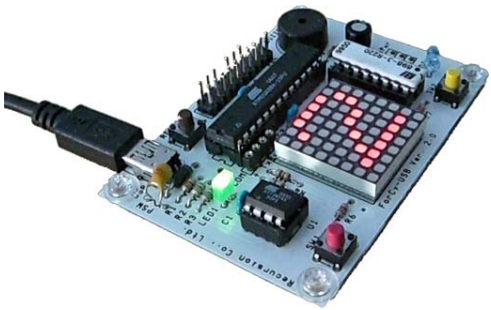
- 内蔵コンパイラ ForCy で学ぶ -
組み込みプログラミング



ForCy – USB 2.0

初版	2007/08/03
第 2 版	2009/02/03

有限会社リカージョン

- 内蔵コンパイラ ForCy で学ぶ -

組み込みプログラミング

－ 目次 －

FORTH C + 4 / :ForCy

はじめに

ForCy-USB は、マイコン制御、組み込みプログラミングを手軽に体験していただくための教材です。煩雑な準備や難解な予備知識なしに「プログラム制御」の雰囲気を楽しむことができます。

昨今の電子技術の急激な進歩によって社会生活のいたるところにコンピュータが浸透してきましたが、それとともに、高性能パソコンを操作してインターネットやワープロを使うだけでは、コンピュータ本来の姿を理解することが難しくなってきています。本教材では、手順を記述して計算やデジタル信号の入出力を行うプログラム制御の考え方を、家電製品や産業機器に使われる安価なマイクロコントローラを使って学びます。この小さな制御の積み重ねが今日のコンピュータ技術であることを実感してください。

１．特徴

マイクロコントローラ（マイコン）でプログラム制御するには、パソコンでアプリケーションを作る以上に面倒な準備と手続きが必要になります。パソコン上に開発用ソフトウェアをインストールし、その使い方やプログラム作法を習得しなければなりません。マイコンチップにプログラムを書き込む専用の装置も必要です。

ForCy-USB では、こうしたプログラム開発に必要なしくみを小さくしてあらかじめマイコンに組み込んであります。パソコンに USB 接続し、メモ帳で記述したプログラムをハイパーターミナルへコピー＆ペーストするだけで動作します。

プログラムには、小規模マイコンに内蔵できるよう開発した ForCy という簡易言語を使います。小型ながら、プロ向けの高水準言語に匹敵する制御構文を備えており、「処理の流れ」を自在に記述することができます。また、より本格的な開発技術の学習にも使えるよう、C 言語やアセンブラ環境で作成した実行形式のプログラムを起動するしくみも備えています。

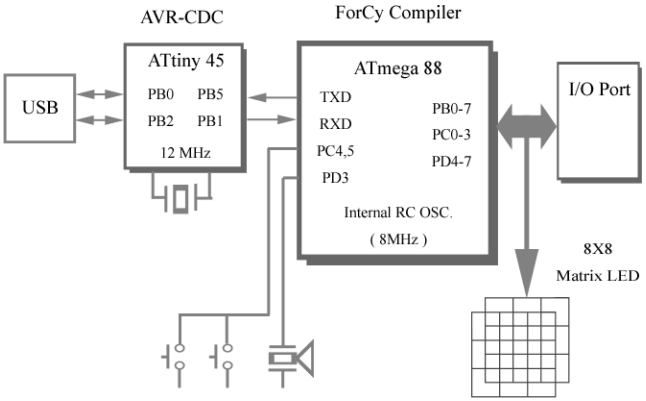
２．基本構成

コンピュータ演算の中枢となる部分はマイクロプロセッサと呼ばれ、これにメモリや周辺装置を接続すると、コンピュータ・システムになります。家電製品などでは、メモリや周辺機能をも同居させたマイクロコントローラが多く使われています。製品ごとに異なるボタンやセンサ、表示パネルなどを接続すれば、電子制御の部分ができあがります。

ForCy-USB には家電製品や産業機器に組み込まれる８ビットのマイクロコントローラ（Atmel 社 ATmega88）が使われています。搭載メモリの少ない数百円のチップですが、２進数８桁の演算を１秒間に８００万回も処理する能力があります。これは初期のデスクトップ PC を凌駕する性能です。

このマイコンチップのなかに、パソコンと通信し、送り込まれたプログラムテキストをマイコンが扱いやすいよう変換して実行するしくみを入れてあります。プログラムは、人がコンピュータを動かすために記述した独特の文章表現ですが、通常はこれをコンピュータがより効率的に処理できる形式に変換してから実行します。ForCy では、プログラムテキストを予め中間コードに変換（コンパイル）しておき、そのコードに応じた小さな処理をコンピュータが解釈実行（インタープリタ）する方式になっています。なお、少々ややこしいですが、このコンパイル処理もマイコン上のプログラムとして実行されています。

ForCy-USB 基板には、PC で作成したプログラムテキストを USB 通信でやりとりするためにもうひとつのマイコン（ATtiny45、小さいほうの IC）を使っています。また、ドットマトリクス LED と圧電ブザー、押しボタンスイッチでさまざまなプログラム制御を実験することができます。



３．インストール

ForCy-USB は単独で動作可能なコンピュータ・システムですが、プログラムテキストの編集にパソコンを使います。

パソコン：　　Windows XP/Vista。USB ポートがあるもの。

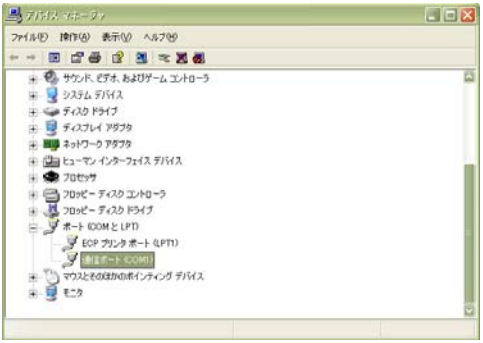
接続ケーブル：　USB ケーブル(A-B)

Windows XP/Vista

(1) Web サイト（http://www.recursion.jp/forcy-usb/）から INF ファイル(avrcdc_inf.zip)とサンプルプログラム（sample.zip）をダウンロードし解凍（右クリックですべて展開）します。

(2) PC にケーブルをつなぎ、基板を接続すると、ドライバ選択のダイアログが表示されます。ここで、ネット接続や自動検索せず、検索場所として展開した INF ファイルのフォルダ（/inf/bulk/xp2k または /inf/bulk/vista)を指定します。警告が出たら承認してください。USB ドライバがロードされます。

(3) 仮想 COM ポートが正しく生成されているか確認します(コントロールパネル／システム／ハードウェア／デバイスマネージャ／ポート（COM と LPT)）。 ここで COM ポートの番号を覚えておきます。

