[第3章]

使いやすいアセンブラと効率の良いC言語

AVRのソフトウェア開発方法と環境

MCUをシステムに仕立てるには，ハードウェアの開発もさることながらソフトウェアの開発がなくてはならないものです．AVRのソフトウェアを開発するにはマシン語に対応したニモニック・コードをマシン語に変換するアセンブラを用いて行う方法，高級言語と呼ばれるCプログラムなどをマシン語に翻訳するCコンパイラなどを用いる方法があります．

アセンブラの説明では，第4章以降を読むのに最低限必要な項目をまとめていきます．詳細についてはアトメル社からリリースされているアセンブラ・マニュアルを参照ください．アドレッシング・モード，プログラムの書式，命令コードの概要と，誤りやすいと思われる事柄を織り込んで説明していきます．実際のアセンブラの使用法は，第1章のデモンストレーションと第5章 AVRStudioのアセンブラの項目を参照願います．

C言語はすでにAVRユーザの過半数の方が使用していると思われます．多くの8ビットMCUでは，Cで書くことはできても実際システムにするとコード・サイズがとても膨らんでしまったり，速度の著しい低下を招いたりして実用には供さないと称される製品が多いように聞いていますが，AVRはCとの相性がよく，Cを利用したシステム構築が可能です．

本書では，筆者の好みでイメージクラフト社のIccAVRを用いて，組み込み型コンピュータに特有なフォームと注意点を主に説明していきます．

3-1　アセンブラでのプログラミング

最近筆者は，初期設定などのわずらわしさから，アセンブラを使うことが少なくなり，C言語でプログラムを書くことが多くなっています．しかし，今回この本の執筆を依頼されて，じっくりと命令セットを見ていった結果，AVRが非常にわかりやすい命令体系をとっていることを再認識しました．

通常アセンブラでは，全命令の3割程度の命令しか使われていないといわれているようですが，AVRでは100を超える命令があっても縦横に命令を駆使できるのではないかと感じています．それは，命令の構成が非常にシンプルで，目的にあった命令を探しやすいのが大きな要因に思えます．とくにブランチ命令などでは，I/Oの状況をすぐにシーケンスに反映できるなど，短いステップで効率良くプログラムを書くことができ，AVRの吟味されたメカニズムによく対応していることが実感されます．

(1) アセンブラ・ソース

アセンブラ・ソース・コードのプログラム例を**リスト3-1**に示します.

アセンブラのステートメントは，下記のように記述します.

```
label:          directive       op          ; コメント
label:          instruction     op1, op2    ; コメント
```

命令(instruction)のオペランド，op1，op2は，op1 ← op2という方向で見ます.

すなわち，データ転送命令のmovを考えると，

リスト3-1　LEDを順次点灯するサンプル・プログラム

```
; program3.asm
; 基本的要素を含むプログラム例
; STK500のLEDを右から左へ順に点灯，これを繰り返すプログラム
; I/Oポート(出力ポート)の初期化
; サブルーチンを使用するためにスタック・ポインタの初期化
; サブルーチン・コール
; などを含む

.include "m8def.inc"                ; 定義ファイルをインクルード
.def Temp = R16                     ; レジスタにシンボル名をつける擬似命令
.def CNT1 = R17                     ①
.def CNT2 = R18
                                    ; 空白行を入れることも可能
.org 0x0000                         ; プログラム・コードの始まるロケーションを宣言
        rjmp    RESET               ; リセット時にプログラムはここからスタートする
; 通常ここに割り込みベクタを定義する(割り込みを使用していないので未定義)

Delay1: ser     CNT1                ; ディレイ・サブルーチンのエントリ，CNT1 ← 0xFF
Loop1:  ser     CNT2                ; CNT2 ← 0xFF
Loop2:  dec     CNT2                ; カウンタ値をマイナス1，結果が0になるとSREG:Z=1
        brbc    1, Loop2            ; SREGのビット1(Z)がクリアされていればLoop2へブランチ
        dec     CNT1
        brbc    1, Loop1
        ret                         ; サブルーチン・リターン

RESET:                              ; ラベル
        ldi     Temp, low(RAMEND)   ; RAMの最終番地をバイト単位で
        out     SPL, Temp               ; スタック・ポインタへセット
        ldi     Temp, high(RAMEND)      ; RAMENDは各.defファイルの中で定義されている
        out     SPH, Temp
        ser     Temp                    ; Temp ← 0xFF
        out     DDRB, Temp              ; PORTBのデータ・ディレクション・レジスタを出力にセット

Loop:
        sbis    PORTB,7             ; PORTBのMSBがセット(1)ならスキップ
        ser     Temp                ; Temp ← 0xFF
        out     PORTB, Temp         ; Tempレジスタのデータを PORTBにセット
        rcall   Delay1              ; ディレイ・サブルーチンを相対コール
        lsl     Temp                ; Tempレジスタのビットをそれぞれ左へ1ビット・シフト
        rjmp    Loop
```

```
        mov     Rd, Rr
```

と記述でき，Rr(ソース)レジスタの内容をRd(ディスティネーション)レジスタに転送することを意味しています．

さらにいくつかの例を示します．

```
        sub     Rd, Rr   ;  Rd ← Rd-Rr
        add     Rd, Rr   ;  Rd ← Rd+Rr
```

減算，加算の結果がディスティネーション・レジスタにストアされます．

リスト3-1中の①にあるように，擬似命令(directive)は，“ .def ”などのように“ . ”で始まります．def はレジスタにシンボル名をつける擬似命令です．擬似命令はオペレーション・コードには直接変換されません．メモリのプログラム・ロケーションを調節したり，マクロを定義したり，メモリの初期化を行ったりするときに使用します．擬似命令に関してはこの節の(4)を参照してください．

