豊四季タイニーBASIC for Arduino STM32 V0.5

リファレンスマニュアル RV0.5

作成日 2017 年 4 月 8 日

作成者 たま吉さん

目次

1 概要

1.1 はじめに

1.2 システム構成

1.3 実行環境

（以降作成中）

2 実行環境を用意しよう！

この章では何もない状態から、「Tiny BASIC for Arduino STM32」を動かすまでの環境構築について解説します。“なにもない状態から“といっても、次の機材は必要となります。

① Ｗｉｎｄｏｗｓパソコン（Ｌｉｎｕｘ、Ｍａｃに詳しい人はそれをつかうのも良い）
② Bule Pill ボード(STM32F103C8T6 搭載のマイコンボード)
③ USB ケーブル

（以降作成中）

3 使ってみよう！

この章では、「レガシーBASIC（古き良き時代の BASIC 言語）の雰囲気を知らない方々にその雰囲気をつかんでもらいましょう」ということで、軽いフットワークで解説進めます。

準備 OK？ 前章の実行環境は完璧？ それでは電源を入れて使ってみよう！
TeraTarm あたりのターミナルソフトを起動して繋げてみよう！

3.1 入力可能状態は OK■（カーソル）

```
OK
■
```

この状態は、「いつでもこいや〜」状態。
きみのコマンド入力を待っている状態。
何か打ち込んでみよう！

3.2 Hello,world を表示してみる

直接、次の文字を打ってみよう。

```
print “Hello,world” Enter
```

入力ミスの修正は[BS]キー、[DEL]キー、カーソルキーなどいつものキーが使えるよ。
入力出来たら、最後に[Enter]キーを押してみよう。

```
print “Hello,world” Enter
Hello,world
OK
■
```

表示たね。PRINT は文字列を表示するコマンドなのです！
入力は大文字、小文字どちらでも OK！

3.3 Hello,world を Hello,Tiny BASIC に変更してみる

カーソルキー[↑][↓] [←] [→]でカーソルを動かして、world の w に移動してみよう。

```
print “Hello,world”
Hello,world
```

[DEL]キーで world を削除して、代わりに Tiny BASIC と入力しよう。
入力後に、[Enter]キーを押します。

```
print “Hello,Tiny BASIC” Enter
Hello,tiny BASIC
OK
■
```

こんな感じで[Enter]がコマンド実行の合図だ。
[Enter]を入力する際、カーソルは行のどこにあっても構わない。

3.4 PRINT の前に 10 を追加してプログラムにしよう

再びカーソル操作で print の前に 10 を挿入して[Enter]キーを押してみよう。

```
10 print “Hello,Tiny BASIC” Enter
Hello,tiny BASIC
OK
```

これでプログラムとして登録された。LIST コマンドで確認してみよう。

カーソルを OK の下の空き行に移動して、list と入力

```
OK
list Enter
10 PRINT “Hello,Tiny BASIC”
OK
■
```

登録したプログラムの内容が表示されたね。

まだ、行番号 10 しか登録していないので、その１行だけの表示だ。

このように、先頭に数字(=行番号)をつけると行番号の後ろのコマンドがプログラムとして登録されるのだ。

3.5 プログラムを実行してみよう

う～ん、気分的に、画面を綺麗にしてから表示したくなった。

画面をきれいにしてから、３回実行してみよう。

```
cls Enter
OK
run Enter
Hello,tiny BASIC
OK
run Enter
Hello,tiny BASIC
OK
run Enter
Hello,tiny BASIC
OK
```

３回同じ処理が実行出来たね。

CLS は画面を消すコマンド、RUN はプログラムを実行。よく使うコマンド。

3.6 プログラムに追加する

今の“画面きれいにしてから、３回表示”をプログラムにやらせてみよう！

次をように入力して、プログラムとして登録するよ。

```
5 cls Enter
7 for i=1 to 3 Enter
20 next i Enter
```

入力したプログラムを確認！

```
List Enter
5 CLS
7 FOR I=1 TO 3
10 PRINT “Hello,Tiny BASIC”
20 NEXT I
OK
```

入力したプログラムを実行するよ。

```
run Enter
Hello,tiny BASIC
Hello,tiny BASIC
Hello,tiny BASIC
OK
```

3 行表示出来た。

プログラムの実行順番は行番号順。CLS に最初に実行されるよ。

FOR 文は繰り返しを行う命令。3 回 10 行の表示を行ったよ。

3.7 行番号を並びなおす

行番号の 5,7,10,20 と並びがちょっと美しくないですね。

直してみよう。

RENUM コマンドを使ってプロフラムの行番号を 10 ずつに整えるよ。

```
retum Enter
OK
List Enter
10 CLS
20 FOR I=1 TO 3
30 PRINT “Hello,Tiny BASIC”
40 NEXT I
OK
```

きれいに並んだね。

3.8 やっぱり 10 行は不要、削除したい

やっぱり、直前の表示は画面は消したくない。

10 行はいらない。削除したい。行番号だけ入力して[Enter]を押してみよう。

```
10 Enter
```

これで 10 行が削除されるよ。確認してみよう。

```
List Enter
20 FOR I=1 TO 3
30 PRINT “Hello,Tiny BASIC”
40 NEXT I
OK
```

10 行が無くなったね。

20 行から始まるのは美しくない。行番号をまた振り直そう。

```
retum Enter
10 FOR I=1 TO 3
20 PRINT “Hello,Tiny BASIC”
30 NEXT I
OK
```

こんな感じ行番号だけの入力で、行削除。よく使う操作だよ。

3.9 プログラムを保存したい

今作成したプログラムは SRAM 上に保存されているんだ。

マイコンの電源を切ると消えてしまう・・・

消えないで欲しい・・・。そんな時、SAVE コマンドを使うよ。

SAVE コマンドはプログラムをフラッシュメモリに保存するよ。

フラッシュメモリに保存されたプログラムは電源を切っても消えないよ。

次のコマンドを打ってみましょう。

```
save Enter
OK
```

これで保存できたはずだ。

3.10 保存されたプログラムを読み込み

本当に保存されているか確認してみましょう。

SRAM 上のプログラムを消してフラッシュメモリか読み込みと試してみよう。

NEW コマンドを打ってプログラムを初期化するよ。

```
new Enter
OK
```

これで消えたはず。

LIST コマンドで確認。

```
list Enter
OK
```

何も出てこない。本当に消えた。

次に LOAD コマンドでフラッシュメモリからプログラム読み込むよ。

```
loadEnter
OK
list Enter
10 FOR I=1 TO 3
20 PRINT “Hello,Tiny BASIC”
30 NEXT I
OK
```

成功！ 復活出来た！ ・・・ これで基本的な操作は終了だ。

基本はバッチリ、準備万端！ Tiny BASIC を使った更なるプログラミングをエンジョイしよう！

4 画面操作

ターミナルソフト TeraTerm にて接続した場合に利用できるキーです。

入力可能文字　：半角英数字、半角記号

キーボード操作

カーソルの上下左右移動	: [←][→][↑][↓]
行の先頭に移動	: [HOME]
行の末端に移動	: [END]
カーソルの前の文字の削除	: [BackSpace]
カーソル位置の文字の削除	: [Delete], [CTRL-X]

編集モードの切り替え

挿入モード・上書きボード切り替え	: [Insert]

その他

入力行の確定	: [Enter]
画面の再表示	: [PageUP],[PageDown],[CTRL-R]
実行プログラムの強制終了	: [CTRL-C], [ESC]2 回押し
画面の表示内容消去	: [CTRL-L]

画面表示制御コマンド

画面の表示内容消去	: CLS
画面の文字色の設定	: COLOR
画面の文字表示属性の設定	: ATTR
画面の再表示	: REDRAW
画面カーソル移動	: LOCATE
文字の表示	: PRINT
日付表示	: DATE

5 プログラムの要素の説明

（ここにサンプルプログラムのソース）

（構成要素の説明）

（構成要素の一覧）

コマンド

行番号

コマンド区切り

数値・数値定数

変数

配列変数

代入文

式・演算子

文字列

関数（数値関数、文字列関数）

制御文

（以降作成中）

5.1 コマンド

コマンドとは何らかの処理を行う命令文です。

コマンドにはプログラム作成や環境設定等に利用するシステムコマンドと、プログラム中で利用する一般コマンドがあります。

システムコマンド：

LIST	プログラムリストの表示
RENUM	行番号の振り直し
RUN	プログラムの実行
NEW	プログラムの消去
SAVE	内部フラッシュメモリへのプログラム保存
LOAD	内部フラッシュメモリからプログラムを読み込む

システムコマンドは、コマンドライン直接入力のみ利用できます。

プログラム中での記述は可能ですが無視されます。

一般コマンド：

REM	コメント
LET	変数に値を代入
PRINT	画面への文字表示
INPUT	数値の入力
CLS	画面表示内容の全消去
COLOR	文字色の設定
ATTR	文字表示属性の設定
LOCATE	カーソルの移動
REDRAW	画面表示の再表示
DATE	現在時刻の表示
GETDATE	日付の取得
GETTIME	時刻の取得
SETDATE	時刻の設定
RESETTICK	起動からの経過時間カウントのリセット
WAIT	時間待ち
GPIO	GPIO 機能設定
OUT	デジタル出力
SHIFTOUT	デジタルシフトアウト出力
POKE	指定アドレスへのデータ書き込み
EEPFORMAT	仮想 EEPROM のフォーマット
EEPWRITE	仮想 EEPROM のへのデータ書き込み

一般コマンドは、コマンドライン直接入力での実行のほか、プログラム中での利用も可能です。

システムコマンド、一般コマンドの詳細については「8 各コマンド・関数」を参照して下さい。

5.2 関数

関数とは何らかの値を返す命令文です。

関数名()

