GRAPHIC DISPLAY

TMS9918Aの機能と使い方

本章では，２次元のアニメーションを行うのに非常に便利なスプライトという機能があるTMS9918Aについて詳細に解説します．

椎尾一郎/中村拓男

TI社のビデオ・ディスプレイ・プロセッサ(VDP)TMS9918Aは，32枚のスプライト表示機能を持つユニークなLSIです．このLSIはソードやトミーの低価格パーソナル・コンピュータに使用されています．また最近では，米マイクロソフト社などにより提案され，国内主要メーカ14社が賛同してすすめられている，ホーム・パーソナル・コンピュータ規格統一仕様MSX[1]のCRTコントローラにも採用され注目を集めています．

TMS9918Aの特徴

TMS9918A(以下VDP)は，16色のカラーCRTディスプレイの表示および画面制御をするためのLSIです．またV-RAM用の４K，８K，16K D-RAMの自動リフレッシュ機能があり，表示データ用RAMを簡単に接続できます(図１)．

VDP内部のレジスタとV-RAMの書き込み，読み出しはマイクロコンピュータ側と接続される８ビットのパラレル・ポートにより行われます．

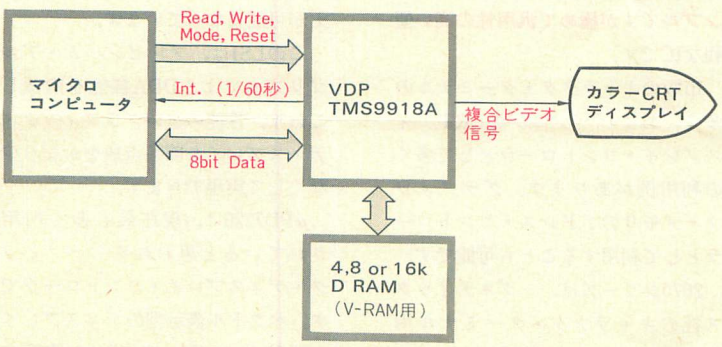
VDPの最大の特徴は32枚のスプライト(動画のためのグラフィック・パターン)表示機能です．図２にスプライトを使用した画面の例を示します．

スプライト面には，V-RAMに登録されたスプライト・データが表示されます．これは８×８または16×16画素で，それぞれ２倍に拡大することができます．また，スプライトの色指定(16色のうち１色)と表示位置も同様にV-RAM上で指定します．この表示位置座標を書き換えることにより，画面内を高速に移動させることができます．この時，スプライト面の優先順位(#０が最優先)にしたがって，順位の低いスプライトは順位の高いスプライトに消され，その結果３次元的な奥行きの効果が得られます．

また，スプライトの重なりによりVDP内のレジスタが変化します．VDPからは１フレームごと(1/60秒)の割り込み信号が出されていますが，この割り込みによってスプライト衝突の判断ルーチンを起動すれば無駄のない処理ができます．なお，スプライトが同一走査線上に５個以上存在すると，優先度の高い４個のスプライトだけが表示されるので注意が必要です．

VDPには，このようなスプライト表示機能のほかに，表１に示す４

〈図１〉システム構成



〈表１〉VDPの表示モード

モード	解像度	パターン数	色指定	スプライト(動画)
グラフィックⅠ	192×256画素	256種類	16色	使用可
グラフィックⅡ	192×256画素	768種類	16色	使用可
マルチカラー	48×64画素	—	16色	使用可
テキスト	24行×40列	256種類	16色のうち2色	使用不可

種類の静止画表示機能があります．これは図2のキャラクタ・パターン面に表示されるもので，192×256画素のグラフィック・モード，48×64画素のマルチカラー・モード，24行×40列のテキスト(キャラクタ)・モードがあります．テキスト・モードにはスプライト面を重ねることはできません．

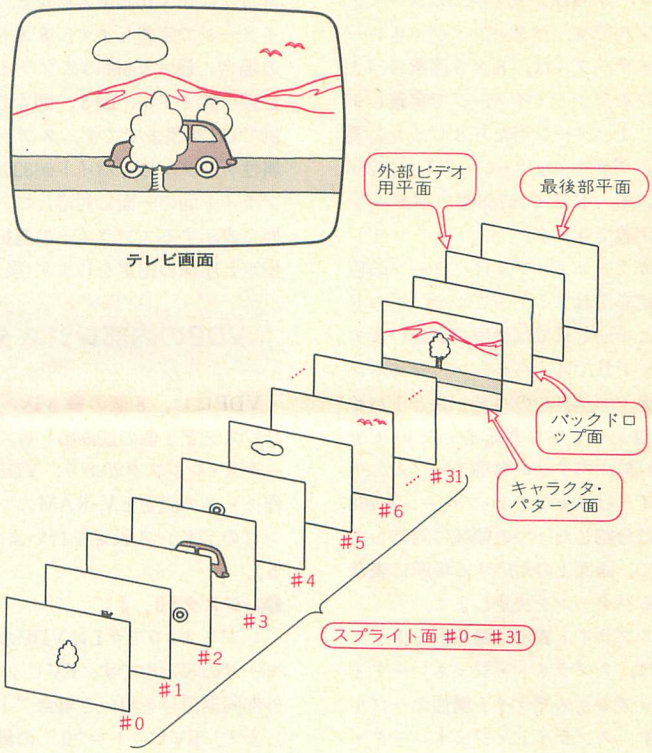
グラフィック・モードⅠとⅡはV-RAM上に定義した8×8画素のパターンをキャラクタ・プレーン上に24行×32列(768個)を表示するモードです．グラフィック・モードⅠとⅡの違いは定義可能なパターン数と色指定方法にあります．モードⅠでは256種，モードⅡでは768種のパターンが定義可能です．このため，モードⅠでは画面上の768区画の8×8画素の領域に任意の形のパターンを表示することはできません．パターンの種類が256種に制限されるからです．

一方，モードⅡでは画面上に表示できるパターンと同じ数のパターンを定義できます．したがって，グラフィック・モードⅡは，通常，ビット・イメージ・グラフィックとして使われ，表示される図形に制限はありません．すなわち，定義されたパターンの画面上の表示位置をあらかじめ決めてしまっておいて，画面の変更はパターンの定義の変更で行うわけです．

グラフィック・モードⅠとⅡでは色指定の方式も違います．モードⅠは8×8画素のパターン1個につき2色の色指定(すなわちビット1とビット0の色を指定)をしますが，モードⅡでは8×8画素のパターンの水平方向8ビットごとに2色の色指定が可能です．

モードⅡは色指定に関しても自由度が大きくなっていますが，1ドットごとの色を16色から自由に選択することはできません．一つのドットの色指定は，モードⅠでは隣接する8×8ドットの色指定と同じもの，モードⅡでは隣接する水平方向8ドットの色指定と同じものになります．

〈図2〉[2] 表示画面構成図



このように，モードⅡはモードⅠに比べて強力ですが，12Kバイト以上のV-RAMが必要です．しかし，現在のメモリICの市場価格から見て4K，8K D-RAMが使用されることはほとんどないと思いますので，モードⅡがVDPの標準モードといってよいでしょう．

VDPの他の静止画表示モードは，マルチカラー・モードとテキスト・モードです．この二つのモードは，定義できるパターン数が256個という点でグラフィック・モードⅠの変形です．マルチカラー・モードは，解像度を48×64画素とすることで，任意の点に16色のうちの任意の色指定を可能にしています．一方，テキスト・モードはグラフィック・モードⅠのパターンに文字フォントを定義して，VDPを文字ディスプレイとして使用するためのモードです．文字フォントとしてなら256個あれば十分です．テキスト・モードでは，表示するパターン数が24行×40列(960個)に増えています．これはパターンの大きさを8×6画素としたもので，定義された8×8画素パターンのMSB 6ビットが表示されます．次に，VDPがスプライトおよび静止画の表示と制御のために使用する，V-RAMのアドレス・マッピングを説明します．

V-RAMのマッピング

VDPはV-RAMを数個のテーブルに分けて使用します．図3はスプライトとグラフィック・モードⅠを用いる場合，図4は同じくグラフィック・モードⅡを使用する場合のV-RAMの分割例です．各テーブルの先頭番地は，VDP内部の各レジスタに登録して指定します．テーブルのとり方はこの図と同じである必要はありませんが，後のプログラム例ではこの分割を採用しています．

静止画表示に関係するテーブルは，

パターン・ジェネレータ・テーブル，パターン名称テーブル，カラー・テーブルです．パターン・ジェネレータ・テーブルは，8×8画素のパターンをビット・イメージで定義します．1パターン当たり8バイト必要で，グラフィック・モードⅠでは256個，モードⅡでは768個のパターンが定義できます．

カラー・テーブルはパターンの色指定を定義するテーブルで，モードⅠとモードⅡでは使用法が違います．モードⅡでは，パターン・ジェネレータ・テーブルのパターンと1バイト対1バイトの対応をしています(後述)．パターン名称テーブルの1バイトはキャラクタ・プレーンを768個に分割した一つの領域に対応していて，画面上の対応する場所に表示するパターンを決定します．

スプライト表示に関係するテーブルは，スプライト・ジェネレータ・テーブルとスプライト属性テーブルです．スプライト・ジェネレータ・テーブルは，スプライトの8×8画素または16×16画素の形をビット・イメージで定義します．8×8画素の場合，最大256個のスプライトが定義可能ですが，図3，図4の分割例では128個までです．スプライト属性テーブルの4バイトが32面のスプライト面の一面に対応して，この面に表示するスプライトの選択，色指定と座標の指定をします(後述)．

VDPの内部レジスタ

VDPには，8個の書き込み専用レジスタと1個の読み出し専用ステータス・レジスタがあり，VDPの動作モードの設定とV-RAMのマッピングのアドレス設定を行います(図5)．

❶レジスタ#0，#1

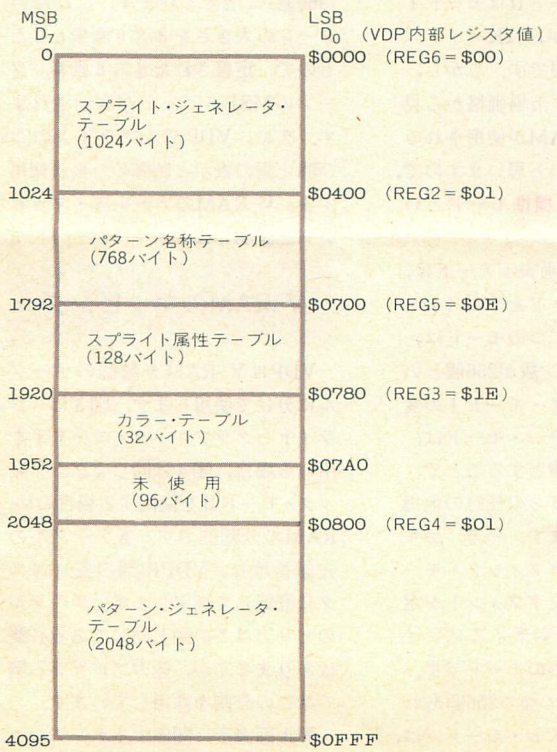
レジスタ#0と#1はVDPの動作モードの設定用です．EVビットは，外部同期信号を用いる場合“1”とします．EVビット=“0”の場合，VDP自身の同期信号となります．IEビットは，割り込み要求出力のON/OFF(1/0)です．これを“1”にすると一画面走査終了時にVDPの$\overline{\text{INT}}$ピンがアクティブ(“L”)になります．$\overline{\text{INT}}$出力は，VDPのステータス・レジスタの読み込みまたはVDPリセット時にリセットされます．

M1～M3ビットは，表示モードを表2に示す組み合わせで指定します．SIZEビットは，スプライトの大きさを8×8画素または16×16画素のいずれかに切り替えます．MAGビットは表示スプライトを拡大するものです．これを“1”とすると倍の大きさ(面積は4倍)で表示されます．

