

 **commodore**



VIC-1001

USER'S MANUAL

PERSONAL COMPUTER

by **commodore**

VIC-1001

PERSONAL COMPUTER USER'S MANUAL

 **commodore**

●注意●

- (1)本書の無断転載は禁止いたします。
- (2)内容は予告なく変更することがあります。
- (3)本書の内容は十分にチェックしておりますが、万一誤り、記載もれ、御不審な点など、お気づきのことがありましたら御連絡下さい。
- (4)(3)項にかかわらず、使用した結果の影響については、責任を負いかねますので御了承下さい。

はじめに

本書には、VIC-1000シリーズ・パーソナル・コンピューター・システムを初めて使用される方のために、コンピューターの基本的なしくみ、使用方法、ハードウェアの構成、VIC-1001パーソナル・コンピューターのプログラミング言語CBM BASIC (VERSION 2) 等に関する説明がなされています。

各種のオプションを付加することにより、あらゆる目的に用いることのできるVIC-1000シリーズ・パーソナル・コンピューター・システムは、TVゲーム、パズル、個人的情報管理から各種科学技術計算、各種ビジネスデータの処理、シミュレーション等々まで、幅広い対応が可能となります。

はじめに

第1部	ハードウェア編	9
第1章	コンピューターのしくみ	10
第2章	VIC-1001 パーソナル・コンピュータ	12
第1節	VIC-1000シリーズ・システム構成	12
第2節	VIC-1000シリーズ・ファミリー	14
第3章	VIC-1001の使用方法	16
第1節	電源投入の前に	16
第2節	電源の投入	18
第3節	ディスプレイ	18
1.	画面の文字数	18
2.	カラー	18
3.	キャラクター・コード	19
第4節	キーボード	23
1.	キーボード配列	23
2.	キャラクター・モードの選択	24
3.	特殊キーの働き	30
第4章	BASICによるプログラミング	32
第1節	BASICとは	32
第2節	プログラムの書き方	32
(1)	ダイレクト・モード	33
(2)	プログラム・モード	35
第3節	プログラムの修正	37
第4節	スクリーン・エディター	41
第5章	カセット・ドライブの使い方	43
第1節	プログラムのSAVE	43
第2節	プログラムのVERIFY	44
第3節	プログラムのLOAD	45
第4節	データのLOADおよびSAVE	46

第6章	ハードウェアの説明	51
第1節	概論	51
第2節	外観	51
第3節	システム構成	52
第4節	機能仕様	54
第5節	メモリー・アドレスマップ	56
第6節	キーボード	58
第7節	インターフェイス	63
第8節	CRT制御方式	67

第2部 ソフトウェア編 69

第1章	CBM BASICの概要	70
第1節	初期状態	70
第2節	動作モード	70
第3節	ラインの書式	71
第4節	ライン・ナンバー	71
第5節	キャラクター・セット	71
第6節	定数	74
	(1)整数形式	74
	(2)浮動小数点形式	74
第7節	変数	75
	(1)変数名および型宣言文字	75
	(2)配列変数	75
第8節	型の変換	76
第9節	式と演算	77
	(1)算術演算	77
	(2)関係演算子	78
	(3)論理演算子	78
	(4)関数	80
	(5)文字列の演算	80
第10節	スクリーン・エディター	81
第11節	エラー・メッセージ	81

第2章	CBM BASICのコマンドとステートメント	82
2. 1	CLOSE	83
2. 2	CLR	84
2. 3	CMD	84
2. 4	CONT	84
2. 5	DATA	85
2. 6	DEF FN	86
2. 7	DIM	87
2. 8	END	87
2. 9	FOR~NEXT	88
2. 10	GET, GET #	90
2. 11	GOSUB~RETURN	91
2. 12	GOTO	92
2. 13	IF~THEN, IF~GOTO	92
2. 14	INPUT	94
2. 15	INPUT #	95
2. 16	LET	96
2. 17	LIST	96
2. 18	LOAD	97
2. 19	NEW	98
2. 20	ON~GOSUB, ON~GOTO	98
2. 21	OPEN	99
2. 22	POKE	100
2. 23	PRINT, PRINT #	100
2. 24	READ	102
2. 25	REM	104
2. 26	RESTORE	104
2. 27	RUN	104
2. 28	SAVE	105
2. 29	STOP	105
2. 30	SYS	106
2. 31	VERIFY	106
2. 32	WAIT	107

第3章	CBM BASICの関数	108
3. 1	ABS	110
3. 2	ASC	110
3. 3	ATN	110
3. 4	CHR\$	111
3. 5	COS	111
3. 6	EXP	111
3. 7	FRE	112
3. 8	INT	112
3. 9	LEFT\$	112
3. 10	LEN	113
3. 11	LOG	113
3. 12	MID\$	113
3. 13	PEEK	114
3. 14	POS	114
3. 15	RIGHT\$	114
3. 16	RND	115
3. 17	SGN	115
3. 18	SIN	116
3. 19	SPC	116
3. 20	SQR	116
3. 21	STATUS	117
3. 22	STR\$	118
3. 23	TAB	118
3. 24	TAN	119
3. 25	TIME	119
3. 26	TIME\$	120
3. 27	USR	120
3. 28	VAL	121
第4章	CBM BASICのプログラム例	122
1.	はじめまして	123
2.	VIC SQUIGGLE (スキィグル)	124
3.	レコーディングその1	126
4.	レコーディングその2	128

5. VICオルガン	129
6. 壁こわしゲーム	131
7. ひらがな	132
8. 文字パターン作成プログラム	133
付録	135
A: CBM BASICのコード表	136
B: CBM BASICの省略形	137
C: 他のBASICからCBM BASICへの変換	138
D: エラーメッセージ一覧表	139
E: 誘導関数	147
F: カラーコントロール	148
G: サウンド	153
H: ハイ・レゾリューション	155

第3部 コンポーネント・データカタログ編163

1. MPS6500マイクロプロセッサ	165
2. MPS6560ビデオ・インターフェイス・チップ (VIC)	181
3. MPS6522汎用インターフェイス・アダプタ	199
4. MPS2364スタティックROM	231

第1部



ハードウェア編

第1章 コンピューターのしくみ

コンピューターを初めて使用される方々のために、そのしくみを簡単に説明しておきます。

コンピューターは、記憶装置と入・出力装置から構成されています。VIC-1001の記憶装置を構成している主要部品は次の通りです。

CPU (CENTRAL PROCESSING UNIT) またはMPU (MICROPROCESSOR UNIT) : MPS6502A×1

ROM (READ ONLY MEMORY) : MPS2364×2 および2332×1

RAM (RANDOM ACCESS MEMORY) : MPS2114×10

VIA (VERSATILE INTERFACE ADAPTOR) : MPS6522×2

VIC (VIDEO INTERFACE CHIP) : MPS6560×1

CPU (またはMPU) は、人間にたとえれば脳に相当する部分であり、上記の部品のコントロールをおこなっています。

ROMは、一定の動作をするためだけの部品です。BASICは、ROMに書き込まれており、たとえば、RUNと入力された時には、RAMの中にあるプログラムを実行する指示をCPUに与えます。電源を切っても、内容は消えません。

RAMは、ユーザー・プログラムおよびデータが格納される部分です。電源を切れば、内容は消えてしまいます。

VIAは、入・出力装置の制御をする部品です。

VICは、その名の示す通り、ビデオ・インターフェイス・チップであり、CRTへの表示をコントロールしています。

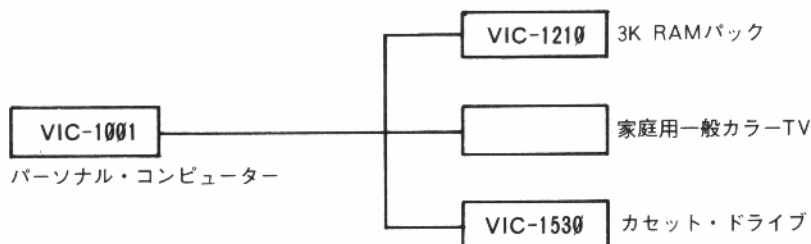
● これらを中心とする記憶装置全体を人間の脳とすれば、目・耳に相当する部分を入・出力装置と呼び、VIC-1001本体に組み込まれているものに、キーボードがあります。キーボードは、入力のみのために使用されます。VIC-1001本体の外部につけられるの入・出力装置としてはCRTディスプレイ（家庭用一般カラーTVまたは専用カラー・モニター）、カセット・ドライブ、プリンター、ジョイスティック、ライトペン等があります。

第2章 VIC-1001パーソナル・コンピューター

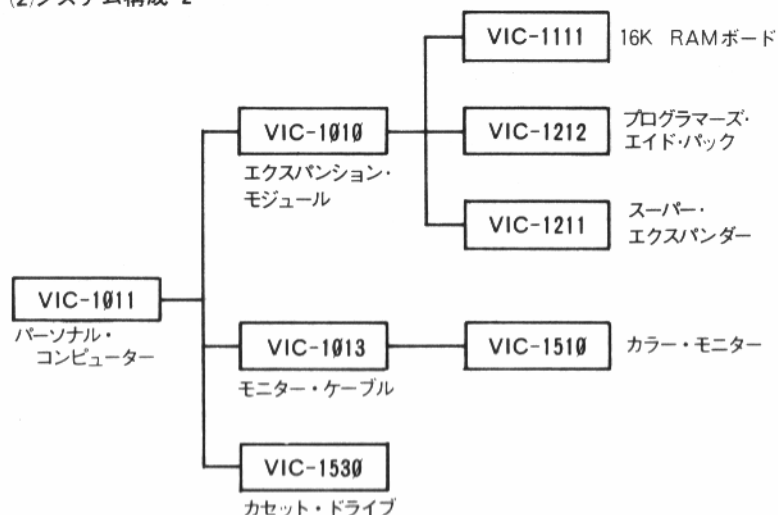
第1節 VIC-1000シリーズ・システム構成

VIC-1000シリーズ・パーソナル・コンピューター・システムは、その用途に応じて、拡張可能です。以下では、その組み合わせ例を図で説明します。

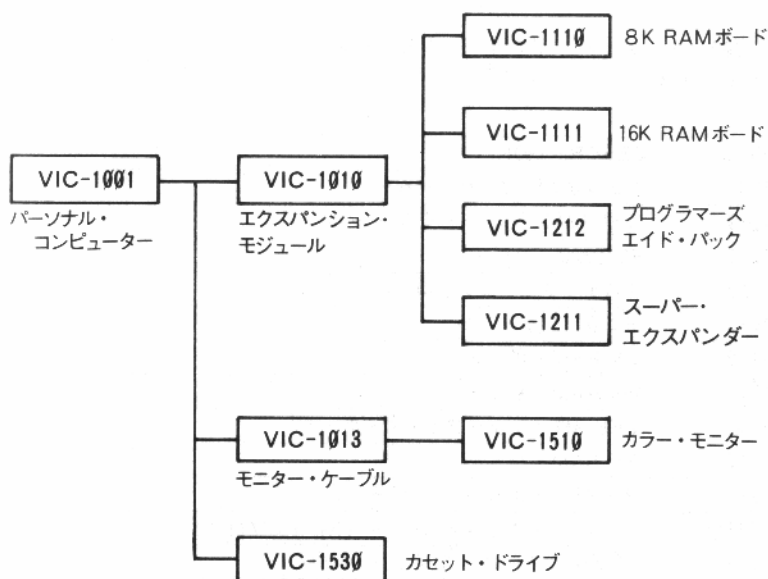
(1)システム構成-1



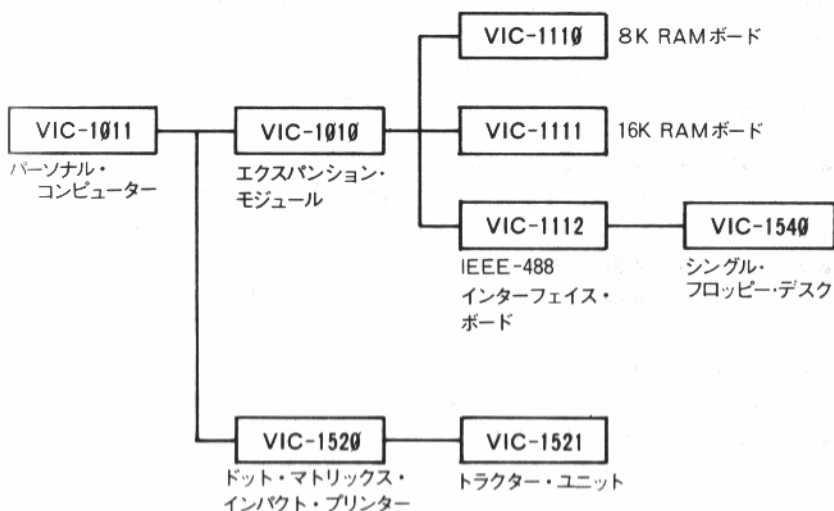
(2)システム構成-2



(3)システム構成-3



(4)システム構成-4



第2節 VIC-1000シリーズ・ファミリー

VIC-1000シリーズ・パーソナル・コンピューター・システムには、以下のファミリーがあります。

MODEL NO.	DESCRIPTION
VIC-1001	パーソナル・コンピューター
VIC-1010	エクспанション・モジュール
VIC-1011	RS-232C・アダプター・ボード
VIC-1012	マルチプル・コントロール・ボード
VIC-1013	モニター・ケーブル
VIC-1110	8K RAM・ボード
VIC-1111	16K RAM・ボード
VIC-1112	IEEE-488インターフェイス・ボード
VIC-1210	3K RAM・バック
VIC-1211	スーパー・エクパンダー
VIC-1211M	スーパー・エクパンダー (3K RAM付き)
VIC-1213	プログラマーズ・エイド・バック
VIC-1310	ライト・ペン
VIC-1311	ジョイ・スティック
VIC-1312	パドル
VIC-1510	カラー・モニター
VIC-1520	ドット・マトリックス・インパクト・プリンター
VIC-1521	トラクター・ユニット
VIC-1530	カセット・ドライブ
VIC-1540	シングル・フロッピー・ディスク

●VIC-1010：エクспанション・モジュール

エッジカード・コネクター4本実装（2本追加可能）。VIC-1001の接続ケーブル・電源・簡易キャビネットつき。VIC-1001のエクспанション・バスを利用。

●VIC-1011：RS-232C・アダプター・ボード

RS-232Cを持つ各種機器と、VIC-1001を接続。VIC-1001のユーザーポートに接続。

●VIC-1012：マルチプル・コントロール・ボード

VIC-1001のコントロール・ポートと接続。ジョイ・スティック4、接続可能。

●VIC-1013：モニター・ケーブル

VIC-1001のカラービデオ・インターフェイスと専用カラーモニター接続用。家庭

用TV使用の場合は、RFモデレーターについているケーブルを使用。

●VIC-1110：8K RAMボード/VIC-1111：16K RAMボード

エクспанション・モジュールのコネクターに接続。

●VIC-1112：IEEE-488 インターフェイス・ボード

IEEE-488を持つ各機器とVIC-1001の接続用。エクспанション・モジュールのコネクターに接続。

●VIC-1210：3K RAMパック

VIC-1001のエクспанション・バス、または、エクспанション・モジュールのコネクターに接続。

●VIC-1211：スーパー・エクспанション

176ドット×160ドットの高解像グラフィックおよび音楽演奏を可能にする拡張パック。ファンクション・キーには、使用頻度の高い12のコマンドをアサイン（ユーザーが変更することも可能）。VIC-1211Mは、3K RAM付き。VIC-1001のエクспанション・バスまたはVIC-1010エクспанション・モジュールに接続。

●VIC-1212：プログラマーズ・エイド・パック

BASICのプログラミング、プログラムのデバックを大幅に効率化する“TOOL”KIT”のコモドル版。ファンクション・キーには、使用頻度の高い12のコマンドをアサイン（ユーザーが変更することも可能）。VIC-1001のエクспанション・バスまたはVIC-1010エクспанション・モジュールに接続。

●VIC-1510：専用カラーモニター

コンポジット・ビデオ・インプット、スピーカー内蔵。ボーダー8色、バックグラウンド16色、キャラクター8色。11インチ。モニターケーブルで、VIC-1001のカラービデオ・インターフェイスに接続。

●VIC-1520：ドット・マトリックス・インパクト・プリンター

80桁。印字スピード150CPS。グラフィック・キャラクター印字も可能。VIC-1001のユーザーポートに付属ケーブルで、ダイレクトに接続。

●VIC-1521：トラクター・ユニット

ドット・マトリックス・インパクト・プリンター用のトラクター・フィールド・メカニズム。

●VIC-1530：カセット・ドライブ

記憶容量 約160Kバイト（C-30 使用時）。VIC-1001のカセット・インターフェイスに接続。

●VIC-1540：シングル・フロッピー・デスク

記憶容量 170Kバイト。VIC-1112 IEEE-488インターフェイス・ボードに接続。

●VIC-1310：ライトペン/VIC-1311：ジョイ・スティック/VIC-1312パドル

VIC-1001のコントロールポート、またはマルチプル・コントロール・ボードに接続。

第3章 VIC-1001の使用法

VIC-1001の梱包を解いたら、まず下記の製品が入っていることを確認してください。

VIC-1001パーソナル・コンピューター

電源トランス

家庭用テレビ・アダプター

ユーザーマニュアル

保証書

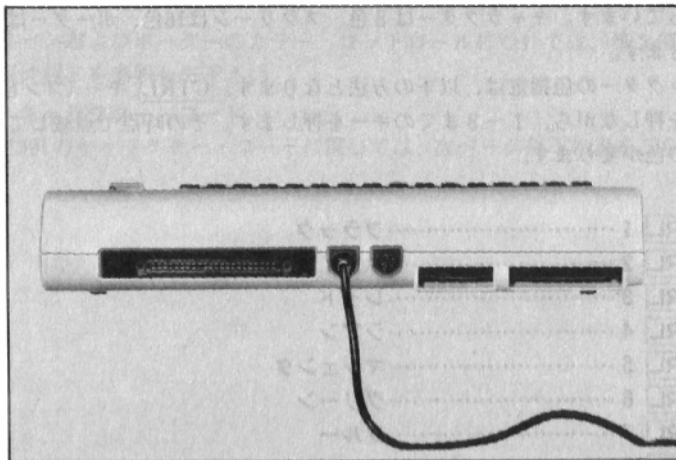
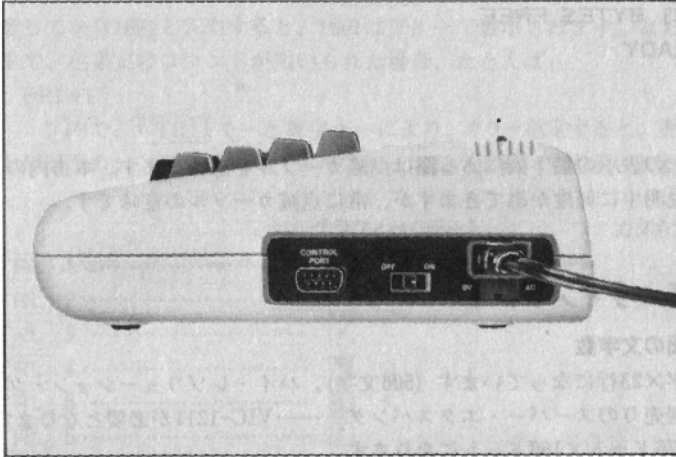
一点でも不足している場合、お買い上げになりました弊社販売店までご連絡下さい。なお、梱包材料は、修理etcの理由により必要になる場合がありますので、保存しておくことをおすすめします。

第1節 電源投入の前に

まず、電源トランスのACプラグをコンセントに差し込み、それからジャックをVIC-1001の右側面にあるAC9Vという表示のあるソケットに差し込んで下さい。そのさい、電源スイッチがOFF状態になっていることを確認して下さい。

次に、CRTディスプレイをVIC-1001に接続して下さい。家庭用TVをご使用のさいは、同梱されていますアダプターを家庭用TVのVHFアンテナ端子に接続後、一方のジャックをカラー・ビデオ・インターフェイスに接続して下さい。一方、専用カラー・モニター (VIC-1510) を御使用の場合は、別売りのモニター・ケーブル (VIC-1013) を用いて接続して下さい。このさい、TVアダプターは必要ありません。

注意 ●カセット・ドライブ (VIC-1530) その他の周辺機器、各種オプションとVIC-1001本体との接続方法については、その周辺機器、オプションのマニュアルをお読み下さい。



第2節 電源の投入

まずTVの電源を投入してから、VIC-1001の電源をONにして下さい。電源スイッチは、右側面に位置しています。この時、ディスプレイの画面が次の表示をすれば、VIC-1001は正常ということになります。

```
***** CBM BASIC V2 *****
3583 BYTES FREE
READY
```



注意：上の表示の最下列にある■は点滅カーソルを意味します。本書内の表示の説明中に何度か出てきますが、常に点滅カーソルの意味です。

第3節 ディスプレイ

1. 画面の文字数

22文字×23行になっています(506文字)。ハイ・レゾリューション・グラフィック(別売りのスーパー・エクспанダー——VIC-1211が必要となります)の場合は、176ドット×160ドットになります。

2. カラー

電源投入時、キャラクターはブルー、スクリーンはホワイト、ボーダーはシアンになっています。キャラクターは8色、スクリーンは16色、ボーダーは8色を指定できます。

キャラクターの色指定は、以下の方法となります。**CTRL** キー(コントロールキー)を押しながら、1～8までのキーを押します。その時点で点滅しているカーソルの色が変わります。

CTRL 1	ブラック
CTRL 2	ホワイト
CTRL 3	レッド
CTRL 4	シアン
CTRL 5	マゼンタ
CTRL 6	グリーン
CTRL 7	ブルー
CTRL 8	イエロー

したがって、COMMODORE VIC-1001の、COMMODOREをレッドで、VIC-1001をシアンで、そして1001をブルーでスクリーン上に表示する場合、その操作の仕方は、**CTRL** キーを押しながら3を押し、いったん離してから、COMMODOREと入力すると、これら9つのキャラクターはレッドで表示されます。次に**CTRL** キーを押しながら4を押し、いったん離してから、VIC-と入力すると、シアンでVIC-と表示されます。最後に**CTRL** キーを押しながら7を押し、いったん離してから1001と入力すると、1001はブルーで表示されます。なお、プログラム中で、色指定のコマンドが用いられた場合、たとえば

10 PRINT " " "

の" " 内で、**CTRL** キーと数字キーにより、カラー指定すると、押された数字により以下のシンボルがリバース・フィールド（反転）で表示されます。

	グラフィックモード	カタカナモード
CTRL 1		
CTRL 2		
CTRL 3		
CTRL 4		
CTRL 5		
CTRL 6		
CTRL 7		
CTRL 8		

スクリーンおよびボーダーのカラー・コントロールについては、第2部ソフトウェア編付録Fを参照して下さい。

3-4・キャラクター・コード

VIC-1001のキャラクター・コードに関しては、次ページ以下の表をごらん下さい。

キャラクター・コード表

ASCII(アスキー)		スクリーン コード	キャラクター		ASCII(アスキー)		スクリーン コード	キャラクター	
10進	16進		グラフィック モード	カタカナ モード	10進	16進		グラフィック モード	カタカナ モード
0	0		NULL	NULL	38	26	38	&	
1	1				39	27	39	'	
2	2				40	28	40	(
3	3				41	29	41)	
4	4				42	2A	42	*	
5	5				43	2B	43	+	
6	6				44	2C	44	,	
7	7				45	2D	45	-	
8	8				46	2E	46	.	
9	9				47	2F	47	/	
10	A		LF	LF	48	30	48	0	
11	B				49	31	49	1	
12	C				50	32	50	2	
13	D		CR	CR	51	33	51	3	
14	E				52	34	52	4	
15	F				53	35	53	5	
16	10				54	36	54	6	
17	11		カーソル下		55	37	55	7	
18	12		反 転		56	38	56	8	
19	13		HOME	HOME	57	39	57	9	
20	14		DEL	DEL	58	3A	58	:	
21	15				59	3B	59	:	
22	16				60	3C	60	<	
23	17				61	3D	61	=	
24	18				62	3E	62	>	
25	19				63	3F	63	?	
26	1A				64	40	0	@	
27	1B				65	41	1	A	
28	1C		レッド		66	42	2	B	
29	10		カーソル右		67	43	3	C	
30	1E		グリーン		68	44	4	D	
31	1F		ブルー		69	45	5	E	
32	20	32	SPACE	SPACE	70	46	6	F	
33	21	33	!		71	47	7	G	
34	22	34	''		72	48	8	H	
35	23	35	#		73	49	9	I	
36	24	36	\$		74	4A	10	J	
37	25	37	%		75	4B	11	K	

ASCII (アスキー)		スクリーン コード	キャラクター		ASCII (アスキー)		スクリーン コード	キャラクター	
10進	16進		グラフィック モード	カタカナ モード	10進	16進		グラフィック モード	カタカナ モード
76	4C	12	L		116	74	84		ヤ
77	4D	13	M		117	75	85		ユ
78	4E	14	N		118	76	86		ヨ
79	4F	15	O		119	77	87		ラ
80	50	16	P		120	78	88		リ
81	51	17	Q		121	79	89		ル
82	52	18	R		122	7A	90		レ
83	53	19	S		123	7B	91		リ
84	54	20	T		124	7C	92		ワ
85	55	21	U		125	7D	93		リ
86	56	22	V		126	7E	94	π	π
87	57	23	W		127	7F	95		ヲ
88	58	24	X		128	80			
89	59	25	Y		129	81			
90	5A	26	Z		130	82			
91	5B	27	[131	83			
92	5C	28	¥		132	84	127		ソ
93	5D	29)		133	85		f 1	
94	5E	30	↑		134	86		f 3	
95	5F	31	←		135	87		f 5	
96	60	64			136	88		f 7	
97	61	65		チ	137	89		f 2	
98	62	66		ツ	138	8A		f 4	
99	63	67		テ	139	8B		f 6	
100	64	68		ト	140	8C		f 8	
101	65	69		ナ	141	8D		シフトリターン	
102	66	70		ニ	142	8E			
103	67	71		ヌ	143	8F			
104	68	72		ネ	144	90		ブラック	
105	69	73		ノ	145	91		カーソル上	
106	6A	74		ハ	146	92		反転オフ	
107	6B	75		ヒ	147	93		CLR	CLR
108	6C	76		フ	148	94		INST	INST
109	6D	77		ヘ	149	95			
110	6E	78		ホ	150	96			
111	6F	79		マ	151	97			
112	70	80		ミ	152	98			
113	71	81		ム	153	99			
114	72	82		メ	154	9A			
115	73	83		モ	155	9B			

ASCII (アスキー)		スクリーン コード	キャラクター		ASCII (アスキー)		スクリーン コード	キャラクター	
10進	16進		グラフィック モード	カタカナ モード	10進	16進		グラフィック モード	カタカナ モード
156	9C		マジエンタ		196	C4	68		ト
157	9D		カーソル左		197	C5	69		ナ
158	9E		イエロー		198	C6	70		ニ
159	9F		シアン		199	C7	71		ヌ
160	A0		SPACE	SPACE	200	C8	72		ネ
161	A1	111		○	201	C9	73		ノ
162	A2	98		イ	202	CA	74		ハ
163	A3	99		ウ	203	CB	75		ヒ
164	A4	100		エ	204	CC	76		フ
165	A5	101		オ	205	CD	77		ヘ
166	A6	95		ヲ	206	CE	78		ホ
167	A7	103		キ	207	CF	79		マ
168	A8	104		ク	208	D0	80		ミ
169	A9	105		ケ	209	D1	81		ム
170	AA	106		◇	210	D2	82		メ
171	AB	107		ト	211	D3	83		モ
172	AC	108		ス	212	D4	84		ヤ
173	AD	109			213	D5	85		ユ
174	AE	110			214	D6	86		ヨ
175	AF	111		○	215	D7	87		ラ
176	B0	64			216	D8	88		リ
177	B1	97		ア	217	D9	89		ル
178	B2	114			218	DA	90		レ
179	B3	115			219	DB	120		ロ
180	B4	116		年	220	DC	92		ワ
181	B5	117		月	221	DD	121		ン
182	B6	102		カ	222	DE	106		◇
183	B7	103		キ	223	DF	95		ヲ
184	B8	104		ク	224	E0	94	π	π
185	B9	105		ケ	225	E1	91		
186	BA	122		コ	226	E2	93		
187	BB	123		サ	227	E3	99		ウ
188	BC	124		シ	228	E4	100		エ
189	BD	108		ス	229	E5	101		オ
190	BE	126		セ	230	E6	102		カ
191	BF	127		ソ	231	E7	103		キ
192	C0	119		タ	232	E8	104		ク
193	C1	65		チ	233	E9	105		ケ
194	C2	66		ツ	234	EA	106		◇
195	C3	67		テ	235	EB	107		

ASCII (アスキー)		スクリーン モード	キャラクター		ASCII (アスキー)		スクリーン モード	キャラクター	
10進	16進		グラフィック モード	カタカナ モード	10進	16進		グラフィック モード	カタカナ モード
236	EC	108		ス					
237	ED	109							
238	EE	110							
239	EF	111		。					
240	F0	112							
241	F1	113							
242	F2	114							
243	F3	115							
244	F4	116		年					
245	F5	117		月					
246	F6	118		日					
247	F7	119		夕					
248	F8	120		ロ					
249	F9	121		ン					
250	FA	122		コ					
251	FB	123		サ					
252	FC	124		シ					
253	FD	125							
254	FE	126		セ					
255	FF	94		π					

第4節：キーボード

1. キーボード配列

VIC-1001には66種類のキーがあり、そのキー配置は次ページの写真の通りです。(JIS準拠)。

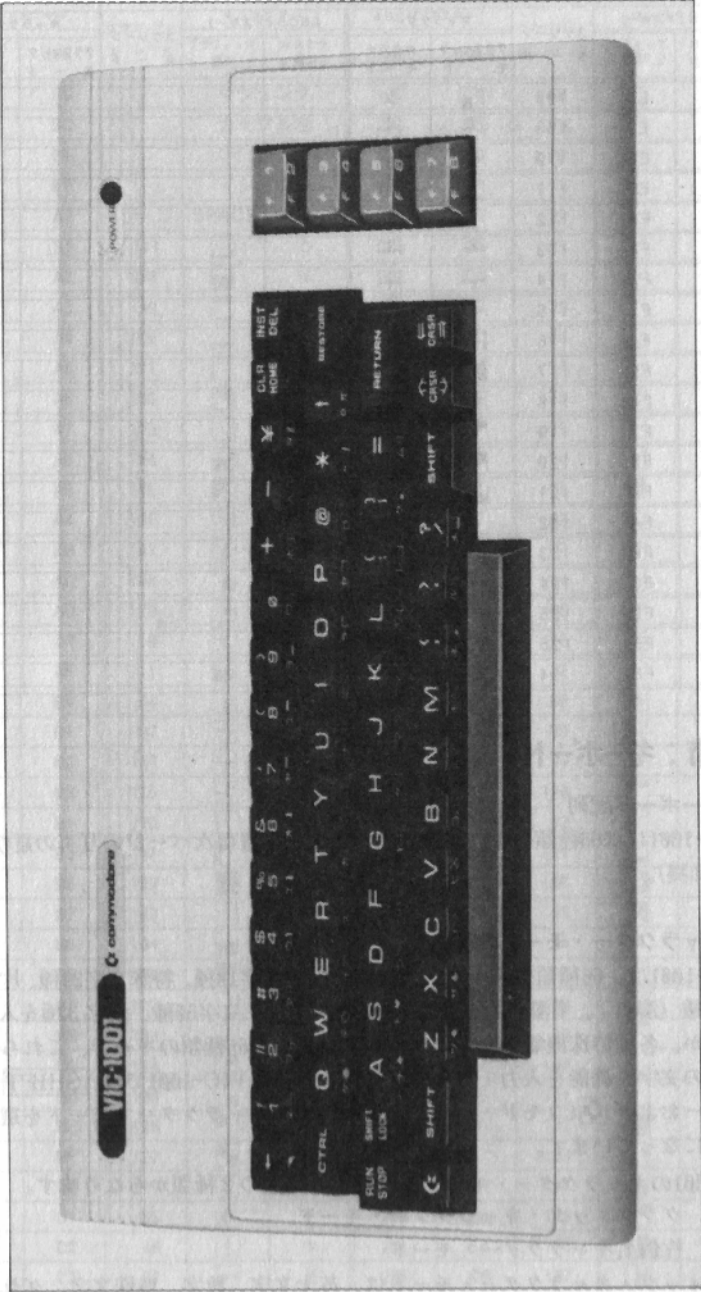
2. キャラクター・モードの選択

VIC-1001は、66種類のキーで英大文字26種、数字10種、特殊文字29種、片仮名文字48種(濁点°、半濁点〃を含む)、グラフィック文字50種、漢字3種を入力できるほか、各種特殊機能をもつようになっています。66種類のキーで、これらのたくさんの文字、機能を入力できるようにするため、VIC-1001では、**[SHIFT]**(シフト)キーおよび**[C]**(コモドル)キーを用いて、キャラクター・モードを選択するようになっています。

VIC-1001のキャラクター・モードは、基本的に次の2種類からなります。

- a. グラフィック・キャラクター・モード
- b. 片仮名キャラクター・モード

グラフィック・キャラクター・モードは、英大文字、数字、特殊文字、グラフィ



ック文字からなり、片仮名キャラクター・モードは、英大文字、片仮名文字、漢字および特殊文字、グラフィック文字の一部から成っています。

(1) グラフィック・キャラクター・モード

電源投入時には、キャラクター・モードはグラフィック・モードに設定され、**[SHIFT LOCK]** キー=シフト・ロック・キーが押されていないければ、何もしないで、それぞれのキーを押すと、ディスプレイには図1に相当するキャラクターが表示されます。電源をいったん切って、再度投入し、好きなようにタイプしてみてください。

次に、**[SHIFT]** キーを押しながら、それぞれのキーを押してみてください。ディスプレイには図2に相当するキャラクターが表示されます。

[SHIFT] キーの代りに、**[C]** キーを押しながら、それぞれのキーを押すと、ディスプレイには図3に相当するキャラクターが表示されます。

(2) 片仮名キャラクター・モード

片仮名キャラクター・モードに移るには、まず、**[SHIFT]** キーと**[C]** キーを同時に押します。(1)でタイプしたグラフィック文字が片仮名に変わるのがわかるでしょう。

片仮名モードはさらに、片仮名モードⅠと片仮名モードⅡに分かれます。片仮名モードⅠでは、図1および図4に相当するキャラクターが表示され、片仮名モードⅡでは、図5および図6に相当するキャラクターが表示されます。

電源投入後、グラフィック・モードから片仮名モードへ移ったさいには、片仮名モードⅠに設定されています。**[SHIFT]** キーを押さないで、それぞれのキーを押すと、図1のキャラクターが表示されます。**[SHIFT]** キーを押しながら各キーを押すと、図4のキャラクターが表示されます(これで最上列の特殊文字をタイプするのに用います)。

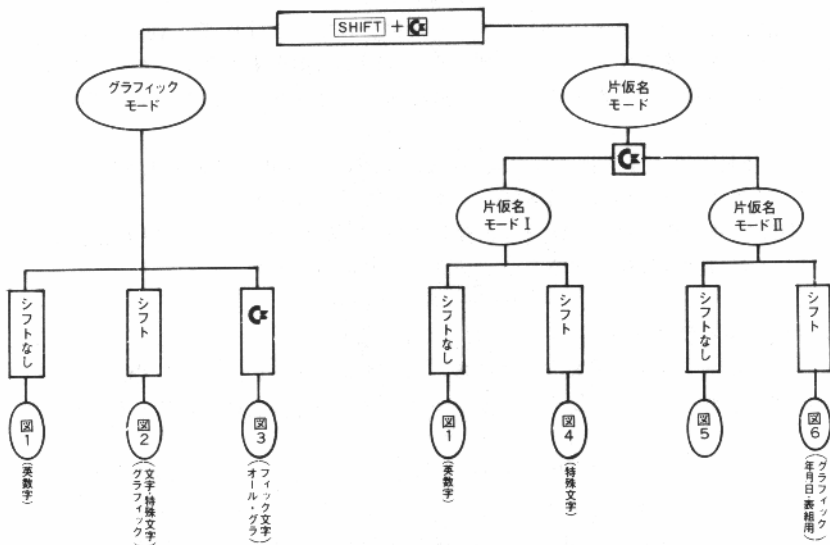
次に、**[C]** キーを押して離すと、片仮名モードⅠから片仮名モードⅡに移ります(**[C]** キーは、一度押すと、その状態でソフト的にロックがかかります)。片仮名モードⅡで、**[SHIFT]** キーを押さないで各キーを押すと、図5のキャラクター(フル片仮名)が表示されます。**[SHIFT]** キーを押しながら、各キーを押すと、図6に示すキャラクター(年月日および表組用グラフィック文字)が表示されます。

ここで再び**[C]** キーを押すと、片仮名モードⅡから片仮名モードⅠに移ります。つまり、**[C]** キーが片仮名モードⅠとモードⅡの切換えスイッチになっています。

片仮名モードからグラフィック・モードに移るには、**[SHIFT]** キーと**[C]** キーを同

時に押します。

グラフィック・モードと片仮名モード（ⅠとⅡ）との関係は、次の図のようになります。

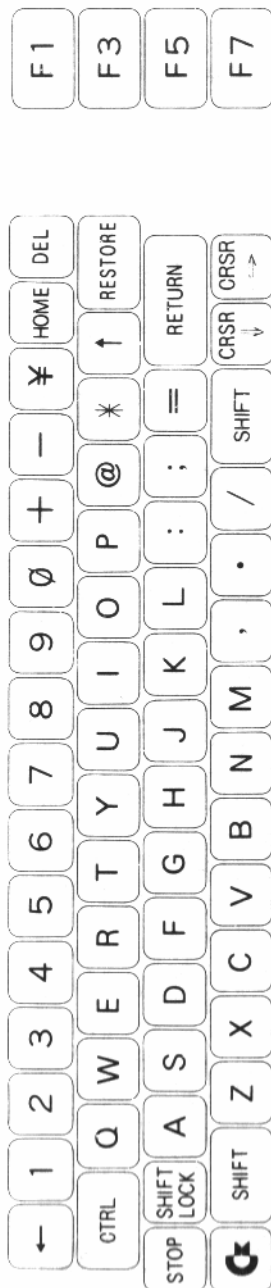


片仮名の打ち方 プログラム中、片仮名を使うことができるのは、“ ”で囲んだ中だけです。

```
例) PRINT "カタカナ"  
INPUT "キカン"; K  
REM "オセロ"  
A$="コモドール"
```

したがって、一般に、プログラム中で片仮名を使いたいときは、次のようにします。まず、プログラム中で片仮名を使うかどうかを決め、グラフィック・モードから片仮名モードに変換します。そして、たとえば `PRINT "カタカナ"` と打とうとして、`[1]` を押して、`カ` が表示されたら、`[C]` キーを押します。そしてそのまま、シフトなしでタイプすれば `10 PRINT` とタイプできます。それから、`[SHIFT]` キーを押しながら `[2]` を押します。そのあと、`[C]` キーを押して離してから、`カタカナ` とタイプします。片仮名が打ち終わったら、`[C]` キーを押して離し、`[SHIFT]` キーを押しながら、`[2]` を押します。これで、`10 PRINT "カタカナ"` とタイプできたことになります。

☒ 1



☒ 2

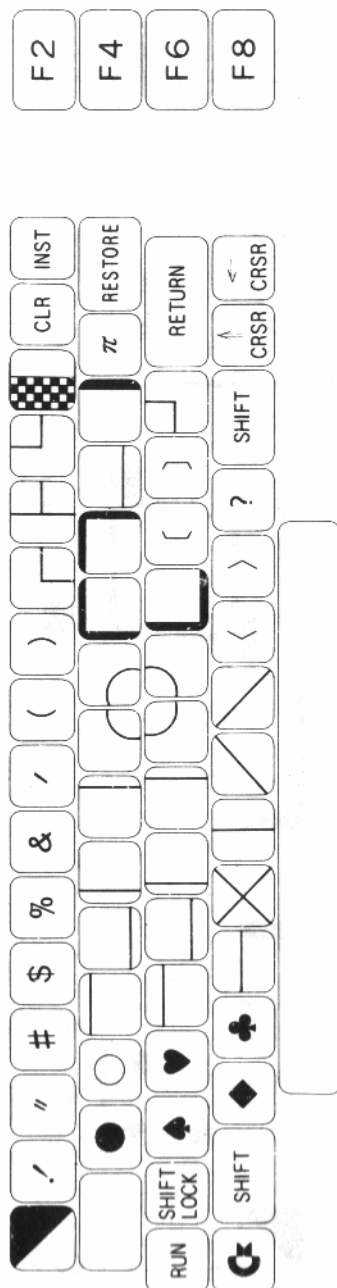


図 3

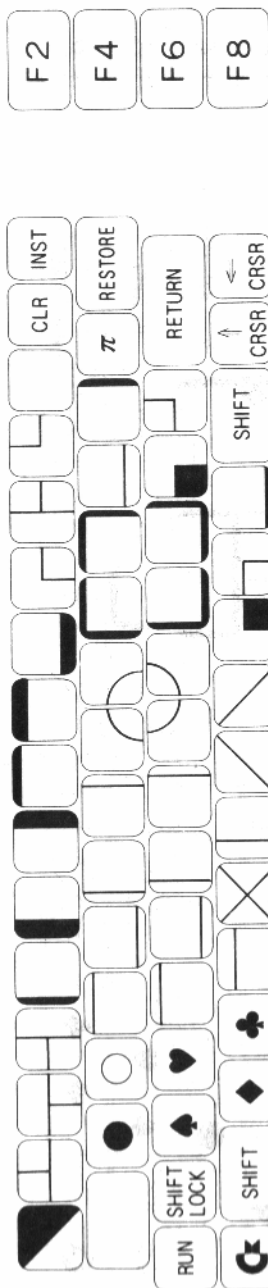


図 4



図5



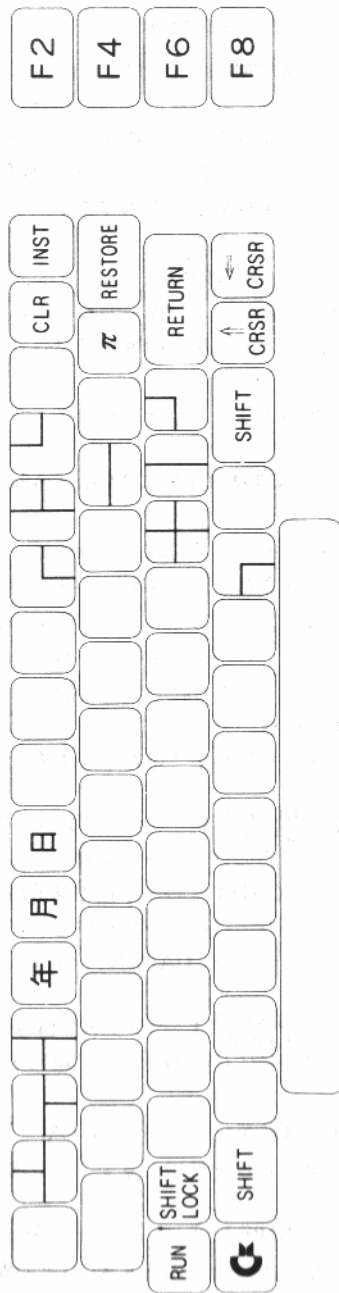
F1

F3

F5

F7

図6



F2

F4

F6

F8

3・特殊キーの働き

(1) **CTRL** キー (コントロール・キー)

このキーは、次の3つの用途に用います。

- a. キャラクターのカラー・セット
- b. リバース・フィールド(反転文字)のセットおよび解除
- c. 実行スピードのダウン

キャラクターのカラー・セットは、第4章第3節3のカラーの項を参照して下さい。
リバース・フィールドのセットは、**CTRL** キーを押しながらRを押します。リバース・フィールドの解除は**CTRL** キーを押しながら \emptyset を押します。

画面上にABCをリバース・フィールドで表示する場合

? "**CTRL** ↓ R **CTRL** ↑ ABC **CTRL** ↓ \emptyset **CTRL** ↑ "

と押します。この時点で画面は、? "**R** ABC \square "と表示され、**RETURN** キーが押されると、ABCがリバース・フィールドで表示されます。

なお、カタカナ・モードの場合は、表示は、? "**R** A B C \square "となります。

また、**CTRL** キーの3番目の使い方は以下の通りです。プロタラム実行中、このキーを押し続けている間、プログラムの実行スピードをおとすことができます。同様に、プログラムをLIST中にこのキーを押すことにより、スクロール・スピードをおとすことができます。

(2) **RUN STOP** キー (ラン/ストップ・キー)

実行中のプログラムを停止する時、そして、プログラムをカセット・ドライブからLOADするのを中止させる時に用います。**SHIFT** (または **C**) キーを押しながら、このキーを押すことにより、カセット・ドライブから最初にみつかったプログラムをLOAD開始します。

(3) **SHIFT LOCK** キー (シフト・ロック・キー)

SHIFT 状態のモードに保ちます。

(4) **C** キー (コモドール・キー)

他のキーと併用します。(その働きについては、前項を参照して下さい)

(5) **SHIFT** キー (シフト・キー)

他のキーと併用します。(その働きについては、前項を参照して下さい)

(6) **CLR HOME** キー (クリア/ホーム・キー)

このキーを押すと、カーソルがホーム・ポジション (左上端) に戻ります。

SHIFT キーと同時に押すと、画面をクリアし、そしてカーソルをホーム・ポジションに戻します。

(7) **INST DEL** キー (インサート / デリート・キー)

このキーを押すと、カーソルの左側にあるキャラクターを1字削除し、それより右の文字が左へ詰められます。リピート機能がついています。**SHIFT** キーと同時に押すと、カーソルの位置とその左側のキャラクターの間にスペースを入れ、キャラクターの挿入を可能にします。

(8) **RESTORE** キー (レストア・キー)

プログラム実行中、何らかの理由によりプログラムが暴走した場合、**RUN STOP** キーとともにこのキーを押すことにより、メモリー中のデータを失うことなく実行を停止させることができます。

(9) **RETURN** キー (リターン・キー)

カーソルを次の左端に移します。カーソルが最下段にある場合は、画面をスクロールします。

(10) **CRSR** キー (カーソル・アップ / ダウン・キー)

このキーを押すと、カーソルを1列下に下げます。**SHIFT** キーと併用した場合は、1列上に上げます。リピート機能がついています。

(11) **CRSR** キー (カーソル・レフト / ライト・キー)

このキーを押すと、カーソルを1キャラクター分右へ移動させ、**SHIFT** キーと併用した場合は、左へ移動させます。

(12) f1~f8キー (ファンクション・キー)

スーパー・エキスパンダー (VIC-1211) またはプログラマース・エイド・パック (VIC-1212) を使用することにより、12種類の機能を持たせることが可能になります。VIC-1211または1212をご使用にならない場合でも、プログラムにより、一定のファンクションを持たせることが可能です。なお、VIC-1211、1212をご使用の場合はそれぞれのマニュアルをご参照下さい。

第4章 BASICによるプログラミング

この章は、VIC-1001の操作方法に習熟していただくために、特に初心者の方を対象に、BASIC（ベーシック）言語でプログラムをどのように書き、編集するか、またそれをどのように実行させるかなど、基本的な事項の説明をおこないます。VIC-1001のBASIC言語であるとCBM BASIC V2（バージョン2）の詳しい説明については、第2部ソフトウェア編を参照して下さい。

第1節 BASICとは

BASIC (Beginner's All-purpose Symbolic Instruction Gode) は、その名の通り、初心者向けコンピューター言語として米国で開発され、今ではほとんどのパーソナル・コンピューターに、BASIC言語がとりいれられるようになりました。対話形式を基本としたその使いやすさが受け入れられており、CBM BASIC V2は、使われる方々の立場に立って特に開発されたものです。

BASIC言語は、パーソナル・コンピューター・メーカーや、その機種によって命令の書き方や機能が少しずつ異なっていますので、必ず第2部のソフトウェア編も読んで下さい。

第2節 プログラムの書き方

VIC-1001に電源を入れると、

```
***** CBM BASIC V2 *****  
3583 BYTES FREE  
READY.  
■
```

と表示されます(■は、点滅カーソルを意味します)。ディスプレイ上に点滅カーソルが出ている場合、コンピューターはREADY状態にあり、入力待ちを意味します。たとえば、キーボードからでたらめな文字(数字はこのさい含めないで下さい)を入力し、**RETURN** キーを押して下さい。

```
? SYNTAX  
ERROR  
READY.  
■
```

とディスプレイに表示されるはずですが、SYNTAX ERRORとは、コンピューターの理解できない文章が入力された場合に示されるメッセージです。この他にもエラー・メッセージは何種類もあります。どのような場合にどのようなエラー・メッセージが表示されるかは、第2部付録Dを参照して下さい。

VIC-1001に何らかの動作をさせるには、(1)ダイレクト・モード(2)プログラム・モードの2つの方法があります。ここでは、この2つのモードをBASICの面から説明します。

(1)ダイレクト・モード

ライン・ナンバー(行番号——プログラム・モードの項を参照して下さい)をつけずに、BASICコマンドを直接入力することにより、コンピューターに命令を与えることをダイレクト・モードと呼びます。たとえば、

```
PRINT "GOOD MORNING" RETURN
```

と入力して下さい。その時、ディスプレイは次のように表示します。

```
PRINT "GOOD MORNING"  
GOOD MORNING
```

```
READY.  
■
```

●注意● 入力の終わりをコンピューターに知らせるために、入力の最後には必ずRETURN キーを押します。RETURN キーが押されないと、コンピューターはいつまでも入力があるものと考え、実行に移りません。

ダイレクト・モードは、コンピューターに命令を記憶させずに、直接結果を得ようとする場合に用います。たとえば、

```
PRINT 456 * 346 / 3  
52592
```

```
READY.  
■
```

```
PRINT SIN (3 * π / 360 * 60)
```

```
1
```

READY.



