PIC プログラミングの考え方をシミュレータを使って理解する

# プログラミングは手順の分解

# 光永 法明

プログラミングとは、やりたいことを手順に分解していくことです.小学校で掛け算や割り算を筆算で 九九ができる単位に分けて計算していったように、少しずつ簡単な手順に分けていくことなのです.本章 では、パソコン上のシミュレータを使って、PICがどのように動作するのかを見ていき、プログラミング のこつを紹介していきます.ぜひシミュレータを使ってPICの動作を自分で確かめてください.

## 3-1 シミュレータを使ってみる

シミュレータは,実際のPICの内部の動作をパソコン上で模擬してくれるものです.これを使うと,普通は見えないPICの内部動作が目に見えるようになります.MPLABのインストール,シミュレータの使い方は Appendix A を見てください.

まずは足し算をPICで実行するシミュレーションをしてみましょう.プログラムは**リスト**3-1のとおりです.エディタ・ウィンドウで入力してください.

行頭の空白はタブです.スペース・バーではなく,タブ・キーで入力します.入力が終わったら,アセンブルです.Project タブから Make を選ぶかファンクション・キー F10を押します.

図3-1のように, BUILD SUCCEEDEDと表示されればアセンブルは成功です.うまくアセンブルで きない場合には,文字が間違っていないか見直してください.

リスト3-1 足し算プログラム

	list #inclu	p=16f877a de p16f877a.inc	; ;	PIC16F877A 用のプログラムであることを宣言 PIC16F877A 用のヘッダ・ファイルを読み込む
REGA	equ	0x21	ד צ	ドレス0x21のレジスタをREGA いう名前をつけて使う
main				
	movlw	0x1	;	w レジスタに1を代入( w = 1 )
	addlw	0x2	;	w レジスタに 2 を加算( w = w + 2 = 1 + 2 )
	movwf	REGA	;	REGA に w レジスタの値を代入( REGA w = 3 )
STOP	HERE			しここを
	goto	STOP_HERE	;	スリープ命令( プログラムはここで止まる)) ∫ずっと 繰り返す
	end		;	プログラムの最後には必ず書く



### 図3-1 うまくアセンブルできたときのOutput ウィンドウ

MPLAB でアセンブルするとOutput ウィンドウが表示される. アセンブルに成功したときには"BUILD SUCCEEDED "と表示されるので確認する.また タイプ・ミスなどではエラー(error)や警告(warning)が表示される.



#### 図3-2 MPLAB SIM にチェック・マークをつける

シミュレータを使うにはDebugger Select Tool で選択できる" MPLAB SIM "を選択し, チェック・マークが表示された状態にしておく.

Special Funct	ion Registers									
Address⊽	SFR Name	Hex	Decimal	Binary	Char 🔺					
	WREG	00	0	00000000						
0000	INDF		<b>+</b> –		-					
0001	TMRO	00	0	0000000						
0002	PCL	00	0	0000000						
0003	STATUS	18	24	00011000						
0004	FSR	00	0	0000000						
0005	PORTA	00	0	0000000						
0006	PORTB	00	0	0000000						
0007	PORTC	00	0	0000000	. 🔻					

(0であることを確認する)

図3-3 Special Function Registers(特殊機能レジスタ, SFR) )ウィンドウを見る SFRを表示させることができる.ウィンドウ内にはSFRのアドレス,名前,値(16進,10 進,2進と文字)が表示される.

つぎに, MPLAB SIM が使えるようになっているか確認しましょう.図3-2のように, Debugger Select Tool 3 MPLAB SIM のところにチェック・マークがついていればOK です.

View タブから, Special Function Registersをクリックし, ウィンドウを出しましょう(図3-3). この ウィンドウを見るとwレジスタ(WREG)の値は0ですね.0でなかったらWREGのHEX(16進表記)か Decimal(10進表記)の数字をクリックして,0を入力します.もう一つ, View タブから, File Registers ウィンドウを出しましょう(図3-4).