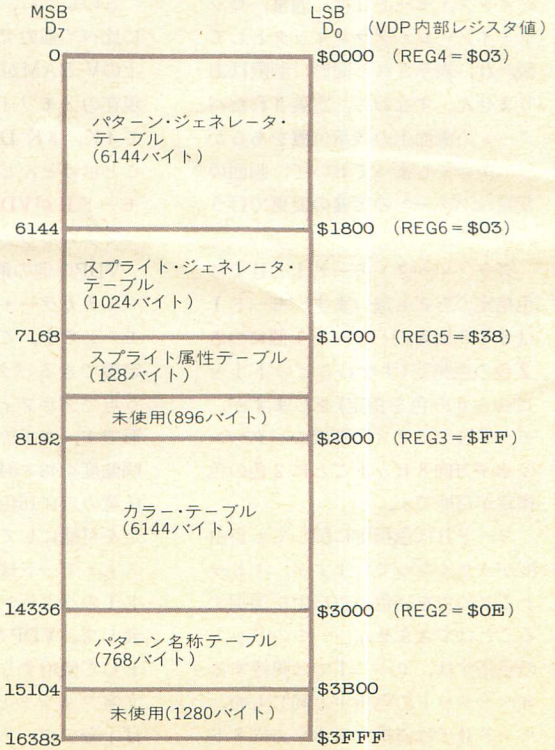
❷レジスタ#2

パターン名称テーブルの先頭番地の上位4ビットを下位4ビットに設定します〔図5(b)〕．パターン名称テーブルが$0400番地から始まる場合(図3参照)は$01となります．

〈図3〉4K V-RAMマッピング例(グラフィック・モードⅠ)



〈図4〉16K V-RAMマッピング例(グラフィック・モードⅡ)



〈表2〉表示モードの設定

M1	M2	M3	表示モード
0	0	0	グラフィックⅠ
0	0	1	グラフィックⅡ
0	1	0	マルチカラー
1	0	0	テキスト

❸レジスタ#3

(a)モードⅠ(M3＝“0”)の場合

カラー・テーブルの先頭番地の上位8ビットを設定します〔図5(c)〕．カラー・テーブルが$0780番地から始まる場合(図3参照)$1Eとなります．

(b)モードⅡ(M3＝“1”)の場合

カラー・テーブルの先頭番地の上位1ビットを，上位1ビットに設定し，他のビットはすべて“1”とします〔図5(d)〕．カラー・テーブルが$2000番地から始まる場合(図4参照)$FFとなります．

❹レジスタ#4

(a)モードⅠ(M3＝“0”)の場合

パターン・ジェネレータ・テーブルの先頭番地の上位3ビットを，下位3ビットに設定します〔図5(e)〕．パターン・ジェネレータ・テーブルが$0800番地から始まる場合(図3参照)，$01となります．

(b)モードⅡ〔M3＝“1”〕の場合

パターン・ジェネレータ・テーブルの先頭番地の上位1ビットを，下位から3ビット目に設定します．ビット1とビット0は“1”に設定します〔図5(f)〕．パターン・ジェネレータ・テーブルが$0000番地から始まる場合(図4参照)，$03となります．

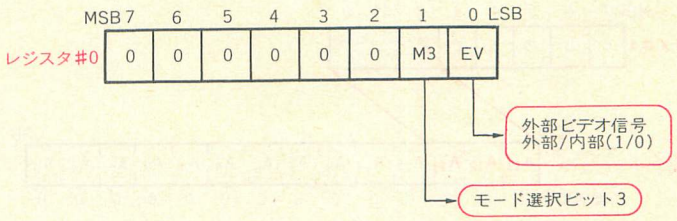
❺レジスタ#5

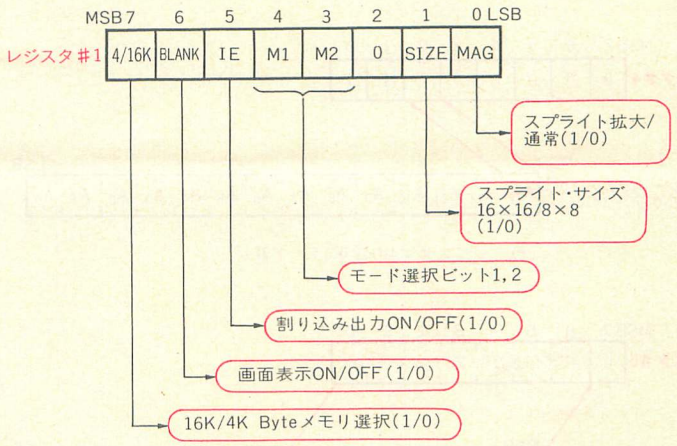
スプライト属性テーブルの先頭番地の上位7ビットを，下位7ビットに設定します〔図5(g)〕．スプライト属性テーブルが$0700番地から始まる場合(図3参照)，$0Eとなります．

❻レジスタ#6

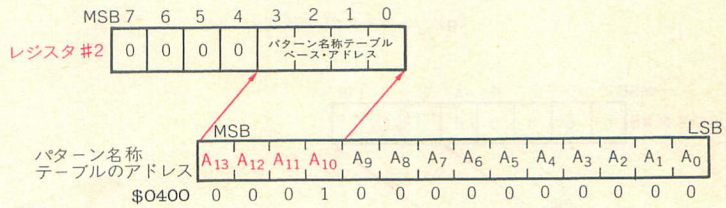
スプライト・ジェネレータ・テーブルの先頭番地の上位3ビットを，下位3ビットに設定します〔図5(h)〕．

〈図5〉VDPの内部レジスタ

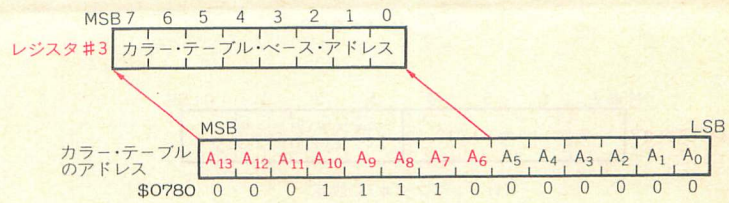




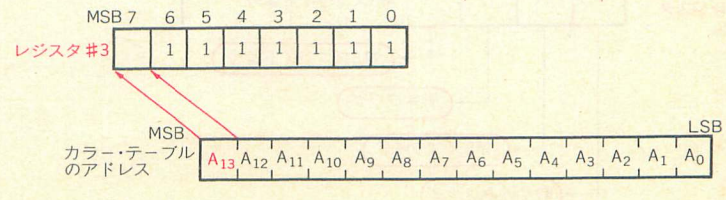
(a) レジスタ#0, #1の設定



(b) レジスタ#2の設定

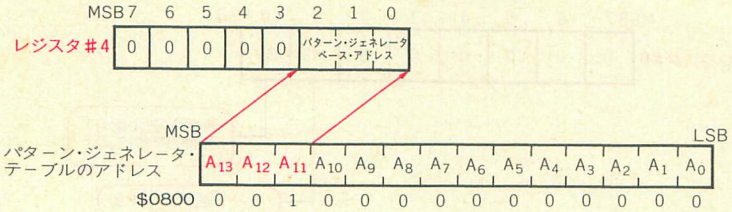


(c) レジスタ#3の設定(モードⅠ)

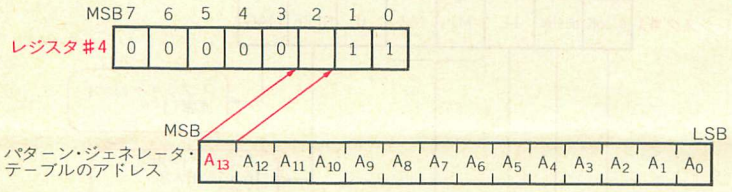


(d) レジスタ#3の設定(モードⅡ)

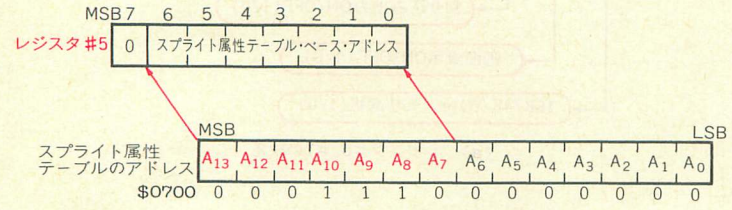
〈図5〉VDPの内部レジスタ(つづき)



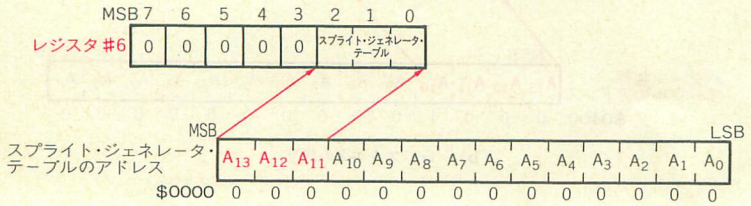
(e) レジスタ#4の設定(モードI)



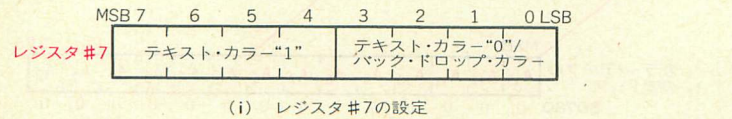
(f) レジスタ#4の設定(モードII)



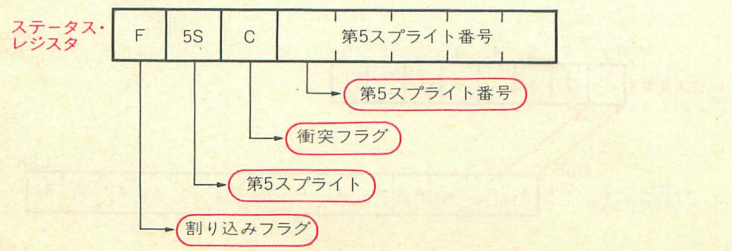
(g) レジスタ#5の設定



(h) レジスタ#6の設定



(i) レジスタ#7の設定



(j) ステータス・レジスタ

スプライト・ジェネレータ・テーブルが$00000番地から始まる場合(図3参照), $00となります.

❼レジスタ#7

テキスト・モードの色と, 他のモードのバック・ドロップ面の色を指定します〔図5(i)〕. 上位4ビットでキャラクタ・データの"1"の色, 下位4ビットは"0"の色となります. また, 下位4ビットで他のモードのバック・ドロップ面の色を指定します. VDPの色指定は$0から$Fまでの16種のコードで行います. 各コードと色の対応を表3に示します.

❽ステータス・レジスタ

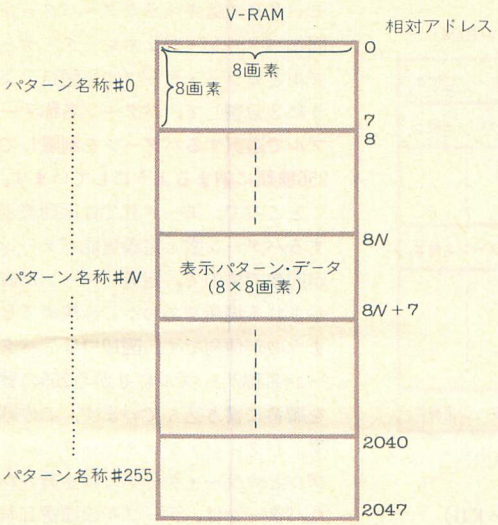
ステータス・レジスタの内容を図5(j)に示します. Fフラグは割り込みフラグで, 一画面走査終了時(1/60秒ごと)に"1"になります. レジスタ#1のIEビットが"1"の場合, $\overline{\text{INT}}$出力はアクティブ("L")になりますが, ステータス・レジスタの読み込みで$\overline{\text{INT}}$出力は"H"にリセットされます.

5Sフラグは5個以上のスプライトが同一水平ラインに存在し, かつFフラグ="0"の時"1"にセットされます. この時, 表示優先度が5番目以下のスプライトは表示されません. 優先度5番目のスプライト面番号は, ステータス・レジスタの下位5ビットに現れます.

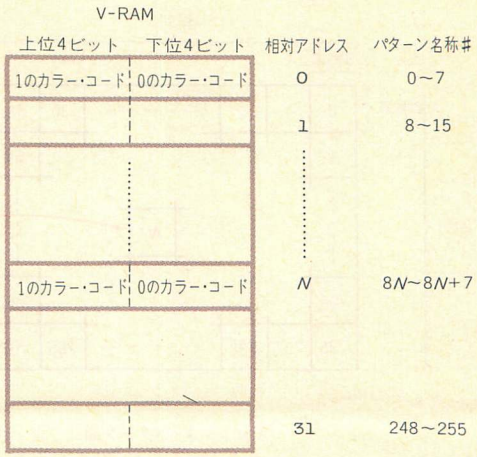
Cフラグは衝突フラグで, 二つ以上のスプライトが衝突した時"1"にセットされます. ステータス・レジスタはVDPのリセットとステータス・レジスタの読み出しによってリセットされます. 衝突フラグなどは, スプライトの一致で一度セットされると, スプライトが離れてもセットされたままです. 衝突の検出に先立ってステータス・レジスタの空読みが必要です.