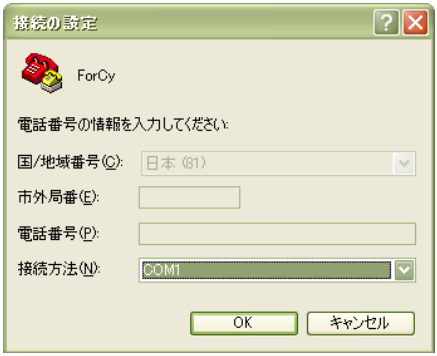


スタートメニューのプログラムから、アクセサリ／通信／ハイパーターミナルを選びます。「新しい接続」で、適当な名前を入れてください。

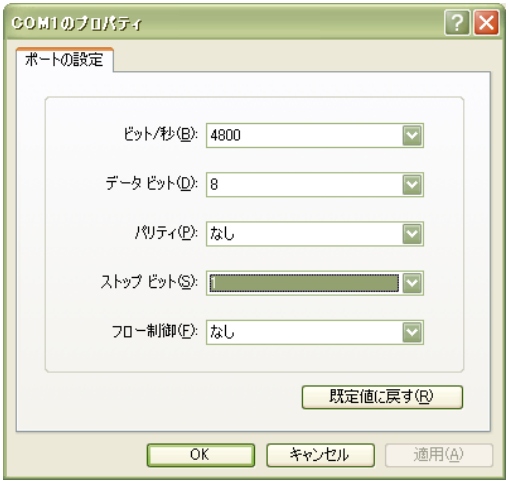
（ハイパーターミナルが見つからないときは、コントロールパネル／プログラムの追加と削除／Windows コンポーネントの追加、でアクセサリのハイパーターミナルを追加します。）

Vista にはハイパーターミナルが付属しません。弊社の HypoTerminal、もしくは TeraTerm をお使いください。TeraTerm では文字送信間隔を設定するか、ファイル転送機能を用います。

「接続の設定」で、先の COM ポート番号を指定します。



つぎに、「COM？のプロパティ」で、4800 bps、8 ビット、パリティなし、ストップビット 1、フロー制御なし、 に設定します。



（ここをあとから再設定するときは、メニュー「通信」を「切断」してからメニュー「ファイル」－「プロパティ」－「接続の設定」－「モデムの構成」を選んでください。）

これでターミナルのウィンドウが開きますが、ここでメニュー「ファイル」－「プロパティ」－「設定」でファンクションキーを「Windows キー」にしておけば Ctrl-V でウィンドウにペーストできます。

最後にメニュー「通信」で「電話」を選びます。通信に成功すると、

```
.......
```

のように、１秒ごとに「. 」が表示されます。RESET ボタンを押すと

```
                /*-- ForCy 1.00 by O.Tamura --*/
.......
```

が表示されます。Enter キーを入力すると改行することを確認してください。

Macintosh OS 10.2～10.4

Web サイトからサンプルプログラム（sample.zip）をダウンロードします。Mac では、USB 接続するだけでシステムに識別されますので、アップルメニューの「この Mac について」／（詳しい情報..）／ハードウェア／USB で USB 機器として認識（USB-232）されていることを確認します。Mac ではシリアル通信プログラムとして IntelMac も含め安定に動作するのはシェアウェアの ZTerm です。設定は、ボーレート 4800bps，データ長 8bit，パリティなし，１ストップビット，フロー制御なし。漢字コードは SJIS(Shift-JIS)、必要なら送受信の改行コード設定も調整してください。USB デバイス名は cu.usbmodem のあとに数字や文字が並んだものになるようです。Zterm では Settings/Text Pacing.. の Delay はどちらも 0 にしてください。テキストエディタは、OS 付属のテキストエディットで十分です。

４．操作手順

一秒ごとに'.'が表示されている状態で、ForCy-USB はつぎのコマンドを受け付けます。コマンドはすべて半角文字です。

コマンド	機能
#	コンパイルモードに入り、「>」を表示します。このあと、メニュー「転送」－「テキストファイルの送信」から、あるいはコピー&ペーストでソースプログラムを送信します。コンパイル中は、新規登録ワード名が順に表示されます。プログラム先頭行に「#」を入れておけば、このキー入力を省略できます。
Esc または ¥	コンパイル処理からコマンド待ち状態に戻ります。最後のワード（「main」など）が表示されたら、キー入力してコマンドモードに戻ってください。プログラム最終行を「¥」としておけば、自動的に戻ります。
g	ユーザ領域にある中間コードを実行します。プログラム最終行で「¥」のあとに入れておけば、自動的に開始します。
d	ユーザ領域にある中間コードをダンプします。先頭８バイトがヘッダで、２，３バイト目が中間コードのサイズです。
$	オートスタートを設定し、「！」を表示します。電源投入直後にユーザ領域にある中間コードを実行します。解除するときは、押しボタンスイッチ「SW2」を押したまま RESET ボタンを押します。

（使い方）

1．サンプルプログラム（hello.txt など）をメモ帳で開き、すべて選択（Ctrl-A）してからコピー（Ctrl-C）します。

2．ハイパーターミナルに移り＃を入力、＞表示の後、ペースト（Ctrl-V）します。

3．:main が表示されたら「Esc」キーを入力します。

4．ｇで実行します。

注１）プログラムの末尾 :main などの後には、必ず改行やスペースを含めてコピーペーストしてください。この区切り文字がないと main 処理がシステムに登録されません。

注２）プログラムが暴走したときはリセットボタンだけを押してリセットしてください。それでも暴走が繰り返されるときは、SW2 を押したままリセットしてください。

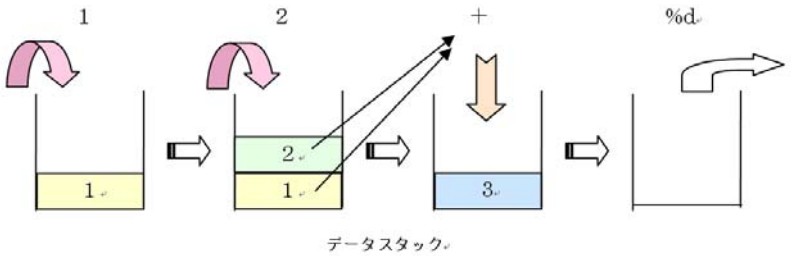
とりあえず、添付のサンプルプログラムをすべて実行してみましょう。どれぐらいのプログラムを書けばどういったことができるのか感触を掴んでください。

以降の文中のプログラムはそのまま実行できます。オンラインのマニュアル（PDF）を開き、テキスト選択モードでコピーして使ってください。

５．プログラミング

ForCy-USB で使われる ForCy 言語は、プログラミング言語としてはやや特殊な部類に属します。小規模なマイコンチップのなかで解釈実行できるよう特別に設計されたものですが、慣れるまで時間がかかるかもしれません。干し草を上へ上へと山積みしては上から取り崩して使う作業をイメージしたデータ処理方式を採用しています。専門的にいうと、ForCy は「仮想スタックマシンを操作する後置記法型プログラミング言語」です。

後置記法プログラムでは、必要なデータを揃えておいてから、それらをどう処理するか指定します。１＋２　の代わりに、１　２　＋　と記述します（１に２を足す、日本語読みと同順）。コンピュータの中では、１が入力されると、１はデータなのでとりあえず積んでおきます。つぎの２も同様、１の上に積み上げます。そして＋が来ると、これは演算子（オペレータ）なので、処理を始めます。＋は２つのデータを加算するので、積み上げられたデータを２つ取り出し、これらを加算して、得られた結果の３を再び積み上げます。%d は積み上げられた一番上のデータを取り出して表示するオペレータです。１　２　＋　%d とすれば、３が表示されます。



これを実際にプログラムとして実行するときは、処理手順をワードとしてシステムに登録してから実行する手続きが必要です。適当な名前（ここでは main）を付けてください。

```
1 2 + %d :main （改行）
```

をメモ帳に書いてコピーし、プログラムモード（#）でペーストして Esc し実行（g）してください。数字や記号の間とプログラム末尾にはスペースや改行を入れます。（２＋３）＊（７－４）なら

```
2 3 + 7 4 - * %d :main （改行）
```

