ZBK－SBASIC（整数BASICコンパイラ）取扱説明書

有限会社中日電工

［注意］SBASICはコンパイラです。Z80マシン語のバイナリファイルを生成します。コンパイル操作はZBKボードの接続を必要としませんが、コンパイル後のZ80マシン語ファイルはZBKボードにロードする必要があります。Z80のマシン語ファイルですがZBKボード上のシステムROM内サブルーチンをコールしているため、ZBKボード（ZB10K～ZB28K）以外のZ80ボード上では実行することができません。また**ZBKボードは05年6月以後出荷のROM（050617以後）を搭載している必要があります。**

ZBK－V3BASICはインタプリタですがSBASICはコンパイラです。あらかじめマシン語に翻訳してから実行するのでインタプリタよりも高速で実行できます。V3BASICほどの機能は必要としないが、もっと高速で処理したい、という目的で作成しました。SBASICはV3BASICと異なり整数しか扱えません。I／Oを中心とする多くの処理ではそれで十分な場合が多いと考え、より高速であることを重視しました。

使い方は簡単です。文法はV3BASICとほとんど同じです。ソースプログラムはWindows上でテキストエディタ（notepadなど）で作成します。拡張子が　. txt　のファイルを作成します。コンパイルはDOSプロンプト（DOS窓）で行いますがV3BASICとは異なり、ZBKボードを接続、起動しておく必要はありません。コンパイルによって作成されたマシン語プログラムを実行する時点でZBK開発システムを起動して、マシン語プログラムをロードします。

SBASIC. COMとソースプログラムはどのフォルダに置いても、そのDOS窓でフォルダに行ってSBASIC. COMを実行しコンパイルすることはできますが、どうせZBKシステムで実行、デバッグすることになりますから、ZBK. COMと同じフォルダにSBASIC. COMもコピーしておいて、ソースプログラムも同じフォルダに作成するとよいでしょう。

とりあえずSBASICでのコンパイルを体験するだけなら、すでに説明したようにZBKシステムを接続、起動しておく必要はありません。しかし実用上は、コンパイルによって作成されたマシン語プログラムはすぐにZBKシステムにロードして実行デバッグするはずですし、大抵は一回では完了せず、その結果を見た上でソースプログラムを修正し、再コンパイルし、再びZBKシステムにロードし実行デバッグするということを繰り返すはずです。したがってZBKシステムもあらかじめ起動しておいた方が望ましいのですが、ZBKシステムは一度起動するとそのDOS窓上ではZBK以外の作業は行えなくなります。ソースプログラムを修正するだけならDOS窓からnotepadにマウスを移動するだけで簡単にできますが、SBASICを再実行するためにはその度にZBKシステムを終了しなければなりません。その不便を解消するために、ZBKシステムを起動するDOS窓と別にSBASICを実行するDOS窓を開いておくことを推奨します（どちらのDOS窓もZBK. COM及びSBASIC. COMのある同じフォルダを指定します）。そうすることによりマウスを移動するだけでSBASICのコンパイルとZBKシステム上でのロード、デバッグとをそれぞれ継続して行うことができます。

以下具体的な操作方法を説明します。

1．ソースプログラムの作成・実行

1．1　プログラムの作成

ソースプログラムはテキストエディタ（notepad）などを使って作成します。ファイル名はMSDOSのルールにしたがいます（最大8文字の英数字＋拡張子. txt）。

［簡単なプログラム例］ ファイル名　TEST1. TXT

```
' TESTPROGRAM
        ORG=$4004
        A%=0:B%=2
*LOOP:PRINT A%,B%
        A%=A%+5:B%=B%+3
        IF A%<50 GOTO *LOOP
        PRINT "END"
        RETURN
```

