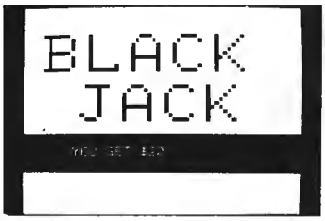
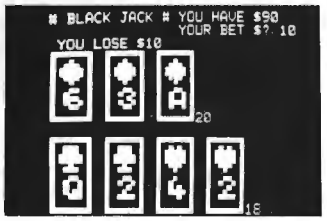
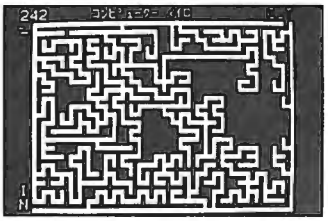
NAKAMOZU TINY BASIC







山下春生

今までのＴＩＮＹＢＡＳＩＣの機能はすべて含み，より速く，より高度な処理を目指して製作された6800系ＢＡＳＩＣインタープリタの決定版．高速，命令が強力で非常に多い，グラフィック命令を持つ，bit処理，割り込み処理が可能などの多くの特徴を持つ究極的ＴＩＮＹ ＢＡＳＩＣ．

NAKAMOZU TINY BASIC とは？

ＢＡＳＩＣが日本に上陸以来，数年で早くもマイコンの標準言語の座を得，現在，整数型及び実数型の種々のＢＡＳＩＣが発表（発売？）されています．

その中で，ホビーストに最も愛用されているものは，ゲームに適したＴＩＮＹ ＢＡＳＩＣであると思われます．ＡＳＣII誌上にも，ＰＡＬＯ ＡＬＴＯ，ＧＡＭＥ，電大版等が，毎回の様に登場しています．最近の傾向として，ゲームを取ってみてもＴＴＹベースものの焼き直し（スクロール型）から，ＣＲＴベースやグラフィックを使用したもの（リアルタイム型）に変って来ており，この様な機能とリアルタイムに使用出来るだけの高速なものが必要とされています．

ここで紹介する〝ＮＡＫＡＭＯＺＵ ＴＩＮＹ ＢＡＳＩＣ（ＮＴＢ）〟は，他のＴＩＮＹ ＢＡＳＩＣの機能は全て含み，より速くを目指して製作された整数型インタープリタです．ＧＡＭＥがＶＴＬを基にして作られた様に，ＮＴＢは68系では高速を誇る電大版を基に発展させました．

特徴としては，マニュアルに示した様に

1) 高速．
2) 命令数が非常に多い．
3) ＣＲＴベースの命令，及びグラフィック命令を持つ．
4) 機械語的な仕事が可能（bit 処理等）．
5) 割り込み許容モードで働く．

が挙げられ，特にゲーム用には，文字入力，リアルタイム入力があり，ＰＲＩＮＴ文中にカタカナやグラフィック記号を直接入れる事が可能である他，ＰＡＬＯ ＡＬＴＯと同じ省略形がある点も便利であると思われます．

現在，数名の仲間に使われて，様々なプログラムが出来ており，機能面での問題や不満が指摘されておりませんので発表する事になりました．以前は．ゲーム等はＶＴＬを主に制作していましたが，（1月号のマイクロＴＲＥＫも仲間の合作）今ではＮＴＢに乗り換えています．

特に，スピードに関しては，**表2**のベンチマーク・テストの結果では，電大版より少し速く，ＶＴＬと同程度ですが，実際のプログラムでは数倍速くなります．この傾向は，プログラムが複雑で長くなる程顕著になって来ますので，ベンチマーク・テストなどは，最も本来のスピードを示していないものと言えるかも知れません．高速化の要因は，基になった電大版がすぐれていたのに加え，ＣＯＭＭＡＮＤ，ＳＴＡＴＥＭＥＮＴ，ＦＵＮＣＴＩＯＮのテーブルをサーチする回数を数分の1に減少したためです．

加えて，ＮＴＢには，スタックを用いてループを作るＤＯ ＵＮＴＩＬがありますので，ＦＯＲ ＮＥＸＴループと合わせると，全くＧＯＴＯ文の無いプログラムが作れ，大幅なスピードアップが可能です．（ＤＯ ＵＮＴＩＬは，ＰＡＳＣＡＬやＰＬＡＮのＲＥＰＥＡＴ ＵＮＴＩＬと同じもので，プログラムの構造化に役立つ）

