ND80ZⅢ ZB3BASIC操作説明書

(有)中日電工

目次

〒463-0067 名古屋市守山区守山2－8－14
パレス守山305
有限会社中日電工
TEL052-791-6254 Fax052-791-1391
E-mail thisida@alles.or.jp
Homepage http://www.alles.or.jp/~thisida/

2010. 9. 25 Rev. 1. 0

1章　基本操作

1. ZB3BASICの起動

ZB3BASIC（ND80Z3BASIC）はND80ZⅢとパソコンとを接続して、リモート接続プログラムから起動します。
まずは「リモート接続プログラム操作説明書」に従って、ND80ZⅢとパソコンをUSBケーブルと接続して、リモート接続プログラムを実行して、ND80ZⅢをリモート操作できるところまで準備してください。

リモート接続プログラムからZB3BASICを起動するには、[z]キーを入力します。

```
logfile nd80zlog¥09071137.txt open

ＮＤ８０ＺⅢに接続しました
0001 0000 - ｚ              ［ｚ］キーを入力します
```

すると画面がクリアされて、下の表示になります。

```
*** nd80z3 basic ****
>
```

これ以後は、BASICのコマンドとマシン語モニタコマンドが入力できます。

[注記]リモート接続プログラムでは、ND80ZⅢの7セグメントLEDには、コマンドプロンプト画面の表示と同時にアドレス表示、データ表示が行われますが、ZB3BASICにエントリするとND80ZⅢ本体のLED表示のためのDMAが禁止されるため、7セグメントLEDは消灯します。

ZB3BASICシステムにはBASICインタプリタのほかにマシン語モニタが含まれていて、どちらも簡単な操作で使えるように考慮されています。
ここでは両機能に共通な基本操作について、まず説明します。例として簡単なBASICプログラムの作成、実行の仕方を説明します。
基本操作について説明するのが目的なので、BASICの命令そのものの説明や使い方については、特に説明してはいません。ここではとりあえず使用例の通りに操作してみて、基本的な操作方法をマスターして下さい。

ZB3BASICが起動すると、MS－DOSと同じように入力プロンプトマーク（＞）が出てキー入力待ちの状態になります。
このときBASICの命令をキーボードから入力すると、その命令はただちに実行されます。

＞PRINT　2＋3[Enter]

と入力してみて下さい。
＋マークのようにキーの上側に書いてある文字は[Shift] を押しながら入力します。
入力の最後は必ず[Enter]を押します。
上のようにキーを操作するとすぐ下に2＋3が計算されて、5が表示されます。

[注記1]
[Enter]、[Shift] のように[　] で囲んだキーはキーボードの左右、下部にある特殊機能キーを示します。
[注記2]
ZB3BASICの命令、コマンドは英大文字のみを使います。
リモート接続プログラムでは英小文字を使いますから、ZB3BASICに文字コードを渡すルーチンが小文字→大文字にコード変換を行っています。
ですから小文字のまま、
＞print　2＋3[Enter]と入力してもエラーにならずに正しく実行されます。

2.　スクリーンエディタ（画面編集機能）

ZB3BASICモードでは通常のキー入力待ちの状態では常にスクリーンエディタが作動していていつでも簡単にプログラムの作成や修正ができるようになっています。
スクリーンエディタはとても便利な機能ですが、その性質を理解しないで使うと、入力ミスをしてしまいます。

スクリーンエディタのポイントとして以下のことを理解しておいて下さい。
キーボードから入力していくと、一つキーを押すごとにディスプレイ画面にその文字が表示されますが、この段階では画面（つまりスクリーン）に表示されるだけで、システムソフトの入力データとしてはまだ一字も送られていません。
ですからこの時点では入力した文字の訂正や追加が自由にできます。最後に[Enter]キーを押すとその一行分の文字が初めて入力されます。
つまりスクリーンエディタでの入力のカギは[Enter]キーです。
[Enter]を押した時、その一行が入力データとして扱われる、ということをよく理解しておいて下さい。
そしてその一行分の文字は、いまキーボードから入力した文字に限らず、プログラムで画面に表示された文字でも、或いはLISTコマンドなどで表示されたプログラムリストでも構いません。とにかく現在ディスプレイ画面に表示されている全ての文字が入力の対象になります。
例えば DM C000,C030 というマシン語モニタコマンドを入力したいとき、キーボードからそのまま

＞DM　C000,C030[Enter]

と入力すればよいのですが、たまたま画面上で同じコマンドが使われていて、仮に10行位上に DM C000,C030 と表示されていたら、［↑］［↓］ などのカーソル移動キーを使って、カーソルをその DM C000,C030 と表示されている行まで移動してから[Enter]を押せばよいのです。
またこのとき DM C000,C030 ではなくて、DM C100,C150 と入れたければ、これも下のようにカーソルをその変更位置に持って行って部分的に直したあとで[Enter]を押します（カーソル移動キーの使い方については「3.　特殊機能キー」を参照して下さい）。

＞DM　C000,C030　　カーソルを変更したい文字に移動する
＞DM　C100,C030　　1をキー入力する。
＞DM　C100,C030　　カーソルを移動する
＞DM　C100,C150　　1、5と入力する
＞DM　C100,C150_　　[Enter]を押す（カーソルはこの行の中ならどこにあっても構わない）

この機能はBASICのプログラムを作成、修正するときに利用すると効果的です。
はじめにプログラムを入力するときは1行ずつキー入力していくのですが、入力したプログラムの文の一部を修正しなければならないときがよくあります。
そのようなとき、その行を初めから入れなおすのでは大変です。そこでこのスクリーンエディタの機能が生きてきます。
LIST命令によって画面上にその行を表示した上で、カーソルを移動して必要な修正を行った後[Enter]を押せば修正が完了します。

[例]下のように入力してみて下さい。
＞10　A＝0[Enter]
＞20　PRINT　A,　[Enter]
＞30　A＝A＋1[Enter]
＞40　GOTO　25[Enter]　（本当は GOTO 20　が正しいがここではプログラム訂正の例として、わざと間違えて入力する）

プログラムが入ったかどうか、念のために確認してみます。
プログラムを確認するにはLISTコマンドを使います。
LIST[Enter]と入力すると、今キーボードから入力したプログラムがそのまま表示されます。

```
＞LIST[Enter]
    10　A＝0
    20　PRINT　A,
    30　A＝A＋1
    40　GOTO　25
＞_
```

BASICのプログラムはこの例のように行番号（ここでは10～40）をつけて入力します。
それではこのプログラムを実行してみます。BASICプログラムを実行するにはRUNコマンドを使います。
RUN[Enter]と入力すると、次のように表示されます。

```
0                       最初の実行結果（行番号10～20）
ERR：31                 エラーコード
```

```
    40  GOTO  25          エラーが発生した行が表示される
>_                        キー入力待ちになる
```

そこで［↑］、［→］を使ってすぐ上に表示されている 40 GOTO 25 のところへカーソルを移動します（カーソル移動キーの使い方については「3.　特殊機能キー」を参照して下さい）。

```
    40  GOTO  25          カーソルをセットする
    40  GOTO  20_         0を入力し、［Enter］を押す。
>_
```

これで修正は完了です。もう一度RUN[Enter]と入力してみて下さい。
今度は正しくプログラムが実行され、画面に連続して実行結果が表示されます（このプログラムは0、1、2、…と順に画面に数を表示するプログラムです）。
なおBASICのプログラムを中止するときは、[Ctrl]B　キーを使います。
［Ctrl］を押しながら　B　を押すとBASICプログラムの実行が中止され、もとのキー入力待ちの状態に戻ります。

［注意］
既に説明した通り[Enter]を押すことによってカーソルのある行全体が入力され、その場合カーソルが行のどの位置にあっても結果は同じであることに十分注意して下さい。

```
  10  PRINT  A ,B ,C  _      %＄AB      123
                      ↑
```

下線部だけを入力するつもりで、ここで[Enter]を押しても同じ行にある文字はすべて入力されてしまいます（カーソルマークの後ろの　%＄AB　　123　も一緒に入力されてしまいます）。
上で[Enter]入力後、LIST　10[Enter]と入力すると上の一行がそのまま表示されます。

```
>LIST  10
    10  PRINT  A ,B ,C        %＄AB      123
>_
```

［注記］
スクリーンエディタの使用例として、プログラムの修正について説明してきましたが、スクリーンエディタはプログラムだけではなくて、キー入力される全てに対して動作します。
下のようにキー入力してみます。

```
>A＝0[Enter]
>A＝A＋1：PRINT  A[Enter]
1                          上の命令が実行された結果が表示される。
>_
```

BASICの命令を行番号をつけないで入力すると、その命令はただちに実行されます（ダイレクトモード）。
ここで［↑］キーを使ってもう一度カーソルを上の命令文（A=A+1:PRINT A　）の位置まで移動して、[Enter]を押してみて下さい（［→］で文の終わりにカーソルを合わせる必要はありません。カーソルはその行のどこにあっても[Enter]の効果は同じです）。
表示されていた数が＋1されることに気が付くでしょう（結果が1だったところが2になります）。
このように[Enter]を押したときにその一行分の文字が入力されるのであって、その文が画面のどこにあっても、またいつ表示されたものであっても同じように入力されることをよく理解して下さい。

3.　ページモード

スクリーンエディタは現在表示されている画面の中でしか使えません。[↑]を押し続けて画面の一番上までいくとそこから上の行（スクロールアップして消えてしまった行）を再表示させることはできません。
そのような場合に「ページモード」にするとスクロールアップして消えてしまった行を再表示させることができます。
［PageUp］キーを押すと画面左下に［page　mode］と表示されます。この表示のときに［PageUp］キーを押すと押す度にちょうどページを上にさかのぼるように一行ずつ戻って表示されます。［PageDown］キーを押すと［PageUp］キーと逆の動きをします。
［Home］キーを押すとバッファの先頭行までさかのぼって表示されます。
［End］キーを押すと［page　mode］が解除され、通常表示されていた最後の画面に戻ります。

ページモード表示のときもスクリーンエディタは働いています。ページモード表示でもつねに現在表示されている一画面の中でのみ[↑][↓][→][←]キーでカーソルを移動して表示されているプログラムを書き換えることができます。またLIST表示やその他のコマンド入力もできます。しかしページモードはすでに表示し終わった過去の画面を表示しているのですから、その途中で現在の実行結果を上書き表示すると、勘違いをするもとになりますから、プログラム行の修正とか過去の表示の再確認以外の作業は[End]キーでページモードを終了してから行った方がよいでしょう。
ページモードの記憶バッファの容量は6400行分です。

4. 特殊機能キー

4．1 [Shift]

＝、＋、＊などの特殊記号や、英大文字を入力するときは、[Shift] キーを押しながら、そのキーを押します。
[Shift] は同時に押すことによって、そのキーの上側にマークしてある文字の入力を選択する働きをします。

4．2 [Enter]

一行分の文字列を入力バッファに転送します。
[Enter]を押さない限りは、文字キーを押して、それが画面に表示されていても、ただ表示されているだけで、本当に入力はされていません。
[Enter]を押すことによってはじめて、システムが入力データとして受け付けます。

4．3 [↑] [↓] [←] [→]

カーソルを矢印方向に1字ずつ移動します。

4．4 [Insert]

現在カーソルのある文字から後を1字分だけ右に移動して、カーソルのある位置にスペースを作ります。
文字を追加する場合に使用します。

4．5 [Deiete] [Backspace]

[Insert]とは逆に文字を消したいときに使います。現在カーソルのある位置の左側の1文字を消して、それより右にある文字を順に左につめます。

4．6 [Ctrl]B

BASICプログラムの実行やAUTOモードあるいはLISTコマンドによるリスト出力などを途中で打ち切るときなどに使用します。([Ctrl]キーを押しながら[B]キーを押します)
BASICプログラムの場合には、CONTコマンドを入力することにより、実行を再開させることができます。
なおマシン語プログラムの実行中は、[Ctrl]B では打ち切ることができません(それらを打ち切るには、[Ctrl]Cを入力するかMSDOSプロンプト(コマンドプロンプト)の右上の[×]ボタンをマウスでクリックします)。

4．7 [Ctrl]C

ZBKシステムの実行を打ち切ります。通常は／EコマンドでZBKシステムを終了しますが、マシン語サブルーチンの実行中や何らかの理由でND80ZⅢが暴走してしまって[Ctrl]B入力が受け付けられない場合に使います。暴走した状態では[Ctrl]Cも受け付けられないときがあります。そのような場合にはMSDOSプロンプト(コマンドプロンプト)の右上の[×]ボタンをマウスでクリックします。警告メッセージがでますが、「はい」をクリックして強制終了します。
[Ctrl]Cまたはコマンドプロンプトの右上の[X]をクリックして強制終了するとログファイルは保存されません。

4．8 [PageUp]

ページモードにエントリします。[PageUp]キーを押す度に表示画面が1行ずつ戻って表示されます。

4．9 [PageDown]

ページモード中に[PageDown]を押すと、[PageUp]と逆に表示が1行ずつ進みます。通常表示の最終画面まで進むとページモードが解除されます。

4．10 [Home]

[Home]キーを押すとページモードバッファの先頭の画面が表示されます。

4．11 [End]

[End]キーを押すとページモードが解除され、通常表示の最終画面に戻ります。

●オートリピート機能

キーを少し長く押したままにしていると、そのキーの文字が続けて入力されます。

カーソル移動キー［↑］［↓］［←］［→］や[Backspace][Delete][Insert][PageUp][PageDown]はそのキーの機能が連続して働きます。

1 プログラムの作成

1．1 キーボードからの入力

下のように入力して下さい。行の最後で[Enter]を押すことについては既に説明した通りです。命令の意味についてはここでは説明しません。この章ではまず基本的なプログラムの作成方法や実行方法について説明します。
入力ミスに気がついたときは、[Enter]を押す前ならばその場でカーソルを移動して修正できます。

```
＞10　A＝2[Enter]
＞20　PRINT　A[Enter]
＞30　A＝A＋5[Enter]
＞40　GOTO　20[Enter]
＞_
```

これである作業をするBASICプログラムが作成されました。
この例のような、ある働きをする命令のまとまりをプログラムと言います（プログラムファイルとよぶ場合もあります）。
プログラムは、TXTファイルとしてハードディスクに保存しておき、あとからいつでも使うことができます。
BASICプログラムは、この例のように必ず行番号をつけます。
BASICプログラムを行番号をつけないで入力すると、その命令はただちに実行されてしまいます。
行番号は1～32767の整数で、プログラムは入れた順番とは関係なく、行番号の順に整理されます。
普通は上の例のように、10、20、30、と10番飛びに番号をつけます。こうしておくと、（BASICプログラムは行番号の順に整理されるため）あとからの追加が楽にできます。
例えば上の例で行番号10と20の行の間に、もう一行追加したくなったときは、行番号を15にしてその行を書けば、後から追加したその行番号15の行は、行番号40の後ではなくて、行番号10と行番号20の行の間に挿入されます。（もしも行番号を1、2、3とつけておくと、その間に新しい行を挿入することができません。）
AUTOコマンドを使用すると行番号をシステムが作成して表示するのでプログラム作成の能率が上がります（4章参照）。
RENコマンドで行番号を整理して付け直すことができます。

[注記]
このシステムのBASICではプログラムの終わりを示す命令（END文）は必要ありません。また特に必要があって使う以外はSTOP文も書く必要はありません。プログラムの最後にSTOP文が書いて無くても、省略されているものとして正常に実行されます。

1．2 プログラムの確認

全部入力を終わったら、本当に正しく入力できたかどうか確認してみます。入力したプログラムを確認するには、LISTコマンドを使います。

```
＞LIST[Enter]
　　10　A＝2
　　20　PRINT　A
　　30　A＝A＋5
　　40　GOTO　20
＞_
```

1．3　プログラムの実行

プログラムの入力が完了し、正しく入力されていることを確認したら、いよいよ実行してみます。
BASICプログラムを実行するにはRUNコマンドを使います。このコマンドの入力によりRUNモードになりプログラムが実行されます。

```
＞RUN[Enter]
2
7
12
```

17
.
.

上のように、2、7、12……と、5ずつ増えた数が縦にどんどん表示されます。
この例と違う数字が表示されたり、ERR：××に続いてプログラムの一部が表示された場合には、入力したプログラムに誤りがあります。もう一度LISTコマンドでプログラムを表示させて、よく確認して下さい。
誤りがみつかったら、この章の「1．6　プログラムの修正」の説明に従って、正しく直してから、もう一度RUNコマンドを入力して下さい。

［参考］
この実行例で、表示される数字がディスプレイ画面の一番下の行までくると、そのあとは画面全体が下から上に移動しながら、一番下の行に新しい数字が表示されていきます。
これをスクロール（画面が一杯になったら、画面を順に一行ずつ上に移動させ、文字が画面の下にかくれてしまわないようにするコントロール方法）機能といいます。
ZB3BASICの表示は、すべてスクロール表示になっています。
画面の上端から消えてしまった行は［PageUp］キーによって表示を戻すことができます（1章2．2参照）。

1．4　プログラムの中止（［Ctrl］B）

前記のプログラムは、どんどん数字を表示していくだけで、どこまで行っても止まりません。
実はそのままにしておけば、ある時点では止まるのですが、それはかなり先のことになります（やがて計算結果がオーバーフローすれば止まる）。
そこでとにかくもうこのへんで実行を中止したい、という場合に、プログラムをブレイクすると言います。
[Ctrl]Bを使います。
[Ctrl] を押しながら英字のBキーを押すとブレイクが受け付けられBASICプログラムの実行が中止されます。

```
    break  in  30（または20、40）
```

の表示が出て、キー入力待ちの状態になります。プログラムの実行が中断した状態ですが、ここでプログラムの修正をするとかその他の作業を自由に行うことができます。

［注記］
break　in　××　の表示は、その××の行を実行中にブレイクしたことを示しています。
［注意］
マシン語プログラムの実行中はブレイクはできません。
マシン語のプログラムの実行を中止したい時は、［Ctrl］Cを押すかMSDOSプロンプト（コマンドプロンプト）の右上の［×］ボタンをマウスでクリックします。

1．5 プログラムの実行再開

前項でブレイクしたあと、その続きから実行を再開させることができます。
CONT［Enter］と入力してみて下さい。再びプログラムの実行が開始されて、さきほどの続きの数値から表示していきます。
CONTコマンドの入力前に変数の内容を確認したり、その値を変更することもできます。
変数の内容を確認する例として、PRINT命令をダイレクトに実行させてみます。

```
＞PRINT　A［Enter］
112
＞_
```

さきほどブレイクした直前に表示されていた値か、その次の値が表示されます（ここでは表示例として、112にしています）。
次に変数の値を変更してみます。

```
＞A＝12345［Enter］
＞_
```

これで変数Aに12345がセットされました。この状態でCONTコマンドを入力して、実行を再開してみます。

```
＞CONT[Enter]
12350
12355
12360
  ・
  ・
```

変更した値から表示されることが確認できます。
このようにBASICプログラムは、実行中に任意の時点でプログラムの実行を中止しその時の変数の値を確認したり、また必要ならば変数の値を変更した後で、再び中止した所から実行を再開させることができます。

[注意]
ブレイクした後で、プログラムを修正した場合には、CONTコマンドで正しく実行が再開されない場合があります。また修正によっては、CONTコマンドの入力によって、システムが暴走してしまう可能性がありますから、プログラム修正をしたあとでCONTコマンドを入力してはいけません。
プログラム修正をしたあとでも、RUNコマンドではじめから実行を開始した後に、ブレイクして、それからCONTコマンドを入力することは構いません。またLISTコマンドでプログラムの内容を確認するだけならば問題はありません。
[ブレイク]→[プログラム修正]→[CONT]……不可
[ブレイク]→[プログラム修正]→[RUN]→[ブレイク]→[CONT]……可

1．6　プログラムの修正

まず、プログラムを表示してみます。

```
＞LIST[Enter]
      10  A＝2
      20  PRINT  A
      30  A＝A＋5
      40  GOTO  20
＞_
```

[↑]と[→]を使って、30　A＝A＋5　の＋記号のところにカーソルをセットします。

```
      30  A＝A＋5

      30  A＝A＊5      ＋を＊に変更します。そのあとここで[Enter]を押します。
＞_   40  GOTO20       カーソルが次の行に移動して次の修正ができるようになります。
```

修正がもう必要ないときは[↓]を押してリストの終わりまでカーソルを移動します。
念のためにもう一度LIST[Enter]と入力してみて下さい。＋が＊に変更されているはずです。
このように修正したい文をもう一度入れ直さなくても簡単に修正できるのがスクリーンエディタの特徴です。
なお、よく注意してみると気がつくように、はじめ入力するときは行番号を＞の次にすぐ入力し、行番号と文との間をつめて入力しても、LIST出力すると行番号は6字分を占め、その次に必ず1字の空白がとられます。
その分だけ文全体が右に移動するため、もし行一杯に文を書いた場合は、LIST出力させると終わりの方が行からはみ出てしまいます。
はみ出た部分は下の行に表示されます（つまり2行に渡って表示されます）が、実行にはさしつかえありません。
しかし、このようにして2行に渡って表示されている文に対して修正を行い[Enter]を押すと、2行目の部分は無視されてしまいます。したがって余り長い文を書かないように注意して下さい。

[注記]
ここではスクリーンエディタの特徴をはっきり出すため、LIST[Enter]でプログラム全体を表示させましたが、この例のようにある一行だけ（ここでは行番号30）修正すればよい場合には、LIST　30[Enter]と入力すれば、その一行だけ表示させることができます。そうしておいてから必要な修正を行ったほうが、この場合は能率的です。

2.　エラーコードとエラーメッセージ

2．1　エラーコード

前項で修正したプログラムを再び実行してみます。
RUN[Enter]と入力すると、今度は5倍ずつ大きい数が表示され、しばらくすると下のような表示が出て実行が中止されます。

```
＞RUN[Enter]
2
10
50
250
1250
 ・
 ・
0．888178E＋36
0．444089E＋37
0．222044E＋38
ERR：　8
      30　A＝A＊5
＞_
```

このようにBASICインタプリタは実行中のエラーを教えてくれます。
これは行番号30の実行中に計算結果がオーバーフローしたことを示しています。
プログラムによっては実行中に様々なエラーが発生します。そのような時に、BASICインタプリタは、そのエラーの原因をエラーコードで表示するとともに、エラーが発生した行を表示してブレイクします。
エラーコードとその意味については9章で説明します。

2．2　エラーメッセージ

BASICプログラムの実行中以外にエラーが発生した場合にも、そのエラーを知らせるメッセージが表示されます。
それらのメッセージについては、それぞれの機能やコマンドの項で説明してありますから、それを参照して下さい。

3.　命令の省略形

命令の省略形が認められるのがZB3BASICの大きな特徴の一つです。
例えばいちいちPRINTと入力しなくてもP．でもPR．でも構いません（このPRINTに限っては、他のパーソナルコンピュータのBASICでも、？マークを省略形として使えるものが多いようですが、その他の命令については省略形が使用できないのが一般的です）
省略形で入力するようにするとプログラム作成の能率があがります。
省略形で入力した命令も、次にLISTコマンドで確認するとちゃんと省略しないもとの命令形で表示されます。
省略形については、各命令の解説のところに掲げてあります。

4.　スペースについて

BASICプログラムの作成をする場合に、命令と変数等との区切にはスペースが必要です。
例えば、10　PRINT　A　を、10　PRINTA　と入力するとエラーになります（PRINTAという変数名として受け取られてしまいます）。ただし省略形の後ろにはピリオド（．）があって区切りが判るためスペースはなくても構いません（その場合でもLIST出力では、間にスペースを挿入した形で表示されます）。
別の例で、IF　A＝B　THEN　PRINT　A　という文で、A＝Bのように記号と変数、数値の間にはスペースは必要ありません。その他の部分のスペースを省略すると命令と変数名がつながって区別できなくなるため、省略はできません。
また逆に各単語のつづりの中にスペースを入れることは許されません。
例えば、10　P　R　INT　A　は、エラーになります。
なお単語の区切としてのスペースは2桁以上続けて使用しても、LIST出力時にはカットされて1桁のスペースしか表示されません。
ただし、REM文や” ”の中、またはDATA文の文字データのスペースはカットされずにもとのまま残ります。

5.　プログラムの復活

BASICで書かれたプログラムはNEWコマンドを実行することにより消去されます。[Ctrl]CでZB3BASICを終了したり、ND80ZⅢの電源を切ることでも消去されてしまいます。しかしその場合でもボタン電池によってRAMの中身をバックアップしているので、メモリの中から完全に消えたわけではありません。
またマシン語サブルーチンを実行中はブレイクが利かないので、中止するには[Ctrl]Cによる強制終了しか方法はありません。
このようにやむなく終了したために消えてしまったBASICプログラムはHELPコマンドで復活させることができます。
この章で作成したプログラムがまだ残っていることをLISTコマンドで確認してからNEW[Enter]と入力してみてください。
次にLISTコマンドを入力しても、プログラムは1行も表示されません。
そこで今度はHELPコマンドを入力してみます。

```
＞HELP[Enter]
TEXT　4004－4049
ヘンスウ　DFFB－DFFF
＞_
```

LISTコマンドを入力してみて下さい。消えたはずのプログラムが元通り表示されます。

[注意1]
例外として、変数を全く使用しないか、または変数名A%～Z%、A＄～H＄のみを使用するプログラムが書かれている時にNEWを実行した場合（またはZB3BASICを終了した場合）にはHELPコマンドの入力によってシステムのコントロールが正しく行われなくなります。このような場合には再びNEWコマンドを実行してください（この条件の時にはNEWを繰り返しても、HELPコマンドは使用できません）。
[注意2]
マシン語プログラムが暴走したときなどに[Ctrl]CでZB3BASICを強制終了したあと、再びZB3BASICにエントリしてHELPコマンドを入力しても、BASICプログラムは復活できないこともあります。 プログラムの暴走以外の場合でも、一度ZB3BASICを終了して、その後にZB3BASICに再エントリするとRAMの一部が書き換わってHELPコマンドによって完全なプログラムが復活できないときがあります。
[注意3]
この機能は、NEWやZB3BASICの終了や電源のOFFによって仮にクリアされたプログラムを復活させるもので、プログラムの上書き、変更によって書き換えたりマシン語プログラムの暴走によって破壊された場合には元に戻すことはできません。不測の事態や操作ミスなどに備えて、作成中のプログラムはできるだけこまめにハードディスクに保存してください（／SAVEコマンドでファイル名をつけてプログラムをハードディスクに保存することができます）。