PRINT TIS\$

000152

READY.



VIC-1001ではBASICのいろいろなコマンドを省略形を用いて書くことができます。そのうち、最も頻繁に用いられるものに?があります。これは、PRINTの代用になります。省略形を用いて、上記の入力をする場合、以下ようになります(省略形については第2部ソフトウェア編付録Bを参照して下さい)。

? 456 * 346 / 3

52592

READY.

? SIN (3 * π / 360 * 60)

1

READY.

? TIS\$

000325

READY.



また、変数を1つ1つ定義し、最後に変数を用いた数式を書いて値を求めることもできます。たとえば、球の体積を求める場合、半径が10であるならば、

R=10

READY.

? 4/3 * π * R ↑ 3

4188.7902

READY.



また、直方体(20×30×40)の体積を求める場合、

A=20 : B=30 : C=40

READY.

? A*B*C

24000

READY.



ダイレクト・モードでは、上で説明したように、結果はすぐに得られますが、コンピューターの特長である「人間の命令した手順通り、複雑な計算を何回もおこなったり、大量のデータを自動的に処理する」ことはできません。

(2) プログラム・モード

ダイレクト・モードではライン・ナンバー（行番号）をつけませんでした。プログラム・モードではライン・ナンバーをつけることにより、コンピューターに実行の順番を指示します。CBM BASICでは、1ライン・ナンバーごとに88字（ライン・ナンバーおよびスペース、そして **RETURN** キーを含めて）までを入力することが可能です。コンピューターはRUNという入力があると、ライン・ナンバー順に実行開始します。BASICの命令を順序よく考えて、組み合わせ、処理の手順として記憶させていく作業をプログラミングと呼びます。ここでは、球の体積を求めるプログラムを作ってみましょう。半径をRとします。

```
10? "ハンケイ?" RETURN
20INPUT R RETURN
30? "タイセキ="; RETURN
40? 3/4* $\pi$ *R3 RETURN
```



と入力した後、LISTと入力して下さい(LISTの後にも **RETURN** キーを押すことを忘れないで下さい)。ディスプレイは、以下を示します。

```
10 PRINT "ハンケイ?"
20 INPUT R
30 PRINT "タイセキ=";
40 PRINT 3/4* $\pi$ *R3
```

お気付きのように、省略形を入力時に用いたにもかかわらず、LIST(入力したプログラムの内容を確認する場合に用いるコマンド) すると、BASICコマンドが、完全な形で表示されます。また ライン・ナンバーの後にスペースを入れなくても、

LIST時には、スペースが入り、読みやすくなります。

なお、プログラミングを開始する前には、メモリーの内容をクリアーするためにNEWというコマンドを用います。ただし、いったんNEWというコマンドを入れますと、画面上には残っていても、メモリー中からは除去されてしまいますので、気をつけて下さい。

第3節 プログラムの修正

まず、前記のプログラムが実際に、どのように動くかを確かめるために、RUN **RETURN** と入力して下さい。

```
10 PRINT "ハンケイ?"
20 INPUT R
30 PRINT "タイセキ=";
40 PRINT 3/4 * π * R ^ 3
PEADY.
RUN
ハンケイ?
? ■
```

となります。この時点で、半径の値を入れます。たとえば半径=15として、15 **RETURN** と入力すると、画面は

```
ハンケイ?
? 15
タイセキ=7952.1564
READY.
■
```

となり、体積が求められました。ところがこのプログラムの場合、次にたとえば半径が20の球の体積を求めようとすると、ふたたびRUN **RETURN** と入力する必要があります。この手間を省くためにはプログラムの修正をする必要があります。また、画面をいったんクリアしてから、実行に移るという修正も加えてみることにします。

まず、プログラムをLISTして下さい。そして、点滅しているカーソルの位置から以下を入力して下さい。

```
5 ? " SHIFT CLR HOME "
50 GOTPI0
```

そして再度LISTと入力して下さい。画面には、以下が表示されるはずで

```

5 PRINT "♥"
10 PRINT "ハンケイ?"
20 INPUT R
30 PRINT "タイセキ=";
40 PRINT 3/4 * π * R ↑ 3
50 GOTP 10
READY.

```

お気付きのように、後に加えた2行のラインが、ライン・ナンバー順に並び変えられました。このように、後に加えたプログラムも、ライン・ナンバーにより必要なところへ挿入が可能になります。

そこで、この修正されたプログラムをRUNし、15を入力して下さい。すると、画面がいったんクリアーされ、スクリーンの左端から、

```

ハンケイ?
? 15
タイセキ=7952.1564

```

```

? SYNTAX
ERROR IN 50
READY.

```

と表示されます。

まず、ライン・ナンバー5を加えたことにより、キーボードの項で説明しましたように、**SHIFT** + **CLR HOME** により、いったん画面がクリアーされました。

次に、"?SYNTAX ERROR IN50"というメッセージにより、ライン・ナンバー50に文法上の誤りがあったことがわかります。そこで、LIST50と入力して下さい。すると、画面は以下のようになります。





```





LIST 50

50 GOTP 10
READY.


```

ここで、BASICに登録されていないGOTPという言葉があることが判ります。し


たがって、カーソル・コントロール・キー (、、 および ) を用いて、修正をおこないます。

上の場合は、 キーと  を用いて、カーソルを50 GOTP 50の位置まで持って行き、その次に  キーで、カーソルをPを上に移し、その位置でOを打ち、 キーを押します。そして、カーソルをREADYの下の行まで持って行き、LISTと入力して下さい。ここで、ライン・ナンバー50が以下のように訂正されている事を確認して下さい。

50 GOTO 10

●注意1●訂正後、 キーを使用せずに、カーソル・コントロール・キーを用いて下の行に移った場合、その行は訂正されません。

●注意2●プログラムのライン・ナンバーを10とか100おきにするのは、後になって行を追加する場合がありますからです。ライン・ナンバーは正の整数である必要があります、最大63999までになっています。

つぎに、このプログラムをRUNして、今度は正確に動くことを確認して下さい。半径の値を変える事により、次々に体積を瞬時に計算し、ディスプレイに表示します(このプログラムの実行を停止させるには、 キーを押します)。

次に、簡単に、ライン・ナンバーごとに、どのような処理がなされるのか説明します。

まずライン・ナンバー5では、前に述べましたように、画面をクリアし、カーソルをホーム・ポジションに持っていくという処理がなされます(PRINT " "の間にカーソル・コントロール・キーを用いれば、好みの場所に文字を表示できます。ライン・ナンバー5を修正して、実際に研究してみてください。)ライン・ナンバー10で、ハンケイ?という文字列をディスプレイ上に表示させます。ライン・ナンバー20では、Rという変数に対して、数字が入力されるのを待ちます(数字以外が入力されると、コンピューターは?REDO FROM START...最初からやり直して下さい……という表示をします)。ライン・ナンバー30では、10と同様にタイセキ=という文字列をディスプレイ上に表示させます。その行の終りにある;は、次の表示を、タイセキ=の右どなりに表示せよという意味を持っています(例1参照)。;を取ると、例2のように表示されます。

ライン・ナンバー40では、入力されたRの値を実際に数式にあてはめ、その結果を、上記タイセキ=の右側に表示します。PRINTの後に、" "が無い事に注目して下さい。" "をつけた場合、数式そのままを表示します。

例 1 : ハンケイ ?
? 15
タイセキ=7952.1564
READY.
■

例 2 : ハンケイ ?
? 15
タイセキ=
7952.1564

ある行全体を削除する場合は、該当する番号を入力し、**RETURN** キーを押します。たとえば、前の例の中のライン・ナンバー 5 を削除する場合は、

5 **RETURN**

と入力します。

また、LISTコマンドは以下の使い方があります。

1. ある 1 行 (ライン・ナンバー 10 とします) をリストする場合

LIST10 **RETURN**

2. ライン・ナンバー 10 ~ 150 までを LIST する。

LIST10-150 **RETURN**

3. ライン・ナンバー 150 までを LIST する。

LIST-150 **RETURN**

4. ライン・ナンバー 150 以降を LIST する。

LIST150- **RETURN**

スクロール (画面のせり上がり) をストップさせるには、**RUN STOP** キーを用い、ゆっくりスクロールさせたい時は **CTRL** キーを用います。

あるライン・ナンバーに修正を加え、そのライン・ナンバーの文が23字より長くなる場合、自動的に行間隔があげられます。たとえば、

```
10 PRINT"1 2 3 4 5 6 7 8 9 0"  
20 PRINT"ABCDEF"
```

とあり、ライン・ナンバー10の1234567890の代りに123456789012345とする場合、カーソルを0の後の" の位置へ戻し、1 2 3と追加するわけですが、3を入力した時点で、ライン・ナンバー20は自動的にスクロール・ダウンします。

第4節 スクリーン・エディター

プログラムの編集に関するまとめおよび補足をします。

- 1) 1ライン・ナンバーを書き終ったあと、そして修正し終った後、必ず **RETURN** キーを押して下さい。たとえば10PRINT"ABC"と入力し、**RETURN** キーを押さずに、**CRSR** キーを用いて下段にカーソルを移動し、LISTしても、何もディスプレイされない事で、**RETURN** キーの必要性は、おわかりいただけると思います。一方、あるライン・ナンバーを書き終えていない場合は、カーソルが画面の右端にあったとしても、その次の行には自動的にカーソルは移動しますから、**RETURN** キーは押さないで下さい。**RETURN** キーが押されると、入力が終了したとみなされます。
- 2) あるライン・ナンバーを持つ行全体を消す場合は、そのライン・ナンバーを入力し **RETURN** キーを押します。
- 3) あるライン・ナンバーと同じ内容の行を違ったライン・ナンバーとともに用いたい場合、そのライン・ナンバーの上にカーソルを移動し、新しい番号を入力し、**RETURN** キーを押すことにより、新しい行を発生させることができます。たとえば

```
10 PRINT"HELLO";
```

と入力して下さい。そして、1の上にカーソルを戻し、20 **RETURN**、そして次にカーソルを2の上に戻し、30 **RETURN** と入力し、LISTしてみて下さい。

```
10 PRINT"HELLO";  
20 PRINT"HELLO";  
30 PRINT"HELLO";
```

となります。

第6章 カセット・ドライブの使い方

VIC-1001では、電源を切るとメモリー内のプログラムおよびデータは消えてしまいます（ROMに書き込まれているものは消えません）。したがって、そのプログラムやデータを再度用いる場合には、あらかじめそれらを外部記憶装置に記憶させておきます。VIC-1001の外部記憶装置には、カセット・テープおよびフロッピー・ディスクがあります。VIC-1000シリーズの初心者用システムおよび標準システムでは、カセット・テープを用いますので、ここではカセット・ドライブ（VIC-1530）の使い方について説明します。まず、カセット・ドライブのマニュアルに従って、カセット・ドライブをVIC-1001に接続し、所定の初期テストおよび録音・再生チェックをおこなったうえで、以下をおこなってください。

次の簡単なプログラムをキーボードから入力して下さい。

```
10 FOR I=1 TO 20  
20 ? "HELLOW VIC";  
30 NEXT I
```

このプログラムをカセット・テープに記憶させるには、SAVEというコマンドを用います。

第1節 プログラムのSAVE

まず、SAVEする前に、このプログラムに名前をつける必要があります。プログラム名は、最大84文字使用できますが、実際にテープに書き込まれるのは、16文字までです。したがって、上のプログラム名をABCDEFGHIJKLMN OPQRSTUVWXYZという26文字としてSAVEしたとしても、テープ上にはPまでしか書き込まれません。つまり、もし、2番目のプログラムが、ABCDEFGHIJKLMN OPZZZという名称でSAVEされた場合、後にコンピューターは、これら2つのプログラムとも、プログラム名は、ABCDEFGHIJKLMN OPであると判断します。

さて、実際にこのプログラムにTESTという名称をつけてSAVEすることになります。

SAVE "TEST" **RETURN** と入力して下さい。

画面に

PRESS RECORD & PLAY ON TAPE

と表示されます。画面の指示にしたがい、カセット・ドライブのRECORDボタンを押しながら、PLAYボタンを押すと、画面に、

OK

SAVING TEST

が表示され、SAVEが終了すると、画面にREADY.が表示され、カーソルが点滅します。

●注意● 実際に、SAVINGという表示がスクリーンに出てから約10秒間は、カセット・ドライブのモーターの回転が安定化を待つために録音は開始しません。約10秒経過後、ヘッダーと呼ばれるものを書き込みます。そして録音終了後、エンド・マークを書き込みます。録音開始まで約10秒間ありますので、問題はないと思いますが、クリアフィーダー（磁性体の塗ってない部分）が異常に長い場合、ヘッダーがカセット・テープに録音されず、後に再生できない場合があります。

第2節 プログラムのVERIFY

上記の原因の他にも、うまくSAVEできないことになる原因はあります。たとえば、テープの傷、RECORDボタンの押し忘れが典型的なケースです。これらを防止するために、SAVE終了後、VERIFYしておく習慣をつけて下さい。

前例のプログラム「TEST」をSAVE終了後、カーソルが点滅し始めた時にいったん巻き戻し、キーボードからVERIFY TESTと入力します。すると、画面には、

PRESS PLAY ON TAPE

と表示されますので、表示にしたがい、カセット・ドライブのPLAYボタンを押して下さい。

画面には次のようなメッセージが出てきます。

OK
SEARCHING FOR TEST
FOUND TEST

VERIFYING

OK

READY.

なお、前に述べた理由により、SEARCHING……のメッセージとFOUND TESTのメッセージの間に約10秒間の間隔があります。

上記のメッセージが表示されれば、プログラムは正しくSAVEされました。

もし、SAVEが正しくなされていない場合は、次のような結果となります。

- 1) もし、ヘッダーがクリアフィーダーの終わらないうちに書き込まれた場合、ヘッダーが検知されないことにより、FOUND TESTのメッセージがいつまで待っても出てきません。
- 2) もし、何らかの理由により、うまくSAVEされていない場合には、最後のOKの代わりにVERIFY ERRORというメッセージが表示されます。

以上でおわかりと思いますが、VERIFYとは、コンピューターのメモリー中にあるプログラムと、指示されたプログラム名で記録されたカセット・テープ内のプログラムを比較し、一致しているかどうかを「確認する」ことをいうのです。

ここで、いったんコンピューターのメモリーをクリアーするためにNEWというコマンドを入力して下さい。その次にLISTと入力し、メモリーがクリアーされたことを確認して下さい。

第3節 プログラムのLOAD

ここでは、プログラムをカセット・テープからLOADする方法について説明します。

次のように、キーボードから入力して下さい。

LOAD "TEST"

画面に次のようなメッセージが表示されます。

PRESS PLAY ON TAPE

指示にしたがい、カセット・ドライブのPLAYボタンを押すと、画面に次のよ

うなメッセージが出ます。

OK

SEARCHING FOR TEST
FOUND TEST
LOADING

READY.

これで、プログラムのLOADが終了したことになります。

この場合、このプログラムを実行するには、キーボードからRUNを入力する必要がありますが、LOAD後自動的にRUNさせる場合(ただし、プログラム名は指定できません)、**SHIFT**キー(あるいは **C** キー)と **RUN STOP** キーを同時に押すことにより、可能です。いったんテープを巻き戻して、操作してみてください。

また、その時次の実験をしてみてください。LOAD "XYZ" と入力し、画面の指示にしたがい、カセット・ドライブのPLAYボタンを押して下さい。画面には次のようなメッセージが表示され、モーターは回転し続けるはずです。

SEARCHING FORXYZ
FOUND TEST

おわかりのように、指定されたプログラム名以外のプログラムは、その名前だけを表示し、LOADはしません。

第4節 データのLOADおよびSAVE

この項を説明するには、2本のテープが必要になります。以下の説明をお読みになる前にまず、2本のテープをご用意下さい。

以下のプログラムをキーボードから入力し、前項で説明した要領でPGM 1 というプログラム名でSAVEして下さい。

```
10 OPEN 1, 1, 1  
20 FOR J=1 TO 20  
30 PRINT # 1, "I AM A VIC. /" (? を用いないで、PRINT...として下さい)
```

```

40 NEXT J
50 CLOSE 1
60 PRINT "REWIND YOUR TAPE AND THEN PRESS A SPACE KEY"
70 GET A$:IF A$=" "THEN70(クオート"を続けて2回タイプして下さい)
80 OPEN 1
90 FOR J=1 TO 20
100 INPUT#1, X$
110 PRINT X$
120 NEXT J
130 CLOSE 1
140 PRINT "WONDERFUL!!"

```

そのカセットを巻き戻し、PGM 1と書いたラベルをはって置いて下さい。別のカセットを入れて下さい。完全に巻き戻されていることを確認後、上のプログラムをRUNさせて下さい。

PRESS RECORD & PLAY ON TAPE

という表示が出ますから、指示にしたがいカセット・ドライブのRECボタンを押しながらPLAYボタンを押して下さい。

すると、OKが表示され、その約25秒後に、以下が表示されます。

REWIND YOUR TAPE AND THEN PRESS A SPACE KEY

指示にしたがい、REWボタンを押してテープを巻き戻し、キーボードのスペース・キーを押します。すると、

PRESS PLAY ON TAPE

と表示されますので、指示にしたがいカセット・ドライブのPLAYボタンを押して下さい。すると、OKと表示が出て、約25秒待つとI AM A VIC!という文章が20行プリントされ、最後にWONDERFUL!!と表示し、READY状態となります。

●注意● この時、I AM A VIC!を16行プリントし、テープがいったん止まりますが、これはバッファがいっぱいになるためで、支障はありません。

ここで、テープを巻き戻し、DATA 1と書いたラベルをはって置いて下さい。

以上で、一連の作業は終わりました。

PGM 1とマークされたテープがプログラム・テープであり、DATA 1とマークされたテープがデータ・テープです。ここで、ちょっとした実験をしてみましょう。DATA 1とマークされたテープをカセット・ドライブに入れ、**SHIFT** キーを押しながら **RUN STOP** キーを押して下さい。画面に

PRESS PLAY ON TAPE

が表示されます。この表示にしたがい、カセット・ドライブのPLAYボタンを押して下さい。

画面には、
OK

SERCHING
FOUND

と表示されるだけで、LOADINGというメッセージは出てきません。つまり、データ・テープは、プログラム・テープとは違ってLOADすることはできません。以下でライン・ナンバー順に何がなされているかを説明します。

ライン・ナンバー10

ロジカル・ファイル1（最初の1）をデバイス・ナンバー1（2番目の1……カセット）用に、しかも書き込み用（最後の1）にOPENする（開く）。

ライン・ナンバー20~40

テープに I AM A VIC! を20回、書き込む。

ライン・ナンバー50

ロジカル・ファイル1をCLOSEする（閉じる）。

ライン・ナンバー60~70

テープを巻き戻すメッセージを表示し、スペース・キーが押されるまで待つ。メッセージは、スペース・キーを押すとありますが、実際には、何でもキーが押されれば、次のライン・ナンバーに進みます。

ライン・ナンバー80

ロジカル・ファイル1をOPENする。

ライン・ナンバー90~120

テープから I AM A VICを20回読み取り、表示する。

ライン・ナンバー130

ロジカル・ファイル1をCLOSEする。

ライン・ナンバー140

WONDERFUL!!を表示する。

もう少し詳しく説明しましょう。ライン・ナンバー10は、カセット・ドライブ（周辺機器には必ず、デバイス・ナンバーが割り当てられており、VIC-1001の場合、カセット・ドライブが1となっています）用に、チャンネルを、書き込み用にOPENしたわけです。つまり、最初の1はOPENしたいチャンネル番号を示し、次の1はデバイス・ナンバーを示し、最後の1は、そのチャンネルを、書き込み用にOPENすることを意味します。

VIC-1001では、チャンネル（正確にはロジカル・ファイルと呼ばれます）を10本までOPENすることが可能です。10本以上のチャンネルがOPENされると、

**? TOO MANY FILES
ERROR**

となります。

また、ロジカル・ファイル・ナンバーは1から255までの間の整数であれば、いくつでもかまいません。256以上の数字を用いると、

**? ILLEGAL QUANTITY
ERROR**

となります。

最後の1は、書き込み用にチャンネルをOPENすることを意味しています（テープから、読み取る場合は、0となります。——ライン・ナンバー80でもう少し説明します）。

まとめをかねて、もう一度説明しますと、データをカセット・テープやフロッピー・ディスクに書き込むには、

OPEN A, B, 1

とし、その場合Aは、ロジカル・ファイル・ナンバーであり、Bがデバイス・ナンバーとなります。カセットの場合、Bは1です。

ライン・ナンバー20から40では、OPENしたロジカル・ファイルに20回の I AM A VIC! を送り込んでいます。そしてライン・ナンバー50で、先にOPENされたロジカル・ファイル1をCLOSEします。この場合、OPENしたロジカル・ファイル・ナンバーと同じナンバーをCLOSEしないと、ライン・ナンバー80でエラーとなります。つまり、いったんOPENしてあるロジカル・ファイルをCLOSEせずに、再度OPENすると、

```
? FILE OPEN
  ERROR IN 80
```

となります。

ライン・ナンバー70以下では、書き込まれたデータの読み込みをします。ライン・ナンバー80で、先ほどと同様にロジカル・ファイル1をOPENしています。このライン・ナンバーは、実際には、

```
80 OPEN 1, 1, 0
```

であるべきものです。しかし、特に指示しないかぎり、デバイス・ナンバーは1であり、ロジカル・ファイルは読み取り (0—これをセカンダリー・アドレスと呼びます) にOPENされます。ライン・ナンバー10で、デバイス・ナンバーおよびセカンダリー・アドレスを定義したのは、CBM BASICでは常にOPEN<ロジカル・ファイル・ナンバー>、<デバイス・ナンバー>、<セカンダリー・アドレス>の順に書く必要があります、ライン・ナンバー10ではセカンダリー・アドレス0でなく、1であるため、必然的にデバイス・ナンバーも必要になったわけです。

前に、プログラムのLOADの所で、PGM 1をLOADする場合、

```
LOAD "PGM 1"
```

としましたが、これも実際には

```
LOAD "PGM 1", 1
```

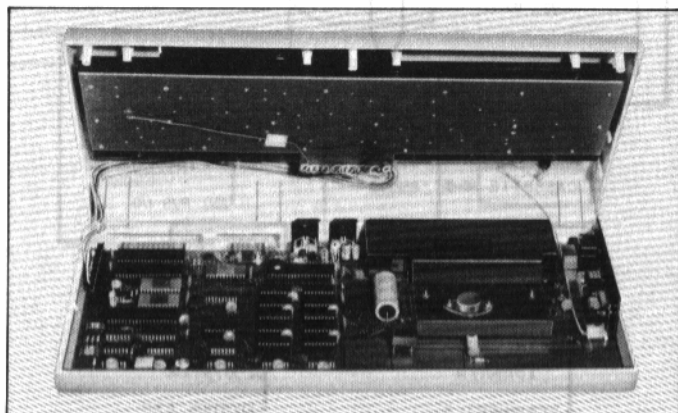
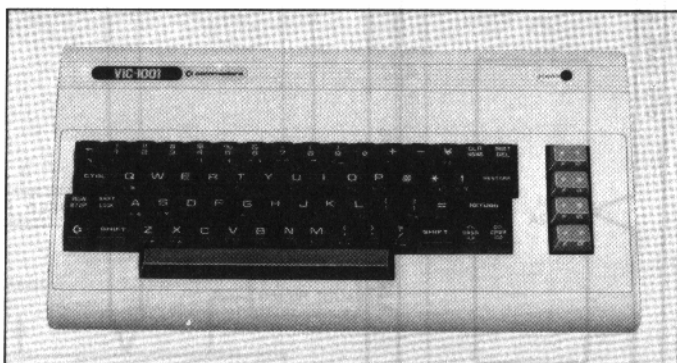
であったわけで、この1は、デバイス・ナンバーです。したがって、1であるため、省略ができたわけです。

第6章 ハードウェアの説明

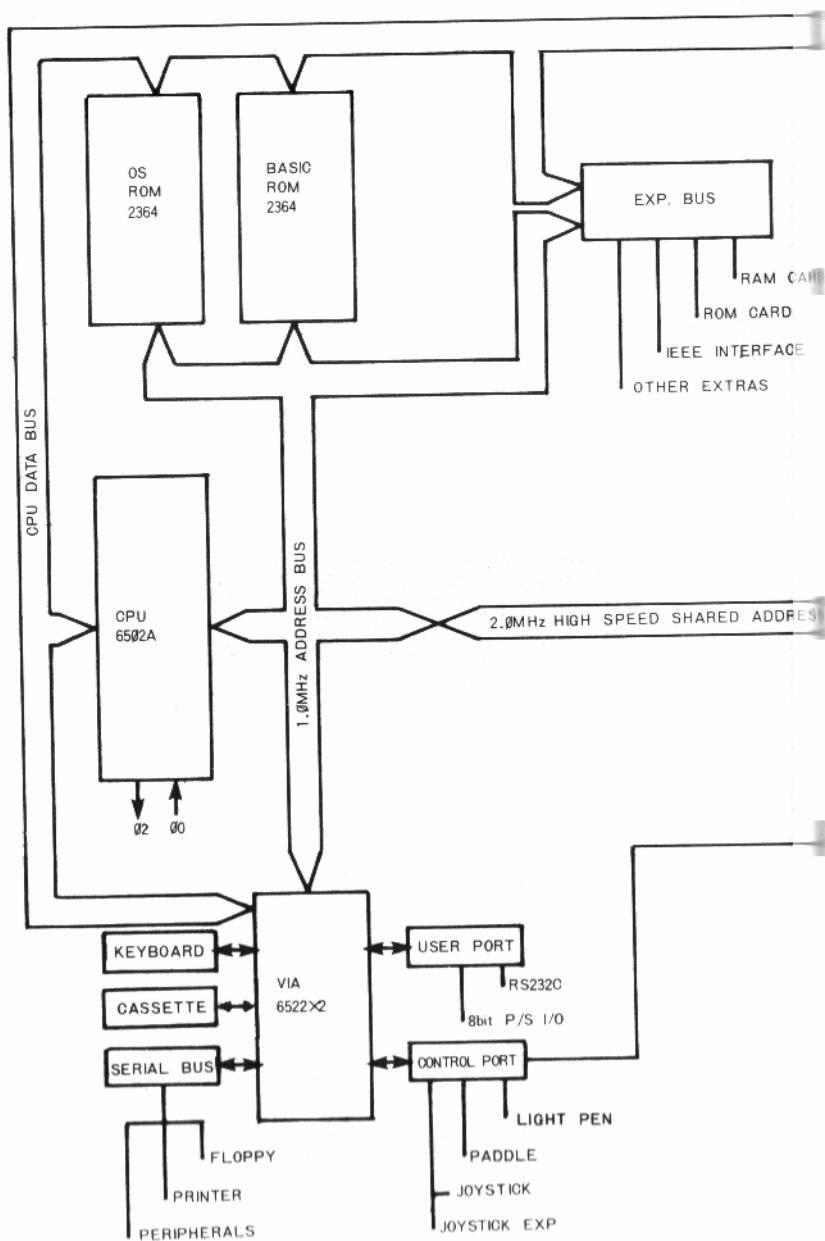
第1節 概論

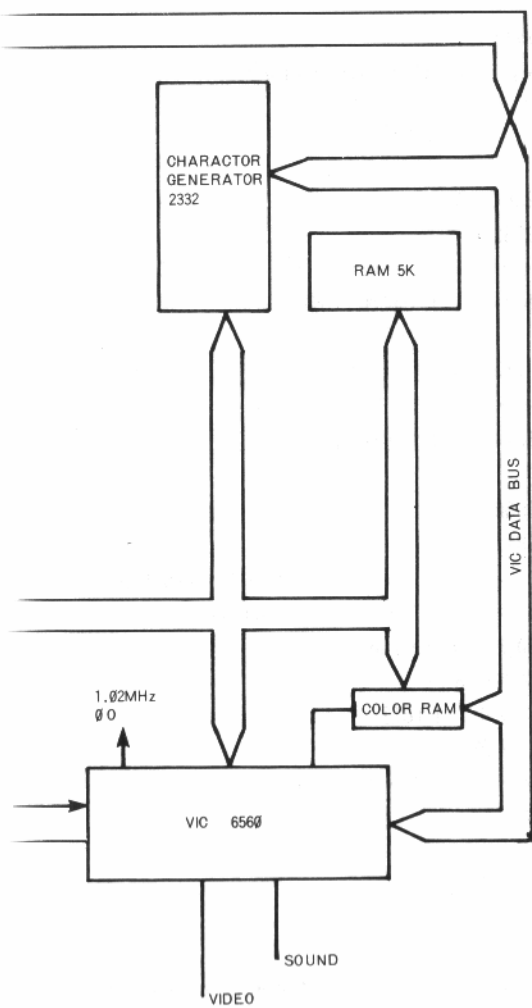
VIC-1001は、CPUにMOS TECHNOLOGY社（コモドールの半導体部門）製のMPS-6502Aが使用されているパーソナル・コンピュータです。このクラスでは最も充実したインターフェイスを内蔵しており、システムの拡張が多様かつ容易に可能です。以下では、VIC-1001のハードウェアについて説明します。

第2節 外観



第3節 システム構成





第4節 機能仕様

(1)CPU

使用CPU : MPS-6502A
CLOCK 周波数 : 1.0227MHz (14.31818MHzを14分周)
RESET : パワー・オン
割込み : カード・エッジに開放

(2)ROM

使用ROM : MPS 2364×2
: MPS 2332×1
容量 : 20Kバイト
: 最大32Kバイトまで拡張可能

(3)RAM

使用RAM : MPS 2114×10
容量 : 5Kバイト
: 最大32Kバイトまで拡張可能

(4)CRTインターフェイス

使用コントローラ : MPS 6560
スクリーン構成 : 22桁×23行
文字構成 : 8×8ドットマトリックスキャラクター128種類
: 8×8グラフィックキャラクター64種類
グラフィック機能 : 176×160または88×160
カラー機能 : キャラクター8色
: バックグラウンド16色
: ボーダー8色
カーソル : リバース・ブリンク・カーソル
その他 : リバース

(5)カセット・インターフェイス

方式 : コモドル方式
ボーレート : 500ボ

(6)サウンド・ジェネレーター

構成 : サウンド・ジェネレーター×3(別個にコントロール可能)
: ホワイト・ノイズ・ジェネレーター×1
音域 : 128Hz~16KHz

(7)ユーザー・ポート

方式 : 8ビット・パラレル / 8ビット・シリアル入出力兼用

(8)シリアル・インターフェイス

方式 : コモドル方式
ボーレート : 50/70/134.6/150/300/600/1200/2400/3600

(9)キーボード

- 方式 : ソフトウェア・スキャン
- キー : J I S 準拠フルキーボード
4 ファンクション・キー、カーソル・コントロール・キー、
コントロール・キー、コモドル・キー、レストア・キー

(10)電源

- 電源電圧 : AC100V±10%、50/60Hz
- 消費電力 : 18W

(11)使用条件

- 使用温度 : 0°C~40°C
- 使用湿度 : 20%~80% (ただし結露しないこと)
- 保在温度 : -5°C~60°C

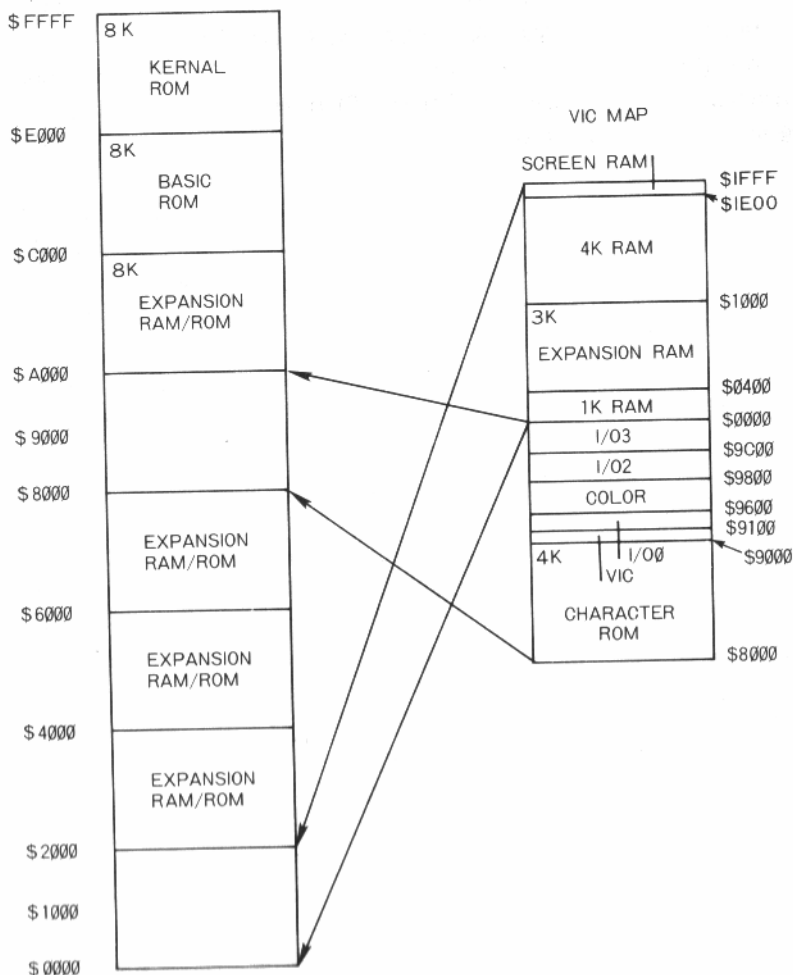
(12)外形寸法 : 404(W)×204(D)×74(H)mm

(13)重量 : 1.8kg

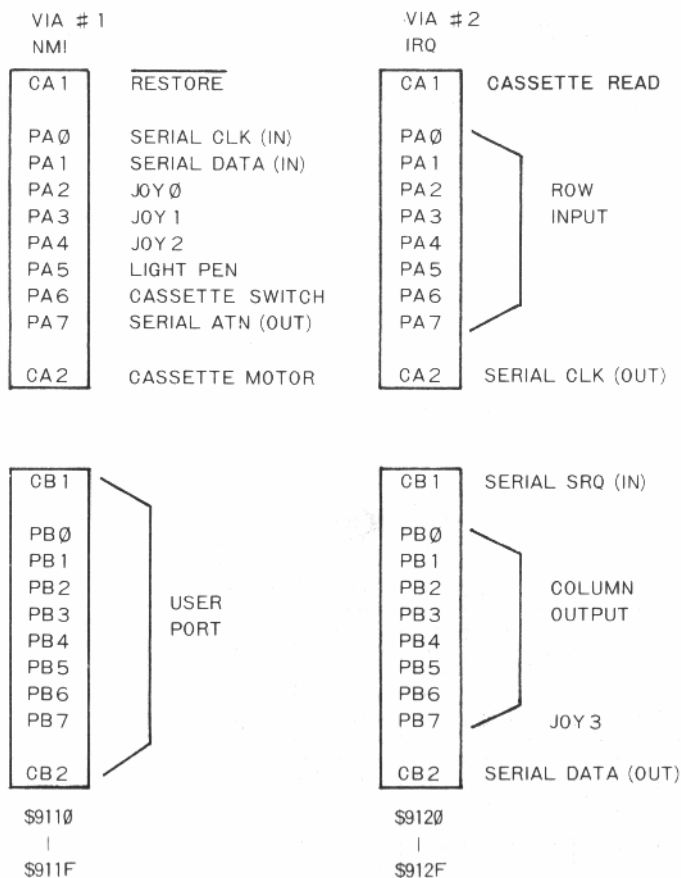
第5節 メモリー・アドレス・マップ

VIC-1001のメモリーには、BASICのインタープリターが収納されているマスクROM (MPS2364) と、スタックRAM (MPS2114) が使われており、ROM20Kバイト、RAM5Kバイトが実装されています。ROMおよびRAMはともに、32Kバイトまで拡張可能になっています。

次にVIC-1001のメモリー・マップを示します。



MPS-6522のポート・アサメイメントは下図を参照して下さい。



第6節 キーボード

VIC-1001のキーボードは、ソフトウェア・スキャン方式をとっています。マトリックス状に配置されたキースイッチは、CPUのインプット命令によりスキャンされ、押されたキーの情報は、ソフトウェアによりコードに変換されます。

次に、キーボードマトリックスを示します。

図-1

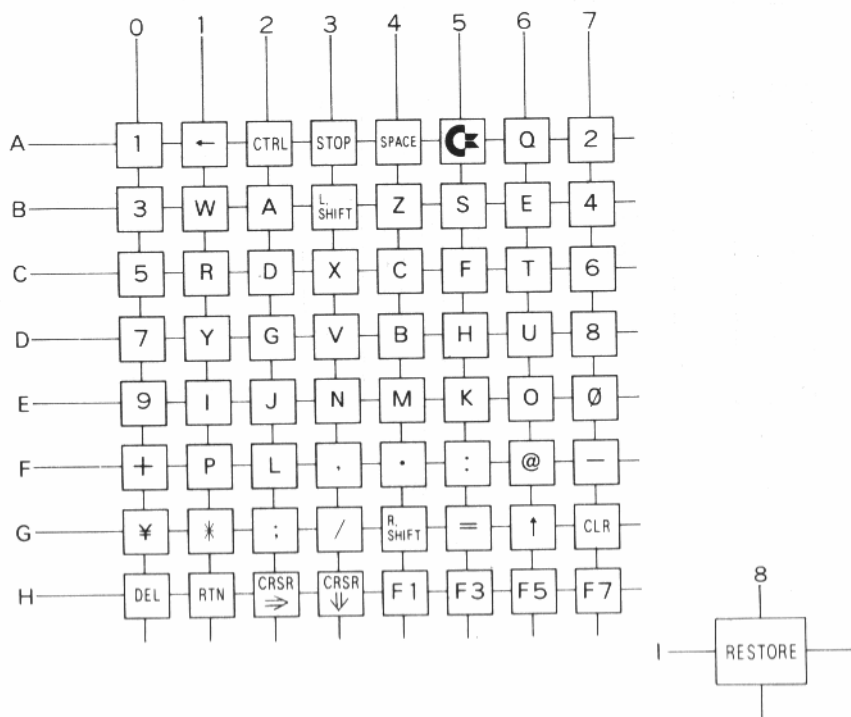


图-2

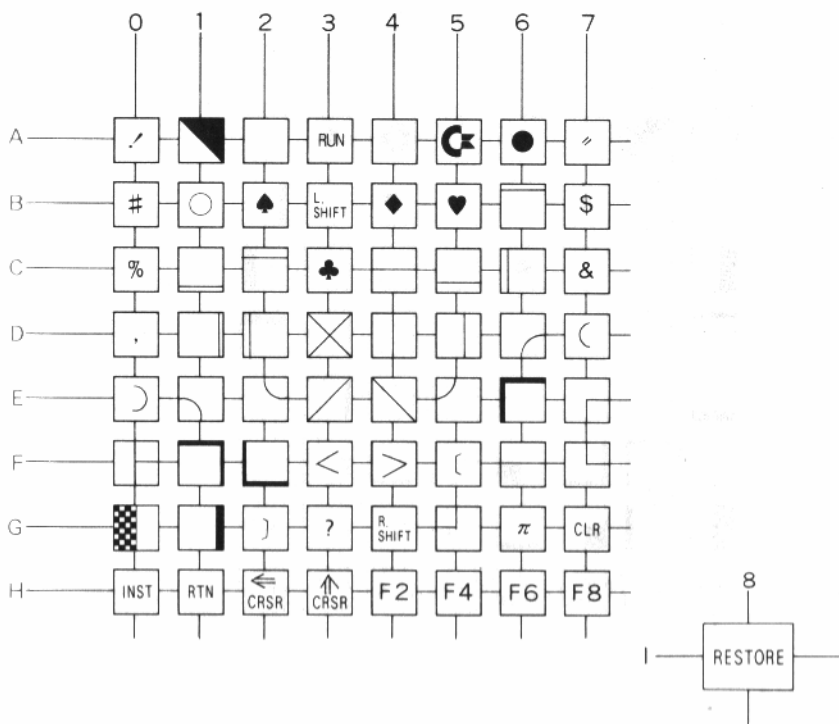


图-3

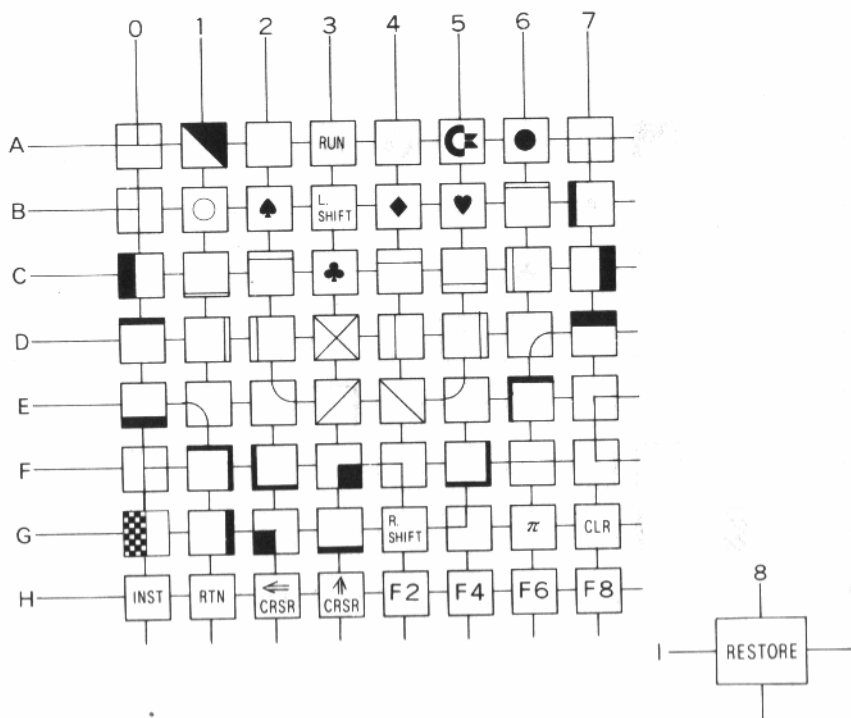


図-4

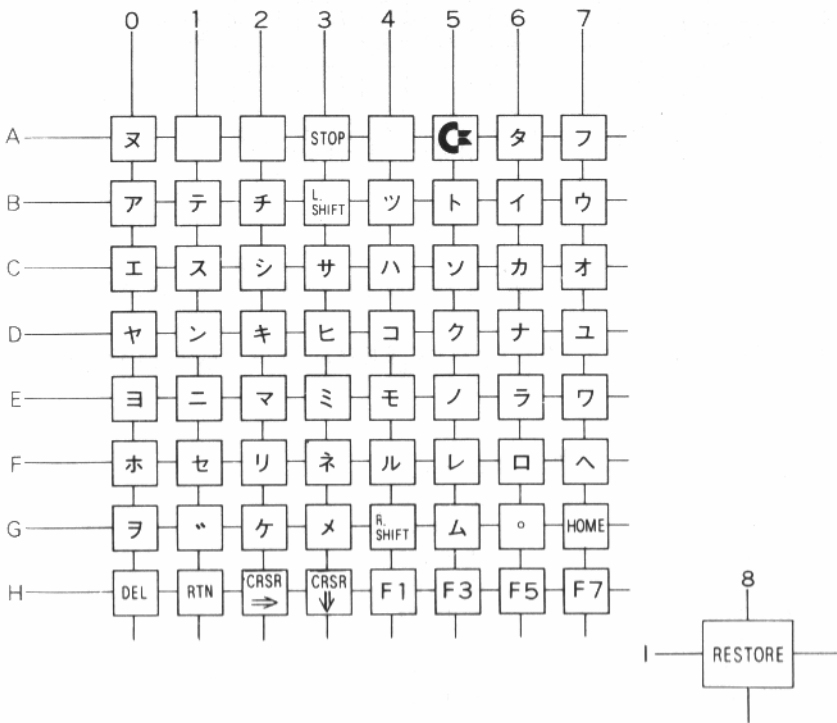
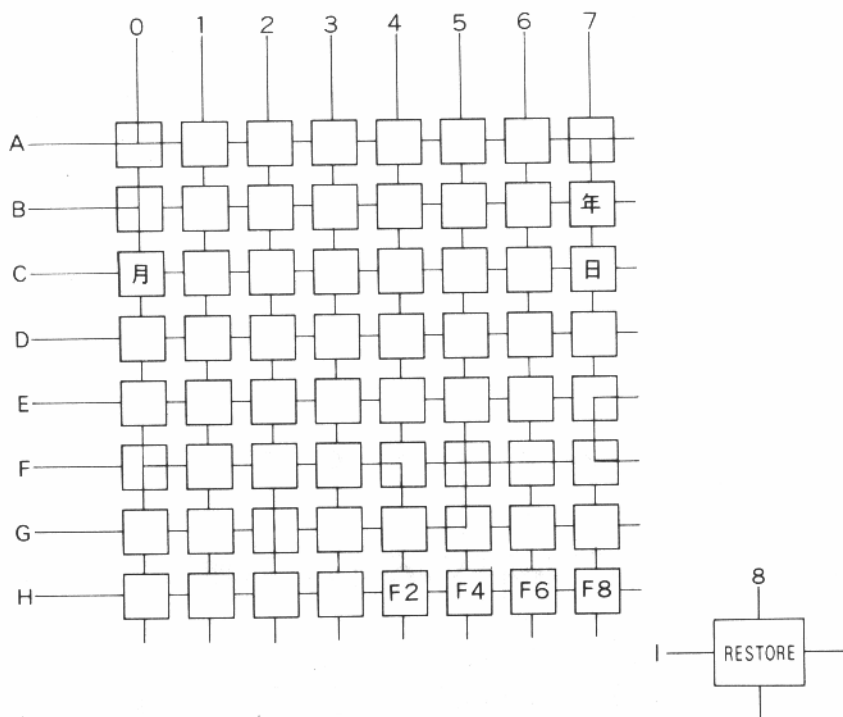


図-5



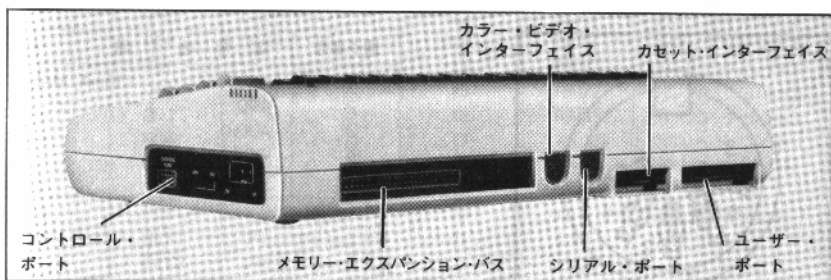
また、キーボードコネクターの端子と信号の関係は以下のようにになっています。

- | | |
|-------------------|----------|
| 1) GND | 11) COL1 |
| 2) KEY | 12) COL0 |
| 3) <u>RESTORE</u> | 13) ROW7 |
| 4) +5V | 14) ROW6 |
| 5) COL7 | 15) ROW5 |
| 6) COL6 | 16) ROW4 |
| 7) COL5 | 17) ROW3 |
| 8) COL4 | 18) ROW2 |
| 9) COL3 | 19) ROW1 |
| 10) COL2 | 20) ROW0 |

第7節 インターフェイス

VIC-1001には、以下に示すインターフェイスが標準装備されています。

- 1) ユーザー・ポート
- 2) カセット・インターフェイス
- 3) カラー・ビデオ・インターフェイス
- 4) シリアル・ポート
- 5) メモリー・エクспанション・バス
- 6) コントロール・ポート

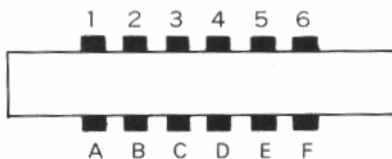


1) ユーザー・ポート



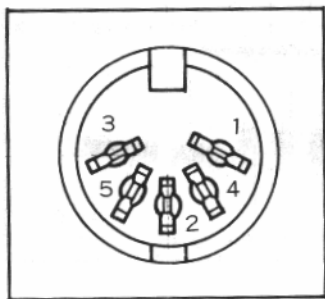
端子番号	信号名	備考	端子番号	信号名	備考
1	GND		A	GND	
2	+5V	100mA MAX.	B	CB1	
3	RESET		C	PB0	
4	JOY0		D	PB1	
5	JOY1		E	PB2	
6	JOY2		F	PB3	
7	LIGHT PEN		H	PB4	
8	CASSETTE SWITCH		J	PB5	
9	SERIAL ATN IN		K	PB6	
10	+9V	100mA MAX.	L	PB7	
11	GND		M	CB2	
12	GND		N	GND	

2) カセット・インターフェイス



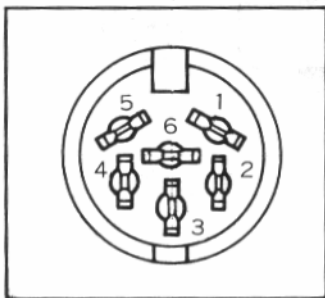
端子番号	信号名
A-1	GND
B-2	+5V
C-3	CASSETTE MOTOR
D-4	CASSETTE READ
E-5	CASSETTE WRITE
F-6	CASSETTE SWITCH

3) カラー・ビデオ・インターフェイス



端子番号	信号名	備考
1	+6V	10mA MAX
2	GND	
3	AUDIO	
4	VIDEO LOW	
5	VIDEO HIGH	

4) シリアル・ポート



端子番号	信号名
1	SERIAL SRQ IN
2	GND
3	SERIAL ATN IN/OUT
4	SERIAL CLK IN/OUT
5	SERIAL DATA IN/OUT
6	NC

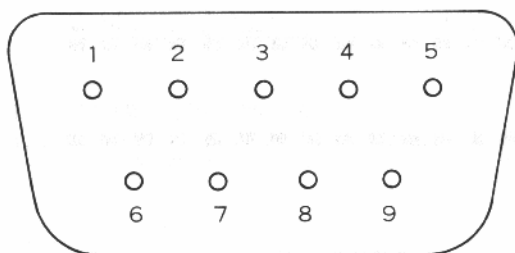
5) メモリー・エクспанション・バス



端子番号	信号名	備考	端子番号	信号名	備考
1	GND	CPUデータ	12	<u>BLK3</u>	ブロックセレクト3
2	CD0	IN/OUT	13	<u>BLK5</u>	ブロックセレクト5
3	CD1	◇	14	<u>RAM1</u>	ラム・ブロックセレクト1
4	CD2	◇	15	<u>RAM2</u>	ラム・ブロックセレクト2
5	CD3	◇	16	<u>RAM3</u>	ラム・ブロックセレクト3
6	CD4	◇	17	VR/W	VICリード/ライト
7	CD5	◇	18	CR/W	CPUリード/ライト
8	CD6	◇	19	<u>TRQ</u>	
9	CD7	◇	20	NC	ノーコネクション
10	<u>BLK1</u>	ブロックセレクト1	21	+5V	MAX 500mA
11	<u>BLK2</u>	ブロックセレクト2	22	GND	

端子番号	信号名	備考	端子番号	信号名	備考
A	GND		N	CA10	CPUアドレス10
B	CA0	CPUアドレス0	P	CA11	◇ 11
C	CA1	◇ 1	R	CA12	◇ 12
D	CA2	◇ 2	S	CA13	◇ 13
E	CA3	◇ 3	T	I/02	I/Oセレクト2
F	CA4	◇ 4	U	I/03	◇ 3
H	CA5	◇ 5	V	S02	システム・クロック2
J	CA6	◇ 6	W	<u>NMI</u>	
K	CA7	◇ 7	X	<u>RESET</u>	
L	CA8	◇ 8	Y	NC	ノーコネクション
M	CA9	◇ 9	Z	GND	

6) コントロール・ポート



端子番号	信号名	備考
1	JOY0	
2	JOY1	
3	JOY2	
4	JOY3	
5	POT Y	
6	LIGHT PEN	
7	+5V	MAX. 100mA
8	GND	
9	POT X	

第8節 CRT制御方式

VIC-1001では、画面に文字および図形を表示するために専用のビデオ・インターフェイス・チップ (MPS-6560) を使用しています。

画面に表示される文字および図形のデータは、RAM上に配置されたビデオRAM上に格納されます。DMAコントローラは、このビデオRAM内のデータをDMA転送 (ダイレクト・メモリ・アクセス) により、文字および図形データをそのバッファに一時記憶し、画面に同期してビデオ信号発生回路に出力します。

ビデオ信号発生回路は、タイミング回路によって発生された同期信号と、映像信号とを合成して、コンポジットビデオ信号を発生させます。

ビデオRAM

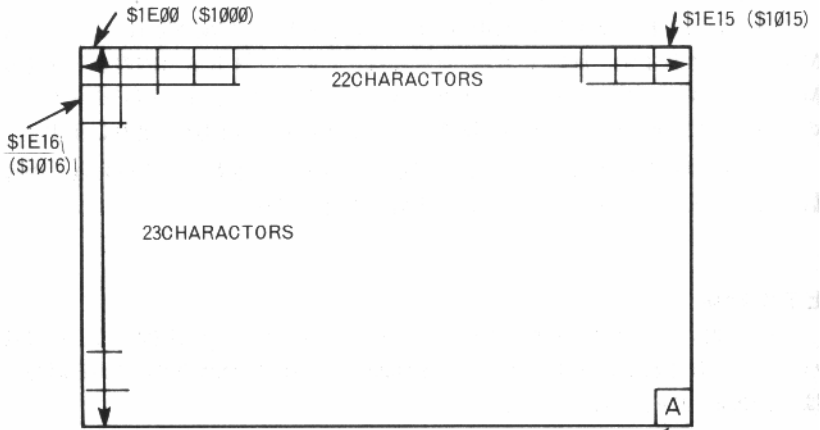
VIC-1001のビデオRAMは、メイン・メモリーのアドレス\$1E00~1FFFの512バイトに配置されています (ただし、\$2000以上にRAMが増設されている場合には、\$1000~11FFとなります)。

次ページに画面とビデオRAMの対応を示します。

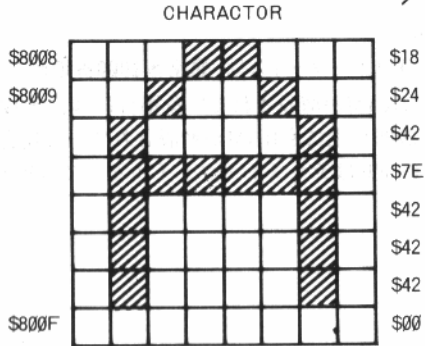
BASICテキストファイル

VIC-1001のBASICテキスト・ファイルは、通常\$1000番地からスタートします。ただし、\$0400~0FFFにRAMが増設されている場合には、\$0400がスタート番地になります。しかし\$2000以上にもRAMが増設されている場合は、\$1000番地からスタートし、\$0400~0FFFはテキスト・ファイルとしては扱われません。

ビデオRAMの構成



() 内のアドレスはRAM EXPを\$2000以上に置いた場合



第2部



ソフトウェア編

第1章 CBM BASICの概要

CBM BASIC (V 2)は、VIC-1001のプログラミング言語です。パーソナル・コンピューターのパイオニアであるPET/CBMのコモドールBASICに基づいており、コモドールBASIC同様、数多くあるBASIC言語の中でも、特に分りやすく、使いやすい言語です。

この章では、CBM BASICの概要を説明し、第2章でコマンド・ステートメントの、第3章で関数の説明をおこないます。第4章では、CBM BASIC を使ったプログラム例をいくつかあげます。

第1節：初期状態

コールドスタートするためには、次の3通りの方法があります。

- ①電源の投入
システムのリセット、およびOS (オペレーションシステム) とBASIC の変数をすべて初期状態にします。
- ②リセット信号
メモリー拡張用コネクタよりリセット信号を送ります。
- ③コマンドによるリセット
SYSまたは、ブランチ命令により、リセット番地へジャンプします。
以上、いずれの場合も、ディスプレイ上は、次のように表示されます。

***** CBM BASIC V2 *****

3583 BYTES FREE

READY.

