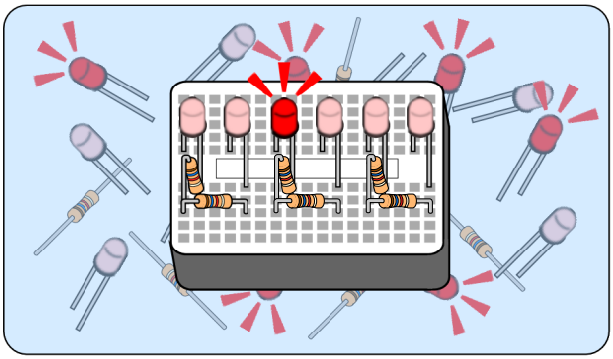
IchigoJam 電子工作パーツセット
Lチカ実験セット
LED-jamP

LEDs Flasher Parts Set

IchigoJam 0.9.7 / 1.0.1対応

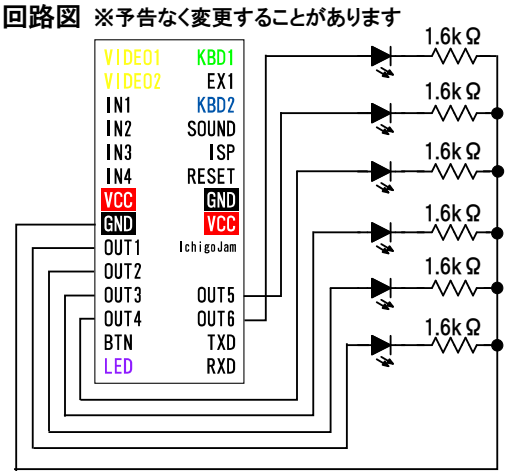
0 概要



マイコン電子工作の定番、LEDを6個使用したLEDの点滅実験セットです。
BASICプログラムからLEDの点滅を自在に操ってみましょう。

本キットのプログラムはIchigoJam バージョン0.9.7及びバージョン1.0.1で動作確認しています。

回路図 ※予告なく変更することがあります



部品表 ※予告なく変更することがあります

	品名/型番/値	数量	備考		品名/型番/値	数量	備考
1	LED 赤	6		3	ジャンプワイヤ 10cm	9	
2	抵抗 1.6kΩ	6	茶青赤金の色帯	4	ブレッドボード	1	

【企画・販売元】 オーディオ・マイコン・メカトロ・電子パーツ
デジット
年中無休・営業時間: AM11:00～PM8:00
〒556-0005 大阪市浪速区日本橋4-6-7
[TEL]06-6644-4555 / [FAX]06-6644-1744
[HP]http://digit.kyohritsu.com
[Blog]http://blog.digit-parts.com [Twitter]@0666444555

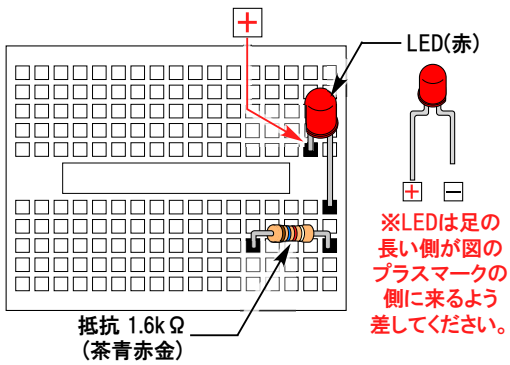
【販売窓口】
■ シリコンハウス (大阪・日本橋店舗) 06-6644-4446
■ デジット (大阪・日本橋店舗) 06-6644-4555
■ 法人営業部 (B2B/学校・官公庁) 06-6646-0707
■ 通販営業部 (インターネット通販) 06-6644-6116