🔲 File Regis	ters																_	
Address	00	.01	02	03	04	05	06	07	08	09	OA	OB	OC	OD	OE	OF	ASCII	<b></b>
0000		03	00	18	00	00	00	00	00	00	00	00	00	00	00	00		
0010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
,0020,	00)	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0030	00	00	00	00	00)	00	00	00	00	00	00	00	00	00	00	00		
0040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0080		FF	00	18	00	ЗF	FF	FF	FF	07	00	00	00	00	00		?	
0090		00	FF	00	00				02	00			07	00	00	00		
0040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
OOEO	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
OOFO	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0100		03	00	18	00		00				00	00	00	00	00	00		-
Hex Syn	polic																	
$\geq$																	L .	
表示方法	を変	E更	) (	0x	21	σι	ッジ	スタ	ョは	00	20	)				文字	にすると、どんな文字	
するタブ	-		儿	の行	ĴΟ	01	თკ	- T	37	を見	る	J			1	かを	表示してくれる	J

図3-4 File Registers ウィンドウ(実行前)

リスト2-1のプログラムを実行する前にファイル・レジスタを表示させたところ.表の見方 としてはファイル・レジスタ・アドレスを16進数で表記したときの上の3桁の行の,下1 桁の値が一致するところを見る.

File Reg	isters											000					and the second	_ 🗆 ×
Address	s 00	01	02	03	04	05	06	07	08	09	OA	OB	0C	OD	OE	OF	ASCII	<b></b>
0000		00	(03)	(18)	00	00	00	00	00	00	00	00	00	00	00	00		
0010	00	00	<u> </u>	00)	δą	00	00	00	00	00	00	00	00	00	00	00		
0020	00	03	00 (	φo	00	ØÖ	00	00	00	00	00	00	00	00	00	00		
0030	00	<i>†</i> oo	00	ρo	00	00	00	00	00	00	00	00	00	00	00	00		
0040	00	00	00	00/	00	00	ÒQ	00	00	00	00	00	00	00	00	00		
0050	οø	00	00	00	90	00	00	Ø	00	00	00	00	00	00	00	00		
0060	- 90	00	00	00	90	00	00	00/	00	00	00	00	00	00	00	00		
0070	φo	00	00	00	ρo	00	00	00	ÔQ	00	00	00	00	00	00	00	•••••	1000
0080	/	FF	$\underline{03}$	(18	roo/	ЗF	FF	FF	FF	Q7.	00	00	00	00	00		?	
0090	/	00	FF'	90	00	1-			02	οQ			07	00	00	00		
00400	/ 00	00	00	ଂକ୍	00	60	60	00	00	00	99	00	00	00	00	00		
0080	00	00	00	00,	00	90	00	00	~	00	oq	00	00	00	00	00		
0000	00	00	00	00	20	90	00	00	00	9	00	20	00	00	00	00	•••••	
0000	00	00	00	00	00	40	00	00	00	00	20	UN.	00	00	00	00		
UUEU	00	00	00	00	00	77	00	00	00	00	00	200	00	00	00	00		
UUFU	00	00	ő		00	461	00	υu	υu	υu	00	UN	NU.	00	00	00		
0100		00	03		-	-+	UU				UU	UU	Nh.	UU	υu	UU		•
Hex 5	ymbolic			_	_							-	$\mathbb{Z}$					
			_								_	_	$\supset$					
(0x031	こ変化	Ł)	P	CL	レシ	バス	タ(	0x	c O C	02	2)	S	ΤĀ	ΤU	s٦	ッジ	スタ(0x0003,	
している	3	J	0	x0	08	2,	02	c01	L02	2)	J	0	x0	08	з,	02	x0103)	

図3-5 File Registers ウィンドウ(実行後)

リスト2-1のプログラムを実行後のファイル・レジスタの値.

前回にレジスタを表示していたときから変化があったところは赤く表示される.

では早速実行してみましょう. Debugger タブから, Runの中のRunをクリックするか, ファンクション・キー F9を押します.すぐに, このプログラムは終わるので, Debug タブから, Runの中のHaltをクリックするか, ファンクション・キー F5を押し, シミュレーションを止めます.

Special Function Register ウィンドウの w の値と File Registers ウィンドウの 0x21 アドレスの値が 3 になりましたね(図3-5). Symbolic タブをクリックすると図3-6のようにレジスタの名前が表示されるの でわかりやすいかもしれません.