①行番号はつけません（つけてもコンパイラは無視します）

②最初の実行文より前にORG文が必要です。翻訳後のマシン語プログラムの開始アドレスを指定します。通常は＄4004にします(ZBK－BASICのマシン語サブルーチンと同じです)。
ZBK－V3BASICと同じくコメント文以外は全て英大文字を使用します(全角は使えません。必ず半角にしてください)。
　行の始めや命令と変数名などの区切りには任意の桁数のスペース(空白)を入れることができますから、Tabやスペースを使って見やすいレイアウトにすることができます。
③使える変数名はA%～Z%、A＄～H＄と一次配列@( )、及びメモリ参照用の[ ]のみです。
:を使って複数の命令を書くことができます。
④GOTO、GOSUB文のためにはラベルを使います。＊に続けて1～13桁の英数字およびー、＿が使えます。
⑤IF文は、IF. . . GOTO しかありません。IF GOTO文の後ろに : をつけて複数の命令文を書くとその文はIFが不成立の時に実行される文になります(IF GOTOの次の行とみなされる)。
⑥このコンパイラはZBK－V3BASICのマシン語サブルーチン作成用です。単独に実行されることを考慮していません。ZBK－V3BASICのUSR命令によってコールされることで実行されます。そのためプログラムの最後にはRETURN文が必要ですが、例のようにプログラムの最終行がそのまま実行の終了ポイントであるような場合に限ってRETURN文を省略することができます。コンパイラはファイルの最後にRETURNコード(C9)を付け加えるようになっています。

1．2 プログラムのコンパイル
　あらかじめZBK．COMとおなじフォルダにSBASIC．COMもCOPYしておいてください。ソースプログラムも同じフォルダにSAVEします。
　ZBK－V3BASICと同じようにDOS窓を開いて、ZBKシステムのあるフォルダに移動します(コンパイルするだけならZBKボードを接続してZBK．COMを起動する必要はありません)。
　SBASIC．COMを実行します。

```
C:￥zbk＞SBASIC　TEST1．TXT[Enter]
TEST1.BIN ...OK　$006C(108)bytes
TEST1.WK ...OK
C : ￥zbk＞
```

　SBASIC．COMが正常に終了すると、マシン語バイナリファイル(TEST1．BIN)と翻訳情報ファイル(TEST1．WK)が作成されます。何らかのエラーがあるとWHAT？またはHOW？に続いてエラーの発生した行が表示され、翻訳は中止されます。エラーを訂正して上書き保存してから再度SBASIC．COMを実行してください。
　翻訳情報ファイルは拡張子は．wkですがテキストファイルなのでnotepadなどで開けます。通常は参照する必要はありません。BASIC命令がマシン語に翻訳されたアドレスの情報が記録されています。
[作成されたTEST1．WKの内容]

```
0000 ' TESTPROGRAM
0000      ORG=$4004
4004      A%=0:400A B%=2
4010 *LOOP:4010 PRINT A%,B%
402A      A%=A%+5:4038 B%=B%+3
4046      IF A%<50 GOTO *LOOP
405D      PRINT "END"
406E      RETURN
406F                ←コンパイラによって追加されたＲＥＴＵＲＮコードがある
```

1．3 プログラムの実行
　SBASICコンパイラによって作成されたバイナリファイルTEST1．BINはZBKシステムを起動させて、／LDコマンドで＄4004番地にロードしたあと、USR(＄4004)命令によって実行することができます。

下にTEST1．BINのロード、USR（＄4004）のダイレクト命令による実行結果を示します。また参考までにロードされたマシン語プログラムをDAコマンドで逆アセンブル表示しました。

```
DEBUG TOOL FOR KL5C8012
(C)Copyright CHUNICHIDENKO 2000,2001,2004,2005 Rev.1.4
02
t/0202
)Z
>/LD TEST1.BIN,4004                    ＴＥＳＴ１．ＢＩＮのロード
006CBYTES LOAD
>USR($4004)                            ＴＥＳＴ１．ＢＩＮの実行
0      2
5      5
10      8
15      11
20      14
25      17
30      20
35      23
40      26
45      29
END
>DA 4004                               ＴＥＳＴ１．ＢＩＮの逆アセンブル表示
4004 210000     LD HL,$0000
4007 2240F4     LD ($F440),HL
400A 210200     LD HL,$0002
400D 2242F4     LD ($F442),HL
4010 2A40F4     LD HL,($F440)
4013 0E00       LD C,00
4015 CD5D10     CALL $105D
4018 0605       LD B,05
401A CD3910     CALL $1039
401D 10FB       DJNZ FB
401F 2A42F4     LD HL,($F442)
4022 0E00       LD C,00
4024 CD5D10     CALL $105D
4027 CD1B10     CALL $101B
402A 2A40F4     LD HL,($F440)
402D E5         PUSH HL
402E 210500     LD HL,$0005
4031 D1         POP DE
4032 CD0628     CALL $2806
4035 2240F4     LD ($F440),HL
4038 2A42F4     LD HL,($F442)
403B E5         PUSH HL
403C 210300     LD HL,$0003
403F D1         POP DE
4040 CD0628     CALL $2806
4043 2242F4     LD ($F442),HL
4046 2A40F4     LD HL,($F440)
```

```
4049 E5         PUSH HL
404A 213200     LD HL,$0032
404D D1         POP DE
404E EB         EX DE,HL
404F B7         OR A
4050 ED52       SBC HL,DE
4052 CB7C       BIT 7,H
4054 210100     LD HL,$0001
4057 2001       JR NZ,01
4059 2D         DEC L
405A C21040     JP NZ,$4010
405D C36440     JP $4064
4060 45         LD B,L
4061 4E         LD C,(HL)
4062 44         LD B,H
4063 00         NOP
4064 116040     LD DE,$4060
4067 AF         XOR A
4068 CD1810     CALL $1018
406B CD1B10     CALL $101B
406E C9         RET
406F C9         RET

>/EXIT
```