第2節：動作モード

VIC-1001に電源を入れ、BASICが走るようにすると、ディスプレイ上で、カーソルが点滅しています。これは、BASICが、コマンドを受け取る準備のできていることを意味します。BASICは、ダイレクト・モードか、または、プログラム・モードのどちらかで使用することができます。

ダイレクト・モード……この場合、BASICのステートメントおよびコマンドにライン・ナンバーをつけません。また命令が入力された時点で、すぐに実行されます。算術および論理演算の結果を直ちに表示したり、後で使うために格納することができますが、命令は実行後失なわれます。このモードは、デバッグをおこなう場合や、BASICを完全なプログラムを必要としない簡単な計算をすぐにおこなうカリキュレーターとして使うのに便利です。

プログラム・モード……このモードはプログラムを入力する場合に使用します。プログラムの各ライン（行）は、ライン・ナンバーではじまり、メモリーに格納されます。メモリーに格納されたプログラムはRUNコマンドを入力することにより実行されます。

第3節：ラインの書式

ラインはBASICプログラムの基本的な単位です。BASICのラインの書式は次のとおりです。

nnnnn BASICステートメント[:BASICステートメント……] (キャリッジ
ライン・ナンバー リターン)

プログラムは1ラインに1つ以上のBASICステートメントを入れることができますが、各々のステートメントはコロン(:)で区切られていなくてはなりません。

BASICプログラムは必ずライン・ナンバーではじまり、キャリッジリターンで終わります。1ラインにつき最大88文字まで入れることができます。

第4節：ライン・ナンバー

BASICプログラムは必ずライン・ナンバーではじまります。ライン・ナンバーはプログラムがメモリーに格納される順番をあらわしています。また、プログラム中での分岐や編集をおこなう場合の目印として使用されます。

ライン・ナンバーは0から63999までの整数を使います。

第5節：キャラクターセット

CBM BASICのキャラクターセット（使用文字）は、英文字、数字、グラフィック文字、カナ文字、そして特殊文字があります。特殊文字および特殊キーは次の通りです。

特殊文字	名称
	ブランク (空白)
;	セミコロン
=	イコール、等号
+	プラス、加算の演算子
-	マイナス、減算の演算子
*	アスタリスク、乗算の演算子
/	スラッシュ、除算の演算子
↑	上向き矢印、べき乗の演算子
(左かっこ
)	右かっこ
%	パーセント
#	ナンバー記号
\$	ドル記号
!	感嘆符
[左かぎかっこ
]	右かぎかっこ
,	コンマ
.	終止符 (ピリオド) または小数点
'	シングルクォーテーション (アポロストロフィ)
"	ダブルクォーテーション (引用符)
:	コロンの
&	アンド
?	疑問符 (クエスチョン・マーク)
>	不等号 (より小さい)
<	不等号 (より大きい)
@	アット・マーク (単価記号)
←	左向き矢印
¥	円記号
π	円周率 (3.14159265)

特殊キー	機 能
〈CLR〉	ディスプレイ画面を消去して、カーソルをホームポジション（左上スミ）に戻します。
〈CRSR⇒〉	カーソルを右へ1文字ずらします。
〈CRSR⇐〉	カーソルを左へ1文字ずらします。
〈CRSR↑〉	カーソルを上へ1文字ずらします。
〈CRSR↓〉	カーソルを下へ1文字ずらします。
〈CTRL〉	数字の1から8と併用することにより、ディスプレイ画面の色を変えることができます。また、CTRL+Rで、文字を反転させることができ、CTRL+Øで反転を解除できます。さらにまた、このキーを押すことにより、スクロール速度を落とすことができます。
〈DEL〉	カーソルの左側にある文字を1字削除し（それより右の文字を左へ詰め）ます。リピート機能付き。
〈HOME〉	カーソルをホームポジション（左上スミ）に戻します。
〈INST〉	カーソルの位置とその左側の文字との間にスペースを入れ、文字の挿入を可能にします。
〈RESTORE〉	[STOP] キーと同時に押すことにより、システムのリセットをおこなわずに、BASICを再スタートできます。
〈RETURN〉	1行の入力を終了させます。そのさい、カーソルを次列の左端に移します。カーソルが最下段にある場合は、画面をスクロールします。
〈RUN〉	LOAD+RUNのコマンドを入力したのと同じ働きをします。
〈SHIFT〉	キャラクター・セットの変換に用います。
〈SHIFT LOCK〉	SHIFT状態のキャラクター・セットに保ちます。
〈STOP〉	プログラムの実行を中止して、BASICのダイレクト・モードに戻します（ディスプレイにはREADY.が表示され、カーソルが点滅します）。また、プログラムをカセット・ドライブからLOADするのを中止させる時にも用います。
〈C〉	[SHIFT] キーと同様の働きをします。[SHIFT] キーと併用して、グラフィック・キャラクター・モード↔片仮名キャラクター・モードの変換をおこないます。また、片仮名モードⅠとⅡの切換えをおこないます。

第6節：定数

定数はBASICが実行時にそのまま使用する値です。定数には文字定数および数値定数の2種類があります。

文字定数は引用符(ダブルクォーテーション)に囲まれた最大255文字までの文字の列です。文字定数の例をあげましょう。

"COMMODORE"

"1980"

"コモドール"

数値定数は正または負の数です。BASIC中での数値定数は、","(コンマ)を持つことができません。

数値定数には次の2つの形式があります。

1. 整数形式

-32768から+32767までのすべての整数。整数は小数点をふくむことはありません。

2. 浮動小数点形式

指数形式で表現された正または負の数値です。浮動小数点形式では数値(仮数部)に続き文字Eそして符号(+のばあいはなくてもよい)つきの整数(指数部)で表現されます。指数部の範囲は-39から+38までです(CBM BASICで扱える実数は、 $2.93873588E-39 \sim 1.70141183E+38$ までです)。CBM BASICでは、浮動小数点形式の数値定数は、10桁の精度で格納され、9桁までの桁数で表示されます。

浮動小数点形式の数値は、次のいずれかに該当します。

1. 10桁以下の数値

2. 指数形式で表示

3. 浮動小数点形式の変数として割りあてられた場合、

例) 28.9

-4.09E-05

3.7

第7節：変数

変数とはBASICプログラム中で使われる値をあらわすために用いられる名前です。変数の値はプログラマーにより定義されたり、プログラムによる演算の結果が割りあてられたりします。変数は値が割りあてられるまでは「0」（ゼロ）とみなされます。

1) 変数名および型宣言文字

BASICでは変数名は何文字でもかまいませんが、最初の2文字だけが意味を持ち、それにより区別されます。最初の1文字は必ず英文字でなければなりません。残りの文字は英文字または数字または型宣言文字のどれでもかまいません。

変数名は予約語（BASICで使用される、コマンド・ステートメント・関数名など）であってはなりません。また予約語を含んでもいけません。たとえばB00という変数名はB00という予約語を含んでいるので認められません。

変数は数値または文字列のいずれもあらわすことができます。文字変数名は最後に **\$**（ドル記号）をつけて表現します。

例) **A\$**="ウリアゲニッポウ"
AB\$="ABC"

この **\$**（ドル記号）は、変数が文字列であることを宣言する型宣言文字です。数値変数が、整数変数であることを宣言するには変数名の最後に「%」（パーセント記号）をつけておこないます。

それら以外の変数名を用いる場合は、浮動小数点形式の変数になります。

例) (変数名)

MI	浮動小数点形式の変数
LIMIT%	整数変数
N\$	文字変数

2) 配列変数

配列は、同じ変数名で参照することのできる値の集まり、またはテーブルです。

配列のそれぞれの要素は、整数または整数表記による添字を伴った配列変数により参照されます。配列変数名はその配列の次元の数と同じ個数の添字を持ちます。たとえば、V(10)は1次元配列の値として、D(1,4)は2次元配列の値としてあつかわれます。2次元以上の場合も同様です。

第8節：型の変換

BASICは、必要に応じて数値定数の型を、ある型から他の型に自動的に変換します。この場合、以下に述べる点について留意して下さい。

1. ある型の数値変数が他の異なった型の数値変数に割りあてられた場合は、その変数名により宣言された型に変換して格納されます。(もし文字変数が数値に、またはその逆が割りあてられた場合は「TYPE MISMATCH」エラーが発生します。)

```
例) 10 A%=12.34
     20 PRINT A%
     RUN
     12
```

2. 式を計算する場合、算術または論理演算のオペランドはすべて浮動小数点形式の同じ精度でそろえられます。整数は評価のためにいったん浮動小数点形式に変換され、再度整数形式に戻されます。
3. 論理演算では、オペランドは整数に変換され、整数の結果が得られます。オペランドが-32768から+32767の範囲内でない場合は「OVER FLOW」エラーが発生します。
4. 浮動小数点形式で表わされた数値が整数に変換される場合には、小数部分は丸められ、変換された結果の整数はもとの浮動小数点形式の数値より小さいか等しい値をもちます。

```
例) 10 AZ=12.24
     20 PRINT A%
     RUN
     12
```

第9節：式と演算

式とは単に文字または数値定数、または変数、あるいはある値を得るために定数や変数を演算子で結合したものです。

演算子は与えられた値について、算術または論理演算を実行します。BASICに備わっている演算は大きくわけて次の4つに分類することができます。

1. 算術演算
2. 関係演算
3. 論理演算
4. 関数

1 算術演算

算術演算子を演算の順位にしたがって、ならべると次のようになります。

(演算子)	(演算)	(例)
↑	指数	$X \uparrow Y$
—	否定 (負号)	$-X$
*, /	乗算、浮動小数点の除算	$X * Y, X / Y$
+, -	加算、減算	$X + Y, X - Y$

演算の順序を変える場合は、カッコ“(”、“)”)”を使用します。カッコ内の演算が最初におこなわれます。カッコの内部では通常の演算順位が適用されます。

演算をおこなう場合、カッコのネスティング (入れ子) 使用は最大10個までです。

通常の数式表現とそれらをBASICで表現した例を示しましょう。

数式表現	BASICでの表記
$X + 2 Y$	$X + 2 * Y$
$X - Y \div Z$	$X - Y / Z$
$X \times Y \div Z$	$X * Y / Z$
$(X^2) Y$	$(X \uparrow 2) \uparrow Y$
$X^{(Y^2)}$	$X \uparrow (Y \uparrow 2)$
$X (-Y)$	$X * (-Y)$ ……演算子が連続する場合はカッコで区切って下さい。

●注意●

0による除算とオーバフロー：数式にたいする演算処理の実行中に除数が0の場合には、“DIVISION BY ZERO”エラーが発生します。また、オーバフローが生じた場合には、“OVER FLOW”エラーが発生します。

(2)関係演算子

関係演算子は2つの値を比較するのに用いられます。比較の結果は「真」(−1)、または「偽」(0)になります。この結果を用いてプログラムの流れの決定したりすることができます(2-13を参照)。

演算子	説明	表記
=	等しい	X=Y
<>	等しくない	X<>Y
<	より小さい	X<Y
>	より大きい	X>Y
<=	等しいか小さい	X<=Y
>=	等しいか大きい	X>=Y

(符号は変数に値を代入するのにも使われます。2-16 LETを参照)

1つの式の中で関係演算子と算術演算子が同時に使われている場合、算術演算子が先に実行されます。

たとえば

$$X+Y < (T-1) / Z$$

この場合、X+Yの値が(T-1)÷Zの値より小さければ真となります。関係演算子の使用例としては、次のような例があります。

```
IF SIN(X) < 0 GOTO 1000  
IF 1-INT(I/J) < > 0 THEN K=K+1
```

(3)論理演算子

ビット操作や、論理演算をおこなったり、いくつもの関係を調べたりするため論理演算子を使用します。論理演算子はそのオペランドの値により、真または偽の値を結果として与えます。1つの式の中では、論理演算は算術および関係演算のあとで実行されます。論理演算の結果を次の表に示します。各演算子は演算の順位にしたがってリストされています。

NOT (否定)

X	NOT X
1	0
0	1

AND (論理積)

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

OR (論理和)

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

関係演算子がプログラムの流れを決定するのに使用できるのと同様に、論理演算子も2つまたは、それ以上の関係を結びつけて、判断のための真または偽の値を与えることができます(2-13 IFを参照)。

例) IF B > 200 AND F < 5 THEN 100
IF I < 10 OR J < 0 THEN 50
IF NOT P THEN 200 (Pが-1以外はすべてライン・ナンバー200へジャンプします)

論理演算子は、オペランドを-32768から、+32767までの16ビットの符号付き、2の補数表示の整数に変換してから演算をおこないます(もしオペランドがこの範囲になければエラーが発生します)。

指定された演算はこの整数に対し、ビットごとに対応させておこなわれます。つまり、結果の各ビットは2つのオペランドの対応するビットの状態によって決定されます。したがって、バイト単位のデータを、あるビットパターンに対して調べるのに論理演算を使うことができます。

たとえば、あるI/Oポートのステータスバイトのある1ビットだけをマスクするのに、AND演算子を使うことができます。またOR演算子は、2つのバイトデータを併合して1つの2進数を得ることができます。

次の例は、論理演算子の働きを理解するのに役に立つでしょう。

$$\begin{array}{r}
 63 \text{ AND } 9 = 9 \qquad 63 = 111111 \text{ (2進)} \\
 \qquad \qquad \qquad 9 = 001001 \text{ (2進)} \\
 \hline
 \text{ANDの結果 } 9 = 001001 \text{ (2進)}
 \end{array}$$

$$\begin{array}{r}
 15 \text{ AND } 6 = 6 \qquad 15 = 1111 \text{ (2進)} \\
 \qquad \qquad \qquad 6 = 0110 \text{ (2進)} \\
 \hline
 \text{ANDの結果 } 6 = 0110 \text{ (2進)}
 \end{array}$$

$$\begin{array}{r}
 -1 \text{ AND } 21 = 21 \qquad -1 = 1111111111111111 \text{ (2進)} \\
 \qquad \qquad \qquad 21 = 00000000000010101 \text{ (2進)} \\
 \hline
 \text{ANDの結果 } 21 = 00000000000010101 \text{ (2進)}
 \end{array}$$

$$\begin{array}{r}
 5 \text{ OR } 3 = 7 \qquad 5 = 101 \text{ (2進)} \\
 \qquad \qquad \qquad 3 = 010 \text{ (2進)} \\
 \hline
 \text{ORの結果 } 7 = 111 \text{ (2進)}
 \end{array}$$

$$\begin{array}{r}
 5 \text{ OR } 5 = 5 \qquad 5 = 0101 \text{ (2進)} \\
 \qquad \qquad \qquad 5 = 0101 \text{ (2進)} \\
 \hline
 \text{ORの結果 } 5 = 0101 \text{ (2進)}
 \end{array}$$

$$\begin{array}{r}
 -1 \text{ OR } -4 = -1 \qquad -1 = 1111111111111111 \text{ (2進)} \\
 \qquad \qquad \qquad -4 = 1111111111111100 \text{ (2進)} \\
 \hline
 \text{ORの結果 } -1 = 1111111111111111 \text{ (2進)}
 \end{array}$$

NOT X = -(X + 1) 任意の数の2の補数は各ビットを反転させたものに1を加えたものです。

(4)関数

関数は式の中で、あるオペランドに対して決められた演算操作を呼びだすのに使用されます。CBM BASICにはSQR(ルート)やSIN(サイン)などの組み込み関数が用意されています。組み込み関数については第3章で説明します。CBM BASICではまた、ユーザーにより、関数を定義することができます(2・6のDEF FNを参照)。

(5)文字列の演算

文字列は“+”によってつなぐことができます。たとえば

```

例) 10 A$="VIC":B$="1001"
    20 PRINT A$+B$
    30 PRINT "COMMODORE"+A$+B$
    RUN
    VIC1001
    COMMODOREVIC1001

```

また、数値演算に用いたのと同じ次の比較演算子によって比較することもできます。

= <> < > <= >=

文字列の比較は1度にそれぞれの文字列から1文字ずつ、そのASCII(アスキー)コードの比較をおこないます。もしすべてのASCII(アスキー)コードが等しければ、その2つの文字列は等しくなります。もしASCII(アスキー)コードが異なった場合は、小さいコードのものが数値的に小さいとされます。もし比較の途中で一方の文字列が終わりになった場合は、その短い文字列の方が小さいとされます。文字列の前と後ろのスペース(空白)も意味をもちます。

```

例) "AA"<"AB"
    "COMMODORE"="COMMODORE"
    "X$">"X#"
    "CL ">"CL"
    "VIC"<"VIC1001"
    A$=" 8 / 10 / 80" の場合
    A$<" 9 / 10 / 80"

```

第9節 スクリーン・エディター

第1部ハードウェア編を参照して下さい。

第10節 エラーメッセージ

BASICプログラムを実行中に何か処理を中断させるエラーが検出された場合、エラーメッセージが表示されます(付録Cにエラーメッセージの一覧表があります)。

第2章 CBM BASICの コマンドとステートメント

この章ではCBM BASICの持つすべてのコマンドとステートメントについて、以下の形式で説明します。

書式： 命令の正しい書式（フォーマット）を示します。使用される記号の意味については下記を参照して下さい。

目的： なんのためにその命令が使用されるのかを説明します。

説明： 命令の使用法を詳しく説明します。

例： 命令の使用法を示すプログラム例をあげます。

書式の記法

各コマンド・ステートメントの書式は、以下の記法にしたがっています。

1. アルファベットの大きい文字で記された項目は、そのままの形、または省略形で入力して下さい（省略形については付録Bを参照して下さい）。
2. 山カッコ（〈 〉）で、囲まれた項目はユーザーが指定する項目です。
3. かぎカッコ（〔 〕）で、囲まれた項目は、オプションです。
4. 山カッコおよびかぎカッコ以外のコンマ、丸カッコ、セミコロン、ハイフオン、等号などの記号は、示された位置に正しく入力して下さい。
5. 省略記号…の続く項目は、一行の許す長さ内で任意の回数繰り返すことができます。
6. たて線（|）で区切られている項目は、各々のうち1個を選択します。

CBM BASICコマンド・ステートメント一覧表

CLOSE	2. 1	LIST	2.17
CLR	2. 2	LOAD	2.18
CMD	2. 3	NEW	2.19
CONT	2. 4	ON~GOSUB, ON~GOTO	2.20
DATA	2. 5	OPEN	2.21
DEF FN	2. 6	POKE	2.22
DIM	2. 7	PRINT, PRINT #	2.23
END	2. 8	READ	2.24
FOR~NEXT	2. 9	REM	2.25
GET, GET #	2.10	RESTORE	2.26
GOSUB~RETURN	2.11	RUN	2.27
GOTO	2.12	SAVE	2.28
IF~THEN, IF~GOTO	2.13	STOP	2.29
INPUT	2.14	SYS	2.30
INPUT #	2.15	VERIFY	2.31
LET	2.16	WAIT	2.32

2.1 CLOSE

書式： CLOSE〈ロジカルファイル番号〉

目的： ファイルに対する入出力を終了する。

説明： 〈ロジカルファイル番号〉は、あるファイルがオープンされた時に使われた番号です。CLOSEが実行されますと、特定のファイルとファイル番号の間のつながりはなくなります。そしてそのファイルは、異なったファイル番号ないしは同じ番号で再度オープンすることができます。同様に、一度使用されたファイル番号を他のファイルに用いることもできます。

シーケンシャル出力ファイル作成時にCLOSEを実行しますと、最後の出力バッファを書きこみます。

例： OPEN 4, 4
 PRINT # 4, "THIS GOES TO PRINTER"
 CLOSE 4

2.2 CLR

書式： CLR

目的： すべての数値変数を0に、文字変数をヌルコード((00)₁₆)にセットし、メモリーのエンド、スタック、配列をリセットします。

説明： CLRがBASICプログラム中で実行された後、次に述べる以外はプログラムを続行することができます。

- エラーになり続行できない場合
- スタックを使用する処理を実行しようとした場合 (GOSUB~RETURN, FOR~NEXTなど)
- 配列を使用する場合、つまりDIMで配列を宣言した後、CLRを実行すると、その後では配列は宣言されていないものとして扱われます。

例： 10 X=25
20 CLR
30 PRINT X
RUN
0

2.3 CMD

書式： CMD <ロジカルファイル番号> [, 印字する数値または文字]

目的： シリアルバスと機器をつなぎ、その機器をリスナー状態にします。

説明： CMDはPRINT# (2.23参照)と同じパラメータを用います。

例： REM CREATE HARDCOPY LISTING
OPEN 4, 4
CMD 4, "PROGRAM LISTING"
LIST
PRINT# 4, "TURNS OFF CMD"
CLOSE 4

2.4 CONT

書式： CONT

目的： ストップキー入力後、またはSTOP (2.29参照) やEND (2.8参照)を実行した後に、プログラムを再実行する。

説明： 処理を中止した個所から実行が継続されます。INPUT (2.14参照) ステートメントからプロンプト出力後に処理を中止した場合、CONTを実行すると、再びプロンプトを出力するところから実行を開始します。通常 CONTは、デバッグのためにSTOPと共に用いられます。プログラムの実行を途中で中止し、ダイレクトモードステートメントにより、変数などの中間結果を調べたり変更することができます。そして、その後、CONTやGOTO (2.12参照) を用いて処理を再開することができます。GOTOの場合は、処理再開するスタートのプログラム・ライン・ナンバーを指定することができます。

実行の中断中に、プログラムが変更ないし編集された場合は、CONTは無効になります。また中断中、ダイレクトモード実行中にエラーが発生した場合は処理を続行することができません。

プログラムの実行がエラーにより中断された場合はCONTにより実行を再開することはできません。

例： 2.29 STÖPの項を参照

2.5 DATA

書式： DATA<定数>[, <定数>…]

目的： プログラム中のREAD (2.24参照) により読まれる数値および文字定数を格納する。

説明： DATAステートメントは、非実行ステートメントで、プログラム中どこに置いてかまいません。DATAステートメントは、(コンマに区切られて) 1行中にはいるだけの定数をいれることができ、またプログラム中にいくつでもDATAステートメントを使うことができます。READステートメントは、ライン・ナンバーの順にDATAステートメントを読んでゆきます。そしてそこに含まれているデータはその行にいくつデータがあるのか、またはその行がプログラム中のどこに置かれているかにはかわらず、1つの連続したデータの一部分と考えることができます。

<定数>は任意の形式の数値定数、すなわち固定小数点、浮動小数点、または整数を含むことができます(数式は許されません)。DATAステートメントに含まれる文字定数は、その中にコンマ(,)、コロン(:)または前後に意味のある空白(スペース)を含む場合にはクォーテーション(")で定数を囲んで下さい。それ以外の場合には、その必要はありません。

READステートメントにより、指定される変数の型（数値または文字）は対応するDATA文の定数と一致しなくてはなりません。

DATAステートメントは、RESTORE (2.26参照) により最初から読むことができます。

例 : 2.24 READの項を参照して下さい。

2.6 DEF FN

書式 : DEF FN〈名前〉[(〈パラメータ〉)] = 〈関数の定義〉

目的 : ユーザーによって書かれた関数を定義し名前を付ける。

説明 : 〈名前〉は正しい変数名でなければなりません。FNに引き続く〈名前〉が関数の名前になります。〈パラメータ〉は、その関数が呼ばれた時に置き換えられる、関数の定義式の中の変数名です。〈関数の定義〉は、その関数の演算内容を記述する式で、1行の範囲にかぎられます。この式にあらわれる変数名は、単に関数を定義するだけで、同じ名前をもつプログラム中の変数には何の影響も及ぼしません。また関数の定義式に使われた変数名はパラメータ中にあってもなくてもかまいません。もしあれば、そのパラメータの値は関数が呼ばれた時に与えられます。そうでなければ、その変数の現在の値が使われます。

このユーザ定義関数は、数値関数のみであって文字関数は許されていません。関数名によって型が指定されたなら、式の値は、その型として扱われます。もし関数名により指定された型と引数（アーギュメント）の型が一致しない場合は、"TYPE MISMATCH" エラーが発生します。

DEF FNステートメントはそれが定義する関数が呼ばれる前に実行されなければなりません。もし定義される前に関数が呼ばれた場合は、"UNDEF'D FUNCTION" エラーが発生します。

なお、DEF FNはダイレクトモードでは使用できません。

```
例 : 10 DEF FNAB(X)=X↑3/Y↑2
      20 I=2:Y=3
      30 T=FNAB(I)
      40 PRINT T
      RUN
      .888888889
```

ライン・ナンバー10で関数FNABを定義します。

ライン・ナンバー30では、10で定義した関数が呼ばれます。

2.7 DIM

書式： DIM(変数名)(添字の最大値)

目的： 配列変数の添字の最大値を指定し、同時にメモリー領域を割り当てる。

説明： DIMステートメントなしに配列変数名を用いた場合、その添字の最大値は、10と見なされます。もし指定された値より大きい添字が使われた場合は、`BAD SUBSCRIPT` エラーが発生します。添字の最小値はつねに0です。

DIMステートメントは指定された配列のすべての要素の値を初期値0に設定します。

1次元以上、255次元までの配列を宣言することができます。添字の最大値は32767まで使用することができます。配列全体の大きさは、使用できる、メモリーの容量により制限されます。

例： 10 DIM A(20)
20 FOR I=0TO20
30 READA(I)
40 NEXT
50 DATA1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21

多次元配列の場合

10 DIM R3(5,5)
10 DIM D\$(2,3,2)

2.8 END

書式： END

目的： プログラムの実行を終了させ、すべてのファイルを閉じ（クローズ）、その後、コマンドレベルに戻る。

説明： ENDステートメントはプログラムの実行を終了させるために、プログラム中のどこに置いておかまいません。

STOP(2.29参照)とは異なり、ENDステートメントではBREAKメッセージは出力されません。また、プログラムが一番最後のENDはなくてもかまいません。ENDステートメントを実行するとBASICはつねにコマンドレベルに戻ります。

例： 100 IF K>1000 THEN END

2.9 FOR~NEXT

書式： FOR〈変数名〉=〈X〉TO〈Y〉[STEP〈Z〉]
 :
 NEXT〔〈変数名〉〕[,〈変数名〉…]
ただしX, YおよびZは数値式

目的： 一連の命令を指定した回数分、繰り返し実行する。

説明： 〈変数名〉で指定した変数は繰り返しの回数をかぞえるためのカウンターとして使用されます。最初の数値式〈X〉は、カウンターの初期値、二番目の数値式〈Y〉はカウンターの最終値を設定します。次にFORステートメントからNEXTステートメントまでのプログラム文が実行されます。そしてNEXTをみつけた時点でカウンターにはSTEPによって指定された値だけ加算されます。もしSTEPが省略された場合は、+1加算されます。そしてカウンターの値が最終値より大きいかどうか調べられ、もし大きくないならBASICは、FORの次のステートメントに戻ります。もし大きいならば、NEXTに続くプログラム文が実行されます。

STEPで指定する値が負の場合は、カウンターの初期値は最終値より大きい値を設定して下さい。この場合カウンターの値が初期値より小さくなるまで、FORステートメントからNEXTステートメントまでのプログラム文の実行が繰り返されます。

多重ループについて（ネスティング）

FOR...NEXTループは入れ子構造（ネスト）にすることができます。つまり、FOR...NEXTループの中に、他のFOR...NEXTループを置くことができます。ループが多重になった場合は、それぞれのループのカウンターには、各々異なった独自の変数名を使用しなくてはなりません。内側のループのNEXTステートメントは、外側のループのものより以前にあらわれなければなりません。多重ループが同じ場所で終わる場合は、それら全部のループに対して、1つのNEXTステートメントに、カウンター変数をループの順番に、コンマで区切って書くことで、対応することができます。NEXTステートメントが、最近のFORステートメントに対応する場合には、NEXTの変数名は省略することができます。もし対応するFORステートメントのないNEXTがあらわれたなら“NEXT WITHOUT FOR”エラーが発生し、プログラムの実行は停止します。

スタックの容量に制限があるため、多重ループで使用できるFOR...NEXTの数は最大9個までです。

例 1 : 多重ループ

```
10 FOR I=1 TO 3
20 FOR J=1 TO 3
30 PRINT I;J
40 NEXT J,I
```

RUN

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```

例 2 : ループカウンタ設定後、使用した変数名に別の値を代入

```
10 K=10
20 FOR I=1 TO K STEP 2(カウンタは1~10に設定)
30 PRINT I;
40 K=K+10
50 PRINT K
60 NEXT
```

RUN

```
1 20
3 30
5 40
7 50
9 60
```

例 3 : カウンタの最終値が初期値より小さい場合

```
10 J=0
20 FOR I=1 TO J
30 PRINT I
40 NEXT I
```

RUN

```
1
```

この場合、カウンタが、最終値 (J) より大きいということは、NEXT が実行された段階で、はじめてわかるため、ループが1回だけ実行されます。

```

例 4 : 10 FOR I=1 TO 1+5
        20 PRINT I;
        30 NEXT
        RUN
         1   2   3   4   5   6

```

この場合、ループ回数は6回です。ループカウンターの初期値が最終値より先に変数名にセットされます。

注 : 整数型変数 (I%, A%のように変数名の後に%をつけてあらわします) を、ループのカウンター変数として使用することはできません。

```

10 FOR I%=1 TO 10
20 PRINT I%
30 NEXT I%
RUN
?SYNTAX ERROR IN 10

```

2.10 GET, GET#

書式 : GET{# (ロジカルファイル番号),}(変数名)

目的 : ファイルから1文字読みこんで、変数に代入する。

説明 : ロジカルファイル番号をとみなわないGETは、キーボードバッファから、数字でも文字でも、1文字だけをバッファ内から入力します。もしバッファ内に何もなければ、ヌルコード (= (00) 16)、値は0を入力します。

GET#は、ロジカルファイル番号により指定されたファイルより1文字入力します。ただし、OPENステートメント (2.21参照) により指定されたデバイス番号 (機器番号) が0の場合は、前述のGETと同じ動作をします。デバイス番号が1 (=カセット) の場合E0F (=エンドオブファイル) のキャリッジリターン (= (0D) 16) を入力したかどうか、ST (ステータス) を調べることによって検出することができます。

```

例 : 10 PRINT "HANGS UNTIL KEY PRESSED"
      20 GET A$
      30 IF A$="" THEN 20
      40 PRINT "OK"

```

2.11 GOSUB~RETURN

書式： GOSUB〈ライン・ナンバー〉
 :
 RETURN

目的： サブルーチンへの分岐および戻り。

説明： 〈ライン・ナンバー〉は、サブルーチンの最初の行のライン・ナンバーです。

サブルーチンのRETURNを実行すると、BASICは最近のGOSUBステートメントの次のステートメントに戻ります。1つのサブルーチン中にRETURNがいくつあってもかまいませんが、プログラムのロジックは、サブルーチン中の正しいRETURNステートメントを指示するようになっていなくてはなりません。サブルーチンは、プログラムのどこに置かれてもかまいませんが、メインプログラムと、一目で区別できるように書かれることをおすすめします。また不用意にサブルーチンに飛びこんでしまうことを防止するためには、STOP、ENDないしはサブルーチンを飛びこすよう指示するGOTOステートメントをサブルーチンの前に置くようにして下さい。

サブルーチンを、プログラムの先頭にもってきて、小さいライン・ナンバーを与えることにより、実行速度をたかめることができます。

スタックの容量に制限があるため、GOSUB~RETURNをネスティング(2.9 FOR~NEXT参照)で使用した場合、ネスティングできるのは最大23個です。

例： 10 GOSUB 40
 20 PRINT"END"
 30 END
 40 PRINT"SUBROUTINE";
 50 PRINT"SUBROUTINE";
 60 PRINT"SUBROUTINE"
 70 RETURN
 RUN
 SUBROUTINE SUBROUTINE SUBROUTINE
 END

2.12 GOTO

書式： GOTO〈ライン・ナンバー〉

目的： 無条件に、通常のプログラムの流れから指定された行へ分岐する。

説明： 指定された〈ライン・ナンバー〉の行にあるステートメントが、実行文であれば、その部分およびそれに引き続くステートメントを実行します。もし非実行文であれば、〈ライン・ナンバー〉の行の後の最初の実行文より実行が継続されます。

例： 10 READ R
20 PRINT "R=";R,
30 A=3.14*R↑2
40 PRINT "AREA=";A
50 GOTO10
60 DATA5,1,15
RUN
R= 5 AREA= 78.5
R= 1 AREA= 3.14
R= 15 AREA= 706.5
?OUT OF DATA
ERROR IN 10

2.13 IF~THEN, IF~GOTO

書式： IF〈論理式〉THEN〈ステートメント〉|〈ライン・ナンバー〉

IF〈論理式〉GOTO〈ライン・ナンバー〉

目的： 〈論理式〉によって記述される条件にしたがって、プログラムの実行の流れに関する判断をおこなう。

説明： 〈論理式〉の結果が真(≠0)ならばTHENまたはGOTO文が実行されます。THENには、分岐するライン・ナンバーまたは実行するステートメントのどちらも続けることができます。

もし〈論理式〉の結果が、偽(=0)ならば、THENまたはGOTO文は無視され、次の行のステートメントの実行がおこなわれます。

IFステートメントの多重化(ネスティング)

IF~THENを多重(ネスト)にすることができます。その限度は行の長

さによって制限されます。

```
IF A=B THEN IF B=C THEN PRINT "A=C"
```

IF~THEN〈ライン・ナンバー〉を実行した場合、指定されたライン・ナンバーの文がなければ、'UNDEF'D STAEMENT' エラーが発生します。

```
例 1 : 10 I=5  
20 IF I THEN GET I:PRINT I  
30 GOTO10
```

この場合、ライン・ナンバー20では、もし $I \neq 0$ ならば、キーボードから1文字読みこもうとします（数字以外をキーボードから入力するとSYNTAX ERRORになります）。

```
例 2 : 10 IF (I>10) AND (I<20) THEN DB=1980 : GOTO300  
20 PRINT "OUT OF RANGE"  
      :  
300 END
```

この例では、もしIが10より大きくかつ20より小さい場合、DBの値が計算され、ライン・ナンバー300へ分岐します。それ以外は、ライン・ナンバー20から以降へと実行を続けます。

注 : 整数以外の数はその内部表現（2進法）が必ずしも正確でないため（たとえば、.2は正確に表わすことはできません）、整数でない数を調べる場合はある範囲をとって調べることを、おすすめします。たとえば、変数Aの計算結果が1.0になった場合、1かどうかを調べる確かな方法は

```
IF ABS (A-1.0) <=1.0E-6 THEN...
```

です。もしこの結果が真なら、 10^{-6} 分の1の精度でAは1に等しいことになります。

2.14 INPUT

書式： INPUT ["<プロンプト>";]<変数名>

目的： プログラム実行時にキーボードから入力することを可能にする。

説明： INPUTステートメントが実行されると、プログラムの実行は停止し、データの入力を待っている疑問符(?)がプリントされ、カーソルが点滅します。もし"<プロンプト>"がある場合は、疑問符の前にそれがプリントされます。そこで、キーボードからデータを入力して下さい。入力されたデータは、<変数名>で与えられた変数に代入されます。入力データの数はリスト中の変数名の数と一致してはなりません。各データの項目はコンマで区切ります。もしINPUTステートメントで指定した個数よりも少ない数のデータしか入力されなかった場合には、さらに2つの疑問符を表示してもっとデータの必要なことを示します。また多すぎる個数のデータが入力された場合、またデータとしてコンマ(,)、コロン(:)を入力した時、"?EXTRA IGNORED"というメッセージをプリントし、余分なデータ、コンマおよびコロンは無視して、プログラムの実行を継続します。

<変数名>は、(添字付きを含み)数値変数名でも文字数名でもかまいませんが、入力される各データは、変数名で指定した型に一致してはなりません。INPUTステートメントに対する文字(ストリング)の入力は、クォーテーション(")で囲む必要はありません。

一度に入力できるデータの長さは、最大88文字までです。しかし、プロンプト(?+1スペース)の分が最低2文字必要なので実際は86文字です(リターンキーを押すと、それも1文字分にかぞえます)。

次にもし入力されたデータの型が、対応する変数名のものと一致しない場合は、"?REDO FROM START"のメッセージがプリントされ、もう一度繰り返しデータの入力を求め、まちがって入力した値は無視されます。

また、キーボードからリターンキーだけを押した場合、<変数名>の値は、INPUTを実行する前に、代入された値がそのまま残っています。

例1： 10 INPUT X
20 PRINT X"ジジョウ ハ"X↑2
30 END
READY.
RUN
? 5 (画面に?があらわれ、カーソルが点滅したらキーボードより5を入力して下さい)
5 ジジョウ ハ 25

```
例 2 : 10 PI=3.14159265
        20 INPUT"ハンケイ";R
        30 A=PI*R↑2
        40 PRINT"メンセキ";A
        50 PRINT
        60 GOTO20
```

RUN

ハンケイ?7.4 (キーボードより7.4を入力して下さい)

メンセキ 172.033614

ハンケイ?

:

```
例 3 : 10 INPUT A, B$, C
```

2 15 INPUT #

書式: INPUT #〈ロジカルファイル番号〉,〈変数名〉

目的: ファイルからデータを読みこみ、プログラムの変数に代入する。

説明: 〈ロジカルファイル番号〉は、データ入力のために、OPEN (2.21参照)によりオープンされたファイルに使われた番号を用います。〈変数名〉は、ファイルのデータを割りあてる変数名です。入力するデータの型は、〈変数名〉により指定された型と一致しなくてはなりません。INPUT#ステートメントでは、INPUTのように疑問符がプリントされることはありません。

ファイル中のデータは、INPUT文に対して入力するデータと同じ型になっていなくてはなりません。数値の場合は、先に続く空白(スペース)、キャリッジリターン、ラインフィードは無視されます。つまり、空白、キャリッジリターン、ラインフィード以外の最初の文字が、数値データの始まりとされるわけです。数値は空白、キャリッジリターン、ラインフィード、またはコンマによって区切られます。型が異なっていた場合は、"? REDO FROM START"のかわりに"? FILE DATA ERROR"がプリントされます。EOIまたはタイムアウトによりINPUT#は1個のデータの入力を終わり、次へすすみます。

INPUT#で1回で入力できるデータの最大の長さは88文字です。

例: 2.21 OPENの項を参照して下さい。

2.16 LET

書式： [LET]〈変数名〉=〈式〉

目的： 〈式〉であらわされる値を変数に代入する。

説明： LETはなくてもかまいません。つまり、式の値を変数名に代入するには、等号だけでよいのです。

例： 10 LET A=23
20 LET B=23↑2
30 LET C=23↑4
40 LET SM=A+B+C

は

10 A=23
20 B=23↑2
30 C=23↑4
40 SM=D+E+F

としても同じです。

2.17 LIST

書式1： LIST [〈ライン・ナンバー〉]

書式2： LIST [〈ライン・ナンバー〉]—[〈ライン・ナンバー〉]

目的： メモリー内にあるプログラムの全部、または一部のリストを出力機器（たとえば、ディスプレイ、プリンターなど）に出力する。

説明： 書式1)

1. 〈ライン・ナンバー〉が指定されない場合は、プログラムの全部のリストを出力します。
2. 〈ライン・ナンバー〉が指定されている場合、その番号に該当する行のリストだけを出力します。

書式2)

1. 最初の〈ライン・ナンバー〉だけが指定された場合、その行から最後までリストが出力されます。
2. 2番目の〈ライン・ナンバー〉だけが指定された場合、プログラ

ムの先頭から指定された行までのリストが出力されます。

3. 〈ライン・ナンバー〉が両方共指定されている場合は、指定された範囲のリストが出力されます。

LISTコマンドを実行し終わると、BASICは、つねにコマンドレベルに戻ります。

例	:	書式 1)	LIST	プログラム全部のリスト
			LIST500	ライン・ナンバー500のリスト
		書式 2)	LIST50-	ライン・ナンバー50から最後までのリスト
			LIST-100	最初からライン・ナンバー100までのリスト
			LIST50-100	ライン・ナンバー50から100までのリスト

2.18 LOAD

書式: LOAD〈"ファイル名"〉[, 〈デバイス番号〉]

目的 外部記憶装置からメモリーにプログラムを読みこむ。

説明: 〈ファイル名〉は、プログラムが、SAVE (2.28参照) された時に、使用された名前です。

デバイス番号を指定しない場合は、自動的に1 (=カセット) が割りあてられます。

LOADコマンドは〈ファイル名〉で指定したプログラムを読みこむまえにメモリー内に現在あるすべての変数、プログラムを消去し、かつオープンされていたファイルをすべてクローズします。

もしLOADコマンドをプログラム中で実行した場合はオープンされていたファイルはそのままオープンされたままで、新しいプログラムを読みこみ (=LOAD)、実行開始 (=RUN) します。

このようにしてLOADコマンドはいくつかのプログラム (または同じプログラムのいくつかのセグメント) を連結して実行するのに使えます。また、この場合、各プログラム間 (ないしはセグメント間) で変数は消去されませんので、データの受け渡しをおこなうことができます。

〈ファイル名〉および〈デバイス番号〉の指定がない場合は、カセットから1番最初に見つけたプログラムファイルを読みこみます。

例 : LOAD "BASIC BASIC"

この場合、デバイス番号=1 (カセット) の機器から`BASIC BASIC`
という名前のプログラムファイルをメモリーに読みこみます。

2.19 NEW

書式 : NEW

目的 : 現在メモリー内にあるプログラム、およびすべての変数を消去する。

説明 : 新しいプログラムを入力する前に、メモリーをクリアする時に、コマンドレベルでNEWを入力します。

BASICは、つねにNEWを実行した後、コマンドレベルに戻ります。

2.20 ON~GOSUB, ON~GOTO

書式 : ON<式> GOSUB<ライン・ナンバー> [, <ライン・ナンバー>...]

 ON<式> GOTO<ライン・ナンバー> [, <ライン・ナンバー>...]