KYOHRITSU 共立電子産業株式会社 共立電子 検索

1 組み立てかた

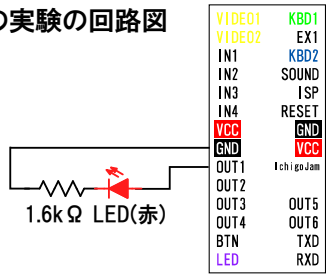
最初の実験はLED1個で行います。

①

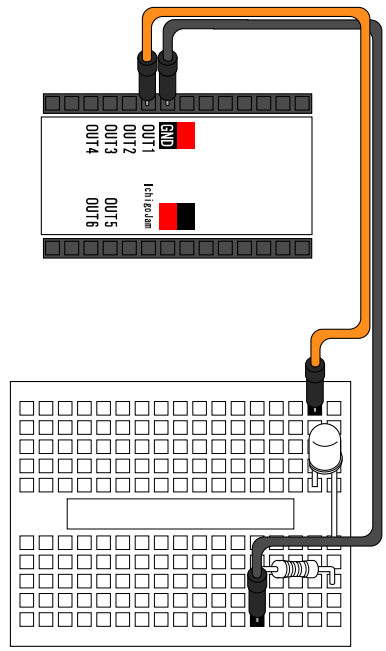


※LEDは足の長い側が図のプラスマークの側に来るよう差してください。

最初の実験の回路図



② IchigoJamとの接続



2 動かしてみよう

組み立てた回路のチェックを兼ねて、早速動かしてみましょう。

IchigoJamにはBASICのコマンドを即時実行する機能があります。この機能を使用すると、組み立てた回路の動作を手軽に試せます。

```
OUT 1,1↵
OK
```

行番号なしでBASICの文(コマンド)を入力し、「ENTER↵」キーを押すと、入力した文がすぐに実行されます。

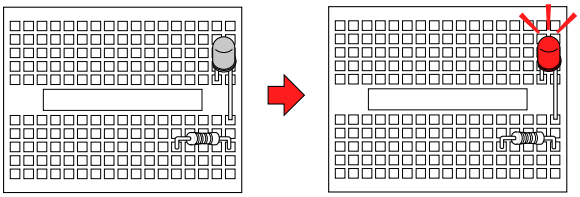
実験① LEDを点灯させてみよう

最初に、ブレッドボード上のLEDを点灯させてみます。次のように入力し、「ENTER↵」キーを押してください。ブレッドボード上のLEDが点灯します。

```
OUT 1,1↵
OK
```

正しく入力したのにLEDが点灯しない場合は、LEDの取り付け向きなどブレッドボードの組み立てをチェックしてください。

LEDが点灯します



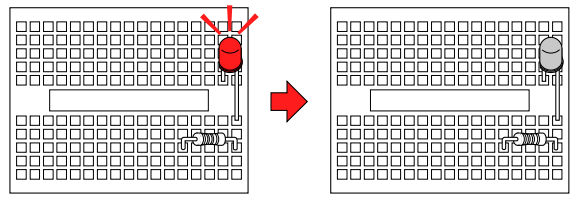
実験② LEDを消灯させてみよう

次に、点灯しているLEDを消灯させてみます。ブレッドボード上のLEDが点灯した状態で次のように入力し、「ENTER↵」キーを押してください。

ブレッドボード上のLEDが消灯します。

```
OUT 1,0↵
OK
```

LEDが消灯します



消灯したLEDを再び点灯させたい場合は、再び「OUT 1,1↵」と入力するとLEDを点灯させることができます。

何度かLEDを点灯させたり消灯させたりして、感触をつかんでください。

実験③ LEDをつけて消す動作を自動化しよう

先の実験では手動でBASICの文を打ち込み、LEDを点灯させたり消灯させたりしましたが、次にLEDを点灯させて消灯させる動作を自動化してみます。

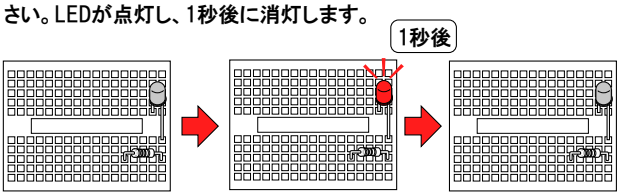
LEDが点灯している場合は「OUT 1,0↵」でLEDを消灯させてから、次のように入力してください。

```
10 OUT 1,1↵
20 WAIT 60↵
30 OUT 1,0↵
```

先の実験のときと違って、行の頭に数字(行番号)を付けて入力することに注意してください。

前の実験のときのように1行入力してもLEDが点灯したり消灯したりしませんが、それで正常です。

入力が終わったら、今度は行番号なしで「RUN↵」と入力してください。LEDが点灯し、1秒後に消灯します。



モニタ画面に「OK」が出てから再び「RUN↵」を入力すると何度でも(IchigoJamの電源を切らない限り)繰り返して実行できます。試してください。

「RUN↵」を入力したとき「Syntax Error in xx」と表示されて途中で止まってしまった場合は止まった行の入力が間違っています。
確認して修正してください。