6.　プログラムの消去

現在メモリに記憶されているプログラムを全部消したいときはNEWコマンドを使います。また、ある行だけを消したいときはその行番号のみを入力すれば消去できます。

```
10　A＝A＋1
20　PRINT　A
30　GOTO　60
40　PRINT　B
```

例えば、上のようなプログラムのうち、行番号30の行を消したいときは下のように、その行番号（ここでは30）だけを入力します。

```
＞30
＞_　　これで、30の行は消去されています。
```

またある範囲のプログラムを消したい場合には、DELETEコマンドを使います。

```
＞DELETE　100－150[Enter]
```

というように使用します。この場合には行番号100～150の間のプログラムが削除されます。

7.　プログラムの保存（ファイルの作成）

　BASICプログラムはファイル名をつけて、テキストファイル形式で保存することができます。／SAVEコマンドを使います。

［書式］／SAVE　ファイルネーム

　ファイル名のみを指定した場合にはZB3BASICが存在するフォルダ（通常はnd80z3フォルダ）にSAVEされます。同じファイル名があると上書きされます。
　ファイル名にドライブ名やフォルダ名を含めて記述することで、別のドライブやディレクトリにSAVEすることができます（使用例③）。
　ファイル名は名前部分が英数字及び、－_で8字以内、拡張子は3字以内で任意につけられます。TXTである必要はありません。つけなくても構いません。
　ハードディスクにはテキストイメージで保存されます。Notepad（メモ帳）などのテキストエディタで参照したり、修正、追加、削除などができます。またプリンタに出力することもできます。
　／SAVEコマンドではBASICシステムによって1行ずつBASIC内部コードをテキスト形式に変換していきます。このときプログラムは1行ずつ画面にリスト表示されます。

［使用例①］
＞／SAVE　TEST．TXT［Enter］
［使用例②］
＞／SAVE　MIHON［Enter］　　……拡張子はなくてもよい。
［使用例③］
＞／SAVE　D：￥BASIC￥TESTPRO．TXT［Enter］
［注記1］
　使用例③のように別のドライブや別のフォルダにSAVEすることもできます（上の①②例ではZB3BASICの存在するフォルダにSAVEされます）。
［注記2］
　使用例②のように拡張子をつけないで保存してもZB3BASICでの作業には支障ありません。Windowsではフォルダ内の表示をしたときにテキストファイルのアイコンがつかないため、整理がしにくいかもしれないことと、アプリケーションから開くことしかできない欠点があります。

8.　プログラムの読み込み（テキストファイルのLOAD）

　テキスト形式で保存されたファイルをBASICプログラムとしてND80ZⅢのRAMエリアに読み込むことができます。／LOADコマンドを使います。

［書式］／LOAD　ファイルネーム，aaaa

　ファイル名のみを指定した場合にはZB3BASICが存在するフォルダ（通常はnd80z3フォルダ）からLOADします。ファイルがみつからないときは　FILE　NOT　FOUND　と表示されます。ファイル名にドライブ名やフォルダ名を含めて記述することで、別のドライブやフォルダにあるファイルをLOADすることができます（使用例③）。
　aaaaは4桁の16進数で、省略することもできます。省略した場合には現在のテキストエリアの先頭からLOADされます。aaaaをつけるとaaaa番地からLOADされます。
　／LOADコマンドでテキスト形式のファイルが読み込まれるとき、BASICシステムによって1行ずつBASIC内部コードに変換してRAMに格納していきます。このときプログラムは1行ずつ画面にリスト表示されます。内部コードに変換するときに文法エラーが見つかるとそこでエラーコードが表示されLOAD作業は打ち切られます。
　メモ帳などのテキストエディタで作成したプログラムも文法的に正しければ／SAVEで保存されたファイルと同じようにLOADすることができます。

［注意1］
　LOADコマンドの動作は、NEW　＋　／LOAD、またはNEW　aaaa　＋　／LOAD、というように必ずNEWコマンドの動作を伴っています（LOAD前にRAMに存在したプログラムは失われます）。
　なおこのことはBASICテキストファイルについてのみあてはまります。バイナリファイル（マシン語プログラム、データファイル）を／LDコマンドでLOADしてもテキストエリアには影響を与えません。
［注意2］

メモ帳（Notepad）は問題ありませんが、WriteやWordでは通常はそれぞれの形式でファイルが作成されます。保存するときにテキスト形式を指定してもゴミが混じる場合があります。Writeなどで作成したファイルがうまくLOADできないときは、一度メモ帳（Notepad）で開いてゴミを削除して、メモ帳で保存してからLOADしてみてください。

[使用例①]
＞／LOAD　TEST. TXT[Enter]
[使用例②]
＞／LOAD　MIHON[Enter]　　……テキスト形式のファイルなら拡張子のついていないファイルでもよい
[使用例③]
＞／LOAD　D:￥BASIC￥TESTPRO. BAS, 9000[Enter]
この例のように別のドライブや別のフォルダにあるファイルをLOADすることもできます。上の①②例ではプログラムは現在のテキストエリアの先頭からLOADされます。③では9000番地からLOADされます。

[注意]
テキスト形式（拡張子の種類に関わらず内容がテキスト形式になっている）以外のファイルをLOADするとZB3BASICシステムが暴走してハングアップすることがあります。

9. ZB3BASICの終了

1）BASICを終了してリモート接続プログラムに戻るには、／REMOTEコマンドを使います。

＞／remote
BASICを終了しました
1001 C300 -

2）ZB3BASICを終了して、リモート接続の終了するには、[Ctrl]を押しながら、[E]キーを押します。

>0000 00C3 -
ndremote.exe を終了しました
logfile closed at Tue Sep 14 22:23:37 2010

10. ログファイルの保存

ログファイルを一旦閉じたあと、新しいログファイルを開くには／CLOSEコマンドを使います。
ログファイルはZB3BASIC＋リモート接続を終了した時点でCLOSEされて、ファイルとして保存されますが[Ctrl][C]によって強制終了すると、ログファイルは保存されずに破棄されてしまいます。
ZB3BASIC＋リモート接続プログラムを終了することなく、ログファイルだけを一旦CLOSEするために／CLOSEコマンドを使います。
現在開いているログファイルが閉じられて保存されたあと、新しいファイル名でロクファイルがOPENされます。

>/close
logfile closed at Wed Sep 15 14:15:48 2010

open new logfile

3章　BASICの文法

1．動作モード

BASICインタプリタの動作には、コマンドモードとプログラム実行モードの二通りがあります。コマンドモードでは命令を直接実行したり、プログラムの編集を行います。プログラム実行モードではあらかじめ作成した一連のステートメントを自動実行します。

1．1　コマンドモード

＞（プロンプトマーク）が表示されカーソルマーク ＿ が表示されているいる時（つまり普通のキー入力待ちの時）がコマンドモードで、このときに命令を入力するとただちに実行されます。
例えば、下のようにPRINT命令に続いて計算式を実行すると、ただちに実行されてその演算結果が表示されます。

```
＞PRINT　2＋3－7［Enter］
　　－2　　　　計算結果がすぐに表示される。
＞＿　　　　再び、次の入力待ちとなる。
```

もともとこのモードはBASICの機能というよりも、システムそのものの基本的なモードというほうが適切で、BASICに限らず、このシステムで使うコマンドを入力してただちに実行することができます（ZB3BASICにエントリ後は必ずこのモードになります。またBASICプログラムの実行中にブレイクしたときもこのモードになります）。
コマンドを実行する以外にもうひとつ、このモードの大きな機能があります。それはプログラムの作成、編集です。
プログラムは行番号をつけたステートメント（命令文）の集まったものです。
命令に行番号をつけて入力すると、上のようにすぐに実行されるのではなくて、そのかわりにユーザーズエリア（TEXTエリアとも言います）内にプログラムとして記憶されます。
つまりこのモードはコマンドモードであると同時に、プログラム作成モードであるとも言えます。

1．2　プログラム実行モード（RUNモード）

このモードは表記のように別名RUNモードといい、RUN［Enter］と入力すると、このモードになり、TEXTエリアにあるBASICプログラムをインタプリタが順に解読しながら自動的に実行します。（もしTEXTエリア内にプログラムが無い場合には、すぐにコマンドモードに戻りますが、別にエラーメッセージは表示されません）

2．BASICプログラムの構造

BASICプログラムは1以上の行からなっており、行には命令実行のための文が1つ以上含まれています。

2．1　行

行は行番号と文で構成されます。1行の長さは最大80バイトです。
行番号は1つだけ必要ですが、文は1つの行に複数個記述することができます。文と文との区切には：（コロン）を使います。

2．2　行番号

1～32767の範囲の符号無し整数を使用します。RUNコマンドでプログラムを実行させると、一部の命令を除いて、原則的にはこの行番号の順に実行されます。
行番号はあとからの追加修正を容易にするために10番飛び位で記述するのが便利です。

2．3　文（ステートメント）

文はBASICインタプリタが実行する場合の命令実行単位で、1つの行に複数記述することも許されます（マルチステートメント）。この場合は前述のように、：（コロン）で区切ります。

```
　10　A＝A＋1：PRINT　A：GOTO　30
　↑　　↑　　↑　　↑
行番号　　文　　文　　文
```

3.　BASIC文の構成要素

　BASIC文は命令語だけではなく、変数、定数、演算子等様々な構成要素によって成り立っています。

3．1　コマンド

　コマンドモードでのみ実行可能で、RUNモードでは実行不可能です（つまり行番号をつけてBASICプログラムの中で使うことはできない）。
　BASICのプログラムを作成したり、修正したり、またはディスクに保存したりするときに使用します。
　コマンドとしてはBASICに全く関係のない、マシン語プログラムを作成するときなどに使用する、マシン語モニタコマンドもあります。
　したがってそれらのコマンドを全部まとめてBASICの機能として説明するには、少し無理があります。
　BASICでは使用しないコマンドについてはそれぞれ必要な説明書で説明することにしてこの項ではBASICで使用するコマンドのみを示します。
　BASICで使用するコマンドには次のものがあります。

AUTO、CONT、DELETE、HELP、LIST、／LOAD、NEW、REN、RUN、／SAVE、SL、XL

　これらのコマンドの詳細については、次の章で説明します。

3．2 コマンド以外の命令

　一般にRUNモードのもとで実行される命令です。
　コマンドモードでも実行可能ですが、実行しても意味のないもの（REM、STOP）や、マルチステートメント以外では正しく実行されないものもあります（FOR～NEXTなど）。

CLEAR、DATA、DEF　FN、DIM、FOR…NEXT、GOSUB～RETURN、GOTO、IF…THEN…ELSE、IF…GOTO…ELSE、INPUT、LET、ON　ERROR　GOTO、ON…GOSUB、ON…GOTO、OUT、POKE、PRINT、READ、REM、RES、RESTORE、RESUME、SET、STOP、SWAP、TRON、TROFF、USR、WRITE

3．3　関数

　ZB3BASICでは下記の関数を使用することができます。

ABS、AND、ASC、ATN、BCD、BI＄、BIT、CHR＄、COD、COS、DEC、EXP、FIX、HEX＄、IN、INKEY＄、INPUT＄、INSTR、INT、LEFT＄、LEN、LN、LOG、MID＄、OR、PEEK、RIGHT＄、RND、SEARCH、SGN、SID、SIN、SPACE＄、SPC、SQR、STR＄、TAB、TAN、VAL、XOR

3．4　システム変数・システム定数

　一般の変数は、その値をユーザーが定義しますが、これとは逆にシステムが値を与えてユーザーはそれを利用することはできるが、ユーザーが勝手に値を与えることはできない特殊な変数、定数です。

ERL、ERR、PI、TIME＄

3．5　定数

　ZB3BASICで扱う数には、ある特定の値を示す「定数」と、値が可変な「変数」及び「配列」があります。ここではまずその「定数」について説明します。

3．5．1　十進定数

　ZB3BASICで扱う十進定数は10の37乗　～10の－37乗　の範囲にある正、負数で、有効桁数は6桁の単精度実数と16桁の倍精度実数があります。
　表現形式は①整数型、②固定小数点型、③指数型の3通りがあります。各々の例を下に示します。

①整数型

　単精度　　123456　　－365
　倍精度　　123456789012345
②固定小数点型
　単精度　　123．45　－0．00135　．12345
　倍精度　　－1234567．89012345　123．45＃
・指数型
　単精度　　0．2345E＋2　－0．235E－13
　　（E＋2は×10の2乗 のことです）
　倍精度　　－1234567．89012345D＋15　0．3D－5
　　（D＋15は×10の 15 乗のことです）
［注記］
　123．45＃のように後ろに＃をつけて表すと倍精度の数として扱われます。
　整数の場合には、例えば12345という数は単精度でも倍精度でも同じ値になりますが、実数（小数部分を持つ数）は、BASIC内部で10進数→2進数変換の過程で誤差を含む近似値になります。
　123．45は2進数に変換される時に有効数字6桁の数として扱われます。これに対して123．45＃は有効数字16桁の数として変換されるため、より真の値に近い数値となります。
　倍精度の計算を行う時にはこの違いに十分注意する必要があります。
　計算式の中で定数を表現するのに＃をつけないでおくと期待する精度が求められません。
　なお1234567．89012345のように単精度の表現を越えている場合には、自動的に倍精度数として扱われるため、＃をつける必要はありません。

3．5．2　16進定数

　ZB3BASICではマシン語との結合やI／Oポートの操作に便利なように2桁又は4桁の16進定数が扱えるようにしてあります。（2桁、4桁以外は扱えません）
　16進定数は10進数と区別するために、＄マークをつけて表します。
　16進定数は式の中で使用することもできます。

［例］
　A＝＄FF
　B＝C＊K＋＄B000

3．5．3　文字定数

　文字定数はPRINT文等で文字列を表示したり、文字変数に値を代入するのに使用します。
　文字定数は、”　”（クォーテーションマーク）ではさんで表示します。
　文字定数は1行（最大80字）の中であれば、桁数の制限はありませんが、文字変数に代入する場合には39桁を越えるとエラーになります。
　”　”（クォーテーションマーク）の中には、英大文字、英小文字、カナ、記号のどのような組合わせでも許されます。

［注記］
　クォーテーションマーク（”）をデータとして用いたい時には、CHR＄（＄22）を使います。

［使用例］
```
>10 PRINT ″ABCDE″
>20 PRINT CHR$($22);″abcde″;CHR$($22)

>RUN
ABCDE
″abcde″
```

3．6　変数

　変数にはその取りうる値によって、実数型、倍精度実数型、整数型及び文字型の区別があります。
　どの型の変数も最初の1文字がA～Zで始まる、1～5桁の英数字（英字は大文字に限る）で表された変数名を使用しますが、その後ろに倍精度実数型は＃をつけ、整数型は%をつけ、文字型は＄をつけて区別します。
　この場合、変数ABCとABC＃、ABC%、ABC＄はそれぞれ別の変数として扱われます。

メモリ内部では実数型変数のデータ部分は指数部1バイト、仮数部3バイトの4バイトを占め、倍精度実数型変数は指数部1バイト、仮数部7バイトの8バイト、整数型は2バイト、文字型は40バイトを占めます。
いずれの変数も、BASICの命令、関数、システム変数と同じ並びの名前は使用できません。
以下に変数名の正、誤例を示します。

[正しい例]
ABC　AA　ALPHA　C302　Z
ABC＃　AA＃　ALPHA＃　C302＃　Z＃
ABC%　AA%　ALPHA%　C302%　Z%
ABC＄　AA＄　ALPHA＄　C302＄　Z＄

[間違っている例]
ABCDEF……（桁数が多い）
3DATA……（1桁目が英字でない）
Alpha……（小文字は使えない）
DATA……（BASIC命令のDATAと同じつづりになっている。この場合XDATA、DATA1、DATAXなどは許される）
FNC……（FNで始まる名前はユーザー関数になるためFNC（××）の形でしか使用できない。ユーザー関数については3．9参照）

3．6．1　整数型変数

最初の1文字がA～Zで始まる、1～5桁の英数字に続けて%をつけて表します。
－32768～＋32767（16進数の＄0000～＄FFFF）の範囲の整数しか代入することはできません。
```
10 A%=123.45
```
のように右辺に実数を書いてもエラーにはなりませんが、この場合にはその値の小数部が切り捨てられて、整数値（この例では 123 ）が代入されます。
[注記]
ZB3BASICでは処理速度をあげるため、整数のみの演算は高速の整数演算ルーチンによって処理しています。整数演算ルーチンは、式の左辺に整数型の変数がある場合に限って起動されます。例えば
```
10 A%=123.45+12.67 ……・①
```
のように右辺に整数が無い場合でも整数演算が行われます。この場合には A%の値は 136 ではなくて 135 になります。つまり整数演算では全ての数の小数部が先に切り捨てられて、整数のみにしたあとで計算が行われます。上の例では A%=123+12 の計算が行われます。
これに対しFIX関数を使用して
```
10 A%=FIX(123.45+12.67)　……・②
```
のように書いた場合には A%の値は 136 になります。
なお①のように、整数型変数の代入文の右辺に、実数型の定数を書くことは許されますが、実数型の関数（SINやSQR等）を書くことはできません。その必要があるときは②のFIX関数の（ ）の中に記述します。

3．6．2　実数型変数

最初の1文字がA～Zで始まる、1～5桁の英数字で表します。
この型の変数には、整数でも実数でも代入できます。整数型変数では－32768～＋32767の範囲外の整数を代入することはできませんが、この実数型変数はそれよりも大きな整数を扱うこともできます。
例えば、
```
10 A=999999
20 B=123456E+30
```
などの記述ができます。したがって特に処理速度を要求しない場合には、整数型変数を使用するよりも実数型変数を用いた方が、制限が少ないだけプログラムが組み易くなります。

3．6．3　倍精度実数型変数

最初の1文字がA～Zで始まる、1～5桁の英数字に続けて＃をつけて表します。
この型の変数は、有効数字が16桁の数が扱えるため、精度を必要とする計算や大きな整数値を扱う時に使用します。
ZB3BASICでは通常の実数計算ルーチン（浮動小数計算ルーチン）と倍精度実数計算ルーチン（倍精度浮動小数計算ルーチン）とが用意されていて、どちらが選択されるかは次の条件によって決定されます。
①代入文の左辺が倍精度型の変数、配列のとき、右辺の計算は倍精度で行われる。

②代入文の左辺が単精度型の変数、配列のとき、右辺の計算は単精度で行われる。
③PRINT文での計算式では、式の第一項の型により決定する。第一項が関数の場合には、その（ ）内の式の第一項の型により決定する。

①で右辺の計算式の中に単精度の値が含まれていると、結果的には単精度並の精度しか求められません。下の例を比べて下さい。
[例1]10 A#=123.45＊10
　　20 PRINT A#
（実行結果）
　　1234.499969482422
[例2]10 A#=123.45#＊10
　　20 PRINT A#
（実行結果）
　　1234.5

関数の計算も同じルールで行われます。
　　10 A#=SID(25)+COD(47)
上の文のように左辺が倍精度型ならば、右辺の関数も倍精度型で計算されますから、この例の場合には、10 A#=SID(25#)+COD(47#) にする必要はありませんありません（整数値には誤差はない）。しかし下の例では同じように倍精度で計算されますが、（ ）の中の値が単精度で誤差が含まれているため期待する精度は求まりません。
　　10 A#=SQR(12.456)
この場合には、10 A#=SQR(12.456#)にする必要があります。

②で右辺の計算式に倍精度の値が含まれていると、その値を単精度に直してから計算されます。
③の意味は次の例で理解して下さい。
　10 PRINT 10.5+1.23456789012
　20 PRINT 1.23456789012+10.5
上の文では式の第一項が単精度なので、単精度で計算が行われ、下の文では第一項が倍精度なので、倍精度で計算が行われます。実行して結果を比べてみて下さい。
　　30 PRINT 1－SIN(0.123#)
という文で結果を倍精度で出したいときは下のように第一項に倍精度の数を置くようにします。
　　30 PRINT 1#－SIN(0.123#)
または
　　30 PRINT　－SIN(0.123#)+1
または式の先頭にダミーの 0# を入れて
　　30 PRINT 0#+1－SIN(0.123#) のようにします。

３．６．４　文字型変数

最初の１文字がA～Zで始まる、１～５桁の英数字に続けて＄をつけて表します。
各変数に代入できる文字列の長さは最大３９桁です。
文字型変数に値を代入する場合の右辺は、文字型定数、文字型変数、文字型配列または文字型関数でなければエラーになります。
　10 A$=″abcABC″
　20 B$=A$
　30 C$=CHR$($41)
などの書き方をします。また文字型定数、文字型変数、文字型配列、文字型関数を＋でつないで、
　40 D$=A$+″XYZ″+C$
のように使うこともできます。この場合に左辺の変数には、右辺の文字列を順につないだものが、代入されます。右辺の文字列の桁数の合計が３９を越えるとエラーになります。

３．７　配列

配列も変数と同じように、整数型、実数型、倍精度実数型および文字型の４通りがあります。名前も変数と同じように１～５桁の英数字（及びそのあとに%、＃、＄をつける）でその後ろに（ ）をつけて要素を特定します。
最大３次まで使用でき、（ ）内の数値（添字）は０及び正の整数で、変数、式も使うことができます。ただし配列または配列を含む式を添字として使うことはできません。
配列はDIM文により使用する範囲を定義しておく必要があります。

ABC(1, 2)とABC%(7)というように、同じ文字の並びでも後ろに%や＄がついていれば、別の配列として区別されるため、使用上は問題ありませんが、配列と変数で同じ名前を使用することはできません。
例えば数値変数でABCを使っていて、数値配列でABC(1, 2)のように使うことはできません(数値変数でABCを使っていて、文字配列でABC＄(1, 2)のように使うことは問題ありません)。
DIM文で定義する配列の数や添字の大きさに制限はありませんが、TEXTエリアの範囲で、プログラムが占めるメモリの残りに割りつけられるため、余り大きな配列を指定すると、メモリオーバーになってしまいます。その場合はエラーメッセージが出されます。
配列に対する演算規則は変数の場合と全く同じです。

[整数型配列の例]
ABC%(1, 2)　AA%(A, B, C)　Z%(ALPHA, BETA)
[実数型配列の例]
ABC(1, 2)　AA(A, B, C)　Z(ALPHA, BETA)
[倍精度実数型配列の例]
ABC#(1, 2)　AA#(A, B, C)　Z#(ALPHA, BETA)
[文字型配列の例]
ABC＄(1, 2)　AA＄(A, B, C)　Z＄(ALPHA, BETA)

3．8　式

式には算術式と関係式及び論理式があり、算術式は一般の数値演算に用いられ、関係式、論理式は主にIF文の中で用いられます。

3．8．1　算術式

算術式は数値定数、数値変数、数値配列及び数値関数を算術演算子及び( )カッコでつないだもので、演算の優先順位は一般の数学における計算と同じですが、代数式のように乗算記号を省略することはできません。
算術演算子を優先順位の高いものから順に並べると次のようになります。

①^(ベキ乗)②＊(乗算)及び／(除算)③＋(加算)及びー(減算)

( )がある場合は( )内の計算が優先されます。( )の多重使用に制限はありません。
なお、特に除算を含む計算式の場合は表現を考える必要があります。次の例を参考にして下さい。

(代数表現)　$\frac{K}{A(B+C)} \times (C+D)$

(算術式表現)　K／(A＊(B＋C))＊(C＋D)

これを下のように書くと正しい結果は得られません。注意して下さい。

K／A＊(B＋C)＊(C＋D)

なお文字型の定数、変数、配列、関数に対しても＋記号のみを使用してつなぐことができます(「3．6．4　文字型変数」参照)。

3．8．2　関係式

関係式は2つの要素(変数、定数、配列、関数またはこれらを組合わせた式)を関係演算子でつないだもので、ふつうはIF文の中で用いられ、2つの値の比較の結果により真又は偽の値をとります。真のときは1、偽のときは0の値をもちます。

関係演算子	使用例	意味、その値
＞	A＞B	A＞Bのとき真(1)、それ以外偽(0)
＜	A＜B	A＜B 〃 〃 〃 〃
＞＝	A＞＝B	A≧B 〃 〃 〃 〃
＜＝	A＜＝B	A≦B 〃 〃 〃 〃
＝	A＝B	A＝B 〃 〃 〃 〃

＜＞　　A＜＞B　A≠B　〃　　〃　　〃　　〃

[使用例]
10 IF A>0 GOTO 30
20 IF ABC＜=XYZ+100 THEN PRINT ″ABC=OK″:GOTO 30

文字型の変数、定数、配列、関数などの比較も同じようにできます。
文字型の場合には桁数が大きい方が大となり、同じ桁数なら始めの桁からひと桁ずつ各桁の文字のキャラクタコードを比較していって、はじめに大きいコードが出てきた方が大になります（キャラクタコードについては23章を参照して下さい）。
全く同じ文字の並びで桁数も同じ場合には、その値は等しいことになります。

3．8．3　論理式

論理式は関係式を論理演算子＊（論理積）及び＋（論理和）で結ぶことによって表わされ、真（1）又は偽（0）の値をもちます。主としてIF文の中で使用されます。
各々の関係式は（　）で囲う必要があります。
[使用例]
10 IF (A＞=0)＊(B=C) GOTO 300

上例の意味は「A≧0で同時にB＝Cならば行番号300へジャンプせよ」です。
上例で＊のかわりに＋（論理和）を用いると意味は「A≧0か又はB＝Cのとき行番号300へジャンプせよ」になります。

3．9　ユーザー関数

FNで始まる最大5桁の英数字名は、ユーザー関数として使用されます。
ユーザー関数とはその名の通りユーザーが自由に定義できる関数のことです。
変数が異なるだけで計算式は同じという処理をプログラムのあちこちで使用する場合にユーザー関数を使うとプログラムがすっきりとまとまります。
ユーザー関数は、DEF　FN文で定義します。下に使用例を示します。

[使用例]
```
10 DEF FNX(A, B)=A＊B+10．5
　　:
50 C=Z－FNX(X, Y)
　　:
80 K=FNX(L, M)+S＊3
```

行番号50のX、Yの値を行番号10の右辺の式のA、Bにあてはめて計算した結果がFNX（X, Y）の値になります。同様に行番号80のL、Mの値を行番号10の右辺の式のA、Bにあてはめて計算した結果がFNX（L, M）の値になります。
DEF　FN文の（　）内に記述した変数（1個以上複数個記述できる）は等号の右側の数式を表すために仮に使用するだけなので、他のプログラム部分でその変数（この例ではA及びB）を使用していても、互いに影響は与えません。
DEF　FN文は幾つでも定義でき、またプログラムの途中に置くことも出来ますが、その関数が引用される以前に実行されている必要があります。
ユーザー関数名はFNで始まる最大5桁の英数字でなければいけません。
整数型の関数にするには普通の変数名と同じように、名前の最後に%をつけます。倍精度型の関数にするには最後に＃をつけます。
文字型は扱うことができません。
（　）内の変数名はDEF　FN文と引用される文とで型が一致している必要はありませんが、計算の型はその（　）内の変数の型ではなくて、関数の型に支配されます。
使用例で、FNX%(A, B) にした場合には、FIX(A＊B+10．5) ではなくて、FIX(A)＊FIX(B)+FIX(10．5) が計算されます。

