

---

---

# 豊四季タイニーBASIC for Arduino STM32 V0.85

## リファレンスマニュアル RV03

---

---

作成日 2017 年 11 月 10 日

作成者 たま吉さん



---

## 目次

---

豊四季タイニーBASIC for Arduino STM32 V0.85 リファレンスマニュアル RV03i

目次 .....	i
1. 仕様や構成など .....	1
1.1 はじめに .....	1
1.2 関連情報 .....	2
1.3 著作権・利用条件について .....	3
1.4 システム構成 .....	2
1.5 利用環境のバリエーション .....	3
1.6 仕様 .....	7
1.7 結線図 .....	8
1.8 パーツ一覧 .....	12
1.9 ボードピンを利用 .....	13
1.10 ボード上のピン一覧 .....	14
1.11 メモリーマップ .....	15
1.12 文字(フォント)コード .....	16
1.13 キー入力コード .....	17
1.14 編集操作操作キー .....	20
2. 実行環境の構築(執筆中) .....	21
2.1 最小限構成の実行環境構築 .....	21
2.2 圧電サウンダー(圧電スピーカー)の追加 .....	21
2.3 SD カードモジュールの追加 .....	21
2.4 NTSC ビデオ出力の追加 .....	21
2.5 PS/2 キーボード インタフェースの追加 .....	22
3. 利用環境設定 .....	23
3.1 ファームウェア(スケッチ)更新後の初期設定 .....	23
3.2 PS/2 キーボードに関する設定 .....	23
3.3 コンソール画面の表示設定 .....	24
3.4 シリアルコンソールの通信条件の設定 .....	25
3.5 NTSC ビデオ出力の同期信号の調整 .....	26
3.6 起動時のコンソール画面の選択 .....	26
3.7 内部 RTC の時刻設定 .....	27
3.8 プログラムの保存と読み込み .....	27
3.9 パソコンからのプログラム読み込と保存 .....	28
3.10 フラッシュメモリのエクスポート .....	28
3.11 プログラムの自動起動設定 .....	29

3.12	仮想 EEPROM 領域の初期化.....	29
3.13	プログラム保存領域の初期化.....	29
3.14	MMC 互換カード (SD カード含む) の初期化 .....	30
4.	使ってみよう！ .....	31
4.1	入力可能状態は OK■ (カーソル) .....	31
4.2	Hello,world を表示してみる.....	31
4.3	Hello,world を Hello,Tiny BASIC に変更してみる .....	31
4.4	PRINT の前に 10 を追加してプログラムにしよう .....	32
4.5	プログラムを実行してみよう.....	32
4.6	プログラムに追加する.....	33
4.7	行番号を並びなおす .....	33
4.8	やっぱり 10 行は不要、削除したい.....	34
4.9	プログラムを保存したい.....	34
4.10	保存されたプログラムを読み込み.....	34
5.	プログラム構成要素の説明 (コマンド・関数等) .....	35
5.1	プログラムの構造と形式.....	35
5.2	行と命令文 .....	36
5.3	制御文 .....	37
5.4	コマンド・関数.....	37
5.5	コマンド・関数一覧 .....	38
5.6	数値 .....	41
5.7	文字・文字列 .....	41
5.8	変数・配列変数.....	42
5.9	演算子 .....	42
5.10	式 .....	44
5.11	定数.....	44
5.12	文字列利用における制約 .....	45
6.	各制御文の詳細.....	47
6.1	GOTO 指定行へのジャンプ (制御命令) .....	47
6.2	GOSUB サブルーチンの呼び出し (制御命令) .....	48
6.3	RETURN GOSUB 呼び出し元への復帰 (制御命令) .....	49
6.4	IF 条件判定 (制御命令) .....	50
6.5	FOR TO ~ NEXT 繰り返し実行 (制御命令) .....	51
6.6	END プログラムの終了 (制御命令) .....	52
7.	各コマンド・関数の詳細.....	53
7.1	RUN プログラムの実行 (システムコマンド) .....	53
7.2	RENUM 行番号の振り直し (システムコマンド) .....	54
7.3	DELETE プログラムの指定行の削除 (システムコマンド) .....	55

7.4	CONSOLE	コンソール画面切替	56
7.5	SCREEN	スクリーンモードの設定	57
7.6	WIDTH	シリアルターミナルの画面サイズの設定	58
7.7	LIST	プログラムリストの表示	59
7.8	NEW	プログラムの消去	60
7.9	LOAD	内部フラッシュメモリ・SD カードからプログラムを読み	61
7.10	SAVE	内部フラッシュメモリ・SD カードへのプログラム保存	62
7.11	FILES	内部フラッシュメモリ保存・SD カード内のプログラムの一覧表示	63
7.12	BLOAD	SD カードからバイナリデータ読込	65
7.13	BSAVE	SD カードへのバイナリデータ保存	66
7.14	REM	コメント	67
7.15	LET	変数に値を代入	68
7.16	CLV	変数領域の初期化	69
7.17	LRUN	指定プログラム番号の実行	70
7.18	EXPORT	内部フラッシュメモリの内容のエクスポート	71
7.19	CONFIG	システムの設定	73
7.20	SAVECONFIG	システムの設定の保存	74
7.21	ERASE	内部フラッシュメモリ上のプログラム削除	75
7.22	MKDIR	ディレクトリの作成	76
7.23	RMDIR	ディレクトリの削除	77
7.24	REMOVE	ファイルの削除	78
7.25	CAT	ファイルの内容表示	79
7.26	ABS	絶対値の取得（数値関数）	80
7.27	ASC	文字から文字コードへの変換（数値関数）	81
7.28	FREE	プログラム領域の残りバイト数の取得（数値関数）	83
7.29	INKEY	キー入力の読み取り（数値関数）	84
7.30	RND	乱数の発生（数値関数）	85
7.31	LEN	文字列の長さの取得（数値関数）	86
7.32	MAP	数値のスケール変換（数値関数）	87
7.33	RGB	3 原色から RGB コード変換（数値関数）	88
7.34	CHR\$	文字コードから文字への変換（文字列関数）	90
7.35	BIN\$	数値から 2 進数文字列への変換（文字列関数）	91
7.36	HEX\$	数値から 16 進数文字列への変換（文字列関数）	92
7.37	DMP\$	整数の小数付き数値文字列への変換（文字列関数）	93
7.38	STR\$	変数が参照する文字列の取得・文字列の切り出し	94
7.39	WAIT	時間待ち	95
7.40	CLT	起動からの経過時間カウンタのリセット	96
7.41	TICK	経過時間取得（数値関数）	97

7.42	POKE 指定アドレスへのデータ書き込み.....	98
7.43	PEEK 指定アドレスの値参照（数値関数） .....	99
7.44	PRINT 画面への文字表示.....	100
7.45	INPUT 数値の入力.....	102
7.46	CLS 画面表示内容の全消去 .....	103
7.47	COLOR 文字色の設定 .....	104
7.48	ATTR 文字表示属性の設定.....	105
7.49	LOCATE カーソルの移動 .....	106
7.50	REDRAW 画面表示の再表示.....	107
7.51	CSCROLL キャラクタ画面スクロール.....	108
7.52	VPEEK 画面指定位置の文字コード参照（数値関数） .....	109
7.53	PSET 点の描画 .....	110
7.54	LINE 直線の描画.....	111
7.55	RECT 矩形の描画 .....	112
7.56	CIRCLE 円の描画.....	113
7.57	BITMAP ビットマップ画像の描画 .....	114
7.58	GPRINT 文字列の描画 .....	115
7.59	GSCROLL グラフィックススクロール .....	116
7.60	LDBMP ビットマップファイルの読み込み .....	118
7.61	DWBMP BMP ファイルの表示.....	120
7.62	GPEEK 画面上の指定位置ピクセルの参照（数値関数） .....	122
7.63	GINP 指定領域のピクセルの有無判定（数値関数） .....	123
7.64	TONE 単音出力 .....	124
7.65	NOTONE 単音出力停止 .....	125
7.66	DATE 現在時刻の表示 .....	126
7.67	GETDATE 日付の取得 .....	127
7.68	GETTIME 時刻の取得 .....	128
7.69	SETDATE 時刻の設定.....	129
7.70	GPIO GPIO 機能設定 .....	130
7.71	OUT デジタル出力 .....	132
7.72	POUT PWM パルス出力 .....	133
7.73	SHIFTOUT デジタルシフトアウト出力.....	134
7.74	IN デジタル入力（数値関数） .....	136
7.75	ANA アナログ入力（数値関数） .....	137
7.76	SHIFTIN デジタルシフトアウト入力（数値関数） .....	138
7.77	PULSEIN 入力パルス幅の計測力（数値関数） .....	139
7.78	I2CR I2C スレーブデバイスからのデータ受信（数値関数） .....	141
7.79	I2CW I2C スレーブデバイスへのデータ送信（数値関数） .....	143

7.80	SMODE シリアルポート機能設定 .....	145
7.81	SOPEN シリアル通信オープン .....	147
7.82	SCLOSE シリアル通信クローズ .....	148
7.83	SPRINT シリアル通信文字列出力 .....	149
7.84	SWRITE シリアル通信 1 バイト出力 .....	151
7.85	SREADY シリアル通信データ受信データ確認（数値関数） .....	152
7.86	SREAD シリアル通信 1 バイト受信（数値関数） .....	153
7.87	EEPFORMAT 仮想 EEPROM のフォーマット .....	154
7.88	EEPWRITE 仮想 EEPROM のへのデータ書き込み .....	155
7.89	EEPREAD 仮想 EEPROM のからのデータ読み込み（数値関数） .....	156
8.	応用プログラム(執筆中) .....	157
8.1	ボード上の LED の点滅 .....	157
8.2	追加 LED の点滅 .....	158
8.3	PWM による LED 輝度の制御 .....	158
8.4	スライド抵抗をつかった LED の輝度調節 .....	158
8.5	シフトレジスタ 74HC595 を使った 8 個の LED の制御 .....	158
8.6	I2C インタフェースの利用（基本編） .....	158
8.7	シリアル通信 .....	158
8.8	OLED(SSD1306)ディスプレイの利用 .....	158

## 1. 仕様や構成など

### 1.1 はじめに

「豊四季タイニーBASIC for Arduino STM32」は整数型 BASIC インタープリタ開発・実行環境です。Tetsuya Suzuki 氏が開発及び公開している「TOYOSHIMI Tiny BASIC for Arduino」をベースに、安価な Blue Pill ボード (STM32F103C8T6 搭載) 向けに移植し機能拡張を行いました。開発環境としては、Arduino STM32 を採用しています。

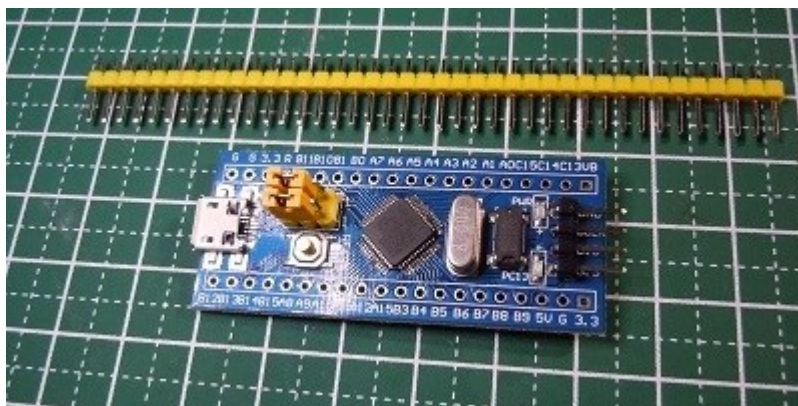


図 1 Blue Pill ボード



図 2 ブレッドボード上の実装例

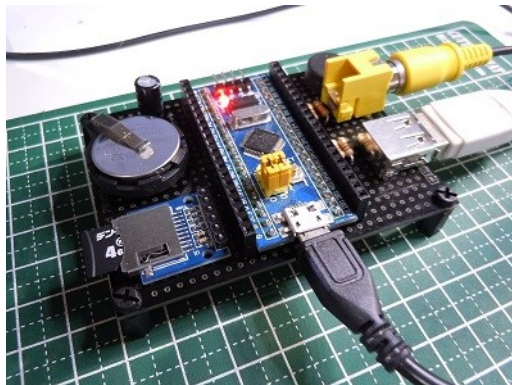


図 3 ユニバーサル基板上的実装例

機能拡張としては、プログラム編集・開発機能の強化（スクリーンエディット機能、プログラム保存機能、シリアルコンソール）、周辺機器のサポート（V モニターディスプレイ、PS/2 キーボード、MMC 互換カード対応、単音出力）、電子工作利用のための制御機能（デジタル入出力、PWM 出力、アナログ入力、I2C バス通信、シリアル通信）を追加しました。



図 4 起動直後の画面

本リファレンスマニュアルでは、オリジナル版の TinyBASIC の文法に加え、新しく追加した機能、コマンドについて解説・説明いたします。本書の構成としては、前半に概要、システム構成、仕様、レガシーBASIC 環境を始めて利用する方向けの解説、中盤にリファレンス、後半に応用の内容となります。



## 1.2 関連情報

「豊四季タイニーBASIC（オリジナル版）」に関する情報

### ❑ 開発者公開ホームページ

電脳伝説 Vintagechips 豊四季タイニーBASIC 確定版

<https://vintagechips.wordpress.com/2015/12/06/豊四季タイニーbasic-確定版/>

### ❑ 開発者公開リソース

[https://github.com/vintagechips/ttbasic\\_arduino/](https://github.com/vintagechips/ttbasic_arduino/)

### ❑ 開発者執筆書籍

タイニーBASIC を C で書く

<http://www.socym.co.jp/book/1020>



原本のご紹介

タイニー BASIC を C で書く  
パソコンでもマイコンでも走り、機能拡張し放題。

著者 鈴木哲哉

定価 3,200 円 + 税

判型 / ページ数 B5 変形 / 288 ページ

ISBN 978-4-8026-1020-9

「豊四季タイニーBASIC for Arduino STM32」に関する情報

### ❑ 公開サイト

豊四季タイニーBASIC for Arduino STM32

[https://github.com/Tamakichi/ttbasic\\_arduino/tree/ttbasic\\_arduino\\_ps2\\_ntsc](https://github.com/Tamakichi/ttbasic_arduino/tree/ttbasic_arduino_ps2_ntsc)

### ❑ Arduino STM32 に関する情報

- Arduino STM32 公式 HP (モジュール配布サイト)

rogerclarkmelbourne/Arduino\_STM32

[https://github.com/rogerclarkmelbourne/Arduino\\_STM32](https://github.com/rogerclarkmelbourne/Arduino_STM32)

- Arduino STM32 公式フォーラム

<http://www.stm32duino.com/>

- STM32duino wiki

<http://wiki.stm32duino.com>

- Blue Bill ボードに関する情報

[http://wiki.stm32duino.com/index.php?title=STM32F103\\_boards](http://wiki.stm32duino.com/index.php?title=STM32F103_boards)

- DEKO のあやしいお部屋。 - STM32F103C8T6

<http://ht-deko.com/arduino/stm32f103c8t6.html>

### 1.3 著作権・利用条件について

「豊四季タイニーBASIC for Arduino STM32」の著作権は移植・機能拡張者の「たま吉」、オリジナル版開発者の「Tetsuya Suzuki 氏」にあります。

本著作物の利用条件は、オリジナル版「豊四季タイニーBASIC」に従うものとします。

オリジナル版は GPL (General Public License) でライセンスされた著作物であり、「豊四季タイニーBASIC for Arduino STM32」もそれに従い、GPL ライセンスされた著作物となります。

更に利用条件として以下を追加します。

- 1) 不特定多数への再配布においてはバイナリーファイルのみの配布を禁止、必ずソースを添付すること。
- 2) 「豊四季タイニーBASIC for Arduino STM32」の明記、配布元の URL を明記する。
- 3) 営利目的のNAVER 等のまとめサイト(まとめ者に利益分配のあるもの)へのいかなる引用の禁止

#### 補足事項

オリジナル版において開発者 Tetsuya Suzuki 氏は次の利用条件を提示しています。

- 1) TOYOSHIMI Tiny BASIC for Arduino

[https://github.com/vintagechips/ttbasic\\_arduino](https://github.com/vintagechips/ttbasic_arduino)

の記載内容/

*(C)2012 Tetsuya Suzuki*

*GNU General Public License*

- 2) 豊四季タイニーBASIC のリファレンスを公開

<https://vintagechips.wordpress.com/2012/06/14/豊四季タイニーbasic リファレンス公開/>

の記載内容/

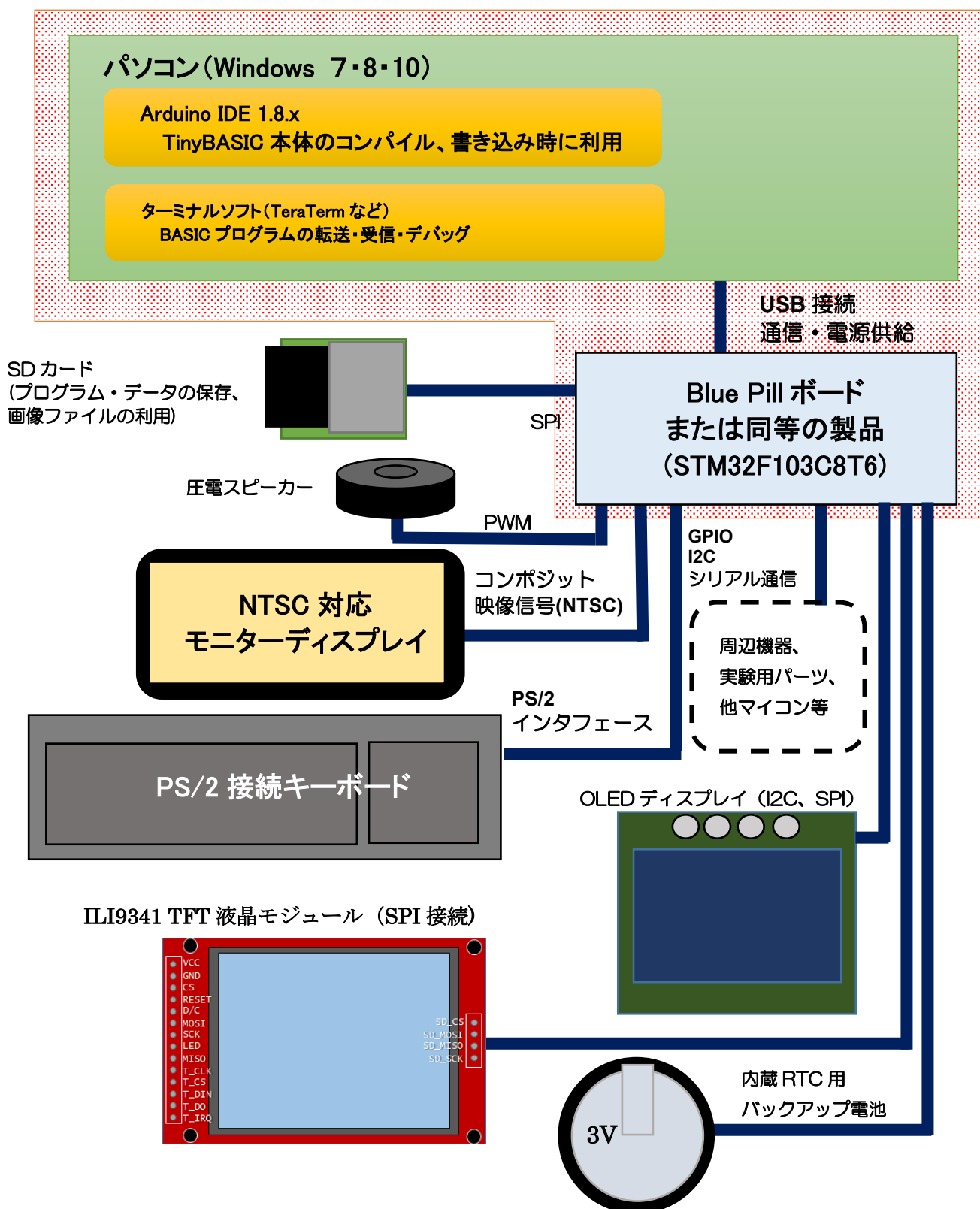
*豊四季タイニーBASIC がもたらすいかなる結果にも責任を負いません。*

*著作権者の同意なしに経済的な利益を得てはいけません。*

*この条件のもとで、利用、複写、改編、再配布を認めます。*

## 1.4 システム構成

「豊四季 Tiny BASIC for Arduino STM32」では、次のような構成での利用で利用可能です。

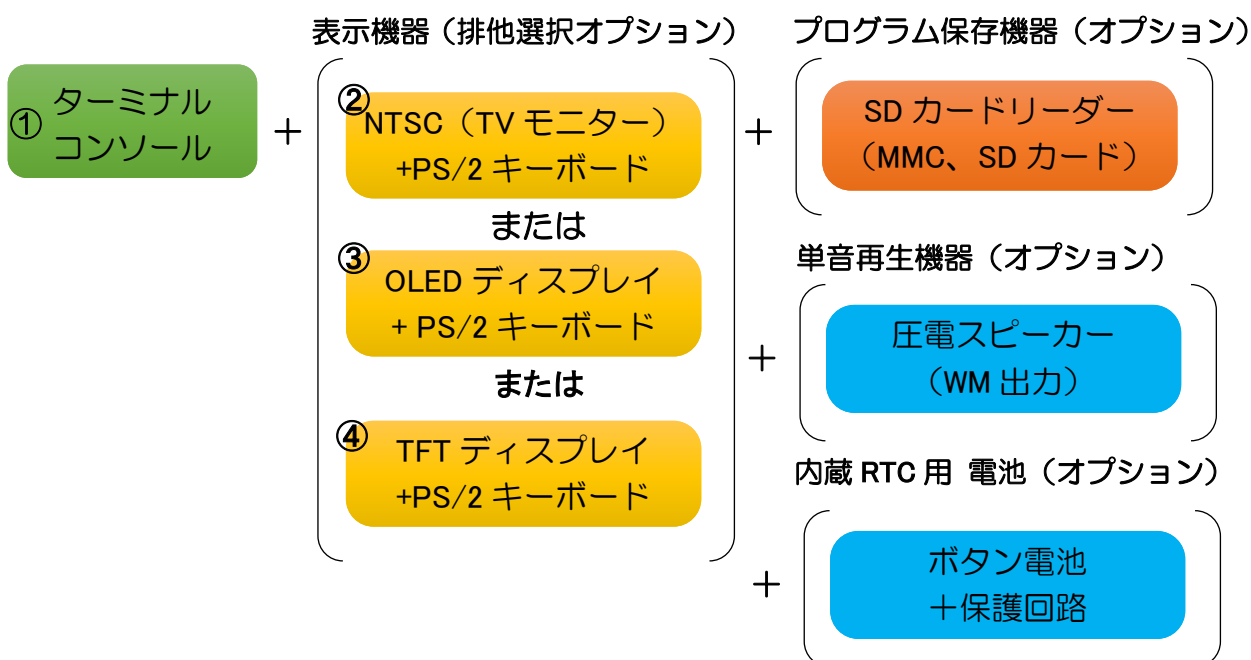


**図み内** は利用においての最小限の構成です。シリアル経由でのプログラム入力が可能です。

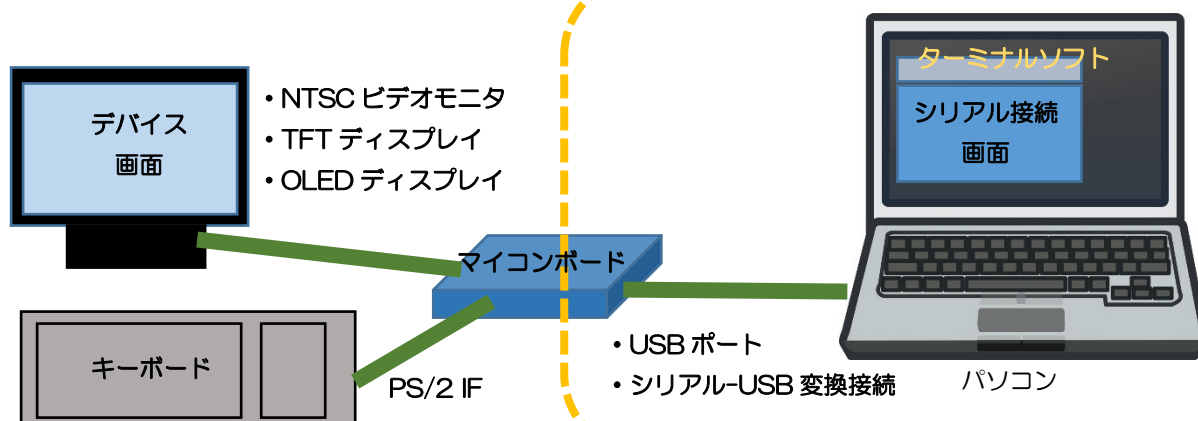
周辺機器は必須ではありません。また、同時利用は出来ないものがあります。

## 1.5 利用環境のバリエーション

豊四季 Tiny BASIC for Arduino STM32 では、次のような組み合わせで利用環境を構築することが可能です。



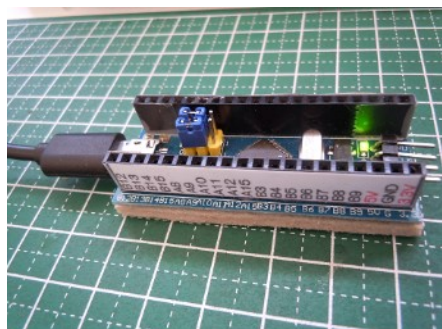
## 具体的な利用イメージ



BASIC プログラムを対話編集・実行する環境でバリエーションを分類すると、次の4種類(①～④)となります。

## ① ターミナルコンソール版

USB ポートまたは、シリアルポートによるシリアルコンソール画面を使った最小限のプログラミング環境です。



標準装備の USB ポートにてパソコンに接続し、パソコン画面上で TeraTerm 等のターミナルソフトを使ったプログラム開発を行うことができます。シリアルコンソール画面は、USB シリアル接続とは別に、ボード上のシリアルポートに随時切り替えて使うことも可能です。

USB シリアル  
コンソール画面  
(デフォルト)

GPIO シリアルポート  
(コンソール画面化可  
能)

必要に応じて、SD カードリーダー、圧源スピーカー、内蔵 RTC 用バックアップ電池を追加して利用出来ます。

## ② NTSC 版

NTSC 対応モニターと PS/2 キーボードを使う環境です。

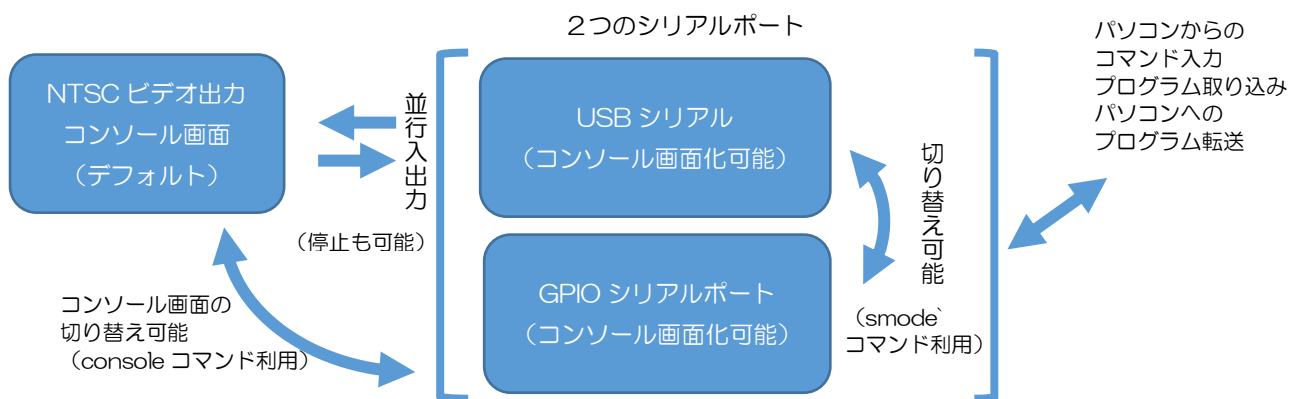


NTSC ビデオ出力画面をプログラム作成画面（コンソール画面）として利用します。

画面に出力されるテキスト文字列は、シリアルポート（デフォルトでは USB シリアルポート）にも出力されます。シリアルポートからの入力は、PS/2 キーボードと並行してキーボード入力として受付ます。

NTSC ビデオ出力画面表示が行えない状況でも、シリアルポートからのコマンド入力を行うことが可能です。

シリアルポートを利用してパソコンからの BASIC プログラムの取り込みや、パソコンへの転送も用意を行うことができます（ただし、スクリーン編集を行うには、コンソール画面の切替が必要です）。



これにより、パソコンからの上からの文字入力を並行して受け、かつマイコンボードからのプログラム実行結果を随時、パソコン側で取り込むことができます。

また、USB シリアルポートの代わりに、GPIO シリアルピンのシリアルを利用することも可能です。

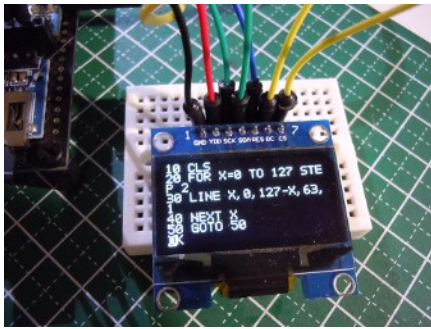
さらに、コンソール画面を NTSC ビデオ出力ではなく、「①シリアルコンソール版」と同様に、シリアルポートを使ったシリアルコンソール画面に随時切り替えることができます。

必要に応じて、SD カードリーダー、圧源スピーカー、内蔵 RTC 用バックアップ電池を追加して利用出来ます。  
ボード上の入出力端子を使った、各種制御実験等を行うことができます。

