NAKAMOZU

BASIC

COMPILER VI

- 6800用 高速 1パス コンパイラ
- 任意長の英数字列による変数 および IX配列
  変数とIX配列の合計で 100個まで使用可能
  IX配列には 2バイト型と 1バイト型を使い分けて
  見やすく、効率の良いプログラム開発ができる
- 任意長の英数字列によるラベル および名前が使用できる。特に、数字のみのラベルも許されているので一般のBASICのような、行番号表記も可能
- ラベルのコンスタント宣言 (アセンブラのEQUに相当)
- 演算式中にラベルを定数として使用可
- 構造化 ループ文 および ESCAPE文

REPEAT	WHILE	LOOP	FOR
ESCAPE	ESCAPE	ESCAPE	
UNTIL	ENDWHILE	EXIT	NEXT

- 1バイト および 2バイト単位でデータ、ラベル、ストリングを置くことのできる DATA文
- スタック操作ができるので再帰呼出しも可能
- 関数呼出し、手続き呼出しで、引数を受け渡しができる。もちろんマシン語のプログラムとも容易に、リンクできる
- コンパイル時に、ループの完全なチェックを行なう
- オペランド書き換え等をしていないので、ROM化可能
- 最適化されたオブジェクトを作るので、高速で、しかも、メモリー効率が良い。
- 乗除算は高速化されている。
- 特に、定数を乗除する場合、*2, *3, *4, *5, *8, *10, *16, *256, /2, /4, /8, /16 は、シフト命令を作るので、さらに高速化される。
- 単独で動作するモードとサブルーチンとして動作するモードの2つのエントリーアドレスをもっている。
- RUN時、システムによって変更を要するのはRTP(Run Time Package)のみで、オブジェクトは互換性がある。

May 21, 1982 Written by KXM

ラベル
- 英数字の任意長の文字列（すべて識別）
  @, A～Z, 0～9
- 数字のみからなる文字列も許されているので、行番号とみなす事ができる。
- ラベルは、GOTO, GOSUB, の飛先等にだけ、つければよい。
- ラベルは1行中の先頭から始める必要がある。

変数　英字で始まる任意長の英数字列（すべて識別）
A～Z, 0～9, および @
すべてゼロページに使用順に割当てられる。
合計100個まで可。

（例）　A, ABC01, FLAG

配列　2バイト型 $\boxed{\text{変数}}$ [e], 1バイト型 $\boxed{\text{変数}}$ #[e]
変数が配列の先頭のポインタとなる
e：式

（例）
XYZ[W], ARRAY#[Y*10+X]
A= $1000
A[3] = 500 ……

$1000		A[0]
$1001		
$1002		A[1]
$1003		
$1004		A[2]
$1005		
$1006	$01	A[3]
$1007	$F4	
$1008		

<DIMENSION宣言>（2バイト型）

```
A=$1000 : B=A+200 : C=B+60
```

<BASIC>

```
DIM A(100), B(30), C(10)
```

定数

- 10進定数　−32768 ～ 32767
- 16進定数　$0 ～ $FFFF（$をつける）
- リテラル　"ストリングス"

  1文字または2文字

  1文字 …… 0 文字 (ASCII)

  2文字 …… 1文字目 2文字目 (ASCII)

  DATA, PRINT 文中は任意長
- ラベル定数　&ラベル

  ラベルの示すアドレスを定数として使用する場合。

  ラベルは、定数で定義する事もできる → CONST

(例)

```
TABLE DATA "GAME OVER", ....
        :
FOR L=&TABLE TO &TABLE+100
        :
NEXT L
```

式

左から優先

優先の順序を変えるときは（ ）を使用する。

A=(B*C)+(D/E) の場合、前のカッコは、オブジェクトを作らないので、カッコを活用してリストを見やすくすることができる。

演算子

+, −, *, /

比較演算子

符号付 { >, >=, <, <= }　符号なし { >>, >>=, <<, <<= }　共通 { =, <> }

関数

e：式

RND[e]	0～$\lvert e \rvert - 1$ の乱数
ABS[e]	$\lvert e \rvert$
MOD[e_1, e_2]	$\lvert e_1 \rvert / \lvert e_2 \rvert$ のあまり
AND[e_1, e_2, …, e_n]	$e_1 \wedge e_2 \wedge \cdots \wedge e_n$ ビットごとのAND
OR[e_1, e_2, …, e_n]	$e_1 \vee e_2 \vee \cdots \vee e_n$ ビットごとのOR
XOR[e_1, e_2, …, e_n]	$e_1 \oplus e_2 \oplus \cdots \oplus e_n$ ビットごとのEX-OR
SGN[e]	eの符号 $\begin{cases} e > 0 & 1 \\ e = 0 & 0 \\ e < 0 & -1 \end{cases}$
GET$	文字入力、キー入力待ちをする。
KEY$	リアルタイムキー入力。ASCIIコード キーを押していない場合の値は $00
!P[e_1, e_2]	座標$(x, y) = (e_1, e_2)$のドット が点燈していれば1，消燈していれば 0を値とする。
SWAP[e]	BA ← AB
BYTE[e]	符号付16bit → 符号付8bit
WORD[e]	〃 8bit → 〃 16bit

FUNC[e_0, e_1, e_2, ……, e_n]

PC ← e_0 , STACK ← e_1 ~ e_{n-1} BA ← e_n

e_1 ~ e_n を引数として、アドレス e_0 のサブルーチンを実行した後のBAの内容を値とする。

関数側の引数を受ける変数は、PARAMETER 文で指定できる。

(例 1.)