の形式で記述します。関数には数値関数と文字列関数の２種類があります。

以下に利用可能な数値関数、文字列関数を示します。

数値関数：

ABS()	絶対値の取得
ANA()	アナログ入力
ASC()	文字から文字コードへの変換
EEPREAD()	仮想 EEPROM のからのデータ読み込み
FREE()	プログラム領域の残りバイト数の取得
I2CR()	I2C スレーブデバイスからのデータ受信
I2CW()	I2C スレーブデバイスへのデータ送信
IN()	デジタル入力
INKEY()	キー入力の読み取り
PEEK()	指定アドレスの値参照
RND()	乱数の発生
SHIFTIN()	デジタルシフトアウト入力
TICK()	起動からの経過時間取得
VPEEK()	画面指定位置の文字コード参照

数値関数は数値を返します。

数値関数は式、代入式、各コマンドや関数の引数として利用できます。

文字列関数：

CHR$()	画文字コードから文字への変換
BIN$()	数値から 2 進数文字列への変換
HEX$()	数値から 16 進数文字列への変換

文字列関数は文字列を返します。

文字列関数は PRINT 文の引数にて利用できます。

各関数の使い方については、「8 各コマンド・関数」を参照して下さい。

5.3 数値・変数・演算子・式・文

■ 数値

「Tiny BASIC for Arduino STM32」で使える数値は整数型のみとなります。
また整数型は 16 ビット幅、有効範囲は-32767～32767 となります。式、数値定数、数値関数、変数、配列変数はすべてこれに従います。

数値の表記は次の形式が可能です。

10 進数表記(-32767～32767)　　：-1、-32767，100
16 進数表記($+1～4 桁)　　：$1345，$abcd，$Abcd

■ 変数

変数とは数値を格納できる保存領域です。数値と同様に式に利用できます。
格納できる数値は 16 ビット幅、有効範囲は-32767～32767 となります。
変数は通常の変数の他に配列変数があります

変数の表記

通常の変数　　：変数名 A～Z の 1 文字で表記、26 個利用可能
例) A=4, Z=Z+1
配列変数　　：@(添え字)の形式で添え字は数値で 0～99(@(0)～@(99))まで利用可能
添え字の数値には式、変数等の利用が可能
例)@(0)=30/5, @(A+1)=5

■ 演算子

数値演算で利用できる演算子を示します。
記述例の A,B には数値、変数、配列変数、カッコで囲んだ式が利用できます。

算術演算子

演算子	説明	記述例
+	足し算	A+B　A と B を足す
-	引き算	A-B　A から B を引く
*	掛け算	A*B　A と B の積
/	割り算	A/B　A を B で割る
%	剰余算	A%B　A を B で割った余り

ビット演算子

演算子	説明	記述例
&	ビット毎の AND 演算	A&B　A と B のビット毎の AND
|	ビット毎の OR 演算	A|B　A と B のビット毎の OR
>>	ビット右シフト演算	A>>B　A を右に B ビットシフト
<<	ビット左シフト演算	A<<B　A を左に B ビットシフト
~	ビット毎の反転	~A　A の各ビットを反転
^	ビット毎の非排他 OR	A^B　A と B の非排他 OR

比較演算、論理演算の論理反転は、0 が偽、0 以外が真となります。
0 以外が真となりますので、1、-1 はとも真となります。

比較演算子

演算子	説明	記述例
=	等しいかを判定	A=B　A と B が等しければ真,異なれば偽
!=	異なるかを判定	A<>B　A と B が異なれば真,等しければ偽
<	小さいかを判定	A<B　A が B 未満であれば真、そうでなければ偽
<=	小さいまたは等しいかを判定	A<=B　A が B 以下であれば真、そうでなければ偽
>	大きいかを判定	A>B　A が B より大きければ真、そうでなければ偽
>=	大きいまたは等しいかを判定	A>=B　A が B 以上であれば真、そうでなければ偽

論理演算子

演算子	説明	記述例
AND	論理積	A AND B　A.B が真なら真、でなければ偽
OR	論理和	A OR B　A,B どちらかが真なら真、でなければ偽
!	否定	!A　A が真なら偽、偽なら真

■ **演算子の優先順序**

演算子の優先度を下記に示します。優先度の数値が小さいほど優先度が高くなります。
計算結果が意図した結果にならない場合は、優先度の確認、括弧をつけて優先度を上げる等の対応を行って下さい。

演算子の優先度

優先度	演算子
1	括弧で囲った式
2	!, ~
3	*, / , % , &, | . << , >>,^
4	+ , - ,
5	=, <> , != , >, >= , < , <= , AND , OR ,

■ 式

式とは 1、1+1、A、A+1、ABS(-1) などの演算・値の評価を伴う記述をいいます。
式はプログラム実行にて計算・評価が行われ、１つの整数値の値として振る舞います。
変数への値の代入、コマンド、条件判定(IF 文)、関数に引数に式が用いられた場合は、評価後の値がコマンドおよび関数に渡されます。

式の利用：

変数への値の代入(代入命令 LET は省略可能)

```
LET A=(1+10)*10/2
A=5*5+3*2
@(I+J) = I*J
```

コマンド・関数の引数

```
LOCATE X+I*2, J:PRINT “*”
OUT PC13, I>J
A=EEPREAD(I+J+1)/2
```

■ 文

文とはコマンド、式、関数、コマンド、コメント、ラベルを組み合わせて記述した命令です。
コマンドはコロン:を使って１行に記述することができます。

```
10 I=I+1:LOCATE 0,I:COLOR 7,0:PRINT “Hello”:GOTO 50
```

コメント文

文において、REM、および省略形のシングルクオーテーション’を使った以降の文はコメントとなります。

例：

```
10 ‘Sample program
20 REM Sample program
30 PRINT “Hello”:’print hello
```

ラベル

行の先頭のダブルクオテーションはラベルとして動作します。
ラベル自体は実行時に何も行いません。GOTO 文、GOSSUB 文のジャンプ先の指定で利用します。ラベルの後ろにはコマンドを続けて記述することができます。

```
10 “LOOP”:PRINT “Hello”
30 GOTO “LOOP”
```

5.4 制御文

制御文は制御命令を使った文です。複数のキーワードを使って制御文として動作します。

FOR 文の例：

```
FOR A=0 TO 10 STEP 5
PRINT A
NEXT A
```

プログラムは RUN コマンドを実行することにより、先頭行から行番号順に逐次実行されます。
この逐次実行の流れは、制御文を用いることで条件分岐、繰り返しを行うことが出来ます。
この制御文としては、次の命令があります。

GOTO	指定行へのジャンプ
GOSUB	サブルーチンの呼び出し
RETURN	GOSUB 文によるサブルーチン呼び出しからの復帰
IF	条件判定
FOR	繰り返し
END	プログラムを終了する

各制御文の詳細については「6 各制御文」を参照して下さい。

6 各制御文の詳細

6.1 GOTO 指定行への分岐（制御命令）

■ 書式

GOTO 行番号

GOTO “ラベル”

■ 説明

プログラムの実行を行番号またはラベルで指定した行に移ります。

行番号には数値、式（変数、関数を含む）が利用できます。

ラベルを指定した場合は、行先頭に同じラベルがある行に移ります。

ラベルはダブルクオテーション(“)で囲って指定します。

```
10 I=0
20 “LOOP”
30 PRINT “@”
40 I=I+1:IF I=5 GOTO 60
50 GOTO “LOOP”
60 END
```

上記の例では、40 行の GOTO で 60 行に移動、50 行のラベル指定で 20 行に移動しています。

ラベルを使うとプログラムの流れがわかりやすくなります。

ラベルの使用は行番号に比べるとラベル文字列検索の分、若干処理に時間がかかります。

行先頭以外にラベルがある場合は、GOTO 文、GOSSUB 文のジャンプ先として参照はされません。

■ エラーメッセージ

Undefined line number or label	: 条件
Overflow	: 指定した数値が-32768 ～ 32767 を超えている
Syntax error	: 文法エラー、書式と異なる利用を行った

■ 利用例

```
100 GOSUB "SUB01"
110 GOSUB "SUB02"
120 N=0
130 "LOOP"
140 PRINT "N=";N
150 N=N+1:IF N<5 GOTO "LOOP"
160 END
170 "SUB01"
180 PRINT "SUB01"
190 RETURN
200 "SUB02"
210 PRINT "SUB02"
220 RETURN
```

6.2 GOSUB サブルーチンの呼び出し（制御命令）

■ 書式

GOSUB 行番号

GOSUB “ラベル”

■ 説明

プログラムの実行を一旦、行番号またはラベルで指定した行に移ります。

移った先で RETURN 命令により、呼び出した GOSUB 命令の次の位置に戻ります。

行番号には数値、式（変数、関数を含む）が利用できます。

ラベルを指定した場合は、行先頭に同じラベルがある行に移ります。

ラベルはダブルクオテーション(“)で囲って指定します。

```
10 GOSUB “PRN_LOGO”
20 GOSUB “PRN_DATE”
30 GOSUB 200
40 END
50 “PRN_LOGO”
60 PRINT “Tiny BASIC for Arduino STM32”
70 RETURN
80 “PTN_DATE”
90 PRINT “Edition V0.3 2017/04/01”
100 RETURN
200 PRINT “Ready”
210 RETURN
```

上記の例では、10 行、20 行でラベルを指定してサブルーチンを呼び出しています。

30 行では、行番号を指定してサブルーチンを呼び出しています。

ラベルを使うことで、プログラムの実行がわかりやすくなります。

ラベルの使用は行番号に比べるとラベル文字列検索の分、若干処理に時間がかかります。

行先頭以外にラベルがある場合は、GOTO 文、GOSSUB 文のジャンプ先として参照はされません。

■ エラーメッセージ

Undefined line number or label	: 条件
Overflow	: 指定した数値が-32768 ～ 32767 を超えている
GOSUB too many nested	: GOSUB のネスト数が規定を超えた
Syntax error	: 文法エラー、書式と異なる利用を行った

■ 利用例

```
100 GOSUB "SUB01"
110 GOSUB "SUB02"
120 N=0
130 "LOOP"
140 PRINT "N=";N
150 N=N+1:IF N<5 GOTO "LOOP"
160 END
170 "SUB01":PRINT "SUB01":RETURN
200 "SUB02":PRINT "SUB02":RETURN
```