## ③ OLED 版

OLED モジュールを表示機として利用する環境です。PS/2 キーボードを併用します。

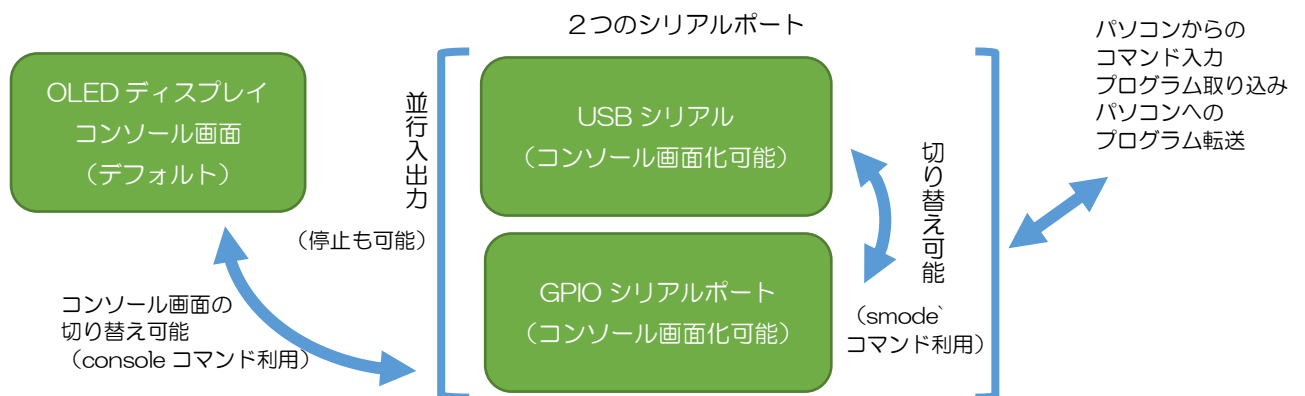
必要に応じて、SD カードリーダー、圧源スピーカー、内蔵 RTC 用バックアップ電池を追加して利用出来ます。



OLED モジュールの接続は I2C と SPI をサポートします。

I2C 接続は、接続が簡単でボード上のポートを占有することなく利用出来ますが、SPI 接続に比べると表示レスポンスが遅いです。高速な描画を行いたい場合は SPI 接続にて利用して下さい。

NTSC 版と同様に、画面に出力されるテキスト文字列は、シリアルポート（デフォルトでは USB シリアルポート）にも出力されます。シリアルポートからの入力は、PS/2 キーボードと並行してキーボード入力として受付ます。シリアルポートを利用してパソコンからの BASIC プログラムの取り込みや、パソコンへの転送も用意に行うことが出来ます（ただし、スクリーン編集を行うには、コンソール画面の切替が必要です）。



コンソール画面をシリアルポートに切り替えた場合、OLED ディスプレイ上での入力テキストは表示出来ませんが、グラフィック描画を行うコマンドは利用することが出来ます。

## ④ TFT 版

TFT カラーグラフィック液晶と PS/2 キーボードを使う環境です。

カラー表示をサポートします。

必要に応じて、SD カードリーダー、圧源スピーカー、内蔵 RTC 用バックアップ電池を追加して利用出来ます。



NTSC 版、OLED 版と同様に、画面に出力されるテキスト文字列は、シリアルポート（デフォルトでは USB シリアルポート）にも出力されます。シリアルポートからの入力は、PS/2 キーボードと並行してキーボード入力として受付ます。シリアルポートを利用してパソコンからの BASIC プログラムの取り込みや、パソコンへの転送も用意を行うことができます（ただし、スクリーン編集を行うには、コンソール画面の切替が必要です）。



コンソール画面をシリアルポートに切り替えた場合、OLED ディスプレイ上での入力テキストは表示出来ませんが、グラフィック描画を行うコマンドは利用することができます。

## 1.6 仕様

## (1) ソフトウェア仕様

項目		概要
プログラミング環境		CUI（キャラクユーザーインタフェース） スタンドアロンのスクリーンエディタ+実行環境
NTSC キャラクタ表示画面		37×27 文字、28×13 文字、28×13 文字、24×13 文字
NTSC グラフィック表示画面		224×216 ドット、224×108 ドット、112×108 ドット
ターミナルスクリーン画面		16×10 文字 ～ 127 文字×45 文字の範囲で任意指定
OLED キャラクタ表示画面		21x8 文字、10x4 文字、7x2 文字（画面回転可能）
OLED グラフィック表示画面		128x64 ドット（画面回転可能）
TFT キャラクタ表示画面		53x30、26x15、17x10、13x7、10x6、8x5 文字（画面回転可能）
TFT グラフィック表示画面		320x240 ドット（画面回転可能）
B A S I C 言 語 仕 様	形式	整数型 BASIC（符号付 16 ビット整数 -32768～32767）
	プログラム領域	4096 バイト
	利用文字コード	1 バイト JIS コード（半角記号・英数字、半角カタカナ）＋ キャラクタ文字（IchigoJam 互換）
	利用可能変数	変数名：208 個 A～Z の英字 1 文字 26 個、 A0..A6 ～ Z0..Z6 の英字 1 文字+数字 1 文字 182 個 配列変数@:100 個 @(0)～@(99)
	ユーザー開放 作業用メモリ	SRAM 1024 バイト 内部フラッシュメモリ 250 ワード（16 ビット×250）
	プログラム保存 方式	内部フラッシュメモリ 8 本（32,768 バイト） 外部 MMC 互換カード（SD カードも含む） ⇒ 保存可能数は SD カードの容量による

## (2) ハードウェア仕様

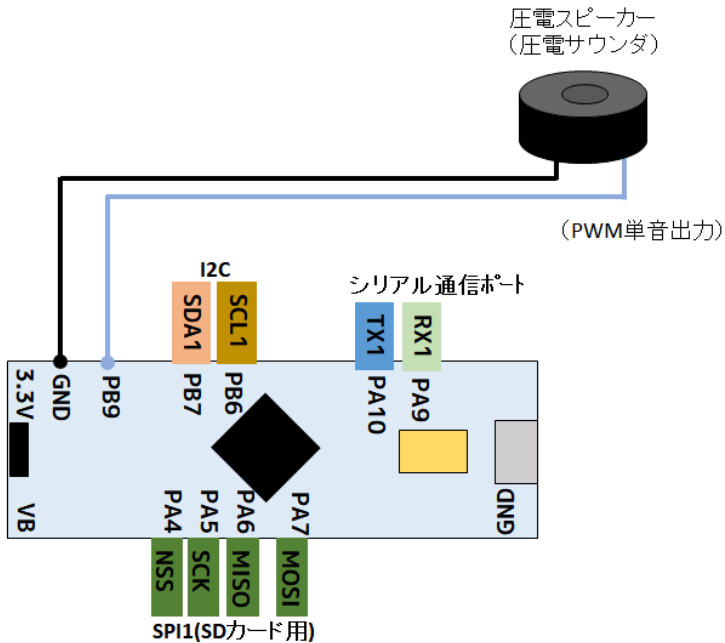
項目	概要
動作クロック	72MHz
時計機能	内部 RTC 利用 (電池によるバックアップ可能) 精度: ±10 分/月
映像出力	モノラル (2色白黒) NTSC ビデオ出力 接続は RCA 端子 (コンポジット映像信号)
キーボード I/F	PS/2 5V 対応 (5V トレラント ピン利用) 日本語キーボード、US キーボード対応
サウンド	圧電スピーカーによる単音モノラル再生 (3.3V 矩形波 1~32767Hz)
USB I/F	DFU 利用ファームウェア書込み USB-シリアル通信
シリアル通信	1 チャンネル: 通信速度 ~921,600、データ長 8 ビット、 ストップビット 1 パリティ None
I2C インタフェース	1 チャンネル (マスター利用のみ) アドレス 7 ビット、400kbps
GPIO	<ul style="list-style-type: none"> <li>デジタル入力 1 ビット× 24 ポート</li> <li>デジタル出力 1 ビット× 24 ポート</li> <li>PWM 出力 7 チャンネル (周波数可変、デューティ比 1/4096 刻み)</li> </ul>
アナログ入力	9 チャンネル (分解能 12 ビット)
MMC 互換カード (SD カード含む)	FAT16 / FAT32 32G バイトまで対応 SPI 接続
OLED ディスプレイ 対応モジュール	SSD1306、SSD1309、SH1106 コントローラ利用モジュール 画面サイズ 0.96 インチ~22.4 インチ 128x64 ドットの解像度の製品 対応インタフェース: I2C、SPI
TFT ディスプレイ 対応モジュール	ILI9341 コントローラ利用グラフィック液晶モジュール 画面解像度 320x240 の製品



## 1.7 結線図

Blue Pill の場合の各パーツの取り付け・結線は次の図のようになります。

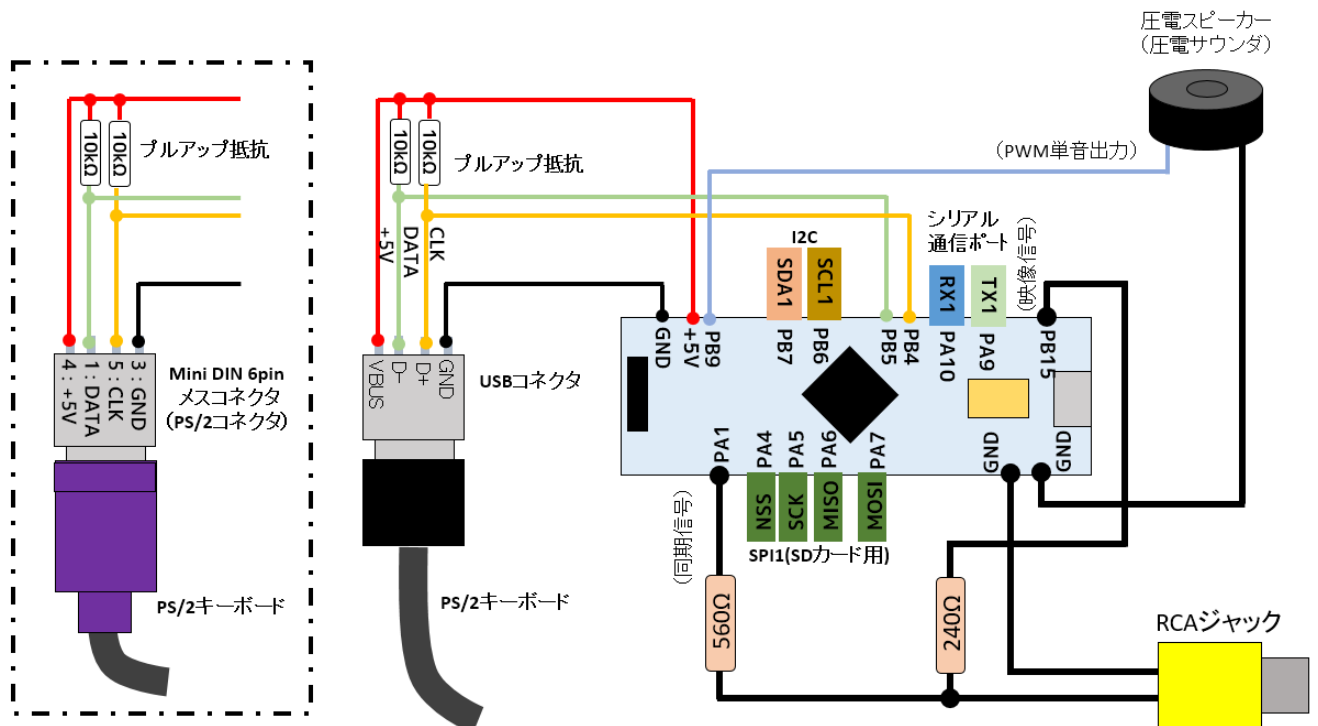
## ① ターミナルコンソール版



圧電スピーカーはオプションです。必要に応じて接続して下さい。

RTC 用バックアップ電池、SD カードモジュールも必要に応じて追加できます。

## ② NTSC 版



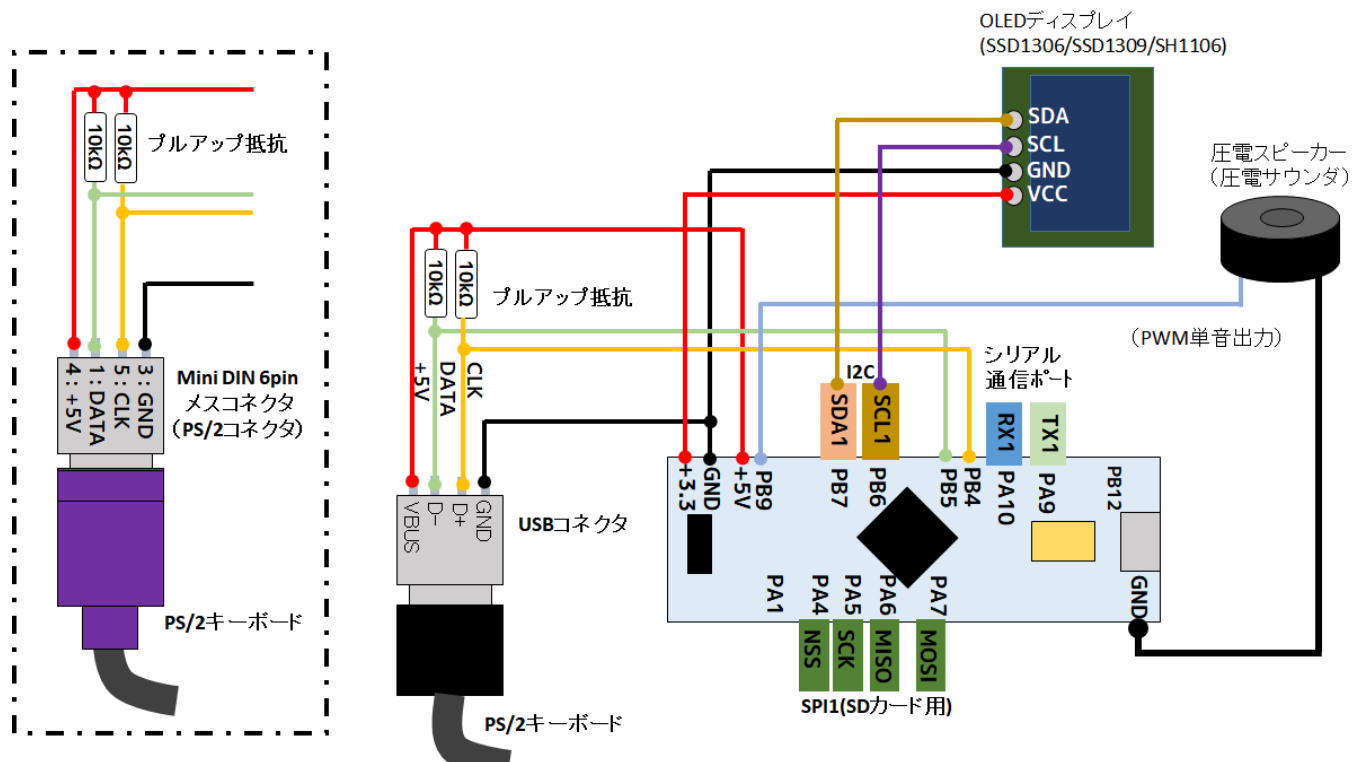
PS/2 キーボード用の接続コネクタは、USB コネクタ、Mini DIN6 ピンのどちらかを用途に応じて選択します。

圧電スピーカーはオプションです。必要に応じて接続して下さい。

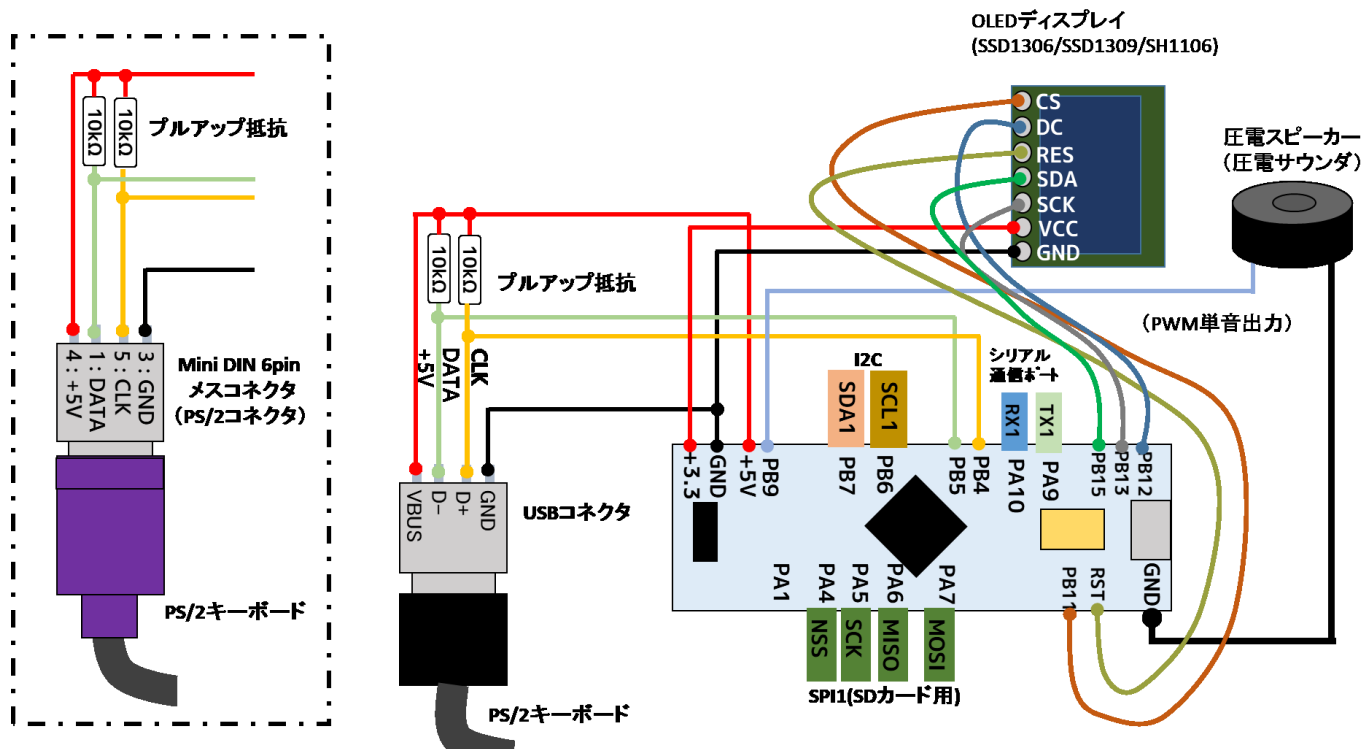
RTC 用バックアップ電池、SD カードモジュールも必要に応じて追加できます。

### ③ OLED 版

#### (1) 利用する OLED モジュールが I2C インタフェースの場合



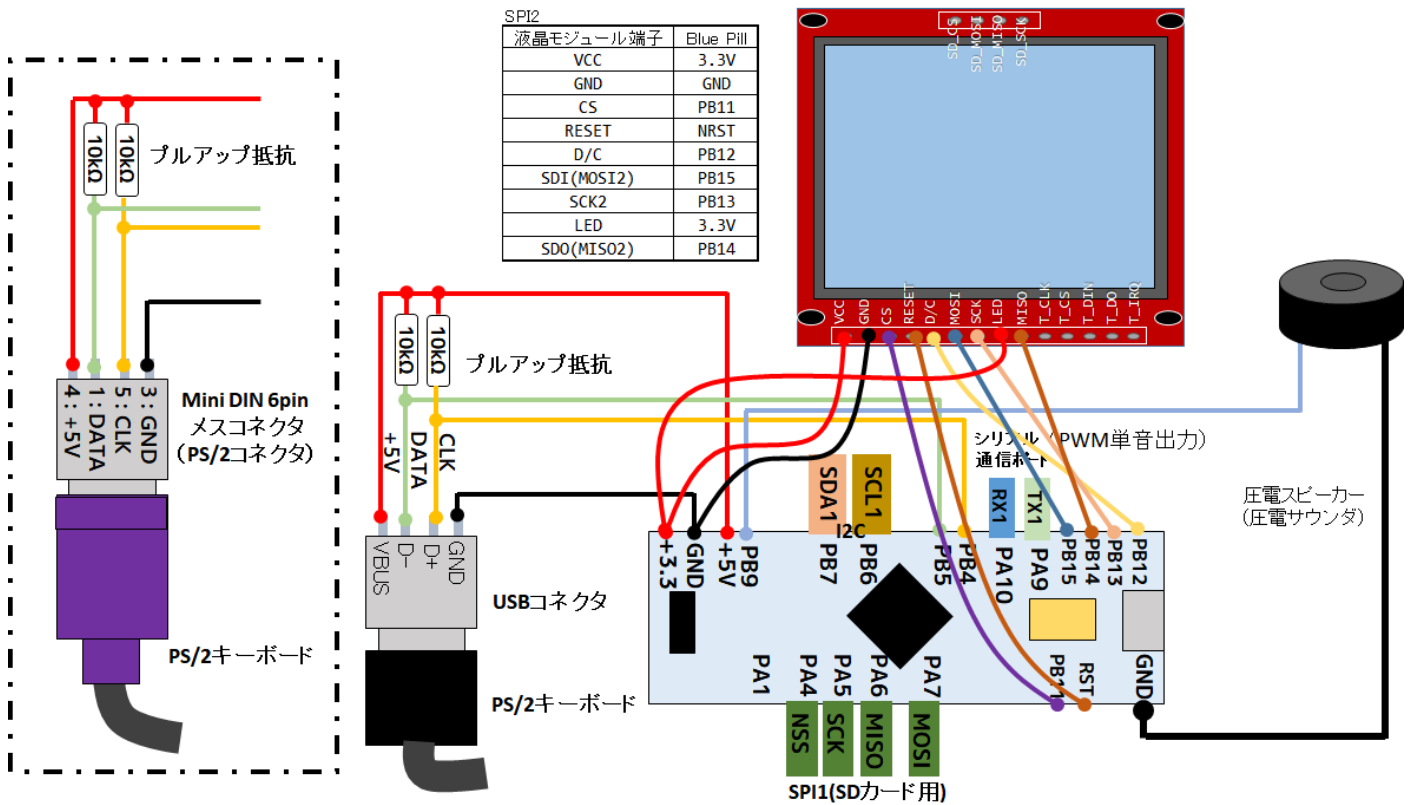
#### (2) 利用する OLED モジュールが SPI インタフェースの場合



両タイプとも、圧電スピーカーはオプションです。必要に応じて接続して下さい。

また RTC 用バックアップ電池、SD カードモジュールも必要に応じて追加できます。

④ TFT 版



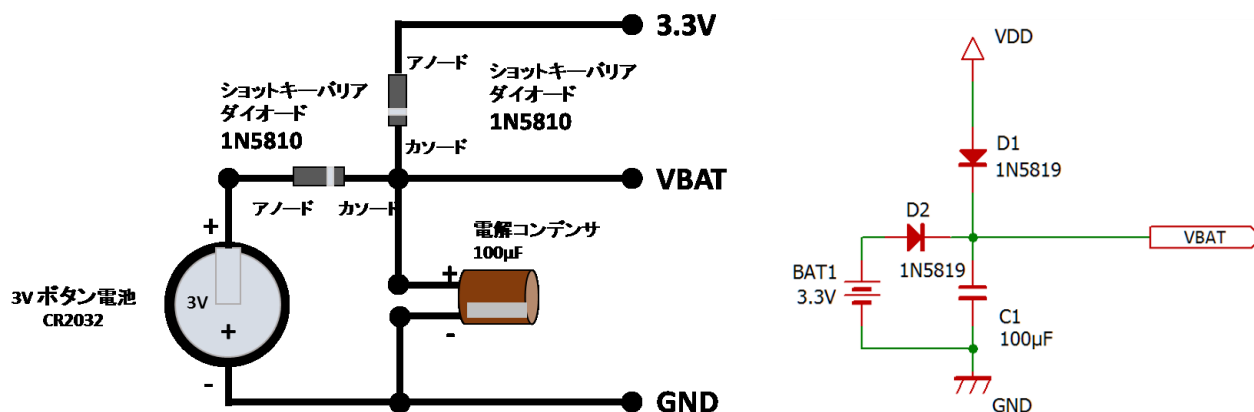
圧電スピーカーはオプションです。必要に応じて接続して下さい。

RTC 用バックアップ電池、SD カードモジュールも必要に応じて追加できます。

利用する TFT モジュールに SD カードリーダーが搭載している場合は、その SD カードリーダーの利用が可能です。

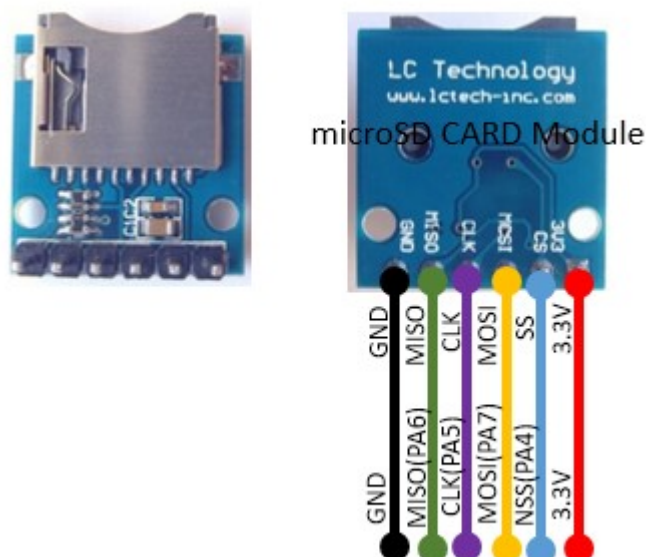
⑤ RTC 用バックアップ電池（オプション）

RTC 用バックアップ電池を利用することで、電源を切った状態でも RTC の時刻を保持することが出来ます。以下の接続例を示します。利用するダイオードは、VF 値が低い(0.5 以下) ものを推奨します。



⑥ SD カードモジュール（オプション）

市販のプルアップ抵抗及び、ダンピング抵抗が組み込まれたモジュールの利用を推奨します。



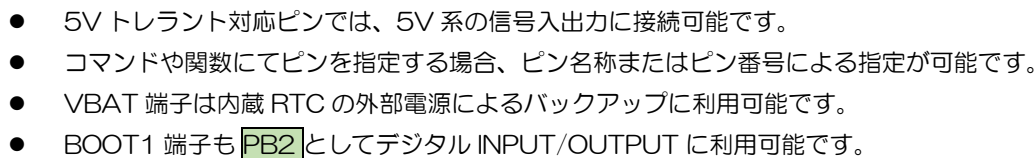
## 1.8 パーツ一覧

ブレッドボード上に実装する場合を想定したパーツ一覧(※パーツの入手先、製品、価格は目安です)

No.	パーツ名称	個数	説明	価格 (円)
1	Blue Pill マイコンボード	1	マイコン本体 amazon (HiLetgo) STM32F103C8T6 ARM STM32 Minimum システム 開発ボードモジュール	290
2	RCA ジャック (メス)	1	NTSC モニター接続用端子 秋月電子 RCA ジャック DIP 化キット(黄) AE-RCA-DIP-Y	100
3	USB コネクタ A メス ※1	1	PS/2 キーボード接続用端子 秋月電子 USB コネクタ DIP 化キット(A メス) AE-USB-A-DIP	120
4	Mini DIN 6pin コネクタ ※1	1	PS/2 キーボード接続用端子 秋月電子 ミニ DIN コネクタピッチ変換キット AE-MiniDIN-6	150
5	圧電スピーカー	1	サウンド用 秋月電子 圧電スピーカー(17mm) PKM17EPP-4001-B0	40
6	抵抗 560Ω 1/4W	1	NTSC ビデオ信号生成用	10
7	抵抗 240Ω 1/4W	1	NTSC ビデオ信号生成用	10
8	抵抗 10kΩ 1/4W	2	PS/2 キーボード用	20
9	SD カードモジュール ※2	1	SD カード利用 amazon EasyWordMail SD カードスロットソケットリーダーモジュール Arduino 用 HiLetgo Micro SD/ TF カードモジュール シールド モジュール 6 ピン SPI	150
10	ボタン電池 CR2032 ※3	1	RTC バックアップ用 ダイソー ボタン電池 CR2032	100
11	ボタン電池用ボックス ※3	1	RTC バックアップ用 秋月電子 ボタン電池基板取付用ホルダー CR2032 用 (小型タイプ)	50
12	ショットキーバリアダイオード 1N5810, 1S3 など ※3	2	RTC バックアップ用 秋月電子 整流用ショットキーダイオード(30V1A) 1S3 VF が 0.5V 以下が目安	40
13	電解コンデンサ 100μF ※3	1	RTC バックアップ用 秋月電子 電解コンデンサー100μF25V85℃(ルビコン OK)	10
14	OLED ディスプレイ	1	SSD1306/SSD1309/SH1106 コントローラ 解像度 128x64 ドット、I2C、SPI に対応 1.3 インチまたは 1.54 インチの製品推奨	500
15	TFT グラフィック液晶	1	ILI9341 コントローラ利用グラフィック液晶モジュール SPI 接続、画面解像度 320x240 の製品	800
16	ブレッドボード ユニバーサル基板、線材	適宜		500