目的 : <式>の評価により得られた値に基づき、指定されたいくつかのライン・ナンバーの該当する行の1つに分岐する。

説明 : <式>の値が、どの行に分岐するかを決定します。たとえば<式>の値が3であったなら、3番目に書かれた行が、分岐先となります (もし値が整数にならない時は、小数部分は切り捨てられます)。

ON~GOSUBステートメントでは、各ライン・ナンバーは、サブルーチンの最初の行のライン・ナンバーでなければなりません。

もし<式>の値が負になった場合、`ILLEGAL QUANTITY`エラーが発生します。またもし、値が、0または、ライン・ナンバーの個数より大きい場合には、次の行へプログラムの制御が移り、エラーは起こりません。

例 : 10 ON A-1 GOTO 20, 30, 40, 50

この場合、

A=4ならば、プログラムの制御は、ライン・ナンバー40に移ります。

A=6ならば、制御は、ライン・ナンバー10の次の行に移ります。

21 OPEN

書式： OPEN<ロジカルファイル番号>[,<デバイス番号>[,<セカンダリィアドレス>[, " <ファイル名>"]]]

目的： I/Oチャンネルとシリアルバス、または内部デバイス間の連絡を確立する。

説明： ロジカルファイル番号は次の範囲内で指定しなくてはなりません。

1 ≤ロジカルファイル番号 ≤255

デバイス番号を指定しない場合は、自動的に1 (=カセット)が割りあてられます。セカンダリィアドレスおよびファイル名を指定しない場合は、ないものとみなされます。シリアルバスを使用した場合は、GET #、INPUT #、およびPRINT #が送られるのといっしょに必ずセカンダリィアドレスと、デバイス番号も送られます。

ファイルのタイプは、もし`S` (シーケンシャル) が指定されなければ、プログラムファイルになります。また、シーケンシャルファイルの場合 (`S` で指定) `W` (ライト) が指定されなければ、読みこみ用のREAD (リード) ファイルとして扱われます。

ファイルは、カセット (デバイス番号=1)、プリンター (デバイス番号=4)、ディスプレイ (デバイス番号=3)、およびキーボード (デバイス番号=0) に対してオープンすることができます。

カセットテープを使用した場合

セカンダリィアドレスは

- 0 カセットテープからファイルを入力します (読みこみ)。
- 1 カセットテープにファイルを出力します (書きこみ)。
- 2 カセットテープにファイルを出力します (書きこみ。ただし、ファイルの最後にエンド `オブ` テープ=EOTを書きこみません)。

です

例： 10 OPEN2,1,1,"DATAFILE"
20 FOR I=1 TO10
30 PRINT #2, CHR\$(I+48)
40 NEXT
50 CLOSE 2
60 PRINT "REWIND TAPE"
70 PRINT "PRESS ANY KEY TO READ"
80 GET O\$:IF O\$=" "THEN80
90 OPEN2,1,0"DATAFILE"

```
100 FOR I= 1 TO10
110 INPUT #2, A$
120 PRINT VAL(A$)
130 NEXT
140 CLOSE 2
```

2.22 POKE

書式： POKE I, J ($0 \leq I \leq 65535, 0 \leq J \leq 255$)
ただし I および J は整数で表記

目的： メモリーの指定した番地(= I)に1バイトのデータ(= J)を書きこむ。

説明： 整数 I は書きこむメモリーの番地です。J は書きこむデータで0から255の範囲でなくてはなりません。また I は0から65535の範囲です。

POKEと逆の働きをする関数にPEEKがあります。PEEKの引数は、1バイトのデータを読みだしたいメモリーのアドレスです (3.13参照)。

POKEとPEEKは、効率的なデータの格納、機械語サブルーチンの書きこみや、機械語サブルーチンとのデータや計算結果などの受け渡しなどに役立ちます。

例： 10 POKE 4096,1

この場合4096番地に1が書きこまれます。

2.23 PRINT, PRINT

書式： PRINT[#<ロジカルファイル番号>],[<式>,...]

目的： データをディスプレイ、または指定したチャンネルに出力する。

説明： <式>が省略された場合、1行ブランク(空白)が出力されます。<式>がある場合、数値や文字(ストリング)が出力されます。文字の場合はクォーテーション(")で囲む必要があります。

プリント位置

句読点で各項目を区切ることにより、項目を区切って出力することができます。つまりBASICでは、88文字(画面での4行分になります)を8個のゾーンに区切り、各項目をコンマ(,)で区切った場合、次のゾーン

の先頭から次の値が出力されます。セミコロン(;)で区切った場合は、前の値に続いて次の値が出力されます。項目間にスペースを1個以上いれても同様の結果が得られます。もし、〈式〉の最後が、コンマやセミコロンで終わった場合、次のPRINTステートメントは、ひきつづき同じ行に出力されます。

コンマやセミコロンがない場合は、行の最後に自動的にキャリッジリターンがつけられ改行します。もし画面への出力が22文字をこえた場合、自動的に次の行へ出力されます。

数値がプリントされると、そのうしろにスペースが1個挿入されます。また数値の場合、正の数は、数値の前にスペースが1個、負の場合はマイナス符号がつけられます。数値は有効桁数の範囲内であれば、そのまま、そうでなければ、指数形式で表示されます。ある数値Xが $0 > X < .01$ であるならば、Xの型には関係なく、指数形式で表示されます。

キャリッジリターン(= (0D) 16)は、各PRINT#ステートメントの最後に自動的につけられます。しかし、〈式〉の最後にセミコロン(;)をつけることによって避けることができます。

```
例 1 : 10 X = 3
        20 PRINT X+7, X-7, X*(-7), X↑7
        30 END
        RUN
          10           -4
        -21           2187
```

この場合、出力する各項目をコンマで区切っているため、各項目は、それぞれのゾーンの先頭から出力されます。

```
例 2 : 10 INPUT X
        20 PRINT X "ジジョウ="X↑2
        30 PRINT X "サンジョウ="X↑3
        40 PRINT
        50 GOTO 10
        RUN
        ? 9      (9を入力します)
        9 ジジョウ= 81.0000001
        9 サンジョウ= 729.000001
```

```

? 8      (8を入力します)
8 ジジョウ= 64
8 サンジョウ= 512

```

この場合、ライン・ナンバー20の最後はセミコロンをつけることにより、次のPRINTステートメントが、同じ行に出力されます。ライン・ナンバー40では、1行、ブランクを出力します。

(注) 9の2乗、3乗を計算した結果が、机上での計算結果と異なっています。これはVICがもっているアルゴリズムの結果の誤差です。たとえば、9の2乗(9↑2)を計算する場合、次のようになります。

```

2 loge 9 = xとすると
9 ↑ 2 は exでもとめられます。
∴ ex = 81.0000001
(VICでは e = 2.71828183, 有効桁数は9桁のため以上の結果がえられます。)
```

```

例3 : 10 FOR X=1 TO 5
      20 J=J+2
      30 K=K+4
      40 ? J;K;
      50 NEXT X
      RUN
           2   4   4   8   6   12   8
           16  10  20

```

この場合、セミコロンにより、数値は、前の値にひき続いて(ただし、符号分としての1スペースおよびおわりの1スペースが必ず出力されますが)出力されます。ライン・ナンバー40では?がPRINTのかわりに使われています。ただし、LISTで、プログラムのリストをとりなおしますと、今度はPRINTと表示されます。

2.24 READ

書式: READ<変数名>[,<変数名>...]

目的: DATAステートメントよりデータを読み、変数に割りあてる(2.5参照)。

説明: READステートメントは、必ずDATAステートメントと組み合わせて使

わなければいけません。READステートメントはDATAステートメントのデータを1対1対応の方法で変数に割りあててゆきます。READステートメントの変数は、数値変数でも、文字変数でもかまいません。しかし読み出された値と変数の型は一致していなければなりません。もし一致していない場合は、"? SYNTAX" エラーが発生します。

1つのREADステートメントが、1つ以上のDATA文を参照したり（複数の場合は順番に参照されます）、またいくつかのREADステートメントが1つのDATAステートメントを参照することができます。もし、〈変数名〉の個数がDATAステートメントの個数を越えてしまった場合は、"OUT OF DATA" エラーが発生します。

指定された変数の数が、DATAステートメントのデータの個数よりも少ない場合には、続いてある次のREADステートメントが続きのデータから読み始めます。もしREADステートメントが、それ以上なければ、残りのデータは無視されます。

DATAステートメントの先頭からもう一度読みなおす場合はRESTORE (2.26参照) を使用して下さい。

```
例 1 : 10 FOR I=1 TO 10
        20 READ A (I)
        30 NEXT
        40 DATA 7.06, 4.19, 3, 3.98, 1
        50 DATA 2, 3, 5.09, 7, 1, 8, 4
           :
```

この場合、DATAステートメントからデータを読み、配列Aに代入します。実行するとA(1)は7.06、A(2)は4.19と、各要素に順番に割りあてられます。

```
例 2 : 10 PRINT " CITY                ZIP"
        20 READ C$, Z
        30 DATA "SANTA CLARA", 95050
        40 PRINT C$; Z
        RUN
        CITY                ZIP
        SANTA CLARA        95050
```

この場合、文字と数値データを読みこんでいます。

2.25 REM

書式： REM〈コメント〉

目的： プログラム中に注釈、コメントを入れる。

説明： REMステートメントは実行はされませんが、プログラムのリストを取ると入力した内容（カタカナのコメントは"で囲みます）がそのままリストされます。REMステートメントの行へGOTOやGOSUBにより分岐することができます。この場合REMステートメントの後にくる最初の実行文から実行を開始します。

例： 100 REM CBM BASIC MANUAL
110 REM "コモドール ジャパン"

2.26 RESTORE

書式： RESTORE

目的： DATAステートメントを最初から読めるようにする。

例： 10 READ A, B, C
20 RESTORE
30 READ D, E, F
40 DATA 57, 80, 91
：

この場合実行すると、変数AとDには57が代入されます。

2.27 RUN

書式： RUN〔〈ライン・ナンバー〉〕

目的： 現在メモリー内にあるプログラムの実行を開始する。

説明： 〈ライン・ナンバー〉を指定すると、その行から実行がはじまります。指定のない場合には、最も小さいライン・ナンバーから実行がはじまります。

次の場合に、プログラムは実行を終了し、BASICはいつでもコマンドモードに戻ります。

- ①実行するライン・ナンバーが、もう、ない場合
- ②ENDまたはSTOPステートメントが実行された場合
- ③プログラム実行中に、何かエラーが発生した場合

2.28 SAVE

書式: SAVE ["<ファイル名>" [, <デバイス番号>] [, <コマンド>]]

目的: BASICプログラムファイルに書きこまれる(セーブする)。

説明: デバイス番号を指定しない場合は、自動的に1 (=カセット)が割りあてられます。
コマンドが∅の場合……プログラムのセーブ終了後テープの最後にエンドオブテープ (EOT) が書かれません。
コマンドが∅以外の場合……セーブ終了後、エンドオブテープ (EOT) が書かれます。
<ファイル名>を指定しない場合は、自動的にヌルコード(=(00)₁₆) が割りあてられます。

例: SAVE
SAVE "COMMODORE"
SAVE (A\$)

2.29 STOP

書式: STOP

目的: プログラムの実行を停止してコマンドレベルに戻る。

説明: STOPステートメントはプログラムの実行を停止するためにプログラムのどこで使用してもかまいません。
STOPを実行すると次のようなメッセージが出力され実行を停止します。
BREAK IN nnnnn
 ライン・ナンバー

ENDステートメントとちがい、STOPステートメントは、ファイルをクローズしません。

STOPが実行されるとBASICは必ずコマンドレベルに戻ります。またCONTコマンド (2.4参照) により実行は再開されます。

例: 10 INPUT A, B, C
20 K=(A+3)/2: L=B*3
30 STOP
40 M=C*K+100: PRINT M
RUN

```

? 1, 2, 3      (キーボードから入力します)
BREAK IN 30
READY.
PRINT L      (ダイレクトモードで入力します)
  6
READY.
CONT      (ダイレクトモードで入力します)
  106
READY.

```

2.30 SYS

書式： SYS<変数名>[[<引数>, ...]]

目的： 機械語サブルーチンを呼び出します(3.27 USR関数も参照して下さい)。

説明： <変数名>は、機械語ルーチンのメモリー上でのスタート番地です。<変数名>として配列変数名は使用できません。
<引数>は、機械語のサブルーチンへ受け渡す引数です。

例： 100 JA=30200
110 SYSJA

2.31 VERIFY

書式： VERIFY["<ファイル名>"[, <デバイス番号>]]

目的： メモリー内のプログラムとテープ上のプログラムファイルとの内容を比較してその違いを教える。

説明： デバイス番号が指定されない場合は、自動的に1(=カセット)が割り当てられます。<ファイル名>を指定しない場合は、ヌルコード(=(00)₁₆)が割り当てられます。カセットを使用する時のみファイル名がなくてもかまいません。

例： VERIFY "KFILE"
PRESS PLAY ON TAPE # 1
OK
VERIFYING
VERIFY ERROR
READY.

232 WAIT

書式： WAIT〈番地〉,I[,J]

目的： コンピューターの入力ポートをモニター(監視)する間、プログラムの実行を停止する。

説明： WAITステートメントを実行すると、指定した番地(アドレス)のビットパターンが指定した状態になるまでプログラムの実行が中断されます。指定した番地から読みこんだデータと整数表現JとのEOR(排他的論理和)の結果とIのAND(論理積)が取られます。もしその結果が0なら、BASICは、もう一度指定した番地の状態を読み込み同じ操作を繰り返します。もし結果が0でなければ、プログラムの実行は次の文に移ります。Jが省略された場合は、0とみなします。

例： 10 WAIT37151,64,64

この場合、カセットのキーを押すまで、プログラムの実行を中断しています。

注： WAITステートメントの実行により、無限ループにはいつてしまう場合があります。その場合には、コンピューターをリセット(例、電源OFF)しなければなりません。

第3章 CBM BASICの関数

この章では、CBM BASICの持っている組み込み関数について、以下の形式で説明します。これらの関数は、どのプログラムでも、何の定義の必要もなく使用することができます。

書式：関数の正しい書式（フォーマット）を示します。関数に渡す引数（アーギュメント）については、下記を参照して下さい。関数にも省略形が使えます（付録Bを参照して下さい）。

機能：その関数の機能を示します。

例：その関数の使用法を示すプログラム例をあげます。

関数に渡す引数（アーギュメント）は、つねにカッコ（ ） で囲みます。この章での関数の書式で、引数は次のように表現します。

X , Y 任意の数式をあらわします。

I , J 任意の整数をあらわします。

X\$, Y\$ 任意の文字列をあらわします。

整数が必要な場合に、浮動小数点形式であらわされた数値が与えられた場合、BASICは、小数部分を切り捨て、整数部分のみを使います。

CBM BASIC関数の一覧表

関数		結果	
		ニューメリック	ストリング
ABS	3. 1	X	
ASC	3. 2	X	
ATN	3. 3	X	
CHR\$	3. 4		X
COS	3. 5	X	
EXP	3. 6	X	
FRE	3. 7	X	
INT	3. 8	X	
LEFT\$	3. 9		X
LEN	3.10	X	
LOG	3.11	X	
MID\$	3.12		X
PEEK	3.13	X	
POS	3.14	X	
RIGHT\$	3.15		X
RND	3.16	X	
SGN	3.17	X	
SIN	3.18	X	
SPC	3.19		X
SQR	3.20	X	
STATUS	3.21	X	
STR\$	3.22		X
TAB	3.23		X
TAN	3.24	X	
TIME	3.25	X	
TIME\$	3.26		X
USR	3.27	X	
VAL	3.28	X	

3.1 ABS

書式： ABS (X)

機能： 式Xの絶対値を与えます。

例： PRINT ABS (7*(-5))
35

3.2 ASC

書式： ASC(X\$)

機能： スtring X \$の最初の文字のASCII (アスキー)コードを与えます (ASCIIコードについては付録Dを参照)。

例： 10 X\$="VIC"
20 PRINT ASC(X\$)
RUN
86

ASCIIコードから文字への変換については、CHR\$関数 (3.4)を参照して下さい。

3.3 ATN

書式： ATN (X)

機能： Xのアーктンジェントの値 (ラジアン単位) を与えます。演算結果は、 $-\pi/2$ から $\pi/2$ の範囲になります。式Xはどのような型の数値であってもかまいませんが、ATNの演算は常に浮動小数点形式でおこなわれます。

例： 10 INPUT X
20 PRINT ATN (X)
RUN
? 3 (キーボードより3を入力します。)
1.24904577

3.4 CHR\$

書式： CHR\$(I)

機能： ASCII (アスキー) コードが I である文字を与えます (ASCII コードについては付録Dを参照)。通常、CHR\$は特別な文字を出力するのに用いられます。たとえばCHR\$(147)を出力することにより、画面をクリア (消去) し、カーソルをホームポジションに戻すことができます。

例： PRINT CHR\$(65)
A

文字をASCIIコードに変換するのは、ASC関数 (3.2) を参照して下さい。

3.5 COS

書式： COS(X)

機能： X (ラジアン単位) のコサインの値を与えます。COS(X)の演算は浮動小数点形式でおこなわれます。

例： 10 X=2*PI
20 PRINT X
RUN
1.08060461

3.6 EXP

書式： EXP(X)

機能： eをX乗した値を与えます。Xは88.02969191より小さいか等しい値でなければいけません。

もしEXP関数がオーバーフローした場合、"OVERFLOW" エラーが発生します。

例： 10 X=3
20 PRINT EXP(X-1)
RUN
7.3890561

3.7 FRE

書式： FRE(X)

機能： メモリー内でBASICが未使用のバイト数を与えます。FRE関数で使う引数Xはダミーですので、任意の数値を使用して下さい。

例： PRINT FRE(0)
3553 (その時に未使用のバイト数が表示されます)

3.8 INT

書式： INT(X)

機能： Xより小さいか等しい最大の整数を与えます。

例： PRINT INT(89.72)
89

PRINT INT(-23.17)
-24

3.9 LEFT\$

書式： LEFT\$(X\$,I)

機能： X\$の左側からI個の文字列を与えます。Iは0から255までの範囲の数でなければなりません。

もしIがX\$の桁数(=LEN(X\$))より大きければ、X\$全体を結果として与えます。

もしIが0ならば、ヌルコード((00)₁₆, 長さは0)を与えます。

```
10 A$="COMMODORE JAPAN"  
20 B$=LEFT$(A$,9)  
30 PRINTB$  
RUN  
COMMODORE
```

MID\$, RIGHT\$関数も参照して下さい。

3.10 LEN

書式： LEN (X\$)

機能： X\$を構成する文字の長さ（桁数）を与えます。プリントされない文字や空白（スペース）も数えます。

例： 10 X\$="COMMODORE JAPAN"
20 PRINT LEN(X\$)
RUN
15

3.11 LOG

書式： LOG (X)

機能： Xの自然対数を与えます。Xは必ず0より大きくなければなりません。

例： PRINT LOG(37/5)
2.00148

3.12 MID\$

書式： MID\$(X\$,I[,J])

機能： X\$の左側よりI番目からJ個の文字列を与えます。IおよびJは、0から255までの範囲の数でなければいけません。もしJが省略されていたり、I番目の文字より右にJ個より少ない文字しかない場合は、I番目より右側の文字全部を関数の値として与えます。

もしI>LEN(X\$)、つまりX\$の桁数がIより小さい場合は、関数の値として、ヌルコード(=(00)16,長さ0)を与えます。

例： 10 A\$="GOOD"
20 B\$="BY LUCK"
30 PRINT A\$;MID\$(B\$,3,5)
RUN
GOOD LUCK

LEFT\$, RIGHT\$も参照して下さい。

3.13 PEEK

書式: PEEK (I)

機能: I番地のメモリーの内容を読みだした値 (0から255までの10進整数) を与えます。Iは0から65535までの範囲の数でなければなりません。PEEKは、POKE (2.22参照) の逆の働きをする関数です。

例: A=PEEK (4096)

3.14 POS

書式: POS (X)

機能: 画面上のカーソルの現在の横位値を与えます (左端が0)。引数Xはダミーです。

例: 100 PRINT " ";POS(X)
RUN
7

3.15 RIGHT\$

書式: RIGHT\$ (X\$,I)

機能: X\$の右側からI個の文字列を与えます。
もし、 $I \geq \text{LEN}(X\$)$ 、つまりX\$の桁数とIが等しいまたは大きいならば、結果はX\$全部を与えます。
もし、 $I = 0$ ならば、ヌルコード ($(00)_{16}$ 、長さは0) を与えます。

例: 10 A\$="COMMODORE JAPAN"
20 B\$=RIGHT\$(A\$,5)
30 PRINTB\$
RUN
JAPAN

MID\$, LEFT\$も参照して下さい。

3.16 RND

書式： RND (X)

機能： Xの正負0により1、-1、0を与えます。
X<0の場合、その数字に割りあてられた1個の乱数を常に与えます。
X=0の場合、使用のたびに同一の順序の乱数系列を与えます。
X>0の場合、使用のたびに新しい乱数系列を与えます。

例： 10 FOR I=1 TO 3
20 PRINT INT (RND (1) * 100);
30 NEXT
RUN
66 1 90 (ここで表示される数は乱数ですので必ずしも左の
数と同じではありません。)

3.17 SGN

書式： SGN (X)

機能： Xの正負0により1、-1、0を与えます。
X>0の場合、SGN (X) = 1
X=0の場合、SGN (X) = 0
X<0の場合、SGN (X) = -1

例： 5 INPUT X
10 ON SGN(X)+2 GOTO 100,200,300
100 PRINT "X<0":END
200 PRINT "X=0":END
300 PRINT "X>0":END

この場合、X<0の時、ライン・ナンバー100へ、
X>0の時、ライン・ナンバー300へ、
X=0の時、ライン・ナンバー200へ
分岐します。

3.18 SIN

書式： SIN (X)

機能： X (ラジアン単位) のサインの値を与えます。SIN(X)の演算は、浮動小数点形式でおこないます。

例： PRINT SIN(1.5)
 .997494987

3.19 SPC

書式： SPC (I)

機能： I個の空白(スペース)で構成される文字列を与えます。SPC関数は、PRINTと共にのみ使用されます。Iは0から255までの範囲の数でなければいけません。

例： PRINT "GOOD" SPC(10) "LUCK"
 GOOD LUCK

3.20 SQR

書式： SQR (X)

機能： Xの平方根を与えます。ただしXは必ず正または0でなければなりません。

例： 10 FOR X=2 TO 10 STEP 2
 20 PRINT X;SQR(X)
 30 NEXT
 RUN
 2 1.41421356
 4 2
 6 2.44948974
 8 2.82842713
 10 3.16227766

3.21 STATUS

書式： ST

機能： カセット、画面（スクリーン）、キーボード、シリアルバスなどで最後に
入出力操作をおこなった時のコンピューターの状態を与えます。

ステータス ビット位置	ステータス値	カセットリード	シリアルバスリード/ ライト	カセット ベリファイ・ロード
0	1		リスナータイムアウト	
1	2		トーカー・タイムアウト	
2	4	ショートブロック		ショートブロック
3	8	ロングブロック回復		ロングブロック
4	16	不能なリードエラー		ミスマッチ
5	32	チェックサムエラー		チェックサムエラー
6	64	エンドオブファイル		
7	-128	エンドオブテープ	当番機器なし	エンドオブテープ

(ショートブロック) ST=4

テープからブロックを読み込んでいる時、読み込まれるべきバイト数に達しないうちにスペース音が入ってきたことを示します。原因としては、LOADされるはずの短いプログラムファイルがデータファイルとして読まれた場合が考えられます。

(ロングブロック) ST=8

テープからのブロック読み込み時に、読まれるべきバイト数を超えたにもかかわらず、スペース音が来ない場合、原因としては、長いプログラムファイルをデータファイルと誤まって読んだ場合が考えられます。

(回復不能なリードエラー) ST=16

原因として冗長ブロックのうちの最初のブロックに31個以上のエラーがあった場合、もしくは冗長ブロックと基本ブロックの双方同一ヶ所にエラーがおこって修復不可能な場合が考えられます。

(チェックサムエラー) ST=32

LOADや、データの読み込みの終了後、RAM中の各バイトについてチェックサムが計算され、入力デバイスから受け取ったバイトと比較されます。もし一致しなければ、チェックサムエラーとなります。

(エンドオブファイル) ST=64

データテープに、エンドオブデータファイルマークがあるとSTに64がセットされます。

(エンドオブテープ) ST=-128

EOT (エンドオブテープ) を読んだ場合、STに128がセットされます。

例 : 10 OPEN 2
20 INPUT # 2, A\$
30 IF ST=0 OR ST=64 THEN50 (ST=0は動作が正常に
40 GOTO20 終了したことを示します)
50 PRINT A\$
60 CLOSE 2

3.22 STR\$

書式: STR\$(X)

機能: Xの数値を表わす文字列を与えます。

例 : 5 REM ARITHMETIC FOR KIDS
10 INPUT "TYPE A NUMBER";N
20 ON LEN(STR\$(N)) GOSUB30,50,70,90
25 END
30 PRINT "LENGTH= 1 ":RETURN
50 PRINT "LENGTH= 2 ":RETURN
70 PRINT "LENGTH= 3 ":RETURN
90 PRINT "LENGTH= 4 ":RETURN

VAL関数(3.28)も参照して下さい。

3.23 TAB

書式: TAB(I)

機能: 画面(スクリーン)またはターミナルのI桁目の位置まで空白(スペース)をプリントします。もし現在のプリント位置がIを越えていれば、TABは無効です。TAB(0)が左端になります。右端は画面の桁数から1ひいた数がIになります。

Iは0から255までの範囲でなければなりません。また、TABは、PRINTステートメントでしか使用できません。

例 : 10 PRINT "NAME" TAB (15) "HEIGHT":PRINT
20 READ A\$, B\$
30 PRINT A\$;TAB (15);B\$
40 DATA "BOB", "180.19"

RUN

NAME	HEIGHT
BOB	180.19

3.24 TAN

書式 : TAN (X)

機能 : X (ラジアン単位) のタンジェントの値を与えます。TAN (X) の演算は、浮動小数点形式でおこなわれます。TAN関数がオーバーフローした場合は、"OVERFLOW" エラーが発生します。

例 : 10 Y=P *TAN (X) / 3

3.25 TIME

書式 : TI

機能 : 内部タイマーを読むのに使用されます。約1/60秒 (1ジフィー) の単位で内部タイマーの時間が与えられます。これは、実際の時計とはちがいます。

例 : 10 A=TI
20 IF TI-A<120THEN20
30 PRINT " 2 SECONDS"

この場合TIは、約1/60秒毎に値がかわってくるために、プログラムを実行しますとライン・ナンバー10でAにスタート時間をセットして、約2秒後に、" 2 SECONDS" をプリントします。

3.26 TIMES

書式： TIS

機能： 内部タイマーを読んで、その結果を "HHMMSS" (HH:時, MM:分, SS:秒) の文字列 (ストリング) で与えます。電源を投入した直後に内部タイマーはすべて000000にセットされます。したがって、内部タイマーに初期値を代入しない場合は、TISはコンピュータの使用時間をあらわします。

内部タイマーの値に初期値を与えるためには、INPUTステートメントといっしょに、または、代入式 (下記の例を参照) を用います。

```
例 : 10 TIS="000000"  
      20 FOR I=1 TO 1000 :NEXT  
      30 PRINT TIS  
      RUN
```

000001 (実行すると、約1秒後にこれが表示されます)

3.27 USR

書式： USR(X)

機能： メモリーの1番地、2番地にストアされている番地に、プログラムのコントロールを移し、引数をフローティングアキュムレーターに格納します。

例： 828番地にダイレクトモードでキーボードから、次のようにして機械語ルーチンを書いておきます。

```
POKE828, 169
```

```
POKE829, 01
```

```
POKE830, 141
```

```
POKE831, 0
```

```
POKE832, 30 > 画面(V-RAM)は7702番地から開始しています。
```

```
POKE833, 96
```

```
POKE1, 60
```

```
POKE2, 3
```

画面をクリアして、カーソルをホームポジションに戻して下さい。そして、

```
B=USR(0)
```

と入力します。

すると、画面の左上に 'A' の文字が出力されます。次に

PRINT B と入力しますと

10

と表示されます。

3.28 VAL

書式： VAL (X \$)

機能： 文字列 (ストリング) X \$ の表わす数値を与えます。もし X \$ の最初の文字が +, -, または数字でなければ VAL(X \$) の値は 0 になります。

例： **10 A\$="123"**
20 B\$="+123"
30 C\$="-123"
35 D\$="ABC"
40 PRINT VAL (A\$)
50 PRINT VAL (B\$)
60 PRINT VAL (C\$)
70 PRINT VAL (D\$)
RUN
123
123
-123
0

数値を文字列 (ストリング) に変える場合は、STR\$関数 (3.22) を参照して下さい。

第4章 CBM BASICのプログラム例

この章は、これまでに説明したCBM BASICのコマンド、ステートメントを用いて作成した、簡単なプログラム例を集めてみました。いずれも短かくて、かつ結果がスクリーンにすぐに表示されるもの、音が出力されるものばかりです。何はともあれ、プログラム・リストを見ながら、キーボードからプログラムを入力して下さい。それを実行させてエラーが発生した場合、エラーメッセージ一覧表などを参照して、原因を究明して下さい。また、これらのプログラムを参考にして、より楽しい、高度なプログラムをたくさん作って下さい。

カーソル制御、反転指定、キャラクター色指定のプログラム

CBM BASICでは、カーソル制御、反転および反転解除指定、キャラクター色指定が、簡単にプログラムできます。コマンドは、ダブルクォーテーション(“)で囲んで、用います。

コ マ ン ド	キ ー 操 作	画面にあらわれる文字
画面クリア	"CLR"	□ (□)
ホーム	"HOME"	☺
カーソル右	"CRSR⇒"	▶
カーソル左	"CRSR⇐"	◀
カーソル下	"CRSR↓"	⤵
カーソル上	"CRSR↑"	⤴
一字追加	"INST"	■ (■)
反 転	"CTRL R"	☺
反転解除	"CTRL Ø"	☐
ブラック	"CTRL 1"	■ (■)
ホワイト	"CTRL 2"	☐
レッド	"CTRL 3"	■ (■)
シアン	"CTRL 4"	▲ (■)
マゼンタ	"CTRL 5"	■ (■)
グリーン	"CTRL 6"	■ (■)
ブルー	"CTRL 7"	■ (■)
イエロー	"CTRL 8"	■ (■)

1. はじめまして

```
10 REM *****
20 REM * HOW DO YOU DO ? *
30 REM *****
40 REM
50 A$="          "
60 PRINT " * * * HOW DO YOU DO ? * *"
70 PRINT A$ " * "
80 PRINT TAB(5) "I ♪ V ♪ I"
90 PRINT TAB(5) " | | "
100 FOR I=1TO500:NEXT
110 PRINT A$ " "
120 PRINT TAB(5) "I ♪ ♪ I"
130 PRINT TAB(5) " | | "
140 FOR I=1TO500:NEXT
150 PRINT A$ " * "
160 PRINT TAB(5) "I ♪ V ♪ I"
170 PRINT TAB(5) " | | "
180 END
```

VICぼうやがおじぎをします。

ライン・ナンバー70~90とライン・ナンバー150~170は両方共、顔をあげた状態です。

ライン・ナンバー110~130は頭をさげた状態です。

ライン・ナンバー100、140は、その間、画面を停止させます。

2. VIC SQUIGGLE (スキィグル)

```
1   C$=" ■■■■■■ "
5   PRINT"□";
10  DATA "|", "-", "]", "L", "r", "┐", "└"
20  DATA 1,0,5,6
30  DATA 0,1,4,3
40  DATA 3,6,2,0
50  DATA 4,5,0,2
60  DIM A$(5), B(5,5)
70  FOR I=0TO5
80  READ A$(I)
90  NEXT
100 FOR I=1TO4
110 FOR J=1TO4
120 READ B(J, I)
130 NEXT
140 NEXT
190 T1=1
200 T2=1
210 X=20
220 Y=12
300 REM ***START
310 T1=4*RND(1)+1
320 IF B(T1, T2)=0 THEN310
325 GOSUB2000
330 T2=T1
340 ON T1 GOTO400,410,420,430
400 Y=Y-1 : GOTO500
410 Y=Y+1 : GOTO500
420 X=X+1 : GOTO550
430 X=X-1 : GOTO550
500 IF Y<1 THEN Y=20 : GOTO300
510 IF Y>20 THEN Y=1 : GOTO300
550 IF X<1 THEN X=20 : GOTO300
560 IF X>20 THEN X=1 : GOTO300
570 GOTO300
2000 PRINT"□";
2010 FOR I=1TOY
2020 PRINT"■";
2030 NEXT
2040 FOR I=1TOX
```

```
2050 PRINT "11";
2055 NEXT
2060 PRINT "2" MID$(C$,RND(1)*7+1,1);
2080 PRINT A$(B(T1,T2)-1);
2090 RETURN
```

ライン・ナンバー2060を次のようになおすと、また、ちがった画面を楽しむことができます。

```
2060 PRINT MID$(C$,RND(1)*7+1,1);
```

3. レコーディング その1

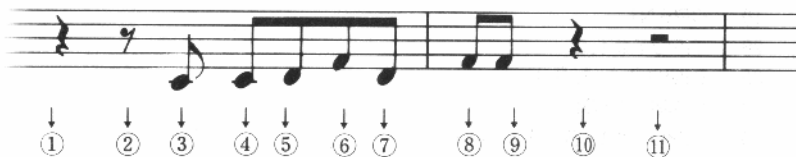
```
1  REM *****
2  REM * RECORDING *
3  REM *****
4  REM
10 INPUT "LENGTH";L
20 DIM S(L), L(L)
30 FOR I=1TOL
40 INPUT A, B
50 S(I)=A:L(I)=B
60 NEXT
70 PRINT "WRITE OR RUN(W OR R) ?"
80 GET O$: IF O$="R" THEN 110
90 IF O$("<">"W" THEN 80
100 GOSUB 200
110 POKE 36878, 15
120 FOR I=1TOL
130 POKE 36874, S(I)
140 FOR J=0TOL(I) * 300
150 NEXT : NEXT
160 END
200 OPEN 1, 1, 1
210 PRINT #1; L
220 FOR I=1TOL : PRINT #1, S(I) : PRINT #1, L(I)
230 NEXT : CLOSE 1 : GOTO 70
```

音符のかわりにキーボードから、コードを入力して、VICに演奏させましょう。
入力したコードは、テープにセーブして残しておくことができます。

〈実行手順〉

- ① RUN **RETURN**
- ② 画面に LENGTH? と表示され、カーソルが点滅します。
- ③ キーボードから音符の数をいれます。

たとえば、次の場合



11と入れて下さい。

④ 次に、音のコードと長さをいれます。

③の例の場合、

∅,2 **RETURN**

∅,1 "

191,1 "

191,1 "

198,1 "

2∅7,1 "

198,1 "

2∅7,1 "

2∅7,1 "

∅,2 "

∅,4 "

⑤ ③で指定した数だけコードと長さを入れると、画面は

WRITE OR RUN (W OR R) ?

と表示されます。

R を押すと、④で作った音が流れてきます。

W を押すと、PRESS PLAY & RECORD ON TAPEと表示されますので、テープをセットしてデータを書きこんで下さい。

⑥ 書きこみが終了すると⑤に戻ります。

参照：第2部付録Gサウンド

4. レコーディング その2

```
10 OPEN1
20 INPUT #1,L
30 DIMS(L),L(L)
40 FORI=1TOL
50 INPUT #1,S(I),L(I)
60 NEXT
70 PRINT"END OR RUN(E OR R)?"
80 GETOS:IFOS="R"THEN110
90 IFO$<>"E"THEN80
100 END
110 POKE36878,15
120 FORI=1TOL
130 POKE36874,S(I)
140 FORJ=0TOL(I)*100
150 NEXT:NEXT
160 GOTO70
```

レコーディング その1で作成したテープを読んで音を出します。

- ① RUN **RETURN**
- ② PRESS PLAY ON TAPE と画面に表示されます。テープをセットして **PLAY** を押して下さい。
- ③ END OR RUN (E OR R) ? と表示されます。
- ④ E を押すと、プログラムを終了します。
R を押すと、音が出ます。終わった後、③へ戻ります。

5. VICオルガン

```

100 REM
110 REM VIC ORGAN
120 REM
130 DIM K$(8), KY(9), MU(9)
140 FOR I=1 TO 9:READ KY(I):NEXT
145 FOR I=1 TO 9:READ MU(I):NEXT
150 DATA 0,56,1,57,2,58,3,59,4
155 DATA 63,70,76,79,85,89,93,95,0
160 CR$="          "
170 SP$="          "
180 FOR I=1 TO 8
190 K$(I)=MID$(CR$,I,1)+SP$
200 NEXT I
205 K$(0)=" "+SP$
210 VA$="          "
220 PRINT" "
230 FOR I=1 TO 8
240 PRINTVA$;SPC((I-1)*2);K$(0)
250 NEXT
260 A=PEEK(197):IFA=63 THEN GOSUB1000
270 FOR I=1 TO 8
280 IF A=KY(I) THEN J=I:I=8:NEXT:GOTO300
290 NEXT I:GOTO260
300 POKE36878,15:PRINTVA$;SPC((J-1)*2);K$(J)
305 POKE36874,(MU(J)+128)
310 GOTO260
1000 POKE36874,0:FOR H=15 TO 0 STEP 1:POKE36878,H:NEXT:
RETURN

```

① RUN RETURN

- ②
- | | |
|--|--|
| <p>1を押すと、ドの音が出ます。</p> <p>2 " レ "</p> <p>3 " ミ "</p> <p>4 " ファ "</p> <p>5 " ソ "</p> | <p>6を押すと、ラの音が出ます。</p> <p>7 " シ "</p> <p>8 " ド "</p> <p>0 " 音が消えます。</p> |
|--|--|

参照：第2部付録F カラーコントロール、付録G サウンド

6. 壁こわしゲーム

```

100 REM *****
110 REM *VIC WALL DESTROY*
120 REM *****
130 GOSUB620:REM TITLE WRITE
140 VR=7680
150 DEF FNA(X)=(V-1)*22+X+VR:REM VRAM ADDRESS
160 DEF FNB(X)=(PEEK(FNA(X+1))=32)
170 V=2:REM V=VERTICAL
180 SC$="0000":WAS="":PDS="":SP$=" "
190 VA$=" "
200 TIS="000000"
210 PRINT "SCORE:0000 TIME:00":PRINTPDS
220 GOSUB410
230 GETAS
240 PRINT " ";SPC(16):RIGHT$(TIS,2)
250 IFTIS>"000100"THEN500
260 IFAS=" "THEN230
270 IFAS="|"THENDT=1:GOSUB320:GOTO300
280 IFAS="|"THENDT=-1:GOSUB320:GOTO300
290 IFAS=" "THENGOSUB360
300 IFWA=0THENGOSUB410
310 GOTO230
320 PRINT LEFT$(VA$,V):SP$:V=V+DT
330 IFV>20THENV=20
340 IFV<2THENV=2
350 PRINT LEFT$(VA$,V):PDS:RETURN
360 POKE 36879,128+2
365 POKE FNA(0),81:POKEFNA(0),97
370 FORI=1TO20:POKEFNA(I),81:POKE FNA(I),32
380 IFFNB(I)=-1THEN400
390 GOSUB460
400 NEXTI:POKE 36879,27:RETURN
410 Z$=" ":FORI=1TO19
420 IFRND(1)>.7THENZ$=Z$+WAS:WA=WA+1:GOTO440
430 Z$=Z$+SP$
440 NEXTI
450 PRINT " "SPC(20)Z$:RETURN
460 POKEFNA(I),32:POKE FNA(I+1),42:POKE FNA(I+1),32
470 WA=WA-1:I=20:SC=SC+10
480 PRINT " "RIGHT$(SC$+MID$(STR$(SC),2),4)
490 RETURN

```

```

500 PRINT "SPC(16) "*"
510 PRINT LEFT$(VA$,10) " GAME OVER "
520 PRINT "PRESS RETURN TO START"
530 GETA$:IFAS<>CHR$(13)THEN530
540 RUN
550 DATA "VIC WALL DESTROY."
560 DATA "KEYBORD COMMAND."
570 DATA ""
580 DATA "CRSR UP . . . . . MOVE UP."
590 DATA "CRSR DOWN . . . . . MOVE DOWN."
600 DATA "SPACE . . . TO FIRE."
610 DATA "*"
620 PRINT ""
630 READ A$:IFAS="*"THENRESTORE:GOTO720
640 FORI=1TOLEN(A$)
650 PRINT MID$(A$,I,1) " ";
660 FORJ=1TO5
670 GETB$:IFBS<>" "THENJ=5
680 NEXTJ
690 IFBS<>" "THENI=LEN(A$):NEXTI:GOTO760
700 NEXTI
710 PRINT " ":GOTO630
720 FORI=1TO5000
730 GETA$:IFAS<>" "THENI=5000
740 NEXTI
750 IFAS=" "THEN620
760 RETURN

```

LET'S PLAY!!

- (注) ライン・ナンバー140のVRは、スクリーンのスタートアドレスです。拡張メモリーの8Kまたは16Kを使用している場合は、140 VR=4096に変更して下さい。

7. ひらがな

```
1 POKE51,0:POKE52,19
2 POKE55,0:POKE56,19
3 CLR
5 FORI=1TO40
10 READ A
20 POKE1023+4096+I,A
30 NEXT
40 GOTO100
50 DATA16,124,16,58,84,154,178,82
60 DATA0,2,129,129,129,129,65,2
61 DATA24,0,24,36,4,4,4,8
62 DATA48,0,120,16,32,84,136,0
63 DATA32,248,34,33,120,164,164,68
100 POKE36867,PEEK(36867)AND254
105 POKE36866,PEEK(36866)OR128
110 POKE36869,253
140 END
```

- ① RUN RETURN と押して実行させます。
- ② @ を押すと、画面に「あ」が表示されます。
- | | | | |
|----------|---|---|---|
| <u>A</u> | ” | ” | ” |
| <u>B</u> | ” | ” | ” |
| <u>C</u> | ” | ” | ” |
| <u>D</u> | ” | ” | ” |

参照：第2部付録Hハイ・レゾリューション

8. 文字パターン作成プログラム

```
100 REM *****
110 REM MAKE PROGRAMMABLE CHARACTER *
120 REM *
130 REM *FOR VIC 1001 *
140 REM *****
150 VR=7680
160 VS="  " :WS="....."
170 PRINT" WHICH ?"
180 PRINT" (HIT NUMERIC KEY.)"
190 PRINT" 1 .8 *8 DOT."
200 PRINT" 2 .8 *16 DOT."
210 GETAS:IFAS="" THEN210
220 IFAS<>"1"ANDAS<>"2"THEN210
230 CH=0:IFAS="2"THENCH=1
240 CL=7:IFCH=1THENCL=15
250 PRINT":
260 J=8:IFCH=1THENJ=16
270 FORI=1TOJ:PRINTW$:NEXTI
280 X=VR:POKEX,(PEEK(X)OR128)
290 GETAS:IFAS="" THEN290
300 IFAS=CHR$(20) THEN590
310 IFAS=CHR$(13) THEN650
320 FORI=1TO8
330 IFMID$(VS,I,1)=AS$ THENJ=I:I=8:NEXTI:GOTO360
340 NEXTI
350 GOTO290
360 ONJGOTO370,400,430,460,490,520,560
370 REM CURSOL DOWN
380 X0=22:Y%=Y%+1:IFY%>CL THENY%=CL:X0=0
390 GOSUB630:GOTO290
400 REM CURSOL UP
410 X0=-22:Y%=Y%-1:IFY%<0 THENY%=0:X0=0
420 GOSUB630:GOTO290
430 REM CURSOL RIGHT
440 X0=1:X%=X%+1:IFX%>7 THENX%=7:X0=0
450 GOSUB630:GOTO290
460 REM CURSOL LEFT
470 X0=-1:X%=X%-1:IFX%<0 THENX%=0:X0=0
480 GOSUB630:GOTO290
490 REM CURSOL HOME
```

```

500 X%=0:Y%=0
510 POKE X,(PEEK(X)AND127):X0=0:X=VR:GOSUB630:GOTO290
520 REM PATTERN CLR
530 X%=0:Y%=0:PRINT" ";
540 Y=8:IFCH=1 THEN Y=16
550 FOR I=1 TO Y:PRINTW$:NEXT:GOTO280
560 REM DOT!
570 POKE X,81:X0=1:X%=X%+1:IFX%>7 THEN X%=7:X0=0
580 GOSUB630:GOTO290
590 REM DELETE DOT
600 POKE X,46:X0=-1:X%=X%-1:IFX%<0 THEN X%=0:X0=0
610 GOSUB630:GOTO290
620 REM RVS SUBROUTINE
630 POKE X,(PEEK(X)AND127):X=X+X0:POKE X,(PEEK(X)OR128)
640 RETURN
650 REM CALCULATE
660 PRINT" ";
670 Z=7:IFCH=1 THEN Z=15
680 POKE X,(PEEK(X)AND127)
690 FOR I=VR TO VR+22 *Z STEP 22
700 FOR J=0 TO Z
710 IF PEEK(I+J)<>81 THEN 730
720 Y=Y+2↑(7-J)
730 NEXT J
740 IF I=VR THEN 750
750 PRINT TAB(10)MID$(STR$(Y),2):Y=0
760 NEXT I
770 PRINT:PRINT" 実行 RUN TT":END

```

7. のひらがなプログラムのライン・ナンバー50から63までのDATA文に書かれた数字を簡単に見つけることができます。

付 録

- A : CBM BASICのコード表
- B : CBM BASICの省略形
- C : 他のBASICからCBM BASICへの変換
- D : エラーメッセージ一覧表
- E : 誘導関数
- F : カラーコントロール
- G : サウンド
- H : ハイ・レゾリューション

A : CBM BASICのコード表

BASICはコマンド、ステートメント、関数などを、1文字 (=1バイト) に変換してメモリー内部に格納します。変換するコードは次表の通りです。

コード	BASICの表現	コード	BASICの表現	コード	BASICの表現
128	END	154	CONT	180	SGN
129	FOR	155	LIST	181	INT
130	NEXT	156	CLR	182	ABS
131	DATA	157	CMD	183	USR
132	INPUT #	158	SYS	184	FRE
133	INPUT	159	OPEN	185	POS
134	DIM	160	CLOSE	186	SQR
135	READ	161	GET	187	RND
136	LET	162	NEW	188	LOG
137	GOTO	163	TAB (189	EXP
138	RUN	164	TO	190	COS
139	IF	165	FN	191	SIN
140	RESTORE	166	SPC	192	TAN
141	GOSUB	167	THEN	193	ATN
142	RETURN	168	NOT	194	PEEK
143	REM	169	STEP	195	LEN
144	STOP	170	+	196	STR\$
145	ON	171	-	197	VAL
146	WAIT	172	*	198	ASD
147	LOAD	173	/	199	CHR\$
148	SAVE	174		200	LEFT\$
149	VERIFY	175	AND	201	RIGHT\$
150	DEF	176	OR	202	MID\$
151	POKE	177	>	203	GO
152	PRINT #	178	=	204	?SYNTAX
153	PRINT	179	<		ERROR (注)

(注) 変換後、このエラーメッセージが出力されます。

B : CBM BASICの省略形

CBM BASICでは、コマンド、ステートメント、関数などのキーワードを省略形であらわすことができます。たとえば、LISTの場合、まずLをプリントし、**[SHIFT]** キーを押しながらIをプリントします。もし、グラフィック・キャラクター・モードならL□と表示され、片仮名キャラクター・モードなら、Lノとなります。どちらの場合でも、LISTのかわりになります。CBM BASICのこの特徴をうまく利用すれば、プログラム作成時のバイト数節約にもなります。

以下に、キーワードの省略形の一覧表を示します。○で囲んだアルファベットは**[SHIFT]** キーを押しながら、そのキーを押すことを示します。その左に、ディスプレイに表示される形を示します。() 内は片仮名モードの場合です。

キーワード	省略形	キーワード	省略形
AND	A [Ⓝ] A [Ⓝ] (Aホ)	RUN	R [Ⓞ] R [Ⓝ] (Rユ)
NOT	N [Ⓞ] N [Ⓝ] (Nマ)	SAVE	S [Ⓜ] S [♠] (Sチ)
		STEP	ST [Ⓜ] ST [Ⓝ] (STナ)
CLOSE	CL [Ⓞ] CL [Ⓝ] (CLマ)	STOP	ST [Ⓝ] S [Ⓝ] (Sヤ)
CLR	CL [Ⓞ] C [Ⓝ] (Cリ)	SYS	S [Ⓞ] S [Ⓝ] (Sル)
CMD	CM [Ⓜ] C [Ⓝ] (Cへ)	THEN	TH [Ⓜ] T [Ⓝ] (Tネ)
CONT	CO [Ⓞ] C [Ⓝ] (Cマ)	VERIFY	VE [Ⓜ] V [Ⓝ] (Vナ)
DATA	DA [Ⓜ] D [♠] (Dチ)	WAIT	WA [Ⓜ] W [♠] (Wチ)
DEF	DE [Ⓜ] D [Ⓝ] (Dナ)		
DIM	DI [Ⓞ] D [Ⓝ] (Dノ)	ABS	A [Ⓜ] A [Ⓝ] (Aツ)
END	EN [Ⓝ] E [Ⓝ] (Eホ)	ASC	AS [Ⓞ] A [♥] (Aモ)
FOR	FO [Ⓞ] F [Ⓝ] (Fマ)	ATN	AT [Ⓜ] A [Ⓝ] (Aヤ)
GET	GE [Ⓜ] G [Ⓝ] (Gナ)	CHR\$	CH [Ⓜ] C [Ⓝ] (Cネ)
GOSUB	GO [Ⓞ] GO [♥] (GOモ)	EXP	E [Ⓝ] E [♣] (Eリ)
GOTO	GO [Ⓞ] G [Ⓝ] (Gマ)	FRE	FR [Ⓜ] F [Ⓝ] (Fメ)
INPUT#	I [Ⓝ] I [Ⓝ] (Iホ)	LEFT\$	LE [Ⓜ] LE [Ⓝ] (LEニ)
LET	LE [Ⓜ] L [Ⓝ] (Lナ)	MID\$	M [Ⓞ] M [Ⓝ] (Mノ)
LIST	LI [Ⓞ] L [Ⓝ] (Lノ)	PEEK	PE [Ⓜ] P [Ⓝ] (Pナ)
LOAD	LO [Ⓞ] L [Ⓝ] (Lマ)	RIGHT\$	RI [Ⓞ] R [Ⓝ] (Rノ)
NEXT	NE [Ⓜ] N [Ⓝ] (Nナ)	RND	RN [Ⓞ] R [Ⓝ] (Rホ)
OPEN	OP [Ⓜ] O [Ⓝ] (Oミ)	SGN	SG [Ⓞ] S [Ⓝ] (Sヌ)
POKE	PO [Ⓞ] P [Ⓝ] (Pマ)	SIN	SI [Ⓞ] S [Ⓝ] (Sノ)
PRINT	? ?	SPC (SP [Ⓜ] S [Ⓝ] (Sミ)
PRINT#	PR [Ⓜ] P [Ⓝ] (Pメ)	SQR	SQ [Ⓞ] S [♣] (Sム)
READ	RE [Ⓜ] R [Ⓝ] (Rナ)	STR\$	ST [Ⓜ] ST [Ⓝ] (STメ)
RE-	RE [Ⓞ] RE [♥] (REモ)	TAB (TA [Ⓜ] T [♠] (Tチ)
STORE		USR	US [Ⓞ] U [♥] (Uモ)
RETURN	RE [Ⓝ] RE [Ⓝ] (REヤ)	VAL	VA [Ⓜ] V [♠] (Vチ)

