NAKAMOZU TINY BASIC

連載第2回

NTBインタープリタの構造

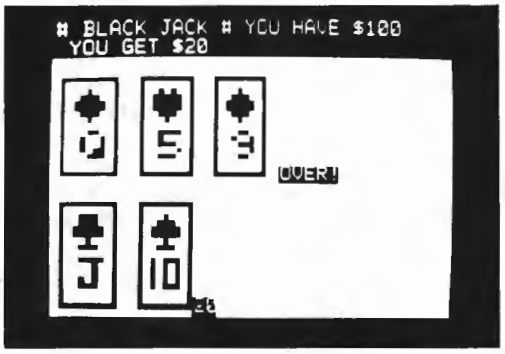
山下春生

最近，構造化プログラミングの波がマイコン・ホビーストの世界にも押し寄せて来ており，ＢＡＳＩＣに代ってＰＡＳＣＡＬのような手続き型言語が注目されています．このような言語のプログラムは，全体の構造がモジュール化されていて見やすく，これらと比較するとＢＡＳＩＣのプログラムを見るのが厭になってきます．

今の時点では，望み過ぎの感もありますが，この違いは，インタープリタとコンパイラの違いと思っても良いかもしれません．インタープリタでは，見やすくすればそれだけメモリーを食い実行速度が落ちるからです．

そこで，インタープリタの特徴を生かし，インタープリタでしかできない方向へ進んで来たもののひとつがＴＩＮＹ　ＢＡＳＩＣであると思います．数ｋバイトのメモリーで走り，エディタ機能を内蔵し，速度は遅いが一応リアルタイムで働く等．

ＮＴＢは，インタープリタでしかできない事は思いつく限り可能に，少しでも構造化言語に近づけようとしてできたものです．



ＮＴＢインタープリタの構造

ＮＴＢの主要管理ルーチンは，一部を除いて電大版と同じですが，68系の性質を生かして非常に良くできていますので解説しておきます．

ＰＡＬＯ－ＡＬＴＯなど80系のＴＩＮＹＢＡＳＩＣと最も異なる点は，インタープリタが使用する各種スタックの考え方です．

インタープリタに必要なスタックは，マシン語レベルで使用するハードウェア・マシン・スタック以外に算術演算用スタック，サブルーチン・コール用スタック，ＦＯＲ・ＮＥＸＴ用スタックなどが必要です．80系ではスタック操作命令は強力であるが，メモリ操作は弱いのでハードウェア・スタックを工夫して各種スタックと兼用しています．反対に68系は，メモリー操作のアドレッシングモードの強力さを生かして，各々のスタックを別々にソフトウェアで作ることが多いと言えます．ＶＴＬのような簡単な言語（算術演算の優先順位がなくＦＯＲ－ＮＥＸＴがない．）では，算術スタックをハードウェア・スタックと兼用させています．

どちらの方式も一長一短がありますが，一般的には，後者の方が理解しやすく浮動小数点形にもそのまま使える点で有利と思います．

ＮＴＢでは，マシン・スタック以外に算術演算用のＣ－スタック，サブルーチン・コール用のＩＸ－スタックそしてＦＯＲ－ＮＥＸＴスタックを持っており，各々のスタック・ポインタをベース・ページに設定しています．

算術スタックが一番複雑で，ＰＵＳＨとＰＵＬＬのサブルーチンおよびスタック内で逆ポーランド演算を行なうサブルーチンが用意されており，全演算はレジスタではなく算術スタック内で行なわれます．（実際にはＩＸアドレッシングで行なう訳ですが．）この処理の様子を図１に示します．

上位管理ルーチン

最上位の管理ルーチンは，図２のように２つのモードから成立しています．

エディト・モードは，行番号付きのＴＥＸＴを入力バッファからロードして編集を行ないます．エディト・モードから行実行モードへは，コマンドとすべてのダイレクト・モードを投入することで移行し，実行終了後エディット・モードに還ります．