①＄４０６０～＄４０６３はＤＡコマンドでは正しく表示されません。ここは「ＥＮＤ」のＡＳＣＩＩ文字列（最後はエンドコード００）です。
②ここではＵＳＲ（＄４００４）をダイレクト実行していますが、一般的には次のようにＢＡＳＩＣプログラムとして実行させます（マシン語サブルーチンのメモリエリアを確保するためにＢＡＳＩＣプログラム作成前にＮＥＷコマンドの実行が必要です。この例ではＮＥＷ, ４１００で十分ですがマシン語プログラムのサイズに合わせて十分な値を確保する必要があります）。

```
>NEW,4100
>10 USR($4004)
>RUN
```

③ＤＡコマンドによる逆アセンブル結果でもわかるように、ＳＢＡＳＩＣコンパイラはＺＢＫシステムＲＯＭ内のシステムルーチンをコールする形で必要な処理ルーチンを組み込んでいます。したがってＳＢＡＳＩＣコンパイラによって作成されたバイナリファイルは、Ｚ８０マシン語プログラムですがＺＢＫボード以外のＺ８０ボード上では実行させることはできません。
④マシン語サブルーチンで実行されるＰＲＩＮＴ文はＬＣＤ表示用としても実行可能ですが、基本的にはデバッグ用の機能と考えてください。作成したマシン語プログラムが正しく実行されているかどうかを確認するための途中経過の表示用として位置付けています。
　コンパイラは文法の誤りは指摘しますが、作成されたマシン語プログラムはいつも正しく実行されるとは限りません。ＢＡＳＩＣインタプリタならば実行中の誤りもエラー表示してブレイクしますが、マシン語プログラムにはその機能はありません。致命的なエラー（ＧＯＳＵＢ～ＲＥＴＵＲＮ、ＦＯＲ～ＮＥＸＴが論理的に正しくない場合など）ではシステムが暴走したりハングアップしたりします。適切なポイント毎にＰＲＩＮＴ文をいれておけば、暴走した場合などでも強制的にＤＯＳプロンプトを終了させたあとでログを分析すれば暴走の原因を追求することができます。

2. SBASICの文法

基本的にはZBK－V3BASICと共通ですが部分的に相違しているところがありますから一通りは目を通して

ください。

2．1　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。

2．1．1　行

行は文で構成されます。行番号はつけてもエラーにはなりませんがコンパイラによって無視されます。ソースプログラムの1行の長さに明示的な制限はありませんがコンパイル後にトラブルが発生する可能性があります。V3BASICと同じく、おおむね80字以内にしてください。

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。

2．1．2　文（ステートメント）

文は命令実行単位で、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。ラベルも文の一種ですが行の先頭に書きます。

```
＊LOOP1：A%＝A%＋1：PRINT　A%：GOTO　＊LOOP2
　　↑　　↑　　↑　　↑
ラベル　　文　　文　　文
```

2．2　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。

2．2．1　命令

FOR～NEXT、GOSUB～RETURN、GOTO、IF…GOTO、OUT、PRINT

以上はV3BASICにもある命令ですが、SBASICのみに使用可能な命令として以下のものがあります。

INC、DEC、IFNZ　GOTO、IFZ　GOTO

2．2．2　関数

ABS、AND、ASC、CHR＄、HEX＄、IN、LEFT＄、LEN、MID＄、OR、RIGHT＄、SGN、STR＄、VAL、XOR

2．2．3　定数

SBASIC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。

① 十進定数

SBASICでは0および－32768～＋32767の範囲の整数しか扱えません。

② 16進定数

SBASICでは2桁又は4桁の16進定数が扱えます。（2桁、4桁以外は扱えません）

16進定数は10進定数と区別するために、＄マークをつけて表します。

［例］

A%＝＄FF

B%＝C%＊K%＋＄B000

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（半角に限ります）。

[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]
```
PRINT "ABCDE"
PRINT CHR$($22)+"ABCDE"+CHR$($22)
```
[実行結果]
```
ABCDE
"ABCDE"
```