※2, ※3 RTC 用電池、SD カードモジュールは必要に応じて実装

画像は各販売元または秋月電子の通販サイトのものを利用させていただいています。



## 1.10 ポート上のピン一覧

ピン 名称	ピン 番号	用途	説明
PA0	0	アナログ入力、デジタル IN/OUT	汎用
PA1	1	アナログ入力、デジタル IN/OUT 、NTSC 同期信号	汎用、NTSC 版では占有(利用禁止)
PA2	2	アナログ入力、デジタル IN/OUT	汎用
PA3	3	アナログ入力、デジタル IN/OUT	汎用
PA4	4	アナログ入力、デジタル IN/OUT、SPI(NSS)	汎用、SD カード
PA5	5	アナログ入力、デジタル IN/OUT、SPI(SCK)	汎用、SD カード
PA6	6	アナログ入力、デジタル IN/OUT、PWM、SPI(MISO)	汎用、SD カード
PA7	7	アナログ入力、デジタル IN/OUT、PWM、SPI(MOSI)	汎用、SD カード
PA8	8	デジタル IN/OUT、PWM	汎用
PA9	9	デジタル IN/OUT、PWM、シリアル通信	汎用、シリアル通信（送信）
PA10	10	デジタル IN/OUT、PWM、シリアル通信	汎用、シリアル通信（受信）
PA11	11	USB D-	占有、他の利用禁止
PA12	12	USB D+	占有、他の利用禁止
PA13	13	S-LINK、デジタル IN/OUT	S-LINK、汎用
PA14	14	S-LINK、デジタル IN/OUT	S-LINK、汎用
PA15	15	デジタル IN/OUT	汎用
PB0	16	アナログ入力、デジタル IN/OUT、PWM	汎用
PB1	17	アナログ入力、デジタル IN/OUT、PWM	汎用
PB2	18	BOOT1、デジタル IN/OUT ※プルアップ抵抗あり 起動時にレベルの状態ですターミナルモード選択	汎用、BOOT1 モード指定 ターミナルモード選択
PB3	19	デジタル IN/OUT	汎用
PB4	20	デジタル IN/OUT 、PS/2 キーボード I/F CLK	汎用、PS/2 キーボード利用時占有(利用禁止)
PB5	21	デジタル IN/OUT 、PS/2 キーボード I/F DATA	汎用、PS/2 キーボード利用時占有(利用禁止)
PB6	22	I2C SCL1	占有、他の利用禁止
PB7	23	I2C SDA1	占有、他の利用禁止
PB8	24	デジタル IN/OUT	汎用
PB9	25	SOUND(PWM 出力)	占有、他の利用禁止
PB10	26	デジタル IN/OUT	汎用
PB11	27	デジタル IN/OUT、OLED_CS、TFT_CS	汎用、OLED(SPI)、TFT 利用時占有(利用禁止)
PB12	28	デジタル IN/OUT、OLED_SC、TFT_DC	汎用、OLED(SPI)、TFT 利用時占有(利用禁止)
PB13	29	デジタル IN/OUT、OLED_SCK、TFT_SCK	汎用、OLED(SPI)、TFT 利用時占有(利用禁止)
PB14	30	デジタル IN/OUT、TFT_SDO	汎用、TFT 利用時占有(利用禁止)
PB15	31	デジタル IN/OUT、NTSC 映像信号、OLED_SDA、TFT_SDI	汎用、NTSC、OLED(SPI)、TFT 利用時占有
PC13	32	LED、デジタル IN/OUT ※プルアップ抵抗あり	汎用
PC14	33	RTC へのクロック供給	占有、他の利用禁止
PC15	34	RTC へのクロック供給	占有、他の利用禁止

- **色塗り部**は GPIO コマンドでの利用する機器のよっては利用禁止となる
- 各ピン 20mA（ソース、シンク利用）まで電流を流すことが可能、推奨 8mA
- 全ピン合計では 150mA まで利用可能

## 1.11 メモリーマップ

「豊四季タイニーBASIC for Arduino STM32」では STM32F103CT6 のサポート外のフラッシュメモリ領域 64 バイトを含む合計 128k バイトの領域を利用しています。SRAM は 20k バイトの領域を利用しています、

## フラッシュメモリ

128k バイトの領域はページ単位(1024 バイト)で領域が管理されています。

領域の利用は下記の通りです。

アドレス	ページ番号	ページ数	用途
0x08000000-0x08001FFF	0	8	Arduino STM32 ブートローダー (DFU)
0x08002000-0x080157FF	8	86	Tiny BASIC インタープリタ
0x08015800-0x0801F7FF	94	32	プログラム保存用 (4k バイト×8)
0x0801F800-0x0801FFFF	126	2	仮想 EEPROM (config 用、ユーザー利用用)

## 仮想アドレス

Tiny BASIC では、仮想アドレス利用することで SRAM およびフラッシュメモリ上のデータの参照・書き込みを行うことができます。仮想アドレスは次の構成となります。

仮想アドレス	定数名	領域サイズ(バイト)	用途
\$0000	VRAM	可変・最大 5,760	キャラクタスクリーン表示用メモリ (CW×CH) 最大 128×45 = 6,400 バイト
\$1900	VAR	420	変数領域 (A~Z, A0:A6~Z0:Z6) + 4 バイト
\$1AA0	ARRAY	200	配列変数領域 (@ (0) ~ @ (99) )
\$1BA0	PRG	4,096	プログラム領域
\$2BA0	MEM	1,024	ユーザーワーク領域
\$2FA0	FNT	2,048	フォント 256 文字 (フラッシュメモリ)
\$37A0	GRAM	可変・最大 6,048	グラフィック表示用メモリ NTSC、OLED 版でのみ利用可能。ただし、 ターミナルモードの場合 (CONSOLE 1) 領域サイズは 0 となります。 サイズは GW/8*GH で計算できます。

各領域のアドレス先頭は定数 VRAM, VAR, ARRAY, PRG, MEM, FNT, GRAM に定義されています。



## 1.12 文字(フォント)コード

NTSC ビデオ表示で利用しているデフォルトフォントのキャラクター表を下記に示します。

フォントはサイズは 6x8 ドットです。IchigoJam と互換性があります。

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
\$00	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
\$10	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
\$20	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/	
\$30	0	1	2	3	4	5	6	7	8	9	:	;	=	>	?	
\$40	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	
\$50	p	q	r	s	t	u	v	w	x	y	z	[	\	]	^	_
\$60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
\$70	p	q	r	s	t	u	v	w	x	y	z	[	\	]	^	_
\$80	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
\$90	~	!	!"	!"	!"	!"	!"	!"	!"	!"	!"	!"	!"	!"	!"	!"
\$A0	#	\$	%	&	'	(	)	*	+	,	-	.	/	:	;	=
\$B0	>	?@	A	B	C	D	E	F	G	H	I	J	K	L	M	N
\$C0	O	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^
\$D0	_	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n
\$E0	o	p	q	r	s	t	u	v	w	x	y	z	[	\	]	^
\$F0	_	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n

(備考)

キャラクター表は下記のプログラムで作成しています。

```

10 'font map
20 N=16
30 GOSUB "@hr"
40 FOR C=0 TO 255 STEP N
50 GOSUB "@line(C,N)"
60 GOSUB "@hr"
70 NEXT C
80 END
90 "@line(C,N)"
100 FOR L0=0 TO 7
110 ?" |";
120 FOR I0=C TO C+N-1
130 D0=PEEK(FNT+8*I0+L0)
140 FOR J0=0 TO 5
150 IF D0&($80>>J0) ?"#"; ELSE ?" ";
160 NEXT J0
170 ?" |";
180 NEXT I0
190 ?
200 NEXT L0
210 RETURN
220 "@hr"
225 ?" ";
230 FOR I0=0 TO 111
240 ?"-";
250 NEXT I0
260 ?
270 RETURN

```

ライセンスに関する表記



[CC BY IchigoJam](https://creativecommons.org/licenses/by/4.0/)

フォントデータは IchigoJam 1.2.1 のフォントを参考にして作成しています。

(絵文字部分は IchigoJam の 8x8 フォントを見ながら自作しました)

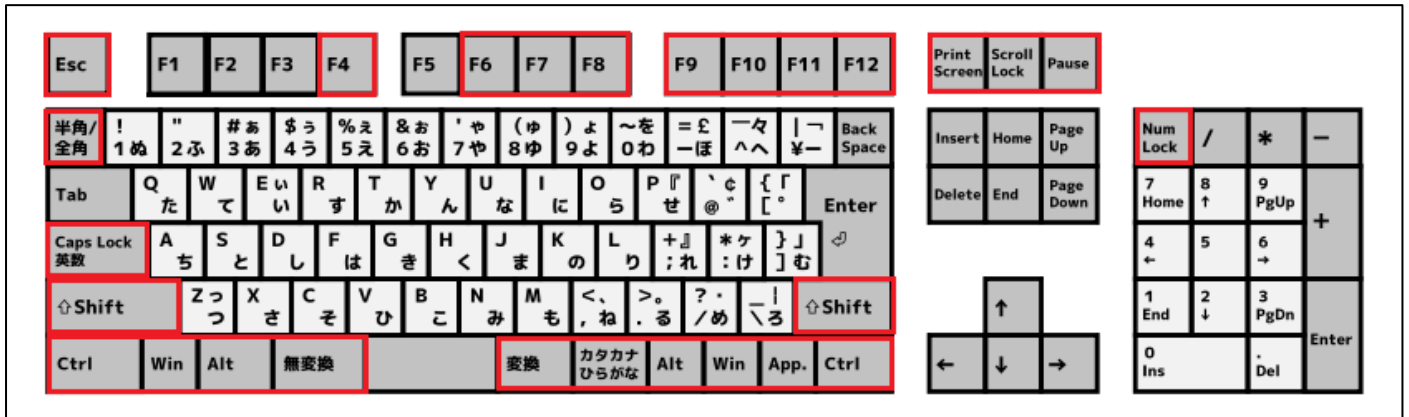
## 1.13 キー入力コード

INKEY()関数等で入力したキーを判定する際は、キー入力コードを用います。

大部分のキー入力コードは入力する文字コード、ASCIIコードと一致します。カーソルキー等の入力制御用のキーは、「豊四季タイニーBASIC for Arduino STM32」固有のコードとなります。

□ キー入力コードが取得出来ないキー

赤枠のキーは、制御用または利用不可キーとなっています。



(注意)ターミナルコンソールの通常状態(SCREEN 0 以外のモード)では、BackSpace キーを覗き、制御・編集用キーは利用出来ません。ターミナルコンソールの SCREEN 0 では、F4、F6~F12 は値を返しますが、サポート範囲外のため利用をは推奨しません。

106/109 キーボード (※画像は ウィキペディア <https://ja.wikipedia.org/wiki/キー配列> より拝借)

□ キー入力コード一覧

キーボード入力にて取得出来るキー入力コードを以下に示します。

コードに数値の記載のあるキーに限りコードの取得が可能です。

キー	併用なし		Shift		CTRL		カタカナ		SHIFT+カタカナ	
	文字	コード	文字	コード	文字	コード	文字	コード	文字	コード
[ESC]										
[F1]		12		12		12		12		12
[F2]		4		4		4		4		4
[F3]		14		14				14		14
[F4]										
[F5]		18		18		18		18		18
[F6]										
[F7]										
[F8]										
[F9]										
[F10]										
[F11]										
[F12]										

■ はサポートしていないキーです。

キー	併用なし		Shift		CTRL		カタカナ		SHIFT+カタカナ	
	文字	コード	文字	コード	文字	コード	文字	コード	文字	コード
[Backspace]		8						8		8
[Enter]		13						13		13
[Tab]		9						9		9
[CapsLock]										
[Shift]										
[Ctrl]										
[Win]										
[Alt]										
[無変換]										
[空白 (Space)]	空白	32					空白	32	空白	32
[変換]										
[カカヒらがなローマ字]										
[メニュー]										
[Insert]		134						134		134
[Home]		132						132		132
[PageUp]		136						136		136
[Delete]		133						133		133
[End]		137						137		137
[PageDown]		135						135		135
[↑]		129						129		129
[↓]		128						128		128
[←]		130						130		130
[→]		131						131		131
[PrintScreen]										
[ScrollLock]										
[Pause/Break]										
[! め]	1	49	!	33			1	49	!	33
[" ふ]	2	50	"	34			2	50	"	34
[# ああ]	3	51	#	35			3	51	#	35
[\$ うう]	4	52	\$	36			4	52	\$	36
[% ええ]	5	53	%	37			5	53	%	37
[& おお]	6	54	&	38			6	54	&	38
[ ' やや]	7	55	'	39			7	55	'	39
[(8 ゆゆ]	8	56	(	40			8	56	(	40
[)9 よよ]	9	57	)	41			9	57	)	41
[ 0 をわ]	0	48					0	48	0	48
[ - ほ]	-	45	=	61	CTRL -	29	-	176	=	61
[ ~ へ]	^	94	~	126			^	94	~	126
[   ￥]	¥	92		124			¥	92		124

はサポートしていないキーです。

キー	併用なし		Shift		CTRL		カタカナ		SHIFT+カタカナ	
	文字	コード	文字	コード	文字	コード	文字	コード	文字	コード
[Q た]	q	113	Q	81	CTRL Q	17	q	113	Q	81
[W て]	w	119	W	87	CTRL W	23	w	119	W	87
[E いい]	e	101	E	69	CTRL E	5	エ	180	エ	180
[R す]	r	114	R	82	CTRL R	18	r	114	R	82
[T か]	t	116	T	84	CTRL T	20	t	116	T	84
[Y ん]	y	121	Y	89	CTRL Y	25	y	121	Y	89
[U な]	u	117	U	85	CTRL U	21	ウ	179	ウ	179
[I に]	i	105	I	73	CTRL I	9	イ	178	イ	178
[O ら]	o	111	O	79	CTRL O	15	オ	181	オ	181
[P せ]	p	112	P	80	CTRL P	16	p	112	P	80
[`@` ]	@	64	`	96			`	222	`	96
[ [ [ ` ] ] ]	[	91	{	123			「	162	°	223
[A ち]	a	97	A	65	CTRL A	1	ア	117	ア	117
[S と]	s	115	S	83	CTRL S	19	s	115	S	83
[D し]	d	100	D	68	CTRL D	4	d	100	D	68
[F は]	f	102	F	70	CTRL F	6	f	102	F	70
[G き]	g	103	G	71	CTRL G	7	g	103	G	71
[H く]	h	104	H	72	CTRL H	8	h	104	H	72
[J ま]	j	107	J	74	CTRL J	10	j	107	J	74
[K の]	k	107	K	75	CTRL K		k	107	K	75
[L り]	l	108	L	76	CTRL L	12	l	108	L	76
[+:れ]	;	59	+	43			;	59	+	43
[*:け]	:	58	*	42	CTRL *	28	:	58	*	42
[ ] り む]	]	93	}	125			」	163	}	125
[Z っ]	z	122	Z	90	CTRL Z	26	z	122	Z	90
[X さ]	x	120	X	88	CTRL X	24	x	120	X	88
[C そ]	c	99	C	67	CTRL C		c	99	C	67
[V ひ]	v	118	V	86	CTRL V	22	v	118	V	86
[B こ]	b	98	B	66	CTRL B	2	b	98	B	66
[N み]	n	110	N	78	CTRL N	14	ン	221	ン	221
[M も]	m	109	M	77	CTRL M	13	m	109	M	77
[<,、ね]	,	44	<	60			、	164	<	60
[>。る]	.	46	>	62	CTRL >	30	。	161	>	62
[?/・め]	/	47	?	63	CTRL ?	31	/	47	?	63
[¥_ろ]	¥	92	_	95			¥	92	_	95

■はカタカナローマ字入の子音のため、2 文字入力にてコードが確定します。

## 1.14 編集操作操作キー

「豊四季タイニーBASIC for Arduino STM32」では、NTSC ビデオ出力画面、シリアルコンソール画面(CONSOLE 0)でのフルスクリーン編集をサポートしています。

## 編集キー一覧

編集キー	機能
[ESC]	プログラム中断、シリアルコンソールでは要2回押し
[F1]	画面の全消去
[F2]	カーソル位置の行消去
[F3]	カーソルの次行に空行挿入
[F5]	画面の再表示
[BackSpace]	カーソル前の文字の削除
[Insert]	上書きモード、挿入モードも切り替え
[Home]	カーソルを行の先頭に移動
[END]	カーソルを行の末尾に移動
[PageUP]	カーソルを画面右上に移動、画面のスクローダウン
[PageDown]	カーソルを表示している最終行に移動、スクロールアップ
[Delete]	カーソル位置の文字の削除
[←]	カーソルを左に移動
[→]	カーソルを右に移動
[↑]	カーソルを上移動
[↓]	カーソルを下移動
[Enter]	行入力の確定、改行
[NumLock]	テンキーのロック、ロック解除
[カタカナ/ひらがな/ローマ字]	カタカナ入力のON、OFF
[Ctrl] + C	プログラム中断
[Ctrl] + D	カーソル位置の行削除
[Ctrl] + K	カタカナ入力のON、OFF
[Ctrl] + L	画面の全消去
[Ctrl] + N	カーソルの次行に空白挿入
[Ctrl] + R	画面の再表示
[Ctrl] + X	カーソル位置の文字の削除

## 2. 実行環境の構築（執筆中）

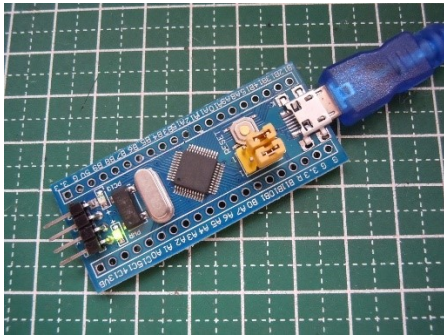
この章では、安価な Blue Pill ボードで「豊四季タイニーBASIC for Arduino STM32」を稼働させるための作業について説明します。初めて構築する場合は、最小構成から構築し問題なく動作したらパーツを追加して機能拡張を行っていくことをお勧めします。

### 2.1 最小限構成の実行環境構築

最小構成「Blue Pill ボード」＋「USB ケーブル」＋「ホストパソコン」で利用する場合の環境構築について説明します。この構成では、Blue Pill ボードに追加するパーツ類は不要です。

手順：

- ① Blue Pill ボードの入手
- ② 購入直後の動作確認
- ③ 豊四季 Tiny BASIC for Arduino STM32 の書込み



#### 1) 購入直後の動作確認

USB ケーブルにてパソコンや AC アダプタの USB 端子に接続し、ボード上の 2 つの LED の点灯を確認します。Blue Pill ボードには動作確認用のサンプルプログラムが書き込まれています。

#### チェックポイント

電源 ON を示す LED：常時点灯

プログラムから制御できる PC13 ポートに接続している LED：1 秒周期で点滅

**（注意）** パソコンに接続した場合、不明な USB デバイスとして警告画面が表示される場合があります。この段階では、警告を無視して下さい。

#### 2) Arduino STM32 用ブートローダの書込み

Blue Pill ボードへのブートローダの書込み、  
パソコンへのドライバーソフトのインストール

### 2.2 圧電サウンダー（圧電スピーカー）の追加

### 2.3 SD カードモジュールの追加

### 2.4 NTSC ビデオ出力の追加

## 2.5 PS/2 キーボード インタフェースの追加

### 3. 利用環境設定

本章では、「豊四季タイニーBASIC for Arduino STM32」を使ってプログラムの開発を行うための環境及びその設定について、解説します。

#### 3.1 ファームウェア(スケッチ)更新後の初期設定

「豊四季タイニーBASIC for Arduino STM32」のファームウェア（スケッチ）を初めての利用、またはバージョンの更新を行った場合、内部フラッシュメモリのプログラム保存及び CONFIG データ保存用の領域の初期化が必要となります。

初期化は次の作業を行って下さい。

##### ① プログラム保存領域の初期化

内部フラッシュメモリの初期化を行います。

詳細については下記の項目を参照して下さい。

⇒ 「3.13 プログラム保存領域の初期化」

##### ② 仮想 EEPROM 領域の初期化

システム設定情報を保存する領域の初期化を行います。

詳細については下記の項目を参照して下さい。

⇒ 「3.12 仮想 EEPROM 領域の初期化」

初期化を行った場合、再設定が必要となります。

#### 3.2 PS/2 キーボードに関する設定

PS/2 キーボードは、日本語キーボードと US キーボードに対応しています。

デフォルトでは日本語キーボード対応になっています。

利用するキーボードのタイプを一時的に切り替えるには [CONFIG](#) コマンドを利用します。

書式

CONFIG 1, キーボード選択 (0:日本語キーボード 1:US キーボード)

例: US キーボードに切り替える

```
CONFIG 1,1
```

例: 日本語キーボードに切り替える

```
CONFIG 1,0
```

設定した状態を保存するのは、[SAVECONFIG](#) コマンドを実行します。

```
SAVECONFIG
```

これにより、次回以降の利用にも反映されます。



## 3.3 コンソール画面の表示設定

コンソール画面は、画面解像度、画面文字サイズ、画面の向きなどの設定変更が可能です。

表示設定は利用しているデバイスコンソール画面（シリアル、NTSC、OLED、TFT）により異なります。

下記に表示設定を行う、コマンドを利用することで、表示設定を行うことができます。

コンソール画面	コマンド	説明
NTSC ビデオ出力画面	SCREEN n	画面解像度を設定します。 n：スクリーンモード 1：画面(解像度 224×216ドット、テキスト 37×27文字 2：画面(解像度 224×108ドット、テキスト 37×13文字 3：画面(解像度 112×108ドット、テキスト 18×13文字
TFT 液晶画面	SCREEN n [, m]	画面フォントサイズ、画面向きを設定します。 n：スクリーンモード 1：画面(解像度 320×240、ノーマルフォント テキスト 53×30 文字 2：画面(解像度 320×240、2 倍角フォント テキスト 26×15 文字 3：画面(解像度 320×240、3 倍角フォント テキスト 17×10 文字 4：画面(解像度 320×240、4 倍角フォント テキスト 13×7 文字 5：画面(解像度 320×240、5 倍角フォント テキスト 10×6 文字 6：画面(解像度 320×240、6 倍角フォント テキスト 8×5 文字  m：画面向き 0～4：時計回りに 90° づつ回転（デフォルト：3）
	COLOR n [, m]	テキスト文字色、背景色を設定します。 n：文字色 0～8 、\$0000～\$FFFF m：背景色 0～8 、\$0000～\$FFFF  COLOR コマンドの色指定については、「7.47 COLOR 文字色の設定」を参照して下さい。  COLOR コマンド実行後、CLS コマンドを実行するか、[F1]キーを押すことで画面全体に色設定が反映されます。
OLED 画面	SCREEN n [, m]	n：スクリーンモード 1：画面(解像度 128×64、ノーマルフォント テキスト 21×8 文字 2：画面(解像度 128×64、2 倍角フォント テキスト 10×4 文字 3：画面(解像度 128×64、3 倍角フォント テキスト 7×2 文字  m：画面向き 0～4：時計回りに 90° づつ回転（デフォルト：0）
シリアルコンソール	WIDTH n , m	テキスト表示の縦・横文字数を設定します。 n：横文字数 16～128（デフォルト 80） m：縦文字数 10～45（デフォルト 24）  USB シリアルポートまたは、GPIO シリアルポート接続をコンソール画面として利用している場合に設定可能です。
	SMODE n [, m]	2 つのシリアルポートの機能を切り替えます  SMODE 0 USB シリアルポートをコンソール画面として利用し、GPIO シリアルポートはデータ通信用に利用します（デフォルト）。  SMODE 1, “通信速度” GPIO シリアルポートをコンソール画面として利用し、USB シリアルポートはデータ通信用に利用します。第 2 引数には通信速度を指定します。 例：SMODE 1, “115200”
共通	CONSOLE n	デバイスコンソール画面（NTSC、TFT、OLED）から、シリアルコンソール画面利用への切り替えまたは、その逆を行います。 0：デバイスコンソール画面の利用に切り替えます 1：シリアルコンソール画面の利用に切り替えます  シリアルコンソール画面では PS/2 キーボードは利用出来ません。

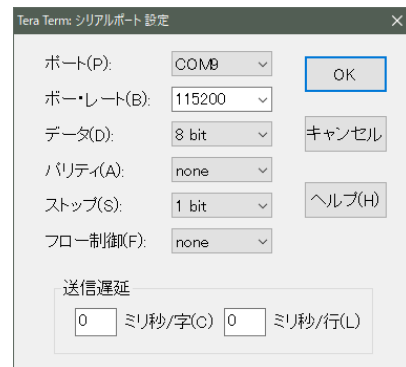
## 3.4 シリアルコンソールの通信条件の設定

ここでは、Windows 10 パソコン上でターミナルソフト TeraTerm を利用する場合の設定について解説します。  
Blue Pill ボードでは、USB ポートを使った接続とシリアルポートを使った2つの方法にて利用出来ます。

## 1) USB 接続（Blue Pill ボードの micro USB コネクタ経由）の場合

## ①通信条件の設定

ポート	設定値
ポート	各自の環境に合わせて設定
ボー・レート	115200(任意)
データ長	8 bit
パリティビット	無し
ストップビット	1bit
フロー制御	無し



USB 接続の場合、ボー・レートの設定任意です。

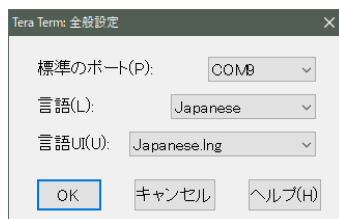
ボー・レートの設定は無視され USB 通信の最適な速度で通信を行うようです（推測）。

TeraTerm の設定は、メニュー[設定] - [シリアルポート]から[シリアルポート設定]画面を開いて行います。

## ②言語等に関する設定

「豊四季 Tiny BASIC for Arduino STM32」には 1 バイトコードの半角カタカナの利用をサポートします。  
半角カタカナ利用のためには、利用する文字列コードとしてシフト JIS を指定します。

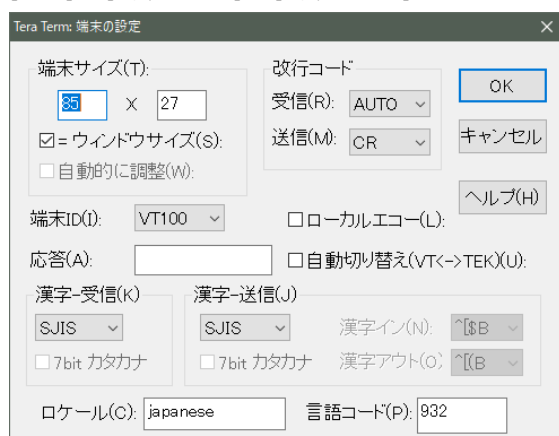
[設定] - [全般設定] - [全般設定]画面 の設定



言語 : Japanese

言語 UI : Japanese.lng

[設定] - [端末の設定] - [端末の設定]画面 の設定



漢字-受信 : SJIS

漢字-送信 : SJIS

改行コード受信 : AUTO、送信 CR

ウィンドウサイズ : ☒チェックを入れる

## 2) GPIO シリアルポート（PA9、PA10）の場合

接続には USB-シリアル変換モジュール等が必要となります。

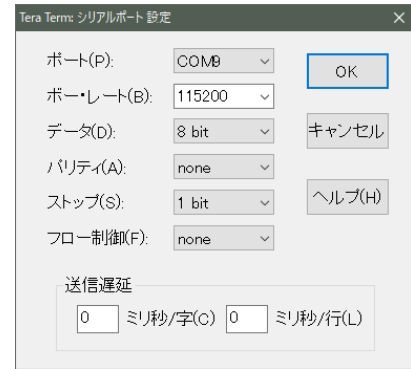
## ①通信条件の設定

ポート	設定値
ポート	各自の環境に合わせて設定
ボー・レート	115200(任意)
データ長	8 bit
パリティビット	無し
ストップビット	1bit
フロー制御	無し

ボー・レートの設定は利用する USB-シリアル変換モジュールの性能の範囲で自由に選択して下さい。115200 以上での利用を推奨します。

Blue Pill ボードでは、TeraTerm で設定可能な最高速度 921600 での通信も可能です。

TeraTerm の設定は、メニュー[設定] - [シリアルポート]から[シリアルポート設定]画面を開いて行います。



## ②言語等に関する設定

設定は、「1) USB 接続 (Blue Pill ボードの micro USB コネクタ経由) の場合」と同じです。

## 3.5 NTSC ビデオ出力の同期信号の調整

NTSC ビデオモニターの映像がスクロールする等、正常に表示出来ない場合は、次の調整を試してみてください。改善される場合があります。

NTSC 垂直同期信号補正

CONFIG 0, 補正 (0~2 デフォルト値:0)

例:

```
CONFIG0,2
```

設定した状態を保存するのは、[SAVECONFIG](#) コマンドを実行します。

```
SAVECONFIG
```

これにより、次回以降の利用にも反映されます。

## 3.6 起動時のコンソール画面の選択

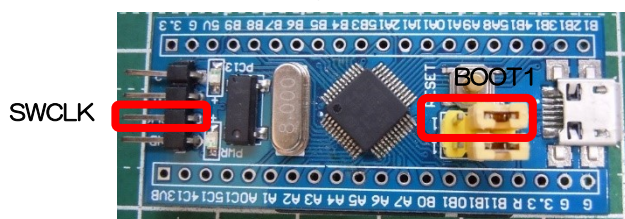
デフォルトでは、ボードを起動するとデバイス画面が、コンソール画面となります。

NTSC 版では、NTSC ビデオ出力画面、OLED 版では、

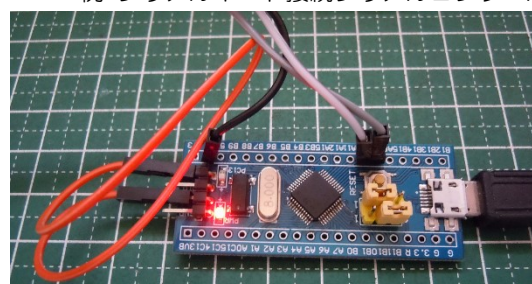
起動時に次の設定にて利用するスクリーン画面を選択することが出来ます。

コンソール画面	BOOT1	SWCLK	備考
デバイスコンソール画面(NTSC/OLED/TFT)	0	-	
USB ポート接続シリアルコンソール	1	OPEN or LOW	
シリアルポート接続シリアルコンソール	1	HIGH	通信速度 115,200bps