一般に使用されているＩＦ ＴＨＥＮ（ＧＯＴＯ）ループは，ループ回数だけ行番号サーチを行なうので，プログラムが長くなれば非常に時間がかかります．（ＰＲＯＧ・1とＰＲＯＧ・2はその例）

NTBを走らす方法は？

Ｉ／Ｏルーチン

基本的には，他の68系インタープリタと同様に，ＭＩＫＵＧ程度のＩ／Ｏルーチンを用意すれば良いと言えます．ＴＴＹでも動かせますが，有効に使うにはカタカナとグラフィック記号が使えるＶ―ＲＡＭが最適です．

ＣＲＴベースの命令（ＣＬＲ，ＣＵＲＳ），グラフィック命令（！Ｗ，！Ｂ，！Ｒ，！Ｐ，ＮＥＧ）及びリアルタイム入力関数（ＫＥＹ）等，システムに依存する命令に関する分だけパッチを入れる箇所が増えますし，グラフィックなどは各自のシステムにあったルーチンへのリンクが必要です．

ハードウェア

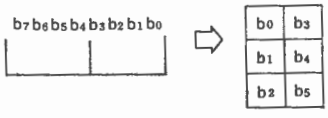
私のシステムは，Ｖ―ＲＡＭとしてＴＶＤ―02をグラフィック改造したもの（サンベックのＶ―ＲＡＭやＴＲＳ―80と同様）を使用しており，ＰＲＯＧ・3の実行結果の通りのＡＳＣIIコードに割り合ててあります．（グラフィック以外はＪＩＳと同じ）

1キャラクターを6つに割って，ＡＳＣＩＩコードの下位6bitをそのままパターンにしたもので，プロットルーチンで計算する事により，64×48のグラフィックとして使え，かつ，そのままグラフィック記号としても使え

ナカモズ TINY BASICの特徴

1．一般のTINY BASICの2倍半の命令数．(52)
2．電大版よりさらにSPEED UP．
3．CRTベースの命令を持つ．
4．PALO ALTO TINY BASICと同じ省略形が使用できる．
5．10進数，16進数，ASCIIの定数及び入出力可能．
6．完全なLOAD，SAVE，APPENDの機能を持つ．
7．READ，DATA，RESTOREがあり，関数の定義ができる．
8．自由に使える37の変数．(10のIX変数を含む)
9．2バイト単位のPEEK，POKEにも使える10種のIX配列．
10．配列的に使用できるPEEK，POKE(1バイト単位)
11．4種のグラフィック制御命令．
12．PRINT文中にカタカナ，及びグラフィック記号が使える．
13．PRINT USING文がある．
14．INPUT時のKEY-IN ERRORによる．プログラム中断をふせぐRE-ENTER．
15．高速ループを作る．DO・UNTIL．
16．リアル・タイム入力用KEY関数．
17．IRQ許容モードで働く．
18．17種類のERROR MESSAGE．
19．VTLのプログラムをそのままおきかえられるGOTO文．
20．使い易いMOD関数．
(その他，電大版TINY BASICの特長はすべて持つ．)

メモリビット・パターン　グラフィックパターン



るもので，他のキャラクターと同時に表示出来る点が便利です．プログラム中で，コントロールコード以外の全てのコードの使用を認めると，プログラム末を示すＥＯＦマークに困りますので，負の数で使用する必要の無い＄80をＥＯＦマークとし，キーボードから＄80の入力を禁止しました．

＄80は，このグラフィック記号のスペースに当るもので，使用しないコードだからです(Ｖ－ＲＡＭ上に何種類もスペースがあるのは困りものです．)したがって，キーボードから＄80が出る人は，ソフトで禁止するか，＄20に変換してください．

システム編集方法

先ず，オブジェクト・ダンプ・リストからメモリーにそのままロードして，全体のサムチェックをして下さい．(サムチェック・プログラム)

この結果，ＡＣＣＢが＄３Ｃで，ＡＣＣＡが＄ＤＣならＯＫです．変更箇所は，主に入出力，ＢＲＥＡＫ判定，イニシャライズ・ルーチンと，マシンスタック，ＩＸスタック及びＴＥＸＴエリア等で一応動きます．

他にＣＲＴ命令，グラフィック命令，ＫＥＹ関数等を使う人は，それぞれのルーチンとパッチしなければなりません．(ＡＤＲＳとＤＡＴＡの変更程度)

