初の１個はSWAP文(ファイル単位のGOSUB)専用となっており，一般ファイルは63個までとなっています。また，階層ディレクトリは，親ディレクトリに名前と長さ(子ディレクトリの長さ＝２Kバイト)等が登録され，子ディレクトリには自分自身(″.″)と親ディレクトリ(″..″)のデータが入ります。従って，一般ファイルは61個となります。

32バイトの詳しい状況は次の通りです。



〈モード〉

00…未使用領域であることを示す(01H～1FHが無効になる)

01…OBJ型(マシン語プログラム・データ…BSAVE・BLOAD・モニタのL・Sで使用)

02…BTX型(BASICのプログラム…SAVE・LOAD等で使用)

03…BSD型(シーケンシャルファイル…OPEN ″I″, ″O″, ″A″, SAVE～, Aで使用)

04…BRD型(ランダムファイル…OPEN ″R″・GSAVE・GLOADで使用)

05～0E…未使用(05＝RB, 07＝LIB, 0A＝SYS, 0B＝GRが一応登録されている。F-DOSのなごり)

0F…DIR型(階層ディレクトリ)

80…SWAPされたファイルがないことを示す(各ディレクトリの最初の一つのみに存在する。01H～1FHは無効)

81…SWAPされたファイルがあることを示す(各ディレクトリの最初の一つのみに存在する。14H・15H，1EH・1FHのみ有効)

〈ファイル名〉

最後に0DHを必ずつけるので，実質的には16文字以内。

〈属性〉

b_0＝0…SET～,""の状態(ノーマル)

b_0＝1…SET～,"P"の状態(書き込み・消去禁止ファイルで，FILES時モードの後に"＊"が付く)

b_7＝0…BSD型のように，1レコード毎に次のレコード番号が入ったファイルである(モードが05H以上の時のみ有効で，ファイルを削除する時に参照される)

b_7＝1…OBJ・BSD型のように，連続したファイルである(モードが05H以上の時のみ有効で，ファイルを削除する時に参照される)

〈ファイル長〉

BRDでは長さを32(20H)で割ったもの，それ以外ではそのまま入る。

〈ロード開始アドレス・実行開始アドレス〉

共にBINの時のみ有効

〈ファイル作成日時〉

1AH		1BH				1CH			1DH		
b_7 b_4	b_3 b_0	b_7	b_6 b_3	b_2 b_1	b_0	b_7 b_5	b_4 b_3	b_2 b_0	b_7	b_6 b_4	b_3 b_0
10年の位(0～9)	1年の位(0～9)	10月の位(0～1)	1月の位(0～9)	10日の位(0～3)		1日の位(0～9)	10時の位(0～2)	1時の位(0～9)	10分の位(0～5)		1分の位(0～9)

のようになっています。各桁毎に区切られているので，BCD形式(RP-5C15…カレンダクロックで使われている)との相互変換がシフトだけでできるようになっています。

〈開始レコード番号〉

ファイルの先頭が格納されているレコード番号が，下位・上位の順で入っています。2DDの場合は原理上0000H～09FFH実際は0030H～09FFHになります。

OBJ・BTXとモードが05H以上で「属性」のb7が1のファイル(DIRもこれになります)では，この番号からファイル長の上位バイト(下位バイトが0以外では切り上げを行い，上位バイト＋1になる)レコード分だけが，ファイルの内容です。レコード番号が00F0H，ファイル長9000HのOBJファイルなら，レコード番号00F0H～017FHが，そっくりそのままファイルの内容になります。BSDとモードが05H以上で「属性」のb7が0のファイルでは，この番号のレコードが最初の254(FEH)バイト分のデータとなります。1レコードは256バイトですので，2バイト余りますが，その最後の2バイトが0000Hならファイルはそこで終わりです。しかし，それが0000H以外なら，それは次の254バイト分のデータが入っているレコードの番号になります。そして，そのレコードに

入っている最後の2バイトがさらに次のレコード番号を指しています(または0000Hで終了を示している)。これをずっとファイルが終わるまで続けるようになっているのです。要するに,「最後の2バイト」でレコードがチェインしているというわけです。

最後に，BRDモードのファイルでは，OBJ・BTXの特徴とBSDの特徴を兼ね備えたものになっています。つまり，OBJ・BTXでは一挙にLOAD・SAVEできるため高速である反面SAVE時に連続した空きエリアが必要となるため，256バイト(1レコード)の空きエリアが仮に10個あったとしても，それが全く連続しておらずバラバラに存在しているなら257バイトしかないプログラムでもSAVEできないことになります。BSDでは1レコード単位でチェインするため無駄なく最後まで使用できる反面，読出し・書込み・消去に時間がかかります。BRDではこれらをうまく組み合わせ，容量・スピード共，ある程度良好な効率が得られるようにしています。

まず，開始レコード番号で示されたレコードには，128個のレコード番号が入っています。そして，1個目のレコード番号から連続した4Kバイト分(16レコード分)，2個目のレコード番号からの4Kバイト分，3個目のレコード番号からの4Kバイト分，…128個目のレコード番号からの4Kバイト分が，実際の内容となります。ただし，レコード番号が0000Hの時は，そこでデータは終わりになります。また，BRDファイルは，最大512Kまで利用可能です(他のモードは14H・15Hのファイル長の部分の都合で64K未満です)。

0018H～0???－1(00EFH)　(SysLoader等)

BASICのシステムディスクやテレホンソフトでは2段階ブートでBASIC・テレホンソフトが起動しますが，ここには2段目を読み出すためのSysLoaderの他に，グラフィック・プリンタ・カナ漢字変換・文節変換・ファイル管理・マウス・音声合成等を行うためのプログラムが入っています。これについては別に触れている通りです。また，その中のプリンタルーチンで使用する，プリンタ選択ユーティリティで設定したデータが，レコード0018Hに入っています。

また，レコード0019Hには，レコード0001H～0003Hで指定しているプログラムのうちのどれを起動するかの情報と，アルゴ機能の情報が入っています。BASICのディスクではアルゴマネージャ，電卓，カラーシミュレーション，オートダイヤラーが登録されており，通常はオートダイヤラー以外の二つについてロードするフラグがセットされています。

SysLoaderはまた，HELPキーによる起動ソフトの切換え，アルゴキーによるアルゴ機能の切換え等もサポートしています。「BASIC-M25(6Z002)をロード中です」等と表示されている最中，青の「0」マークが順に赤に変わってゆきますが，これもSysLoaderの機能の一つです。ちなみにこれは，BASIC(テレソフト)は8Kバイトにつき1個，アルゴ機能はファイル一つにつき1個ずつ変化してゆきます。例えばアルゴ機能を全部「使用せず」にしてしまうと，「0」マークは5個になります。

BASIC-M25の隠し命令 ❹

●PRINT文等での〝～〟中にコントロールシンボルキャラクタが使える

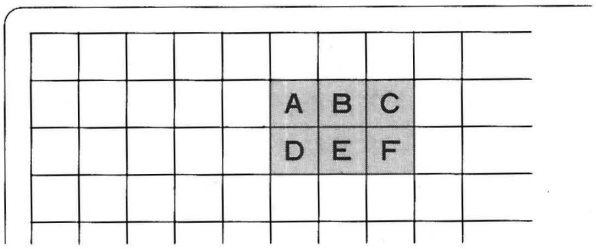
これはフォーマット&コピーのプログラム等でシャープのプログラマも使用しているので，既にお気づきの方もおられるかと思いますが，要するにCHR$を使わなくとも〝～〟中に直接コントロールコードを置けるということです。コントロールコードはCHR$(13)以外(↵以外)なら何でも良く，CTRL+EやCTRL+L，⇨⇦⇧⇩等も自由に使えます。

例えば，

LOCATE 5，1：PRINT〝ABC⇩⇦⇦⇦DEF〟

とすると，

右のような表示にできます。つまり，LOCATEの相対指定ができるわけです。前項のPRINT [ ]同様，LOCATEとPRINTが交互に出て来てわかりづらくなることがかなり防げます。また，この他のコントロールコードでも，〝～〟とCHR$が長々と連なってしまう点が改善されるでしょう。



なお，⇨等の文字は，GRAPHを押してカーソルが◆になっている状態で，カーソルキーやCTRL+アルファベットキーを押すと入力することができます。ただ，これらによって入力した文字は，プリンタでリストをとる時に印字できない場合がありますので注意して下さい(漢字プリンタ使用時もビットイメージ印字を行うようにすると，印字可能になります)。

●KILL

OPEN〝O〟～で出力シーケンシャルデータファイルを使用する時，場合によってはそのファイルを正式登録せずにキャンセルしたいことがありますが，CLOSE #1等とする代りにKILL #1のようにすると，OPENで指定したファイルは作らず，そのファイル番号のバッファを解放することができます。また，ただのCLOSEで全ファイルCLOSEとなるのと同じように，ただのKILLで全ファイルキャンセルとなります。マイクロソフト系のBASICでは一般に，CLOSEで一度正式登録した後で，ファイル名を指定したKILLによりそのファイルを消すといった方法がとられていますので操作が簡単になるでしょう。

●IPLの隠し機能

ここではBASICの機能ではなく，マニュアルにないIPLの機能を紹介しておきます。なお，これらは1～4，／，E等同様，それらのキーを押しながらIPLを起動させることで利用可能です。また，原則としてこれらは組み合わせて使うことができます。

B…メモリのテストを行います。対象となるのは標準装備の128KシステムRAM（最初の16Kバイト分を除く），128K増設システムRAM，標準装備の64KグラフィックRAM，64K増設グラフィックRAMで，02_H～$2F_H$のブロック番号と合否(GでOK，NでRAM未実装か異常あり)とが表示されます。本体出荷時にメーカーがチェック用に使用しているものと思われます。なお，チェック後はCPUが停止状態となりますので，結果を確認したらIPLスイッチを再び押すなどして下さい。

C…カセットデッキにあるソフトを利用したい時は，マニュアルによるとドライブ1に何も入れずIPLを起動させ，エラーが発生した後でCキーを押すことになっていますが，Cキーを押しながらIPLを起動させると，一発でカセットデッキからロードさせることができます。

G…IPLでは，通常グラフィックRAMの内容を消していますが，このキーを押しながらIPLを立ち上げた場合は，ソフト的にはこの動作を特に行いません。ただ，IPL信号がCPUのリセット入力となり，その間使用しているRAMへのリフレッシュ動作が停止するため，厳密にはグラフィックRAMの内容が1ビットたりとも欠けずに残る，という保障はされていません(このためマニュアルにこの機能の説明がない)。ですが実際にはそっくりそのまま残るのが普通ですから活用すると良いでしょう。

X…MZ-80Bモードにします。つまり，カンガルーポケット内のスイッチがMZ-2500の位置にあっても，Xを押しながらIPLを起動すると，MZ-80Bのソフトが利用可能になります。また，この時，クロックは2500モード同様6MHzになるので，すべてのソフトがスピードアップします。ただ，この結果カセットデッキのスピードまで上がってしまうため，この機能はディスクしか利用しない場合の専用機能と考えて良いでしょう。

Z…MZ-2000モードにする点を除き，Xと同様の働きをします。



第9章 ゲーム&グラフィックプログラム集

「6MHzのMZが出る」って聞いた時，真っ先に思ったのが「ああ，時代も進歩したな」ってこと。だって僕がコンピュータを始めた頃，MZはまだ出たばかり。セミキットのMZ-80Kは2MHzだった。

グラフィックも今とはくらべものにならないくらい貧弱で，80×50ドットのモノクロセミグラフィック。RAMも標準で20Kしかなかった。

それが，たったの6年でCPUは3倍速，グラフィックはアナログ式の640×400ドット。RAMなんか標準でも128Kだもんね。当時は夢にも思わなかった。

そしてもう一つ夢にも思わなかった事。それはそのスーパーコンピュータのプログラムを僕が作ることになるなんて事。依頼を二つ返事で引き受けたのはいいけれど，知れば知るほど奥が見えてくる機能のスゴさ。山の様なコマンド群。うーん，本当に僕なんかに使いこなせるのだろうか？

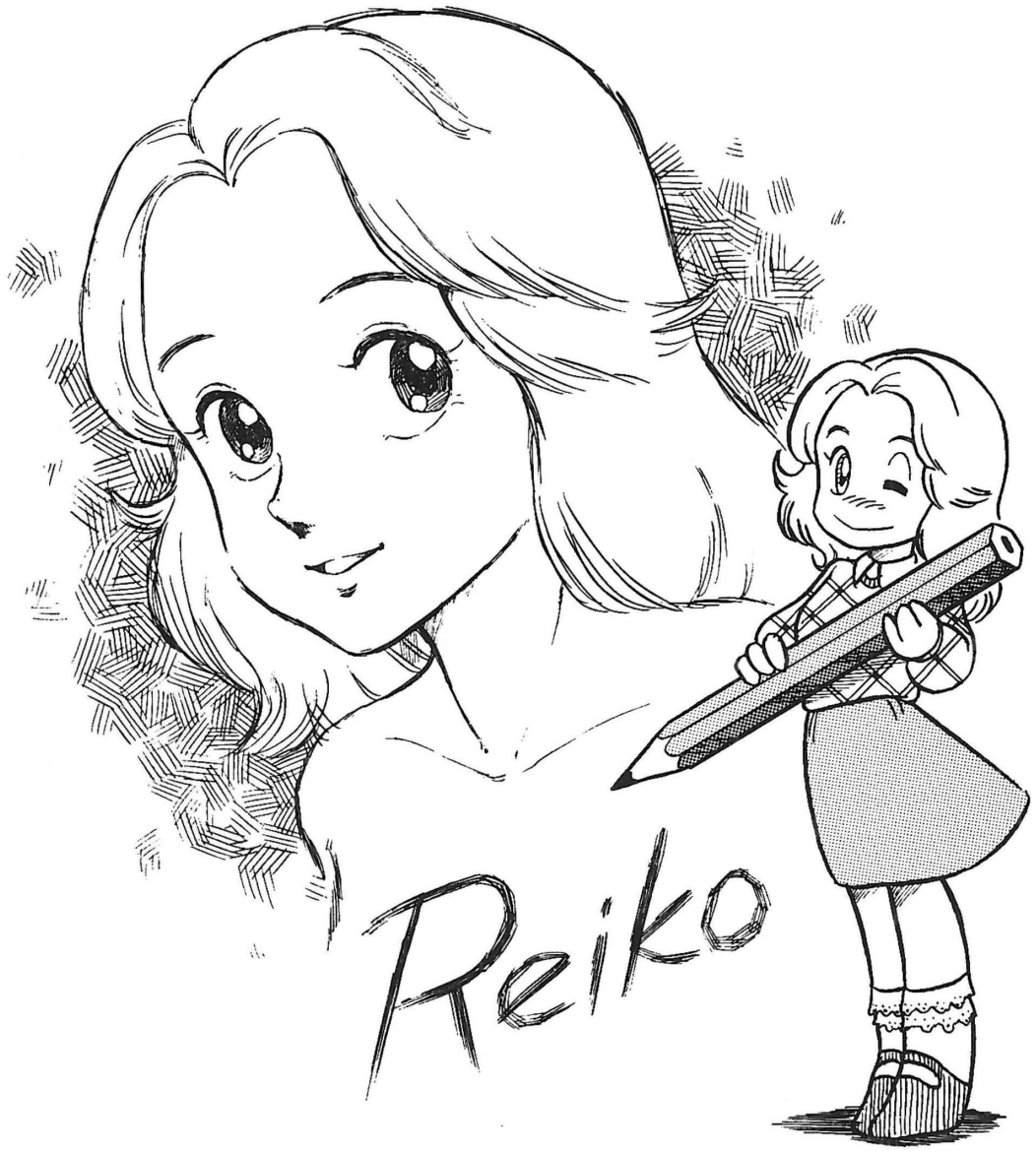
とにかく全力投球でいくしかなさそうだ！

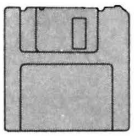
多部田　俊雄

9-1 グラフィックエディタ「RERU」

ゲームなどに使うキャラクタを作る時は普通，キャラクタエディタを使って作ります。スーパーMZにもアプリケーションにPCGエディタがあり，それを使ってテキスト画面に出力するキャラクタを再定義できますが，グラフィック画面に出力する「グラフィックキャラクタ」を作るエディタはありません。

そこでマイコン本誌でおなじみ女の子の名前シリーズ！グラフィック・エディタ「RERU」を紹介しましょう。





★RERUとは

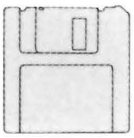
グラフィックエディタ「ＲＥＲＵ」とは，次の機能を持ったゲーム作成用ツールです。

●簡単にグラフィックキャラクタを作れる

●作ったキャラクタをＢＡＳＩＣのＤＡＴＡに直せる

●座標指定でそのキャラクタを高速で動かせる

方眼紙に手書きでキャラクタを書いたりＰＵＴ命令で画面表示をしたりしていると手間も処理速度もかかってしまいます。これを解消してくれるのが，このゲームエディタシリーズなのです。ちなみに，「ＲＥＲＵ」のネーミングは後輩の□□礼子のニックネームからとっています（今までに発表したシリーズ名は「CHIEmkⅡ」「FUMI」「MAYU」「YUMImkⅡ」などなど）。

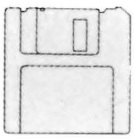


グラフィックモード

スーパーＭＺのグラフィックモードは標準装備でも**第１－１表**のように多彩です。しかしゲームなどに使うとなると①は色不足ですし，③は横のドットが粗くなります。④の256色モードも魅力的ですが，エディタでサポートしきれないため，結局「ＲＥＲＵ」では②の**「640×200ドット16色」**をサポートすることにしました。

《第1－1表》スーパーＭＺのグラフィックモード

①	640×400 ドット 16色中４色選択
②	640×200 ドット 16色
③	320×200 ドット 16色（２画面）
④	320×200 ドット 256色



使用方法

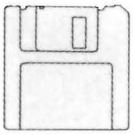
ＲＥＲＵを使ってゲームを作るには，だいたい次の様な順序で行います。

①**エディタ（リスト1－1）**でキャラクタを作成。そのデータをいったんメモリ上に置いておく。

②**データメイカ（リスト1－2）**でデータをＤＡＴＡ文にする（データメイカの中には**キャラクタライタ**も含まれています）。

③ＤＡＴＡ文以前の行番号中に，あなたがゲームをプログラミングし，ＰＯＫＥ文で座標とキャラクタナンバーを指定。ＵＳＲ文でキャラクタをコールすることによってキャラクタを書込む。

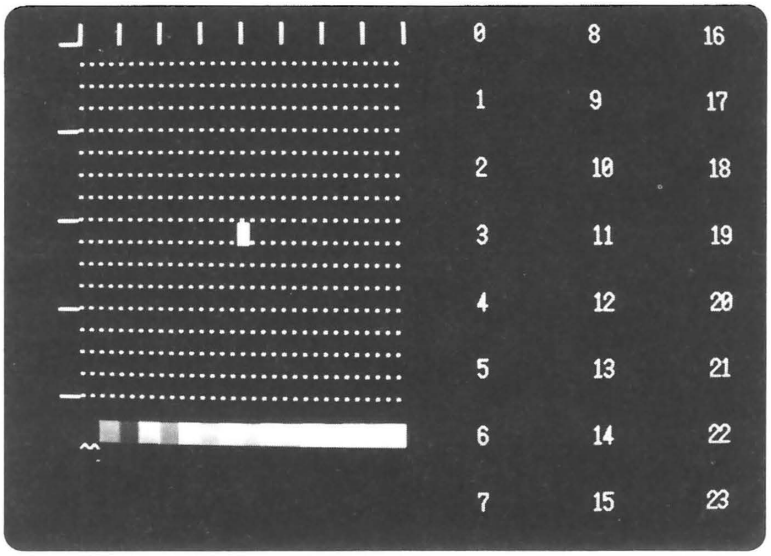
以上の様になります。とにかく百聞は一見にしかずと言います。まずはインプットして下さい。



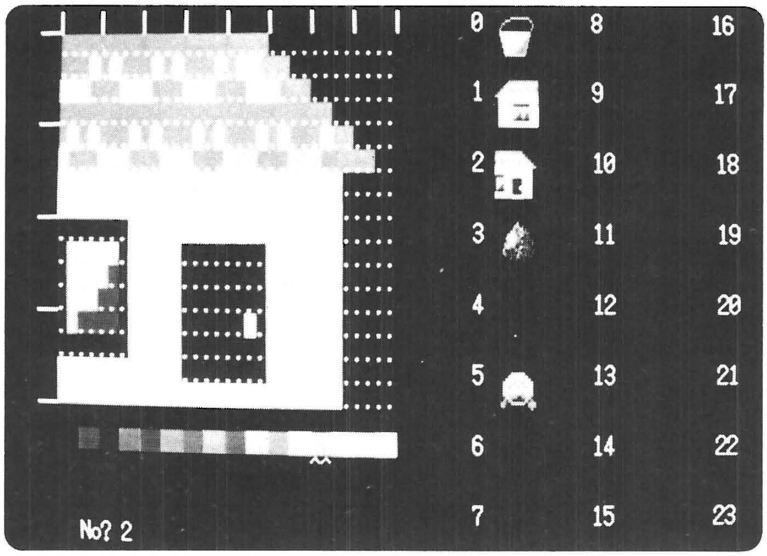
エディタ

NEW ON 0↵を実行後，ＲＵＮすると**写真1－1**の様になります。左上には32×16のドット，左下には指定色のカラーバー，その下にカラーカーソル。画面半分にはキャラクタナンバーが表示されています。

すべての表示が終わるとカーソル（■）がドットのマス中央に現れ，**第1－2表**のコマンドの入力が可能になります。



《写真1-1》RUN直後のエディットモード



《写真1－2》セットモードで
No.2のデータを表示

《第1－2表》 RERUの使用法

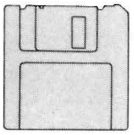
モード	操作キー	内　　容
エディットモード	8	カーソル上へ。
	2	カーソル下へ。
	4	カーソル左へ。
	6	カーソル右へ。
	スペース	カラーカーソルの指す色でカーソルの指す所に色をセットします。
	W	32×16マス中のキャラクタをデータに直し，メモリに置くのと同時に原寸で画面に表示します。 キャラクタナンバーをたずねてくるので0～23の数を入れて下さい。
	C	カラーカーソルの指す色でマス中をクリアします。
	S	セットモードにします。
セットモード	4	カラーカーソル左へ。
	6	カラーカーソル右へ。
	X	マス中のキャラクタを左右入れかえます。
	Y	マス中のキャラクタを上下入れかえます。
	G	メモリ上のデータをキャラクタに直し，マス中に表示します。 キャラクタナンバーを入れて下さい。
	R	エディットモードにします。

(注)　スタート時はエディットモードになっています。

《リスト１－１》RERUエディタ・プログラム

```
   10 clear &HA000:def int A-Z:dim B(31,15),R(31,15),G(31,15),I(31,15)
   20 init "CRT2:640,200,16":init "CRT1:80,25,1,1":kmode 0:cls 3:locate 0,0:print "  |   |   |   |   |
|   |   |   |"
   30 for D=0 to 19 step 4:locate 0,D:print "__":next D
   40 for D=1 to 16:locate 2,D:print ".............................":next D
   50 for D=0 to 2:for DD=0 to 7:locate 40+D*11,DD*3:print D*8+DD:next DD:next D
   60 for D=0 to 15:line (16+16*D,144)-(31+16*D,151),D,BF:next D:gosub 820
   70 locate 2,19:print "^^":X=16:Y=8:CO=0:ZZ=0:gosub 490
   80 G$=inkey$:if G$="8" then locate 2+X,Y+1:print ".":Y=Y-1:if Y<0 then Y=15
   90 if G$="2" then locate 2+X,Y+1:print ".":Y=Y+1:if Y>15 then Y=0
  100 if G$="4" then locate 2+X,Y+1:print ".":X=X-1:if X<0 then X=31
  110 if G$="6" then locate 2+X,Y+1:print ".":X=X+1:if X>31 then X=0
  120 if G$="5" then locate 2+X,Y+1:print ".":X=16:Y=8
  130 if G$=" " then B(X,Y)=BC:R(X,Y)=RC:G(X,Y)=GC:I(X,Y)=IC:line (16+X*8,8+Y*8)-(23+X*8,15+Y*8),CO,BF:lo
cate 2+X,Y+1:print ".":X=X+1:if X>31 then X=0
  140 locate 2+X,Y+1:print "■":if G$="c" then ZZ=0:gosub 490
  150 if G$<>"w" then 300
  160 locate 4,22:input "No";LL:if LL>23 then 160
  170 PP=&HA800+LL*256:PX=int(LL/8)*88+344
  180 PY=int(LL-int(LL/8)*8)*24:for D=0 to 15:for DD=0 to 3
  190 DB=0:DR=0:DG=0:DI=0:for EE=0 to 7:Q1=EE+8*DD
  200 pset (PX+Q1,PY+D),B(Q1,D)+R(Q1,D)*2+G(Q1,D)*4+I(Q1,D)*8:J9=int(2^EE)
  210 DB=DB+B(Q1,D)*J9
  220 DR=DR+R(Q1,D)*J9
  230 DG=DG+G(Q1,D)*J9
  240 DI=DI+I(Q1,D)*J9
  250 next EE:poke PP+DD+D*4,DB
  260 poke PP+DD+D*4+64,DR
  270 poke PP+DD+D*4+128,DG
  280 poke PP+DD+D*4+192,DI
  290 next DD:next D:play "C32"
  300 if G$<>"s" then 80
  310 G$=inkey$:if G$="4" then locate 2+CO*2,19:print "  ":CO=CO-1:if CO<0 then CO=15
  320 if G$="6" then locate 2+CO*2,19:print "  ":CO=CO+1:if CO>15 then CO=0
  330 locate 2+CO*2,19:print "^^"
  340 if G$="r" then ZZ=1:gosub 490:goto 80
  350 if G$<>"g" then 620
  360 locate 4,22:input "No";OO:if OO>23 then 360
  370 poke &HA3FF,OO:def usr=&HA000:def usr1=&HA00D:A=usr(0)
  380 for YY=0 to 15:for XX=0 to 31:B(XX,YY)=peek(&HA400+XX+YY*32):next XX:next YY
  390 A=usr1(0)
  400 for YY=0 to 15:for XX=0 to 31:R(XX,YY)=peek(&HA400+XX+YY*32):next XX:next YY
  410 A=usr1(0)
  420 for YY=0 to 15:for XX=0 to 31:G(XX,YY)=peek(&HA400+XX+YY*32):next XX:next YY
  430 A=usr1(0)
  440 for YY=0 to 15:for XX=0 to 31:I(XX,YY)=peek(&HA400+XX+YY*32):next XX:next YY
  450 for YY=0 to 15:for XX=0 to 31
  460 line (16+XX*8,8+YY*8)-(23+XX*8,15+YY*8),B(XX,YY)+R(XX,YY)*2+G(XX,YY)*4+I(XX,YY)*8,BF
  470 next XX:next YY:play "C32"
  480 goto 310
  490 PP=CO:IC=0:if PP>7 then PP=PP-8:IC=1
  500 on PP+1 goto 510,520,530,540,550,560,570,580
  510 BC=0:RC=0:GC=0:goto 590
  520 BC=1:RC=0:GC=0:goto 590
  530 BC=0:RC=1:GC=0:goto 590
  540 BC=1:RC=1:GC=0:goto 590
  550 BC=0:RC=0:GC=1:goto 590
  560 BC=1:RC=0:GC=1:goto 590
  570 BC=0:RC=1:GC=1:goto 590
  580 BC=1:RC=1:GC=1
  590 if ZZ=1 then return
  600 for D=0 to 15:for DD=0 to 31:B(DD,D)=BC:R(DD,D)=RC:G(DD,D)=GC:I(DD,D)=IC:next DD:next D
  610 locate 2+X,Y+1:print ".":X=16:Y=8:line (16,8)-(271,135),CO,BF:return
  620 if G$<>"x" then 720
  630 for YY=0 to 15:for XX=0 to 15:A=B(XX,YY):B(XX,YY)=B(31-XX,YY)
  640 B(31-XX,YY)=A:next XX:next YY
  650 for YY=0 to 15:for XX=0 to 15:A=R(XX,YY):R(XX,YY)=R(31-XX,YY)
  660 R(31-XX,YY)=A:next XX:next YY
  670 for YY=0 to 15:for XX=0 to 15:A=G(XX,YY):G(XX,YY)=G(31-XX,YY)
  680 G(31-XX,YY)=A:next XX:next YY
```

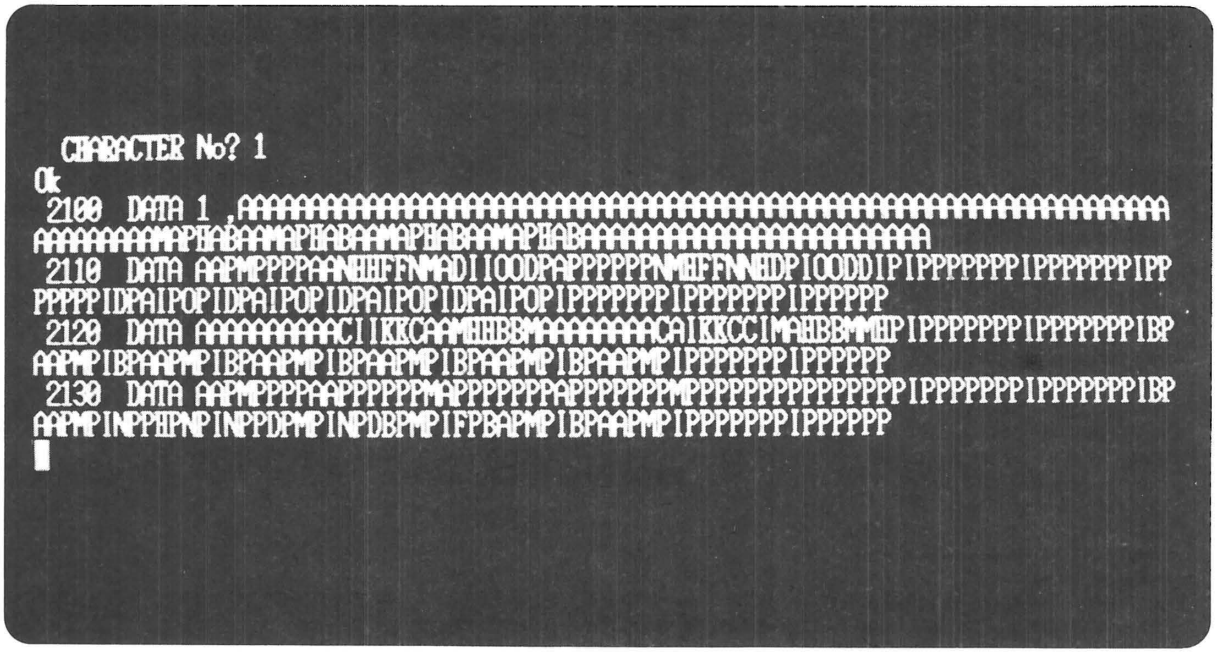
```
  690 for YY=0 to 15:for XX=0 to 15:A=I(XX,YY):I(XX,YY)=I(31-XX,YY)
  700 I(31-XX,YY)=A:next XX:next YY
  710 goto 450
  720 if G$<>"y" then 310
  730 for XX=0 to 31:for YY=0 to 7:A=B(XX,YY):B(XX,YY)=B(XX,15-YY)
  740 B(XX,15-YY)=A:next YY:next XX
  750 for XX=0 to 31:for YY=0 to 7:A=R(XX,YY):R(XX,YY)=R(XX,15-YY)
  760 R(XX,15-YY)=A:next YY:next XX
  770 for XX=0 to 31:for YY=0 to 7:A=G(XX,YY):G(XX,YY)=G(XX,15-YY)
  780 G(XX,15-YY)=A:next YY:next XX
  790 for XX=0 to 31:for YY=0 to 7:A=I(XX,YY):I(XX,YY)=I(XX,15-YY)
  800 I(XX,15-YY)=A:next YY:next XX
  810 goto 450
  820 AD=&HA000:for D=0 to 2:read ST$:for DD=1 to 127 step 2:poke AD,(asc(mid$(ST$,DD,1))-65)*16+(asc(mid
$(ST$,DD+1,1))-65):AD=AD+1:next DD:next D:return
  830 data DKPPKDGHCOAABBAAKIBJCCPOKDAOEANNCBAAKECKPOKDHOMLEHMCCDKANNDGAAAAMDCHKANNDGAAABMLEPMCDDKANNDGAB
AAMDDHKANNDGABABMLFHMCEDKANNDGACAA
  840 data MDEHKANNDGACABMLFPMCFDKANNDGADAAMDFHKANNDGADABMLGHMCGDKANNDGAEAAMDGHKANNDGAEABMLGPMCHDKANNDGAF
AAMDHHKANNDGAFABMLHHMCIDKANNDGAGAA
  850 data MDIHKANNDGAGABMLHPMCJDKANNDGAHAAMDJHKANNDGAHABBBAIAANNBJCDHJNGABEPMCBGKACCPOKDMJAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```



データメイカ

エディタをBREAKで止め，データメイカをロードします。ＲＵＮするとキャラクタナンバーをたずねてくるので0-23の数を入れます。ＤＡＴＡ文を作り，ＥＮＤになります。次のデータを作るには，もう一度ＲＵＮさせてください(つまり24個のキャラクタを作るには24回ＲＵＮさせることになるのです)。

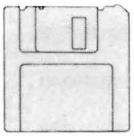
(注)　キャラクタナンバーと作成されるＤＡＴＡ文の行番号は，行番号の2000＋100×(キャラクタナンバー)の関係にあります。



《写真１－３》データメイカ実行画面

《リスト１－２》RERUデータメイカ・プログラム