C：他のBASICからCBM BASICへの変換

他のBASICで書かれたプログラムをVIC-1001パーソナル・コンピュータで実行する場合、多少プログラムをCBM BASIC用に変更する必要があります。どのようにするのかの例を次に説明します。

C.1 文字列(ストリング)配列

文字列(ストリング)の長さを宣言するのに使用されるすべてのステートメントを消去して下さい。たとえば、DIM A\$(I, J)は、Jの要素を持った長さIの文字配列を表わしますが、これはCBM BASICでは、DIM A\$(J)に書き直して下さい。

いくつかのBASICでは、コンマ(,)やアンド(&)を文字列(ストリング)を連結するのに使用していますが、これはCBM BASICでは、プラス符号(+)を用いて下さい。

CBM BASICでは、MID\$, RIGHT\$, LEFT\$関数を、文字列(ストリング)の中から指定した文字数分だけとりだして、文字列を作るのに使用します。他のBASICでは、A\$(I)はA\$の中から第I番目の1文字をとりだします。またA\$(I, J)はA\$の第I番目からJ番目の文字列をとりだします。これをCBM BASICに変換する場合は次のようにします。

他のBASIC	CBM BASIC
A\$(I)=X\$	A\$=LEFT\$(A\$, I-1)+X\$+MID\$(A\$, I+1)
A\$(I, J)=X\$	A\$=LEFT\$(A\$, I-1)+X\$+MID\$(A\$, J+1)

C.2 複数の代入文

いくつかのBASICでは次のような形を用いて、BとCに同時に \emptyset を代入することができます。

```
10 LET B=C= $\emptyset$ 
```

しかしCBM BASICでは2番目の等号(=)を論理演算子として処理するために、Cが \emptyset の場合、Bには-1が代入されます。

そこで、CBM BASICでBおよびCに \emptyset を代入するためには次のようにします。

```
10 C= $\emptyset$ :B= $\emptyset$ 
```

C.3 マルチステートメント

複数のステートメントを1行上にならべて書く場合、それぞれのステートメントをバックスラッシュ(\)で区切って書くBASICがありますが、CBM BASICの場合は、コロン(:)を使用して区切ります。

MAT関数

MAT関数を用いている場合、CBM BASICでは、FOR-NEXTループを使用して書きなおして下さい。

D：エラーメッセージ一覧表

BASICエラー	
エラーメッセージ	原 因
BAD SUBSCRIPT	<p>DIMステートメントで確保された配列の範囲を越えたような場合、またDIM宣言をせずに、配列を使用した場合（ただし、BASICの文法上、配列の添え字が0～9までの範囲では、DIM宣言する必要はありません）。</p> <p>例) DIM A (2,2) と宣言したあと A (1,1,1) =2 を実行すると ?BAD SUBSCRIPT ERRORになります。</p> <p>DIM宣言をしないで直接 A (10,10) =2 を実行すると ?BAD SUBSCRIPT ERRORになります。</p>
CAN'T CONTINUE	<p>通常、STOPコマンドの実行、もしくは STOP キーを押すことにより、プログラムの実行を中止させた場合に、CONTによりSTOPを解除して、引き続き実行を再開できますが、次の4通りの場合に、CONTを実行させようとすると、このエラーが発生します。</p> <ol style="list-style-type: none"> 1) プログラムが存在しない場合 2) STOPした後に、新しい行を追加した場合 3) プログラムがRUNコマンドにより実行されていない場合 4) エラーが発生して実行が中断された場合
DIVISION BY ZERO	<p>除算（割り算）において分母を0で実行しようとした場合に、このエラーがおきます。</p> <p>もしプログラム実行中に、このエラーメッセージが出力されたなら、次の方法で、分母が0であることを、確認することができます。</p>

エラーメッセージ	原因
DIVISION BY ZERO	<p>? DIVISION BY ZERO ERROR IN 10</p> <p>LIST 10 (エラーの発生した行のリストを調べます)</p> <p>10 A=B/C (このような計算式だったとします)</p> <p>PRINT C (ダイレクトモードで分母Cの値を調べますと、0になっていることがわかります)</p> <p>0</p>
FORMULA TOO COMPLEX	<p>文字列式(STRING式)を評価するさいに、文字列式が複雑すぎて、メモリー内の文字列のポインタの領域をこえてしまった場合に発生します。</p> <p>文字列式をいくつかにわけて短かくすることにより、解決することができます。</p>
ILLEGAL DIRECT	<p>BASICで、INPUTまたはGETでデータを入力する場合、88文字 (=88バイト) 長の入力バッファを使用します。この同じバッファが、ダイレクトモードで入力されたコマンドをとりこんで実行させるためにも使用されます。そのため、INPUTおよびGETを、ダイレクトモードで使用すると、このエラーが発生します。</p> <p>DEFもまた、多少ちがいますが、同じように、バッファの関係でダイレクトモードで使うことができません。つまり関数名は、関数定義をおこなった式のポインタと一しょにBASICの変数エリアに格納されます。そのため関数は、入力バッファの中にだけ残されるため、次のコマンドが何か入力されると、関数がなくなってしまいます。</p> <p>DEFをダイレクトモードで使用しますとこのエラーが発生します。</p>
ILLEGAL QUANTITY	<p>関数に、規定にあわない数値や変数を用いて実行しようとした場合、このエラーが発生します。</p> <p>1) 配列の添字が次の範囲外の場合</p> <p>$0 \leq (\text{添字}) \leq 32767$</p> <p>たとえば $A(-5) = 5$ は</p> <p>? ILLEGAL QUNTITY ERRORになります。</p> <p>2) $\text{LOG}(X)$ ($X > 0$)</p> <p>Xが負または0の場合エラーになります。</p>

エラーメッセージ	原因
	<p>3) SQR (X) (X ≥ 0) Xが負の場合、エラーになります。</p> <p>4) A ↑ B Aが負で、かつBが整数でない場合、エラーになります。</p> <p>PRINT (-5) ↑ 2.3 ?ILLEGAL QUANTITY ERROR</p> <p>5) 機械語ルーチンをセットしないでUSR関数を使用するとエラーになります。</p> <p>6) MID\$, RIGHT\$, LEFT\$の文字列の長さを指定するパラメータの数値が下記の範囲外の場合、エラーになります。</p> <p>1 ≤ (パラメーター) ≤ 255</p> <p>7) ON (式) GOTO (2.20参照) のインデックス (式の評価結果) が次の範囲外の場合、エラーになります。</p> <p>0 ≤ (インデックス) ≤ 255</p> <p>8) PEEK, POKE, WAIT, SYSで指定するアドレスが次の範囲外の場合、エラーになります。</p> <p>0 ≤ アドレス ≤ 65535</p> <p>9) WAIT, POKE, TAB, SPCで使用するパラメータが次の範囲外の場合、エラーになります。</p> <p>0 ≤ (パラメータ) ≤ 255</p> <p>たとえば、POKE 826, 1000を実行すると ?ILLEGAL QUANTITY ERROR になります。</p>
NEX WITHOUT FOR	FOR...NEXTのネスティングが正しくプログラムされなかったか、または、NEXTに対応するFORがない場合に、

エラーメッセージ	原因
NEX WITHOUT FOR	<p>このエラーが発生します。</p> <p>例) FOR I=1 TO 10 : NEXT : NEXT ?NEXT WITHOUT FOR ERROR</p> <p>FOR I=1 TO 10 : NEXT J ?NEXT WITHOUT FOR ERROR</p>
OUT OF DATA	<p>READを実行して読みこもうとするデータの数が、DATAで定義されているデータの数より多い場合に、エラーが発生します。</p> <p>また、ディスプレイの画面に、READY.の文字が表示されている同じ行にカーソルを移動し、RETURN(リターン)キーを押すとこのエラーが発生します(なぜならREADY.をBASICがREADY.という命令として実行しようとするからです)。</p>
OUT OF MEMORY	<p>プログラムを作成または編集している場合、BASICテキストエリア(メモリーマップ参照)がいっぱいになった場合、エラーになります。</p> <p>プログラム実行時には、変数を割りつけたり、作られたりすることにより、メモリーがいっぱいになった場合、エラーになります。</p> <p>プログラムが短かくても、DIMにより配列宣言をすると、大きなメモリー領域を確保しますので注意して下さい。</p> <p>また、FOR...NEXT、GOSUB...RETURNのように、スタックを使用する場合(2.9,2.11参照)、スタックの容量をこえた場合、メモリーに関係なく、このエラーが発生します。たとえば</p> <pre>10 GOSUB10 : GOTO10 RUN ?OUT OF MEMORY ERROR IN 10</pre>

エラーメッセージ	原因
	<p>実際にメモリー容量が足りなくなったのか、単にスタックの容量が一杯になってエラーになったのかはFRE関数を用いて残りのメモリー容量を調べて判断することができます。</p> <pre>PRINT FRE (0)</pre>
OVERFLOW	<p>計算結果や入力した数値が1.70141184E+38より大きい場合、BASICでは扱うことができず、エラーになります。なお、UNDERFLOWはエラーとされませんが、2.93873587E-39より小さい数値は、すべて0として扱われます。</p> <pre>PRINT 1E40 ?OVERFLOW ERROR</pre>
REDIM'D ARRAY	<p>一度宣言した配列を同じ変数名でもう一度宣言しなおすとエラーになります。</p> <p>例) A(5)=6 DIM A(10,10) ?REDIM'D ARRAY ERROR</p>
REDO FROM START	<p>これは他のエラーとちがって、プログラムの実行を中止することはありません。</p> <p>INPUT実行時に、INPUTで指定したのとちがう型のデータ、(たとえば数値 =ニューメリック を指定しているのにアルファベットを入力した場合) を入力した場合に発生します。</p> <p>例) 10 INPUT A RUN ?ABC ("ABC" を入力します) ?REDO FROM START ? (再び入力を要求します)</p> <p>正しくデータが入力されるまでによる入力要求がくり返されます。</p>

エラーメッセージ	原因
RETURN WITHOUT GOSUB	<p>対応するGOSUBがなしで、RETURNを実行しようとした場合、エラーになります。</p> <p>例) CLR (2.2参照) ?RETURN WITHOUT GOSUB ERROR</p>
STRING TOO LONG	<p>255文字より長い文字列(STRING)を作ろうとした場合に、このエラーが発生します。</p> <p>例) A\$="A":FORI=1TO10:A\$=A\$+A\$:NEXT ?STRING TOO LONG ERROR</p>
SYNTAX	<p>BASICの文法上の誤り、つまり、コマンド、ステートメント、関数として、CBM BASICが定めていない形式、文字を使用した場合、文字のつづりをまちがえた場合に、このエラーが発生します。</p> <p>また、かっこの数があわない場合にも、エラーになります。</p> <p>例) A=(1+8 ?SYNTAX ERROR</p>
TYPE MISMATCH	<p>代入文などで式の左右の形が一致していない場合に、このエラーになります。</p> <p>例) A\$=12 ?TYPE MISMATCH ERROR</p>
UNDEF'D FUNCTION	<p>定義していない関数を使用しようとした場合、エラーが発生します。</p> <p>例) X=FNA(3) ?UNDEF'D FUNCTION ERROR</p>
UNDEF'D STATEMENT	<p>GOTO, GOSUB, THENで指定したライン・ナンバーがプログラム中不在の場合、エラーが発生します。</p> <p>例) GOTO A ?UNDEF'D STATEMENT ERROR</p>

オペレーティング・システム (OS) エラー

エラーメッセージ	原因
BAD DATA	<p>指定した機器 (デバイス) よりデータを入力した場合、数値 (ニューメリック) データであるべきなのに文字 (アルファ) データが入力された時に、このエラーが発生します。つまりファイル上にあるデータの形式がまちがっている場合に、発生します。</p>
DEVICE NOT PRESENT	<p>シリアルバスにつながっていない機器 (デバイス) を使用しようとした場合に、発生します。この時、ST (ステータス、3.21参照) はタイムアウトを示す2になります。OPEN, CLOSE, CMD, INPUT #, GET #, PRINT # を実行した時に、このエラーが調べられます。</p>
FILE NOT FOUND	<p>OPENまたはLOADコマンドで指定されたファイル名が、この時いっしょに指定された機器 (デバイス) から、発見することができなかった場合、エラーになります。</p>
FILE NOT OPEN	<p>ファイルをOPENしないで使用した場合に、エラーになります。</p> <p>例) CLR INPUT #10, A ?FILE NOT OPEN ERROR</p>
FILE OPEN	<p>同じファイルを2度OPENしようとした場合、このエラーが発生します。</p> <p>例) OPEN1,4,1 OPEN1,4,1 ?FILE OPEN ERROR</p>
LOAD	<p>カセットテープからプログラムを読みこんでいる (ローディング) 時に発生します。これはテープの最初のブロックに31個以上のエラーがあったか、または、2つのブロックの同じ位置にエラーが発見された場合に、発生します。</p>

エラーメッセージ	原因
<p>NOT INPUT FILE</p>	<p>カセットテープの場合、書きこみ(ライト)用にOPENされたファイルからデータを読みこもう(リード)しようとした場合に、このエラーが発生します。ただし、書きこみ用にOPENしたファイルをCLOSEして、その後、読みこみ(リード)用にOPENした場合は、エラーにはなりません。</p> <p>例) 1Ø OPEN1,1,1 2Ø INPUT #1,A RUN : ?NOT INPUT FILE ERROR</p>
<p>NOT OUTPUT FILE</p>	<p>カセットテープの場合、ファイルからデータを読みこむのと同時に、書きこむことができません。そのため読みこみ(リード)用にOPENされたファイルに、書きこみ(ライト)をおこなうと、このエラーが発生します。またキーボードは入力(読みこみ)のみをおこなう機器(デバイス)ですから、キーボードになにかを出力(書きこみ)することはできません。そこで、次のような場合にはエラーになります。</p> <p>例) 1Ø OPEN1,Ø (キーボードのデバイス番号=Ø) 2Ø PRINT #1,A\$ RUN ?NOT OUTPUT FILE ERROR</p>
<p>VERIFY</p>	<p>メモリーに格納されているプログラムの内容と、指定されたファイルの内容がちがっている場合、このエラーが発生します。</p>

E: 誘導関数

CBM BASICが組み込み関数として用意していない関数のうち、いくつかは組み込み関数を用いて作ることができます。以下の数式を参考にして下さい。(誤差の範囲に注意が必要です)。

関数名	計算方法
SECANT	$SEC(X) = 1 / \cos(X)$
COSECANT	$CSC(X) = 1 / \sin(X)$
COTANGENT	$COT(X) = 1 / \tan(X)$
INVERSE SINE	$ARCSIN(X) = ATN(X / \sqrt{-X * X + 1})$
INVERSE COSINE	$ARCCOS(X) = -ATN$ $(X / \sqrt{-X * X + 1}) + \pi / 2$
INVERSE SECANT	$ARCSEC(X) = ATN(X / \sqrt{X * X - 1})$
INVERSE COSECANT	$ARCCSC(X) = ATN(X / \sqrt{X * X - 1})$ $+ (SGN(X) - 1) * \pi / 2$
INVERSE COTANGENT	$ARCOT(X) = ATN(X) + \pi / 2$
HYPERBOLIC SINE	$SINE(X) = (EXP(X) - EXP(-X)) / 2$
HYPERBOLIC COSINE	$COSH(X) = ((EXP(X) + EXP(-X)) / 2$
HYPERBOLIC TANGENT	$TANH(X) = EXP(-X) / EXP(X)$ $+ EXP(-X)) * 2 + 1$
HYPERBOLIC SECANT	$SECH(X) = 2 / (EXP(X) + EXP(-X))$
HYPERBOLIC COSECANT	$CSCH(X) = 2 / (EXP(X) - EXP(-X))$
HYBERBOLIC COTANGENT	$COTH(X) = (EXP(-X) / (EXP(X)$ $- EXP(-X)) * 2 + 1$
INVERSE HYPERBOLID SINE	$ARCOSINH(X) = LOG(X + \sqrt{X * X + 1})$
INVERSE HYPERBOLIC COSINE	$ARCCOSH(X) = LOG(X + \sqrt{X * X - 1})$
INVERSE HYPERBOLIC TANGENT	$ARCTANH(X) = LOG((1 + X) / (1 - X)) / 2$
INVERSE HYPERBOLIC SECANT	$ARCSECH(X) = LOG((\sqrt{-X * X + 1}$ $+ 1) / X)$
INVERSE HYPERBOLIC COSECANT	$ARCCSCH(X) = LOG((SGN(X)$ $* \sqrt{X * X + 1}) / X)$
INVERSE HYPERBOLIC COTANGENT	$ARCCOTH(X) = LOG((X + 1) / (X - 1)) / 2$

F: カラーコントロール

VIC1001でカラーをコントロールするのに幾通りかの方法があります。

バックカラーは、スクリーンカラーとボーダーカラーを同じように、または、各自別々にコントロールすることができます。

キャラクターカラーのコントロールは、**CTRL** キーを使うことにより、また PRINTあるいはPOKEによりそれぞれの方法でおこなうことができます。

F.1. バックカラーのコントロール

バックカラーのコントロールをおこなうためには、メモリー中の (900F)₁₆ 番地 (= (36879)₁₀ 番地) にコントロール用のデータを書きこみます。書きこむデータは0から255の範囲の数値です ($0 \leq (\text{データ}) \leq 255$)。

では実際にバックカラーをコントロールするプログラム例をあげましょう。

```
例) 10 R=INT(RND(1)*255)+1.....書きこむデータを作ります。
    20 C=36879 .....(900F)16番地
    30 POKE C, R
    40 FOR I=1 TO 200 : NEXT .....次の色へ移るまでの時間を調節します。
    50 GOTO 10
    RUN
```

この場合、Rは1から255までの範囲内の数値をランダムに36879番地にかきこみ次々と色をかえていきます。この時、キャラクターカラーは変わらず、また、Rの値を2進法であらわした場合、2³ビットがON(オン)の場合、画面の文字はリバース(反転)になって出力されます。たとえば、R=139の場合、Rを2進数で表現すると、R=10001011となり2³ビットが1(=ON)なので、この場合はリバースで出力されます。

次にもうひとつカラーをコントロールするプログラム例をあげて説明しましょう。

```
例) 10 C=9*16+3+15 .....(900F)16番地
    20 X=INT(RND(1)*15) .....スクリーンカラーコードを作ります
    30 Y=INT(RND(1)*7) .....ボーダーカラーコードを作ります。
    40 R=X*16+8+Y .....スクリーン、ボーダーカラーコードをそれぞれ
                           .....の位置にセット、8はリバースにする
                           .....ため
```

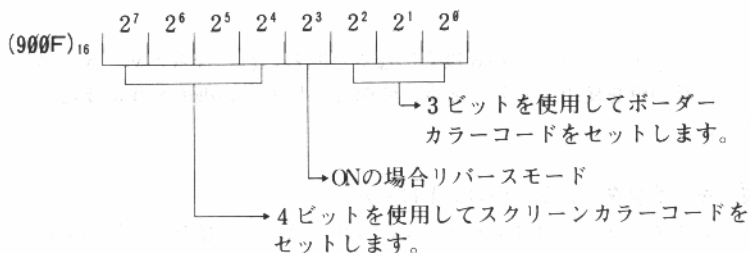
```

50 POKE C, R          ...カラーコードを書きこみます。
60 FOR I=1 TO 200 : NEXT ...次の色へ移るまでの時間を調節します。
70 GO TO 20
RUN

```

この場合、スクリーンとボーダーカラーが各々、変化していきます。

ライン・ナンバー40についてももう少し説明しますと、カラーコントロールするために、(900F)₁₆番地の1バイト (= 8ビット) を使用しますが、このバイトの各ビットは次のような意味をもちます。



〈スクリーンカラーコード〉

コード	色
0	ブラック
1	ホワイ
2	レッド
3	シアン
4	マゼン
5	グリーン
6	ブルー
7	イエロー
8	オレンジ
9	ライト オレンジ
10	ピンク
11	ライト シアン
12	ライト マゼン
13	ライト グリーン
14	ライト ブルー
15	ライト イエロー

〈ボーダーカラーコード〉

コード	色
0	ブラック
1	ホワイ
2	レッド
3	シアン
4	マゼン
5	グリーン
6	ブルー
7	イエロー

次のプログラムによりスクリーンカラーがコードによりどのような色になるか
たしかめて下さい。

```
10 INPUT A
20 POKE 36879, A * 16 + 8
30 GOTO 10
RUN
? ..... → 0 から 15 までのコードをキーボードから入力して下さい
:
:
:
```

スクリーンカラーコードが8以上になると明るい色になるため、キャラクター
カラーに濃い色を使用すると、非常に鮮明な、見やすい画面を作成することがで
きます。

F.2 キャラクターカラーのコントロール

キャラクターカラーのコントロールの方法には次の3通りの方法があります。

1) キーボードからコントロールキーと数字を組みあわせて同時に入力すること
によりコントロールできます。

たとえば

CTRL キー + **1** とおすとブラックになります。

CTRL キー + **7** とおすとブルーになります。

数字は **1** ~ **8** まで使用できます(第1部ハード編第4章第3節を参照)。

2) カラー用のコントロール文字をPRINTで出力することによりコントロールす
ることができます。

例をあげますと、

```
10 C$ = "          "    ...コントロール文字
20 C = INT(RND(1) * 8) + 1 ...カラー指定のために1~8のランダムな
                           数を作ります。
30 PRINT MID$(C$, C, 1); "A"; ...Aの文字を色々なカラーで出力します。
40 GOTO 20              (ⓂAが表示されていない部分はバックカラー
                           と同じ色のため見えませんからです。)
```

カナモードの場合は C\$ は以下のようになります。

```
10 C$=" 1234567890 "
20 C=INT(RND(1)*B)+1
30 PRINTMID$(C$,C,1);" A";
40 GOTO20
RUN
```

ライン・ナンバー10でカラーのコントロール文字を作るには、まず、クォーテーション (") をキーボードから入力したあと、**CTRL** キーを押しながら、数字の1から8まで順番に押していきます。

3) スクリーンに直接POKEを用いて文字を書いたり、かつカラーをコントロールすることができます。カラーコントロールを説明する前にまず、画面に文字を出力するために用いるスクリーンコードを、ASCII (アスキー) コードから作りだすプログラムを書いてみましょう。

```
5 INPUTX%
10 W%=X%AND63
20 IF(X%AND128)<>0THENW%=W%OR64
30 X%=W%
40 PRINTX%
```

最初にX%にスクリーンコードに変換したいASCIIコード、たとえば文字Aに対応する (65)₁₀を代入して、このプログラムを実行すると、結果のスクリーンコードX% (ライン・ナンバー30) には1が代入されます (第1部第4章第3節のキャラクター・コード表中のスクリーンコードを参照して下さい)。画面に文字をリパースにして出力させる場合は、スクリーンコードに128を加算します。

画面 (スクリーン) は、506文字分のエリアを持っています。メモリー内のどこにあるのかは、次のようにメモリーの容量により2通りあります。

- 1) RAMエリアが(1000)₁₆~(1FFF)₁₆番地および3K拡張メモリーを装備して、(0400)₁₆~(1FFF)₁₆番地の場合、スクリーンは(1E00)₁₆~(1FFF)₁₆番地にあります。
- 2) RAMエリアが(1000)₁₆から(2000)₁₆番地以降最大(7FFF)₁₆番地までである場合、(この時(0400)₁₆~(0FFF)₁₆番地のRAMの有無には無関係)、スクリーンは、(1000)₁₆~(11FF)₁₆番地にあります。

さて、この506文字の各々についてのカラーの指定ができます。スクリーンのエリア506文字に対応して (9600)₁₆ ~ (97FF)₁₆ 番地までが、それぞれのカラーのコントロールをおこないます。カラーコントロールコードは、ボーダーカラーコード (D.1参照) と同じです。

では、画面いっぱいに真赤なハートを書いていくプログラムを作ってみましょう。

```
10 SM=7680 (またはSM=4096)…前述の①、②を参照して下さい。
20 SC=38400
30 FORI=0TO505
40 M=SM+I : C=SC+I
50 POKEM,83 : POKEC,2
60 NEXT
RUN
```

ライン・ナンバー10でスクリーンエリアのスタート番地は、前述の①、②の2通りありますから、自分の機械にあわせて選択して下さい。

ライン・ナンバー20のカラー指定のためのエリアのスタート番地は、スクリーンエリアの場所が変わっても、常に一定の38400番地からはじまります。

G：サウンド

カラーをコントロールするのと同じように、ある決められた場所に数値をPOKEにより代入することにより、サウンド（音）を作ることができます。決められた場所をサウンド（音）をコントロールするレジスターといいます。レジスターは5個あり、各々に、(900A)₁₆、(900B)₁₆、(900C)₁₆、(900D)₁₆、(900E)₁₆番地のアドレスが割りあてられています。各レジスターに、次のような役割があります。

レジスターアドレス	〈機能〉
(900A) ₁₆ 番地	コードにより周波数の異なった音を出します（音の高低がつくれます）。コードはサウンドコード表を参照して下さい。
(900B) ₁₆ 番地	機能は(900A) ₁₆ 番地と同じですが、音色（トーン）がちがっています。
(900C) ₁₆ 番地	機能は上の2つと同じですが、音色（トーン）がちがっています。
(900D) ₁₆ 番地	ノイズを作ります。
(900E) ₁₆ 番地	コードにより音の振幅を決定します。（音の大小を作ります）この場合のコードは0から15までの範囲で15が一番大きい音をだします。

音の高低はサウンドコード（ $128 \leq (\text{コード}) \leq 255$ ）により指定することができます。コードの値が大きくなれば、音も高くなります。ただし例外として255の場合は、低くなってしまいます。また0をレジスターに代入すると音が消えます。

〈サウンドコード一覧表〉

音階	コード	音階	コード
ド (C)	128	ソ (G)	213
ド# (C#)	134	ソ# (G#)	215
レ (D)	141	ラ (A)	217
レ# (D#)	147	ラ# (A#)	219
ミ (E)	153	シ (B)	221
ファ (F)	159	ド (C)	223
ファ# (F#)	164	ド# (C#)	225
ソ (G)	170	レ (D)	227
ソ# (G#)	174	レ# (D#)	228
ラ (A)	179	ミ (E)	230
ラ# (A#)	183	ファ (F)	231
シ (B)	187	ファ# (F#)	232
ド (C)	191	ソ (G)	234
ド# (C#)	195	ソ# (G#)	235
レ (D)	198	ラ (A)	236
レ# (D#)	201	ラ# (A#)	237
ミ (E)	204	シ (B)	238
ファ (F)	207	ド (C)	239
ファ# (F#)	210	ド# (C#)	240

(注) この一覧表の音階は絶対音階ではありません。

H: ハイ・レゾリューション

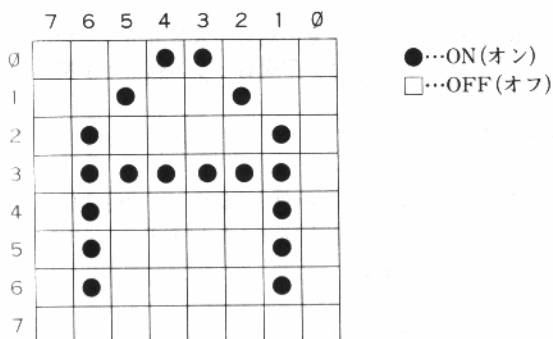
ハイ・レゾリューションとは高分解能ということです。この機能を利用すると、音符の ` ` や、 `木`、 `☒` などのキーボードにない漢字や、グラフィックなどを自由に表示することができます。

自分の好きな文字やグラフィックを作るのに、次の3通りの方法があります。

- 8×8 ドット構成
- 8×16ドット構成
- 4×8ドット、または4×16ドット構成でマルチカラーコントロールをおこなう。

まず、8×8ドットとか8×16ドットというのは、具体的にどういうことなのか説明しましょう。

画面に表示される文字は、各々、横に8個、縦に8個の点(ドット)で構成されており、それらの点をON (オン)、OFF (オフ)、すなわち、明るくしたり、暗くしたりすることにより作られています。たとえば、 `A` という文字は、



という形でつくられています。では、もし `☒` というグラフィック文字を8×8ドット構成で作るとどうなるかみてみましょう。

	7	6	5	4	3	2	1	0
0	●	●	●	●	●	●	●	●
1		●	●	●	●	●	●	
2			●	●	●	●		
3				●	●			
4				●	●			
5			●	●	●	●		
6		●	●	●	●	●	●	
7	●	●	●	●	●	●	●	●

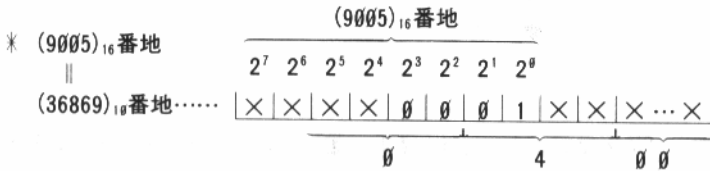
では、こうした方法で、 8×8 または 8×16 ドットの好きなパターンを作り、それを画面に表示する方法について説明しましょう。

まず、表示するパターンが 8×8 ドットか 8×16 ドットかを選択するために、 $(9003)_{16}$ 番地 (= $(36867)_{10}$ 番地) の 2^0 ビットを ON (オン) か OFF (オフ)、つまり 1 か 0 にしなければなりません。この場合、 2^0 が 0 の時は 8×8 ドット構成、1 の時は 8×16 ドット構成であることを示します。

例) $(9003)_{16}$ 番地 = $(36867)_{10}$ 番地の 2^0 ビットが 0 の場合… 8×8 ドット構成
 = 1 の場合… 8×16 ドット構成

次に指定したドット構成で作成した文字パターンコードを格納する場所を指定します。

たとえば、 $(0400)_{16}$ 番地からを使用する場合は、 $(9005)_{16}$ 番地 (= $(36869)_{10}$ 番地) の下 4 ビット (2^0 、 2^1 、 2^2 、 2^3 ビット) に次のようにセットします。



(注) ×は何がはいっていても 0 として扱います。

また、 $(3000)_{16}$ 番地からを文字パターンコードの格納番地として使用する場合は、次のようになります。



では、文字パターンコードを実際につくってみましょう。

パターン①

	7	6	5	4	3	2	1	0
0	●	●	●	●	●	●	●	●
1		●	●	●	●	●	●	
2			●	●	●	●		
3				●	●			
4				●	●			
5			●	●	●	●		
6		●	●	●	●	●	●	
7	●	●	●	●	●	●	●	●

このような場合、左端から右横へ8ドットの各ドットに値をもたせ、その合計を1バイトのデータとして、8行分、8バイトのコードを作ります。つまり●でぬりつぶしてあるところに、それに対応する値を代入して、その合計をつくりま

す。パターン①の場合、

上の行から

0	●	●	●	●	●	●	●	●
	↓	↓	↓	↓	↓	↓	↓	↓
	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	+64	+32	+16	+8	+4	+2	+1
	=255							

1		●	●	●	●	●	●	
	↓	↓	↓	↓	↓	↓	↓	↓
	0	+64	+32	+16	+8	+4	+2	0
	=126							

2			●	●	●	●		
	↓	↓	↓	↓	↓	↓	↓	↓
	0	+0	+32	+16	+8	+4	+0	+0
	=60							

3			●	●				
	↓	↓	↓	↓	↓	↓	↓	↓
	0	+0	+0	+16	+8	+0	+0	+0
	=24							

4

				●	●				
--	--	--	--	---	---	--	--	--	--

$$\downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow$$

$$\emptyset + \emptyset + \emptyset + 16 + 8 + \emptyset + \emptyset + \emptyset = 24$$

5

		●	●	●	●				
--	--	---	---	---	---	--	--	--	--

$$\downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow$$

$$\emptyset + \emptyset + 32 + 16 + 8 + 4 + \emptyset + \emptyset = 60$$

6

	●	●	●	●	●	●			
--	---	---	---	---	---	---	--	--	--

$$\downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow$$

$$\emptyset + 64 + 32 + 16 + 8 + 4 + 2 + \emptyset = 126$$

7

●	●	●	●	●	●	●	●		
---	---	---	---	---	---	---	---	--	--

$$\downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow$$

$$128 + 64 + 32 + 16 + 8 + 4 + 2 + \emptyset = 255$$

各行 (0~7) の右側に書かれている合計値が文字パターンコードになります。
 ■を表示するためには、この8個のそれぞれの合計値 (8バイト使用します) が必要になります。

文字パターンコードの作り方の例をいくつかあげてみましょう。

例1) 音符♪の場合

	7	6	5	4	3	2	1	0	
0				●	●				→ $\emptyset + \emptyset + \emptyset + 16 + 8 + \emptyset + \emptyset + \emptyset = 24$
1				●		●			→ $\emptyset + \emptyset + \emptyset + 16 + \emptyset + 4 + \emptyset + \emptyset = 20$
2				●		●			→ $\emptyset + \emptyset + \emptyset + 16 + \emptyset + 4 + \emptyset + \emptyset = 20$
3				●			●		→ $\emptyset + \emptyset + \emptyset + 16 + \emptyset + \emptyset + 2 + \emptyset = 18$
4			●	●					→ $\emptyset + \emptyset + 32 + 16 + \emptyset + \emptyset + \emptyset + \emptyset = 48$
5		●	●	●					→ $\emptyset + 64 + 32 + 16 + \emptyset + \emptyset + \emptyset + \emptyset = 112$
6		●	●						→ $\emptyset + 64 + 32 + \emptyset + \emptyset + \emptyset + \emptyset + \emptyset = 96$
7									→ $\emptyset + \emptyset + \emptyset + \emptyset + \emptyset + \emptyset + \emptyset + \emptyset = 0$

例2) '▲'の場合

	7	6	5	4	3	2	1	0	
0									→ 0+0+0+0+0+0+0+0+0=0
1				●	●				→ 0+0+0+16+8+0+0+0+0=24
2			●	●	●	●			→ 0+0+32+16+8+4+0+0+0=60
3		●	●	●	●	●	●		→ 0+64+32+16+8+4+2+0=126
4	●	●	●	●	●	●	●	●	→ 128+64+32+16+8+4+2+1=255
5				●	●				→ 0+0+0+16+8+0+0+0+0=24
6			●			●			→ 0+0+32+0+0+4+0+0+0=36
7		●					●		→ 0+64+0+0+0+0+0+2+0=66

例3) 8×16ドット構成の文字2つを使って漢字の'君'を作ってみました。
 まず左半分そして右半分です。

	7	6	5	4	3	2	1	0	
0									→ 0
1									→ 0
2				●	●	●	●	●	→ 15
3						●			→ 4
4				●	●	●	●	●	→ 31
5						●			→ 4
6				●	●	●	●	●	→ 15
7				●		●			→ 4
8				●	●	●	●	●	→ 15
9				●		●			→ 20
10				●		●			→ 20
11				●		●	●	●	→ 23
12									→ 0
13									→ 0
14									→ 0
15									→ 0

	7	6	5	4	3	2	1	0	
0									→ 0
1									→ 0
2	●	●	●	●					→ 240
3				●					→ 16
4	●	●	●	●	●				→ 248
5				●					→ 16
6	●	●	●	●					→ 240
7									→ 0
8	●	●	●	●					→ 240
9				●					→ 16
10				●					→ 16
11	●	●	●	●					→ 240
12									→ 0
13									→ 0
14									→ 0
15									→ 0

8×16ドット構成の場合は、1つの文字を作るために16バイト必要になります。

以上の例1.2.3.で文字パターンコードの作り方がわかったところで、これらを指定した格納番地に格納しましょう。

文字は8×8ドット構成で、格納スタート番地は(1400)₁₆番地とします。格納する文字パターンコードは、例1、例2で作成したものを使用します。

- ① 8×8ドット構成であることを指定します。

POKE36867, PEEK(36867) AND 254

これで(9003)₁₆番地 (= (36867)₁₆番地) の2ndビットを0にして8×8ドット構成であることが指定されます。なお8×16ドット構成の指定はPOKE36867、PEEK(36867) OR 1でおこないます。

- ② 格納スタート番地が(1400)₁₆番地、ビデオRAMのスタート番地が(1E00)₁₆番地であることを指定します。

POKE36869,253

POKE36866, PEEK(36866) OR 128

注) 第3部のMPC 6560ビデオ・インターフェース・チップの項を参照して下さい。

- ③ 文字パターンを格納します。

例1の文字パターンを(1400)₁₆番地から8バイト分格納します。

POKE5120,24

POKE5121,20

POKE5122,20

POKE5123,18

POKE5124,48

POKE5125,112

POKE5126,96

POKE5127,0

次に例2の文字パターンをつづいて格納します。

```
POKE5128,0
POKE5129,24
POKE5130,60
POKE5131,126
POKE5132,255
POKE5133,24
POKE5134,36
POKE5135,66
```

④次にこれらの文字パターンを画面に表示するには、キーボードの[@] (アットマーク)を押して下さい。すると例1の文字パターンが画面にあらわれます。次に[A]を押して下さい。例2の文字パターンが表示されます。格納した文字パターンは、格納順にスクリーンコードと(スクリーンコード表参照)対応します。最大512種類までの文字パターンを作成することができます。

⑤①～③までを実際にプログラムにつくってみましょう。

```
1 POKE51,0:POKE52,19
2 POKE55,0:POKE56,19
3 CLR
4
5 FORI=1TO16
10 READ A
20 POKE1023+4096+I,A
30 NEXT
40 GOTO100
50 DATA24,20,20,18,48,112,96,0
60 DATA0,24,60,126,255,24,36,66
100 POKE36867,PEEK(36867)AND254
105 POKE36866,PEEK(36866)OR128
110 POKE36869,253
140 END
```

BASICプログラムのテキストエリアは、
(1300)₁₆番地までであることを宣言しています。

プログラムをうちおわったら、まちがっていないかたしかめて下さい。

RUN [RETURN] と入力して下さい。

[@] キーをおすと画面に♪があらわれます。

次に[A]キーをおすと▲があらわれます。

このプログラムでは、2種類のパターンしか作っていませんが、これを参考に
して、たくさんのあなただけのパターンを作ってみてください。

なお、このプログラムを終わらせるには [RUN
STOP] キー + [RESTORE] キー
押して下さい。

第3部



コンポーネント データカタログ編

コンポーネント・データカタログ

1. MPS6500マイクロプロセッサ
2. MPS6560ビデオ・インターフェイス・チップ (VIC)
3. MPS6522汎用インターフェイスアダプタ
4. MPS2364スタティックROM

MPS-6500マイクロプロセッサ

概説

MPS6500シリーズのマイクロプロセッサは、ソフトウェア・コンパチブルなマイクロプロセッサ・ファミリーの最初のもので、この製品のファミリーは、ソフトウェア・コンパチブルで、アドレス可能なメモリー範囲、割込入力、クロック発振チップおよびドライブの選択が可能です。MPS6500グループのマイクロプロセッサはすべて、グループ内でソフトウェア・コンパチブルで、M6800製品とバス・コンパチブルです。

ファミリー内の5つのマイクロプロセッサは、クロック発振とドライブを持ち、4つのマイクロ・プロセッサは、外部クロックによりドライブされます。内部クロックは高いパフォーマンスと低コストを目的とし、タイム・ベースにシングル・フェーズで、クリスタルまたはRC入力を用いています。外部クロックは、マルチ・プロセッサ・システムに適用され、タイミングの制御が必要な場合用います。マイクロ・プロセッサはいずれのバージョンでも最大周波数が1MHzか2MHzが可能です（"A" は製品番号の添字です）。

特長

- 5V単一電源
- Nチャンネル・シリコン・デプレッションロード
- 8ビット並列処理
- 56インストラクション
- 10進および2進演算
- 13アドレス・モード
- プログラマブル・スタック・ポインタおよび可変長スタック
- 任意のメモリー使用可能
- 1または2MHzで動作
- パイプライン構造

信号の説明

クロック (ϕ_1 、 ϕ_2)

MPS651Xは、Vcc電圧レベルの非重複2相クロックが必要です。

MPS651Xのクロックは、内部クロック・ジェネレータにより与えられます。クロックの周波数は、外部から制御できます。

アドレスバス (AB0-AB15)

出力は、TTLコンパチブルで、標準TTL 1個と130pFの負荷をドライブできます。

データバス (DB0-DB7)

データバス用に8ピンあります。双方向性で、マイクロプロセッサからデバイス、ペリフェラルへ、またはその逆のデータ転送をおこないます。出力はスリーステートバッファで、標準TTL 1個と130pFの負荷をドライブできます。

データバス・イネーブル (DBE)

TTLコンパチブル入力であり、スリーステートデータの出力バッファを制御し、高レベルで、バスドライバーを動作可能にします。通常は、DBEは、 ϕ_2 クロックでドライブされるので、マイクロプロセッサは ϕ_1 の間だけ入力が可能です。リードサイクルの間、データバスドライバは内部的にディスエーブルの状態であり、実質的には、オープン回路となります。外部からデータバスドライバをディスエーブルにするにはDBEは、低レベルに保持する必要があります。

レディ (RDY)

入力信号であり、書込みサイクル以外のすべてのサイクルで、マイクロプロセッサを単一サイクルにします。 ϕ_1 の間に低レベルに変化すると、現在フェッチしているカレント・アドレスをアドレスラインに出力し、ホルト状態にします。この状態は、その後の ϕ_2 まで続き、レディ信号は低レベルを保ちます。したがって、マイクロプロセッサは、最高2サイクルのダイレクト・メモリ・アクセス (DMA) と同じように、低スピードのPROMとのインタフェースも可能です。書込みサイクルの間、レディ信号が低レベルの場合は、次の読み動作まで、書込みは無視されます。

インタラプト・リクエスト (\overline{IRQ})

TTLコンパチブル信号であり、入力されるとマイクロプロセッサ内部で、インタラプト・シーケンスが開始します。マイクロプロセッサは割込の入る前に実行していた命令を完結させます。同時に、ステータス・コード・レジスタの割込マスクビットをチェックします。割込マスクフラグがセットされていないなら、マイクロプロセッサは、インタラプト・シーケンスを実行しはじめます。プログラムカウンタ、ステータス・レジスタは、スタックに待避されます。次にマイクロプロセッサは、割込マスクフラグをハイにして、それ以外のインタラプトには応じません。その後、プログラム・カウンタの下位をFFFE番地から、上位をFFFF番地からロードして、そのアドレスに制御を移します。RDY信号は、インタラプトの受けつけ時には高レベルでなくてはなりません。ワイヤOR操作には3KΩの外部抵抗をういます。

ノンマスクابل・インタラプト (\overline{NMI})

この立下がり入力により、マイクロプロセッサ内部で、ノンマスクابل・インタラプト・シーケンスが発生します。

\overline{NMI} は無条件割込みです。現在実行中の命令を完了し、割込マスクフラグの状態に関係なく \overline{IRQ} の動作シーケンスが開始します。プログラム・カウンタにロードされるアドレスは、下位、上位、それぞれFFFA番地、FFFB番地にあり、そのアドレスにプログラム制御を移します。そこにロードしてある命令により、メモリー内のノンマスクابل・インタラプト・ルーチンにジャンプします。ワイヤOR操作には、Vccに対して3kΩの外部抵抗をういます。

\overline{IRQ} および \overline{NMI} 入力は、 ϕ_2 でのハードウェア・インタラプト・ラインであり、現在実行中の命令を完了してから、 ϕ_1 の立下がりにより検出されます。

セット・オーバフロー・フラグ (SO)

この立下がり入力により、ステータス・コード・レジスタのオーバフロービットをセットします。この信号は ϕ_1 の立下がりにより検出されます。

SYNC

出力ラインであり、マイクロプロセッサがOP CODEのフェッチを実行している間のサイクルを識別します。SYNCラインはOP CODEのフェッチの ϕ_1 の間に高レベルになり、そのサイクルの間、高レベルを保持します。SYNC信号が高レベルになろうとする ϕ クロックの間に、RDY信号が落ちると、プロセッサはその状態でストップし、RDY信号が高レベルになるまで、その状態を保ちます。したがって、SYNC信号はRDYを制御して、一命令の実行に使用されます。

リセット ($\overline{\text{RES}}$)

この入力、マイクロプロセッサをパワーダウン状態からのリセット、スタートに使われます。この信号が低レベルを保持している間は、マイクロプロセッサからのまたはマイクロプロセッサへの書きこみは禁止されます。立上がり入力があると、マイクロプロセッサはすぐに、リセット・シーケンスを開始します。

システム初期化の6クロック・サイクルの後、割込マスクフラグがセットされ、メモリーのFFFC番地およびFFFD番地から、プログラム・カウンタをロードします。これが、プログラム制御の開始番地になります。パワーアップ・ルーチンでVccが4.75ボルトになった後、リセット信号は、少なくとも2クロック・サイクルの間、低レベルを保持されなくてはなりません。この間にR/Wおよび(SYNC)信号が有効になります。

リセット信号が、その2クロック・サイクルの後、高レベルである間、マイクロ・プロセッサは、上に述べた正常なリセット動作を続行します。

ユーザコード	オペレーション	イニシャル		デビッド		アドレス		オペレーション		レジスタ		アドレス		レジスタ		オペレーション		レジスタ		ユーザコード	
		OP	N	OP	N	OP	N	OP	N	OP	N	OP	N	OP	N	OP	N	OP	N		
L D X	M → X	(1)	AJ	2	2	AE	4	3	A6	3	2										
L D Y	M → Y	(1)	AB	2	2	AC	4	3	A4	3	2										
L S R	0 → R	(1) → C	4E	6	3	46	5	2	4A	2	1										
N O P	NO OPERATION		EA	2	1																
O R A	AVM → A		04	2	2	0D	4	3	05	3	2										
P H A	A → M	S → I → S	48	3	1																
P H P	P → M	S → I → S	08	3	1																
P L A	S → I → S	M → A	68	4	1																
P L P	S → I → S	M → P	28	4	1																
R O L	0 → C →		ZE	6	3	26	5	2	2A	2	1										
R O R	C → 7	0 →	6E	6	3	66	5	2	6A	2	1										
R T I	RTN INT		48	6	1																
R T S	RTN SUB		68	6	1																
S B C	A → M → C → A	(1)	E9	2	2	ED	4	3	E5	3	2										
S E C	I → C		88	2	1																
S E D	I → D		F8	2	1																
S E I	I → I		78	2	1																
S T A	A → M		8D	4	3	85	3	2													
S T X	X → M		BE	4	3	86	3	2													
S T Y	Y → M		BC	4	3	84	3	2													
T A X	A → X		AA	2	1																
T A Y	A → Y		AB	2	1																
T S X	S → X		8A	2	1																
T X A	X → A		8A	2	1																
T X S	X → S		9A	2	1																
T Y A	Y → A		98	2	1																

注意

1. ページバウンダリにかかった場合はNに1加算されます。
2. 同一ページ内でのジャンプの場合はNに1加算されます。
3. キャリはゼロと一致しません。
4. 10進モードでは、Zフラグは不正。
結果ゼロのチェックはアキュムレータでおこないます。

X インデックス
Y インデックス
A アキュムレータ
M 実効アドレスの内容
Ms スタック・ポインタの内容
+ 加算
- 減算

< 論理積
> 論理和
V 排他的論理和
L レジスタの値あり
- レジスタの値なし
M7 メモリーのビット7
M6 メモリーのビット6

N # サイクル数
バイト数

命令セット—アルファベット順

ADC	メモリーとキャリーをアキュムレータに加算します。
AND	メモリーとアキュムレータの論理積をとります。
ASL	左へ1ビットシフト(メモリおよびアキュムレータ)します。
BCC	キャリーがOFFの場合、ブランチします。
BCS	キャリーがONの場合、ブランチします。
BEQ	結果がゼロの場合、ブランチします。
BIT	アキュムレータの内容により、メモリーのビットテストをおこないます。
BMI	結果がマイナスの場合、ブランチします。
BNE	結果がゼロ以外の場合、ブランチします。
BPL	結果がプラスの場合、ブランチします。
BRK	強制ブレイクします。
BVC	オーバーフローがOFFの場合、ブランチします。
BVS	オーバーフローがONの場合、ブランチします。
CLC	キャリーフラグをOFFにします。
CDL	10進モードをクリアします。
CLI	割込禁止フラグをクリアします。
CLV	オーバーフローフラグをクリアします。
CMP	メモリーとアキュムレータを比較します。
CPX	メモリーとインデックスレジスタXを比較します。
CPY	メモリーとインデックスレジスタYを比較します。
DEC	メモリーから1減算します。
DEX	インデックスレジスタXから1減算します。
DEY	インデックスレジスタYから1減算します。
EOR	メモリーとアキュムレータの「排地的論理和」をとります。
INC	メモリーに1加算します。
INX	インデックスレジスタXに1加算します。
INY	インデックスレジスタYに1加算します。
JMP	新しいロケーションにジャンプします。
JSR	リターンアドレスをセーブして、新しいロケーションにジャンプします。
LDA	メモリーをアキュムレータにロードします。
LDX	メモリーをインデックスレジスタXにロードします。
LDY	メモリーをインデックスレジスタYにロードします。
LSR	右へ1ビットシフト(メモリーおよびアキュムレータ)します。
NOP	無操作
ORA	メモリーとアキュムレータの論理和をとります。