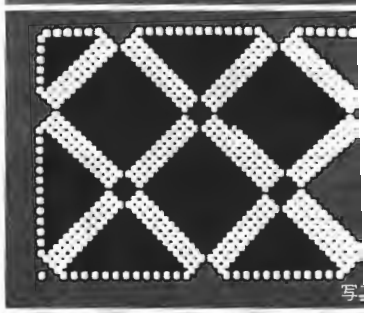
このインタープリタは，ＴＥＸＴエリア変更が容易なように，ＴＥＸＴ開始ＡＤＲＳ，最終ＡＤＲＳをＤＡＴＡとして，インタープリタの先頭にもっています．(インタープリタの後に，Ｉ／Ｏルーチンを付ける人は，開始ＡＤＲＳをその後にしておく)

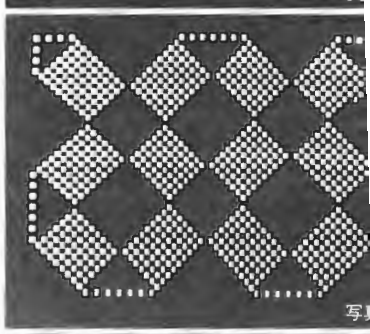
サンプル・プログラム1

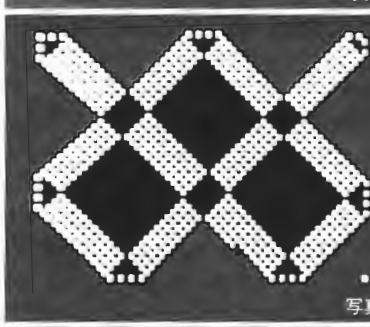
```
10 P."START"
100 REM AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
110 REM BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
120 REM CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
130 REM DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
140 REM EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
150 REM FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
160 REM GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG
170 REM HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
180 REM IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
190 REM JJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ
500 K=0
510 DO
520 K=K+1
530 UNTIL K=1000
540 P."END"
900 REM 4.2 SEC
```

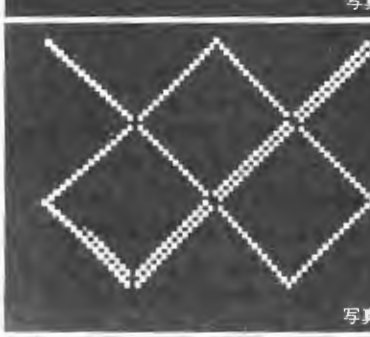
サンプル・プログラム2

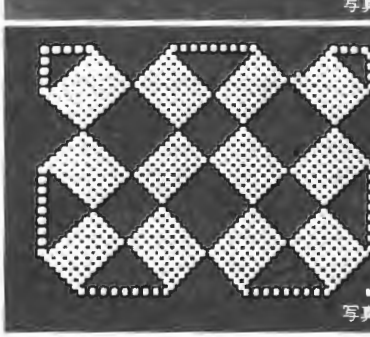
```
10 P."START"
100 REM AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
110 REM BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
120 REM CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
130 REM DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
140 REM EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
150 REM FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
160 REM GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG
170 REM HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
180 REM IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
190 REM JJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ
500 K=0
520 K=K+1
530 IF K<1000 THEN520
540 P."END"
900 REM 13 SEC
```











サンプル・プログラム 3

```
20 CLR
100 %0=$C000
110 F.I=0TO$FF
120 %0(I)=$FFFF-" "-I*$100
130 N.I
140 COPY
```