6.3 RETURN GOSUB 呼び出し元への復帰（制御命令）

■ 書式

RETURN

■ 説明

直前に呼び出された GOSUB の次の処理に復帰します。

詳細は「6.2 GOSUB サブルーチンの呼び出し（制御命令）」を参照して下さい。

■ エラーメッセージ

Undefined line number or label	: 条件
Overflow	: 指定した数値が-32768 ～ 32767 を超えている
RETURN stack underflow	: GOSUB の呼び出しがないのに RETURN を実行
Syntax error	: 文法エラー、書式と異なる利用を行った

■ 利用例

```
100 GOSUB "SUB01"
110 GOSUB "SUB02"
120 N=0
130 "LOOP"
140 PRINT "N=";N
150 N=N+1:IF N<5 GOTO "LOOP"
160 END
170 "SUB01"
180 PRINT "SUB01"
190 RETURN
200 "SUB02"
210 PRINT "SUB02"
220 RETURN
```

6.4 IF 条件判定（制御命令）

■ 書式

IF 条件式 実行文 1

IF 条件式 実行文 1 ELSE 実行文 2

■ 説明

条件式の真偽判定を行い、真の場合は時実行文 1 を実行します。

偽の場合 ELSE 文の指定があれば実行文 2 を実行します。なければ次の行にスキップします。

真偽は次の条件となります。

真：値が 0 以外の数値

偽：値が 0

真偽判定を行う、条件式には次の指定が可能です。

式　　：例　A > 5、A+1、A & 1 、INKEY() = ASC(“A”)

定数　　：例　HIGH 、LOW、PA1

数値　　：例　123、$50

実行文には IF、GOTO、GOSUB 等の制御命令を使うことも可能です。

（注意）ELSE 利用の制約(ぶら下がり ELSE に関する補足)

IF 文をネストして利用した場合、ELSE 文は直前の IF 文に対応します。

例：`IF A=1 IF B=1 ? "A,B=1" ELSE ? "A=1,B<>1"`

上記の ELSE は 2 番目の IF 文に対応します。

この ELSE 文は直前の IF 文に対応の条件で、ELSE 文にさらに IF 文をネストすることが出来ます。

例: `IF A=1 IF B=1 ? "A,B=1" ELSE IF B=2 ? "A=1,B=2" ELSE IF B=3 ?"A=1,B=3"`

上記では、`A=1,B=1,2,3` の条件に対して対応する表示メッセージを表示する例です。

エラーメッセージ

IF without condition　：条件

Overflow　　：指定した数値が-32768 ～ 32767 を超えている

■ 利用例

```
10 A=INKEY()
20 IF A=ASC(“Y”) | PRINT
```

6.5 FOR TO ～ NEXT 繰り返し実行（制御命令）

■ 書式

FOR 変数=初期値 TO 最終値　実行文(複数行可能) NEXT 変数名

FOR 変数=初期値 TO 最終値 STEP 増分　繰り返し実行文(複数行可能) NEXT 変数名

■ 説明

変数を初期値から最終値まで増やし、FOR から NEXT の間の実行文を繰り返し実行します。

変数が最終値に達した時点で繰り返しを止め、NEXT の次の命令の実行を行います。

STEP にて増分を指定しない場合、増分は 1 となります。

STEP を用いた場合は、マイナス値を含め任意の増分の指定が可能です。

```
10 PRINT “start.”
20 FOR I=0 TO 5 STEP 2
30 PRINT I
40 NEXT I
50 PRINT “done.”
```

実行結果

```
start
0
2
4
done.
```

上記の例では I を 0 から 5 まで、2 ずつ増加して 30 行の命令を繰り返し実行します。

I が 4 の時、増分 2 を足すと終了値 5 を超えた 6 となるので繰り返しを終了し、50 行を実行します。

FOR 文はネストも可能です。

■ エラーメッセージ

FOR without variable	: FOR 文で変数を指定していない
FOR without TO	: FOR 文で TO を指定していなし
NEXT without counter	: NEXT に対応する FOR 文が無い
FOR too many nested	: FOR 文のネストが多すぎる
Syntax error	: 文法エラー、書式と異なる利用を行った
Overflow	: 指定した数値が-32768 ～ 32767 を超えている

■ 利用例

画面の指定位置に*を表示する（FOR 文のネストの例）

```
10 CLS
20 FOR Y=5 TO 10
30 FOR X=5 TO 10
40 LOCATE X,Y:PRINT "*"
50 NEXT X
60 NEXT Y
```

6.6 END プログラムの終了（制御命令）

■ 書式

END

■ 説明

プログラムの実行を終了します。

```
100 GOSUB "SUB01"
110 GOSUB "SUB02"
120 N=0
130 "LOOP"
140 PRINT "N=";N
150 N=N+1:IF N<5 GOTO "LOOP"
160 END
170 "SUB01"
180 PRINT "SUB01"
190 RETURN
200 "SUB02"
210 PRINT "SUB02"
220 RETURN
```

■ エラーメッセージ

Syntax error　　：文法エラー、書式と異なる利用を行った

■ 利用例

特定の条件でプログラムを終了する

```
10 I=0
20 PRINT I
30 IF I=5 END
40 GOTO 20
```

7 定数

「Tiny BASIC for Arduino STM32」では次の定数が利用出来ます。
定数はコマンドの引数や式の中で数値関数と同等に利用出来ます。

■ 1 ビット入出力値

HIGH, LOW

HIGH は 1,LOW は 0 が割り当てられています。
各コマンドの引数、IF 文、式にて利用出来ます。

■ メモリ領域先頭アドレス定数

MEM, VRAM, VAR, ARRAY

SRAM の先頭アドレスからの相対アドレスを参照するための定数です。
詳細は次の通りです。

MEM	: ユーザーワーク領域	サイズ 1024 バイト
VRAM	: 画面表示用メモリ（80×25）	サイズ 2048 バイト
VAR	: 変数領域（A～Z）	サイズ 52 バイト
ARRAY	: 配列変数領域(@(0)～@(99))	サイズ 200 バイト

PEEK、POKE、I2CW、I2CR 等の SRAM メモリを利用するコマンドで利用します。

■ ピン番号定数

```
PA00, PA01, PA02, PA03, PA04, PA05, PA06, PA07, PA08
PA09, PA10, PA11, PA12, PA13, PA14, PA15
PB00, PB01, PB02, PB03, PB04, PB05, PB06, PB07, PB08
PB09, PB10, PB11, PB12, PB13, PB14, PB15
PC13, PC14, PC15
```

GPIO、OUT、IN、ANA、SHIFTOUT、SHIFTIN コマンド・関数のピン番号の指定に利用します。
各ピン番号定数に実際のピン番号 0 ～ 34 が割り当てられています。

■ GPIO モード設定定数

OUTPUT_OD, OUTPUT, INPUT_PU, INPUT_PD, ANALOG, INPUT_FL

GPIO コマンドのモード設定を行うための定数です。

■ ビット方向定数

LSB, MSB

ビット送信等で上位、下位を指定するための定数です。
SHIFTIN,SHFITOUT コマンドで利用します。

8 各コマンド・関数の詳細

1.1　LIST　プログラムリストの表示（ｃコマンド）

■ 書式

LIST

LIST 表示開始行番号

LIST 表示開始行番号, 表示終了行番号

■ 引数

表示開始行番号：表示を開始する行番号（1 ～ 32767）

表示終了行番号：表示を終了する行番号（1 ～ 32767）

■ 説明

プログラムリストの表示を行います。

引数を指定しない場合は、全てのプログラムを表示します。

表示開始番号を指定した場合は、その番号以降のプログラムリストを表示します。

表示開始番号、表示終了番号を指定した場合は、そこ範囲のプログラムリストを表示します。

表示したプロフラムリストは、カーソルを移動して編集することが出来ます。

編集後は必ず[ENTER]キーを押して入力確定を行って下さい。

■ エラーメッセージ

Syntax error　　：文法エラー、書式と異なる利用を行った、引数に数値以外を指定した

Overflow　　：指定した数値が-32768 ～ 32767 を超えている

■ 利用例

```
LIST
10 I=0
20 PRINT "*";
30 I=I+1:IF I<10 GOTO 20
40 PRINT
50 END
OK

list 20,30
20 PRINT "*";
30 I=I+1:IF I<10 GOTO 20
OK
```

1.2　REM　コメント（一般コマンド）

■ 書式

REM コメント

‘ コメント

■ 引数

コメント　　：任意の文字列

■ 説明

プログラムに説明等の記載を行います。

‘(シングルクオート)は REM の省略形です。

REM および’以降の文字以降はすべてコメントとみなし、プログラムとして実行されません。

プログラムの先頭行にコメントを付けた場合は、FILES コマンドで保存プログラム一覧を表示したたきに、各プログラムの見出しとなります。

■ エラーメッセージ

Syntax error　　：文法エラー、書式と異なる利用を行った

■ 利用例

```
LIST
1 REM Print starts
10 I=0:’initialize value
20 PRINT "*";
30 I=I+1:IF I<10 GOTO 20
40 PRINT
50 END
```

1.3　LET　変数に値を代入（一般コマンド）

■ 書式

LET 変数=値

LET 配列変数=値,値,値,値

■ 引数

変数　　：変数 A ～ Z、または配列変数 @(0) ～ @(99)

配列変数　：配列変数 @(0) ～ @(99)

値　　：式、数値、変数、配列変数、数値定数

■ 説明

値を変数に代入します。

値は式、数値、定数、変数、配列変数等の整数です。

配列変数への代入は、複数の値を指定できます。指定した添え字を起点に順番に代入します。

LET @(3)=1,2,3,4,5

上記の記述は、

LET @(3)=1: LET @(4)=2: LET @(5)=3: LET @(5)=4: LET @(5)=5

と同じです。

LET は省略可能です。次の 2 は同じ結果となります。

LET A=A+1

A=A+1

■ エラーメッセージ

Syntax error　　：文法エラー、書式と異なる利用を行った

Overflow　　：指定した数値が-32768 ～ 32767 を超えている

■ 利用例

```
LIST
1 REM Print starts
10 LET I=0:’initialize value
20 PRINT "*";
30 LET I=I+1:IF I<10 GOTO 20
40 PRINT
50 END
```