BOOT1 と SWCLK 端子



例: シリアルポート接続シリアルコンソール選択



起動時に次の設定にて利用するスクリーン画面を選択することが出来ます。

(注意) SWCLK を 3.3V に結線した場合、2 点間に  $80\mu\text{A}$  (実測) の電流が流れます (0.265mW)。消費電力を抑えたい場合は、間に大き目の抵抗 ( $100\text{K}\Omega\sim 1\text{M}\Omega$ ) を間に入れて下さい。  $1\text{M}\Omega$  の抵抗を入れた場合、 $3\mu\text{A}$  (0.001mW) まで抑えられます。

### 3.7 内部 RTC の時刻設定

内部 RTC の時刻設定は次のコマンドで行います。

SETDATE 年,月,日,時,分,秒

例 : SETDATE 2017,8,10,12,30,0

設定した時刻は DATE コマンドで確認できます。

例 : DATE

2017/08/10 [Thr] 12:32:45

RTC の設定時刻は、SD カード利用した場合のファイル作成・更新時刻としても利用されます。

### 3.8 プログラムの保存と読み込み

作成したプログラムは内部フラッシュメモリに保存可能です。

フラッシュメモリへの保存

[SAVE](#) プログラム番号 (0~7)

フラッシュメモリからの読み込み

[LOAD](#) プログラム番号 (0~7)

フラッシュメモリに保存しているプログラムのリスト表示

[FILES](#)

```
files
0:CLS
1:'oscilloscope
2:'シコリョウ
3:(none)
4:(none)
5:(none)
6:(none)
7:(none)
OK
```

FILES コマンドはプログラムの先頭行を表示します。

プログラム番号の格納領域にプログラムが保存されていない場合は、(none)が表示されます。

フラッシュメモリ上のプログラムは ESASE コマンドにて削除することが出来ます。

[LRUN](#) コマンドを用いると、指定したプログラムをロードして実行することが出来ます。

```
LRUN 1
```

マイコンボードの起動時に予め指定したプログラムを起動することも可能です。

詳細は、「3.11 プログラムの自動起動設定」を参照して下さい。

また、SD カードモジュールを接続している場合、MMC 互換カード (SD カード含む) へのプログラムの保存、読み込みが可能となります。

MMC 互換カード(SD カード含む)への保存

[SAVE](#) "ファイル名"

MMC 互換カード(SD カード含む)からの読み込み

[LOAD](#) "ファイル名"

MMC 互換カード(SD カード含む)に保存しているプログラムのリスト表示

[FILES](#) ""

ファイルの保存先には、ディレクトリ名の指定も可能です。

詳細については、各コマンドのリファレンスを参照して下さい。

### 3.9 パソコンからのプログラム読込と保存

パソコンからマイコンボードにプログラムを転送する方法としては、  
ターミナルソフトの画面にプログラムソースコピー＆ペーストする方法が簡単でおすすめです。

通常、ペースト時には一部の制御用文字（BS キーの文字コードなど）が機能してしまい、正しく転送出来ない場合があります。この場合は、シリアルポートのコンソール画面利用をしていない状態で、

[SMODE](#) コマンドを使って一時的に制御用文字コードの機能を無効化してから転送して下さい。

制御用文字コードの機能を無効化

```
SMODE 3,0
```

制御用文字コードの機能を有効化

```
SMODE 3,1
```

(注意) シリアルポートをコンソール画面（スクリーン編集可能な画面として利用している場合）として利用している場合、  
SMODE コマンドによる制御用文字コードの機能を無効化は行うことが出来ません。

### 3.10 フラッシュメモリのエクスポート

内部フラッシュメモリに保存されている、プログラムをパソコンにバックアップする機能として、  
[EXPORT](#) コマンドが用意されています。

パソコン上のターミナルソフトから接続した状態で、EXPORT コマンドを実行します。

ターミナルソフト上に保存されているプログラムが全て出力されますので、その内容をコピー＆ペーストにてメモ帳等に張り付けて保存します。

```
export
NEW
10 CLS
20 GETTIME H,M,S
30 @(0)=H/10: @(1)=H%10
40 @(2)=10
50 @(3)=M/10: @(4)=M%10
60 @(5)=10
70 @(6)=S/10: @(7)=S%10
80 FOR I=0 TO 7
90 BITMAP I*24+10,20,FNT,@(I)+ASC("0"),6,8,4
100 NEXT I
110 GOTO 20
SAVE 0

NEW
10 'oscilloscope
20 CLS
```

```

30 GPIO PB1,ANALOG
40 "LOOP"
50 R=ANA(PB1)
60 Q=R/20
70 LOCATE 0,0:?"R;"
80 PSET GW-1,GH-Q-4,1
90 WAIT 50
100 GSCROLL 8,208,3
110 GOTO "LOOP"
SAVE 1

NEW
1 "ｼﾞｭｶﾅﾅｼﾞ"
5000 CLS
5010 GETTIME H,M,S
5020 @(0)=H/10:@(1)=H%10
5030 @(2)=10
5040 @(3)=M/10:@(4)=M%10
5050 @(5)=10
5060 @(6)=S/10:@(7)=S%10
5070 FOR I=0 TO 7
5080 BITMAP I*24+10,20,FNT,@(I)+ASC("0"),6,8,4
5090 NEXT I
5100 GOTO 20
SAVE 2

OK

```

保存したプログラムをマイコンボードに読み込むには「3.9 パソコンからのプログラム読み込みと保存」の内容を参考にして、保存していたテキストの「NEW～SAVE X」の単位でコピー＆ペーストしてプログラム番号単位で保存していきます。作業については、「7.18EXPORT 内部フラッシュメモリの内容のエクスポート」の内容も参照して下さい。

### 3.11 プログラムの自動起動設定

マイコンボードの起動時にフラッシュメモリー上の指定したプログラム番号のプログラムを起動することが出来ます。この設定は CONFIG コマンド、SAVECONFIG コマンドにて行います。プログラム番号 0 を自動起動する場合は、次のコマンドを実行します。

```

CONFIG 2,0
OK
SAVECONFIG
OK

```

CONFIG コマンドの第 2 引数にプログラム番号(0～7)を指定します。

自動起動を解除する場合は、次のコマンドを実行します。

```

CONFIG 2,-1
OK
SAVECONFIG
OK

```

自動起動するプログラムには、SD カード上のプログラムを指定することは出来ませんが、フラッシュメモリーから起動されるプログラムから RLUN コマンドを使って間接的に起動することが出来ます。

```

10 LRUN "STAT.BAS"

```

### 3.12 仮想 EEPROM 領域の初期化

仮想 EEPROM 領域には、システム設定情報とユーザー用仮想 EEPROM 用に利用します。この領域を初期化は [EEPFORMAT](#) コマンドを実行して下さい。具体的には、次のコマンドを実行して下さい。

```

EEPFORMAT
OK

```

### 3.13 プログラム保存領域の初期化

内部フラッシュメモリの初期化は、[ERASE](#) コマンドを利用します。具体的には、次のコマンドを実行して下さい。

ERASE 0,7 OK
-----------------

### 3.14 MMC 互換カード(SD カード含む)の初期化

MMC 互換カード(SD カード) の初期を行い場合は、FAT16 たまは FAT32 形式で行って下さい。

通常、購入した SD カードは FAT16 または FAT32 にてフォーマットされており、そのまま利用することが出来ます。

なお、MMC 互換カード(SD カード) は電氣的に弱い面もありますので、万が一の故障を想定し、重要なデータを保存する用途との併用は避け、なるべく専用のカードを用意して下さい。

## 4. 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！

### 4.1 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！

### 4.2 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
■
```

表示出来たね。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！

### 4.3 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]キーを押すのを忘れずに！

```
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。



#### 4.4 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

```
10 print "Hello,Tiny BASIC" Enter
Hello,Tiny BASIC
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。

#### 4.5 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。よく使うコマンドだよ。

## 4.6 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 
7 for i=1 to 3 
20 next i 
```

入力したプログラムを確認！

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

今入力した 5,7,10 行が追加されたね。プログラムは行番号順に表示されるよ。

それでは実行！ run コマンドで実行するよ。

```
run 
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 が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。

## 4.7 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グラムの行番号を 10 ずつに整えるよ。

```
return 
OK
List 
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。

#### 4.8 やっぱり 10 行は不要、削除したい

やっぱり、直前の表示内容は消したくない。

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。

```
renum 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。

#### 4.9 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のだ。

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。

#### 4.10 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を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！

## 5. プログラム構成要素の説明（コマンド・関数等）

本章では、「豊四季タイニーBASIC for Arduino STM32」がサポートするプログラム言語について解説します。

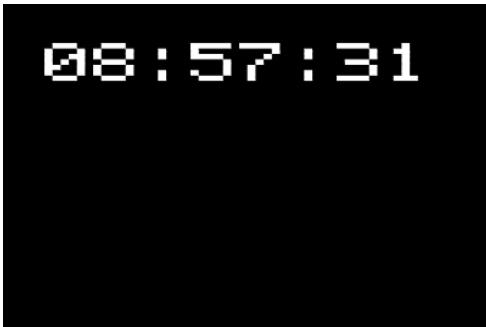
### 5.1 プログラムの構造と形式

次のプログラムリストは、画面に時刻を表示するプログラムです。

実行すると NTSC ビデオ画面に時刻を更新表示します。

```
10 CLS:SETDATE 2017,8,1,8,57,0
20 GETTIME H,M,S
30 @(0)=H/10: @(1)=H%10
40 @(2)=10
50 @(3)=M/10: @(4)=M%10
60 @(5)=10
70 @(6)=S/10: @(7)=S%10
80 FOR I=0 TO 7
90 BITMAP I*24+10,20,FNT,@(I)+ASC("0"),6,8,4
100 NEXT I
110 GOTO 20
```

実行結果



プログラムの構造的は・・・

プログラムは複数の行（行番号＋命令文）で構成されます。

例）10 CLS: SETDATE 2017,8,1,8,57,0 ：画面の表示内容を全消去し、時刻を設定する

命令文は1 つまたは複数のコマンドや式で構成されます。

例）30 @(0)=H/10: @(1)=H%10 ：2つの代入式を記述 配列変数@(0)、@(1)に値を代入する

コマンドはコマンド名と引数で構成されます。

例）20 GETTIME H, M, S ：変数 H, M, S に現在の時間、分、秒を代入する

引数は式、変数、定数にて構成されます。

例）BITMAP I\*24+10,20,FNT,@(I)+ASC("0"),6,8,4：引数に式を記述しその計算結果を引数として渡している

基本的に、プログラムは行番号順にシーケンシャルに実行されます。

例）10 行から 70 行までは行番号順に実行し配列変数@(0)から@(6)に時刻情報を設定します

制御命令により、分岐や繰り返しを行うことが出来ます。

例）80 行～100 行は FOR ～ NEXT により、8 回処理を繰り返します。

例）20 行～110 行は GOTO 文により無限に処理を繰り返します。

このようにプログラムは様々な要素で構成れます。  
次節からは、これらの各構成要素について解説します。

## 5.2 行と命令文

### □ 行

行とは利用者がコンピュータに対して指示を行う単位です。行は命令文にて構成されます。  
「豊四季タイニーBASIC for Arduino STM32」では、行の単位で利用者からの指示を処理します。

スクリーンエディタ、およびコンソールから、

```
PRINT "Hello" Enter
Hello
OK
```

のように **Enter** キーの入力により、行の単位で指示を受け取りその処理を行います。  
また、行の先頭に行番号を付けた場合、

```
10 PRINT "Hello" Enter
```

#### 「指定した行番号で行（命令文）をプログラムに登録せよ」

との指示であると判断し、命令文は実行せずに、行（命令文）をプログラムとしてメモリーに登録します。

```
10 Enter
```

のように行番号だけを指定した場合は、

#### 「指定した行番号で行（命令文）を削除せよ」

との指示であると判断し、該当する行をプログラムから削除します。

### □ 命令文

命令文とはコマンド、式、関数、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行に複数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:'サンプル
```

#### コメント文

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

例：

```
10 'Smple program
20 REM Sample program
30 PRINT "Hello":'print hello
```

#### ラベル

行の先頭のダブルクォーテーションで囲った文字列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。

例：

```
10 "LOOP":PRINT "Hello"
20 GOTO "LOOP"
```

### 5.3 制御文

制御文は制御命令を使ったプログラムの記述文です。複数のキーワード、式の組み合わせで構成されます。プログラムは RUN コマンドを実行することにより、通常は先頭行から行番号順に逐次実行されます。この逐次実行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できます。

FOR 文の例：繰り返し処理を行う

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

以下に制御命令の一覧を示します。

制御命令	説明
<a href="#">GOTO</a> 行番号 <a href="#">GOTO</a> ラベル	指定行へのジャンプ
<a href="#">GOSUB</a> 行番号 <a href="#">GOSUB</a> ラベル	サブルーチンの呼び出し
<a href="#">RETURN</a>	GOSUB 文サブルーチン呼び出しからの復帰
<a href="#">IF</a> 条件式 実行文 1 <a href="#">IF</a> 条件式 実行文 1 ELSE 実行文 2	条件判定
<a href="#">FOR</a> 変数=初期値 TO 最終値 実行文(複数行可能) NEXT 変数 <a href="#">FOR</a> 変数=初期値 TO 最終値 STEP 増分 繰り返し実行文 NEXT 変数	繰り返し
<a href="#">END</a>	プログラムを終了する

各制御命令の詳細については、「6 各制御文の詳細」を参照して下さい。

### 5.4 コマンド・関数

#### コマンド

コマンドとは何らかの処理を行うプログラム要素です。コマンドには一般コマンドとシステムコマンドの2種類があります。一般コマンドはコマンドラインからの直接利用とプログラム内での利用が可能です。

システムコマンドは、プログラムの管理を行うコマンドです。プログラム中で利用した場合、不整合が生じるため、コマンドラインでのみ利用可能としています。

システムコマンドをプログラム中で利用した場合、エラー("Cannot use system command")となります。

例：プログラム内にシステムコマンド [DELETE](#) を記述

```
10 ? "SAMPLE"
20 DELETE 10
30 END
RUN
Cannot use system command in 20
20 DELETE 10
OK
```

#### 関数

関数とは何らかの値を返す命令文です。関数には数値関数と文字列関数の2種類があります。

数値関数は数値を返す関数です。文字列関数は文字列を返す関数です。

数値関数はコマンドの引数や式のなどの数値として利用します。

文字列関数は PRINT,SPRINT,GPRINT の引数、ファイル名の引数にて利用できます。

例：関数の利用

```
10 A=ASC("A")*8+FNT
20 FOR I=0 TO 7
30 PRINT BIN$(PEEK(A+I),8)
40 NEXT I
50 END
```

```
run
01110000
10001000
10001000
10001000
11111000
10001000
10001000
00000000
OK
```

上記のプログラムは文字“A”のフォントパターンを2進数で表示します。

10行にて数値関数 [ASC\(\)](#) を利用しています。30行で文字列関数 [BIN\\$\(\)](#) を利用しています。

関数単独での利用は出来ません。コマンドや関数の引数、式中でのみ利用可能です。

例：関数の利用は文法エラーとなる

```
ASC("A")
Syntax error
OK
```

## 5.5 コマンド・関数一覧

機能別に分類したコマンド及び関数の一覧を示します。

各コマンド及び関数の利用方法の詳細については、「7.各コマンド・関数の詳細」を参照して下さい。

### □ システムコマンド

<a href="#">RUN</a>	プログラムの実行
<a href="#">RENUM</a> <a href="#">RENUM</a> 先頭行番号 <a href="#">RENUM</a> 先頭行番号,間隔	行番号の振り直し 先頭行番号：振り直し開始先頭行 間隔：行間の増分
<a href="#">DELETE</a> 行番号 <a href="#">DELETE</a> 先頭行番号,末尾行番号	プログラムの指定行の削除

### □ プログラム関連

コマンド

<a href="#">CONSOLE</a> モード	コンソール画面の切替 モード OFF(0)：デバイスコンソール ON(1)：シリアルコンソール
<a href="#">SCREEN</a> モード, [画面向き]	スクリーンモードの変更 モード NTSC 版 1: 224x216 ドット、2: 224x108 ドット、3: 112x108 ドット OLED 版 1: 21x8 文字、2: 10x4 文字、3: 7x2 文字 TFT 版 1: 53x30、2: 26x15、3: 17x10、4: 13x7、5: 10x6、6: 8x5 文字  画面向き OLED、TFT 版でのみ指定可能 0~3 で時計回りで 90 度ずつ回転
<a href="#">WIDTH</a> 横文字数,縦文字数	シリアルターミナルの画面サイズの設定
<a href="#">LIST</a> <a href="#">LIST</a> 開始行番号 <a href="#">LIST</a> 開始行番号,終了行番号	プログラムリストの表示 開始行番号：表示を開始する行 終了行番号：表示を終了する行
<a href="#">NEW</a>	プログラムの消去
<a href="#">LOAD</a> <a href="#">LOAD</a> プログラム番号 <a href="#">LOAD</a> "ファイル名"[, モード]	内部フラッシュメモリ/SD カードからプログラム読み込み プログラム番号：読み出し元番号 (0~9) "ファイル名":SD カード上のファイル名 モード:0:上書き読み込み 1:追記読み込み

<a href="#">SAVE</a> <a href="#">SAVE</a> [プログラム番号] <a href="#">SAVE</a> “ファイル名”	内部フラッシュメモリ/SD カードへのプログラム保存 プログラム番号: 保存先番号 (0~9) “ファイル名”:SD カード上のファイル名
<a href="#">FILES</a> <a href="#">FILES</a> 開始[,終了] <a href="#">FILES</a> “パス名”	内部フラッシュメモリ/SD カードのプログラム一覧表示 開始,終了:プログラム番号 “パス名”:フォルダ名、ファイル名 (ワイルドカード可能)
<a href="#">BLOAD</a> “ファイル名”,アドレス ,バイト数	SD カードからバイナリデータ読込
<a href="#">BSAVE</a> “ファイル名”,アドレス ,バイト数	SD カードへのバイナリデータ保存
<a href="#">REM</a> [コメント文] or “[コメント文]	コメント 省略形の (シングルクォーテーション) も可能
<a href="#">LET</a> 変数=式 <a href="#">LET</a> @(添え字)=n1 [,n2,n3,n4…]	変数・配列変数に値を代入 (LET は省略可能) 配列変数では連続代入指定が可能
<a href="#">CLV</a>	変数領域の初期化
<a href="#">LRUN</a> プログラム番号 <a href="#">LRUN</a> プログラム番号,行番号 <a href="#">LRUN</a> プログラム番号,“ラベル” <a href="#">LRUN</a> “ファイル名” <a href="#">LRUN</a> “ファイル名”,行番号 <a href="#">LRUN</a> “ファイル名”,“ラベル”	指定したプログラムを内部フラッシュ/SD カードから読込んで実行する
<a href="#">EXPORT</a> <a href="#">EXPORT</a> 対象プログラム番号 <a href="#">EXPORT</a> 開始プログラム番号,終了プログラム番号	内部フラッシュメモリの内容をエクスポートする 対象番号                      エクスポートする番号 開始番号, 終了番号      エクスポートする範囲
<a href="#">CONFIG</a> 項目番号,設定値	システムの設定
<a href="#">SAVECONFIG</a>	システム設定の保存
<a href="#">ERASE</a> プログラム番号 <a href="#">ERASE</a> 開始プログラム番号,終了プログラム番号	内部フラッシュメモリ上のプログラム削除
<a href="#">MKDIR</a> “ディレクトリ名”	ディレクトリの作成
<a href="#">RMDIR</a> “ディレクトリ名”	ディレクトリの削除
<a href="#">REMOVE</a> “ファイル名”	ファイルの削除
<a href="#">CAT</a> “ファイル名”	ファイルの内容表示

## 数値関数

<a href="#">ABS</a> (整数)	絶対値の取得
<a href="#">ASC</a> (変数) <a href="#">ASC</a> (変数,文字位置) <a href="#">ASC</a> (文字列) <a href="#">ASC</a> (文字列,文字位置)	変数が参照する文字列の取得/文字列の切り出し 例 A=ASC(“ABCD”,2) S=“ABC”:B=ASC(S,2)
<a href="#">FREE</a> ()	プログラム領域の残りバイト数の取得
<a href="#">INKEY</a> ()	キー入力の読み取り
<a href="#">RND</a> ()	乱数の発生
<a href="#">LEN</a> (変数) <a href="#">LEN</a> (文字列)	文字列長の取得
<a href="#">MAP</a> (値,開始範囲 1,終了範囲 1, 開始範囲 2,終了範囲 2)	数値のスケール変換
<a href="#">RGB</a> (赤, 緑, 青)	3 原色から RGB コード変換

## 文字列関数

<a href="#">CHR\$</a> (文字コード)	画文字コードから文字への変換
<a href="#">BIN\$</a> (数値[,桁指定])	数値から 2 進数文字列への変換
<a href="#">HEX\$</a> (数値[,桁指定])	数値から 16 進数文字列への変換
<a href="#">DMP\$</a> (数値) <a href="#">DMP\$</a> (数値,小数桁数) <a href="#">DMP\$</a> (数値,小数桁数,整数部桁数)	整数の小数付き数値文字列への変換 小数桁数 0~4 整数部桁数 0~8
<a href="#">STR\$</a> (変数) <a href="#">STR\$</a> (変数,先頭,長さ) <a href="#">STR\$</a> (文字列) <a href="#">STR\$</a> (文字列,先頭,長さ)	変数が参照する文字列の取得

## □ 時間待ち・時間計測関連

## コマンド

<a href="#">WAIT</a> ミリ秒	時間待ち
<a href="#">CLT</a>	TICK()の経過時間のリセット

## 数値関数

<a href="#">TICK</a> () <a href="#">TICK</a> ([モード])	起動からの経過時間取得 モード:0 ミリ秒単位、1: 秒単位
---	-----------------------------------

## □ 記憶領域操作関連

## コマンド

<a href="#">POKE</a> アドレス,データ[,データ...データ]	指定アドレスへのデータ書き込み
---	-----------------

## 数値関数

<a href="#">PEEK</a> (アドレス)	指定アドレスの値参照
-----------------------------	------------



## □ キャラクタ表示関連

## コマンド

<a href="#">PRINT</a> [#n.] 数値・文字列[: 数値・文字列…][:]	画面への文字表示 文末:で改行の抑制可能
<a href="#">INPUT</a> 変数	数値の入力
<a href="#">INPUT</a> 変数,オーバーフロー時の既定値	例: INPUT "A=";A,-1
<a href="#">INPUT</a> プロンプト, 変数	
<a href="#">INPUT</a> プロンプト, 変数, オーバーフロー時の既定値	
<a href="#">CLS</a>	画面表示内容の全消去
<a href="#">COLOR</a> 文字色	文字色の設定 ※ターミナル版でのみ利用可能
<a href="#">COLOR</a> 文字色,背景色	
<a href="#">ATTR</a> 属性	文字表示属性設定 ※ターミナル版でのみ利用可能
<a href="#">LOCATE</a> 横位置,縦位置	カーソルの移動
<a href="#">REDRAW</a>	画面表示の再表示
<a href="#">CSCROLL</a> x1,y1,x2,y2,方向	指定領域内での文字スクロール

## 数値関数

<a href="#">VPEEK</a> (横位置,縦位置)	画面指定位置の文字コード参照
---------------------------------	----------------

## □ グラフィック表示関連

## コマンド

<a href="#">PSET</a> x, y, 色	点の描画
<a href="#">LINE</a> x1,y1,x2,y2,色	直線の描画
<a href="#">RECT</a> x1,y1,x2,y2,色,モード	矩形の描画
<a href="#">CIRCLE</a> x, y,半径,色,モード	円の描画
<a href="#">BITMAP</a> x, y, アドレス, インデックス, 幅, 高さ [倍率]	ビットマップ画像の表示
<a href="#">GPRINT</a> X,Y,[#n.] 数値・文字列[:数値・文字列…][:]	指定位置に文字列描画
<a href="#">GSCROLL</a> x1,y1,x2,y2,方向	指定領域内でのグラフィックススクロール
<a href="#">LDBMP</a> "ファイル名",アドレス,bx,by, 幅, 高さ [色指定]	SD カードから BMP ファイルをメモリにロード
<a href="#">DWBMP</a> "ファイル名",x, y, bx, by 幅, 高さ [色指定]	SD カードから BMP ファイルをロードして表示

## 数値関数

<a href="#">GPEEK</a> (横位置,縦位置)	画面上の指定位置ピクセルの参照
<a href="#">GINP</a> (横位置,縦位置,高さ,幅,色)	指定領域のピクセルの有無判定

## □ サウンド関連

## コマンド

<a href="#">TONE</a> 周波数,出力期間	指定周波数のパルス出力
<a href="#">NOTONE</a>	パルス出力の停止

## □ RTC（時刻）関連

## コマンド

<a href="#">DATE</a>	現在時刻の表示
<a href="#">GETDATE</a> 変数 1,変数 2,変数 3,変数 4	日付の取得 変数 1 から順に年,月,日,曜日コードを格納
<a href="#">GETTIME</a> 変数 1,変数 2,変数 3	時刻の取得 変数 1 から順に時,分,秒を格納
<a href="#">SETDATE</a> 年,月,日,時,分,秒	時刻の設定

## □ GPIO・入出力関連

## コマンド

<a href="#">GPIO</a> ピン名 ピン番号,機能名	GPIO 機能設定 ピン名利用時は GPIO 記述省略可能
<a href="#">OUT</a> ピン番号,出力値	デジタル出力
<a href="#">POUT</a> ピン番号, デューティ値	PWM パルス出力
<a href="#">POUT</a> ピン番号, デューティ値,周波数	
<a href="#">SHIFTOUT</a> データピン,クロックピン, 出力形式,出力データ	デジタルシフトアウト出力

## 数値関数

<a href="#">IN</a> (ピン番号)	デジタル入力
<a href="#">ANA</a> (ピン番号)	アナログ入力
<a href="#">SHIFTIN</a> (データピン番号, クロックピン番号, 入力形式)	デジタルシフトイン入力
<a href="#">SHIFTIN</a> (データピン番号, クロックピン番号, 入力形式,条件)	
<a href="#">PULSEIN</a> (パルス入力ピン番号,検出信号,タイムアウト)	入力パルス幅の測定
<a href="#">PULSEIN</a> (パルス入力ピン番号,検出信号,タイムアウト, スケール)	
<a href="#">I2CR</a> (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)	I2C スレーブデバイスからのデータ受信
<a href="#">I2CW</a> (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)	I2C スレーブデバイスへのデータ送信

## □ シリアル通信関連

## コマンド

<a href="#">SMODE</a> モード	GPIO シリアルポート、USB シリアルポート機能切り替え
<a href="#">SMODE</a> モード, “通信速度”	
<a href="#">SOPEN</a> “通信速度”	シリアルポートのオープン
<a href="#">SCLOSE</a>	シリアルポートのクローズ
<a href="#">SPRINT</a> [#n.] 数値・文字列; 数値・文字列;	シリアルポートへの文字列出力
<a href="#">SWRITE</a> データ	シリアルポートからの 1 バイト送信

## 数値関数

<a href="#">SREADY()</a>	シリアル通信 受信データ確認
<a href="#">SREAD()</a>	シリアル通信 1 バイト受信

## □ 仮想 EEPROM 関連

## コマンド

<a href="#">EEPFORMAT</a>	仮想 EEPROM のフォーマット
<a href="#">EEPWRITE</a> アドレス,データ	仮想 EEPROM のへのデータ書き込み

## 数値関数

<a href="#">EEPREAD</a> (アドレス)	仮想 EEPROM のからのデータ読み込み
--------------------------------	-----------------------

システムコマンド、一般コマンドの詳細については「7 各コマンド・関数」を参照して下さい。

## 5.6 数値

## □ 数値

「Tiny BASIC for Arduino STM32」で使える数値は整数型のみとなります。

整数型は 16 ビット幅、有効範囲は-32768~32767 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(-32768~32767) : -1, -32767, 100  
 16 進数表記(1 桁~4 桁) : \$1345, \$abcd, \$Abcd

## (注意)

数値を中間コードに変換(文字列数値を 2 バイト型数値に変換)する際、オーバーフローが発生した場合は overflow エラーとなります。

例①:

```
?32768
Overflow
```

評価・演算においてオーバーフローが発生した場合はエラーとはなりません。

例②:

```
?32767+1
-32768
```

例①、例②は一見、同じような記述ですが、結果に差異が発生することをご理解下さい。

## 5.7 文字・文字列

## □ 文字・文字列

「Tiny BASIC for Arduino STM32」では半角英数字の文字列を扱うことができます。

文字列の表記は次の通りです。

- “文字列” : ダブルクォーテーションで囲みます。  
 文字列は 0~255 文字まで指定可能です。

例:

```
?PRINT "Hello"
Hello
OK
```

## 5.8 変数・配列変数

## □ 変数

変数とは数値を格納する保存領域です。数値と同様に式に利用できます。

変数には、通常の変数の他に**配列変数**があります

変数に格納する数値は 16 ビット幅、有効範囲は-32768～32767 となります。

**変数**は数値型のみ利用可能ですが、文字列の格納アドレスを代入することにより、文字列の参照を行うことが出来ます。

変数の表記

通常の変数：変数名は英字 A～Z の 1 文字、または英字+数字(0～6)の 2 文字の利用が可能です。

例:

```
10 A0=200
20 A1=300
30 A=A0+A1
40 S="Hello"
```

**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:

```
10 A=5
20 @(0)=30/5,
30 @(A+1)=5
```

代入式において、指定した添え字を起点として連続代入が可能

例:

```
10 @(10)=100,200,300,400,500
```

## 5.9 演算子

## □ 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、関数、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 XOR	A^B A と B の XOR