```
   10 clear &HA000:goto 900
  900 init "crt1:80,25,1,1":cls 3:locate 2,2:input "CHARACTER No";NO:if NO>23 then 900
  910 PE=&HA800+256*NO:LI=2000+NO*100:locate 0,4:print LI;" DATA";NO;".";:for D=0 to 63:A=peek(PE+D)/16:B
=int(A):C=(A-B)*16
  920 print chr$(B+65);chr$(C+65);:next D:print
  930 for DD=1 to 3:print LI+DD*10;" DATA ";:for D=0 to 63:A=peek(PE+D+64*DD)/16:B=int(A):C=(A-B)*16:prin
t chr$(B+65);chr$(C+65);:next D
  940 print:next DD:locate 0,3:key 0,chr$(13)+chr$(13)+chr$(13)+chr$(13):end
 1000 AD=&HA000:for D=0 to 1:read ST$:for DD=1 to 127 step 2:poke AD,(asc(mid$(ST$,DD,1))-65)*16+(asc(mid
$(ST$,DD+1,1))-65):AD=AD+1:next DD:next D
 1010 read A:if A=810 then 100
 1020 AD=&HA800+256*A:for D=0 to 3:read ST$:for DD=1 to 127 step 2:poke AD,(asc(mid$(ST$;DD,1))-65)*16+(a
sc(mid$(ST$,DD+1,1))-65):AD=AD+1:next DD:next D:goto 1010
 1070 data PDDKAAKHGHDKABKHGPDKACKHOFGHCOAABBAAK1BJOLOBNFEMHNCBAAAABBEAABAGA1CJIHDAABBJBAPJBBAAMABJACAAAJ
NBOLONHDAFKHDBEMAAONFDADKHDOCADCAH
 1080 data KHDOAGNDLEDKAHKHNDLFDMNDLFABPPBAONFLADKHONKAONKAONKAONKAOLDJOLBAPDDKAHKHMGACDCAHKHPOC1CANEONHL
AFKHPLMJAAAAAAAAAAAAAAAAAAAAAAAA
 6000 data 810
```



フィニッシュ

こうしてＤＡＴＡ文の芸術ができあがったら900～940行までをＤＥＬＥＴＥし，後は10行に

10　CLEAR　&HA000：INIT〝CRT2：640, 200, 16″：DEF　USR＝&HA000：GOTO 1000

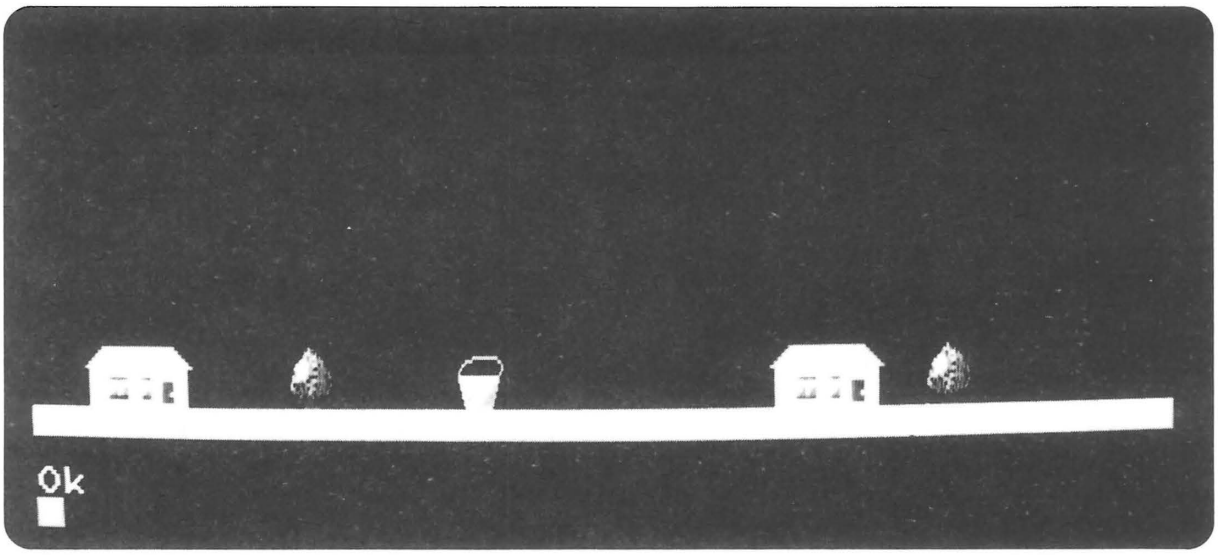
を入れ，100行からあなたがゲームを組めばいいのです。なお，Ｘ，Ｙ座標，キャラクタナンバーをＰＯＫＥするアドレスは

POKE　&HA700，Ｘ座標（０～76）

POKE　&HA701，Ｙ座標（０～46）

POKE　&HA702，キャラクタナンバー（０～23）

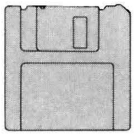
以上のようになっています。後はＡ＝ＵＳＲ（０）などを実行すれば画面上にキャラクタが現れます。**リスト１－３**のようなサンプルを組んでみたので参考にして下さい。



《写真１－４》サンプル画面

《リスト1－3》RERUサンプル・プログラム

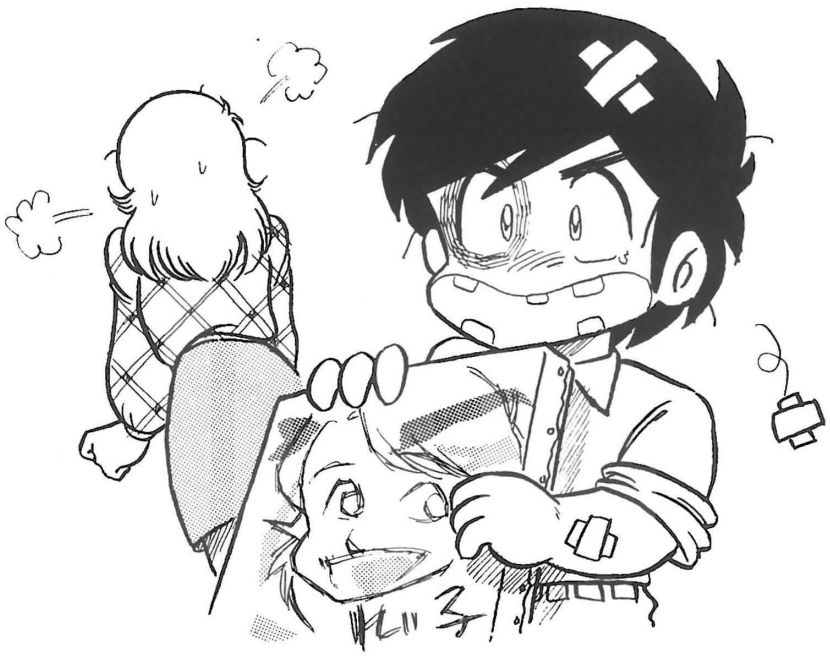
```
  10 clear &HA000:init "crt2:640,200,16":def usr=&HA000:goto 1000
 100 init "crt1:40,25,0,0":kmode 0:klist 0
 110 color 1:locate 0,20:print "■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■"
 120 X=&HA700:Y=X+1:C=X+2
 130 poke Y,36:poke X,4:poke C,1:A=usr(0):poke X,8:poke C,2:A=usr(0)
 140 poke X,18:poke C,3:A=usr(0)
 150 poke X,30:poke C,0:A=usr(0)
 160 poke X,52:poke C,1:A=usr(0):poke X,56:poke C,2:A=usr(0)
 170 poke X,62:poke C,3:A=usr(0)
 200 end
1000 AD=&HA000:for D=0 to 1:read ST$:for DD=1 to 127 step 2:poke AD,(asc(mid$(ST$,DD,1))-65)*16+(asc(mid
$(ST$,DD+1,1))-65):AD=AD+1:next DD:next D
1010 read A:if A=810 then 100
1020 AD=&HA800+256*A:for D=0 to 3:read ST$:for DD=1 to 127 step 2:poke AD,(asc(mid$(ST$,DD,1))-65)*16+(a
sc(mid$(ST$,DD+1,1))-65):AD=AD+1:next DD:next D:goto 1010
1070 data PDDKAAKHGHDKABKHGPDKACKHOFGHCOAABBAAKIBJOLOBNFEMHNCBAAAABBEAABAGAICJIHDAABBJBAPJBBAAMABJAGAAAJ
NBOLONHDAFKHDBEMAAONFDADKHDOCADCAH
1080 data KHDOAGNDLEDKAHKHNDLFDMNDLFABPPBAONFLADKHONKAONKAONKAONKAOLDJOLBAPDDKAHKHMGACDCAHKHPOCICANEONHL
AFKHPLMJAAAAAAAAAAAAAAAAAAAAAAAA
2000  data 0 ,AAAAAAAAAAAAAAAAAPAPPADAAAPAAAMMAAAAABACAAAAABABAAAAAAIPAPPPPAHPAPPPPAHOAPPHPACOAPPDPACMA
PPBPABMAPPAHABIAPPIDAAIAHPIAAAAAPPHPAA
2010  data AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAPAPPPPADPAPPPPAHOAPPHPACOAPPDPACMAPPB
PABMAPPAHABIAPPIDAAIAHPIAAAAAPPHPAA
2020  data AAAAAAAAAAAAAAAAAPAPPADAAAPAAAMMAAAAABACAAAAABABAAAAAAIPAPPPPAHPAPPPPAHOAPPHPACOAPPDPACMAPPB
PABMAPPAHABIAPPIDAAIAHPIAAAAAPPHPAA
2030  data AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAABAAAAMAABAAAAO
AAAAAAAPIAAAAAAHMAAAAIAHPAAAAAAAAAA
2100  data 1 ,AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAMAPHABAAAMAPHABAA
MAPHABAAMAPHABAAAAAAAAAAAAAAAAAAAAAAAA
2110  data AAPMPPPPAANHHFFNMADIIOODPAPPPPPPNMHFFNNHDPIOODDIPIPPPPPPPPIPPPPPPPIPPPPPPPIDPAIPOPIDPAIPOPIDPA
IPOPIDPAIPOPIPPPPPPPIPPPPPPPIPPPPPP
2120  data AAAAAAAAAACIIKKCAAMHHBBMAAAAAAAACAIKKCCIMAHBBMMHPIPPPPPPPIPPPPPPPIBPAAPMPIBPAAPMPIBPAAPMPIBPA
APMPIBPAAPMPIBPAAPMPIPPPPPPPIPPPPPP
2130  data AAPMPPPPAAPPPPPPMAPPPPPPPAPPPPPPPMPPPPPPPPPPPPPPPIPPPPPPPIPPPPPPPIBPAAPMPINPPHPNPINPPDPMPINPD
BPMPIFPBAPMPIBPAAPMPIPPPPPPPIPPPPPP
2200  data 2 ,AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADOAAAAAADOAACAAADO
AAAAAADOAAAEAAAAAAAAAAAAAAAAAAAAAPAAPAA
2210  data PPPPAPAANHHFDNAADIIOODAAPPPPPPADHFFNNHANIOODDIDOPPPPPPAHPPPPPPAHPPPPPPAHMBPPPPAHMBPPNPAHMBPPP
PAHMBPPPPAHPPPPPPAHPPPPPPAHPPAPPAAH
2220  data AAAAAAAACIIKACAAMHHBBMAAAAAAAAAAIKKCCIACHBBMMHABPPPPPPAHPPPPPPAHIAPPPPAHIAAPPAAHIAAPNAAHIAAPP
AAHIAAPPEAHIAAPPAAHPPAPPAAHPPPPPPAH
2230  data PPPPAPAAPPPPPDPAAPPPPPPAAPPPPPPADPPPPPPAPPPPPPPDPPPPPPPAHPPPPPPAHIAPPPPAHLOAPPAAHJOAPPAAHIOAPP
AAHICAPPEAHIAAPPAAHPPAPPAAHPPPPPPAH
2300  data 3 ,AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
2310  data AAAAAAAAAAAAAAAAAAAABAAAAAAABAAAAAAAAAAAAAAAABAAAAAAABAAAAAAAKAAAAAADAAAAAAAFAAAAAAAFAAAAAAA
EAAAAAAADAAAAIAAHAAAAMAAPAAAAOADPAA
2320  data AAAAABAAAAMABHAAAAOAFOAAAAPINKABAAPOLNABAAPPOGADAAPPLKAKIAPOOBBJIAHHKMADEAMKKCBCMAFJFKAJIAFHF
DAFAAJICEAAAAAAAAAAAAAAAAAAAAAAAA
2330  data AAAAABAAAAEAABAAAACAAMAAAAFIBCAAAAOOAMAAAAPPAAAAAANKAIAAIAFOAAAAIACHAMAAEAIAAAAAEAAAACAAAAAAA
BAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
6000 data 810
```



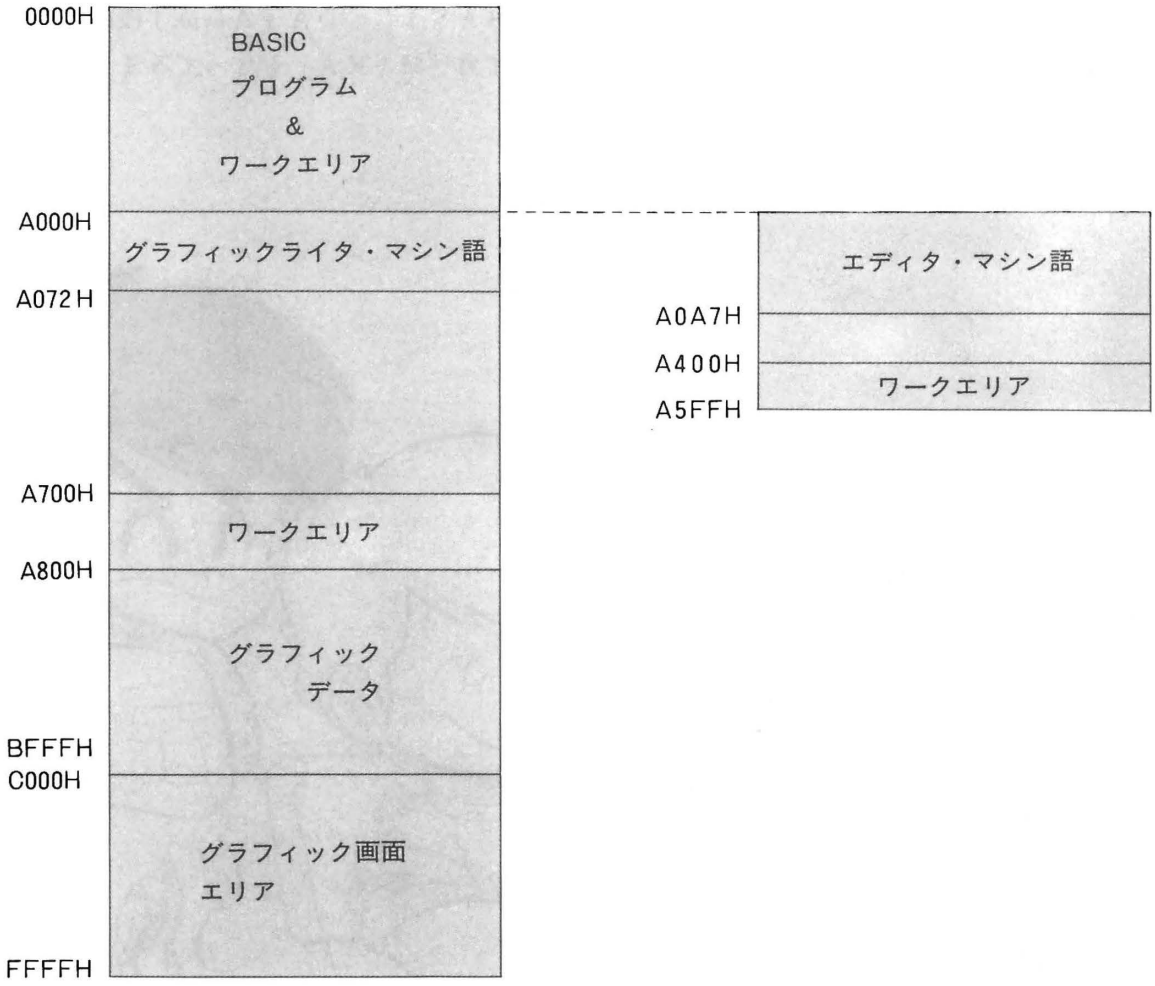
マシン語について

エディタにもデータメイカにも行の終りの方にAからPまでのアルファベットが並んでいますが，実はこれが0～FHまでに対応しているマシン語なのです（例えばAAは00H，MJはC9H）。

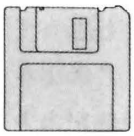
エディタの方のマシン語は1バイトのキャラクタデータをメモリから読出し，1ビットずつに分解するためにあり，データメイカの方はグラフィックを画面に高速表示するためにあります。



両方ともまあ短めに組めたと思っているのですが，グラフィック表示ルーチンの方はまだ改造の余地がありそうです（ＬＤＩ命令でＰＵＴしていますが，ＰＯＰとＬＤ命令を組み合わせた方が高速になるはずです）。メモリマップ（**第１－１図**）を載せておきますので改造にチャレンジする方はがんばって下さい。



《第１－１図》 RERUメモリマップ



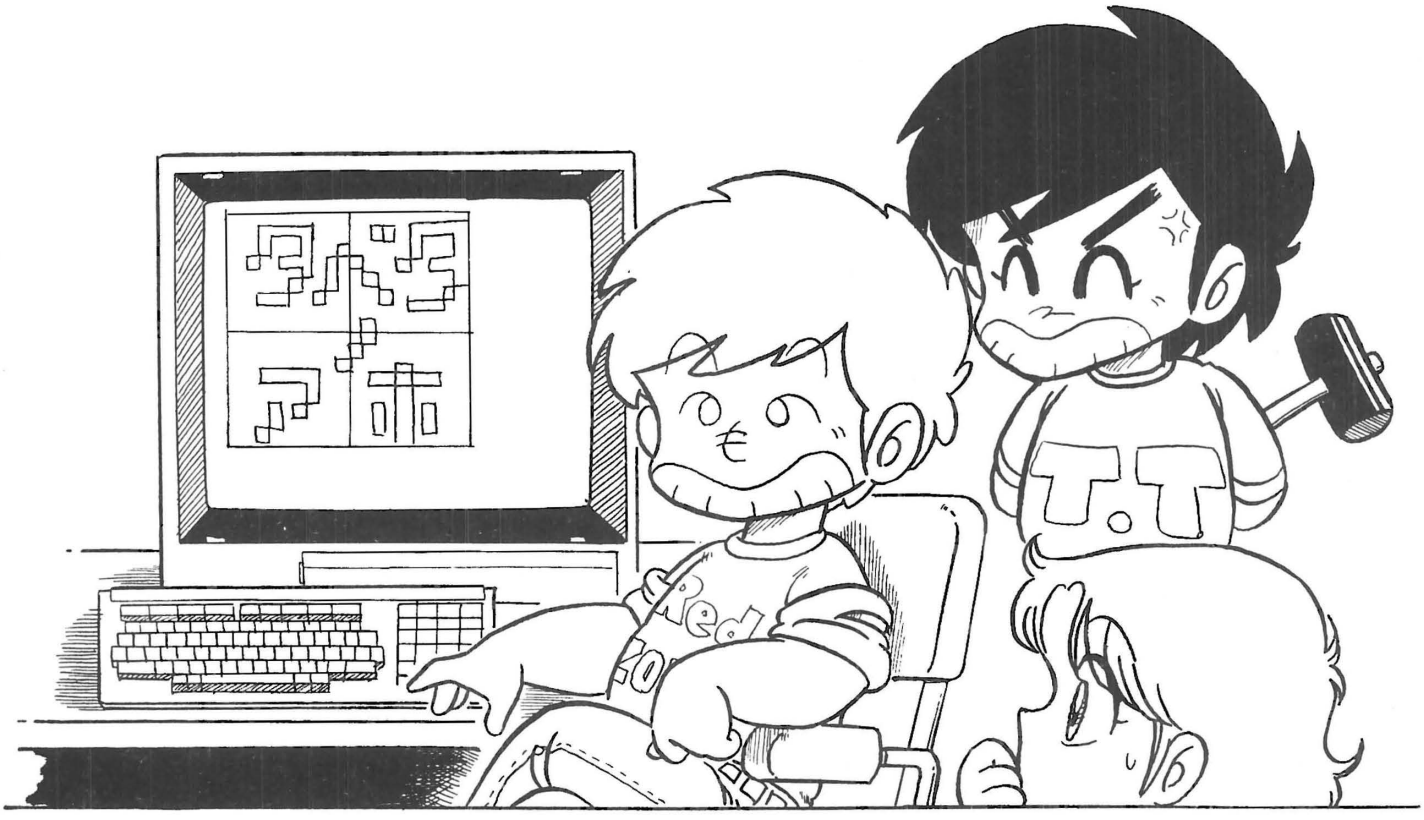
VS・テキスト画面

さて，普通のパソコンのゲームならグラフィックキャラクタとテキスト画面に出力するＰＣＧキャラクタは「グラフィックに背景，テキストに移動物」といったぐあいに重ね合わせに使われますが，スーパーＭＺは標準装備でもテキスト画面を２枚持っていて重ね合わせもできるため，単に合成をするだけならグラフィックキャラクタは必要ありません。

しかしあの**「プライオリティ機能」**を使うとなると話は別で，どうしてもグラフィックによる背景が必要になります。そこで次はプライオリティの復習もかねてＰＣＧデータメイカとプライオリティのサンプルプログラムを載せてみました。

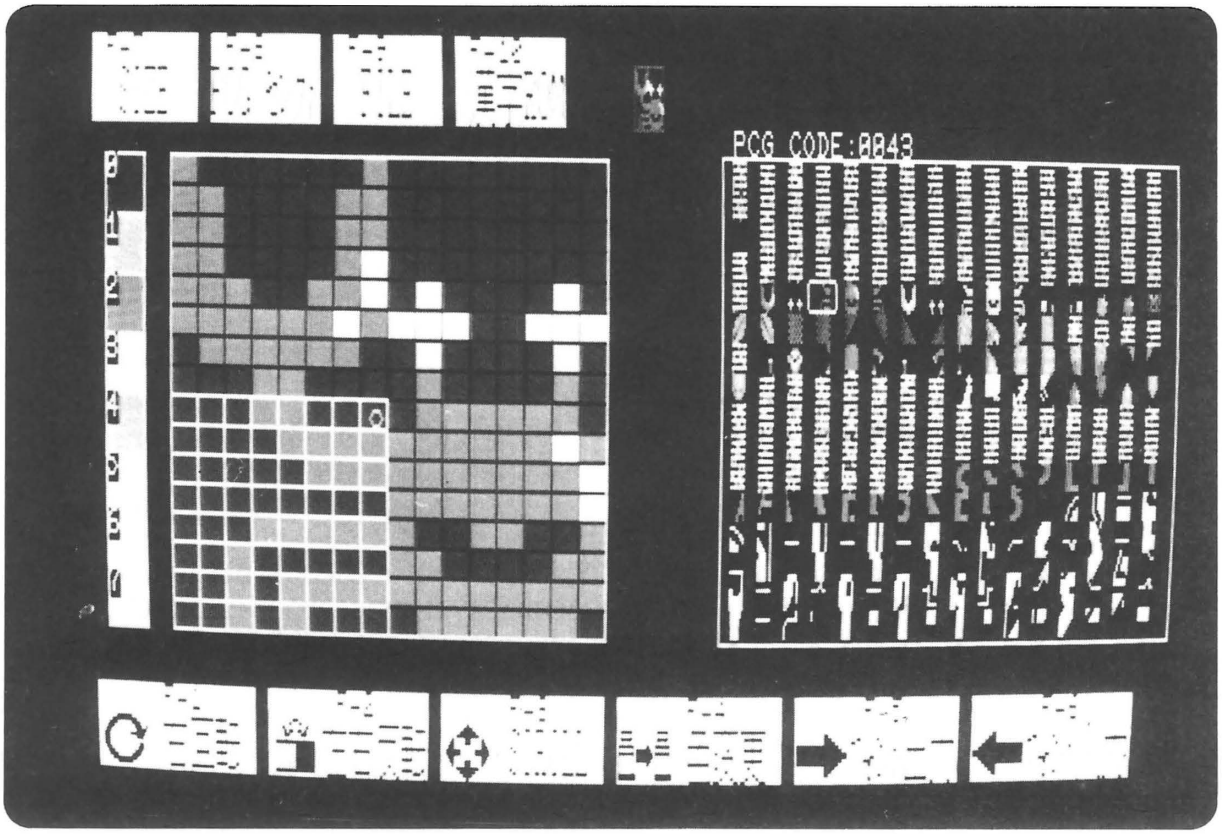
9-2 PCGデータメイカ「MIKA」

先にも書きましたがスーパーＭＺのアプリケーションの中には使い勝手のよい**「ＰＣＧエディタ」**がありますが，そこで作ったキャラクタを直接ＢＡＳＩＣのＤＡＴＡに直す機能はありません。そこで簡単なプログラムながら**ＰＣＧデータメイカ「ＭＩＫＡ」**を作ってみました。



使用方法

ＰＣＧエディタでキャラクタを作り終わったらF 9を押してＢＡＳＩＣに戻り,「ＭＩＫＡ」(リスト２－１）をロードします。ＲＵＮするとキャラクタナンバーをたずねてくるので0～255の数を入れます。2000行以降に20000＋10×(キャラクタナンバー）の関係でＤＡＴＡ文を作成し，ＥＮＤになります。次のＤＡＴＡ文を作るには再びＲＵＮさせて下さい。



《写真２－１》スーパーMZユーティリティ「PCGエディタ」

```
  CHARACTER No? 70
Ok
 2700  DATA  70 ,81C3C3C3E7FF7E1881C3C3C3E7FF7E1800000000101020000
list
   10 init "crt1:80,25,1,1":cls 3:locate 2,2:input "CHARACTER No";NO:if NO>255 t
hen 10
   20 B$=cgpat$(NO):print
   30 print 2000+NO*10;" ";"DATA ";NO;",";ascchr$(B$):locate 0,3
   50 key 0,chr$(13):end
 1000 read A:if A=300 then 100
 1010 read B$:def chr$(A)=hexchr$(B$):goto 1000
 2700  data  70 ,81C3C3C3E7FF7E1881C3C3C3E7FF7E1800000000101020000
 5000 data 300
Ok
```

《写真２－２》MIKA実行画面

こうしてＤＡＴＡ文のかたまりができたら10行～50行をＤＥＬＥＴＥし，10行に
INIT〝CRT1：40，25，0，0″：CGEN　1：GOTO　1000
を入れます。

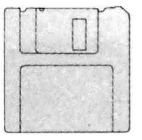
後は100行からプログラミングするだけです。**リスト２－２**の様なサンプルを組んでみたので参考にして下さい。

《リスト２－１》MIKA（PCGデータメイカ）プログラム

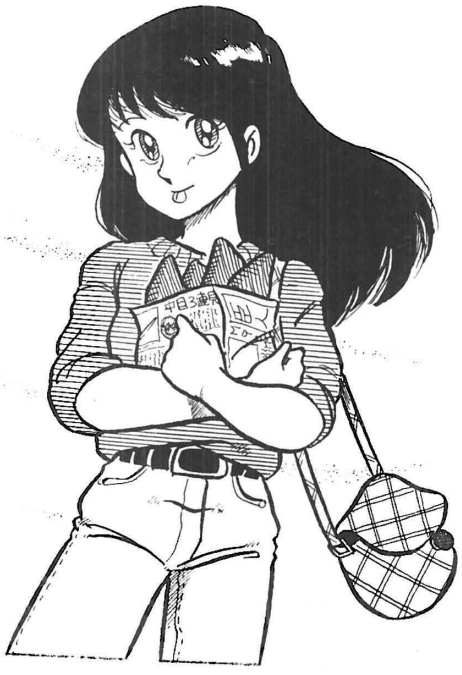
```
  10 init "crt1:80,25,1,1":cls 3:locate 2,2:input "CHARACTER No";NO:if NO>255 then 10
  20 B$=cgpat$(NO):print
  30 print 2000+NO*10;" ";"DATA ";NO;",";ascchr$(B$):locate 0,3
  50 key 0,chr$(13):end
1000 read A:if A=300 then 100
1010 read B$:def chr$(A)=hexchr$(B$):goto 1000
5000 data 300
```

《リスト２－２》MIKAサンプルプログラム

```
   10 init "CRT1:40,25,0,0":cgen 1:goto 1000
  100 locate 19,18:print "FGF":locate 19,19:print "HIJ"
  110 end
 1000 read A:if A=300 then 100
 1010 read B$:def chr$(A)=hexchr$(B$):goto 1000
 2700  data  70 ,81C3C3C3E7FF7E1881C3C3C3E7FF7E1800000000101020000
 2710  data  71 ,0000000000000004200000000042E7424200000000042E74200
 2720  data  72 ,180F07001F202F28180F07001F202F2800000000000000000
 2730  data  73 ,7EFFFFFF99C3FF7E7EFFFFFF99C3FF7E0002010100000000
 2740  data  74 ,18F0E000F804F41418F0E000F804F4140000000000000000
 5000 data 300
```



プライオリティ

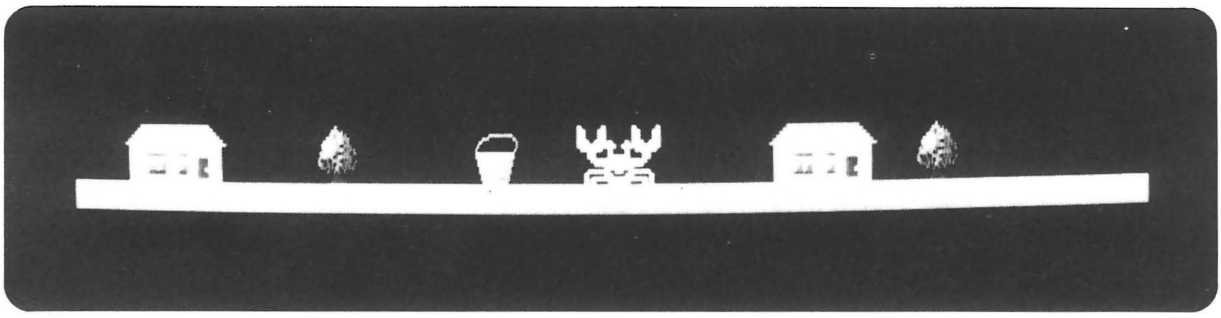


スーパーＭＺの特長として**「グラフィック画面とテキスト画面のパレットごとのプライオリティ」**機能があることは，みなさん既に御存知でしょう。ゲームなどで，この機能を使うと結構面白いものができます。

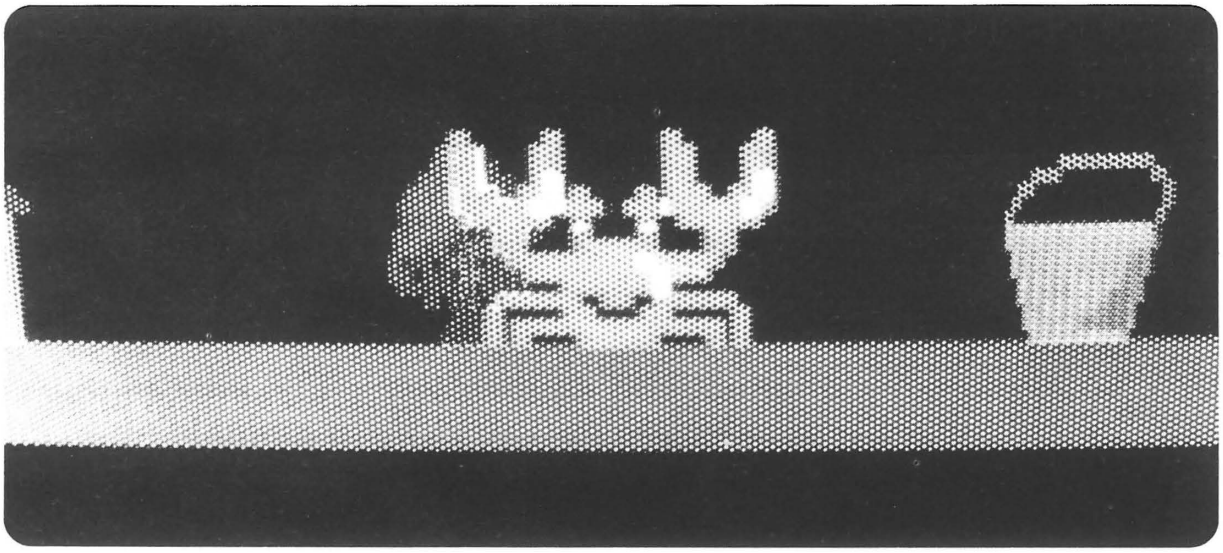
ためしに**リスト１－３**と**リスト２－２**を組合わせ，**リスト２－３**の様なサンプルを組んでみました。グラフィックデータなどはまったくそのままですが120行のＰＲＩＯＲＩＴＹ命令でパレットコードの５～８をテキスト画面よりも優先に表示させています。

家や木の前では**写真２－４**の様にカニは前に出て来ますが，５～８のコードで書かれているバケツと重なると**写真２－５**の様に後にまわります。

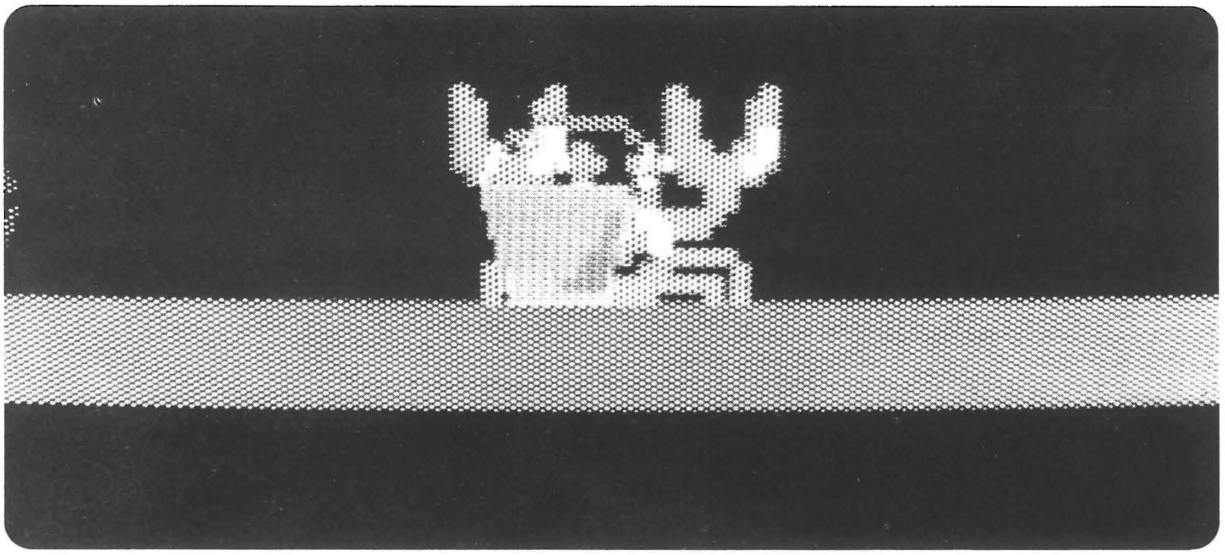
「ＲＥＲＵ」や「ＭＩＫＡ」，そしてプライオリティ機能を使ってさらに面白いものを作ってみて下さい。



《写真２－３》 RERUとMIKAが仲良く合体



《写真２－４》家や木の前ではカニが優先

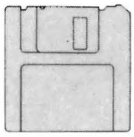


《写真2－5》カニとバケツではバケツが優先

《リスト２－３》RERU＋MIKAサンプルプログラム

```
 10 clear &HA000:init "crt2:640,200,16":def usr=&HA000:goto 1000
 100 init "crt1:40,25,0,0":kmode 0:klist 0
 110 color 1:locate 0,20:print "[redacted]"
 120 priority 32+64+128+256
 130 X=&HA700:Y=X+1:C=X+2:poke Y,36:poke X,4:poke C,1:A=usr(0):poke X,8:poke C,2:A=usr(0)
 140 poke X,18:poke C,3:A=usr(0)
 150 poke X,30:poke C,0:A=usr(0)
 160 poke X,52:poke C,1:A=usr(0):poke X,56:poke C,2:A=usr(0)
 170 poke X,62:poke C,3:A=usr(0)
 180 KANIX=18:repeat on ,4:ogen 1
 190 G$=inkey$
 200 if G$="4" then KANIX=KANIX-1:if KANIX<0 then KANIX=0
 210 if G$="6" then KANIX=KANIX+1:if KANIX>35 then KANIX=35
 220 locate KANIX,18:print " FGF ":locate KANIX,19:print " HIJ "
 230 for D=0 to 1000:next D
 240 goto 190
1000 AD=&HA000:for D=0 to 1:read ST$:for DD=1 to 127 step 2:poke AD,(asc(mid$(ST$,DD,1))-65)*16+(asc(mid
$(ST$,DD+1,1))-65):AD=AD+1:next DD:next D
1010 read A:if A=810 then 5000
1020 AD=&HA800+256*A:for D=0 to 3:read ST$:for DD=1 to 127 step 2:poke AD,(asc(mid$(ST$,DD,1))-65)*16+(a
sc(mid$(ST$,DD+1,1))-65):AD=AD+1:next DD:next D:goto 1010
1070 data PDDKAAKHGHDKABKHGPDKACKHOFGHCOAABBAAKIBJOLOBNFEMHNCBAAAABBEAABAGAICJIHDAABBJBAPJBBAAMABJAGAAAJ
NBOLONHDAFKHDBEMAAONFDADKHDOCADCAH
1080 data KHDOAGNDLEDKAHKHNDLFDMNDLFABPPBAONFLADKHONKAONKAONKAONKAOLDJOLBAPDDKAHKHMGACDCAHKHPOC1CANEONHL
AFKHPLMJAAAAAAAAAAAAAAAAAAAAAAAA
2000  data 0 ,AAAAAAAAAAAAAAAAAPAPPADAAAPAAAMMAAAAABACAAAAABABAAAAAAIPAPPPPAHPAPPPPAHOAPPHPACOAPPDPACMA
PPBPABMAPPAHABIAPPIDAAIAHPIAAAAAPPHPAA
2010  data AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAPAPPPPADPAPPPPAHOAPPHPACOAPPDPACMAPPB
PABMAPPAHABIAPPIDAAIAHPIAAAAAPPHPAA
2020  data AAAAAAAAAAAAAAAAAPAPPADAAAPAAAMMAAAAABACAAAAABABAAAAAAIPAPPPPAHPAPPPPAHOAPPHPACOAPPDPACMAPPB
PABMAPPAHABIAPPIDAAIAHPIAAAAAPPHPAA
2030  data AAAAAAAAAAAAAAAAAAAA
```

9-3 やまたのおろち



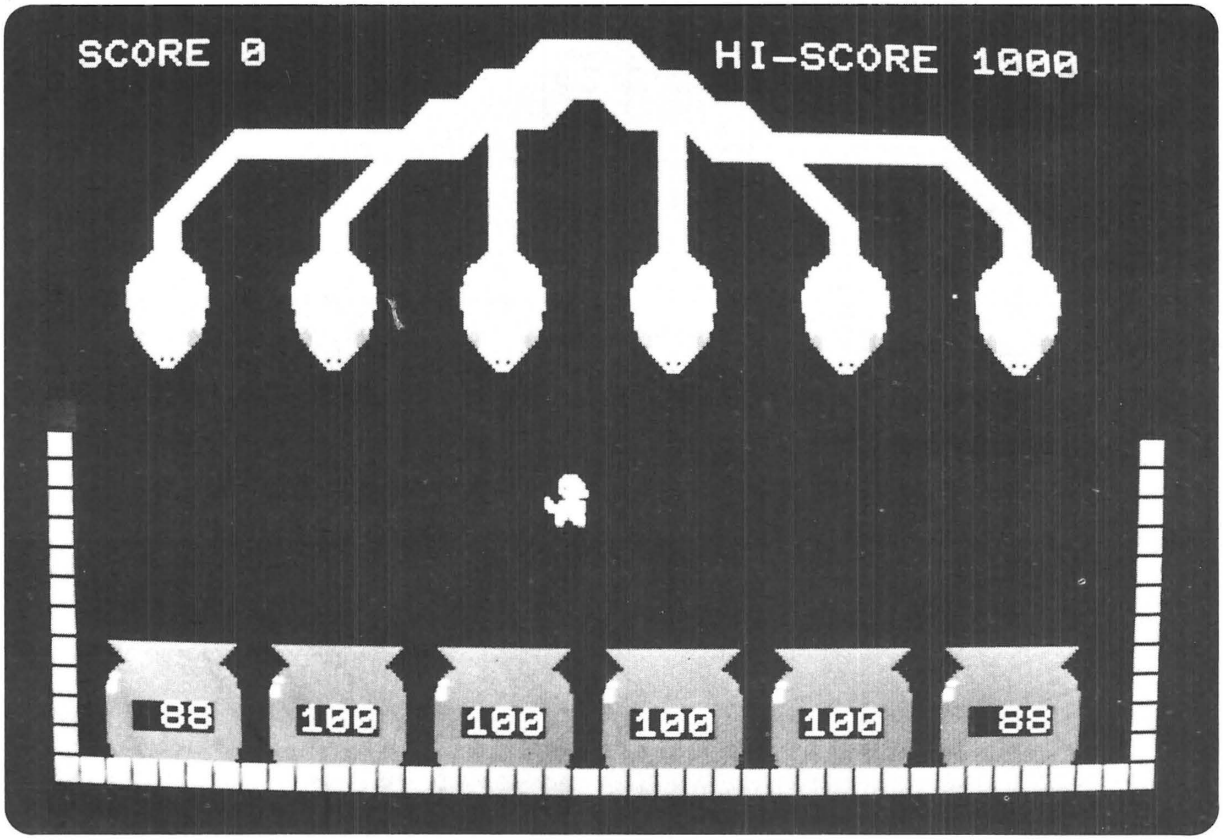
ストーリー

昔々の天上界にスサノオという青年がいました。彼はその若さゆえのエネルギーを持て余し，夜な夜なディスコでフィーバーしたり，カフェバーでバーボンを飲んで暴れまわったりしていましたが，ある日姉の寝室にラッコの毛皮を投げ込み，それがもとで親戚一同のひんしゅくをかい，とうとう天上界を追い出されてしまいました。

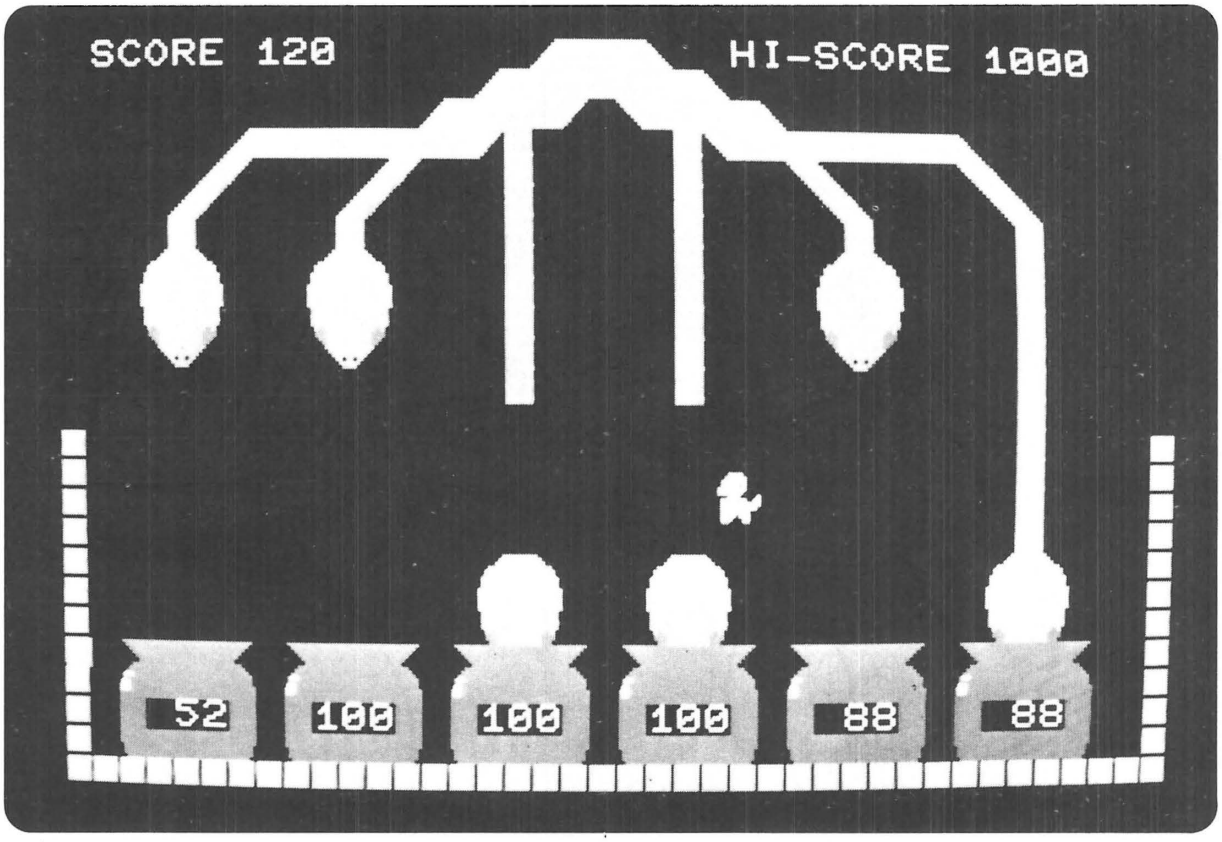
さて，下界にやって来た彼は，悲しそうに泣いている一人の若い娘に出会いました。

「Hey，そこのギャル！何泣いてんのさ。よかったら俺と一緒にTea飲まない？」

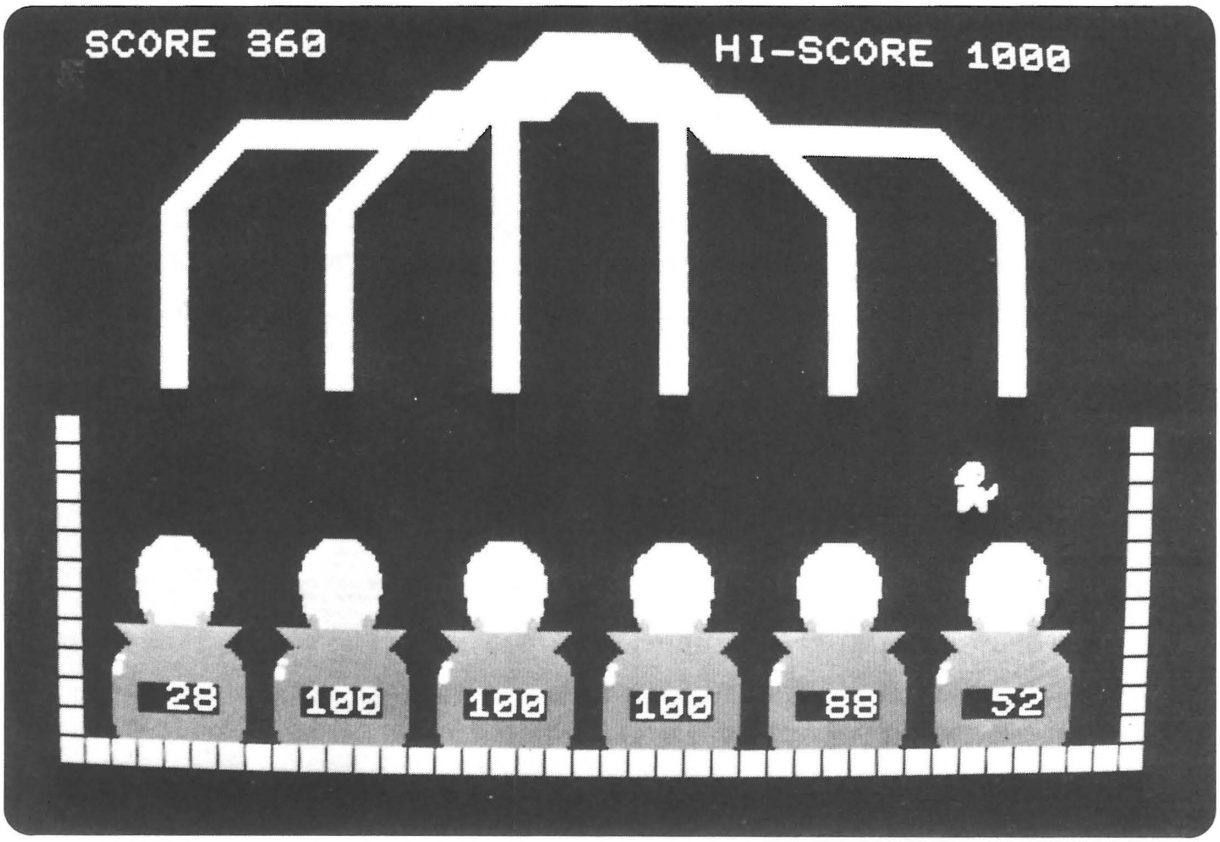




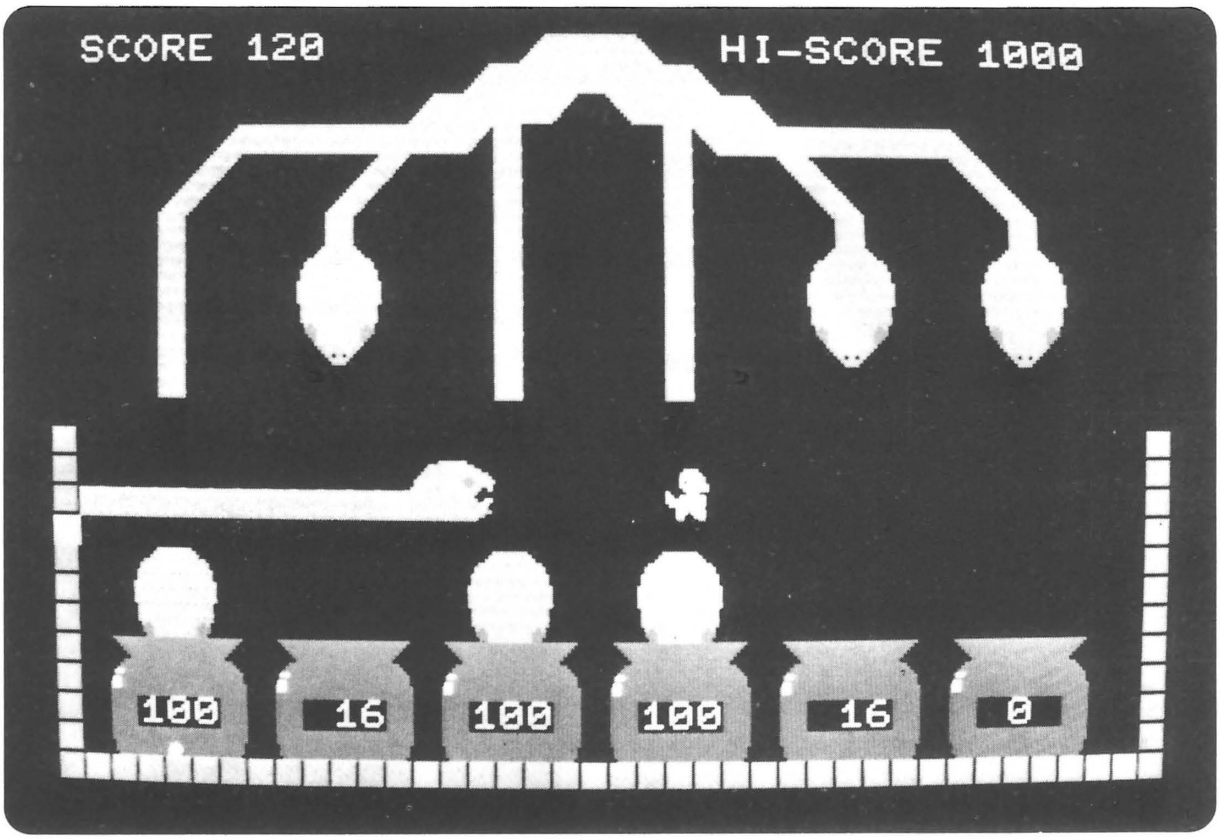
《写真3－1》スタート画面。中央にいるのがスサノオ



《写真3－2》酒ガメに首をのばす蛇をすかさず切り倒す



《写真３－３》六つの首を切り落とすと１面クリア



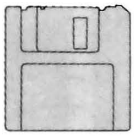
《写真３－４》酒ガメがカラになるとサイドスネークが出現する

と声をかけ，彼はアマンドにその娘を連れ込みました。窓辺の席でチーズケーキを食べながらその娘が言うには

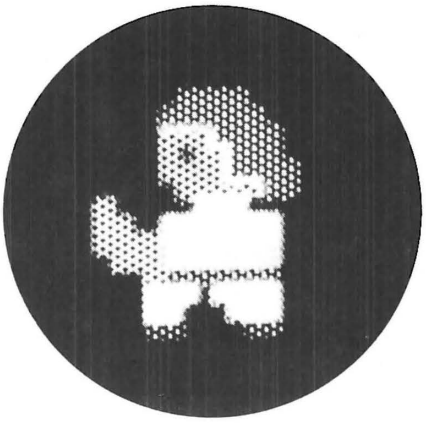
「実は次の満月の夜，八つの首を持つ大蛇がローンの取りたてにやって来るのです。そこで我家では私を借金の肩代わりにしたのです」。

泣きながら身の上を語る彼女に，彼は下心丸出しで同情し，大蛇と戦う決意を固めました。

そしてやってきた満月の晩。彼は大蛇の好物である酒を用意し，泥酔した蛇を倒すという世にもヒキョウな作戦を練り上げ，大蛇を迎え撃ったのでした。さてスサノオの運命やいかに!?

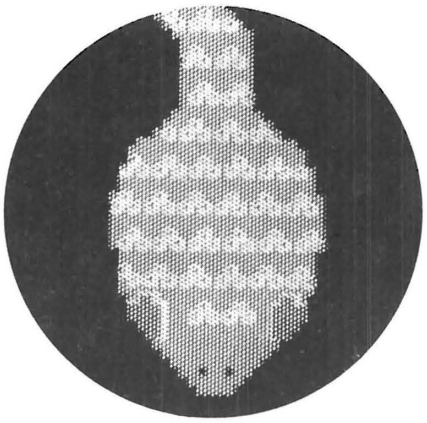


ルール



①スサノオ

力ばっかり余っている青年。[4]，[6]のキーで左右に動き，スペースキーで蛇の首を切ることができますが，蛇が首をのばした時に真下にいると喰われてしまいます。



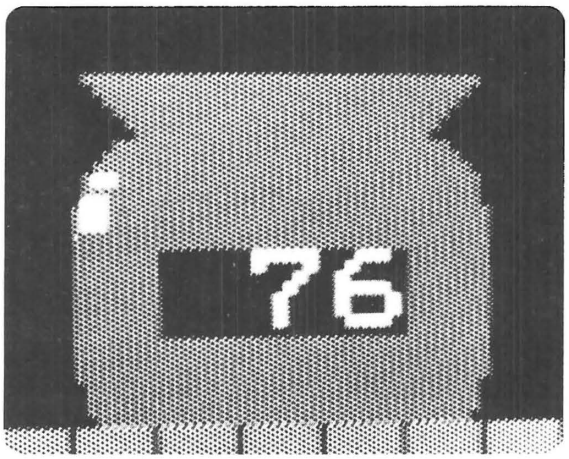
②ヤマタノオロチ

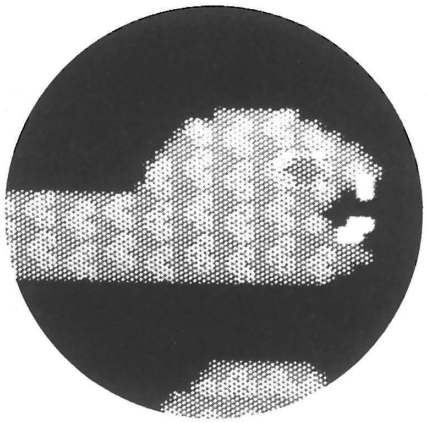
「ヤマタ」というわりには首が6本しかありませんが，実は1ピキはどこかに隠れていて，もう1ピキは自我に目覚めて1人立ちしてしまっていないのです。ま，とにかく，これがヤマタノオロチです。

画面下にある酒ガメにむかって首をのばしてきて，ある程度飲むと引っ込み，またしばらくすると首をのばします。これを繰返すのですが，もし酒ガメがカラになると……。

③酒ガメ

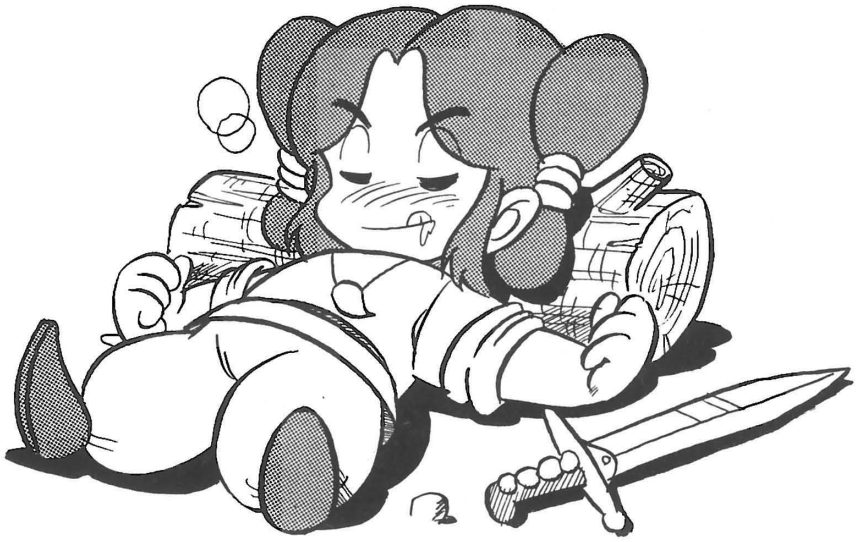
中央に酒の残量がパーセントで表示されています。





④サイドスネーク

酒ガメがカラになると左横から現れ，スサノオに襲いかかります。



スコア

蛇の首を切ると上がりますが，その蛇の飲んだ酒の量に比例して高得点になります。10点から120点ぐらいでしょう。

1面クリア&ゲームオーバー

蛇の首を6本とも切り落とすと1面クリアになり，さらに難しい次の面に進みます。またスサノオが3回やられるとゲームオーバーとなります。

プログラム

各行の働きは**第3表**のようになっています。注意しなければならないことは10行でキーリピートを最高速にしているため，テストランなどをさせた後は[CTRL]－[D]でキャラクタとリピートスピードをノーマルに戻してから次の作業をして下さい。

《第3表》プログラム説明

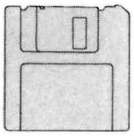
20～100	メインルーチン
110～180	首切り
190～310	蛇の首の動き
320～330	サイドスネークの動き
340～390	ゲームオーバー
400～680	初期設定
690～820	デモ
830～	ＰＣＧデータ

《リスト 3》やまたのおろち・プログラム