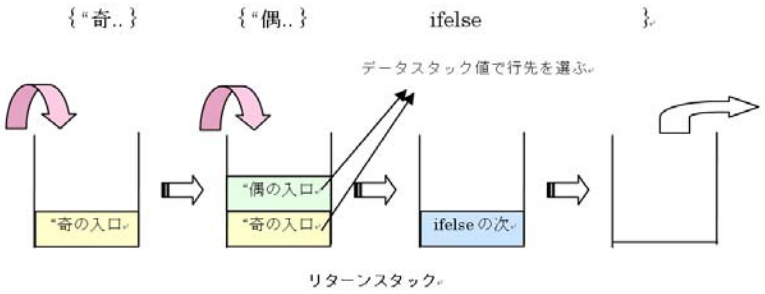
です。このように、データを積み上げては崩し処理を進めるしくみはスタックマシンと呼ばれます。データの積み崩しはデータスタックで行われます。

コンピュータは、状況によって処理の流れを変化させることで、複雑なデータ処理を行います。この流れの制御もまたスタックを使って行われます。

```
        @c '0' -          // 文字から数値へ変換
        2 %               // 2で割った余り
        { "奇数" %s } { "偶数" %s } ifelse
:main
```

1行目で、キー入力した1文字を数値に変換します。@c でキー入力文字をデータスタックに積み、文字0の値を引くことで文字コードを数値にします。2行目で、この値を2で割った余りを計算します。%はデータスタック上から2番目の値を1番目の値で割った余りを、これらの値を取り出したあとに積み上げます。3行目で余りが0でないなら奇数、0なら偶数を表示します。%s は " " のなかの文字列を表示します。

プログラム中で { を見つけると、コンピュータは、{ の入口の場所を別のスタック（リターンスタック）に積み、} までを処理せず通過します。この場合は "奇数"... と "偶数"... の入口が順に積まれます。ifelse は、データスタックに積んである値が0ならリターンスタックの上から1番目の入口から、0以外なら2番目の入口から処理を始めます。また、このときリターンスタックの2つの入口は取り出し、ifelse の次の入口を積んでおきます。選んだ処理が } まで来ると、今度はリターンスタックから入口を取り出して処理を始めます（つまり、ifelse 以降に進みます．．ここでは終了）。




```
        { "繰り返し " %s } do
:main
```

do はリターンスタックトップの入口に入ると同時に do 自身の入口をリターンスタックに積みます。よって、} までくると、また do が実行されるので、いつまでも { } 内が繰り返されることになります。なお、この繰り返しは {} 内の while, break 文による条件判断で do 以降に抜け出すことができます。

このように、データスタックとリターンスタックを用いることで、高度なデータ処理や機器制御を行うことができます。ForCy 基板では、あらかじめマイコンがスタックマシンの動作をするようプログラムしてあります。ユーザがプログラムを流し込めば、スタックを操作しながら処理が進められます。

大きなプログラムは、小さなプログラムを積み重ねて作ります。ForCy ではプログラムの単位をワード（語）と呼びます。繰り返し使う処理、内容的にまとまりのある処理はワードとして切り出し名前をつけておき、以降のプログラムで呼び出します。また、データスタックを使って値をワードに引き渡すこともできます。

```
        0 { "おいでやす。" %s ++ } for ..
:挨拶
        10 挨拶
:main
```

とすれば、「挨拶」というワードが main から呼ばれますが、そのときデータスタック経由で値 10 が渡されています。「挨拶」ではこの回数だけ表示を繰り返します。

ワードを呼び出すとき、そのワードは必ずその前のどこかで定義されている必要があります。また、定義したワードのなかでデータスタックがいくつ増減するか、常に注意してください。スタックの動きを正確にイメージし操作することが、スタックマシン・プログラミングの勘所です。ワード定義を上手に使うとプログラムが読み易くなります。

後置記法のプログラミングは、処理順序を分析せずともスタックの上だけを見て処理を進められる、コンピュータに都合のよい記述方法です。欧米で制御システム開発にしばしば使われる FORTH や、プリンタ制御と PDF ファイルの内部記述に使われる PostScript がよく知られています。BASIC, Java, C などの一般に広く普及しているプログラミング言語では、式や構文をもう少し人が見慣れた形式で書けるようにしてありますが、その分、実行前に処理順序を正しく判断しておく手間が必要になります。前置記法型の言語には、研究や教育によく用いられる Lisp や Scheme があります。

６．デジタル信号の制御

プログラミングに慣れたら、さまざまな部品を I/O ポートに接続して制御の実験をしてみてください。ここでは基本的な部品のつなぎかたについて概説します。

● 発光ダイオード（ＬＥＤ）

I/O ポートを出力に設定しＨレベル電位とすると、4.5V ぐらいの電圧が生じます。発光ダイオードを光らせたときの電圧降下は赤色で 1.6V，緑色で 2.0V ぐらいですから、直結すると無理な電流が流れて I/O ポートや LED を傷めます。間に電位差を吸収し電流を制限する抵抗が必要です。LED は 2-10mA 程度の電流で十分に光りますから、330～1KΩぐらいの抵抗を入れてください。

● 圧電ブザー

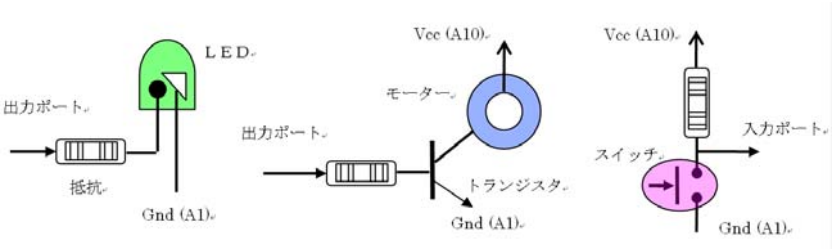
圧電ブザーは圧電素子加える電圧変化により音を発生します。コンデンサと等価です。保護用に１KΩ程度の抵抗を入れます（ LED と同様）。

● モーター

モーターには大きな電流（0.1A 以上）が流れます。マイコンの I/O ポートはモーターを動かすだけの電流を出力できませんので、トランジスタや FET で駆動します。ON/OFF だけなら１個で済みますが、逆転させるのであれば４個でブリッジ回路を組みます。

● スイッチ

スイッチの状態を I/O ポートで読み取るときは、スイッチの ON/OFF が電圧の H/L として得られるようにします。ある程度の電流を流すと安定しますので、3.3-10KΩぐらいの抵抗でプルアップ、プルダウンします。



LED の点滅

LED の点滅は、LED の接続されたポートの電位をＨ（High)にし、しばらく待ってＬ（Low)、しばらく待ってまたＨ … の繰り返しです。もっとも初歩的なプログラムですが、このなかにはマイコン制御技術の多くが含まれています。

まずはポート機能を設定し、マイコンチップの足ごとの電圧上げ下げの準備をします。これらはマイコン内部にある特殊機能レジスタ（SFR)に値を設定することで行われます。まず、その足を電圧の出力に使うか、それとも外部の電圧を読み取るのに使うか、データ方向レジスタに指定します。その上で、出力ならデータレジスタに値を書き込み、入力ならデータレジスタの値を読み出します。この設定は足を８本ずつまとめて行います。SFR には機能ごとに異なる番地が割り当てられています。ハードウェアの制御には、マイコンのデータシートを熟読する必要があります。

注１）16 進表記

Ｈ，Ｌの２状態を持つデジタル信号は２進数で表現すると分かり易いですが、信号数分の桁だと扱い難くなります。一般には２進数４桁ずつをまとめた１６進表記が多く用いられます。８ビット（２進８桁）であれば、0～9,a,b,c,d,e,f の１６進２桁で表現できます。１６進数の数値には頭に 0x をつけます。