リスト2-1のプログラムは,1+2を実行してwとREGA(0x21)に代入するプログラムです.数字を適当に変えてアセンブル,実行してみてください.wとアドレス0x21の値が変わりますね.

🔲 File Regist	ers					
Address	Hex	Decimal	Binary	Char	Symbol Name	<u> </u>
0015	00	0	00000000		CCPR1	
0016	00	0	00000000		CCPR1H	
0017	00	0	00000000		CCP1CON	
0018	00	0	00000000		RCSTA	
0019	00	0	00000000		TXREG	
001A	00	0	00000000		RCREG	
001B	00	0	00000000		CCPR2	> レジスタの名前
001C	00	0	00000000	•	CCPR2H	
001D	00	0	00000000	•	CCP2CON	
001E	00	0	00000000	•	ADRESH	
001F	00	0	00000000	•	ADCONO	
0020	00	0	00000000	•		
0021	03	3	00000011		REGA .	)
0022	00	0	00000000	•		
0023	00	0	00000000	•		
0024	00	0	00000000	•		
0025	்	Ŷ	00000000	÷		<b>~</b>
Hex Sym	ibolic	t t				
					)	

(アドレス)(16進数)(10進数)(2進数)(文字)

図3-6 File Registers **ウィンドウ**(実行後,シンボリック表記) ファイル・レジスタの値は別の表示方法も選べる.こちらは一覧性はないが,SFRの表記 のように一つの値については見やすい.

今度はステップ実行を試してみましょう.

Debug タブから, Runの中のReset をクリックするかファンクション・キーF6を押します.これで PICがリセットされた状態になります.次はDebug タブから, Runの中のStep Intoをクリックするかフ ァンクション・キーF7を押します.Step IntoのクリックまたはF7を押すたびに1命令ずつ実行されま す.goto命令のところでは,同じアドレスにジャンプしているので,プログラムは先へは進みません. 最後にアニメーション実行を試しましょう.

Debug タブから, Run の中の Reset をクリックするかファンクション・キー F6 を押し, リセットします.次は Debug タブから, Run の中の Animate をクリックします. Debug タブから, Run の中の Halt を クリックするか, ファンクション・キー F5 を押すとアニメーション実行は止まります.

三つのシミュレーションいずれも,リセットしなければ,止まったところから,シミュレーションを再開します.止まっているときには,レジスタの値を変えて動作がどうなるかを見ることもできます.たと えば,wレジスタの値を変えて2+3や4+3の結果を見ることができます.

うまくいかないときは, MPLABをいったん終了し, ダウンロード・サービス(http://mycomputer. cqpub.co.jp/の本書のタイトルからリンク)から入手したファイルをプロジェクトの作成したファ イルに上書きして, MPLABを起動し直し, 試してください.

## Column 1 プログラムの最後で無限ループにしているのは

プログラムの最後で無限ループにしているのは, 次の行を実行してほしくないからです.

マイコンは常に次の命令を実行しようとします. プログラムを書いていないメモリ領域であっても実 行します.PICの場合にはプログラムを書いていな い領域は addlw 0xff 命令が並んでいるため,そ れが実行されます.

これを防ぐには,無限ループで停止するのが簡単 なのです.

実際のプログラムでは何かを実行しつづけるの で,何もしない無限ループで停止にすることは少な いです.

## 3-2 アセンブラに慣れよう

ちょっと,わざとエラーを起こしてみましょう.

行頭の空白を消してみる.

ラベルを1文字変えてみる.

listの行のプロセッサ名を変えてみる.pic16f877zではどうでしょうか.

includeの行のファイル名を変えてみる.pic16f877xではどうでしょうか.

命令を間違えてみる.movlwをmovfwにしてみましょう.どうなりますか.

命令を間違えてみる.movlwをmovlfにしてみましょう.

同じ名前のラベルをつくってみる.movlwの行頭にREGAと書いてみましょう.

同じ名前のラベルをつくってみる.movlwの行頭にSTOP HEREと書いてみましょう.