行実行モードでは，テーブルをサーチして各コマンド，ステートメント処理ルーチンをコールしますが，ＲＵＮコマンドがコールするルーチンは，行実行モードそのものですのでＴＥＸＴの実行に移り，同様にＴＥＸＴ中のＥＮＤステートメントを実行するとエディト・モードをコールします．この２つのモードに主従関係はなく，共にメイン・ルーチンになっています．この管理ルーチンをもう少し詳しいフローチャートで示したものが図３で，これ自身がＮＥＷ，ＥＮＤ，ＲＵＮの処理ルーチンにもなっています．

コマンドとステートメントには，何ら差はなく単にキーワード・テーブルが異なるだけで，コマンド・テーブルはステートメント・テーブルの前にあるので，コマンドでなければ自動的にダイレクト・モードのステートメントと解釈します．また，ダイレクト・モードというものは，単にポインタを入力バッファ先頭にセットし一行実行ルーチンをコールするだけであり，テキストの実行とまったく区別はありません．一行のみしか存在しないので，ダイレクト・モードでは，一行実行後ＥＤＩＴモードに戻ります．

考えようによっては，コマンドとダイレクト・モードがメインであると見なすことができます．というのは，テキストの実行というものは，ＲＵＮというコマンドが呼び出すルーチンがテキスト実行ルーチンであるからです．

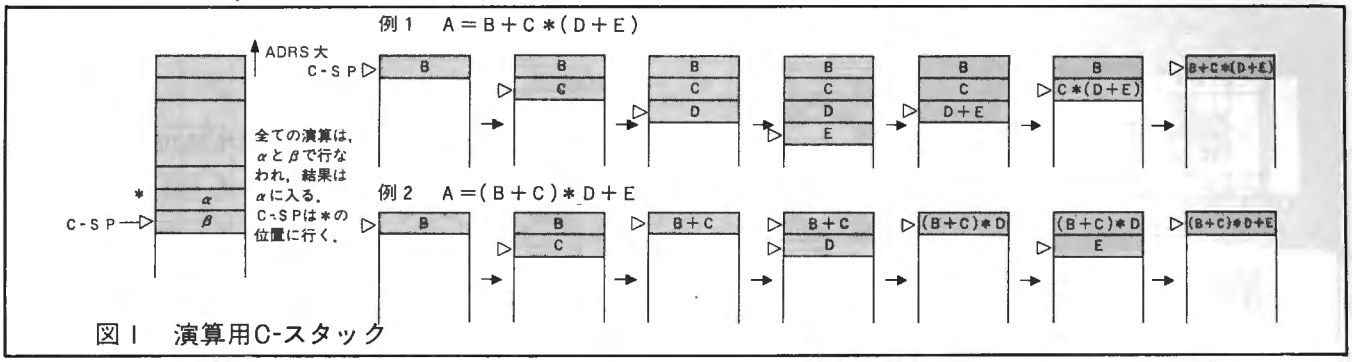


図1　演算用C-スタック

代入文処理ルーチン

インタープリタにおいて，上位管理ルーチンと共に重要なのが算術演算を行なう代入文処理ルーチンで，このＢＡＳＩＣでは，論理演算処理も含んでいます．

演算子の優先順位は，

1．乗除算
2．加減算
3．論理演算（比較演算）

になっており，演算子より優先順位の高いものがカッコ処理と関数処理です．

したがって，代入文処理ルーチンは，演算子に3レベルの管理が必要であり，カッコ，関数を含むんで4レベルの管理を行なっています．

図4は，代入文処理を各レベル別の比較的詳しいフローチャートで表わしたもので，演算子の優先順位を実現する構成になっています．この種のプログラムで重要なのは，再帰性という概念で，このフローチャートで一番下位ルーチンであるＦＵＮＣのレベル処理中，カッコまたは関数が表われると，自分を呼んでいる上位ルーチンをコールします．実際の演算のレベルは，カッコのネスティングまたは，関数の引数内の関数のネスティングの深さによって変動する訳です．この処理を簡単に実現する方法が**図1**で説明したスタックを用いた逆ポーランド演算であり，インタープリタ中最も功妙な部分です．

