第1部 基礎知識編

第2章 クロス開発の流れとデバッガのいろいろ

組み込みマイコンの開発環境

熊谷 あき

組み込みマイコンのプログラムを開発するには，Windows のプログラム開発とは異なるツールや考え方が要求される．ここでは組み込みシステム開発の一般論として，どのようなプログラム作成手順を踏むのか，デバッグ・ツールにはどのような形態のものがあるのかなど，組み込みプログラム開発の基礎知識について解説する．（編集部）

1 プログラム開発の流れ

● 機械語とアセンブリ言語

機械語とは 0 と 1 のバイナリまたは 16 進数で表現される，CPU が直接理解できる言葉です．しかしそのままでは，とても人間が理解できるものではありません．

機械語は数値の羅列に過ぎないので，それを多少でも人間に理解しやすい言語に置き換えたのがアセンブリ言語といえます．アセンブリ言語のインストラクション（ニモニックとオペランド）は，機械語よりは人間が理解しやすいように，英単語をベースとした省略形の英数字を使って記述します．

アセンブリ言語と機械語は，基本的には 1 対 1 で対応するので，CPU の動作一つ一つをすべて細かく記述することができます．しかし CPU アーキテクチャが異なると，アセンブリ言語はその記述フォーマットがまったく異なるので，CPU アーキテクチャごとにアセンブリ言語を覚える必要があります．

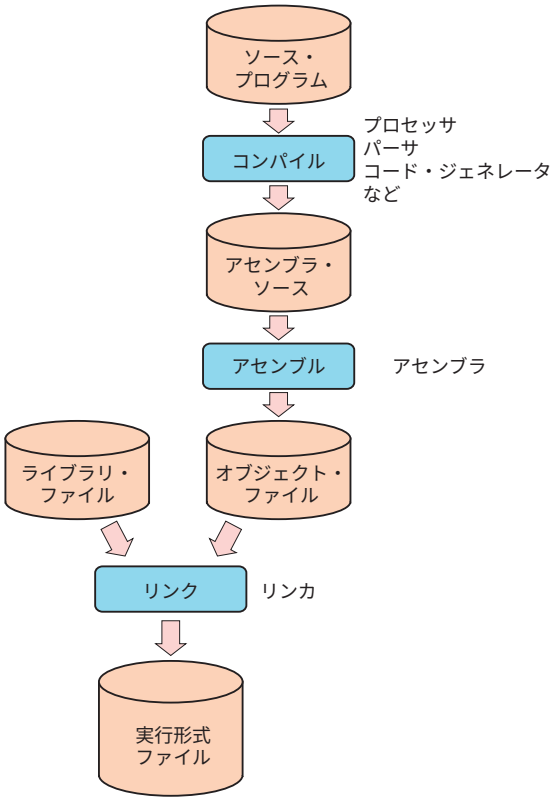


図 1　コンパイラの基本的な動作

● 高級言語

アセンブリ言語では CPU アーキテクチャごとにニモニックなどを覚える必要があり，作成したプログラムの移植もままなりません．

そこでより人間の言語により近い形式でプログラミングしたいという要求から，高級言語が開発されました．現在，組み込みの世界でも良く使われる代表的な高級言語として，C 言語や C++ 言語があります．

● プログラム開発の流れ

C 言語で記述したソース・ファイルは，そのままでは単なるテキスト・ファイルです．CPU が直接実行できる形式のファイルではありません．このソース・ファイルは CPU が直接実行できる形式（機械語）に変換する必要があります．このためのツールを総称してコンパイラと呼びます．

コンパイラは，テキスト・ベースのソース・ファイルをコンパイル（翻訳）し，各種ライブラリやモジュールなど必要なファイルをリンク（結合）し，最終的にバイナリ・コード（機械語）へと変換します．

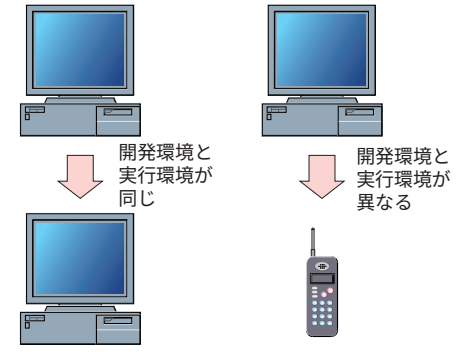
コンパイラの仕様にもよりますが，一般的なコンパイラの動作を**図 1** に示します．図で示すように，内部ではいろいろなツールが呼び出され変換作業を行いますが，一般的に“ コンパイル ”と言った場合は，最終的な実行形式ファイル（機械語：バイナリ・コード）が一番最後に生成されます．このバイナリ・コー