```
   10 dim Q(5),Z(5):init "CRT1:40,25,0,0":klist 0:click off:cls 3:repeat on ,4:kmode 0:console 0,25:HS=10
00:gosub 840:goto 690
   20 if M0=0 then MS=int(rnd(1)*6):if Z(MS)=1 then gosub 190:if DE=1 then goto 340
   30 if M0=1 then D0=D0+1:if D0=DR then Q(MS)=Q(MS)-12:D0=0:gosub 250:if DE=1 then goto 340
   40 GE$=inkey$
   50 if GE$="4" then M$="♦AB ♦♦♦♦CD ":locate MM,14:print M$:RL=-1:KN=-1
   60 if GE$="6" then M$="♦ EF♦♦♦♦ GH":locate MM,14:print M$:RL=1:KN=1
   70 if GE$=" " then gosub 110:if EN=1 then play "E16C16E16C16E16G16G16G16F16D16F16D16E16C16C16":play "R
32":goto 410
   80 if peek@(&H38,560+MM+RL*2)=32 then MM=MM+RL:locate MM,14:print M$
   90 RL=0:for D=0 to 600:next D
  100 goto 20
  110 PP=peek@(&H38,560+MM+KN*2):if PP<>81 then return
  120 play "G16"
  130 if KN=1 then locate MM,15:print "♦KL"
  140 if KN=-1 then locate MM,15:print "♦♦IJ"
  150 play "F32":locate XX+1,12:print " ♦♦ ♦♦ ♦♦ ♦♦ "
  160 locate MM,14:print M$:SC=SC+10*D0:locate 6,0:cgen 0:print SC:D0=0:cgen 1
  170 Z(MS)=0:M0=0:for D=0 to 5:if Z(D)=1 then return
  180 next D:EN=1:return
  190 M0=1:XX=3+6*MS:for D=7 to 9:locate XX,D
  200 print " Q ♦♦♦♦hQi♦♦♦♦jQk♦♦♦♦lQm♦♦♦♦npo":next D
  210 SV=560+XX:for D=0 to 2:if peek@(&H38,SV+D)<>32 then DE=1:return
  220 next D:play "o3c6o4":for D=10 to 15:locate XX,D
  230 print " Q ♦♦♦♦hQi♦♦♦♦jQk♦♦♦♦lQm♦♦♦♦npo":next D:locate XX,16
  240 print " Q ♦♦♦♦hQi♦♦♦♦jQk♦♦♦♦lQm":return
  250 if Q(MS)<0 then Q(MS)=0
  260 locate XX,22:print "   "
  270 cgen 0:locate XX,22:print str$(Q(MS)):cgen 1
  280 for D=15 to 7 step -1:locate XX,D
  290 print "hQi♦♦♦♦jQk♦♦♦♦lQm♦♦♦♦npo♦♦♦♦   ":next D
  300 if Q(MS)=0 then gosub 320
  310 M0=0:return
  320 play "03C404":for D=1 to MM:locate D,14:print " WXV♦♦♦♦♦VVVZ":for DD=0 to 250:next DD:next D
  330 DE=1:return
  340 for D=0 to 3:play "C32":locate MM,14:print "MN♦♦♦OP":play "G32":locate MM-1,14:print "MN♦♦♦OP":next
 D
  350 for D=0 to 3000:next D
  360 ME=ME-1:if ME>0 then goto 410
  370 locate 16,12:cgen 0:print "G A M E♦♦♦♦♦♦♦♦♦O V E R":play "E16C16E16C16E16G16G16G16F16D16F16D16E16C1
6C16"
  380 if SC>HS then HS=SC:locate 13,16:print "HI-SCORE";HS
  390 play "R32":for D=0 to 4000:next D:goto 690
  400 SC=0:ME=3:DR=15
  410 D0=0:DR=DR-2:if DR<5 then DR=5
  420 for D=0 to 5:Q(D)=100:Z(D)=1:next D
  430 M0=0:RL=0:KN=0:EN=0:DE=0
  440 M$="♦AB♦♦♦CD":MM=19:cgen 1:cls 1:cgen 0
  450 print " SCORE";SC
  460 locate 24,0:print "HI-SCORE";HS:cgen 1
  470 locate MM,14:print M$
  480 for D=0 to 39:poke@ &H38,960+D,91:next D
  490 for D=13 to 24:poke@ &H38,40*D,91:poke@ &H38,40*D+39,91:next D
  500 locate 0,0
  510 print "♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦UQQQT"
  520 print "               UQQQQQQQT"
  530 print "             UQQQQR SQQQQT"
  540 print "      UQQQQQQQQRQ     QSQQQQQQQQT"
  550 print "     UR    UR   Q     Q   ST    ST"
  560 print "    UR    UR    Q     Q    ST    ST"
  570 print "    Q     Q     Q     Q     Q     Q"
  580 print "   hQi   hQi   hQi   hQi   hQi   hQi"
  590 print "   jQk   jQk   jQk   jQk   jQk   jQk"
  600 print "   lQm   lQm   lQm   lQm   lQm   lQm"
  610 print "   npo   npo   npo   npo   npo   npo"
  620 locate 0,20
```

```
630 print "♦♦abbbc abbbc abbbc abbbc abbbc abbbc"
640 print "♦♦dbbbe dbbbe dbbbe dbbbe dbbbe dbbbe"
650 print "♦♦bbbbb bbbbb bbbbb bbbbb bbbbb bbbbb"
660 print "♦♦fbbbg fbbbg fbbbg fbbbg fbbbg fbbbg♦♦♦♦♦"
670 cgen 0:locate 3,22:print "100♦♦♦100♦♦♦100♦♦♦100♦♦♦100♦♦♦100":cgen 1
680 goto 20
690 cgen 1:width 40
700 for D=0 to 39
710 poke@ &H38,200+D,91
720 poke@ &H38,240+D,91
730 poke@ &H38,280+D,91:next D
740 cgen 0:locate 12,6:print "YAMATA NO OROCHI"
750 cgen 1:locate 21,12:print "AB♦♦♦♦IJD":for D=0 to 2000:next D
760 for D=0 to 14:locate D,12:print " WXY♦♦♦♦♦VVVZ":for DD=0 to 250:next DD:next D
770 cgen 0:locate 0,16
780 print "          YOUR KEY   RIGHT    (6)♦"
790 print "                     LEFT     (4)♦"
800 print "                     ATTACK(SPACE)♦"
810 print "          START      (SPACE)"
820 GE$=inkey$:if GE$=" " then goto 400
830 goto 820
840 read A:if A=300 then return
850 read B$:def chr$(A)=hexchr$(B$):goto 840
860  data  32 ,00000000000000000000000000000000000000000000000000000000
870  data  65 ,0000070F0100000000000000000E0B0F07000000000E0B0F07
880  data  66 ,000080C0E0E0F03000000000000000C0000000000000C0
890  data  67 ,83CFFF7F101D1800030F0F0F081D181803CFFF7F1F1D1800
900  data  68 ,80E0E0E000E0E00080E0E0E000E0E06080E0E0E0E0E0E000
910  data  69 ,000001030707070C0000000000000003000000000000000003
920  data  70 ,0000E0F080000000000000000070D0F0E000000000070D0F0E0
930  data  71 ,010707070007070001070707000707060107070707070700
940  data  72 ,C1F3FFFE00B81800C0F0F0F000B81818C1F3FFFEF8B81800
950  data  73 ,F07F1F0000000000000000000000000F07E1E0000000000
960  data  74 ,037F7F3F001D180003FFFF3F001D181803FFFF3F0F1D1800
970  data  75 ,C0FEFEFC00B81800C0FFFFFC00B81818C0FFFFFCF0B81800
980  data  76 ,0FFEF8000000000000000000000000000F7E700000000000
990  data  77 ,000000000000000000000000121110905000000000000000
1000  data  78 ,000000000000000000000000081020400000000000000000
1010  data  79 ,0000000000000000007C0005091121010000000000000000
1020  data  80 ,0000000000000000007C0040281000000000000000000000
1030  data  81 ,0044AA0000225500000000000000000FFBB55FFFFDDAAFF
1040  data  82 ,0044A8000020400000000000000000FFBA54F8F0C08000
1050  data  83 ,00442A0000020100000000000000000FF3B151F0F050201
1060  data  84 ,0040A0000020540000000000000000080040F0F8DCAAFF
1070  data  85 ,0000020000225500000000000000000103050F1F1D2AFF
1080  data  86 ,0224422002244220000000000000000FDDBBDDFFDDBBDDF
1090  data  87 ,0004020002244220000000000000000030D1F3D5B3DDF
1100  data  88 ,0224422002244220000000001030100FDDBBDDFFCD8BCDF
1110  data  89 ,0000402000040222000000C0E0C20200C0B0DC3E1B3ED2
1120  data  90 ,0000460C00204020000060C00000000E0C0A6FCFCDEB8C0
1130  data  91 ,FEFEFEFEFEFEFE0000000000000000000000000000000000
1140  data  97 ,0000000000000007F3F1F0F07030F1F0000000000000000
1150  data  98 ,000000000000000FFFFFFFFFFFFFFFF0000000000000000
1160  data  99 ,000000000000000FEFCF8F0E0C0F0F0000000000000000
1170  data  100 ,0030006060600003F7F7F7FFFFFFFF0030006060600000
1180  data  101 ,000000000000000FCFEFEFEFFFFFFFF0000000000000000
1190  data  102 ,000000000000000FFFFFFFF7F7F7F3F0000000000000000
1200  data  103 ,000000000000000FFFFFFFFFEFEFEFC0000000000000000
1210  data  104 ,0000020000225500000000000000000103051F3F1D2A7F
1220  data  105 ,0040A0000020540000000000000000080040F8FCDCAAFE
1230  data  106 ,0044AA0000225500000000000000007FBB55FFFFDDAA7F
1240  data  107 ,0044AA0000225500000000000000FEBB55FFFFDDAAFE
1250  data  108 ,00442A0000000000000000183C3C1C7F3B557F27030303
1260  data  109 ,0044AA0000000000000000183C3C38FEBA54FEE4C0C0C0
1270  data  110 ,000000000000000C0000000000000030703010000000
1280  data  111 ,00000000000000030000000000000C0E0C080000000000
1290  data  112 ,000000000000000000000000000000FFFFFFFFDB7E3C00
1300 data 300
```

9-4 サルカニウォーズ



ストーリー

昔々，１ピキの老ガニが植物学上，超画期的な生育方法を発見しました。あの筑波の「水耕栽培法」にもひけを取らない「脅迫栽培法」がそれでした。それは植物に水を与えるたびに恐い顔をして，

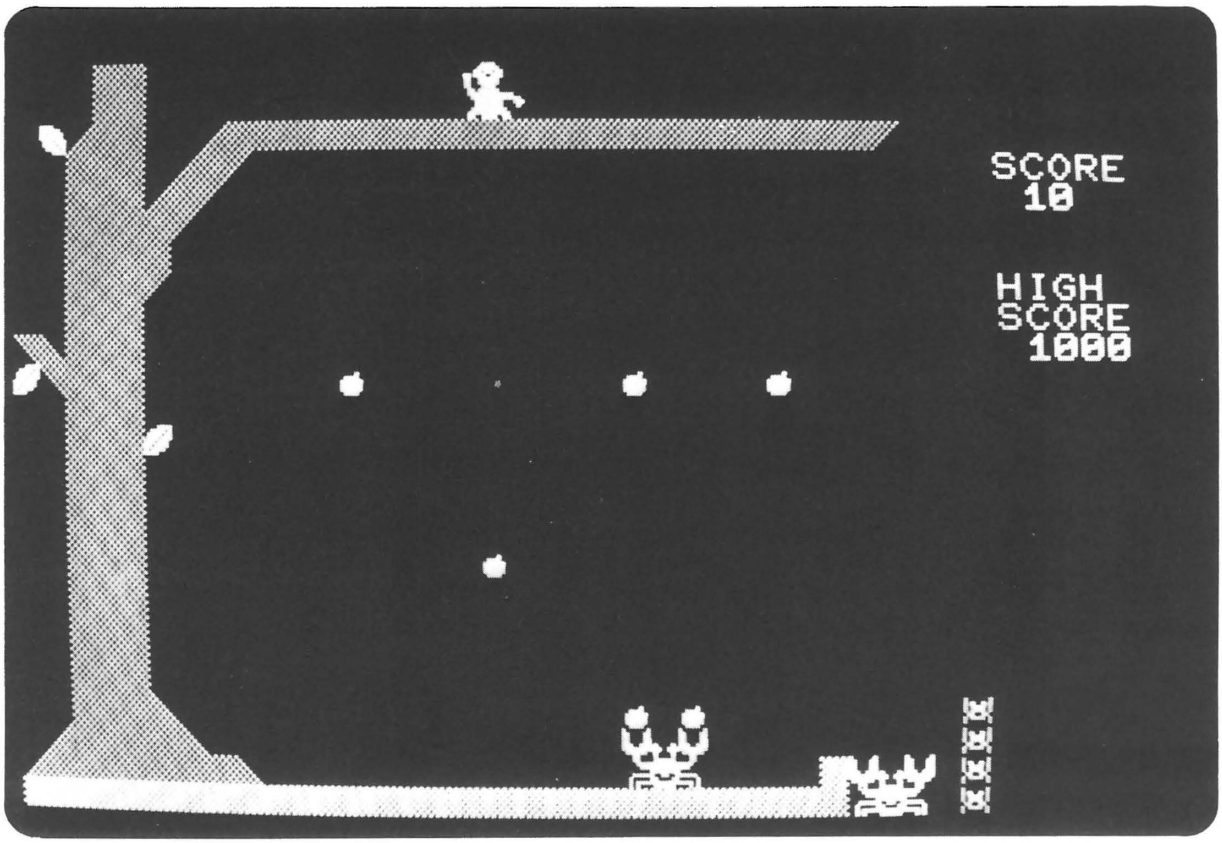
「おいコラァ，はよ大きくならんかい！ならんとハサミでチョン切るでェ」

と関西弁で脅迫するというもので，この方法で育てられたカキは天をつくような巨木に生長しました。

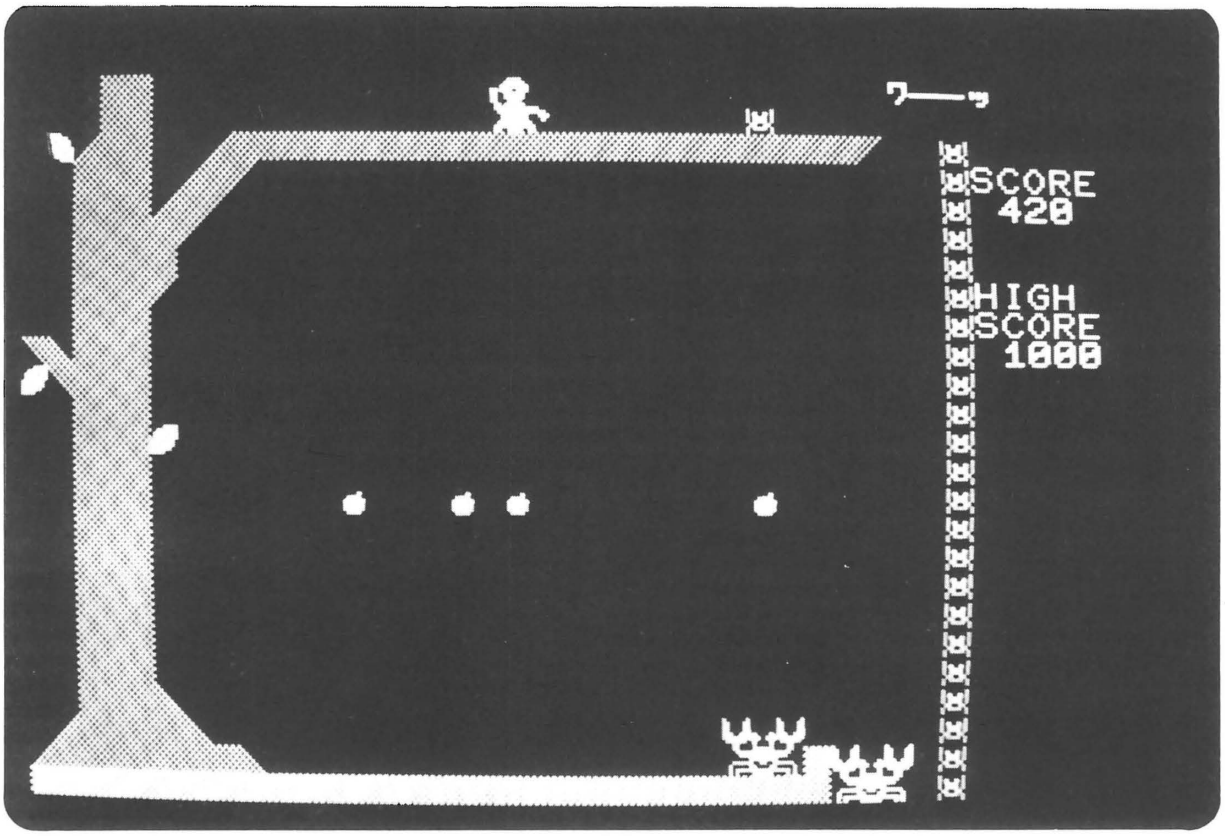
ところが老ガニは彼の成果をねたむ助手の猿に暗殺されてしまい，栽培法の論文は猿の手に渡ってしまいました。もしこの論文が公表されればノーベル賞は確実。賞金で国債を買い込めば利息生活も夢ではないのです。

それを知った息子ガニは彼の妻とともに敢然と猿に戦いをいどんだのでした。

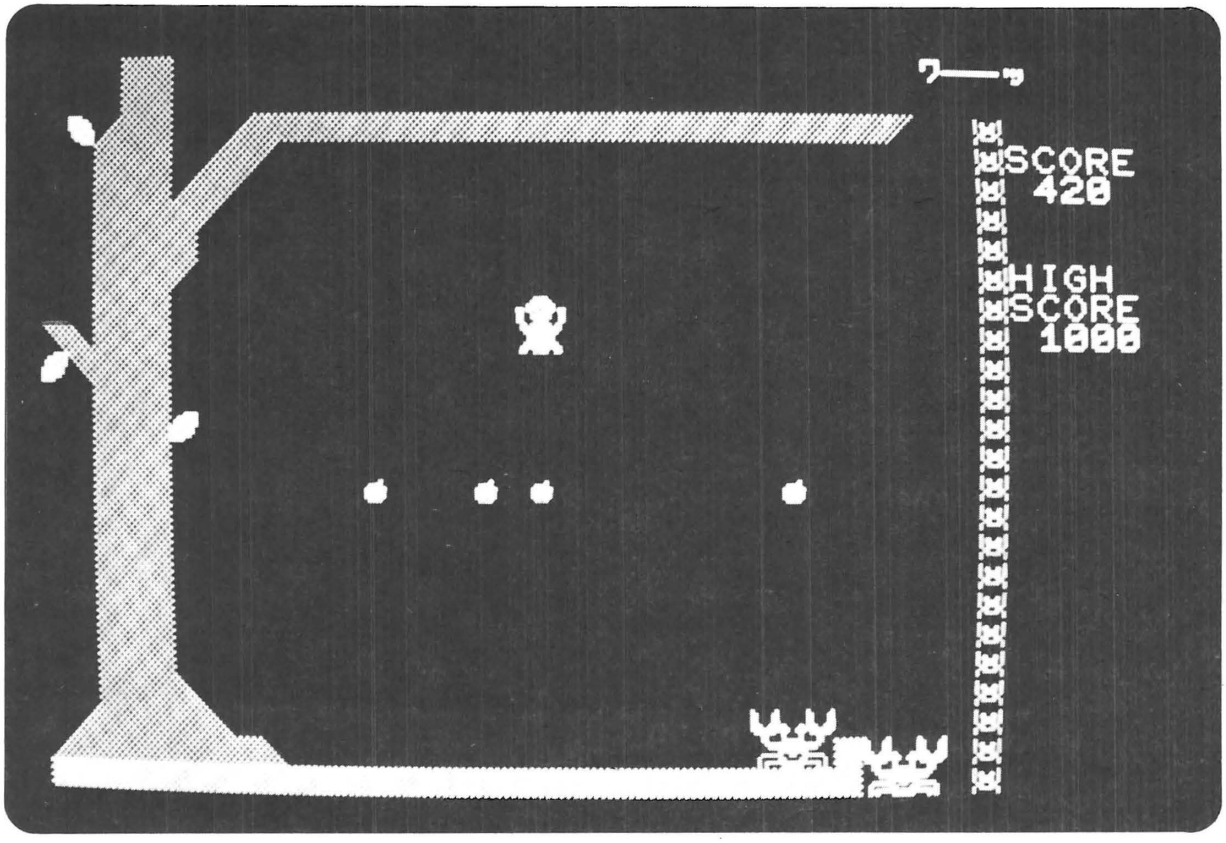




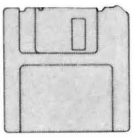
《写真４－１》赤いカキは片手に一つまで。すかさず奥さんにパス



《写真４－２》子ガニが積み上がると，猿にむかって突撃する



《写真4－3》子ガニに足をとられて，猿が木から落ちてくる

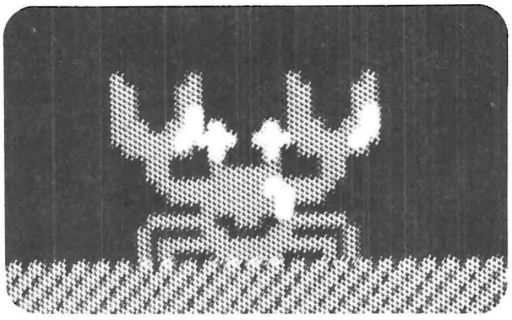


ルール

キャラクタ別に紹介します。

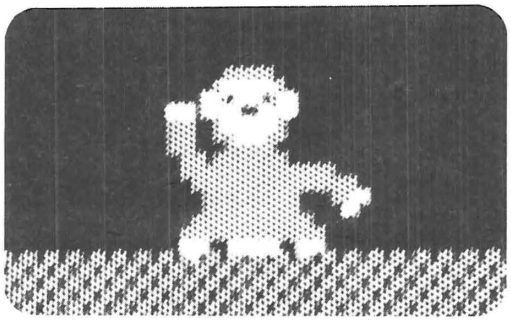
①カニ

4，6のキーで左右に動かし，サルが投げつけてくるカキの中の赤い甘ガキをハサミで受け止め，右下で待っているカニの奥さんに渡さなければなりません。持てるカキは両のハサミに一つずつだけで，それ以上持とうとするとハサミが取れてしまいます。



②猿

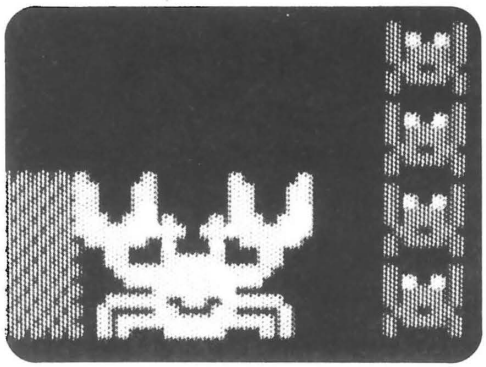
カキの木の上でチョコまかと動きまわっています。枝をゆさぶってシブガキを落とし，自身もカキを投げてきます。カニは直接は猿と戦えませんが，カニの奥さんが生む子ガニを猿のいる枝の高さまで積み上げると，猿に飛びかかり，木から転落させることができます。



③奥さんと子ガニ

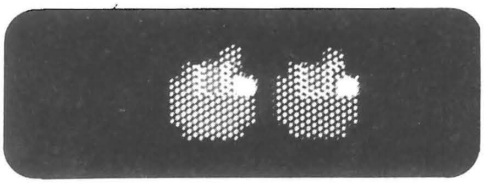
カニの奥さんはカニからカキを受け取るとそれを食べ，１個食べるごとに１ピキの子ガニを産みます。

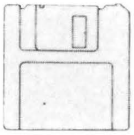
産まれた子ガニはタテに重なり，やがて枝の高さまで積み上がりますが，ある一定の時間で数が減っていき，全滅してしまうとカニはショックでコケてしまいます。



④カキ

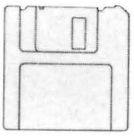
赤いのが甘ガキで，青いのがシブガキです。カニはシブガキに当たると無条件でコケてしまいますし，甘ガキでも頭にくらうとコケてしまいます。





スコア

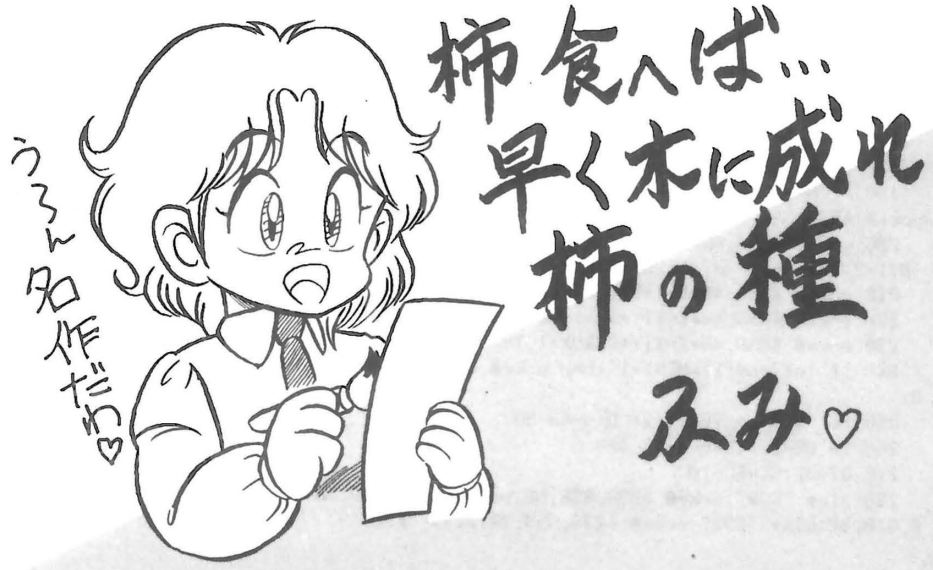
カニが甘ガキをキャッチすると５点，それを奥さんに渡すと10点，そして猿を木から落とすことに成功するとボーナスで500点で加算されます。

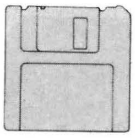


1面クリア&ゲームオーバー

サルを木から落とすと１面クリアになり，より忙しくなる次の面に進みます。

また，カニが３回コケるとゲームオーバーとなります。





プログラム

各行の働きは**第4表**のようになっています。このプログラムもやはりキーリピートを最高速にしているので，BREAK後は[CTRL]－[D]でキャラクタとリピートをノーマルに戻して下さい。

《第4表》プログラム説明

20～250	メインルーチン
270～330	子ガニの表示
340～390	子ガニの突進
	サルが落ちる
400～450	カニがコケる
460	スコアの表示
470～630	初期設定
640～800	デモ
810～	PCGデータおよび設定

《リスト4》サルカニウォーズ・プログラム

```
   10 init "CRT1:40,25,0,0":klist 0:click off:cls 3:repeat on ,4:kmode 0:console 0,25:HS=1000:gosub 810:g
oto 640
   20 if peek@(&H38,881+KANIX)<>66 then goto 400
   30 if (O1=78) and (peek@(&H38,800+KANIX)=78) then H1=32:O1=32:play "C32G32"
   40 if H1=32 then 70
   50 VL=peek@(&H38,840+KANIX):if (O1=32) and (VL=78) then KAKIY=3:O1=78:SC=SC+5:gosub 460
   60 if VL=77 then goto 400
   70 if (O2=78) and (peek@(&H38,802+KANIX)=78) then H2=32:O2=32:play "C32G32"
   80 if H2=32 then 110
   90 VR=peek@(&H38,842+KANIX):if (O2=32) and (VR=78) then KAKIY=3:O2=78:SC=SC+5:gosub 460
  100 if VR=77 then goto 400
  110 XX=KANIX:GE$=inkey$:if GE$="4" then KANIX=KANIX-1:if KANIX<9 then KANIX=9
  120 if GE$="6" then KANIX=KANIX+1:if KANIX>26 then KANIX=26:if (O1=78) or (O2=78) then goto 260
  130 poke@ &H38,840+XX,32,32,32:poke@ &H38,880+XX,32,32,32:poke@ &H38,920+XX,32,32,32
  140 poke@ &H38,840+KANIX,O1,32,O2:poke@ &H38,880+KANIX,H1,66,H2:poke@ &H38,920+KANIX,67,68,69
  150 RN=0:if (int(rnd(1)*7)=0) and (KAKIY<18) then RN=-1+int(3*rnd(1))
  160 MM=MM*-1:XX=SARUX:SARUX=SARUX+RN:if SARUX<9 then SARUX=8
  170 if SARUX>26 then SARUX=25
  180 poke@ &H38,XX,32,32,32:poke@ &H38,XX+40,32,32,32
  190 if MM=1 then poke@ &H38,SARUX,97,98:poke@ &H38,SARUX+40,99,100,101: else poke@ &H38,SARUX+1,98,102:
poke@ &H38,SARUX+40,103,100,104
  200 poke@ &H38,40*BY1+BX1,32:poke@ &H38,40*BY1+BX2,32:poke@ &H38,40*BY1+BX3,32:BY1=BY1+1:if BY1=24 then
 BY1=3:BX1=int(rnd(1)*20)+9:BX2=int(rnd(1)*20)+9
  210 poke@ &H38,40*BY1+BX1,77:poke@ &H38,40*BY1+BX2,77:poke@ &H38,40*BY1+BX3,77
  220 poke@ &H38,40*KAKIY+XX+1,32:KAKIY=KAKIY+1:if KAKIY=24 then KAKIY=3
  230 poke@ &H38,40*KAKIY+SARUX+1,78
  240 if int(rnd(1)*MEN)=1 then poke@ &H38,994-40*EN,32:EN=EN-1:play "G32C32","E32":if EN=-1 then goto 40
0
  250 for D=0 to 750:next D:goto 20
  260 if O2=32 then goto 320
  270 O2=32:SC=SC+10
  280 play "G32":poke@ &H38,868,32:poke@ &H38,909,78:play "G32":poke@ &H38,909,32,78:play "G32":poke@ &H3
8,910,32:play "F32":poke@ &H38,993,79:play "F32"
```

```
 290 poke@ &H38,993,32:EN=EN+1:poke@ &H38,994-40*EN,79
 300 gosub 460:if EN=23 then goto 340
 310 if O1=32 then goto 130
 320 O1=32:SC=SC+10
 330 play "G32":poke@ &H38,866,32:poke@ &H38,868,78:goto 280
 340 cgen 0:locate 32,0:print "ﾜｰｰｯ":cgen 1
 350 for D=33 to SARUX+1 step -1:locate D,1:print "O ":play "C32":next D
 360 for D=0 to 21:poke@ &H38,80+SARUX,83,83,83:locate SARUX,D:print "   ****abf****cdh**":for DD=0 to 2
00:next DD:next D
 370 play "t230ecececgggfdfdecc"
 380 cgen 0:locate 13,8:print "ｻﾙﾓ ｷｶﾗ ｵﾁﾙ":locate 14,10:print "BONUS 500":SC=SC+500:gosub 460
 390 play "r32":goto 480
 400 for D=0 to 3:poke@ &H38,881+KANIX,75:play "F16"
 410 poke@ &H38,881+KANIX,76:play "B16":next D:for D=0 to 5000:next D
 420 KANI=KANI-1:if KANI>0 then goto 490
 430 if SC>HS then HS=SC
 440 cgen 0:locate 15,8:print "G A M E":locate 15,10:print "O V E R"
 450 for D=0 to 10000:next D:goto 640
 460 cgen 0:locate 35,4:print str$(SC):cgen 1:return
 470 SC=0:KANI=3:MEN=90
 480 MEN=MEN-10:if MEN<50 then MEN=50
 490 cgen 1:width 40:EN=3
 500 play "T200 O4E8E8E8E8C8C8E8F8E8F8G8F8R8G8F8E8E8E8C8O3B8B8O4C8D8C2O5C1"
 510 print "***SS***SS*****QWSS":locate 0,9:print "UV***PU"
 520 locate 5,12:print "P"
 530 for D=3 to 23:locate 2,D:print "SSS";:next D
 540 locate 5,21:print "V**SV***SSSV";:locate 1,22:print "W***WS";
 550 locate 0,24:print "RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRHIJ*****RFGF**"
 560 locate 5,7:print "T**S**ST***WST***WST***WSSSSSSSSSSSSSSSSSSSSSST"
 570 locate 26,22:print "HBA****CDE**"
 580 locate 34,24:print "O**O**O**O":locate 14,0:print "ab***cde"
 590 cgen 0:color 4:locate 35,3:print "SCORE":locate 35,7:print "HIGH*****SCORE"
 600 color 7:locate 35,4:print str$(SC):locate 35,9:print str$(HS)
 610 BY1=3:BX1=int(rnd(1)*20)+9:BX2=int(rnd(1)*20)+9:BX3=int(rnd(1)*20)+9
 620 H1=65:H2=65:SARUX=14:O1=32:O2=32:KAKIY=3:KANIX=26:MM=1
 630 cgen 1:goto 20
 640 width 40:print:cgen 1
 650 print "    bbbbb  bbb  bbbbb b   b"
 660 print "    b   b b   b b   b b   b"
 670 print "    bb    bbbbb b  b  b   b"
 680 print "      bbb b   b bbbb  b   b"
 690 print "        b b   b b   b b   b"
 700 print "    bbbbb b   b b   b bbbbb**"
 710 print "          0   0  000  00  0  000 "
 720 print "          0  0  0   0 00  0   0  "
 730 print "          0 0   00000 0 0 0   0  "
 740 print "          00    0   0 0 0 0   0  "
 750 print "          0 00  0   0 0  00   0  "
 760 print "          0   0 0   0 0   0  000 ";:cgen 0:print "WARS"
 770 print "***          YOUR KEY  <4> & <6>**"
 780 print "            START <SPACE> KEY"
 790 GT$=inkey$:if GT$=" " then goto 470
 800 goto 790
 810 restore 840
 820 read A:if A=255 then return
 830 read B$:def chr$(A)=hexchr$(B$):goto 820
 840  data  65 ,0000000101020000081C3C3C3E7FF7E180000000101020000
 850  data  66 ,00000000000000000000000042E742420000000042E74200
 860  data  67 ,0000000000000000180F07001F202F280000000000000000
 870  data  68 ,0002010100000000FEFFFFFF99C3FF7E0002010100000000
 880  data  69 ,00000000000000018F0E000F804F41400000000000000000
 890  data  70 ,81C3C3C3E7FF7E1881C3C3C3E7FF7E1800000000101020000
 900  data  71 ,00000000000000420000000042E742420000000042E74200
 910  data  72 ,180F07001F202F28180F07001F202F280000000000000000
 920  data  73 ,7EFFFFFF99C3FF7E7EFFFFFF99C3FF7E0002010100000000
 930  data  74 ,18F0E000F804F41418F0E000F804F41400000000000000000
 940  data  75 ,00000000000000000000000000E0E2E242000000000E0E2E200
 950  data  76 ,00000000000000000000000007474742000000000007474700
 960  data  77 ,000004181A000000000000000200000000000024667E7E3C
```