注２）コメント

// で始まる行、あるいは /* */ で挟まれた部分はプログラムを読み易くするためのコメントです。プログラム処理には関係ないので省いても動作します。

```
0x24 :DDRB          // ポート B のデータ方向レジスタ
0x25 :PORTB         // ポート B のデータレジスタ
0x27 :DDRC
0x28 :PORTC
0x2a :DDRD
0x2b :PORTD

0xff DDRB =sfr      // ポート B を出力方向
0x0f DDRC =sfr      // ポート C 下位４bit を出力方向
0xfa DDRD =sfr      // ポート D 上位４bit と下位２bit を出力方向

0xff PORTB =sfr     // ポート B の出力データをすべて H
0x00 PORTC =sfr     // ポート C の出力データをすべて L
0x00 PORTD =sfr     // ポート D の出力データをすべて L
@c
:main
```

８X８マトリクスLED のアノード側（正電位を加える側）８本はポートＢに接続されています。カソード側（０電位側）の８本は、上４本がポートD,下４本がポートＣに接続されています。ポートＢをすべてＨに、ポートD,Cの４ビットをＬにして、すべてのＬＥＤを点灯しています。@c でなにかキー入力するまでプログラム終了を待たせます。

（ループで時間調整）

点滅させるときは、繰り返し処理のなかで出力のＨＬを切り替えますが、このとき待ち時間を入れて適当な周期にします。

```
0x24 :DDRB
0x25 :PORTB
0x27 :DDRC
0x28 :PORTC
0x2a :DDRD
0x2b :PORTD

    0xff DDRB =sfr   // ポートＢを出力
    0x0f DDRC =sfr   // ポートＣ下位４bit を出力
    0xfa DDRD =sfr   // ポートＤ上位４bit を出力
    0x00 PORTB =sfr
    0x00 PORTC =sfr
    0x00 PORTD =sfr
:LED_初期化

    PORTB sfr ~ PORTB =sfr
:LED_反転

    0 { ++ } for ..          // ここで時間待ち
:待ち

    LED_初期化
    {
            LED_反転
            30000 待ち       // この数値で調整
            @c? break        // キー入力待ち
    } do
:main
```

原理は簡単ですが、これだと待ち時間を正確に合わせるのが面倒です。

（カウンタで時間調整）

０．５秒とか２秒を正確に設定するには、クロックを使います。基板が動き出してからの経過時間が clk に２ミリ秒単位で、sec に秒単位で格納されています。これらの値はプログラム実行中も裏で自動的に増えていきます。

```
0x24 :DDRB
0x25 :PORTB
0x27 :DDRC
0x28 :PORTC
0x2a :DDRD
0x2b :PORTD

    0xff DDRB =sfr   // ポート B を出力
    0x0f DDRC =sfr   // ポート C 下位４bit を出力
    0xfa DDRD =sfr   // ポート D 上位４bit を出力
    0x00 PORTB =sfr
    0x00 PORTC =sfr
    0x00 PORTD =sfr
:LED_初期化

    PORTB sfr ~ PORTB =sfr
:LED_反転

    clk + { clk != while } do .       // clk の指定増分だけ待つ
:待ち

    LED_初期化
    {
            LED_反転
            250 待ち
            @c? break        // キー入力待ち
    } do
:main
```

今の clk 値に 250 を足した値に clk が増えるまで待つので、０．５秒ごとに明滅します。これで時間はかなり正確ですが、LED を点滅させるだけに高性能マイコンを占有しています。マイコンは、ON/OFF の瞬間以外はただ時間待ちで遊んでいます。

（割り込み処理）

普段は別の仕事をさせておいて、ON/OFF の瞬間だけ LED 制御を行うには割り込み処理を使います。割り込みは、外部信号の変化やタイマ設定した時刻でプログラムを中断して予め決めておいた処理を実行するしくみです。ForCy では２ミリ秒と１秒のタイマ割り込みを使えます。

```
0x24 :DDRB
0x25 :PORTB
0x27 :DDRC
0x28 :PORTC
0x2a :DDRD
0x2b :PORTD

    0xff DDRB =sfr   // ポート B を出力
    0x0f DDRC =sfr   // ポート C 下位４bit を出力
    0xfa DDRD =sfr   // ポート D 上位４bit を出力
    0x00 PORTB =sfr
    0x00 PORTC =sfr
    0x00 PORTD =sfr
:LED_初期化

    PORTB sfr ~ PORTB =sfr
:LED_反転

    ;wait
    wait 0 ==
    {
            LED_反転
            . 250
    } if
    -- =wait         // wait を１減らす
:２m秒割り込み

    LED_初期化
    0 =wait
    { ２m秒割り込み 0 } interrupt    // 割り込み処理の登録
    @c .             // キー入力待ち。ここで別のプログラムを処理可能
:main
```

２ミリ秒ごとに一瞬だけ呼び出される以外は、普段はまったく別の仕事をさせておけます。また、wait1, wait2,.. と複数の待ちカウント処理を用意すれば、複数の LED を違う周期で点滅させることもできます。

（タイマ機能）

ここまでの考え方は、パソコンでのプログラミングにも共通します。組込みプログラミングでは、もうひとつ、マイコンチップに内蔵される周辺機能を使う方法があります。タイマ／カウンタを用いれば、簡単な計数処理はソフトウェアを煩わせずにハードウェアで済ませることができます。タイマの出力ポートにLEDを接続し、特殊機能レジスタでタイマを初期設定しておきさえすれば、プログラムの処理とは無関係にLEDを点滅させ続けることができます。タイマ出力ビットが接続された１列だけ点滅します。

```
0x24 :DDRB
0x25 :PORTB
0x27 :DDRC
0x28 :PORTC
0x2a :DDRD
0x2b :PORTD
0x80 :TCCR1A                // タイマ・カウンタ１制御用
0x81 :TCCR1B                // タイマ・カウンタ１制御用
0x88 :OCR1AL                // タイマ・カウンタ１比較用
0x89 :OCR1AH                // タイマ・カウンタ１比較用

    0xff DDRB =sfr
    0x0f DDRC =sfr
    0xfa DDRD =sfr
    0x00 PORTB =sfr
    0x00 PORTC =sfr
    0x00 PORTD =sfr

    0x40 TCCR1A =sfr
    0x0d TCCR1B =sfr        // 8MHz/1024 をクロック入力
    15 OCR1AH =sfr          // 3906(15*256+66)毎に反転
    66 OCR1AL =sfr
  @c .              // キー入力待ち。ここで別のプログラムを処理可能
    0 TCCR1A =sfr           // タイマ解除
:main
```

特殊機能レジスタの設定方法は、ATmega88のデータシートをよく読んで理解してください。マイコンの周辺機能をうまく使いこなすことが組み込みプログラミングの難しさであり面白さかもしれません。

ブザー

圧電ブザーから音を出すことは、LED を点滅させることと同じです。違いは、その周期が音として聞こえるぐらい短いことです。

```
0x2a :DDRD
0x2b :PORTD

    clk + { clk != while } do .       // clkの指定増分だけ待つ
:待ち

    0x0a DDRD =sfr                    // ブザー端子（PD3） を出力
    {
            PORTD sfr 8 ^ PORTD =sfr // 出力反転
            2 待ち
    } do
:main
```