2．2．4 変数

SBASICでは整数型のA%～Z%及び文字型のA＄～H＄しか扱えません。

この変数はローカル変数ではなくてメインプログラムであるV3BASICと共通のアドレスを割り当てられます。メインプログラムでUSR(＄4004)を実行する前のA%～Z%、A＄～H＄の値はUSR(＄4004)によってコールされたSBASICマシン語サブルーチンの中で同じ名前で参照することができます。またSBASICマシン語サブルーチン内でA%～Z%、A＄～H＄の値を書き換えると、V3BASICメインプログラムに戻ったあとのA%～Z%、A＄～H＄も同じように書き換わった値を持ちます。

2．2．5 配列

SBASICでは整数型1次配列@( )しか使えません。( )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。配列または配列を含む式を添字として使うこともできます。

V3BASICとは違い@( )はDIM文でその大きさを定義しません。V3BASICの変数エリアに固定的に割り付けられます。@(0)は＄DFFE、＄DFFFに、@(1)は＄DFFC、＄DFFDに割り付けられ、以下@(2)は＄DFFA、＄DFFBにというように順に割り付けられます。

もしV3BASICでA%～Z%、A＄～H＄以外の変数や配列を使用すると、その変数や配列は定義された順に＄DFFFから前の方に割り付けられていきます。その結果、SBASICで@( )を使うとV3BASICの変数、配列のエリアと競合することになります。これを避けるためには、SBASICで使用しようとしている@( )の最大個数をV3BASICプログラムの先頭でダミーの整数配列として定義するようにします。例えば

```
10 DIM ATT%(100)
```

というように。こうすることでSBASICの@(0)～@(100)とV3BASICのATT%(0)～ATT%(100)のエリアが完全に一致するためV3BASICの他の変数や配列とSBASICの@( )とが競合することが避けられます。

2．2．6 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。

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＊(乗算)及び／(除算) ②＋(加算)及び－(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。
なお文字型の定数、変数、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。

② 関係式

関係式は2つの要素(変数、定数、配列)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。V3BASICとは違い、要素に計算式、関数を使うことはできません。また関係演算子の左側には定数を置くことはできません。左側には変数、配列のみ置くことができます。

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］
IF A%＞0 GOTO ＊END
IF @(B%)＜=C% GOTO ＊LOOP

文字型の変数、定数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。
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 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］
IF (A%＞=0)＊(B%=C%) GOTO ＊KEISAN

上例の意味は「A%≧0で同時にB%＝C%ならば＊KEISAN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＊KEISANへジャンプせよ」になります。

2．2．7 ラベル名
SBASIC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13桁の英数字および－（マイナス、ハイフン）、＿（アンダーバー）で表します（＊を含めて最大14桁）。先頭に＊を置くこと以外に規則はありません。＊12345とか＊＿XYZなども許されます。
GOTO文、GOSUB文の行き先にラベル名を書くことで、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］
```
IF S%=0 GOTO *TASU
IF S%=1 GOTO *HIKU
GOTO *KAKERU
'
*TASU:A%=B%＋C%:PRINT A%:RETURN
*HIKU:A%=B%－C%:PRINT A%:RETURN
*KAKERU:A%=B%*C%:PRINT A%:RETURN
```

ラベル名は行の一番先頭に書かなければいけません。
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。

```
＊TASU
A%=B%+C%:PRINT A%:RETURN
```

2．2．8 メモリ参照用変数

V3BASICには無い機能です。1バイトのメモリを直接扱います。V3BASICのPOKE、PEEKの機能に相当しますが、マシン語の(HL)に置き換わるため高速で処理が行われます。特定のメモリエリアにデータを書き込みたいとか、置き換えたいときなどに効果的です。書式は ［a］ です。aには定数、変数、配列のほか計算式も書くことができます。[a]はaの値で示されるアドレスの1バイトのメモリを示しています。例えばB%＝＄8000であるとき、A%＝[B%]は＄8000番地のメモリの値をA%に入れるという働きになります。また[B%]＝0は＄8000番地のメモリの値を0にするという働きになります。