サンプル・プログラム 4

```
10 REM 1000 ﾏﾃﾞﾉ ｿｽｳ
20 F.A=5 TO995 S.6:%0(A)=0:%0(A+2)=0:N.A
30 F.A=5 TO31 S.2:IF%0(A) N.A
40 F.B=2*A TO1000 S.A:%0(B)=1:N.B
50 N.A
80 P.C.7;
90 P."■■ 1000 ﾏﾃﾞﾉ ｿｽｳ ■■"
100 P.U.**2,U.**3,
110 F.A=5 TO995 S.6
115 IF%0(A)=0 P.U.**A,
120 IF%0(A+2)=0 P.U.**A+2,
130 N.A
```

```
■■ 1000 ﾏﾃﾞﾉ ｿｽｳ ■■
*****2  *****3  *****5  *****7  ****11
****13  ****17  ****19  ****23  ****29
****31  ****37  ****41  ****43  ****47
****53  ****59  ****61  ****67  ****71
****73  ****79  ****83  ****89  ****97
***101  ***103  ***107  ***109  ***113
***127  ***131  ***137  ***139  ***149
***151  ***157  ***163  ***167  ***173
***179  ***181  ***191  ***193  ***197
***199  ***211  ***223  ***227  ***229
***233  ***239  ***241  ***251  ***257
***263  ***269  ***271  ***277  ***281
***283  ***293  ***307  ***311  ***313
***317  ***331  ***337  ***347  ***349
***353  ***359  ***367  ***373  ***379
***383  ***389  ***397  ***401  ***409
***419  ***421  ***431  ***433  ***439
***443  ***449  ***457  ***461  ***463
***467  ***479  ***487  ***491  ***499
***503  ***509  ***521  ***523  ***541
***547  ***557  ***563  ***569  ***571
***577  ***587  ***593  ***599  ***601
***607  ***613  ***617  ***619  ***631
***641  ***643  ***647  ***653  ***659
***661  ***673  ***677  ***683  ***691
***701  ***709  ***719  ***727  ***733
***739  ***743  ***751  ***757  ***761
***769  ***773  ***787  ***797  ***809
***811  ***821  ***823  ***827  ***829
***839  ***853  ***857  ***859  ***863
***877  ***881  ***883  ***887  ***907
***911  ***919  ***929  ***937  ***941
***947  ***953  ***967  ***971  ***977
***983  ***991  ***997
```

グラフィックパターン デモンストレーション・プログラム

```
500 CLR
510 CU.(14,2):P."▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁"
520 CU.(14,3):P."■▌▌ NCC-1701■▛";
530 P."▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁▁"
540 P."▀▀▀▀▀▀▀▀▀▀▀▀       H"
550 P.T.11;"▀▚▙   H"
560 P.T.13;"▜▛▙   ▙▁▁"
570 P.T.10;"▟▛▜▜▛   ▀▀▛"
580 P.T.11;""▀▀▀▀▀▀▀"
```

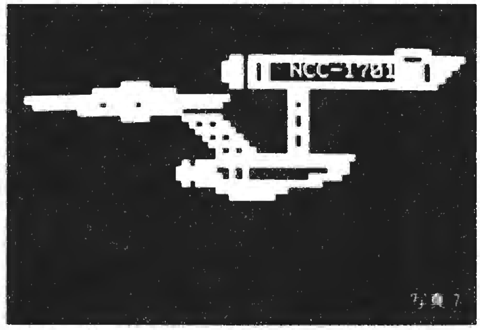


写真 7

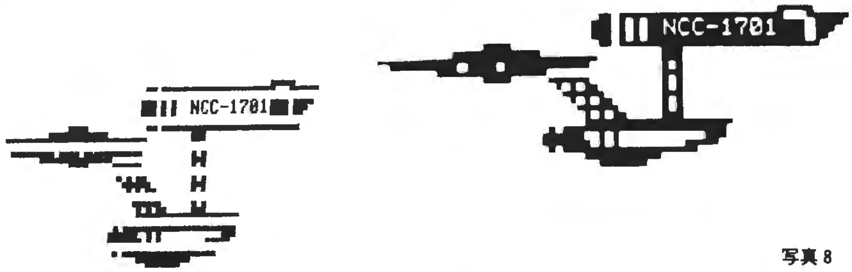


写真 8

入出力仕様は，MIKBUGと同じで，パラメータはACCA，他は要保存で，入力ルーチンでは，エコーバックを行なう必要があります.〔H68の人は，KEY IN後，タイマー割込の禁止を行なう.(実行スピードが落ちる)($E007)←$04，単にSEIだけでは解除されてしまう．SEIは必要ない.〕入出力ルーチンは，CASET LOAD，SAVEの切り換えをコントロール・コード（$11～$14）で行なうようにし，出来ればソフトウェア発振音（$06→LOW，$07→HIGH）が出るようにしておくと使いよいでしょう．

BREAK判定ルーチンは，コントロールキーが押されたら，Zフラグをセットしてリターンする様にします.〔Cフラグが都合の良い人は，（$0188）を$24に変更〕

NTBの4つのスタックの内，マシンスタックとIXスタックは，$A000～$A07Fにあり，メモリーの無い人は変更．

以上の最低限必要な変更箇所は表3に，その他の変更箇所は表4に掲げました．

使用しないSTATEMENTや関数のある人は，TABLEのADRSを各エラー処理のADRESに変更すれば安全です．KEY関数は，ハードに直接依存するため，H68以外の人は，作り換える必要があります．H68用KEY関数は，ソフトでスキャンしているため，引数を設けていますが，他のキーボードの人は，2つ以上のKEYを押しても無意味なため，引数は必要ありません.（$0AEF～$0B19のエリアで充分おさまる.）