VDPのマニュアルでは, ステータス・レジスタの読み出しをVDP割り込み発生時のみ実行することを推薦しています. これはステータス・レジスタが一画面走査ごとに更新さ

〈図6〉パターン・ジェネレータ・テーブル(モードⅠ)



〈図7〉カラー・テーブル(モードⅠ)



れるためです．ステータス・レジスタの読み出しや，スプライトの移動の直後にステータス・レジスタの読み出しを行うと，衝突の検出に失敗することがあります．ステータス・レジスタの読み出しは，割り込みと同期して行うか，割り込みを使用しない場合は，一画面走査時間(1/60秒)以上の待ち時間をとってから読み出すとよいでしょう．

静止画の表示

VDPのキャラクタ・パターン面に表示される静止画面は表1に示したように，グラフィック・モードⅠ，モードⅡ，マルチカラー，テキストの四つのモードがあります．しかし，分解能の点と，モードⅡがモードⅠの機能を含んでいる点から，グラフィック・モードⅡが使用されることが多いでしょう．

❶グラフィック・モードⅠ

グラフィック・モードは，V-RAM上のパターン・ジェネレータ・テーブル，パターン名称テーブルおよびカラー・テーブルの三つのテーブルを用いて表示されます．

パターン・ジェネレータ・テーブルには，表示される画像がビット・パターンで定義されます(図6)．パターンは8バイト(8×8画素)ごとにパターン名称が付けられます．パターン名称は数字の0から255までで，パターン・ジェネレータ・テーブルの8N番目のアドレスから8N+7までにあるパターンは，パターン名称#Nとなります．

カラー・テーブルは，パターンの色指定を行うテーブルです．32バイトのテーブルで，1バイトで8個のパターンの色を指定します(図7)．すなわち，カラー・テーブルのNバイト目はパターン名称#8Nから(8N+7)の8個のパターンの色指定です．カラー・テーブルの1バイトの上位4ビットはパターンの“1”の色を，下位4ビットは“0”の色を決めます．したがって，色指定はパターン単位で行われ，連続する8個のパターン名称#を持つパターンは同じ色指定となります．

パターン名称テーブルは，768バイトの領域でこの1バイトが画面上の8×8画素の区画一つに対応していて，この場所に表示するパターンをパターン名称#で指定します(図8)．

❷グラフィック・モードⅡ

モードⅡは，モードⅠと基本的には同じです．違っている点は定義可能なパターン数が768個に増えた点と，より細かい色指定が可能になった点です．

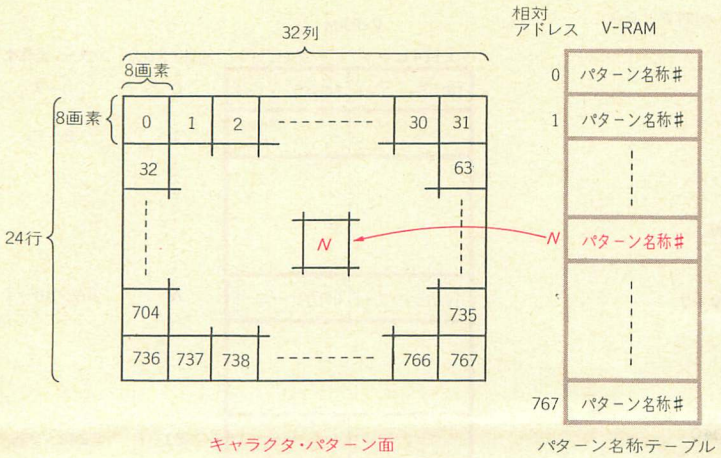
〈表3〉VDPのカラー・コード

コード	色	コード	色
0	透明	8	赤
1	黒	9	明るい赤
2	緑	A	暗い黄
3	明るい緑	B	明るい黄
4	暗い青	C	暗い緑
5	明るい青	D	マゼンタ
6	暗い赤	E	灰
7	シアン	F	白

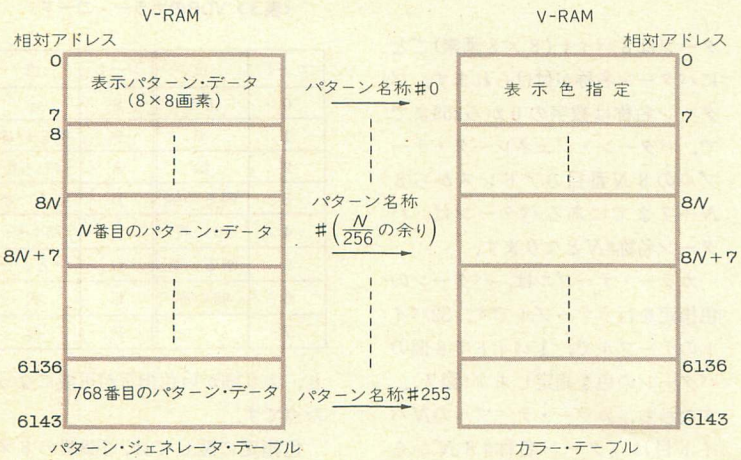
色指定に関しては，768個の8×8画素パターン1個につき8バイトの色指定情報を持つカラー・テーブルが用意されます(図9)．このため，パターン・ジェネレータ・テーブルとカラー・テーブルの大きさはそれぞれ等しくなります．パターン・ジェネレータ・テーブルで定義される表示パターン・データと，この表示色指定を行うカラー・テーブルの関係は図10に示しておきました．各パターンの水平方向の8画素ごとに，“1”と“0”に対応する色が指定できます．

パターン数が768個に増えたため，モードⅠの時のように単純にパターン名称#を順番に0から767とつけたのでは，パターン名称テーブルの1バイト・データでパターン名称を指

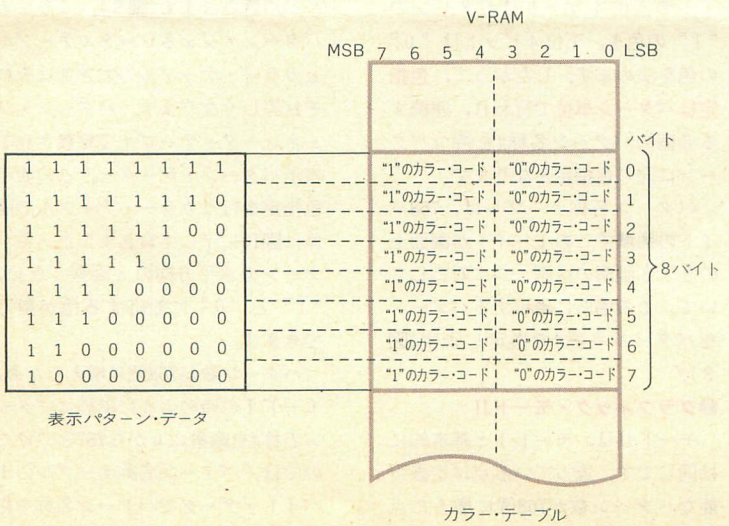
〈図8〉キャラクタ・パターン面とパターン名称テーブル



〈図9〉パターン・ジェネレータ・テーブルとカラー・テーブル(モードII)



〈図10〉グラフィック・モードIIのカラー・テーブル



定することができません．そこで，モードIIではキャラクタ・パターン面とパターン・ジェネレータ・テーブルとカラー・テーブルを図11のように3分割して，パターン名称テーブルで選択するパターンを制限して，256種類に納まるようにしています．

ところで，モードIIでは画面表示するパターン数と定義可能パターンが同数ですので，定義パターンが表示される場所をあらかじめ決めてしまうのが便利です．図12では，パターン名称テーブルに0から255の数を順番に書き込んでいます．この結果，パターン・ジェネレータ・テーブルとカラー・テーブルに定義されたパターンは，テーブルの順番どおりに画面左上から表示されます．表示の変更は，定義テーブルのデータを変更して行います．この方法では，パターン名称テーブルは全く無駄になっていますが，画面操作が容易で実用的な使い方です．

❸マルチカラー・モード

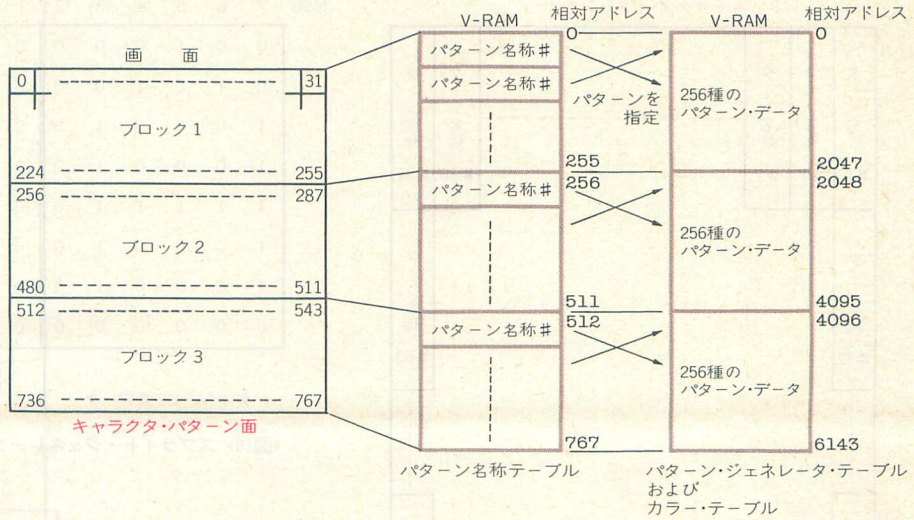
このモードの分解能は48×64画素ですが，各ドットに対して独立に色指定可能です．このモードは，パターン名称テーブルとパターン・ジェネレータ・テーブルを使用します．

キャラクタ・パターン面上の24行×32列区画の一つ一つに対応するパターン名称テーブルに，この場所に表示するドット組をパターン名称で指定します．画面上の24行×32列区画の一つにはマルチカラー・モードのドットが2×2個入ります．この一組をマルチカラー・パターンと呼びます(図13)．

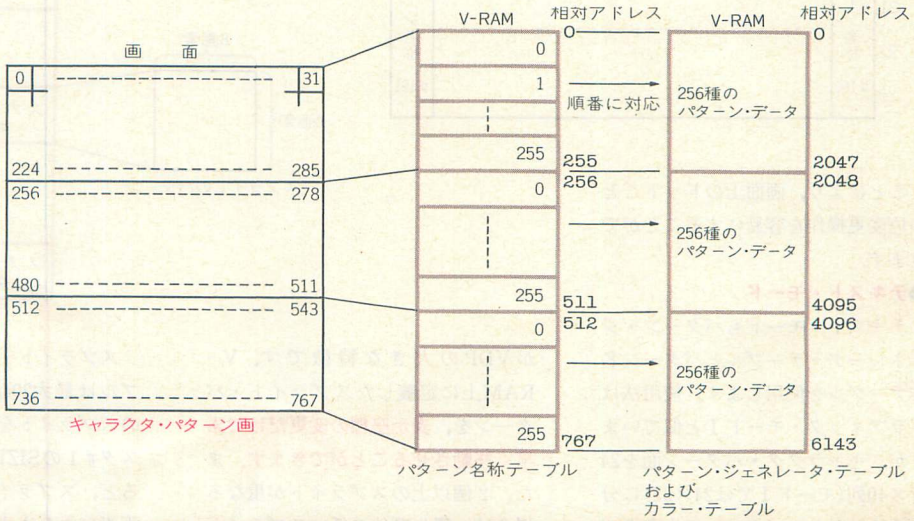
一つのパターン名称につき4個のマルチカラー・パターンが定義され，どれが適用されるかはこのパターンが表示される行(0～23行)数の4の剰余により決まります．このため，192個のパターンによって画面上を埋めることができます．