プログラムのリストは「LIST↵」で見ることができます。

プログラムを1行だけ修正するには

プログラムを1行だけ修正したいときは、修正したい行番号で文を新しく書き、最後に[ENTER↵]キーを押すと新しく書いた文で置き換わります。

もとのプログラム

```
20 WAIT 60
```

新しく文を書きます

```
20 WAIT 30↵
```

[ENTER]キーを押すと20行が新しく書いた文で置き換わります

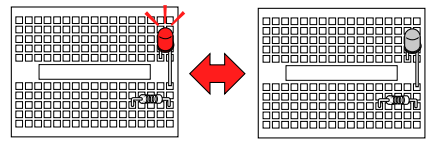
実験④ LEDを自動点滅させよう

実験③では1回LEDを点灯させて消灯したら実行が終わってしまいましたが、今度は繰り返し点滅させてみましょう。

```
10 OUT 1,1↵
20 WAIT 60↵
30 OUT 1,0↵
40 WAIT 60↵
50 GOTO 10↵
```

次のように入力し、「RUN↵」と入力してください。

LEDが1秒ごとに点滅します。キーボードの「ESC」キーを押して実行を中断するまでずっと点滅を繰り返します。



キーワード

- OUT文
- WAIT文
- GOTO文

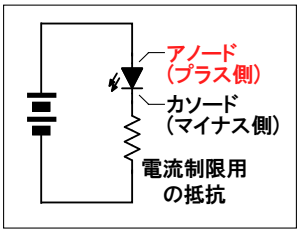
回路と部品の説明

(1) LEDの使い方

LEDは「発光ダイオード」(Light Emitting Diode)の略称で、電子工作の定番部品としておなじみです。使ったことがある人も多いのではないでしょうか?

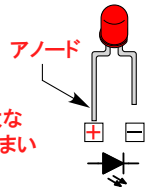
その名の通り、**電流を流すと光るダイオード**で、トランジスタやICと同じ半導体の仲間です。

発光色は赤、橙、黄、緑、青、白など色々で、中には「ピンク色」や「紫色」といった変り種もあります。



LEDは一方向にしか電流を流しませんので、アノードが回路のプラス側になるように接続します。

キットに入っているLEDを見ると足の長さに長い短いの区別があります。足の長い側がアノード(プラス)側です。



LEDを使用するときは、LEDに流れる電流を制限するための抵抗を直列に入れます。

電流制限用の抵抗を入れないとLEDに過大な電流が流れます。最悪の場合LEDを壊してしまいます。注意しましょう。

直列抵抗の値は、次のように求めることができます。

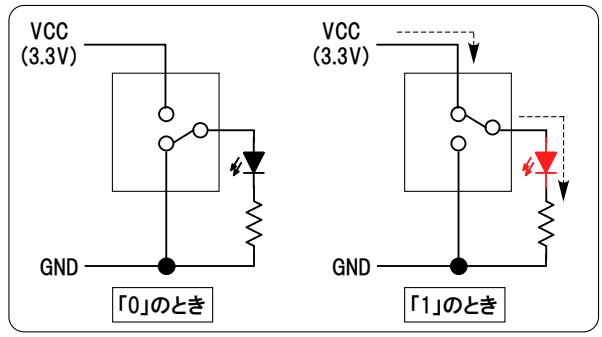
$$\text{直列抵抗の値}(\Omega) = \frac{\text{電源電圧} - \text{LED順方向電圧(V)}}{\text{LEDに流す電流(A)}}$$

LEDの順方向電圧はLEDの色により異なり、赤、黄、黄緑のLEDなら約1.8V～2V程度、純緑、青、白のLEDの場合は約3Vから3.6V程度です。LEDに流す電流は数mA程度が普通です。

(2) IchigoJamの出力ポートのはたらき

IchigoJamにはOUT1～OUT6の6本の出力ポートがあります。出力ポートはBASICのOUT文で操作できます。

出力ポートのはたらきは下の図のように、VCC(3.3V)とGND(0V)の間で切り替わる切り替えスイッチを想像するとイメージがつかみやすいと思います。
(実際にも半導体素子による切り替えスイッチになっています)



