

このソフトはマイクロチップ製PICマイコン用のアセンブラです。
動作環境はMS-DOSが走るマシンなら機種を問いません。

■起動方法

前もって必要なファイルをHDD等にコピーしておきます。

DOSプロンプトで

```
C>pa [オプション] <ソースファイル>
```

ソースファイルに作成したファイルを指定してください。
拡張子は省略するとASMが自動に付加されます。

◆オプションについて

- l ソースのリストファイル出力します。
拡張子は.lst です。

■文法

[ラベル] [ニーモニック] [オペランド],[オペランド].. ;[コメント]

コメント以外はすべて半角文字で書いてください。

「ラベル」は1カラム(1桁目)から書き始めなければなりません。
ラベルには'a-z'のアルファベットの他に,'_','0-9'が使用できます。
(数字は2文字目以降)予約語はラベルとして使用できません。
大文字・小文字は区別します。

「ニーモニック」はタブでインデントしてから、又は2カラム以降に書いてください。
マイクロチップのインストラクションと、PAインストラクションが使用できます。
大文字・小文字は区別しません。

「オペランド」はニーモニックで指定された数を書いてください。
オペランドには式を置くことができます。

「コメント」は';'(半角)文字以降、行末までです。
漢字も使用できます。

◆定数

分類	説明	記述例
10進数	そのまま記述すればOKです	10,100,-123
16進数	値の末尾に'h'を付加します 頭が'a-f'で始まる16進数は ラベルと区別するため、頭に'0'を つけます。	10h,12h,80h 0a0h,0ffh

2進数 値の末尾に'b'を付加します 1000b,1111b,1010b

文字定数 文字列を'(シングルコーテーション)
で囲みます。囲った文字のASCII
コードが定数となります 'A','a','Z'

文字列定数 文字列を'(シングルコーテーション)
で囲みます 'Hello!','sample'

\$ アセンブラのプログラムアドレスを goto \$+1
示します

% アセンブラのデータアドレスを示します mov d_adr,%

オペランドの即値データを表わします mov ra,#100

◆演算子

種類	説明	記述例
()	括弧	mov fr,#2*(3+4)
~	全ビット反転(NOT)	mov fr,#~data
&	ビットAND	mov fr,#cont & mask
	ビットOR	mov fr,#cont bit
^	ビットXOR	mov fr,#cont ^ mask
*	乗算	mov fr,#word*2
/	除算	mov fr,#data/4
+	加算	mov fr,#count + 100
-	減算	mov fr,#count - 100
<<	左ビットシフト	mov fr,#data<<4
>>	右ビットシフト	mov fr,#data>>4
.	ビット修飾子	bsf fr.bit

演算子は優先順位の高い順に表記してあります。

◆アセンブラ擬似命令

命令	オペランド	説明・詳細
include	{filename}	filename で指定されたファイルをインクルードします。

ディレクトリ指定がない場合はカレントディレクトリから読み込みます。アセンブル時にPICそれぞれのシンボルの設定が必要ですので、各PICに用意されたヘッダファイルをインクルードしてください。

ファイル名に(*.H)に'x'の文字が含まれている場合はインクルードした後にデバイス設定(次項参照)が必要です。(シリーズ共通ヘッダファイルとなっているため)

<例>

- include 16f84.h ;PIC16F84をインクルードと
; 同時にPIC16F84 にセット
-

	include	16c5x.h	;PIC16C5xをインクルード	eedata	{value[,value..]}	データEEPROMの初期値をセットします。 この命令はPIC16x84のみ有効です。 この命令でライターの書き込みと同時にデータEEPROMの内容を書き込みます。
			; この状態ではまだデバ			
			; スが設定されていません。			
		.16c57	;PIC16C57にデバイスセット			
.device			デバイス設定。この命令は省略できません。	<例>		
				eedata	0,1,'SAMPLE'	
	<例>					
		.16c84	;PIC16C84	eeorg	{value}	データEEPROMの内部アドレス値をセットしま
		.16f84	;PIC16F84	す		
		.12c508	;PIC12C508			この命令はPIC16x84のみ有効です。 アセンブラ内部のEEPROMアドレス値を変更します
.osc	{type}		発振タイプを設定します。 セットできるタイプは以下の通りです	<例>		
		LP	ローパワー発振子	eeorg	8	
		HS	高速クリスタル	eedata	'Sample'	; アドレス8hから定義します
		RC	RC発振			
		XT	クリスタル・セラロック	org	{value}	現在のプログラムアドレス・ファイルレジスタアドレ
		EXTRC	外部RC発振(12c5xxのみ)	スの		値をセットします。使用するデバイスによって value に設定
		INTRC	内蔵RC発振(12c5xxのみ)			できる最大値が異なります。指定がない場合は0から始まり
	<例>					ります。
		.osc	xt	equ	{value}	シンボルをセットします。 一度指定したシンボルは equ, = 文では再定義できません。
		.osc	intrc			<例>
.wdt	{on off}		WDTの有効・無効の設定 省略時はONです。	limit	equ	100
	<例>			outpin	equ	ra.0
		.wdt	off			
.pwrt	{on off}		パワーアップタイムの有効・無効の設定 PICによってはこの機能がないものがあります。	mov	rb,#limit	
	<例>			setb	outpin	
		.pwrt	off	=	{value}	シンボルをセットします。 = 文は何度でも再定義できます。
.mclre	{enable disable}		マスタークリア(MCLR)有効・無効の設定 (12c5xxのみ) 省略時はENABLEです。	<例>		
	<例>			limit	=	100
		.mclre	disable	limit	=	120 ; 再定義OK
.protect	{on off}		プロテクト設定 省略時はOFFです。	ds	{value}	領域を指定したバイト数だけ確保します。 (内部 org 値を指定した値分加算します)
reset	{label}		プログラムの開始アドレスをセットします。 この命令はPIC16C5xシリーズのみ有効です。 5xシリーズはリセット時PCが最上位アドレスを 指している為、この擬似命令でスタートアドレスを 指定します。	<例>		
	<例>			org	0ch	
		reset	start	cn	ds	1
		org	0	work	ds	1
		start		data	ds	2
				null	ds	0 ; 0でもエラーにはならない
				id	{value}	PICマイコンのIDを設定します。 (各ICのシリアル番号・バージョン管理等に使用します) 指定した値はライターでPICに書き込まれます。 0000h~FFFFhまで指定可能です。
	<例>					<例>