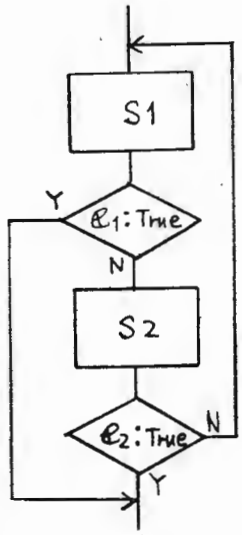
```
      A= FUNC[&XXX, B]

XXX   PARAMETER PT
      M= PT*V+W
      RETURN[M]

      B= FUNC[ FUNC[&ABC, D], FUNC[&PQR, S]]

      C= FUNC[ $FOB1, X, Y, Z]
```

ループ制御文

$\mathcal{E}$: 式　　$\mathcal{E}$: True …… $\mathcal{E} \neq 0$

◦ REPEAT - UNTIL

```
REPEAT
    S1
ESCAPE ℰ1
    S2
UNTIL ℰ2
```

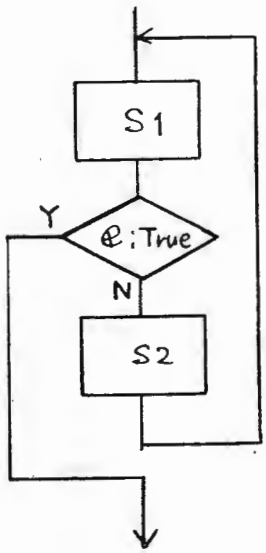


(BASIC)

```
100  S1
     IF ℰ1<>0 THEN 200
     S2
     IF ℰ2=0 THEN 100
200
```

◦ WHILE - ENDWHILE

```
WHILE
    S1
ESCAPE ℰ
    S2
ENDWHILE
```

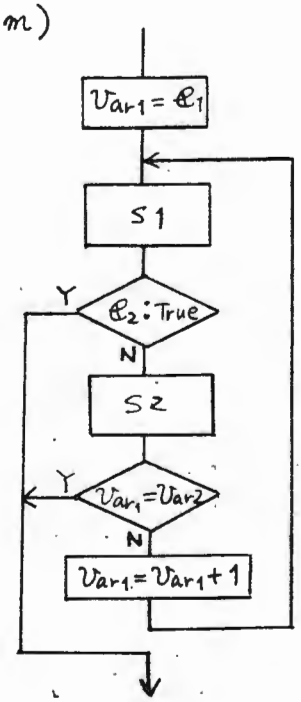


(BASIC)

```
100
     IF ℰ<>0 THEN 200
     GOTO 100
200
```

○ LOOP - EXIT　　e:式, Var:変数, m:定数

LOOP　$Var_1 = e_1$ TO　Var_2 (または m)



ループカウンタに IX レジスタを使用し、CPX で条件判断をするので FOR-NEXT ループよりもはるかに高速である。
ステップは 1 または −1 (TO を DOWNTO に変える)で、最終値が変数または定数でなければならない点が FOR-NEXT と異なる。
Var_2 の値は、ループ内で変更してはいけない。

○ FOR - NEXT

FOR　$Var = e_1$ TO　e_2 (STEP e_3)

NEXT　Var

一般の BASIC と同じ。
スタック操作を含むため、ループへの飛び込みやループ外への飛び出しはできない。よって、ESCAPE も使用できない。
e_3 が 1 または −1 で、e_2 が変数のみか、定数のみの場合は LOOP - EXIT におきかえる事により高速化できる。

・ESCAPE の説明

REPEAT－UNTIL , WHILE － END WHILE
LOOP－EXIT ループ内で、ある条件によって、ループから脱出する目的で使用する。

同一ループ内でいくつ用いても良いし、使わなくても良い。

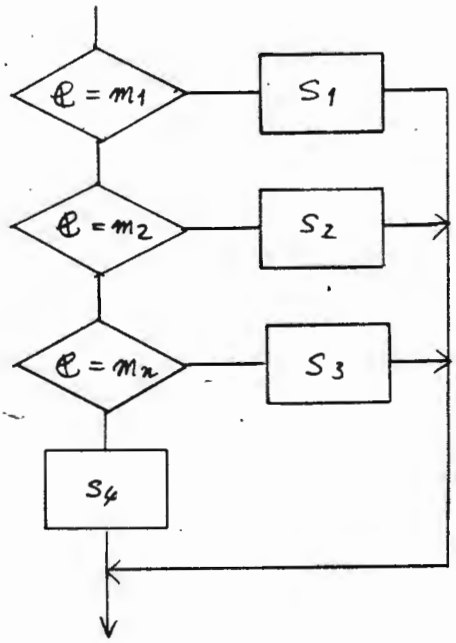
WHILE － END WHILE 内で ESCAPE を使わない場合、終了条件がなくなり、無限ループを構成する事になるので、注意する必要がある。