（キー入力をチェックしていないので、RESETで止めてください）

待ち時間調整では思うような音を出し難いです。こうした処理こそはタイマ機能を使うべきです。

```
0x2a :DDRD
0xb0 :TCCR2A
0xb1 :TCCR2B
0xb3 :OCR2A

    ;i

    "¥x8d¥x7e¥x70¥x69¥x5e¥x54¥x4a¥x46"        // 周期データ
    i @s OCR2A =sfr           // タイマ２周期設定
    i ++ 7 & =i
:音階

    0xfa DDRD =sfr            // ブザー端子（PD3） を出力
    0x44 TCCR2B =sfr          // タイマ２モード設定
    0x12 TCCR2A =sfr          // タイマ２クロック設定

    0 =i
    { 0 音階 } interrupt      // 一秒ごとに音発生
    @c .
    0 TCCR2A =sfr             // タイマ２停止
:main
```

スイッチの読み取り

押しボタンスイッチが押されているか調べるときは、スイッチが接続されているポートを入力に設定します。スイッチは開放状態でHレベルになるようにしてある（プルアップ）ので、データレジスタの値を読み出して、そのスイッチのつながれたビットが０なら短絡、つまり押されていることになります。

```
0x26 :PINC
0x27 :DDRC

    0x0f DDRC =sfr
    {
            PINC sfr 0x10 & 0 == { '1' %c } if .       // SW1
            PINC sfr 0x20 & 0 == { '2' %c } if .       // SW2
            @c? break
    } do
:main
```

このプログラムでは押している間は反応し続けます。押されたら１度だけ処理するには、前に押されてから一度スイッチが離れたか確認する必要があります。また、スイッチは押された瞬間に接点がばたつくことがありますので、その対策も必要です。スイッチの読み取りは案外難しいかもしれません。

７．アナログ値の入出力

明るさ、重さといった物理量は基本的に連続的（アナログ）な値を持っています。これらはセンサ素子によって電気信号に変換されますが、これもまた連続的な電圧値を出力します。この電圧値をマイコンで読み込むには、アナログ－デジタル変換のしくみが必要になります。

ForCy-USB のマイコンにはアナログ値を 1024 段階のデジタル値（10bit）に変換できる機能が備わっています。

（温度計測）

温度計測には温度センサを用います。抵抗値が変化する素子（サーミスタ）が手軽ですが、抵抗値から温度への換算が少々面倒です。入手容易で精度が得やすいことから、ここでは温度センサＩＣの LM35DZ を使います。10ｍＶ／℃ の出力なので換算も簡単です。３本足の小さな部品です（通販などで購入してください）。

ＡＤ変換に使えるポートは PC0～5 までで同時に６つまで素子を接続できますが、一度に変換できるのはひとつだけです。内部で切り替えて使います。ここでは PC0（ADC0）に接続します。

LM35 の GND	→	Gnd	
Vout	→	PC0	ADC0
+Vs	→	PC1	+4.5V

ＡＤ変換ではリファレンス電圧に注意します。アナログ入力の最大値より大きくしますが、通常は 1.1V の基準電圧かマイコンの電源電圧とします。LM35 の場合、1.1V 基準電圧を使って 10bit 計測すると、分解能がほぼ１ｍＶとなります。つまり得られたデジタル値がそのまま ０．１℃単位の温度となります。１０で割って間に小数点を入れ表示しました。もし温度センサの代わりに可変抵抗で電源電圧を分圧して加えるときなどは、ADMUX を 0x40 に設定してリファレンス電圧を電源電圧とします。

（一定時間ごとの計測）

温度変化などを一定時間ごとに計測するときは、必要な間隔だけ秒数を数えてから AD 変換します。ForCy-USB のマイコンで作られるクロックには１～２％の誤差があります。より正確な時間計測には外部の水晶発振器からクロックを加え、タイマでカウントするなどの工夫が必要になります。

（照度計測）

Ver 2.0 基板には照度センサ兼用の青 LED が SW2 に接続されています。計測前に出力ポートにして短絡し、一定時間経過後にＡＤ入力に切り替えて電圧を読み取ってください。

```
0x24 :DDRB
0x25 :PORTB
0x27 :DDRC
0x28 :PORTC
0x78 :ADCL
0x79 :ADCH
0x7a :ADCSRA
0x7b :ADCSRB
0x7c :ADMUX
0x7e :DIDR0

    0xff DDRB =sfr              // ポートB を出力
    0 PORTB =sfr
    2 DDRC =sfr                 // ポートC0 を入力, C1 を出力
    0xfe PORTC =sfr             // 残りをプルアップ
:ポートの初期化

    0x01 DIDR0 =sfr             // ADC0 を使用
    0xc0 ADMUX =sfr             // ADC0 を選択
    0x87 ADCSRA =sfr            // ADC を起動
:ADC_初期化

    0xc7 ADCSRA =sfr
:ADC_開始

    ADCL sfr ADCH sfr 8 << |
:ADC_読み

    0 ADCSRA =sfr
    0 DIDR0 =sfr
:ADC_終了

    ADC_読み
    10 div swap %d '.' %c %d "¥r¥n" %s          // 温度表示
    ADC_開始
:温度計測

    ポートの初期化
    ADC_初期化

    { 0 温度計測 } interrupt            // １秒ごとに温度計測
    @c .
    ADC_終了
:ＡＤ変換
```

（アナログ値の出力）

ForCy-USB のマイコンにはデジタル値をアナログ値に変換する機能はありません。高速高精度の変換には専用の回路を外付けする必要があります。しかしながら、カウンタの PWM 機能を用いれば擬似的にアナログ値を作ることができます。LED の明るさやモーターのトルクを制御する程度なら十分に使えます。

ＰＷＭ（パルス幅変調）とは、デジタル信号のＨＬレベルを高速に切り替えるとき、Ｈの期間とＬの期間を調整することで、平均的にＨＬの中間の値を生成する方法です。Ｈ：Ｌが１：１ならＨレベルの 50%, ３：１なら 75% となります。コンデンサで平滑化しますが、PWM 周期が短ければ LED の点灯などにはそのまま使えます。直流モーターの PWM 制御では、２～３段階ぐらいのトルク切り替えが無難でしょう。下の例では PWM 出力に接続されたマトリクス LED の一列だけが表示されます。

```
0x24 :DDRB
0x25 :PORTB
0x27 :DDRC
0x28 :PORTC
0x2a :DDRD
0x2b :PORTD
0x80 :TCCR1A                    // タイマ・カウンタ１制御用
0x81 :TCCR1B                    // タイマ・カウンタ１制御用
0x88 :OCR1AL                    // タイマ・カウンタ１比較用
0x89 :OCR1AH                    // タイマ・カウンタ１比較用

    0xff DDRB =sfr
    0x0f DDRC =sfr
    0xfa DDRD =sfr
    0x00 PORTB =sfr
    0x00 PORTC =sfr
    0x00 PORTD =sfr
    0x81 TCCR1A =sfr            // PhaseCorrect PWM
    0x03 TCCR1B =sfr            // 8MHz/64 をクロック入力
:初期化

    0 OCR1AH =sfr OCR1AL =sfr           // カウンタを再設定
    5 clk + { clk != while } do .       // 少し待ち
:輝度

    初期化
    {
            255 0 { dup 輝度 ++ } for ..
            0 255 { dup 輝度 -- } for ..
            @c? break
    } do
    0 TCCR1A =sfr               // PWM モード解除
:main
```