```
 970  data  78 ,0000000002000000000000024667E7E3C000004181A000000
 980  data  79 ,000000000000000000081A5BDBD66BD8100002400000000000
 990  data  80 ,00000000000000000000000000000000000F3D7B77EEDEBCF0
1000  data  81 ,000000000000000000000000000000000F0BCDEEE777B3D0F
1010  data  82 ,AA55AA55AA55AA55000000000000000000000000000000000
1020  data  83 ,0000000000000000AA55AA55AA55AA55000000000000000000
1030  data  84 ,0000000000000000AA54A850A040800000000000000000000
1040  data  85 ,0000000000000000AA552A150A050201000000000000000000
1050  data  86 ,00000000000000008040A050A854AA55000000000000000000
1060  data  87 ,000000000000000000010205 0A152A55000000000000000000
1070  data  97 ,0000000000000000000000000606060600000000060000000
1080  data  98 ,000000000000000003C7EDBE77E7E3C00003C99A53C1800
1090  data  99 ,00000000000000000303010000010303000000000000000003
1100  data  100 ,000000000000003C7EFFFFFFFFFFFF3C0000000000000000
1110  data  101 ,0000000000000000000E0F81C0880C0C0000000040800000C0
1120  data  102 ,000000000000000000000000060606060000000006000000
1130  data  103 ,00000000000000000000071F3810010303000000201000003
1140  data  104 ,000000000000000C0C080000080C0C0000000000000000C0
1150  data  32 ,00000000000000000000000000000000000000000000000000
1160 data 255
```

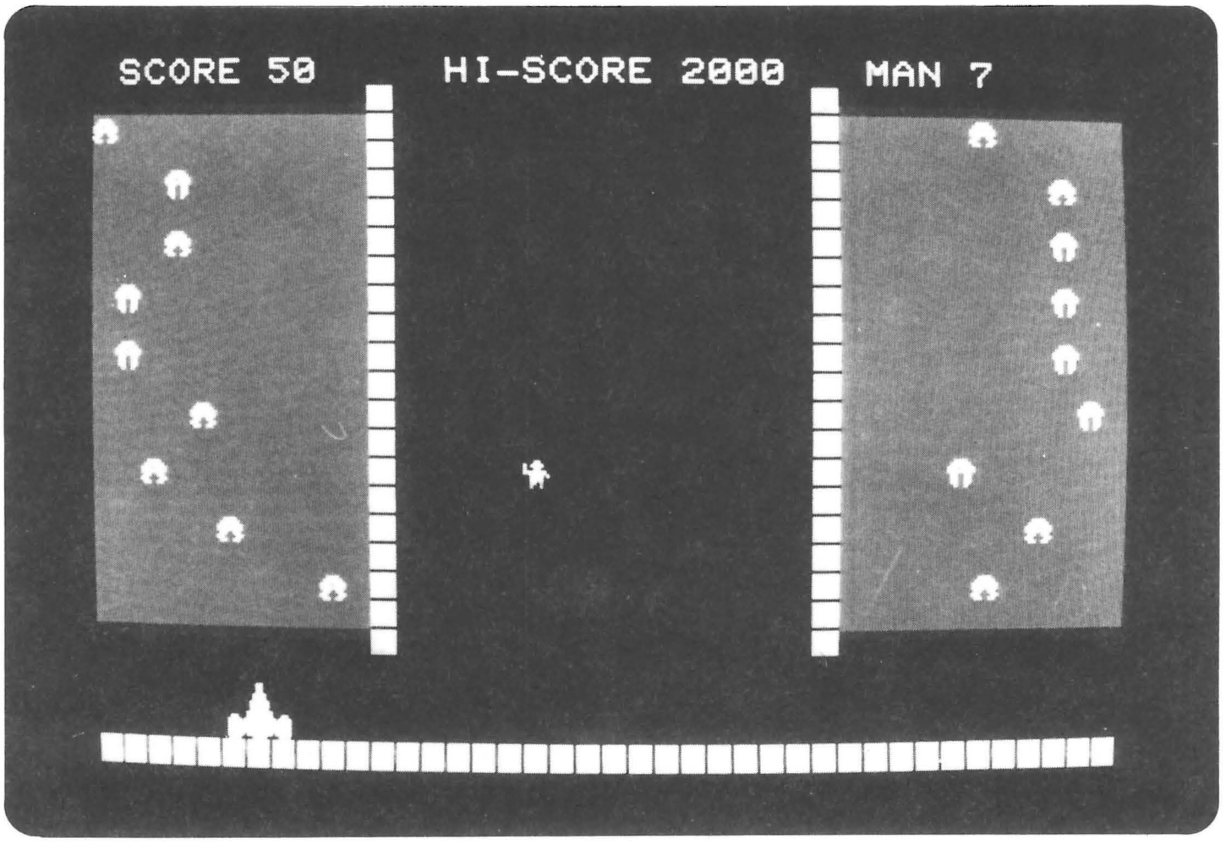
9-5 金門橋エイリアン

ストーリー

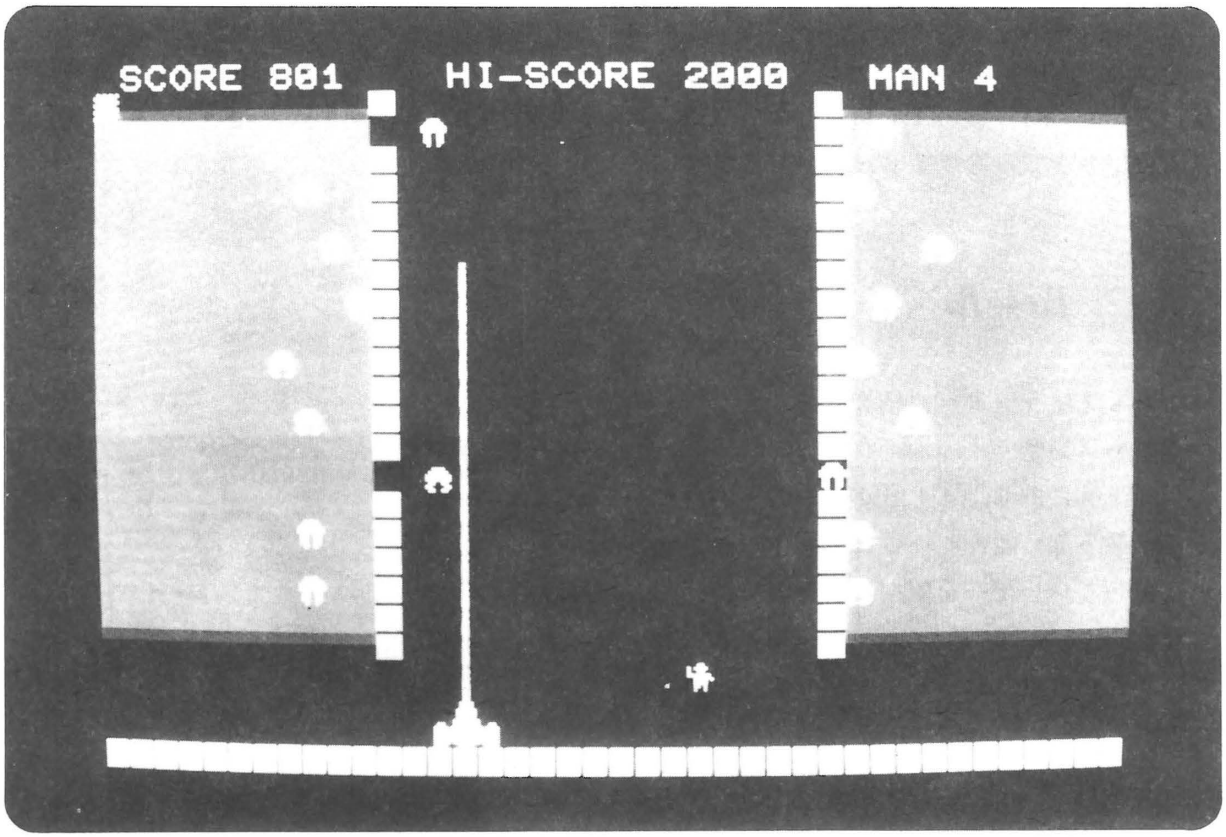
西暦1985年9月。多部田俊雄は白紙の原稿用紙を見つめながら考えていた。

「うーん，そろそろ啓蒙的でない昔話を考えるのにもあきたな。うん。だいたいアクションゲームのストーリーって何でこう取ってつけたようなものしかないんだろ。『あなたは××の平和を守るため○○と戦わなければならないのだ！』—こんなのばっかりだもんね。よく，『ストーリーがあると感情移入がしやすくなる』と言うけど，あれ本当かね。ありきたりのストーリーを読まされると，かえってやる気がなくなったりして。つまんないよな，本当。」

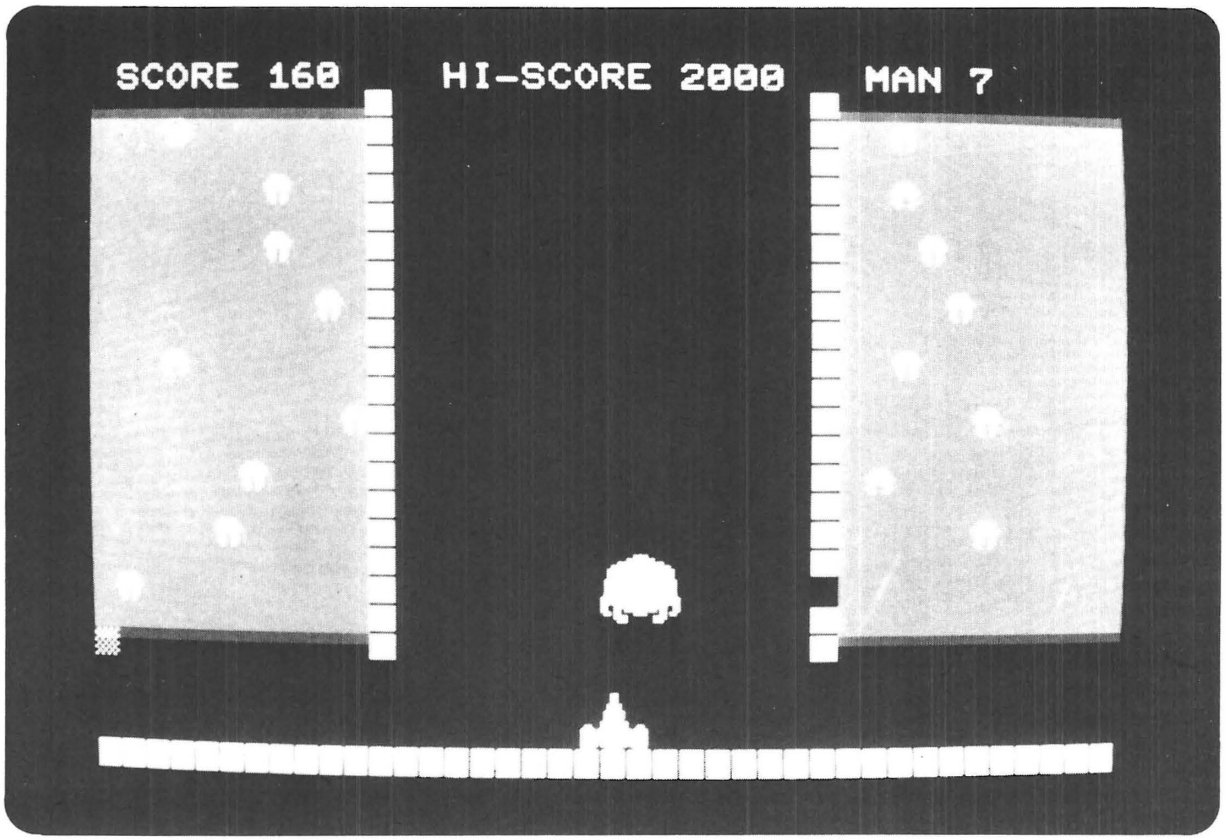




《写真５－１》橋の左右から迫ってくるガ・オーク。中央にいるのが人

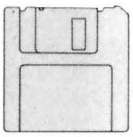


《写真5－2》輝きながら走るビーム。橋の上ならガ・オークを殺せる



《写真5－3》ガ・オークがガラサに変態。人が食べられてしまう

そう言ったわけでこのゲームにはストーリーはありません。どうしてもストーリーが欲しい方は，ゲームを行ったあとで自作して下さい。なお，これは決して，本当に，絶対に，完全無欠に著者の手ヌキではありません。信じて！

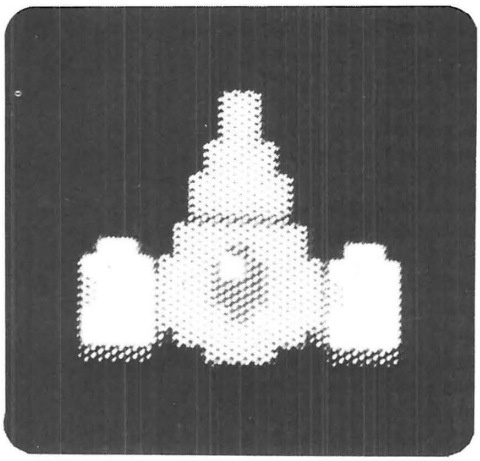


ルール

これもキャラクタ別に紹介します。

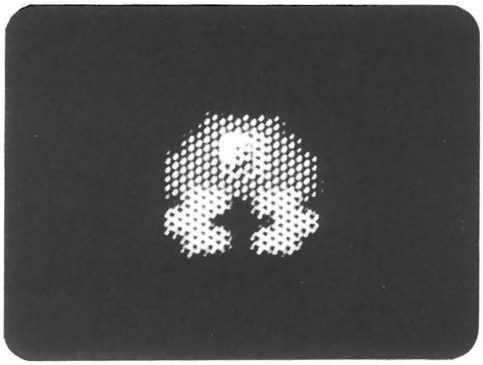
①キャノン砲

画面下部にあり，4と6のキーで左右に動かします。そして最下部に並んでいるエネルギー弾を使ってスペースキーでレーザーを撃ち，橋を渡って行く人の援護射撃をするのですが，エネルギー弾が0になったり，人がガラサに喰われてしまったりすると自爆してしまいます。



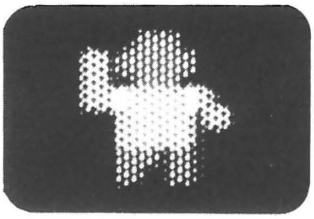
②ガ・オーク

ガラサの幼虫で左右の川から橋を渡る人をねらって徐々に近寄って来ます。水中にいる時はレーザーで撃ってもスタート地点まで後退するだけで，殺すことはできません。上陸すると殺すことができるようになるのですが，ガ・オークも狂暴になり，真横に人が来るといきなりガラサに変身し，飛びかかって喰いついてしまいます。



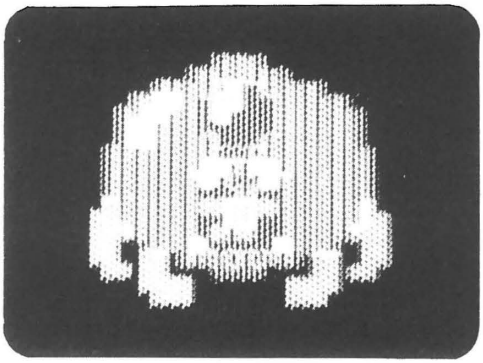
③人

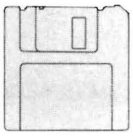
この非常時に橋を渡って行くという非常識な人間。



④ガラサ

ガ・オークの成体。人を喰うとその場で消滅してしまいます。これが本当の喰い逃げだという話も……。





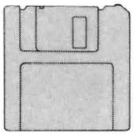
スコア

第５－１表のようになっています。

《第５－１表》スコア

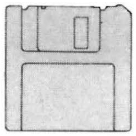
上陸前のガ・オーク	5
上陸後のガ・オーク	20
１面クリア時の残り弾一つにつき	2
１人渡らせるごとに	50
人を殺す	−100





1面クリア&ゲームオーバー

ガ・オークを全て倒すと1面クリアになり，次の面ではさらに足の速い敵が出現します。
また人が全滅するとゲームオーバーとなります。



プログラム

各行の働きは**第5－2表**のようになっています。これもやはりテストランの後は[CTRL]－[D]でキャラクタとリピートスピードをノーマルに戻して下さい。

《表5－2表》プログラム説明

10～20	ハード的初期設定
30～200	メインルーチン
210～330	ビーム発射スコアアップ
340～400	ガラサ出現
410～	人の数表示
420～	スコアの表示
430～460	ゲームオーバー
470～500	1面クリア
570～670	初期設定
680～810	デモ
820～1120	ＰＣＧ設定とデータ

《リスト5》金門橋エイリアン・プログラム

```
   10 init "CRT2:320,200,16"
   20 init "CRT1:40,25,0,0":klist 0:click off:cls 3:repeat on ,4:kmode 0:console 0,25:HS=2000:goto 820
   30 for D=0 to RERU:MO=int(rnd(1)*9):if RE(MO)=1 then goto 40: else next D:goto 70
   40 poke@ &H38,40*RY(MO)+RX(MO),32:RX(MO)=RX(MO)-1:if RX(MO)<26 then RX(MO)=26
   50 poke@ &H38,40*RY(MO)+RX(MO),74+int(rnd(1)*2)
   60 if MY=RY(MO) and RX(MO)<29 then play "G24":poke@ &H38,MY*40+RX(MO)-1,74,32:play "G24":poke@ &H38,MY
*40+RX(MO)-2,75,32:play "03C8O4":TT=0:goto 340
   70 for D=0 to RERU:MO=int(rnd(1)*9):if LE(MO)=1 then goto 80: else next D:goto 110
   80 poke@ &H38,40*LY(MO)+LX(MO),32:LX(MO)=LX(MO)+1:if LX(MO)>13 then LX(MO)=13
   90 poke@ &H38,40*LY(MO)+LX(MO),74+int(rnd(1)*2)
  100 if MY=LY(MO) and LX(MO)>10 then play "G24":poke@ &H38,MY*40+LX(MO),32,74:play "G24":poke@ &H38,MY*4
0+LX(MO)+1,32,75:play "03C8O4":TT=1:goto 340
  110 MRUN=MRUN*-1:MC=72:if MRUN=-1 then MC=73
  120 poke@ &H38,40*MY+MX,32:MY=MY-1:if MY=0 then SC=SC+50:gosub 420:MY=21:MX=15+int(rnd(1)*10)
  130 poke@ &H38,40*MY+MX,MC:LL=0
  140 for D=0 to 400:next D:GE$=inkey$
  150 XX=CX:if GE$="4" then CX=CX-1:if CX<0 then CX=0
  160 if GE$="6" then CX=CX+1:if CX>37 then CX=37
  170 if GE$=" " then goto 210
  180 poke@ &H38,XX+881,32:poke@ &H38,XX+920,32,32,32
  190 poke@ &H38,CX+881,67:poke@ &H38,CX+920,68,69,70
  200 if LL=0 then for D=0 to 30:next D:LL=1:goto 140: else goto 30
  210 if CA=-1 then goto 430
  220 poke@ &H38,960+CA,91:CA=CA-1
  230 for B=21 to 1 step -1:CC=peek@(&H38,CX+1+40*B):poke@ &H38,CX+1+40*B,66:if CC=32 then next B:B=1
  240 for C=21 to B step -1:poke@ &H38,CX+1+40*C,32:next C
  250 if CC=72 or CC=73 then play "O5D32O4":ME=ME-1:gosub 410:SC=SC-100:gosub 420:if ME=0 then goto 430:
else MY=21:MX=15+int(rnd(1)*10):goto 30
  260 if CC<74 then goto 30
```

```
 270 MO=(C+1)/2-1:if CX>20 then goto 310
 280 play "B32":if LX(MO)<11 then SC=SC+5:gosub 420:LX(MO)=0:goto 30
 290 SC=SC+20:gosub 420:LE(MO)=0:AL=AL-1:if AL=0 then goto 470
 300 goto 30
 310 play "B32":if RX(MO)>28 then SC=SC+5:gosub 420:RX(MO)=39:goto 30
 320 SC=SC+20:gosub 420:RE(MO)=0:AL=AL-1:if AL=0 then goto 470
 330 goto 30
 340 if TT=0 then poke@ &H38,MY*40+RX(MO)-2,32: else poke@ &H38,MY*40+LX(MO)+2,32
 350 locate MX-1,MY-1:print "RST****UVW****XYZ":for D=0 to 1000:next D
 360 locate MX-1,MY-1:print "   ****   ****   "
 370 ME=ME-1:gosub 410:if ME=0 goto 430
 380 MO=MY/2-1:MY=21:MX=int(rnd(1)*10)+15:if TT=0 then goto 400
 390 LE(MO)=0:AL=AL-1:if AL=0 then goto 470: else goto 30
 400 RE(MO)=0:AL=AL-1:if AL=0 then goto 470: else goto 30
 410 cgen 0:locate 33,0:print ME:cgen 1:return
 420 cgen 0:locate 6,0:print SC:cgen 1:return
 430 locate CX+1,22:print " "
 440 for D=0 to 5:locate CX,23:print "LMN**":play "O3G32":locate CX,23:print "OPQ**":play "O3D32":next D
:play "O4"
 450 cgen 0:locate 16,7:print "G A M E**********O V E R":if SC>HS then HS=SC
 460 for D=0 to 7000:next D:goto 680
 470 if CA>-1 then for E=CA to 0 step -1:poke@ &H38,960+E,91:play "A32":SC=SC+2:gosub 420:next E
 480 locate 15,9:cgen 0:print "NEXT PLAY!":cgen 1
 490 for D=0 to 4000:next D
 500 ME=ME+1:goto 520
 510 SC=0:ME=7:RERU=0
 520 cgen 1:width 40:cgen 0:locate 1,0:print "SCORE";SC
 530 locate 14,0:print "HI-SCORE";HS
 540 locate 30,0:print "MAN";ME:cgen 1
 550 RERU=RERU+1:if RERU>4 then RERU=4
 560 for D=0 to 19:poke@ &H38,51+40*D,65:poke@ &H38,68+40*D,65:next D
 570 line (0,16)-(87,159),1,BF:line (232,16)-(319,159),1,BF
 580 line (0,13)-(87,15),2,BF:line (232,13)-(319,15),2,BF
 590 line (0,160)-(87,162),2,BF:line (232,160)-(319,162),2,BF
 600 for D=0 to 8:RX(D)=39:RY(D)=2+D*2:RE(D)=1:poke@ &H38,(2+D*2)*40+39,74:next D
 610 for D=0 to 8:LX(D)=0:LY(D)=2+D*2:LE(D)=1:poke@ &H38,(2+D*2)*40,74:next D
 620 CX=18:MY=21:MX=int(rnd(1)*10)+15:MRUN=1:AL=18
 630 for D=0 to 39:poke@ &H38,960+D,71:next D:CA=39
 640 locate 19,22:print "C***DEF"
 650 locate 17,9:cgen 0:print "START!":cgen 1:for D=0 to 4000:next D
 660 locate 17,9:print "      "
 670 goto 30
 680 cgen 1:width 40:cls 2
 690 for D=0 to 39:poke@ &H38,40+D,65:poke@ &H38,80+D,65:poke@ &H38,120+D,65:next D
 700 locate 8,2:cgen 0:print "GOLDEN GATE BRIDGE ALIEN":goto 730
 710 for D=0 to 1200:next D:cgen 1:return
 720 for D=0 to 2000:next D:cgen 0:return
 730 locate 13,7:print "CANNON.....";:gosub 710:print "DEF***C":gosub 720
 740 locate 13,10:print "GARASA.....";:gosub 710:print "*RST****UVW****XYZ":gosub 720
 750 locate 13,13:print "GA-ORC......";:gosub 710:print "J":gosub 720
 760 locate 13,16:print "MAN    ......";:gosub 710:print "H":gosub 720
 770 locate 15,19:cgen 0:print "YOUR KEYS"
 780 locate 11,21:print "RIGHT <6>  LEFT <4>"
 790 locate 10,23:print "FIRE & START <SPACE>"
 800 MH$=inkey$:if MH$=" " then 510
 810 goto 800
 820 read A:if A=300 then 680
 830 read B$:def chr$(A)=hexchr$(B$):goto 820
 840  data  32 ,00000000000000000000000000000000000000000000000000000000
 850  data  65 ,00000000000000000FFFFFFFFFFFFFFF00FFFFFFFFFFFFFF00
 860  data  66 ,18001800180018180018180000181818000000181818181818
 870  data  67 ,1818183C3C7E7E00000000000000000000181818 3C3C7E7E7E
 880  data  68 ,00183D3D3D3D3C0000182C1C1C3C3C3C00183D3F3F3F3E00
 890  data  69 ,FFFFFFFFFFFFFF3C000010000000000FFE7D3C3C3E7FF3C
 900  data  70 ,0018BCBCBCBC3C0000182C1C1C3C3C3C0018BCFCFCFC7C00
 910  data  71 ,FEC682828282C6FE00387C7C7C7C380000386C5C5C7C3800
 920  data  72 ,183C183C3C3C0000008080C2010000000008080FE3D3C3424
 930  data  73 ,183C183C3C3C000000010143800000000001017FBC3C2C24
 940  data  74 ,0000100000000000000018180066C366003C66E7FF66C366
```