したがって，グラフィック・モードIIの場合と同様に，マルチカラー・パターンの占める画面上の位置を，例えば図14のように決めてしま

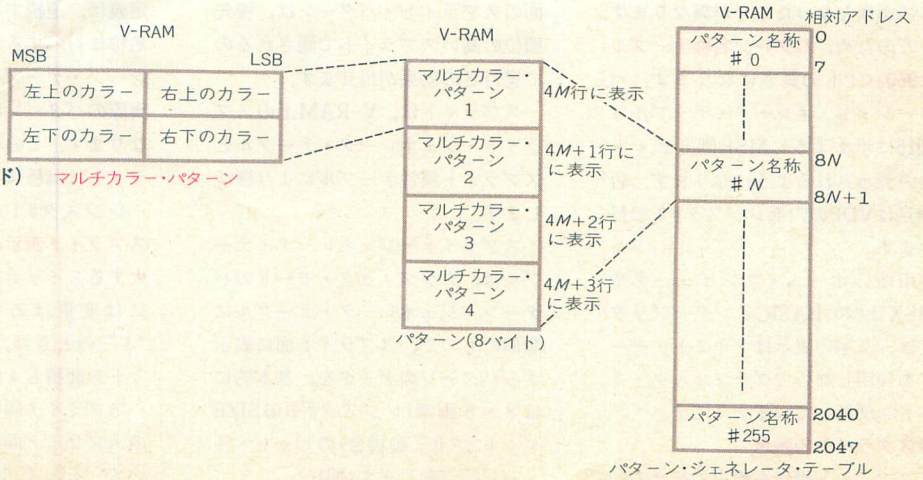
〈図11〉
モードIIのパターン名称テーブル



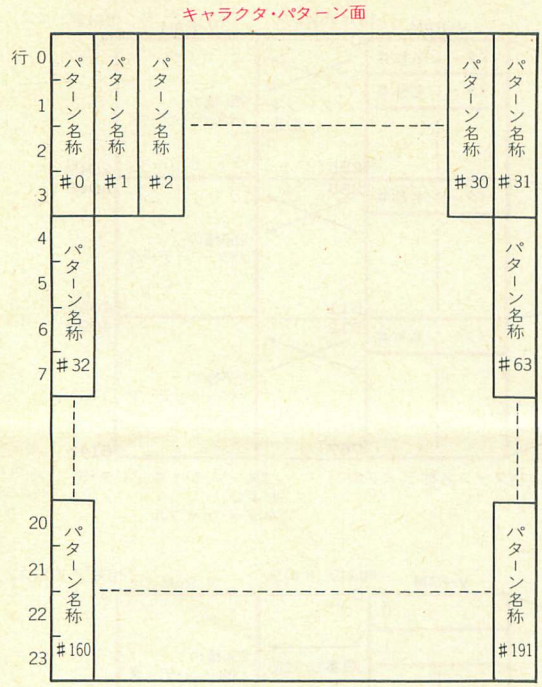
〈図12〉
モードIIの実用的なパターン名称テーブル設定



〈図13〉
パターン・ジェネレータ・テーブル
（マルチカラー・モード）



〈図14〉マルチカラー・モードの実用的なパターン配置

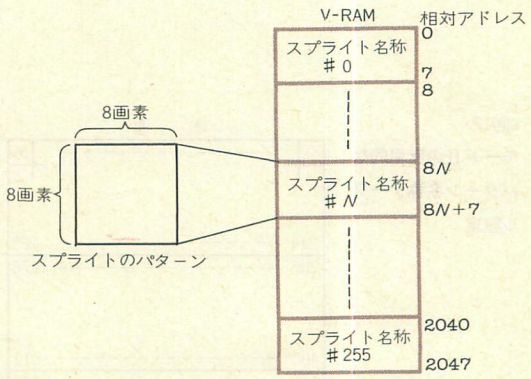


〈図15〉テキスト・モードでのパターン定義例

MSB 7	6	5	4	3	2	1	0 LSB	
0	0	1	0	0	0	0	0	20
0	1	0	1	0	0	0	0	50
1	0	0	0	1	0	0	0	88
1	0	0	0	1	0	0	0	88
1	1	1	1	1	0	0	0	F8
1	0	0	0	1	0	0	0	88
1	0	0	0	1	0	0	0	88
0	0	0	0	0	0	0	0	00 (16進)

文字"A"の2進データ　　表示されない

〈図16〉スプライト・ジェネレータ・テーブル



うことにより，画面上のドットごとの色変更操作を容易にすることができます．

❹テキスト・モード

テキスト・モードもパターン・ジェネレータ・テーブルとパターン名称テーブルを使用します．使用法はグラフィック・モードⅠと似ていますが，キャラクタ・パターン面を24行×40列(モードⅠでは24×32)に分割することと，パターンの大きさが8×6画素となったことが異なります．

このため，**パターン名称テーブルは960バイト**の長さになります．パターン・ジェネレータ・テーブルは図15に示すようにMSB側6ビットのみ表示されるようになります．色指定はVDPの内部レジスタ#7で行います．

市販のホーム・コンピュータやMSX仕様のBASICインタープリタでは，文字の表示はテキスト・モードを使用しないでグラフィック・モードを使用しているようです．

❺スプライトの表示

スプライト操作を簡単に行えるのがVDPの大きな特徴です．V-RAM上に定義したスプライト・パターンを，**表示座標の変更だけで素早く移動させることができます**．また，2個以上のスプライトが重なる場合は，優先順位の低いスプライト面のスプライト・パターンは，優先順位の高いスプライトで隠されるので奥行きの効果が出せます．

スプライトは，V-RAM上のスプライト・ジェネレータ・テーブルとスプライト属性テーブルにより機能します．

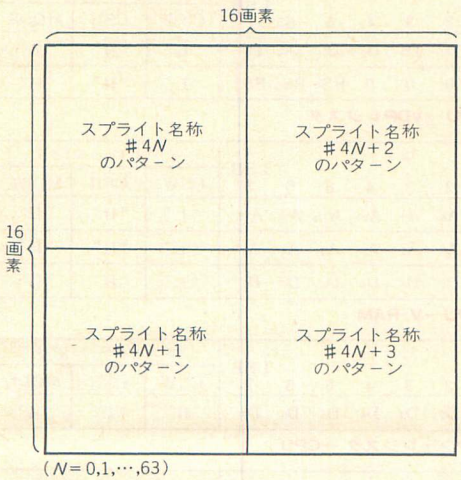
スプライト・ジェネレータ・テーブルは，グラフィック・モードのパターン・ジェネレータ・テーブルに類似のもので，スプライト面に表示するパターンのデータを，基本的には8×8画素(レジスタ#1のSIZEビット＝"0"の場合)のビット・イメージで定義します(図16)．

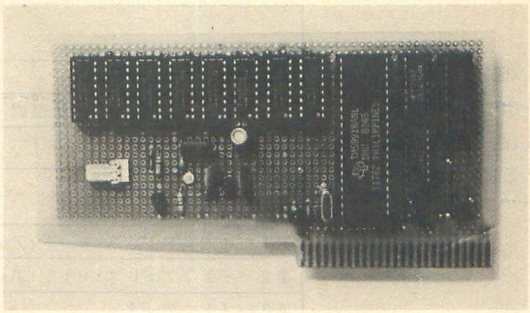
スプライト・ジェネレータ・テーブルは最大2048バイトで，256種類のスプライトを定義できます．レジスタ#1のSIZEビット＝"1"とすると，スプライトの大きさが16×16画素になります．このスプライトの定義は，連続する4個のスプライト名称に対応するスプライト・ジェネレータ・テーブルを用いて行います．実際のパターンは図17に示すものとなります．この時，定義可能なスプライトは64種類となります．

レジスタ#1の**MAGビットは，スプライト表示の際に面積4倍に拡大**するスイッチで，スプライト定義には変更はありません．MAG＝"1"のときは，スプライトの1ドットの面積も4倍になります．

スプライト属性テーブルは，32面のスプライト面に1対1に対応していて，そのスプライト面に表示する

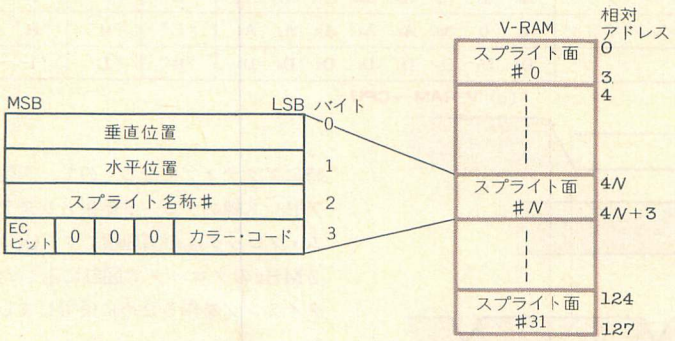
〈図17〉16×16画素のスプライトのパターン





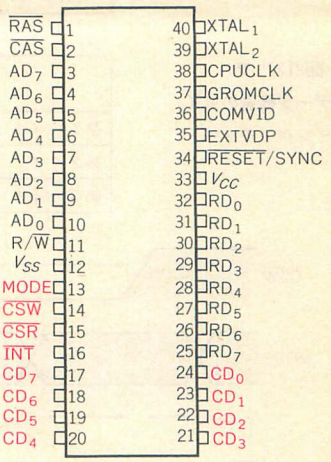
〈写真1〉製作したAppleII用VDPボード

〈図18〉スプライト属性テーブル



〈図19〉[2] VDPのピン配置

(注：MSB＝ビット0と表記されている)



スプライト名称#，色指定と表示座標を設定するものです(図18)．スプライト属性テーブルの先頭が，スプライト面#0の設定であり，これの優先度が最も高くなります．

スプライト表示の優先順位は，スプライト名称#ではなく，スプライト面#，すなわちスプライト属性テーブルの順番になることに注意してください．垂直，水平位置は画面左上が〔－1(＄FF)，0〕で右下が(190，255)です．一つのスプライト面#の垂直位置に＄D0を書き込むと，これ以後のスプライト面は表示されません．

カラー・コードはスプライトの色指定で，スプライトのパターンの“1”の部分の色を設定します．“0”の部分は透明となります．ECビット(Early Clock ビット)を“1”とすると，そのスプライト面が32画素分左にシストします．

レジスタ#1のSIZE＝“1”とした場合，スプライト属性テーブルのスプライト名称#には，スプライトを構成する4個のスプライト名称#(図17)のうち一つを指定します．

CPUとのインターフェース

VDPとCPUのインターフェースは，8ビット・パラレルで行います．CPUと接続するピンは図19のうちMODE，$\overline{CSW}$，$\overline{CSR}$，CD_0～CD_7および$\overline{INT}$です．CPUとのパラレル・データ転送の形式を図20に，この時のタイミング・チャートを図21に示します．

図20(a)は，VDPレジスタへの書き込み手順です．データに引き続き，レジスタ#をLSB 3ビットで転送します．図20(b)，(d)は，V-RAMのアクセスの手順です．V-RAMアドレスを2バイトに分けて転送して，次にデータの転送をします．

V-RAMのアドレスは，一度設定されると次のデータのアクセス，すなわち，MODE＝“L”としたV-RAMデータの転送を繰り返すことによって，自動的にインクリメントされます．

VDPとの1データ転送の際に，2μsの遅延時間が必要です．またV-RAMをアクセスする操作には，最悪で6μsの遅延がさらに必要です．

AppleII用VDPボード

図22にAppleIIの50ピン・バス拡張スロット用のVDPボード例を示します．$\overline{DEV.SEL}$は，拡張スロットに割り当てられた番地のアクセス

〈図20〉(2) ▶
CPUとデータ転送
(注：MSB＝ビット0と表記されている)

	ビット MSB 0	1	2	3	4	5	6	LSB 7	コントロール信号 $\overline{CSW}$	$\overline{CSR}$	MODE
第1バイト：データ	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	"L"	"H"	"H"
第2バイト：レジスタの選択	1	0	0	0	0	RS_0	RS_1	RS_2	"L"	"H"	"H"

(a) CPU→VDPレジスタ

	ビット MSB 0	1	2	3	4	5	6	LSB 7	コントロール信号 $\overline{CSW}$	$\overline{CSR}$	MODE
第1バイト：アドレス・セット・アップ	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}	A_{13}	"L"	"H"	"H"
第2バイト：アドレス・セット・アップ	0	1	A_0	A_1	A_2	A_3	A_4	A_5	"L"	"H"	"H"
第3バイト：データ	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	"L"	"H"	"L"

(b) CPU→V-RAM

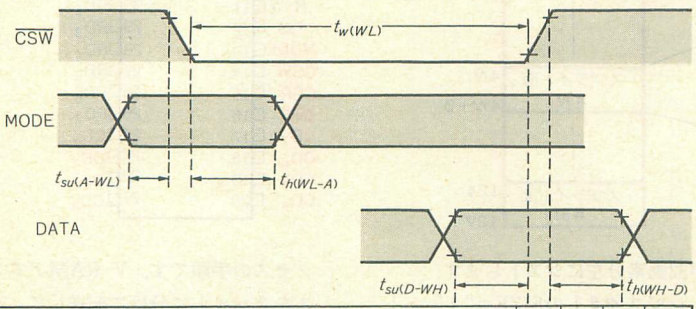
	ビット MSB 0	1	2	3	4	5	6	LSB 7	コントロール信号 $\overline{CSW}$	$\overline{CSR}$	MODE
ステータス・レジスタのデータ	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	"H"	"L"	"H"