3．10　ラベル名

ZB3BASICでは行位置を示すマークとしてラベル名を使用することができます。ラベル名は先頭に＊マークをつけた最大5桁の英数字（＊を含めて6桁）で表します。

GOTO文、GOSUB文の行番号の代わりにラベル名を書くことによって、そのラベルの置かれている行へジャンプすることができます。

[使用例]

```
    10 INPUT B,C,SW
    20 ON SW GOTO *TASU,*HIKU,*KAKE
    30 STOP
    40 *TASU:A=B+C:PRINT A:GOTO 10
    50 *HIKU:A=B-C:PRINT A:GOTO 10
    60 *KAKE:A=B*C:PRINT A:GOTO 10
>r.
B?1
C?2
SW?1
3
B?2
C?5
SW?2
-3
B?15.7
C?3.6
SW?3
56.52
```

ラベル名は行番号のすぐ次の位置、つまり行の一番先頭に書かなければいけません。
10 A=1:＊ABC:N=N+1 というように行の途中に置くことは許されません。
上の例ではマルチステートメントになっていますが次のようにラベル名だけを書くこともできます。

```
    30 ＊TASU
    35   A=B+C:PRINT A:STOP
```

変数名とラベル名の＊から後ろの部分とが同じつづりであっても構いません。

4. 扱うことのできる数値の範囲

このBASICで使用できる数には定数、変数及び配列があります。これらは主にLET文の中などで組合わせて加減乗除等の計算を行います。
数には－∞～＋∞の範囲がありますが、BASICで扱うことのできる数には一定の範囲があります。範囲外の数値を入力したり、計算の結果が範囲外になったりすると、エラーコードが表示されます。
整数型の計算（左辺が整数型変数か整数型配列であるような式の計算）では－32768～＋32767の範囲の整数しか扱うことができません。
それ以外の実数型計算で使用できる数値は10の－37 乗 ～10の+37 乗の範囲の正、及び負数です。

5. 計算の精度

計算の精度は、単精度の場合、有効数字6桁～7桁で計算の内容によって多少の幅があります。しかし計算結果の表示は6桁に統一されています。
倍精度の場合は有効数字16桁で計算されます。

6. 計算の誤差

整数型の演算の場合、有効数字の範囲内ならば、加減算を繰り返してもその値に誤差は含まれてきませんが、実数型の演算では誤差が累積されてくる場合があります。
実数型の演算は、十進数を2進の浮動小数に変換して行っているため、その変換の際の誤差が原因なのですが、2進演算の宿命でどうすることもできません。
例えば下のプログラムを実行した場合の結果は、システムのエラーではなくて、誤差のせいであることを理解して下さい。

```
＞10 FOR    A=0. 1    TO    0. 9    STEP    0. .1
＞20 PRINT    A,
＞30 NEXT    A

＞RUN
0. 1              0. 2              0. 3              0. 4              0. 5              0. 6
0. 7              0. 8
```

もし誤差がなければ最後は 0. 9 で終わるはずですが……。

7. 予約語

「3. 6 変数」のところでも説明しましたが、BASICプログラムの変数名や配列名には、BASIC命令や関数名と同じ文字列で始まる名前を使用することはできません。 変数名として使用できない文字の並びを「予約語」といいます。

ZB3BASICでは、予約語はBASIC命令（5章）、BASIC関数・システム変数（6章）のみで、コマンド類はプログラム中で変数名として使用可能です（例えばRUN、CONT、HELPなどは変数名として使用してもよい）。

以下にZB3BASICの予約語を示します。プログラムの実行中に、全く関係ないと思われるようなエラーコードが表示された場合には、その行の中で以下の予約語と同じ変数名を使用していないか、確認してみて下さい。

ABS、AND、ASC、ATN、BCD、BEEP、BI＄、BIT、CHR＄、CLEAR、CLOSE、CLS、COD、COS、CSRLIN、CURSOR、DATA、DATE＄、DEC、DEF FN、DIM、ELSE、EOF、ERL、ERR、EXP、FIELD、FIX、FN、FOR、GET、GOSUB、GOTO、HEX＄、IF、IN、INKEY＄、INPUT、INPUT＄、INSTR、INT、INTON、INTOFF、LEFT＄、LEN、LET、LN、LOCATE、LOG、LPRINT、MID＄、NEXT、ON ERROR GOTO、ON INT GOSUB、ON GOSUB、ON GOTO、OPEN、OR、OUT、PEEK、PI、POKE、PRINT、PUT、READ、REM、RES、RESTORE、RESUME、RETURN、RIGHT＄、RND、SEARCH、SET、SGN、SID、SIN、SPACE＄、SPC、SQR、STEP、STOP、STR＄、SWAP、TAB、TAN、THEN、TIME＄、TO、TRON、TROFF、USR、VAL、WRITE、XOR

予約語のなかには以前のZB11BASICで使用できた命令や関数が含まれていますが、そのうち下記のものはこのZB3BASICでは使用できません。命令、関数として使用するとプログラムが正しく実行されなくなる可能性がありますから注意してください。

BEEP、CLOSE、CSRLIN、CURSOR、LOCATE、FIELD、GET、LOCATE、OPEN、PUT

　コマンドはコマンドモードでのみ実行可能なもので、プログラム内部で使用することはできません。プログラム中で使用するとエラーコードが表示されます。
　ここではBASICプログラムを作成したり実行したりするときに使用するコマンドをまとめて説明します。

AUTO　[省略形]AU. AUT.

[書式]AUTO　n, d

　プログラム作成時に行番号を自動表示します。
　nは1～32767の範囲の整数で、この数から自動表示されます。dは増分で1～32767の範囲の整数で示します。dを省略すると増分は10になります。

[使用例]
＞AUTO　100, 5　[Enter]
＞　　100_　　……行番号を表示してカーソルが表示されるので文を入力します。

＞　　100XYZ＝123[Enter]　……[Enter]を入力すると
＞　　105_　　……次の行番号が表示されます。続けて次の行の文を入力します。

　このように[Enter]を入力するとその行をメモリに格納するとともに次の行番号を表示するため、毎回行番号を入力する手間が省けます。
　行番号が表示される他は通常の入力モードと同じです。したがって［↑］キーなどを使って入力済の行を修正することもできますが、修正作業後に[Enter]を押すと、次の行番号表示が一つ飛んだ値になります（行番号が飛ぶことさえ気にしなければ、特に支障はありません）。例えば上の例で行番号105が表示されているときに［↑］キーで上の100行に戻って内容を修正してみます。

＞　　100ABC＝123[Enter]
＞　　110_
　下線部を書き直して[Enter]を入力すると次の行番号105はすでに表示済なので、その次の110が[Enter]を入力した下の行位置に表示されます。
　AUTO機能を打ち切って通常の入力に戻るには[Ctrl]Bを入力するか、次の行番号が表示されたときに何も入力しないで[Enter]キーを押します。

[注意]
　すでに作成済のプログラムの修正、追加作業でAUTO機能を使用するときは、行番号が重ならないように注意して下さい。AUTOによって表示された行番号がすでに存在する場合、そのまま新しい文を入力して[Enter]を押すと、同じ行番号の古い文は消されて新しい文に書き換えられてしまいます。

CONT　[省略形]CO. CON.

　ブレイクしたポイントからプログラムの実行を再開します。
　STOP文の実行や、エラー発生により、または[Ctrl]B　入力により、プログラムの実行は中止（ブレイク）されますが、その後でこのCONTコマンドを入力すると、さきほどブレイクしたポイントから再びプログラムの実行を開始します（INPUT文の途中でブレイクした場合には、そのINPUT文の始めから再実行されます）。
　ブレイク後にPRINT文をダイレクト実行して変数の値を参照したり、LET文をダイレクト実行して変数の値を変更したあとで、CONTコマンドを実行することもできます。

[注意1]
　ブレイク後にプログラムの一部を書き換えた後に、CONTコマンドを入力すると、プログラムの実行は正しく行われません（LISTコマンドでプログラムを表示させることは構いません）。
[注意2]
　マシン語サブルーチンで割込みを行う設定になっているときは、ブレイク後にCONTで再実行しても割込みは禁止されたままになります。

DELETE　[省略形]DE. DEL. DELE. DELET.

プログラムの一括削除を行います。

[書式①]DELETE　L1－L2
[書式②]DELETE　－L2
[書式③]DELETE　L1－

L1は削除を開始する行番号、L2は削除を終了する行番号です。
[書式①]ではL1～L2の行番号の行がまとめて削除されます。
もしL1またはL2に該当する行番号が存在しなければ、L1より大きくてL2より小さい範囲の行番号の行が削除されます。
[書式②]のようにL1を省略した場合は、始めからL2までの行が削除されます。
[書式③]のようにL2を省略した場合はL1から最後までの行が削除されます削除されます。

[注意1]
L1－L2を省略してDELETEのみを入力すると、プログラム全体が削除されてしまいます。
[注意2]
DELETEコマンドや、行番号のみの入力により削除したプログラムは、HELPコマンドによっても復活させることはできません。操作には充分注意して下さい。

[使用例]
DELETE　100－230　……行番号100から230まで削除
DELETE　－320　……はじめから行番号320まで削除
DELETE　560－　……行番号560から最後まで削除

HELP　[省略形]H. HE. HEL.

プログラムを復活させます。
NEWコマンドを入力したり、[Ctrl]E、[Ctrl]CでZB3BASICを終了したのち再びZB3BASICにエントリすると、BASICプログラムは消えてしまいます。
そのような場合にHELPコマンドを入力すると、BASICプログラムが元通りに復活します。
ただしマシン語のプログラムが暴走してメモリ内容を破壊してしまったときなどは、このHELPコマンドを入力してもBASICのプログラムはもとに戻らないことがあります。
HELPコマンドを使ってメモリの使用状況を調べることもできます。
BASICプログラムが長くなってしまった場合やたくさんの変数を使用する場合にはプログラムサイズだけではなくて、変数領域がどの位とられているかを確認する必要があります。
HELPコマンドを使うと、プログラム領域（TEXTエリア）と変数データ領域を表示します（BROMコマンドもTEXTエリアを表示しますが変数領域は表示しません）。

[注意]
変数名を全く使用しないか、またはA%～Z%、Ａ$～Ｈ$のみを使用するプログラムを作成した場合には、HELPコマンドを使うと、それ以後はプログラムの作成や実行が正しく行われなくなりますから、そのような場合には、HELPコマンドは使わないように注意して下さい（誤って使用してしまったときは[Ctrl]CでZB3BASICを終了したのち再びZB3BASICにエントリして下さい）。
A%～Z%、Ａ$～Ｈ$のみを使用するプログラムでも、ダミー命令としてプログラムの先頭に、10　A＝0　などという行を書いておけば、万一の場合にHELPコマンドを使用することができます。

[使用例]
HELP[Enter]
TEXT　8004－8083
ヘンスウ　DF80－DFFF

LIST　[省略形]．(ピリオドだけでもよい!!)L. LI. LIS.

[書式①]LIST　L1－L2
[書式②]LIST　－L2
[書式③]LIST　L1－
[書式④]LIST
[書式⑤]LIST　L1

L1、L2は1～32767の整数で行番号を示します。
［書式①］は行番号L1～行番号L2までの範囲にあるプログラムの内容を行番号の小さい順に出力（ディスプレイ画面に表示）します。
L1、L2の値に、そのプログラムに存在しない行番号を指定すると、L1より大きくてL2より小さい行番号の行が順に出力されます。
［書式②］のようにL1を省略するとプログラムの始めからL2までが出力されます。
［書式③］のようにL2を省略するとL1からプログラムの最後までが出力されます。
［書式④］のようにL1－L2を省略すると、プログラム全部が出力されます。
［書式⑤］のようにー（ハイフン）をつけないでL1だけを指定すると、そのL1の1行だけが出力されます。（この時存在しない行番号を指定すると、なにも出力されませんが、特にエラーメッセージは出ません）

［使用例］
＞LIST　30－120［Enter］　……行番号30（なければその次の行番号）から行番号120（なければその前の行番号）までが出力される
＞．30－120［Enter］　……省略形を使う。上と同じ結果になる
↑
ピリオド
＞LIST　－300［Enter］　……プログラムの始めから行番号300（なければその前の行番号）までが出力される
＞LIST　70－［Enter］　……行番号70（なければその次の行番号）からプログラムの最後までが出力される
＞LIST　100［Enter］　……行番号100の行だけが出力される。行番号100が存在しなければ何も出力されない
＞LIST［Enter］　……プログラム全部が出力される

プログラムリスト出力の中止

1画面に表示しきれない時はどんどんスクロールされて（画面の下に新しい行が追加表示されて、それとともに一番上の行が画面の外に押し出されて消えて）行きます。
適当なところでリスト出力を打ち切りたいときは、［Ctrl］を押しながらBを押して下さい。リスト出力が中止されます。

／LOAD　［省略形］なし

［書式］
／LOAD　ファイルネーム，aaaa

テキスト形式で保存されたファイルをBASICプログラムとしてND80ZⅢのRAMエリアに読み込みます。／がついたコマンドでは省略形は使えません。
ファイルネームがファイル名のみの場合にはZB3BASICが存在するフォルダ（通常はnd80z3フォルダ）からLOADします。ファイルがみつからないときは　FILE　NOT　FOUND　と表示されます。ファイル名にドライブ名やフォルダ名を含めて記述することで、別のドライブやフォルダにあるファイルをLOADすることができます（使用例③）。
aaaaは4桁の16進数で、省略することもできます。省略した場合には現在のテキストエリアの先頭からLOADされます。aaaaをつけるとaaaa番地からLOADされます。
／SAVEでファイルを作成したときのテキストエリアのアドレス情報は保存されません。／LOAD時に新しく決定されます。
／LOADコマンドでテキスト形式のファイルが読み込まれるとき、BASICシステムによって1行ずつBASIC内部コードに変換してRAMに格納していきます。このときプログラムは1行ずつ画面にリスト表示されます。内部コードに変換するときに文法エラーが見つかるとそこでエラーコードが表示されLOAD作業は打ち切られます。
メモ帳などのテキストエディタで作成したプログラムも文法的に正しければ／SAVEで保存されたファイルと同じようにLOADすることができます。

［注意1］
LOADコマンドの動作は、NEW　＋　／LOAD、またはNEW　aaaa　＋　／LOAD、というように必ずNEWコマンドの動作を伴っています（LOAD前にRAMに存在したプログラムは失われます）。
なおこのことはBASICテキストファイルについてのみあてはまります。バイナリファイル（マシン語プログラム、データファイル）を／LDコマンドでLOADしてもテキストエリアには影響を与えません。
［注意2］

メモ帳（Notepad）は問題ありませんが、WriteやWordでは通常はそれぞれの形式でファイルが作成されます。保存するときにテキスト形式を指定してもゴミが混じる場合があります。Writeなどで作成したファイルがうまくLOADできないときは、一度メモ帳（Notepad）で開いてゴミを削除して、メモ帳で保存してからLOADしてみてください。

［使用例①］
＞／LOAD　TEST. TXT［Enter］
［使用例②］
＞／LOAD　MIHON［Enter］　……テキスト形式のファイルなら拡張子のついていないファイルでもよい
［使用例③］
＞／LOAD　D：￥BASIC￥TESTPRO. TXT, 9000［Enter］
この例のように別のドライブや別のフォルダにあるファイルをLOADすることもできます。上の①②例ではプログラムは現在のテキストエリアの先頭からLOADされます。③では9000番地からLOADされます。

［注意1］
ファイル名は名前部分が半角英数8文字以内で拡張子は3文字以内に限られます。テキストファイルであってもそれ以外の名前のファイルはLOADできません。
［注意2］
テキスト形式（拡張子の種類に関わらず内容がテキスト形式になっている）以外のファイルをLOADするとZB3BASICシステムが暴走してハングアップすることがあります。

NEW ［省略形］N. NE.

［書式］NEW　aaaa

ユーザーズエリア（TEXTエリア）内のユーザープログラムを消去します。aaaaは4桁の16進数です。
16進アドレス（aaaa）を指定すると、ユーザープログラムを消去すると同時にTEXTエリアの先頭アドレスをaaaaにセットします。これ以後はプログラムを書くとaaaa番地からそのプログラムが格納されます。
aaaaを省略すると、ユーザープログラムは消去されますが、TEXTエリアの先頭アドレスは変更されません。
ZB3BASICエントリ後は、NEW　8004が自動的に実行されます。（TEXTエリアの先頭アドレスは＄8004になります）
ここではプログラムを消去する、と説明しましたが正確には「消去」するのではなくてTEXTエリアの中に書かれたプログラムの始めと終わりのアドレスを管理しているレジスタをクリアするだけで、プログラムそのものは、まだ消えずにそのまま残っています。
HELPコマンドを実行すると、このレジスタの値を元に戻すため、プログラムが復活することになります。

［使用例］
＞NEW［Enter］　……テキストエリアクリア。エリアの先頭アドレスは変更されない
＞NEW　A000［Enter］ ……テキストエリアクリア。エリアの先頭アドレスは＄A000になる

REN ［省略形］RE.

［書式］REN　n1, n2, n3

プログラムの行番号を整理して新しく付け直します。行のはじめにある行番号だけではなくGOTO、GOSUB、ON ERROR　GOTO、ON　INT　GOSUB、ON　n　GOSUB、ON　n　GOTO、RESTORE、RESUMEの行番号部分も正しく更新されます。
n1は新しくつける先頭の行番号で省略すると10が設定されます。
n2は旧行番号を指定します。この行番号から後ろを新しい行番号に書き換えます。省略するとプログラムの先頭行が指定されます。
n3は新しい行番号の増分で、省略すると10が設定されます。
前または途中にあるパラメータを省略するときは、区切りのカンマ（, ）は省略できません。
後にあるパラメータを省略するときは、その後ろのカンマ（, ）は省略できます。

［使用例］
REN［Enter］　先頭行番号を 10 にして、その後 20、30、……と置き換える
REN 1000, 520, 5［Enter］　行番号 520 を 1000 にしてその後 1005、1010、……と置き換える
REN 1000, 520［Enter］　上と同じことを 1000 の後 1010、1020、……とする
REN, , 20［Enter］ 先頭行番号を 10 にして、その後 30、50、……と置き換える

[注意]
作業用エリアとしてBASICプログラムの後のRAM領域を使用するため、RENコマンドの実行によって、それ以前にBASICプログラムを実行した結果の変数の値が破壊されることがあります。またBASICプログラムより後のアドレスにマシン語プログラムやデータがある場合には、それらが破壊されることがあります。

RUN [省略形]R. RU.

BASICプログラムを実行するためのコマンドです。
RUN[Enter]とキーボードから入力すると、一番先頭の行から実行が開始されます。
RUN 100[Enter]のように後ろに行番号をつけると、その行から実行を開始します。

／SAVE [省略形]なし

[書式]／SAVE ファイルネーム

BASICプログラムをテキスト形式でSAVEします。
ファイルネームがファイル名のみの場合にはZB3BASICが存在するフォルダ（通常はnd80z3フォルダ）にSAVEされます。同じファイル名があると上書きされます。
ファイル名にドライブ名やフォルダ名を含めて記述することで、別のドライブやフォルダにSAVEすることができます（使用例③）。
現在のテキストエリアのアドレス情報は保存されません。／LOAD時に新しく決定されます。
／SAVEコマンドではBASICシステムによって1行ずつBASIC内部コードをテキスト形式に変換していきます。このときプログラムは1行ずつ画面にリスト表示されます。

[使用例①]
＞／SAVE TEST. TXT[Enter]
[使用例②]
＞／SAVE MIHON[Enter] ……拡張子はなくてもよい。
[使用例③]
＞／SAVE D:￥BASIC￥TESTPRO. TXT[Enter]
[注記1]
使用例③のように別のドライブや別のフォルダにSAVEすることもできます（上の①②例ではZB3BASICの存在するフォルダにSAVEされます）。
[注記2]
使用例③のようにTXT以外の拡張子をつけて保存してもZB3BASICでの作業には支障ありません。Windowsではディレクトリ表示をしたときにテキストファイルのアイコンがつかないため、整理がしにくいかもしれないことと、アプリケーションから開くことしかできない欠点があります。

SL [省略形]なし

プログラム中から、指定する文字列を含む行を捜し出して、表示します。

[書式]SL n1

n1はプログラム中から捜し出したい文字列で、空白や記号を含む文字列でも構いません。
XLのパラメータではカンマ（ , ）を含むことはできませんが、SLではカンマを含む文字列を指定することもできます。
SLコマンドはXLコマンドとは異なり、表示を行うだけでプログラムそのものには何の影響も与えません。

[使用例]
＞SL ABC[Enter] ……ABCという文字列を含む行を全て表示する

[注意]プログラム中の数値や、GOTO、GOSUBのオペランドとしての行番号はサーチできますが、行のはじめに書いてある行番号はサーチできません。
10 GOTO 100
↑ ↑
サーチ不可 サーチ可

[注記]文字定数” ”の中だけはサーチできません。”を含めてサーチしてください。
(例)
30 PRINT ”ABC”

SL AB …検出できない
SL ”AB
30 PRINT ”ABC”

XL [省略形]X.

プログラム中の文字列を別の文字列で置き換えます。

[書式]XL n1,n2

n1はプログラム中で置き換えの対象にする文字列で、n2は新しい文字列です。
XLコマンドを実行すると、プログラムを1行ずつチェックして行き、n1と同じ並びの文字列をみつけると、その文字列をn2の文字列で置き換えます。
この場合にn1とn2の文字数が一致している必要はありません。
また必ずしも意味のある単位で区切った文字列である必要はありません(たとえば命令や変数名の一部分を指定しても構いません)。
文字列の中に記号や空白を含めることもできますが、カンマ(,)を含めることはできません。

[使用例]
XL ABC, XYZ[Enter] ……プログラム中のABCという文字列をXYZで置き換えます。

[注意1]プログラム中の数値や、GOTO、GOSUBのオペランドとしての行番号は置き換えできますが、行のはじめに書いてある行番号の置き換えはできません。
10 GOTO 100
↑ ↑
置き換え不可 置き換え可

[注意2]変数名の一部を変更したり、1～2桁の短い文字列の置き換えの場合には、意外なところに同じ文字列が使用されていて、それが全部置き換えられてしまうため、プログラムエラーが作り出されてしまうことがあります。
たとえば変数名XをABCに変更するつもりで、XL X, ABC[Enter]と入力すると変数名のXだけではなくて、NEXT文までがNEABCTに置き換えられてしまいます。
いきなりXLコマンドを使用しないで、SLコマンドで確認してから、XLを実行するようにして下さい。

[注記]文字定数” ”の中だけは置き換えできません。”を含めて指定してください。
(例)
30 PRINT ”ABC”

XL AB, XY …置き換えできない

XL ”AB, ”XY
30 PRINT ”ABC”

命令（ステートメント）はBASICプログラム中でそれぞれの動作を指定するのに使われます。
この命令と次章で説明する関数が、BASICの主要な構成要素です。
したがってこの命令と関数の働きを理解すれば、BASICプログラムは自由に使いこなすことができます。
これらの命令はコマンドモードでダイレクトに実行することもできます。

CLEAR ［省略形］CLE. CLEA.

全ての変数を初期化します。数値変数には0が代入され、文字変数には1桁の空白が代入されます。

［使用例］
10 CLEAR

CLS ［省略形］CL.（CL と . です）

表示画面をクリアしてカーソルを画面のトップに移動します。

［使用例］
10 CLS

DATA ［省略形］D. DA. DAT.

READ文で読み込まれる数値、文字定数を定義します。
READ文より前にあっても、後ろにあっても構いません。また幾つDATA文を書いても構いません。READ文は小さい行番号のDATA文から順に読んで行きます。
整数、実数、倍精度実数、文字定数をカンマ（ , ）で区切って複数個記述できますが、その型がREAD文の変数の型と一致しない場合は実行時にエラーになります（整数を実数型変数に読み込むことはできますが、実数を整数型変数に読み込むことはできません。また数値を文字型変数に読み込むことはできますが、文字を整数、実数型変数に読み込むことはできません）。
文字定数は” ”でくくらないでそのまま表記します。（” ”を使うと” ”も文字列の一部とみなされます）

［使用例］
```
10  DATA  123, ABCDE, 10. 5
        :
50  READ  X%, Y$
        :
90  READ  Z
100  PRINT  X%, Y$, Z

＞RUN
123          ABCDE       10. 5
```

［注意］DATA文はマルチステートメント記述（複数の命令を：で区切って同じ行に続けて書くこと）はできません。ただしその行にラベルをつけて下のように記述することは許されます。

［許される使用例］
```
100  ＊ABC：DATA  123, ABCDE, 10. 5
```

DEF　FN ［省略形］なし

ユーザー関数を定義します。ユーザー関数名はFNで始まる最大5桁の英数字（整数型はその後に%をつける）で示します。
文字型は使用できません。

［使用例］
```
10  DEF  FNX（A, B）＝A＊B＋10. 5
        :
```

50 C＝Z－FNX(X, Y)

DEF FN文の( )内に記述した変数(1個以上複数個記述できる)は等号の右側の数式を表すために仮に使用するだけなので、他のプログラム部分でその変数(この例ではA及びB)を使用していても、互いに影響は与えません。
DEF FN文は幾つでも定義でき、またプログラムの途中に置くことも出来ますが、その関数が引用される以前に実行されている必要があります。
( )内の変数名はDEF FN文と引用される文とで型が一致している必要はありませんが、計算の型はその( )内の変数の型ではなくて、関数の型に支配されます。
使用例で、FNX%(A, B) にした場合には、FIX(A＊B+10. 5) ではなくて、FIX(A)＊FIX(B)+FIX(10. 5) が計算されます。

DIM [省略形]DI.