OUT文で出力ポートを「0」にすると、IchigoJam内部の切り替えスイッチは「GND(0V)」側に切り替わります。本キットのように出力ポートとGND間にLEDを接続してある場合、LEDには電流は流れません(消灯しています)

出力ポートを「1」にすると、内部の切り替えスイッチは「VCC(3.3V)」側に切り替わります。するとLEDに電流が流れ、LEDが点灯します。

プログラムの説明

プログラムとは何か?

実験①と②のときはLEDの点灯状態を変化させるため、手動でBASICの命令を入力しました。

```
OUT 1,1↵
OK
```

行番号なしでBASICの命令を入力し、「ENTER↵」キーを押すと、ただちに入力した命令が実行されます。

実験③と④のときは頭に行番号をつけてBASICの命令を書きました。BASICの命令を入力したとき、ただちには実行されず、「RUN↵」と入力してはじめて命令が実行されました。

これは実験③と④で**「プログラム」**を書いたためです。「プログラム」とは、コンピュータ(IchigoJamも立派なコンピュータです)の命令をある決まりごと(文法)にしたがって書いたものです。

行番号をつけてBASICの命令を入力すると、入力した命令はすぐには実行されず、IchigoJam内部のプログラム用のメモリに置かれます。

プログラムを書いたあと、行番号なしで「RUN↵」を入力すると、プログラムに書いてあることを、行番号の若い順に**「プログラムに書いてある通り」**に自動で実行します。

RUNは「入力したプログラムを実行せよ」と命令する文です。

コンピュータの命令と書き方の決まりごとのことをまとめて「プログラミング言語」といいます。プログラミング言語にはいろいろありますが、IchigoJamの場合は、「BASIC」というプログラミング言語でプログラムを書きます。

```
(例) 50 GOTO 10
     行番号 命令
```

「プログラムの10行にジャンプせよ」という文です。

BASICでは、まず行番号を書き、行番号に続いて命令(文)を書くという決まり(文法)があります。

実行しようとしているプログラムがBASICの文法上間違っている場合は「Syntax Error in xx」(xxはエラーのある文の行番号)というエラーメッセージを出して停止します。

プログラムのセーブとロードのしかた

作成したプログラムをSAVEコマンドで保存することができます。

```
      ⋮
130 GOTO 30

SAVE0 ↵
```

プログラムをSAVEしないと、IchigoJamの電源をOFFにしたときに作成したプログラムが消えてしまいます。

上の例のようにSAVEコマンドの後ろにプログラム番号を指定して実行すると、指定した番号に保存されます。

プログラム番号はIchigoJam本体に保存するとき0～3(バージョン0.9.7では0～2)、外部のEEPROMカセットに保存するときは100～131番になります。

! SAVEすると前に入っていたプログラムは上書きされます(消えてなくなります)ので十分注意してください。

```
LOAD0 ↵
Loaded xxxbyte
OK
```

SAVEコマンドで保存したプログラムは、LOADコマンドでファイル番号を指定すると呼び出すことができます。

☆プログラム番号を省略したとき

プログラム番号を省略してプログラムをロード/セーブすることもできます。このときどの番号のプログラムが対象になるかはIchigoJamのバージョンによって次のように異なります。注意してください。

◎ IchigoJam バージョン1.0.1の場合

SAVEコマンドのときは最後に実行したLOADまたはSAVEコマンドで指定した番号に上書き保存されます。

LOADコマンドのときは最後に実行したLOADまたはSAVEコマンドで指定した番号からプログラムを呼び出します。

(※電源をONしてはじめてLOADコマンドやSAVEコマンドを実行した場合、番号を省略すると0番が対象になります)

◎ IchigoJam バージョン0.9.7の場合

SAVEコマンドのときは0番に上書き保存されます。

LOADコマンドのときは0番からプログラムを呼び出します。

EEPROMカセットを使用する場合

EEPROMカセットからプログラムをロードし、その後EEPROMカセットを取り外した場合、番号を省略してセーブしようとするとファイルエラーになりセーブできません。