1.4　RENUM　行番号の振り直し（システムコマンド）

■ 書式

RENUM

RENUM 開始行番号

RENUM 開始行番号, 増分

■ 引数

開始番号：振り直しをする新しい行番号の開始番号 （1 ～ 32767）

増　　分：行番号の増分（1 ～ 32767）

■ 説明

プログラム全体の行番号を指定した条件にて振り直します。

行番号を指定した開始番号から指定した増分で振り直します。

行番号は 1～32767 の範囲まで振ることが出来ます。

引数を省略した場合は、行番号を 10 行から 10 間隔で振り直します。

開始番号だけを指定した場合は、指定した開始番号から 10 間隔再振り直します。

開始番号と増分を指定した場合は、指定した開始番号から指定した増分で振り直します。

振り直しにおいて、GOTO 文、GOSUB 文のとび先の行番号も正しく更新されます。

開始行番号には変数、式の指定は行えません。必ず数値を指定して下さい。

■ エラーメッセージ

Syntax error	：文法エラー、書式と異なる利用を行った、引数に数値以外を指定した
Illegal value	：振り直しをする新しい行番号が有効範囲を超えている
Overflow	：指定した数値が-32768 ～ 32767 を超えている

■ 利用例

```
LIST
10 I=0
20 PRINT "*";
30 I=I+1:IF I<10 GOTO 20
40 PRINT
50 END
OK
RENUM 100,10
OK
LIST
100 I=0
110 PRINT "*";
120 I=I+1:IF I<10 GOTO 110
130 PRINT
140 END
```

1.5　RUN　プログラムの実行（システムコマンド）

■ 書式

RUN

■ 引数

なし

■ 説明

プログラムの実行を行います。
実行中のプログラムを強制終了するには[ESC]キーを 2 回連続して押すか、[CTRL+C]キーを押して下さい。WAIT 命令で長い時間待ちを行っている場合は、強制終了に時間がかかる場合があります。

実行中は、カーソルが非表示となります。
プログラム実行中にエラーが発生した場合は実行を終了します。

■ エラーメッセージ

Syntax error　　：文法エラー、書式と異なる利用を行った、引数に数値以外を指定した

■ 利用例

```
LIST
10 I=0
20 PRINT "*";
30 I=I+1:IF I<10 GOTO 20
40 PRINT
50 END
OK

RUN
**********
OK
```

1.6　NEW　プログラムの消去（システムコマンド）

■ 書式

NEW

■ 引数

なし

■ 説明

プログラム領域のプログラム、変数、配列変数を消去します。

■ エラーメッセージ

Syntax error　　：文法エラー、書式と異なる利用を行った、引数に数値以外を指定した

■ 利用例

```
LIST
10 I=0
20 PRINT "*";
30 I=I+1:IF I<10 GOTO 20
40 PRINT
50 END
OK

NEW
OK
LIST
OK
```

1.7　FREE　プログラム領域の残りバイト数の取得（数値関数）

■ 書式

FREE()

■ 引数

なし

■ 戻り値

プログラム領域の残りバイト数 0 ～ 2047

■ 説明

プログラム領域の残りバイト数を返します。
プログラム領域のサイズは 2048 バイトあります。そのうち 1 バイトは終端チェック用に利用しています。残り 2047 バイトがプログラミングに利用出来る容量です。
プログラムサイズは次のように求めることが出来ます。

```
PRINT 2047-FREE()
```

■ エラーメッセージ

Syntax error　　：文法エラー、書式と異なる利用を行った

■ 利用例

プログラムサイズを調べる

```
PRINT 2047-FREE()
287
OK
```

1.8　SAVE　内部フラッシュメモリへのプログラム保存（一般コマンド）

■ **書式**

SAVE

SAVE プログラム番号

■ **引数**

プログラム番号：　0 ～ 3

■ **説明**

プログラムをマイコン内のフラッシュメモリに保存します。

プログラムは最大で 4 つ保存可能です。保存先はプログラム番号 0～3 で指定します。

プログラム番号を指定しない場合、プログラム番号に保存します。

プログラム番号の指定には、変数や式の指定は出来ません。

（注意）SAVE コマンドはプログラム内での実行は出来ません。実行しても無視されます。
直接コマンド入力を行った場合のみ有効です。

■ **エラーメッセージ**

Syntax error	: 書式と異なる利用を行った、プログラム番号に変数、式を指定した
Illegal value	: プログラム番号の指定が 0～3 の範囲外である
Overflow	: 指定した数値が-32768 ～ 32767 を超えている

■ **利用例**

プログラム番号 1 にプログラムを保存する

```
SAVE 1
OK
```

1.9　LOAD　内部フラッシュメモリからプログラムを読み込む（一般コマンド）

■ 書式

LOAD

LOAD プログラム番号

■ 引数

プログラム番号：　0 ～ 3

■ 説明

マイコン内のフラッシュメモリからプログラムを読み込みます。
読み込み前のプログラム、変数、配列変数は消去されます。

プログラム番号 0～3 で指定します。プログラムは最大で 4 つ保存可能です。
プログラム番号の指定しない場合、プログラム番号 0 を読み込みます。
プログラム番号の指定には、変数や式の指定は出来ません。

（注意）LOAD コマンドはプログラム内での実行は出来ません。実行しても無視されます。
直接コマンド入力を行った場合のみ有効です。

■ エラーメッセージ

Syntax error	: 書式と異なる利用を行った、プログラム番号に変数、式を指定した
Illegal value	: プログラム番号の指定が 0～3 の範囲外である
Program not found	: 指定したプログラム番号にプログラムが保存されていない
Overflow	: 指定した数値が-32768 ～ 32767 を超えている

■ 利用例

プログラム番号 1 を読み込む

```
LOAD 1
OK
```

1.10 PRINT　画面への文字表示（一般コマンド）

■ 書式

PRINT [桁指定][区切り][文字列|数値] [区切り]…

※ カッコ[ ]は省略可能を示す

※ |はいずれのうち１つを示す

※ …は可変を示す

■ 引数

桁指定　　: #数値 または #-数値 の形式で指定する
　　例:#3 、 #-3

区切り　　: セミコロン';' または カンマ','

文字列　　: 文字列定数または文字列関数

数値　　: 数値定数、変数、配列変数、または数値関数

■ 説明

指定した文字列、数値をカーソル位置に表示します。

表示要素である文字列、数値は区切り文字のセミコロン';'、またはカンマ','にて連結して表示することが出来ます。

連結表示において、数値は桁指定により指定した桁幅にて等間隔で表示します。

桁指定においてマイナスの数値を指定した場合は、間隔を 0 で補完します。

正の数値の場合は空白文字で補完します。

PRINT 文の引数の最後に';'または','が付加されている場合は改行しません。

付加されていない場合は引数の内容を表示後、改行します。

■ エラーメッセージ

Syntax error　　: 文法エラー、書式と異なる利用を行った、引数に数値以外を指定した

Overflow　　: 指定した数値が-32768 ～ 32767 を超えている

■ 利用例

```
PRINT #2,1;":";2;":";3
 1: 2: 3
OK

PRINT #-2,1;":";2;":";3
01:02:03
OK

A=123:PRINT A;"=$";HEX$(A)
123=$7B
OK
```

1.11 INPUT　数値の入力（一般コマンド）

■ 書式

INPUT 変数
INPUT 変数,オーバーフロー時の既定値
INPUT プロンプト 変数
INPUT プロンプト 変数, オーバーフロー時の既定値

■ 引数

変数	: 入力した値を格納する変数または配列変数 A ～ Z、@(0) ～ @(99)
プロンプト	: 文字列定数 “プロンプト”
オーバーフロー時の既定値	: -32767～32767

■ 説明

現在のカーソル位置にて数値の入力をし、指定した変数にその値を格納します。
キーボードから入力できる文字は数値 0～9、入力訂正の[BS]、[DEL]、入力確定の[Enter]です。
それ以外の入力はできません。

引数に変数のみを指定した場合は、”変数名=”を表示しその後ろの位置から数値を入力します。
プロンプトを指定した場合は、そのプロンプトを表示しその後ろに位置から数値を入力します。

オーバーフロー時の既定値を指定した場合、入力値した数値がオーバーフローを発生した場合にオーバーフロー時の既定値を変数に設定します。オーバーフロー既定値を設定していない場合に、オーバーフローが発生した場合は、Overflow エラーとなります。

■ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った、引数に数値以外を指定した
Overflow	: 指定した数値が-32768 ～ 32767 を超えている

■ 利用例

```
INPUT A
A=1234
OK

INPUT “Value=” A
Value=1234
OK
```

1.12 CLS　画面表示内容の全消去（一般コマンド）

■ 書式

CLS

■ 引数

なし

■ 説明

画面上に表示している内容を全て消します。
行番号を指定した開始番号から指定した増分で振り直します。
カーソルは画面の先頭左上に移動します。

（注意）直前に COLOR コマンドにて文字色、背景色の指定を行っている場合、画面全体に背景色が適用されます。

■ エラーメッセージ

Syntax error　　：文法エラー、書式と異なる利用を行った

■ 利用例

```
CLS
OK
```

1.13 COLOR　文字色の設定（一般コマンド）

■ 書式

COLOR 文字色

COLOR 文字色, 背景色

■ 引数

文字色： 色コード　0～9

背景色： 色コード　0～9

■ 説明

文字色の設定を行います。指定した色は以降の文字表示に反映されます。

文字色、背景色で指定する色コードに対する色は次の表の通りです。

表 1 色コード

色コード	色
0	黒
1	赤
2	緑
3	茶
4	青
5	マゼンタ
6	シアン
7	白(デフォルト)
8	黄

（注意）利用するターミナルソフトにより色が正しく表示されない場合があります。
属性指定との併用では正しく表示されない場合があります。
画面を[CTRL-R]、[Page UP]、[Page Down]キーにて再表示した場合、
色情報は欠落します。

■ エラーメッセージ

Syntax error　：文法エラー、書式と異なる利用を行った
Illegal value　：色コードに範囲外の値を指定した
Overflow　：指定した数値が-32768 ～ 32767 を超えている