(c) VDPステータス・レジスタ→CPU

〈図21〉(2) ▼
データ転送のタイミング・チャート

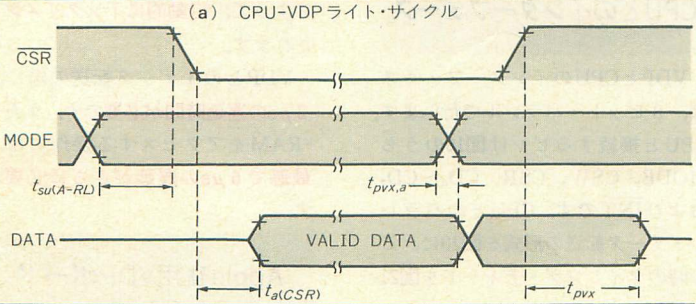
	ビット MSB 0	1	2	3	4	5	6	LSB 7	コントロール信号 $\overline{CSW}$	$\overline{CSR}$	MODE
第1バイト：アドレス・セット・アップ	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}	A_{13}	"L"	"H"	"H"
第2バイト：アドレス・セット・アップ	0	0	A_0	A_1	A_2	A_3	A_4	A_5	"L"	"H"	"H"
第3バイト：データ	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	"H"	"L"	"L"

(d) V-RAM→CPU



PARAMETER	min	typ	max	unit
t_{su}(ARL) Address setup time before CSR low		0		ns
t_{su}(AWL) Address setup time before CSW low		2	30	ns
t_h(WLA) Address hold time after CSW low		30	60	ns
t_{su}(DWH) Data setup time before CSW high		20	100	ns
t_h(WHD) Data hold time after CSW high		−8	30	ns
t_w(WL) Pulse width, CSW low	180	200		ns
t_w(CSH1) Pulse width, chip select high (requesting memory access)		8		μs
t_w(CSH2) Pulse width, chip select high (not requesting memory access)		3		μs

(a) CPU-VDPライト・サイクル



PARAMETER		TEST CONDITIONS	min	typ	max	unit
t_a(CSR)	Data access time from $\overline{CSR}$ low	C_L=300pF		100	150	ns
t_{pvz}	Data disable time after $\overline{CSR}$ high			65	100	ns
$t_{pvx,a}$	Data invalid time from address changes			0		ns
f_{CPUCLK}	CPU clock output frequency (f_{ext}÷3)			3.58		MHz
$f_{GROMCLK}$	GROM clock output frequency (f_{ext}÷24)			447.5		kHz

(b) CPU-VDPリード・サイクル

時にアクティブになるもので，上位アドレス線のデコード結果とシステム・クロックϕ_2の論理積です．Q_3は2 MHzのクロックで図21に示したタイミングを作るために使用しました．

VDP制御ソフトウェア

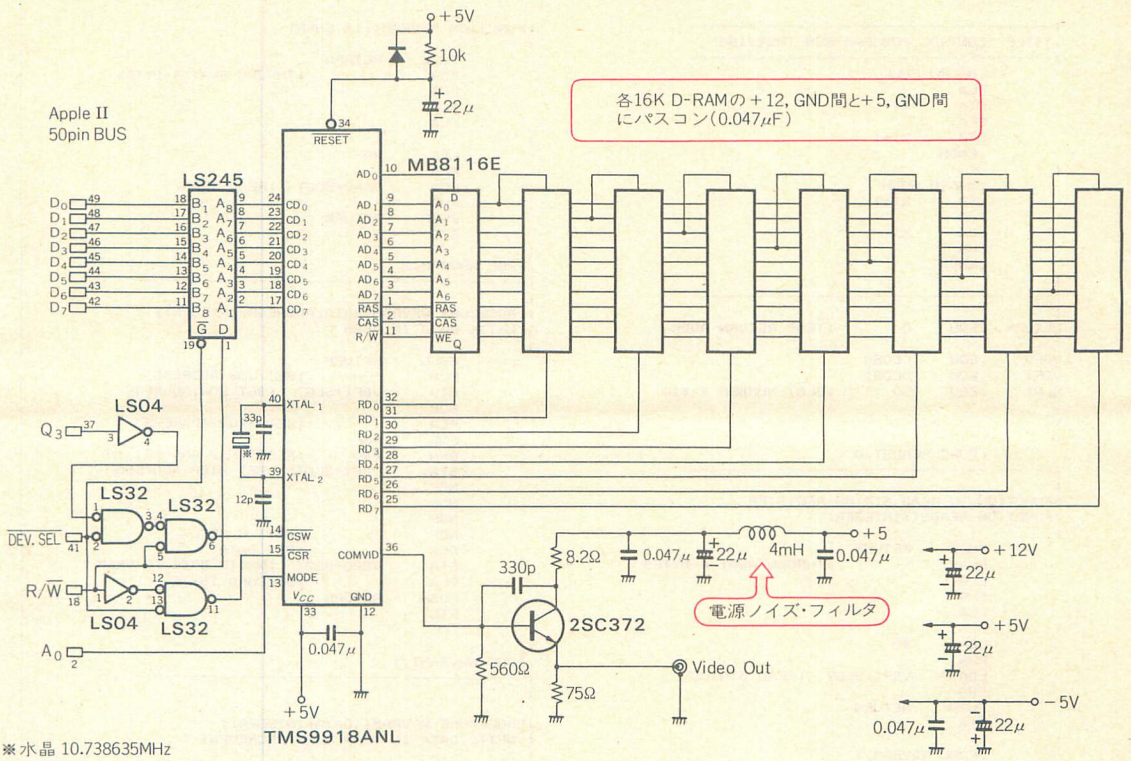
VDPの内部レジスタとV-RAMとのデータ転送を行うための基本的なアセンブラ・ルーチンを，**リスト1**に示します．Apple IIのUCSD-Pascalシステムで開発したもので，Pascalから呼び出して使用します．

Pascalとアセンブラ・ルーチンの引数と結果の受け渡しは，図23のように6502のスタックを介して行います．

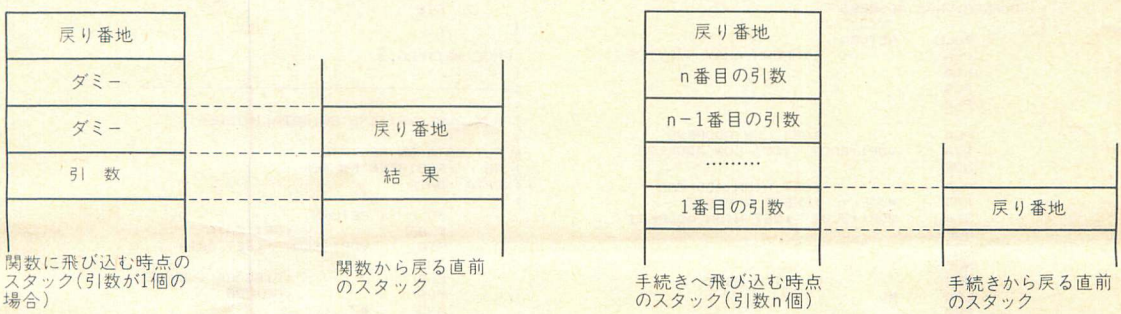
リスト1に示した基本ルーチンは，

①関数ＲＥＡＤＳＴ
　ステータス・レジスタの読み込み
②関数ＲＤＶＲＡＭ
　V-RAMのアドレス指定と読み込み
③関数ＲＤＶＲＡＭＩ
　V-RAMの読み込み(アドレスは

〈図22〉 AppleII用VDPボード



〈図23〉[4] Apple UCSD-Pascalの引数の渡し方



（a） 関数の引数と結果の返し方

（b） 手続きへの引数の渡し方

自動インクリメントされる）

④手続きWRVRAM

VRAMのアドレス指定と書き込み

⑤手続きWRVRAMI

V-RAMの書き込み(アドレスは自動インクリメントされる)

⑥手続きWRITERG

VDPのレジスタへの書き込み

の6種類です．いずれのルーチンでも，VDPとのやりとりに2μs，V-RAMのアクセスに8μs以上の遅延時間となるようにNOPを挿入しています．

なお，VDPのアクセスに6502のインデックス・アドレッシングを用いると余分のメモリ読み出し動作が生じ，VDPが誤動作することがあります．

以上の基本ルーチンで，VDPとのデータ転送はすべて可能になるため，この他のVDP制御ルーチンはすべてPascalで記述することができます．例えば，**リスト6**の手続きＩＮＩＴＶＤＰは，VDPの8個のレジスタにデータを書き込み，VDPの初期化を行います．これはVDPをグラフィック・モードIIにして，V-RAMのテーブルを**図4**に示したマッピングにしたがって設定し

〈リスト1〉VDPレジスタとV-RAMのアクセス

```
;----------------------------------------
.TITLE "CONTROL PROGRAM FOR TMS9918A"
;----------------------------------------
        .MACRO PULL
        PLA
        STA     %1
        PLA
        STA     %1+1
        .ENDM
;
        .MACRO PUSH
        LDA     %1+1
        PHA
        LDA     %1
        PHA
        .ENDM
;
;----------------------------------------
;
RETURN  .EQU    0       ;TEMP RETURN ADDR
;
VDP0    .EQU    0C080
VDP1    .EQU    0C081
SLOT    .EQU    50      ;SLOT NUMBER * $10
;
;
;
        .FUNC READST,0
;----------------------------------------
;
;FUNCTION TO READ STATUS REGISTER
;FUNCTION READST:INTEGER;
;
        PULL    RETURN
        PLA             ;THROWN AWAY 4-BYTES
        PLA
        PLA
        PLA
;
        LDA     #0
        PHA
        LDA     VDP1+SLOT  ;READ STATUS
        PHA
        PUSH    RETURN
        RTS
;
        .FUNC RDVRAM,1
;----------------------------------------
;
; FUNCTION TO READ VRAM POINTED BY ADDRESS
;
;FUNCTION RDVRAM(ADDRESS:INTEGER):INTEGER;
; ADDRESS:VRAM ADDRESS
;
        PULL    RETURN
        PLA             ;THROWN AWAY 4-BYTES
        PLA
        PLA
        PLA
;
        PLA             ;GET LOW ADDRESS
        STA     VDP1+SLOT  ;SET LOW ADDRESS
        NOP
        PLA             ;GET HIGH ADDRESS
        AND     #03F    ;RESET 2 MSB'S
        STA     VDP1+SLOT  ;SET HIGH ADDRESS
        NOP
        NOP
        NOP
        NOP             ;WAIT 8 MSEC
        LDA     #0
        PHA
        LDA     VDP0+SLOT  ;READ VRAM
        PHA
        PUSH    RETURN
        RTS
;
;
        .FUNC RDVRAMI,0
;----------------------------------------
;
; FUNCTION TO READ VRAM AUTO INCREMENT
;
;FUNCTION RDVRAMI:INTEGER;
;
        PULL    RETURN
        PLA             ;THROWN AWAY 4-BYTES
        PLA
        PLA
        PLA
;
        LDA     #0
        PHA
        LDA     VDP0+SLOT  ;READ VRAM
        PHA
        PUSH    RETURN
        RTS
;
.PROC WRVRAM,2
;----------------------------------------
;
;PROCEDURE WRVRAM(DATA,ADDRESS:INTEGER);
; WRITE DATA IN VRAM
;
        PULL    RETURN
        PLA             ;GET LOW ADDRESS
        STA     VDP1+SLOT  ;SET LOW ADDRESS
        NOP
        PLA             ;GET HIGH ADDRESS
        AND     #3F
        ORA     #40     ;RESET MSB AND SET 1B
        STA     VDP1+SLOT  ;SET HIGH ADDRESS
        NOP
        NOP
        NOP
        NOP             ;WAIT 8 MSEC
        PLA             ;GET DATA
        STA     VDP0+SLOT  ;WRITE DATA IN VRAM
        PLA             ;DISCARD 1BYTE
        PUSH    RETURN
        RTS
;
;
.PROC WRVRAMI,1
;----------------------------------------
;
;PROCEDURE WRVRAMI(DATA:INTEGER);
; WRITE DATA IN VRAM AUTO INCREMENT
;
        PULL    RETURN
        PLA             ;GET DATA
        STA     VDP0+SLOT  ;WRITE DATA IN VRAM
        PLA             ;DISCARD 1BYTE
        PUSH    RETURN
        RTS
;
;
.PROC WRITERG,2
;
;----------------------------------------
;
;PROCEDURE WRITERG(RG,DATA:INTEGER)
;
; PUT DATA ON REGISTER
; RG   :REGISTER NUMBER
; DATA :DATA
;
        PULL    RETURN
        PLA             ;GET DATA
        STA     VDP1+SLOT  ;SET DATA
        NOP
        PLA             ;DISCARD 1 BYTE
        PLA             ;GET RG
        AND     #7
        ORA     #80     ;SET MSB
        STA     VDP1+SLOT
        PLA             ;DISCARD 1 BYTE
        PUSH    RETURN
        RTS
;
        .END
```