```
 950  data  75 ,0010000000000000000181800666666243C66E7FF66666624
 960  data  76 ,0000000000000000000421050F2F7F7F0000000000000000
 970  data  77 ,00000000000000002288C2F7FFFFFFFF0000000000000000
 980  data  78 ,0000000000000000002084A0F0F4FEFE0000000000000000
 990  data  79 ,0000000000000000092100111B1F3F7F0000000000000000
1000  data  80 ,000000000000000041143EFFFFFFFF0000000000000000
1010  data  81 ,0000000000000000908400088D8F8FCFE0000000000000000
1020  data  82 ,00000005171D1D1D0000000060C1818000000020E2E3A3A
1030  data  83 ,0014555131210141000002C4E5E7E3C0028AA92B0A08082
1040  data  84 ,000000405054545400000000000000000000000A0A8A8A8A8
1050  data  85 ,555555551515150500000000040C0E0C02A2A2A2A6AEAEACA
1060  data  86 ,456F5E0818180081007 6E3F7E7E7FFFFBAE642007E182881
1070  data  87 ,5454545454545050000000000020307 03AAAAAAAAAAABAFA3
1080  data  88 ,0100000000000000CC6C0F070000000CC6C0F0700000000
1090  data  89 ,55000000000000000000000000000000AA3C000000000000
1100  data  90 ,0000000000000003336F0E000000000B336F0E000000000
1110  data  91 ,FEFEFEFEFEFEFEFE0000000000000000000000000000000000
1120 data 300
```

BASIC-M25の隠し命令 5

●倍速スムーススクロール

これはBASIC等の隠し機能と言うよりもBASIC等の改造情報と言った方が良いと思います。

POKE　&H2C4,　&H1E,　&H2,　&HCD,　&H52,　&H4C,　&H15,　&H20,　&HFA,　&HC9：PO
KE　&H1FB,　&HC4,　&H2

としますと，スムーススクロールの速度が2倍になります。通常の状態ではブラウン管に1画面分の画像が映し出される約$\frac{1}{60}$秒間に1度，次の1画面を表示するための準備として，画面が一時的に消える期間(垂直ブランキング期間)があるのですが，その時にCPUに割込みをかけ，1文字の縦方向$\frac{1}{8}$ずつ(200ラインディスプレイ時は1ドットずつ，400ラインの8×16ドットフォント利用時は2ドットずつ)スクロールアップさせるという手法により，全く乱れのないスクロールを実現しています。このことから1行分のスクロールアップには原理上，常に約$\frac{8}{60}$秒(0.133…秒)かかることがわかりますが，さきほどのちょっとしたプログラムでは，1回当たりのスクロール量を縦方向$\frac{2}{8}$($\frac{1}{4}$)文字分にすることで，スクロール速度を倍(1行につき$\frac{4}{60}$秒)にしているのです。

なお，この状態からさらに

POKE　&H2C5,　&H4

としますと，4倍速スクロールも可能です。ただ，ここまで速くしますと，画面に追いついてゆくのがかなりきつくなり，目が疲れ易くなります。それに，モニタのDコマンドやFILES命令のように元々スピーディに表示が行われるものでは問題ありませんが，LIST命令の際には使われている予約語や実数の定数の数が多い行(一行が長い行)で中間コード等→文字列変換にかかる時間が長くなり，スクロールがガタついてかえって見づらくなることがあります。また，同様に表示のための前処理が長いものでは，やはり乱れが生じることがありますので，2倍速くらいが頃度良いかと思います。

なお，

POKE　&H2C5,　&H1

とすると通常の1倍速の状態にすることが可能です。

また，

POKE　&H1FB,　&H52,　&H4C

として，完全に元の状態に戻すこともできます。

●ラインスクロールモードで起動する

これも前項と似たようなものです。ただし，改造ではありません。BASIC起動時の画面はスムーススクロールモードになっていますが，auto-run.m25に，

980　POKE　&H5ED,　0：POKE　&H5E9,　&H18：KLIST　2：PRINT　CHR$(4)；

を追加すると，ラインスクロールモードで起動できます(行番号は211～989の中で空いている所ならどこでもOKです)。INIT文を使った場合と異なり，初期メッセージを消さずにこのモードにできますので，大変便利です。

付録

EXPRESSION OF SENSIBILITY & COMMUNICATION

SuperMZ

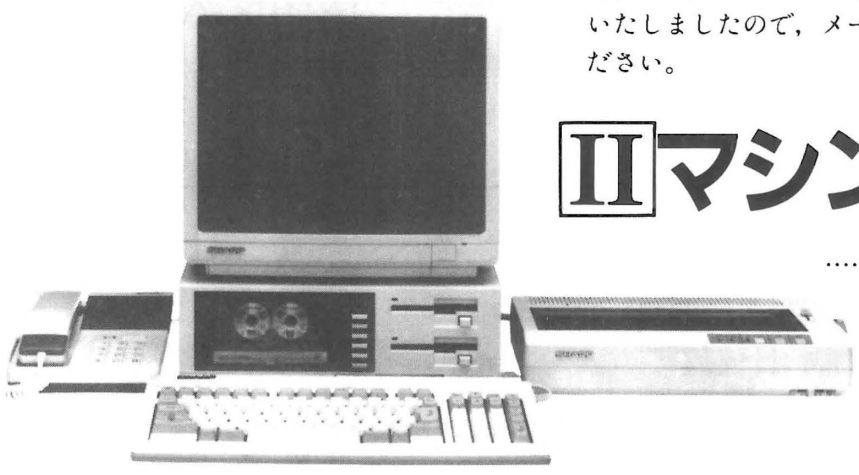
Ⅰ 全回路図

① CPU・メモリコントロール部 …… 296
② IPL・BIOS・拡張ROM・メインRAM部 …… 297
③ グラフィック部 …… 298
④ テキストV-RAM・キャラクタジェネレータ部 …… 299
⑤ CRTコントロール部 …… 300
⑥ I/O・割込みコントロール部 …… 301
⑦ RS-232C・マウス・キーボード制御部 …… 302
⑧ カセット及び音声合成インターフェース・サウンド部 …… 303
⑨ アナログ・デジタルディスプレイ切換え部 …… 304
⑩ カレンダクロック・リモート・各種スロット部 …… 305
⑪ キーボード・マウス端子(前面)部 …… 306
⑫ FDC(フロッピーディスクコントローラ)部 …… 307
⑬ ジョイスティックインターフェース部 …… 308
⑭ 拡張RAMボード(MZ-1R26) …… 309
⑮ 辞書ROMボード(MZ-1R28) …… 310
⑯ パレットボード(MZ-1M10)パレットレジスタ部 …… 311
⑰ パレットボード(MZ-1M10) D/A変換部 …… 312
⑱ 拡張グラフィックRAMボード(MZ-1R27) …… 313
⑲ カセットデッキワンチップマイコン部 …… 314
⑳ カセットデッキアンプ部 …… 315
㉑ 電源部 …… 316
㉒ ボイスボード(MZ-1M08) …… 317
㉓ キーボード部 …… 318

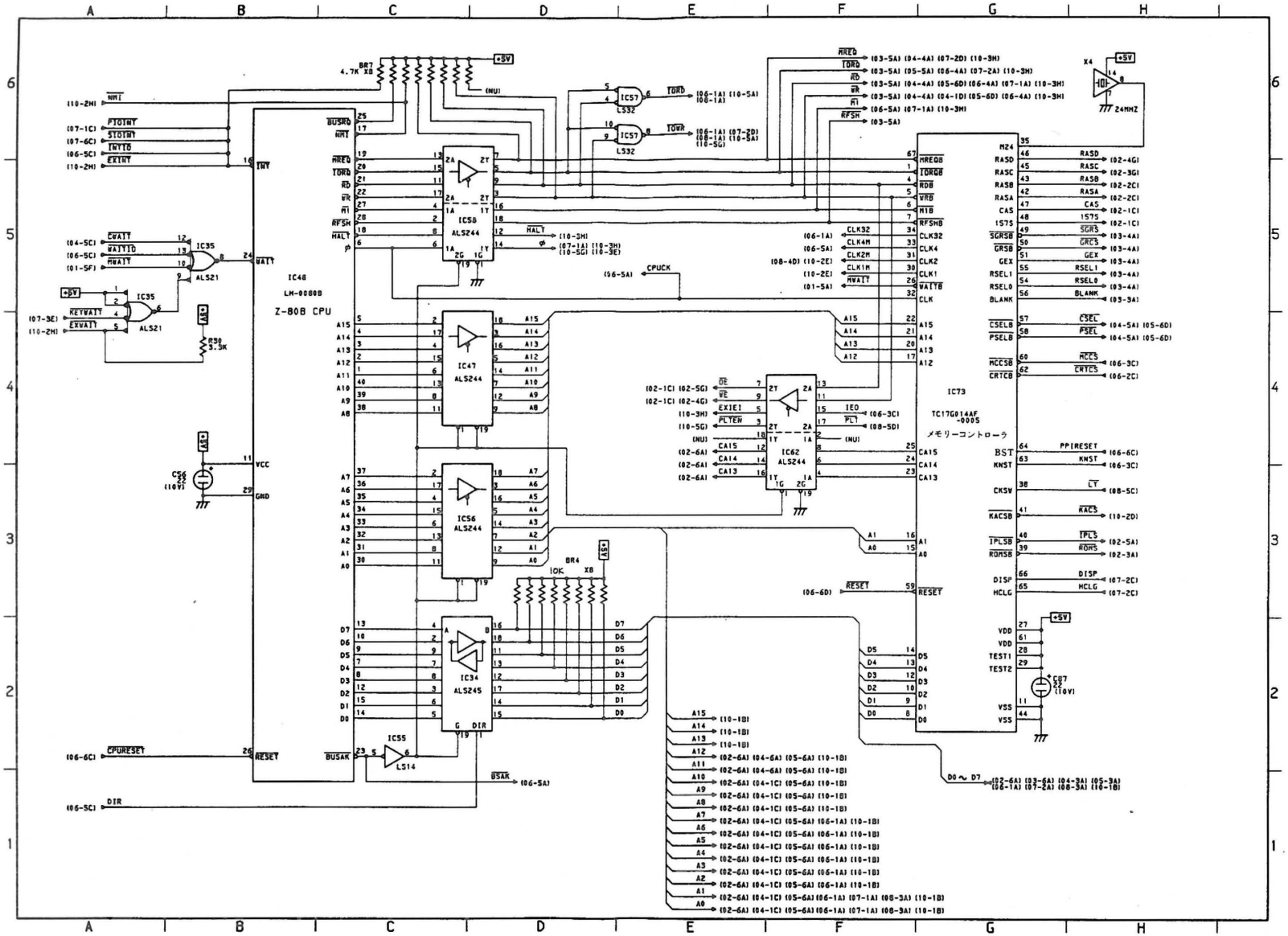
※本回路図はマイコン編集部において参考資料として掲載いたしましたので，メーカーへのお問合わせはご遠慮ください。

Ⅱ マシン語活用表

…… 319



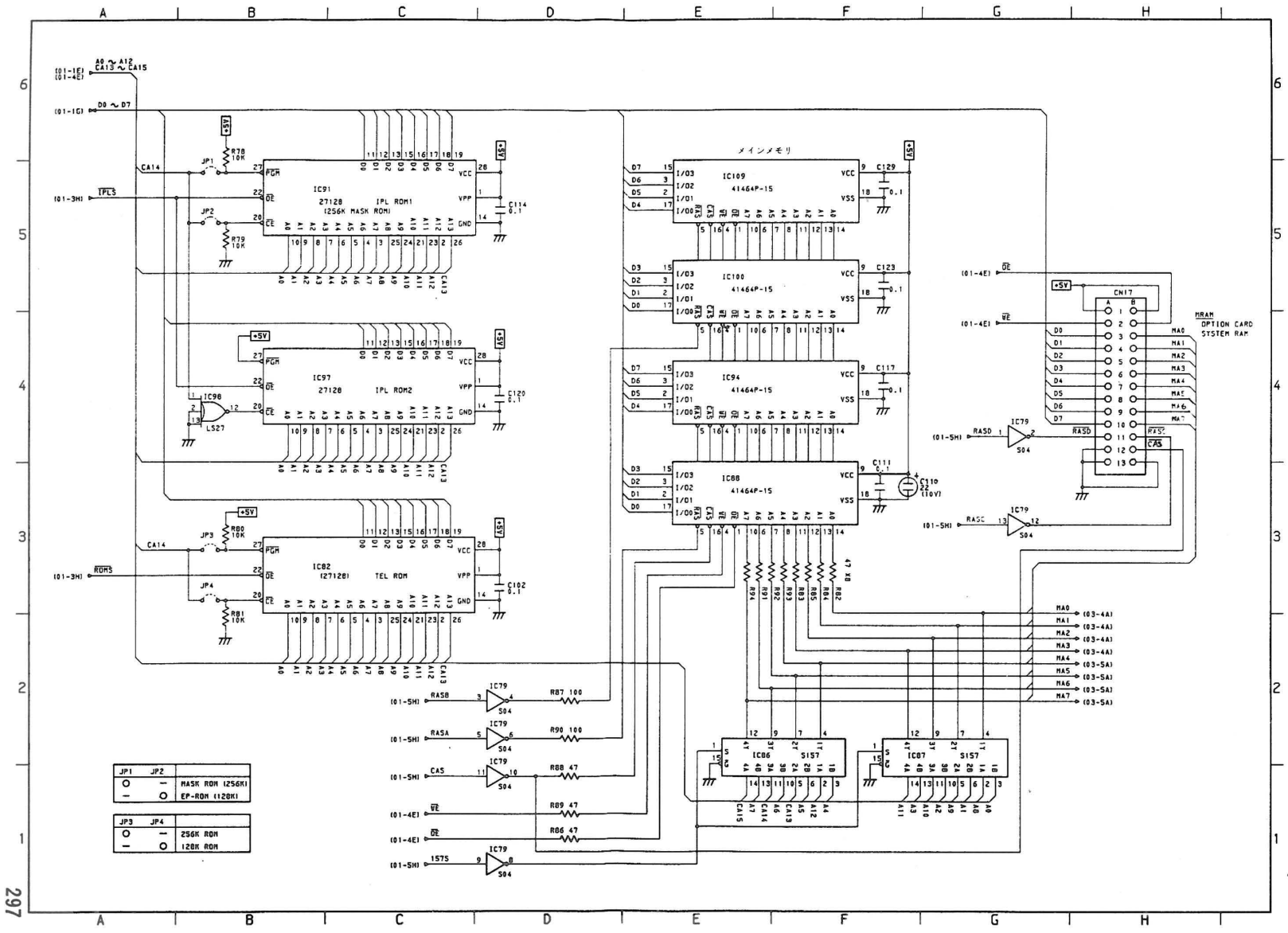
① CPU・メモリコントロール部



左側のZ-80B CPUとあるが，スーパーMZの中枢，セントラルプロセッシングユニット（CPU）です。A_{15}～A_0はCPUからメモリやI/Oポートを選択するための信号線，アドレスバスで，D_7～D_0は選択したメモリ等のデータをやり取りを行うための信号線です。また$\overline{MREQ}$はメモリをアクセスする，$\overline{IORQ}$はI/Oポートをアクセスする，$\overline{RD}$は読出しを行う，$\overline{WR}$は書込みを行う，$\overline{RF\ SM}$はD-RAMのリフレッシュを行う，M_1は命令の1バイト目を(これがCB$_H$，DD$_H$，ED$_H$，FD$_H$のいずれかなら2バイト目も)読出し中であることを示す，HALTは$\overline{HALT}$命令(76$_H$)を実行して停止していることを示す，$\overline{BUSAK}$はDMAによってバスを空け渡していることを示す，出力信号です。$\overline{INT}$は通常の割込みを行う，$\overline{NMI}$はノンマスカブル割込みを行う，$\overline{WAIT}$はデータバスを使用する時間引き延ばしを行う，$\overline{RESET}$はリセットを行う，$\overline{BUSRQ}$はDMAを行うことを，それぞれCPUにリクエストするためのもの，ϕはクロックを供給するもので，CPUにとっては入力信号線になります。CPUからの出力は，多数のICに伝達するために，バスバッファを入れています。なお，VCCとGNDは電源です。

一方，右側のTC17G1 4AF-0005は，CPUのメモリ空間を広げるためのメモリマッピングレジスタを持ち，メモリやCPUの細かな制御信号を発生させたりもするカスタムLSIで，C-MOS 1400ゲートで構成されるゲートアレイです。ポートB4$_H$～B7$_H$でプログラムからアクセスできます。CPUのクロック4MHz/6MHzの切換えや，メモリアクセス時のWAIT発生，MZ-2000/80Bモードの V-RAM等のバンク切換え等もここで行っています。

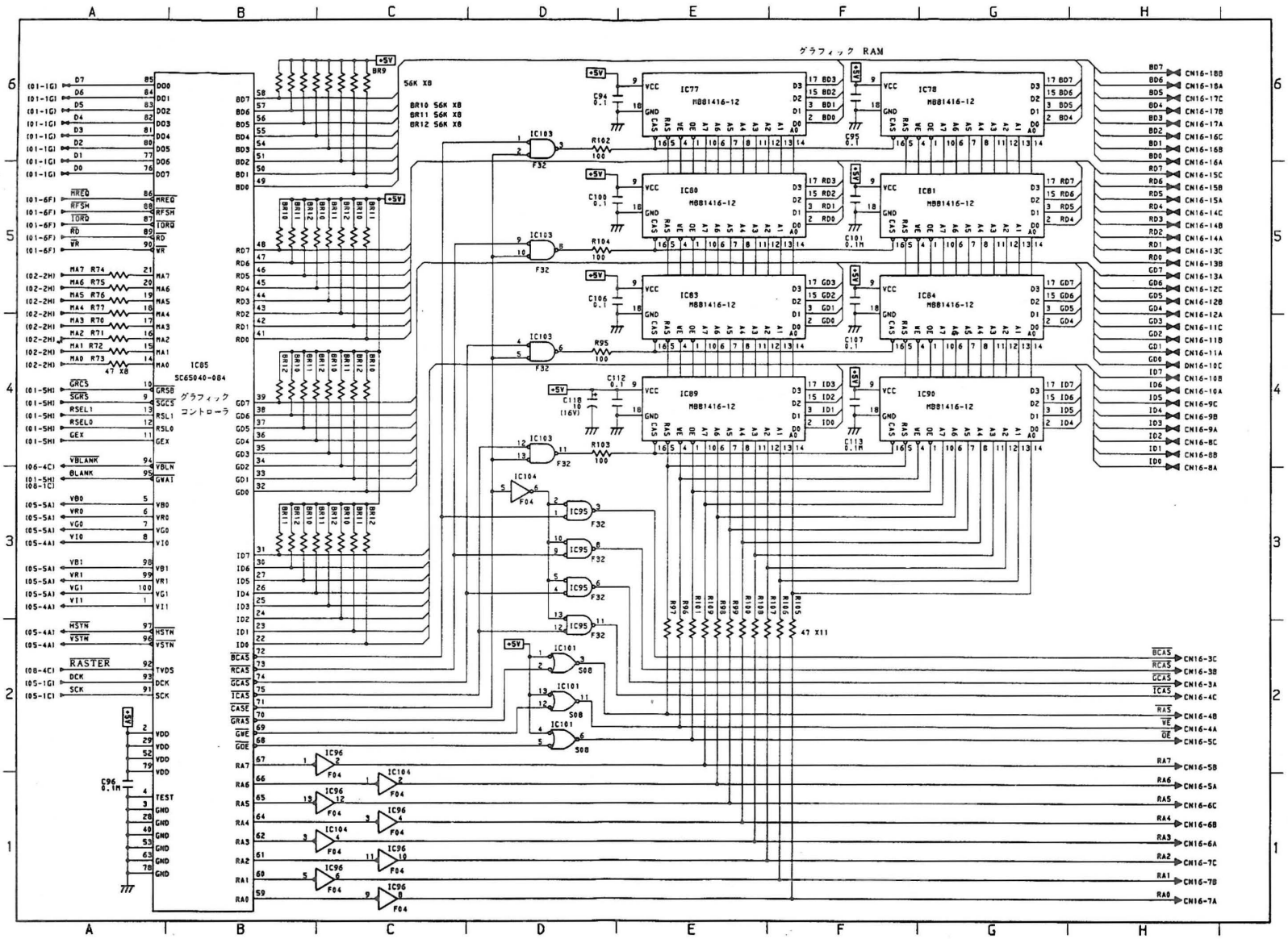
② IPL・BIOS・拡張ROM・メインRAM部



左上から縦に三つある「27128」は16KバイトのP-ROM（プログラマブルROM）で，上の二つはIPL・BIOSを2分割したもの，下の一つは通常ソケットだけが実装されている拡張（通信用）ROMです。IPL・BIOS ROMは近い将来マスクROM1個で置き替えられるよう，拡張ROMは将来的に大容量のプログラムを入れられるよう，32KバイトのROMも接続できるようになっています。

右側で最近登場した4ビット×64K（256Kビット）のD-RAM「41464」（μPD-41254C）を使い，これ4個だけで128Kバイトを実現しています（1ビット×64Kの「4164」なら16個必要）。また，拡張用RAM（MZ-1 R26）のソケットがさらに右にあります。

③ グラフィック部

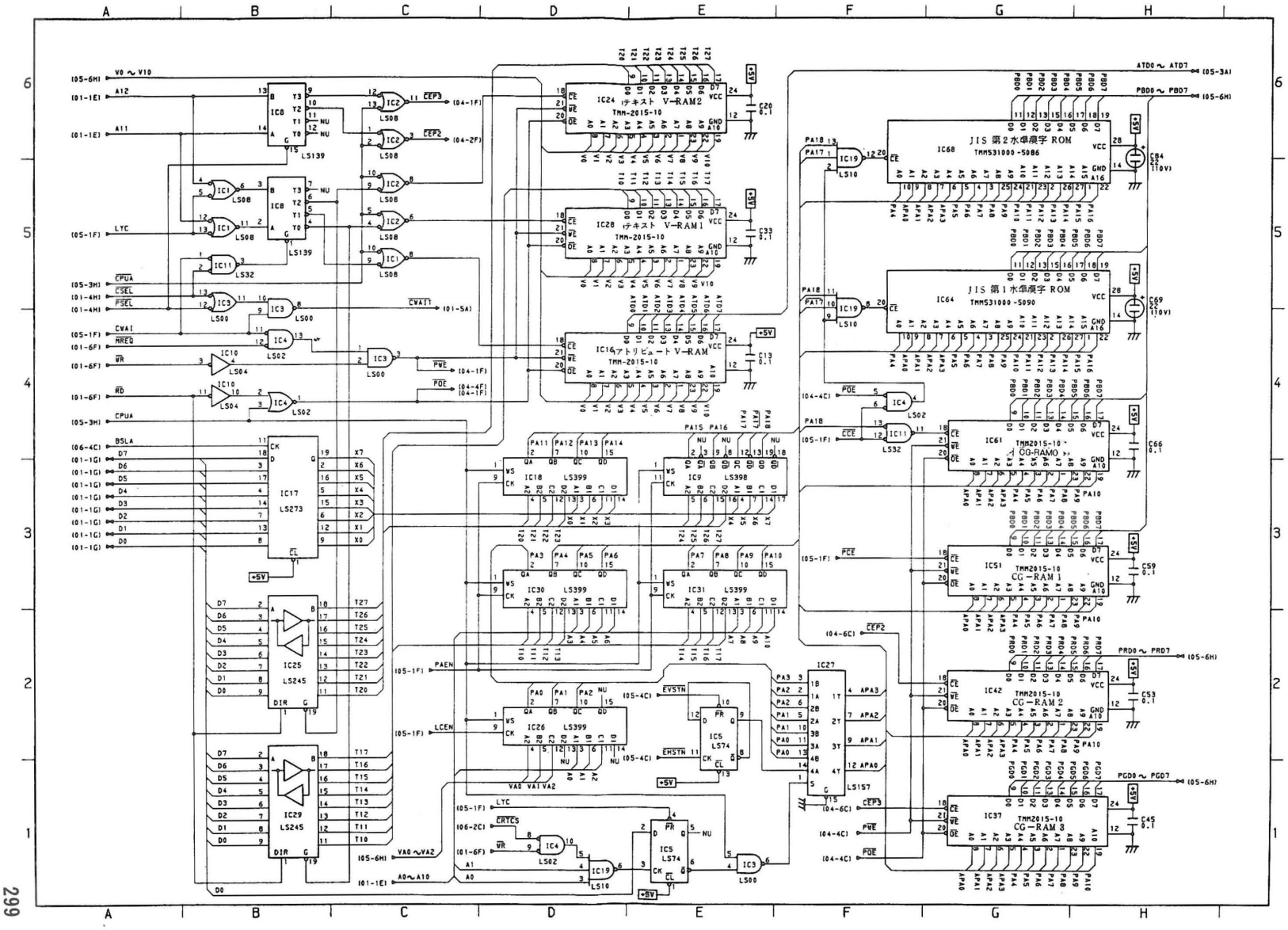


この部分はSC65S040-084というグラフィックコントローラ(G-CRTC)とグラフィックRAMです。

G-CRTCはポートBC$_H$～BF$_H$でアクセスされるカスタムLSI（ゲートアレイ）で，グラフィックRAMのアドレス発生，256色モードや2面画重ね合わせの制御，リードモディファイライト，サーチモード，グラフィックのスムーズスクロール，グラフィックのハードウェアビューポート，描画色とタイリングパターンの設定等は，すべてこれで行っています。

グラフィックRAMは256色モード等の実現のため4ビット×16K（64Kビット）のD-RAM「4416」（MB-81416）を使用し，1度のアクセスだけで4バイトを同時に読み書きできるようにしています。

④ テキストV-RAM・キャラクタジェネレータ部

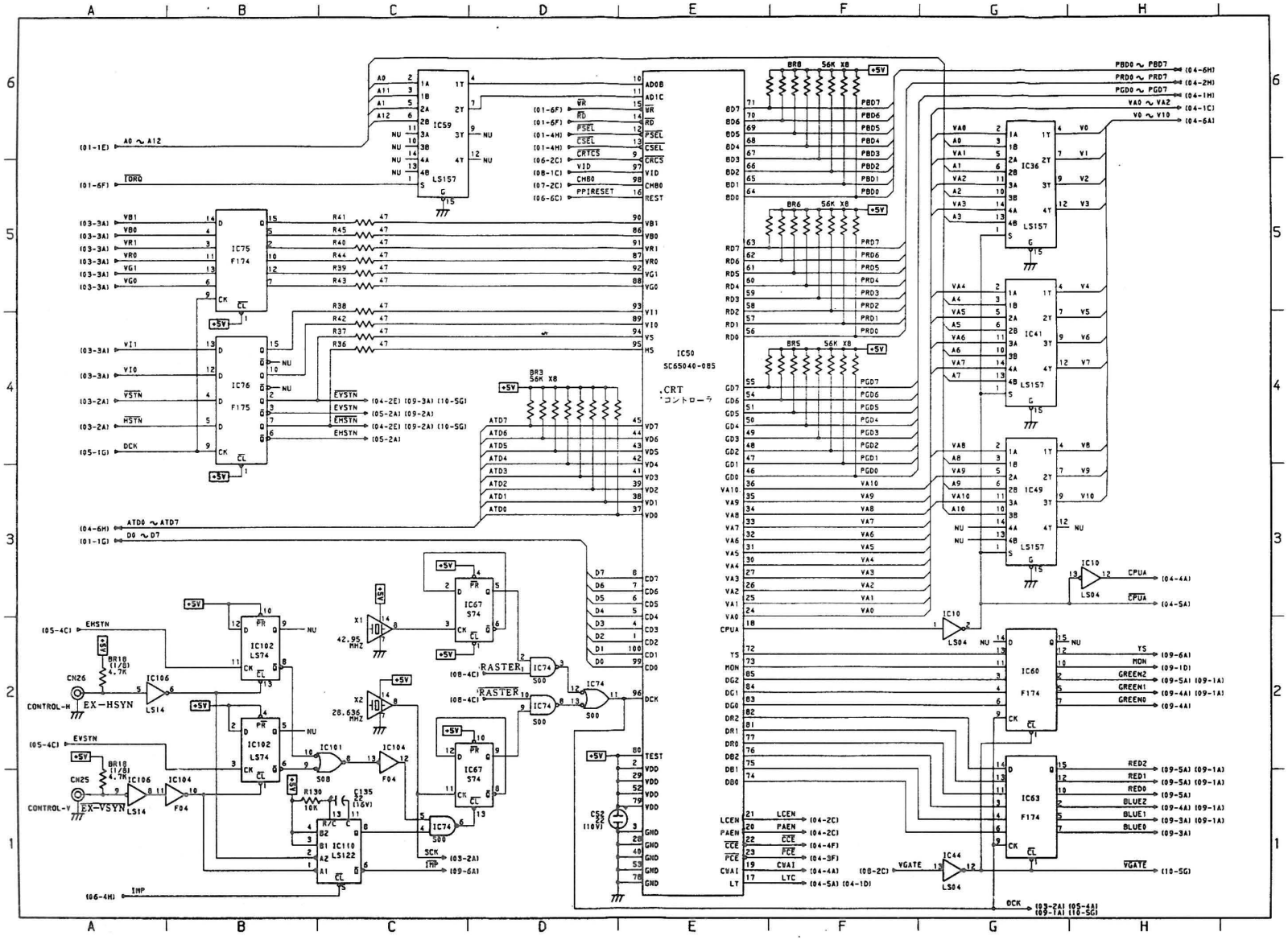


この部分は⑤のCRTCと密接な関係にあります。中央上部の8ビット×2Kのスタティック RAM「2016」(TMM-2015) はテキストV-RAM 1・2とアトリビュートV-RAMで，右にあるのは漢字フォントを格納した2個の128Kバイトマスク ROM「231000」(TMM-31000)，プログラマブルキャラクタジェネレータ (PCG) 0～3を担当する4個の「2016」(TMM-2015) です。

これらのメモリはCRTC系とCPUの両方からアクセスされますので，左下にあるバスバッファ (74LS245×2) でその切換えを行っています。また，その上の (74) LS273は，ポートCFHで出力したキャラクタジェネレータの選択状態を保持するTTLです。

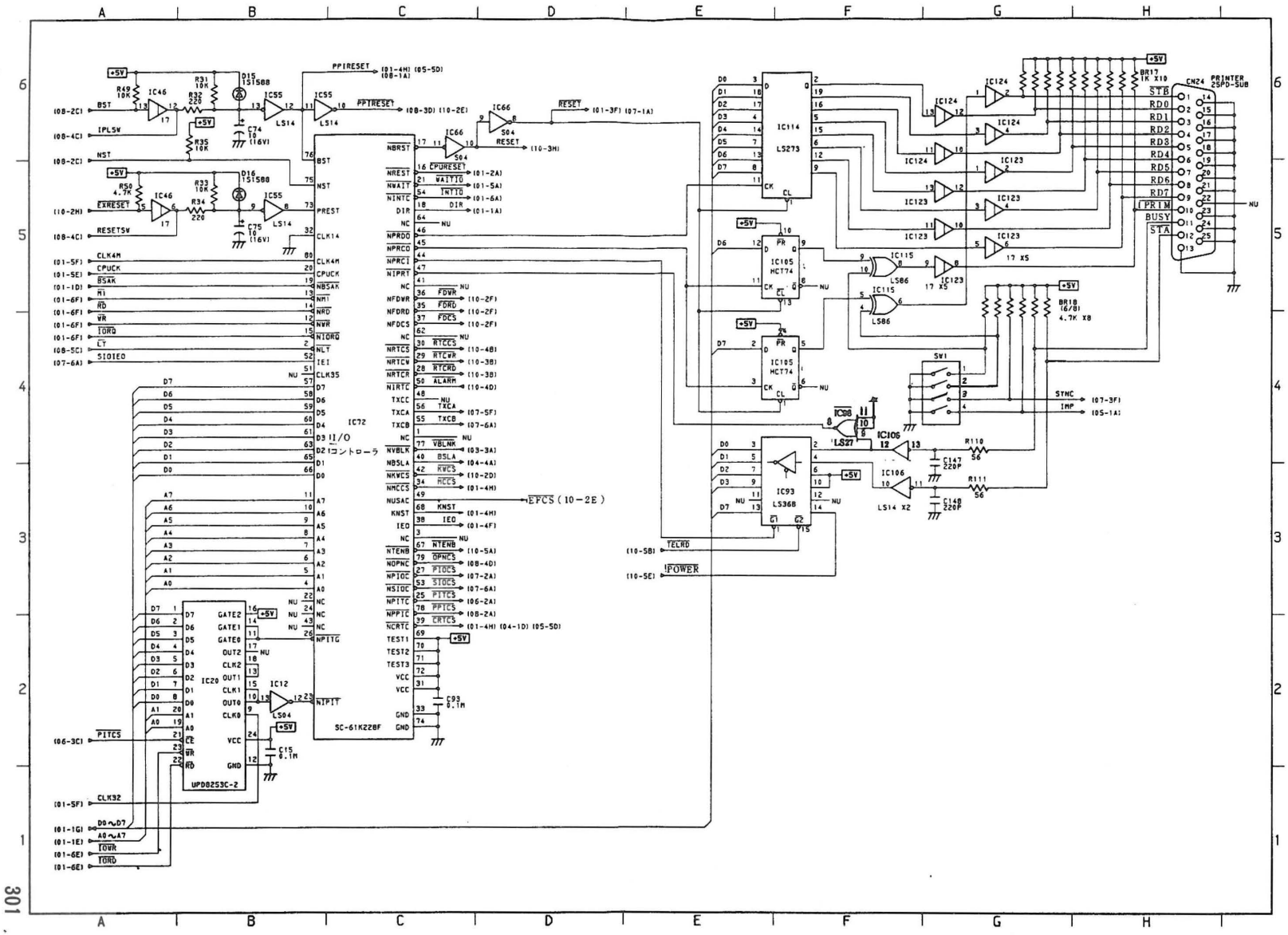
左下にあるTTLで構成される部分は，400ライン時の内部/外部同期（スーパーインポーズ用）の切換えを行っています。この信号はCRTCと③のG-CRTCに供給されています。

⑤ CRTコントロール部



中央にあるSC65S040-085というCRTコントローラ（CRTC）はポートF4H～F7HでアクセスされるカスタムlSI（ゲートアレイ）で，テキストV-RAM（含．アトリビュート）のアドレス発生，テキストのスムーズスクロール，テキストのハードウエアビューポート，グラフィックのパレットレジスタ，テキストとグラフィッのプライオリティ付き合成等を行っています。また，内部レジスタの設定により，自分自身のアクセス方法を変えることで，MZ-2000/80Bモードにも対応しています。

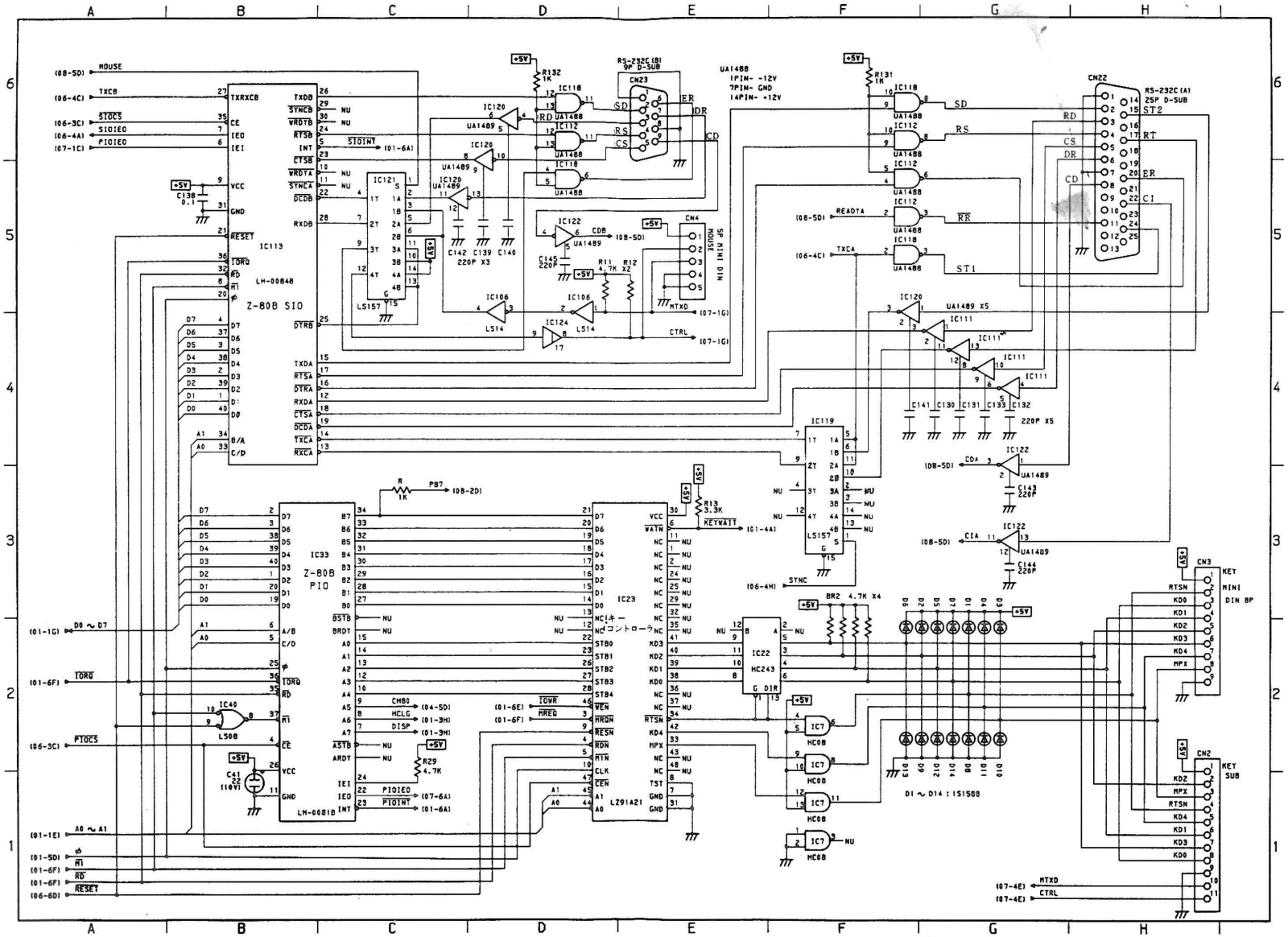
⑥ I/O・割込みコントロール部



I/Oコントローラ「SC-61K228F」はC-MOS 1000デートで構成されるカスタムLSI（ゲートアレイ）で，各種LSIへのポートアドレスのデコードや，垂直同期信号・インターバルタイマ・プリンタ・カレンダクロックからのモード2割込み発生代行，SIOアドレス・ボーレイト選択等を行っています。また，一部のI/Oポートアクセス時のWAIT発生と，CPU等のリセット信号（RESETスイッチではCPUやこのLSI，電源投入・IPLスイッチでは全部のICとLSIをリセットする）発生も行っています。ポートアドレス$C6_H$～$C7_H$とCD_Hをアクセスすると，このLSIが選択されます。

また，左下にはインターバルタイマとして利用している8080系用の汎用カウンタ「8253」（μPD8253C）が，右上にはTTLで構成されるプリンタインターフェースがあります。ここでは，ポートCA_Hのb7（CPUから見て入力）のボイスコミュニケーションインターフェースボード実装時，電話着信により電源が入ったかどうかも処理しています。

⑦ RS-232C・マウス・キーボード制御部

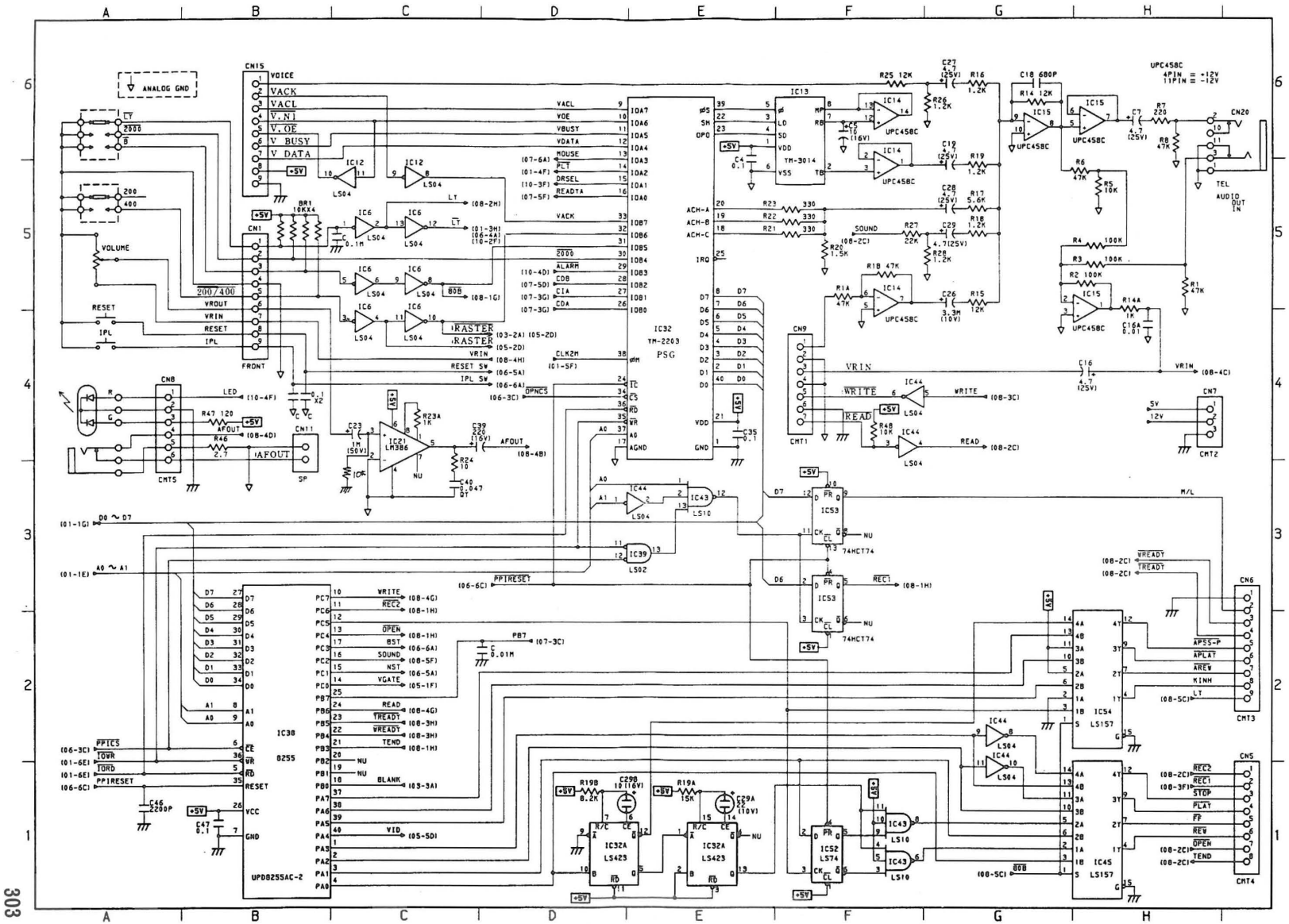


左上のZ-80B SIOは汎用シリアル入出力LSIで，AチャンネルはRS-232CのAポートに，Bチャンネルは切換えによってRS-232CのBポートとマウスインターフェースに使われています。

左下のZ-80B PIOは，汎用パラレル入出力LSIで，キーボードやMZ-2000/80B時のバンク切換えを行っています。

中央下部の「LZ91A21」は，キーボードをセパレートタイプにし，しかも従来機との互換性をとるために開発されたカスタムLSI(ゲートアレイ)で，4ビットの双方向線をカラムアドレス出力，データ入力(下位)，同上位入力を時分割で行うために使われます。また，キーボードコネクタを電源供給中（動作中）に抜き差ししてしまった場合の保護用として，ダイオードが14個使用されています。

⑧ カセット及び音声合成インターフェース・サウンド部

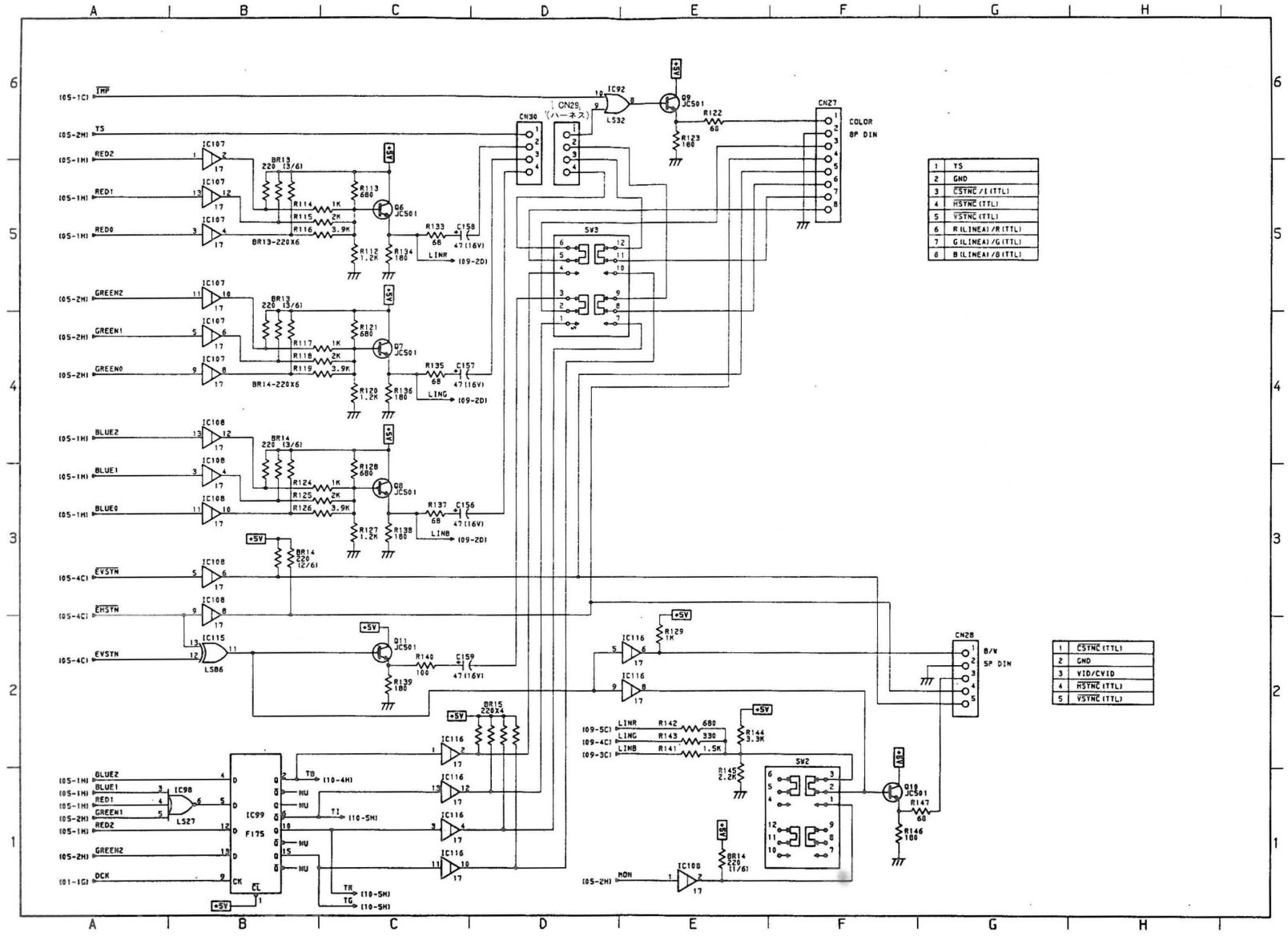


中央上部にあるLSI，「YM-2203」は，AY-3-8910コンパチのSSG音源と，独自のFM音源（OPN）により合計8オクターブ6和音＋1効果音のサウンドを発生するもので，右隣のYM-3014はその内のFM音源部の信号をD/A変換にり通常のアナログ信号に変換するものです。また，その左には，音声合成ボード（MZ-1M08）の端子があります。YM-2203には汎用入力ポートが付いていますので，それを利用して音成合成やモードスイッチ等とのコミュニケーションを行っています。

左下にはi8080系用の汎用パラレル入力LSI，「8255」（μPD8255AC）があり，主にカセット関係の制御に使われています。右端にはカセットデッキ部との接続用のコネクタがあり，その途中にはMZ-2500・2000と80Bモードの切り換え回路が入っています。

また，右上にはYM-2203のSSG部，OPN部，音声合成，8255のCポート b_2 のダイレクトサウンド出力，カセットデッキのアナログ部再生信号の合成回路（抵抗器を介し，そのまま接続されている）があり，この信号は背面のAUDIO端子（モデムホン）に接続されています。また，この信号とモデムホンからの入力音声とが合成され，カセットデッキのアナログ録音部（ライン）や，さらにアンプを通した上で内蔵スピーカーと前面のイヤホン端子に接続されています。

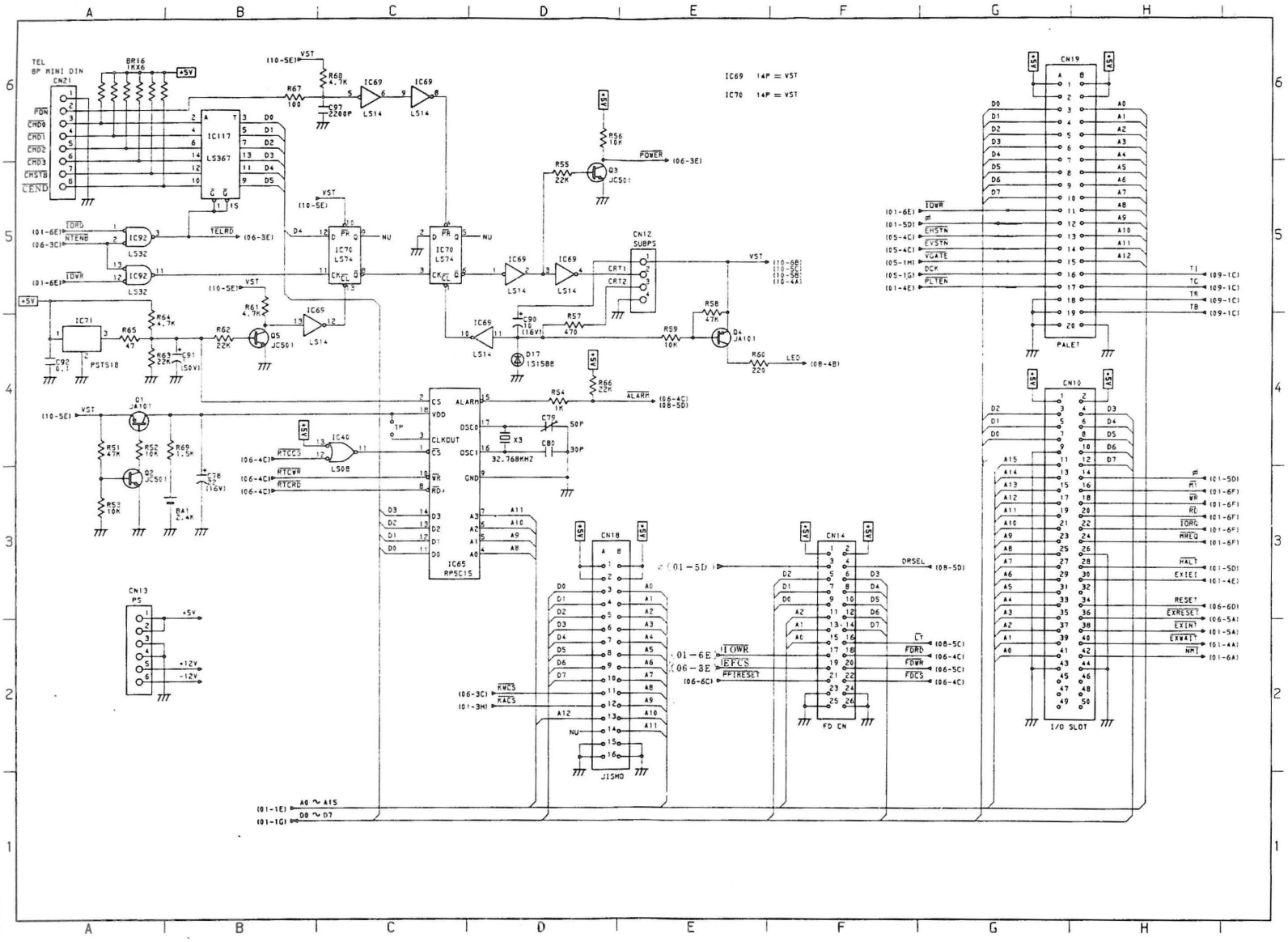
⑨ アナログ・デジタルディスプレイ切換え部



ここでは，3ビット8階調のR・G・B信号を，D/A変換後アナログRGBとして出力するか，その最上位ビットだけを採用してデジタルRGB（+I）として出力するか，また，モノクロディスプレイに対してアナログRGBをR：G：B≒5：10：2で混合してアナログで出力するか，ORをとってデジタルで出力するか（この時はMZ-2000・80B内蔵ディスプレイコンパチとなり，テキストのみの出力とグラフィック合成の出力とを選択できる）を選択します。

中央上部のCN30とCN29（コネクタ）は通常そのまま接続されていますが，4096色パレットボード（MZ-1M10）を使用する時は，この途中にパレットレジスタを置くような仕組みになっています。

⑩ カレンダクロック・リモート・各種スロット部



左上部にはポートCA$_H$でアクセスされるモデムホンとのインターフェース回路があります。

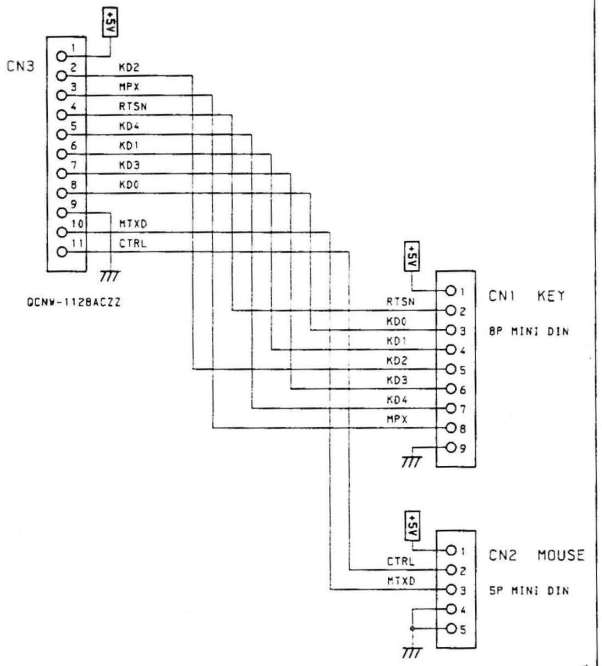
その下にはカレンダクロック「RP-5C15」と，そのバッテリバックアップ回路があります。バックアップには 2.4 V のニッカド電池が使われています。また，RP5C15のレジスタ選択信号 A_3～A_0には，Z-80のA_{11}～A_8が接続されているのがわかります。

右の方には，4096色パレットボード(MZ-1M10)，拡張I/Oポート (MZ-1U09)，辞書ROMボード(MZ-1R28)，FDC・ジョイスティック部との接続コネクタがあります。

なお，バックアップ用バッテリの充電用回路とIC69～70(74LS14・74LS74)は，メインパワースイッチ ONでパワーセレクトスイッチ＝リモート（電話待機中）の状態でも通電されており，電話着信により，それを電源部に伝え，システム全体の電源を ON にしてしまいます。

⑪ キーボード・マウス端子(前面)部

MINI DIN PWB



FD I/F PWB

CN1

1	+5V	2	+5V
3	ø	4	DRSEL
5	D2	6	D3
7	D1	8	D4
9	D0	10	D5
11	A2	12	D6
13	A1	14	D7
15	A0	16	$\overline{LT}$
17	$\overline{IOWR}$	18	$\overline{FDRD}$
19	$\overline{FFCS}$	20	$\overline{FDWR}$
21	PPIRESET	22	$\overline{FDCS}$
23	GND	24	GND
25	GND	26	GND

CN2

1	GND	2	NC
3	GND	4	NC
5	GND	6	DRIVE SELECT 4A
7	GND	8	INDEX
9	GND	10	DRIVE SELECT 1A
11	GND	12	DRIVE SELECT 2A
13	GND	14	DRIVE SELECT 3A
15	GND	16	MOTOR ON
17	GND	18	DIRECTION SELECT
19	GND	20	STEP
21	GND	22	WRITE DATA
23	GND	24	WRITE GATE
25	GND	26	TRACK 00
27	GND	28	WRITE PROTECT
29	GND	30	READ DATA
31	GND	32	SIDE ONE SELECT
33	GND	34	NC

CN3

1	NC	20	GND
2	NC	21	GND
3	DRIVE SELECT 4B	22	GND
4	INDEX	23	GND
5	DRIVE SELECT 1B	24	GND
6	DRIVE SELECT 2B	25	GND
7	DRIVE SELECT 3B	26	GND
8	MOTOR ON	27	GND
9	DIRECTION SELECT	28	GND
10	STEP	29	GND
11	WRITE DATA	30	GND
12	WRITE GATE	31	GND
13	TRACK 00	32	GND
14	WRITE PROTECT	33	GND
15	READ DATA	34	GND
16	SIDE ONE SELECT	35	GND
17	GND	36	GND
18	GND	37	GND
19	GND		

CN4

1	FWD1
2	BACK1
3	LEFT1
4	RIGHT1
5	+5V
6	TRGA1
7	TRGB1
8	COM1
9	GND

CN5

1	FWD2
2	BACK2
3	LEFT2
4	RIGHT2
5	+5V
6	TRGA2
7	TRGB2
8	COM2
9	GND

CN6

1	ø
2	$\overline{IOWR}$
3	A7
4	A6
5	A5
6	A4
7	A3

CPU PWB

CN16 GRAM

	A	B	C
1	+5V	+5V	+5V
2	+5V	+5V	+5V
3	$\overline{GCAS}$	$\overline{RCAS}$	$\overline{BCAS}$
4	$\overline{VE}$	$\overline{RAS}$	$\overline{ICAS}$
5	RA6	RA7	$\overline{OE}$
6	RA3	RA4	RA5
7	RA0	RA1	RA2
8	ID0	ID1	ID2
9	ID3	ID4	ID5
10	ID6	ID7	GD0
11	GD1	GD2	GD3
12	GD4	GD5	GD6
13	GD7	RD0	RD1
14	RD2	RD3	RD4
15	RD5	RD6	RD7
16	BD0	BD1	BD2
17	BD3	BD4	BD5
18	BD6	BD7	GND
19	GND	GND	GND
20	GND	GND	GND

CN22 RS-232C(A)

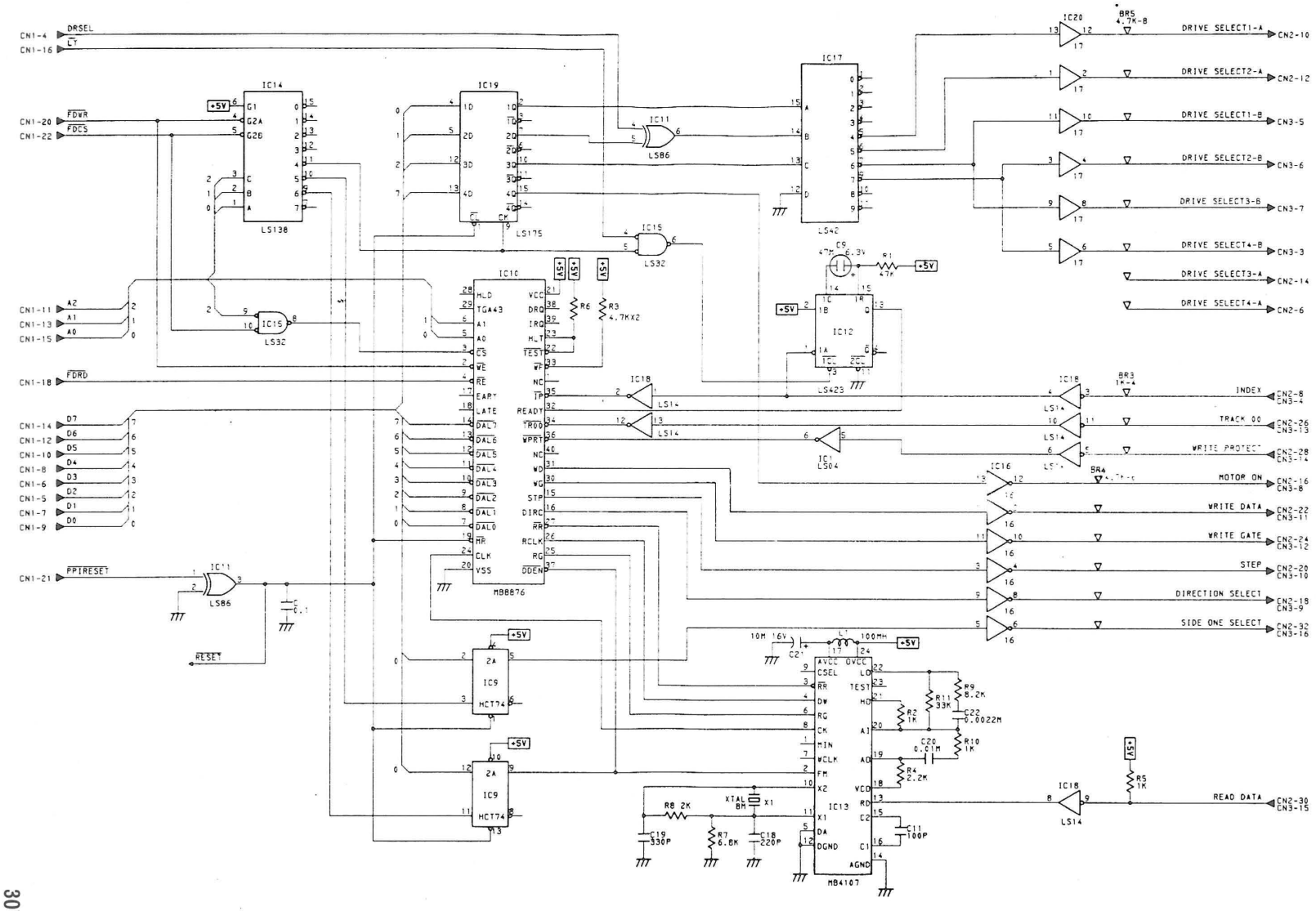
1	FG	14	
2	SD	15	ST2
3	RD	16	
4	RS	17	RT
5	CS	18	
6	DR	19	
7	SG	20	ER
8	CD	21	
9		22	CI
10		23	
11	$\overline{RR}$	24	ST1
12		25	
13			

CN23 RS-232C(B)

1	FG	6	ER
2	SD	7	DR
3	RD	8	SG
4	RS	9	CD
5	CS		

これは，本体前面(カンガルーポケットの隣)にあるキーボード・マウス端子と，FDC・ジョイスティック基板上の各種コネクタ(CN1＝メイン基板とのコミュニケーション，CN2＝内蔵3.5″ FDDへ接続，CN3＝外部FDDへの接続，CN4・5＝ジョイスティック端子，CN6＝I/Oコントローラにジョイスティックデコード回路が入るまでの間，FDD・ジョイスティック基板側でデコードを行うために必要な信号)，メイン基板上の各種コネクタ(拡張グラフィックRAMとRS-232C用)の端子番号説明図です。

⑫ FDC(フロッピーディスクコントローラ)部(FDC/ジョイスティック基板上)

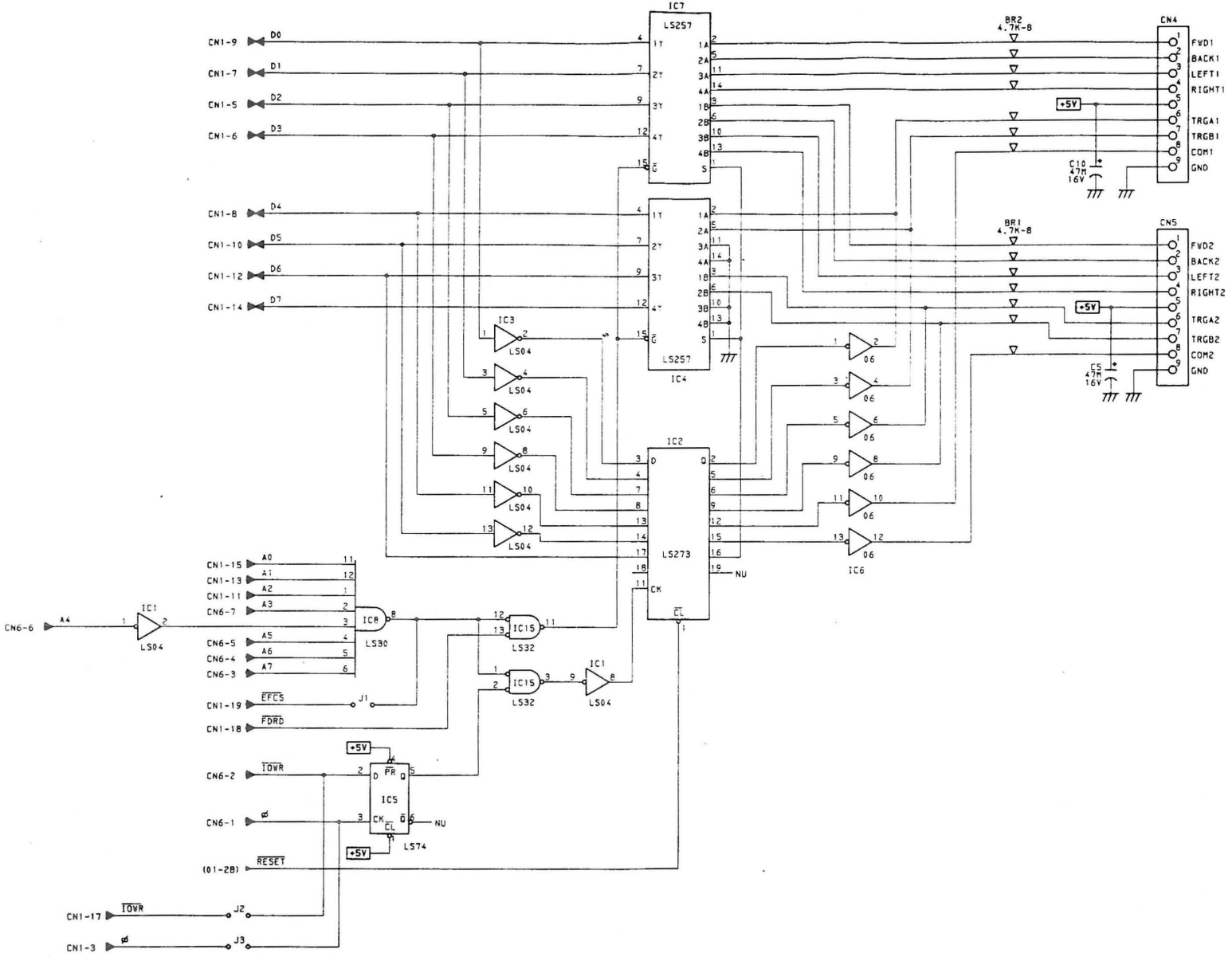


ほぼ中央にあるのがフロッピーディスクコントローラ(FDC)の「8876」(MB8876A)です。このLSIはデータバスが負論理(DAL 7～DAL 0)なのにCPUのデータバス(正論理)に直結しているため，データの入出力等は反転しています。

下部の「MB4107」はVFO用のLSIで，MFM(倍密度)の読出しの際，利用されています。

他に，モータON・OFF，ドライブ選択，MFM/FM(単密度)の選択，ヘッド(面・側)の選択が，TTLで行われています。

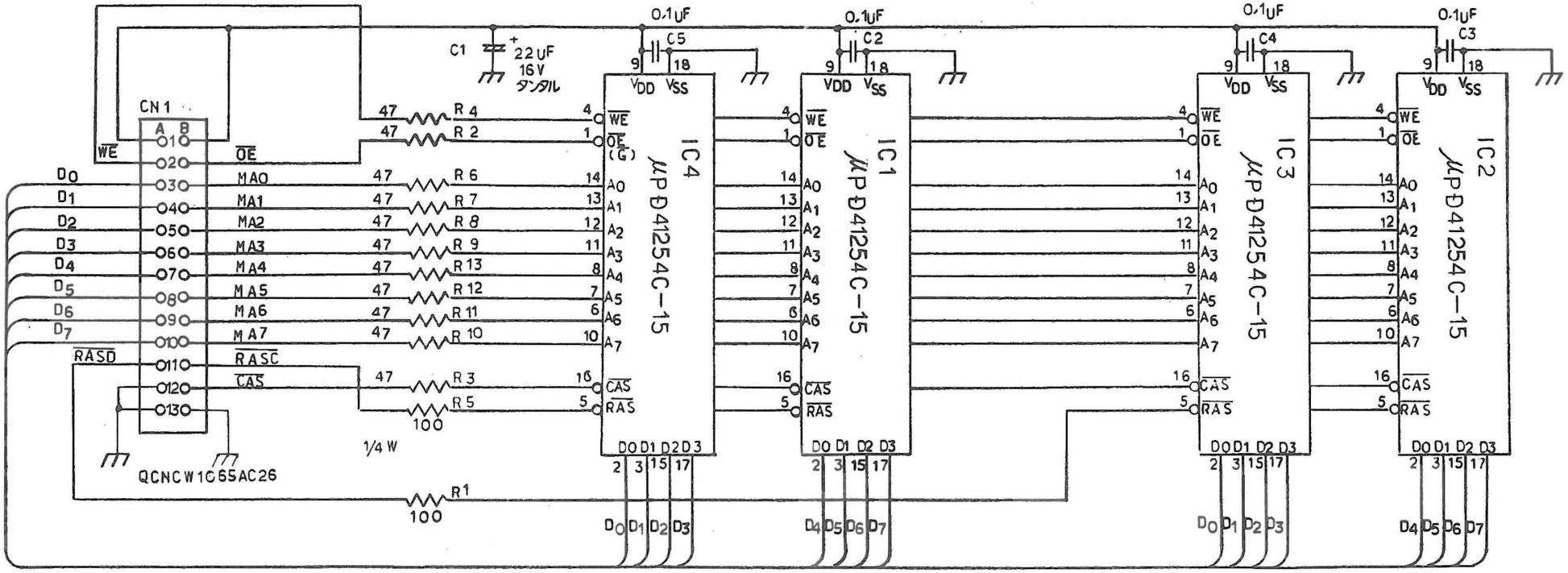
⑬ ジョイスティックインターフェース部(FDC/ジョイスティック基板上)



この部分はジョイスティック（本来入力のみ）とのインターフェースを行っていますが，汎用の入出力ポートとしても使えるよう，回路が複雑になっています。1ポート当たり，入力7本としても，入力4本・出力3本としても利用可能です。

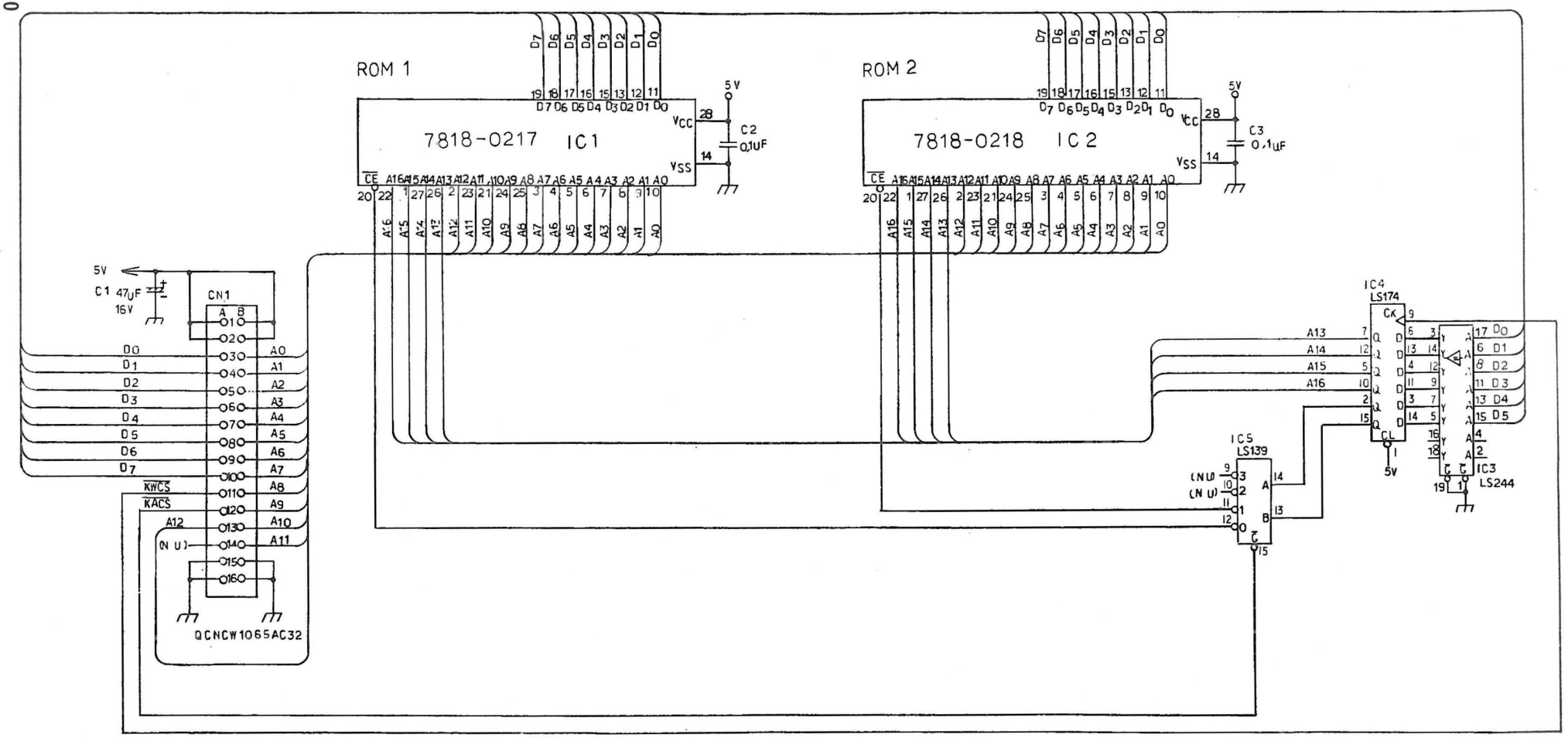
左下にはポートEF$_H$のデコード回路がありますが，近い将来I/Oコントローラ内にこの機能が入るようになった時には，(74)LS30は省かれ，J1がショート(接続)されることになると思います。また，その時にはCN6も省略される筈です(機能的な差はない)。

⑭ 拡張RAMボード(MZ-1R26)



この基板には4ビット×64Kの「4464」(μPD41254C)というD-RAMと，ノイズ軽減用の抵抗器・コンデンサが入っているだけで，デコード回路は本体側にあります。たいへんシンプルなボードなので，周辺装置のメーカーから，安価なコンパチボードが出て来ることも考えられます。

⑮ 辞書ROMボード(MZ-1R28)

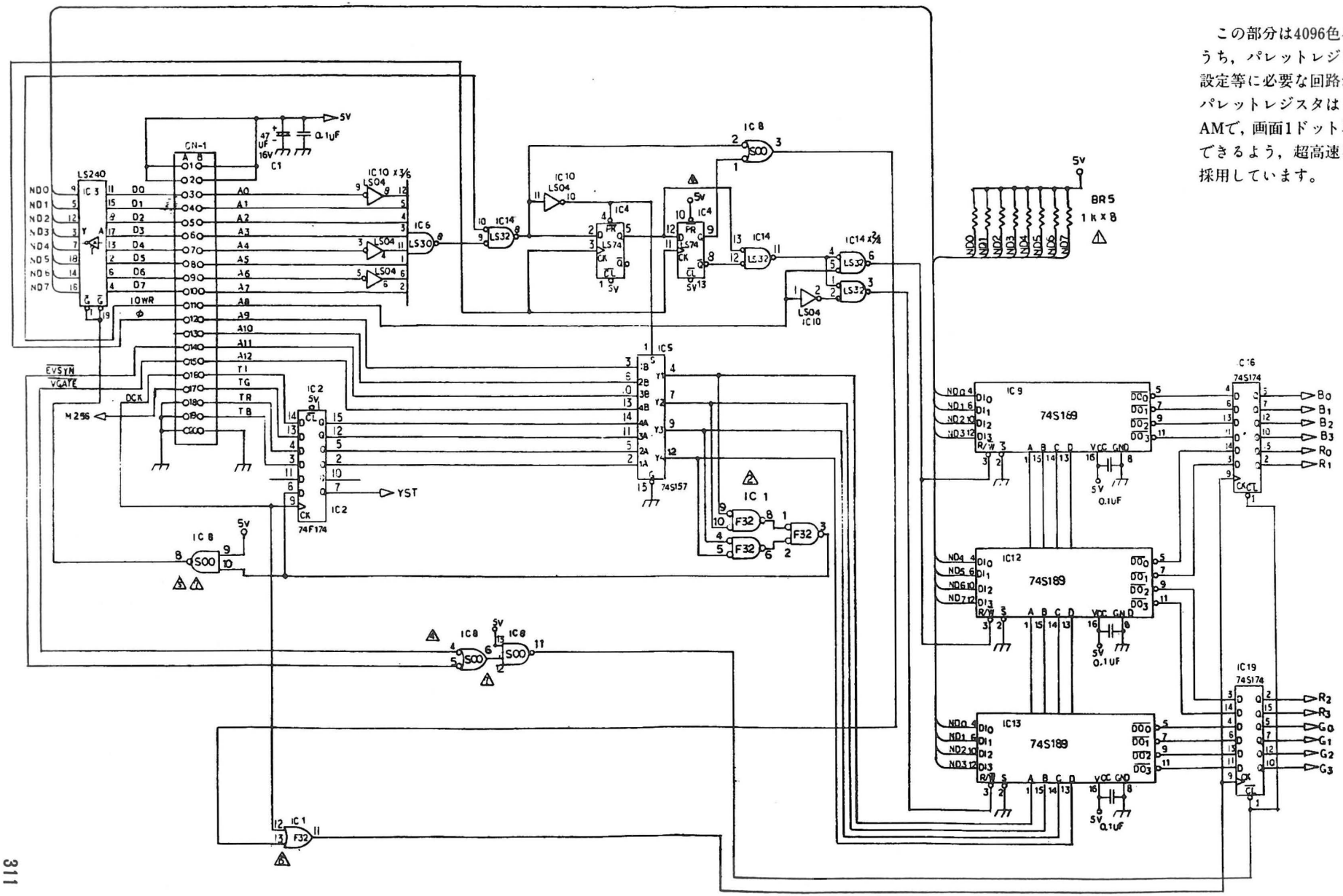


この基板には2個の128Kバイトマスク ROM「231000」(SC7818-0217・0218)が載っており，右下のTTL，(74)LS139でその振り分けを行っています。回路上では辞書ROMは倍の容量(512K)まで拡張できるように配慮されているのがわかります。また，将来的には，128K2個でなく，256KバイトのROM 1個(512K時も2個)で置き換えることもできるようになっています(74LS139は不要になる)。

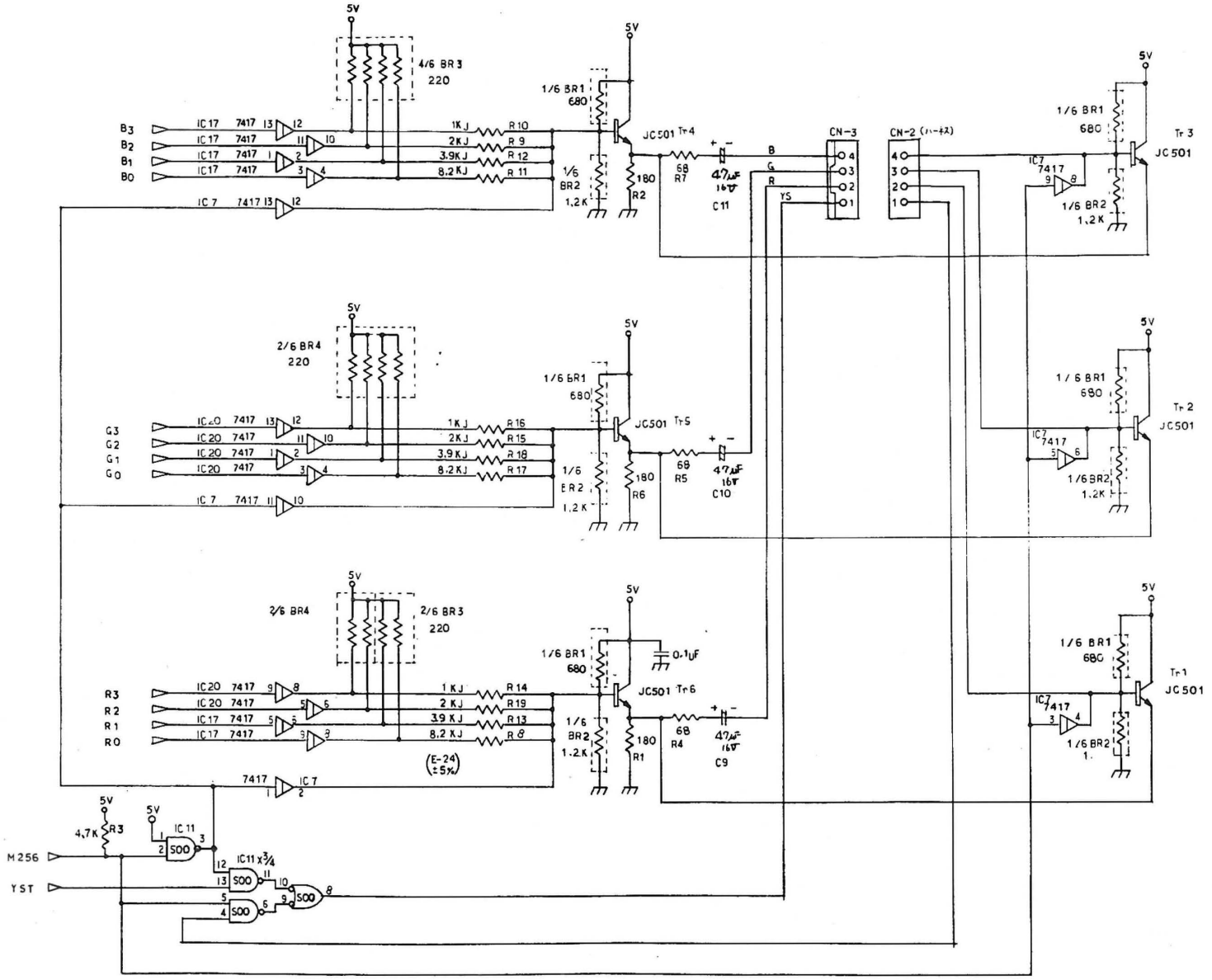
なお，(74)LS174はポートCF$_H$で設定される辞書ROMアドレス A$_{13}$以上を保持するために使われています。

⑯ パレットボード(MZ-1-M10)パレットレジスタ部

この部分は4096色パレットボードのうち，パレットレジスタ×３と，その設定等に必要な回路から成っています。パレットレジスタは４ビット×16のRAMで，画面1ドット単位の変化に追随できるよう，超高速なTTL(74S189)を採用しています。



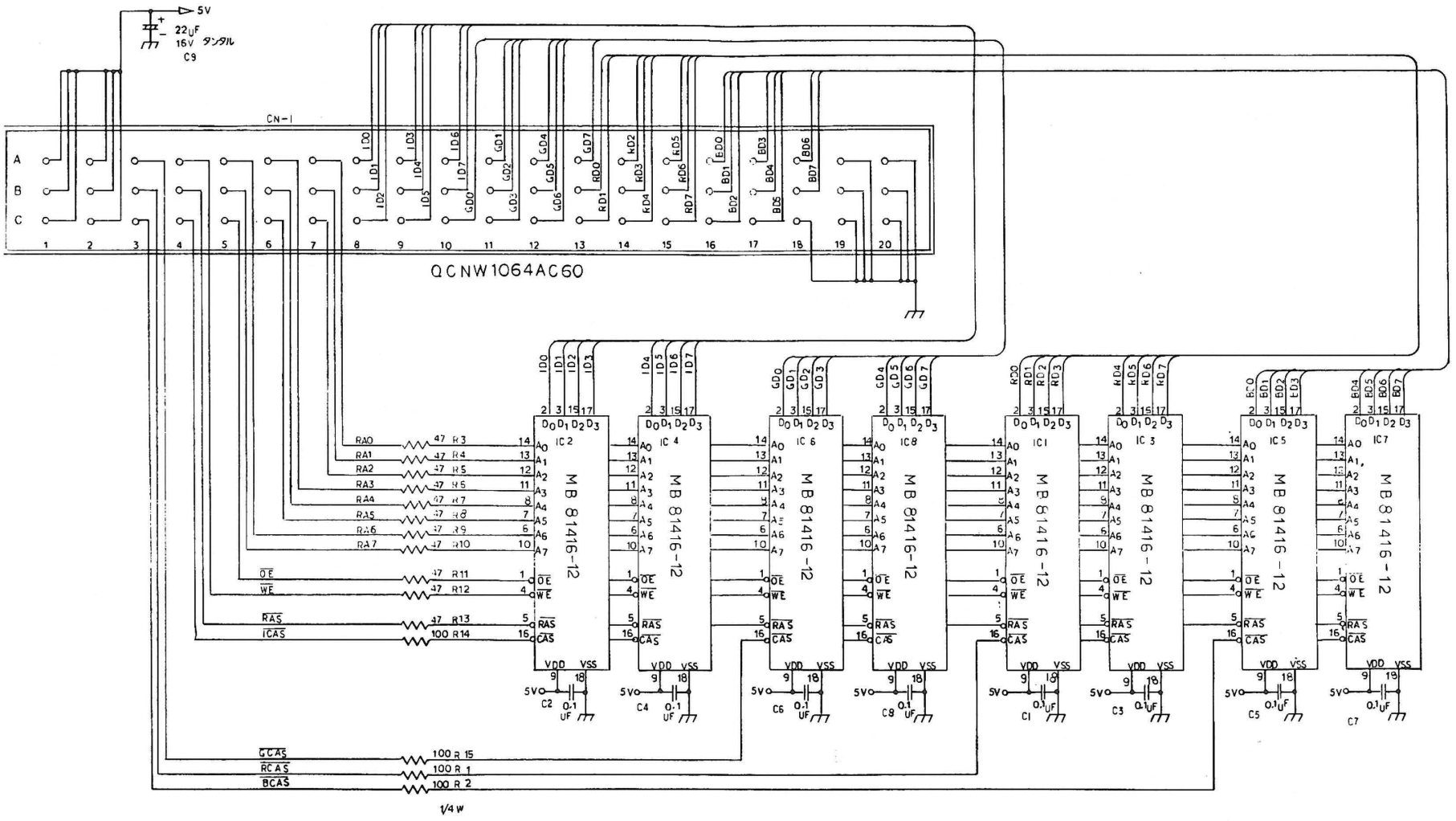
⑰ パレットボード(MZ-1R27)D/A変換部



この部分は4096色パレットボードのうち，D/A変換部と，それを無効にする回路から成っています。256色モード時やこのボードを使用しない時はYM-2203ポートA(レジスタ0E$_H$)のb2を1にして，右側のトランジスタつまりCN-2から来る本体側の信号を有効にし，COLOR=(1，0，0，7)のようにして4096色パレットレジスタを使う場合はb2を0にして，左側のトランジスタつまり本ボードで作られたアナログRGB信号を有効にするわけです。

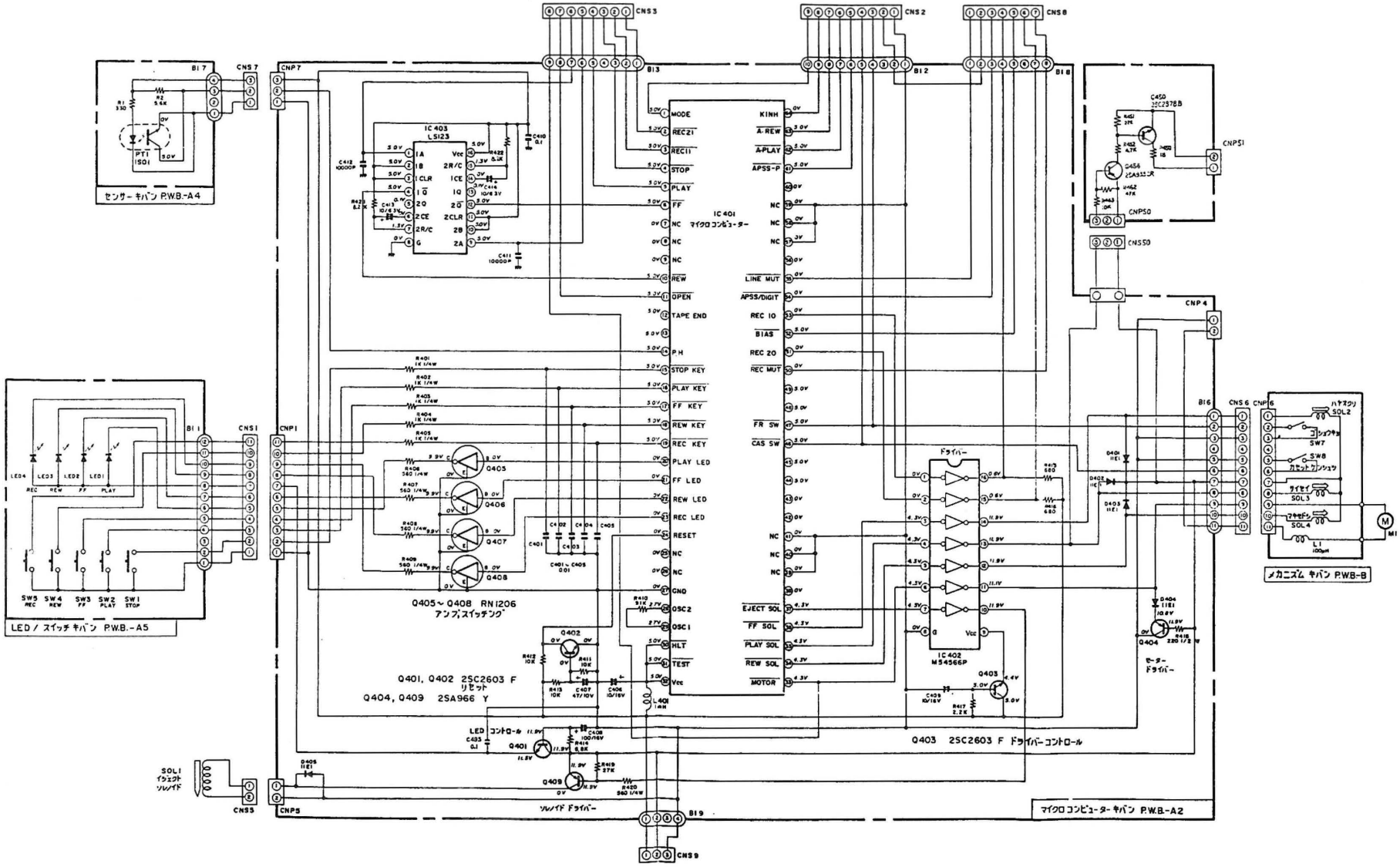
なお，CN3を本体側のCN29(⑨図参照)と，CN2を本体側のCN30と接続して初めて，本ボードは有効になります。