■ 利用例

ランダムな色で＊を表示する

```
10 FOR I=0 TO 10
20 FOR J=0 TO 10
30 COLOR RND(8): ? "*";
35 WAIT 100
40 NEXT J
50 ?
60 NEXT I
```

1.14 ATTR　文字表示属性の設定（一般コマンド）

■ 書式

ATTR 属性

■ 引数

属性： 属性コード 0 ～ 4

■ 説明

文字の表示属性を設定します。指定した表示属性は以降の文字表示に反映されます。

属性に指定する属性コードは次の表の通りです。

表 2 属性コード

属性コード	機能
0	標準(デフォルト)
1	下線
2	反転
3	ブリンク
4	ボールド

（注意）利用するターミナルソフトにより色が正しく表示されない場合があります。
色指定との併用指定を行った場合、正しく表示されない場合があります。
画面を[CTRL-R]、[Page UP]、[Page Down]キーにて再表示した場合、
色情報は欠落します。

■ エラーメッセージ

Syntax error　：文法エラー、書式と異なる利用を行った
Illegal value　：属性コードに範囲外の値を指定した
Overflow　：指定した数値が-32768 ～ 32767 を超えている

■ 利用例

Hello,world を反転表示する

```
10 CLS
20 LOCATE 5,5
30 ATTR 2:? "Hello,world"
40 ATTR 0
```

1.15 LOCATE　カーソルの移動（一般コマンド）

■ 書式

LOCATE 横位置, 縦位置

■ 引数

横位置： 画面上の横位置 0 ～ 79（コンソール出力の場合）

縦位置： 画面上の縦位置 0 ～ 24（コンソール出力の場合）

■ 説明

カーソルを指定した位置に移動します。

縦横位置それぞれの指定に 0 以下の数値を指定した場合、それぞれの位置は 0 となります。
横位置に 79 を超える数値を指定した場合、横位置は 79 となります。縦位置に 24 を超える数値を指定した場合、縦位置は 24 となります。

■ エラーメッセージ

Syntax error　　：文法エラー、書式と異なる利用を行った

Overflow　　：指定した数値が-32768 ～ 32767 を超えている

■ 利用例

画面の指定位置にメッセージを表示する

```
10 CLS
20 LOCATE 5,5:PRINT "Hello,world."
30 LOCATE 6,6:PRINT "TinyBASIC"
40 LOCATE 7,7:PRINT "Thank you."
```

実行結果

```
Hello,world.
 TinyBASIC
  Thank you.
```

1.16 VPEEK　画面指定位置の文字コード参照（数値関数）

■ 書式

VPEEK(横位置 , 縦位置)

■ 引数

横位置： 0～79

縦位置： 0～24

■ 戻り値

指定位置に表示されている文字の文字コード（0 ～ 255）

■ 説明

画面上の指定位置に表示されている文字の文字コードを取得します。

引数の指定位置が範囲外の場合は 0 を返します。

■ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Overflow	: 指定した数値が-32768 ～ 32767 を超えている
'(' or ')' expected	: ’(‘ または ‘)’が無い

■ 利用例

画面最上位の行に表示されている文字を表示する

```
10 FOR I=0 TO 79
20 C=VPEEK(I,0)
30 PRINT CHR$(C);
40 NEXT I
50 PRINT
```

1.17 ABS　絶対値の取得（数値関数）

■ 書式

ABS(値)

■ 引数

値　　：-32768 ～ 32767
　　式、変数、配列変数、数値、数値定数の指定が可能

■ 戻り値

指定した値の絶対値

■ 説明

指定した値の絶対値を返します。

■ エラーメッセージ

Syntax error	：文法エラー、書式と異なる利用を行った
Overflow	：指定した数値が-32768 ～ 32767 を超えている
'(' or ')' expected	：’(‘ または ‘)’が無い

■ 利用例

```
10 FOR I=-3 TO 3
20 PRINT ABS(I)
30 NEXT I

RUN
3
2
1
0
2
3
```

1.18 RND　乱数の発生（数値関数）

■ 書式

RND(値)

■ 引数

値　　：0 ～ 32767

式、変数、配列変数、数値、数値定数の指定が可能

■ 戻り値

0 から指定した値未満の乱数

■ 説明

1 から指定した値を含む範囲の乱数を発生させ、その値を返します。

■ エラーメッセージ

Syntax error	：文法エラー、書式と異なる利用を行った
Overflow	：指定した数値が-32768 ～ 32767 を超えている
'(' or ')' expected	：’(‘ または ‘)’が無い

■ 利用例

ジャンケンの表示

```
10 FOR I=1 TO 5
20 R=RND(3)
30 IF R=0 PRINT “Gu”
40 IF R=1 PRINT “Cyoki”
50 IF R=2 PRINT “Paa”
60 NEXT I

RUN
Gu
Paa
Paa
Cyoki
```

1.19 ASC　文字から文字コードへの変換（数値関数）

■ 書式

ASC(文字)

■ 引数

文字： “文字”

“A” の形式とし、ダブルクオテーションで文字を囲みます

■ 戻り値

指定文字に対応する文字コード（0 ～ 255）

■ 説明

指定した文字に対応する文字コードを返します.

指定した文字列が 1 文字で無い場合は、エラーとなります。

■ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Illegal value	: 文字数が不当
'(' or ')' expected	: ’(‘ または ‘)’がない

■ 利用例

アルファベット A～Z を表示する

```
10 C=ASC("A")
20 FOR I=0 TO 25
30 PRINT CHR$(C+I);
40 NEXT I
50 PRINT

run
ABCDEFGHIJKLMNOPQRSTUVWXYZ
OK
```

1.20 CHR$　画文字コードから文字への変換（文字列関数）

■ 書式

CHR$(文字コード)

■ 引数

文字コード：　0～127, 160～255

■ 戻り値

指定位置に表示されている文字の文字コード（0 ～ 255）

■ 説明

指定した文字コードに対応する文字を返します。

本関数は PRINT の引数として利用可能です。

範囲外の値を指定した場合は空白文字(" ")を返します。

■ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
'(' or ')' expected	: '(' または ')'が無い
Overflow	: 指定した数値が-32768 ～ 32767 を超えている

■ 利用例

アルファベット A～Z を表示する

```
10 C=ASC("A")
20 FOR I=0 TO 25
30 PRINT CHR$(C+I);
40 NEXT I
50 PRINT

run
ABCDEFGHIJKLMNOPQRSTUVWXYZ
OK
```

1.21　BIN$　数値から 2 進数文字列への変換（文字列関数）

■ 書式

BIN$(数値)

BIN$(数値 , 桁数)

■ 引数

数値：　変換対象の整数値(-32768 ～ 32767)

桁数：　出力桁数(0 ～ 16)

■ 戻り値

2 進数文字列(1 桁～16 桁)

■ 説明

指定した数値を 2 進数文字列に変換します。

PRINT 文の引数にて利用可能です。

桁数を指定した場合は数値が指定した桁数に満たない場合は、0 で桁を補います。

指定した桁数を超える場合は数値の桁数を優先します。

桁数をしない場合は、先頭の 0 は付加されません。

■ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
'(' or ')' expected	: '(' または ')'がない
Illegal value	: 桁数の値が正しくない
Overflow	: 指定した数値が-32768 ～ 32767 を超えている

■ 利用例

2 進数で変数の内容を表示する

```
10 A=1234:B=1
20 PRINT BIN$(A)
30 PRINT BIN$(B)
40 PRINT BIN$(A,4)
50 PRINT BIN$(B,4)

run
10011010010
1
10011010010
0001
OK
```

1.22 HEX$　数値から 16 進数文字列への変換（文字列関数）

■ 書式

HEX$(数値)

HEX$(数値 , 桁数)

■ 引数

数値： 変換対象の整数値(-32768 ～ 32767)

桁数： 出力桁数(0 ～ 4)

■ 戻り値

16 進数文字列(1 桁～4 桁)

■ 説明

指定した数値を 16 進数文字列に変換します。

PRINT 文の引数にて利用可能です。

桁数を指定した場合は数値が指定した桁数に満たない場合は、0 で桁を補います。

指定した桁数を超える場合は数値の桁数を優先します。

桁数をしない場合は、先頭の 0 は付加されません。

■ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
'(' or ')' expected	: ’(‘ または ‘)’がない
Illegal value	: 桁数の値が正しくない
Overflow	: 指定した数値が-32768 ～ 32767 を超えている

■ 利用例

1.23 REDRAW　画面表示の再表示（一般コマンド）

■ 書式

REDRAW

■ 引数

なし

■ 説明

表示用メモリ（VRAM）の内容を画面に再表示を行います。
[CTRL-R]、[Page UP]、[Page Down]キーによる再表示をコマンドにて行います。
再表示においては、カーソルの移動は行いません。

再表示においては、個々に色付しけした文字の色、背景色、属性は欠落します。
再表示後は最後に指定した文字色、背景色、属性に統一されます。

■ エラーメッセージ

Syntax error　　：文法エラー、書式と異なる利用を行った

■ 利用例

表示用メモリ（VRAM）に直接書き内容を画面に反映させる

```
10 POKE VRAM+80*5+50,ASC("b")
20 REDRAW
```

1.24 INKEY キー入力の読み取り（数値関数）

■ 書式

INKEY()

■ 引数

なし

■ 戻り値

押したキーの文字コード
キーが押されていない場合は 0

■ 説明

キーボード上の押しているキーの文字コードを取得します。
キーが押されていない場合は、0 を返します。

（注意）[ESC]、[CTRL-C]はプログラム中断用のため、キー入力の読み取りは出来ません。
また、コンソールの制約により読み取れないキーがあります。

■ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
'(' or ')' expected	: ’(‘ または ‘)’がない

■ 利用例

1.25 DATE　現在時刻の表示（一般コマンド）

■ 書式

DATE

■ 引数

なし

■ 説明

内蔵 RTC から現在の時刻を読み、その情報を画面に表示します。

（注意）内蔵 RTC 用のバックアップ電池を搭載していないボードでは、時刻情報は電源 OFF により初期化されます。その場合は SETDATE コマンドにて時刻設定を行って下さい。

