
マイコンを正しく操縦するための作法

# 基礎から学ぶC言語講座

岡田 好一
*Yoshikazu Okada*

## 第7回 スタートアップ・ファイルの詳細とアセンブリ・プログラミング

C言語はmain関数からスタートします．しかし，スタックの設定や大域変数の初期化はそれ以前に行われています．

つまり，C言語のmain関数をいきなり動作させることはできず，少量でもアセンブリ言語による初期設定ルーチンが必要です．これが，スタートアップ・ファイルの存在理由です．

開発環境でプロジェクトを普通に作成すると，二つのスタートアップ・ファイル

ncrt0.a30
sect30.inc

が自動生成されます．どちらもアセンブリ言語のソース・プログラムです．sect30.incはncrt0.a30に組み込まれて使用されます．内容は，参考文献(1)と(2)に解説されています．

パソコンでプログラムを組むならば，C言語の知識以外にOSが用意する環境についての知識が必要です．それと同様に，スタートアップ・ファイルの動作を知ることは，R8C/1Bの標準開発環境が提供するソフトウェア上の環境を知ることになります．難しくはないので，しばらくお付き合いください．

ということで，アセンブリ言語を読む必要が生じてきました．プログラムを読むための要点を述べてみます．

## アセンブラの基礎知識

アセンブラは，機械語命令に名前を付けた「ニーモニック(mnemonic)」とアドレスを表す「ラベル(label)」で書かれたアセンブリ言語プログラムを機械語に変換するプログラムです．アセンブリ言語のソース・プログラムは「アセンブラで書かれたプログラム」と言い習わされています．

ニーモニックと機械語は原則として1対1に対応するので，当然ながらCPUが異なれば機械語が異なりますから，ニーモニックも違ってきます．また，通常はCPUの開発元が指定する書式が使われますが，アセンブラの開発元の都合により微妙に変化してしまう場合があります．

それやこれやで，C言語並みの可搬性をアセンブラに求めることはできません．あくまで，機械語を便利に記述するための方便です．

### ● アセンブリ・プログラムの書式

アセンブリ・プログラムの1行は，

ラベル　命令　オペランド

の形をしています．R8C純正のアセンブラでは，ラ

**Keyword 1　機械語(machine language)**

CPUが命令として直接解釈できるビット・パターンで書かれたプログラムです．

アドレス可能なメモリ上に展開され，ビット・パターンはCPUの制御装置のデコーダで解釈され，実行されます．現存するほぼすべてのCPUに見られる階層でしょう．

ビット・パターンそのものは命令コード(instruction code/operation code)と呼ばれます．

通常，addやmovなどと，開発元によってニーモニックが与えられます．原則として，機械語命令とニーモニックは，1対1対応です．

しかし，mov命令のように非常に異なるビット・パターンに同一のニーモニックが与えられていたり，adjnz命令とsbjnz命令のように同じコードの場合があります．また，ldintb命令のような暗黙のマクロが紛れ込む場合があります．

表7－1　R8C/Tinyの純正アセンブラの指示命令（一部）

| | |
|---|---|
| .glb | 外部でも使われるシンボル／ラベル |
| .equ | シンボルの設定 |

（a）ラベル（アドレス）とシンボル（定数）

| | |
|---|---|
| .section | セクションの始まり |
| .org | アドレスの指定 |
| .end | ソース・プログラムの終了 |
| .include | ファイルの内容の挿入 |

（b）セクション（一続きのメモリ）

| | |
|---|---|
| .byte | ROM上の1バイト |
| .word | ROM上の2バイト |
| .addr | ROM上の3バイト |
| .lword | ROM上の4バイト |
| .blkb | 1バイト単位のRAM領域 |
| .align | 偶数アドレスに調整 |

（c）領域確保

| | |
|---|---|
| .if | 条件により以下をアセンブルする |
| .elif | else ifのこと |
| .else | 条件以外で以下をアセンブル |
| .endif | 条件アセンブルの終わり |

（d）条件アセンブル

| | |
|---|---|
| .macro | マクロ定義の始まり |
| .endm | マクロ定義の終わり |

（e）マクロ

ベルにはコロン「:」が後続します．ラベルは省略可で，大半の行には付いていません．命令とオペランド（命令の対象）は空白またはタブで区切ります．オペランドが複数の場合はカンマで区切ります．

命令は「add」や「rts」などのニーモニックです．R8Cは16ビット機ですが，8ビット処理も同等にこなせるので，movやaddなどのデータ系の命令は.w（word＝16ビット）か.b（byte＝8ビット）を付けてデータの大きさを指定します．

オペランドにはレジスタやラベルがきます．オペランドに数値が直接書かれる場合は「#」，レジスタ間接・相対指定の印には角かっこ[]が用いられます．R8Cのアセンブラの書き方では，オペランドは転送元，転送先の順に書き，x86系のアセンブラとは逆順です．

### ● データ領域の定義を行う指示命令

データ領域の定義には「指示命令（directive）」が使われます．指示命令はニーモニックの位置に書きますが，機械語には変換されず，1バイト，2バイト…や文字列の領域を指定します．つまり，ラベルは位置ですから，文法上，何か位置を占めるものが必要なので，疑似的な命令を仮に書いておく感じです（**表7－1**）．

領域確保以外の指示命令もあります．シンボルと呼ばれる定数の定義，条件アセンブル，マクロなどです．特にマクロは便利なしかけなので，マクロが使えるアセンブラは「マクロ・アセンブラ」と呼ばれます．

### ● 一続きのメモリ領域「セクション」

一続きのメモリ領域がセクションです．記述が別々でも，同名のセクションは一続きにまとめられます．セクションの属性には，CODE，ROMDATA，DATAがあり，それぞれ，プログラム，固定データ，変更可能なデータを記述します．CODEとROMDATAはROMに，DATAはRAMに配置されるように，.org指示命令でアドレスを指定します．

何も指定しなければコード領域のみとなり，RAMが使えませんから，データ・セクションは必要です．

スタートアップ・ファイルにセクションの詳細な記述があり（sect30.inc），コンパイラはそのセクションを利用します．

### ● 条件アセンブル

コンパイラなどのオプションの設定により，アセンブルする部分を変更するためのしかけです．純正のスタートアップ・ファイルにもR8C専用部分があります．

## Keyword 2　指示命令（directive）

領域確保や条件アセンブルなど，ニーモニックの位置に書かれ，アセンブラ自体の動作を指示します．疑似命令（pseudo instruction）と呼ばれることもあります．

逆に言えば，アセンブラへの指示は，何でも指示命令になってしまいます．つまり，ニーモニックから機械語への変換以外のアセンブラの能力そのものです．

初期の8ビット・マイコンの頃は単純でしたが，最近のアセンブラの指示命令は，CPUやOSの発展に対応して高度化しているようです．本連載で使用しているアセンブラも，もちろん現代的なアセンブラです．

微妙な違いはあれ，基本的なところは変わらないので，R8Cのアセンブラで学習しても，他のアセンブラでその知識が役に立ちます．

高級言語と同じく，練習が必要なので，簡単なプログラムで実働するものを作っておくことをお勧めします．