REPEAT－UNTIL 文は、WHILE － END WHILE 文に含まれているが、使用頻度、プログラムの読み易さなどの理由から、採用している。

これらのループはすべて終了条件を設定するので混乱が少なく、デバッグが容易である。

CASE 文

CASE e
OF m_1
S_1
OF m_2
S_2
OF m_n
S_3
DEFAULT
S_4
ENDCASE



○ DEFAULT はなくてもよい

○ m は $m_1, m_2, \cdots m_n$ と書ける
この場合は、or 結合

マルチステートメントは：で区切る。

ステートメント	書式
GOTO	◦ GOTO [ラベル] ◦ GOTO [16進定数] (そのアドレスへ JMP)
GOSUB	◦ GOSUB [ラベル] ◦ GOSUB [16進定数] (そのアドレスへ JSR) ◦ GOSUB [ラベル] [e_1, e_2, …… e_n]
CALL	◦ CALL [e_0, e_1, e_2, …… e_n] PC ← e_0 , STACK ← e_1 ～ e_{n-1} , BA ← e_n ◦ CALL [& [ラベル], e_1, e_2, …… e_n] ラベルを書く場合。

IF

IF e
[S1]
ELSE
[S2]
ENDIF

IF e
[S1]
ENDIF

$$\begin{cases} e \neq 0 \text{ (真)} & S1 \\ e = 0 \text{ (偽)} & S2 \end{cases}$$

CLR　◦ スクリーンをスペースクリアし、カーソルを左上へ戻す。

PUSH, PULL　◦ PUSH Var1, Var2, ……, Varn

[　]

PULL Var1, Var2, ……, Varn

RETURN　◦ サブルーチンのリターン

RETURN[e]　◦ 関数の終りに、eを値として持ち帰るリターン

PARAMETER 変数　◦ 関数の始めに置き、引数を変数にセットする、変数は任意数。

PRINT　　。PRINT　変数またはリテラルなど、

フォーマッタ

; 改行なし

, 8キャラクタ、ゾーン指定

CHR$␣e　1バイト 文字出力

HDT␣e　1バイト 16進出力　□□

HDF␣e　2バイト 16進出力　□□□□

TAB␣e　タブ出力.　TAB(e)と書く方が良い。

INPUT　　。INPUT (リテラル) Var1, Var2, …

リテラルは省略可

例　INPUT A, B, C

INPUT "X=" X

数値入力なので、数字以外の入力をした場合

RE-ENTER の表示が出て再び、入力待ちとなる。

数値の最後は CR でしめくくる。

DELAY[e]　e msec のディレイを行なう。

CURS[e1, e2]　画面の文字座標(e1, e2)へカーソルを移す。

!W[e1, e2]　座標(e1, e2)の点を白にする。

!B[e1, e2]　座標(e1, e2)の点を黒にする。

!R[e1, e2]　座標(e1, e2)の点の白黒をひっくり返す。

NEG　グラフィックの白黒を全画面にわたって、ひっくり返す。

DATA　1バイト単位で、データやプログラムを置く。(FCB相当)
DATA 定数, 定数, ……
リテラルの場合、連続記法が可能.

例　DATA 1, $55, "ABCD", "P", $0D

2バイト単位で、データや、アドレスを置く場合(FDB相当)は、!を前置する。

例　DATA !1, !100, !$1234, !LABEL

混用できるので、ジャンプ・テーブル等が容易に作れる。

例　DATA "INVADER", !UFO

INITIALIZE　スタックポインタの設定、システムのイニシャライズルーチンをコールする。これらは、RTPを用いるので、オブジェクトは同一通常メインルーチンの場合に先頭に書く。

ORG 定数　オブジェクトのロケーションを設定。(ORG相当)
指定のない場合は、RTPで設定された値となる。

CONST　CONST [ラベル1] = m_1, [ラベル2] = m_2, …
ラベルを定数宣言する。(EQU相当)
そのラベルを使う前に宣言しなければならない
m_1, m_2は定数(ラベル定数はのぞく)

STOP　Run Time Package のジャンプベクトル中のストップのベクトルへ行く.

END　プログラムの最後に必要で、コンパイル時には、この文によって、コンパイルを終了する。OBJECT実行時はSTOPと同じ.

*　コメント文.(ラベルエリアには書けない)