KEYWORD ── 機械語，高級言語，C 言語，アセンブラ，クロス・コンパイラ，クロス開発，デバッグ，ICE，JTAG デバッガ，ROM モニタ型デバッガ

リスト1　Cソース・プログラム

```
short int xyz;

main()
{
   xyz = 0x10;
}
```



(a) ネイティブ・コンパイラ　(b) クロス・コンパイラ

図2　ネイティブ・コンパイラとクロス・コンパイラ

リスト2　リスト1をコンパイルしたアセンブラ・ソース

```
.text
    .align    0
    .global   _main
_main:
    mov  ip,sp
    stmfd     sp!,{fp,ip,lr,pc}
    sub  fp,ip,#4
    bl   ___gccmain
    ldr  r3,.L3
    mov  r1,#16
    mov  r2,r1@movhi
    strh r2,[r3, #0]    @movhi
    b    .L2
.L4:
    .align    0
.L3:
    .word     _xyz
.L2:
.L1:
    ldmea     fp,{fp,sp,pc}
.comm    _xyz,4@2
```

(a) ARM用

```
_main:
            ADD         #-4,R15
            MOV         #0,R3
            MOV         R3,R0
            MOV.B       R0,@(3,R15)
            BRA         L207
            NOP
L208:
            MOV.B       @(3,R15),R0
            MOV         R0,R2
            ADD         #1,R2
            MOV         R2,R0
            MOV.B       R0,@(3,R15)
L207:
            MOV.B       @(3,R15),R0
            MOV         R0,R3
            MOV         #10,R2
            CMP/GE      R2,R3
            BF          L208
            ADD         #4,R15
            RTS
            NOP
```

(b) SH用

ドが，CPUが直接実行できる形式となります．

● ソース・ファイルと実行ファイル

リスト1にC言語ソース・ファイルを，**リスト2**(**a**)に**リスト1**をARM用にコンパイルした後のアセンブラ・ソースを，**リスト2**(**b**)にSH用にコンパイルした後のアセンブラ・ソースを示します．まったく同じ**リスト1**のC言語のソース・ファイルが，**リスト2**ではまったく別の文字列の記述に変換されました．CPUが異なると命令体系やアセンブラの記述フォーマットが異なるので，これが本当に同じ動作をするプログラムとは思えないほど，違った内容のファイルが生成されます．

リスト2のファイルはファイル形式としては単なるテキスト・ファイルです．このファイルをアセンブルし，さらにリンクして実行形式ファイルを作成します．

● クロス開発とは

プロローグでも先輩が説明していますが，クロス開発とはプログラム開発環境とプログラム実行環境が異なる場合を呼びます(**図2**)．またそのとき使用するコンパイラをクロス・コンパイラと呼びます．PC/AT互換機上で，第2部で解説するH8やSH，ARM，MIPSのプログラムを開発するのは，まさにクロス開発です．

ちなみに，クロス・コンパイラと対比する意味で，開発環境で実行するプログラムをコンパイルする場合は，ネイティブ・コンパイラと呼びます．

● 入出力ライブラリの問題

パソコン上でのプログラミングに慣れていると，組み込み環境でも，printfやfopenなどのファイル入出力関数を気軽に使ってしまいそうになります．しかし，HDDなどのストレージをもたない組み込みシステムでは，ファイルを保存できる媒体がないので入出力関数は使えません．また，printfを実行した結果を表示するための表示装置をもたない組み込みシステムも多いので，printf文を使おうにも表示先が存在しない場合もあります．

グラフィックや文字を表示できる表示装置がない場合でも，シリアル・ポートの先に接続したターミナルに文字列を表示させることで，表示機能を補う使い方もあります．また逆に，ターミナルで入力した文字をシリアル・ポートを介して入力することもできます．printfの表示先やgetchの入力元などを標準入出力装置とも呼びますが，画面やキーボードをもたない組み込みシステムの場合は，このようにシリアル・ポートを標準入出力として使うことが多いようです．

なお，環境が整備された組み込みシステムでは，printfを実行するとそのシステムで規定された標準出力に表示されるように，ライブラリが用意されている場合もあります．

2 デバッグ環境のいろいろ

● Windowsアプリケーションのデバッグ

Windows環境では，**図3**に示すようなGUIを使った非常に強力なデバッガが用意されており，一般的なWinodwsアプリケーションの開発ならこれで十分なデバッグが可能です．

Winodwsはもともと広い画面やキーボード，マウスを標準的に使える環境なので，デバッグもそれらのリッチなリソースを使って快適にデバッグできます(デバッグ作業自体が快適かどうかは別？)．

● 組み込み機器でのデバッグ手法

最近では，LCD液晶画面を装備した組み込み機器もめずらしくありませんが，すべての組み込みシステムが画面表示装置を