EEPROMカセットを取り外した後の初回のセーブはIchigoJam本体のプログラム番号を指定してIchigoJam本体にセーブしてください。

キーワードの説明

OUT文

OUT文は、IchigoJamの出力端子(OUT1～OUT6)を操作する命令です。「OUT」に続く最初の数字(1～6)で操作する出力端子(OUT1～OUT6)を選び、2番目の数字(0または1)で出力端子の状態を指定します。

(例) OUT 1,1

上の使用例では、OUT1端子に「1」を出力します。「1」を出力すると、OUT端子が約3.3V(「H」レベル)になります。「0」を出力した場合はOUT端子が約0V(「L」レベル)になります。

OUT文を実行したとき、OUT文で選んでいないOUT端子の状態は変化しません。

「OUT」の後に数字を1つしか書かない場合は、OUT1～OUT6端子の状態をまとめて変更します。

(例) OUT 15

この場合は「OUT」に続いて0～63の数字を指定します。

IchigoJam バージョン1.0.1では数の表現として、10進表記以外に16進表記と2進表記も使用できます。用途に合わせて使用するとプログラムがわかりやすくなり便利です。

数字の頭に何も指定しない場合は10進表記です。
数字の頭に「#」記号を指定した場合は16進表記、「`」(バッククオート)記号を指定した場合は2進表記です。

16進表記は「0～9の数字」と「A～Fのアルファベット」で数を表現します。

2進表記は「0と1の数字」の組み合わせで数を表現します。

WAIT文

WAIT文は、プログラムの実行を指定した時間だけ一時停止させる文です。一時停止させる時間は1/60秒(約16.7ミリ秒)単位で指定します。

(例) WAIT 60

この例では、プログラムの実行を1秒(1/60秒 x 60 = 1秒)だけ一時停止させます。

一時停止させる時間として、定数の代わりに変数を含んだ式を指定することもできます。

(例) WAIT 29-M

この例のように変数を含んだ式を指定すると、待ち時間を可変にすることも可能です。

GOTO文

GOTO文は、指定した行番号の行にジャンプする文です。

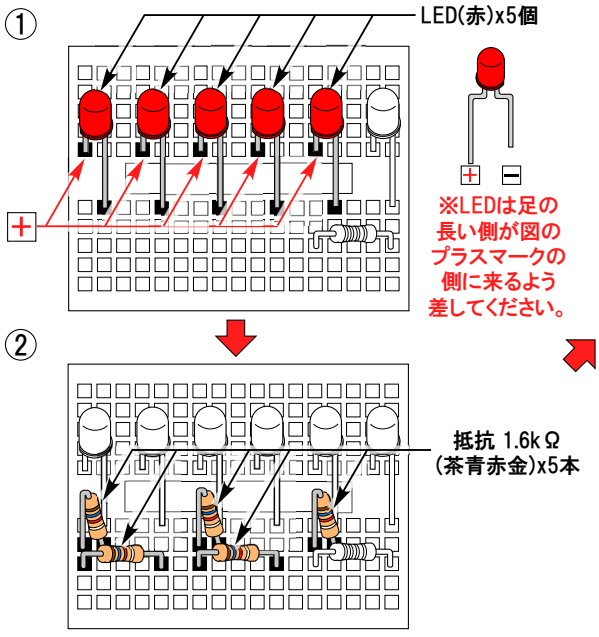
(例) 200 GOTO 20

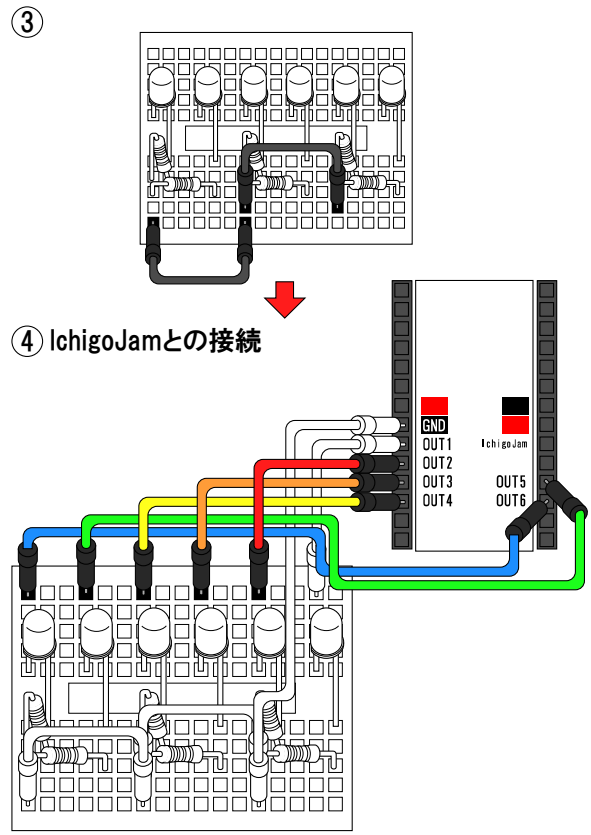
上の例では、プログラムの20行にジャンプします。「GOTO」に続けて、ジャンプ先の行番号を書きます。