配列変数を定義します。
ZB3BASICでは3次までの数値及び文字変数の配列を使うことができます。
普通の変数はLET文で値を入れるときに変数名の定義も同時に行ってしまうのですが配列はまずDIM文で配列名とその配列の大きさを定義してからでないと使うことができません。
ただしDIM文では名前と大きさ(1次～3次の別、要素の数)を定義するだけで個々の配列要素には初期値が入るわけではありません。各配列要素にはDIM文で定義したあとで、LET文で値をセットします(プログラムの中でCLEAR文を実行した場合には全ての配列要素に0またはスペースが代入されます)。
DIM文はプログラム中のどこにあっても構いません。しかしプログラムの途中に置くのはプログラムミスのもとにもなりますから、できるだけプログラムの始めの部分に書いておく方がよいでしょう。
一つのDIM文の中で複数の配列名を定義することもできますし、一つのプログラムの中で何回DIM文を使用しても構いません。
しかし配列はテキストエリアの後ろから割りつけられていき、前からはプログラムが入るので、あまり大きな配列を定義するとメモリ不足になって実行できなくなることもあります。
配列がメモリに割りつけられる時の使用バイト数の目安は、整数型は1要素当たり2バイト、実数型は4バイト、倍精度実数型は8バイト、文字型では40バイトです。

[使用例]
```
10 DIM AKA(3,5,2),KI%(6,3),SIRO$(7)
```

上の文によって実数型配列AKA(a, b, c)(a＝0～3、b＝0～5、c＝0～2)整数型配列KI%(d, e)(d＝0～6、e＝0～3)そして文字型配列SIRO＄(f)(f＝0～7)が定義されます。
DIM文で定義しない配列名を他の文の中で使用したり、あるいは添字(カッコ内の要素を示す数値)の値としてDIM文で定義した数よりも大きい数を用いたりするとエラーコードが出されます。

[注意1]普通の変数名と配列名とで、同じ名前を使用してはいけません。またFNで始まる名前を配列名とすることはできません。
[注意2]DIM命令はダイレクトモードでは使用できません。
プログラム作成時のDIM文にエラーがあった時は、他の文の場合と別の動作になります。
プログラム作成時点でDIM文以外の文でエラーが表示された場合には、その行は登録されません。
例えば、
10 PRINT ABCXYZ[Enter] と入力すると、ERR:1 が表示されてこの行番号10は登録されません。LISTコマンドで確認すると、この行が表示されないことがわかります。
これに対し、DIM文ではエラーの発生した部分以後は切り捨てられますが、それ以前の部分は登録されます。
例えば、
10 DIM A(10), B(10, K), C(10)[Enter] と入力すると下線部に誤りがあるため、ERR:4 が表示されますが、この後LISTコマンドで確認してみると、
10 DIM A(10) と表示されます。

FOR～NEXT [省略形]F. FO. N. NE. NEX. ([注意]コマンドモードではN. NE. はNEWの略になるため、使用できない。プログラム中では使用してもさしつかえない。)

[書式]
FOR 変数名(または配列名)＝開始値 TO 終了値 STEP 増分
:
NEXT 変数名(または配列名)

FOR文からこれに対応するNEXT文までの間の命令を、指定条件が満たされるまで繰り返し実行します。
変数名（または配列名）には開始値がセットされて、FOR文とNEXT文の間にある文が実行され、次にその変数名（または配列名）の値に増分が加算されて、またFOR文とNEXT文の間にある文が実行されます。繰り返し実行された結果、変数名（または配列名）の値が終了値を越えると、このループから抜け出てNEXT文の次の文が実行されます。
変数名、配列名には、整数型、実数型、倍精度実数型が許されますが、文字型は使用できません。

[使用例]

```
10  FOR  A＝3  TO  16  STEP  2
20  PRINT  A,
30  NEXT  A
40  PRINT  ”END”, A
```

```
＞RUN[Enter]
3              5              7              9              11             13
15             END            17
```

この例では、A＝3からはじまって2ずつ増加しながら10～30の文を繰り返し実行し、16を越えたら、つまり17になったところでNEXTの次の文に移ります。
A＝17のときにはループの中の処理は行われないで外に出ます。（A＝15までしか実行されていないことに注意して下さい。）

●開始値、終了値、増分ともにマイナス値を入れることもできます。増分にマイナス値を指定するとカウンタの値は順に減って行きます。増分にマイナス値を指定した場合には、変数の値が終了値よりも小さくなった時点でループから抜け出します。
●STEPを省略することもできますが、このときは増分は1になります。
●開始値、終了値、増分には変数や式を書くこともできます。
その場合、終了値及び増分値は最初のFOR文実行時の状態での計算値がとられ、途中でその値を変えても実行に影響は与えません。

```
10  FOR  A＝B  TO  C＊2  STEP  D
20  B＝6, C＝A, D＝A＋2   ←このような文が入っても終了値、増分は変化しない。
      :
50  NEXT  A
```

●FOR～NEXT文の中に別のFOR～NEXT文を多重に使用することができます。
多重使用（ネスティング）はシステムのスタックエリアをそれだけ多く使用するため、限度を越えるとエラーコードが表示されます。
なおスタックエリアは（ ）を多重に使用した数値演算や、GOSUB文でも使用されるため、FOR～NEXT文を何重に使用できるかは、そのプログラムによって異なります。
FOR～NEXT文を多重使用する場合には、内側のループ文は外側のループ文に完全に含まれていなければなりません。

（正）

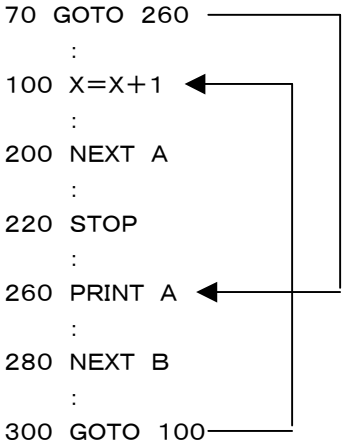
```
10  FOR  A＝1  TO  20
      :
50  FOR  B＝3  TO  60
      :
120  NEXT  B
      :
200  NEXT  A
      :
```

（誤）

```
10  FOR  A＝1  TO  20
      :
50  FOR  B＝3  TO  60
      :
120  NEXT  A
      :
200  NEXT  B
      :
```

誤って使用した場合にはエラーコードが表示されます。
なお、下の例は外にまたがっているようですが、また戻って来ているので、誤りではありません。

```
10  FOR  A＝1  TO  20
      :
50  FOR  B＝3  TO  60
      :
```

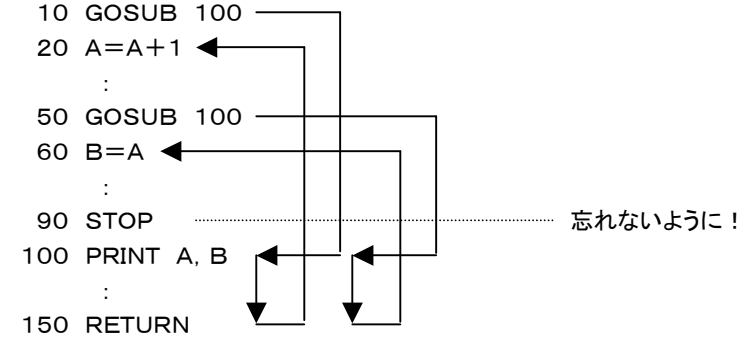


GOSUB～RETURN ［省略形］GOS. GOSU. R.　RE. RET. RETU. RETUR.（［注意］コマンドモードではR.はRUNの　略になるため、使用できない。プログラム中では使用してもさしつかえない。）

　サブルーチンの呼び出し、及びサブルーチンからの戻りを制御します。
 GOSUB文で指定する行から始まるサブルーチンを実行し、RETURN文でさきほどのGOSUB文の次の文に戻ります。
　行番号の指定は正の整数のほか＊マーク付のラベル名も許されます。変数や計算式などは使用できません。
 指定した行番号、ラベル名が存在しないときはエラーコードが表示されます。

 GOTO文と似ていますが、GOSUB文の場合にはRETURN文をみつけると、先程のGOSUB文の場所に戻って、その次の文から実行を続けます。
 サブルーチンはこのように呼ばれた先へまた戻って行くので、同じ処理を違う場所で何回も行いたいときに使うと便利です。

［使用例1］
```
  10 GOSUB 300   ……行番号300から始まるサブルーチンに行く
  50 GOSUB ＊ABC   ……ラベル名＊ABCの書かれたサブルーチンに行く
```
［使用例2］



　1つのプログラムの中でサブルーチンは、いくつ使っても構いません。またサブルーチンの中で別のサブルーチンを呼び出す、サブルーチンの多重使用も許されますがFOR～NEXTと同様にGOSUBとRETURNがペアになっている必要があります。
 サブルーチンの中でGOTO（特に IF GOTOが間違いやすい）を多用するとRETURN文を実行しないでまたもとのGOSUB文を実行してしまうようなプログラムになってしまうことがあります。このような場合には急速にスタックを消費するため、すぐにエラーになってしまいます。
 またGOSUB文を実行していないのにGOTO文でサブルーチンの途中に入ってしまい、RETURN文を実行してしまうようなプログラムになることがあります。この場合もエラーコードが表示されます。

GOTO ［省略形］G. GO. GOT.

 指定する行にジャンプしてそこから実行を続けます。
　行番号の指定は正の整数のほか＊マーク付のラベル名も許されます。変数や計算式などは使用できません。
 指定した行番号、ラベル名が存在しないときはエラーコードが表示されます。

［使用例］
　10 GOTO 300　……行番号300にジャンプする
　50 GOTO ＊ABC ……ラベル名＊ABCの行にジャンプする

IF…THEN…ELSE
IF…GOTO…ELSE ［省略形］I. TH. THE. E. EL. ELS.

IFの後に続く式（関係式または論理式）の結果が真であればTHENの後の文を実行し、（IF…GOTOの場合はそのGOTO文を実行し）、偽であればELSEの後の文を実行します（ELSE以降が無い場合には、その次の行が実行されます）。
IF文の中に別のIF文を書くことが許されますが、勘違いをし易いので避けたほうが賢明です。
THEN、ELSEにすぐ続けてGOTO文を書くことはできますが、THEN 行番号、ELSE 行番号 の形は許されません。

［例1］
10 IF A＜10 THEN PRINT ”YES”:GOTO 100 ELSE PRINT ”NO”:GOTO 50

例1の様にTHEN、ELSEの後ろには複数の文を書くことができます（例1では最後がGOTO文になっていますが常に最後がGOTO文である必要はありません）。

［例2］
10 IF A＜10 THEN PRINT ”YES” ELSE PRINT ”NO”:GOTO 50
20 PRINT ”END”

例2でAの値が10以下の時、PRINT ″YES″ の次に実行されるのは、GOTO 50 ではなくて、この次の行（行番号20）であることを理解して下さい。つまりELSEの後ろに書かれた文はコロン（:）で区切られていても、すべてELSEにかかっています（IF文の次の文として独立させることはできません）。

●文字型の比較
関係演算子（＝、＜、＞、＜＝、＞＝、＜＞）を使って、文字の大小比較ができます。 文字型の比較は文字数の多い方が大になります。
同じ文字数の場合には先頭から1文字ずつキャラクタコードを比較していき、最初に大きいキャラクタコードの文字があらわれた方を大とします。

INPUT ［省略形］IN. INP. INPU.

キーボードから変数または配列にデータを入力します。
入力データとして、定数以外の変数名、配列名や式を入力することはできません。
PRINT文などと同じく要素を ，で区切って複数個記述することができます。

［使用例］
10 INPUT A, B, C
　このプログラムが実行されるとまず、A？と表示して入力待ちになります。
　そしてこれに答えると次に、B？と表示して再び入力待ちになります。
　このようにしてすべてのデータが入力されるまで繰り返し入力が行われます。

●INPUT文にはPRINT文と同じ文字列表示機能があります。
次のように文字列を” ”で囲んで記述することにより、その文字列を表示させることができます。

［使用例］
10 INPUT”DATA”, A, ”NINZU ”, C
＞RUN［Enter］
DATAA？　……これに入力すると、その次が表示される
NINZU C？

文字列と変数との間のカンマ（ , ）を省略すると変数名及び？の表示が省略されます。
10 INPUT”DATA”A, ”NINZU ”C
＞RUN［Enter］

DATA　……これに入力すると、その次が表示される
NINZU

[注記]INPUT文が実行されると、コマンドモードと同じようにカーソルマークが表示されますが、この時はコマンドモードとは別の入力方式になっているため[Insert][Delete][←][→][↑][↓]は使用できません。[Backspase]は使用できます。

LET [省略形]L. LE.（[注意]コマンドモードではL. はLISTの略　になるため、使用できない。プログラム中では使用してもさしつかえない。）

変数、配列に値を代入します
LET A＝3
LET B＝(C＋D)／A
のように左辺は、変数、配列でなければなりませんが、右辺は定数、式も許されます。式の場合はその式を計算した結果が変数に格納されます。
なお、LETは完全に省略することができ、例えば、
LET A＝A＋3　は
A＝A＋3
と表記します。
これは数学的にはおかしいのですが、この＝は等号ではなく代入記号として用いられていると考えて下さい。
つまり A＝(イコール)A＋3ではなくて、AにA＋3の結果を代入する、の意味です。
LET文はPRINT文やINPUT文と同じく要素を ，で区切って複数個記述することができます。

[使用例]
10 LET A＝1, b%＝0, c(i, j)＝3＋A, X$＝"ABC"
LETは省略することができます
10 LET A＝1, b%＝0, c(i, j)＝3＋A, X$＝"ABC"
なお文字型の場合には、＝の右側には、文字定数、文字変数、文字配列を書くことができますが、式としては次の例のように、複数の文字変数、文字定数、文字関数を ＋ でつないで書くことのみが許されます。

[使用例]
10 A$＝"ABC", B$＝"XYZ"
20 C$＝A$＋"HIJK"＋B$＋CHR$($42)
30 PRINT C$
>RUN[Enter]
ABCHIJKXYZB
>
[注意]文字変数(文字配列)には最大39桁の文字列しか代入できません。もしそれよりも大きい文字列を代入しようとすると、エラーコードが表示されます。

ON ERROR GOTO [省略形]ON. ON . ON E. ON ER. ON ERR. ON ERRO.
ON ERROR. ON ERROR . ON ERROR G. ON ERROR GO. ON ERROR GOT.

プログラムの実行中にエラーが発生した場合に、この文で指定する行番号の文が実行されます(このときシステム変数ERLには、エラーが発生した行番号がセットされ、ERRにはエラーの種類を示すコードがセットされます)。
この文はエラーが発生するより前に実行されている必要があります。したがってプログラムの一番先頭に書いておくようにします。
存在しない行番号を指定するとエラーになりますが、行番号として0を指定することは許されます。ON ERROR GOTO 0 の実行後は、エラーが発生してもエラー処理ルーチンは実行されずに、エラーコードを表示してブレイクします。
なおエラー処理ルーチンの終わりは、RESUME文かSTOP文である必要があります([注意](3)参照)。

[使用例]
10 ON ERROR GOTO 100
20 INPUT A, B
30 PRINT A／B
40 GOTO 20
100 PRINT "ERROR！", ERL, ERR

```
110 RESUME NEXT
```
上のプログラムを実行して、Bに0を入力すると下のように表示されます。
```
ERROR！        30        12        …エラーコード12は0除算エラーです
A？
```

[注意]エラー処理ルーチンには、通常の命令文を自由に使用して構いません。また行数の制限はありません。しかしエラー処理ルーチンと普通のルーチンとは、みかけ上は区別できませんが、次の点が異なっています。
(1)エラー処理ルーチンの実行中に[Ctrl]Bによりブレイクした場合、CONTコマンドで実行を再開することはできません。
(2)エラー処理ルーチンの実行中にエラーが発生した場合には、ON ERROR GOTO文は実行されず、エラーコードを表示してブレイクします。この場合もブレイク後にCONTコマンドで実行を再開することはできません。
(3)RESUME文によりエラー処理ルーチンが終了し、メインルーチンの実行継続に必要な情報がセットしなおされます。
たとえばFOR～NEXTやGOSUB～RETURN文の途中でエラーが発生して、ON ERROR GOTO文によりエラー処理が行われたあと、RESUME文によらずにGOTO文などでもとのルーチンに戻ると、スタックの食い違いにより異常な動きになります。

ON n GOSUB [省略形]ON n GOS. ON n GOSU.

[書式]ON n GOSUB L1, L2, ……

GOSUBの後ろに複数個ならべて書かれた行番号のうち、前からn番目にある行番号のサブルーチンを実行します。
nは変数、配列でも構いませんがその値は正の整数に限ります。GOSUBの後ろに書かれた行番号の個数より多い値を指定すると、GOSUB文は実行されません。
L1、L2、……には行番号のほか、＊ではじまるラベル名を書くこともできます。

[使用例]
```
10 ON XYZ GOSUB 100, 550, 230
```
この例ではXYZ＝1のときGOSUB 100が実行され、XYZ＝2のときGOSUB 550が実行され、XYZ＝3のときGOSUB 230が実行されます。

ON n GOTO [省略形]ON n G. ON n GO. ON n GOT.

[書式]ON n GOTO L1, L2, ……

GOTOの後ろに複数個ならべて書かれた行番号のうち、前からn番目にある行番号 の文へジャンプします。
nは変数、配列でも構いませんがその値は正の整数に限ります。GOTOの後ろに書かれた行番号の個数より多い値を指定すると、GOTO文は実行されません。
L1、L2、……には行番号のほか、＊ではじまるラベル名を書くこともできます。

[使用例]
```
10 ON XYZ GOTO 100, 550, 230
```
この例ではXYZ＝1のときGOTO 100が実行され、XYZ＝2のときGOTO 550が実行され、XYZ＝3のときGOTO 230が実行されます。

OUT [省略形]O. OU.

指定したI／Oアドレスに8ビットのデータを出力します。
アドレス、データとして変数、配列や数式を使うこともできます。ただし、I／Oアドレス、データ共に1バイトなのでその値は0～255（＄00～＄FF）に限られます。

[使用例]
```
10 OUT ＄83, ＄80
```
上の文の実行によりI／Oアドレス＄83のI／Oデバイスに、8ビットのデータ、＄80が出力されます（ND80ZⅢの基板上に実装されている82C55のアドレスが＄80～＄83になっていて、＄83は82C55のコントロールワードアドレスになっていますから、上の文は82C55のA～Cポートを出力にセットすることになります）。

[注意]I／Oアドレスの一部はシステム内で特定の機能に使われているため、システムで使用しているI／Oアドレスに対してOUT命令を実行すると暴走する場合があります。注意して下さい。

POKE [省略形]PO. POK.

指定したメモリアドレスに8ビットのデータを書き込みます（読み出しはPEEK関数を使います）。

[使用例]
10 POKE ＄C000, ＄C3 ……＄C000に16進データ＄C3が書き込まれます。

[注意]プログラムや重要なデータの入っているメモリアドレスに対してこの命令を使うと、そのメモリ内容を壊してしまうためプログラムが暴走することもあります。書き込むアドレスはよく検討してから使って下さい。

PRINT ［省略形］P. PR. PRI. PRIN.

ディスプレイ画面に、変数、配列の値、計算式の値、文字列などを表示します。
表示は現在のカーソル位置から行われ、1行で表示できないときは次の行に自動的に改行して出力されます。
なお、LET文と同じく要素をカンマ(,)で区切って複数個記述することができます。

[使用例]
10 A＝123. 45, I%＝200, X＄＝"XYZ"
20 PRINT A, I%, X＄, A＋I%
30 PRINT A;I%;X＄;A＋I%

＞RUN[Enter]
123. 45 200 XYZ 323. 45
123. 45200XYZ323. 45
パラメータをカンマ(,)で区切ると、先頭から13桁毎に間をあけて表示されます。またセミコロン(;)で区切ると間をあけないで表示されます。

●PRINT文の最後にカンマ(,)やセミコロン(;)をつけると、改行が行われません。上の例で、20 PRINT A, I%, X＄, A＋I%; とすると、結果は下のようになります。
＞RUN[Enter]
123. 45 200 XYZ 323. 45123. 45200XYZ323. 45

●PRINTとだけ書いておいた場合には、1行改行命令になります。

[使用例]
10 PRINT "ABC"
20 PRINT
30 PRINT "DEF"
＞RUN[Enter]
ABC

DEF

● ￥記号を使用して、数値を出力する場合にその桁数を指定することができます。下の例でその効果を確認して下さい。

[￥未使用]
```
10 FOR A=0 TO 10
20 PRINT A, 2^A
30 NEXT A
＞RUN
0           1
1           2
2           4
3           8
4           16
5           32
6           64
7           128
```

[￥使用]
```
10 FOR A=0 TO 10
20 PRINT ¥5, A, 2^A
30 NEXT A
 ＞RUN
0            1
1            2
2            4
3            8
4           16
5           32
6           64
7          128
```

8	256	8	256
9	512	9	512
10	1024	10	1024

￥で桁数を指定すると、出力される数値の前に空白をおくことで空白桁＋数値の桁数＝￥で指定した桁数にします。その結果、数値の後ろで桁を揃えて表示することができます。もしも￥で指定した桁よりも大きい数値が出力された場合には、￥の指定に関わらず全桁数が表示されます。
なお￥の効力はそのPRINT文の中でのみ有効です。次のPRINT文では初期値（＝1）に戻ります。

READ ［省略形］REA.

DATA文に書かれているデータを変数に読み込みます。
最初に実行されるREAD文は一番小さい行番号のDATA文のデータから読み込みを開始します。READ文もDATA文も一つのプログラムに幾つ書いても、また幾つの変数を割り当てても構いませんが、READ文の変数とDATA文の定数の型が異なっていたり、DATA文のデータの数が不足すると、エラーになります。
DATA文のデータの方が多い場合には残りのデータが無視されるだけで、エラーにはなりません。
使用例はDATA文の項を参照して下さい。
なおDATA文の最初に戻って、始めのデータから読みなおしたい時や、途中から読み出したい時にはRESTOREを使います。

READ ＃ ［省略形］REA. ＃

他のパソコンなどとデータの送受信（シリアル伝送）をするための命令です。
READ＃命令の実行によって受信データが変数に格納されます。

［使用例］　　READ ＃1, Ａ$, A%

＃はチャンネルを指定します。＃1を指定してください。
第2パラメータは文字変数（文字型配列）でなければなりません。ここに受信データが入れられます。
第3パラメータは整数型変数でなければならず、省略はできません。ここには受信したデータの桁数が入れられます。
データの終わりを示すコード（0D・0A）を受信するとリターンしますが、39桁を受信して文字変数の桁数が一杯になっても、0D・0Aが受信されないときは、Ａ$には39桁のデータをセットするとともにA%に40をセットしてリターンします。
READ＃文を実行したあとで、このA%の値をチェックし、もし40になっていたらまだ受信すべきデータが残っていると判断できます。
次に再びREAD文を実行したとき、データが無くて、すぐに0D・0Aコードを受信した場合には、Ａ$には” ”（1桁のスペース）が入りますがA%には0がセットされます。
したがって、このスペースが受信データか無視すべきデータかは、A%が0であるかどうかで判断できます。

RS232C受信はND80ZⅢのボード上に実装されているPIC18F14K50が行います。
PIC18F14K50はBASIC命令の実行とは関係無く、データが受信される度に割り込みによって受信バッファ（256バイト）に受信データをたくわえていきます。
READ＃命令が実行されると、PIC18F14K50にデータの引渡しが要求され、受信バッファの先頭から最初の0D・0Aコードまでが文字変数に読みこまれます。再びREAD＃命令が実行されると先に読んだ0D・0Aコードの次のデータから次の0D・0Aコードまでが文字変数に読みこまれます。なおいずれの場合でも0D・0Aコードそのものは文字変数には読みこまれません。
READ＃命令が実行された時に受信バッファの中に0D・0Aコードで区切られたデータが複数個格納されている場合には、READ＃命令を実行する度に順にデータが文字変数に読みこまれることになります。そして整数型変数にはその時のデータ長（桁数）が入ります。データ長が40桁以上の場合にはそのデータを読み込む1回目のREAD＃命令で文字変数に最初の39桁が格納され、整数型変数には40が格納されます。再びREAD＃命令を実行することでデータの残りを読み込むことができます。
READ＃命令が実行された時に受信バッファが空の場合には文字変数には1桁の空白が格納され、整数型変数には0が格納されます。
READ＃命令が実行された時に、データが受信の途中で何桁かは受信されているがまだ0D・0Aコードまでは受信されていない場合には、文字変数には受信途中のデータが格納され、整数型変数には－1が格納されます。受信バッファにデータがある場合にはそれが0D・0Aコードがまだ受信されていない途中のデータであっても、READ＃命令によって文字変数に読みこまれてしまいますが、整数型変数が－1かどうかを調べることで、受信途中の半端なデータか0D・0Aコードで区切られたデータかを区別することができます。

[プログラム例]
```
10 KETA%=0
20 READ #1,RDATA$,KETA%
30 IF KETA%<=0 GOTO 20
40 PRINT RDATA$;
50 IF RDATA$="END" THEN STOP
60 IF KETA%=40 GOTO 10 ELSE PRINT:GOTO 10
```

行番号50で転送データの終わりを"END"で判定していますが、これは例であって、データの最後に"END"を送信しなければならない、ということではありません。

[注記1]最初にREAD　#1で受信を開始する前に、必ず整数型変数を0クリアしてください(行番号10参照)。
また0D・0Aコードを受け取るか39バイトを受信して文字変数のデータが確定したあと、新しいデータを読み込む前にも整数型編買うを0クリアしてください(行番号60参照)。このようにしないと正しく受信されません。
[注記2]RS232C通信のボーレートは、ND80ZⅢのディップスイッチDS1の№2、№3 によって決定されます。データは8ビット、ストップビット長1、パリティ無しです。詳しくは「ND80ZⅢ取扱説明書」Ⅷ. RS232Cインターフェースを参照してください。
[注記3]PIC18F14K50はRS232Cデータを8ビットのバイナリデータとして扱いますが、READ　#命令による受信データは、ASCII文字データとして扱います。文字データの区切りコードは0D0Aです。受信した文字データは0D0Aコードを除いて文字変数に格納されたあと、PRINT文などで使われます。ですから一般にコマンドプロンプトで画面に表示できない、00～1Fなどのコードを受信すると、その結果をPRINT文で表示したときに、異常な表示になったり、システムが暴走する可能性がありますから、注意してください。

REM　[省略形]なし

プログラム中に、実行されないコメントを書くのに使います。このREMから行の終わりまでに書かれた文字は、実行時には無視されます。
なおREMの代わりにアポストロフィ(') で代用することができます。

[使用例]
```
10 REM *** TEST PROGRAM ***
20 'Good Morning!
```

RES　[省略形]なし

変数(または配列)の値が0～255の範囲(つまり8ビットの数)であるとき、そのうちの任意の1ビットを0にします。残りの7ビットには影響を与えません。マシン語のRES命令と同じ働きをします。

[使用例]
```
10　A＝＄FF:PRINT　HEX＄(A), BI＄(A)
20　RES　A, 3
30　PRINT　HEX＄(A), BI＄(A)
＞RUN[Enter]
FF                  11111111
F7                  11110111
```

[注意]第2パラメータ(上の例では3)には定数の他に、変数、配列や式を書くこともできますが、その値は0～7の範囲の整数に限られます。
第1パラメータ(上の例ではA)には変数または配列以外は使用できません(共に整数型か実数型に限ります)。

RESTORE　[省略形]RES. REST. RESTO. RESTOR.