ＮＴＢの特長である，すべての引数に式が書けるというのは，引数の計算で単にＬＥＶＥＬ３をコールするのみで実現できます．

ＶＴＬのように，関数が存在せず演算子に優先順位がないものは，１レベルのみでよいため代入文処理は非常に簡単になり，アキュムレータ・ペアをスタック・トップと見なしてアキュムレータで数値を持って回っていますが，ＮＴＢではすべての演算処理は，算術スタック内で行なっており，アキュムレータはフリーです．

GOSUB－RETURN，DO－UNTＩL処理

ＮＴＢでは，小道具サブルーチン内以外では，ＩＸレジスタをテキスト実行アドレス・ポインタにしています．

ＧＯＴＯやＧＯＳＵＢのような行番号による分枝は，行番号をＢＡペアにセットし現在の行より小さければテキスト先頭，大きければその行先頭のアドレスをＩＸレジスタにセットして，ＩＸレジスタを進めながら行番号比較を行ない一致する行を捜します．見つかった時，ＩＸレジスタは，その行を指していますので一行実行モードに戻るだけで分枝したことになります．ＧＯＴＯとＧＯＳＵＢの違いは，実行アドレスをスタックにＰＵＳＨするかしないかの差で，ＧＯＳＵＢの場合テキストポインタ用スタック（ＩＸスタック）にＩＸレジスタをＰＵＳＨするサブルーチンＰＳＨＸをコールした後分枝します．