８．テキストの送受信

ForCy-USB はプログラムの転送に PC とシリアル通信していますが、プログラムの実行中にも文字の送出や受信ができます。マイコンでシリアル通信するには、特殊機能レジスタをいろいろ設定して USART（汎用シリアル非同期送受信ユニット）を動作させるとともに、ばらばらなタイミングで送受信される文字を溜めておくしくみ（バッファリング）も必要になります。こうした手続きをすべて ForCy プログラムで組むとかなり煩雑になるため、基本的な文字入出力のしくみはシステム定義ワードとして用意してあります。

PC のハイパーターミナルからキー入力した文字を ForCy 基板で読み込むには、@c を用います。@c はシリアル回線から１文字受信するまで待って、受信した文字をデータスタックに積みます。この逆に、ForCy プログラム中で生成した文字をハイパーターミナル画面に表示するには %c を用います。%c はスタックトップの文字を取り出しシリアル回線に送出します。

```
{ @c %c } do :main
```

とすれば、入力した文字を出力（エコー）します。@c は文字が入力されるまで待ち続けますが、ここでプログラムを待たせたくないときは、@c? で文字の有無だけを確認します。

```
{ @c? { @c %c } if } do :main
```

@c? は文字がすでに受信されていれば真（０以外）、まだ受信していなければ偽（０）をスタックトップに積みます。

単語など複数の文字からなる文字列は、１文字ずつ受信して配列に格納するなどしてから利用します。文字列を送出するときは、%s を用います。”文字列” %s とすればハイパーターミナル画面に 文字列 が表示されます。文字列中の１文字だけ送るときは、”ABCDEF” 3 @s %c のようにすれば、３番目（０から数えて）の文字 'D' が抜き出されて送出されます。文字列に全角文字があるときは、２文字分と数えて２文字分送ってください。

数字を表示するには、数値データを文字列に変換して送る必要があるのですが、%d, %x を使えばデータスタックトップの数値データを内部で 10 進数、16 進数として文字列変換し、送出します。

９．多桁ＬＥＤの走査

I/O ポートに LED を接続すれば、ポートの数だけの LED を明滅させることができます。数値表示に用いられる７セグメント LED では小数点も含めて８つの LED が入っていますから、８ビット分のポートが必要になります。４桁表示なら 32 ビット。ForCy-USB 基板のマトリクス LED では計 64 個の LED が集積しています。たくさんの IO ポートを持つマイコンは高価なうえ配線もたいへんです。

一般に多数の LED を制御するときは、LED を格子状に結線して縦横の組み合わせで表示させる、「ダイナミックスキャン方式」が用いられます。高速で各桁あるいは各行を切り替え表示すれば、全桁、全行が同時に表示されているように見えます。下のサンプルをベースにいろいろな表示を試してみましょう。

```
    0xff 0x24 =sfr          // DDRB
    0x0f 0x27 =sfr          // DDRC
    0xfa 0x2a =sfr          // DDRD
:ポート設定

    ;行 [8]内容

    0 0x25 =sfr             // いったん消灯

    行 ++ 7 & =行           // 表示行を巡回移動
    1 行 << ~               // 対応する行のビットを0
    dup 0x0f & 0x30 | 0x28 =sfr     // SW1,2 はプルアップ
    0xf0 & 0x2b =sfr

    行 内容 0x25 =sfr       // その行の内容表示
:走査


    8 0 { dup 内容 ~ over =内容 ++ } for ..
:反転

    ポート設定
    0 =行
    8 0 { 0xff over << over =内容 ++ } for ..

    { 走査 反転 } interrupt
    @c .
:ＬＥＤ表示
```

１０．複数タスクの連携

現実の組込みシステムでは、さまざまな役割を担うプログラム群（タスク）が、互いに影響し合いながら協調して複雑に動作しています。あるタスク（例えばキー入力処理）が起動すると、それに関連する別のタスク（計算とか表示とか）がそこから起動されるような連鎖が多発するとき、それらのタスクが満遍なく実行されるよう管理するしくみが必要になります。タスク管理は、Windows や TRON などのオペレーティング・システムの重要な役割です。

下の例では処理０、処理１が互いを１秒ごとに起動し合うだけですが、起動までの遅延時間を指定するよう拡張すれば、擬似マルチタスクと呼ばれる実用的なタスク管理ができるようになります。

```
    ;先頭 ;末尾 [8]行列

    末尾 =行列
    末尾 ++ 7 & =末尾
:登録

    " 0: ->1¥r¥n" %s
    1 登録
:処理0

    " 1: ->0¥r¥n" %s
    0 登録
:処理1

    先頭 末尾 !=
    {
            先頭 行列
            { 処理0 処理1 } of
            先頭 ++ 7 & =先頭
    } if .
:毎秒

    0 =先頭
    0 =末尾
    { 0 毎秒 } interrupt
    0 登録
    @c .
:タスク管理
```

１１．再帰呼び出し

上級向けの話題です。ForCy プログラムでは定義済みのワードを呼び出すことができますが、現在定義中のワード、つまり自分自身を呼び出すこともできます。古典落語の「頭山」に大男が自分の頭にできた池に身投げする悲話がありますが、この技法によって、複雑そうに見える処理が明瞭なアルゴリズムとして実装することができるようになります。

数学で帰納的に定義される式、例えば階乗は次式で表されます。

```
f(n)    = f(n-1) * n     ( n>0 )
        = 1              ( n==0 )
```

これはつぎのようなプログラムになります。

```
        0 > { dup -- self * } { . 1 } ifelse
:階乗
        7 階乗 %d
:main
```

再帰呼び出しを使えば、do 文や for 文を用いずとも、繰り返し処理を記述することができます。

```
        0 > { "いつまでも"%s -- self } if
:繰り返し
        100 繰り返し
:main
```

これで指定回数、「繰り返し」が繰り返されます。再帰呼び出しではデータスタックとリターンスタックが急速に消費され易いことに注意してください。ForCy ではこの例のように、再帰呼び出しがワードの末尾にあるときはリターンスタックの消費が抑えられるようにしてあります（末尾呼び出しの最適化）。

サンプルプログラムのなかで再帰呼び出しを使っているもの

Hanoi.txt	ハノイの塔
Qsort.txt	クイック・ソート
Queen.txt	８王妃の問題
Rcrs.txt	再帰呼び出し数式

１２．文法

16 bit スタックマシンへの後置記法プログラム。ワード、変数、定数、コメントで構成される。

大分類	小分類	要素	例
ワード	システム定義	文字列	dup　+
	ユーザ定義	文字列	:main
変数	変数	符号なし 16 ビット整数	;x　;y　;count
	配列	変数の１次元配列	[10]a　[0x20]b
定数	数値	符号なし 16 ビット整数	34　0x1f　‘Q’
	文字列	バイト列（0 終端）	"string"　“¥x7f¥xff”

変数

16 ビット符号なし整数（0～65535）。使用前に ; で宣言する。（;a など）

計算で範囲を超えると、例えば 65535+1 は 0 に、負値は 65536 から引いた値（−10 なら 65526）となる。

a を読み出し（スタックに積む）：　a　　a に 100 を代入：　100 =a

配列

変数のまとまり。使用前に [] で宣言する。[10]a なら a[0]～a[9] まで。

a[0] を読み出し（スタックに積む）：　0 a　　a[0] に 100 を代入：　100 0 =a

定数

10 進数　　123 など

16 進数　　0x78　　（10 進数だと 16*7+8 = 120）

文字定数　　'A'　ASCII コード値。0x41

文字列　　"abc¥xff¥n"　¥x?? は文字の 16 進数指定。¥n は改行。文字列の末尾には 0 が付加。

ワード登録	... :name	**コロンに続けて名称**
	... self ... :name	**再帰呼び出し**
ワード実行		**最終登録ワードを実行**
変数		
宣言	;name	**セミコロンに続けて名称**
参照	name	**変数の ” 値 ” をスタックトップに積む**
代入	=name	**スタックトップの値を変数に代入**
配列		
宣言	[size]name	size **の配列領域を確保**
参照	index name	name[index]**の値をスタックトップに積む**
代入	index =name	**スタックトップの値を** name[index]**に代入**
文字列	"string"	**中間コード上の文字列開始位置を内部ポインタにセット。文字列は 0 終端**
	index @s	**内部ポインタから** index **番目の文字をスタックトップに積む**
コメント	/* */	**区間をコメント扱い**
	//	**行末までをコメント扱い**
	,	**末尾 ‘ , ’ のワードをコメント扱い**