⑱ 拡張グラフィックRAMボード(MZ-1R27)



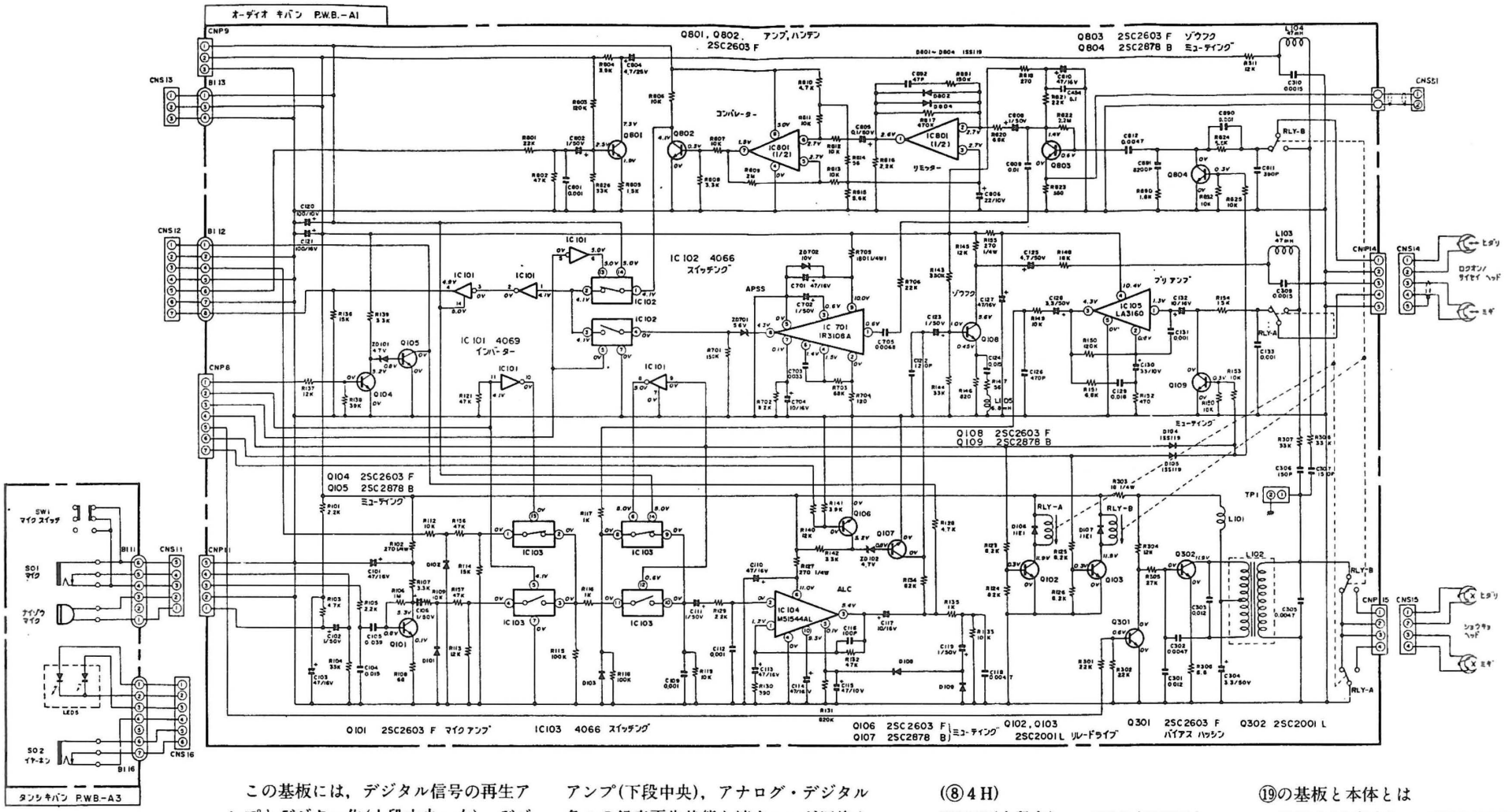
この基板は拡張メインRAMと同様，4ビット×16Kの「4416」(MB 81416) というD-RAMと，ノイズ軽減用の抵抗器・コンデンサが入っているだけで，デコーダ回路は本体側にあります。4バイト同時アクセス用にデータ線が32本もあるため，端子数の特に多いコネクタを使用します。

⑲ カセットデッキワンチップマイコン部



この基板には，電磁メカのカセットデッキを制御するためのワンチップマイコンが載っており，前面パネルのPLAY等のスイッチ，その隣のLED（ランプ），再生・巻き戻し・早送り・イジェクト用のソレノイド(リレーのように金属片を電磁石で物理的に移動させるもの)，キャピスタンや巻き取り用のモータ，カセットが入っているかどうかの検出スイッチ，ライトプロテクト用ツメの検出スイッチ，巻き取り軸(右側)が正常に回転しているかどうかのセンサー(フルオートストップ用)が接続されています。

⑳ カセットデッキアンプ部



この基板には，デジタル信号の再生アンプとデジタル化(上段中央～右)，デジタル信号の録音アンプ(上段左)，アナログ信号の再生アンプ(中段右)，デジタル信号のAPSS時アンプ(中段中央)，APSS時アンプと再生アンプの選択(中段左)，マイク端子・マイク・ラインの切換え(下段左)，アナログ部録再兼用のパワーアンプ(下段中央)，アナログ・デジタル各々の録音再生状態と消去ヘッド切換え用リレー・駆動回路及び消去ヘッド用交流バイアス発生回路(下段右)が載っており，ヘッドやマイク等が接続されています。

なお，この基板と本体とは

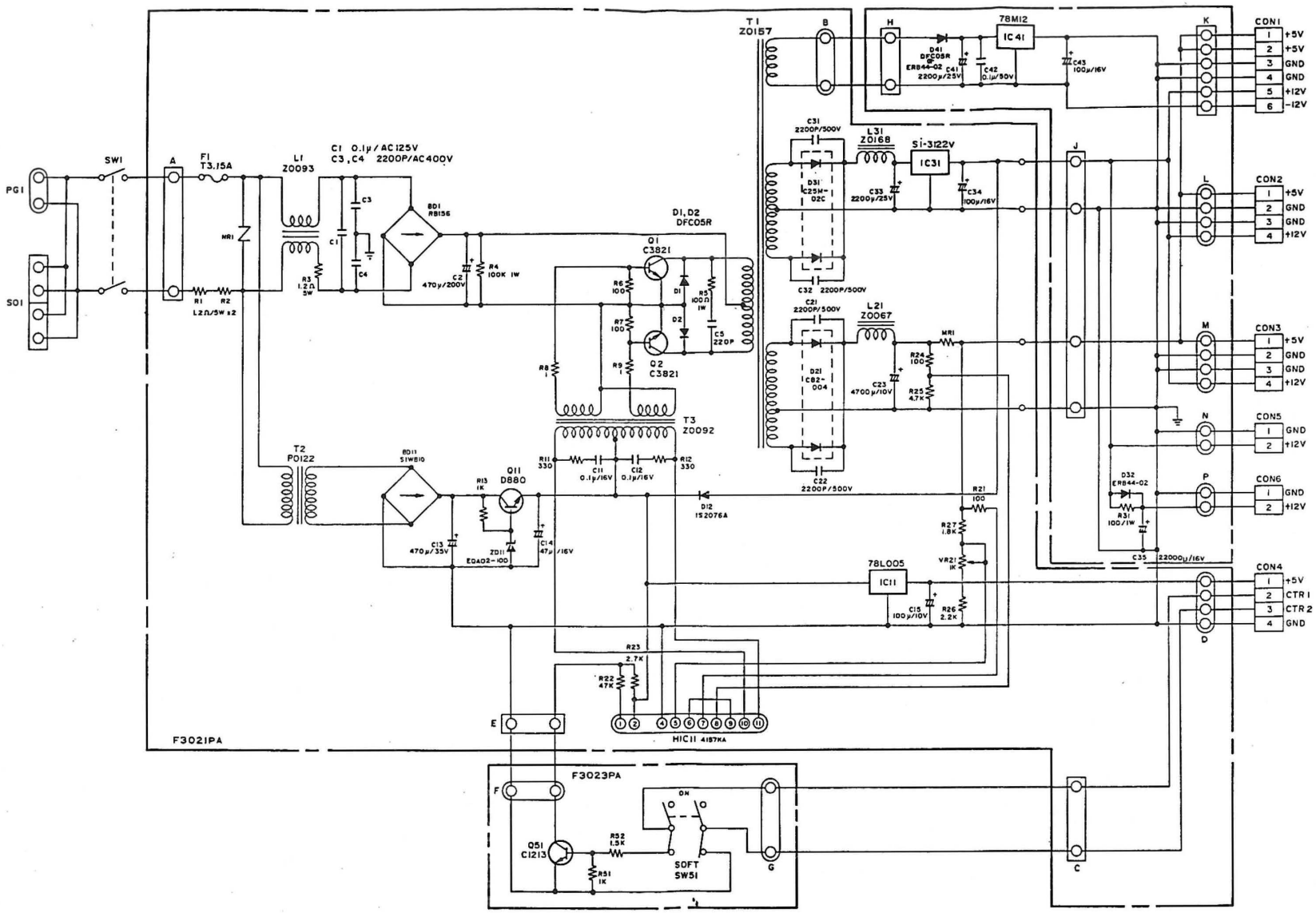
CNS13(上段左) ↔ CN 7 (CMT 2)
(⑧4 H)
CNS12(中段左) ↔ CN 9 (CMT 1)
(⑧4 F)

この基板と⑲のワンチップマイコンの基板とは

CNP 9 (上段左) ↔ CNS 8 (下段中央)
CNP 8 (中段左) ↔ CNS 8 (上段右)
⑲の基板と本体とは
CNS 3 (上段左) ↔ CN 5 (CMT 4)
(⑧1 H～I)
CNS 2 (上段中央) ↔ CN 6 (CMT3)
(⑧2 H～I)

で，それぞれ接続されています。また，この基板のCNS16(左下)と，⑧のCN 8 (CMT 5)，(4 A) とは同じものです。

㉑ 電源部

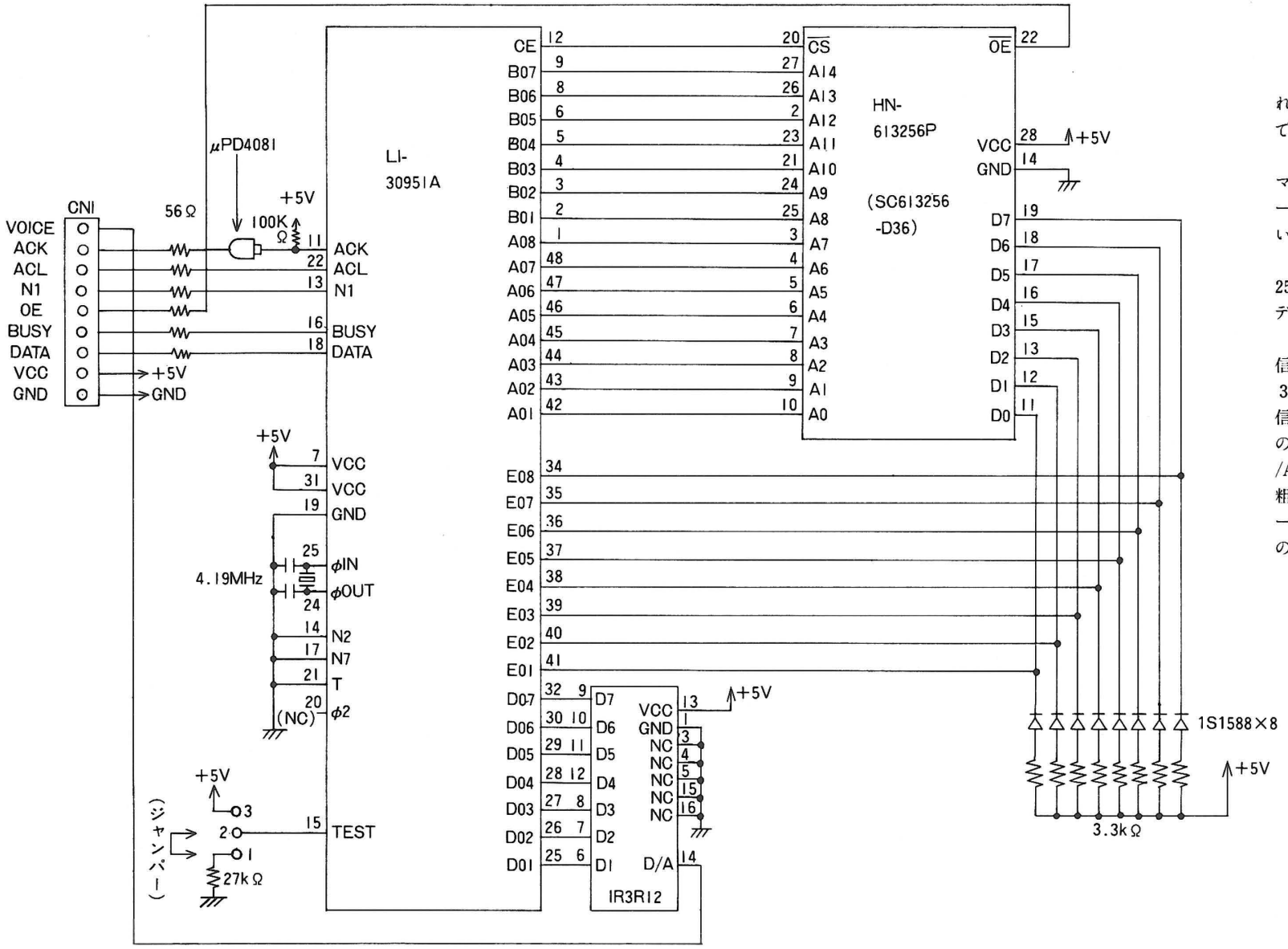


スーパーMZの電源には，効率の良いスイッチングレギュレータが使われています（このクラスでは当然ですが）。つまり，ほとんどすべてのIC（TTL）・LSIの電源である5Vの使用電力に合わせて，交流100Vを消費するわけです。

また，スイッチングレギュレータでは必ずスイッチング用の別電源が必要ですが，スーパーMZではこれ（T2 PO 122やBD17 S1W 810やQ11 D880等で作られる）を本体の一部，つまり⑩で説明した部分に独立して供給し，パワーセレクトスイッチ＝リモート（SOFT SW51＝OFF）でスタンバイ時にはCON4のCTR1＝Hレベルとなり，この図の下部のトランジスタをON状態にしてHIC11 4157KAというICの①をLレベルにし，他へは電源を供給しません。しかし，電話着信時はCTRL1＝Lレベルになり，トランジスタはパワーセレクトスイッチ＝ノーマル（SOFT・SW51＝ON）と時の同様，トランジスタはOFF状態になるため，ICの①がHになってシステム全体がONになります。

なお，SW1（メインパワースイッチ）がOFFの時は，スイッチング用の電源を含め，すべてがOFF状態となるため，モデムホンのリモート信号は無視されます。

㉒ ボイスボード(MZ-1M08)



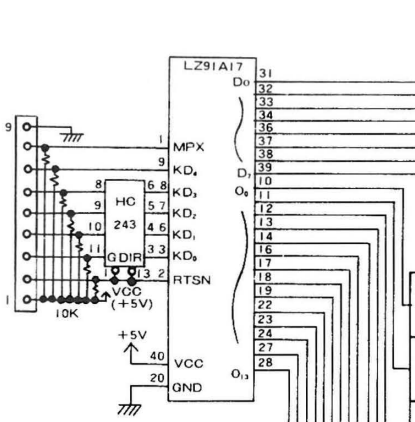
このボードは本来MZ-1500用に開発されたものですが，スーパーMZでも使用できるものです。

LI30951Aというのは一機のワンチップマイコンで，メインCPU（Z80B）とのデータ受け渡しや，音声合成そのものを行います。

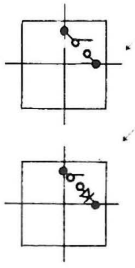
右上には32Kバイトのマスク ROM「23256」（SC613256）があり，ここに音声のデータが入っています。

LI30951Aの出力は7ビットのデジタル信号で行われますが，その右下にあるIR3R12によってD/A変換され，アナログの信号が取出されます。スーパーMZ本体側のアナログディスプレイの回路等にもD/A変換回路はあり，それらは3ビットと粗いので抵抗器で行っていますが，本ボードの場合は7ビットもあるので，精度の高い専用ICを使っています。

㉓ キーボード部



F8	F7	F6	F5	F4	F3	F2	F1
−	+	.	,	9	8	F10	F9
7	6	5	4	3	2	1	0
BREAK	→	←	↓	↑	⏎	(スペース)	TAB
G キ	F ハ	E ィ イ	D シ	C ソ	B コ	A チ	/ ? メ .
O ラ	N ミ	M モ	L リ	K ノ	J マ	I ニ	H ク
W テ	V ヒ	U ナ	T カ	S ト	R ス	Q タ	P セ
, < 、 ネ	. > 。 ル	ー ロ	¥ | ー	^ ヘ	Z ッ ツ	Y ン	X サ
7 ' ャ ヤ	6 & ォ オ	5 % ェ エ	4 $ ゥ ウ	3 # ァ ア	2 " フ	1 ! ヌ	0 ヲ ワ
	[{ ム 「	− @ ゛	− = ホ	; + レ	: * ケ	9) ョ ヨ	8 (ュ ユ
/	*	ESC	⇦	INST DEL	CLR HOME	COPY	] { ム 」
			CTRL	カナ	SHIFT	LOCK	GRAPH
						変換	無変換
						HELP	🎤



個々のキーの様子

SHIFT
CTRL
GRAPH
変換
無変換
キーの場合

LZ-91A17は，本体側のLZ-91A21と一諸になって，実質７本の信号線で97個のキーの状態を転送するために使われています。
SHIFT等のキーは他のキーと同時に押すことが多いので，nキーロールオーバー時のために，逆流防止用のダイオードが入っています(極めて一般的です)。

付録II スーパーMZマシン語活用表

アセンブリ言語	マシン語	ステート数	フラグ変化 SZH P/V NC	働き
ADC A, (HL)	8E	7	↕↕↕V0↕	A+(HL)+Cy→A
ADC A, (IX+nnH)	DD 8E nn	19	↕↕↕V0↕	A+(IX+nnH)+Cy→A
ADC A, (IY+nnH)	FD 8E nn	19	↕↕↕V0↕	A+(IY+nnH)+Cy→A
ADC A, A	8F	4	↕↕↕V0↕	A+A+Cy→A
ADC A, B	88	4	↕↕↕V0↕	A+B+Cy→A
ADC A, C	89	4	↕↕↕V0↕	A+C+Cy→A
ADC A, D	8A	4	↕↕↕V0↕	A+D+Cy→A
ADC A, E	8B	4	↕↕↕V0↕	A+E+Cy→A
ADC A, H	8C	4	↕↕↕V0↕	A+H+Cy→A
ADC A, L	8D	4	↕↕↕V0↕	A+L+Cy→A
ADC A, mmH	CE mm	7	↕↕↕V0↕	A+mmH+Cy→A
ADC HL, BC	ED 4A	15	↕↕↕V0↕	HL+BC+Cy→HL
ADC HL, DE	ED 5A	15	↕↕↕V0↕	HL+DE+Cy→HL
ADC HL, HL	ED 6A	15	↕↕↕V0↕	HL+HL+Cy→HL
ADC HL, SP	ED 7A	15	↕↕↕V0↕	HL+SP+Cy→HL
ADD A, (HL)	86	7	↕↕↕V0↕	A+(HL)→A
ADD A, (IX+nnH)	DD 86 nn	19	↕↕↕V0↕	A+(IX+nnH)→A
ADD A, (IY+nnH)	FD 86 nn	19	↕↕↕V0↕	A+(IY+nnH)→A
ADD A, A	87	4	↕↕↕V0↕	A+A→A
ADD A, B	80	4	↕↕↕V0↕	A+B→A
ADD A, C	81	4	↕↕↕V0↕	A+C→A
ADD A, D	82	4	↕↕↕V0↕	A+D→A
ADD A, E	83	4	↕↕↕V0↕	A+E→A
ADD A, H	84	4	↕↕↕V0↕	A+H→A
ADD A, L	85	4	↕↕↕V0↕	A+L→A
ADD A, mmH	C6 mm	7	↕↕↕V0↕	A+mmH→A
ADD HL, BC	09	11	··↕·0↕	HL+BC→HL
ADD HL, DE	19	11	··↕·0↕	HL+DE→HL
ADD HL, HL	29	11	··↕·0↕	HL+HL→HL
ADD HL, SP	39	11	··↕·0↕	HL+SP→HL
ADD IX, BC	DD 09	15	··↕·0↕	IX+BC→IX
ADD IX, DE	DD 19	15	··↕·0↕	IX+DE→IX
ADD IX, IX	DD 29	15	··↕·0↕	IX+IX→IX
ADD IX, SP	DD 39	15	··↕·0↕	IX+SP→IX
ADD IY, BC	FD 09	15	··↕·0↕	IY+BC→IY
ADD IY, DE	FD 19	15	··↕·0↕	IY+DE→IY
ADD IY, IY	FD 29	15	··↕·0↕	IY+IY→IY
ADD IY, SP	FD 39	15	··↕·0↕	IY+SP→IY
AND (HL)	A6	7	↕↕1P00	A AND (HL)→A
AND (IX+nnH)	DD A6 nn	19	↕↕1P00	A AND (IX+nnH)→A
AND (IY+nnH)	FD A6 nn	19	↕↕1P00	A AND (IY+nnH)→A
AND A	A7	4	↕↕1P00	A AND A→A
AND B	A0	4	↕↕1P00	A AND B→A
AND C	A1	4	↕↕1P00	A AND C→A
AND D	A2	4	↕↕1P00	A AND D→A
AND E	A3	4	↕↕1P00	A AND E→A
AND H	A4	4	↕↕1P00	A AND H→A

アセンブリ言語	マシン語	ステート数	フラグ変化 SZHP/vNC	働き
AND L	A5	4	↕↕1P00	A AND L→A
AND mmH	E6 mm	7	↕↕1P00	A AND mmH→A
BIT 0, (HL)	CB 46	12	?↕1?0・	(HL) のb0が0ならZ=1
BIT 0, (IX+nnH)	DD CB nn 46	20	?↕1?0・	(IX+nnH) のb0が0ならZ=1
BIT 0, (IY+nnH)	FD CB nn 46	20	?↕1?0・	(IY+nnH) のb0が0ならZ=1
BIT 0, A	CB 47	8	?↕1?0・	Aのb0が0ならZ=1
BIT 0, B	CB 40	8	?↕1?0・	Bのb0が0ならZ=1
BIT 0, C	CB 41	8	?↕1?0・	Cのb0が0ならZ=1
BIT 0, D	CB 42	8	?↕1?0・	Dのb0が0ならZ=1
BIT 0, E	CB 43	8	?↕1?0・	Eのb0が0ならZ=1
BIT 0, H	CB 44	8	?↕1?0・	Hのb0が0ならZ=1
BIT 0, L	CB 45	8	?↕1?0・	Lのb0が0ならZ=1
BIT 1, (HL)	CB 4E	12	?↕1?0・	(HL) のb1が0ならZ=1
BIT 1, (IX+nnH)	DD CB nn 4E	20	?↕1?0・	(IX+nnH) のb1が0ならZ=1
BIT 1, (IY+nnH)	FD CB nn 4E	20	?↕1?0・	(IY+nnH) のb1が0ならZ=1
BIT 1, A	CB 4F	8	?↕1?0・	Aのb1が0ならZ=1
BIT 1, B	CB 48	8	?↕1?0・	Bのb1が0ならZ=1
BIT 1, C	CB 49	8	?↕1?0・	Cのb1が0ならZ=1
BIT 1, D	CB 4A	8	?↕1?0・	Dのb1が0ならZ=1
BIT 1, E	CB 4B	8	?↕1?0・	Eのb1が0ならZ=1
BIT 1, H	CB 4C	8	?↕1?0・	Hのb1が0ならZ=1
BIT 1, L	CB 4D	8	?↕1?0・	Lのb1が0ならZ=1
BIT 2, (HL)	CB 56	12	?↕1?0・	(HL) のb2が0ならZ=1
BIT 2, (IX+nnH)	DD CB nn 56	20	?↕1?0・	(IX+nnH) のb2が0ならZ=1
BIT 2, (IY+nnH)	FD CB nn 56	20	?↕1?0・	(IY+nnH) のb2が0ならZ=1
BIT 2, A	CB 57	8	?↕1?0・	Aのb2が0ならZ=1
BIT 2, B	CB 50	8	?↕1?0・	Bのb2が0ならZ=1
BIT 2, C	CB 51	8	?↕1?0・	Cのb2が0ならZ=1
BIT 2, D	CB 52	8	?↕1?0・	Dのb2が0ならZ=1
BIT 2, E	CB 53	8	?↕1?0・	Eのb2が0ならZ=1
BIT 2, H	CB 54	8	?↕1?0・	Hのb2が0ならZ=1
BIT 2, L	CB 55	8	?↕1?0・	Lのb2が0ならZ=1
BIT 3, (HL)	CB 5E	12	?↕1?0・	(HL) のb3が0ならZ=1
BIT 3, (IX+nnH)	DD CB nn 5E	20	?↕1?0・	(IX+nnH) のb3が0ならZ=1
BIT 3, (IY+nnH)	FD CB nn 5E	20	?↕1?0・	(IY+nnH) のb3が0ならZ=1
BIT 3, A	CB 5F	8	?↕1?0・	Aのb3が0ならZ=1
BIT 3, B	CB 58	8	?↕1?0・	Bのb3が0ならZ=1
BIT 3, C	CB 59	8	?↕1?0・	Cのb3が0ならZ=1
BIT 3, D	CB 5A	8	?↕1?0・	Dのb3が0ならZ=1
BIT 3, E	CB 5B	8	?↕1?0・	Eのb3が0ならZ=1
BIT 3, H	CB 5C	8	?↕1?0・	Hのb3が0ならZ=1
BIT 3, L	CB 5D	8	?↕1?0・	Lのb3が0ならZ=1
BIT 4, (HL)	CB 66	12	?↕1?0・	(HL) のb4が0ならZ=1
BIT 4, (IX+nnH)	DD CB nn 66	20	?↕1?0・	(IX+nnH) のb4が0ならZ=1
BIT 4, (IY+nnH)	FD CB nn 66	20	?↕1?0・	(IY+nnH) のb4が0ならZ=1
BIT 4, A	CB 67	8	?↕1?0・	Aのb4が0ならZ=1
BIT 4, B	CB 60	8	?↕1?0・	Bのb4が0ならZ=1
BIT 4, C	CB 61	8	?↕1?0・	Cのb4が0ならZ=1

アセンブリ言語	マシン語	ステート数	フラグ変化 SZHP/VNC	働き
BIT 4, D	CB 62	8	?↕1?0・	Dのb4が0ならZ=1
BIT 4, E	CB 63	8	?↕1?0・	Eのb4が0ならZ=1
BIT 4, H	CB 64	8	?↕1?0・	Hのb4が0ならZ=1
BIT 4, L	CB 65	8	?↕1?0・	Lのb4が0ならZ=1
BIT 5, (HL)	CB 6E	12	?↕1?0・	(HL) のb5が0ならZ=1
BIT 5, (IX+nnH)	DD CB nn 6E	20	?↕1?0・	(IX+nnH) のb5が0ならZ=1
BIT 5, (IY+nnH)	FD CB nn 6E	20	?↕1?0・	(IY+nnH) のb5が0ならZ=1
BIT 5, A	CB 6F	8	?↕1?0・	Aのb5が0ならZ=1
BIT 5, B	CB 68	8	?↕1?0・	Bのb5が0ならZ=1
BIT 5, C	CB 69	8	?↕1?0・	Cのb5が0ならZ=1
BIT 5, D	CB 6A	8	?↕1?0・	Dのb5が0ならZ=1
BIT 5, E	CB 6B	8	?↕1?0・	Eのb5が0ならZ=1
BIT 5, H	CB 6C	8	?↕1?0・	Hのb5が0ならZ=1
BIT 5, L	CB 6D	8	?↕1?0・	Lのb5が0ならZ=1
BIT 6, (HL)	CB 76	12	?↕1?0・	(HL) のb6が0ならZ=1
BIT 6, (IX+nnH)	DD CB nn 76	20	?↕1?0・	(IX+nnH) のb6が0ならZ=1
BIT 6, (IY+nnH)	FD CB nn 76	20	?↕1?0・	(IY+nnH) のb6が0ならZ=1
BIT 6, A	CB 77	8	?↕1?0・	Aのb6が0ならZ=1
BIT 6, B	CB 70	8	?↕1?0・	Bのb6が0ならZ=1
BIT 6, C	CB 71	8	?↕1?0・	Cのb6が0ならZ=1
BIT 6, D	CB 72	8	?↕1?0・	Dのb6が0ならZ=1
BIT 6, E	CB 73	8	?↕1?0・	Eのb6が0ならZ=1
BIT 6, H	CB 74	8	?↕1?0・	Hのb6が0ならZ=1
BIT 6, L	CB 75	8	?↕1?0・	Lのb6が0ならZ=1
BIT 7, (HL)	CB 7E	12	?↕1?0・	(HL) のb7が0ならZ=1
BIT 7, (IX+nnH)	DD CB nn 7E	20	?↕1?0・	(IX+nnH) のb7が0ならZ=1
BIT 7, (IY+nnH)	FD CB nn 7E	20	?↕1?0・	(IY+nnH) のb7が0ならZ=1
BIT 7, A	CB 7F	8	?↕1?0・	Aのb7が0ならZ=1
BIT 7, B	CB 78	8	?↕1?0・	Bのb7が0ならZ=1
BIT 7, C	CB 79	8	?↕1?0・	Cのb7が0ならZ=1
BIT 7, D	CB 7A	8	?↕1?0・	Dのb7が0ならZ=1
BIT 7, E	CB 7B	8	?↕1?0・	Eのb7が0ならZ=1
BIT 7, H	CB 7C	8	?↕1?0・	Hのb7が0ならZ=1
BIT 7, L	CB 7D	8	?↕1?0・	Lのb7が0ならZ=1
CALL C, nnmmH	DC mm nn	17/10	・・・・・・	Cy=1 (セット) ならCALL
CALL M, nnmmH	FC mm nn	17/10	・・・・・・	S=1 (セット) ならCALL
CALL NC, nnmmH	D4 mm nn	17/10	・・・・・・	Cy=0 (リセット) ならCALL
CALL NZ, nnmmH	C4 mm nn	17/10	・・・・・・	Z=0 (リセット) ならCALL
CALL P, nnmmH	F4 mm nn	17/10	・・・・・・	S=0 (リセット) ならCALL
CALL PE, nnmmH	EC mm nn	17/10	・・・・・・	P・V=1 (セット) ならCALL
CALL PO, nnmmH	E4 mm nn	17/10	・・・・・・	P・V= (リセット) ならCALL
CALL Z, nnmmH	CC mm nn	17/10	・・・・・・	Z=1 (セット) ならCALL
CALL nnmmH	CD mm nn	17	・・・・・・	無条件にサブルーチンを呼び出す
CCF	3F	4	・・↕・0↕	Cy XOR 1→Cy (Cyフラグを反転。Hフラグは実行前のCyと同じ)
CP (HL)	BE	7	↕↕↕V1↕	A− (HL) →結果は捨てる
CP (IX+nnH)	DD BE nn	19	↕↕↕V1↕	A− (IX+nnH) →結果は捨てる
CP (IY+nnH)	FD BE nn	19	↕↕↕V1↕	A− (IY+nnH) →結果は捨てる

アセンブリ言語	マシン語	ステート数	フラグ変化 SZHP/VNC	働き
CP A	BF	4	↕↕↕V1↕	A－A→結果は捨てる
CP B	B8	4	↕↕↕V1↕	A－B→結果は捨てる
CP C	B9	4	↕↕↕V1↕	A－C→結果は捨てる
CP D	BA	4	↕↕↕V1↕	A－D→結果は捨てる
CP E	BB	4	↕↕↕V1↕	A－E→結果は捨てる
CP H	BC	4	↕↕↕V1↕	A－H→結果は捨てる
CP L	BD	4	↕↕↕V1↕	A－L→結果は捨てる
CP mmH	FE mm	7	↕↕↕V1↕	A－mm→結果は捨てる
CPD	ED A9	16	↕↕↕↕1・	A－(HL)→結果は捨てる・HL－1→HL・BC－1→BC・BC≠0なら1→P/Vフラグ
CPDR	ED B9	16/21	↕↕↕↕1・	A－(HL)→結果は捨てる・HL－1→HL・BC－1→BC・BC≠0なら1→P/Vフラグ・これをBC＝0かA＝(HL)になるまで繰り返す
CPI	ED A1	16	↕↕↕↕1・	A－(HL)→結果は捨てる・HL＋1→HL・BC－1→BC・BC≠0なら1→P/Vフラグ
CPIR	ED B1	16/21	↕↕↕↕1・	A－(HL)→結果は捨てる・HL＋1→HL・BC－1→BC・BC≠0なら1→P/Vフラグ・これをBC＝0かA＝(HL)になるまで繰り返す
CPL	2F	4	・・1・1・	A XOR FFH→A (全ビット反転)
DAA	27	4	↕↕↕P・↕	直前の演算が10進数同士とみなし補正
DEC (HL)	35	11	↕↕↕V1・	(HL)－1→(HL)
DEC (IX+nnH)	DD 35 nn	23	↕↕↕V1・	(IX+nnH)－1→(IX+nnH)
DEC (IY+nnH)	FD 35 nn	23	↕↕↕V1・	(IY+nnH)－1→(1Y+nnH)
DEC A	3D	4	↕↕↕V1・	A－1→A
DEC B	05	4	↕↕↕V1・	B－1→B
DEC BC	0B	6	・・・・・・	BC－1→BC
DEC C	0D	4	↕↕↕V1・	C－1→C
DEC D	15	4	↕↕↕V1・	D－1→D
DEC DE	1B	6	・・・・・・	DE－1→DE
DEC E	1D	4	↕↕↕V1・	E－1→E
DEC H	25	4	↕↕↕V1・	H－1→H
DEC HL	2B	6	・・・・・・	HL－1→HL
DEC IX	DD 2B	10	・・・・・・	IX－1→IX
DEC IY	FD 2B	10	・・・・・・	IY－1→IY
DEC L	2D	4	↕↕↕V1・	L－1→L
DEC SP	3B	10	・・・・・・	SP－1→SP
DI	F3	4	・・・・・・	割り込みを禁止する (0→IFF)
DJNZ $+2+nnH	10 nn	13/8	・・・・・・	B－1→B・B≠0なら相対ジャンプ
EI	FB	4	・・・・・・	割り込みを許可する (1→IFF)
EX (SP), HL	E3	19	・・・・・・	スタックトップとHLの値を入れ換える
EX (SP), IX	DD E3	23	・・・・・・	スタックトップとIXの値を入れ換える
EX (SP), IY	FD E3	23	・・・ ・・	スタックトップとIYの値を入れ換える
EX AF, AF'	08	4	(F' の値)	AとA', FとF' の値を入れ換える
EX DE, HL	EB	4	・・・・・・	DとH, EとLの値を入れ換える

アセンブリ言語	マシン語	ステート数	フラグ変化 SZHP/vNC	働き
EXX	D9	4	・・・・・・	BとB', C C', DとD', EとE', HとH', LとL' の値を入れ換える
HALT	76	4	・・・・・・	割り込みかリセットがかかるまで停止
IM 0	ED 46	8	・・・・・・	モード0割り込みを行う(割り込み時RSTかCALLを周辺側が発生)
IM 1	ED 56	8	・・・・・・	モード1割り込みを行う(割り込み時自動的にRST 38Hを実行)
IM 2	ED 5E	8	・・・・・・	モード2割り込みを行う(テーブル参照型。MZ−2500はこのモード)
IN A, (C)	ED 78	12	↕↕0P0・	Cで示すポートからの入力値→A
IN A, (mmH)	DB mm	11	・・・・・・	ポートmmHからの入力値→A
IN B, (C)	ED 40	12	↕↕0P0・	Cで示すポートからの入力値→B
IN C, (C)	ED 48	12	↕↕0P0・	Cで示すポートからの入力値→C
IN D, (C)	ED 50	12	↕↕0P0・	Cで示すポートからの入力値→D
IN E, (C)	ED 58	12	↕↕0P0・	Cで示すポートからの入力値→E
IN H, (C)	ED 60	12	↕↕0P0・	Cで示すポートからの入力値→H
IN L, (C)	ED 68	12	↕↕0P0・	Cで示すポートからの入力値→L
INC (HL)	34	11	↕↕↕V0・	(HL) +1→ (HL)
INC (IX+nnH)	DD 34 nn	23	↕↕↕V0・	(IX+nnH) +1→ (IX+nnH)
INC (IY+nnH)	FD 34 nn	23	↕↕↕V0・	(IY+nnH) +1→ (IY+nnH)
INC A	3C	4	↕↕↕V0・	A+1→A
INC B	04	4	↕↕↕V0・	B+1→B
INC BC	03	6	・・・・・・	BC+1→BC
INC C	0C	4	↕↕↕V0・	C+1→C
INC D	14	4	↕↕↕V0・	D+1→D
INC DE	13	6	・・・・・・	DE+1→DE
INC E	1C	4	↕↕↕V0・	E+1→E
INC H	24	4	↕↕↕V0・	H+1→H
INC HL	23	6	・・・・・・	HL+1→HL
INC IX	DD 23	10	・・・・・・	IX+1→IX
INC IY	FD 23	10	・・・・・・	IY+1→IY
INC L	2C	4	↕↕↕V0・	L+1→L
INC SP	33	10	・・・・・・	SP+1→SP
IND	ED AA	16	?↕??1・	Cで示すポートからの入力値→(HL)・HL−1→HL・B−1→B・B≠0なら0→Zフラグ
INDR	ED BA	16／21	?1??1・	Cで示すポートからの入力値→(HL)・HL−1→HL・B−1→B・これをB=0になるまで繰り返す
INI	ED A2	16	?↕??1・	Cで示すポートからの入力値→(HL)・HL+1→HL・B−1→B・B≠0なら0→Zフラグ
INIR	ED B2	16／21	?1??1・	Cで示すポートからの入力値→(HL)・HL+1→HL・B−1→B・これをB=0になるまで繰り返す
JP (HL)	E9	4	・・・・・・	HL→PC。つまりHL番地にジャンプ
JP (IX)	DD E9	8	・・・・・・	IX→PC。つまりIX番地にジャンプ
JP (IY)	FD E9	8	・・・・・・	IY→PC。つまりIY番地にジャンプ

アセンブリ言語	マシン語	ステート数	フラグ変化 SZH$P/_V$NC	働き
JP C, nnmmH	DA mm nn	10	・・・・・・	Cy＝1（セット）ならジャンプ
JP M, nnmmH	FA mm nn	10	・・・・・・	S＝1（セット）ならジャンプ
JP NC, nnmmH	D2 mm nn	10	・・・・・・	Cy＝0（リセット）ならジャンプ
JP NZ, nnmmH	C2 mm nn	10	・・・・・・	Z＝0（リセット）ならジャンプ
JP P, nnmmH	F2 mm nn	10	・・・・・・	S＝0（リセット）ならジャンプ
JP PE, nnmmH	EA mm nn	10	・・・・・・	P・V＝1（セット）ならジャンプ
JP PO, nnmmH	E2 mm nn	10	・・・・・・	P・V＝0（リセット）ならジャンプ
JP Z, nnmmH	CA mm nn	10	・・・・・・	Z＝1（セット）ならジャンプ
JP nnmmH	C3 mm nn	17	・・・・・・	無条件にジャンプ（nnmmH→PC）
JR C, $+2+nnH	38 nn	12／7	・・・・・・	Cy＝1（セット）なら相対ジャンプ
JR NC, $+2+nnH	30 nn	12／7	・・・・・・	Cy＝0（リセット）なら相対ジャンプ
JR NZ, $+2+nnH	20 nn	12／7	・・・・・・	Z＝1（リセット）なら相対ジャンプ
JR Z, $+2+nnH	28 nn	12／7	・・・・・・	Z＝1（セット）なら相対ジャンプ
JR $+2+nnH	18 nn	12	・・・・・・	無条件に相対ジャンプ