PHA	アキュムレータをスタックにプッシュします。
PHP	プロセッサ・ステータスをスタックにプッシュします。
PLA	スタックからアキュムレータにプルします。
PLP	スタックからプロセッサ・ステータスにプルします。
ROL	左へ1ビット回転します(メモリーおよびアキュムレータ)。
ROR	右へ1ビット回転します(メモリーおよびアキュムレータ)。
RTI	インタラプトからリターンします。
RTS	サブルーチンからリターンします。
SBC	アキュムレータからメモリーおよびポローを減算します。
SEC	キャリフラグをセットします。
SED	10進モードにセットします。
SEI	割込禁止フラグをセットします。
STA	アキュムレータをメモリーにストアします。
STX	インデックスXをメモリーにストアします。
STY	インデックスYをメモリーにストアします。
TAX	アキュムレータをインデックスXに移します。
TAY	アキュムレータをインデックスYに移します。
TSX	スタックポインタをインデックスXに移します。
TXA	インデックスXをアキュムレータに移します。
TXS	インデックスXをスタックポインタに移します。
TYA	インデックスYをアキュムレータに移します。

アドレッシング・モード

アキュムレータ・アドレッシング

1バイト命令で、アキュムレータに対する操作です。

イミディエイト・アドレッシング

イミディエイト・アドレッシングでは、命令の2バイト目にオペランドがあり、メモリーのアドレスは必要ない。

アブソリュート・アドレッシング

アブソリュート・アドレッシングでは、命令の2バイト目が実効アドレスの下位8ビットで、3バイト目が上位8ビットになります。アブソリュート・アドレッシングモードでは、アドレス可能メモリの65Kバイト全体のアクセスが可能です。

ゼロ・ページ・アドレッシング

ゼロページ命令では、上位アドレスバイトを0として、命令の2バイト目のみをフェッチすることにより、コードと実行時間の節約をします。ゼロページを注意深く使えば、コードの効率が非常に増します。

インデックス・ゼロ・ページ・アドレッシング (X、Yインデックス)

このアドレッシングは、インデックス・レジスタに関連して使われ、「ゼロページのX」または「ゼロページのY」として参照されます。実効アドレスは、インデックスレジスタの内容に、第2バイトを加えることにより計算されます。「ゼロページ」アドレッシングの形をとっているため、第2バイトの内容は、ゼロページを参照します。さらに、メモリの上位8ビットにはキャリは加えられず、ページ・パラランダリーの交叉は起きません。

インデックス・アブソリュート・アドレッシング (X、Yインデックス)

このアドレッシングは、X、Yインデックスレジスタに関して使用され、「アブソリュートのX」または「アブソリュートのY」として参照されます。実行アドレスは、X、Yの内容に、命令の第2、第3バイトの内容であるアドレスを加えることにより計算されます。このモードでは、インデックス・レジスタにはインデックスまたはカウント値を、命令には、ベース・アドレスを使用します。このタイプのインデックスでは、すべてのロケーションを参照でき、インデックスは、複数のフィールドを指すことができるので、コーディングと実行時間を減らすことができます。

インプライド・アドレッシング

インプライド・アドレッシング・モードでは、オペランドが持っているアドレスは、命令のオペレーションコード内にあります。

リラティブ・アドレッシング

リラティブ・アドレッシングは、ブランチ命令のみで使われ、条件ブランチに対して、デスティネーションを与えます。命令の第2バイトは、プログラム・カウンタが、次の命令をさした時、その下位8ビットの内容に加えられたオフセットとして、オペランドになります。オフセットの範囲は、次の命令から、-128から+127バイトとなります。

インデックス・インダイレクト・アドレッシング

インデックスは、インダイレクト・アドレッシング (インダイレクト、X) では、命令の第2バイトは、メインデックス・レジスタの内容に、キャリを無視して加算されます。その結果、ページ・ゼロのメモリーロケーションを指し、

その内容は、実効アドレスの下位8ビットになります。ページ・ゼロ内の次のメモリーロケーションの内容は、実効アドレスの上位8ビットになります。実効アドレスの上下バイトを指定するメモリーロケーションは、いずれもページ・ゼロになくってはなりません。

インダイレクト・インデックス・アドレッシング

インダイレクト・インデックス・アドレッシング (インダイレクト、Y) では、命令の第2バイトは、ページ・ゼロのメモリーロケーションを指しています。そのメモリーロケーションの内容は、Yインデックス・レジスタの内容と加算して、実効アドレスの下位8ビットになります。キャリーは、ページ・ゼロ内の次のメモリーロケーションの内容と加算され、実効アドレスの上位8ビットになります。

アブソリュート・インダイレクト

命令の第2バイトの内容は、メモリーロケーションの下位8ビットになります。上位8ビットは、命令の第3バイトになります。そのメモリーロケーションの内容が実効アドレスの下位バイトであり、次のメモリーロケーションの内容が上位バイトになり、それが16ビットのプログラム・カウンターにロードされます。

最大定格

項 目	記 号	定 格	単 位
電源電圧	V_{CC}	$-0.3 \sim +7.0$	V_{dc}
入力電圧	V_{IN}	$-0.3 \sim +7.0$	V_{dc}
動作温度	T_A	$0 \sim 70$	$^{\circ}C$
保存温度	T_{STG}	$-5 \sim +150$	$^{\circ}C$

●注意●

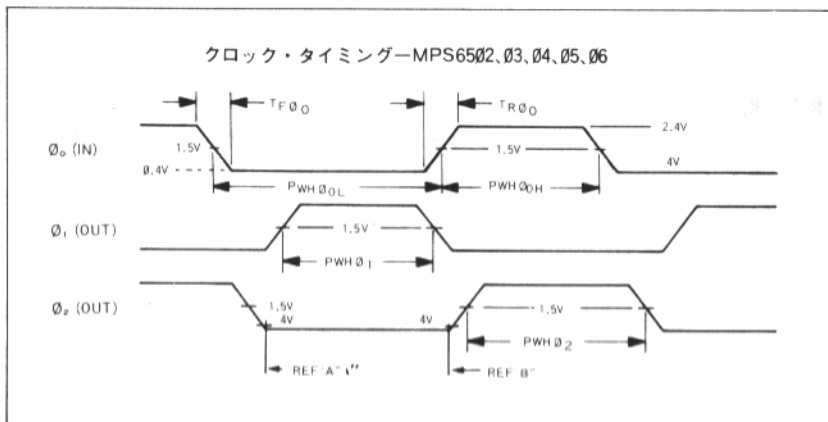
このチップは、静電気あるいは電界に対して保護されていますが、最大定格を超える電圧をかけないよう注意して下さい。

電気特性 (V_{CC}=5.0V±5%、V_{SS}=0、T_A=25°C)

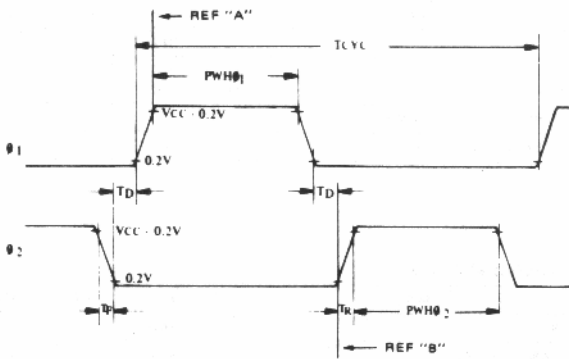
φ₁ φ₂はMPS6512、13、14、15に適用、φ₀(in)はMPS6502、03、04、05、06に適用

記号	項目	最小	標準	最大	単位	テスト条件
V _{IN}	"H"入力電圧	V _{SS} +2.4 V _{CC} -0.2		V _{CC} V _{CC} +0.25	V _{dc}	ロジック、φ ₀ (in) φ ₁ 、φ ₂
V _{IL}	"L"入力電圧	V _{SS} -0.3 V _{SS} -0.3		V _{SS} +0.4 V _{SS} +0.2	V _{dc}	ロジック、φ ₀ (in) φ ₁ 、φ ₂
V _{INT}	"H"ヌレシュオールド電圧	V _{SS} +2.0			V _{dc}	RES、NMI、RDY、IRQ、Data、S.O.
V _{ILT}	"L"ヌレシュオールド電圧			V _{SS} +0.8	V _{dc}	RES、NMI、RDY、IRQ、Data、S.O.
I _{IN}	入力リーク電流			2.5 100 10.0	μA μA μA	(V _{IN} =0から5.25V、V _{CC} =0) ロジック (RDY、S.O以外) φ ₁ 、φ ₂ φ ₀ (in)
I _{TSI}	オフ状態スリーステート入力電流			10	μA	(V _{in} =0.4から2.4V、V _{CC} =5.25V) データライン
V _{OH}	"H"出力電圧	V _{SS} +2.4			V _{dc}	(I _L OAD=-100μAdc、V _{CC} =4.75V) SYNC、Data、A ₀ -A ₁₅ 、R/W
V _{OL}	"L"出力電圧			V _{SS} +0.4	V _{dc}	(I _L OAD=1.6mAdc、V _{CC} =4.75V) SYNC、Data、A ₀ -A ₁₅ 、R/W
P _D	消費電力		.25	.70	W	
C	容量				pF	(V _{IN} =0、T _A =25°C、f=1MHz)
C _{IN}				10 15		ロジック データ
C _{OUT}				12		A ₀ -A ₁₅ 、R/W、SYNC
C _{φ0(m)}				50		φ ₀ (in)
C _{φ1}			30	50		φ ₁
C _{φ2}			50	80		φ ₂

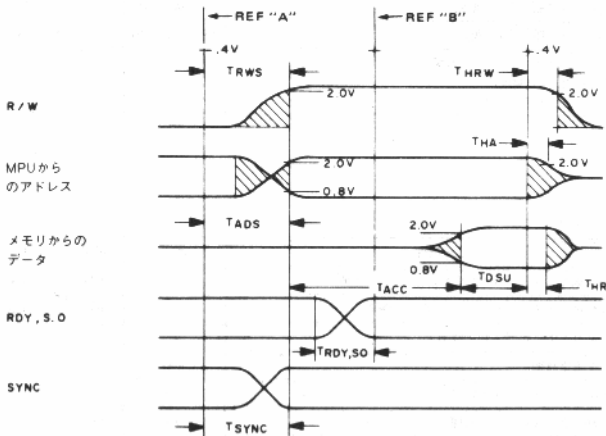
注意：IRQとNMIは、3Kのプルアップ抵抗が必要です。



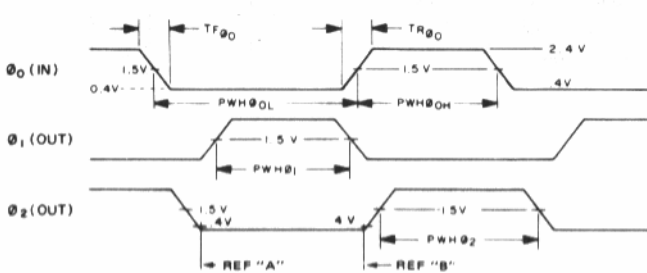
クロック・タイミング—MPS5612, 13, 14, 15



リードタイミング



ライトタイミング



注：REFはクロックのレファレンスポイントの意味です。

1MHz タイミング

クロック タイミング—MPS6512, 13, 14, 15

記号	特性	最小	標準	最大	単位
T_{CYC}	サイクルタイム	1000			nsec
PWH $\phi 1$	クロックパルス幅 ($V_{CC} = 0.2V$ で測定)	$\phi 1$	430		nsec
PWH $\phi 2$		$\phi 2$	470		
T_F	立下がり時間 ($0.2V \sim V_{CC} = 0.2V$ で測定)			25	nsec
T_D	クロックの遅延時間 ($0.2V$ で測定)	0			nsec

クロック タイミング—MPS6502, 03, 04, 05, 06

記号	特性	最小	標準	最大	単位
T_{CYC}	サイクルタイム	1000			ns
PWH ϕ_0	ϕ_0 (IN)パルス幅(1.5Vで測定)	460		520	ns
TR ϕ_0 , TF ϕ_0	ϕ_0 (IN)立上がり立下がり時間			10	ns
T_D	クロック遅延時間(1.5Vで測定)	5			ns
PWH ϕ_1	ϕ_1 (OUT)パルス幅(1.5Vで測定)	PWH $\phi_{OL} - 20$		PWH ϕ_{OL}	ns
PWH ϕ_2	ϕ_2 (OUT)パルス幅(1.5Vで測定)	PWH $\phi_{OH} - 40$		PWH $\phi_{OH} - 10$	ns
T_R , T_F	ϕ_1 (OUT)立上がり 立下がり時間 (8Vから2.0Vで測定) (ロード=30PF+1TTL)			25	ns

リード/ライト タイミング

記号	特性	最小	標準	最大	単位
T_{RWS}	MPS6500からのリード/ライトセットアップ時間		100	300	ns
T_{ADS}	MPS6500からのアドレスセットアップ時間		100	300	ns
T_{ACC}	メモリー・リードアクセス時間			575	ns
T_{DSU}	データ安定時間	100			ns
T_{HR}	読込時、データホールド時間	10			ns
T_{HW}	書込時、データホールド時間	30	60		ns
T_{MDS}	MPS6500からのデータセットアップ時間		150	200	ns
T_{RDY}	RDY, S.O.セットアップ時間	100			ns
T_{SYNC}	MPS6500からのSYNCセットアップ時間			350	ns
T_{HA}	アドレスホールド時間	30	60		ns
T_{HRW}	R/Wホールド時間	30	60		ns

2MHzタイミング

クロックタイミング—MPS6512, 13, 14, 15, 16

記号	特性	最小	標準	最大	単位
T_{CYC}	サイクルタイム	500			nsec
PWH $\phi 1$ PWH $\phi 2$	クロックパルス幅 ($V_{CC} - 0.2V$ で測定)	$\phi 1$ 215 $\phi 2$ 235			nsec
T_F	立下がり時間(0.2V $\sim V_{CC} - 0.2V$ で測定)			12	nsec
T_D	クロック遅延時間(0.2Vで測定)	0			nsec

クロックタイミング—MPS6502, 03, 04, 05, 06

記号	特性	最小	標準	最大	単位
T_{CYC}	サイクルタイム	500			ns
PWH ϕ_0	$\phi_{0(IN)}$ パルス幅(1.5Vで測定)	240		260	ns
TR ϕ_0 , TF ϕ_0	$\phi_{0(IN)}$ 立上がり立下がり時間			10	ns
T_D	クロック遅延時間(1.5Vで測定)	5			ns
PWH ϕ_1	$\phi_{1(OUT)}$ パルス幅(1.5Vで測定)	PWH $\phi_{ol} - 20$		PWH ϕ_{ol}	ns
PWH ϕ_2	$\phi_{2(OUT)}$ パルス幅(1.5Vで測定)	PWH $\phi_{oH} - 40$		PWH $\phi_{oH} - 10$	ns
T_R , T_F	$\phi_{1(OUT)}$ $\phi_{2(OUT)}$ 立下がり時間 (0.8Vから2.0で測定) (ロード=30pF+1TTL)			25	ns

リード/ライトタイミング

記号	特性	最小	標準	最大	単位
T_{RWS}	MPS6500からのリード/ライトセットアップ時間		100	150	ns
T_{ADS}	MPS6500Aからのアドレスセットアップ時間		100	150	ns
T_{ACC}	メモリーリードアクセス時間			300	ns
T_{DSU}	データ安定時間	50			ns
T_{HR}	読込時データホールド時間	10			ns
T_{HW}	書込時データホールド時間	30	60		ns
T_{MDS}	MPS6500Aからのデータセットアップ時間		75	100	ns
T_{RDY}	RDY, S.O. セットアップ時間	50			ns
T_{SYNC}	MPS6500AからのSYNCセットアップ時間			175	ns
T_{HA}	アドレスホールド時間	30	60		ns
T_{HRW}	R/Wホールド時間	30	60		ns

ピン パッケージ

MPS6502			
V _{ss}	1	40	RES
RDY	2	39	φ ₂ (OUT)
φ ₁ (OUT)	3	38	S 0
IRQ	4	37	φ ₀ (IN)
N.C.	5	36	N.C.
NMI	6	35	N.C.
SYNC	7	34	R/W
V _{cc}	8	33	DB0
AB0	9	32	DB1
AB1	10	31	DB2
AB2	11	30	DB3
AB3	12	29	DB4
AB4	13	28	DB5
AB5	14	27	DB6
AB6	15	26	DB7
AB7	16	25	AB15
AB8	17	24	AB14
AB9	18	23	AB13
AB10	19	22	AB12
AB11	20	21	V _{ss}

特徴

- 65Kバイト アドレスラブル
- $\overline{\text{IRQ}}$ 割込
- $\overline{\text{NMI}}$ 割込
- 内蔵クロック
 - TTLレベル単相入力
 - RCベースタイマ入力
 - クリスタルベースタイマ入力
- 同期信号
 - (命令実行に用いられる)
- レディ信号
 - (1サイクル実行に用いられる)
- サポートチップに対し2相クロック出力

MPS-6560

ビデオ・インターフェイス・チップ(VIC)

概要

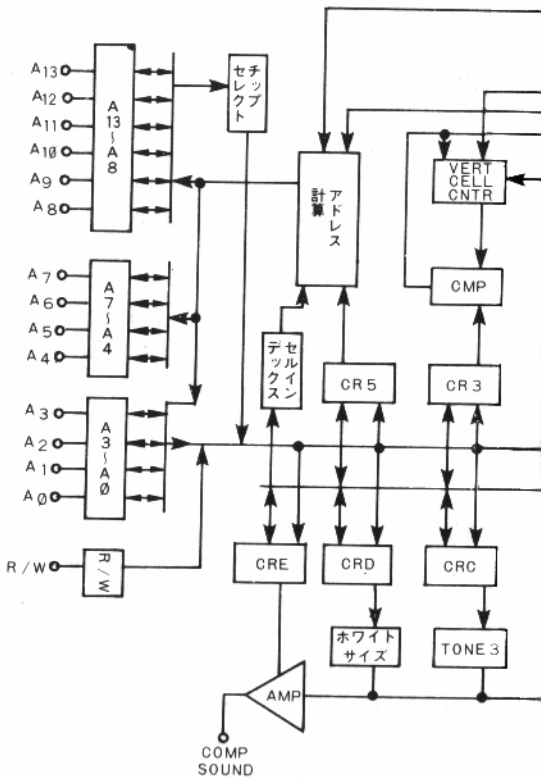
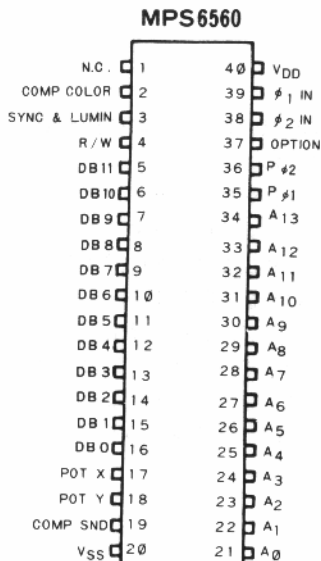
MPS6560ビデオ・インターフェイス・チップ (VIC) は、低コストのCRT端末、生医学的なモニタ、コントロール・システム・ディスプレイ、アーケード/ホームビデオ・ゲーム、ホーム・コンピューターなどのようなカラー・ビデオ・グラフィック応用製品のために設計されています。カラーでプログラマブルなキャラクターグラフィックを高分解能で生成する回路を提供します。また、VICは、ビデオ・ゲームに用いられるA/Dコンバータと音声効果を提供します。

チップの音声システムには、3つの独立で、プログラマブルな音声発生器、ホワイト・ノイズ発生器、振幅変調器があります。したがって、スクリーンのリフレッシュの間のCPUの空時間をなくし、プログラマブルなスイッチにより、組合せ、非組合せの操作をすることができます。MPS6560は、2種類のモードで、カラー操作します。

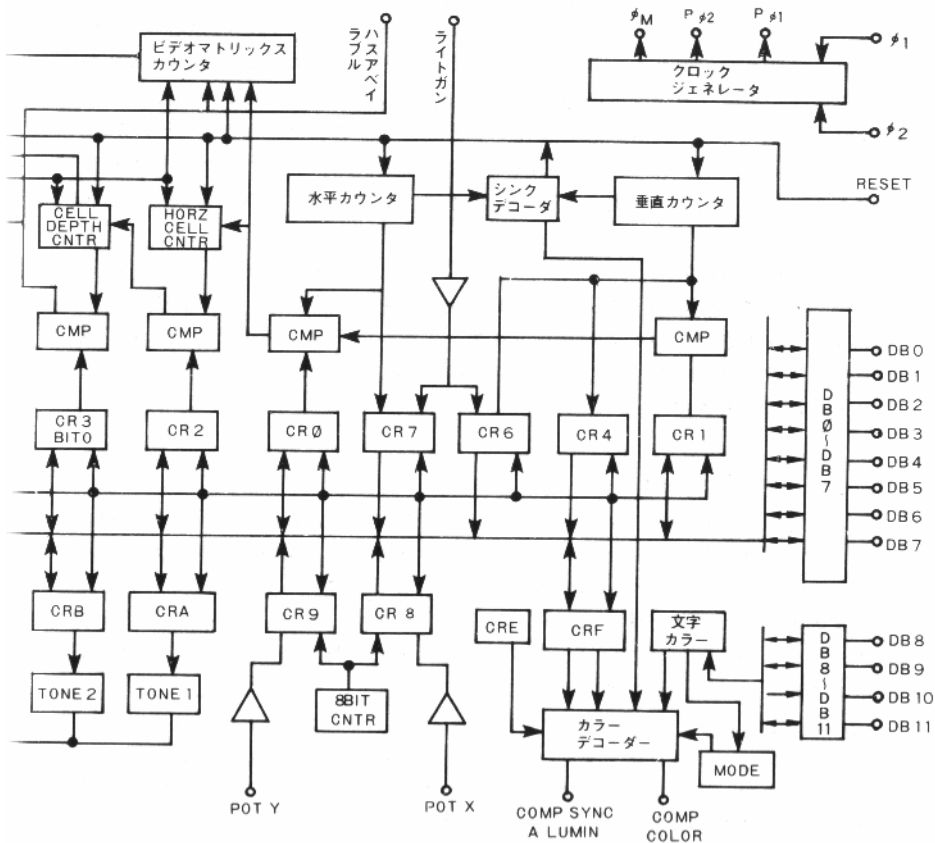
特長

- 16Kバイトアドレス空間に完全拡張可能
- マスク・プログラマブル同期ジェネレーション
- オン・チップ・カラージェネレーション
- 600のプログラマブルでムーバブルなバックグラウンド・ロケーション
- 192×200のスクリーン・グリッド
- キャラクタ・サイド2種
- オン・チップ・音声システム
- オン・チップ・DMA、アドレス・ジェネレーション
- 16アドレス可能コントロール・レジスタ
- ターゲット・ゲームにはライトガン/ペン使用可能

ピン配置図



ブロック図



信号の説明

アドレスバス (A₀—A₁₃)

14ビットのアドレスバス (A₀—A₁₃) は双方向性です。P_φ = 1の間、アドレス・ピンは入力モードであり、マイクロプロセッサは、VICの16のコントロール・レジスタのいずれもアクセス可能です。アドレスバスの上位ピン (A₈からA₁₃) は、チップセレクトの役割をします。実際のチップ・セレクトは、A₁₃ = A₁₁ = A₁₀ = A₉ = A₈ = 0でA₁₂ = 1の場合で、これは、VICのチップ・セレクト・アドレスが、16進の1000の場合になります。アドレスバスの下位4ビット (A₀からA₃) は、入力アドレスによるコントロールレジスタ選択に使われず。

P_φ = 1の間は、VICアドレスピンは、データ (キャラクタ・ポインタ、あるいは、キャラクタ・セル) がフェッチされている場合は、出力モードになります。VICはフェッチされているメモリー・ロケーションのアドレスを出力します。VICのアドレスは、P_φのエッジの立ち上がりから、50nsで有効になり、P_φのエッジの立ち上がりまで続きます。

リード/ライト (R/W)

この信号は入力のみで、VICとマイクロプロセッサとのデータの流れを制御します。R/W信号が低レベルで、VICのチップセレクト条件が満足されている場合、マイクロプロセッサは、選択されているVICコントロール・レジスタにデータの書込みが可能です。R/W信号が高レベルで、チップセレクト条件が満たされている場合、マイクロプロセッサは、選択されているVICコントロール・レジスタからデータの読み込みが可能です。

VICとマイクロプロセッサとのデータの受渡しは、すべてP_φ = 1の場合のみであることは、注目に値します。P_φの間、VICはディスプレイ用のデータをメモリーからフェッチし、VICがメモリーロケーションのどこへも、書込みしないようにR/W信号を高レベルに保たれなくてはなりません。

データバス (DB₀—DB₁₁)

12ビットデータバス (DB₀—DB₁₁) は、2つの部分に分けられます。下位8ビット (DB₀からDB₇) はマイクロプロセッサとディスプレイに必要なデータのインタフェイスに使われ、上位4ビットは色とモード情報に使われます。下位8ビット (DB₀からDB₇) は、さらに2つの分類に分けられます。すなわち、マイクロプロセッサ側のインタフェイスとビデオデータ側のインタフェイスの2種類です。P_φ = 1の間、DB₇からDB₀はマイクロプロセッサVICのデータの受渡しに使われます。P_φ = 1の間は、DB₇からDB₀はディスプレイデータのフェッチに使われます。

クロック

マスタ・オシレータ・クロック入力 (ϕ_1 、 ϕ_2)

6560は14.31818MHz(NTSC)、2相クロックが必要です。クロック信号は、+5Vで非重複でなくてはなりません。

システムクロック (P ϕ_1 、P ϕ_2)

これは、VICのタイミング・ジェネレータです。+5V、非重複、1.02MHzで、6512マイクロプロセッサのドライブが可能です。

メモリークロック (オプション、 ϕ_M)

これは単一2.04MHzクロックで、アドレスバスが有効になった後、VICのメモリーに、ストロープが必要な場合、用いられます。37ピンに設置可能なオプションです。

アナログ・ディジタルコンバータ (POTX、POTY)

これは入力ピンで、マイクロプロセッサの判別できる8ビットの16進数に、分圧値を変換します。単一RC時定数積分により実行されます。分圧計は、ポート・ピンに接続された外部コンデンサの測定に用います。

コンポジット・サウンド (COMP SND)

このピンは、VICブロック図に示すように、6560の音の合成を出力します。ハイ・インピーダンス出力(約1K Ω)で、スピーカーをドライブするには、外部に増幅器およびバッファを必要とします。

コンポジット・シンクおよびルミナス (SYNC & LUMIN)

このピンはオープン出力で、標準テレビジョンに必要なビデオ同期と輝度に関する情報を与えます。

コンポジット・カラー (COMP COLOR)

この信号は、標準テレビジョンが受けとる色の情報を与えます。コンポジット・カラー・ピンは、ハイ・インピーダンス出力バッファであり、3.579545MHzで、突発信号とカラー逆変換および振幅情報を与えます。

リセット

オプションの37ピン入力で、水平・垂直同期カウンタを外部信号と同期をとります。

バス・アベイラブル

オプションの37ピン出力で、ビデオのメモリー・フェッチに対して、VICの状態を示します。VICがメモリーアクセスをおこなう2 μ sec前に低レベルになり、スクリーンがリフレッシュされるまで、低レベルを保ちます。

ライト ガン/ペン

オプションの37ピン入力で、立下がりにより、コントロールレジスタ6、7にラッチされているスクリーン上にスキャンされている瞬間のドット位置を与えます。このピンは、「ターゲット・ショット」等のゲームやライトペンに用いられるフォトディテクターに使われます。

動作説明

プログラマブル・カラー文字をつくるために、VICは、3つに分かれたエリア、すなわち文字ポインタ、ディスプレイ文字、カラーポインタから、外部メモリのアクセスをおこないます。文字ポインタはRAMの1ブロック(ビデオ・マトリックスとよばれる506バイトが代表的)であり、そこに表示したい文字のポインタがあります。文字エリアは、8か16バイトのブロックからなっており(通常セルとよばれる)そこに表示しようとする実際のドットパターンがあります。文字セルはRAMでもROMでもかまわず、スクリーンに表示する方法によります。カラーポインタ・エリアはRAMの小さなブロックにあります(ふつう506個の4ビット群がカラー・マトリックスとよべれます)。4ビットカラーポインタは表示文字の色を決定し、2つのモードのうち1つを選択します。

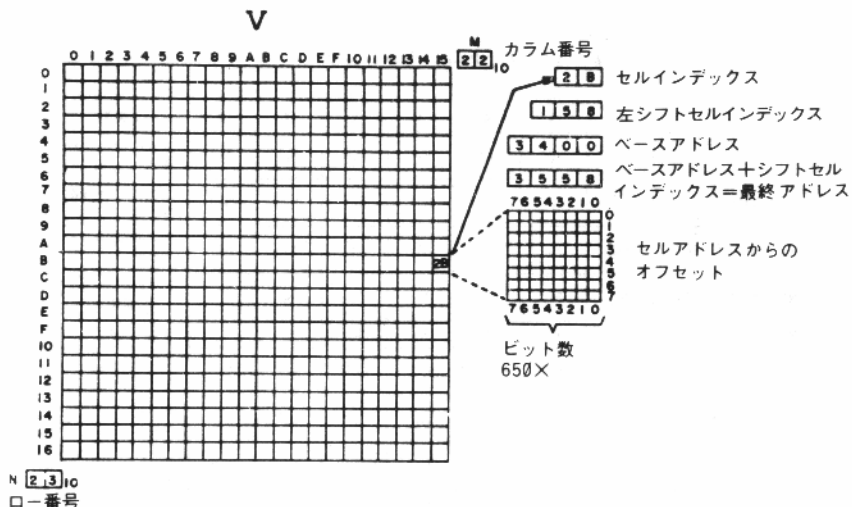
ビデオ・マトリックス、カラー・マトリックス、文字セルを結びつけて、必要なデータをスクリーンに表示するのは、外部のマイクロプロセッサによります。

VICの動作についてより完全に理解するために図1を示します。典型的なビデオ・マトリックスで、水平方向に22文字、垂直方向に23文字表示可能で、スクリーンの水平方向176ドット、垂直方向184ドットに対して506文字を表示します。文字表示位置は、対応するポインタまたはインデックスをもち、特定の位置に文字を表示します。

たとえば、(B、15)には2Bというインデックスがあります。つまり2B番目の文字が、その位置に表示されるのです。VICは文字インデックス値2Bをフェッチし、表示する文字位置のアドレス計算をおこないます。計算方法は簡単で、8 \times 8の文字セルの場合、インデックスは3回左にシフトし(8倍)、VICコントロールレジスタCR5にあるスタートアドレスを加算します。今の場合、文字セルの開始アドレスは\$3400であり、文字インデックスの左シフトした値と加算され、文字位置として、メモリ上\$3558が得られます。

ある特定文字の表示回数は無制限です。同じ文字インデックス(たとえば2B)を使用して、文字データは何度でも表示されます。簡単なソフトウェア・ドライバにより、十分なRAM(約4KバイトのセルRAM)があれば、ビット単位の表示システムとしてVICを使うことができます。

図1 典型的なビデオマトリックス(23×22)



補色/バックグラウンド・カラー

- 0 ブラック
- 1 ホワイト
- 2 レッド
- 3 シアン
- 4 マジェンタ
- 5 グリーン
- 6 ブルー
- 7 イエロー
- 8 オレンジ
- 9 ライトオレンジ
- A ピンク
- B ライトシアン
- C ライトマジェンタ
- D ライトグリーン
- E ライトブルー
- F ライトイエロー

ボーダー/キャラクター・カラー

- 0 ブラック
- 1 ホワイト
- 2 レッド
- 3 シアン
- 4 マジェンタ
- 5 グリーン
- 6 ブルー
- 7 イエロー

レジスタ機能

6560には8ビット構成のコントロール・レジスタが16個あり、マイクロプロセッサがVICのすべてのオペレーティング・モードでのコントロールを可能にしています。コントロール・レジスタとその機能については次に記述します。図2はレジスタのロケーションと内容を示しています。

1000 ORIGIN

図2 VICコントロールレジスタ

	7	6	5	4	3	2	1	0	ビット番号
CR ₀ 1000	I	S _X ⁶	S _X ⁵	S _X ⁴	S _X ³	S _X ²	S _X ¹	S _X ⁰	スクリーンX座標 原点
CR ₁ 1001	S _Y ⁷	S _Y ⁶	S _Y ⁵	S _Y ⁴	S _Y ³	S _Y ²	S _Y ¹	S _Y ⁰	スクリーンY座標 原点
CR ₂ 1002	B _V ⁹	M ₆	M ₅	M ₄	M ₃	M ₂	M ₁	M ₀	ビデオマトリックス 列番号
CR ₃ 1003	R ₀	N ₅	N ₄	N ₃	N ₂	N ₁	N ₀	D	ビデオマトリックス 行番号
CR ₄ 1004	R ₈	R ₇	R ₆	R ₅	R ₄	R ₃	R ₂	R ₁	ラスタ値
CR ₅ 1005	B _V ¹³	B _V ¹²	B _V ¹¹	B _V ¹⁰	B _C ¹³	B _C ¹²	B _C ¹¹	B _C ¹⁰	ベースアドレス コントロール
CR ₆ 1006	L _H ⁷	L _H ⁶	L _H ⁵	L _H ⁴	L _H ³	L _H ²	L _H ¹	L _H ⁰	ライトペン 水平
CR ₇ 1007	L _V ⁷	L _V ⁶	L _V ⁵	L _V ⁴	L _V ³	L _V ²	L _V ¹	L _V ⁰	ライトペン 垂直
CR ₈ 1008	P _X ⁷	P _X ⁶	P _X ⁵	P _X ⁴	P _X ³	P _X ²	P _X ¹	P _X ⁰	ポットX
CR ₉ 1009	P _Y ⁷	P _Y ⁶	P _Y ⁵	P _Y ⁴	P _Y ³	P _Y ²	P _Y ¹	P _Y ⁰	ポットY
CR _A 100A	S ₁	F ₁ ⁶	F ₁ ⁵	F ₁ ⁴	F ₁ ³	F ₁ ²	F ₁ ¹	F ₁ ⁰	F _N ⁽¹⁾
CR _B 100B	S ₂	F ₂ ⁶	F ₂ ⁵	F ₂ ⁴	F ₂ ³	F ₂ ²	F ₂ ¹	F ₂ ⁰	F _N ⁽²⁾
CR _C 100C	S ₃	F ₃ ⁶	F ₃ ⁵	F ₃ ⁴	F ₃ ³	F ₃ ²	F ₃ ¹	F ₃ ⁰	F _N ⁽³⁾
CR _D 100D	S ₄	F ₄ ⁶	F ₄ ⁵	F ₄ ⁴	F ₄ ³	F ₄ ²	F ₄ ¹	F ₄ ⁰	F _N ⁽⁴⁾
CR _E 100E	C _A ³	C _A ²	C _A ¹	C _A ⁰	A ₃	A ₂	A ₁	A ₀	振幅
CR _F 100F	C _B ³	C _B ²	C _B ¹	C _B ⁰	R	C _T ²	C _T ¹	C _T ⁰	カラー コントロール

注) NU=未使用

図2

CR₀

ビット0-6はテレビ・スクリーンの左端から最初に現われる文字カラムの距離を決定し、スクリーンのビデオ・マトリックスの横方向のサイズに使用されます。ビット7はインターレース・スキャン・モード (I=1) を選択します。

CR1

テレビ・スクリーンの上から最初に現われる文字ローの距離で、スクリーンのビデオ・マトリックスの垂直方向のサイズに使用されます。

CR2

ビット0-6はビデオ・マトリックスのカラム数をセットします。ビット7は、CR5のビデオ・マトリックス・アドレスの一部です。

CR3

ビット1-6はビデオ・マトリックスのローの数をセットします。ビット0は8×8の文字マトリックス(D=0)か16×8の文字マトリックス(D=1)に使用されます。ビット7はCR4のラスタ値の一部です。

CR4

テレビのラスタ・ビームがスキャンする現在のライン・ナンバーを示します。

CR5

ビット0-3は文字のセル・スペースがはじまるアドレスを決めます(ビットA13からA10はアクチュアル・アドレスです)。ビット4-7およびCR2のビット7はビデオ・マトリックスのスタート・アドレスを決めます(ビットA13からA9はアクチュアル・アドレスです)。

CR6

ライトペンの水平方向のラッチ・ポジションです。

CR7

ライトペンの垂直方向のラッチ・ポジションです。

CR8

POTXのデジタル値です。

CR9

POTYのデジタル値です。

CRA

ビット0-6は最初のオーディオ・オシレーターの周波数をセットします。ビット7はオシレーターがオンで1、オフで0です。

CRB

2番目のオーディオ・オシレーター用で、CRAと同様です。

CRC

3番目のオーディオ・オシレーター用で、CRAと同様です。

CRD

ノイズの周波数をセットし、CRAと同様です。

CRE

ビット0-3は、オーディオ・シグナルのボリュームをセットします(いかなる音を作る時も最小1つのサウンド・ジェネレーターはオンの状態にしておかねばなりません)。ビット4-7はマルチカラー・モードのオペレーションで使用される補色のコードを表わしています。

CRF

ビット4-7は16種のバックグラウンドのカラーのうち1つを選択し、ビデオ・マトリックス内のバックグラウンド・エリアにそのカラーをセットします。ビット0-2はビデオ・マトリックスの外側のボーダーエリアに8色のうち1つを選択します。ビット3はビデオ・マトリックスで共通のバックグラウンド・カラーのとき $R=1$ で、反転するときは $R=0$ です。全文字を同じバックグラウンド・カラーにしたり、個々異なるバックグラウンドにするのは、カラーRAM内のコードによって決めます（マルチカラー・モードのときはRビットは関係なく、CRF機能もこのモードでは異なります。詳細はオペレーティング・モードのセクションを参照して下さい）。

カラー・オペレーティング・モード

VICには2つのカラー・オペレーション・モード、HI-RES（ハイ・レゾリューション）モードとマルチカラー・モードがあります。このオペレーティング・モードは、基本的にはどのようにしてキャラクター・セルに情報を与えてテレビ・スクリーン上の点を変化させるかを決めます。オペレーティング・モードは、ビデオ・マトリックスの個々のキャラクター・ロケーションで示されるカラー・ポインタのMSBで決定されます。もしキャラクターのカラー・ポインタのMSBがゼロの時は、そのキャラクターはHI-RESモードで表示されます。MSBが1になると、そのキャラクターはマルチカラー・モードで表示されます。

HI-RESモードではキャラクター・セルとスクリーン上の表示される点が1対1に対応しています。たとえばキャラクターの全ビットが1のとき表示される色もあれば、全ビットがゼロの時に表示される色もあります。文字のフォアグラウンド・カラーはキャラクター・カラー・ポインタの残り3ビットで指定され、バックグラウンド・カラーはCRFで指定されます。

マルチカラー・モードではキャラクター・セルの個々2ビットがスクリーン上の1点と対応し、その点の色は2ビット・コードで決まります。HI-RESモードは1つの文字では2色までしか表示できません。マルチカラー・モードは1つの文字で4色まで可能ですが、これはセル・データの2ビットがスクリーン上の1点と対応しているため、HI-RESモードを水平に半分したにすぎません。これは 8×8 のキャラクター・セルのメモリー・マップをスクリーン上で 8×4 のキャラクターにしたこととなります。ここで注意しなければならないのは、 8×4 のマルチカラー・キャラクターと 8×8 のHI-RESキャラクターの要求するメモリーの量は同じであり、単にスクリーン上のマップが異なるだけであるということです。

マルチカラー・モードでは2ビットが点を構成する4色のうちの1色を選択します。これらの2ビットで構成される4つのコードがVICに点のカラー情報を与えます。これにより、その点の色はバックグラウンド、ボーダー、補色、フォアグラウンドのどれかになります。

マルチカラー・モードで選択されるコード

00-バックグラウンド・カラー (CRF)
 01-ボーダー・カラー (CRF)
 10-フォアグラウンドカラー (カラーRAM)
 11-補色 (CRE)

●注意● 2ビットのコード自身はカラー・コードではなく、3または4ビットで構成される4つの異なったカラー・コードのポインターにすぎません。

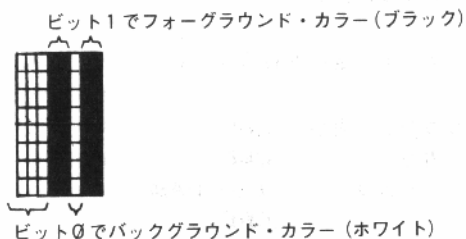
例)

CRF=1F 文字のバックグラウンド・カラーはホワイト (1)
 ボーダー・カラーはイエロー (7)
 反転は選択されていません (R=1)
 CRE=6X 補色はブルー (6)

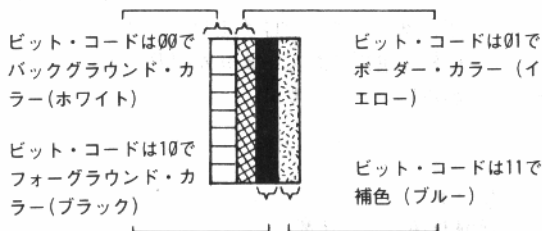
文字の定義

バイト	ビット	16進
	76543210	
0	00011011	1B
1	00011011	1B
2	00011011	1B
3	00011011	1B
4	00011011	1B
5	00011011	1B
6	00011011	1B
7	00011011	1B

もしカラー・ポインターが指定する文字が0 (0000) のときは、フォアグラウンド・カラーはブラック (0) でHI-RESモード (MSB=0) が選択されています。この文字は次のようにスクリーンに表示されます。



もしカラーの指定している文字が8 (1000) のときは、フォアグラウンド・カラーはブラック (0) でマルチカラー・モード (MSB=1) が選択されています。この文字は次のようにスクリーンに表示されます。

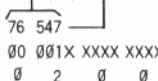
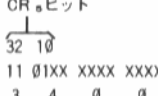


●注意● これは単なる例であり、多くのテレビ・セットでは接近した点の異なる色は鮮明には出ません。

文字の表示モードはキャラクター・カラー・ポインターで指定され、スクリーン上のキャラクター・ロケーションは個々独立したカラー・ポインターを持っているので、HI-RESとマルチカラー・キャラクターを自由にまぜることが可能です。これによりアルファニュメリックをHI-RESモードのキャラクターで表示でき、マルチカラー・モードでは文字の幅を広くすることもできるなど、表示機能の柔軟性を大幅に増大しています。

VICコントロールレジスタの使用例：

簡単にするために全部の文字がHI-RESモードであり、VICレジスタは次の値に位置するとします。

レジスター	内容 (16進)	2 進	結 果
CR0	03	0/000 0011	スクリーンのビデオ・マトリックスを左端から3×4ドット幅だけ移動します。 インターレースは選択されていません (I=0)。
CR1	19	0001 1001	スクリーンの上からビデオ・マトリックスを16進の19×2ドット高さだけ下へ移動します。
CR2	96	1/001 0110	ビデオ・マトリックスのカラムを16進の16 (=22、ベースは10) にセットします。 (ビット7はCR5で使用)。
CR3	2E	X/010 1110	ビデオ・マトリックスのローを(01011 (=23、ベースは10) にセットします。 8×8キャラクターのマトリックスが選択されます (D=0)。
CR5	特定のシステムのメモリー・ロケーションをアクセスするためにセットされます。たとえばビデオ・マトリックスのスタート・アドレスが16進の3400になっているのを16進の0200に変える場合に、このCR5がセットされます。		
CR5	0D 14ビットのアドレスで構成されます。 CR ₅ ビット CR ₂ ビット  00 001X XXXX XXXX 0 2 0 0 これはビデオ・マトリックス用です。 キャラクター・マトリックスも同様に14ビットで構成されません。 CR ₅ ビット  11 01XX XXXX XXXX 3 4 0 0	0000 1101	CR2のビット7は1にセット。

CRA	00	0/000 0000	オシレーター1はオフ。
CRB	9A	1/001 1010	オシレーター2はオンで、相対周波数は1A。
CRC	00	0/000 0000	オシレーター3はオフ。
CRD	A5	1/010 0101	ノイズ・ジェネレーターはオンで、相対周波数は25。
CRE	XF	XXXX1111	音量が最大にセットされます。
CRF	0E	0000/1/110	バックグラウンド・カラーは共通で全文字がブラック(0)でボーダー・カラーはダークブルー(6)、個々の文字はそれ自身の色で黒いバックグラウンドの上に表示されています(R=1)。

これらのレジスタ値は、23×22文字のビデオ・マトリックスのスクリーンを作り、個々の文字は黒のバックグラウンドの上に表示され、ビデオ・マトリックスを囲んでいるボーダー・エリアはダークブルーです。それにサウンド・ジェネレーター#2とホワイト・ノイズが出力します。

これら全部のレジスタにより異なった効果を作り出すことができます。

例：

CR0の数を増すとビデオ・マトリックス領域は右へシフトします。CRBの数を減らすとオシレーター2の周波数はダウンします。CRFを06に変えるとビデオ・マトリックスのボーダーはダークブルーで残りますが、文字はブラックになり、バックグラウンドは別の異なった色に変わります。

スクリーン上に絵を作るためには、VICにローとカラムの数および中心の値を適切なレジスタにロードしなければなりません。

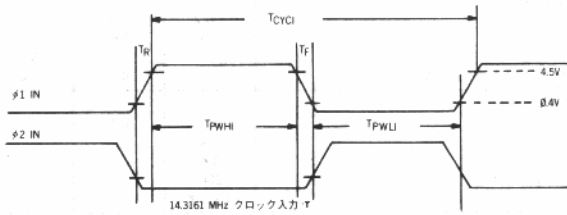
直流特性 $T_A = 0^\circ\text{C} \sim +50^\circ\text{C}$, $V_{DD} = 5V \pm 5\%$ (他の指定がないとき)

パラメーター	最小	最大	タイプ	単位
リード/ライトリセット(オプション) アドレスとデータ入力の状態 V_{IL} V_{IH} 入力容量 入力漏れ(全出力はハイインピーダンス状態)	-0.2 2.4	0.4 5.6 8.0 10.0	5.0 1.0	Volts Volts pF μA
アドレスとデータ出力状態 V_{OL} V_{OH} I_{OL} (シンクカレント) $V_{OL} = 0.4$ I_{OH} (ソースカレント) $V_{OH} = 2.4$ 3状態のインピーダンス	2.4 2.4 200 1×10^6	0.4		Volts Volts mA μA Ohms
クロック入力(ϕ_1 と ϕ_2 の入力) 周波数 容量 V_{IL} V_{IH}	-0.2 4.5	10.0 0.3	14.3 18 18 5.0	MHz pF Volts Volts
クロック出力($P\phi_1$, $P\phi_2$) V_{OL} I_{OL} (≈ 0.3 Volts V_{OL}) V_{OH} I_{OH} (≈ 4.7 Volts V_{OH}) ローディング 周波数	1.6 $V_{DD} - 2$ 200	0.3V 120.0	1.02	Volts mA Volts μA pF MHz

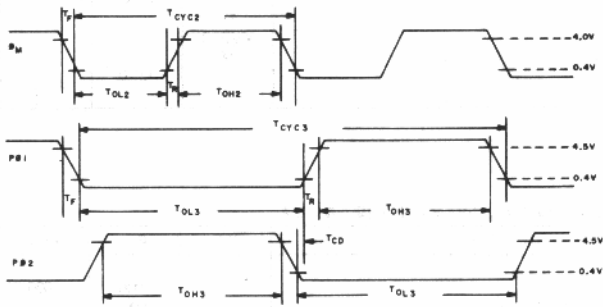
直流特性 $T_A = 0^\circ\text{C} \sim +50^\circ\text{C}$, $V_{DD} = 5V \pm 5\%$ (他の指定がないとき)

パラメーター	最小	最大	タイプ	単位
音の構成 出力インピーダンス 最大カレント(シンクまたはソース) 出力オフセットボルテージ V_{OH} (最大振幅) V_{OL} (最大振幅) V_{OH} (最小振幅) V_{OL} (最小振幅)	2.2 3.2 2.55	2.8 2.8 1.8 2.45	1000	Ω μA 2.5 3.5 1.5 2.6 2.4 Volts Volts Volts Volts
Pot入力 $V_{TRIGGER}$ (ライジングエッジ) Potリセット V_{OL} I_{OL} ($\approx V_{OL} - 0.2$)	2.2 500	2.8 0.2	2.5	Volts μA
ライトベン入力(オプション) $V_{TRIGGER}$ (フォーリングエッジ)	2.8	2.2	2.5	Volts
ϕM (オプション) V_{OL} I_{OL} (≈ 0.3 Volts V_{OL}) V_{OH} I_{OH} (≈ 4.7 Volts V_{OH}) ローディング 周波数	1.6 $V_{DD} - 7$ 100	0.4 60	2.04	Volts mA Volts μA pF MHz
バスの使用(オプション) V_{OL} I_{OL} V_{OH} I_{OH} V_{DD} I_{DD}	1.6 2.4 100 4.75	0.3 5.25 150	5.00 120	Volts mA Volts μA Volts mA