・ワードはスペース、タブもしくは改行で区切る（半角文字のみ）。

・ワード、変数名、配列名は半角 15 文字まで。

・最終登録ワードから実行する。名称は任意。

・文字列は半角 255 文字まで。Shift-JIS コード可。

・コメント /* */, // の長さは任意。ただし ‘ , ’ コメントは半角 14 文字まで。

フロー制御	{ ... } do { ... **条件式** while ... } do { ... **条件式** break ... } do { ... **条件式** continue ... } do { ... **指値更新** ... } **終値 指値** for **終値 指値** { ... **指値更新** ... } for **条件式** { ... } if **条件式** { ... } { ... } ifelse **比較値** { **値１** { ... } case **値２** { ... } case ... } switch return n { w0 w1 ... wk } of { w0 w1 ... wk } interrupt	**無限ループ** **真で継続（偽で脱出）** **真で脱出（偽で継続）** **真で{} 節の先頭から再開（偽で継続）** **指値が終値に等しくなるまで繰り返し** **（指値更新：スタックトップに** ++,--**など）** **真で{} 節を実行** **真で左{} 節、偽で右{} 節を実行** **比較値と一致する値の左節のみ実行** **一致がなければ末尾を実行** **実行中のワードからリターン** n **番目のワードを実行** **割り込み実行ワードのテーブル設定）** w0: 10mS **毎に呼び出されるワード** w1: 1Sec **毎に呼び出されるワード**

do

{ } のなかの処理を繰り返す。この区間でのデータスタックの増減が０になるよう注意。

while, break, continue

データスタックトップの値が真（０以外）か偽（０）によってループを脱出（while,break）するか、ループ先頭へ戻る(continue）。データスタックトップは取り崩される。

for

・終了後に指値と終値が残るので破棄 .. する。
・continue 成立時、以降はスキップされる。

```
{
    ...             // ここで指値操作する
    条件 continue
    ++              // 成立時は実行されない！
} n 0 for ..
```

・指値（ループカウンタ）を変数で使うときは

```
;i
{
    dup =i
    ...
    ++
} 10 0 for ..
```

・指値が終値を飛び越すときは

```
{
    ...
    3 +
    10 min
} 10 0 for ..
```

などとするか while で記述する。

if, ifelse

・条件式は、{} 節の後でもよい。
・{} 節の直下で while, break, continue は使えない。

switch

・{} 節の直下で while, break, continue は使えない。
・終了後に比較値が残るので破棄 . する。

return

・データスタック管理はプログラマが行うこと。

of, interrupt

・ユーザ定義ワードのみ指定可。

比較演算	==	（u1 u2　u1 flag）
	!=	**スタックから”１項”を除き、比較の結果を積む。**
	<	
	<=	
	>	
	>=	
単項演算	++	（u1 -- u2）
	--	**スタックから１項を除き、結果を積む。**
	~	
	!	
二項演算	+	（u1 u2 -- u3）
	-	**スタックから２項を除き、結果を積む。**
	*	
	/	
	%	
	&	
	|	
	^	
	&&	
	||	
	<<	**左シフト**
	>>	**右シフト**
	min	**小さいほう**
	max	**大きいほう**

スタック操作	.	（X --）
		スタックから１項目を除く。Forth:drop
	..	（X1 X0 --）
		スタックから２項目を除く。Forth:2drop
	@	（Xu .. X0 u -- Xu .. X0 Xu）
		スタックトップから指定番目 u の値をスタックに積む。
	dup	（X -- X X）
		スタックトップをコピーする。（ 0 @ と等価）
	over	（X1 X0 -- X1 X0 X1）
		スタックの２項目をコピーする。（ 1 @ と等価）
	=	（Xu+1 .. X1 X0 u -- X0 .. X1）
		スタックトップの値を指定番目 u の項目にコピーする。
	nip	（X1 X0 -- X0）
		スタックの２項目を削除する。（ 0 = と等価）
	swap	（X1 X0 -- X0 X1）
		スタック上位の２項目を交換する。
	_ds	（Xu-1 .. X0 -- Xu-1 .. X0 u）
		データスタック上の項目数をスタックに積む。（デバッグ用）
	_rs	（ -- u）
		リターンスタック上の項目数をスタックに積む。（デバッグ用）

比較演算について

・2 値の比較後は始めの値が残るので不要なら破棄する。

x y < nip { ... } if

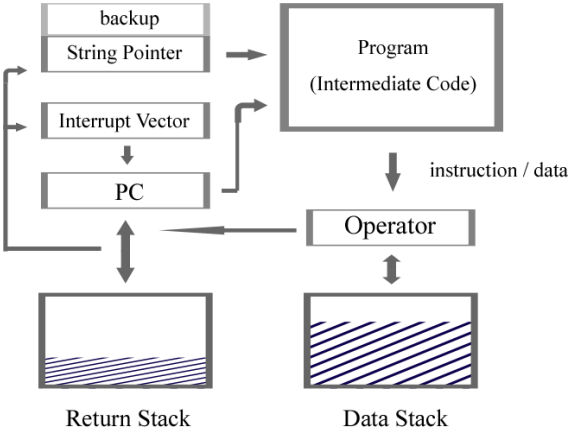
x y < { ... } if .　　// if の直後で破棄

・データは符号なし整数なので、x 0 >= は常に成立（無意味）。

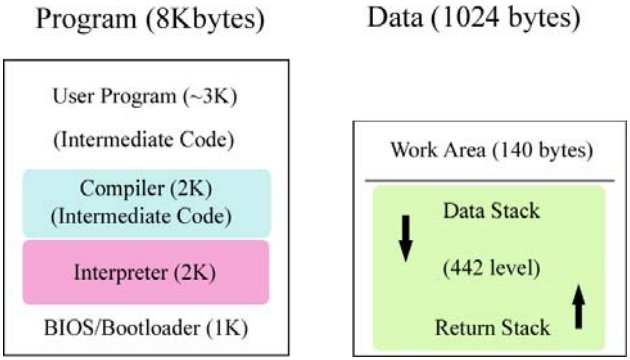
コンソール入出力	@c?	(-- flag) **コンソール文字入力の有無をスタックに積む。**
	@c	(-- char) **コンソール入力文字をスタックに積む。**
	@s	(u -- char) **最後出の文字列定数のスタックトップ番目の文字をスタックに積む。**
	%c	(char --) **スタックトップの値をコンソールに出力する。**
	%s	(--) **最後出の文字列定数をコンソールに出力する。**
	%d	(u --) **スタックトップの値を十進数でコンソールに出力する。**
	%x	(u --) **スタックトップの値を十六進数でコンソールに出力する。**
システム	sfr	(u -- u) **特殊機能レジスタ** [u] **の値をスタックに積む。**
	=sfr	(u1 u2 --) **特殊機能レジスタ** [u2] **に値** u1 **をセットする。**
	ep	(u -- u) **プログラムメモリ** [u] **の値をスタックに積む。**
	=ep	(u1 u2 --) **プログラムメモリ** [u2] **に値** u1 **をセットする。**
	ed	(u -- u) **データメモリ** [u] **の値をスタックに積む。**
	=ed	(u1 u2 --) **データメモリ** [u2] **に値** u1 **をセットする。**
	clk	(-- u) **システム経過時間（**10mS **単位）をスタックに積む。**
	sec	(-- u) **システム経過時間（秒単位）をスタックに積む。**