KEYが押されていればASCIIコードを，押されていなければ，100（$64）をACCAに入れ，ACCBをCLRして，$05E4にJMPすれば良いです.（IX要保存）

注意としては，押しているかどうかの判断を，PIAのCA1，2やCB1，2を行なうと，KEY関数を実行する前に押したフラグが残っているため，使いづらいです．

入力ポートからのDATA（例えば，ダイナミックのエンコーダ使用の人なら，3回読んで同じコードなら押している，異なれば押してない）で判断する必要があります．

COPYは，オプションSTATEMENTですので，$098DのADRSを自由に変更して使用して下さい.（私は，V−RAM1ページをプリントアウトするようにしている．コマンド名も4文字のものになら変更可．レジスタは，SP以外破壊しても良い.）

ナカモズBASICによるサムチェック・プログラム

```
20 IN." ｱﾄﾞﾚｽ FROM  "A
30 IN." ｱﾄﾞﾚｽ  TO  "B
40 N=1:S=0:A=A/8*8
50 DO:C=0:P."■■■■ NO. ";N;" ■■■■":N=N+1
60 DO:P.HDFA;T.9;
70 DO:C=C+#A:F=#A/16:GOS.200
80 F=MOD:GOS.200:P." ";:A=A+1
90 U.(AND(A,7)=0)+(A>B):P.
100 U.(AND(A,$7F)=0)+(A>B)
110 P." SUM :";HDFC:S=S+C
120 U.A>B:P.:P."TOTAL CHECK SUM :";HDFS
130 END
200 P.C.(F>9)*7+"0"+F;:RET
```

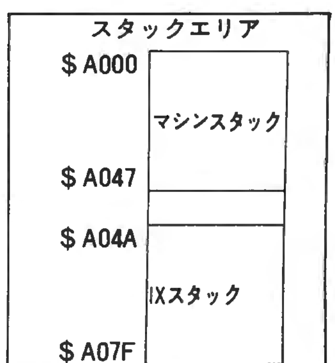


図1　ベース・ページ　ワークエリア（＄0000～＄00FF）

ADRS（下位）	内　　容	ADRS	内　　容	ADRS	内　　容	ADRS	内　　容
00	a) VTOP	40	%2	80	↑	C0	EOBF ↑
1		1		1		1	
2	A	2	%3　IX変数	2		2	
3		3		3		3	
4	B	4	%4	4		4	
5		5		5		5	
6	C	6	%5	6		6	
7		7		7		7	⊥
8	D	8	%6	8		8	C STACK ↑
9		9		9		9	
A	E	A	%7	A		A	
B		B		B		B	
C	F	C	%8	C		C	
D		D		D		D	
E	G	E	%9	E		E	
F		F		F		F	
10	H	50	WKA	90		D0	
1		1		1		1	
2	I	2	WKB	2		2	
3		3		3		3	
4	J	4	WKC	4		4	
5		5		5		5	
6	K	6	XS	6		6	
7		7		7		7	
8	L	8	SPS	8		8	
9		9		9		9	
A	M	A	MDS	A	BUFFER	A	
B		B		B		B	
C	N	C	ADP　FS　※	C		C	
D		D		D		D	
E	O　変数	E		E		E	
F		F		F		F	
20	P	60	RNDS	A0	INPUTLINE　　C STACK	E0	FOR-NEXT STACK
1		1		1		1	
2	Q	2	FLOD　※	2		2	
3		3	FAUT　※	3		3	
4	R	4	LPCT　※	4		4	
5		5	CHCT　※	5		5	
6	S	6		6		6	
7		7	WKUSE	7		7	
8	T	8	空	8		8	
9		9		9		9	
A	U	A	LNB	A		A	
B		B		B		B	
C	V	C	EOB	C		C	
D		D		D		D	
E	W	E	EADRS	E		E	
F		F		F		F	
30	X	70	XSP	B0		F0	
1		1		1		1	
2	Y	2	CSP	2		2	
3		3		3		3	
4	Z	4	FRP	4		4	
5		5		5		5	
6	[　BOP	6	DP	6		6	
7		7		7		7	
8	¥　EOP	8	BFFR	8		8	
9		9		9		9	
A	]　MEMEND	A		A		A	
B		B		B		B	
C	%0　IXV	C		C		C	
D		D		D		D	
E	%1	E		E		E	
F		F	↓	F	↓	F	⊥