したがって，ＲＥＴＵＲＮの処理は，ＰＵＬＸをコールするだけで済みます．

ループを作るＤＯ，ＵＮＴＩＬも同じようにＩＸスタックを使用しています．

ＤＯは，現在のＩＸをＰＵＳＨするだけで，

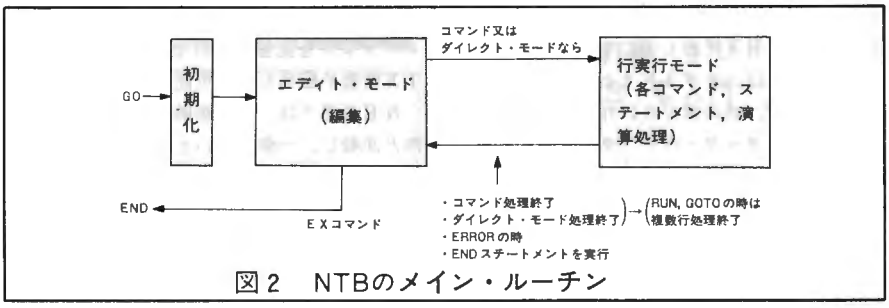


図2　NTBのメイン・ルーチン

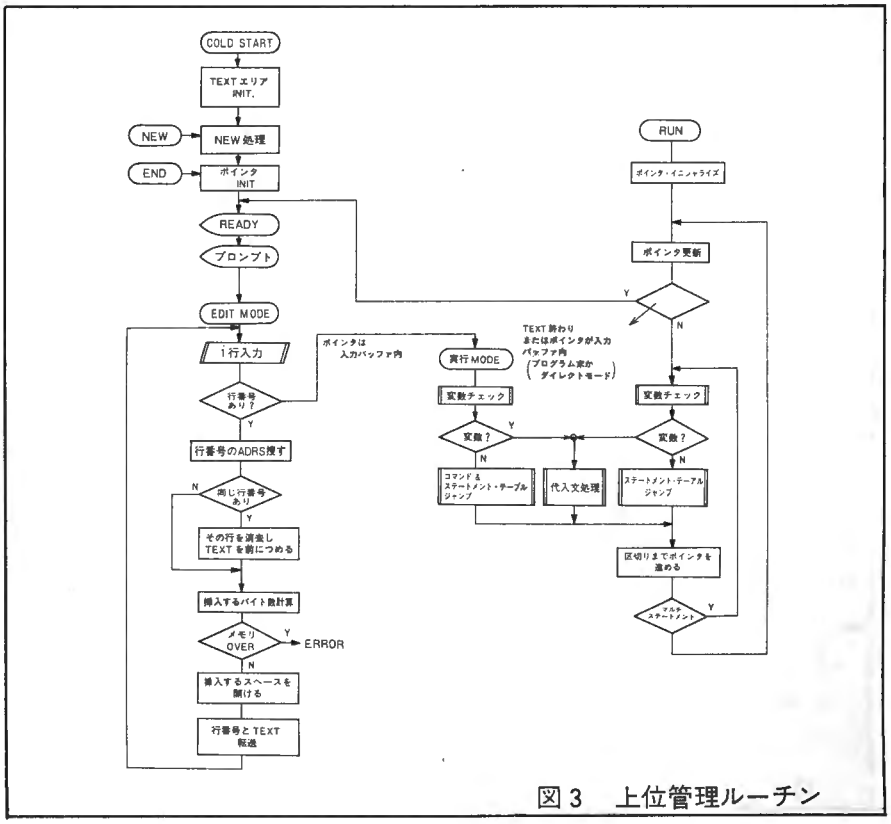


図3　上位管理ルーチン

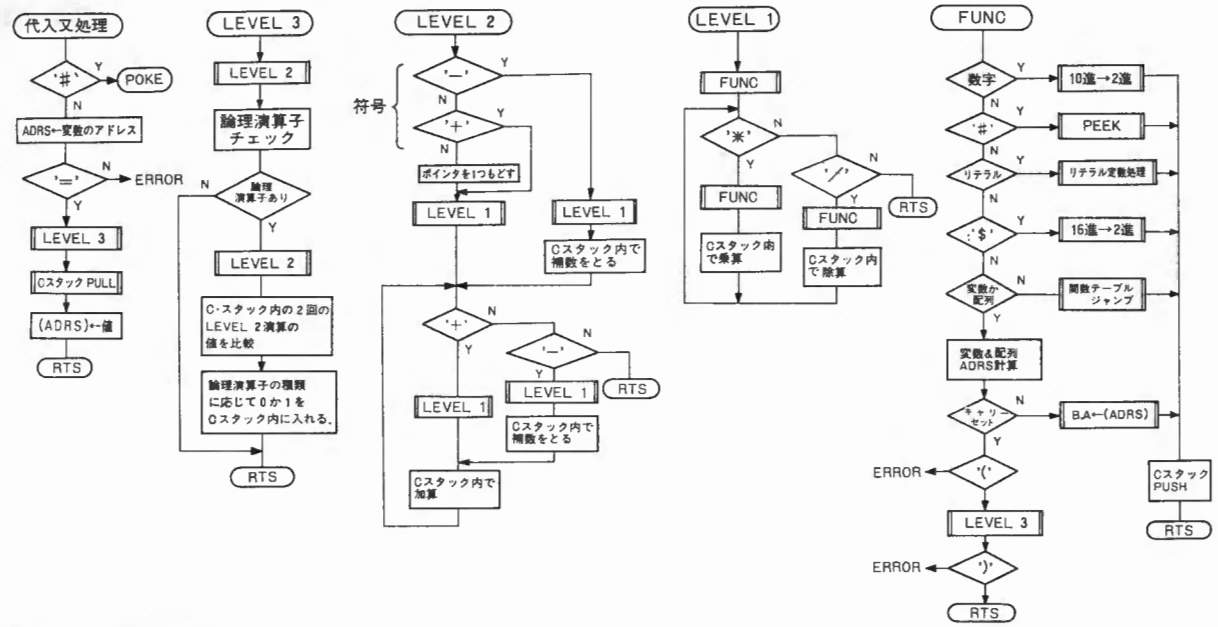


図4　代入文処理

分枝の必要はありませんので，ＤＯ処理ルーチンは，ＰＳＨＸそのものになります．

ＵＮＴＩＬは，まずＩＦ文と共通の比較ルーチンを呼び，式の値が０以外（成立）のときは，ＩＸスタック・ポインタを２バイト戻してから次の文に移り，式の値が０（不成立）のときは，ＩＸスタック・ポインタは変更せずにスタック内からＩＸの値をロードすることによりループを回ります．