ます．**リスト6**では他にスプライトの定義，移動，衝突検出などのルーチンをPascalで記述しています．

なおVDP本来のスプライト面座標は，左上(0，−1)右下(255，190)となっていますが，本稿のプログラムでは左下(0，0)右上(255，191)としています．

一方，Pascalでは時間がかかるルーチンをアセンブリ言語で用意しました(**リスト2～リスト5**)．以下に示す4種のルーチンです．いずれもグラフィック・モードIIの静止画面(キャラクタ・プレーン)に関するものです．

①手続きＰＬＯＴ１(**リスト2**)

キャラクタ・プレーン上の座標(X，Y)に指定された色の点を打

〈リスト2〉キャラクタ・プレーンに点を打つ手続き

```
;----------------------------------------
.TITLE "CONTROL PROGRAM FOR TMS9918A"
;----------------------------------------
        .MACRO PULL
        PLA
        STA     %1
        PLA
        STA     %1+1
        .ENDM
;
        .MACRO PUSH
        LDA     %1+1
        PHA
        LDA     %1
        PHA
        .ENDM
;
;----------------------------------------
;
RETURN  .EQU    0       ;TEMP RETURN ADDR
;
;
;
.PROC PLOT1,3
;
;----------------------------------------
;
;PROCEDUE TO PLOT '1' ON CHARACTER PLANE
;TMS 9918 IS GRAPHIC II
;PGEN=0,CTABLE=$2000
;
;PROCEDURE PLOT1(X,Y,COLOR);
;
        .REF    RDVRAM,WRVRAM
XSAVE   .EQU    10      ;X SAVE AREA
YSAVE   .EQU    11      ;Y SAVE AREA
BITP    .EQU    12      ;BIT PATTERN SAVE AREA
COLOR   .EQU    13      ;COLOR SAVE AREA
;
        PULL    RETURN+2
        PLA             ;GET COLOR CODE
        STA     COLOR   ;SAVE COLOR CODE
        PLA             ;DISCARD 1 BYTE
        PLA             ;GET Y
        CMP     #192.
        BCC     PL1     ;IF Y>=192 THEN END
        PLA
        PLA
        PLA             ;DISCARD 3 BYTE
        JMP     PLOTEND
PL1     STA     YSAVE   ;YSAVE:=191-Y
        LDA     #191.
        SEC
        SBC     YSAVE
        STA     YSAVE   ;SAVE Y
        PLA             ;DISCARD 1 BYTE
        PLA             ;GET X
        STA     XSAVE   ;SAVE X
        AND     #7      ;MAKE BIT PATTERN FROM X
        TAX
        PLA             ;DISCARD 1 BYTE
        INX
        LDA     #0
        SEC
PL2     ROR     A
        DEX
        BNE     PL2
        STA     BITP    ;SAVE BIT PATTERN

        LDA     XSAVE   ;CALCULATE POINT ADDRESS
        AND     #0F8     ;(Y AND $F8)*32+(X AND $F8)+(Y AND $7)
        STA     XSAVE
        LDA     YSAVE
        AND     #7
        ORA     XSAVE
        STA     XSAVE
        LDA     YSAVE
        AND     #0F8
        LSR     A
        LSR     A
        LSR     A
        STA     YSAVE

;GET PATTERN FROM VRAM

        PHA             ;SET HIGH ADDRESS
        LDA     XSAVE   ;SET LOW ADDRESS
        PHA

        PHA
        PHA
        PHA
        PHA             ;PUSH 4 BYTE
        JSR     RDVRAM
        PLA             ;GET PATTERN FROM VRAM
        ORA     BITP    ;PUT PATTERN
        NOP
        NOP
        NOP
        NOP             ;WAIT 8 MSEC
        PHA             ;SET DATA
        LDA     YSAVE   ;SET HIGH ADDRESS
        PHA
        LDA     XSAVE   ;SET LOW ADDRESS
        PHA
        JSR     WRVRAM  ;WRITE PATTERN IN VRAM
        LDA     COLOR   ;PUT COLOR CODE ON VRAM
        ASL     A       ;'1' COLOR=COLOR CODE   '0' COLOR=0
        ASL     A
        ASL     A
        ASL     A
        PHA             ;SET DATA
        PHA
;
;VRAM ADRS(COLOR)=VRAM ADRS(PATTERN)+$2000
;
        LDA     YSAVE   ;
        CLC
        ADC     #20
        PHA             ;SET HIGH ADDRESS
        LDA     XSAVE
        PHA             ;SET LOW ADDRESS
        JSR     WRVRAM
PLOTEND PUSH    RETURN+2
        RTS
```

ちます．

②手続きＬＩＮＥ(リスト3)

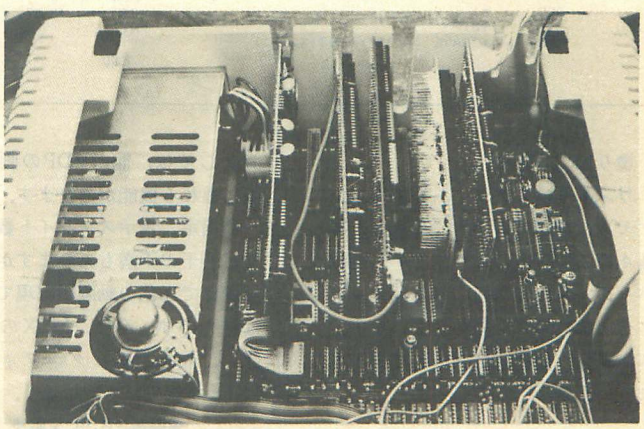
キャラクタ・プレーン上の2点間を指定された色の線で結びます．

③手続きＣＬＳ(リスト4)

キャラクタ・プレーンを消去します．パターン名称テーブルをイニシャライズして，パターン・ジェネレータ・テーブルをリセットし，カラー・テーブルを“1”＝白，“0”＝透明に設定します．

④ＰＡＩＮＴ(リスト5)

キャラクタ・パターンで座標指定された点を含む領域を指定の色で



〈写真2〉Appleの拡張スロットに挿入したところ

〈リスト3〉キャラクタ・プレーンに線を引く手続き

```
;---------------------------------------
.TITLE "CONTROL PROGRAM FOR TMS9918A"
;---------------------------------------
        .MACRO PULL
        PLA
        STA     %1
        PLA
        STA     %1+1
        .ENDM
;
        .MACRO PUSH
        LDA     %1+1
        PHA
        LDA     %1
        PHA
        .ENDM
;
;---------------------------------------
;
RETURN  .EQU    0       ;TEMP RETURN ADDR
;
;
        .PROC   LINE,5
;
;---------------------------------------
;
;PROCEDURE TO DRAW LINE ON CHARACTER PLANE
;
;PROCEDURE LINE(X1,Y1,X2,Y2,COLOR);
;
        .REF    PLOT1
        .DEF    CALPLOT
X1SAVE  .EQU    14
Y1SAVE  .EQU    15
X2SAVE  .EQU    16
Y2SAVE  .EQU    17
XDIF    .EQU    18
YDIF    .EQU    19
XDIR    .EQU    1A
YDIR    .EQU    1B
COLOR   .EQU    1C
RX      .EQU    1D
RY      .EQU    1E
;
        ;GET PARAMETERS
        PULL    RETURN+4
        PLA             ;GET COLOR
        STA     COLOR
        PLA
        PLA             ;GET Y2
        STA     Y2SAVE
        PLA
        PLA             ;GET X2
        STA     X2SAVE
        PLA
        PLA             ;GET Y1
        STA     Y1SAVE
        PLA
        PLA             ;GET X1
        STA     X1SAVE
        PLA
        ;
        LDA     #0      ;SET DIRCTION +,+
        STA     XDIR
        STA     YDIR
        LDA     Y2SAVE  ;YDIF:=ABS(Y2-Y1)
        SEC
        SBC     Y1SAVE
        BCS     LN1
        INC     YDIR    ;YDIR:= -
        EOR     #0FF     ;A:=NEG(A)
        ADC     #1
LN1     STA     YDIF
        LDA     X2SAVE  ;XDIF:=ABS(X2-X1)
        SEC
        SBC     X1SAVE
        BCS     LN2
        INC     XDIR    ;XDIR:= -,
        EOR     #0FF     ;A:=NEG(A)
        ADC     #1
LN2     STA     XDIF
        ORA     YDIF    ;IF POINT1=POINT2 THEN PLOT(POINT1) AND END
        BNE     LN7
        JSR     CALPLOT
        JMP     LINEEND
LN7     LDA     YDIF    ;IF B7 OF YDIF =1 THEN LN8
        AND     #80
        BNE     LN8
        LDA     XDIF    ;IF B7 OF XDIF =1 THEN LN8
        AND     #80
        BNE     LN8
        ASL     YDIF    ;YDIF:=YDIF*2
        ASL     XDIF    ;XDIF:=XDIF*2
        JMP     LN7
        ;
LN8     LDA     #0      ;CLEAR REGISTERS FOR DDA
        STA     RX
        STA     RY
        ;
LN9     LDA     RY      ;RY:=RY+YDIF
        CLC
        ADC     YDIF
        STA     RY
        BCC     LN11    ;IF NO CARRY FROM RY REGISTER THEN LN11
        LDA     YDIR
        BNE     LN10
        INC     Y1SAVE  ;IF YDIR= + THEN INC Y1
        JMP     LN11    ;           ELSE DEC Y1
LN10    DEC     Y1SAVE
;
LN11    LDA     RX      ;RX:=RX+XDIF
        CLC
        ADC     XDIF
        STA     RX
        BCC     LN13    ;IF NO CARRY FROM RX REGISTER THEN LN13
        LDA     XDIR
        BNE     LN12
        INC     X1SAVE  ;IF XDIR= + THEN INC X1
        JMP     LN13    ;           ELSE DEC X1
LN12    DEC     X1SAVE
LN13    JSR     CALPLOT;PLOT(X1,Y1)
        LDA     Y2SAVE  ;IF Y1=Y2 AND X1=X2 THEN END
        CMP     Y1SAVE  ;                   ELSE LN9(CONTINUE PLOTTING)
        BNE     LN9
        LDA     X2SAVE
        CMP     X1SAVE


        BNE     LN9
;
LINEEND PUSH    RETURN+4
        RTS
;
;
;SUBROUTINES FOR CALL PLOT1
CALPLOT LDA     X1SAVE   ;SET PARAMETERS
        PHA
        PHA
        LDA     Y1SAVE
        PHA
        PHA
        LDA     COLOR
        PHA
;       PHA
        JSR     PLOT1   ;CALL PLOT1
        RTS
;
        .END
```

塗りつぶします．

リスト6はスプライトを使用したデモ・プログラムです．**写真3**と**写真4**はこのデモのようすです．空はバック・ドロップ面，家並はキャラクタ・プレーンに描かれています．車，電柱，雲はスプライトで表示されています．

今回紹介できなかったVDPの機能としては，他のVDPの画面を背後に重ねる機能があります．

色指定，表示スプライト数の制限など若干の不満もありますが，簡単な操作で16色の動画が実現できる点は魅力です．今後，MSXの普及に伴い，最も手軽なグラフィック用LSIとなっていくでしょう．