■ エラーメッセージ

Syntax error　　：文法エラー、書式と異なる利用を行った

■ 利用例

時刻の設定と表示を行う

```
SETDATE 2017,4,1,12,00,00
OK
DATE
2017/04/01 [Sat] 12:00:03
OK
```

1.26 GETDATE　日付の取得（一般コマンド）

■ 書式

GETDATE 年格納変数,月格納変数,日格納変数,曜日格納変数

■ 引数

年格納変数：取得した西暦年を格納する変数を指定
月格納変数：取得した月を格納する変数を指定
日格納変数：取得した日を格納する変数を指定
曜日格納変数：取得した曜日コードを格納する変数を指定

■ 説明

内蔵 RTC から日付情報を取得し、その値を指定した変数に格納します。
格納される値を次の通りです。
年格納変数：西暦年 4 桁整数 1900 ～ 2036
月格納変数：1 ～ 12
日格納変数：1 ～ 31
曜日格納変数：曜日コード 0 ～ 6（0:日 1:月 2:火 3:水 4:木 5:金 6:土）

（注意）内蔵 RTC 用のバックアップ電池を搭載していないボードでは、時刻情報は電源 OFF により初期化されます。その場合は SETDATE コマンドにて時刻設定を行って下さい。

■ エラーメッセージ

Syntax error　：文法エラー、書式と異なる利用を行った

■ 利用例

現在の日付を取得する

```
DATE
2017/04/01 [Sat] 12:03:05
OK
GETDATE A,B,C,D
OK
PRINT a;b;c;d
2017416
OK
```

1.27 GETTIME　時刻の取得（一般コマンド）

■ 書式

GETTIME 時格納変数,分格納変数 秒格納変数

■ 引数

時格納変数： 取得した時を格納する変数を指定

分格納変数： 取得した分を格納する変数を指定

秒格納変数： 取得した秒を格納する変数を指定

■ 説明

内蔵 RTC から日付情報を取得し、その値を指定した変数に格納します。

格納される値を次の通りです。

時格納変数： 0 ～ 23 整数

分格納変数： 0 ～ 59 整数

秒格納変数： 0 ～ 61 整数(うるう秒考慮)

（注意）内蔵 RTC 用のバックアップ電池を搭載していないボードでは、時刻情報は電源 OFF により初期化されます。その場合は SETDATE コマンドにて時刻設定を行って下さい。

■ エラーメッセージ

Syntax error　　：文法エラー、書式と異なる利用を行った

■ 利用例

現在の時間を取得する

```
DATE
2017/04/01 [Sat] 12:10:51
OK
GETTIME A,B,C
OK
PRINT #-2,a;":";b;":";c
12:11:01
OK
```

(補足) PRINT 文の #-2 は数値を 2 桁(0 付き)で表示する指定です。

1.28 SETDATE　時刻の設定（一般コマンド）

■ 書式

SETDATE 年,月,日,時,分,秒

■ 引数

年：1900 ～ 2036　西暦年 4 桁の整数
月：1 ～ 12　整数
日：1 ～ 31　整数
時：0 ～ 23　整数
分：0 ～ 59　整数
秒：0 ～ 61　整数（うるう秒考慮）

■ 説明

指定した時刻を内蔵 RTC に設定します。

（注意）内蔵 RTC 用のバックアップ電池を搭載していないボードでは、時刻情報は電源 OFF により初期化されます。その場合は SETDATE コマンドにて時刻設定を行って下さい。

■ エラーメッセージ

Illegal value　：指定した引数の数値が有効範囲以外
Syntax error　：文法エラー、書式と異なる利用を行った
Overflow　：指定した数値が-32768 ～ 32767 を超えている

■ 利用例

時刻の設定と表示を行う

```
Y=2017:M=4:D=1:H=12:N=0:S=0
SETDATE Y,M,D,H,N,S
OK
DATE
2017/04/01 [Sat] 12:00:03
OK
```

1.29 TICK　起動からの経過時間取得（数値関数）

■ 書式

TICK()

TICK(モード)

■ 引数

モード：　LOW（または 0）　下位 15 ビットの値取得

　　HIGH（または 1）　上位 15 ビットの値取得

■ 戻り値

モード　LOW 時：　0 ～ 32767　（0.1 秒単位）　　約 54.6 分迄

モード　HIGH 時：　0 ～ 32767　（3276.8 秒単位）　　約 54.6 分×取得数値

■ 説明

起動からの経過時間を返します。

モード指定無、または LOW 指定の場合は 0.1 秒単位の経過時間を返します。

モード指定が HIGH 場合は上位のビットを返します。

LOW の桁は約 54.6 分で桁溢れして 0 に戻ります。桁溢れの発生を判定する場合は HIGH 指定にて上位ビットを参照して下さい。

秒単であれば、内蔵 RTC を使う GETTIME を使う方法もあります。

■ エラーメッセージ

Syntax error	：文法エラー、書式と異なる利用を行った
'(' or ')' expected	：’(‘ または ‘)’がない
Overflow	：指定した数値が-32768 ～ 32767 を超えている

■ 利用例

5 秒間時間待ちループを行う

```
10 RESETTICK
20 IF TICK()<50 GOTO 20
30 PRINT "5sec passed"
```

1.30 RESETTICK　起動からの経過時間カウントのリセット（一般コマンド）

■ 書式

RESETTICK

■ 引数

なし

■ 説明

起動からの経過時間カウンタをリセット（0 を設定）します。

TICK()関数使う前に経過時間カウントをリセットすることで 0 からのカウントを行うことが出来ます。

■ エラーメッセージ

Syntax error　　：文法エラー、書式と異なる利用を行った

■ 利用例

5 秒間時間待ちループを行う

```
10 RESETTICK
20 IF TICK()<50 GOTO 20
30 PRINT "5sec passed"
```

1.31 WAIT　時間待ち（一般コマンド）

■ 書式

WAIT 待ち時間(ミリ秒)

■ 引数

待ち時間： 0 ～ 32767 （単位 ミリ秒）

■ 説明

引数で指定した時間(ミリ秒単位)、時間待ち(ウェイト)を行います。
最大で 32767 ミリ秒（32.8 秒）の時間待ちが可能です。
長い時間待ちを行う必要がある場合は、TICK()や GETTIME を使った方法を検討してください。

（注意）時間待ち中はキー操作によるプログラム中断を行うことは出来ません。
短い時間の指定にてループ処理を行う等の対策を行ってください。

■ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Illegal value	: 待ち時間に範囲外の値を指定した
Overflow	: 指定した数値が-32768 ～ 32767 を超えている

■ 利用例

画面上の指定位置に時刻を 1 秒間隔で更新表示する

```
10 SETDATE 2017,4,1,12,0,0
20 CLS
30 LOCATE 5,5
40 DATE
50 WAIT 1000
60 GOTO 30
```

1.32　GPIO　GPIO 機能設定（一般コマンド）

■ 書式

GPIO ピン番号, モード

■ 引数

ピン番号：0 ～ 34

ピン番号は数値の他に次のピン名（定数）での指定も可能です。

```
PA00, PA01, PA02, PA03, PA04, PA05, PA06, PA07, PA08,
PA09, PA10, PA11, PA12, PA13, PA14, PA15,
PB00, PB01, PB02, PB03, PB04, PB05, PB06, PB07, PB08,
PB09, PB10, PB11, PB12, PB13, PB14, PB15,
PC13, PC14, PC15
```

モード：次の定数

```
OUTPUT_OD：デジタル出力（オープンドレイン）
OUTPUT    ：デジタル出力
INPUT_FL  ：デジタル入力（フロート状態：Arduino の INPUT 指定と同じ）
INPUT_PU  ：デジタル入力（内部プルアップ抵抗有効）
INPUT_PD  ：デジタル（内部プルダウン抵抗有効）
ANALOG    ：アナログ入力
```

■ 説明

ボード上の GPIO ピンの機能設定を行います。

Arduino の pinMode()に相当します。

GPIO ピンを使って信号の入出力を行う場合は、必ず本コマンドによる設定が必要です。

（注意）ピン番号の指定範囲及び定数は、ST32F103C8T6 での利用を想定したものです。
ST32F103C8T6 以外の MPU にて使う場合は、機能設定できない場合があります。

■ エラーメッセージ

Syntax error	：文法エラー、書式と異なる利用を行った
Illegal value	：ピン番号、モードに範囲外の値を指定した
Overflow	：指定した数値が-32768 ～ 32767 を超えている

■ 利用例

PB01 ピンからアナログ入力値を読み取り、その値を画面に随時表示します。

```
10 CLS
20 GPIO PB01,ANALOG
30 A=ANA(PB01)
40 LOCATE 5,5: ? A; "     "
50 GOTO 30
```

1.33 OUT デジタル出力（一般コマンド）

■ 書式

OUT ピン番号, 出力値

■ 引数

ピン番号：0 ～ 34

ピン番号は数値の他に次のピン名（定数）での指定も可能です。

```
PA00, PA01, PA02, PA03, PA04, PA05, PA06, PA07, PA08,
PA09, PA10, PA11, PA12, PA13, PA14, PA15,
PB00, PB01, PB02, PB03, PB04, PB05, PB06, PB07, PB08,
PB09, PB10, PB11, PB12, PB13, PB14, PB15,
PC13, PC14, PC15
```

出力値：

```
LOW または 0　　：0V を出力する
HIGH or 0 以外の値：3.3V を出力する
```

■ 説明

指定ピンから、指定した出力を行います。

出力を行う場合は事前に GPIO コマンドによる機能設定（出力設定）が必要です。

（注意）ピン番号の指定範囲及び定数は、ST32F103C8T6 での利用を想定したものです。
ST32F103C8T6 以外の MPU にて使う場合は、機能設定できない場合があります。

■ エラーメッセージ

Syntax error	：文法エラー、書式と異なる利用を行った
Illegal value	：ピン番号、モードに範囲外の値を指定した
Overflow	：指定した数値が-32768 ～ 32767 を超えている

■ 利用例

Blue Pill ボードの搭載 LED（PC13 ピン）を点滅させます。

```
10 P=PC13
20 GPIO P,OUTPUT
30 OUT P,HIGH
40 FOR I=1 TO 10
50 OUT P,LOW
60 WAIT 300
70 OUT P,HIGH
80 WAIT 300
90 NEXT I
```