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

比較演算子と論理演算は優先度が同レベルです。比較演算子の演算を先に行う場合は、括弧を使って優先度を上げて下さい。

例：

```
?1<5 and 2>3
0
?(1<5) and (2>3)
1
```

## 5.10 式

## □ 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
```

## 5.11 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

VRAM, VAR, ARRAY, PRG, MEM, FNT, GRAM

データ領域を参照するための定数です。各定数の詳細は次の通りです。

定数名	値	領域サイズ	用途
VRAM	\$0000	(可変最大) 6,400	スクリーン表示用メモリ (CW×CH)
VAR	\$1000	440	変数領域 (A~Z, A0~Z6) + 4 バイト
ARRAY	\$1240	200	配列変数領域 (@ (0) ~ @ (99) )
PRG	\$1340	4,096	プログラム領域
MEM	\$2340	1,024	ユーザーワーク領域
FNT	\$2740	(可変最大) 2,048	フォント 256 文字 (フラッシュメモリ)
GRAM	\$2F40	(可変最大) 6,048	グラフィック表示用メモリ

定数値のアドレスは仮想的なアドレスです。実アドレスとは異なります。

PEEK、POKE、I2CW、I2CR、BITMAP、LDBMP のメモリ領域を利用するコマンドで利用できます。

## □ 画面表示の定数

画面サイズを参照する定数です。

CW : キャラクタ画面の横文字数 (6x8 フォントの場合 37)  
 CH : キャラクタ画面の縦行数 (6x8 フォントの場合 27)  
 GW : グラフィック画面の横ドット数 (224)  
 GH : グラフィック画面の縦ドット数 (216)

## □ 方向を示す定数

CSCROLL, GSCROLL コマンドでスクロール方向を指定に利用出来ます。

UP : 値 0  
DOWN: 値 1  
RIGHT: 値 2  
LEFT : 値 3

## □ ピン番号定数

GPIO、OUT、IN、ANA、SHIFTOUT、SHIFITIN 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。

PA0, PA1, PA2, PA3, PA4, PA5, PA6, PA7, PA8  
PA9, PA10, PA11, PA12, PA13, PA14, PA15  
PB0, PB1, PB2, PB3, PB4, PB5, PB6, PB7, PB8  
PB9, PB10, PB11, PB12, PB13, PB14, PB15  
PC13, PC14, PC15

## □ GPIO モード設定定数

GPIO コマンドのモード設定を行うための定数です。

OUTPUT\_OD, OUTPUT, INPUT\_PU, INPUT\_PD, ANALOG, INPUT\_FL

## □ 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。

SHIFITIN, SHFITOUT コマンドで利用します。

## □ OFF/ON 定数

OFF : 値 0  
ON : 値 1

## 5.12 文字列利用における制約

「Tiny BASIC for Arduino STM32」では制約付きで文字列の利用をサポートしています。

変数への代入も可能です。

例:

```
10 A="ABCDE"
20 PRINT A
RUN
ABCDE
OK
```

変数への文字列代入は、文字列格納アドレスを変数に代入することにより実現しています。

C 言語のポインタによる参照方式と同等です。

ただし、「Tiny BASIC for Arduino STM32」では文字列情報を[長さ+文字列]の形式で管理しているため、文字列の参照を代入した変数の値を変更して利用することは行えません。

下記のような利用は出来ません。行った場合、意図しない動作となります。

例：

```
10 A="ABCDE"
20 A=A+1
30 PRINT A
```

また、コマンドラインでの利用は行えません。文字列参照が正しく行うことが出来ません。

例：

```
A="ABCDE"
OK
?A
```

変数に代入した文字列の操作を行う次の関数を用意しています。

- LEN() 文字列の長さの取得

```
10 A="ABCDE"  
20 L=LEN(A)  
RUN  
OK
```

- STR\$() 変数で参照している文字列の取得、部分切り出し

```
10 A="ABCDE"  
20 PRINT STR$(A)  
30 PRINT STR$(A,4,1)  
RUN  
ABCDE  
D  
OK
```

- ASC() 指定位置の文字コードの取得 :

```
10 A="ABCDE"  
20 C=ASC(A,4,1)  
RUN  
68  
OK
```

## 6. 各制御文の詳細

### 6.1 GOTO 指定行へのジャンプ（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、"ラベル"で指定した行にジャンプします。

行番号には数値、式（変数、関数を含む）が利用可能です。

ラベルを指定した場合、行先頭に同じラベルがある行にジャンプします。

ラベルはダブルクォーテーション( ")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 行にジャンプ、50 行のラベル指定で 20 行にジャンプしています。

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。

行の先頭に無いラベルは、ジャンプ先としての利用は出来ません。

```
10 GOTO "hoge"
20 END
30 ? "Hello": "hoge": ? "test"
run
Undefined line number or label in 10
10 GOTO "hoge"
OK
```

行番号には式や変数の指定が可能ですが、RENUM コマンドによる行番号付け替えに対応出来ない場合があります。

①GOTO N\*100+500

②GOTO 500+N\*100+500

①では RENUM コマンドは何もしません。②は 500 を番号とみなして更新します。

ただし①②とも計算結果が正しい行番号となることは保証出来ません。ご注意下さい。

#### □ エラーメッセージ

Undefined line number or label	: 指定した行、ラベルが存在しない
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

ラベルを使ったループ処理

```
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。  
ラベルはダブルクォーテーション( ")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 "PRN_DATE"
90 PRINT "Edition V0.84 2017/08/01"
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。  
行の先頭に無いラベルは、GOSUB 文の呼び出し先としての参照はされません。

```
10 GOSUB "hoge"
20 END
30 ? "Hello":"hoge":?"test":RETURN
run
Undefined line number or label in 10
10 GOSUB "hoge"
OK
```

行番号には式や変数の指定が可能です。RENUM コマンドによる行番号付け替えに対応出来ない場合があります。

①GOSUB N\*100+500

②GOSUB 500+N\*100+500

①では RENUM コマンドは何もしません。②は 500 を行番号とみなして更新します。  
ただし①②とも計算結果が正しい行番号となることは保証出来ません。ご注意ください。

## □ エラーメッセージ

Undefined line number or label	: 指定した行、ラベルが存在しない
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(=10)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

ラベルを使ったサブルーチンの呼び出し例

```
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	: 指定した行、ラベルが存在しない
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

ラベルを使ったサブルーチンの呼び出し例

```

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、真の場合は実行文 1 を実行します。

真の場合に ELSE がある場合、ELSE 文以降の命令文は実行しません。

偽の場合、ELSE がある場合、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：評価した値が 0 以外である

偽：評価した値が 0 である

真偽判定を行う条件式には定数、数値、関数を含む計算式が記述できます。

式の例：

```
IF A > 5 B=B-1
IF A+1 PRINT A
IF A & 1 TONE 440,200
IF INKEY() = ASC("A") GOTO 100
```

定数の例

```
IF HIGH GOTO 100
```

数値の例

```
IF 123 PRINT "123"
```

関数の例

```
IF RND(1) X=X+1
```

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 IF 文をネスト出来ます。

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 : 条件式が定義されていない

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

### □ 利用例

y を押したら、“Y key” を表示して終了、その他のキーなら“End” を表示して終了する

何も押されていないならば 10 行から 30 行をループする。

```
10 A=INKEY()
20 IF A=ASC("y") PRINT "Y key":END ELSE IF A<>0 PRINT "End":END
30 GOTO 10
```

## 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

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

## □ 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。

例：

```
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.
ok
```

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(利用例を参照)。

FOR~NEXT の繰り返しは、利用している変数が繰り返し条件を満たさなくなった時点で繰り返しを終了します。

最初の例に `35 I=6` を追加し、ループ内で条件を満たさなくすることでループを抜けることが出来ます。

```
10 PRINT "start."
20 FOR I=0 TO 5 STEP 2
30 PRINT I
35 I=6
40 NEXT I
50 PRINT "done."
```

実行結果

```
start
0
done.
ok
```

## □ 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
FOR without variable	: FOR 文で変数を指定していない
FOR without TO	: FOR 文で TO を指定していない
NEXT without counter	: NEXT に対応する FOR 文が無い
FOR too many nested	: FOR 文のネスト数が規定数(=10)を超えた
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。

### □ 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. 各コマンド・関数の詳細

### 7.1 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は [ESC]キーまたは、[CTRL+C]キーにて強制終了することが出来ます。

シリアルコンソール上で[ESC]キーにて中断する場合は、[ESC]キーを2回押す必要があります。

プログラムの実行中は、カーソルが非表示となります。

プログラムの実行中は実行すべき次の行が無い場合、実行を終了します。

また END コマンドにより実行を終了します。

プログラムの実行中にエラーが発生した場合は実行を終了します。

**（注意）** WAIT 命令で長い時間待ちを行っている場合は、時間待ちが終了するまで、[ESC]キー、[CTRL+C]キーによる強制終了を行うことが出来ません。

#### □ 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、引数に数値以外を指定した

#### □ 利用例

##### プログラムを実行する

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

RUN
*****
OK
```

## 7.2 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

~~RENUM 開始行番号,増分,振り直し開始行番号、振り直し終了行番号~~

## □ 引数

開始番号 : 振り直しをする新しい行番号の開始番号 (0 ~ 32767)

増分 : 行番号の増分 (1 ~ 32767)

~~振り直し開始行番号 : 振り直し対象の開始行番号 (0 ~ 32767)~~~~振り直し終了行番号 : 行番号の増分 (0 ~ 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。

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。

開始番号のみ指定した場合、指定した開始番号から 10 間隔で振り直します。

開始番号と増分を指定した場合、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更新されます。

~~行番号 0、1 は振り直しの対象から除外されます。~~~~これは、常に先頭行と利用しプログラム説明等のコメントを記載するためです。~~

(注意) GOTO,GOSUB に存在しない行番号を指定している場合、更新は行われません。  
 また、行番号に計算式を利用している場合、正しい更新が行われない場合があります。  
 GOTO 100+N\*10  
 の記載の場合、先頭の数値 100 を行番号とみなして更新します。

## □ 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  
 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

行番号を 100 から 10 間隔で振り直す

```
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
```

## 7.3 DELETE プログラムの指定行の削除（システムコマンド）

## □ 書式

DELETE 行番号

DELETE 先頭行番号,末尾行番号

## □ 引数

行番号 : 削除対象の行番号 1～32767

先頭番号 : 削除対象範囲の先頭番号 1～32767

末尾番号 : 削除対象範囲の末尾番号 1～32767

## □ 説明

プログラム内の指定した行、指定した範囲（先頭行番号、末尾行番号）の行を削除します。

## □ 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

Illegal value : 先頭番号と末尾番号の指定に矛盾がある

## □ 利用例

プログラム内の 20 行から 50 行の範囲を削除する

```

10 ? "AAAAAA"
20 ? "BBBBBB"
30 ? "CCCCCC"
40 ? "DDDDDD"
50 ? "EEEEEE"
60 ? "FFFFFF"
70 ? "GGGGGG"

DELETE 20,50
OK
LIST
10 ? "AAAAAA"
60 ? "FFFFFF"
70 ? "GGGGGG"
OK

```



## 7.4 CONSOLE コンソール画面切替

## □ 書式

CONSOLE モード

## □ 引数

モード：0 ～ 1 の整数

モード	機能	説明
OFF (0)	デバイスコンソール	コンソール画面をデバイス画面に切り替えます。 デバイス画面はNTSC、OLED ディスプレイ、TFT ディスプレイです。
ON (1)	シリアルコンソール	コンソール画面をシリアルターミナル画面に切り替えます。 USB シリアル、シリアルポート接続のシリアルコンソールが利用可能です。

## □ 説明

コンソール画面の切替を行います。

デバイスコンソールを指定した場合、利用しているデバイス（NTSC、OLED、TFT）がコンソール画面となります。

シリアルコンソールを指定した場合、USB シリアルまたはシリアルポート接続のターミナルコンソール画面となります。

シリアルコンソールでは、シリアル接続したパソコン上のターミナルソフト等でのスクリーン編集を行うことが出来ます。デフォルトでは2つのシリアルポート（USBシリアル、シリアル端子 PA9、PA10）のうち、USBシリアルがスクリーン編集用として機能します。シリアル端子をスクリーン編集用に利用する場合、SMODE コマンドにて切り替えて下さい。

例

```
SMODE 1, "115200"
```

**（注意）**シリアルコンソールでは、デバイスコンソール、PS/2 キーボードは機能しません。  
切り替えにおいてはご注意ください。

シリアルコンソールを利用している状態で SCREEN コマンドを実行した場合、デバイスコンソールに切り替わり、指定したスクリーンモードの設定が反映されます。

シリアルコンソールからデバイスコンソールに切り替えた場合、スクリーンモードはデフォルトの設定となります。

## □ 補足

起動直後のモードについて

デフォルトではモード：0（デバイスコンソール）となります。

ただし、次の設定を行うことでモード：1（シリアルコンソール）に切り替えることが出来ます。

**設定：BOOT1 のジャンパーピンを 1 にセットして起動する**

さらに、上記の設定と併用して、SWCLK (PA14) ピンが HIGH (VCC 接続) の場合、USB シリアルではなくシリアルポート（PA9、PA10）をシリアルコンソール画面として利用することが出来ます。

## □ 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、引数に数値以外を指定した  
Illegal value : 指定した引数の値が範囲外である  
Overflow : 指定した数値が -32767 ～ 32767 を超えている

## □ 利用例

コンソール画面をシリアルコンソールに切り替える

```
CONSOLE ON
```

## 7.5 SCREEN スクリーンモードの設定

## □ 書式

SCREEN モード [画面向き]

## □ 引数

モード : 1 ~ 6 の整数

画面向き : 0 ~ 3 の整数

## □ 説明

デバイスコンソールのスクリーンモードの設定を行います。

設定可能なモード、画面向きは利用するデバイスコンソールにより異なります。

## ①デバイスコンソールがNTSC ビデオ出力画面の場合

## ・モードの設定値

モード	機能	機能
1	NTSC ビデオ画面 モード1 (デフォルト)	画面解像度 224×216ドット テキストスクリーン 37×27文字
2	NTSC ビデオ画面 モード2	画面解像度 224×108ドット テキストスクリーン 37×13文字
3	NTSC ビデオ画面 モード3	画面解像度 112×108ドット テキストスクリーン 18×13文字

## ・画面向き

指定出来ません

## ②デバイスコンソールがOLED ディスプレイの場合

## ・モードの設定値

モード	機能	機能
1	ノーマルモード (デフォルト)	画面(解像度 128×64、 テキストスクリーン 21×8 文字
2	フォント倍角	画面(解像度 128×64、 テキストスクリーン 10×4 文字
3	フォント3倍角	画面(解像度 128×64、 テキストスクリーン 7×2 文字

## ・画面向き

0(デフォルト) ~ 3の指定で90度ずつ回転することが出来ます。

## ③TFT(ILI9341)ディスプレイの場合

## ・モードの設定値

モード	機能	機能
1	ノーマルモード (デフォルト)	画面(解像度 320×240、 テキストスクリーン 53×30 文字
2	フォント倍角	画面(解像度 320×240、 テキストスクリーン 26×15 文字
3	フォント3倍角	画面(解像度 320×240、 テキストスクリーン 17×10 文字
4	フォント4倍角	画面(解像度 320×240、 テキストスクリーン 13×7 文字
5	フォント5倍角	画面(解像度 320×240、 テキストスクリーン 10×6 文字
6	フォント6倍角	画面(解像度 320×240、 テキストスクリーン 8×5 文字

## ・画面向き

0(デフォルト) ~ 3の指定で90度ずつ回転することが出来ます。

## □ 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、引数に数値以外を指定した  
 Illegal value : 指定した引数の値が範囲外である  
 Overflow : 指定した数値が-32767 ~ 32767 を超えている

## □ 利用例

スクリーンモードをモード1に設定する

SCREEN 1

## 7.6 WIDTH シリアルターミナルの画面サイズの設定

## □ 書式

WIDTH 横文字数, 縦文字数

## □ 引数

横文字数：ターミナル画面上のスクリーンの横文字数 16 ～ 128（デフォルト 80）

縦文字数：ターミナル画面上のスクリーンの縦文字数 10 ～ 45（デフォルト 24）

## □ 説明

シリアルターミナル画面上のスクリーンサイズの設定を行います。

この設定にてスクリーンエディタの表示文字数、スクロール範囲の変更を行うことができます。

スクリーンサイズは、最小サイズ 16 桁×10 行 ～ 最大 128 桁×45 行 となります。

**（注意）** デバイスコンソール画面で WIDTH コマンドを実行した場合、設定は無視されます。  
また、シリアルコンソール画面への変更直後はデフォルトの 80×24 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、引数に数値以外を指定した
Illegal value	: 指定した引数の値が範囲外である
Overflow	: 指定した数値が-32768 ～ 32767 を超えている

## □ 利用例

スクリーンサイズを 80 桁×30 行に設定する

```
WIDTH 80,30
```

## 7.7 LIST プログラムリストの表示

## □ 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（0 ～ 32767）

表示終了行番号：表示を終了する行番号（0 ～ 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、その範囲のプログラムリストを表示します。

表示したプログラムリストは、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

```

## 7.8 NEW プログラムの消去

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

## □ 利用例

プログラムを消去する

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

run
*****
OK
?I
10
OK
NEW
OK
LIST
OK
?I
0
OK
```

## 7.9 LOAD 内部フラッシュメモリ・SD カードからプログラムを読み

## □ 書式

LOAD

LOAD プログラム番号

LOAD "ファイル名"

LOAD "ファイル名", モード

## □ 引数

プログラム番号 : 0 ~ 7 内部フラッシュメモリのプログラム番号

"ファイル名" : ディレクトリパス(省略可能)+ファイル 8 文字+拡張子 3 文字 のファイル名 (63 文字まで)

モード : 0:上書きロード (デフォルト)、1:追記ロード

## □ 説明

内部フラッシュメモリ、または外付け SD カードからプログラムを読み込みます。

引数を省略した場合は、内部フラッシュメモリのプログラム番号を読み込みます。

引数にプログラム番号を指定した場合は、指定したプログラム番号を読み込みます。

プログラム番号 0~7 で指定します。読込の際、変数領域は初期化されます。

引数にファイル名を指定した場合、SD カードからプログラムを読み込みます。

ファイル形式は、テキスト形式またはバイナリ形式(SAVE 時にバイナリ形式指定) 対応しています。

ファイル名にはディレクトリパスを付けることが出来ます。

ファイル名には大文字小文字の区別はありません。

例：

```
LOAD "/SAMPLE/TEST.BAS"
LOAD "/PRG1.BAS"
LOAD "PRG1.BAS"
```

SDカードからの読み込みには、引数のモードにて 0:上書きロード、1:追記ロードを指定することが出来ます。

上書きロードの場合、メモリー上のプログラムの初期化後に読み込みます。変数領域は初期化されます。追記モードの場合、メモリー上のプログラムに追記して読み込みを行います。変数領域は初期化されません。

(注意) バイナリーモードで SAVE したプログラムを読み込む場合、メモリー上のプログラムの行番号よりも若い番号が付いている場合、行番号の重複や行並びに不整合が生じます。

## □ 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 0~9 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  
 Bad file name : 指定したファイル名が正しくない  
 SD I/O error : SD カードの利用が出来ない  
 File read error : SD カードからのファイル読み込みに失敗した  
 Overflow : 指定した数値が -32767 ~ 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
```

## 7.10 SAVE 内部フラッシュメモリ・SD カードへのプログラム保存

## □ 書式

SAVE

SAVE プログラム番号

SAVE “ファイル名”

SAVE “ファイル名”,保存形式

## □ 引数

プログラム番号 : 0 ~ 7

“ファイル名” : ディレクトリパス(省略可能)+ファイル 8 文字+拡張子 3 文字 のファイル名 (63 文字まで)

保存形式 : 1 テキスト形式 (デフォルト) 、0 バイナリ形式

## □ 説明

プログラムをマイコン内のフラッシュメモリ、または SD カードに保存します。

引数の省略、数値を指定した場合は内部フラッシュメモリに保存します。

プログラムは最大で 8 つ保存可能です。保存先はプログラム番号 0~7 で指定します。

引数の省略した場合、プログラム番号 0 に保存します。

ファイル名を指定した場合は、SD カードにプログラムを保存します。

保存形式の指定により、テキスト形式またはバイナリ形式にて保存が行えます。

保存形式を指定しない場合は、テキスト形式で保存されます。

バイナリ形式での保存はプログラム領域 4k バイトの全てを保存します。

(注意) NTSC 表示版では SAVE コマンド実行時に画面表示が乱れる場合があります。

## □ 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 0~7 の範囲外である
Bad file name	: 指定したファイル名が正しくない
SD I/O error	: SD カードの利用が出来ない
File write error	: SD カードへのファイルの書込みに失敗した
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
```

## 7.11 FILES 内部フラッシュメモリ保存・SD カード内のプログラムの一覧表示

## □ 書式

FILES

FILES 開始プログラム番号

FILES 開始プログラム番号,終了プログラム番号

FILES "ファイルパス"

## □ 引数

開始プログラム番号 : 表示開始プログラム番号 0~7

終了プログラム番号 : 表示終了プログラム番号 0~7

"ファイルパス" : SD カード内のディレクトリまたはファイル名(ワイルドカード指定可能)

## □ 説明

マイコン内のフラッシュメモリに保存されているプログラムまたは、SD カード内のファイルの一覧を表示します。

引数を指定しない場合は、内部フラッシュメモリ内のプログラム番号 0~7 の先頭行をリスト表示します。

開始プログラム番号,終了プログラム番号を指定した場合、その範囲のリストを表示します。

プログラム番号にプログラムが保存されていない場合は(none)と表示されます。

プログラム先頭行にコメントをつけると、一覧表示でのプログラムの内容が分かり易くなります。

例:

```
files
0:'Edit bitmap
1:'RTC TEST
2:(none)
3:(none)
4:(none)
5:(none)
6:(none)
7:1'LED Blink
OK
```

引数にファイルパスを指定した場合、SD カード内のファイルを一覧表示します。

ファイル名は 8.3 形式となります。

ファイルパスには、ディレクトリ名、ディレクトリ名+ファイル名(ワイルドカード指定可能)の記述が可能です。ワイルドカードは、\*と?が利用可能です。ディレクトリ名指定の上位、下位の"/"は省略可能です。""と"/"は同じ出力となります。

```
FILES ""
FILES "/"
FILES "/*"
FILES "/SRC/"
FILES "/SRC"
FILES "/SRC/SAMPLE/"
FILES "SRC/"
FILES "SRC/*.BAS"
FILES "SRC/*.B??"
```

出力されるファイル一覧の順番は不定となります。ディレクトリファイルの場合は、ファイル名の最後に\*が付きます。

```
files "*.bas"
TEST.BAS      TIME.BAS
TM.BAS        SOKUDO.BAS
トイ.BAS      トイ2.BAS
TEST2.BAS     TESTBMP.BAS
AA.BAS        1.BAS
2.BAS         3.BAS
サッチャン.BAS SUB.BAS
ケイワ.BAS    DAT*
```

(注意) 豊四季 Tiny BASIC (ファームウェア) の新規利用または更新を行った直後は、フラッシュメモリ上の既存データの内容の不整合により正しく表示できない場合があります。その場合は、ERASE コマンドにてフラッシュメモリ上のプログラムの消去を行って下さい。



## □ 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	: 指定した開始プログラム番号,終了プログラム番号の値が正しくない。
Bad file name	: 指定したファイル名が正しくない
SD I/O error	: SD カードの利用が出来ない
File read error	: SD カードからのファイル読み込みに失敗した
Overflow	: 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

シリアルコンソール画面で SD カード内のファイル一覧を表示する。

(補足) 表示する横の列数は、スクリーンの横のサイズに応じて調整されます。

```

COM9:115200baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
OK
files ""
TEST.BAS    SYSTEM~1*    FONT.BIN    FOUND.000*   TIME.BAS
TM.BAS      SOKUDO.BAS    トゲイ.BAS  IMG.BMP      トゲイ2.BAS
FONTLCD.BIN IMG2.BMP      IMG3.BMP    IMG4.BMP     IMG5.BMP
IMG6.BMP    LOGO1.BMP     LOGO2.BMP   TEST2.BAS    TESTBMP.BAS
AA.BAS      DIR1*         VRAM.BIN    A.BIN        B.BIN
1.BAS       2.BAS        3.BAS      サッチャン.BAS SUB.BAS
ケイック.BAS SAMPLE*       TT.BAS      TEST*        MEM.BIN
SS1306.BAS  DS3231.BAS   OLED.BAS    OLED2.BAS    RR.BAS
LOGO24.BMP  LOGO25.BMP   OLED30.BAS  CAT*         CAT.JPG
CAT2.JPG    INDEX.HTM    NEXT.HTM    CAT.BMP      TT.BMP
LOGO.BMP    LOGO3.BMP    OLED00*.BAS TEST.TXT     TEST1.BAS
TEST3.BAS   TEST4.BAS    メモリ-MAP.BAS CHAROUT.BAS  HAROUT.BAS
T1.BAS      T2.BIN       T1.BIN
OK

```

## 7.12 BLOAD SD カードからバイナリデータ読込

## □ 書式

BLOAD "ファイル名",アドレス ,バイト数

## □ 引数

"ファイル名" : ディレクトリパス(省略可能)+ファイル 8 文字+拡張子 3 文字 のファイル名

アドレス : 読み込みデータ格納アドレス \$0000 ~

バイト数 : 読み込みデータサイズ 1~32767

## □ 説明

SD カードから指定したファイルの内容を指定したアドレスに指定バイト数分、読み込みます。

アドレスには仮想アドレスを有効範囲内で指定します。

仮想アドレスについては「1.11 メモリーマップ」の仮想アドレスを参照下さい。

BSAVE で保存したユーザーワーク領域、変数領域等内容を読み込んで利用することが出来ます。

ファイル名には、PRINT 文と同じ書式の記述が可能です。

例:

```
10 N=1
20 BLOAD #-2,"DATA";N;".BIN",MEM,1024
OK
```

上記の例では、ファイル "DATA01.BIN" をユーザー作業領域に読み込みます。

(注意) 読み込むファイルは BSAVE コマンドで保存したファイルに限られます。オリジナルのファイルを読み込む場合は、先頭に 2 バイト x00,x00 を付加して下さい。

(注意) プログラム領域への読み込みを行った場合の動作がおかしくなる場合があります。

## □ 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	: 指定した引数が有効範囲を超えている
Bad file name	: 指定したファイル名が正しくない
SD I/O error	: SD カードの利用が出来ない
File read error	: SD カードからのファイル読み込みに失敗した
Overflow	: 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

画面表示内容（グラフィック表示）をファイルに保存し、そのファイルを読み込み再表示を行う

```
BSAVE "GDATA.BIN",GRAM,6048
OK
CLS
OK
BLOAD "GDATA.BIN",GRAM,6048
OK
```

## 7.13 BSAVE SD カードへのバイナリデータ保存

## □ 書式

BSAVE “ファイル名”,アドレス ,バイト数

## □ 引数

“ファイル名” : ディレクトリパス(省略可能)+ファイル 8 文字+拡張子 3 文字 のファイル名

アドレス : 保存データ格納アドレス \$0000~

バイト数 : 保存データサイズ 1~32767

## □ 説明

指定アドレスに格納されいてるバイナリデータを SD カードに保存します。

アドレスには仮想アドレスを有効範囲内で指定します。

仮想アドレスについては「1.11 メモリーマップ」の仮想アドレスを参照下さい。

ファイル名には、PRINT 文と同じ書式の記述が可能です。

例:

```
10 N=1
20 BSAVE #-2,"DATA";N;".BIN",MEM,1024
OK
```

上記の例では、ユーザー作業領域をファイル名 “DATA01.BIN”に保存します。

(注意) 保存データの先頭に 2 バイト x00,x00 が付加されます。

## □ 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
Illegal value	: 指定した引数が有効範囲を超えている
Bad file name	: 指定したファイル名が正しくない
SD I/O error	: SD カードの利用が出来ない
File write error	: SD カードへのファイル書き込みに失敗した。
Overflow	: 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

グラフィック表示内容の保存と読み込み

```
BSAVE "GDATA.BIN",GRAM,6048
OK
CLS
OK
BLOAD "GDATA.BIN",GRAM,6048
OK
```

## 7.14 REM コメント

## □ 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。

‘(シングルクォート)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時に、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

コメントの記述例

```
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
```

## 7.15 LET 変数に値を代入

## □ 書式

LET 変数=値

LET 配列変数=値,値,値,値...

変数=値

配列変数=値,値,値,値...

## □ 引数

変数 : 変数、または配列変数

配列変数 : 配列変数 @(値)

配列の値(添え字)には式、数値、変数、配列変数、数値定数の利用が可能

値 : 式、数値、変数、配列変数、数値定数

## □ 説明

変数に値を代入します。

値は式、数値、定数、変数、配列変数等を評価した整数値です。

配列変数への代入は、複数の値を指定した連続代入が可能です。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 'Print starts
10 LET I=0:'initialize value
20 PRINT "*";
30 LET I=I+1:IF I<10 GOTO 20
40 PRINT
50 END

```

## 7.16 CLV 変数領域の初期化

## □ 書式

CLV

## □ 引数

なし

## □ 説明

変数領域（変数、配列）の初期化を行います。

変数 A～Z、配列@(0)～@(99)はすべて 0 に初期化されます。

LRUN コマンドにてプログラムをロードして実行する場合、変数領域は初期化されません。

これにより呼び出し元のプログラムの変数をひ引き継ぐことが出来ます。

呼び出されたプログラムにて明示的に変数領域の初期化を行う場合は、CLV コマンドを使って下さい。

## □ 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

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

## □ 利用例

```
a=123
OK
?a
123
OK
CLV
OK
?a
OK
```

## 7.17 LRUN 指定プログラム番号の実行

## □ 書式

LRUN プログラム番号  
 LRUN プログラム番号,行番号  
 LRUN プログラム番号,"ラベル"  
 LRUN "ファイル名"  
 LRUN "ファイル名",行番号  
 LRUN "ファイル名","ラベル"