BASICのプログラムは行番号の若い順に順次実行されますが、GOTO文はプログラムの流れを変更する文です。(プログラムの流れを変更する文には他にGOSUB文、RETURN文などがあります)

点滅させるLEDを増やそう（入門プログラム）

LED1個の点滅で感触をつかんだところで、点滅させるLEDを増やしましょう。ブレッドボードに次のように部品と配線を追加します。





プログラムリスト(その1)

```
1 REM LED ﾁｶﾁｶ
10 OUT 0:PRINT "LED ﾁｶﾁｶ"
20 OUT 6,0:OUT 1,1:PRINT "LED1 : ON"
30 WAIT 30
40 OUT 1,0:OUT 2,1:PRINT "LED2 : ON"
50 WAIT 30
60 OUT 2,0:OUT 3,1:PRINT "LED3 : ON"
70 WAIT 30
80 OUT 3,0:OUT 4,1:PRINT "LED4 : ON"
90 WAIT 30
100 OUT 4,0:OUT 5,1:PRINT "LED5 : ON"
110 WAIT 30
120 OUT 5,0:OUT 6,1:PRINT "LED6 : ON"
130 WAIT 30
140 GOTO 20
```

LED順送りの例

プログラムリスト(その2)

```
1 REM LED ﾁｶﾁｶ
10 OUT 0:PRINT "LED ﾁｶﾁｶ"
20 OUT 1,0:OUT 6,1:PRINT "LED6 : ON"
30 WAIT 30
40 OUT 6,0:OUT 5,1:PRINT "LED5 : ON"
50 WAIT 30
60 OUT 5,0:OUT 4,1:PRINT "LED4 : ON"
70 WAIT 30
80 OUT 4,0:OUT 3,1:PRINT "LED3 : ON"
90 WAIT 30
100 OUT 3,0:OUT 2,1:PRINT "LED5 : ON"
110 WAIT 30
120 OUT 2,0:OUT 1,1:PRINT "LED6 : ON"
130 WAIT 30
140 GOTO 20
```

LED逆送りの例

プログラムの説明

プログラムをRUNすると、ブレッドボード上のLEDが右端(OUT1)から左端(OUT6)まで順送りに点灯します。

6個のLEDが順番に点灯したでしょうか? もし点灯しないLEDがある場合はブレッドボード上の組み立て(特にLEDの逆差し)、IchigoJamとの間の配線をチェックしてください。

前に点灯してあるLEDを消灯して、次のLEDを点灯させる操作をLEDの数だけ繰り返しています。

ブレッドボード上のLEDを上のプログラムと逆順に点灯させるプログラムは右の「プログラムリスト(その2)」です。

キーワード

- REM文(コメント)
- PRINT文の使い方
- コロン「:」記号

キーワードの説明

REM文(コメント) (1行)

REM文は、プログラムにコメント(メモ)を入れるために使います。

(例) REM ｽｲｯﾁ ｦ ｵｽﾏﾃﾞ ﾏﾂ

「REM」の後ろはプログラムの実行時は無視されます。

PRINT文の使い方(10行)