[注意]V3BASICのPOKE文と同様、[a]もメモリを直接扱うため、aの値には十分注意が必要です。注意しないで使用すると、システムのアドレスやV3BASICの変数エリアを書き換えてしまう可能性もあり、その場合にはシステムが暴走したり、プログラムが異常な動作をするかもしれません。

3．BASIC命令（ステートメント）

3．1 DEC

V3BASICのDEC関数ではありません。変数の値を－1減算する命令です。ある処理を一定回数繰り返す時に変数のカウンタを用意して一回処理するごとに変数カウンタの値を－1していき、変数の値が0になるまで繰り返すようなプログラムはよく使われます。そのような時にDECとあとで説明するIFNZを組み合わせて使えばより短い時間で実行できます。

[使用例]

```
        A%=100
*LOOP:@(A%)=0
        DEC A%
        IFNZ *LOOP
```

［注記１］ＤＥＣはＩＦＮＺと組合わせて使わなければいけないという制約はありません。Ｘ％＝Ｘ％－１の代わりにＤＥＣ　Ｘ％と書くことができます。変数Ａ％～Ｚ％のほかに＠（　）に対して、ＤＥＣ　＠（１０）のように使うこともできます。

［注記２］ＤＥＣと後述のＩＮＣに限って、下例のようにＩＦＮＺまたはＩＦＺと組み合わせて１行に書くことができます。

［例］ＤＥＣ　Ａ％　ＩＦＮＺ　＊ＬＯＯＰ

3．2 FOR～NEXT

[書式]

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

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、A%～Z%、@( )が使えますが、Ａ＄～Ｈ＄は使用できません。

[使用例]

```
FOR A%=3 TO 16 STEP 2
PRINT A%,
NEXT
PRINT "END",A%
```

[実行結果]

```
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PRINT A%、NEXT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。）

●V3BASICとは違ってNEXTに変数名、配列名を書く必要はありません（書いてもコンパイラは無視します）。
●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、配列、定数を書くことはできますが式を書くことはできません。
●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、限度を越えると暴走する可能性があります。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文を何重まで安全に使用できるかは、そのプログラムによって異なります。メインプログラムも含めてスタックの使用が深くなりすぎると思われる場合にはFOR～NEXTを使わず、IF GOTO文によるループを使用するようにした方が安全です。
なおFOR～NEXTを多重に使用する場合に、NEXTが重なっても省略することはできません。

```
FOR A%＝0 TO 100 ①
FOR B%＝5 TO 50  ②

NEXT      ②に対するNEXT
NEXT      ①に対するNEXT
```

3．3 GOSUB～RETURN

［書式］
GOSUB ラベル名

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文で指定するラベル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。
サブルーチンの指定は＊マーク付のラベル名により行います。
指定したラベル名が存在しないときはコンパイル時にHOW？が表示されます。

3．4 GOTO

［書式］
GOTO ラベル名

指定する行にジャンプしてそこから実行を続けます。
行の指定は＊マーク付のラベル名によって行います。
指定したラベル名が存在しないときはコンパイル時にHOW？が表示されます。

3．5 IF…GOTO

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GOTO文を実行し、偽であれば次の行が実行されます。関係演算子（＝、＜、＞、＜＝、＞＝、＜＞）の左側には、変数、配列しか書けません。右側には定数も書けますが、計算式は書けません。

●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の長い方が大になります。
同じ長さ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。

3．6 IFNZ GOTO

直前の命令(INCまたはDEC)の実行結果が0でなければGOTO文を実行します。結果が0ならば次の行の命令を実行します(IFNZは1語です。IF NZと書いてはいけません)。

3．7 IFZ GOTO

直前の命令(INCまたはDEC)の実行結果が0ならばGOTO文を実行します。結果が0でなければ次の行の命令を実行します(IFZは1語です。IF Zと書いてはいけません)。

3．8 INC

変数の値を＋1加算する命令です。DEC命令と同様の機能です。DECが変数の内容を－1するのに対しINCは＋1します。A%＝A%＋1とINC A%では結果は全く同じです。

3．9 OUT

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、定数、計算式を使うことができます。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]

```
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(ZB10K、ZB20K以外のZBKボードでは基板上に実装されている1番目の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1番目の82C55のA～Cポートを出力にセットすることになります)。