READ文で読み込みを開始するDATA文を指定します。
普通READ文は小さい行番号のDATA文から読み取りを開始し、順に読んで行きますが、処理によってはもう一度前のDATA文に戻って読み取りたい場合や、間をとばして先のDATA文を読みたい場合があります。
READ文に先立って、読み取りを始めたいDATA文の行番号を、このRESTORE文で指定しておくことによって、任意のDATA文から読み取りを開始することができます。

RESTOREの後に行番号を指定しない場合は、一番小さい行番号のDATA文が指定されたことになります。

[使用例]
```
10 DATA ABC, XYZ, 123. 45
20 DATA 10, 20, 30
30 READ A$, B$
40 RESTORE 20
50 READ I, J
60 RESTORE
70 READ C$
80 PRINT A$, B$, C$, I, J
>RUN[Enter]
ABC             XYZ             ABC             10              20
```

RESUME [省略形]RESU. RESUM.

ON ERROR GOTO文でエラー発生時の処理をしたあと、次に実行する文を指定します。
RESUME のように後ろに何もつけない場合は、エラーの発生した文から再実行します。RESUME NEXT と後ろにNEXTをつけると、エラーの発生した文の次の文から実行を開始します。
RESUME 50 のように後ろに行番号をつけると、その行から実行を開始します。

[使用例]
```
10 ON ERROR GOTO 100
20 INPUT A, B
30 PRINT A／B
40 GOTO 20
100 PRINT "ERROR！", ERL, ERR
110 RESUME NEXT
>RUN[Enter]
A？10[Enter]
B？0[Enter]
ERROR！          30              12
A？
```

RETURN [省略形]R. RE. RET. RETU. RETUR.

GOSUB文で指定したサブルーチンの終わりを示します。RETURN文が実行されると、GOSUB文の次の文に戻って処理を続けます。
詳しくは、GOSUB～RETURNの説明を参照して下さい。

SET [省略形]SE.

変数（または配列）の値が0～255の範囲（つまり8ビットの数）であるとき、そのうちの任意の1ビットを1にします。残りの7ビットには影響を与えません。マシン語のSET命令と同じ働きをします。

[使用例]
```
10 A＝＆0F:PRINT HEX＄(A), BI＄(A)
20 SET A, 7
30 PRINT HEX＄(A, 2), BI＄(A)
>RUN[Enter]
0F              00001111
8F              10001111
```

[注意]第2パラメータ（上の例では7）には定数の他に、変数、配列や式を書くこともできますが、その値は0～7の範囲の整数に限られます。
第1パラメータ（上の例ではA）には変数または配列以外は使用できません（共に整数型か実数型に限ります）。

STOP [省略形]S. ST. STO.

プログラムの実行を中止してブレイクします。このとき画面には次の表示が出ます。

Break　in　LLLLL　（LLLLL はSTOP文の書かれている行番号）

この後キーボードからCONTコマンドを入力すれば、このSTOP文の次の文から実行を再開させることができます（ただしプログラムの最後にSTOP文が書かれていて、そのSTOP文でブレイクした場合には、実行を再開させることはできません）。

SWAP

［書式］SWAP　A, B

2つの変数、配列の値を交換します。双方の型は同じでなければいけません。
型が同じならば変数と配列との間での値の交換もできます。

［使用例］
```
10　SWAP　ABC%, XYZ%
20　SWAP　AB#, XY#
30　SWAP　AA$(3), XX$
```

TRON
TROFF　［省略形］なし

BASICプログラムの実行をトレースします。
プログラム中にTRON命令を書いておくとそのTRON命令実行後は、行番号が実行順に画面に表示されます。
行番号は通常の出力データと区別できるように、[　]で囲んで出力されます。
TROFF命令が書かれている行を実行するとそれ以後は行番号が出力されなくなります。
BASICプログラムのデバッグの過程で、プログラムが実行される順序を追跡したい場合があります。そのような時にこのコマンドを必要と思われる区間に挿入することで、簡単に実行のトレースができます。
プログラムの中に追加しなくても、TRON命令を行番号なしでダイレクトに実行したあとRUNコマンドを入力すればプログラムのはじめからトレースします。
またプログラムの実行途中でブレイクし、TRON命令をダイレクト実行したあと、CONTコマンドを入力すれば実行再開時点からトレースされます。
TRONはプログラム中またはブレイク後にダイレクトモードでTROFF命令が実行されるまで機能し続けます。
TROFF命令の実行の他NEWコマンドの実行、[Ctrl]B、[Ctrl]CによるZB3BASICの終了によってもTRONは解除されます。

［使用例］
```
10　PRINT　”START”
20　TRON
30　FOR　A＝1　TO　3
40　PRINT　”A＝”;A
50　NEXT　A
60　TROFF
70　PRINT　”END”
＞RUN[Enter]
START
[30][40]A＝1
[50][40]A＝2
[50][40]A＝3
[50][60]END
＞
```
TRON実行前の行番号10、20及びTROFF実行後の70は出力されていないことに注目して下さい。
この命令は同じBASICプログラム中で何回使用しても構いません。

USR　［省略形］U. US.

［書式］USR（＄bbbb）

マシン語サブルーチンを呼び出して実行します。
＄bbbbのところにコールしたいマシン語サブルーチンプログラムの先頭アドレスを＄マーク付の16進数4桁で表記します（16進数の代わりに、変数、配列、数式を書くこともできます。本当は16進数ではなくて10進数でもよいのですが、メモリアドレスは16進数で扱いますから、それをわざわざ10進数に直すのは意味がありません）。
マシン語サブルーチンの実行終了後、このUSR文の次の文から再びBASICプログラムが実行されます。
なお、マシン語プログラムに行く前に必要なレジスタの退避は行われているため、AF、BC、DE、HL、AF’、BC’、DE’、HL’、IX、IYの各レジスタは自由に使用できます。（ただしSPを不用意にいじると、もとのBASICに戻れなくなってしまうので注意して下さい）
マシン語サブルーチンにエラーがあっても、BASICとは異なりエラーメッセージは出ません。場合によっては暴走したりハングハングアップすることもすることもありますから、マシン語プログラム部分は予め十分デバッグをしてエラーのないようにしておいて下さい。

［使用例］
10 USR(＄8050)

［注意］当然のことですがマシン語サブルーチンの最後は必ずRET命令でなければなりません。
また、スタックを使用した場合はもとのレベルに戻しておかなければ正しくBASICに戻ることはできません（PUSHとPOPの数の食い違いやCALL、RETの組合わせなどに注意）。
なおマシン語プログラムの実行中は[Ctrl]B を入力してもブレイクはできません（中止するには、［Ctrl］Cを入力するかDOS窓の右上の［×］ボタンをマウスでクリックします）。

WRITE ＃［省略形］W. ＃ WR. ＃ WRI. ＃ WRIT. ＃
他のパソコンなどとデータの送受信（シリアル伝送）をするための命令です。
WRITE＃命令によって、変数または定数データが送信されます。

［使用例］
10 WRITE ＃1 ,”ABC”,A＄ ,……

＃はチャンネルを指定します。＃1のみが使用できます。
第2パラメータ以降のデータは文字定数か文字変数（文字型配列）でなければなりません。
区切マークの ,（カンマ）の代わりに;（セミコロン）を使うことができます。
上記例のように複数の文字型データを ,（カンマ）または;（セミコロン）で区切って記述した場合には、各データはひとつのデータとして送信されます。
最後に ,（カンマ）または;（セミコロン）がある場合には、データの終わりを示すコード（0D・0A）は送信されません。
,（カンマ）も;（セミコロン）も無しで終わると、データのあとに、0D・0Aコードが送信されます。
［例］
WRITE ＃1 ,”ABC”,”XYZ”,
送信されるデータ（16進数で示す）……41・42・43・58・59・5A
WRITE ＃1 ,”ABC”,”XYZ”
送信されるデータ……41・42・43・58・59・5A・0D・0A

関数はいままで説明してきたコマンドや命令（ステートメント）とは少し性質が異なっています。
関数は命令（ステートメント）のようにBASIC文の中で単独で使うことはできず、LET文やPRINT文の中で、ある値を持つ数値として扱われます。
なお関数は（ ）の中に記述するパラメータをもとにして、それぞれの処理をした結果を、その関数の値としてもちますが、システム変数はパラメータが不要で、ある特定の値をシステムが与えます。

ABS ［省略形］A. AB.

［書式］ABS（a）

絶対値を計算します。
（ ）の中には、定数、変数、配列、数式、関数（いずれも文字型を除く）が記述できます。

［使用例］
```
10 A＝ABS（B＊C－10）
```

AND ［省略形］AN.

［書式］AND（a, b）

8ビットの2数の論理積（AND）を計算します。マシン語のAND命令と同じ働きをします。
（ ）の中には、定数、変数、配列、数式、関数（いずれも文字型を除く）が記述できますが、その値は0～255の範囲の整数でなければ、正しい結果は得られません。
またこのAND関数の取りうる値の範囲も0～255（＄00～＄FF）です。

［使用例］
```
10 A＝＄37
20 B＝AND（A, ＄0F）
30 PRINT HEX＄（A, 2）, BI＄（A）
40 PRINT HEX＄（B, 2）, BI＄（B）
＞RUN［Enter］
37              00110111
07              00000111
```

ASC ［省略形］AS.

［書式］ASC（a）

文字列の最初の1文字のキャラクタコードを与えます。（ ）の中には文字定数、文字変数が記述できます。

［使用例］
```
10 INPUT A＄
20 CD＝ASC（A＄）:PRINT HEX＄（CD）,
30 IF （＄30＜＝CD）＊（CD＜＝＄39） THEN PRINT ”NUMERIC” ELSE PRINT ”ALPHA”
40 GOTO 10
＞RUN［Enter］
A＄？12345［Enter］
31              NUMERIC
A＄？ABC［Enter］
41              ALPHA
A＄？
```

ATN ［省略形］AT.

［書式］ATN（a）

逆正接(tan－1) を計算します。

BCD ［省略形］BC.

［書式］BCD(a)

0～99の範囲の整数をBCD2桁に変換します。( )の中には定数、変数、配列、数式、関数(いずれも文字型を除く)が記述できますが、その値は0～99の範囲の整数でなければ、正しい結果は得られません。
LED(7セグメント)表示回路などでは0～9の数1桁を4ビットで扱います(これをBCD表現)といいます。
例えば10進数の56は計算機内部では16進数の38(00111000)として扱われており、したがってこの数をI／Oポートから出力すると当然16進数の38が出力されます。これではBCD表現の回路では10進数の「38」として受け取られてしまいます。BCD表現の回路では10進数の56が16進数の56(01010110)として表される必要があります。
この変換を計算で求めることもできますが、BCD関数を使えば簡単に値が求まります。BCD関数の機能はI／Oポートを介してBCD数の受渡しをすることが目的ですから8ビットの数、BCD2桁の数の変換しか行えません。8ビットの2進数(16進数)は0～255(00～FF)までありますが、BCD数は同じ8ビットでも0～99までしか表現できません(10進数の100はBCDでは000100000000になって3桁になります)。BCD(a)のaの値が0～99以外の場合にはエラーになります。

［参考］BCD(a)と逆の働きをする関数にDEC(a)があります。

［使用例］
```
10 A%＝99
20 B%＝BCD(A%)
30 PRINT HEX＄(A%), HEX＄(B%)
＞RUN[Enter]
63              99
```

BI＄ ［省略形］なし

［書式］BI＄(a)［省略形］なし

8ビットの数をビット表現の文字列にします。( )の中には、定数、変数、配列、数式、関数(いずれも文字型を除く)が記述できますが、その値は0～255の範囲の整数でなければ、正しい結果は得られません。

［使用例］
```
10 A＝＄37
20 B＝AND(A, ＄0F)
30 PRINT HEX＄(A), BI＄(A)
40 PRINT HEX＄(B), BI＄(B)
＞RUN[Enter]
37              00110111
07              00000111
```

BIT ［省略形］B. BI.

［書式］BIT(a, n)

8ビットの数の任意のビットを調べ、それが0のときにこの関数の値も0になり、1のときこの関数の値も1になります。マシン語のBIT命令と同じ働きをします。
( )の中には、定数、変数、配列、数式、関数(いずれも文字型を除く)が記述できますが、その値は、第1パラメータは0～255、第2パラメータは0～7の範囲の整数でなければ、正しい結果は得られません。
BIT関数の値は、0か1の2値のみです。
この関数はI／Oポートからの入力データの特定のビットの状態を知りたい時などに使うと便利です。
下の例はI／OのAポートのビット7にLレベルの信号が入力された時に、”DATA IN”と表示させるプログラムです。

［使用例］

```
10 OUT ＆83, ＆90
20 A＝IN(＆80)
30 IF BIT(A, 7)＝0 THEN PRINT "DATA IN" ELSE GOTO 20
```

CHR＄ [省略形]CH. CHR.

[書式]CHR＄(a)

8ビットのデータをキャラクタコードとみなして、そのコードに対応する1桁の文字を発生します。
( )の中には、定数、変数、配列、数式、関数(いずれも文字型を除く)が記述できますが、その値は0～255の範囲の整数でなければ正しい結果は得られません。

[使用例]

```
10 A＄＝CHR＄(41)＋CHR＄(42)＋CHR＄(43)
20 PRINT A＄
>RUN[Enter]
ABC
```

COD [省略形]CO.

[書式]COD(a)

余弦(cos)を計算します。( )の中には、定数、変数、配列、数式、関数(いずれも文字型を除く)が記述できます。
( )内の値の単位は度です。

[使用例]

```
10 FOR A＝0 TO 90 STEP 10
20 S＝SID(A), C＝COD(A)
30 PRINT A, S, C, :IF C＜＞0 THEN PRINT S／C ELSE PRINT "－"
40 NEXT A
>RUN[Enter]
```

0	0	1	0
10	0.173648	0.984808	0.176327
20	0.34202	0.939693	0.36397
30	0.5	0.866025	0.57735
40	0.642787	0.766044	0.839099
50	0.766044	0.642788	1.19175
60	0.866025	0.5	1.73205
70	0.939692	0.34202	2.74748
80	0.984808	0.173648	5.67128
90	1	0	-

COS [省略形]なし

[書式]COS(a)

余弦(cos)を計算します。( )の中には、定数、変数、配列、数式、関数(いずれも文字型を除く)が記述できます。
( )内の値の単位はラジアンです。

[使用例]

```
10 FOR A=0 TO 90 STEP 10
20 D=A*PI/180
30 S=SIN(D),C=COS(D)
40 PRINT A,S,C,:IF C<>0 THEN PRINT TAN(D) ELSE PRINT "-"
50 NEXT A

>RUN[Enter]
```

0	0	1	0
10	0.173648	0.984808	0.176327
20	0.34202	0.939693	0.36397
30	0.5	0.866025	0.57735
40	0.642787	0.766044	0.839099
50	0.766044	0.642788	1.19175
60	0.866025	0.5	1.73205
70	0.939692	0.34202	2.74748
80	0.984808	0.173648	5.67128
90	1	0	-

DEC ［省略形］無し

［書式］DEC（a）

　0～99の範囲のBCD2桁の数を10進数に変換します。（　）の中には定数、変数、配列、数式、関数（いずれも文字型を除く）が記述できますが、その値は0～99の範囲のBCD数（0～63の範囲の10進数）でなければ、正しい結果は得られません。
　LED（7セグメント）表示回路などでは0～9の数1桁を4ビットで扱います（これをBCD表現）といいます。
　例えば10進数の56はBCD数では（01010110）と表され、これは16進数の56になります。したがってこの数をI／Oポートから入力すると16進数の56＝10進数の86として入力されてしまいます。入力された数がBCD数の場合には16進数の56をそのまま10進数の56に変換する必要があります。
　この変換を計算で求めることもできますが、DEC関数を使えば簡単に値が求まります。DEC関数の機能はI／Oポートを介してBCD数の受渡しをすることが目的ですから8ビットの数、BCD2桁の数の変換しか行えません。8ビットの2進数（16進数）は0～255（00～FF）までありますが、BCD数は同じ8ビットでも0～99までしか表現できません（10進数の100はBCDでは000100000000になって3桁になります）。BCD数の0～99はその表現を16進数として扱えば、0～＄99つまり10進数の0～153になります。したがってDEC（a）のaの値がBCD数の0～99以外の時（10進数の0～153以外の時）はエラーになります。
［参考］DEC（a）と逆の働きをする関数にBCD（a）があります。

［使用例］
```
10　A%＝＄99
20　B%＝DEC（A%）
30　PRINT　A%, B%
＞RUN［Enter］
153　　99
```

ERL
ERR ［省略形］なし

　BASICの実行中にエラーが発生した時、ERRにそのエラーコードがセットされ、ERLにはエラーが発生した行番号がセットされます。
　この値はIF文などで数値データとして参照可能です。

［使用例］
```
10　ON　ERROR　GOTO　100
20　INPUT　A, B
30　PRINT　A／B
40　GOTO　20
100　PRINT　”ERROR！LINE＝”;ERL, ”CODE＝”;ERR
110　RESUME　NEXT
RUN［Enter］
A？10［Enter］
B？0［Enter］
ERROR！LINE＝30　　CODE＝12
A？
```

EXP ［省略形］EX.

[書式]EXP(n)

指数関数 e のn乗 を計算します。
( )の中には、定数、変数、配列、数式、関数(いずれも文字型を除く)が記述できます。

[使用例]
10 PRINT EXP(0. 5)

[参考]e以外の一般的な値に対する指数を計算したいときは、演算記号の^(ベキ乗)を使います。
例えば2. 573の－2. 5乗は2. 573^(－2. 5)と書けばよいのです。
なお平方根はSQR関数を使いますが、その代わりにa^0. 5と書いて求めることもできます。
ある数のn乗根は、このベキ乗を使ってa^(1／n)と書けば求まります。

FIX [省略形]FI.

[書式]FIX(a)

( )内の値の小数点以下を切り捨てて整数化します。
( )の中には、定数、変数、配列、数式、関数(いずれも文字型を除く)が記述できます
左辺に整数型の変数を置いて、
10 A%＝123. 45
のようにしても整数化はできますが、この場合には－32768～＋32767の範囲の数しか扱えません。しかしFIXの場合にはこれより大きな数でもエラーにならずに整数化できます。
整数化の関数としてはFIXの他にINTがあります。
FIXとINTは( )内の値が正の場合には同じ結果が求まります。
( )内の値が負の時に、INTはもとの値を越えない最大の整数を返しますが、FIXはもとの値の整数部分のみを返します。

[使用例]
10 A＝123. 45
20 B＝－123. 45
30 PRINT FIX(A), FIX(B), INT(A), INT(B)
＞RUN[Enter]
123 －123 123 －124

HEX＄ [省略形]H. HE. HEX.

[書式]HEX＄(a, n)

aの値を16進数化してその文字列を与えます。
nを省略した場合には、16進数で表したときの上位の桁が0のときに、その桁が省略されます。
nを指定すると上位桁が0でも、nで指定する桁数だけ有効になります。逆に例えば16進数で3桁の数なのに、nに2や1を指定すると上位桁は無視されてしまいます。
( )の中には、定数、変数、配列、数式、関数(いずれも文字型を除く)が記述できますがaの値は－32768～＋32767、nの値は1～4の範囲の整数に限ります。
[使用例]
10 A＝1234
20 PRINT A, HEX＄(A), HEX＄(A, 2), HEX＄(A, 4)
30 PRINT －A, HEX＄(－A), HEX＄(－A, 2), HEX＄(－A, 4)
＞RUN[Enter]
１２３４ ４Ｄ２ Ｄ２ ０４Ｄ２
－１２３４ ＦＢ２Ｅ ２Ｅ ＦＢ２Ｅ

IN [省略形]なし

[書式]IN(a)

I／Oポート等のI／Oアドレスから8ビットのデータを入力します。
OUT命令の反対の働きをします。マシン語のIN命令に相当します。
（ ）の中には、定数、変数、配列、数式、関数（いずれも文字型を除く）が記述できますが、その値は0～255の範囲の整数に限ります。
IN関数の取り得る値の範囲は0～255（＄00～＄FF）です。

[使用例]
```
10 ABCIN=IN($80)
```

INKEY＄ ［省略形］INK. INKE. INKEY.

この変数を参照したときにキーボードが押されていると、そのキーの文字が得られます。
押されていないときは、1桁の空白が得られます。

[使用例]
```
10 A$=INKEY$
20 IF A$=" " GOTO 10
30 PRINT A$
40 GOTO 10
```

このプログラムを実行するとキーを押すたびに、その文字が表示されます。
[注記]USBで交信しているため、キー入力しても実際に応答があるまでに、0．5秒から1秒ていどかかるときもあります。
上のプログラムでは、キーを押しつづけると連続してキーコードが入力されます。
最初の1回だけ、キー入力を受け付けるには、下のようにします。

```
10 B$=" "
20 A$=INKEY$
30 IF A$=" "GOTO 20
40 IF A$=B$ GOTO 20
50 PRINT A$;
60 B$=A$
70 GOTO 20
```

INPUT＄ ［省略形］IN. ＄ INP. ＄ INPU. ＄

[書式]INPUT＄(k, #n)

RS232C受信データバッファ、またはキーボードから指定する桁数の文字を読み取って、その桁数の文字列データとして格納します。
kは読み取る桁数で1～39の範囲の整数で、定数の他、変数、配列、式を書くこともできます。nはRS232Cの回線番号で1を指定します。
＃1を省略するとキーボードからの入力データを読み込みます。
INPUT＄はREAD＃と違って、RS232C受信データバッファが空の時は指定桁数のデータが受信されるまで待ち続けます。
＃1を省略してキーボード入力にした場合、指定した桁数の文字がキーボードから入力されるまで待ち続けます。
INPUT＄はREAD＃やINKEY＄と違って文字コード以外の制御コードも1文字として読み取ります。例えばRS232C受信データの終わりを示す、0D、0Aなどもコードとして読み取ることができます。

[使用例]
```
10 SWICH=0
20 A$=INPUT$(1)
30 IF A$=CHR$($0D) GOTO 80 ……$0D は[CR]の内部コードです
40 PRINT A$;
50 IF SWICH=0 THEN IDATA$=A$ ELSE IDATA$=IDATA$+A$
60 SWICH=1
70 GOTO 20
80 PRINT:PRINT IDATA$
>RUN
```

ABC　……A、B、C、とキーを押したあと[Enter]キーを押す
ABC

[注記]RS232C入力の場合は、00～FFの全てのコードを入力することができます。
キーボードからの場合は、[→][Home][F1]などの特殊キーのコードは読み取れない場合があります。またそれらのコードは80～FFに置き換えて入力されます。

INSTR ［省略形]INS. INST.

[書式]INSTR(n, 文字列1, 文字列2)

文字列の中から任意の文字列を捜してその文字の位置を与えます。
文字列1は対象になる文字列で文字変数または文字型の配列を置きます。文字列2は捜したい文字列で文字変数、文字型配列の他、文字定数を置くこともできます。文字定数を書く場合には　”　”で囲んで表記します。
nは捜し始める位置で、省略すると文字列(A$)の始めから捜し始めます。
指定する文字列が見つかった場合には、その文字位置(先頭から数えて〇桁目)を返します。みつからなければ0を返します。

[使用例]
```
10 ABC$="AABBCCDEFGHIJJJKLMN"
20 PRINT INSTR(3,ABC$,"IJJJK")
>RUN
12
```

INT ［省略形]無し

[書式]INT(a)

( )内の値の小数点以下を切り捨てて整数化します。
( )の中には、定数、変数、配列、数式、関数(いずれも文字型を除く)が記述できます
整数化の関数としてはINTの他にFIXがあります。
FIXとINTは( )内の値が正の場合には同じ結果が求まります。
( )内の値が負の時に、INTはもとの値を越えない最大の整数を返しますが、FIXはもとの値の整数部分のみを返します。

[使用例]
```
10 A=123.45
20 B=-123.45
30 PRINT FIX(A),FIX(B),INT(A),INT(B)
>RUN
123           -123          123           -124
```

LEFT$ ［省略形]LEF. LEFT.

[書式]LEFT$(文字列 ,n)

文字列の左端から任意の長さだけ取り出した文字列を与えます。
( )内の文字列は定数、変数、文字型の配列のいずれでもよく、また長さを指定する数値は定数、変数、配列、式のいずれでもよいのですがその値は1～39の範囲の整数に限ります。

[使用例]
```
10 A$="ABCDEFG"
20 L$=LEFT$(A$,3),M$=MID$(A$,4,3),R$=RIGHT$(A$,3)
30 PRINT L$,M$,R$
>RUN[Enter]
ABC           DE            EFG
```

LEN ［省略形]なし

[書式]LEN(文字列)

文字列の文字数を与えます。
( )の中には文字定数、文字変数、文字型の配列が記述できます。

[使用例]
```
10 A$="ABCDE"
20 PRINT LEN(A$)

>RUN[Enter]
5
```

LN [省略形]なし

[書式]LN(a)

自然対数 loge X を計算します。
( )の中には、定数、変数、配列、数式、関数(いずれも文字型を除く)が記述できますが、その値は正の数である必要があります。

[使用例]
```
10 PRINT LN(0.1)
```

LOG [省略形]なし

[書式]LOG(a)

常用対数 log10 X を計算します。
( )の中には、定数、変数、配列、数式、関数(いずれ文字型を除く)が記述できますが、その値は正の数である必要があります。

[使用例]
```
10 PRINT LOG(0.1)
```

MID$ [省略形]M. MI. MID.