PRINT文は変数の値(計算結果)やメッセージを表示するのに使用します。
基本的な使用方法は次のとおりです。

(例) PRINT K

この例では変数Kの値を画面に表示します。

(例) PRINT ﾞﾎﾝｼﾞﾂﾊ ｾｲﾃﾝﾅﾘﾞ

この例では「ﾎﾝｼﾞﾂﾊ ｾｲﾃﾝﾅﾘ」というメッセージの文字列を画面に表示します。文字列はダブルクオーテーション「"」で囲みます。

PRINT文で表示させる項目をセミコロン「;」記号で区切って続けて書くと、複数の項目を1行に表示させることができます。

(例) PRINT W/10;".";W%10;"V"

この例ではまず変数Wの値を10で割った値(小数点以下は切捨てとなります)を表示し、そのあとに小数点記号(ピリオド「.」記号)を表示し、そのあとに変数Wの値を10で割った余りを表示、最後に文字「V」を表示します。

PRINT文の後ろに何も指定しない場合は、何も表示せず、改行のみ行います。

コロン(「:」)記号 (10行)

BASICのプログラムでは、1行に複数の文を書くことができます。1行に複数の文を書くときは、文と文の間をコロン(「:」)記号で区切って書きます。

(例) V=ANA():PRINT V

参考 (プログラム中のカタカナについて)

IchigoJamにPCを接続し、PC上の端末ソフトからプログラムを入力して使用している人も多いと思います。

この場合、端末ソフトの設定によってはREM文やPRINT文中で使用しているカタカナが文字化けして正常に表示されなかったり、「Syntax Error」になってうまく実行できなかったりすることがあります。

これはIchigoJamのカタカナの文字コードとPC側の文字コードが一致しないため、端末ソフト側で別の文字になったり「不明の文字」として「?」に置き換えられたりするために起こります。

端末ソフトの設定で文字コードが「UTF-8」になっている場合は「JIS」または「Shift-JIS」に変更すると比較的うまくいくようです。それでも直らない場合はカタカナの部分をローマ字表記にするなどして対策してください。

※本キットのプログラムはIchigoJamに接続したキーボードから直接入力する前提で作成しています。

---

4 上級プログラム

入門プログラムで行ったLEDの順送りをもう少しスマートな方法でやってみます。

プログラムリスト

```
1 REM LED ﾁｶﾁｶ
10 PRINT "LED ﾁｶﾁｶ"
20 I=1
30 OUT I:PRINT I
40 WAIT 5
50 I=I<<1
60 IF I<64 THEN GOTO 30
70 I=32
80 OUT I:PRINT I
90 WAIT 5
100 I=I>>1
110 IF I>1 THEN GOTO 80
120 GOTO 20
```

キーワード

- 変数I
- IF文と条件判断
- ビットシフト演算子

プログラムの説明

(1) プログラムの動作について

プログラムをRUNすると、ブレッドボード上のLEDが右から順送りに点灯し、左端まで来たら右方向に逆送りに点灯します。

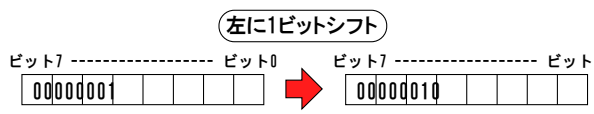
プログラム中でOUT1～OUT6に出力する値を変数として用意しておき、OUT文で一度に出力するようにしています。

(2) ビットのシフトについて

IchigoJamのBASICには変数の値を指定したビット数だけ左右にシフトする「シフト演算子」(「<<」「>>」)があります。

シフト演算子についてはキーワードの説明を見てください。

IchigoJamのBASICで扱える数値は-32768～+32767の範囲の整数(16ビット符号つき)ですが、シフトするとどうなるのかわかりやすいように、8ビットの整数を左に1ビットシフトする例で説明します。



今、シフト前の変数の中身が上のように一番下のビット(ビット0)のみ「1」、ほかのビットは全て「0」だとします。

この値を左に1ビットシフトすると、ビット6の内容がビット7、ビット5の内容がビット6･･･ビット0の内容がビット1にというように、ビットの内容が左に1個ずつずれます。その結果、「1」のビットはビット1に移動します。

シフトした結果はみだしてしまったビットは破棄されます
(消えてなくなります)

キーワードの説明

変数 I (20行)

変数とは、プログラム中で使用する数値を入れるための「箱」です。箱を区別するために、箱には「名前」をつけます。(これを「変数名」といいます)

IchigoJamのBASICでは、変数名は「A」から「Z」までのアルファベット1文字ですので、全部で26個の変数が使用できます。

コピー(代入)