1.34　IN デジタル入力（数値関数）

■ 書式

IN(ピン番号)

■ 引数

ピン番号：0 ～ 34

ピン番号は数値の他に次のピン名（定数）での指定も可能です。

```
PA00, PA01, PA02, PA03, PA04, PA05, PA06, PA07, PA08,
PA09, PA10, PA11, PA12, PA13, PA14, PA15,
PB00, PB01, PB02, PB03, PB04, PB05, PB06, PB07, PB08,
PB09, PB10, PB11, PB12, PB13, PB14, PB15,
PC13, PC14, PC15
```

■ 戻り値

取得した値　0(LOW) または　1(HIGH)

■ 説明

指定ピンの入力値を読み取り、その値を返します。

入力を行う場合は事前に GPIO コマンドによる機能設定（入力設定）が必要です。

（注意）ピン番号の指定範囲及び定数は、ST32F103C8T6 での利用を想定したものです。
ST32F103C8T6 以外の MPU にて使う場合は、機能設定できない場合があります。

■ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Illegal value	: ピン番号、モードに範囲外の値を指定した
Overflow	: 指定した数値が-32768 ～ 32767 を超えている
'(' or ')' expected	: ’(‘ または ‘)’がない

■ 利用例

1.35　ANA アナログ入力（数値関数）

■ 書式

ANA(ピン番号)

■ 引数

ピン番号：　0 ～ 7, 16, 17 または以下の定数

```
PA00, PA01, PA02, PA03, PA04, PA05, PA06, PA07,
PB00, PB0
```

■ 戻り値

取得した値　0～4095(12 ビット)

■ 説明

指定ピンのアナログ入力値を読み取り、その値を返します。

アナログ入力を行う場合は事前に GPIO コマンドによる機能設定（アナログ入力）が必要です。

（注意）ピン番号の指定範囲及び定数は、ST32F103C8T6 での利用を想定したものです。
ST32F103C8T6 以外の MPU にて使う場合は、機能設定できない場合があります。

■ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Illegal value	: ピン番号、モードに範囲外の値を指定した
Overflow	: 指定した数値が-32768 ～ 32767 を超えている
'(' or ')' expected	: '(' または ')'がない

■ 利用例

PB01 ピンからアナログ入力値を読み取り、その値を画面に随時表示します。

```
10 CLS
20 GPIO PB01,ANALOG
30 A=ANA(PB01)
40 LOCATE 5,5: ? A; "     "
50 GOTO 30
```

1.36　SHIFTIN デジタルシフトアウト入力（数値関数）

■ 書式

SHIFTIN(データピン番号, クロックピン番号, 入力形式)

■ 引数

データピン番号：　0 ～ 34　データを入力するピン

クロックピン番号：　0 ～ 34　クロックを出力するピン

上記のピン番号は数値の他に次のピン名（定数）での指定も可能です。

```
PA00, PA01, PA02, PA03, PA04, PA05, PA06, PA07, PA08,
PA09, PA10, PA11, PA12, PA13, PA14, PA15,
PB00, PB01, PB02, PB03, PB04, PB05, PB06, PB07, PB08,
PB09, PB10, PB11, PB12, PB13, PB14, PB15,
PC13, PC14, PC15
```

入　力　形　式：入力するデータの順番を下記にて指定

LSB または 0：下位ビットから入力する

MSB または 1：上位ビットから入力する

■ 戻り値

入力値（1 バイト）

■ 説明

クロックにて同期を行い、データピンから 1 バイト分のデータを 1 ビットずつ入力します。

Arduino の shiftIn()と同等の動作をします。

データピン、クロックピンは事前に GPIO コマンドによる機能設定（デジタル入出力）が必要です。

（注意）ピン番号の指定範囲及び定数は、ST32F103C8T6 での利用を想定したものです。

ST32F103C8T6 以外の MPU にて使う場合は、機能設定できない場合があります。

■ エラーメッセージ

Syntax error	：文法エラー、書式と異なる利用を行った
Illegal value	：ピン番号、モードに範囲外の値を指定した
Overflow	：指定した数値が-32768 ～ 32767 を超えている
'(' or ')' expected	：’(‘ または ‘)’がない

■ 利用例

1.37　SHIFTOUT デジタルシフトアウト出力（一般コマンド）

■ 書式

SHIFTOUT データピン番号, クロックピン番号, 出力形式, 出力データ

■ 引数

データピン番号：　0 ～ 34　データを出力するピン

クロックピン番号：　0 ～ 34　クロックを出力するピン

上記のピン番号は数値の他に次のピン名（定数）での指定も可能です。

```
PA00, PA01, PA02, PA03, PA04, PA05, PA06, PA07, PA08,
PA09, PA10, PA11, PA12, PA13, PA14, PA15,
PB00, PB01, PB02, PB03, PB04, PB05, PB06, PB07, PB08,
PB09, PB10, PB11, PB12, PB13, PB14, PB15,
PC13, PC14, PC15
```

出　力　形　式：出力するデータの順番を下記にて指定

LSB または 0：下位ビットから出力する

MSB または 1：上位ビットから出力する

出力データ：　出力するデータ（下位 8 ビットのみ有効）

■ 説明

クロックにて同期を行い、データピンから 1 バイト分のデータを 1 ビットずつ出力します。

Arduino の shiftOut()と同等の動作をします。

データピン、クロックピンは事前に GPIO コマンドによる機能設定（デジタル出力）が必要です。

（注意）ピン番号の指定範囲及び定数は、ST32F103C8T6 での利用を想定したものです。
ST32F103C8T6 以外の MPU にて使う場合は、機能設定できない場合があります。

■ エラーメッセージ

Syntax error	：文法エラー、書式と異なる利用を行った
Illegal value	：ピン番号、モードに範囲外の値を指定した
Overflow	：指定した数値が-32768 ～ 32767 を超えている

■ 利用例

1.38　I2CR　I2C スレーブデバイスからのデータ受信（数値関数）

■ 書式

I2CR(デバイスアドレス,コマンドアドレス,コマンド長,データアドレス,データ長)

■ 引数

デバイスアドレス　　: 0 ～ 127 (\$00 ～ \$7F)（7 ビット指定）
　　I2C スレーブアドレスを 7 ビット形式で指定
コマンドアドレス　　: 0 ～ 32767(\$0000 ～ \$1FFFF)
　　送信するコマンドが格納されている SRAM 内相対アドレス
コマンド長　　: 0 ～ 32767
　　送信するコマンドのバイト数
受信データアドレス　: 0 ～ 32767(\$0000 ～ \$1FFFF)
　　受信データを格納する SRAM 内相対アドレス
データ長　　: 0 ～ 32767
　　送信するデータのバイト数

■ 説明

I2C スレーブデバイスからデータを受信します。
送信先はデバイスアドレスにて I2C スレーブアドレスを指定します。

受信において、コマンド等の制御データの送信が必要な場合は、コマンドアドレス、コマンド長にて送信するデータを指定します。受信のみを行う場合は、コマンドアドレス、コマンド長に 0 を設定します。送信受信するデータは SRAM 先頭からの相対アドレスにて指定します。
相対アドレスの指定には次の定数を利用することで有用な領域への書込み・参照が簡単に行えます。

MEM	: ユーザーワーク領域	サイズ 1024 バイト
VRAM	: 画面表示用メモリ（80×25）	サイズ 2048 バイト
VAR	: 変数領域（A～Z）	サイズ 52 バイト
ARRAY	: 配列変数領域(@(0)～@(99))	サイズ 200 バイト

I2C 通信には次の接続ピンを利用します。

PB6：SCL（I2C クロック）
PB7：SDA（I2C データ）

（注意）SRAM 容量を超える相対アドレス（STM32F103C8T6 の場合は 20480）への参照はエラーとなります。

■ 戻り値

0：正常終了
1：通信バッファに対してデータが長すぎる
2：アドレス送信に NACK が返された
3：データ送信に NACK が返された
4：その他のエラー

■ **エラーメッセージ**

Syntax error	: 文法エラー、書式と異なる利用を行った
Illegal value	: 指定した引数の値が不当である
Out of range value	: 指定した値が有効範囲を超えている
Overflow	: 指定した数値が-32768 ～ 32767 を超えている

■ **利用例**

I2C EEPROM(AT24C256 スレーブアドレス $50)にデータの読み書きを行う

```
100 POKE MEM+0,$00,$00
110 POKE MEM+2,64,65,66,67
120 R=I2CW($50,MEM,2,MEM+2,4)
130 ? "snd r=";R
140 POKE MEM+6,0,0,0,0
150 WAIT 5
160 R=I2CR($50,MEM,2,MEM+6,4)
170 ? "rcv r=";R
180 ? "rcv data:";
190 FOR I=0 TO 3
200 ? PEEK(MEM+6+I);" ";
210 NEXT I
220 ?
```

実行結果

```
run
snd r=0
rcv r=0
rcv data:64 65 66 67
OK
```

1.39 I2CW　I2C スレーブデバイスへのデータ送信（数値関数）

■ 書式

I2CW(デバイスアドレス,コマンドアドレス,コマンド長,データアドレス,データ長)

■ 引数

デバイスアドレス　：0 ～ 127 (\$00 ～ \$7F)（7 ビット指定）
　I2C スレーブアドレスを 7 ビット形式で指定
コマンドアドレス　：0 ～ 32767(\$0000 ～ \$1FFFF)
　送信するコマンドが格納されている SRAM 内相対アドレス
コマンド長　：0 ～ 32767
　送信するコマンドのバイト数
データアドレス　：0 ～ 32767(\$0000 ～ \$1FFFF)
　送信するデータが格納されている SRAM 内相対アドレス
データ長　：0 ～ 32767
　送信するデータのバイト数

■ 説明

I2C スレーブデバイスにデータを送信します。
送信先はデバイスアドレスにて I2C スレーブアドレスを指定します。
送信するデータは SRAM 先頭からの相対アドレスにて指定します。
送信するデータはあらかじめ、設定しておく必要があります。
コマンドとデータの区別はありません。デバイスに対しては、単純にコマンド、データの順に送信しています。
相対アドレスの指定には次の定数を利用することで有用な領域への書込み・参照が簡単に行えます。