## □ 引数

プログラム番号 : 実行するプログラムの番号 0 ~ 7  
 "ファイル名" : ディレクトリパス(省略可能)+ファイル 8 文字+拡張子 3 文字 のファイル名  
 行番号 : プログラムの実行開始行番号 1~32767  
 "ラベル" : プログラムの実行開始行のラベル指定

## □ 説明

プログラム番号またはファイル名で指定したプログラムを実行します。

第 1 引数に数値を指定した場合は、内部フラッシュメモリから指定したプログラム番号のプログラムを読み込んで実行します。ファイル名を指定した場合は、SD カードから指定ファイルを読み込んで実行します。

第 2 引数に行番号またはラベルの指定がある場合は、その行からプログラムの実行を開始します。

ファイル名の指定には、PRINT 文と同等の記述が可能です。

変数 N の値でファイル名を作成してプログラムを実行する。

```
10 INPUT "PROG NO=",N,0
20 IF N=0 OR N>9 GOTO 10
20 LRUN #-2,"PRG";"N;".BAS"
```

上記の例では、N の値が 1~9 の場合には 対応する"PRG01.BAS"~"PRG09.BAS"をロードして実行します。

(注意) LRUN にてプログラムを実行した場合、変数領域の初期化は行われません。直前のプログラムが設定した変数の値を引き継ぎます。初期化が必要な場合は呼び出されたプログラム中で CLV コマンドを実行して下さい。

## □ 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  
 Illegal value : プログラム番号の指定が 0~7 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  
 Bad file name : 指定したファイル名が正しくない  
 SD I/O error : SD カードの利用が出来ない  
 File read error : SD カードからのファイル読み込みに失敗した  
 Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

入力したプログラム番号を実行する。

```
10 INPUT N
20 LRUN N
```

## 7.18 EXPORT 内部フラッシュメモリの内容のエクスポート

## □ 書式

EXPORT

EXPORT 対象番号

EXPORT 開始番号,終了番号

## □ 引数

対象番号 : エクスポートするプログラム保存番号 0 ~ 7

開始番号,終了番号 : エクスポートするプログラム保存番号の範囲 0 ~ 7

## □ 説明

マイコン内のフラッシュメモリに保存されているプログラム(プログラム番号0~7)を出力表示します。

ターミナルソフト等を利用することにより、出力表示されたプログラムリストをコピーすることで、内部保存されたプログラムのバックアップを行うことができます。

引数に何も指定しない場合は、0 ~ 7 までの全てのプログラムを画面に出力します。

対象番号を指定した場合は、該当するプログラム番号の内容のみ出力します。

開始番号と終了番号を指定した場合はその範囲のプログラムを出力します。

出力形式は次の形式となります。

```
NEW
10 'oscilloscope
20 CLS
30 GPIO PB1,ANALOG
40 "LOOP"
50 R=ANA(PB1)
60 Q=R/20
70 LOCATE 0,0:?"R;" "
80 PSET 223,221-Q,1
90 WAIT 50
100 GSCROLL 0,8,GW-1,GH-1,LEFT
110 GOTO "LOOP"
SAVE 1
```

プログラム番号毎のプログラムリストの先頭に“NEW”コマンド、末尾に“SAVE プログラム番号”が付加されます。

ターミナルソフトにペーストすることにより、フラッシュメモリ内に再登録することができます。

## □ 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 : 指定した引数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

プログラム番号 1,2 に保存されているプログラムのエクスポート

```
EXPORT 1,2
NEW
10 'oscilloscope
20 CLS
30 GPIO PB01,ANALOG
40 "LOOP"
50 R=ANA(PB01)
60 Q=R/20
70 LOCATE 0,0:?" "
80 PSET GW-1,GH-Q-4,1
90 WAIT 50
100 GSCROLL 8,208,3
110 GOTO "LOOP"
SAVE 1

NEW
10 CLS
20 RECT 0,0,GW,GH,1,1
30 RESETTICK
40 FOR I=0 TO 215
50 FOR X=0 TO 200 STEP 16
60 GSCROLL X,0,8,GH,0
70 GSCROLL X+8,0,8,GH,1
80 NEXT X
100 NEXT I
110 ?TICK()
120 GOTO 120
SAVE 2

OK
```

## 7.19 CONFIG システムの設定

## □ 書式

CONFIG 項目番号, 設定値

## □ 引数

項目番号: 0 ~ 2

設定値: 選択した項目の設定値 (項目により設定値の条件は異なる)

## □ 説明

「豊四季タイニーBASIC for Arduino STM32」の環境設定を行います。

項目番号	設定名称	設定値	説明
0	NTSC 垂直同期信号補正	0,1,2	NTSC の垂直同期信号の補正値を設定します。 デフォルトは 0 です。 利用するモニターの表示がスクロールする場合、映像が乱れる場合は、この値を設定して調整してください。
1	キーボード設定	0,1	利用するキーボードのレイアウト設定を行います。 <b>0: 日本語キーボード(デフォルト)</b> <b>1: US キーボード</b>
2	自動起動プログラム設定	-1 0~7	電源 ON 後に自動起動するプログラムを指定します。 0~7 はフラッシュメモリに保存しているプログラム番号を指定します。-1 の設定で自動起動無しの設定となります。

設定はコマンド実行後、直ちに反映されます。本設定は電源を落とすまで有効です。

次回以降の起動にも適用したい場合は、SAVECONFIG コマンドで保存して下さい。

初めてボードに書き込みを行う場合、SAVECONFIG コマンドで書き込めない場合があります。

その場合は、EEPFORMAT コマンドにて保存領域の初期化を行って下さい。

## □ 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: 指定した項目番号、設定値が正しくない。

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

## □ 利用例

利用しているキーボードを US キーボードに設定する。

```
CONFIG 1,1
OK
SAVECONFIG
OK
```

## 7.20 SAVECONFIG システムの設定の保存

## □ 書式

SAVECONFIG

## □ 引数

なし

## □ 説明

CONFIG コマンドの設定変更をフラッシュメモリに保存します。次回以降の起動でも設定を有効にします。  
フラッシュメモリの利用状況により、保存に失敗する場合があります。  
その場合は、フラッシュメモリの初期化を EEPFORMAT にて行って下さい。

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

## □ 利用例

利用しているキーボードを us キーボードに設定する。

```
CONFIG 1,1
OK
SAVECONFIG
OK
```

## 7.21 ERASE 内部フラッシュメモリ上のプログラム削除

## □ 書式

ERASE プログラム番号

ERASE 開始プログラム番号,終了プログラム番号

## □ 引数

プログラム番号 : 0 ~ 7

開始プログラム番号 : 0 ~ 7

終了プログラム番号 : 0 ~ 7

## □ 説明

マイコン内のフラッシュメモリに保存されているプログラム(プログラム番号 0~7)の削除を行います。

削除対象プログラムは、コマンド引数にてプログラム番号を指定するか、

削除するプログラム番号の範囲（開始プログラム番号、終了プログラム番号）にて指定します。

(注意) 豊四季 Tiny BASIC のスケッチを書き込み後、FILES コマンドにてプログラム一覧を表示した場合、意味不明の文字列が大量に表示される場合があります。  
この場合は ERASE コマンドを実行して、プログラム保存領域を初期化して下さい。

## □ 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 0~7 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

フラッシュメモリに保存されている全てのプログラムの削除を行う。

```
ERASE 0,9
OK
FILES
0: (none)
1: (none)
2: (none)
3: (none)
4: (none)
5: (none)
6: (none)
7: (none)
OK
```

## 7.22 MKDIR ディレクトリの作成

## □ 書式

MKDIR "ディレクトリ名"

## □ 引数

“ディレクトリ名”： 1～63 文字までのディレクトリ名

## □ 説明

SD カードにディレクトリを作成します。

指定するディレクトリ名は、上位ディレクトリ名を含めて最大 63 文字まで指定可能です。

作成するディレクトリの名称は 8 文字可能です。大文字小文字の区別はありません。

上位のディレクトリ名は"/"です（名前指定において省略可能）。

次の2つは同じ動作をします。

MKDIR "/DIR1"

MKDIR "DIR1"

サブディレクトリの作成は次のように記述します。

MKDIR "/DIR1/SUBDIR"

または

MKDIR "DIR1/SUBDIR"

上位のディレクトリがない場合にサブディレクトリ作成を行った場合、親ディレクトリも作成されます。

次のコマンドでは、DIR2 と DIR2/SUBDIR が作成されます。

MKDIR "DIR2/SUBDIR"

## □ 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

Bad file name : 指定したファイル名が正しくない

SD I/O error : SD カードの利用が出来ない

File write error : SD カードへのファイル書き込みに失敗した。

## □ 利用例

ディレクトリ SRC を作成する

```
MKDIR "SRC"
OK
```

## 7.23 RMDIR ディレクトリの削除

## □ 書式

RMDIR "ディレクトリ名"

## □ 引数

"ディレクトリ名": 1～63 文字までのディレクトリ名

## □ 説明

SD カード内の指定したディレクトリを削除します。

指定したディレクトリにサブディレクトリやファイルが存在する場合、そのファイルも削除されます。

指定するディレクトリ名は、上位ディレクトリ名を含めて最大 63 文字まで指定可能です。

作成するディレクトリの名称は 8 文字可能です。大文字小文字の区別はありません。

上位のディレクトリ名は"/"です（名前指定において省略可能）。

次の2つは同じ動作をします。

```
RMDIR "/"DIR1"
```

```
RMDIR "DIR1"
```

サブディレクトリの削除は次のように記述します。

```
RMDIR "/"DIR1/SUBDIR"
```

または

```
RMDIR "DIR1/SUBDIR"
```

（注意）ルートディレクトリの削除は出来ません。

RMDIR "/" は "Bad file name" 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
Bad file name	: 指定したファイル名が正しくない
SD I/O error	: SD カードの利用が出来ない
File write error	: SD カードへのファイル書き込みに失敗した。

## □ 利用例

ディレクトリ SRC を削除する

```
RMDIR "SRC"
OK
```

## 7.24 REMOVE ファイルの削除

## □ 書式

REMOVE "ファイル名"

## □ 引数

"ファイル名" : ディレクトリパス(省略可能)+ファイル 8 文字+拡張子 3 文字 のファイル名 (63 文字まで)

## □ 説明

SD カード内の指定したファイルを削除します。

ファイル名にはディレクトリ名+ファイル名の形式での記述が可能です。

上位のディレクトリ名は"/"です (名前指定において省略可能)。

次の2つは同じ動作をします。

```
REMOVE "/SAMPLE.BAS"
```

```
REMOVE "SAMPLE.BAS"
```

ディレクトリ内のファイルを削除する場合はそのディレクトリ名も指定します。

```
REMOVE "/SRC/SAMPLE.BAS"
```

または

```
REMOVE "SRC/SAMPLE.BAS"
```

## □ 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
Bad file name	: 指定したファイル名が正しくない
SD I/O error	: SD カードの利用が出来ない
File write error	: SD カードへのファイル書き込みに失敗した。

## □ 利用例

ファイルを削除する

```
REMOVE "/SRC/SAMPLE.BAS"
OK
```

## 7.25 CAT ファイルの内容表示

## □ 書式

CAT "ファイル名"

## □ 引数

"ファイル名" : ディレクトリパス(省略可能)+ファイル 8 文字+拡張子 3 文字 のファイル名 (63 文字まで)

## □ 説明

SD カード内の指定したテキストファイルの内容を画面に表示します。

表示テキストには先頭にコメントコマンド(!)が付加されます。

ファイル名にはディレクトリ名+ファイル名の形式での記述が可能です。

上位のディレクトリ名は"/"です (名前指定において省略可能)。

次の2つは同じ動作をします。

CAT "/SAMPLE.BAS"

CAT "SAMPLE.BAS"

ディレクトリ内のファイルを表示する場合はそのディレクトリ名も指定します。

CAT "/SRC/SAMPLE.BAS"

または

CAT "/SRC/SAMPLE.BAS"

(注意) バイナリ形式のファイルには対応していません。

## □ 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
Bad file name	: 指定したファイル名が正しくない
SD I/O error	: SD カードの利用が出来ない
File read error	: SD カードからのファイル読み込みに失敗した

## □ 利用例

ファイルの内容を表示する

```
CAT "/SRC/SAMPLE.BAS"
!10 PRINT "Hello Tiny BASIC"
!20 PRINT "Hello Arduino STM32"
!30 END
OK
```



## 7.26 ABS 絶対値の取得（数値関数）

## □ 書式

ABS(値)

## □ 引数

値 : -32767 ~ 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。

（注意）-32768 の絶対値 32768 はオーバーフローとなるため、-32768 を指定した場合はオーバーフローエラーとなります。

## □ 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  
 Overflow : 指定した数値が-32767 ~ 32767 を超えている  
 '(' or ')' expected : '(' または ')' が無い

## □ 利用例

変数の値の絶対値を表示する

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

RUN

3

2

1

0

2

3

OK

## 7.27 ASC 文字から文字コードへの変換（数値関数）

## □ 書式

ASC(文字列)

ASC(文字列,文字位置)

ASC(変数)

ASC(変数,文字位置)

## □ 引数

文字列： “文字列”  
 “ABC” の形式とし、ダブルクォーテーション文字を囲みます

文字位置： 1～32767  
 変換対象となる左からの文字位置を指定します。

変数： 文字列参照している変数または配列変数

## □ 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。

文字列のみを指定した場合、先頭の文字コードを返します。

```
10 ?ASC("ABCD")
RUN
65
OK
```

文字位置を指定した場合、指定した位置の文字コードを返します。

```
10 ?ASC("ABCD",3)
RUN
67
OK
```

変数を指定した場合、変数が参照している文字列のコードを返します。

```
10 A="ABCDEF"
20 ?ASC(A)
30 ?ASC(A,3)
RUN
65
67
OK
```

**（注意）** 変数の文字列参照はプログラム中にのみ有効です。コマンドラインでは文字列参照を正しく行うことが出来ません。

## □ 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

Illegal value : 文字指定位置が不当

'(' 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

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
```

## 7.28 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 ～ 4095

## □ 説明

プログラム領域の残りバイト数を返します。

作成したプログラムサイズを確認する場合は、下記の記述にて行うことが可能です。

```
PRINT 4095-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

'(' or ')' expected : 引数部の記述が正しくない

## □ 利用例

プログラムサイズを調べる

```
PRINT 4095-FREE()
287
OK
```

## 7.29 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 0 ～ 255

キーが押されていない場合は 0 を返します

## □ 説明

キーボード上の押しているキーの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。

押したキーが返すコードについては「1.13 キー入力コード」を参照下さい。

**（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。  
 キーボードにおいて利用出来ないキーはコードの取得が出来ません。

## □ 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

## □ 利用例

入力したキーのコードを調べる

```
10 CLS
20 I= INKEY()
30 IF I>0 LOCATE 0,0:?"#3,I
40 GOTO 20
```

## 7.30 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

0 から指定した値-1 の範囲の乱数を発生させ、その値を返します。

R=RND(10)

の場合、変数 R には 0~9 範囲の値が代入されます。

## □ 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  
 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 @(0)="グー","チョキ","ハー"
20 FOR I=1 TO 5
30 R=RND(3)
40 PRINT I,":";STR$(@(R))
50 NEXT I
run
1:チョキ
2:ハー
3:チョキ
4:グー
5:ハー
OK

```

## 7.31 LEN 文字列の長さの取得（数値関数）

## □ 書式

LEN(変数)

LEN(文字列)

## □ 引数

変数 : 変数、または配列変数

文字列 : “文字列”の形式（ダブルクォーテーションで囲み）

## □ 戻り値

文字列の長さ 0 ～ 32767

## □ 説明

文字列定数、変数で参照してる文字列の文字数をカウントし、その値を返します。

```

10 ?LEN("12345678")
20 A="ABCDEF"
30 @(0)="abcdef"
40 ?LEN(A)
40?LEN(@(0))
RUN
8
6
6
OK

```

（注意）「豊四季タイニーBASIC for Arduino」の文字列のサポートは限定的です。

他の文字列関数と組み合わせた利用はサポートしていません。

例) 下記のような記述は出来ません。

```
?LEN(BIN$(100))
```

## □ 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

文字を拡大表示する

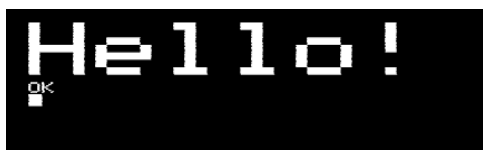
変数 A が参照している文字列の長さ LEN()関数で取得し、その長の分、文字を拡大表示します。

```

10 CLS
20 A="Hello!"
30 FOR I=1 TO LEN(A)
40 BITMAP (I-1)*30,0,FNT,ASC(A,I),6,8,5
50 NEXT I
60 LOCATE 0,5

```

実行結果



## 7.32 MAP 数値のスケール変換（数値関数）

## □ 書式

MAP(値,範囲下限, 範囲上限, 新範囲下限, 新範囲上限)

## □ 引数

値 : 変換対象の数値 -32768 ~ 32767  
 範囲下限 : 対象数値の数値の下限 -32768 ~ 32767  
 範囲上限 : 対象数値の数値の上限 -32768 ~ 32767  
 新範囲下限 : 新しい対象数値の数値の下限 -32768 ~ 32767  
 新範囲上限 : 新しい新しい対象数値の数値の下限 -32768 ~ 32767

## □ 戻り値

スケール変換後の値 -32767 ~ 32767

## □ 説明

指定した値のスケール変換を行い、その値を返します。

例：

```
A=MAP(ANA(PA07),0,4095,0,99)
```

上記の例ではアナログ入力値 0~4095 を 0 ~ 99 のレンジに変換し、その値を返します。

アナログ入力や画面座標指定、PWM パルス出力等にて MAP 関数を使うことで、簡単にレンジ変換を行うことができます。

## □ 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

アナログ入力値 0~4095 を 0~3.3V で表示する

```
10 CLS
20 GPIO PB1,ANALOG
30 V=MAP(ANA(PB1),0,4095,0,3300)
40 LOCATE 0,0: ? DMP$(V,3)
50 WAIT 200
60 GOTO 30
```



## 7.33 RGB 3原色からRGBコード変換（数値関数）

## □ 書式

RGB(赤, 緑, 青)

## □ 引数

赤 : 赤色のレベル 0 ~ 31 (5ビット)

緑 : 緑色のレベル 0 ~ 31 (5ビット)

青 : 青色のレベル 0 ~ 31 (5ビット)

## □ 戻り値

RGBコード (16ビット)

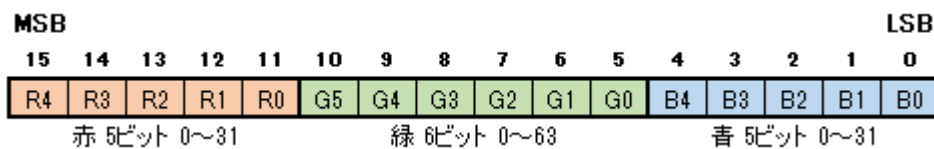
## □ 説明

赤、緑、青の各色成分のレベルを合成した16ビットのRGBコードを返します。

本関数はカラー表示可能なTFTディスプレイの色指定を補助する関数です。

TFT(ILI9341)ディスプレイでは16ビットの色指定が可能です。

RGBコードは次の構成となります。



TFT(ILI9341)ディスプレイではRGBコードとして \$0009 ~ \$FFFF の色指定が可能です。

\$0000~\$0008 は、次の色に割り当てています。

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

RGBコードは、緑のみ6ビット(0~63)です。

本関数では、他の色を同じように色ビットの重みを同じ扱いにするために緑も0~31を指定範囲とし、

内部にて値を2倍にしています。この場合、0~62までの値しか取れないため、緑のみ32の値の指定が可能とし、32を指定した場合、内部処理にて63を設定するものとします。

## □ 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 : 指定した数値が有効範囲を超えている

'(' or ')' expected : '(' または ')' が無い

## □ 利用例

TFT ディスプレイ画面全体を、黒から白に段階的に変化させる

```
10 CLS
20 FOR C=0 TO 31
30 RECT 0,0,GW-1,GH-1,RGB(C,C,C),1
40 NEXT C
```

## 7.34 CHR\$ 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： 0～255

## □ 戻り値

指定した文字コードに対する文字

## □ 説明

指定した文字コードに対応する文字を返します。

PRINT、SPRINT、GPRINT の引数にて利用可能です。

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
Illegal value	: 文字コードに 0～255 以外の数値を指定した
'(' 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

```

## 7.35 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。

数値には式、変数、定数等の指定が可能です。

PRINT、SPRINT、GPRINT 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。

桁数を指定しない場合は、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

```

## 7.36 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( -32767 ~ 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。

数値には式、変数、定数等の指定が可能です。

PRINT、SPRINT、GPRINT 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

入力した整数値を 16 進数表示する

```
10 INPUT "value=",V
20 PRINT HEX$(V,4)
30 GOTO 10
```

## 7.37 DMP\$ 整数の小数付き数値文字列への変換（文字列関数）

## □ 書式

DMP\$(数値)

DMP\$(数値,小数点桁数)

DMP\$(数値,小数点桁数,整数部桁数)

## □ 引数

数値 : 変換対象整数 -32767 ~ 32767

小数点桁数 : 小数点以下の桁数を指定 0 ~ 4（省略時は 2）

整数部桁数 : 小数点以上の桁数を指定 0 ~ 8（省略時は 0）

## □ 戻り値

小数点を付加した数値文字列

## □ 説明

指定した数値を小数点付きの文字列数値に変換します。

引数の小数点以下の桁数を  $n$  とした場合、数値  $\div 10^n$  の計算を行い、小数点以下の数値を含めて表示します。

```
PRINT DMP$(3141,3)
3.141
```

小数点以下の桁数を指定しない場合、小数点桁数は 2 となります。

```
PRINT DMP$(3141)
3.14
```

整数部桁数を指定した場合、桁数に満たない場合は空白で補完します。

本関数は PRINT、SPRINT、GPRINT の引数にて利用可能です。

## □ 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	: 指定した引数の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
Illegal value	: 桁数の値が正しくない
'(' or ')' expected	: '(' または ')' が無い

## □ 利用例

アナログ入力値 0~4095 を 0~3.3V で表示する

```
10 CLS
20 GPIO PB1,ANALOG
30 V=MAP(ANA(PB1),0,4095,0,3300)
40 LOCATE 0,0:PRINT DMP$(V,3)
50 WAIT 200
60 GOTO 30
```

実行結果

```
run
2.011
2.215
2.294
2.330
```

アナログ入力値 0~4095 を MAP 関数にて 0~3300 に変換し、

0~3300 の数値を DMP\$() 関数にて "0.000" ~ "3.300" の文字列数値に変換して表示しています。

## 7.38 STR\$ 変数が参照する文字列の取得・文字列の切り出し

## □ 書式

STR\$(変数)

STR\$(変数,先頭位置,長さ)

STR\$(文字列)

STR\$(文字列,先頭位置,長さ)

## □ 引数

変数 : 文字列を参照している変数または配列変数

先頭位置 : 切り出す文字位置先頭 1~32767

長さ : 切り出す文字の長さ 1~32767

文字列 : ダブルクォーテーションで囲った文字列定数(“文字列”)

## □ 戻り値

指定した文字列および変数が参照している文字列を全部または一部を切り出して返します。

## □ 説明

指定した文字列および変数が参照している文字列を全部または一部を切り出して返します。

引数に先頭位置、長さを指定した場合は、その条件にて文字列を切り出して返します。

先頭位置は 1 からの指定となります。

PRINT、SPRINT、GPRINT の引数にて利用可能です。

例:

```
10 S="123456789"
20 ?STR$(S)
30 ?STR$(S,5,2)
40 ?STR$("ABCDE",5,1)
```

実行結果

```
run
123456789
56
E
OK
```

上記の例では、変数 S が参照している文字列“123456789”に対して、  
 20 行は全て出力、30 行は 5 番目からの文字から 2 文字 “56”を出力、  
 40 行は直接指定した文字列“ABCDE”の 5 番目の文字から 1 文字を出力しています。

## □ 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 : 指定した文字位置、長さの値が不当

'(' 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

変数が参照している文字列を切り出して表示する

```
10 'モリツツサ サンプル
20 S="Hello,Tiny BASIC"
30 L=LEN(S)
40 PRINT STR$(S);" LEN=";L
50 PRINT STR$(S,1,5)
60 C=ASC(S,12)
RUN
Hello,Tiny BASIC LEN=16
Hello
```

## 7.39 WAIT 時間待ち

## □ 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
```



## 7.40 CLT 起動からの経過時間カウンタのリセット

## □ 書式

CLT

## □ 引数

なし

## □ 説明

TICK()の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タをリセットすることで0からのカウントを行うことができます。

## □ 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

1 万回ループの実行時間を測定する（プログラム：TICK.BAS）

```
10 A=0
20 CLT
30 FOR I=1 TO 10000
40 A=A+1
50 NEXT I
60 PRINT A
70 PRINT TICK();"msec"
```

```
RUN
10000
225msec
```

## 7.41 TICK 経過時間取得（数値関数）

経過時間を取得する。

## □ 書式

TICK()

TICK(モード)

## □ 引数

モード 0：経過時間をミリ秒単位で取得する（デフォルト）

1：経過時間を秒単位で取得する

## □ 戻り値

モード指定に従った経過時間を返す。

モード 0： 0 ～ 32767 ミリ秒の経過時間を返す(約 32.767 秒でオーバーフロー)

モード 1： 0 ～ 32767 秒の経過時間を返す(約 9.1 時間でオーバーフロー)

## □ 説明

起動からの経過時間を返します。経過時間は CLT コマンドにて初期化が可能です。

CLT コマンドと組み合わせて使うことで、処理時間の測定を行うことができます。

モード指定無し、または 0 を指定の場合、ミリ秒単位の経過時間を返します。

1 を指定した場合は、秒単位の経過時間を返します。

（注意）オーバーフローが発生しますので、長時間の測定には利用することが出来ません。

## □ 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、引数の指定が正しくない
Illegal value	: モードの指定値が正しくない
Overflow	: モードの指定値が -32767 ～ 32767 を超えている

## □ 利用例

1 万回ループの実行時間を測定する（プログラム：TICK.BAS）

```
10 A=0
20 CLT
30 FOR I=1 TO 10000
40 A=A+1
50 NEXT I
60 PRINT A
70 PRINT TICK();"msec"

RUN
10000
225msec
```

## 7.42 POKE 指定アドレスへのデータ書き込み

## □ 書式

POKE 仮想アドレス,データ

POKE 仮想アドレス,データ,データ, ... データ (可変個数指定)

## □ 引数

仮想アドレス : 仮想アドレス(16ビット) \$0000 ~

データ : 書き込むデータ (下位 8ビットのみ有効)

## □ 説明

指定した仮想アドレスに指定したデータを書き込みます。

仮想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き込みが簡単に行えます。

VRAM	: 画面表示用メモリ (CW×CH)	サイズは SCREEN モードにより可変
VAR	: 変数領域	サイズ 416 バイト
ARRAY	: 配列変数領域(@ (0) ~ @ (99) )	サイズ 200 バイト
PRG	: プログラム領域	サイズ 4096 バイト
MEM	: ユーザーワーク領域	サイズ 1024 バイト
GRAM	: グラフィック表示用メモリ	サイズは SCREEN モードにより可変

仮想アドレスの詳細については、「1.11 メモリーマップ」の「仮想アドレス」を参照下さい。

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。

指定した仮想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 への文字の書き込みは次のようになります。

VRAM に書き込んだ内容を画面の表示に反映するには REDRAW コマンドを実行します。

```
10 POKE VRAM+Y*80+X,ASC("A")
20 REDRAW
```

## □ 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 : S 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 の内容をユーザーワーク領域に保存する

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

## 7.43 PEEK 指定アドレスの値参照（数値関数）

## □ 書式

PEEK(仮想アドレス)

## □ 引数

仮想アドレス： 参照を行う SRAM 領域先頭からの相対バイトアドレス(16 ビット) \$0000 ~

## □ 戻り値

指定した仮想アドレスに格納されている 1 バイトデータ（0~255）

## □ 説明

指定した仮想アドレスに格納されている値（1 バイト）を返します。

仮想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。

VRAM	： 画面表示用メモリ（CW×CH）	サイズは SCREEN モードにより可変
VAR	： 変数領域	サイズ 416 バイト
ARRAY	： 配列変数領域(@ (0) ~ @ (99) )	サイズ 200 バイト
PRG	： プログラム領域	サイズ 4096 バイト
MEM	： ユーザーワーク領域	サイズ 1024 バイト
FNT	： フォントデータ	サイズ 2048 バイト
GRAM	： グラフィック表示用メモリ	サイズは SCREEN モードにより可変

仮想アドレスの詳細については、「1.11 メモリーマップ」の「仮想アドレス」を参照下さい。

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\*CW+X)

## 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 : 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

文字“A”のフォントデータを参照し、2 進数で表示する

```
10 FOR I=0 TO 7
20 D=PEEK(FNT+ASC("A")*8+I)
30 PRINT BIN$(D,8)
40 NEXT I
```

実行結果

```
run
01110000
10001000
10001000
10001000
11111000
10001000
10001000
00000000
OK
```

## 7.44 PRINT 画面への文字表示

## □ 書式

PRINT

PRINT 文字列|数値

PRINT 文字列|数値;

PRINT 文字列|数値; 文字列|数値;...