[書式]MID$(文字列 ,n, m)

文字列の左端からn番目から始まってm個を取り出した文字列を与えます。
文字列の部分には文字定数、文字変数、文字型の配列が記述できます。
n、mは定数、変数、配列、式のいずれでもよいのですがその値は1～39の範囲の整数に限ります。
nが文字列の桁数より大きいとエラーになります。mが文字列の桁数より大きいときはもとの文字列全体が与えられます。

[使用例]
```
10 A$="ABCDEFG"
20 L$=LEFT$(A$,3),M$=MID$(A$,4,2),R$=RIGHT$(A$,3)
30 PRINT L$,M$,R$
>RUN[Enter]
ABC             DE              EFG
```

OR [省略形]なし

[書式]OR(a, b)

8ビットの2数の論理和(OR)を計算します。マシン語のOR命令と同じ働きをします。( )の中には、定数、変数、配列、数式、関数(いずれも文字型を除く)が記述できますが、その値は0～255の範囲の整数でなければ、正しい結果は得られません。

またこのOR関数の取り得る値の範囲も0～255（＄00～＄FF）です。

[使用例]
```
10 A=$37
20 B=OR(A,$0F)
30 PRINT HEX$(A),BI$(A)
40 PRINT HEX$(B),BI$(B)

>RUN[Enter]
37          00110111
3F          00111111
```

PEEK ［省略形］PE. PEE.

［書式］PEEK（a）

POKE文の逆の働きをする関数です。指定するメモリアドレスの内容がこの関数の値になります。したがって取り得る値の範囲は8ビットの整数（0～255）になります。
（ ）の中には、定数、変数、数式、関数（いずれも文字型を除く）が記述できますがその値はメモリアドレスの範囲（16ビットの数）に限られるため16進数の＄0000～＄FFFF（10進数の－32768～＋32767）でなければなりません。

[使用例]
```
10 A=PEEK($F800)
```

PI
PI# ［省略形］なし

πの値をとるシステム定数です。三角関数等の計算でπが必要な時に使います。
PI は実数型（有効数字6桁）でPI#は倍精度実数型（有効数字16桁）です。

[使用例]
```
10 D=A*PI/180
```

RIGHT＄ ［省略形］RI. RIG. RIGH. RIGHT.

［書式］RIGHT＄（文字列 ,n）

文字列の右端から任意の長さだけ取り出した文字列を与えます。
（ ）内の文字列は定数でも変数でもよく、また長さを指定する数値は定数、変数、配列、式のいずれでもよいのですがその値は1～39の範囲の整数に限ります。

[使用例]
```
10 A$="ABCDEFG"
20 L$=LEFT$(A$,3),M$=MID$(A$,4,2),R$=RIGHT$(A$,3)
30 PRINT L$,M$,R$
>RUN[Enter]
ABC             DE              EFG
```

SEARCH ［省略形］SEA. SEAR. SEARC.

［書式］SEARCH（配列名，n，s，d）

配列名で指定した配列要素の中から整数値nを捜し、最初にみつかった要素・を返します。最後まで捜してもnを値とする配列要素がみつからなかったときは－1を返します。
配列名として指定できる配列は整数型の一次元配列でなければいけません。
nは捜したい値で、整数型の変数、定数、配列のいずれでも指定できます。

sは配列のどこから捜しはじめるかを示す値で、整数型の変数、定数、配列のいずれでも指定できます。sを省略すると配列の最初の要素(添字の値＝0)から捜しはじめます。
dはステップ値で整数型の変数、定数、配列のいずれでも指定できます。dを省略するとd＝1が指定されたことになり、配列要素を全てサーチします。

[使用例]
```
10 DIM A%(10)
20 FOR N=0 TO 10
30 A%(N)=N+100
40 NEXT N
50 NN=SEARCH(A%,105)
60 PRINT NN,A%(NN)
>RUN[Enter]
5           105
```

SGN [省略形]SG.

[書式]SGN(a)

( )内の数値の符号を調べます。値が正のときは1を返します。値が0のときは0を負のときは－1を返します。
( )の中には、定数、変数、配列、数式、関数(いずれも文字型を除く)が記述できます。

SID [省略形]SI.

[書式]SID(a)

正弦(sin)を計算します。
( )の中には、定数、変数、配列、数式、関数(いずれも文字型を除く)が記述できます。( )内の値の単位は度です。

[使用例]
```
10 FOR A=0 TO 90 STEP 10
20 S=SID(A),C=COD(A)
30 PRINT A,S,C,:IF C<>0 THEN PRINT S/C ELSE PRINT "-"
40 NEXT A

>RUN[Enter]
0               0               1               0
10          0.173648        0.984808        0.176327
20          0.34202         0.939693        0.36397
30          0.5             0.866025        0.57735
40          0.642787        0.766044        0.839099
50          0.766044        0.642788        1.19175
60          0.866025        0.5             1.73205
70          0.939692        0.34202         2.74748
80          0.984808        0.173648        5.67128
90          1               0               -
```

SIN [省略形]なし

[書式]SIN(a)

正弦(sin)を計算します。
( )の中には、定数、変数、配列、数式、関数(いずれも文字型を除く)が記述できます。( )内の値の単位はラジアンです。

[使用例]
```
10 FOR A=0 TO 90 STEP 10
20 D=A*PI/180
```

```
30 S=SIN(D),C=COS(D)
40 PRINT A,S,C,:IF C<>0 THEN PRINT TAN(D) ELSE PRINT "-"
50 NEXT A

>RUN[Enter]
0               0               1               0
10          0.173648        0.984808        0.176327
20          0.34202         0.939693        0.36397
30          0.5             0.866025        0.57735
40          0.642787        0.766044        0.839099
50          0.766044        0.642788        1.19175
60          0.866025        0.5             1.73205
70          0.939692        0.34202         2.74748
80          0.984808        0.173648        5.67128
90          1               0               -
```

SPACE＄ ［省略形］SP. SPA. SPAC. SPACE.

［書式］SPACE＄(n)

任意の桁数の空白を与えます。
( )の中には、定数、変数、配列、数式、関数(いずれも文字型を除く)が記述できますが、その値は1～39の範囲の整数に限ります。

```
[使用例]
10 FOR N=1 TO 10
20 A$="*"+SPACE$(N)+"*"
30 PRINT A$
40 NEXT N

>RUN[Enter]
* *
*  *
*   *
*    *
*     *
*      *
*       *
*        *
*         *
*          *
```

SQR ［省略形］SQ.

［書式］SQR(a)

平方根を計算します。
( )の中には、定数、変数、配列、数式、関数(いずれも文字型を除く)が記述できますが、その値は0または正の数に限ります(負数はエラーになります)。

```
[使用例]
10 FOR A=0 TO 10
20 PRINT A,SQR(A)
30 NEXT A

>RUN[Enter]
0               0
1           1
```

```
2           1.41421
3           1.73205
4           2
5           2.23607
6           2.44949
7           2.64575
8           2.82843
9           3
10          3.16228
```

STR＄ ［省略形］STR.

［書式］STR＄(a)

（ ）内の値を示す文字列を与えます。
（ ）の中には、定数、変数、配列、数式、関数(いずれも文字型を除く)が記述できます。VAL関数と逆の働きをします。

［使用例］

```
10 X=12.34
20 A$=STR$(X/3)
30 PRINT A$

>RUN[Enter]
4.11333
```

TAN ［省略形］TA.

［書式］TAN(a)

正接(tan)を計算します。
（ ）の中には、定数、変数、配列、数式、関数(いずれも文字型を除く)が記述できます。（ ）内の値の単位はラジアンです。
なお、度の単位の tan を求めるには SID(X)/COD(X) を計算するか、TAN(X*PI/180) を計算します。

［使用例］

```
10 FOR A=0 TO 90 STEP 10
20 D=A*PI/180
30 S=SIN(D),C=COS(D)
40 PRINT A,S,C,:IF C<>0 THEN PRINT TAN(D) ELSE PRINT "-"
50 NEXT A

>RUN[Enter]
0           0           1           0
10          0.173648    0.984808    0.176327
20          0.34202     0.939693    0.36397
30          0.5         0.866025    0.57735
40          0.642787    0.766044    0.839099
50          0.766044    0.642788    1.19175
60          0.866025    0.5         1.73205
70          0.939692    0.34202     2.74748
80          0.984808    0.173648    5.67128
90          1           0           -
```

TIME＄ ［省略形］TI. TIM. TIME.

現在の時刻(時、分、秒)を持つシステム変数です。

TIME＄は8桁の文字型変数で、HH:MM:SSの形をしています。HHは00～23、MMとSSは00～59の範囲の整数です。
TIME＄はWindowsシステムの時刻データを引用しています。
この値はPRINT文やLET文などで常時参照することができます。
TIME＄は読み取り専用です。TIMS＄に値を書き込むことはできません。

[使用例]
```
10 PRINT "START",TIME$
```

VAL [省略形]V. VA.

[書式]VAL(文字列)

数字の文字列を、計算できる数値に変換します。STR＄と逆の働きをします。
12345という数値はプログラムのなかで計算したり変数に代入することができますが、"12345"というように文字型で表現したものはこのままでは計算に利用することはできません。VAL関数は文字列が示している数値を計算できる値に変換する働きをします。
下はTIME＄の内容を計算できる値に変換する例です

[使用例]
```
10 T$=TIME$
20 H=VAL(LEFT$(T$,2))
30 M=VAL(MID$(T$,4,2))
40 S=VAL(RIGHT$(T$,2))
50 PRINT T$,H,M,S
```

XOR [省略形]X. XO.

[書式]XOR(a, b)

8ビットの2数の排他的論理和(XOR)を計算します。マシン語のXOR命令と同じ働きをします。
排他的論理和とは2つの数をビット毎に比較し共に1の場合及び共に0の場合には結果のそのビットを0にし、一方が1で他方が0のときは結果のそのビットを1にする演算です。
( )の中には、定数、変数、配列、数式、関数(いずれも文字型を除く)が記述できますが、その値は0～255の範囲の整数でなければ、正しい結果は得られません。
またこのXOR関数の取り得る値の範囲も0～255(＄00～＄FF)です。

[使用例]
```
10 A=$37
20 B=XOR(A,$0F)
30 PRINT HEX$(A),BI$(A)
40 PRINT HEX$(B),BI$(B)
>RUN[Enter]
37          00110111
38          00111000
```

[参考]＄FFとのXORをとることによりその値を反転させることができます。

[使用例]
```
10 A=$37
20 B=XOR(A,$FF)
30 PRINT HEX$(A),BI$(A)
40 PRINT HEX$(B),BI$(B)
>RUN[Enter]
37            00110111
C8            11001000
```
つまりXOR(××,＄FF)はマシン語のCPL命令と同じ働きをすることになります。

7章 RS232C制御

ND80ZⅢボードの9PinD－SUBコネクタ（CN1）にRS232Cケーブルを接続することで他のパソコンや測定器などとデータの送受信を行うことができます。
実際にRS232Cの送受信を行うのは、ND80ZⅢボード上のPIC18F14K50です。
BASICの命令を使うことで、RS232C送信、受信を行うプログラムを簡単に書いて実行することができます。

1. 準備

ボーレートの設定、ケーブルの接続、その他の注意事項などについては、「ND80ZⅢ取扱説明書」Ⅷ. RS232Cインターフェース を参照してください。

2. 送信命令 WRITE ＃1

[使用例]
10 WRITE ＃1, ”ABC”, A$, ……

＃1はRS232Cの回線番号で＃1を指定します。
第2パラメータ以降のデータは文字定数か文字変数（または文字型配列）でなければなりません。数値を送信したい場合にはその数値をSTR＄関数で文字型に換えてから送信します。 区切マークの ,（カンマ）の代わりに ;（セミコロン）を使うことができます。
上記例のように複数の文字型データを ,（カンマ）または ;（セミコロン）で区切って記述した場合には、各データはひとつのデータとして送信されます。
最後に ,（カンマ）または ;（セミコロン）がある場合には、データの終わりを示すコード（0D・0A）は送信されません。
,（カンマ）または ;（セミコロン）も無しで終わると、データのあとに、0D・0Aコードが送信されます。
例:
WRITE ＃1,”ABC”,”XYZ”,
送信されるデータ（16進数で示す）……41・42・43・58・59・5A
WRITE ＃1,”ABC”,”XYZ”
送信されるデータ……41・42・43・58・59・5A・0D・0A

[送信プログラム例]

```
10 PRINT "write start"
20 FOR A=0 TO 20
30 WRITE #1,STR$(A);",";
40 NEXT A
50 WRITE #1,CHR$($0D);CHR$($0A);"end"
```

[受信（相手側）データ]

```
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,
end
```

3. 受信命令 READ ＃1

[使用例]
10 READ ＃1 ,A＄ ,X%

READ＃命令の実行の有無やタイミングには関係なく、データが受信される度に割り込みによってPIC18F14K50の受信バッファ（256バイト）にたくわえられていきます。
READ＃命令が実行されると、受信バッファの先頭から最初の0D・0Aコードまでが文字変数に読みこまれます。再びREAD＃命令が実行されると先に読んだ0D・0Aコードの次のデータから次の0D・0Aコードまでが文字変数に読みこまれます。なおいずれの場合でも0D・0Aコードそのものは文字変数には読みこまれません。
READ＃命令が実行された時にPIC18F14K50の受信バッファに0D・0Aコードで区切られたデータが複数個格納されている場合には、READ＃命令を実行する度に順にデータが文字変数に読みこまれることになります。そして整数型変数にはその時のデータ長（桁数）が入ります。データ長が40桁以上の場合にはそのデータを読み込む1回目のREAD＃命令で文字変数に最初の39桁が格納され、整数型変数には40が格納されます。再びREAD＃命令を実行することでデータの残りを読み込むことができます。

READ＃命令が実行された時に受信バッファが空の場合には文字変数には1桁の空白が格納され、整数型変数には0が格納されます。
READ＃命令が実行された時に、データが受信の途中で何桁かは受信されているがまだ0D・0Aコードまでは受信されていない場合には、文字変数には受信途中のデータが格納され、整数型変数には－1が格納されます。受信バッファにデータがある場合にはそれが0D・0Aコードがまだ受信されていない途中のデータであっても、READ＃命令によって文字変数に読みこまれますが、整数型変数が－1かどうかを調べることで、受信途中の半端なデータか0D・0Aコードで区切られたデータかを区別することができます。

[プログラム例]
```
10 KETA%=0
20 READ #1,RDATA$,KETA%
30 IF KETA%<=0 GOTO 20
40 PRINT RDATA$;
50 IF RDATA$="END" THEN STOP
60 IF KETA%=40 GOTO 10 ELSE PRINT:GOTO 10
```

行番号50で転送データの終わりを”END”で判定していますが、これは例であって、データの最後に”END”を送信しなければならない、ということではありません。

[注記1]最初にREAD ＃1で受信を開始する前に、必ず整数型変数を0クリアしてください（行番号10参照）。
またOD・0Aコードを受け取るか39バイトを受信して文字変数のデータが確定したあと、新しいデータを読み込む前にも整数型編買うを0クリアしてください（行番号60参照）。このようにしないと正しく受信されません。
[注記2]RS232C通信のボーレートは、ND80ZⅢのディップスイッチDS1の№2、№3 によって決定されます。データは8ビット、ストップビット長1、パリティ無しです。詳しくは「ND80ZⅢ取扱説明書」Ⅷ. RS232Cインターフェースを参照してください。
[注記3]PIC18F14K50はRS232Cデータを8ビットのバイナリデータとして扱いますが、READ ＃命令による受信データは、ASCII文字データとして扱います。文字データの区切りコードは0D0Aです。受信した文字データは0D0Aコードを除いて文字変数に格納されたあと、PRINT文などで使われます。ですから一般にコマンドプロンプトで画面に表示できない、00～1Fなどのコードを受信すると、その結果をPRINT文で表示したときに、異常な表示になったり、システムが暴走する可能性がありますから、注意してください。

4. INPUT＄()

PIC18F14K50のデータバッファにたくわえられた受信データを読み出すのにREAD＃命令の他にINPUT＄関数を使うことができます。
READ＃命令はデータの区切りマーク（0D・0Aコード）までのひとまとまりのデータを読み取るのに対して、INPUT＄関数は区切りマークに関係なく、バッファの先頭から任意の桁数の文字を読み出します。READ＃命令は文字コードしか読み取りませんが、INPUT＄関数は0D、0Aコードも1桁の文字として読み取ります。

[書式]INPUT＄(k, ＃n)……k＝1～39、n＝1～2

kは読み取る桁数で1～39の範囲の整数で、定数の他、変数、配列、式を書くこともできます。nはRS232Cの回線番号で＃1を指定します。
なお＃1を省略するとキーボードからの入力データを読み込む関数になります。
INPUT＄はREAD＃と違って、RS232C受信データバッファが空の時は指定桁数のデータが受信されるまで待ち続けます。

[使用例]
```
    10 ON ERROR GOTO 60
    20 PRINT "start"
    30 A$=INPUT$(5,#1)
    40 PRINT A$;
    50 GOTO 30
    60 PRINT "err line=";ERL,"err=";ERR,"232c err=";HEX$(PEEK($FFB8),2)
>r.
start
ABCXYZ
123
```

```
abcdefghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ
/end
END
```

5. エラーコード

READ #命令やINPUT$関数の実行中に、受信エラーが発生すると、エラーコード73が表示されます。
READ #の場合には、READ #で使っている整数型変数にエラーの種類を示す値がセットされます。また受信エラーを記録する特殊なメモリアドレス$FFB8にも、同じ値がセットされます。
INPUT$の場合には、メモリアドレス$FFB8に、エラーの種類を示す値がセットされます。

[受信エラーの種類]
01 受信バッファオーバーフロー
READ #またはINPUT$命令が実行されるよりも前に256バイト以上のデータが送信されてきて、それを受信してしまったか、BASICのプログラムで、なにか別の作業、たとえばINPUT文やあるいはIF GOTO文などの実行によって、READ #文やINPUT$文が実行されない間に256バイト以上のデータが送られてきてしまった場合にこのエラーが発生します。

02 オーバーランエラー
通常は発生しません。一時的にPIC18F14K50がビジーになるなど、異常な状態が発生したことが考えられます。

04 フレーミングエラー
ディップスイッチで設定しているボーレートと異なるボーレートのデータが送られてきた場合に発生します。

[注記]受信エラーによってBASICプログラムがブレイクしたあと、再度受信プログラムを実行すると、再び同じエラーが発生する場合があります。
また、エラーが発生したあとの最初の受信命令の実行によって、正しくないデータが読み込まれる場合があります。

8章　マシン語プログラムの作成

リモート接続プログラム＋ZB3BASICシステムのメリットは8ビットCPUボードの制御をBASICプログラムで行うことができる点にあります。
しかし処理によってはマシン語サブルーチンと併用したいときも出て来ます。リモート接続プログラム＋ZB3BASICはマシン語プログラムの開発にも適しています。

1．Z80アセンブラ

マシン語プログラムの作成にはアセンブラが便利です。ND80ZⅢ組立キットにはWindowsパソコン上で使えるZ80アセンブラ（ZASM．COM）とZ80逆アセンブラ（ZDAS．COM）が含まれています。マシン語サブルーチンの作成にはZASM．COMを使えば簡単にZ80マシン語プログラムを作成することができます。ZASM．COMはリモート接続＋ZB3BASICを起動して使うのではなく、MSDOSプロンプト（コマンドプロンプト）上で単独に使います。
ZB3BASICを起動させた状態で別のMSDOSプロンプト（コマンドプロンプト）を開いてそこでZASM．COMを実行し、マシン語プログラムを作成してからZB3BASICのMSDOSプロンプト（コマンドプロンプト）に戻って、いま作成したマシン語プログラムのバイナリファイルをロードする、といった使い方もできます。
Z80アセンブラの使い方については「Z80アセンブラZASM操作説明書」を参照してください。

2．マシン語モニタコマンド

少し長いマシン語のプログラムを作成しようとした場合に、今やアセンブラは欠かせません。
しかしアセンブラを使うためにはまずソースプログラムを作成し、次にそのソースプログラムをZASMでバイナリファイルに翻訳し、最後に／LDコマンドでメモリにロードする、という手続きをする必要があります。
もし、ほんとに短いマシン語プログラムでしたら、ND80ZⅢのキーボードから16進コードで直接メモリに書いた方が早いかもしれません。
そのような場合には、ND80ZⅢの16進コード入力と同じように、マシン語コードをそのまま入力するCMコマンドはとても便利です。
また数バイトではなくて数十バイトのメモリの内容を確認するにはDMコマンドが便利です。
アセンブラを利用し出作成したマシン語プログラムをデバッグするにはマシン語モニタコマンドの助けが必要です。BP／RT（ブレークポイント／リターンコマンド）はマシン語プログラムのデバッグには欠かせない機能です。
マシン語モニタコマンドの詳細については10章を参照してください。

3．逆アセンブラ

これも有りがたい機能です。マシン語プログラムのソースプログラムがどこかへいってしまって見つからないという場合に威力を発揮します。
CMコマンドやDMコマンドを使えば書いてある内容を16進コードで確認することはできます。しかしその一部を変更するとか、別のシステムに移植するとかになると、マシン語コマンドでそれを行うのは至難の技です。
ND80ZⅢ組立キットに附属しているZ80逆アセンブラZDAS．COMは、マシン語のバイナリファイルを読みこんで、それを再アセンブル可能なZ80アセンブラソースプログラムにまで作り上げてしまう、本格的な逆アセンブラです。
Z80逆アセンブラについては「Z80逆アセンブラZDAS操作説明書」を参照してください。

4．マシン語サブルーチン

マシン語のプログラムは、JPコマンドを使って、単独で実行させることができます。
また、ZB3BASICのサブルーチンとして、BASICの実行中にコールして実行させることもできます。
BASICのサブルーチンとして使うのではなくて、マシン語プログラムとして単独で使う場合には、BASICのシステムワークエリア以外のどこにでもマシン語のプログラムを置くことができます。
BASICのシステムワークエリアとしては、E000～FFFFのアドレスがリザーブされていますから、それ以外のRAMエリア、8000～DFFFなら、どこに置いても支障はありません。

4．1　マシン語サブルーチンのためのエリア確保

マシン語のプログラムをBASICプログラムからコールして使う場合には、マシン語のプログラム（サブルーチン）はBASICプログラムや変数エリアを避けて置く必要があります。
BASICプログラムが置かれているメモリアドレスやそのBASICプログラムで使っている変数エリアのアドレスは、HELPコマンドを入力すれば表示されます。
BASICプログラムは8004番地から後ろのアドレスに向かって置かれていきます。

BASICの変数エリアはDFFF番地から前に向かって割り当てられていきます。
ですから、マシン語のサブルーチンは、その間のプログラムでも変数エリアでも使われていないメモリアドレスに置くことができます。
しかしプログラムが追加変更されていくうちに、最初に置いていたマシン語サブルーチンのエリアと重なってしまうかもしれません。
また大きな配列変数などを追加したりすることによっても、マシン語のサブルーチンを壊してしまうことになるかもしれません。
そのようなことを避けるためと、もうひとつ後で説明する別の理由から、マシン語のサブルーチンはBASICプログラムよりも前に置くようにしてください。
BASICプログラムよりも前にマシン語のサブルーチンを置くためには、マシン語サブルーチンのためのエリアを確保する必要があります。
マシン語サブルーチンのためのエリアを確保するためには、2つの方法があります。
NEWコマンドと／LOADコマンドです。

4．2　NEWコマンド

BASICのプログラムを書き始める前に、NEWコマンドを実行します。
NEWコマンドはBASICプログラムをクリアするとともに、BASICプログラムを書き始めるアドレスを指定するコマンドです。
パラメータを付けないで、ただNEW[Enter]と入力した場合には、
NEW，8004[Enter]と入力したことになります。BASICプログラムを書くと8004番地から順に格納されていきます。
この場合には、マシン語のサブルーチンのためのエリアは確保されません。
リモート接続プログラムから、[Z]キーを入力して、ZB3BASICを起動した直後は、NEW，8004が実行された状態になります。
マシン語サブルーチンのためのエリアを確保するためには、必要と思われるメモリアドレス分をずらして、NEWコマンドで設定します。
たとえば　NEW，8800[Enter]とすれば、8004～87FFの範囲のメモリがマシン語サブルーチンのために確保されます。
なお8000～8003はZB3BASICのための制御情報のためにシステムが使いますから、ここにはマシン語サブルーチンなどを書かないようにしてください。

4．3　／LOADコマンド

マシン語サブルーチンのためのエリアを確保する、もう1つの方法は／LOADコマンドを使ってBASICプログラムをロードするときに、ロード先のメモリアドレスを指定する方法です。
／LOAD　TEST．TXT[Enter]
のように、アドレス指定をしない／LOADコマンドを実行すると、BASICプログラムは、8004番地からロードされます。
／LOADコマンドに先だって、NEW，xxxxが実行されていても、その設定はクリアされます。また／LOADの入力前に書かれていたBASICプログラムもクリアされます。
／LOAD　TEST．TXT，9000[Enter]とすれば、8004～8FFFの範囲のメモリがマシン語サブルーチンのために確保されます。

4．4　マシン語プログラムの終わり方

マシン語プログラムの最後が正しく終わるようになっていないと、ZB3BASICシステムがハングアップしてキー入力を受け付けなくなったり、暴走してコマンドプロンプト画面に異常な表示が連続して出されたりします。
そのような状態では、[Ctrl][C]で強制終了するか、コマンドプロンプトの右上の[x]をクリックしてコマンドプロンプトを閉じるしかありません。ND80Z Ⅲもリセットする必要があります。
マシン語プログラムの最後がエンドレスで無限ループにしておくこともできますが、その場合にプログラムを終了するには[Ctrl][C]で強制終了することになります。
マシン語プログラムの最後にZB3BASICのリエントリアドレス、1033へのジャンプ命令を書いておくと、マシン語プログラムの終了後は、＞が表示されて、通常のZB3BASICの入力画面に戻ることができます。
[例]マシン語プログラムの最後に下のコードを書いておきます。
C30010　JP　＄1033

4．5　マシン語プログラムの実行

マシン語プログラムを実行するには、マシン語モニタコマンドのJPコマンドを使います。
マシン語プログラムのスタートアドレスが、8004番地のとき、下のように入力します。

＞JP　8004[Enter]

[参考]マシン語サブルーチンとして、最後をC9(RET)で終わるようにしたプログラムを、BASICのUSR関数を使ってダイレクトモードで実行させることもできます。

＞USR(＄8004)[Enter]

4．6　マシン語サブルーチンの終わり方

ZB3BASICのプログラムの実行中にコールする、マシン語サブルーチンの最後はRET命令でなければなりません。
最後は必ず下のように書きます。
C9　RET

[注意1]マシン語サブルーチンの中でCALL命令や、PUSH命令、POP命令を使うときは、マシン語サブルーチンが開始されるときと、終了するときとでスタックが食い違わないように注意してください。もしもスタックに食い違いがあると、BASICに戻れなくなったり、暴走してしまうことがあります。
[注意2]マシン語サブルーチンの実行中は[Ctrl][B]によるブレイクはできません。