全角英数字や空白で記述してみる.

最後のgotoをなくしてみる.

endをなくしてみる.

それぞれに,どういうエラー・メッセージが出ましたか.これで MPASM のエラーが出たら,どう直 せばよいかわかりますね.movlwとmovfwを間違えても,最後のgotoをなくしてもエラーになりま せんが,動作がおかしくなります.View タブから, Program Memory ウィンドウを出して,ステップ実 行するとよくわかります.気をつけましょう.

## 3-3 加減算と論理演算をシミュレータで実行してみる

加減算と論理演算を実行して,STATUSレジスタの変化を見てみましょう.

リスト3-2をシミュレータでステップ実行してみてください. Special Function Registers ウィンドウを 見ると,<u>キャリ・フラグ</u> ②(<u>ボロー・フラグ</u> ②, STATUS<0>)と<u>zフラグ</u> ③(<u>STATUS<2></u>)が変化し ていくのがわかると思います.計算の順序を変えたり先に STATUS レジスタ内のフラグを変えたりして, ちゃんと計算によってフラグが変化していることを確かめてみてください.

プログラムはそのままでレジスタの値を変えて試すこともできるので,いろいろ試してみてください.

## 3-4 比較-基本は引き算-, 条件分岐にはビット・テスト

1と2はどちらが大きいかは,すぐにわかりますね.でもPICで1と2ではどちらが大きいかは,どうやって調べるのでしょうか.

そういうときには,引き算をして,結果が0か,正か,負かをSTATUSレジスタで調べます.0なら, 両者は一致,正なら引かれたほうが大きい,負なら引かれたほうが小さいということがわかります.

数字が大きい場合には REG\_LARGER に 1 を,一致する場合には REG\_EQUAL に 1 を,小さい場合には REG\_SMALLER に 1 を代入するプログラムは**リスト**3-3のようになります.

プログラムでは,まずREG\_LARGER,REG\_EQUAL,REG\_SMALLERを0にします.これはファ イル・レジスタにリセット時には,どんな値が入っているかわからないからです.つぎに2と1を比較す

	list #include	p=16f877a e p16f877a.inc	; PIC16F877A 用のプログラムであることを宣言 ; PIC16F877A 用のヘッダ・ファイルを読み込む
REGA	A equ	0x20	
	bsf movlw	STATUS, Z 0 ←	; STATUS レジスタの z フラグを 1 にする 
	movlw addlw	D'1' D'1'	<pre>} w = 1 + 1 = 2</pre>
	movlw addlw	D'1' -D'1'	<pre>} w = 1 - 1 = 0 , z = 1</pre>
	movlw addlw	D'255' D'1'	<pre>} w = 255 + 1 0 , C = 1</pre>
	movlw sublw	D'2' D'1'	$\left. \begin{array}{l} \\ \end{array} \right\} w = 1 - 2 = -1 , C = \overline{BO} = 0$
	movlw sublw	D'1' D'1'	<pre>} w = 1 - 1 = 0 , z = 1</pre>
	movlw sublw	D'1' D'2'	<pre>} w = 2 - 1 = 1 , C = BO = 1</pre>
	movlw andlw	0x0f 0xff	<pre></pre>
	movlw iorlw	0x0f 0xff	<pre> } 論理和 w = 0x0f OR 0xff = 0xff ←</pre>
	movlw xorlw	0x0f 0xff	
	movlw movwf comf	0x0f REGA REGA, w	否定       1111 0000         w レジスタで演算でき       0000 1111         ないので REGA に代入・       0 0 と1 が入れ替わる         してから演算する       1111 0000
	goto	\$	- (ここで無限ループで止まる)
	end		

るため,2-1の引き算を実行しています.そして,引き算の結果変化した STATUS レジスタをチェック して,該当するレジスタを1にしています.

まず, zフラグをチェックします.zフラグが0ならば,次の行(goto)はスキップされ,1(引き算の 結果が0)ならgotoが実行されます.2-1では0にならないので,スキップして,次はCフラグをチェ ックします.Cフラグは引き算の場合ボロー・フラグの反転なので,借り出しがあれば0,なければ1です.