PRINT 文字列|数値; 文字列|数値;... ;

PRINT #桁数,文字列|数値; 文字列|数値;... ;

※ ...は可変指定を示す

※ ; は連結指定、カンマ','も可能

## □ 引数

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、式

連結指定 : セミコロン';' または カンマ','  
文末に付けると改行抑制される

桁数 : #数値 または #-数値 の形式で指定する

例:#3 、 #-3

## □ 説明

指定した文字列、式、関数、変数、数値をスクリーン画面のカーソル位置に表示します。

```
PRINT "Hello,World"
Hello,World
OK
PRINT 123*3
369
OK
PRINT HEX$(123*3)
171
OK
```

文字列、式、関数、変数、数値は連結指定のセミコロン';'、またはカンマ','にて連結して表示することが出来ます。

また最後に';'または','が付加されている場合は改行しません。

```
10 N=5:C=3:K=10
20 PRINT "N=";N;" C=";C;
30 PRINT " K=";
40 PRINT K
RUN
N=5 C=3 K=10
OK
```

## 数値の整形表示

#数値にて桁数を指定することで、任意の桁数（指定桁に満たない場合は空白を入れる）にて等間隔で表示します。

数値の前に-（マイナス）を付加した場合、空白をではなく、0(零)で不足桁を補います。

#数値は任意の位置、任意の回数指定できます。

## 桁数指定なし

```
PRINT 1;" ":"2;" ":"3
1:2:3
```

## 桁数指定あり(空白文字で補間)

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

## 桁数指定あり(0で補間)

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

### 表示位置の指定

LOCATE コマンドを併用することで、スクリーン画面の任意の位置に文字を表示することが出来ます。

```
10 LOCATE 10,10
20 PRINT "Hello!"
```

### 色、属性の指定

スクリーン画面が SCREEN 0 のターミナルコンソールの場合、

COLOR コマンド、ATTR コマンドを併用することで、文字の前景色、背景色の指定、点滅、アンダーライン等の属性を付加することが出来ます。

```
10 COLOR 4,3
20 PRINT "Hello,";
40 ATTR 2
50 PRINT "World".
60 COLOR 7,0:ATTR 0
```

## □ 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

時刻を表示する

```
10 GETTIME A,B,C
20 LOCATE 0,0:PRINT #-2,A;":";B;":";C
30 WAIT 1000
40 GOTO 20
```

実行結果

```
10:52:00
```

## 7.45 INPUT 数値の入力

## □ 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

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 : -32768 ~ 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符号-、+、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。

数値を入力せずに、[Enter]にて入力確定を行った場合、変数には 0 が格納されます。

引数に変数のみを指定した場合、“変数名:”を表示しその後ろの位置から数値を入力します。

```
INPUT A
A:
```

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位置から数値を入力します。

```
INPUT "Value=",A
Value=
```

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は、オーバーフロー時の既定値を変数に設定します。オーバーフロー時の既定値を設定していない場合は、Overflow エラーとなります。

オーバーフロー時の既定値なし

```
INPUT A
A:111111
Overflow
OK
```

オーバーフロー時の既定値あり

```
INPUT A, -1
A:111111
OK
?A
-1
OK
```

## □ 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
```

## 7.46 CLS 画面表示内容の全消去

## □ 書式

CLS

CLS 消去対象画面

## □ 引数

消去対象画面 :

0 メインコンソール画面の全消去 (デフォルト)

1 デバイスコンソール画面の消去

## □ 説明

画面上に表示している内容を全て消します。

引数に消去対象画面を指定した場合、指定した画面の消去を行います。

OLED 版、TFT 版ではシリアルコンソール画面切り替え中でも、デバイスコンソール画面へのグラフィック描画コマンドを使った描画が可能です。その際、CLS コマンドの引数にてどちらの画面を消去するかを指定することが出来ます

## □ エラーメッセージ

Illegal value : 指定した引数が有効範囲でない

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

画面の表示内容を消去する

CLS
OK



## 7.47 COLOR 文字色の設定

## □ 書式

COLOR 文字色

COLOR 文字色, 背景色

## □ 引数

文字色：色コード 0～8、または RGB コード：\$0009 ～ \$FFFF

背景色：色コード 0～8、または RGB コード：\$0009 ～ \$FFFF

## □ 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。

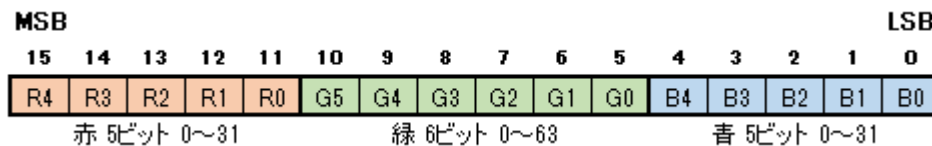
文字色、背景色で指定する色コードに対する色は次の表の通りです。

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

TFT(ILI9341)ディスプレイを利用している場合、上記の色コードに加えて、

RGB コード：\$0009 ～ \$FFFF の色指定が可能です。RGB コードは次の構成となります。



(注意) TFT デスプレイ、ターミナル版のみ利用可能です。

(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

```

## 7.48 ATTR 文字表示属性の設定

## □ 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) ターミナル版のみ利用可能です。

(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 : 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
```

## 7.49 LOCATE カーソルの移動

## □ 書式

LOCATE 横位置, 縦位置

## □ 引数

横位置：画面上の横位置 0 ～ CW-1

縦位置：画面上の縦位置 0 ～ CH-1

CW、CH は画面の横桁数、縦行数の示す定数です。

利用するフォント等により異なります。

## □ 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 CW-1 を超える数値を指定した場合、横位置は CW-1 となります。縦位置に CH-1 を超える数値を指定した場合、縦位置は CH-1 となります。

定数 CW、CH に設定されている値は次のようすることで確認することができます。

```
?CW
37
OK
?CH
27
OK
```

## □ 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.
```

## 7.50 REDRAW 画面表示の再表示

## □ 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

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

## □ 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
```

## 7.51 CSCROLL キャラクタ画面スクロール

## □ 書式

CSCROLL x1,y1,x2,y2,方向

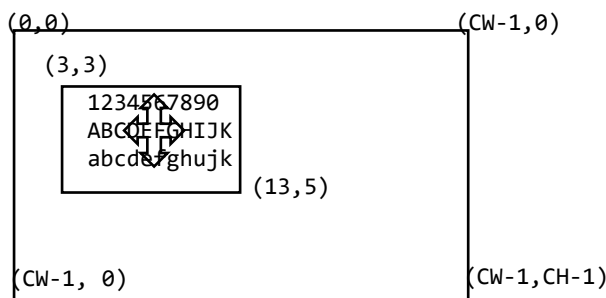
## □ 引数

x1 : 左上横座標 0 ~ CW-1 (最大値は利用フォントにより可変)  
 y1 : 左上縦座標 0 ~ CH-1 (最大値は利用フォントにより可変)  
 x2 : 右下横座標 0 ~ CW-1 (最大値は利用フォントにより可変)  
 y2 : 右下縦座標 0 ~ CH-1 (最大値は利用フォントにより可変)  
 方向 : UP(0): 上、DOWN(1): 下、RIGHT(2): 右、LEFT(3): 左  
 ※方向は定数またはカッコ内の数値の指定が可能

## □ 説明

画面の指定範囲でテキストを1文字単位でスクロールします。

本コマンドは、シリアルコンソール画面、TFT ディスプレイでは利用出来ません。



画面全体をスクロールしたい場合は、

CSCROLL 0,0,CW-1,CH-1,LEFT

のように、画面の左上の位置と画面右下の位置を指定します。

指定した範囲をスクロールしたい場合は、

CSCROLL 3,3,13,5,LEFT

のように範囲を指定します。

## □ 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  
 Illegal value : 指定した数値が不当である  
 Not supported : SCREEN 1~3 以外でコマンドを利用した

## □ 利用例

文字をスクロールする

```

10 CLS
20 LOCATE 10,10:?"Hello,TinyBASIC"
25 WAIT 200
30 FOR I=0 TO 15
40 CScroll 10,10,24,10,LEFT
50 WAIT 500
60 NEXT I

```

## 7.52 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 ~ CW - 1

縦位置: 0 ~ CH - 1

CW, CH は画面の横桁数、縦行数の示す定数です。

利用するフォント等により異なります。

## □ 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
Illegal value	: 指定した引数の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
'(' 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 CW-1
20 C=VPEEK(I,0)
30 PRINT CHR$(C);
40 NEXT I
50 PRINT
```

## 7.53 PSET 点の描画

## □ 書式

PSET 横座標, 縦座標, 色

## □ 引数

横座標 : 0 ~ GW-1 (注意 最大値は利用環境により異なる場合があります)

縦座標 : 0 ~ GH-1 (注意 最大値は利用環境により異なる場合があります)

色 : 0 黒、1 白、2 反転 または RGB コード \$0000~\$FFFF (TFT ディスプレイのみ)

※GW、GHはグラフィック画面の横ドット数、縦ドット数を示す定数です。

この定数は、画面解像度設定等により変わります。

## □ 説明

指定したグラフィック座標に指定した色の点を描画します。

本コマンドはシリアルコンソールでは利用出来ません。

NTSC 版、OLED 版では、色に2をした場合は、座標位置の色を反転した点を描画します。

(注意) 範囲外の座標を指定した場合、範囲内の境界に描画します。

## □ 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

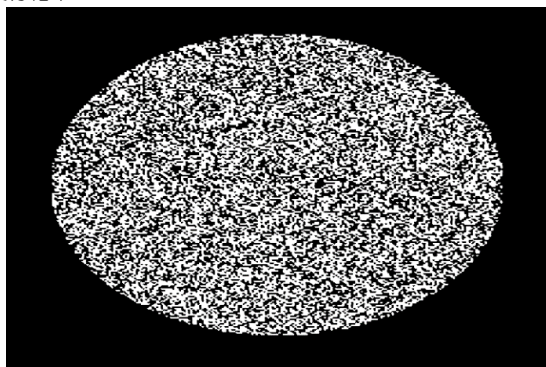
Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

半径 100 ドット内に点を描画する

```
10 CLS
20 FOR N=0 TO 30000
30 IF V!=99 Y=Y+V:V=V+1
40 X=RND(200)-100
50 Y=RND(200)-100
60 IF X*X+Y*Y<10000 PSET X+110,Y+108,1
70 NEXT N
80 GOTO 80
```

実行結果



## 7.54 LINE 直線の描画

## □ 書式

LINE 横座標 1, 縦座標 1, 横座標 2, 縦座標 2, 色

## □ 引数

横座標 1 : 0 ~ GW-1

縦座標 1 : 0 ~ GH-1

横座標 2 : 0 ~ GW-1

縦座標 2 : 0 ~ GH-1

色 : 0 黒、1 白、2 反転 または RGB コード \$0000~\$FFFF (TFT ディスプレイのみ)

※GW、GHはグラフィック画面の横ドット数、縦ドット数を示す定数です。

この定数は、画面解像度により変わります。

## □ 説明

指定したグラフィック座標 (横座標 1, 縦座標 1) と (横座標 2, 縦座標 2) 間を結ぶ直線を指定した色で描画します。色に 2 をした場合は、座標位置の色を反転した点を描画します。

本コマンドは NTSC 出力版またはグラフィック表示デバイス版にて利用可能です。

## □ 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

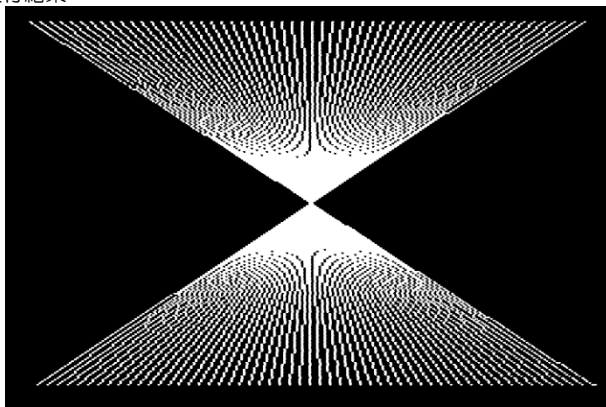
Overflow : 指定した数値が -32768 ~ 32767 を超えている

## □ 利用例

直線を描画する

```
10 CLS
20 FOR X=0 TO 223 STEP 4
30 LINE X,0,223-X,215,1
40 NEXT X
50 GOTO 50
```

実行結果





## 7.55 RECT 矩形の描画

## □ 書式

RECT x1, y1, x2, y2, 色, モード

## □ 引数

x1 : 左上横座標 0 ~ GW-1 (最大値は環境により可変)  
 y1 : 左上縦座標 0 ~ GH-1 (最大値は環境により可変)  
 x2 : 右下横座標 0 ~ GW-1 (最大値は環境により可変)  
 y2 : 右下縦座標 0 ~ GH-1 (最大値は環境により可変)  
 色 : 0 黒、1 白、2 反転 または RGB コード \$0000~\$FFFF (TFT ディスプレイのみ)  
 モード : 0 塗りつぶしなし, 0 以外 塗りつぶしあり

※GW、GH はグラフィック画面の横ドット数、縦ドット数を示す定数です。

この定数は、画面解像度に設定等により変わります。

## □ 説明

指定した位置に指定した色で矩形を描画します。

モードに 0 を指定した場合は線のみを描画します。0 以外を指定した場合は白で塗りつぶします。

## □ 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  
 Illegal value : 指定した数値が不当である

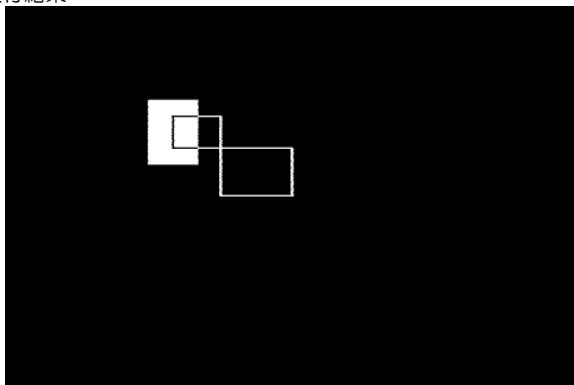
## □ 利用例

矩形を描画する

```

10 CLS
20 RECT 50,50,70,90,1,1
30 RECT 60,60,80,80,2,0
40 RECT 80,80,110,110,1,0
50 GOTO 50
  
```

実行結果



## 7.56 CIRCLE 円の描画

## □ 書式

CIRCLE 横中心座標, 縦中心座標, 半径, 色, モード

## □ 引数

横中心座標 : 0 ~ GW-1

縦中心座標 : 0 ~ GH-1

半径 : 1 ~ GW-1

色 : 0 黒、1 白、2 反転 または RGB コード \$0000~\$FFFF (TFT ディスプレイのみ)

モード : 0 塗りつぶしなし, 0 以外 塗りつぶしあり

※GW、GH はグラフィック画面の横ドット数、縦ドット数を示す定数です。

この定数は、画面解像度に設定等により変わります。

## □ 説明

指定した中心座標(横中心座標, 縦中心座標)を起点に指定した半径の円を色で描画します。

モードに 0 を指定した場合は線のみを描画します。0 以外を指定した場合は指定した色で塗りつぶします。

## □ 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

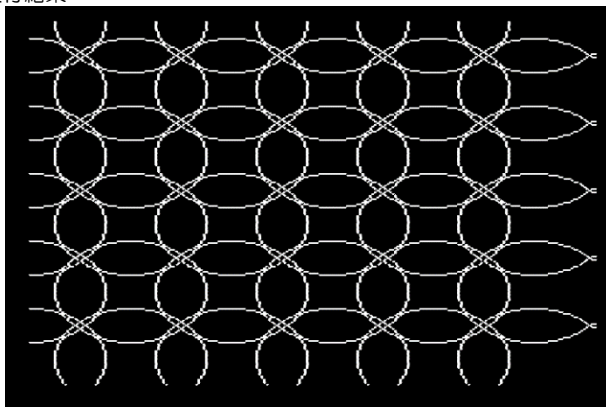
Overflow : 指定した数値が-32768 ~ 32767 を超えている

## □ 利用例

円を描画する

```
10 CLS
20 FOR Y=0 TO 215 STEP 40
30 FOR X=0 TO 223 STEP 40
40 CIRCLE X,Y,30,1,0
50 NEXT X
60 NEXT Y
70 GOTO 70
```

実行結果



## 7.57 BITMAP ビットマップ画像の描画

## □ 書式

BITMAP 横座標, 縦座標, 仮想アドレス, インデックス, 幅, 高さ [, 倍率]

## □ 引数

横座標 : 0 ~ GW-1  
 縦座標 : 0 ~ GH-1  
 仮想アドレス : 数値(任意の仮想アドレス)  
 インデックス : 0 ~ 32767  
 幅 : 1 ~ GW-1  
 高さ : 1 ~ GH-1  
 倍率 : 1 ~ 8

GW、GH はグラフィック画面の横ドット数、縦ドット数を示す定数です。  
 この定数は、画面解像度により変わります。

## □ 説明

指定した座標にビットマップ画像を描画します。

仮想アドレスに描画対象の画像データの先頭格納アドレスを指定します。

インデックスには画像データの先頭格納の格納位置を指定します。

(幅 + 7) / 8 + 高さ \* インデックス の計算にて参照する画像格納位置を移動します。

## □ 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

## □ 利用例

インバーダーのキャラクタをアニメーション表示する（4倍で表示）

```
10 CLS
20 POKE MEM+0,$10,$40,$48,$90,$5F,$D0,$77,$70
30 POKE MEM+8,$3F,$E0,$1F,$C0,$10,$40,$20,$20
40 POKE MEM+16,$10,$40,$08,$80,$1F,$C0,$37,$60
50 POKE MEM+24,$7F,$F0,$5F,$D0,$50,$50,$0D,$80
60 FOR A=0 TO 20
70 BITMAP A,0,MEM,A%2,12,8,4
80 WAIT 200
90 NEXT A
```

実行結果



## 7.58 GPRINT 文字列の描画

## □ 書式

GPRINT 横座標,縦座標

GPRINT 横座標,縦座標文字列|数値

GPRINT 横座標,縦座標文字列|数値;

GPRINT 横座標,縦座標文字列|数値;文字列|数値;…

GPRINT 横座標,縦座標文字列|数値;文字列|数値;… ;

GPRINT 横座標,縦座標#桁数,文字列|数値: 文字列|数値:… ;

※ | はいずれのうち1つを示す

※ …は可変指定を示す

※ ; は連結指定、カンマ','も可能

## □ 引数

横座標 : 横描画位置 0 ~ GW-1 (通常は 223)

縦座標 : 縦描画位置 0 ~ GH-1 (通常は 215)

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、式

連結指定 : セミコロン';' または カンマ','  
文末に付けると改行抑制される

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

## □ 説明

指定したグラフィック座標にグラフィックとして文字列を描画します。

横座標、縦座標以降の記述及び出力結果は PRINT 文と同等です。

出力する文字列の表記方法については、「7.44 PRINT 画面への文字表示」を参照下さい。

## □ 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

時刻を表示する

```
10 GETTIME A,B,C
20 GPRINT 0,0,#-2,A;":";B;":";C
30 WAIT 1000
40 GOTO 10
```

実行結果

10:52:00

## 7.59 GSCROLL グラフィックスロール

## □ 書式

GSCROLL x1, y1, x2, y2, 方向

## □ 引数

x1 : 左上横座標 0 ~ GW-1 (最大値は環境により可変)  
 y1 : 左上縦座標 0 ~ GH-1 (最大値は環境により可変)  
 x2 : 右下横座標 0 ~ GW-1 (最大値は環境により可変)  
 y2 : 右下縦座標 0 ~ GH-1 (最大値は環境により可変)  
 方向 : UP(0): 上、DOWN(1): 下、RIGHT(2): 右、LEFT(3): 左

※方向は定数またはカッコ内の数値の指定が可能

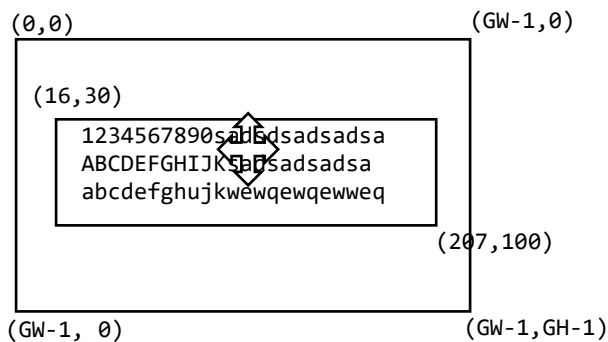
※GW、GHはグラフィック画面の横ドット数、縦ドット数を示す定数です。

この定数は、画面解像度により変わります。

## □ 説明

画面上の指定範囲表示内容を 1 ドット単位でスクロールします。

本コマンドは、シリアルコンソール画面、TFT ディスプレイでは利用出来ません。



NTSC ビデオディスプレイ使用時には次の制約があります。OLED ディスプレイでは制約はありません。

- ・右上横方向x1は8の倍数で指定して下さい (バイト境界指定)。
- ・左下横座標x2は8の倍数-1で指定して下さい (バイト境界指定)。

上記以外の数値を指定した場合、指定数値をバイト境界に補正します。

画面全体を左スクロールしたい場合は、

GSCROLL 0, 0, GW-1, GH-1, LEFT

のように、画面全体の領域を指定します。

指定した範囲をスクロールしたい場合は、

GSCROLL 16,30,207,100, LEFT

のように範囲を指定します。

(注意) GSCROLL でスクロールした場合、見かけ上のテキストとテキスト表示管理の情報にずれが生じます。VPEEK による表示位置の文字コード取得は正しい値を返しません。

## □ 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  
 Overflow : 指定した数値が-32768 ~ 32767 を超えている  
 Illegal value : 指定した数値が不当である

## □ 利用例

### 簡易オシロスコープ

```
10 'oscilloscope
20 CLS
30 GPIO PB1,ANALOG
40 "LOOP"
50 R=ANA(PB1)
60 Q=R/20
70 LOCATE 0,0:?" "
80 PSET 223,221-Q,1
90 WAIT 50
100 GSCROLL 0,8,GW-1,GH-1,LEFT
110 GOTO "LOOP"
```

## 7.60 LDBMP ビットマップファイルの読み込み

## □ 書式

LDBMP “ファイル名”, 仮想アドレス, bx, by, 幅, 高さ[, 色指定]

## □ 引数

“ファイル名” : ディレクトリパス(省略可能)+ファイル 8 文字+拡張子 3 文字 のファイル名  
 仮想アドレス : データの読み込みアドレス(仮想アドレス)  
 Bx : ビットマップ画像の切り出し横座標 0 ~ 32767  
 By : ビットマップ画像の切り出し縦座標 0 ~ 32767  
 幅 : ビットマップ画像の切り出しドット幅 1 ~ 32767  
 高さ : ビットマップ画像の切り出しドット高 1 ~ 32767  
 色指定 : 0 変換なし、1 反転

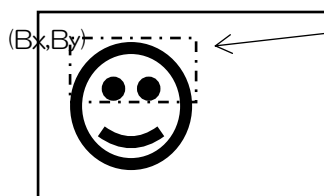
## □ 説明

SD カード内の Windows ビットマップファイル (モノクロ 2 値) の指定領域の画像を切り出して、メモリ上に読み込みます。  
 読み込む画像はファイル名で指定します。

仮想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き込みが簡単に行えます。

VRAM	: 画面表示用メモリ (CW×CH)	サイズは SCREEN モードにより可変
VAR	: 変数領域	サイズ 416 バイト
ARRAY	: 配列変数領域 (@ (0) ~ @ (99) )	サイズ 200 バイト
PRG	: プログラム領域	サイズ 4096 バイト
MEM	: ユーザーワーク領域	サイズ 1024 バイト
GRAM	: グラフィック表示用メモリ	サイズは SCREEN モードにより可変

仮想アドレスの詳細については、「1.11 メモリーマップ」の「仮想アドレス」を参照下さい。



切り出す領域

ビットマップ画像の座標 (Bx,By) から  
幅、高さの画像を連続データとして  
指定したアドレスに格納する。

ビットマップ画像上の座標 Bx、By から指定した幅、高さの画像を指定したアドレスに連続データとして格納します。指定した幅、高さがビットマップ画像よりも大きい場合は、切り出し幅、高さを画像サイズにて調整します。

引数の色指定にて、元の画像の白・黒を反転して読み込む指定が可能です。

**(注意)** 連続したデータが指定した保存可能な領域をオーバーするチェックは行っていません。あらかじめ画像サイズを見積もった上で利用して下さい。ユーザーワーク領域(1024 バイト)を利用した場合、格納できる画像は、  
 128 ドット×64 ドットの画像程度です。  
 大きい画像を表示する場合、グラフィック表示用メモリに直接読み込むか、  
 DWBMP コマンドを利用して下さい。

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

## □ 利用例

下記の横 96 ドット、縦 64 ドットの画像（768 バイト）ファイル“TT.BMP”の全データをユーザーワーク領域(MEM〜)に読み込み、BITMAP コマンドで 2 倍表示する。

画像ファイル      プログラム



```
10 CLS
20 LDBMP "TT.BMP",MEM,0,0,96,64
30 BITMAP 10,30,MEM,0,96,64,2
40 GOTO 40
```

実行結果



同画像を、「豊」「四」「季」「TBASIC」の 4 つに分けて読み込み、BITMAP コマンドで分離して表示する。「豊」は反転して読み込む。

プログラム

```
10 CLS
20 LDBMP "TT.BMP",MEM,0,0,32,32,1
30 LDBMP "TT.BMP",MEM+128,32,0,32,32
40 LDBMP "TT.BMP",MEM+256,64,0,32,32
50 LDBMP "TT.BMP",MEM+384,0,32,96,32
60 BITMAP 10,30,MEM,0,32,32,2
70 BITMAP 74,40,MEM,1,32,32,2
80 BITMAP 138,50,MEM,2,32,32,2
90 BITMAP 10,120,MEM+384,0,96,32,2
```

実行結果

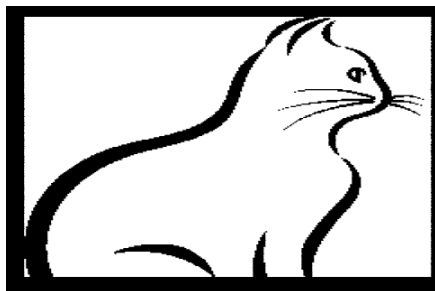


大きいサイズの画像を直接グラフィック表示用メモリに読み込んで画像を表示する。

プログラム

```
10 CLS
20 LDBMP "CAT.BMP",GRAM,0,0,GW,GH
30 GOTO 30
```

実行結果



※画像は Helm42 さんの猫絵を利用させて頂いています。

<http://free-illustrations.gatag.net/tag/%E7%8C%AB-%E3%83%8D%E3%82%B3>



## 7.61 DWBMP BMP ファイルの表示

## □ 書式

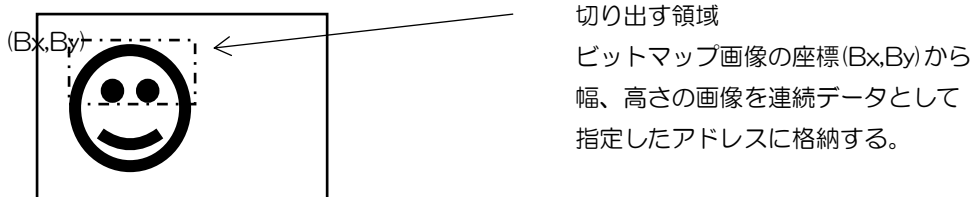
DWBMP “ファイル名”, x, y, Bx, By 幅, 高さ[色指定]

## □ 引数

“ファイル名” : ディレクトリパス(省略可能)+ファイル 8 文字+拡張子 3 文字 のファイル名  
 X : 画像表示位置 横 0 ~ GW-1 ※8 の倍数であること  
 y : 画像表示位置 縦 0 ~ GH-1  
 Bx : ビットマップ画像の切り出し横座標 0 ~ 32767 ※8 の倍数であること  
 By : ビットマップ画像の切り出し縦座標 0 ~ 32767  
 幅 : ビットマップ画像の切り出しドット幅 1 ~ GW ※8 の倍数であること  
 高さ : ビットマップ画像の切り出しドット高 1 ~ GH  
 色指定 : 0 変換なし、1 反転 ※ TFT ディスプレイの場合、指定出来ません

## □ 説明

SD カード内の Windows ビットマップファイルの指定領域の画像を切り出して、指定した座標(x,y)に表示します。  
 NTSC、OLED ディスプレイの場合、ビットマップファイルはモノクロ（白黒2値）画像のみ利用出来ます。  
 TFT デスプレイの場合、16 ビットカラービットマップファイルのみ利用出来ます。



ビットマップ画像上の座標 Bx、By から指定した幅、高さの画像を切り出して指定位置に表示します。  
 表示位置の x、ビットマップ画像の切り出し位置 Bx、切り出し幅はバイト境界である8の倍数である必要があります。8の  
 倍数でない場合は8の倍数に丸めます。

NTSC、OLED ディスプレイの場合、引数の色指定にて、元の画像の白・黒を反転して読み込む指定が可能です。  
 TFT ディスプレイの場合、引数の色指定は利用出来ません。

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

## □ 利用例

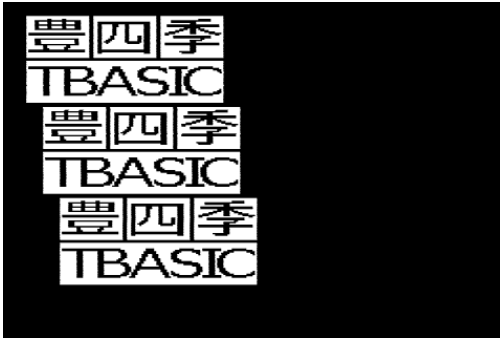
下記の横 96 ドット、縦 64 ドットの画像（768 バイト）ファイル“TT.BMP”を3つ表示する。  
 それぞれの画像表示開始位置は、座標(0,0)、座標(8,64)、座標(16,130)とする。

画像ファイル プログラム



```
10 CLS
20 DWBMP "TT.BMP",0,0,0,0,96,64
30 DWBMP "TT.BMP",8,65,0,0,96,64
40 DWBMP "TT.BMP",16,130,0,0,96,64
50 GOTO 50
```