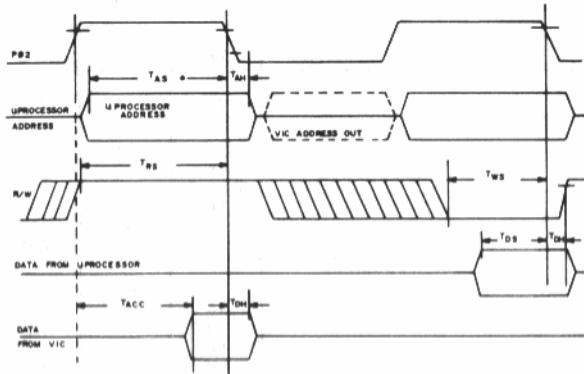
VIC入力クロック

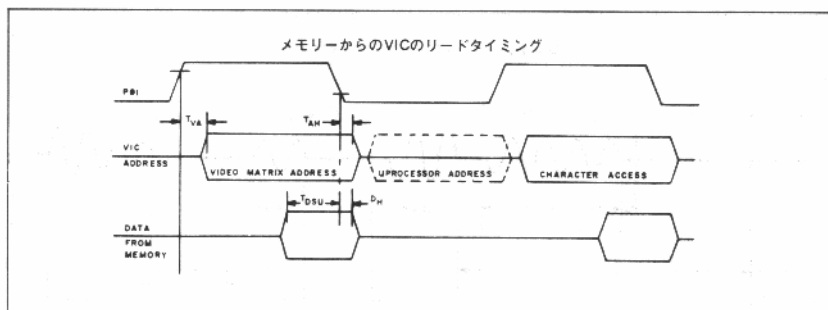


VIC出力クロック



VICに対するマイクロプロセッサのリード/ライトタイミング





VICシステムタイミング

記号	特性	最小	標準	最大	単位
T_{CYC1}	VIC入力クロックタイミング				
T_{PW11}	入力クロックサイクルタイム	69.82		69.84	ns
T_{PWL1}	"H"クロック	20			ns
T_{R1}	"L"クロック	20			ns
T_{F1}	立上がり立下がり時間			10	ns
T_{CYC2}	VIC出力クロックタイミング				
T_{OL2}	2MHzクロックサイクルタイム	480		500	ns
T_{OH2}	"L"出力クロック出力	200		260	ns
T_{OH2}	"H"出力クロック出力	180		250	ns
T_{CYC3}	1MHzマイクロプロセッサクロックサイクルタイム	960		990	ns
T_{OL3}	"L"出力クロック	380		500	ns
T_{OH3}	"H"出力クロック	380		500	ns
T_{CD}	4Vでのクロック遅延時間	5		20	ns
T_{R}	最大Qでの立上がり時間			80	ns
T_{F}	最大Qでの立下がり時間			40	ns
T_{AS}	VICに対するマイクロプロセッサのリード/ライトタイミング				
T_{AH}	アドレスセットアップ時間	375			ns
T_{AH}	アドレス・ホールド時間	5			ns
T_{RS}	読込みセットアップ時間	375			ns
T_{WS}	書き込みセットアップ時間	275			ns
T_{DS}	データセットアップ時間	200			ns
T_{ACC}	データアクセス時間	350			ns
T_{DH}	データホールド時間	30			ns
T_{VA}	メモリーからのVICのリードタイミング				
T_{VA}	P0から有効アドレスまでの時間				ns
T_{AH}	アドレスホールド時間	10			ns
T_{DSU}	データセットアップ時間	60			ns
DH	データホールド時間	20			ns
BLANKING	色と発光タイミングの同期構成				
B_3	ブランキング期間(ビデオなし)	10.0	11.0	12.0	μ s
BURST	ブリースウェイ	3	5	7	μ s
BURST	カラーバーストリファレンス信号	4.0	5.0	6.0	μ s
H_3	出カタイミングの同期構成				
H_1	水平同期パルス	4.0	5.0	6.0	μ s
H_1	水平ライン時間	63.0	63.5	64.0	μ s
$H_{1/2}$	1/2水平ライン時間	30.0	31.5	32.5	μ s
E	イコーリセーション・パルス	2.0	2.5	3.0	μ s
E_1	イコーリセーション時間	188.0	190.5	192.0	μ s
V_3	垂直同期パルス	188.0	190.5	192.0	μ s
V_3 to V_5	垂直同期間の時間		16.66		ms

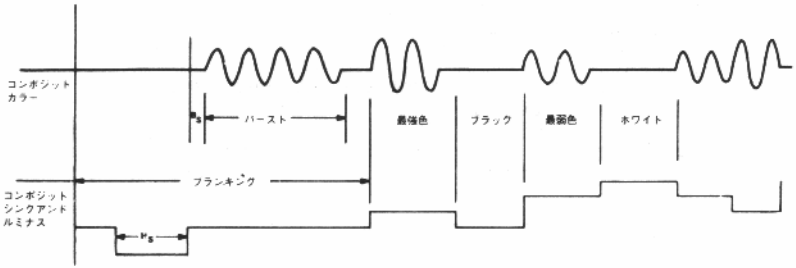
1. 兼

1. カラーバースト信号は、3,579,545MHzのカラー・フェイスで、他のカラー種別はそれにより測定されます。
たとえは、最終層のプルは3,579,545MHzで、バースト信号がH₁の間もあ

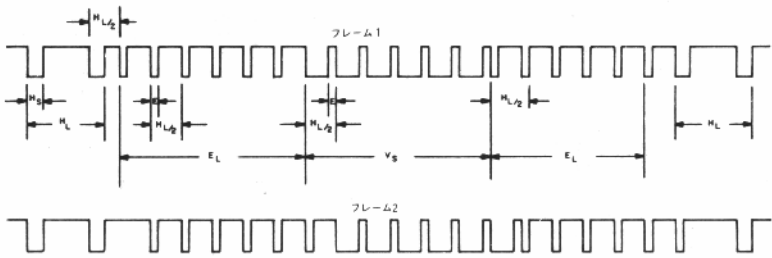
6. なら、135nsの遅れももちます。

2. V₃の間のH₁期間の個数は、混合モードで、262.5です。
3. V₃の間のH₁期間の個数は、非混合モードで、フレームあたり262です。
4. NTSCのみ。

コンポジットシンク、カラー、ルミナス



コンポジットシンク出力



MPS6522汎用インターフェイスアダプタ

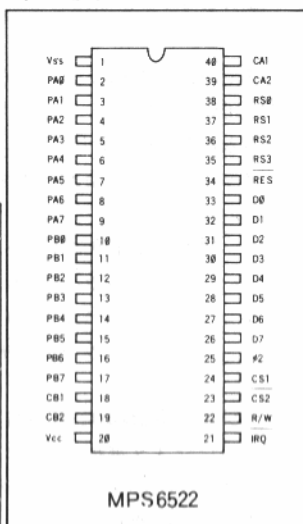
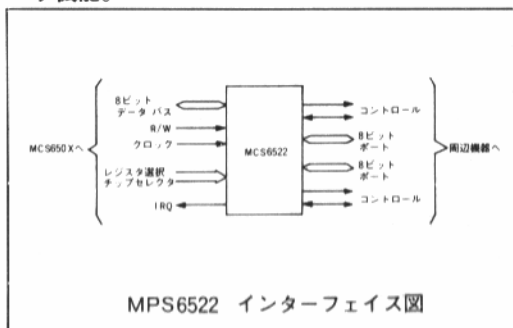
概要

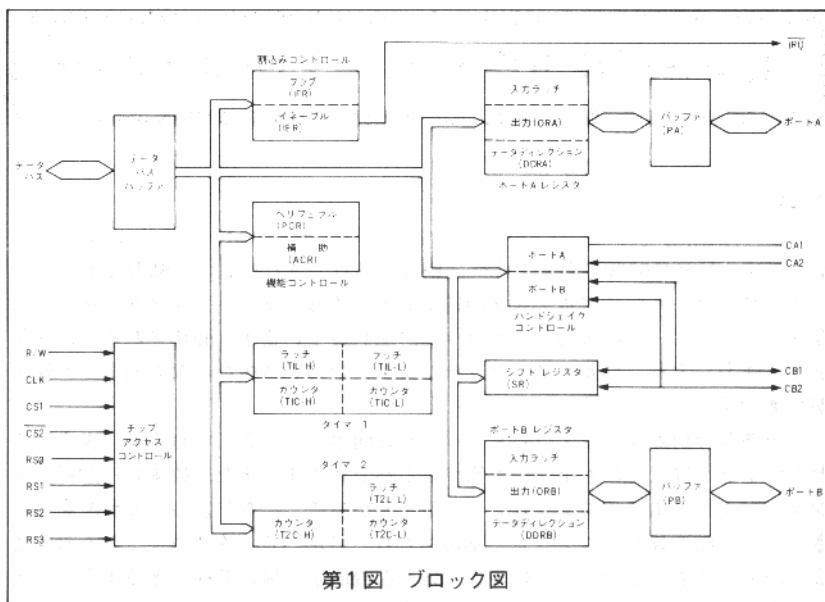
MPS6522汎用インターフェイスアダプタ (Versatile Interface Adapter: VIA) は、MP S 6520の機能をすべて備えており、さらに2つの強力なインターバル・タイマー・シリアルデータの送受信ができるシフト・レジスタ、そしてペリフェラル・ポートの入力データラッチが加わっています。強化されたハンドシェイク能力により、マルチプロセッサシステムでのVIA相互の双方向データ転送が可能です。

周辺機器の制御は主に2つの8ビット双方向ポートによっておこないます。これらのポートは入力にも出力にも使うことができます。また、数本のI/O線をインターバル・タイマでコントロールすることができ、これによって方形波を発生させてその周波数をプログラムで制御したり、外部パルスをカウントしたりすることができます。このチップの多くの強力な機能を容易に発揮できるように、チップ内部のレジスタとして、割込みプラグレジスタ、割込みイネーブルレジスタ、および2つの機能コントロールレジスタがあります。

特長

- MP S 6520をよりパワフルに強化
- Nチャンネル デプレッションロードによる5V単一電源
- 完全スタチック動作、TTLコンパチブル
- CMOSコンパチブルなペリフェラルコントロール線
- プロセッサと周辺機器とのデータ転送を強力にコントロールできる充実したハンドシェイク機能。





第1図 ブロック図

プロセッサインターフェイス

ここでは、MPS6522とプロセッサとのインターフェイスに使うバスとコントロール用の説明をします。このインターフェイスのAC、DC特性は最後の特性表に記してあります。

1. フェーズ2クロック($\phi 2$)

MPS6522とプロセッサとの間のデータのやりとりは、フェーズ2クロック $\phi 2$ の時のみおこなわれます。さらにこの $\phi 2$ は、チップの中のタイマやシフトレジスタのクロックとしても使われます。

2. チップセレクト(CS1, CS2)

2本のチップセレクト入力、プロセッサのアドレスバスから直接またはデコーダを介してつながります。CS1が H でCS2が L の時、MPS6522のある選択されたレジスタがアクセスされます。

3. レジスタ選択線(RS0, RS1, RS2, RS3)

4本のレジスタ選択線はプロセッサのアドレスバスにつなぎ、プロセッサはこれによりアクセスするレジスタを選択します。16通りの組合せでアクセスされるレジスタは次の通りです。

RS3	RS2	RS1	RS0	レジスタ	注	RS3	RS2	RS1	RS0	レジスタ	注
L	L	L	L	ORB		H	L	L	L	T2L-L T2C-L	ラッチへ書き込み カウンタ読み出し
L	L	L	H	ORA	ハンドシェイクの コントロール	H	L	L	H	T2C-H	T2L-LからT2C-Lへの データ転送のトリガ
L	L	H	L	DDRB		H	L	H	L	SR	
L	L	H	H	DDRA		H	L	H	H	ACR	
L	H	L	L	T1L-L T1C-L	ラッチへ書き込み カウンタ読み出し	H	H	L	L	PCR	
L	H	L	H	T1C-H	T1L-LからT1C-Lへの データ転送のトリガ	H	H	L	H	IFR	
L	H	H	L	T1L-L		H	H	H	L	IER	
L	H	H	H	T1L-H		H	H	H	H	ORA	ハンドシェイクに 影響せず

4. リード/ライト線(R/W)

MPS6522とプロセッサとの間のデータのやりとりの方向は、このR/Wで決まります。R/Wが「L」の時、データはプロセッサからMPS6522の選ばれたレジスタへ転送され（書き込み動作）、「H」の時はそのレジスタから外部へ転送されます（読み出し動作）。

5. データバス(DB0-DB7)

MPS6522とプロセッサとのデータ転送はこの8ビット双方向データバスでおこないます。バスドライバは、通常は高インピーダンス状態になっていて、チップセレクト信号が来て（CS1=H, CS2=L）、R/W=「H」、 $\phi 2$ =「H」になると、データバス上のデータが選択されているレジスタに送り込みます。

6. リセット(\overline{RES})

リセット入力はT1、T2、SRを除くすべてのレジスタの内容を0にします。

これにより周辺機器とのインターフェイス線は入力状態になり、タイマ、シフトレジスタ等はディスエーブルされ、このチップからの割込みはディスエーブルとなります。

7. 割込みリクエスト(\overline{IRQ})

割込みリクエスト出力は、内部の割込みプラグがセットされ、その時対応する割込みイネーブルビットが1ならば「L」になります。この出力はオープンドレインになっているのでシステム内の他の割込みリクエストとワイアードORをとることができます。

ペリフェラルインターフェイス

ここでは、MPS6522の内部レジスタが周辺機器をドライブする時に使うバスライン、コントロールラインを簡単に説明します。

1. Aポート(PA0-PA7)

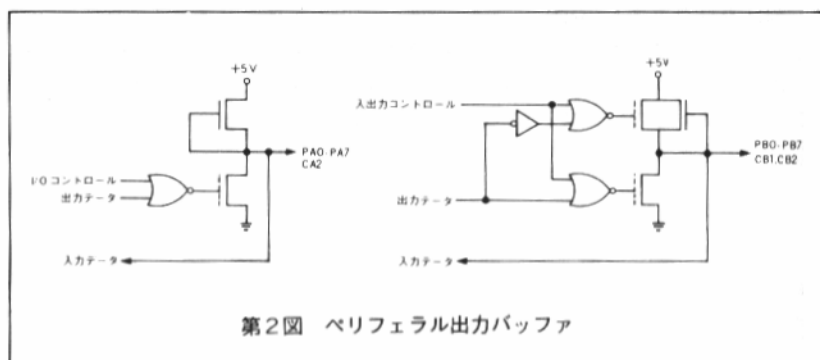
Aポートには8本の線があって、入出力の切替はデータ・ディレクション・レジスタをプログラムすることによって1ビットずつ個別におこなえます。出力ピンのデータは出力レジスタから送られ、入力ピンのデータはCA1のコントロールによって内部のレジスタにラッチされます。こういった動作はすべてプロセッサが、チップ内部の制御レジスタを介してコントロールします。PA0-P A 7は、入力時には標準TTLファンイン1、出力時にはファンアウト1です。

2. Aポートコントロール線(CA1、CA2)

Aポートのコントロール線は2本あって、割込み入力およびハンドシェイク出力として動作します。それぞれ対応する割込みイネーブルビットと共に割込みフラグをコントロールします。さらにCA1はAポートの入力データのラッチをコントロールします。動作モードの切替は、プロセッサがチップ内部のコントロールレジスタを介しておこないます。CA1は高インピーダンスの入力のための線ですが、CA2は標準TTLのファンイン1、ファンアウト1です。

3. Bポート(PB0-PB7)

Bポートも8本の双方向データ線で、出力レジスタとデータ・ディレクション・レジスタにより、Aポートとほぼ同様にコントロールされます。さらにこのポー



トでは、2つのインターバルタイマの片方の出力をPB7に出力し、もう一つのタイマでPB6に出力するパルスを計数することができます。この8本は標準TTLでファンイン1、ファンアウト1で、出力モード時には1.5Vで30mAのソース能力がありますので、ダーリントンのトランジスタスイッチなどを直接ドライブすることができます。

4. Bポートコントロール線(CB1, CB2)

Bポートのコントロール線も割込み入力およびハンドシェイク出力として動作します。CA1、CA2と同様、それぞれ対応する割込みイネーブルビットと共に割込みフラグをコントロールします。またシフトレジスタでコントロールするシリアルポートにもなります。標準TTLでファンイン1、ファンアウト1で、ダーリントランジスタスイッチを直接ドライブできるよう、出力時に1.5Vで1mAのソース能力を持っています。

MPS6522の動作

ここでは第1図に示したいろいろなロジックと、MPS6522の内部の動作について詳しく説明します。

A. データバスバッファ(DB)、ポートAバッファ(PA)、ポートBバッファ(PB)

これらのバッファの電流、電圧のドライブ能力は前の章で述べた通りです。A、C、DC特性は最後の特性表に記してあります。

B. チップアクセスコントロール

チップアクセスコントロールは、チップセレクト9条件を検出し、レジスタセレクト入力をデコードして選択されたレジスタをアクセスするロジックで、データのやりとりの方向とタイミングを決定するR/Wと $\phi 2$ も含まれます。MPS6522にデータを書き込む時は、まずデータは $\phi 2$ でデータ入力レジスタにラッチされ、次に $\phi 2$ ・チップセレクトで選択された内部レジスタへ転送されます。こうすることによってペリフェラルI/Oライン上のデータをグリッチなしに変化させることができます。プロセッサがMPS6522から読み出す時は、データは $\phi 2$ で選択されたレジスタから直接データバスに乗ります。

C. ポートAレジスタ群、ポートBレジスタ群

8ビットのペリフェラルポートをアクセスするのに、各々3つずつのレジスタを使います。まず、データ・ディレクション・レジスタ(DDRA、DDRB)は、ペリフェラルピンを入力にするか出力にするかを決定します。このレジスタのあるビットが0ならば対応するピンは入力となり、1ならば出力となります。

次に出力レジスタ (ORA、ORB) と入力レジスタ (IRA、IRB) があります。ピンが出力にプログラムされていれば、出力レジスタの対応するビットが1ならばピンは「H」に、0ならば「L」になります。入力にプログラムされているピンに対応するビットに任意のデータを書き込むこともでき、この時ピン上の入力データは何の影響も受けません。

パリアフェラルポートから読み込んだデータは入力レジスタ (IRA、IRB) に入り、そこから直接データバスへ送り出すことができます。入力ラッチ動作をディセーブルするとIRAは常にPAのピンと同じ内容になります。これをイネーブルするとCAIの割込みフラグ (IFR1) がセットされる直前のポートAの内容がIRAに入ります。

IRBレジスタも大体似たような動作をしますが、IRAと違うのは、IRBの出力にプログラムされているビットの内容は、ピンの電圧ではなくORBの当該ビットのデータが入ります。これによってプロセッサはIRBを読む時に、ポートBの電圧が十分に振れていなくても (ポートBは1.5Vで30mAのソース能力がある) 正しいデータが読み取れるわけです。ポートBの入力ラッチがイネーブルの時CB1の割込みフラグがセットされると、IRBは上記の組合せのデータを割込みフラグがクリアされるまでラッチします。

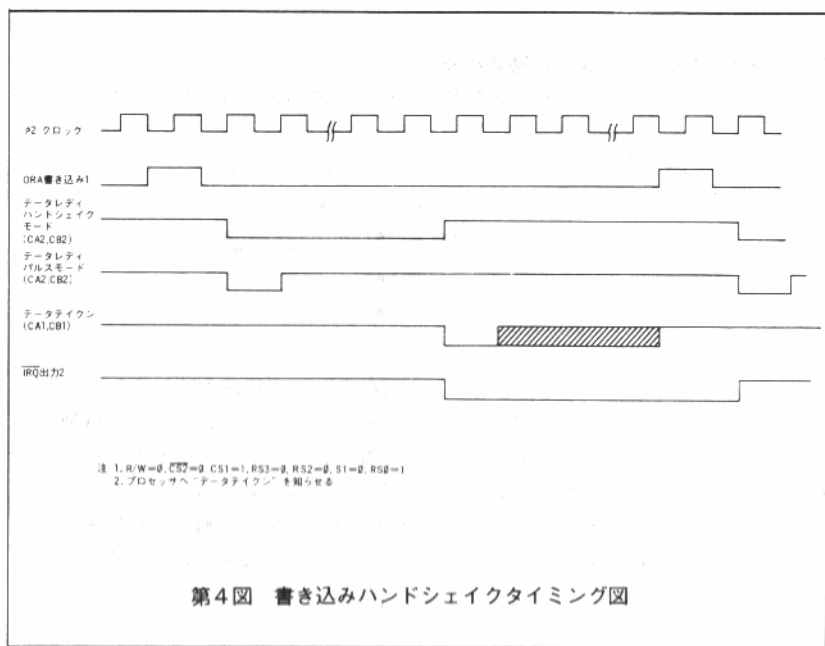
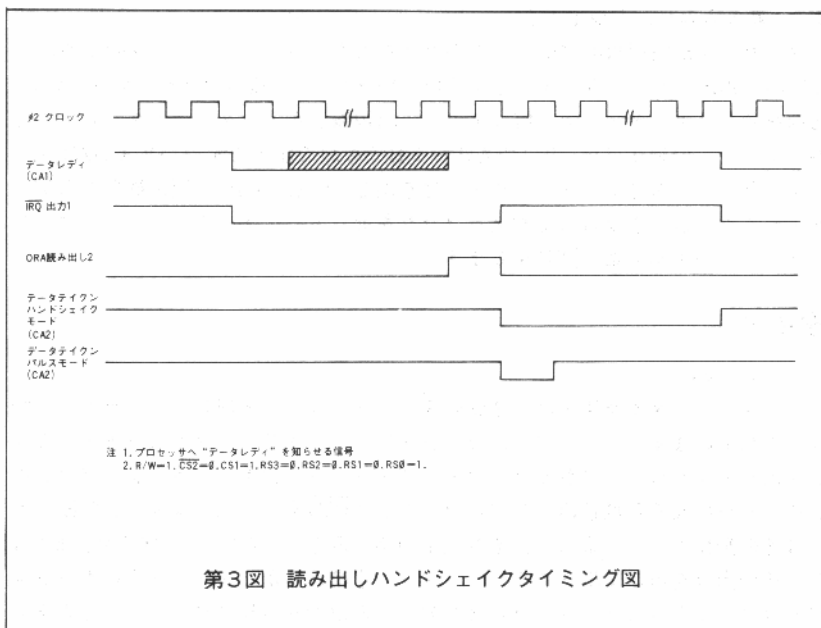
D. ハンドシェイクコントロール

MPS6522は、プロセッサと周辺機器とのデータ転送をハンドシェイクラインの動作によって強かにコントロールします。ポートAのライン (CA1、CA2) は読み出し、書き込み両方のハンドシェイクをおこないますが、ポートBのラインは書き込みのハンドシェイクのみをおこないます。

D-1. 読み出しハンドシェイク

読み出しハンドシェイクは、プロセッサが周辺機器からデータを読み出すさいのコントロールを効率よくおこないます。まず周辺機器は「データ・レディ」信号を出して、パリアフェラルポートにデータが揃っていることをプロセッサに知らせます。プロセッサはこの信号により割込みを起してデータを読み込み、「データ・テイクン」信号を返します。周辺機器はこの信号に対し次のデータ・レディで応答し、データ転送が終るまでこのやりとりを続けます。

MPS6522では、読み出しの自動ハンドシェイクがおこなえるのはポートAだけです。CA1がデータ・レディ信号を受取り、CA2でデータテイクン信号を返します。データ・レディ信号は内部フラグをセットし、プロセッサはこれによって割込みを起すか、またはソフトウェアでポーリングをして検出します。データ・テイクン信号は単発のパルスで出力することも、DCレベルで出力することもできます。DCレベルで出力する場合は、プロセッサが「L」にセットし、次のデータ・レディ信号がこれをクリアします。この動作を第3図に示します。



D-2. 書き込みハンドシェイク

プロセッサが周辺機器にデータを書き込む書き込みハンドシェイクの手順は、前述の読み出しハンドシェイクとよく似ていますが、今度はプロセッサがMC S 6522を介してデータ・レディ信号を出し、周辺機器がデータ・テイクン信号で応答します。この動作はポートAでもポートBでもおこなえます。CA2、CB2はPCレベルまたはパルスでデータ・レディ信号を出し、CA1、CB1が周辺機器からデータ・テイクン信号を受取って割込みフラグをセットし、データ・レディをクリアします。この手順を第4図に示します。

タイマ

E-1. タイマ1

インターパルタイマT1は、2個の8ビットのラッチと16ビットのカウンタで構成されています。ラッチには、カウンタにロードするデータが入ります。カウンタはデータがロードされると、フェーズ2クロックによってカウントダウンを始め、ゼロになると割込みフラグをセットして \overline{IRQ} を“L”にします。この後タイマはその動作モードによって、それ以上の割込みをディスエーブルするか、またはラッチの内容を再び受取ってカウントダウンを続けるかします。さらにカウンタがゼロになるたびにペリフェラルピンPB7の出力を反転させることもできます。以下にタイマの動作の各モードについて詳しく述べます。

E-2. タイマ1のレジスタへの書き込み

T1の四つのアドレスへデータを書き込むと、次のような動作をします。

RS3	RS2	RS1	RS0	動作 (R/W=L)
L	H	L	L	ラッチ下位8ビットへの書き込み
L	H	L	H	ラッチ上位へ書き込み、カウンタ上位へ書き込み ラッチ下位をカウンタ下位へ転送、T1割込みフラグリセット
L	H	H	L	ラッチ下位へ書き込み
L	H	H	H	ラッチ上位へ書き込み、T1割込みフラグリセット

プロセッサはカウンタの下位8ビット (TIC-L) に直接書き込むことはできません。カウンタ上位への書き込みをおこなった時にラッチ下位のデータが自動的に転送されるようになっていきます。タイミング動作はカウンタ上位が書き込まれてから開始されるので、カウンタ下位への直接書き込みは必要ありません。

アドレスセットの後半の2つは、進行中のカウントダウン動作に影響を与えずにラッチに書き込むためのアドレスです。これについては後で詳しく述べます。

E-3. タイマ1のレジスタの読み出し

読み出し時のタイマ1の四つのアドレスの動作は次の通りです。

RS3	RS2	RS1	RS0	動作 (R/W=H)
L	H	L	L	T1カウンタ下位8ビット読み出し T1割込みフラグリセット
L	H	L	H	上位カウンタ読み出し
L	H	H	L	下位ラッチ読み出し
L	H	H	H	上位ラッチ読み出し

E-4. タイマ1の動作モード

タイマ1の動作モードは補助コントロールレジスタの2つのビットが決定します。これらのビットの状態とその時の動作モードは次の通りです。

出力コントロール ACR7	フリーラン・コントロール ACR6	モード
0	0	ワンショット、PB7出力ディセーブル
0	1	フリーランニング、PB7出力ディセーブル
1	0	ワンショット、PB7出力イネーブル
1	1	フリーランニング、PB7出力イネーブル

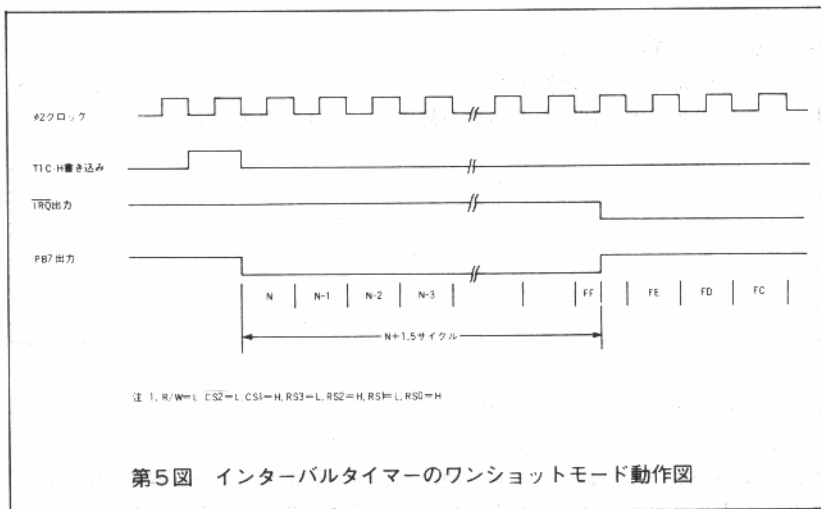
E-5. タイマ1ワンショットモード

インターバルタイマのワンショットモードでは、タイマヘデータをロードするたびに単発の割込みが occurs。他のインターバルタイマと同様、T1C-Hレジスタへの書き込みから割込み発生までの遅れ時間はタイミングカウンタにロードされたデータによって決まります。タイマ1は割込みを起こすだけでなく、ペリフェラルピンPB7に単発のネガティブパルスを出力することができます。出力イネーブル状態 (ACR7=1) でT1C-Hヘデータを書き込むとPB7は「L」になり、タイマ1のタイムアウトで「H」に戻ります。

〈注〉PB7はDDRB7とACR7の両方によって出力モードに指定されますが、どちらも1になっていると、タイマ1がPB7をコントロールし、ORB7は無効となります。

ワンショットモードでは上位ラッチ1の書き込みは動作に何の影響も与えませんが、下位ラッチへはタイマがT1C-Hへの書き込み操作によってカウントダウンを始める前に正しいデータを入れておく必要があります。プロセッサが上位カウンタヘデータを書き込むと、T1の割込みフラグがクリアされ、下位ラッチの内容が下位カウンタへ送られ、タイマはクロック周波数でカウントダウンを始めます。出力イネーブルになっていればこの時PB7はプロセッサによる書き込みの次のフェーズ2で「L」となります。カウンタの内容がゼロになると、T1の割込

みフラグがセットされ、割込みイネーブルならば \overline{IRQ} が L になり、PB7は L になりPB7は H に戻ります。この時もクロックによるカウントダウンは続いていて、プロセッサがカウンタの内容を読んで割込み発生からの経過時間を知ることができるようになっています。しかしT1割込みフラグは、割込みコントロールの項で述べるように、クリアされていなければ再度セットされることはありません。ワンショットモードのタイミング図を第5図に示します。



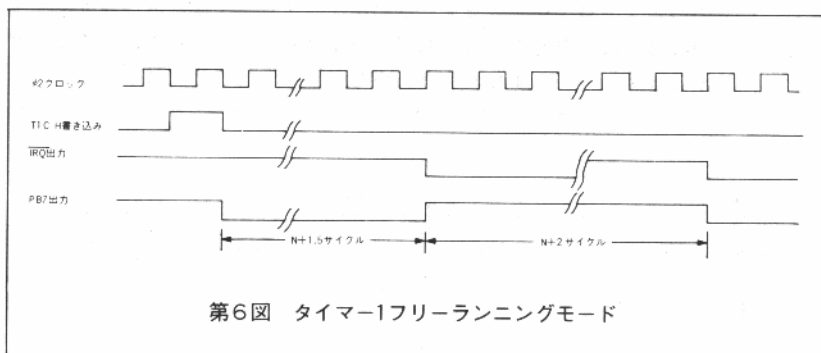
E-6. タイマ1 フリーランニングモード

T1のラッチによる最大の特長は、フリーランニングモードにした時に一定の間隔で連続して割込みをかけることができ、PB7にプロセッサの割込み応答時間の変動によって影響を受けない一定周波数の方形波を出すことができることです。

フリーランニングモード (ACR6=1) では、カウンタがゼロになる毎に割込みフラグがセットされ、PB7の信号が反転します。ゼロになった後、タイマはラッチの内容16ビットをカウンタへ転送し、そこからカウントダウンを始めます。割込みフラグは、T1C-Hへの書き込みか、T1C-Lの読み出しか、後述するフラグへの直接書き込みによってクリアされ、次のタイムアウトの時フラグがセットされるようタイマを書き直す必要はありません。

MCS6500ファミリのインターバルタイマはすべて再トリガが可能です。そしてカウンタの内容を書き替えると、必ずタイムアウトまでの周期も新しくイニシャライズされます。実際にタイマがゼロになる前にプロセッサがタイマの上位カウンタ (T1C-H) の書き替えを続ければ、タイムアウトは決して起りません。しかしラッチへの書き込みは進行中のダウンカウントに影響を与えず、ラッチへ書き

込まれたデータは次のタイムアウトの周期となります。この機能は出力イネーブルの時、特に効果を発揮します。この時はタイムアウトのたびにPB7の信号が反転し、割込みフラグがセットされるので、割込みのたびに新しいデータをラッチに入れて次の半サイクルの周期を決定してやることによってPB7に複雑な波形を発生させることができます。フリーランニングモードのタイミング図を第6図に示します。



F. タイマ2

タイマ2には、ワンショットモードのみのインターバルタイマとしての機能と、ペリフェラルピンPB6に入るネガティブパルスを計数する機能とがあって、補助コントロールレジスタACRのうちの1ビットで切替えます。このタイマは書き込み可能な下位ラッチ(T2L-L)読み出し可能な下位カウンタおよび読み出しと書き込み可能な上位カウンタで構成されています。カウンタレジスタは、φ2でダウンカウントする16ビットカウンタとして働きます。タイマ2の各レジスタのアドレスは次の通りです。

RS3	RS2	RS1	RS0	R/W=0	RW=1
H	L	L	L	T2L-Lへ書き込み	T2C-L読み出し 割込みフラグクリア
H	L	L	H	T2C-Hへ書き込み T2L-LをT2C-Lへ 転送割込みフラグクリア	T2C-H読み出し

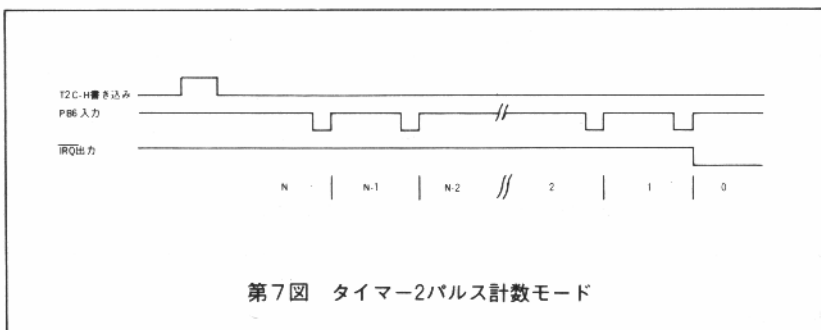
F-1. タイマ2 インターバルタイマモード

インターバルタイマとしては、T2はワンショットモードでT1と同じように働きます。このモードでは、T2C-Hへの書き込みのたびに単発の割込みが発生します。タイムアウト後もカウンタはダウンカウントを続けますが、割込みフラグのセットは、ゼロを過ぎたカウンタが再びセットしないようにディスエーブルされます。これをイネーブルするにはプロセッサはT2C-Hを書き直す必要があります。割込みフラグはT2C-Cの読み出しかT2C-Hへの書き込みでクリアされます。

この動作のタイミング図は第5図です。

F-2. タイマ2パルス計数モード

パルス計数モードでは、T2はあらかじめ決められた数のPB6に入るネガティブパルスを計数します。このため、まずT2に数をロードします。T2C-Hへの書き込みで割込みパルスはクリアされ、カウンタはPB6にパルスが入るたびに減少を始めます。T2がゼロになると割込みフラグがセットされますが、カウンタはさらにPB6のパルスで減少を続けます。次のタイムアウトで割込みフラグをセットさせるためには、T2C-Hを書き替えねばなりません。このモードのタイミングを第7図に示します。パルスはφ2の立ちあがりで「L」でなければなりません。



G. シフト・レジスタ

シフトレジスタは内部の8進カウンタにより、CB2ピンを通じてシリアルデータの送受信をおこないます。シフトパルスは外部からCB1ピンに加えることもできますし、動作モードによっては内部で発生させてCB1ピンに出力し、外部機器でのシフトをコントロールすることもできます。シフトレジスタの動作モードは補助レジスタのビットが決定します。プロセッサはこれらのビットを介して動作モードを決定します。

G-1. シフトレジスタ入力モード

補助コントロールレジスタのビット4は入出力の切替えビットです。シフトレジスタの動作モードは、シフトパルスの発生源によって入力モードに3種類、出力モードに4種類あります。ACR4=0の入力モードは、ACR3とACR2によって次のように決定されます。

ACR4	ACR3	ACR2	モード
0	0	0	シフトレジスタ、ディスエーブル
0	0	1	タイマ2によるシフト入力
0	1	0	システムクロックによるシフト入力
0	1	1	外部パルスによるシフト入力

G-2. モード000シフトレジスタディスエーブル

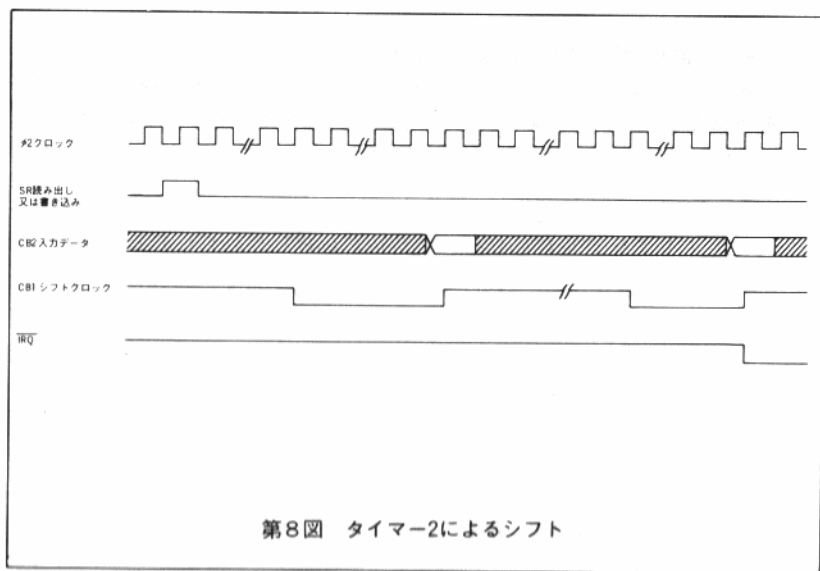
000モードではシフトレジスタはディスエーブルされます。この時プロセッサはシフトレジスタに対し読み出し、書き込みをおこなうことはできませんが、シフト動作はディスエーブルされており、CB1とCB2の動作はペリフェラルコントロールレジスタ (PCR) の該当ビットで決まります。

また、シフトレジスタ割込みフラグもディスエーブルされます。

G-3. モード001タイマ2によるシフト入力

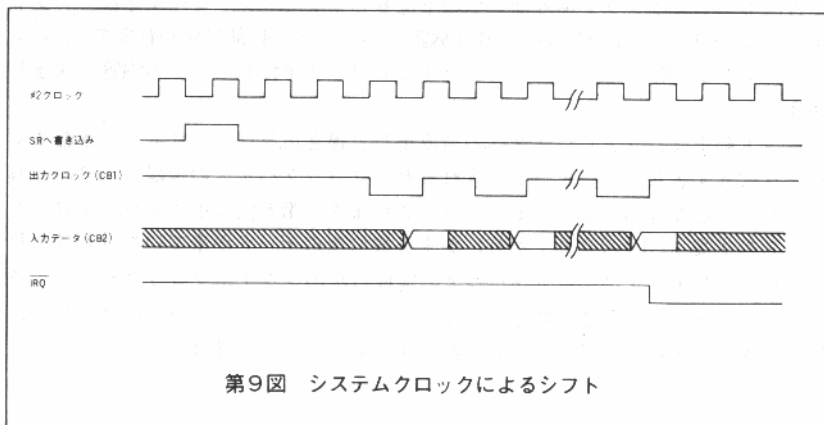
このモードではシフト速度はT2の下位8ビットでコントロールされ、シフトパルスはCB1ピンに出力されて、外部機器でのシフトの制御に使われます。このシフトパルスの周期は、システムロックとT2のラッチ下位8ビットの内容で決まります。

シフト動作はシフトレジスタへの読み出し、書き込みによってトリガされます。データはまずSRの下位ビットに入れられ、クロックパルスの後縁 (CB1の立ち上がり) でとなりの上位ビットにシフトされます。第8図に示すようにクロックパルスの先端 (CB1の立ち上がり) が来る後に次のデータが来ていなければなりません。このデータはクロックパルスの後縁の次のシステムクロックサイクルでシフトレジスタに読み込まれます。そしてクロックパルスが8つ入ると、シフトレジスタの割込みフラグがセットされ、 \overline{IRQ} は「L」になります。



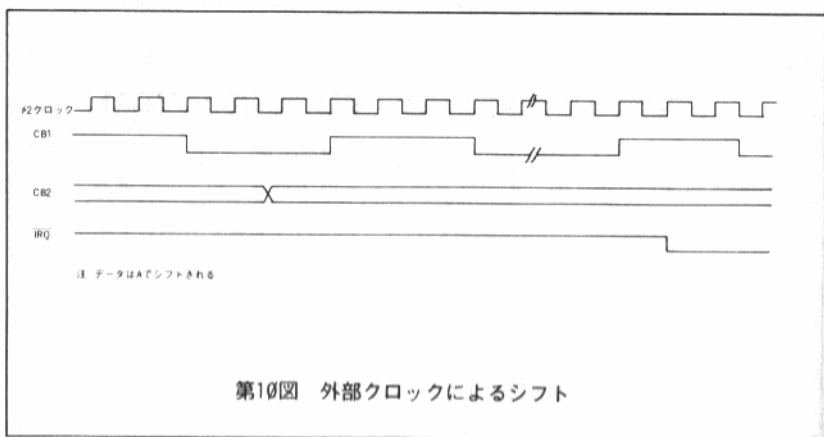
G-4. モード010システムクロックによるシフト入力

このモードではシフトはシステムクロックの周波数でおこなわれます。CB1はシフトパルスを出力し、外部機器をコントロールします。タイマ2はインターバルタイマとして働き、シフトレジスタの動作には関係しません。シフトレジスタへ読み出したり書き込みをおこなうとシフト動作が始まり、データはまずビット0に入り、クロックパルスの後縁で順に高次ビットへ送られます。8クロックパルス経過後、シフトレジスタの割込みフラグがセットされ、CB1のクロックパルス出力は停まります。



G-5. モード011外部クロックによるシフト入力

このモードではCB1はシフトパルスの入力ピンとなって、外部機器が自身の決める転送速度でデータをシフトレジスタに入れることができます。



シフトレジスタのカウンタは8ビット読み込むごとにプロセッサに割込みをかけますが、その後もパルスカウンタとして働き、シフト動作は続きます。シフトレジスタへ読み出したり書き込みをおこなうと割込みフラグはリセットされ、シフトレジスタカウンタはイニシャライズされて次の8ビットをかぞえはじめます。

データは、CB1のシフトパルスの先端の次のシステムクロックサイクルでシフトされます。このため、CB1が‘H’になった後システムクロック1サイクルの間はデータを保持しなければなりません。動作のタイミングを第10図に示します。

G-6. シフトレジスタ出力モード

入出力コントロールビット（ACR4）を1にするとシフトレジスタは出力となり、ACR3とACR2の組合せにより四つのモードが選べます。どのモードでも、シフトレジスタのビット7の内容がCB2ピンに出力され、同時にビット0へシフトバックします。入力モードと同様、CB1はシフトパルスを出力することも、外部から入力することもできます。

4つのモードは次のとおりです。

ACR4	ACR3	ACR2	モード
1	0	0	T2によるフリーランニングモードのシフト出力
1	0	1	T2によるシフト出力、CB1にシフトパルス発生
1	1	0	システムクロックによるシフト出力
1	1	1	外部パルスによるシフト出力

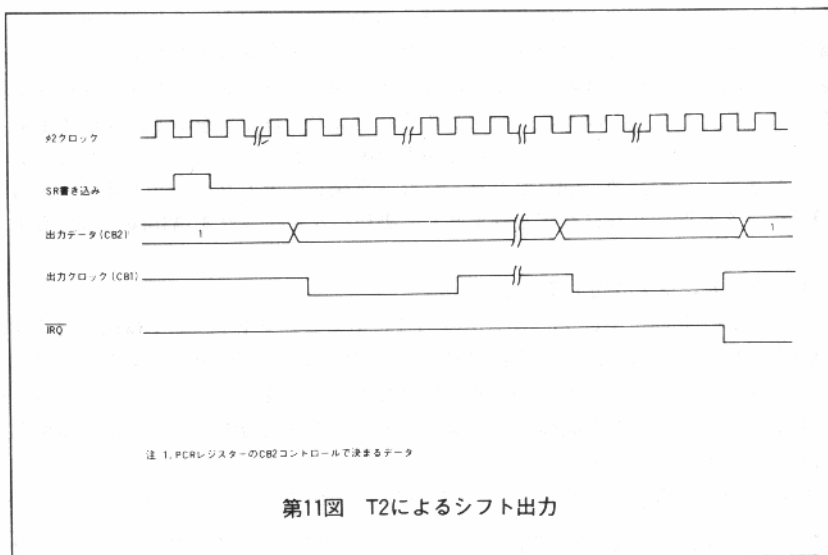
G-7. モード100フリーランニング出力

このモードのシフト速度はモード101と同様T2が決定します。モード101と違うのは、このモードではSRカウンタによってシフト動作が止められることがないという点です。シフトレジスタのビット7（SR7）はビット0に戻されますので、ロードされているデータ8ビットはクロックに従ってCB2に何回でも現われます。このモードではシフトレジスタカウンタはディスエーブルされています。

G-8. モード101T2によるシフト出力

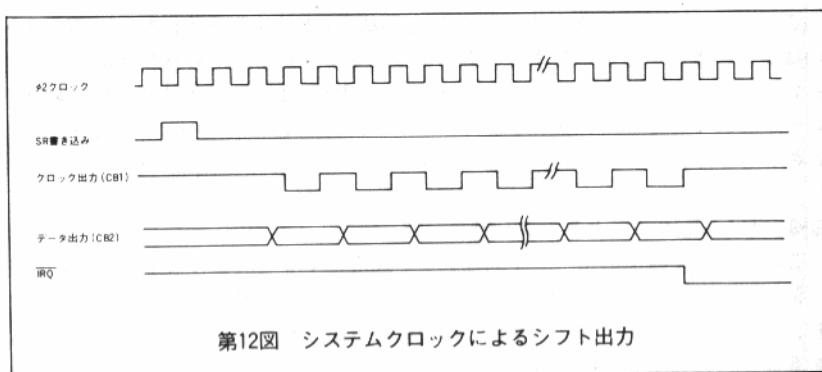
このモードでは前のモードを同様シフト速度はT2で決まります。しかし今度はシフトレジスタへの書き込みまたは読み出しのたびにSRカウンタがリセットされ、データ8ビットがシフトされてCB2に送り出されます。同時にシフトパルスを8発CB1に発生し、外部機器でのシフトをコントロールします。8発のシフトパルスの後はシフト動作は止まり、CB2はペリフェラルコントロールレジスタのCB2コントロールビット（PC5）がコントロールするようになります。

最後のタイムアウトの前にシフトレジスタをロードしなせばシフト動作は続きます。タイミング図を第11図に示します。



G-9. モード110システムクロックによるシフト出力

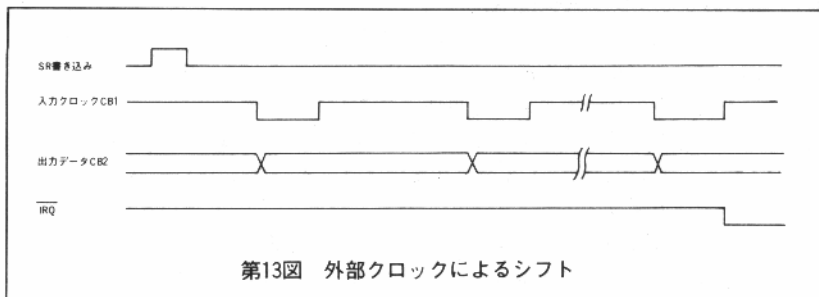
このモードの動作は第11図と似ていますが、シフト速度はφ2ピンのシステムクロックになり、T2はインターバルタイマとしての動作に戻ります。タイミング図を第12図に示します。



G-10. モード111外部パルスによるシフト出力

このモードではシフトは外部機器がCB1ピンに加えるパルスによっておこなわれます。SRカウンタはパルスが8発入るたびにSR割込みフラグをセットしますが、その後もシフトは続きます。プロセッサが読み出しか書き込みをおこなう

たびにSR 割込みフラグはリセットされ、SR カウンタはイニシャライズされて、CB1のシフトパルスのカウントを始めます。シフトパルスが8 発来ると割込みフラグがセットされるので、この時プロセッサは次のデータ1 バイトをロードすることができます。



割込みコントロール

H. 割込みコントロール

MPS6522内部の割込みコントロールには、割込みフラグを立てること、割込みをイネーブルすること、プロセッサに割込みが発生したことを伝えることの3つの動作があります。割込みフラグは、チップ内部で割込み条件が満たされるか、チップに入力があるとセットされ、サービスがおこなわれるまでセットされています。割込みの発生源を知るためにはプロセッサは、フラグレジスタをアキュムレータに読み込んで、左か右にシフトして条件ブランチ命令を実行し、割込みフラグを優先順位の高い方から調べます。

割込みフラグは、割込みイネーブルビットと対になっています。このビットはプロセッサからセット / リセットされ、対応するフラグのプロセッサへの割込みをイネーブルします。割込みの発生によって割込みフラグが1にセットされて、その時相手の割込みイネーブルビットが1になっていれば $\overline{\text{IRQ}}$ は「L」になります。 $\overline{\text{IRQ}}$ はオープンコレクタになっていて、他のデバイスからの割込みリクエストとワイヤードORを取ることができます。