１３．内部構成

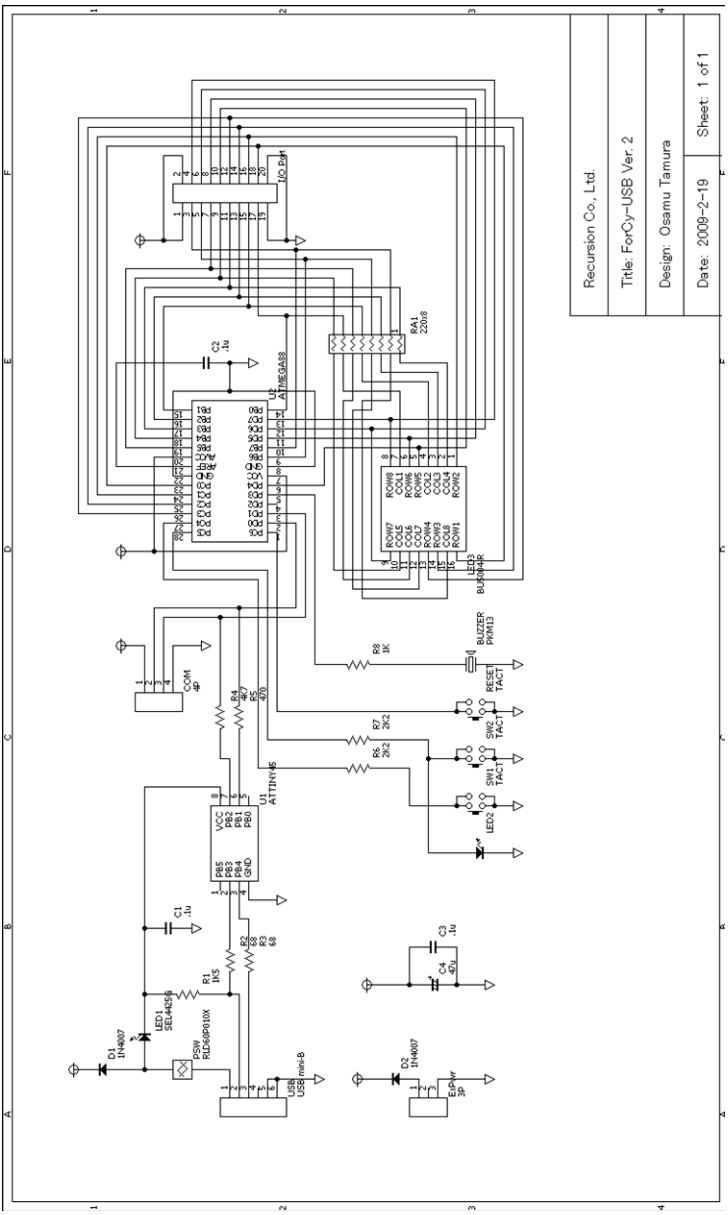
スタックマシン



メモリ配置



回路図


Recursion Co., Ltd.
Title: ForCy-USB Ver. 2
Design: Osamu Tamura
Date: 2009-2-19
Sheet: 1 of 1

１４．マイクロコントローラ概要

ATmega88 の基本仕様と各周辺機能

・特長	高性能、低消費電力の８ビットマイクロコントローラ
・構造	RISC 型アーキテクチャ ８MHz クロックで最大８MIPS の演算性能 ハードウェア乗算器
・メモリ	自己プログラム可能な８K バイトフラッシュメモリ １K バイトスタティック RAM 512 バイトの EEPROM（電源を切ってもデータ保持するメモリ）
・周辺機能	２つの８ビットタイマ・カウンタ １つの１６ビットタイマ・カウンタ PWM 出力（パルス幅変調出力。デジタル・アナログ変換として利用可） ６チャネル１０ビット AD 変換（アナログ値のデジタル変換） USART（シリアル通信） アナログ比較器（アナログ値の大小比較） ２３本の I/O
・その他	電源　2.7-5.5V 動作温度　-40～85℃

※ ForCy-USB では ATmega88 を内蔵ＲＣクロック（８ＭＨｚ校正済）で駆動しています。

● I/O ポートのピンヘッダは半田付けしてありません。自作する基板に合わせて表、裏から半田付けしてください。代わりにピンフレームを用いるか配線を直付けしても構いません。

● ForCy-USB の電源は USB 回線を通して PC から供給されます。外部回路での消費は 100mA 以内にしてください。それより多く消費するときや PC なしで動作させたいときは、電源ポートに 4.5～5.5V を加えます（単三電池３本分）。

● I/O ポートに外部回路を接続すると、マトリクス LED と干渉することがあります。集合抵抗（マトリクス LED 右の白い部品）を IC ソケットからはずせばマトリクス LED を切り離せます。

● ForCy-USB 基板から独自の回路やブレッドボードに AVR マイコンを乗せ替えて使えます。COM の４つの線だけ引き出してつなげば、従来どおり USB 通信によるプログラムができます。

● プログラムが誤作動する原因の多くはスタック上のデータの過不足によるものです。PC 上のシミュレータで基本的なスタックレベルの検査が行えます。ソースコードをダウンロードし、コマンドプロンプトから

```
lint hello 1
```

などとすれば、各定義ワードでのデータスタックの増減を表示、

```
lint hello 2
```

とすれば、すべての定義ワードのデータスタック増減を表示します。

● プログラムの暴走によっては、ForCy コンパイラが破壊されることがあります（利便性のため保護してありません）。ブートモードでコンパイラを再書き込みしてください。

(1) SW1, 2 の両方を押したままRESETを押すと '=' が表示される。
(2) メモ帳で at88x/ForCyAVR.hex を開き、ハイパーターミナルへコピー＆ペーストする。
(3) =ooooooooooooo 中略 ooooooooooooooooo/ が表示され再起動する。

● Ｃ言語、アセンブラでの開発

本基板は、より本格的な開発技術を学べるように、Ｃ言語やアセンブラで作成したプログラムの実行形式をロード実行できるようになっています。PC 側で AVR Studio と GCC で作成したプログラムをコンパイルし、生成した HEX ファイルをブートモードでロードすれば、CPU 能力を最大限に引き出すことができます。詳細は Web ページをご覧ください。

May the ForCy be with you...

※ ForCy は(独)情報処理推進機構(IPA)の未踏ソフトウェア創造事業で 2005 年度上期に採択支援され、(株)ルネサステクノロジ 高田浩和氏の指導により開発しました。
技術情報はすべて無償公開しています。

開発・製造元

有限会社リカージョン

〒600-8148 京都市下京区飴屋町 252 井上ビル 4F

Tel. 075-365-2834

Fax. 075-365-2835

http://www.recursion.jp/forcy-usb/

お問い合わせ

E-mail: info@recursion.jp

[ForCy-USB] 質問、などのタイトルで