## 実行結果



同画像から指定境域の画像を切り出して表示する。

## プログラム

```
10 CLS
20 DWBMP "TT.BMP",0,0,0,0,30,30
30 DWBMP "TT.BMP",8,33,0,0,64,32
40 DWBMP "TT.BMP",16,66,0,32,96,32
50 GOTO 50
```

## 実行結果



## 7.62 GPEEK 画面上の指定位置ピクセルの参照（数値関数）

## □ 書式

GPEEK(横位置, 縦位置)

## □ 引数

横位置: 0 ~ GW - 1

縦位置: 0 ~ GH - 1

※GW、GH はグラフィック画面の横ドット数、縦ドット数を示す定数です。

画面解像度により異なります。

## □ 戻り値

指定位置に表示されているピクセルの色 0 または 1

## □ 説明

画面上の指定位置に表示されているピクセルを参照します。

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

画面上に表示しているドットに有無を 2 進数で表示する

```
10 B=0
10 FOR I=0 TO 7
20 B=(B<<1)+GPEEK(I,0)
30 NEXT I
40 PRINT BIN$(B,8)
50 END
```

## 7.63 GINP 指定領域のピクセルの有無判定 (数値関数)

## □ 書式

GINP(横位置,縦位置,高さ,幅,色)

## □ 引数

横位置 : 0～GW - 1  
 縦位置 : 0～GH - 1  
 幅 : 1 ～ GW-1  
 高さ : 1 ～ GH-1  
 色 : 0 黒、1 白

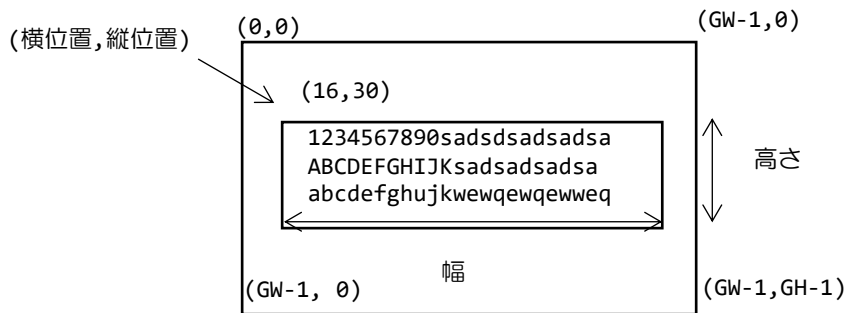
※GW、GH はグラフィック画面の横ドット数、縦ドット数を示す定数です。  
 画面解像度より異なります。

## □ 戻り値

1 : 指定領域内 (境界含む) に指定した色のピクセルが存在する  
 0 : 指定領域内 (境界含む) に指定した色のピクセルが存在しない

## □ 説明

指定した領域内の指定した色のピクセルが存在するをチェックし、その結果を返します。



## □ 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

画面上に表示しているドットに有無を2進数で表示する

```
10 B=0
10 FOR I=0 TO 7
20 B=(B<<1)+GPEEK(I,0)
30 NEXT I
40 PRINT BIN$(B,8)
50 END
```

## 7.64 TONE 単音出力

## □ 書式

TONE 周波数

TONE 周波数,出力期間

## □ 引数

周波数 : 0 ~ 32767 (Hz) 0 の場合は消音

出力期間 : 0 ~ 32767 (ミリ秒) 0 の場合は、継続再生

## □ 説明

PB9 ピンより、指定した周波数のパルス出力(デューティ比 50%)を行います。

PB9 ピンに圧電スピーカ（圧電サウダ）を接続すること音を出すことができます。

出力期間の指定がある場合は、その期間パルスを出力します(ミリ秒単位)。

出力期間の指定がある場合、出力完了待ちを行います。

出力期間の指定がない場合は、NOTONE コマンドで停止指示をするまでパルスを出力し続けます。

(TONE 0 は NOTONE と等価です)

音階・周波数対応表

	ド	ド#	レ	レ#	ミ	ファ	ファ#	ソ	ソ#	ラ	ラ#	シ
1	33	35	37	39	41	44	46	49	52	55	58	62
2	65	69	73	78	82	87	93	98	104	110	117	123
3	131	139	147	156	165	175	185	196	208	220	233	247
4	262	277	294	311	330	349	370	392	415	440	466	494
5	523	554	587	622	659	698	740	784	831	880	932	988
6	1047	1109	1175	1245	1319	1397	1480	1568	1661	1760	1865	1976
7	2093	2217	2349	2489	2637	2794	2960	3136	3322	3520	3729	3951
8	4186	4435	4699	4978	5274	5588	5920	6272	6643	7040	7459	7902

## □ 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

スペースキーを押したら音を鳴らす

```

10 IF INKEY() = 32 TONE 800,50
20 GOTO 10

```

## 7.65 NOTONE 単音出力停止

## □ 書式

NOTONE

## □ 引数

なし

## □ 説明

TONE コマンドによるパルス出力を停止します。

## □ 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

音を停止する

```
10 TONE 400  
20 WAIT 200  
30 NOTONE
```

## 7.66 DATE 現在時刻の表示

## □ 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  
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

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

## 7.67 GETDATE 日付の取得

## □ 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。

格納される値は次の通りです。

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

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

## □ 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
```



## 7.68 GETTIME 時刻の取得

## □ 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

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

## □ 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。

## 7.69 SETDATE 時刻の設定

## □ 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
```

## 7.70 GPIO GPIO 機能設定

## □ 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。

PA0, PA1, PA2, PA3, PA4, PA5, PA6, PA7, PA8, PA9, PA10, PA13, PA14, PA15, PB0, PB1, PB2, PB3, PB4, PB5, PB8, PB10, PB11, PB12, PB13, PB14, PB15, PC13

モード :

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  
 PWM : PWM 出力

## □ 説明

ボード上の指定したピン番号の入出力機能の設定を行います。

Arduino の pinMode() に相当します。

ピン番号の指定には、0~34 の数値または、定数 PA0~PC15 が可能です。

また、GPIO は省略可能です。下記の2つは等価です。

GPIO PA0, OUTPUT  
 PA0, OUTPUT

各モードで利用可能なピン番号

モード	ピン番号
OUTPUT_OD OUTPUT INPUT_FL INPUT_PU INPUT_PD	PA0, PA1, PA2, PA3, PA4, PA5, PA6, PA7, PA8, PA9, PA10, PA13, PA14, PA15, PB0, PB1, PB2, PB3, PB4, PB5, PB8, PB10, PB11, PB12, PB13, PB14, PB15, PC13
ANALOG	PA0, PA1, PA2, PA3, PA4, PA5, PA6, PA7, PB0, PB1
PWM(※1)	グループ 1 PA6, PA7, PB0, PB1
	グループ 2 PA8, PA9, PA10

※1 PWM のグループ内では PWM 周波数が共通設定となります。

**（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。

**（注意）** 利用するシステム構成により、利用出来ないピンがあります。詳細については、「1.10 ポート上のピン一覧」を参照して下さい。

起動直後では各ピンの設定は、デジタル入力（IN()）またはアナログ入力（ANA()）となります。

これは動作を保障するものではありません。GPIO による機能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  
 Illegal value : ピン番号、モードに範囲外の値を指定した  
 Cannot use GPIO fuinction : ピン番号に利用出来ないモード設定を行った  
 Overflow : 指定した数値が-32768 ~ 32767 を超えている  
 Illegal value : 指定した数値が不当である

## □ 利用例

PB1 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(PB1)
40 LOCATE 5,5: ? A; "    "
50 GOTO 30
```

## 7.71 OUT デジタル出力

## □ 書式

OUT ピン番号, 出力値

## □ 引数

ピン番号 : 0 ~ 34

PA0, PA1, PA2, PA3, PA4, PA5, PA6, PA7, PA8, PA9, PA10, PA13, PA14, PA15,  
PB0, PB1, PB2, PB3, PB4, PB5, PB8, PB10, PB11, PB12, PB13, PB14, PB15, PC13

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。

（注意）利用するシステム構成により、利用出来ないピンがあります。詳細については、  
「1.10 ポート上のピン一覧」を参照して下さい。

## □ 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
Cannot use GPIO fuinction	: ピン番号に利用出来ないモード設定を行った
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

```

## 7.72 POUT PWM パルス出力

## □ 書式

POUT ピン番号, デューティ値

POUT ピン番号, デューティ値, 周波数

## □ 引数

ピン番号： 0～34 下記のピンが利用可能です。

グループ 1	PA6, PA7, PB0, PB1
グループ 2	PA8, PA9, PA10

デューティ値： 0 ～ 4095

0 がデューティ比 100%となります。

4095 がデューティ比 100%となります。

周波数： 0 ～ 32767 (単位 Hz)

指定しない場合は 490Hz となります (Arduino Uno 互換)

周波数はピン番号のグループ内で共通

## □ 説明

指定ピンから、PWM パルス出力を行います。

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

パルス出力を停止する場合は、デューティ値に 0 を指定して下さい。

**（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

アナログジョイスティックでサーボモーターの制御を行う

```

1 'サーボ モーターセッ'
5 CLS
10 GPIO PB0, ANALOG
20 GPIO PA8, PWM
30 P=MAP(ANA(PB0), 0, 4095, 102, 491)
40 POUT PA8, P, 50
45 D=MAP(P, 102, 491, -90, 90)
47 LOCATE 0, 0: ?#3, D
50 GOTO 30

```

## 7.73 SHIFTOUT デジタルシフトアウト出力

## □ 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

PA0, PA2, PA3, PA4, PA5, PA6, PA7, PA8, PA9, PA10, PA13, PA14, PA15,  
PB0, PB1, PB2, PB3, PB8, PB10, PB11, PB12, PB13, PB14, PC13

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

シフトレジスタ 74HC595 を使って LED8 個を制御する

```
10 GPIO PB12,OUTPUT
20 GPIO PB13,OUTPUT
30 GPIO PB14,OUTPUT
40 D=$80
50 FOR I=0 TO 6
60 GOSUB "led"
70 D=D>>1
80 WAIT 200
90 NEXT I
100 FOR I=0 TO 6
110 GOSUB "led"
120 D=D<<1
130 WAIT 200
140 NEXT I
150 GOTO 40
160 "led"
170 OUT PB13,LOW
180 SHIFTOUT PB12,PB14,MSB,D
190 OUT PB13,HIGH
200 TONE 600,10
210 RETURN
```



## 7.74 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

PA0, PA2, PA3, PA4, PA5, PA6, PA7, PA8, PA9, PA10, PA13, PA14, PA15,  
PB0, PB1, PB2, PB3, PB8, PB10, PB11, PB12, PB13, PB14, PC13

## □ 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
Cannot use GPIO fuinction	: ピン番号に利用出来ないモード設定を行った
Overflow	: 指定した数値が-32768 ~ 32767 を超えている
'(' or ')' expected	: '(' または ')' がない

## □ 利用例

## 7.75 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, PB1

## □ 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

PB1 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(PB1)
40 LOCATE 5,5: ? A; "    "
50 GOTO 30
```

## 7.76 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)

SHIFTIN(データピン番号, クロックピン番号, 入力形式, 条件)

## □ 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。

PA0, PA2, PA3, PA4, PA5, PA6, PA7, PA8, PA9, PA10, PA13, PA14, PA15,  
PB0, PB1, PB2, PB3, PB8, PB10, PB11, PB12, PB13, PB14, PC13

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

条件 : LOW または HIGH

データピンからのデータを読み取るクロックのタイミングを指定します。

LOW : クロックが LOW の場合にデータを読み取る。

HIGH : クロックが HIGH の場合にデータを読み取る。

## □ 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。

引数に条件を指定した場合、クロックが指定した状態の時にデータピンからデータを読み取ります。

条件を指定していない場合は、条件は HIGH（Arduino の shiftIn() と同様の仕様）となります。

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

ファミコン用ゲームパッドからボタン操作情報を取得する

```

10 CLS
20 GPIO PB12, INPUT_FL
30 GPIO PB13, OUTPUT
40 GPIO PB14, OUTPUT
50 OUT PB13, HIGH
60 OUT PB13, LOW
70 R=SHIFTIN(PB12, PB14, LSB, LOW)
80 LOCATE 0,0: ?BIN$(R,8)
90 WAIT 100
100 GOTO 50

```

## 7.77 PULSEIN 入力パルス幅の計測力（数値関数）

## □ 書式

PULSEIN(パルス入力ピン番号, 検出信号, タイムアウト)

PULSEIN(パルス入力ピン番号, 検出信号, タイムアウト, スケール値)

## □ 引数

パルス入力ピン番号 : 0 ~ 34 データを入力するピン

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。

PA0, PA2, PA3, PA4, PA5, PA6, PA7, PA8, PA9, PA10, PA13, PA14, PA15,  
PB0, PB1, PB2, PB3, PB8, PB10, PB11, PB12, PB13, PB14, PC13

検出信号 : LOW、HIGH または 0、1 測定対象のパルス

タイムアウト : パルス検出待ちタイムアウト時間 0 ~ 32767 (ミリ秒)

スケール値 : 計測時間のスケール変換 1 ~ 1327671 (デフォルト値 1)

## □ 戻り値

正常時: 測定したパルス幅 0 ~ 32767 (単位はスケール値 × マイクロ秒)

タイムアウト時: 0

オーバーフロー時: -1

## □ 説明

**パルス入力ピン番号**で指定したピンから入力される信号のパルス幅を計測し、その値を返します。

測定完了は、入力ピンの状態が**検出信号**の状態になった時点で計測を開始し、**検出信号**の状態でなくなった時点で計測を終了します。たとえば、測定する**検出信号**が **HIGH** の場合、PULSEIN()はピンが **HIGH** になるのを待ち、**HIGH** になった時点で計測を開始し、ピンが **LOW** になるタイミングで計測を終了し、そのパルスの長さを返します。**タイムアウト**内に完全なパルスが受信されなかった場合は 0 を返します。

測定したパルスが整数値 32767 を超えた場合は、オーバーフローとし-1 を返します。

オーバーフローを回避したい場合は、**スケール値**を適宜調整して下さい。

**スケール値**が 1 の場合、1~31767 マイクロ秒までのパルスの測定を行うことが出来ます。この場合、500kHz~15Hz までのパルスを測定出来ます。15Hz より遅い、信号の測定はオーバーフローが発生します。

**スケール値**が 1000 の場合、1~31767 ミリ秒までのパルスの測定を行うことが出来ます。この場合、500Hz~0.015Hz までのパルスの測定が可能です。

本関数は Arduino の pulseIn() 関数を利用して計測しています。

測定値はあくまでも目安であり、精度はあまり高くありません。

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。

NTSC 版では、NTSC ビデオ信号生成を随時並行して行っているため測定値に誤差が生じます。NTSC 版で利用の場合は、コンソールモードにて測定することで精度が向上します。

## □ 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 : 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

パルス幅を計測する

```
10 CLS
20 GPIO PA8, INPUT_PU
30 A=PULSEIN(PA8, LOW, 200, 1)
40 LOCATE 0, 0: ?#8, A
50 GOTO 30
```

補足：PA8 ピンは 5V トレラント（5V の入力が可能）です。パルス計測等には 5V トレラントのピンの利用を推奨します。

## 7.78 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,受信データアドレス,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仮想アドレス  
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内仮想アドレス  
データ長 : 0 ~ 32767  
受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仮想アドレスにて指定します。

仮想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き込み・参照が簡単に行えます。

MEM : ユーザーワーク領域                      サイズ 2048 バイト  
VRAM : 画面表示用メモリ (CW×CH)              サイズ利用フォントにより可変  
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 容量を超える仮想アドレス (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
```

## 7.79 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 内仮想アドレス

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 内仮想アドレス

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 先頭からの仮想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。

仮想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き込み・参照が簡単に行えます。

MEM : ユーザーワーク領域	サイズ 2048 バイト
VRAM : 画面表示用メモリ (CW×CH)	サイズ利用フォントにより可変
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 容量を超える仮想アドレス (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
```

## 7.80 SMODE シリアルポート機能設定

## □ 書式

SMODE 0

SMODE 1, “通信速度”

SMODE 2

SMODE 3, 制御コード無加工指定

## □ 引数

通信速度 : 文字列形式の数値 “110”～ “921600”

制御コードの制御 : 0 なし、1 : あり

## □ 説明

2つのシリアルポート（USBポート、GPIOシリアルポート）に関する機能設定を行います。

## 1) SMODE 0

2つのシリアルポートに次の機能を割り付けます（デフォルト設定）

- ・USBポート : ターミナル対話利用(入力、プログラム転送、デバッグ)
- ・GPIOシリアルポート（PA9、PA10） : データ通信用（SREAD、SWRITE等のコマンド利用用）

## 2) SMODE 1, “通信速度”

2つのシリアルポートに次の機能を割り付けます（デフォルト設定）

- ・USBポート : データ通信用（SREAD、SWRITE等のコマンド利用用）
- ・GPIOシリアルポート（PA9、PA10） : ターミナル対話利用(入力、プログラム転送、デバッグ)

第2引数にはGPIOシリアルポート（PA9、PA10）の通信速度を数字文字列にて指定します。

本コマンドにより、GPIOシリアルポートは指定した通信速度にてオープンされます。

## 3) SMODE 2

シリアル通信禁止

ターミナル対話利用のコンソールでの入出力、データ通信用の入出力は利用出来ません。

出力を行った場合は破棄されます。

## 4) SMODE 3, 制御コード無加工指定

シリアルターミナルからの制御コードの処理を指定します。

0の場合、処理を行いません。1の場合は処理を行います。

シリアルポートをデータ通信に設定した場合は次のコマンドを利用することができます。

SMODE	: GPIOシリアルポート、USBシリアルポート機能切り替え
SOPEN	: シリアル通信開始
SERITE	: 1バイト送信
SREAD	: 1バイト受信
SREADY	: 受信データ有無の確認
SPRINT	: 文字列出力(PRINTと同等の出力)
SCLOSE	: シリアル通信クローズ

**（注意）** シリアル通信をオープンした状態でGPIOコマンドにてPA9,PA10にI/O機能の割り当てを行った場合、正しい動作を行うことが出来ません。  
必ずSCLOSEにてシリアル通信をクローズしてからピンに機能を割り当て下さい。

## □ 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
Illegal value	: 指定した引数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

SMODE 1, "115200"
-------------------

## 7.81 SOPEN シリアル通信オープン

## □ 書式

SOPEN 通信速度

## □ 引数

通信速度 : 文字列形式の数値 “110”～ “921600”  
 例: “115200”

## □ 説明

データ通信設定した GPIO シリアルポート (PA9、PA10) のシリアル通信の利用を開始します。

通信速度は文字列形式の数値で指定します。

GPIO シリアルポートを使ったシリアル通信は次の GPIO ピンを利用することができます。

PA9 : TX 送信  
 PA10 : RX 受信

シリアルポートが既にオープン状態の場合は、シリアルポートをクローズしてからオープンします。

シリアル通信の利用には次のコマンドが用意されています。

SMODE : GPIO シリアルポート、USB シリアルポート機能切り替え  
 SOPEN : シリアル通信開始  
 SERITE : 1 バイト送信  
 SREAD : 1 バイト受信  
 SREADY : 受信データ有無の確認  
 SPRINT : 文字列出力 (PRINT と同等の出力)  
 SCLOSE : シリアル通信クローズ

**(注意)** シリアル通信をオープンした状態で GPIO コマンドにて PA9,PA10 に I/O 機能の割り当てを行った場合、正しい動作を行うことが出来ません。必ず SCLOSE にてシリアル通信をクローズしてからピンに機能を割り当て下さい。

## □ 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  
 Illegal value : 指定した引数に範囲外の値を指定した

## □ 利用例

シリアル通信経由でテキストメッセージを送信する

```
10 SMODE 0
20 SOPEN "9600"
30 SPRINT "Hello,TinyBASIC"
40 SCLOSE
```

## 7.82 SCLOSE シリアル通信クローズ

## □ 書式

SCLOSE 通信速度

## □ 引数

なし

## □ 説明

データ通信設定したシリアルポートのシリアル通信を終了します。

シリアル通信は次の GPIO ピンまたは、USB ポートを利用することが出来ます。

PA9 : TX 送信

PA10 : RX 受信

シリアル通信の利用には次のコマンドが用意されています。

SMODE	: GPIO シリアルポート、USB シリアルポート機能切り替え
SOPEN	: シリアル通信開始
SERITE	: 1 バイト送信
SREAD	: 1 バイト受信
SREADY	: 受信データ有無の確認
SPRINT	: 文字列出力(PRINT と同等の出力)
SCLOSE	: シリアル通信クローズ

(注意) シリアル通信をオープンした状態で GPIO コマンドにて PA9,PA10 に I/O 機能の割り当てを行った場合、正しい動作を行うことが出来ません。必ず SCLOSE にてシリアル通信をクローズしてからピンに機能を割り当て下さい。

## □ 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

シリアル通信経由でテキストメッセージを送信する

```
10 SOPEN "9600"
20 SPRINT "Hello,TinyBASIC"
30 SCLOSE
```

## 7.83 SPRINT シリアル通信文字列出力

## □ 書式

SPRINT

SPRINT 文字列|数値

SPRINT 文字列|数値;

SPRINT 文字列|数値;文字列|数値;...

SPRINT 文字列|数値;文字列|数値;... ;

SPRINT #桁数,文字列|数値; 文字列|数値;... ;

※ | はいずれのうち1つを示す

※ ...は可変指定を示す

※ ; は連結指定、カンマ','も可能

## □ 引数

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、式

連結指定 : セミコロン'; または カンマ','  
文末に付けると改行抑制される桁数 : #数値 または #-数値 の形式で指定する  
例:#3 、 #-3

## □ 説明

指定した文字列、数値をデータ通信設定したシリアルポートに出力します。

書式は PRINT 文と同じです。

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。

SPRINT 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。

シリアル通信は次の GPIO ピンを利用します。

PA9 : TX 送信

PA10 : RX 受信

シリアル通信の利用には次のコマンドが用意されています。

SOPEN : シリアル通信開始

SERITE : 1 バイト送信

SREAD : 1 バイト受信

SREADY : 受信データ有無の確認

SPRINT : 文字列出力 (PRINT と同等の出力)

SCLOSE : シリアル通信クローズ

(注意) シリアル通信をオープンした状態で GPIO コマンドにて PA9,PA10 に I/O 機能の割り当てを行った場合、正しい動作を行うことが出来ません。  
必ず SCLOSE にてシリアル通信をクローズしてからピンに機能を割り当て下さい。

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

## □ 利用例

シリアル通信経由でテキストメッセージを送信する

```
10 SOPEN "9600"  
20 SPRINT "Hello,TinyBASIC"  
30 SCLOSE
```

## 7.84 SWRITE シリアル通信 1 バイト出力

## □ 書式

SERITE 送信データ

## □ 引数

送信データ : 0~255 (1 バイトデータ、変数指定時は上位 8 ビットは無視)

## □ 説明

データ通信設定したシリアルポートに 1 バイト分のデータを出力します。

シリアル通信は次の GPIO ピンを利用します。

PA9 : TX 送信  
 PA10 : RX 受信

シリアル通信の利用には次のコマンドが用意されています。

SMODE : GPIO シリアルポート、USB シリアルポート機能切り替え  
 SOPEN : シリアル通信開始  
 SERITE : 1 バイト送信  
 SPREAD : 1 バイト受信  
 SREADY : 受信データ有無の確認  
 SPRINT : 文字列出力(PRINT と同等の出力)  
 SCLOSE : シリアル通信クローズ

(注意) シリアル通信をオープンした状態で GPIO コマンドにて PA9,PA10 に I/O 機能の割り当てを行った場合、正しい動作を行うことが出来ません。  
 必ず SCLOSE にてシリアル通信をクローズしてからピンに機能を割り当て下さい。

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

## □ 利用例

シリアルポートに'A'から'Z'の文字コードを出力する

```
10 SOPEN "115200"
20 FOR C=ASC("A") TO ASC("Z")
30 SWRITE C
40 NEXT C
50 SCLOSE
```



## 7.85 SREADY シリアル通信データ受信データ確認 (数値関数)

## □ 書式

SREADY()

## □ 引数

なし

## □ 戻り値

0: 受信データなし

1: 受信データあり

## □ 説明

シリアルポートに受信可能なデータが着信しているかを確認します。

シリアルポートから SREAD 関数にてデータの受信を行う際に、事前にデータの有無を確認することが出来ます。

シリアル通信は次の GPIO ピンを利用します。

PA9 : TX 送信

PA10 : RX 受信

シリアル通信の利用には次のコマンドが用意されています。

SMODE	: GPIO シリアルポート、USB シリアルポート機能切り替え
SOPEN	: シリアル通信開始
SERITE	: 1 バイト送信
SREAD	: 1 バイト受信
SREADY	: 受信データ有無の確認
SPRINT	: 文字列出力 (PRINT と同等の出力)
SCLOSE	: シリアル通信クローズ

(注意) シリアル通信をオープンした状態で GPIO コマンドにて PA9,PA10 に I/O 機能の割り当てを行った場合、正しい動作を行うことが出来ません。必ず SCLOSE にてシリアル通信をクローズしてからピンに機能を割り当て下さい。

## □ 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

受信した文字列を画面に表示する

(".":終了、\$OD:改行、それ以外の制御文字は無視)

```

10 SOPEN "115200"
20 IF SREADY()=0 GOTO 20
30 D=SREAD()
40 IF D=ASC(".") GOTO 80
50 IF D=$OD ?
60 IF D>=32 ? CHR$(D);
70 GOTO 20
80 SCLOSE
90 ? :? "Done."
```

## 7.86 SREAD シリアル通信 1 バイト受信 (数値関数)

## □ 書式

SREAD()

## □ 引数

なし

## □ 戻り値

受信データ : 0 ~ 255、データなしの場合は-1

## □ 説明

データ通信設定したシリアルポートから1バイトデータを受信します。

データが無い場合は-1 を返します。データが無い場合の受信待ちは行いません。

シリアル通信は次の GPIO ピンを利用します。

PA9 : TX 送信  
PA10 : RX 受信

シリアル通信の利用には次のコマンドが用意されています。

SMODE : GPIO シリアルポート、USB シリアルポート機能切り替え  
SOPEN : シリアル通信開始  
SERITE : 1 バイト送信  
SREAD : 1 バイト受信  
SREADY : 受信データ有無の確認  
SPRINT : 文字列出力 (PRINT と同等の出力)  
SCLOSE : シリアル通信クローズ

(注意) シリアル通信をオープンした状態で GPIO コマンドにて PA9,PA10 に I/O 機能の割り当てを行った場合、正しい動作を行うことが出来ません。  
必ず SCLOSE にてシリアル通信をクローズしてからピンに機能を割り当て下さい。

## □ 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

受信した文字列を画面に表示する (".":終了、\$0D:改行、それ以外の制御文字は無視)

```
10 SOPEN "115200"
20 IF SREADY()=0 GOTO 20
30 D=SREAD()
40 IF D=ASC(".") GOTO 80
50 IF D=$0D ?
60 IF D>=32 ? CHR$(D);
70 GOTO 20
80 SCLOSE
90 ? :? "Done."
```

## 7.87 EEPFORMAT 仮想 EEPROM のフォーマット

## □ 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 のように利用するものです。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き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き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 250 ワード(250 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。0～32767 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 250 ワードの制約から、250 以内に抑える必要があります。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
```

## 7.88 EEPWRITE 仮想 EEPROM のへのデータ書き込み

## □ 書式

EEPWRITE アドレス,データ

## □ 引数

アドレス : 仮想 EEPROM 内アドレス 0 ~ 32767

データ : 書き込みデータ -32768 ~ 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 のように利用するものです。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き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 250 ワード (500 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。0 ~ 32767 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 250 ワードの制約から、250 以内に抑える必要があります。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
```

## 7.89 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 0 ~ 32767

## □ 戻り値

読みだした 2 バイトデータ (-32768 ~ 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。0 ~ 32767 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

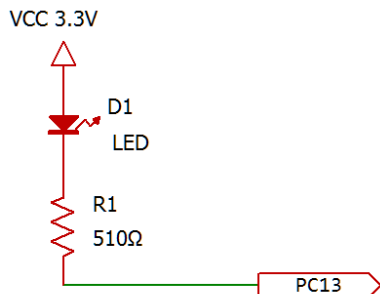
```
EEPWRITE 1234, 123
OK
?EEPREAD(1234)
123
OK
```

## 8. 応用プログラム(執筆中)

### 8.1 ボード上の LED の点滅

#### (1) ボード上の LED の点滅

BluePill ボードの LED は PC13 に接続しています。下記は LED 周りの回路図です。



回路図から PC13 を OV (LOW) にすることで、LED 点灯することが出来ます。

逆に HIGH (3.3V) にすると LED を消灯出来ます。

LED を 1 秒周期で点滅 (0.5 秒間点灯後、0.5 秒間消灯) させてみましょう。

プログラムは次のようになります。

```
1 'L 点滅
10 GPIO PC13, OUTPUT
20 OUT PC13, LOW
30 WAIT 500
40 OUT PC13, HIGH
50 WAIT 500
60 GOTO 20
```

冗長性のある部分を共通化した修正です。繰り返し部分は 1 行短くなりました。

GPIO コマンドも省略形での記述としました。

```
1 'L 点滅 2
10 L=LOW
20 PC13, OUTPUT
30 OUT PC13, L
40 L=!L
50 WAIT 500
60 GOTO 30
```

## 8.2 追加 LED の点滅

## 8.3 PWM による LED 輝度の制御

## 8.4 スライド抵抗をつかった LED の輝度調節

## 8.5 シフトレジスタ 74HC595 を使った 8 個の LED の制御

PWMによるサーボモータの制御

## 8.6 I2C インタフェースの利用（基本編）

I2C バスを利用するには、2.2k~10kΩ程度でプルアップする必要があります。  
（以降作成中）

（以降作成中）

## 8.7 シリアル通信

## 8.8 OLED(SSD1306)ディスプレイの利用