4．7　マシン語サブルーチンの実行

BASICプログラムの中で、マシン語サブルーチンをコールするには、USR()関数を使います。
たとえばアドレス8004から書かれているマシン語サブルーチンをコールするには、BASICプログラムの中で、USR(＄8004)という文を書きます。

[使用例1]
最初にNEWコマンドでマシン語サブルーチンのエリアを確保します。
```
>new,9000
```
CMコマンドで下のようなマシン語サブルーチンを書きます。
```
8004 2A40F4  LD HL,($F440)
8007 23      INC HL
8008 2240F4  LD ($F440),HL
800B C9      RET
```
F440、F441はBASICの整数型変数A%の固定アドレスです。
```
>cm 8004
8004 12-2a
8005 34-40
8006 56-f4
8007 78-23
8008 40-22
8009 F4-40
800A C9-f4
800B 00-c9
800C 7E-
```
次に以下のBASICプログラムを書きます。先にNEW, 9000が実行されていますから、新しく書くプログラムはアドレス9000から格納されていきます。
```
>10a%=12345
>20print a%
>30for i%=0 to 10
>40usr($8004)
>50print a%,
>60next i%
>
```
プログラムを確認してみます。LIST[Enter]を省略して[.][Enter]とすることもできます。
```
>.
    10 A%=12345
```

```
    20 PRINT A%
    30 FOR I%=0 TO 10
    40 USR($8004)
    50 PRINT A%,
    60 NEXT I%
```
プログラムを実行します。RUN[Enter]を省略して[r.][Enter]とすることもできます。
```
>r.
12345
12346         12347         12348         12349         12350         12351
12352         12353         12354         12355         12356
```
8004番地のマシン語サブルーチンをコールする度に、A%の値が＋1されていきます。

[注記1]USR()関数はマシン語サブルーチンをコールするときに、システムで使っているレジスタの値を退避させますから、マシン語サブルーチンでレジスタを退避させる必要はありません。
[注記2]マシン語サブルーチンの中で使うレジスタの値とBASICの変数の値を共有するために、整数型の固定アドレス変数A%～Z%（アドレスF440～F473）、文字型の固定アドレス変数A＄～H＄（アドレスF300～F43F）を使うことができます。

[使用例2]
上の[使用例1]と同じようにして、次のマシン語サブルーチンとBASICプログラムをそれぞれ書いてください。

[マシン語サブルーチン]
```
8004 CD2306  CALL $0623
8007 3240F4  LD ($F440),A
800A C9      RET
```

[BASICプログラム]
```
    10 A%=0
    20 USR($8004)
    30 IF A%=255 GOTO 20
    40 PRINT A%,
    50 GOTO 20
```

マシン語サブルーチンでコールしているアドレス0623はND80Zモニタのキー入力ルーチンです。
ND80ZⅢのキーボードが押されてると、そのキーに対応するキーコードがAレジスタに入ります。キーが押されていないときは、AレジスタにはFFが入れられます。
下はプログラムの実行結果です。
ND80ZⅢのキーを押すたびにそのキーのキーコードが表示されます。
```
>r.
0             1             2             3             21
```

4．8　マシン語プログラムの保存、読み込み

　マシン語プログラム、マシン語サブルーチンをファイル名をつけて保存するには／SVコマンドを使います。
　また／SVで保存したファイルやZASMで作成したバイナリファイルをメモリに読み込むには、／LDコマンドを使います。
　マシン語プログラム、マシン語サブルーチンは作成されたときと異なるアドレスに／LDで読み込んでも正しく実行することはできません。ただのバイナリデータとしてなら、作成、保存されたときと別のアドレスに読み込んでも支障はありません。
　／SV、／LDについては、9章　1．、2．を参照してください。

4．9　マシン語サブルーチンとBASICプログラムと合わせて保存、または読み込む

　マシン語プログラムのためのエリアをBASICプログラムエリアよりも前のアドレスに確保して作成したマシン語サブルーチンと、BASICプログラムは一括してバイナリイメージでファイルに保存することができます。またそのようにして保存されたファイルを読み込んで、BASICプログラムを実行可能な形に戻すことができます。
　詳細については9章5．、6．を参照してください。

9章　プログラムの保存

　ZB3BASICにエントリして作成したマシン語のプログラム、データやBASICのプログラムはZB3BASICを終了すれば消えてしまいます。実際にはボタン電池でRAMのバックアップをしているので消えてしまうことはありません。しかしその上から別のプログラムで上書きすれば無くなってしまいます。
　マシン語プログラム、データやBASICプログラムはファイル名をつけてハードディスクに保存することができます。作成されるファイルはMSDOSの規則にしたがってつくられますから、MSDOSやWindowsのツールで参照したり編集することが可能です。
　マシン語プログラム、データはバイナリ形式のファイルになります。テキストエディタで見ることはできませんが、MSDOSのDEBUGコマンドで編集することができます。
　BASICプログラムはテキスト形式で保存されますから、Notepad（メモ帳）などのテキストエディタで参照、修正、プリンタへの出力などが行えます。
　プログラム、データをSAVE、LOADするためのコマンドはZB3BASICにエントリした状態ですぐに使用することができます。

1．　マシン語プログラム（およびデータ）の保存

　／SVコマンドを使います。

[書式]／SV　ファイルネーム，aaaa，bbbb

　指定アドレス範囲（aaaa～bbbb）のマシン語プログラム（データ）をファイル名をつけてハードディスクに保存します。aaaa，bbbbは4桁の16進数でaaaa＜＝bbbbです。ファイル名はMSDOSの規則に従います。マシン語プログラムについてはZ80アセンブラによってバイナリファイルを作成することができますから、このコマンドを使うことは少ないかもしれません。CMコマンドなどで一部を変更したり、あるメモリ範囲の内容をそのまま保存する場合には有効な手段になります。
　／SVコマンドで作成したバイナリファイルを逆アセンブラ（ZDAS．COM）にかけてアセンブラソースプログラムを作成することができます。

[使用例]
>/sv test_sub.bin,8004,8365[Enter]

[注記]
　ファイル名は名前部が8桁以内の半角英数字および－_で拡張子は3桁以内です。拡張子は任意です。付けなくても構いません。ファイル名にはドライブ名やディレクトリ名も付加することができます。
例：　>/sv E:¥basic¥testpro.rom,8000,8fff[Enter]

2．　マシン語プログラム（およびデータ）のファイルからの読出し

　1．で作成したバイナリファイルをRAMの指定アドレスに読みこみます。／LDコマンドを使います。

[書式]／LD　ファイルネーム，aaaa

　ファイル名にはドライブ名やディレクトリ名も付加することができます。aaaaは4桁の16進数でLOAD先のRAMアドレスを示します。aaaaを省略することはできません。ファイルがみつからないときは　FILE　NOT　FOUND　と表示されます。

[使用例]
>/ld test_sub.bin,8004 [Enter]

3．　BASICプログラムの保存

　ZB3BASICにエントリして作成、編集したBASICプログラムは／SAVEコマンドでハードディスクに保存することができます。／SAVEコマンドについては第4章　BASICコマンド　でも説明していますのでそちらも参照してください。

[書式]／SAVE　ファイルネーム

ファイルネームがファイル名のみの場合にはZB3BASICが存在するディレクトリにSAVEされます。同じファイル名があると上書きされます。
ファイル名にドライブ名やディレクトリ名を含めて記述することで、別のドライブやディレクトリにSAVEすることができます（使用例③）。
現在のテキストエリアのアドレス情報は保存されません。／LOAD時に新しく決定されます。
／SAVEコマンドではBASICシステムによって1行ずつBASIC内部コードをテキスト形式に変換していきます。このときプログラムは1行ずつ画面にリスト表示されます。

[使用例①]
＞／SAVE　TEST. TXT[Enter]
[使用例②]
＞／SAVE　MIHON[Enter]　　……拡張子はなくてもよい。
[使用例③]
＞／SAVE　E:￥BASIC￥TESTPRO. BAS[Enter]
[注記1]
使用例③のように別のドライブや別のディレクトリにSAVEすることもできます（上の①②例ではZB3BASICの存在するディレクトリにSAVEされます）。

4.　BASICプログラムのファイルからの読出し

テキスト形式で保存されたファイルをBASICプログラムとしてZBKボードのRAMエリアに読みこむことができます。／LOADコマンドを使います。

[書式]
／LOAD　ファイルネーム, aaaa

ファイルネームがファイル名のみの場合にはZB3BASICが存在するディレクトリからLOADします。ファイル名にドライブ名やディレクトリ名を含めて記述することで、別のドライブやディレクトリにあるファイルをLOADすることができます（使用例③）。ファイルがみつからないときは　FILE　NOT　FOUND　と表示されます。
aaaaは4桁の16進数で、省略することもできます。省略した場合には現在のテキストエリアの先頭からLOADされます。aaaaをつけるとaaaa番地からLOADされます。
／SAVEでファイルを作成したときのテキストエリアのアドレス情報は保存されません。／LOAD時に新しく決定されます。
／LOADコマンドでテキスト形式のファイルが読み込まれるとき、BASICシステムによって1行ずつBASIC内部コードに変換してRAMに格納していきます。このときプログラムは1行ずつ画面にリスト表示されます。内部コードに変換するときに文法エラーが見つかるとそこでエラーコードが表示されLOAD作業は打ち切られます。
メモ帳などのテキストエディタで作成したプログラムも文法的に正しければ／SAVEで保存されたファイルと同じようにLOADすることができます。

[注意1]
LOADコマンドの動作は、NEW　＋　／LOAD、またはNEW　aaaa　＋　／LOAD、というように必ずNEWコマンドの動作を伴っています（LOAD前にRAMに存在したプログラムは失われます）。
[注意2]
メモ帳（Notepad）は問題ありませんが、WriteやWordでは通常はそれぞれの形式でファイルが作成されます。保存するときにテキスト形式を指定してもゴミが混じる場合があります。Writeなどで作成したファイルがうまくLOADできないときは、一度メモ帳（Notepad）で開いてゴミを削除して、メモ帳で保存してからLOADしてみてください。また全角英数モードで作成されたファイルも読みこめません（異常な表示になります）。必ず半角英数モードで作成してください。

[使用例①]
＞／LOAD　TEST. TXT[Enter]
[使用例②]
＞／LOAD　MIHON[Enter]　　……テキスト形式のファイルなら拡張子のついていないファイルでもよい
[使用例③]
＞／LOAD　E:￥BASIC￥TESTPRO. BAS, 9000[Enter]
この例のように別のドライブや別のディレクトリにあるファイルをLOADすることもできます。上の①②例ではプログラムは現在のテキストエリアの先頭からLOADされます。③では9000番地からLOADされます。
[注意3]
ファイル名は名前部分が半角英数8文字以内で拡張子は3文字以内に限られます。

［注意4］
テキスト形式（拡張子の種類に関わらず内容がテキスト形式になっている）以外のファイルをLOADするとZB3BASICシステムが暴走してハングアップすることがあります。テキスト形式のファイルでも半角英数字以外の文字が使用してあると、やはり暴走してしまいます。

5. マシン語サブルーチンとBASICプログラムの一括保存

マシン語プログラムはZ80アセンブラZASM. COMで作成して、ソースプログラムは□□□. TXT、マシン語コードは□□□. BINというファイル名で保存します。
BASICプログラムは△△△. TXTのようにそれらとはまた別に保存します。テキストエディタで編集を行うためにはこの形でなければいけませんが、プログラムとして完成した時点で保存を考えると、BASICプログラムだけではなくてマシン語サブルーチンと一緒に働くプログラムの場合には、別々に保存するのは面倒でもあります。
この場合、BASICブログラムとマシン語プログラムを一括してバイナリ形式で保存できると便利です。
マシン語プログラムだけの場合と同じように／SVコマントを使いますが、そのままではあとでBASICプログラムをLIST表示可能な状態に戻すことはできません。
／SVコマンドに先だってBROMコマンドでBASICテキストの情報を作成します。BROMについては「第10章　マシン語モニタコマンドおよびシステム操作コマンド」で説明していますから、そちらも参照してください。

［書式］／SV　ファイルネーム，8000，bbbb

［使用例］
＞BROM［Enter］
8500－9732
＞／SV　TESTPRO. ROM，8000，9732［Enter］

［注記］
ファイル名にはドライブ名やディレクトリ名も付加することができます。
例：　／SV　E:￥BASIC￥TESTPRO. ROM，4000，7FFF
［注意1］
BROMコマンド実行後に、BASICプログラムとマシン語サブルーチンを一括保存するときは、／SVの先頭アドレスは必ず8000にします。
先にBROMを実行しておなかいと、／LD作業でBASICプログラムを再生させることができません。
bbbbはBROMの実行により表示される2番目のアドレス値にします。使用例で／SVのアドレス指定は8500，9732ではなくて8000，9732になる点に注意してください。
［注意2］
このようにして保存したファイルにはBASICプログラムも含まれていますがバイナリイメージで保存しているため、Notepadなどのテキストエディタで参照することはできません。

6. マシン語サブルーチンとBASICプログラムを一括保存したファイルからの読み込み

この場合もマシン語プログラムだけの場合と同様に／LDコマントを使いますが、LOAD後にBASICプログラムをLIST表示可能にするため、BSSETコマンドの実行が必要です。ファイルにBASICプログラムが含まれていない場合や／LDで8000以外のアドレスを指定した場合、または／SVでBROMを実行しなかったか、／SV開始アドレスを8000にしなかった場合には、BSSETコマンド入力時にHOW？が表示されます。BSSETについては「第10章　マシン語モニタコマンドおよびシステム操作コマンド」で説明していますからそちらも参照してください。

［書式］／LD　ファイルネーム，8000

［使用例］
＞／LD　E:￥BASIC￥TESTPRO. ROM，8000［Enter］
2543BYTES　LOAD
＞BSSET［Enter］
TEXT　8500－A543
ﾍﾝｽｳ　DBC9－DFFF

メモリの内容をチェックしたり、部分的にマシン語のデータを変更したり、簡単なマシン語プログラムを書いて、すぐに実行してみたい、というようなことがよくあります。
BASICの命令をダイレクトモードで実行してもそのようなことはできるのですが、ZB3BASICに含まれている、マシン語モニタコマンドを利用すると、BASICとは異なった細かい操作が、簡単に行えます。

●コマンドとパラメータの区切りについて

BASICでは命令の後ろの区切り(セパレータ)としてスペースを使っており、またアセンブラニーモニックも命令とオペランドとの区切りはスペースを使っています。
マシン後モニタコマンドはコマンドとパラメータの間の区切りとして、スペースのほかにカンマ(,)も使えますが、以下の説明の書式としては、セパレータとしてスペースを使っています。

BP／RT／CR ［省略形］B.（RT、CRの省略形は無し）

[書式1]BP aaaa
[書式2]BP 0
[書式3]BP D

ZB3BASICではマシン語プログラムの開発、デバッグに役立つようRAM上に書かれたプログラムなら、どこでも実行中にブレイクできるようブレイクポイントが1カ所設定可能です。ただしROM上のプログラムには使用できません。
（BPはbreak pointの、またRTはreturnの略です）
aaaaは4桁の16進数で、ブレイクしたいアドレスを指定します。
例えば、下のようなプログラムを実行する場合を考えます。

```
C000 2100C1        LD HL,$C100
C003 75       LOOP:LD (HL),L
C004 2C             INC L
C005 C203C0         JP NZ,LOOP
C008 C33310         JP $1033
```

このプログラムは＄C100～＄C1FFにデータ＄00～＄FFを書きこむものです。
まず、CMコマンドでC000～C00Aまでに上のマシン語プログラムを1バイトずつ入力します。
これを実行するにはJP, C000でよいのですが、この動作を確認したいと仮定します。
下のように入力してみます。

＞BP C003[Enter]
＞JP C000[Enter]

すると下のような出力が得られます。

```
A F  B C  D E  H L  A'F' B'C' D'E' H'L'  PC   SP   IX   IY  I  SZ H PNC
xx44 xxxx xxxx C100 xxxx xxxx xxxx xxxx C003 F7FE xxxx xxxx xx 01000100
```

BPコマンドでアドレスを指定したうえで、マシン語のプログラムを実行させると、そのアドレスまで実行が進んだところで、そのアドレスの命令を実行する直前でブレイクしてその時の全レジスタの内容を表示してコマンド待ちになります。
このプログラム例ではフラグレジスタ(F)、HLレジスタ、プログラムカウンタ(PC)、スタックポインタ(SP)以外は使用していないため、その他のレジスタの内容には意味はありません(誤解を避けるため、xxxx にしてあります)。
一番最後の8桁はフラグレジスタ(F)の内容をビット毎に表示したものです。
この状態は普通のコマンドモードと変わりないので、他のコマンドを受け付けることができます。例えばDMコマンドでメモリの内容を確認したりすることもできます。
必要な確認が全て済んで、以後の実行を継続したい時は

＞RT[Enter]

と入力します。以後は普通に処理が終了します。

なおBPは続けてセットすることもできます。
上の状態でブレイクしているときに、下のように入力してみて下さい。

＞BP C005[Enter]
＞RT[Enter]

再び、ブレイクしますが今度は下のようにF、HL、PCが変化していることが確認できるはずです。
A F B C D E H L A'F' B'C' D'E' H'L' PC SP IX IY I SZ H PNC
xx00 xxxx xxxx C101 xxxx xxxx xxxx xxxx C005 F7FE xxxx xxxx xx 00000000

＄C005ではフラグの内容がチェックされ、Zフラグが0FFならば再び＄C003に戻って処理が続けられます。フラグの内容を見てみると、右のフラグのビット表示は全て0になっていて、どのフラグもOFFになっていることがわかります。
そこで再びBP C003[Enter]とセットしたうえで、RT[Enter]を入力すると、今度はC003でブレイクします。
なおブレイクポイントを設定するときは、必ず命令の1バイト目のアドレスを指定する必要があります。
上の例で、C000、C003、C004、C005、C008は指定できますが、C001、C002、C006、C007、C009、C00Aを指定すると正しく実行されません。
ところが正しく指定しているにもかかわらず、指定アドレスによってはブレイクがかからないときがあります。
上のプログラム例では全部の命令が少なくとも一回は実行されますが、プログラムによっては、条件付の分岐命令があって、条件によっては全く実行されない部分がでてきます。
そのような場合にはブレイクポイントを設定したにもかかわらず、ブレイクしないで処理が終了してしまいます。
このような場合には次のことに注意して下さい。
このブレイク動作は指定アドレスの命令を、FF（RST 7）で置きかえることで実現しています。CPUはFFコードを見つけると＄0038からの命令を実行しますが、このシステムでは、その＄0038にブレイク処理ルーチンが置いてあります。
ブレイクすると、ブレイク処理が行われると共に、ブレイクアドレスに、FFの代りにもとの命令コードを戻します。ところがブレイクしないとこのFFが残ってしまいます。
例えば上の例で、BP C003[Enter]を入力したあと、すぐにCMコマンドかDMコマンドでC000～C00Aの内容を確認してみて下さい。C003にFFが入っているのが確認できます。
ブレイクしないために残ってしまったFFコードをもとの命令コードに戻すには、

＞BP 0[Enter] ……（書式2）

と入力します（もう一度C000～C00Aの内容を確認してみて下さい。C003にFFが入っていたのが元の75に戻っているはずです）。
ブレイクポイントがセットされたままになっているかどうかはっきりしない時は、下のようにBP D[Enter]と入力すればわかります。セットされているときはそのアドレスが表示されます。

＞BP D[Enter] ……（書式3）
C003 ……BPアドレスが表示される。BPがセットされていない時は何も表示されない

[注意]
リセットすると、BPがセットされたままになっていても、FFコードがプログラム内に残されたままで、BP関係の制御情報はクリアされてしまいます。そうなった後で、BP 0[Enter]を実行してもFFはもとに戻りませんし、BP D[Enter]を実行しても何も表示されません。

[**CR**コマンド]
プレイクしたアドレスから続きを実行するときにレジスタの値を変更すると都合がよい場合があります。ループしているプログラムの動作をデバッグしていて、ループカウンタの途中のある値の近くを詳しく調べたいときなどがあります。1000回のループの終りの20回くらいをチェックしたいときに980回もBP／RTを繰り返すのでは疲れてしまいます。このときループカウンタとしてメモリのワークエリアを使っていれば、ループで一旦ブレイクしループカウンタの値をCMコマンドで変更しておいて（1000を20にしてしまいます。同時にループ回数に伴って変化するワークエリアがあれば、それも合理的な値に直します。）、ループ内の次のプレイクポイントを設定してRTコマンドを実行すれば、980回を一度で済ますことができます。
CRコマンドはA、B、CなどのCPUレジスタもメモリのワークエリアと同じように変更できる機能です。
ブレイクしているときに、CR[Enter]と入力します。
ブレイク直後と同じレジスタ表示が行われます。
>CR[Enter]

```
A F  B C  D E  H L  A'F' B'C' D'E' H'L'  PC   SP   IX   IY  I  SZ H PNC
xx00 xxxx xxxx C101 xxxx xxxx xxxx xxxx C005 F7FE xxxx xxxx xx 00000000
```

↑カーソルがここに来ます。[↑][→]で変更したい値を書き換えます。スクリーンエディタが働いていますからレジスタ全部でも1レジスタでもフラグのみでも変更できます。変更する必要のないレジスタはそのままにしておきます。

例としてA＝50、BC＝1234、キャリーフラグONに書き換えてみます。

```
A F  B C  D E  H L  A'F' B'C' D'E' H'L'  PC   SP   IX   IY  I  SZ H PNC
5000 1234 xxxx C101 xxxx xxxx xxxx xxxx C005 F7FE xxxx xxxx xx  00000001
```

必要な変更が済んだら、[Enter]を入力します。カーソルはこの行の中ならどこにあっても構いません。[Enter]入力によってレジスタが更新されます。そして更新結果がすぐ下に表示されます。

```
A F  B C  D E  H L  A'F' B'C' D'E' H'L'  PC   SP   IX   IY  I  SZ H PNC
5001 1234 xxxx C101 xxxx xxxx xxxx xxxx C005 F7FE xxxx xxxx xx  00000001
>
```

[注意1]フラグレジスタの変更は右のビット表示の部分で行います。左のAFの部分を書き換えても更新はできません。ビット部分を書きかえることで、結果のAFレジスタ部分が更新されます（上例でFレジスタが00から01に変化していることを確認してください）。

[注意2]カーソルを他の行に移動してはいけません。通常の＞表示によるコマンド入力待ちのスクリーンエディタとは違うルーチンで入力しますから、他の行で[Enter]を入力するとエラーになります。

[注意3]CRはブレイク中にレジスタの値を変更し、その後にRTで中断中のプログラムを再開することを前提にしています。その他のタイミングで使用しても無意味です（システムには影響を与えません）。

[注意4]CRは繰り返して実行できますが、値の更新はCRコマンド入力直後のレジスタ表示の時のみ有効です。普通のブレイク表示の時やCRでの値の更新後の[Enter]入力により表示されたレジスタ表示行に対して更新作業はできません。もう一度更新したいときはあらためて、CRコマンドを入力します。

●以上のコマンドを下にまとめて整理しておきます。

①BP aaaa　ブレイクしたいアドレスをセットする。命令コードの1バイト目のアドレスを指定する。先の例でBP C003はよいが、BP C002やBP C001は不可。また、ROM上のプログラムにはセットできない
②BP 0　ブレイクポイントを解除する。
③BP D　ブレイクポイントがセットされているときはそのアドレスを表示する
④RT　ブレイクしていたアドレスから以後を続けて実行する。
⑤CR　ブレイク後にレジスタの値を更新してRTによる実行再開に反映させる

[注記1]すでに説明したように、このブレイクの機能は、RST 7（FF）を利用しているため、ブレイクポイントがセットされているときに、マシン語プログラムのミスなどで暴走した結果、別のアドレスでたまたまFFコードに行きつくと、ブレイクポイント以外の部分でもブレイクしてしまうことがあります。

[注記2]ユーザープログラム内でSPに新しい値を設定しない場合には、システムスタック（＄F490～＄F7FF）がそのまま使用されることになります。

BPの設定により、実行途中でブレイクするといままでユーザーが使用していたスタックをシステムが使用することになります。そのままではスタック内容が変化してしまうため、RTコマンドでユーザープログラムの実行を再開した場合に、正しく実行されなくなります。

そこでこのシステムでは、通常使われるスタックを768バイトと仮定し、＄F500～＄F7FFの内容を、ブレイク時に別のエリア（＄ED00～＄EFFF）に一時待避します。そしてRTコマンドが入力されると、この待避データをもとの＄F500～に戻した上で、ユーザープログラムにリターンします。

ブレイク時のスタックの状態を確認したい場合に、もしシステムスタックをそのまま利用している場合には、この＄ED00～＄EFFFを見るようにします（DM ED00, EFFF[Enter]を入力する）。

BROM　[省略形]BR.　BRO.

BASICプログラムの開始アドレスと終了アドレスを＄8000～＄8003の制御エリアに書込みます。

＄8000～＄8003はBASICプログラムのバイナリイメージファイルを作成するのに必要なエリアです。ここに正しい情報が書き込まれていないと、バイナリイメージファイルを読み込んでもBASICプログラムを復元することができません。

／SVコマンドでBASICプログラムとマシン語プログラムをバイナリファイルとして一括保存する場合には、／SVに先立ってBROMの実行が必要です。

BSSET ［省略形］BS.　BSS.　BSSE.

　バイナリイメージでロードしたBASICプログラムをLIST表示可能な形にします。
／SVコマンドでBASICプログラムとマシン語プログラムを一括保存したバイナリファイルもマシン語のみのバイナリファイルと同様に／LDコマンドでRAMに読み込みますが、そのままではBASICプログラムもバイナリのままなのでLIST表示させることはできません。BSSETコマンドによってLIST表示が可能になり、編集や追加、削除などができるようになります。

［使用例］
＞／LD　TESTPRO. ROM, 8000［Enter］
＞BSSET［Enter］
TEXT　8500－9543
ﾍﾝｽｳ　DBC9－DFFF

／CLOSE［省略形］なし

ZB3BASICが起動中に作成されつつあるログファイルはZB3BASIC（およびリモート接続プログラム）を終了するまではテキストエディタなどで参照することができません。
任意の時点で／CLOSE［Enter］と入力すると一旦ログファイルがクローズされ、その日時をファイルネームとするログファイルが新たに作られます。クローズされたログファイルはZB3BASICを終了しなくてもテキストエディタなどで参照することができます。

CM　［省略形］C.