ＤＯ，ＵＮＴＩＬは，非常に強力な機能ですが，処理としては極めて簡単であり，ＤＯ，ＵＮＴＩＬのために新たに作った処理は，17バイトだけです．

ＦＯＲ－ＮＥＸＴ処理

ＤＯ－ＵＮＴＩＬは，テキスト・ポインタの２バイトをＩＸスタックにセイブするだけであったのに対し，ＦＯＲ－ＮＥＸＴでは，専用のＦスタックに，８バイトセイブします．

演算は，Ｃスタックで行なうので，**図５**と逆の順で演算を行ないＣスタックに８バイト積んでおいてから，逆方向にＦスタックに転送します．制御変数そのものでなく制御変数のアドレスを記憶するので，制御変数には，ＩＸ変数や配列も使用可能です．

ＮＥＸＴでは，まず制御変数を下スタック内と比較し，一致しないときは，ループ内で飛び出したと見なして，Ｆスタックを８バイト戻し比較をくり返します．一致すれば，ＳＴＥＰの値を制御変数に加算し，ＴＯと符号付の比較を行ない終了ならスタックを戻し，終了でなければ，戻りＡＤＲＳをＩＸレジスタにロードします．

以上の処理により，一応ＦＯＲ－ＮＥＸＴループからの飛び出しは許されますが，飛び出した後同じ制御変数のＦＯＲ－ＮＥＸＴを実行するとエラーになります．

原理的に，ＦＯＲ－ＮＥＸＴループの出口は，ＮＥＸＴのみであるべきものであり，ループ中からの飛び出しの後始末は完全には行なえないので，ＦＯＲ－ＮＥＸＴからの乱暴な飛び出しは，プログラムの構造化の点からも避けたいものです．

他の各種コマンド，ステートメント，関数ルーチンは，ソース・リストにコメントを付けておきましたので参考にしてください．

グラフィック用ハードウェア

私のシステムのＶ－ＲＡＭは，先月号で述べたように，ＴＶＤ－02をグラフィック改造をしたものです．サンベック・Ｖ－ＲＡＭ改造の記事がＩ／Ｏ誌（78’ ６）にありましたが，非常に良いアイディアですので同様の方法で改造（アダプタ方式）しました．

この方式は，**図６**のように１キャラクタを６分割してグラフィックとするもので，等価的に64×48のグラフィックになります．

この方法は，改造が簡単でメモリーの追加が不用などの特徴がありますが，最大のメリットは，グラフィック記号としてキャラクタと同様にアスキーコードでアクセスできる他，サブルーチンを利用すると，フルグラフィックとしても使用できます．（アスキー・コードの対応表を**図７**に示します．）

ＴＶＤ－02は，回路が不明で多少サンベックのものと異なっており，カタカナも有効に使用できるようにしたので，Ｉ／Ｏ誌のものと一部異なっています．

図８がその回路で，アダプタ方式になっているため，本体のキャラジェネを抜いたソケットからフラット・ケーブルでアダプタに接続して使用します．つまり，この回路全体をひとつのキャラジェネとして使用する訳です．電源もすべて，ソケットを通して本体から頂く訳で，足りない情報は，ノン・コネクションのピン（10と14）を通して受け渡します．

そのための本体の改造箇所は，**図８**の２点だけです．

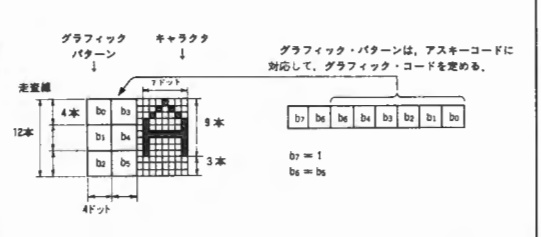


図６　グラフィック・パターンのビット構成

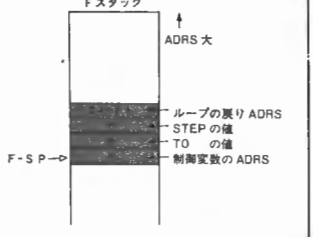
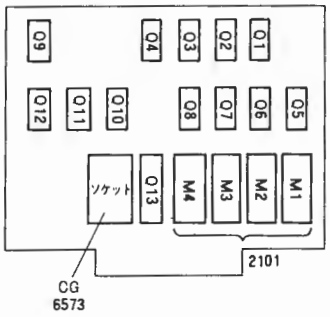