◆参考・引用*文献◆

(1)MSXシステムの全貌，月刊アスキー，1983年8月号，p.110～

(2)*日本テキサスインスツルメンツ㈱，ビデオディスプレイプロセッサユーザーズマニュアル，1982年

(3)日本テキサスインスツルメンツ㈱，ビデオディスプレイプロセッサ　システムデザインハンドブック，初版，共立出版，1983年

(4)*PASCAL入門，マイコンピュータ，CQ出版社，1982年4月号，p.89～

〈リスト4〉キャラクタ・プレーンを消去する手続き

```
;----------------------------------------
.TITLE "CONTROL PROGRAM FOR TMS9918A"
;----------------------------------------
        .MACRO PULL
        PLA
        STA     %1
        PLA
        STA     %1+1
        .ENDM
;
        .MACRO PUSH
        LDA     %1+1
        PHA
        LDA     %1
        PHA
        .ENDM
;
;----------------------------------------
;
RETURN  .EQU    0       ;TEMP RETURN ADDR
;
;
;
        .PROC   CLS,0

;
;----------------------------------------
;
;PROCEDURE TO CLEAR SCREEN(INCLUDE INITIALIZE SCREEN)
;
;PROCEDURE CLS
;
        .REF    WRVRAM,WRVRAMI
PGEN    .EQU    0               ;TOP OF PATTERN GENERATER
PNAME   .EQU    14336.          ;TOP OF PATTERN-NAME
CTABLE  .EQU    8192.           ;TOP OF COLOR TABLE
COUNTER .EQU     10
        PULL    RETURN+2
        ;INITIALIZE PATTERN-NAME TABLE
        LDA     #767. % 256.    ;SET COUNTER 767
        STA     COUNTER
        LDA     #767./256.
        STA     COUNTER+1
        LDA     #0              ;VRAM(PNAME):=0
        PHA
        PHA
        LDA     #PNAME/256.     ;SET ADDRESS PNAME
        PHA
        LDA     #PNAME % 256.
        PHA
        JSR     WRVRAM          ;WRITE VRAM
        LDX     #1              ;VRAM(PNAME+I):=I
CLS1    TXA                     ;SET I
        PHA
        PHA
        JSR     WRVRAMI         ;WRITE VRAM AUTO INCREMENT
        INX                     ;INC I
        SEC
        LDA     COUNTER         ;DEC COUNTER
        SBC     #1
        STA     COUNTER
        LDA     COUNTER+1
        SBC     #0
        STA     COUNTER+1
        ORA     COUNTER
        BNE     CLS1            ;IF COUNTER<>0 THEN CLS1
        LDA     #6143. % 256.   ;SET COUNTER 6143
        STA     COUNTER
        LDA     #6143./256.
        STA     COUNTER+1
        LDA     #0              ;VRAM(PGEN):=0
        PHA
        PHA
        LDA     #PGEN/256.      ;SET ADDRESS
        PHA
        LDA     #PGEN % 256.
        PHA
        JSR     WRVRAM          ;WRITE VRAM
CLS2    LDA     #0              ;VRAM(PGEN+I):=0
        PHA
        PHA
        JSR     WRVRAMI         ;WRITE VRAM AUTO INCREMENT
        SEC
        LDA     COUNTER         ;DEC COUNTER
        SBC     #1
        STA     COUNTER
        LDA     COUNTER+1
        SBC     #0
        STA     COUNTER+1
        ORA     COUNTER         ;IF COUNTER<>0 THEN CLS2
        BNE     CLS2
        LDA     #6143. % 256.   ;SET COUNTER 6143
        STA     COUNTER
        LDA     #6143./256.
        STA     COUNTER+1
        LDA     #0F0            ;VRAM(CTABLE):=$F0  1 =WHITE  0=THROUGH
        PHA
        PHA
        LDA     #CTABLE/256.    ;SET ADDRESS
        PHA
        LDA     #CTABLE % 256.
        PHA
        JSR     WRVRAM          ;WRITE VRAM
CLS3    LDA     #0F0            ;VRAM(CTABLE+I):=$F0  1=WHITE  0=THROUGH
        PHA
        PHA
        JSR     WRVRAMI         ;WRITE VRAM AUTO INCREMENT
        SEC
        LDA     COUNTER         ;DEC COUNTER
        SBC     #1
        STA     COUNTER
        LDA     COUNTER+1
        SBC     #0
        STA     COUNTER+1
        ORA     COUNTER         ;IF COUNTER<>0 THEN CL3
        BNE     CLS3
        PUSH    RETURN+2
        RTS

        .END
```

〈リスト5〉キャラクタ・プレーンを塗りつぶす手続き

```
;---------------------------------------
.TITLE "CONTROL PROGRAM FOR TMS9918A"
;---------------------------------------;
        .MACRO PULL
        PLA
        STA     %1
        PLA
        STA     %1+1
        .ENDM
        ;
        .MACRO PUSH
        LDA     %1+1
        PHA
        LDA     %1
        PHA
        .ENDM
;-------------------------------------
;
RETURN .EQU     0       ;TEMP RETURN ADDR
;
        .MACRO  RGSTP   ;MACRO FOR REGIST POINT
        JSR     SCREEN  ;SCREEN(X,Y)
        BNE     %1      ;IF SCREEN=1 THEN SKIP
        LDX     #TNPTA  ;((NNPT)+TNPT):=X
        LDY     #NNPT
        JSR     ADD16
        LDY     #0
        LDA     XSAVE
        STA     (WORK),Y
        INY             ;((NNPT)+TNPT+1):=Y
        LDA     YSAVE
        STA     (WORK),Y
        LDX     #NNPT   ;(NNP):=(NNP)+2
        JSR     INC2
        JSR     CALPLOT ;PLOT(X,Y,COLOR)
        .ENDM
;
        .proc   PAINT,3
;
;-------------------------------------
;
;procedure to paint character plane
;
;procedure (X,Y,COLOR);
;
        .REF    CALPLOT
        .REF    RDVRAM;
XSAVE   .EQU    14      ;X SAVE AREA
YSAVE   .EQU    15      ;Y SAVE AREA
NOPT    .EQU    16      ;NUMBER OF OLD POINT
NNPT    .EQU    18      ;NUMBER OF NEW POINT
INDEX   .EQU    1A      ;INDEX
COLOR   .EQU    1C      ;COLOR SAVE AREA
WORK    .EQU    1D
WORK2   .EQU    1F
TOPTA   .EQU    21
TNPTA   .EQU    23
;
        PULL    RETURN+4
        LDA     TOP     ;(TOPA):=TOPT   (TNPA):=TNPT
        STA     TOPTA
        LDA     TOP+1
        STA     TOPTA+1
        LDA     TNP
        STA     TNPTA
        LDA     TNP+1
        STA     TNPTA+1
;
        PLA             ;GET COLOR
        STA     COLOR
        PLA
        PLA             ;GET Y
        STA     YSAVE
        PLA
        PLA             ;GET X
        STA     XSAVE
        PLA
        JSR     CALPLOT ;PLOT(X,Y,COLOR)
        LDA     XSAVE   ;(TOPT):=X,Y    TOPT:TOP OF OLD POINT
        STA     TOPT
        LDA     YSAVE
        STA     TOPT+1
        LDA     #2      ;(NOPT):=2
        STA     NOPT
        LDA     #0
        STA     NOPT+1
PA1     LDA     #0      ;(NNPT):=0
        STA     NNPT
        STA     NNPT+1
        STA     INDEX   ;(INDEX):=0
        STA     INDEX+1
        ;
PA10    LDX     #INDEX  ;X:=((INDEX)+TOPT)
        LDY     #TOPTA
        JSR     ADD16
        LDY     #0
        LDA     (WORK),Y
        STA     XSAVE
        INY             ;Y:=((INDEX)+TOPT+1)
        LDA     (WORK),Y
        STA     YSAVE

        ;IF SCREEN(X,Y+1)=0 THEN REGIST POINT(X,Y+1) AND PLOT
        INC     YSAVE
        RGSTP   PA2
        ;
        ;IF SCREEN(X,Y-1)=0 THEN REGIST POINT(X,Y-1) AND PLOT
PA2     DEC     YSAVE
        DEC     YSAVE
        RGSTP   PA3
        ;
        ;IF SCREEN(X+1,Y)=0 THEN REGIST POINT(X+1,Y) AND PLOT
PA3     INC     YSAVE
        INC     XSAVE
        RGSTP   PA4
        ;
        ;IF SCREEN(X-1,Y)=0 THEN REGIST POINT(X-1,Y) AND PLOT
PA4     DEC     XSAVE
        DEC     XSAVE
        RGSTP   PA5
        ;
PA5     LDX     #INDEX  ;(INDEX):=(INDEX)+2
        JSR     INC2
        LDA     INDEX   ;IF (INDEX)<>(NOPT) THEN PA10
        CMP     NOPT
        BNE     PA6
        LDA     INDEX+1
        CMP     NOPT+1
        BNE     PA6
        JMP     PA7
PA6     JMP     PA10
```

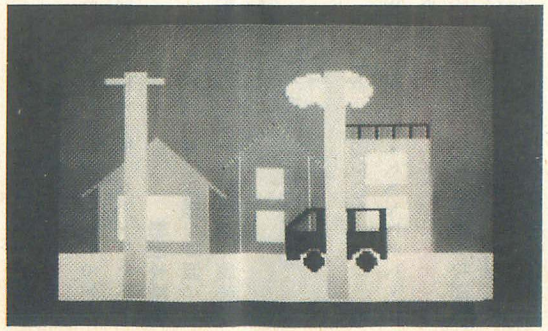
```
PA7     LDA     NNPT    ;IF (NNPT):=0 THEN END ELSE PA8
        ORA     NNPT+1
        BNE     PA8
        PUSH    RETURN+4
        RTS
        ;
PA8     LDA     TOPTA   ;FOR (INDEX):=0 TO (NNPT)-1 DO
        STA     WORK    ;     ((INDEX)+TOPT):=((INDEC)+TNPT)
        LDA     TOPTA+1 ;(WORK):=TOPT
        STA     WORK+1
        LDA     TNPTA   ;(WORK2):=TNPT
        STA     WORK2
        LDA     TNPTA+1
        STA     WORK2+1
        LDA     #0
        STA     INDEX   ;(INDEX):=0
        STA     INDEX+1
PA9     LDY     #0
        LDA     (WORK2),Y       ;((WORK)):=((WORK))
        STA     (WORK),Y
        INY
        LDA     (WORK2),Y       ;((WORK)+1):=((WORK)+1)
        STA     (WORK),Y
        LDX     #WORK
        JSR     INC2            ;(WORK)=(WORK)+2
        LDX     #WORK2
        JSR     INC2            ;(WORK2):=(WORK2)+2
        LDX     #INDEX
        JSR     INC2            ;(INDEX):=(INDEX)+2
        ;
        LDA     NNPT            ;IF (NNPT)<>(INDEX) THEN PA9
        CMP     INDEX
        BNE     PA9
        LDA     NNPT +1
        CMP     INDEX+1
        BNE     PA9
        ;
        LDA     NNPT            ;(NOPT):=NNPT)
        STA     NOPT
        LDA     NNPT+1
        STA     NOPT+1
        JMP     PA1
        ;
TOP     .WORD   TOPT
TNP     .WORD   TNPT
TOPT    .BLOCK  1000.
TNPT    .BLOCK  1000.
;
;SUBROUTINE SCREEN
;
SX      .EQU    10
SY      .EQU    11
BITP    .EQU    12
;
SCREEN  LDA     XSAVE   ;GET PARAMETER
        STA     SX
        LDA     YSAVE
        STA     SY
        LDA     #191.   ;Y:=191-Y
        SEC
        SBC     SY
        STA     SY
        ;
        LDA     SX      ;MAKE BIT PATTERN FROM X
        AND     #7
        TAX
        INX
        LDA     #0
        SEC
SC2     ROR     A
        DEX
        BNE     SC2
        STA     BITP    ;SAVE BIT PATTERN
        ;
        LDA     SX      ;CALCULATE POINT ADDRESS
        AND     #0F8    ;ADDRESS=(Y AND $F8)*32+(X AND $F8)+(Y AND $7)
        STA     SX
        LDA     SY
        AND     #7
        ORA     SX
        STA     SX
        LDA     SY
        AND     #0F8
        LSR     A
        LSR     A
        LSR     A
        PHA             ;SET POINT ADRS
        LDA     SX
        PHA
        ;
        PHA
        PHA
        PHA
        PHA             ;PUSH 4 BYTE
        ;
        JSR     RDVRAM  ;READ PATTERN FROM VRAM
        ;
        PLA             ;GET PATTERN
        PLP
        AND     BITP
        RTS
;
;SUBROUTINE ADD16
;
ADD16   CLC
        LDA     0,X
        ADC     0,Y
        STA     WORK
        LDA     1,X
        ADC     1,Y
        STA     WORK+1
        RTS
;
;SUBROUTINE INC2
;
INC2    CLC
        LDA     0,X
        ADC     #2
        STA     0,X
        LDA     1,X
        ADC     #0

        STA     1,X
        RTS
;
        .END
```