説明　※印のものは1バイトとなっている。

表4　変更箇所（2）

ADRS	DATA	変　更　内　容	対応命令
0A14	DF	VRAMのADRS BUSが反転でない人　＄DF→＄20　V-R	CLR
0A2C	C0	V-RAMの先頭ADRSの上位1バイト	CLR CURS
0A42	A094	I/OのDISPLAY POINTERの上位をSAVEしているADRS	CURS
0A48	A095	〃　下位　〃	CURS
098D	3CB6	OPTIONステートメント　（SP以外破壊可） 各自が定義したステートメントのADRSに変える。 （使用しない人は，＄0A2Eにする。ERR9になる） コマンド名変更は，ソースリストのテーブルを変更する （4文字目の上位1bitを1にすること）	COPY(OPTION)
0992	3C06	グラフィック記号白黒反転ルーチン　（SP以外破壊可） （使用しない人は，＄0A2Eにする。）	NEG
099B	3BF7	グラフィックPick関数ルーチン　（IX, ACCA 破壊可） ACCA：X座標 ACCB：Y 〃 の点が（白→ACCB=1，黒 ACCB=0，キャラクタ ACCB=100(＄64)） にしてリターン	!P(X,Y)
09AD	3BD7	グラフィックSETステートメント　（SP以外破壊可） ACCA：X座標 ACCB：Y座標 の点を白にする。	!W(X,Y)
09B9	3BE8	グラフィックRESETステートメント　（SP以外 〃 ） ACCA :X座標 ACCB :Y座標 の点を黒にする。	!B(X,Y)

表4つづき

09C4	3BF0	グラフィックREVERSステートメント ACCA :X座標 ACCB :Y座標 の点を白黒反転する。　REVERS	!R(X,Y)
09D0	40	グラフィック引数上限指定　X座標（0～63）	
09D4	30	〃　Y座標（0～47）	

表3　変更箇所（1）

ADRS	DATA	変　更　内　容
0106	0C60	TEXT開始ADRS ｝この間がプログラムエリア
0108	33FF	〃 最終 〃
015F	35E6	一文字入力ルーチン　ACCA←ASCII COAD　ACCB, IX, 保存 カセット入力との切り換えも行ない，出力ルーチンでエコーバックしてリターン
0186	369B	BREAK判定（本文参照）　ACCAのみ破壊可
01B3	341C	システムINITIALIZE　SP以外破壊可
046B	3476	一文字出力ルーチン　ACCAにASCIIコード，　ACCB, IX, 保存 CONTROL COADによる処理も含むこと。
01C2	A047	｝マシン・スタック設定
01D4	A047	
02DB	A047	
091C	A047	
025D	A080	IX スタック　上限＋1
06F7	A04A	〃　下限
0709	A080	〃　上限＋1
0B61	F0B1	各自MONITERのADRS

インタープリタの構造と高速化

インタープリタはコンパイラと対比させて，毎回翻訳するという言葉がよく使われますが，インタープリタによるプログラムは，種々のユーティリティー・サブルーチンを接続するデータの集合と考えた方が判り易いと思います．

ＶＴＬの様な記号言語は，一文字を判断するだけで処理ルーチンへ分枝できるので，処理も容易でスピードも速くなります．

ＢＡＳＩＣ等は，不定長のＳＴＡＴＥＭＥＮＴ　ＦＵＮＣＴＩＯＮなどを判断するためには，キーワード・テーブルを順に一文づつ比較して，処理ルーチンのＡＤＲＳを得る訳で，この処理がＢＡＳＩＣの高速化の妨げになっています．また，記号言語と異なり，変数とＳＴＡＴＥＭＥＮＴ　ＦＵＮＣＴＩＯＮとの区別は容易ではないので，大半のＴＩＮＹ　ＢＡＳＩＣ（ＰＡＬＯＡＬＴＯ,電大版を含む）は，まずＳＴＡＴＥＭＥＮＴ　ＴＡＢＬＥを捜して，無かったら代入文と思い，右辺ではＦＵＮＣＴＩＯＮ　ＴＡＢＬＥをサーチし，見つからなかったら変数処理に行くという方法をとっています．

例えば，Ａ＝１＋Ｂ＊Ｃ＋10＊Ｄ　というプログラムでは，ＳＴＡＴＥＭＥＮＴ　ＴＡＢＬＥを１回，ＦＵＮＣＴＩＯＮ　ＴＡＢＬＥを５回サーチする訳です．単なる代入文は，各テーブルを最後までサーチする訳ですので，代入文が一番遅くなります．