図５　FOR NEXTスタック

キャラクタ＋グラフィック・コード表

上位＼下位	0000 0	0001 1	0010 2	0011 3	0100 4	0101 5	0110 6	0111 7	1000 8	1001 9	1010 A	1011 B	1100 C	1101 D	1110 E	1111 F
0	0	16	SP 32	0 48	@ 64	P 80	96	112	128	144	SP' 160	ー 176	タ 192	ミ 208	224	240
1	1	17	! 33	1 49	A 65	Q 81	97	113	129	145	。 161	ア 177	チ 193	ム 209	225	241
2	2	18	" 34	2 50	B 66	R 82	98	114	130	146	「 162	イ 178	ツ 194	メ 210	226	
3	3	19	# 35	3 51	C 67	S 83	99	115	131	147	」 163	ウ 179	テ 195	モ 211		
4	4	20	$ 36	4 52	D 68	T 84	100	116	132	148	、 164	エ 180	ト 196	ヤ 212		
5	5	21	% 37	5 53	E 69	U 85	101	117	133	149	・ 165	オ 181	ナ 197	ユ 213		
6	6	22	& 38	6 54	F 70	V 86	102	118	134	150	ヲ 166	カ 182	ニ 198	ヨ 214		
7	7	23	' 39	7 55	G 71	W 87	103	119	135	151	ア 167	キ 183	ヌ 199	ラ 215		
8	8	24	(40	8 56	H 72	X 88	104	120	136	152	イ 168	ク 184	ネ 200	リ 216		
9	9	25	) 41	9 57	I 73	Y 89	105	121	137	153	ウ 169	ケ 185	ノ 201	ル 217		
A	10	26	* 42	: 58	J 74	Z 90	106	122	138	154	エ 170	コ 186	ハ 202	レ 218		
B	11	27	+ 43	; 59	K 75	[91	107	123	139	155	オ 171	サ 187	ヒ 203	ロ 219		
C	12	28	, 44	< 60	L 76	¥ 92	108	124	140	156	ヤ 172	シ 188	フ 204	ワ 220		
D	13	29	− 45	= 61	M 77	] 93	109	125	141	157	ユ 173	ス 189	ヘ 205	ン 221		
E	14	30	. 46	> 62	N 78	^ 94	110	126	142	158	ヨ 174	セ 190	ホ 206	゛ 222		
F	15	31	/ 47	? 63	O 79	― 95	111	127	143	159	ッ 175	ソ 191	マ 207	° 223		



本体の改造

1) Q10の6PinからGNDに接続されているラインを切断し，CGソケットの10Pin（NC）に接続する

2) 2101（M4）の16PinをCGソケットの14Pin（NC）に接続する．

3) TVD02をデータバスに接続するとき，D_6とD_7を入れ換えて接続する．（こうすると，標準JISコードになる．）

図7　TVD-02本体の改造箇所

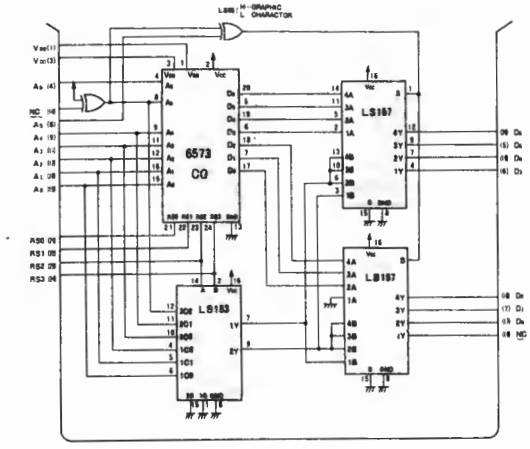


図8　TVD-02用グラフィック改造アダプタ