3．10 PRINT

ディスプレイ画面に、変数、配列の値、文字列を表示します。関数や計算式の値も表示できます。
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。
要素をセミコロン(;)、カンマ(,)で区切って複数個記述することができます。
パラメータをセミコロン(;)で区切ると間をあけないで表示されます。カンマ(,)で区切ると間に5桁の空白が挿入されます。
[使用例]

```
A%＝123:I%＝200:X＄＝"XYZ"
PRINT A%, I%, X＄
PRINT A%;I%;X＄
```

[実行結果]

```
123         200         XYZ
123200XYZ
```

●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、
PRINT A%, I%, X＄; とすると、結果は下のようになります。

```
123         200         XYZ123200XYZ
```

●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。

3．11 RETURN

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。

SBASICでコンパイルするプログラムは全体がV3BASICからUSR（ ）命令でコールされるサブルーチンになりますから、SBASICを終了してV3BASICに戻るときにもRETURN文が必要になります。なおプログラムの最終行がそのプログラムの終わりで、そこからV3BASICに戻る、という場合には最後の行にRETURN文を書くのを省略することができます。SBASICコンパイラはコンパイル後のマシン語プログラムの最後にRETURNコード（C9）を追加します。

4． BASIC関数

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、計算式の中で、ある値を持つ数値として扱われます。関数は（ ）の中の値をもとに計算した結果をその関数の値としますが、（ ）の中の値（変数や配列、定数などの値）は変化しません。次のABS（a）を例にすると、ABS（a）はaの絶対値になりますが、aの値そのものは変化しません。

4．1 ABS

[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]

```
A%＝ABS（B%＊C%－10）
```

4．2 AND

[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]

```
A%＝＄37
B%＝AND（A%，＄0F）
PRINT　HEX＄（A%），HEX＄（B%）
```

[実行結果]

```
37                07
```

4．3 ASC

[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。

4．4 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]

```
A＄＝CHR＄（＄41）＋CHR＄（＄42）＋CHR＄（＄43）
PRINT　A＄
```

[実行結果]

```
ABC
```

4．5 HEX＄

[書式]HEX＄(a)

aの値を16進数化してその文字列を与えます。PRINT文の中で使います。aが1バイトの値であってもつねに2バイト(16進4桁)になります。上位桁の0は省略されません。

( )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]

```
A%＝1234
PRINT A%, HEX＄(A%)
B%＝－A%:PRINT B%, HEX＄(B%)
```

[実行結果]

```
1234            04D2
－1234           FB2E
```

4．6 IN

[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も0～255(＄00～＄FF)です。

4．7 LEFT＄

[書式]LEFT＄(文字列 ,n)

文字列の左端から任意の長さnだけ取り出した文字列を与えます。

( )内の文字列は定数、変数のいずれでもよく、また長さを指定する数値n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正しい結果が得られませんから注意してください。

[使用例]

```
A$="ABCDEFG"
B$=LEFT$(A$,3),C$=MID$(A$,3,3),D$=RIGHT$(A$,3)
PRINT B$,C$,D$
```

[実行結果]

```
ABC             CDE             EFG
```

4．8 LEN

[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、文字変数が記述できます。

[使用例]

```
A$="ABCDE"
A%=LEN(A$):PRINT A%
```

[実行結果]

```
5
```

4．9 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かまたはmが取り出し可能な文字桁数よりも大きいとと正しい結果が得られませんから

注意してください。
[使用例]はLEFT＄の項を参照してください。

4．10 OR

[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。OR関数の取り得る値の範囲も0～255(＄00～ ＄FF)です。
[使用例]

```
A%=$37
B%=OR(A%, $0F)
PRINT HEX$(A%), HEX$(B%)
```

[実行結果]

```
37              3F
```

4．11 RIGHT＄

[書式]RIGHT＄(文字列 ,n)
文字列の右端から任意の長さnだけ取り出した文字列を与えます。
( )内の文字列は定数、変数のいずれでもよく、また長さを指定する数値n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正しい結果が得られませんから注意してください。
[使用例]はLEFT＄の項を参照してください。

4．12 SGN

[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。

4．13 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。

4．14 VAL

[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。

4．15 XOR

[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。XOR関数の取り得る値の範囲も0～255(＄00～＄FF)です。
[使用例]

```
A%=$37
B%=XOR(A%,$0F)
PRINT HEX$(A%),HEX$(B%)
```
[実行結果]
```
37              38
```