LD (BC), A	02	7	・・・・・・	A→（BC）
LD (DE), A	12	7	・・・・・・	A→（DE）
LD (HL), A	77	7	・・・・・・	A→（HL）
LD (HL), B	70	7	・・・・・・	B→（HL）
LD (HL), C	71	7	・・・・・・	C→（HL）
LD (HL), D	72	7	・・・・・・	D→（HL）
LD (HL), E	73	7	・・・・・・	E→（HL）
LD (HL), H	74	7	・・・・・・	H→（HL）
LD (HL), L	75	7	・・・・・・	L→（HL）
LD (HL), mmH	36 mm	10	・・・・・・	mmH→（HL）
LD (IX+nnH), A	DD 77 nn	19	・・・・・・	A→（IX＋nnH）
LD (IX+nnH), B	DD 70 nn	19	・・・・・・	B→（IX＋nnH）
LD (IX+nnH), C	DD 71 nn	19	・・・・・・	C→（IX＋nnH）
LD (IX+nnH), D	DD 72 nn	19	・・・・・・	D→（IX＋nnH）
LD (IX+nnH), E	DD 73 nn	19	・・・・・・	E→（IX＋nnH）
LD (IX+nnH), H	DD 74 nn	19	・・・・・・	H→（IX＋nnH）
LD (IX+nnH), L	DD 75 nn	19	・・・・・・	L→（IX＋nnH）
LD (IX+nnH), mmH	DD 36 nn mm	19	・・・・・・	mmH→（IX＋nnH）
LD (IY+nnH), A	FD 77 nn	19	・・・・・・	A→（IY＋nnH）
LD (IY+nnH), B	FD 70 nn	19	・・・・・・	B→（IY＋nnH）
LD (IY+nnH), C	FD 71 nn	19	・・・・・・	C→（IY＋nnH）
LD (IY+nnH), D	FD 72 nn	19	・・・・・・	D→（IY＋nnH）
LD (IY+nnH), E	FD 73 nn	19	・・・・・・	E→（IY＋nnH）
LD (IY+nnH), H	FD 74 nn	19	・・・・・・	H→（IY＋nnH）
LD (IY+nnH), L	FD 75 nn	19	・・・・・・	L→（IY＋nnH）
LD (IY+nnH), mmH	FD 36 nn mm	19	・・・・・・	mmH→（IY＋nnH）
LD (nnmmH), A	32 mm nn	13	・・・・・・	A→（nnmmH）
LD (nnmmH), BC	ED 43 mm nn	20	・・・・・・	C→(nnmmH)・B→(nnmmH+1)
LD (nnmmH), DE	ED 53 mm nn	20	・・・・・・	E→(nnmmH)・D→(nnmmH+1)
LD (nnmmH), HL	22 mm nn	16	・・・・・・	L→(nnmmH)・H→(nnmmH+1)
LD (nnmmH), HL	ED 63 mm nn	20	・・・・・・	L→(nnmmH)・H→(nnmmH+1)
LD (nnmmH), IX	DD 22 mm nn	20	・・・・・・	IXの下位8ビット→（nnmmH） IXの上位8ビット→（nnmmH＋1）

アセンブリ言語	マシン語	ステート数	フラグ変化 SZH P/V NC	働き
LD (nnmmH), IY	FD 22 mm nn	20	・・・・・・	IYの下位8ビット→(nnmmH) IYの上位8ビット→(nnmmH+1)
LD (nnmmH), SP	ED 73 mm nn	20	・・・・・・	SPの下位8ビット→(nnmmH) SPの上位8ビット→(nnmmH+1)
LD A, (BC)	0A	7	・・・・・・	(BC)→A
LD A, (DE)	1A	7	・・・・・・	(DE)→A
LD A, (HL)	7E	7	・・・・・・	(HL)→A
LD A, (IX+nnH)	DD 7E nn	19	・・・・・・	(IX+nnH)→A
LD A, (IY+nnH)	FD 7E nn	19	・・・・・・	(IY+nnH)→A
LD A, (nnmmH)	3A mm nn	13	・・・・・・	(nnmmH)→A
LD A, A	7F	4	・・・・・・	A→A
LD A, B	78	4	・・・・・・	B→A
LD A, C	79	4	・・・・・・	C→A
LD A, D	7A	4	・・・・・・	D→A
LD A, E	7B	4	・・・・・・	E→A
LD A, H	7C	4	・・・・・・	H→A
LD A, I	ED 57	9	↕↕0↕0・	I→A・IFF2→P／Vフラグ
LD A, L	7D	4	・・・・・・	L→A
LD A, R	ED 5F	9	↕↕0↕0・	R→A・IFF2→P／Vフラグ
LD A, mmH	3E mm	7	・・・・・・	mmH→A
LD B, (HL)	46	7	・・・・・・	(HL)→B
LD B, (IX+nnH)	DD 46 nn	19	・・・・・・	(IX+nnH)→B
LD B, (IY+nnH)	FD 46 nn	19	・・・・・・	(IY+nnH)→B
LD B, A	47	4	・・・・・・	A→B
LD B, B	40	4	・・・・・・	B→B
LD B, C	41	4	・・・・・・	C→B
LD B, D	42	4	・・・・・・	D→B
LD B, E	43	4	・・・・・・	E→B
LD B, H	44	4	・・・・・・	H→B
LD B, L	45	4	・・・・・・	L→B
LD B, mmH	06 mm	7	・・・・・・	mmH→B
LD BC, (nnmmH)	ED 4B mm nn	20	・・・・・・	(nnmmH)→C・(nnmmH+1)→B
LD BC, nnmmH	01 mm nn	10	・・・・・・	mmH→C・nnH→B
LD C, (HL)	4E	7	・・・・・・	(HL)→C
LD C, (IX+nnH)	DD 4E nn	19	・・・・・・	(IX+nnH)→C
LD C, (IY+nnH)	FD 4E nn	19	・・・・・・	(IY+nnH)→C
LD C, A	4F	4	・・・・・・	A→C
LD C, B	48	4	・・・・・・	B→C
LD C, C	49	4	・・・・・・	C→C
LD C, D	4A	4	・・・・・・	D→C
LD C, E	4B	4	・・・・・・	E→C
LD C, H	4C	4	・・・・・・	F→C
LD C, L	4D	4	・・・・・・	L→C
LD C, mmH	0E mm	7	・・・・・・	mmH→C
LD D, (HL)	56	7	・・・・・・	(HL)→D
LD D, (IX+nnH)	DD 56 nn	19	・・・・・・	(IX+nnH)→D
LD D, (IY+nnH)	FD 56 nn	19	・・・・・・	(IY+nnH)→D
LD D, A	57	4	・・・・・・	A→D

アセンブリ言語	マシン語	ステート数	フラグ変化 SZH$^{P}_{V}$NC	働き
LD D, B	50	4	・・・・・・	B→D
LD D, C	51	4	・・・・・・	C→D
LD D, D	52	4	・・・・・・	D→D
LD D, E	53	4	・・・・・・	E→D
LD D, H	54	4	・・・・・・	H→D
LD D, L	55	4	・・・・・・	L→D
LD D, mmH	16 mm	7	・・・・・・	mmH→D
LD DE, (nnmmH)	ED 5B mm nn	20	・・・・・・	(nnmmH)→E・(nnmmH+1)→D
LD DE, nnmmH	11 mm nn	10	・・・・・・	mmH→E・nnH→D
LD E, (HL)	5E	7	・・・・・・	(HL)→E
LD E, (IX+nnH)	DD 5E nn	19	・・・・・・	(IX+nnH)→E
LD E, (IY+nnH)	FD 5E nn	19	・・・・・・	(IY+nnH)→E
LD E, A	5F	4	・・・・・・	A→E
LD E, B	58	4	・・・・・・	B→E
LD E, C	59	4	・・・・・・	C→E
LD E, D	5A	4	・・・・・・	D→E
LD E, E	5B	4	・・・・・・	E→E
LD E, H	5C	4	・・・・・・	H→E
LD E, L	5D	4	・・・・・・	L→E
LD E, mmH	1E mm	7	・・・・・・	mmH→E
LD H, (HL)	66	7	・・・・・・	(HL)→H
LD H, (IX+nnH)	DD 66 nn	19	・・・・・・	(IX+nnH)→H
LD H, (IY+nnH)	FD 66 nn	19	・・・・・・	(IY+nnH)→H
LD H, A	67	4	・・・・・・	A→H
LD H, B	60	4	・・・・・・	B→H
LD H, C	61	4	・・・・・・	C→H
LD H, D	62	4	・・・・・・	D→H
LD H, E	63	4	・・・・・・	E→H
LD H, H	64	4	・・・・・・	H→H
LD H, L	65	4	・・・・・・	L→H
LD H, mmH	26 mm	7	・・・・・・	mmH→H
LD HL, (nnmmH)	2A mm nn	16	・・・・・・	(nnmmH)→L・(nnmmH+1)→H
LD HL, (nnmmH)	ED 6B mm nn	20	・・・・・・	(nnmmH)→L・(nnmmH+1)→H
LD HL, nnmmH	21 mm nn	10	・・・・・・	mmH→L・nnH→H
LD I, A	ED 47	9	・・・・・・	A→I
LD IX, (nnmmH)	DD 2A mm nn	20	・・・・・・	(nnmmH)→IXの下位8ビット (nnmmH+1)→IXの上位8ビット
LD IX, nnmmH	DD 21 mm nn	14	・・・・・・	nnmmH→IX
LD IY, (nnmmH)	FD 2A mm nn	7	・・・・・・	(nnmmH)→IYの下位8ビット (nnmmH+1)→IYの上位8ビット
LD IY, nnmmH	FD 21 mm nn	14	・・・・・・	nnmmH→IY
LD L, (HL)	6E	7	・・・・・・	(HL)→L
LD L, (IX+nnH)	DD 6E nn	19	・・・・・・	(IX+nnH)→L
LD L, (IY+nnH)	FD 6E nn	19	・・・・・・	(IY+nnH)→L
LD L, A	6F	4	・・・・・・	A→L
LD L, B	68	4	・・・・・・	B→L
LD L, C	69	4	・・・・・・	C→L
LD L, D	6A	4	・・・・・・	D→L

アセンブリ言語	マシン語	ステー一数	フラグ変化 SZH P/V NC	働き
LD L, E	6B	4	・・・・・・	E→L
LD L, H	6C	4	・・・・・・	H→L
LD L, L	6D	4	・・・・・・	L→L
LD L, mmH	2E mm	7	・・・・・・	mmH→L
LD R, A	ED 4F	9	・・・・・・	A→R
LD SP, (nnmmH)	ED 7B mm nn	20	・・・・・・	(nnmmH) →SPの下位8ビット (nnmmH+1) →SPの上位8ビット
LD SP, HL	F9	6	・・・・・・	HL→SP (L→SPの下位8ビット・H→SPの上位8ビット)
LD SP, IX	DD F9	10	・・・・・・	IX→SP
LD SP, IY	FD F9	10	・・・・・・	IY→SP
LD SP, nnmmH	31 mm nn	10	・・・・・・	nnmmH→SP
LDD	ED A8	16	・・0↕0・	(HL) → (DE) ・HL−1→HL ・DE−1→DE・BC−1→BC・ BC≠0の時は1→P／Vフラグ
LDDR	ED B8	16／21	・・000・	(HL) → (DE) ・HL−1→HL ・DE−1→DE・BC−1→BC・ これをBC=0になるまで繰り返す
LDI	ED A0	16	・・0↕0・	(HL) → (DE) ・HL+1→HL ・DE+1→DE・BC−1→BC・ BC≠0の時は1→P／Vフラグ
LDIR	ED B0	16／21	・・000・	(HL) → (DE・BC−1→→HL ・DE+1→DE・BC−1→BC・ これをBC=0になるまで繰り返す
NEG	ED 44	8	↕↕↕V1↕	0−A→A
NOP	00	4	・・・・・・	特に何もしない
OR (HL)	B6	7	↕↕0P00	A OR (HL) →A
OR (IX+nnH)	DD B6 nn	19	↕↕0P00	A OR (IX+nnH) →A
OR (IY+nnH)	FD B6 nn	19	↕↕0P00	A OR (IY+nnH) →A
OR A	B7	4	↕↕0P00	A OR A→A
OR B	B0	4	↕↕0P00	A OR B→A
OR C	B1	4	↕↕0P00	A OR C→A
OR D	B2	4	↕↕0P00	A OR D→A
OR E	B3	4	↕↕0P00	A OR E→A
OR H	B4	4	↕↕0P00	A OR H→A
OR L	B5	4	↕↕0P00	A OR L→A
OR mmH	F6 mm	7	↕↕0P00	A OR mmH→A
OTDR	ED BB	16／21	?1??1・	(HL) →Cで示すポートへ出力・HL+1→HL・B−1→B・これをB=0になるまで繰り返す
OTIR	ED B3	16／21	?1??1・	(HL) →Cで示すポートへ出力・HL−1→HL・B−1→B・これをB=0になるまで繰り返す
OUT (C), A	ED 79	12	・・・・・・	A→Cで示すポートへ出力
OUT (C), B	ED 41	12	・・・・・・	B→Cで示すポートへ出力
OUT (C), C	ED 49	12	・・・・・・	C→Cで示すポートへ出力
OUT (C), D	ED 51	12	・・・・・・	D→Cで示すポートへ出力
OUT (C), E	ED 59	12	・・・・・・	E→Cで示すポートへ出力

アセンブリ言語	マシン語	ステート数	フラグ変化 SZH P/V NC	働き
OUT (C), H	ED 61	12	・・・・・・	H→Cで示すポートへ出力
OUT (C), L	ED 69	12	・・・・・・	L→Cで示すポートへ出力
OUT (mmH), A	D3 mm	11	・・・・・・	A→ポートmmHへ出力
OUTD	ED AB	16	?↕??1・	(HL)→Cで示すポートへ出力・HL−1→HL・B−1→B・B≠0なら0→Zフラグ
OUTI	ED A3	16	?↕??1・	(HL)→Cで示すポートへ出力・HL+1→HL・B−1→B・B≠0なら0→Zフラグ
POP AF	F1	10	(復帰値)	A・Fを復帰 (SP)→(ペア)レジスタの下位8ビット・SP+1→SP・(SP)→(ペア)レジスタの上位8ビット・SP+1→SP
POP BC	C1	10	・・・・・・	B・Cを復帰
POP DE	D1	10	・・・・・・	D・Eを復帰
POP HL	E1	10	・・・・・・	H・Lを復帰
POP IX	DD E1	14	・・・・・・	IXを復帰
POP IY	FD E1	14	・・・・・・	IYを復帰
PUSH AF	F5	11	・・・・・・	A・Fを退避 SP−1→SP・(ペア)レジスタの上位8ビット→(SP)・SP−1→SP・(ペア)レジスタの下位8ビット→(SP)
PUSH BC	C5	11	・・・・・・	B・Cを退避
PUSH DE	D5	11	・・・・・・	D・Eを退避
PUSH HL	E5	11	・・・・・・	H・Lを退避
PUSH IX	DD E5	15	・・・・・・	IXを退避
PUSH IY	FD E5	15	・・・・・・	IYを退避
RES 0, (HL)	CB 86	15	・・・・・・	(HL)のb0を0にリセット
RES 0, (IX+nnH)	DD CB nn 86	23	・・・・・・	(IX+nnH)のb0をリにリセット
RES 0, (IY+nnH)	FD CB nn 86	23	・・・・・・	(IY+nnH)のb0を0にリセット
RES 0, A	CB 87	8	・・・・・・	Aのb0を0にリセット
RES 0, B	CB 80	8	・・・・・・	Bのb0を0にリセット
RES 0, C	CB 81	8	・・・・・・	Cのb0を0にリセット
RES 0, D	CB 82	8	・・・・・・	Dのb0を0にリセット
RES 0, E	CB 83	8	・・・・・・	Eのb0を0にリセット
RES 0, H	CB 84	8	・・・・・・	Hのb0を0にリセット
RES 0, L	CB 85	8	・・・・・・	Lのb0を0にリセット
RES 1, (HL)	CB 8E	15	・・・・・・	(HL)のb1を0にリセット
RES 1, (IX+nnH)	DD CB nn 8E	23	・・・・・・	(IX+nnH)のb1を0にリセット
RES 1, (IY+nnH)	FD CB nn 8E	23	・・・・・・	(IY+nnH)のb1を0にリセット
RES 1, A	CB 8F	8	・・・・・・	Aのb1を0にリセット
RES 1, B	CB 88	8	・・・・・・	Bのb1を0にリセット
RES 1, C	CB 89	8	・・・・・・	Cのb1を0にリセット
RES 1, D	CB 8A	8	・・・・・・	Dのb1を0にリセット
RES 1, E	CB 8B	8	・・・・・・	Eのb1を0にリセット
RES 1, H	CB 8C	8	・・・・・・	Hのb1を0にリセット
RES 1, L	CB 8D	8	・・・・・・	Lのb1を0にリセット
RES 2, (HL)	CB 96	15	・・・・・・	(HL)のb2を0にリセット
RES 2, (IX+nnH)	DD CB nn 96	23	・・・・・・	(IX+nnH)のb2を0にリセット
RES 2, (IY+nnH)	FD CB nn 96	23	・・・・・・	(IY+nnH)のb2を0にリセット
RES 2, A	CB 97	8	・・・・・・	Aのb2を0にリセット
RES 2, B	CB 90	8	・・・・・・	Bのb2を0にリセット
RES 2, C	CB 91	8	・・・・・・	Cのb2を0にリセット
RES 2, D	CB 92	8	・・・・・・	Dのb2を0にリセット

アセンブリ言語	マシン語	ステート数	フラグ変化 SZH P/V NC	働き
RES 2, E	CB 93	8	・・・・・・	Eのｂ2を0にリセット
RES 2, H	CB 94	8	・・・・・・	Hのｂ2を0にリセット
RES 2, L	CB 95	8	・・・・・・	Lのｂ2を0にリセット
RES 3, (HL)	CB 9E	15	・・・・・・	(HL) のｂ3を0にリセット
RES 3, (IX+nnH)	DD CB nn 9E	23	・・・・・・	(IX+nnH) のｂ3を0にリセット
RES 3, (IY+nnH)	FD CB nn 9E	23	・・・・・・	(IY+nnH) のｂ3を0にリセット
RES 3, A	CB 9F	8	・・・・・・	Aのｂ3を0にリセット
RES 3, B	CB 98	8	・・・・・・	Bのｂ3を0にリセット
RES 3, C	CB 99	8	・・・・・・	Cのｂ3を0にリセット
RES 3, D	CB 9A	8	・・・・・・	Dのｂ3を0にリセット
RES 3, E	CB 9B	8	・・・・・・	Eのｂ3を0にリセット
RES 3, H	CB 9C	8	・・・・・・	Hのｂ3を0にリセット
RES 3, L	CB 9D	8	・・・・・・	Lのｂ3を0にリセット
RES 4, (HL)	CB A6	15	・・・・・・	(HL) のｂ4を0にリセット
RES 4, (IX+nnH)	DD CB nn A6	23	・・・・・・	(IX+nnH) のｂ4を0にリセット
RES 4, (IY+nnH)	FD CB nn A6	23	・・・・・・	(IY+nnH) のｂ4を0にリセット
RES 4, A	CB A7	8	・・・・・・	Aのｂ4を0にリセット
RES 4, B	CB A0	8	・・・・・・	Bのｂ4を0にリセット
RES 4, C	CB A1	8	・・・・・・	Cのｂ4を0にリセット
RES 4, D	CB A2	8	・・・・・・	Dのｂ4を0にリセット
RES 4, E	CB A3	8	・・・・・・	Eのｂ4を0にリセット
RES 4, H	CB A4	8	・・・・・・	Hのｂ4を0にリセット
RES 5, L	CB A5	8	・・・・・・	Lのｂ4を0にリセット
RES 5, (HL)	CB AE	15	・・・・・・	(HL) のｂ5を0にリセット
RES 5, (IX+nnH)	DD CB nn AE	23	・・・・・・	(IX+nnH) のｂ5を0にリセット
RES 5, (IY+nnH)	FD CB nn AE	23	・・・・・・	(IY+nnH) のｂ5を0にリセット
RES 5, A	CB AF	8	・・・・・・	Aのｂ5を0にリセット
RES 5, B	CB A8	8	・・・・・・	Bのｂ5を0にリセット
RES 5, C	CB A9	8	・・・・・・	Cのｂ5を0にリセット
RES 5, D	CB AA	8	・・・・・・	Dのｂ5を0にリセット
RES 5, E	CB AB	8	・・・・・・	Eのｂ5を0にリセット
RES 5, H	CB AC	8	・・・・・・	Hのｂ5を0にリセット
RES 6, L	CB AD	8	・・・・・・	Lのｂ5を0にリセット
RES 6, (HL)	CB B6	15	・・・・・・	(HL) のｂ6を0にリセット
RES 6, (IX+nnH)	DD CB nn B6	23	・・・・・・	(IX+nnH) のｂ6を0にリセット
RES 6, (IY+nnH)	FD CB nn B6	23	・・・・・・	(IY+nnH) のｂ6を0にリセット
RES 6, A	CB B7	8	・・・・・・	Aのｂ6を0にリセット
RES 6, B	CB B0	8	・・・・・・	Bのｂ6を0にリセット
RES 6, C	CB B1	8	・・・・・・	Cのｂ6を0にリセット
RES 6, D	CB B2	8	・・・・・・	Dのｂ6を0にリセット
RES 6, E	CB B3	8	・・・・・・	Eのｂ6を0にリセット
RES 6, H	CB B4	8	・・・・・・	Hのｂ6を0にリセット
RES 7, L	CB B5	8	・・・・・・	Lのｂ6を0にリセット
RES 7, (HL)	CB BE	15	・・・・・・	(HL) のｂ7を0にリセット
RES 7, (IX+nnH)	DD CB nn BE	25	・・・・・・	(IX+nnH) のｂ7を0にリセット
RES 7, (IY+nnH)	FD CB nn BE	23	・・・・・・	(IY+nnH) のｂ7を0にリセット
RES 7, A	CB BF	8	・・・・・・	Aのｂ7を0にリセット
RES 7, B	CB B8	8	・・・・・・	Bのｂ7を0にリセット

アセンブリ言語	マシン語	ステート数	フラグ変化 SZH P/V NC	働き	
RES 7, C	CB B9	8	・・・・・・	Cのb7を0にリセット	
RES 7, D	CB BA	8	・・・・・・	Dのb7を0にリセット	
RES 7, E	CB BB	8	・・・・・・	Eのb7を0にリセット	
RES 7, H	CB BC	8	・・・・・・	Hのb7を0にリセット	
RES 7, L	CB BD	8	・・・・・・	Lのb7を0にリセット	
RET	C9	10	・・・・・・	無条件にリターン	
RET C	D8	11／5	・・・・・・	Cy＝1（セット）ならリターン	
RET M	F8	11／5	・・・・・・	S＝1（セット）ならリターン	
RET NC	D0	11／5	・・・・・・	Cy＝0（リセット）ならリターン	
RET NZ	C0	11／5	・・・・・・	Z＝0（リセット）ならリターン	
RET P	F0	11／5	・・・・・・	S＝0（リセット）ならリターン	
RET PE	E8	11／5	・・・・・・	P・V＝1（セット）ならリターン	
RET PO	E0	11／5	・・・・・・	P・V＝0（リセット）ならリターン	
RET Z	C8	11／5	・・・・・・	Z＝1（セット）ならリターン	
RETI	ED 4D	14	・・・・・・	通常のマスカブル割り込みから復帰する（周辺LSIもそのことを検知できる）	
RETN	ED 45	14	・・・・・・	ノンマスカブル割り込み（MZ－2500では使用していない）から復帰する	
RL (HL)	CB 16	15	↕↕0P0↕	(HL) をCyを含め左回転	
RL (IX+nnH)	DD CB nn 16	23	↕↕0P0↕	(IX+nnH) をCyを含め左回転	b0←Cy b1←b0
RL (IY+nnH)	FD CB nn 16	23	↕↕0P0↕	(IY+nnH) をCyを含め左回転	b2←b1 b3←b2
RL A	CB 17	8	↕↕0P0↕	AをCyを含め左回転	b4←b3
RL B	CB 10	8	↕↕0P0↕	BをCyを含め左回転	b5←b4
RL C	CB 11	8	↕↕0P0↕	CをCyを含め左回転	b6←b5
RL D	CB 12	8	↕↕0P0↕	DをCyを含め左回転	b7←b6
RL E	CB 13	8	↕↕0P0↕	EをCyを含め左回転	Cy←b7
RL H	CB 14	8	↕↕0P0↕	HをCyを含め左回転	
RL L	CB 15	8	↕↕0P0↕	LをCyを含め左回転	
RLA	17	4	・・0・0↕	AをCyを含め左回転	
RLC (HL)	CB 06	15	↕↕0P0↕	(HL) をCyを含めず左回転	
RLC (IX+nnH)	DD CB nn 06	23	↕↕0P0↕	(IX+nnH) をCyを含めず左回転	b0←b7 b1←b0
RLC (IY+nnH)	FD CB nn 06	23	↕↕0P0↕	(IY+nnH) をCyを含めず左回転	b2←b1 b3←b2
RLC A	CB 07	8	↕↕0P0↕	AをCyを含めず左回転	b4←b3
RLC B	CB 00	8	↕↕0P0↕	BをCyを含めず左回転	b5←b4
RLC C	CB 01	8	↕↕0P0↕	CをCyを含めず左回転	b6←b5
RLC D	CB 02	8	↕↕0P0↕	DをCyを含めず左回転	b7←b6
RLC E	CB 03	8	↕↕0P0↕	EをCyを含めず左回転	Cy←b7
RLC H	CB 04	8	↕↕0P0↕	HをCyを含めず左回転	
RLC L	CB 05	8	↕↕0P0↕	LをCyを含めず左回転	
RLCA	07	4	・・0・0↕	AをCyを含めず左回転	

アセンブリ言語	マシン語	ステート数	フラグ変化 SZH P/V NC	働き
RLD	ED 6F	18	↕↕0P0・	Aのb3～b0←(HL)b7～b4 (HL)b7～b4←(HL)b3～b4 (HL)b3～b0←Aのb3～b0
RR (HL)	CB 1E	15	↕↕0P0↕	(HL)をCyを含め右回転
RR (IX+nnH)	DD CB nn 1E	23	↕↕0P0↕	(IX+nnH)をCyを含め右回転
RR (IY+nnH)	FD CB nn 1E	23	↕↕0P0↕	(IY+nnH)をCyを含め右回転
RR A	CB 1F	8	↕↕0P0↕	AをCyを含め右回転
RR B	CB 18	8	↕↕0P0↕	BをCyを含め右回転
RR C	CB 19	8	↕↕0P0↕	CをCyを含め右回転
RR D	CB 1A	8	↕↕0P0↕	DをCyを含め右回転
RR E	CB 1B	8	↕↕0P0↕	EをCyを含め右回転
RR H	CB 1C	8	↕↕0P0↕	HをCyを含め右回転
RR L	CB 1D	8	↕↕0P0↕	LをCyを含め右回転
RRA	1F	4	・・0・0↕	AをCyを含め右回転
RRC (HL)	CB 0E	15	↕↕0P0↕	(HL)をCyを含めず右回転
RRC (IX+nnH)	DD CB nn 0E	23	↕↕0P0↕	(IX+nnH)をCyを含めず右回転
RRC (IY+nnH)	FD CB nn 0E	23	↕↕0P0↕	(IY+nnH)をCyを含めず右回転
RRC A	CB 0F	8	↕↕0P0↕	AをCyを含めず右回転