MEM	：ユーザーワーク領域	サイズ 1024 バイト
VRAM	：画面表示用メモリ（80×25）	サイズ 2048 バイト
VAR	：変数領域（A～Z）	サイズ 52 バイト
ARRAY	：配列変数領域(@(0)～@(99))	サイズ 200 バイト

I2C 通信には次の接続ピンを利用します。

PB6：SCL（I2C クロック）
PB7：SDA（I2C データ）

（注意）SRAM 容量を超える相対アドレス（STM32F103C8T6 の場合は 20480）への参照はエラーとなります。

■ 戻り値

0：正常終了
1：通信バッファに対してデータが長すぎる
2：アドレス送信に NACK が返された
3：データ送信に NACK が返された
4：その他のエラー

■ **エラーメッセージ**

Syntax error	: 文法エラー、書式と異なる利用を行った
Illegal value	: 指定した引数の値が不当である
Out of range value	: 指定した値が有効範囲を超えている
Overflow	: 指定した数値が-32768 ～ 32767 を超えている

■ **利用例**

I2C EEPROM(AT24C256 スレーブアドレス $50)にデータの読み書きを行う

```
100 POKE MEM+0,$00,$00
110 POKE MEM+2,64,65,66,67
120 R=I2CW($50,MEM,2,MEM+2,4)
130 ? "snd r=";R
140 POKE MEM+6,0,0,0,0
150 WAIT 5
160 R=I2CR($50,MEM,2,MEM+6,4)
170 ? "rcv r=";R
180 ? "rcv data:";
190 FOR I=0 TO 3
200 ? PEEK(MEM+6+I);" ";
210 NEXT I
220 ?
```

実行結果

```
run
snd r=0
rcv r=0
rcv data:64 65 66 67
OK
```

接続している I2C スレーブを調べる

（補足）I2C スレーブアドレスのみ送信し、正常終了(ACK を返した)のアドレスを調べています。

```
10 FOR I=0 TO $7F
20 C=I2CW(I,MEM,0,MEM,0)
30 IF C=0 PRINT HEX$(I,2);" ";
40 NEXT I
50 PRINT :PRINT "done."
```

実行結果

```
run
50
done.
OK
```

1.40　PEEK　指定アドレスの値参照（数値関数）

■ 書式

PEEK(相対アドレス)

■ 引数

相対アドレス：　参照を行う SRAM 領域先頭からの相対アドレス(16 ビット)

■ 戻り値

指定した相対アドレス内の 1 バイトデータ（0～255）

■ 説明

SRAM 先頭からの相対アドレスを指定し、格納されている値（１バイト）を返します。
相対アドレスの指定には次の定数を利用することで有用な領域への参照が簡単に行えます。

MEM	: ユーザーワーク領域	サイズ 1024 バイト
VRAM	: 画面表示用メモリ（80×25）	サイズ 2048 バイト
VAR	: 変数領域（A～Z）	サイズ 52 バイト
ARRAY	: 配列変数領域(@(0)～@(99))	サイズ 200 バイト

ユーザーワーク領域は利用者が自由に利用出来る領域です。
それ以外の領域は BASIC の実行にて利用する領域です。
例として画面上のカーソル位置 X,Y に格納されている文字の参照は次のようになります。

```
C = PEEK(VRAM+Y*80+X)
```

（注意）SRAM の領域外のアドレスを指定した場合、エラーとなりなす。
領域チェックにはマクロ SRAM_SIZE の定義値（デフォルト 20480）を利用しています。
デフォルト値よりも大きい容量のマイコンを利用している場合はこの定義値を修正して下さい。

エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Out of range value	: SRAM の領域外のアドレスを指定した
'(' or ')' expected	: 括弧の指定が正しくない
Overflow	: 指定した数値が-32768 ～ 32767 を超えている

■ 利用例

変数 A から Z をユーザーワーク領域に保存する例

```
10 FOR I=0 TO 51
20 POKE MEM+I,PEEK(VAR+I)
30 NEXT I
```

1.41　POKE　指定アドレスへのデータ書き込み（一般コマンド）

■ 書式

POKE 相対アドレス,データ

POKE 相対アドレス,データ,データ, … データ　（可変個数指定）

■ 引数

相対アドレス：　参照を行う SRAM 領域先頭からの相対アドレス(16 ビット)

データ　　：　書き込むデータ（下位 8 ﾋﾞｯﾄのみ有効）

■ 説明

SRAM 先頭からの指定した相対アドレスに指定したデータを書き込みます。

相対アドレスの指定には次の定数を利用することで有用な領域への書込みが簡単に行えます。

MEM	: ユーザーワーク領域	サイズ 1024 バイト
VRAM	: 画面表示用メモリ（80×25）	サイズ 2048 バイト
VAR	: 変数領域（A～Z）	サイズ 52 バイト
ARRAY	: 配列変数領域(@(0)～@(99))	サイズ 200 バイト

ユーザーワーク領域は利用者が自由に利用出来る領域です。

それ以外の領域は BASIC の実行にて利用する領域です。

指定した相対アドレスに上記以外の領域を指定した場合はエラーとなります。

例として画面上のカーソル位置 X,Y への文字の書込みは次のようになります。

```
POKE VRAM+Y*80+X,ASC(“A”)
```

■ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Out of range value	: SRAM の領域外のアドレスを指定した
Overflow	: 指定した数値が-32768 ～ 32767 を超えている

■ 利用例

変数 A から Z をユーザーワーク領域に保存する例

```
10 FOR I=0 TO 51
20 POKE MEM+I,PEEK(VAR+I)
30 NEXT I
```

1.42 EEPFORMAT　仮想 EEPROM のフォーマット（一般コマンド）

■ 書式

EEPFORMAT

■ 引数

なし

■ 説明

仮想 EEPROM のフォーマットを行います。
仮想 EEPROM とは内部フラッシュメモリの一部をソフトウェアにて EEPROM のように利用しますものです。仮想 EEPROM に書き込んだデータは、マイコンボードの電源を落としても保持されます。
EEPFORMAT コマンドは仮想的に EEPROM として利用するためにフラッシュメモリの初期化処理を行います。既に EEPWTIRE コマンドにてデータの書込みを行っている場合、そのデータを含めて初期化されます。

仮想 EEPROM はフラッシュメモリの２ページ（1024 バイト×2）を利用しています。
フラッシュメモリの書込み回数の制約を考慮し、同じアドレスへの繰り返し書き込みに対して書き込み位置を分散しています。
仮想 EEPROM へのデータ保存は 2 バイト単位（変数のバイト数と同じ）です。
保存可能なデータ量は 254 ワード(508 バイト)となります。

EEPWRITE、EEPREAD コマンドで指定する読書きを指定するアドレスは実際にはアドレスではなく、検索キーです。$0000～$FFFF の間に任意で利用出来ます。
検索キー数はデータ保存量が254ワードの制約から､254以内に抑える必要あります｡仮想EEPROMの機構から数十以内に抑えることを推奨します。

■ エラーメッセージ

Syntax error　　：文法エラー、書式と異なる利用を行った

■ 利用例

```
EEPFORMAT
OK
```

1.43 EEPWRITE　仮想 EEPROM のへのデータ書き込み（一般コマンド）

■ 書式

EEPWRITE アドレス,データ

■ 引数

アドレス　　：仮想 EEPROM 内アドレス $0000 ～ $FFFF(10 進数でも可能)

データ　　：書き込みデータ　32767 ～ 32767（２バイト分）

$0000 ～ $FFFF

■ 説明

仮想 EEPROM の指定したアドレスにデータを書き込みます。

書き込み単位は 2 バイトです。仮想 EEPROM に初めてデータ書き込みを行う場合は事前に EEPFORMAT コマンドで初期化をしておく必要があります。

仮想 EEPROM とは内部フラッシュメモリの一部をソフトウェアにて EEPROM のように利用しますものです。仮想 EEPROM に書き込んだデータは、マイコンボードの電源を落としても保持されます。

仮想 EEPROM はフラッシュメモリの２ページ（1024 バイト×2）を利用しています。

フラッシュメモリの書込み回数の制約を考慮し、同じアドレスへの繰り返し書き込みに対して書き込み位置を分散しています。

保存可能なデータ量は最大で 254 ワード(508 バイト)となります。

EEPWRITE、EEPREAD コマンドで指定する読書きを指定するアドレスは実際にはアドレスではなく、検索キーです。$0000～$FFFF の間に任意で利用出来ます。

検索キー数はデータ保存量が 254 ワードの制約から、254 以内に抑える必要あります。仮想 EEPROM の機構から数十以内に抑えることを推奨します。

■ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
EEPROM out size	: 書き込み容量を超えた
EEPROM bad FLASH	: フラッシュメモリの異常が発生した

■ 利用例

```
EEPWRITE $1234, 123
OK
?EEPREAD($1234)
123
OK
```

1.44 EEPREAD　仮想 EEPROM のからのデータ読み込み（数値関数）

■ 書式

EEPREAD(アドレス)

■ 引数

アドレス　　：仮想 EEPROM 内アドレス $0000 ～ $FFFF(10 進数でも可能)

■ 戻り値

読みだした 2 バイトデータ（-32767～32767, $0000～$FFFF）

■ 説明

仮想 EEPROM の指定したアドレスから 2 バイトデータを取得します。
があります。

EEPWRITE、EEPREAD コマンドで指定する読書きを指定するアドレスは実際にはアドレスではなく、検索キーです。$0000～$FFFF の間に任意で利用出来ます。
指定したアドレスを検索キーとして該当するデータを返します。該当するデータない場合は 0 を返します。

■ エラーメッセージ

Syntax error	：文法エラー、書式と異なる利用を行った
EEPROM bad FLASH	：フラッシュメモリの異常が発生した

■ 利用例

```
EEPWRITE $1234, 123
OK
?EEPREAD($1234)
123
OK
```

9 応用プログラム

（以降作成中）