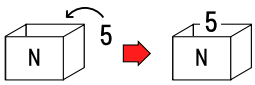
箱に式の計算結果や別の箱の中身を入れる(コピーする)ことを「代入」といいます。
(※コピー元の箱の中身は変化しません)

(参考 :BASICでは変数を使用する前に「 この名前の変数を使用します」と宣言する必要はありません)

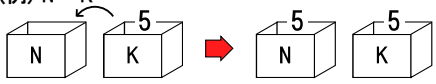
(参考2 :IchigoJamのBASICで扱える数値の範囲は、-32768から32767までの整数です)

変数に数値を代入するには、次のように書きます。

(例) N = 5



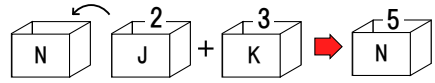
(例) N = K



※コピー元の変数(K)の値は変化しません

足し算や引き算などの計算にも使用します。

(例) N = J + K



IF文と条件判断 (60行)

IF文は、条件によって処理内容を変えたいときに使います。

(例) IF BTN()=1 THEN GOTO 50

条件式　条件式が真のとき実行する文

この例では、BTN()関数が返す値が1と等しいかどうか比較し、もし真ならば(等しければ)「THEN」以下の文を実行します。(偽ならば(等しくなければ)次の行に移ります)

条件式が偽のときに実行する文を書く必要があるときは、条件式が真のときに実行する文に続けて「ELSE」と書き、その後ろに条件式が偽のときに実行する文を書きます。

(例) IF A<B THEN MAX=B ELSE MAX=A

条件式　条件式が真のとき実行する文　条件式が偽のとき実行する文

この例ではAがBより小さいかどうか比較し、もし真ならば(A<Bならば)MAXにBを、偽ならば(A>=Bならば)MAXにAを代入します。

ビットシフト演算子 (50行)

IchigoJamのBASICには、数値を指定したビット数だけ左右にシフトするビットシフト演算子があります。

「<<」は左シフト、「>>」は右シフトを表します。

(例) I=I<<1

上の例は、変数Iの値を左へ1ビットシフトします。同様に

(例) I=I>>1

は、変数Iの値を右へ1ビットシフトします。

シフトする回数は「<<」または「>>」に続けて指定します。

ビットをシフトすると、変数のビットの状態は次のようになります。

<左1シフト>

<右1シフト>

※シフトした結果、変数からはみだしたビットは破棄されます。

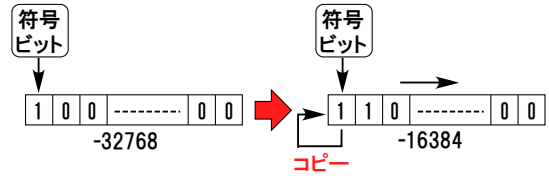
※右シフトの場合と左シフトの場合で変数のプラスマイナスの符号の扱いが異なり、左シフトの場合変数の符号が変化する場合があります。
(本キットのプログラムではOUT1～OUT6に対応する下位6ビットしか使っていないため、問題はありません)

参考 (ビットをシフトすると符号はどうなる?)

1. 右シフトの場合

変数の一番上のビットがシフト後の変数の一番上のビットにコピーされるので、シフト前後で符号は変わりません。

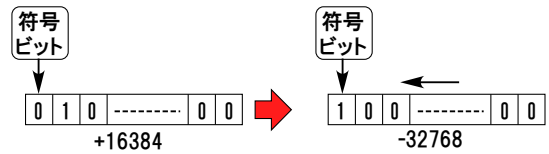
<右1シフト>



2. 左シフトの場合

変数の最上位ビットはシフトしたときに破棄される(消えてなくなる)ため、シフト前後で符号が変化する場合があります。

<左1シフト>



IchigoJamで扱える変数の値は-32768～+32767までの整数です。変数の一番上のビットが符号ビットで、0のときプラスの数、1のときマイナスの数になります。

上の例では、シフト前の一番上のビットが「0」なのでプラスの値ですが、左シフトした結果シフト前に上から2番目にあった「1」のビットが一番上に来たため、一番上のビットが「1」になり符号が変化してしまいます。
(演算結果がオーバーフローしますがエラーにはなりません。注意してください)