RRC B	CB 08	8	↕↕0P0↕	BをCyを含めず右回転
RRC C	CB 09	8	↕↕0P0↕	CをCyを含めず右回転
RRC D	CB 0A	8	↕↕0P0↕	DをCyを含めず右回転
RRC E	CB 0B	8	↕↕0P0↕	EをCyを含めず右回転
RRC H	CB 0C	8	↕↕0P0↕	HをCyを含めず右回転
RRC L	CB 0D	8	↕↕0P0↕	LをCyを含めず右回転
RRCA	0F	4	・・0・0↕	AをCyを含めず右回転
RRD	ED 67	18	↕↕0P0・	Aのb3～b0→(HL)b7～b4 (HL)b7～b4→(HL)b3～b0 (HL)b3～b0→Aのb3～b0
RST 00H	C7	11	・・・・・・	0000Hをサブルーチンコール
RST 08H	CF	11	・・・・・・	0008Hをサブルーチンコール
RST 10H	D7	11	・・・・・・	0010Hをサブルーチンコール
RST 18H	DF	11	・・・・・・	0018Hをサブルーチンコール
RST 20H	E7	11	・・・・・・	0020Hをサブルーチンコール
RST 28H	EF	11	・・・・・・	0028Hをサブルーチンコール
RST 30H	F7	11	・・・・・・	0030Hをサブルーチンコール
RST 38H	FF	11	・・・・・・	0038Hをサブルーチンコール
SBC A, (HL)	9E	7	↕↕↕V1↕	A－(HL)－Cy→A
SBC A, (IX+nnH)	DD 9E nn	19	↕↕↕V1↕	A－(IX+nnH)－Cy→A
SBC A, (IY+nnH)	FD 9E nn	19	↕↕↕V1↕	A－(IY+nnH)－Cy→A
SBC A, A	9F	4	↕↕↕V1↕	A－A－Cy→A
SBC A, B	98	4	↕↕↕V1↕	A－B－Cy→A
SBC A, C	99	4	↕↕↕V1↕	A－C－Cy→A

アセンブリ言語	マシン語	ステート数	フラグ変化 S Z H P/V N C	働き
SBC A, D	9A	4	↕↕↕V1↕	A−D−Cy→A
SBC A, E	9B	4	↕↕↕V1↕	A−E−Cy→A
SBC A, H	9C	4	↕↕↕V1↕	A−H−Cy→A
SBC A, L	9D	4	↕↕↕V1↕	A−L−Cy→A
SBC A, mmH	DE mm	7	↕↕↕V1↕	A−mmH−Cy→A
SBC HL, BC	ED 42	15	↕↕↕V1↕	HL−BC−Cy−HL
SBC HL, DE	ED 52	15	↕↕↕V1↕	HL−DE−Cy−HL
SBC HL, HL	ED 62	15	↕↕↕V1↕	HL−HL−Cy−HL
SBC HL, SP	ED 72	15	↕↕↕V1↕	HL−SP−Cy−HL
SCF	37	4	・・0・01	1→Cy（Cyフラグを強制セット）
SET 0, (HL)	CB C6	15	・・・・・・	(HL) のb0を1にセット
SET 0, (IX+nnH)	DD CB nn C6	23	・・・・・・	(IX+nnH) のb0を1にセット
SET 0, (IX+nnH)	FD CB nn C6	23	・・・・・・	(IY+nnH) のb0を1にセット
SET 0, A	CB C7	8	・・・・・・	Aのb0を1にセット
SET 0, B	CB C0	8	・・・・・・	Bのb0を1にセット
SET 0, C	CB C1	8	・・・・・・	Cのb0を1にセット
SET 0, D	CB C2	8	・・・・・・	Dのb0を1にセット
SET 0, E	CB C3	8	・・・・・・	Eのb0を1にセット
SET 0, H	CB C4	8	・・・・・・	Hのb0を1にセット
SET 0, L	CB C5	8	・・・・・・	Lのb0を1にセット
SET 1, (HL)	CB CE	15	・・・・・・	(HL) のb1を1にセット
SET 1, (IX+nnH)	DD CB nn CE	23	・・・・・・	(IX+nnH) のb1を1にセット
SET 1, (IY+nnH)	FD CB nn CE	23	・・・・・・	(IY+nnH) のb1を1にセット
SET 1, A	CB CF	8	・・・・・・	Aのb1を1にセット
SET 1, B	CB C8	8	・・・・・・	Bのb1を1にセット
SET 1, C	CB C9	8	・・・・・・	Cのb1を1にセット
SET 1, D	CB CA	8	・・・・・・	Dのb1を1にセット
SET 1, E	CB CB	8	・・・・・・	Eのb1を1にセット
SET 1, H	CB CC	8	・・・・・・	Hのb1を1にセット
SET 1, L	CB CD	8	・・・・・・	Lのb1を1にセット
SET 2, (HL)	CB D6	15	・・・・・・	(HL) のb2を1にセット
SET 2, (IX+nnH)	DD CB nn D6	23	・・・・・・	(IX+nnH) のb2を1にセット
SET 2, (IY+nnH)	FD CB nn D6	23	・・・・・・	(IY+nnH) のb2を1にセット
SET 2, A	CB D7	8	・・・・・・	Aのb2を1にセット
SET 2, B	CB D0	8	・・・・・・	Bのb2を1にセット
SET 2, C	CB D1	8	・・・・・・	Cのb2を1にセット
SET 2, D	CB D2	8	・・・・・・	Dのb2を1にセット
SET 2, E	CB D3	8	・・・・・・	Eのb2を1にセット
SET 2, H	CB D4	8	・・・・・・	Hのb2を1にセット
SET 2, L	CB D5	8	・・・・・・	Lのb2を1にセット
SET 3, (HL)	CB DE	15	・・・・・・	(HL) のb3を1にセット
SET 3, (IX+nnH)	DD CB nn DE	23	・・・・・・	(IX+nnH) のb3を1にセット
SET 3, (IY+nnH)	FD CB nn DE	23	・・・・・・	(IY+nnH) のb3を1にセット
SET 3, A	CC DF	8	・・・・・・	Aのb3を1にセット
SET 3, B	CA D8	8	・・・・・・	Bのb3を1にセット
SET 3, C	CB D9	8	・・・・・・	Cのb3を1にセット
SET 3, D	CB DA	8	・・・・・・	Dのb3を1にセット
SET 3, E	CB DB	8	・・・・・・	Eのb3を1にセット

アセンブリ言語	マシン語	ステート数	フラグ変化 SZHP/vNC	働き
SET 3, H	CB DC	8	・・・・・・	Hのb3を1にセット
SET 3, L	CB DD	8	・・・・・・	Lのb3を1にセット
SET 4, (HL)	CB E6	15	・・・・・・	(HL) のb4を1にセット
SET 4, (IX+nnH)	DD CB nn E6	23	・・・・・・	(IX+nnH) のb4を1にセット
SET 4, (IY+nnH)	FD CB nn E6	23	・・・・・・	(IY+nnH) のb4を1にセット
SET 4, A	CB E7	8	・・・・・・	Aのb4を1にセット
SET 4, B	CB E0	8	・・・・・・	Bのb4を1にセット
SET 4, C	CB E1	8	・・・・・・	Cのb4を1にセット
SET 4, D	CB E2	8	・・・・・・	Dのb4を1にセット
SET 4, E	CB E3	8	・・・・・・	Eのb4を1にセット
SET 4, H	CB E4	8	・・・・・・	Hのb4を1にセット
SET 4, L	CB E5	8	・・・・・・	Lのb4を1にセット
SET 5, (HL)	CB EE	15	・・・・・・	(HL) のb5を1にセット
SET 5, (IX+nnH)	DD CB nn EE	23	・・・・・・	(IX+nnH) のb5を1にセット
SET 5, (IY+nnH)	FD CB nn EE	23	・・・・・・	(IY+nnH) のb5を1にセット
SET 5, A	CB EF	8	・・・・・・	Aのb5を1にセット
SET 5, B	CB E8	8	・・・・・・	Bのb5を1にセット
SET 5, C	CB E9	8	・・・・・・	Cのb5を1にセット
SET 5, D	CB EA	8	・・・・・・	Dのb5を1にセット
SET 5, E	CB EB	8	・・・・・・	Eのb5を1にセット
SET 5, H	CB EC	8	・・・・・・	Hのb5を1にセット
SET 5, L	CB ED	8	・・・・・・	Lのb5を1にセット
SET 6, (HL)	CB F6	15	・・・・・・	(HL) のb6を1にセット
SET 6, (IX+nnH)	DD CB nn F6	23	・・・・・・	(IX+nnH) のb6を1にセット
SET 6, (IY+nnH)	FD CB nn F6	23	・・・・・・	(IY+nnH) のb6を1にセット
SET 6, A	CB F7	8	・・・・・・	Aのb6を1にセット
SET 6, B	CB F0	8	・・・・・・	Bのb6を1にセット
SET 6, C	CB F1	8	・・・・・・	Cのb6を1にセット
SET 6, D	CB F2	8	・・・・・・	Dのb6を1にセット
SET 6, E	CB F3	8	・・・・・・	Eのb6を1にセット
SET 6, H	CB F4	8	・・・・・・	Hのb6を1にセット
SET 6, L	CB F5	8	・・・・・・	Lのb6を1にセット
SET 7, (HL)	CB FE	15	・・・・・・	(HL) のb7を1にセット
SET 7, (IX+nnH)	DD CB nn FE	23	・・・・・・	(IX+nnH) のb7を1にセット
SET 7, (IY+nnH)	FD CB nn FE	23	・・・・・・	(IY+nnH) のb7を1にセット
SET 7, A	CB FF	8	・・・・・・	Aのb7を1にセット
SET 7, B	CB F8	8	・・・・・・	Bのb7を1にセット
SET 7, C	CB F9	8	・・・・・・	Cのb7を1にセット
SET 7, D	CB FA	8	・・・・・・	Dのb7を1にセット
SET 7, E	CB FB	8	・・・・・・	Eのb7を1にセット
SET 7, H	CB FC	8	・・・・・・	Hのb7を1にセット
SET 7, L	CB FD	8	・・・・・・	Lのb7を1にセット
SLA (HL)	CB 26	15	↕↕0P0↕	(HL) ×2→ (HL)
SLA (IX+nnH)	DD CB nn 26	23	↕↕0P0↕	(IX+nnH) ×2→ (IX+nnH) b0← 0 b1←b0 b2←b1 b3←b2 b4←b3
SLA (IY+nnH)	FD CB nn 26	23	↕↕0P0↕	(IY+nnH) ×2→ (IY+nnH)
SLA	CB 27	8	↕↕0P0↕	A×2→A

アセンブリ言語	マシン語	ステート数	フラグ変化 SZHP/vNC	働き	
SLA B	CB 20	8	↕↕0P0↕	B×2→B	b5←b4
SLA C	CB 21	8	↕↕0P0↕	C×2→C	b6←b5
SLA D	CB 22	8	↕↕0P0↕	D×2→D	b7←b6
SLA E	CB 23	8	↕↕0P0↕	E×2→E	Cy←b7
SLA H	CB 24	8	↕↕0P0↕	H×2→H	
SLA L	CB 25	8	↕↕0P0↕	L×2→L	
SRA (HL)	CB 2E	15	↕↕0P0↕	(HL) ÷2→ (HL) (2の補数扱い)	
SRA (IX+nnH)	DD CB nn 2E	23	↕↕0P0↕	(IX+nnH) ÷2→ (IX+nnH) (2の補数扱い)	b7→b7
SRA (IY+nnH)	FD CB nn 2E	23	↕↕0P0↕	(IY+nnH) ÷2→ (IY+nnH) (2の補数扱い)	b7→b6 b6→b5 b5→b4
SRA A	CB 2F	8	↕↕0P0↕	A÷2→A(2の補数扱い)	b4→b3
SRA B	CB 28	8	↕↕0P0↕	B÷2→B(2の補数扱い)	b3→b2
SRA C	CB 29	8	↕↕0P0↕	C÷2→C(2の補数扱い)	b2→b1
SRA D	CB 2A	8	↕↕0P0↕	D÷2→D(2の補数扱い)	b1→b0
SRA E	CB 2B	8	↕↕0P0↕	E÷2→E(2の補数扱い)	b0→Cy
SRA H	CB 2C	8	↕↕0P0↕	H÷2→H(2の補数扱い)	
SRA L	CB 2D	8	↕↕0P0↕	L÷2→L(2の補数扱い)	
SRL (HL)	CB 3E	15	↕↕0P0↕	(HL) ÷2→ (HL) (正の値の扱い)	
SRL (IX+nnH)	DD CB nn 3E	23	↕↕0P0↕	(IX+nnH) ÷2→ (IX+nnH) (正の値の扱い)	0→b7
SRL (IY+nnH)	FD CB nn 3E	23	↕↕0P0↕	(IY+nnH) ÷2→ (IY+nnH) (正の値の扱い)	b7→b6 b6→b5 b5→b4
SRL A	CB 3F	8	↕↕0P0↕	A÷2→A(正の値の扱い)	b4→b3
SRL B	CB 38	8	↕↕0P0↕	B÷2→B(正の値の扱い)	b3→b2
SRL C	CB 39	8	↕↕0P0↕	C÷2→C(正の値の扱い)	b2→b1
SRL D	CB 3A	8	↕↕0P0↕	D÷2→D(正の値の扱い)	b1→b0
SRL E	CB 3B	8	↕↕0P0↕	E÷2→E(正の値の扱い)	b0→Cy
SRL H	CB 3C	8	↕↕0P0↕	H÷2→H(正の値の扱い)	
SRL L	CB 3D	8	↕↕0P0↕	L÷2→L(正の値の扱い)	
SUB (HL)	96	7	↕↕↕V1↕	A−(HL)→A	
SUB (IX+nnH)	DD 96 nn	19	↕↕↕V1↕	A−(IX+nnH)→A	
SUB (IY+nnH)	FD 96 nn	19	↕↕↕V1↕	A−(IY+nnH)→A	
SUB A	97	4	↕↕↕V1↕	A−A→A	
SUB B	90	4	↕↕↕V1↕	A−B→A	
SUB C	91	4	↕↕↕V1↕	A−C→A	
SUB D	92	4	↕↕↕V1↕	A−D→A	
SUB E	93	4	↕↕↕V1↕	A−E→A	
SUB H	94	4	↕↕↕V1↕	A−H→A	
SUB L	95	4	↕↕↕V1↕	A−L→A	
SUB mmH	D6 mm	7	↕↕↕V1↕	A−mmH→A	
XOR (HL)	AE	7	↕↕0P00	A XOR (HL)→A	

アセンブリ言語	マシン語	ステート数	フラグ変化 SZHP/vNC	働き
XOR (IX+nnH)	DD AE nn	19	↕↕0P00	A XOR (IX+nnH) →A
XOR (IY+nnH)	FD AE nn	19	↕↕0P00	A XOR (IY+nnH) →A
XOR A	AF	4	↕↕0P00	A XOR A→A
XOR B	A8	4	↕↕0P00	A XOR B→A
XOR C	A9	4	↕↕0P00	A XOR C→A
XOR D	AA	4	↕↕0P00	A XOR D→A
XOR E	AB	4	↕↕0P00	A XOR E→A
XOR H	AC	4	↕↕0P00	A XOR H→A
XOR L	AD	4	↕↕0P00	A XOR L→A
XOR mmH	EE mm	7	↕↕0P00	A XOR mmH→A

〔フラグ変化の記号〕

↕…規則的に変化する
0…必ず0になる(リセットされる)
1…必ず1になる(セットされる)
・…変化しない(以前の状態を保持する)
?…どうなるか決められていない(不規則に変化する)
P/Vフラグの場合
　P…パリティフラグとして規則的に変化する
　V…オーバーフローフラグとして規則的に変化する

※IX・IYレジスタ，JR命令での「+nnH」の部分は，実際には「+00H」～「+7FH」または「−80H」～「−01H」が入る。また，その時のマシン語は，+00H～+7FHの場合にはそのまま，−80H～−01Hの場合には80H～FFH(−80H→80H・−7EH→82H … −02H→FEH・01H→FFH)となる。言い換えると，マシン語での指定値は8ビットであるが，実際にはそのb7をb15～b8にコピーし拡張した16ビット値(00H～7FH→0000H～007FH，80H～FFH→FF80H～FFFFH)とIX等の値が加算され，実効アドレスが決定されると考えても良い(加算の際のオーバーフローは無視する)。

※JR命令で使用している「$」は，その命令の始まるアドレスを示す記号である。つまり，$+2では，(JR命令が2バイト長だから)JR命令の次の命令のアドレスを意味する。つまり，JR命令はその次の命令からの相対アドレス(−80H～+7FH)マシン語で指定することになる。

ステート数(実行単位時間)数値が二つあるのは
　条件CALL・JR・RET…条件成立時(CALL等を実行した時)／
　　条件非成立時(直後の命令に通り過ぎた時)
　その他(ブロックを扱う命令)…最後の1回分(カウンタが0になった時)／
　　それ以外(カウンタが0でなく，再びその命令を実行する時)

ステート数 n の時の実行時間：クロック4MHzの時…n/4000000秒(n/4μ秒)
　クロック6MHzの時…(n+m)/6000000秒(n+m)/6μ秒
　ただし命令の1バイト目がCBH DDH EDH FDHのいずれかの時はm=2そうでない時はm=1(mはM1サイクルにWAITが一つ入るために生じる遅延)

パソコンビジネスにこの一冊

スーパーパソコン ビジネスBASIC ●定価1300円 ●判型B6 ●送料250円

素早くマスター マイコンBASIC講座①実務応用編 ●定価1,500円 ●判型B5 ●送料300円

マイコンBASIC講座②ビジネス活用編 ●定価1,300円 ●判型B5 ●送料300円

マイコンBASIC講座③プログラムマスター編 ●定価1,300円 ●判型B5 ●送料300円

マイコンBASIC講座④データファイル活用編 ●定価1,400円 ●判型B5 ●送料300円

マイコン学入門基本18章 ●定価1,400円 ●判型A5 ●送料300円

ビジネスマイコン入門基本18章 ●定価1,400円 ●判型A5 ●送料300円

マイコン、パソコンとOA入門基本18章 ●定価1,400円 ●判型A5 ●送料300円

パソコン学習基本18章 ●定価1,500円 ●判型A5 ●送料250円

PC-8001、8801マシン語入門① ●定価1,300円 ●判型B5 ●送料300円

PC-8001マシン語入門② ●定価1,300円 ●判型B5 ●送料300円

PC-8801mkIIマシン語入門① ●定価1,500円 ●判型A5 ●送料250円

PC-9801E/F:MS-DOS入門 ●定価1,800円 ●判型A5 ●送料300円

FM-7活用研究 ●定価1,500円 ●判型B5 ●送料300円

漢字L II COBOL入門 ●定価2,000円 ●判型B5 ●送料300円

株価チャート分析入門 ●定価2,000円 ●判型B5 ●送料300円

MZ-2000/2200プログラミングテクニック ●定価1,300円 ●判型B5 ●送料300円

Z80マシン語プログラム入門 ●定価1,500円 ●判型B5 ●送料300円

6809マシン語プログラム入門 ●定価1,500円 ●判型B5 ●送料300円

マイコンOA用語辞典 ●定価2,300円 ●判型B6

マイコン別冊

Super MZ活用研究

ハード&ソフトマスター編

高橋雄一/多部田俊雄 共著

昭和60年12月24日初版発行

©1985 Printed in Japan

定価 2,000円 (送料300円)

発行人 平山秀雄

発行所 (株)電波新聞社

郵便番号141 東京都品川区東五反田1-11-15

電話(03) 445−6111(大代)振替東京5-51961

印刷所 大日本印刷(株)

製本所 (株)堅省堂

ROUND SYSTEM LABORATORY INC.

Super mz MZ-2500　**『スーパー財務/テレビ元帳』¥128,000**

MZ-2500の大容量・超高速をフルに活用した16ビット用ソフトを遥かに超える高速多機能会計ソフトの誕生です。

SUPER MZには「スーパー財務/テレビ元帳」

AI(人工知能)会計ソフト　今これに追随出来るものはありません!

①1枚のディスクに6,000仕訳のデータが入り、これを1ヶ月分としても、12ヶ月分としても使用出来る。
②勘定科目は補助科目を含めて600まで、全部自由設定。期中に追加、変更、割込が自由に出来る。
③指定期間内であれば、以前の月でも来月でも仕訳データの入力、訂正、削除が出来て、処理時間なし。
④仕訳データは日付順に入れなくても、仕訳日記帳も、元帳も、日付順、入力番号順の両方出せて、待時間なし。
⑤データ入力直後に電源が切れても、データの異常は起らない。誤入力のためデータが乱れても修復出来る。
⑥他のソフトの様にデータ量が多くなると処理時間が数分～数十分かかること一切なし。いつでもすぐに出る。
⑦摘要の漢字入力は辞書ROMで文節変換、人名、地名も可。英数字、カナ入力も出来る上、パスワード入力は結合可能。(パスワードプラス機能)
⑧階層メニュー方式、オールメッセージ、誤入力時の警告、コマンドの常時表示のため初めての人でも殆ど説明書不要。
⑨サンプルデータ付のため、入手後すぐ全機能のテスト、プリント、データ入力の練習も出来る。(全自動仕訳と云うものは世界中にありません。)

「スーパー財務/テレビ元帳」は今やあらゆるコンピュータ用会計ソフト中最高位のもので、これより高価なものでも、とうてい及びません。これはSUPER MZの優秀性とラウンドシステムの会計処理のキャリアの相乗効果とも云えます。経理事務の実務上のことを十分に配慮してありますから、実務家各位には十分ご満足頂ける内容です。但し全く簿記も分らない方は、その方の勉強を一寸だけお願いします。(全自動仕訳と云うものは世界中にありません。)

適合機種	あらゆる業種、法人、個人、特殊法人、組合、団体
勘定科目	全部自由設定、簡易科目名漢字入力、カナ漢字変換
補助科目	任意の科目に任意の数の補助科目設定可
勘定科目数	補助科目を含めて600個まで
仕訳件数	1枚のディスクに6,000件、最大12ヶ月分に自動配分
金額	1件、合計共99億円まで。(オプション999億円)
摘要	漢字12字、カナ24字、パスワードプラス機能
マスターファイル	自動月次残高算出機能付ランダムファイル
データファイル	超高速日付順検索付ランダムファイル
使用言語	SUPER BASIC+機械語
演算速度	16ビット機用ソフトの2倍強(当社比)
プリンタースピード	プリンターの限界速度で連続ノンストップ
プリンター用紙	全部普通のストックフォーム、元帳は専用用紙もあり

画面出力	テレビ元帳、テレビ試算表、テレビB/S,P/L、テレビ仕訳日記、テレビ予算実績対比、テレビ資金繰実績、当月、通期利益表
印刷出力	総勘定元帳、補助簿、試算表、貸借対照表、損益計算書、仕訳日記帳、資金繰実績表、予算実績対比表、その他
オプションソフト	特殊法人決算書、部門別利益計算書、工事台帳、手形管理、固定資産台帳(予定)
機器構成	MZ-2500 FD×2、128KB増設RAM MZ-1D22(CRT)又は同等品 MZ-1P18(漢字プリンター)又は1P10A,1P11A,1P06,1P03、辞書ROM
提供メディア	3.5インチ2DDフロッピーディスク×2
附属品	サンプルデータ、予備ソフト、ガイドブック

スーパーシリーズビジネスソフトは、「スーパー給与」「スーパー販売/テレビ台帳」「スーパー仕入/テレビ台帳」等続々発表の予定です。また熱心な自作派ビジネスマンのためにノウハウ公開の新Qシリーズはオールランダムファイルで発表の予定です。またMZ-80B,MZ-2000,2200用の「スーパー財務/テレビ元帳」(カナ)や「スーパー在庫管理」(カナ)やQシリーズ、テープソフトなど引続きサポート中です。詳しくは「SHARP MZ APPLICATION」をごらん下さい。尚、当社は誇大広告や欠点を隠して売ると云ったことは一切ありません。不愉快な電話をおことわりします。業者取引はメーカー認定店に限ります。
資料のご請求は、ソフトの種類を具体的に指定の上、なるべく切手200円同封して下さい。

Dシリーズソフトのユーザーはスーパーシリーズは特別価格
総合カタログMZ版(No.3)〒200(No.4は61年発行)
★ユーザー直接のご注文を歓迎します(会員特典あり)
〈ご注意〉当社ソフトのレンタル、コピイ販売、用紙の複製、商標の無断使用はバチが当たります。

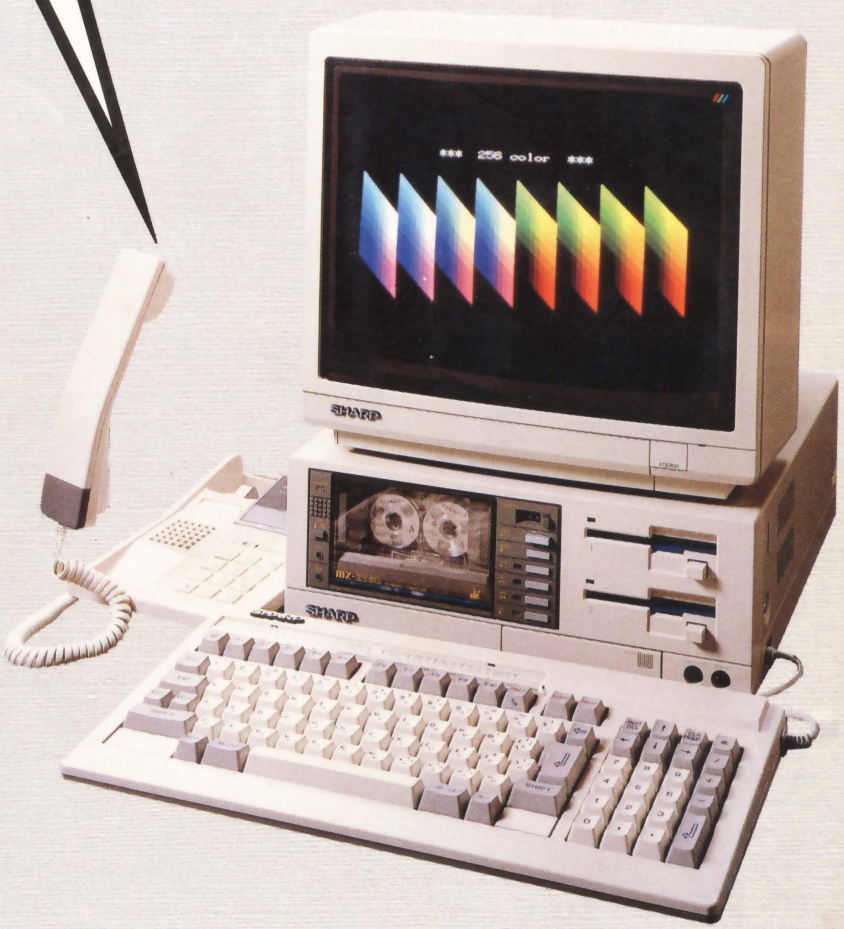
〒560 大阪府豊中市上野西3-2-25 TEL06(849)6982 FAX06(849)6744
株式会社 ラウンドシステム研究所
郵便振替口座/銀行口座 大阪5-95182
三和銀行豊中支店 (普) 313000
三菱銀行豊中支店 (普) 4323108

※ご注意:テレビ元帳は当社の創作語で商標登録申請済です。(無断使用に重ねて警告します)

SHARP

通信機能を搭載し、ニューメディアに対応。

札幌の山田くんと
大阪の細川くんは
パソコンで話せます。
ドキドキするぞ、ファーストコール。



断然おもしろい。
メニュー見ながら相手と話せるよ。

画面に表示されたメニューを見ながら対話形式でやさしく使える通信機能つきカード型データベースソフト「テレホンソフト」を装備。別売のモデムホンやボイスコミュニケーションインターフェイスと併用でデータ通信や「TeleStar」「アスキーネットワーク」など話題のデータベースにアクセスできる大きな楽しみ。

驚くほど速い。
精密グラフィックスがアッという間に現れるゾ。

圧倒的な速さと色彩感覚で迫ります。●最大1000文字の漢字を一度に表示する640×400ドットの高精細度グラフィックス●256色同時表示のきわだつカラー表現●1ドット単位のなめらかなスムーススクロール●瞬時に色変更、16色カラーパレット●最大1024個定義できるPCG●グラフィック専用LSI搭載。

本格的にきれい。
アナログ音声OK、サウンド機能もいっぱい。

プログラムやデータなどのデジタル信号に加え声などのアナログ信号も扱えるボイスレコーダ搭載。ナレーションの必要なアプリケーションに威力を発揮します。また音楽サウンド用シンセサイザICを採用、従来の8オクターブ3重和音(SSG)に加えFM音源(8オクターブ3重和音)も本格的に楽しめます。

みんなが待っていた新時代のパソコン誕生。

●コンパクトながら大容量640KBの3.5インチマイクロフロッピー搭載(Model 20/1基、Model 30/2基)●JIS第1水準漢字ROMに加えJIS第2水準漢字ROMまで標準装備したきわだつ日本語処理機能●強力な漢字BASIC-M25/BASIC-S25採用●最大256KBの大容量メモリ(標準128KB)●50音配列も可能な多機能キーボード●BASICやテレホンソフト使用時にも電卓機能などの特殊機能を利用できる便利なアルゴ(割り込み)機能●スイッチひとつで切替えできるMZ-2200/2000モード、MZ-80Bモードを装備。これまでMZシリーズで培ってきた多くの資産を継承。

写真はModel 30です。写真の14型カラーディスプレイMZ-1D22 標準価格108,000円 モデムホンMZ-1X19 標準価格98,000円は別売です。また装着されているカセットテープは撮影用で、本体の付属品・市販品ではありません。

Super MZ

8ビットパーソナルコンピュータMZ-2500シリーズ

Model 20(MZ-2511・640KB 3.5"FD1基付)……標準価格168,000円
Model 30(MZ-2521・640KB 3.5"FD2基付)……標準価格198,000円

MZ-2500
資料請求券
活用研究

シャープ株式会社 本社 〒545 大阪市阿倍野区長池町22番22号☎(06)621-1221(大代表)●お問い合わせは…本社内国内情報システム営業本部まで。

電波新聞社 〒141 東京都品川区東五反田1-11-15 電話(03)445-6111(大代) 定価2,000円 雑誌68469-27