ワンチップ・マイコン・デザイン・コンテスト　入賞作品

作品名

ロー・エンド・マイコン用の内蔵型コンパイラ

作者：田村 修

● 概要

組み込み機器向け小型マイコンの高性能化が著しいですが，提供される開発環境は初心者には敷居が高いように思われます．

小学生からお年寄りまでが手軽にマイコン応用の製作を楽しめるよう，マイコン・チップに軽量の言語処理系を内蔵して，テキスト・エディタ（メモ帳）と通信ソフトウェア（ハイパーターミナル）だけでプログラミング，実行できるしくみ「ロー・エンド・マイコン用の内蔵型コンパイラ ForCy 」を開発しました．

● 動作概要と特徴

シリアル・ポートから送り込まれたプログラム・テキストは，コンパイラが逐次，中間コードに変換して，内蔵のフラッシュ・メモリに格納します．その後，中間コードはインタープリタ動作で実行します（**図 2-1**）．

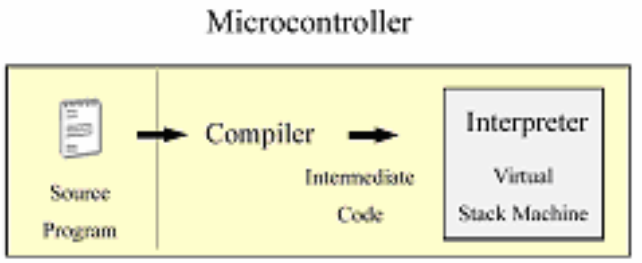

図 2-1　コンパイラの組み込み

コンパイラ自身も自己記述により中間コード化されているので，搭載メモリが ROM 4K ワード, RAM 512 バイト程度のワンチップ・マイコンでも動かすことができます(**図 2-2**).

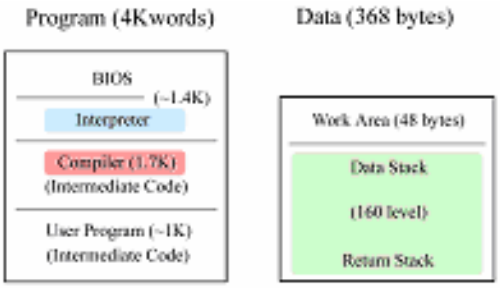

図 2-2　PIC16F88 メモリ・マップ

プログラミング言語は PostScript に似た後置記法のスタック型言語として軽量化しましたが，C と同等の制御構文をもち，BASIC よりもフロー構造を明瞭に記述できます（**リスト 2-1**）．中間コードの短縮，末尾再帰のループ化などの最適化も行いました．処理系は構文解析がほとんどない簡潔な構造のため，改造や拡張が容易に行えます.

- データ型　　16bit 符号なし整数，配列，文字列
- 予約語　　{ } if ifelse do for while break continue switch case
  self of dup nip swap over @ = . .. %c %s %d %x @c @s
- 演算子　　+ - * / % ++ -- == < <= > => << >> min max sfr clk rand ...
- サイズ　　インタープリタ部　1～2.4KB（CPU 依存，アセンブラ記述）
  コンパイラ部　　1.6KB（自己記述による中間コード）
  必要動作メモリ　　2 56 バイト～（定義ワード数による）
- その他　　再帰呼び出し，割り込み，テーブル分岐など

● 実装

これまでのところ，つぎのチップに実装しています.

Microchip 社　　PIC16F88（ROM 4K ワード，RAM 368 バイト）
Microchip 社　　PIC12F683（ROM 2K ワード，RAM 128 バイト，インタープリタのみ）
ルネサス テクノロジ　　R8C/Tiny（ROM 16K バイト，RAM 1K バイト）
Atmel 社　　ATmega88（ROM 8K バイト，RAM 1K バイト）

処理系が小型で改変しやすいことから，マルチプロセッサによる分散制御に応用しました．一つのプログラムを複数のプロセッサに差分コンパイル配信します（**写真 2-1**）．

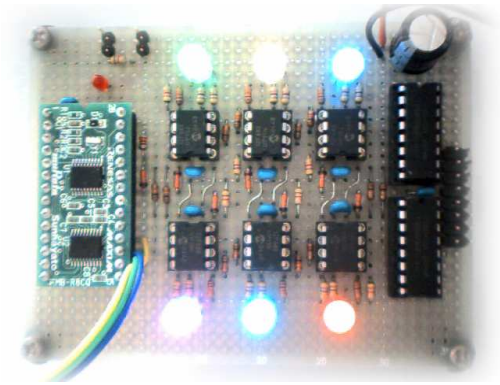

写真 2-1　R8C/Tiny + PIC12F683 X 6

制御以外にも，100 桁ぐらいの円周率計算や，ハノイの塔，8 王妃問題，クイック・ソートといったプログラミングを楽しむこともできます．ATmega88（8MHz）では空ループを毎秒 7 万回繰り返す性能があります（**写真 2-2**）．

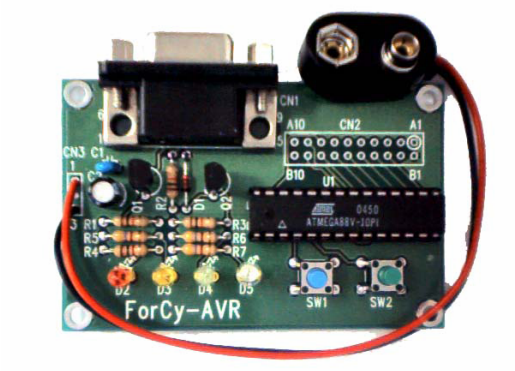

写真 2-2　ATmega88

言語処理系のマイコン内蔵は，初心者にとっての「わかりやすさ」に加えて，言語環境自体を内蔵させることによる高い保守性や，簡素な言語仕様であることによる検証性・信頼性の向上などの利点もあります．

参考 URL

（1）　http://www.recursion.jp/mitou17/

リスト 2-1　ForCy のプログラム例

```
        "¥n¥tHello, World.¥n" %s                //   文字列表示
    :hello

        10 0                                    //   10回繰返し
        {
            hello
            ++
        } for ..
    :hello_10

        ;i [10]a [10]b [10]c                    //   配列の演算

        10 0
        {
            dup =i
            i a    i b    *    i =c             //   c[i] = a[i] * b[i];
            ++
        } for ..
    :c=ab

        ;x ;y                                   //   九九表

        1 =y
        {
            y 9 <= while                        //   1～9まで繰返し
            1 =x
            {
                x 9 <= while                    //   1～9まで繰返し
                x y * %3d                       //   x * y  を表示
                ++ =x
            } do .
            '¥r' %c '¥n' %c                     //   改行
            ++ =y
        } do .
    :multiply

        1 > { dup -- self * } if                //   階乗
    :factorial
```