〈リスト6〉 Pascalで記述したVDP制御のサブルーチンとメイン・プログラム

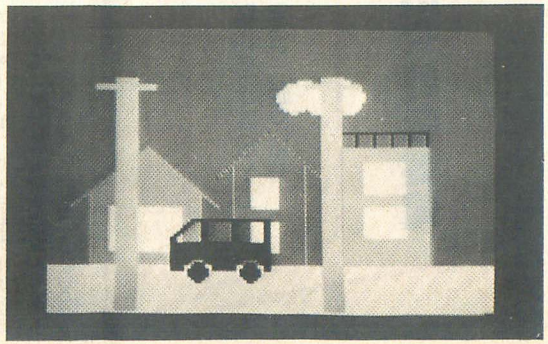
```
Program VDPTEST;
 uses applestuff;
 const PGEN=0;SPGEN=6144;SPATT=7168;CTABLE=8192;PNAME=14336;
 var I,J,K:integer;
     MAG,SIZE:(large,small);
     FLAG1:boolean;

 (*MACINE LANGUAGE SUBROUTINES*)                          機械語プログラムの宣言

 function  READST :integer;external;                      ステータス・レジスタを読む
 function  RDVRAM (ADDRESS:integer):integer;external;    ┐
 function  RDVRAMI:integer;external;                      ├ VDPのVRAMのデータを読む
 procedure WRVRAM (DATA,ADDRESS:integer);external;        ┐
 procedure WRVRAMI(DATA:integer);external;                ├ VDPのVRAMにデータを書き込む
 procedure WRITERG(RG,DATA:integer);external;             VDPのレジスタにデータを書き込む
 procedure PLOT1(X,Y,COLOR:INTEGER);external;             点を書く
 procedure LINE(X1,Y1,X2,Y2,COLOR:integer);external;      線を引く
 procedure CLS;external;                                  スクリーン(キャラクタ面)をクリアする
 procedure PAINT(X,Y,COLOR:integer);external;             色を塗る

 procedure INITVDP;
  var I:integer;
  begin
   (*initialize of registers*)                            VDPのレジスタを初期化する
   WRITERG(0,2);(* graphic ii *)
   WRITERG(1,12*16);(* 16k,on,int off,size 8s*)
   WRITERG(2,14);(*pat-name table 14336*)
   WRITERG(3,255);(*color table 8192*)
   WRITERG(4,3);(*pat-gen table 0*)
   WRITERG(5,3*16+8);(*sp-att table 7168*)
   WRITERG(6,3);(*sp-gen table 6144*)
   WRITERG(7,15*16+1);(*text & back color $f1*)
   (*INITIALIZE SWITCHES*)
   MAG:=small;SIZE:=small;
  end;


 procedure MAGON;
  (*MAG SWITCH ON*)                                       レジスタ#1 MAGビットをONにする
  begin
   if MAG=small then if SIZE=smalL then begin MAG:=large;WRITERG(1,12*16+1) end
                                   else begin MAG:=large;WRITERG(1,12*16+3) end
  end;

 procedure MAGOFF;
  (*MAG SWITCH OFF*)                                      レジスタ#1 MAGビットをOFFにする
  begin
   if MAG=large then if SIZE=small then begin MAG:=small;WRITERG(1,12*16) end
                                   else begin MAG:=small;WRITERG(1,12*16+2) end
  end;

 procedure SIZEON;
  (*SIZE SWITCH ON*)                                      レジスタ#1 SIZEビットをONにする
  begin
   if SIZE=small then
      if MAG=small then begin SIZE:=small;WRITERG(1,12*16+2) end
                   else begin SIZE:=large;WRITERG(1,12*16+3) end
  end;

 procedure SIZEOFF;
  (*SIZE SWITCH OFF*)                                     レジスタ#1 SIZEビットをOFFにする
  begin
   if SIZE=large then
      if MAG=small then begin SIZE:=small;WRITERG(1,12*16) end
                   else begin SIZE:=small;WRITERG(1,12*16+1) end
  end;

 procedure SETSP(SPNAME,PAT1,PAT2,PAT3,PAT4,PAT5,PAT6,PAT7,PAT8:integer);
  (*SET SPRITE PATTERN*)                                  スプライトにスプライトのパターンを書き込む
  begin
   WRVRAM(PAT1,SPGEN+SPNAME*8);WRVRAMI(PAT2);
   WRVRAMI(PAT3);WRVRAMI(PAT4);
   WRVRAMI(PAT5);WRVRAMI(PAT6);
   WRVRAMI(PAT7);WRVRAMI(PAT8);
  end;

 procedure PUTSP(SPNAME,SPPLANE,X,Y,COLOR:integer);
  (*PUT SPRITE ON SPRITE PLANE*)                          スプライトをスプライト面上に置く
  begin
   WRVRAM(190-Y,SPATT+SPPLANE*4);
   WRVRAMI(X);WRVRAMI(SPNAME);
   WRVRAMI(COLOR);
   end;

 procedure MOVESP(SPPLANE,X,Y:integer);                   スプライトを動かす
  (*MOVE SPRITE*)
  begin
   WRVRAM(190-Y,SPATT+SPPLANE*4);
   WRVRAMI(X);
  end;

 procedure CHANGESP(SPNAME,SPPLANE:integer);              スプライト面に置くスプライト番号を変更する
  (*CHANGE SPRITE NAME ON SPRITE PLANE*)
  begin
   WRVRAM(SPNAME,SPATT+SPPLANE*4+2)
  end;

 procedure ENDSPP(SPPLANE:integer);                       指定されたスプライト面以降を無効にする
  (*SET end OF SPRITE PLANE*)
  begin
   WRVRAM(13*16,SPATT+SPPLANE*4);
  end;

 procedure BACKCOLOR(COLOR:INTEGER);                      バック・ドロップ面の色を指定する
  (*SET BACK DROP COLOR*)
  begin
   WRITERG(7,COLOR)
  end;

 function COLLISION:boolean;                              衝突フラグを読む
  (* if collision of splites then true*)
  begin
   COLLISION:=odd(READST div 32);
  end;
 procedure setpattern;                                    自動車，電柱，雲のパターンをスプライトに
  (*set pattern for demo*)                                設定する
  begin(*PATTERN CAR*)
  SETSP( 0,  0,  0,  0,  0,  0,  1,  3);
  SETSP( 1,  6, 12, 24, 24, 31, 31, 31);
  SETSP( 2,  0,  0,  0,  0,127,204,140, 12);
  SETSP( 3, 12, 12, 12, 12,255,255,255,255);

  SETSP( 4,  0,  0,  0,  0,255,  6,  6,  6);
  SETSP( 5,  6,  6,  6,  6,255,255,255,255);
  SETSP( 6,  0,  0,  0,  0,255,  3,  3,  3);
  SETSP( 7,  3,  3,  3,  3,255,255,255,255);
```

```
SETSP( 8, 31, 31, 31, 31, 30, 30,  0,  0);
SETSP( 9,  0,  0,  0,  0,  0,  0,  0,  0);
SETSP(10,255,255,135, 51,121,253,252,252);
SETSP(11,120, 48,  0,  0,  0,  0,  0,  0);

SETSP(12,255,255,255,254,252,253,  1,  1);
SETSP(13,  0,  0,  0,  0,  0,  0,  0,  0);
SETSP(14,255,255, 15,103,243,251,248,248);
SETSP(15,240, 96,  0,  0,  0,  0,  0,  0);

(* end CAR*)

(*POLE*)
SETSP(16,  0,  0,  7,  7,255,255,  7,  7);
SETSP(17,  7,  7,  7,  7,  7,  7,  7,  7);
SETSP(18,  0,  0,224,224,255,255,224,224);
SETSP(19,224,224,224,224,224,224,224,224);

SETSP(20,  7,  7,  7,  7,  7,  7,  7,  7);
SETSP(21,  7,  7,  7,  7,  7,  7,  7,  7);
SETSP(22,224,224,224,224,224,224,224,224);
SETSP(23,224,224,224,224,224,224,224,224);
(*END POLE*)

(*CLOUD*)
SETSP(24,  0,  0,  0,  0,  1,  3,  7, 15);
SETSP(25, 15, 15,  7,  7,  3,  0,  0,  0);
SETSP(26,  0,  0,  0, 60,254,254,255,255);
SETSP(27,255,255,255,255,247,195,  0,  0);

SETSP(28,  0,  0, 14, 31,127,127,255,255);
SETSP(29,255,255,255,255,253,241,112,  0);
SETSP(30,  0,  0,  0,128,224,240,240,248);
SETSP(31,248,248,240,240,224,192,  0,  0);
(*END CLOUD*)
end;

begin(*main*)
 INITVDP;MAGON;SIZEON;
 CLS;(*CLEAR SCREEN*)
 BACKCOLOR(5);(*SET BACK DROP COLOR*)
 SETPATTERN;

 PUTSP(0,8, 0,90,1);PUTSP(4,9,32,90,1);
 PUTSP(8,10,0,58,1);PUTSP(12,11,32,58,1);
```

```
PUTSP(20,0,33,52,14);PUTSP(20,1,33,52+32,14);PUTSP(20,2,33,52+32*2,14);
PUTSP(20,3,33,52+32*3,14);PUTSP(16,12,33,52+32*4,14);

PUTSP(20,4,150,52,14);PUTSP(20,5,150,52+32,14);PUTSP(20, 6,150,52+32*2,14);
PUTSP(20, 7,150,52+32*3,14);PUTSP(16,13,150,52+32*4,14);
PUTSP(24,14,200,180,15);PUTSP(28,15,200+32,180,15);

ENDSPP(16);

LINE(0,50,255,50,15);LINE(0,20,255,20,15);LINE(0,50,0,21,15);   地面，道路を描く
LINE(255,50,255,21,15);LINE(0,20,0,0,1);LINE(255,20,255,0,1);
LINE(0,0,255,0,1);
PAINT(100,40,15);PAINT(100,10,1);
(*HOUSE LEFT*)
LINE(20,90,60,130,3);LINE(20,91,60,131,3);LINE(60,130,100,90,3);  左の家を描く
LINE(60,131,100,91,3);LINE(30,100,30,51,3);


LINE(90,100,90,51,3);LINE(40,90,80,90,15);LINE(80,90,80,60,15);
LINE(80,60,40,60,15);LINE(40,60,40,90,15);PAINT(60,80,15);
PAINT(60,100,3);
(*HOUSE CENTER*)                                          中央の家を描く
LINE(130,140,100,110,8);LINE(130,140,160,110,8);LINE(130,141,100,111,8);
LINE(130,141,160,111,8);LINE(110,120,110,51,8);LINE(150,120,150,51,8);
LINE(120,110,140,110,15);LINE(140,110,140,90,15);LINE(140,90,120,90,15);
LINE (120,90,120,110,15);LINE(120,80,140,80,15);LINE(140,80,140,60,15);
LINE(140,60,120,60,15);LINE(120,60,120,80,15);
PAINT(130,100,15);PAINT(130,70,15);PAINT(130,120,8);
(*HOUSE RIGHT*)                                           右の家を描く
LINE(170,130,220,130,10);LINE(220,130,220,51,10);LINE(170,130,170,51,10);
LINE(180,120,210,120,15);LINE(210,120,210,100,15);LINE(210,100,180,100,15);
LINE(180,100,180,120,15);
LINE(180,90,210,90,15);LINE(210,90,210,70,15);LINE(210,70,180,70,15);
LINE(180,70,180,90,15);
FOR I:=0 TO 5 DO LINE(170+I*10,131,170+I*10,140,1);
LINE(170,140,220,140,1);
PAINT(200,110,15);PAINT(200,80,15);PAINT(200,125,10);
I:=0;REPEAT                                               自動車と雲を動かす
MOVESP(9,I+32,90);MOVESP(8,I,90);MOVESP(11,I+32,58);MOVESP(10,I,58);
MOVESP(15,200-(J DIV 20)+32,180);MOVESP(14,200-(J DIV 20),180);
I:=(I-1)-((I-1) DIV 256)*256;
J:=(J+1)-((J+1) DIV (256*20))*(256*20);
write(' ')
until keypress;
end.
```



〈写真3〉
Pascalで記述したデモ・プログラムによる画面



〈写真4〉
デモ・プログラムによる画面
（自動車と雲が移動）