［書式］CM　aaaa

指定アドレス（aaaa）のメモリの内容を1バイトずつ16進数で表示し、キーボードから入力する16進数と置き換えます。（aaaaは4桁の16進数）
マシン語で簡単なプログラムを書きたい時や、数バイト～数十バイト程度のメモリ内容を書き換えたい時などに使うと便利です。（CMはchange　memoryの略です）
　なおこの機能はRAMに対して有効です。指定したアドレスがROMであった場合でも、エラーメッセージは出ずに一応は正常に実行されますが、メモリの中味は書き換えられません。

［使用例］
①内容を確認する
＞CM　8100［Enter］
8100　2A－［Space］　指定したアドレスとそのメモリの内容が表示される。このままの内容でよい場合には
8101　87－　　スペースキーを押すと、次のアドレスが同じように続けて下に表示される

②内容を更新する
8100　2A－31　下線部のように入力する。［Enter］は必要無い。0～F以外のキーを押すと？が出てもう一
8101　87－　　度同じアドレスが表示される。データ（2桁の16進数）を入力すると、次のアドレスが同じように続けて表示される

③前のアドレスに戻る
8100　2A－31　　入力ミスなどでもう一度前のアドレスに戻りたいときは［←］を押す。前のアドレスとその
8101　87－［←］　　内容が下に表示される
8100　31－

④処理の終了
この作業を打ち切りたい時は［Enter］を押すか又は［Ctrl］［B］を入力します。するとプロンプトマーク（＞）が出てコマンド入力待ちに戻り、カーソルマークが表示されます。
この時までに行った書き換え作業はすでに実行済みなので、処理を打ち切っても取り消しはできません。

［注意］スペースキー、［←］、［Enter］、［Ctrl］［B］は1桁目のデータ入力では有効ですが、2桁目の入力では受け付けられません。？マークが表示されます。

8105　3A－_　　ここでのスペースキー、［←］、［Enter］、［Ctrl］［B］の入力は有効
8105　3A－5_　　ここでスペースキー、［←］、［Enter］、［Ctrl］［B］を入力すると？が出る

CP ［省略形］なし

［書式］CP　aaaa,bbbb,cccc

　指定アドレス範囲（aaaa～bbbb）のメモリの内容を、指定アドレス（cccc）からはじまるメモリの内容と1バイトずつ比較し、一致しない場合はそのアドレスとデータを表示します。一致した箇所は表示されません。（CPはcompareの略です）
　比較作業が完了すると、END＝bbbbと表示して、コマンドモードに戻ります。
　aaaa、bbbb，ccccは4桁の16進数で、ccccに制限はありませんが、aaaa≦bbbbでなければいけません。aaaa＞bbbbのときはHOW？メッセージが出ます。

CS ［省略形］なし

　指定範囲のメモリの内容を1バイトずつ加算して、そのトータル値を16進4桁で表示します（CS＝Check　Sum）。

［書式］CS　aaaa，bbbb

　aaaaは開始アドレス、bbbbは終了アドレスです。

DM ［省略形］D.

［書式］DM　aaaa,bbbb

　CMが1バイトずつの表示であったのに対し、DMコマンドは指定した範囲（aaaa～bbbb）のメモリ内容を1行16バイトずつ表示します。（DMはdump　memory　&　restoreの略です）
　aaaa、bbbbは4桁の16進数でaaaa≦bbbbでなければいけません。aaaa＞bbbbのときはHOW？メッセージが出ます。
　1行16バイトで表示する結果、最後の行が16バイトに満たなくなった場合でも、指定範囲を越えて16バイト分の表示が行われます。
　メモリ内容の表示はCMと同じように16進数2桁で行い、さらに各行の右側にその16進数を文字コード（JIS、ASCII）とみなして、対応する文字を一緒に表示します。（＄00～＄1F、＄80～＄9F、＄E0～＄FFのコードに対しては、ピリオド（　.　）が表示されます）

JP ［省略形］J.

［書式］JP　aaaa

　指定する16進アドレス（aaaa）にジャンプします。（JPはjumpの略です）
　ユーザーが書いたマシン語のプログラムを実行する場合に使います。そのプログラムの先頭アドレスを指定することによって、これ以後はCPUの制御はユーザープログラムに移ります。

●ユーザープログラムの終りの処理
　ユーザープログラムの最後の命令は重要です。BASICプログラムなら終りっぱなしでも構いませんが、マシン語プログラムを終りっぱなしにすると大抵は暴走して止まらなくなったりハングUPしてしまいます。
　ユーザープログラムの最後にはシステムのリエントリアドレスへのジャンプ命令を書いてください。リエントリアドレスは＄1033です。
　JP　＄1033（コード　C33310）と書いてください。
　ユーザープログラムの最後にJP　＄1033があるとユーザープログラム終了後そのままシステムのリエントリプログラムが実行されます。そこではスタックの値やその他システムの処理を継続するのに必要なワークアドレスの値が再設定されるため、ユーザーは終りっぱなしに近い感覚でプログラムを終ることができます。

［参考］USR命令
　マシン語のユーザープログラムを実行するには、このJPコマンドの他にBASICのUSR（＄aaaa）命令も使えます。
　USR（＄aaaa）もコマンドモードで実行すれば、動作はこのJPコマンドと同じになります。
　終わりの処理は、JPの場合には既に説明したように、＄1033へのジャンプ命令でなければいけませんが、このUSR命令の場合には、RET命令で終わることもできます（USR命令をBASICプログラム中で使う場合には、終わりは必ずRET命令でなければいけません）。

ただし終わりがRET命令の場合には、プログラム中でSPの値を不用意に変更してはいけません。PUSH、POPの使い方にも注意して、ユーザープログラムのスタート時のSPの値と、最後のRET命令の実行直前のSPの値が同じになるように注意する必要があります。

／LD ［省略形］なし

［書式］／LD ファイルネーム, aaaa

バイナリファイルをRAMの指定アドレスにロードします。ファイル名にはドライブ名やディレクトリ名も付加することができます。aaaaは4桁の16進数でLOAD先のRAMアドレスを示します。aaaaを省略することはできません。ファイルがみつからないときは FILE NOT FOUND と表示されます。
／SVでBASICプログラムとマシン語サブルーチンを一括して保存作成したファイルの場合にはロード後にBSSETコマンドでBASICプログラムをLIST表示可能な形にすることができます（使用例2）。

［使用例1］
＞／LD TEST__SUB. BIN, 4004［Enter］

［使用例2］
＞／LD A:￥BASIC￥TESTPRO. ROM, 8000［Enter］
＞BSSET［Enter］
TEXT 8500－9543
ﾍﾝｽｳ DBC9－DFFF

MV ［省略形］M.

［書式］MV aaaa,bbbb,cccc

指定アドレス範囲（aaaa～bbbb）のメモリの内容を、指定アドレス（cccc）からはじまるメモリエリアに転送します。（MVはmoveの略です）
この転送作業によってもとのメモリエリア（aaaa～bbbb）の内容は変化しません。
ただしもとのメモリエリアと転送先のメモリエリアに重なりがある場合には、重なった部分のメモリ内容は転送後の内容に置き代わります（メモリエリアの重なり方に制限はありません。どういう重なり方でも、正しく転送されます）。
aaaa、bbbb, ccccは4桁の16進数で、ccccに制限はありませんが、aaaa≦bbbbでなければいけません。aaaa＞bbbbのときはHOW？メッセージが出ます。
転送先のメモリエリアがROMの場合でもエラーにはなりませんが、結果的には書き込みができないため、実行されなかったのと同じことになります。
元のメモリエリアはRAMでもROMでも構いません。

［注意］このコマンドは、指定範囲のメモリ内容がプログラムであっても、ただのデータとしてそのまま転送します。
したがって例えばC000から実行される形で書かれたマシン語のプログラムをこのコマンドで他のアドレス（例えばB000）に移動させた場合、そのアドレスではそのプログラムは実行できないことに注意して下さい。
そのプログラムを他のアドレスで実行できるように移動したい場合には、一度逆アセンブラ（ZDAS. COM）にかけて、ソースプログラムを再生した後、アセンブラ（ZASM. COM）で、そのアドレスにオブジェクトプログラムを作るようにします。

SD ［省略形］なし

［書式1］SD aaaa, bbbb, hhhhhh……
［書式2］SD aaaa, bbbb, ”文字列”

指定アドレス範囲（aaaa～bbbb）のメモリをサーチし、指定データと同じ内容をみつけると、その前後の32バイトを表示します（SD＝Search Data）。
［書式1］は指定データを16進数で示したもので、データの組合せに制限はありません。
［書式2］は指定データを文字列で示します。したがってサーチできるデータは文字コードに限られます。

［使用例］
>SD 9000,97FF,C33310［Enter］

```
91C3  11 55 1F C3 CC 11 11 5B 1F CD 61 11 AF CD 18 10  .U.ﾃﾌ..[.ﾍ a.ｯﾍ..
91D3  C3 33 10 CD E0 11 7C FE FF CA BD 11 C9 21 00 00  ﾃ3.ﾍ..:..ﾊｽ.ﾉ!..

9289  00 DB 01 E6 50 C9 CD 86 12 C0 CD A2 15 CD 03 1A  .ﾛ..Pﾉﾍ .ﾀﾍ｢.ﾍ..
9299  C3 33 10 CD 6F 11 0E 06 1A 6F 13 1A 67 13 CD 5D  ﾃ3.ﾍo....o..g.ﾍ]

>SD 8000,8FFF,"ERR" [Enter]

8641  55 54 D2 45 41 44 D2 45 53 54 4F 52 45 CF 4E 20  UTﾒEADﾒESTOREﾏN
8651  45 52 52 4F 52 20 47 4F 54 4F D2 45 53 55 4D 45  ERROR GOTOﾒESUME
```

／SV　[省略形]C.

[書式]／SV　ファイルネーム, aaaa, bbbb

　指定アドレス範囲(aaaa～bbbb)のマシン語プログラム(データ)をファイル名をつけてディスクに保存します。aaaa, bbbbは4桁の16進数でaaaa＜＝bbbbです。ファイル名はMSDOSの規則に従います。BASICプログラムとマシン語サブルーチンを一括して保存する場合にはこのコマンドが必要になります(使用例2)。その場合には／SVコマンドに先立ってBROMコマンドの実行が必要です。
[使用例1]
＞／SV　TEST__SUB. BIN, 8004, 8365[Enter]

[使用例2]
＞BROM[Enter]
8500－8732
＞／SV　TESTPRO. ROM, 8000, 8732[Enter]

[注記]ファイル名は名前部が8桁以内の英数字および　－_　で拡張子は3桁以内です。拡張子は任意です。付けなくても構いません。ファイル名にはドライブ名やディレクトリ名も付加することができます。
例:　／SV　E:￥BASIC￥TESTPRO. ROM, 8000, 8FFF
[注意1]使用例2ではaaaaは必ず8000にします。先にBROMを実行しておかないと、／LD作業でBASICプログラムを再生させることができません。bbbbはBROMの実行により表示される2番目のアドレス値にします。使用例でaaaa, bbbbは8500, 8732ではなくて8000, 8732になる点に注意してください
[注意2]使用例2で保存したファイルにはBASICプログラムも含まれていますがバイナリイメージで保存しているため、Notepadなどのテキストエディタで参照することはできません。

XD　[省略形]なし

[書式1]XD　aaaa, bbbb, nnnn……, mmmm……

　指定アドレス範囲(aaaa～bbbb)のメモリをサーチし、指定データ(nnnn……)と同じ内容をみつけるとmmmm……で置きかえたあと、その前後の32バイトを表示します(XD＝Exchange　Data)。
　nnnn……, mmmm……は16進数2桁を単位として複数バイト記述できます。nnnn……, mmmm……は同じ長さでなければいけません。

[使用例]
>XD 9000,97FF,C33310,C33010[Enter]

```
91C3  11 55 1F C3 CC 11 11 5B 1F CD 61 11 AF CD 18 10  .U.ﾃﾌ..[.ﾍ a.ｯﾍ..
91D3  C3 30 10 CD E0 11 7C FE FF CA BD 11 C9 21 00 00  ﾃ3.ﾍ..:..ﾊｽ.ﾉ!..

9289  00 DB 01 E6 50 C9 CD 86 12 C0 CD A2 15 CD 03 1A  .ﾛ..Pﾉﾍ .ﾀﾍ｢.ﾍ..
9299  C3 30 10 CD 6F 11 0E 06 1A 6F 13 1A 67 13 CD 5D  ﾃ3.ﾍo....o..g.ﾍ]
```

11章　エラーコード

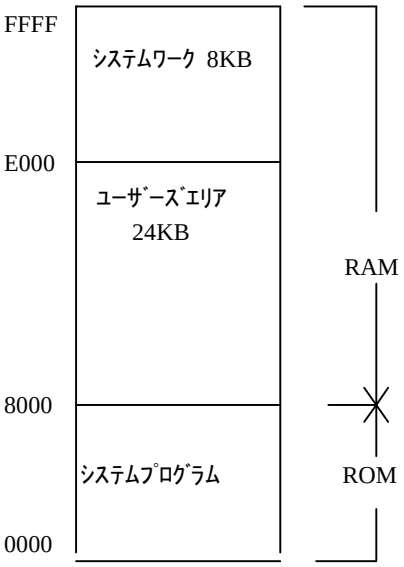
BASICプログラムの実行中にエラーが発生すると、エラーの原因を示す1～3桁のエラーコードが、ERR:　に続けて表示されます。
BASICの命令をコマンドモードで実行したときにエラーが発生しても、このエラーコードが表示されます。
コマンドモードでは、マシン語モニタコマンドなど、その他のコマンドとして登録してある名前以外のものを入力すると、すべてBASIC命令の入力として受け取るようにソフトが組んであります。
したがってコマンドの入力ミスは通常LET文として読み取られ、その結果エラーコードが表示されます（例えばCM 8000[Enter]と入力するところをBM　8000[Enter]と入力してしまった場合、ERR:23が表示されます）。
なおマシン語モニタコマンド（CM、DMなど）の実行中のエラーに対しては、WHAT？、HOW？、SORRYなどの表示になります。
以下にエラーコードとその原因と考えられる主なものを示します（エラーによっては、ここに示した以外の原因によるものもあります）。

1　変数名、ラベル名が5桁（%、$、#、＊付の時は6桁）を越えているか、英数字以外の文字が含まれている。
2　DIM文ですでに定義ずみの配列が再び定義された。またはその配列名がすでに変数名として使用されている。
3　メモリオーバー。プログラムか配列のサイズが大きすぎる。
4　配列の添字に文法エラーがある。または配列名が正しくないかDIM文の後ろに他の文がある。
5　BASIC以外のROMをセットしてROMスタートしようとした。
6　1～32767以外の行番号が入力された。
7　配列名と変数名で同じ名前が使用されたか、配列名に（　）が無い。
8　数値のオーバーフロー。入力された数値の桁数が大きすぎる（実数型は6桁、倍精度実数型は16桁以内）又は計算の結果が処理できる範囲を越えた。
9　マルチステートメント記述でコロン（:）が無いか、文法に合わない文がある。
10　REM文または”　”の中以外で英小文字、カナ文字が使用された。
11　数値のアンダーフロー。計算の結果が処理できる値よりも小さくなった。
12　除数が0の割算が行われた。
13　関数名、配列名に（　）が無いか、（　）内に文法エラーがある。
14　予約語の使い方が間違っている
15　記号の使い方が間違っている。たとえばコロン（:）やカンマ（ ,）が行の先頭にある。
16　GOSUB文が実行されていないのにRETURN文が実行された。
17　FOR文にTOが使われていないか、TOの前におかしなところがある。
18　NEXT文に変数名または配列名が無い。
19　FOR文よりも先にNEXT文が実行された。
20　FOR文とNEXT文で変数名または配列名が異なっている。
21　INPUT文に変数名または配列名が無いか、その前におかしなところがある。
22　計算式に文法エラーがある。例えば演算子＊＋／－などが続けて使われているか、計算に使用できない記号や文字がある。
23　FOR文、IF文、LET文などにおかしなところがある。例えばLET文の左辺に変数以外の文字があるか、THEN　行番号　とした場合など。（ZB3BASICでは、IF～THEN　行番号　の形は使えない。行番号を示す時は、IF～　THEN　GOTO　行番号　を使う）
24　文字定数の長さが39文字を越えるか、右の”が無い。または”が続けて使われている。
25　文字式に文字変数、文字配列、文字定数、文字関数以外のものが使われている。
26　INPUT文の実行中に入力されたデータにエラーがある。例えば数値型の変数に文字列を入力した場合など。
27　IF文でTHENまたはGOTOを使わないでELSEが使われたか、ELSEが余分に使用されている。
28　POKE、OUT、SET、OR文などでパラメータの区切のカンマ（ ,）が無い。
29　READ文に対応するDATA文が無いか、型が合わない。またはDATA文の表記におかしなところがある。
30　READ文に変数およびカンマ（ ,） 以外の文字がある。またはカンマ（ ,） が続けて使用されている。
31　RUN、GOTO、GOSUB文などで指定した行番号が存在しない。または行番号が落ちているか行番号以外の文字が書かれている。または行番号部分に負数が使われている
32　整数演算でオーバーフローが起きた。整数演算は－32768～＋32767の数しか扱えない。整数型の変数、配列や関数に上記の範囲を越える数を与えた場合などに整数オーバフローになる。ただしFIX関数は±2の 23 乗－1までの数が扱える。
33　整数演算で処理できない関数がある。例えばSIN、COSやLOG、SQRなど。
34　配列の添字がDIM文で定義した範囲外の値になっているか、未定義の配列を使用した。
35　FOR～NEXT、GOSUB～RETURNのネスティング（重ね合わせ）が深すぎてスタックオーバーになった

36 16進数の表記に文法エラーがある。16進数は＄××か＄××××以外の表記はできない。
37 数字式の中に文字型の関数やその他の予約語がある。
38 MID＄関数のパラメータが、その文字列の長さより大きい値になっている。
39 文字型関数で扱える数の範囲外を指定した。例えばHEX＄の第2パラメータが1～4以外か、CHR＄で0～255以外の値を指定しているなど。
40 ON ERROR GOTO文が実行されていないのに、RESUME文が実行された。
41 VAL関数で数の桁数が大きすぎるか、数値以外の文字があって変換できない。または値そのものが大きすぎる(オーバーフロー)。
42 PRINT文でカンマ(,)やセミコロン(;)の位置や使い方がおかしい。
43 CONTコマンドが実行できない。もともとRUNが実行されていなかったか、プログラムの終わりでブレイクしたときは、CONTは無効になる。またON ERROR GOTO文によるエラー処理中のブレイクもCONTは無効になる。
44 STOP文にSTOP以外の文字がある。
45 FN関数(ユーザー定義関数)に( )が無いか、余計な文字が含まれている。
46 同じFN関数が2重に定義された。
47 未定義のFN関数が使用された。
48 FN関数名に＄がついているか、パラメータ、定義式に文字型の定数、変数などが使われた。
(注意)DEF FN文に誤りがあっても、そのFN関数が参照された行のエラーになって、その行が表示されることがある。
49 FN関数に別のFN関数が含まれている。FN関数の定義式には数値変数(配列を含む)、数値定数、数値型の関数は使用できるが、別のFN関数は使用できない。
50 参照されたFN関数のパラメータの数がDEF FN文のパラメータの数と合わない。または( )内の表記におかしなところがある。
(注意)DEF FN文に誤りがあっても、そのFN関数が参照された行のエラーになって、その行が表示されることがある。
51 LN、LOGの置数が0かマイナスである。またはSQRの置数がマイナス。
52 べき乗、EXPでオーバーフローがおきた。
53 LOCATE文でパラメータの値が、0≦X≦79、0≦Y≦24の範囲外である。
54
55 TIME＄の使い方が間違っている。
56 RENコマンド実行中に新行番号が32767を越えた。処理を打ち切る。
57 RENコマンドを実行するのに必要なワークエリアが足りなくて処理できない(プログラムか変数名が大き過ぎる)
58 RENコマンドの2番目のパラメータで指定した行番号が存在しない。
59 RENコマンドのパラメータの表記に誤りがある。
60 DATA、DEF FN、DIM文はダイレクトモードでは使用できない。
61 WRITE文で＃が無いかパラメータにミスがある。
62 READ＃文でパラメータにミスがある。
63 数値の指数表現が正しくない(E±……、D±……の表現に誤りがある)。
64 倍精度実数型は使用できない(SET、RES、BITなど)。
65 第一パラメータで配列名が指定されていないか整数型の配列ではない。またはパラメータの表記に誤りがある(SEARCH)。
66 SEARCHで指定した配列がDIM文で定義されていない。または一次元の配列ではない。
67 区切りの ,(カンマ)がないかパラメータの表記が正しくない(SEARCH)。
68 パラメータの表記に誤りがある(SWAP)。
69 SWAP文で指定した変数、配列の型が合っていない。
70 ラベルが二重に定義された。
71
72
73 RS232Cの受信割り込みでREADエラーが発生した(＃1)。
74 DEC、BCD関数で扱える値の範囲を越えている。
75
76
77
78
79 将来のための予約語なので使用できない。

12章　メモリマップ・文字コード表

1．ZB3BASICモードでのメモリマップ



2．詳細メモリマップ（1）

FFFF F800	ｼｽﾃﾑﾜｰｸ
F7FF F490	ｼｽﾃﾑｽﾀｯｸ
F48F F474	ｼｽﾃﾑﾜｰｸ
F473 F300	A%-Z% A$-H$
F2FF F000	ｼｽﾃﾑﾜｰｸ
EFFF E000	ｼｽﾃﾑﾜｰｸ
DFFF aaaa 8000	BASICﾃﾞｰﾀ ﾕｰｻﾞｰｴﾘｱ BASICﾌﾟﾛｸﾞﾗﾑ
7FFF 0000	ｼｽﾃﾑﾌﾟﾛｸﾞﾗﾑ

3. 詳細メモリマップ（2） BASIC固定変数エリア

F472, F473	Z%
F470, F471	Y%
F46E, F46F	X%
F46C, F46D	W%
F46A, F46B	V%
F468, F469	U%
F466, F467	T%
F464, F465	S%
F462, F463	R%
F460, F461	Q%
F45E, F45F	P%
F45C, F45D	O%
F45A, F45B	N%
F458, F459	M%
F456, F457	L%
F454, F455	K%
F452, F453	J%
F450, F451	I%
F44E, F44F	H%
F44C, F44D	G%
F44A, F44B	F%
F448, F449	E%
F446, F447	D%
F444, F445	C%
F442, F443	B%
F440, F441	A%

F43F F418	H$
F417 F3F0	G$
F3EF F3C8	F$
F3C7 F3A0	E$
F39F F378	D$
F377 F350	C$
F34F F328	B$
F327 F300	A$

[注1]BASICプログラムは aaaa 番地から格納されます。またBASIC変数のデータは DFFF から前の方に格納されていきます。aaaa は電源投入後またはリセット後は 8004 番地になります。

マシン語のサブルーチンなどのエリアとしてユーザーが占有する領域が欲しい場合にはNEWコマンドを使って、NEW aaaa と入力すれば 8004～aaaa－1 番地がユーザー用にリザーブされます。またLOADコマンドを使って、／LOAD ″ﾌｧｲﾙﾈｰﾑ″, aaaa と入力することによっても、プログラムLOADと同時にユーザー用のエリアを 8004～aaaa－1 番地に確保することができます。

8000～8003 番地はマシン語のみで使用する場合にはユーザーが使用しても構いませんがBASICプログラムをマシン語サブルーチンと合わせてバイナリイメージファイルとして保存するときはこの 8000～8003 番地が制御情報エリアとして使用されます。したがってもしBASICプログラムをマシン語サブルーチンと一緒にバイナリイメージファイルとして保存する場合には 8000～84003 番地は空きにしておいて、8004～aaaa－1 番地にマシン語プログラムを入れるようにします。

[注2]A%～Z%、A＄～H＄のエリアはBASICプログラムとマシン語サブルーチンの間でデータの受け渡しを楽にできるように設定したものです。

A%～Z%は16ビットの整数（－32768～＋32767）を2進数（16進数）の形で格納します。参考までにA%に＋12345、B%に－12345が格納されている様子を示します。

（B%）F443	CF	…… －12345＝＄CFC7
F442	C7	
（A%）F441	30	…… ＋12345＝＄3039
F440	39	

A＄～Z＄は最長39バイトの文字列をキャラクタコードの形で格納します。参考までにA＄に”ABCDEFG”、B＄に”－12345”が格納されている様子を示します（次ページ図）。

（Ｂ＄）F34F	使用されない		（Ａ＄）F327	使用されない	
F32F			F308		
F32E	35	”５”	F307	47	”Ｇ”
F32D	34	”４”	F306	46	”Ｆ”
F32C	33	”３”	F305	45	”Ｅ”
F32B	32	”２”	F304	44	”Ｄ”
F32A	31	”１”	F303	43	”Ｃ”
F329	2D	”－”	F302	42	”Ｂ”
F328	06	文字列の桁数	F301	41	”Ａ”
			F300	07	文字列の桁数

この例のように文字変数の場合には、その第1バイトにはその文字列のバイト数が入ります。文字コードについては次項を参照して下さい。

４．　文字コード表

上位４ビット

下位４ビット	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0			SP	0	@	P	‘	p			SP	ー	タ	ミ		
1			!	1	A	Q	a	q			。	ア	チ	ム		
2			”	2	B	R	b	r			「	イ	ツ	メ		
3			#	3	C	S	c	s			」	ウ	テ	モ		
4			$	4	D	T	d	t			、	エ	ト	ヤ		
5			%	5	E	U	e	u			・	オ	ナ	ユ		
6			&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
7			’	7	G	W	g	w			ァ	キ	ヌ	ラ		
8			(	8	H	X	h	x			ィ	ク	ネ	リ		
9			)	9	I	Y	i	y			ゥ	ケ	ノ	ル		
A	LF		*	:	J	Z	j	z			ェ	コ	ハ	レ		
B			+	;	K	[	k	{			ォ	サ	ヒ	ロ		
C			,	<	L	¥	l	:			ャ	シ	フ	ワ		
D	CR		-	=	M	]	m	}			ュ	ス	ヘ	ン		
E			.	>	N	^	n				ョ	セ	ホ	゛		
F			/	?	O	_	o				ッ	ソ	マ	゜		

LF＝LINE FEED 、CR＝CARRIAGE RETURN 、SP＝SPACE