ＮＴＢは，テーブル・サーチを必要な時しか行なわないので，上の例では一度もテーブル・サーチを行なわず，プログラム中で最もよく使われる代入文の実行速度があがっている．（ベンチマーク・テストには代入文が少ないので表われてない）

以前，電大版をそのままで，命令数を増したところ，実行速度が半分近くになり，インタープリタの管理ルーチンの変更となりました．

もう一つ．時間のかかるのは，行番号サーチですが，これは前述のように，ＩＦ　ＧＯＴＯによるループをＤＯ　ＵＮＴＩＬに置き換える事で，サーチ回数は大幅に減少します．

このＢＡＳＩＣでは，行番号ＪＵＭＰは飛び先の行番号が現在より大きければ，現在の行から，小さければ先頭からサーチしますので，ループ内で使うサブルーチン等は，ループの直後かプログラムの先頭に置く事が，速いプログラムを作るテクニックです．

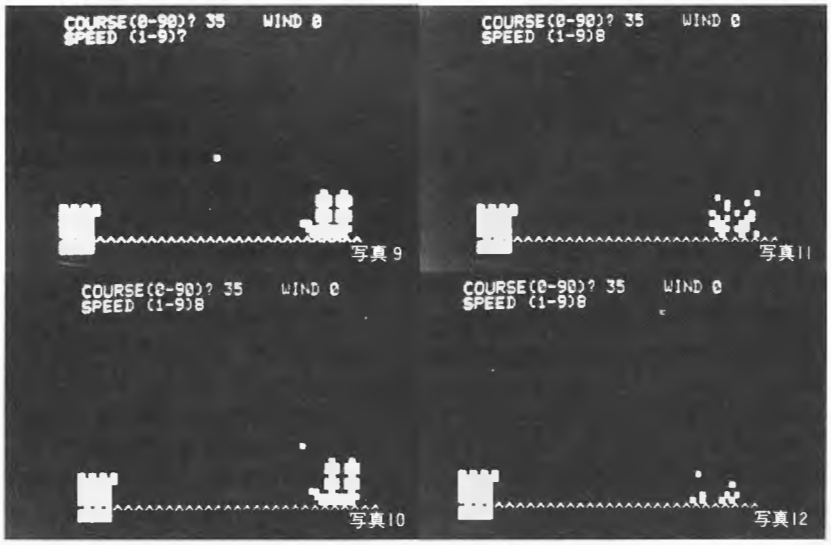
表２．ベンチマークテスト結果（省略形使用）

No. 1	No. 2	No. 3	No. 4	No. 5	No. 6	No. 7	計
0.65	6.2	12.8	12.2	16.0	23	35	105.9
0.65	* 4.2	* 9.2	* 9.0	* 13.0	* 21	* 32	89.05

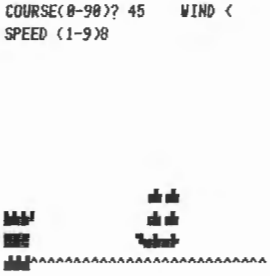
IF THENの代わりにDO UNTILを使用したもの．

サンプル・プログラム

```
10 REM*************************
11 REM* KAIZOKU-SEN  GAME     *
12 REM*  NAKAMOZU TINY BASIC  *
13 REM*************************
20 *(203*K-K*K)/10*V/400
30 B=0:C=0
40 GOSUB 500
50 DO:X=80:Y=330:F=R.(3)-1:CLR(0,1):CURS(20,0):P
."WIND ";
60 P.CHR$(F=-1)*"("+(F=1)*")"+(F=0)*"0"
70 CURS(0,0):IN."COURSE(0-90)"K
80 P."SPEED (1-9)";:V=GET$-"0"
90 IF(V=0)+(V>9)T.70
100 RESTORE20:T=READ
110 K=90-K:RES.20:S=RE.:D=X:E=Y
120 DO:!B(D/10,E/10):IF!P(X/10,Y/10)=1 U.1:U.1:G
.170
125 !W(X/10,Y/10)
130 D=X:E=Y:P.CHR$(7);:X=X+S:Y=Y-T
140 S=S+F:T=T-1:UNTIL(X(0)+(X>630)+(Y(0)+(Y>440)
150 !B(D/10,E/10):B=B+1:UNTIL B>19
160 P.:P."YOU DESTROYED ";C:END
170 IFX>70 GOS.700:C=C+1:B=B+1:G.40
180 NEG:P.C.6;C.4;C.5;"AHO":G.160
500 CLR:CURS(0,13)
520 P."▟▙▟▙":P."▟▙▟▙":P."▟▙▟▙^^^^^^^^^^^^^^^^^^^^
^^^^^^";
530 A=25-R.(12):CU.(A+1,12):P."▟▙▟▙";
540 CU.(A+1,13):P."▟▙▟▙":CU.(A,14):P."▜▟▙▛";
550 RET
700 CU.(A,12):P."▝▖ ▗▘";C.7;
710 CU.(A,13):P."▚▞▝▖▗▘";C.6;
720 CU.(A,14):P."▜▟▗▛▘";C.6;
730 F.G=1TO5:P.C.6;:N.G
740 CU.(A,12):P."     ";C.6;
750 CU.(A,13):P."▝▘  ▗▘";C.6;
760 CU.(A,14):P."▗▟ ▗▟▘";C.6;
770 F.G=1TO5:P.C.6;:N.G:RET
```