MPS6522では、割込みフラグは1つのレジスタに納められていて、そのレジス

ビット	7	6	5	4	3	2	1	0
割込みフラグ レジスタ	IRQ	T1	T2	CB1	CB2	SR	CA1	CA2
割込みイネーブル レジスタ	セット/クリア コントロール	T1	T2	CB1	CB2	SR	CA1	CA2

タの第7ビットはチップ内で割込みが発生すると1になるようになっています。これによって割込み発生源が分散しているシステムでのポーリングが容易におこなえます。

H-1. 割込みフラグレジスタ (IFR)

IFRは、読み出しと、ビットクリアのできるレジスタです。チップセレクトされてこのレジスタが指定されると、IFRの内容はデータバスに乗ります。ビット7は \overline{IRQ} の状態を示し、次のような論理式で表わされます。

$$IRQ = IFR6 \times IER6 + IFR5 \times IER5 + IFR4 \times IER4 + \dots + IFR0 \times IER0$$

(\times はAND、 $+$ はOR) ビット6から0まではラッチになっていて、次の条件でセット/クリアされます。

IFRのビット7はフラグではないので、ここに1を書き込んでもクリアすることはできません。このビットをクリアするには、レジスタの他のビットをクリアするか、次の章で説明するように、起っている割込みをすべてディスエーブルするかしなければなりません。

ビット	セット条件	クリア条件
0	CA2ピンの信号のアクティブ・トランジション	Aポート出力レジスタ (ORA) への読み出し又は書き込み
1	CA1ピンの信号のアクティブ・トランジション	Aポート出力レジスタ (ORA) への読み出し又は書き込み
2	シフト8回終了	シフトレジスタへの読み出し又は書き込み
3	CB2ピンの信号のアクティブ・トランジション	Bポート出力レジスタ (ORB) への読み出し又は書き込み
4	CB1ピンの信号のアクティブ・トランジション	Bポート出力レジスタ (ORB) への読み出し又は書き込み
5	タイマ2のタイムアウト	T2下位カウンタ読み出し T2上位カウンタ書き込み
6	タイマ1のタイムアウト	T1下位カウンタ読み出し T1上位カウンタ書き込み

H-2. 割込みイネーブルレジスタ (IER)

割込みイネーブルレジスタの各ビットは、IFRの各割込みフラグに対応しています。プロセッサはアドレス1110 (IERアドレス) への書き込みで、このレジスタのビットを個別にセットしたりクリアしたりすることによって、他に影響を与えずに個別に割込みをコントロールすることができます。

IERへ書き込むデータのビット7が0ならば、他の1のビットの割込みイネーブルはクリアされ、0のビットについては変化を生じません。逆にIERへ書き込むデータのビット7が1ならば、他の1のビットの割込みイネーブルはセットされ、0のビットについては変化を生じません。このビットを個別にセット、クリアできる機能はシステムの割込みコントロールに大変便利です。

プロセッサはこの他にIERの内容を読み出すこともできます。この時ビット7は0となります。

機能コントロール

1. 機能コントロール

MPS6522の機能と動作モードは、ペリフェラルコントロールレジスタ (PCR) と補助コントロールレジスタ (ACR) の二つのレジスタが主にコントロールします。PCRは4本のペリフェラルコントロールピンの動作を、ACRは二つのインターバルタイマ (T1、T2) と直列ポートの動作モードを主にコントロールします。

1-1. ペリフェラルコントロールレジスタ

ペリフェラルコントロールレジスタは次のような構成になっています。

ビット	7	6	5	4	3	2	1	0
機能	CB2コントロール			CB1 コントロール	CA2コントロール			CA1 コントロール

これらの機能を詳しく説明します。

1. CA1コントロール

PCRのビット0は、CA1の割込み入力信号のアクティブトランジションの方向を決めます。もしこのビットが0ならCA1の割込みフラグは入力パルスの立ち下がり、(HからL)でセットされ、1なら立ち上がり (LからH)でセットされます。

2. CA2コントロール

CA2ピンは割込み入力としてもペリフェラルコントロールとしても使えます。入力モードには割込みフラグのリセットの方法によって異なる2つのモードがあり、どちらもCA1と同様アクティブトランジションの方向を切替えることができます。

出力モードではCA2をCB2と組合わせて働かせることができます。これによってプロセッサはCB1とCB2が先に説明したシリアル動作をするシステムで通常の

PCR3	PCR2	PCR1	モード
0	0	0	入力モード CA2割込みフラグ (IFR0) は入力パルスの立ち下がりでセットされ、ORAへの読み出し又は書き込みでクリアされる。
0	0	1	独立割込み入力モード IFR0はCA2の入力パルスの立ち下がりでセットされるが、ORAへ読み出し、書き込みを行なってもクリアされない。
0	1	0	入力モード IFR0はCA2の入力パルスの立ち上がりでセットされ、ORAへ読み出し、書き込みを行なえばクリアされる。
0	1	1	独立割込み入力モード IFR0はCA2の入力パルスの立ち上がりでセットされるが、ORAへ読み出し、書き込みを行なってもクリアされない。
1	0	0	ハンドシェイク出力モード CA2はORAへの読み出し又は書き込みによって“L”にセットされ、CA1に入力パルスが来るとそのアクティブトランジションで“H”に戻る。
1	0	1	パルス出力モード。ORAの読み出し、書き込みを行なうと、次の1サイクルだけCA2は“C”になる。
1	1	0	マニュアル出力モード。CA2を“L”にする。
1	1	1	マニュアル出力モード CA2を“H”にする。

書き込みハンドシェイクをさせることができるのです。CA2の動作モードを上に表示します。

独立割込み入力モードでは、ORAレジスタへ読み出し、書き込みをおこなっても、CA2割込みフラグはクリアされません。このフラグをクリアするには、IFRの当該ビットに1を書き込んでやらねばなりません。このモードによってプロセッサは、ペリフェラルI/Oポートでおこなわれている動作からは独立して割込み信号を受け付けることができます。

ハンドシェイクとパルス出力モードについては前に説明した通りです。出力信号のタイミングは、動作が読み出しか書き込みかによって少し違います。

3. CB1コントロール

CB1のアクティブ、トランジションのコントロールはCA1の場合とまったく同じです。CB1割込みフラグ (IFR4) は、PCR4が0ならば、CB1入力の立ち下がり、1ならば立ち上がりでセットされ、どちらの場合もORBレジスタの読み出しまたは書き込みでクリアされます。

シフトレジスタがイネーブルされていると、CB1はシフトレジスタのクロックの入出力ピンとなりますが、この時もIFR4はCB1の信号のどちらかのトランジションに対して応答します。

4. CB2コントロール

シリアルポートがディスエーブルの時、CB2ピンの動作はPCRの上位3ビット

が決定します。CB2のモードはCA2の場合とほとんど同じです。

PCR7	PCR6	PCR5	モード
0	0	0	割込み入力モード：CB2の割込みフラグ(FR3)はCB2入力の立ち下がりでセットされ、ペリフェラルBの出力レジスタ(ORB)への読み出し又は書き込みでクリアされる。
0	0	1	単独割込み入力モード：FR3はCB2入力パルスの立ち下がりでセットされるが、ORBへの読み出し又は書き込みではクリアされない。
0	1	0	割込み入力モード：FR3はCB2入力パルスの立ち上がりでセットされ、ORBへの読み出し又は書き込みでクリアされる。
0	1	1	単独割込み入力モード：FR3はCB2入力パルスの立ち上がりでセットされるが、ORBへの読み出し、書き込みではクリアされない。
1	0	0	ハンドシェイク出力モード：CB2は、ORBへの書き込みで“L”になり、CB1入力パルスのアクティブ・トランジションで“H”に戻る。
1	0	1	パルス出力モード：ORBへの書き込み後の1サイクルだけCB2が“L”になる。
1	1	0	マニュアル出力モード：CB2を“L”にする。
1	1	1	マニュアル出力モード：CB2を“H”にする。

1-2. 補助コントロールレジスタ

補助コントロールレジスタの機能についてはすでにほとんど説明してありますがここに改めて一括して説明します。このレジスタの構成は次の通りです。

ビット	7	6	5	4	3	2	1	0
機能	T1コントロール		T2 コントロール	シフトレジスタ コントロール			PBラッチ イネーブル	PAラッチ イネーブル

1. PAラッチイネーブル

MPS6522では、PAポートもPBポートも入力をラッチすることができます。このモードでは、PAポートの入力ピンのデータはCA1の割込みフラグがセットされた時点でラッチされ、PAポート読み出し時にプロセッサへ転送されます。CA1の割込みフラグがセットされている状態でPAポートの入力データが変化してもラッチの内容は変わりません。この入力ラッチはCA2が入力モードでも出力モードでも使えます。

PAポートではプロセッサはペリフェラルピンのデータを読む時は常にラッチを介します。このためPAポートが出力モードの時は読み直す時にORAの内容がラッチに正しく入っているとは限りません。このようにPAポートで出力モードと入力ラッチを組合せて使う時はシステム動作の設計上注意が必要です。

2. PBラッチイネーブル

PBポートの入力ラッチも、PAポートの場合と同様にコントロールされます。ただしPBポートでは、ラッチの同容がペリフェラルピンの電圧になるかORBの内容になるかはそのピンが入力モードか出力モードかによって違います。そしてPAポートの場合と同様、プロセッサは常にラッチの内容を読みます。

3. シフトレジスタコントロール

シフトレジスタの動作モードは次のように決定されます。

ACR4	ACR3	ACR2	モード
0	0	0	シフトレジスタディスエーブル
0	0	1	タイマ2によるシフト入力
0	1	0	システムクロックによるシフト入力
0	1	1	外部クロックによるシフト入力
1	0	0	タイマ2によるフリーランニング出力
1	0	1	タイマ2によるシフト出力
1	1	0	システムクロックによるシフト出力
1	1	1	外部クロックによるシフト出力

4. T2コントロール

タイマ2の動作には2つのモードがあります。ACR5が0だとワンショットモードのインターバルタイマになり、1だとPB6の入力パルスをあらかじめ与えられた数だけカウントします。

5. T1コントロール

タイマ1の動作モードは、ワンショットかフリーランニングかの切替えと、PB7の出力をイネーブルにするかディスエーブルにするかの切替えで下の表のように決まります。

ACR7 出力イネーブル	ACR6 フリーランニング イネーブル	モード
0	0	T1がロードされる毎にタイムアウトで単発の割込み発生 PB7はディスエーブル
0	1	連続割込み発生。PB7はディスエーブル
1	0	T1がロードされる毎に単発割込みを発生し、 PB7にパルスを出力
1	1	連続割込みを発生し、PB7に方形波出力

MPS6522の応用

MPS6522はマイクロプロセッサ用汎用I/Oとして大変優れていますが、その強力な機能は複雑な動作モードと結びついているため、簡単な説明だけでは理解が困難です。ここでは、このチップがマイクロプロセッサシステムの中でどのように使われるかを示して、システム設計の理解の助けとします。

A. MPS6522の割込みコントロール

MPS6522では割込みサービスを容易におこなえるよう、すべての割込みフラグは一つのレジスタにまとめられています。チップ内の7つの割込み発生源に対し、IRQ出力は1本なので、プロセッサは割込みを起した発生源を知るためにフラグを調べなければなりません。このためフラグレジスタの内容をアキュムレータに読み込むのですが、その時場合によっては割込みイネーブルレジスタでディスエーブルされているフラグをマスクして除去しておく必要があります。これは特に割込みがイネーブルでもディスエーブルでもフラグがセットされてしまうエッジ検出の入力動作では重要です。フラグをマスクするには、アキュムレータとIERとのANDを取るか、ANDイミディエート命令を実行します。

この結果フラグがセットされていれば、チップ内で実際に割込みが起っています。この割込みを検出するにはシフトとブランチ命令を続けて実行します。

割込みフラグをクリアするのは簡単で、割込みレジスタの当該ビットに1を書けばよいのです。これは割込みイネーブル/ディスエーブル操作といっしょに次のようにおこなえます。

```
LDA #@10010000 : アキュムレータイニシャライズ
STA IFR        : 割込みフラグクリア
STA IER        : 割込みイネーブルフラグセット
```

または、

```
LDA #@00001000 : アキュムレータイニシャライズ
STA IFR        : 割込みフラグクリア
STA IER        : 割込みディスエーブル
```

もう一つの方法は、読み込んだフラグレジスタの内容をそのまま送り返してやるものです。

```
LDA IFR        : IFRからアキュムレータへ転送
STA IER        : 発生した割込みのフラグをクリア
```

この操作の後もアキュムレータには割込みフラグの情報は残ります。そしてフラグレジスタへの書き込みですでにセットされているフラグだけがクリアされますので、セットされつつあるフラグをクリアしてしまうという誤動作の可能性が少なくなります。

B. タイマ1使用例

タイマ1もMPS6522の強力な機能の一つです。一定間隔での割込み動作とPB7の電圧のコントロール機能によって、いろいろなタイミング動作、データ検出、波形発生などをおこなわせることができます。

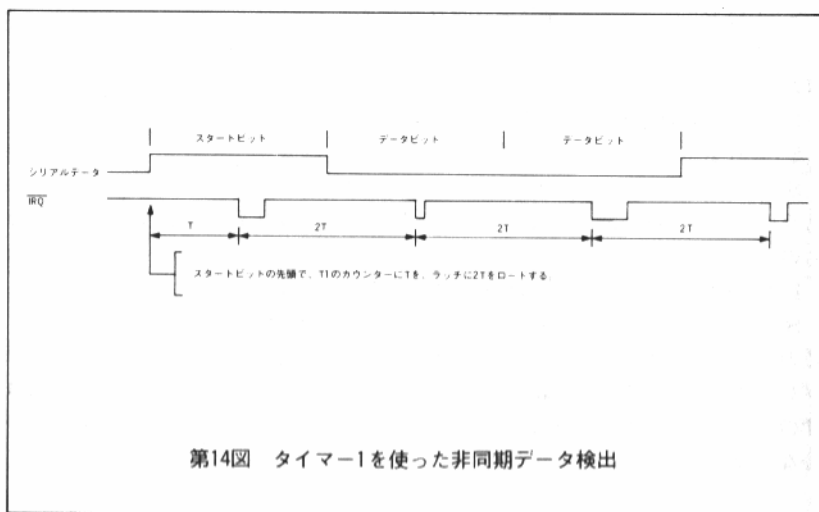
B-1. 時刻表示時計

時刻を表わす時計の機能が必要な場合がよくあります。マイクロプロセッサシステムでは、この機能は通常メモリの中に置かれて、プロセッサに定期的に割込みをかけて割込みサービスルーチンで校正し、メインプログラムで必要な時はいつでも時刻が得られるようにします。

これまでのタイマを使った定期的な割込み動作では、割込みのたびにタイマをロードしなければなりません。さらに、割込みへの応答時間が一定しないと割込みの周期も変動しました。このような問題はタイマ1のフリーランニングモードの割込みではすべて解決されて、クロック周波数の変動以外の時計の誤差の原因はなくなりました。

B-2. 非同期データの検出

非同期シリアルのアスキー信号や、データ収録機器からのシリアルデータ、クロックの検出では、正確にストロブパルスを発生することが重要です。従来のタイマでは前に述べた通り割込み応答時間によってストロブの周期が変動しますが、T1ならば正確な割込みを起すことができ、プロセッサはこの割込みに対して入力データをストロブします。この場合、進行中のカウントダウン動作に



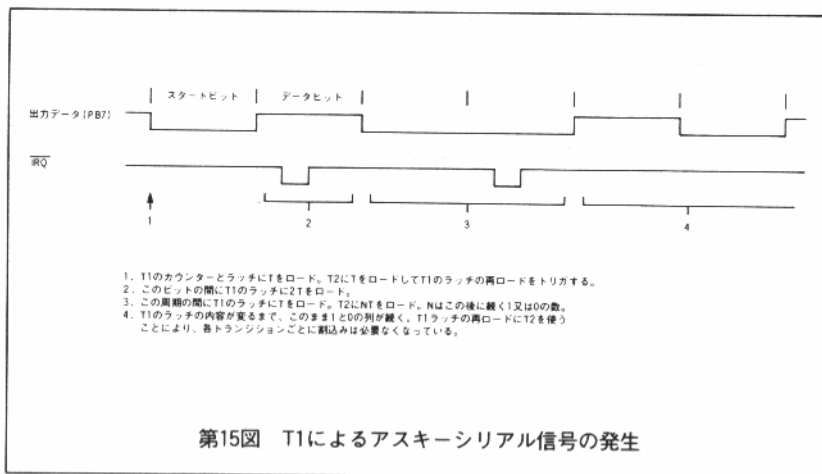
関係なくT1のラッチをロードしなおすことができるのは大変有利で、データ検出の動作中に次のストロブ時間を倍にしたり半分にしたりすることができます。この動作のタイミング図を第14図に示します。

B-3. タイム1による波形発生

T1はプロセッサに割込みをかけるだけでなく、ペリフェラルピンPB7の出力電圧をコントロールすることもできます。ワンショットモードでは単発のネガティブパルスを、フリーランニングモードでは連続波形を発生します。フリーランニングの時はT1のタイムアウトのたびにPB7のレベルが反転します。

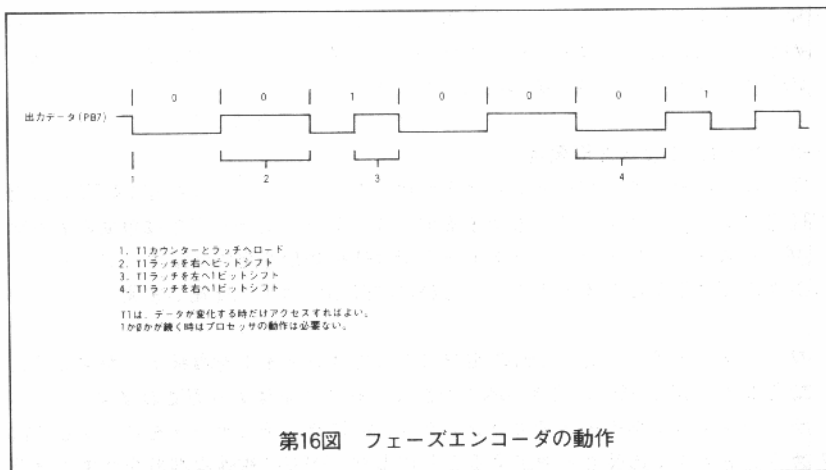
ワンショットモードで、PB7の出力によってソレノイドを直接トリガすることができます。T1C-Hへの書き込みのたびにソレノイドはトリガされます。

フリーランニングモードでカウントダウン中にラッチにデータをロードして次のカウントダウンの長さを決定することにより、PB7に複雑な波形を発生させることもできます。この方法でアスキーのシリアルデータを発生させるタイミング図を第15図に示します。



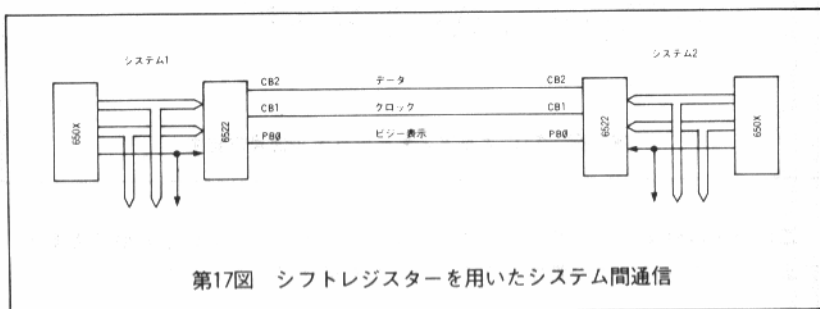
この方法は磁気テープやディスクのフェーズ・エンコーダにも応用できます。フェーズ・エンコーダの動作のタイミング図を第16図に示します。

ここに示した応用例はT1の能力から見ればほんの一部に過ぎません。他にも、パルス幅変調、ビデオゲームの効果音の発生、正確なパルス幅の必要なA/D技術、電子ゲームの波形発生などへの応用が可能です。

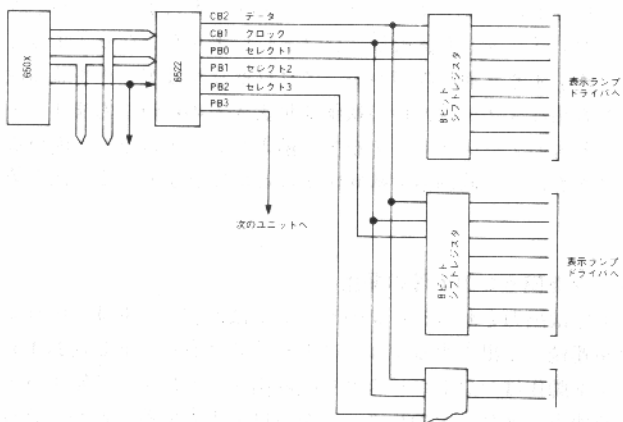


C. シフトレジスタの応用

MPS6522のシフトレジスタは、主にシステム間の同期式シリアルデータ通信用に設計されています。通信をおこなうシステムはシングル・プロセッサで複数のペリフェラル・コントローラを持つシステムの場合もありますしマルチプロセッサ・システムの場合もあります。このシフトレジスタは、CRによってノイズを抑えて比較的低速でのデータ転送に適しており、データ転送はプロセッサが他の仕事をしている間におこなうことができます。2つのプロセッサを持つシステムの例を第17図に示します。MPS6522のシフトレジスタにより、複雑な非周期通信技術を使わないでシステム間の通信をおこなっています。

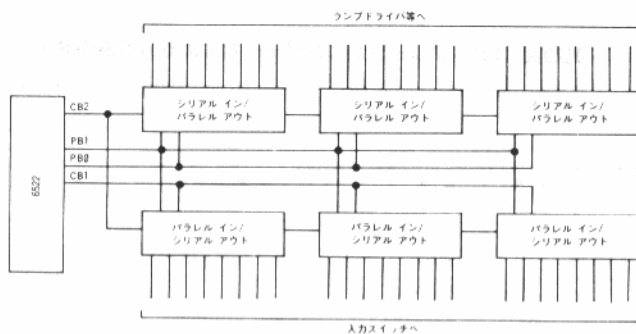


複数の周辺機器を持つシステムでは、シフトレジスタを使ってこれらのインターフェイスにデータを送ることができます。第18図に、複数のステータス表示を



第18図 シフトレジスタを使用した表示インターフェイス

持ったシステムの例を示します。このシステムではそれぞれ独立したコントローラが簡単なドライバを介してステータス表示のランプを働かせます。データ線とクロック線は各ユニットに並列に接続され、さらにポートBの出力でデータを入れるユニットを選択していますが、どのユニットも同じ表示をするならこの選択線は必要ありません。図に示したシステムでは、ユニットを指定し、シフトレジスタへデータを書き込むことによって表示を新しくなっています。



第19図 シフトレジスターによって拡張したI/O

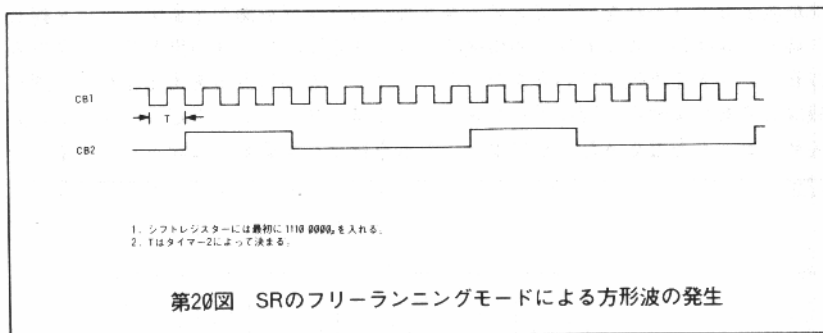
入力機器の制御もまったく同様で、第18図のようにペリフェラルポートの出力のコントロールにより、データをシフトレジスタへ入力することができます。まず読みたい入力のユニットを選択し、シフトレジスタをアクセスするとシフト動作が始まり、終了すると割込みがかってプロセッサはデータを読み取ります。

上に述べた方法はシステムのI/O機能を拡大するのに役立ちます。ステータス表示のランプと入力データスイッチを多く備えたシステムでも、簡単なTTLのシフトレジスタを使って低価格で構成することができます。この例を第19図に示します。

シフトレジスタを使った方形波の発生

出力モードでは出力されるデータのビット7は再びビット0に戻りますので、同じデータを連続して出力する時はシフトレジスタをロードしなおす必要はありません。シフト動作はシフトレジスタを読み出すことによって始まります。

この機能を使うと8ビットのパターンを続けて出力することができるので、周辺機器のクロックとして使えます。この方法を第20図に示します。この時シフト動作はタイマ2によってコントロールされるので、1ビットの時間は最大でクロックパルスの256倍となります。



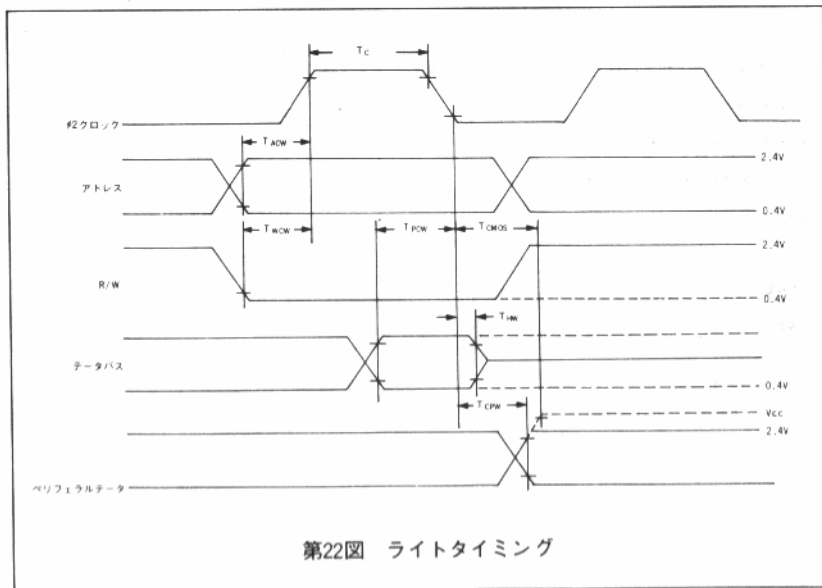
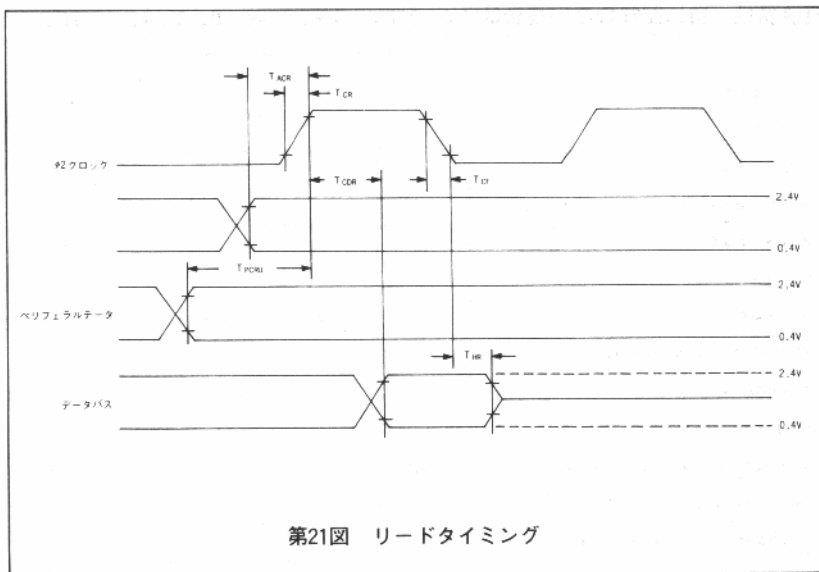
最大定格

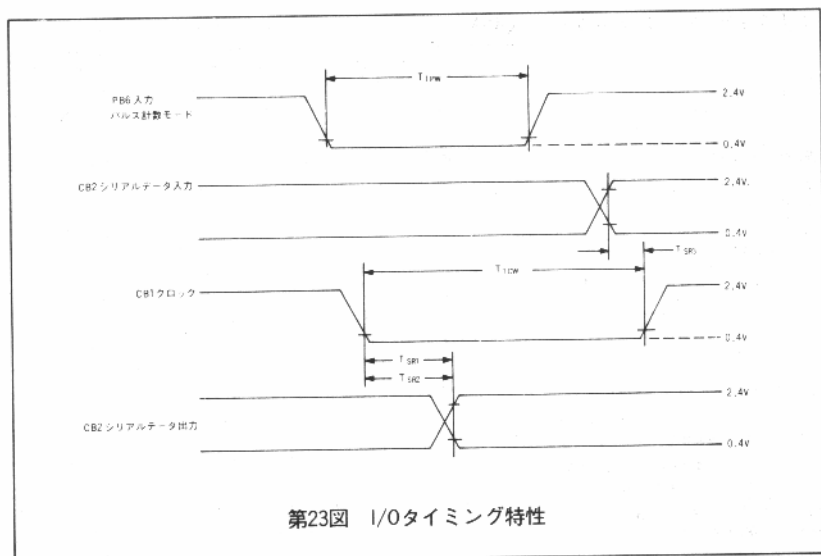
項目	記号	定格	単位
電源電圧	V _{cc}	-0.3~+7.0	Vdc
入力電圧	V _{in}	-0.3~+7.0	Vdc
動作温度	T _A	0~70	°C
保存温度	T _{stg}	-55~+150	°C

このチップには静電気に対する保護回路がついていますが、最大定格を越える電圧をかけないように注意して下さい。

直流特性 (特に記載のない場合は $V_{CC}=5.0V \pm 5\%$ 、 $U_{SS}=0$ 、 $T_A=0 \sim +70^\circ C$)

項 目	記 号	最 小	標 準	最 大	単 位
H. 入力電圧	V_{IH}	+2.4	—	V_{CC}	Vdc
H. 入力電圧	V_{IL}	-0.3	—	+0.4	Vdc
入力リーク電流 $V_{in}=0 \sim 5V_{DC}$ R/W, RE5, RS0, RS1, RS2, RS3, CS1, CS2, CA1, 02	I_{IN}	—	± 1.0	± 2.5	μA_{DC}
オフ状態入力電流 $V_{in}=0.4 \sim 2.4V$ $V_{CC}=\text{Max}$, D0, D7	I_{ISI}	—	± 2.0	± 10	μA_{DC}
H. 入力電流 $V_{IH}=2.4V$ PA0-PA7, CA2, PB0-PB7, CB1, CB2	I_{IH}	-100	-250	—	μA_{DC}
L. 入力電流 $V_{IL}=0.4V$ PA0-PA7, CA2, PB0-PB7, CB1, CB2	I_{IL}	—	-1.0	-1.6	mA _{DC}
H. 出力電圧 $V_{CC}=\text{最小}$ Iload=-100uA PA0-PA7, CA2, PB0-PB7, CB1, CB2	V_{OH}	2.4	—	—	Vdc
L. 出力電圧 $V_{CC}=\text{最小}$ Iload=1.6mA _{DC}	V_{OL}	—	—	+0.4	Vdc
H. 出力電流 (ソース) $V_{OH}=2.4V$ $V_{OH}=1.5V$, PB0-PB7, CB1, CB2	I_{OH}	-100 -3.0	-1000 -5.0	—	μA_{DC} mA _{DC}
L. 出力電流 (シンク) $V_{OL}=0.4V$	I_{OL}	1.6	—	—	mA _{DC}
オフ状態 出力リーク電流 IRQ	I_{OFF}	—	1.0	10	μA_{DC}
入力容量 $T_A=25^\circ C$ $f=1MHz$ R/W, RE5, RS0, RS1, RS2, RS3, CS1, CS2 D0-D7, PA0-PA7, CA2, PB0-PB7, CB1, CB2 02	C_{in}	—	—	7.0 10 20	PF PF PF
出力容量 $T_A=25^\circ C$ $f=1MHz$	C_{out}	—	—	10	PF
消費電力	P_d	—	—	1000	MW





第23図 I/Oタイミング特性

交流特性

リードタイミング (第21図、負荷130pFおよび1 TTLロード)

項目	記号	最小	標準	最大	単位
アドレス確定からクロック立ち上がりまでの遅延時間	T_{ACR}	180	—	—	ns
クロック立ち上がりからバスにデータが乗るまでの遅延時間	T_{CDR}	—	—	395	ns
ベリフェアルデータのセットアップ時間	T_{PCR}	300	—	—	ns
データバス保持時間	T_{HR}	10	—	—	ns
クロック入力立ち上がり、立ち下がり時間	T_{CR} T_{CF}	—	—	25	ns

ライトタイミング (第22図)

項目	記号	最小	標準	最大	単位
イネーブルパルス巾	T_C	0.47	—	25	μ s
アドレス確定からクロック立ち上がりまでの遅延時間	T_{ACW}	180	—	—	ns
データ確定からクロック立ち下がりまでの遅延時間	T_{DCW}	300	—	—	ns
R/W立ち下がりからクロック立ち上がりまでの遅延時間	T_{WCW}	180	—	—	ns
データバス保持時間	T_{HW}	10	—	—	ns
イネーブル立ち下がりからベリフェアルデータ確定までの遅延時間	T_{CPW}	—	—	1.0	μ s
CMUS (Vcc30%)におけるクロック立ち下がりからデータ確定までの遅延時間	T_{CMOS}	—	—	2.0	μ s

ペリフェラルインターフェース特性

項 目	記 号	最 小	標 準	最 大	単 位
CA1, CB1, CA2, CB2 入力の立ち上がり、立ち下がり時間	T _{RF}	—	—	1.0	μs
クロック立ち下がりからCA2立ち下がりまでの遅延時間(読み出しハンドシェイク又はパルスモード)	T _{CA2}	—	—	1.0	μs
クロック立ち下がりからCA2立ち上がりまでの遅延時間(パルスモード)	T _{RS1}	—	—	1.0	μs
CA1 アクティブ・トランジションからCA2立ち上がりまでの遅延時間(ハンドシェイクモード)	T _{RS2}	—	—	2.0	μs
クロック立ち上がりからCA2又はCB2立ち下がりまでの遅延時間(書き込みハンドシェイク)	T _{WHS}	—	—	1.0	μs
ペリフェラルデータ確定からCB2立ち下がりまでの遅延時間	T _{DC}	0	—	1.5	μs
クロック立ち上がりからCA2, CB2立ち上がりまでの遅延時間(パルスモード)	T _{RS3}	—	—	1.0	μs
CB1 アクティブ・トランジションからCA2, CB2立ち上がりまでの遅延時間(ハンドシェイクモード)	T _{RS4}	—	—	2.0	μs
ペリフェラルデータ確定からCA1, CB1 アクティブ・トランジションまでの遅延時間(入力ラッチ)	T _{IL}	300	—	—	ns
CB1立ち下がりからCB2データ確定からCB2データ確定までの遅延時間(内部SRクロック、シフト出力)	T _{SR1}	—	—	300	ns
CB1入力クロック立ち下がりからCB2データ確定までの遅延時間(外部クロック・シフト出力)	T _{SR2}	—	—	300	ns
CB2データ確定からCB1クロック立ち上がりまでの遅延時間(外部又は内部クロックのシフト入力)	T _{SR3}	—	—	300	ns
PB6入力パルスのパルス巾	T _{IPW}	2	—	—	μs
CB1入力クロックのパルス巾	T _{ICW}	2	—	—	μs
PB6入力パルスの間隔	T _{IPS}	2	—	—	μs
CB1入力パルスの間隔	T _{ICS}	2	—	—	μs

MPS 2364

スタティックROM(8192×8)

概要

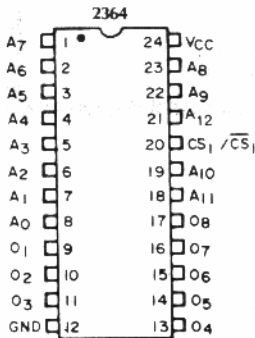
MPS 2364は、65,536ビットのスタティック、リード・オンリーメモリーで、8192×8ビットで構成されています。アクセスタイムが、短い(最大350ns)という特徴をもちます。ハイ・パフォーマンス、大容量、簡単なインタフェイスが重要視される応用製品、およびマイクロプロセッサとコンパチブルであるように設計されています。

2364は、非同期に動作し、クロック入力が必要としません。プログラマブル・チップ・セレクト入力1つにより、2つの64KROMが外部のデコードに回路なしにORで結ばれます。2個の2732EPROMとおきかえることにより、いったんEPROMでプロトタイプされたプリント板をマスク・プログラムROMにかえることによる再設計の必要がありません。

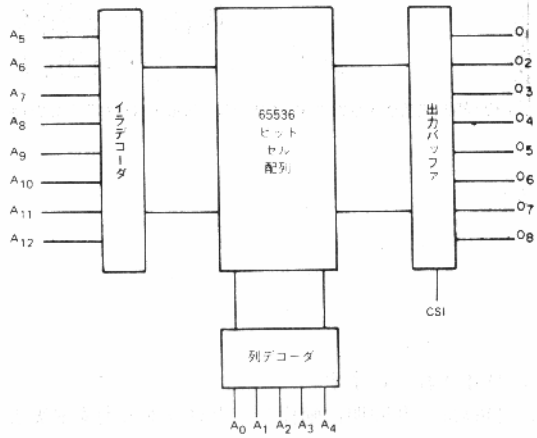
特長

- アクセスタイムは450nsおよび350ns
- スタティック動作
- TTLコンパチブル
- ワイヤーOR増設に対してスリーステート出力
- プログラマブル・チップ・セレクト
- 5V単一電源
- 2716および2732EPROMとピンコンパチブル
- 400mV入力ノイズを無視
- 2716 / 2732EPROMをプログラムデータ入力として受入可能

ピン配置図



ブロック図



直流特性 (特に記載のない場合は、 $T_A = 0^\circ\text{C} \sim +70^\circ\text{C}$ 、 $V_{CC} = 5.0\text{V} \pm 5\%$)

記号	項目	最小	最大	単位	テスト条件
I_{CC1}	供給電流		125	mA	$V_{IN} = V_{CC}$, $V_O = \text{オープン}$, $T_A = 0^\circ\text{C}$
I_{CC2}	供給電流		120	mA	$V_{IN} = V_{CC}$, $V_O = \text{オープン}$, $T_A = 25^\circ\text{C}$
I_O	出力リーク電流		10	μA	チップ・ディセレクト、 $V_O = 0$ から V_{CC}
I_I	入力ロード電流		10	μA	$V_{CC} = \text{最大}$, $V_{IN} = 0$ から V_{CC}
V_{OL}	"L"出力電圧		0.4	Volt	$V_{CC} = \text{最小}$, $I_{OL} = 2.1\text{mA}$
V_{OH}	"H"出力電圧			Volt	$V_{CC} = \text{最小}$, $I_{OH} = -400\mu\text{A}$
V_{IL}	"L"入力電圧	-0.5	0.8	Volt	注1参照
V_{IH}	"H"入力電圧	2.0	$V_{CC} + 1$	Volt	

交流特性 (特に記載のない場合は、 $T_A = 0^\circ\text{C} \sim +70^\circ\text{C}$ 、 $V_{CC} = 5.0\text{V} \pm 5\%$)

記号	項目	2364		2364A		単位	テスト条件
		最小	最大	最小	最大		
t_{ACC}	アドレス・アクセス時間		450		350	nS	
t_{CO}	チップ・セレクト遅延時間		280		200	nS	注2参照
t_{DF}	チップ・ディセレクト遅延時間		175		175	nS	
t_{OH}	アドレス変化からデータの変化までの時間	40		40		nS	

キャパシタンス ($T_A = 25^\circ\text{C}$ 、 $f = 1.0\text{MHz}$ 、注3参照)

記号	項目	最小	最大	単位	テスト条件
C IN	入力キャパシタンス		8	pF	テストピン以外はすべてACグランド接地
C OUT	出力キャパシタンス		10	pF	

注1. -0.5V以下の入力電圧は避けるべきで、チップの破壊の原因となります。

注2. 1TTL+pF使用、入力変化時間: 20ns

タイミング測定: 入力1.5V、出力0.8Vと2.0V、 $C_L = 100\text{pF}$

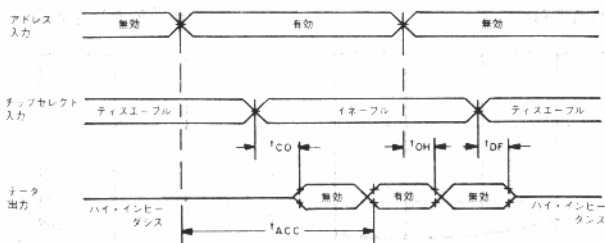
注3. パラメータは周期的なサンプルであり、100%のテストではありません。

最大定格

項目	定 格
動作温度	0°C ~ 70°C
保存温度	-65°C ~ 150°C
電源電圧	-0.5V ~ 7.0V
出力電圧	-0.5V ~ 7.0V
入力電圧	-0.5V ~ 7.0V
消費電力	1.0W

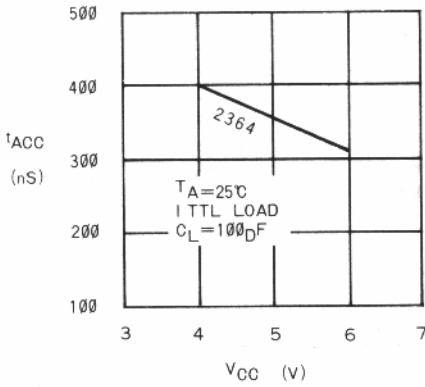
(注) 最大定格を越える場合は、チップを破壊する原因になります。これは、耐久度を述べているのであり、それまたはそれ以上の条件で、本仕様述べている動作をするというではありません。

タイミング図

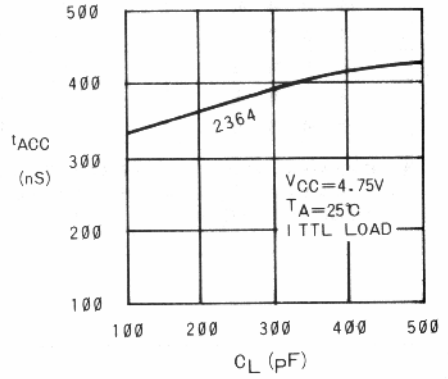


代表特性

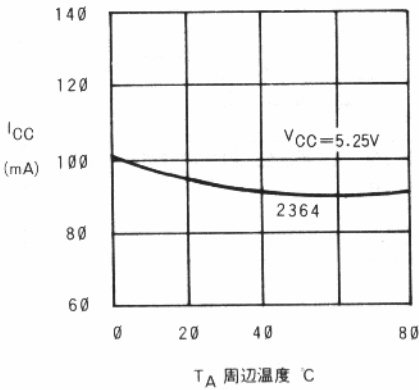
アクセスタイム対電源電圧



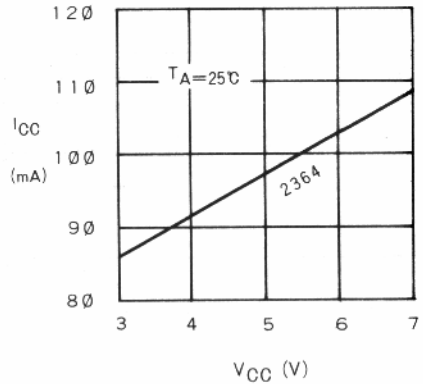
アクセスタイム対
キャパシティブ・ロード



供給電流対周辺温度



供給電流対電源電圧



スクリーン・レイアウト・シート

Program	Programmer	Date
1		22
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		
21		
22		
23		

①スクリーン・エリアのスタート番地は\$E1000=7680、番地または\$E1000=(4096)番地、(\$2000以上にRAMを増設した場合)
②カラー指定のためのエリアのスタート番地は、\$9600=(38400)番地

8×8ドット・パターン作成シート

	7	6	5	4	3	2	1	0	
0									
1									
2									
3									
4									
5									
6									
7									
	128	64	32	16	8	4	2	1	

	7	6	5	4	3	2	1	0	
0									
1									
2									
3									
4									
5									
6									
7									
	128	64	32	16	8	4	2	1	

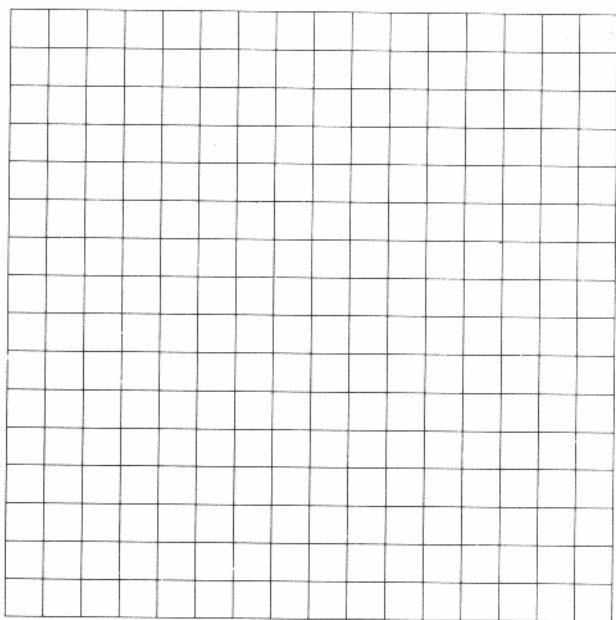
	7	6	5	4	3	2	1	0	
0									
1									
2									
3									
4									
5									
6									
7									
	128	64	32	16	8	4	2	1	

	7	6	5	4	3	2	1	0	
0									
1									
2									
3									
4									
5									
6									
7									
	128	64	32	16	8	4	2	1	

	7	6	5	4	3	2	1	0	
0									
1									
2									
3									
4									
5									
6									
7									
	128	64	32	16	8	4	2	1	

	7	6	5	4	3	2	1	0	
0									
1									
2									
3									
4									
5									
6									
7									
	128	64	32	16	8	4	2	1	

8×16ドット・パターン作成シート



3/4/10

	7	6	5	4	3	2	1	0
0								
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								
11								
12								
13								
14								
15								
	128	64	32	16	8	4	2	1

	7	6	5	4	3	2	1	0
0								
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								
11								
12								
13								
14								
15								
	128	64	32	16	8	4	2	1

MEMO

VIC-1001

PERSONAL COMPUTER USER'S MANUAL

1981年1月15日第2版発行

発行所  コモドール・ジャパン株式会社

東京都港区赤坂8丁目5番32号赤坂山勝ビル6階
〒107 ☎03(479)2131

印刷所 株式会社 坂井印刷所

VIC-1001

 **commodore japan limited**

AKASAKA-YAMAKATSU BLD. 6F 8-5-32 AKASAKA, MINATO-KU TOKYO JAPAN 〒107 PHONE 03-479-2131

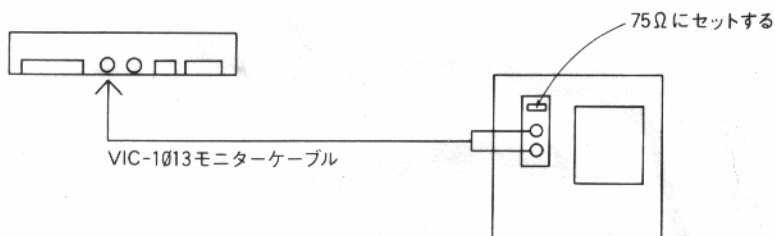
VIC-1001とモニター・テレビ(又は家庭用カラーTV)との接続方法

まず、VIC-1001及びモニター・テレビの電源が切れている事を確認して下さい。

そして、御使用になるTVのタイプにより、下に示してある方法により、VIC-1001とTVを接続し、TVの電源を先に入れ、その後VIC-1001の電源を入れて下さい。画面は約2秒以内に写し出されます。

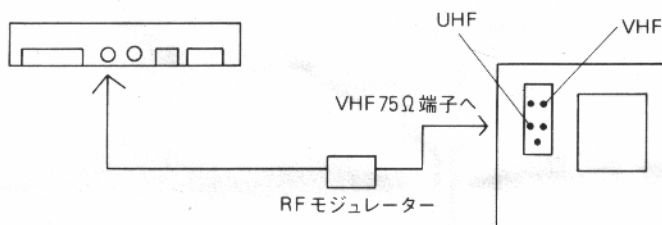
1. モニター・テレビ御使用の場合

VIC-1013モニター・ケーブルをVIC-1001のビデオ・インターフェイス・コネクターに差し込み、一方をモニター・テレビの端子に接続して下さい。その際、モニター・テレビの端子が75Ωにセットされている事を確認して下さい。



2. 家庭用テレビを御使用の場合

同梱されておりますRFモジュレーターのコネクターをVIC-1001のビデオ・インターフェイス・コネクターに差し込み、片方をテレビのVHFのアンテナ端子に接続して下さい。この際、75Ω端子に接続して下さい。



テレビのカラー調整について

VIC-1001を最初に家庭用テレビまたはカラー・モニターに接続し、電源を入れたさいに、ご使用のテレビによっては、スクリーン、ボーダー、キャラクターのカラーが鮮明に出ない場合があります。このような場合には、使用テレビが家庭用テレビのさいには、カラー調整つまみ（色の濃さ、色あい、明るさ、映像など）およびチャンネル微調整つまみによって、最適のカラーになるようにご調節ください。使用テレビがカラー・モニターの場合はカラー調整つまみにより、最適のカラーになるようにご調節ください。