そのほか，アセンブラのソース・コードを記述する要素として，演算子，関数があります〔節(5)参照〕．

(2) アドレッシング・モード

AVRのメモリ・マッピングを第2章の2-1で述べましたが，それぞれのメモリ・スペースをどのようにしてアクセスするのかといった，アクセス方法をここでは説明していきます．

表3-1にアドレッシング・モードをまとめました．

大きく分けて五つのアクセス・モードがあります．

① レジスタ間同士でのアクセス

レジスタ間同士のアクセスでは，32バイトの汎用レジスタの内容に直接操作ができます．これは第1章の例題でも見てきました．シングル・レジスタと二つのレジスタを扱う二つのモードがあります．

② I/Oレジスタとレジスタ間でのアクセス

I/Oレジスタを扱う命令はIN，OUT命令です．入力レジスタからのデータを汎用レジスタに読み出す，あるいは逆に汎用レジスタの内容を出力レジスタに書き出す命令ですが，I/Oとして指定できる範囲が0～63番地までに限られています．この範囲を越える拡張I/Oレジスタは，IN/OUT命令を使うことができません．

64番地以上のI/Oレジスタを扱うには，メモリ・アクセスに使われるLDS/STS命令を使います．このとき，I/Oアドレスとして割り当てられているアドレスに汎用レジスタの0x20番地を加えた番地を使います．

データ・シートの末尾にはRegister Summaryとしてまとめられた表がありますが，ここのAddressの欄に書かれている括弧のついていないアドレスがIN/OUT命令に使用するアドレス，( )でくくられたアドレスがLDS/STS命令でアクセスする場合に使用するアドレスになります(**図3-1**参照)．

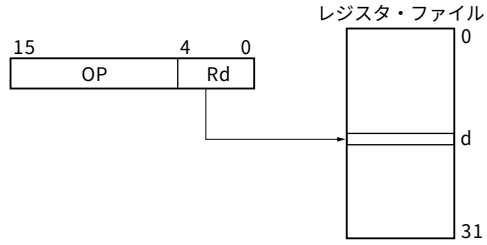
③ レジスタとRAM空間でのアクセス

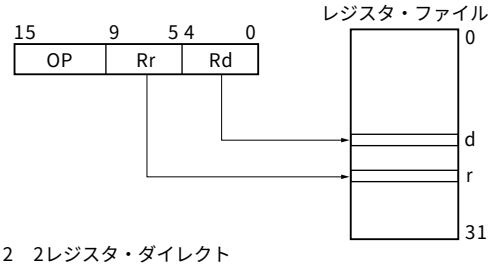
SRAMデータをアクセスするモードです．

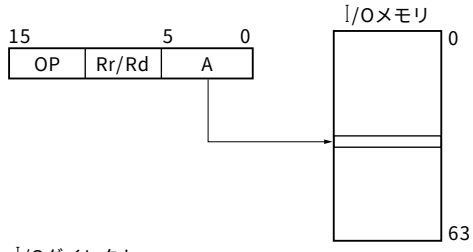
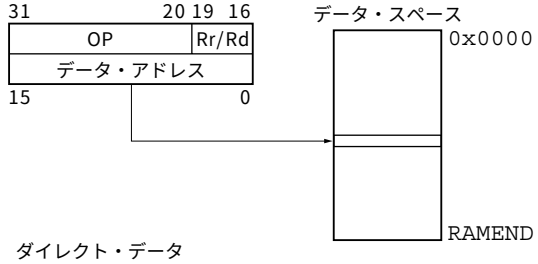
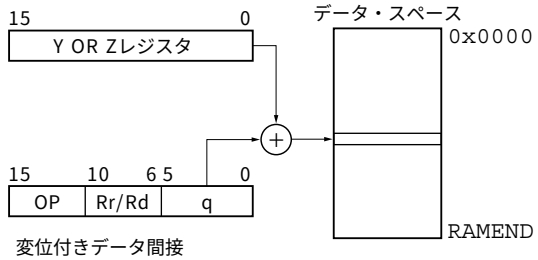
RAMの内容をアクセスするときに，命令の2ワード目に書かれたデータ・アドレスで，直接データのアドレスをポイントできるのがダイレクト・データ・モードです．

AVRでは，汎用レジスタの二つのレジスタをペアにしてポインタとして使うことができます．R26，R27をXレジスタ，R28，R29をYレジスタ，R30，R31がZレジスタと定義されていますが，このポ

表3-1　アドレッシング・モード

モード	例と注意	
 1　シングル・レジスタ・ダイレクト	INC R11 ; R11=R11+1 DEC R12 ; R12=R12-1 COM R8 ; R8=0XFF-R8 NEG R2 ; R2=0X00-R2	①


2　2レジスタ・ダイレクト
	ADD R16,R17 ; R16=R16+R17
	ADC R6,R7 ; キャリを含めた加算
	SUB R7,R6 ; R7=R7-R6
	CP R5,R6 ; R5-R6を演算し，ス
	; テータス・レジスタに
	; 結果をセット
	AND R16,R17 ; R16=R16・R17	

 3　I/Oダイレクト	OUT DDRB,R16 IN R17,PIND I/Oレジスタ・アドレス範囲は0～0x3F. これを超える場合，STS,LDS命令を使用(この場合，I/Oアドレスに0x20を加える)	②
 4　ダイレクト・データ	LDS R3,0x00FF STS 0x0080,R3 命令に含まれる(2ワード命令)アドレス番地の内容を直接アクセス	③
 5　変位付きデータ間接	LDD R4,Y+2 STD Y+5,R4 Y,Zポインタで示されるメモリ・スペース＋変位アドレスのデータをアクセス	