放電プリンタによる実行例１　　放電プリンタによる実行例２



DO-UNTILサンプル・プログラム

```
10 ********************************
15 * DO-UNTIL  SAMPLE PROG.
18 *  GRAPHIC-MIRROR
20 *   BY.H.YAMASHITA
25 ********************************
30 DO:CLR:X=31:Y=23:I=0
40 DO:!W(X,Y):!W(63-X,Y):!W(X,47-Y):!W(63-X,47-Y
)
60 DO:S=R.(3)-1:P=X+S
70 U.(P(0)+(P)31)=0
80 DO:T=R.(3)-1:P=Y+T
90 U.(P(0)+(P)23)=0
100 X=X+S:Y=Y+T:I=I+1
110 U.I=1000
120 COPY
130 U.0
```

I／Oルーチンの作り方

今まで，4K—BASICやVTL，GAME等を動かしている人は，何度もI／Oルーチンを製作されたことと思いますが，NTBは，システム依存性が高いので，全機能を活かすためには，I／Oルーチンも重要です．

私のシステムでは，TTYシュミレータ，ソフト・プリンタ制御ルーチン，ソフト高速カセット，ファイル，グラフィック基本サブルーチンを組み合わせて，独立したTAPE OPERATING SYSTEMを作り，全てのソフトは同一のI／Oルーチンで働いています．その中で，今回NTBに必要なものだけを再アッセンブルしたリストを紹介します．

このI／Oルーチンは，H68用で，ソフトによるBREAK機能，コントロール・コードによるCASET入出力切り換え（ACIAの設定とリモコン：リレー動作).コントロール・コードによるソフト・オシレータ（2種類の音）を含んでいます．

H68のポケッタブル・コンソールをフル活用するため，キーボードの4段シフトを実現させています．コントロール・キーを押すたびに，サイクリックにシフトされ，そのつど螢光表示に'ASCII'，'JIS'，'GRAPHIC'，'CONT'とシフト状態が表示されます．また，シフトの組み合せは，プログラマブルで，SHFTPTの下位4 bitにシフトしたい組み合せをイニシャライズすれば，任意のシフト段数のキーボードとして使用出来ます.(リストでは，SHFTPTに$OFを書き込んでいるので，4段シフトになっている.)

カーソルは，入力待ちの時だけ出るようになっており，BASICから出力の時は，表われません，カーソルとは，人間がキーインしやすいように存在するものですから，BASICからの出力には不要なものですし，こうすると，カーソル移動とプリント文に入れたグラフィック絵を動かす場合，画面がきれいで見やすくなります．

$ODF9以降は，私と同じグラフィック改造（又は製作）された人のためのグラフィック・サブルーチンです．

TVD—02でのグラフィック改造は，I／O誌（'78・6）の回路を一部変更すれば，可能です．尚，私のV—RAMは，データ・バスが反転のための反転でない人は，I／Oルーチンを変更する必要があります．

NTBの後に，このI／Oルーチンを接続する人は，NTBの変更箇所を，このI／OのADRSに修正し，TEXT開始ADRSをI／Oルーチンの後に変更すれば良いです．

NTBは，ゲームに最適な言語だと思いますが，TINY BASICを越えた機能と高速性を活かして，今までBASICでは不可能であった事にも利用出来ると思います．(仮数部桁プログラマブルな浮動小数点演算パッケージや，NTB用画面エディタが出来ています.)

現在，NTBのコンパイラを検討中ですので，完成すればいつか発表したいと思います．