PICアセンブラインストラクションセット

■マイクロチップのインストラクションの他に、下記の拡張インストラクションも使用可能です。

『拡張インストラクションセット』

拡張インストラクションとは、通常のインストラクションをいくつか組み合わせることで特定の処理をする命令を独自に付け加えたものです。例えば比較分岐をする場合、通常のインストラクションでは引き算をして結果をステータスレジスタに反映させた後、そのフラグの変化で分岐をするコードをプログラマがコーディングしなければなりません。特に値の大小(>,<,>=,<=)を比較する場合はコードがややこしくなってしまう。拡張インストラクションはこれを1命令とみなしてプログラム出来るため非常に効率が上がります。(アセンブル時に複数個の命令に展開されます)アセンブラでは拡張インストラクションを意識することなくコーディングできます。

◆オペランドの略号説明

fr...File Register	ファイルレジスタ	ra,fsr,rtcc,...
lite...Literal	定数リテラル	10,0ffh,0ah,...
W...W Register	Wレジスタ	
bit...Bit Filed	ビット位置	status.2
cf...Carry Flag	キャリーフラグ	
zf...Zero Flag	ゼロフラグ	

[PA.EXE Instruction Set]

inst.	operands	Description	words
ニーモニック	オペランド	説明	必要ワード数
add	fr,#lite	fr+=lite	2
add	fr1,fr2	fr1+=fr2	2
add	fr,W	fr+=W	1
add	W,fr	W +=fr	1
addb	fr,bit	fr+=(bit==1) ? 1: 0	2
and	fr,#lite	fr&=lite	2
and	fr1,fr2	fr1&=fr2	2
and	fr,W	fr&=W	1
and	W,fr	W&=fr	1
call	addr8	(*addr8)()	1
サブルーチンをコールします			
cja	fr,#lite,addr9	if(fr>lite) goto addr9	4
比較して結果が真ならジャンプします			
cja	fr1,fr2,addr9	if(fr1>fr2) goto addr9	4

cjae	fr,#lite,addr9	if(fr>=lite) goto addr9	4
cjae	fr1,fr2,addr9	if(fr1>=fr2) goto addr9	4
cjb	fr,#lite,addr9	if(fr<lite) goto addr9	4
cjb	fr1,fr2,addr9	if(fr1<fr2) goto addr9	4
cjbe	fr,#lite,addr9	if(fr<=lite) goto addr9	4
cjbe	fr1,fr2,addr9	if(fr1<=fr2) goto addr9	4
cje	fr,#lite,addr9	if(fr==lite) goto addr9	4
cje	fr1,fr2,addr9	if(fr1==fr2) goto addr9	4
cjne	fr,#lite,addr9	if(fr!=lite) goto addr9	4
cjne	fr1,fr2,addr9	if(fr1!=fr2) goto addr9	4
clc		cf = 0	1
clr	fr	fr = 0	1
clr	W	W = 0	1
clrb	bit	bit = 0	1
clz		zf = 0	1
csa	fr,#lite	if(fr>lite) skip	3
比較して結果が真なら次の命令を			
csa	fr1,fr2	if(fr1>fr2) skip	3
スキップします(次にくる命令は			
csae	fr,#lite	if(fr>=lite) skip	3
1ワード命令でなければなりません)			
csae	fr1,fr2	if(fr2>=fr2) skip	3
csb	fr,#lite	if(fr<lite) skip	3
csb	fr1,fr2	if(fr1<fr2) skip	3
csbe	fr,#lite	if(fr<=lite) skip	3
csbe	fr1,fr2	if(fr1<=fr2) skip	3
cse	fr,#lite	if(fr==lite) skip	3
cse	fr1,fr2	if(fr1==fr2) skip	3
csne	fr,#lite	if(fr!=lite) skip	3
csne	fr1,fr2	if(fr1!=fr2) skip	3
dec	fr	fr--	1
decsz	fr	if(--fr==0) skip	1
djnz	fr,addr9	if(--fr!=0) goto addr9	2
fr から 1 を引き 0 以外ならジャンプする			
ijnz	fr,addr9	if(++fr!=0) goto addr9	2
inc	fr	fr++	1
incsz	fr	if(++fr==0) skip	1
jb	bit,addr9	if(bit==1) goto addr9	2
jc	addr9	if(cf==1) goto addr9	2
jmp	addr9	goto addr9	1
普通の無条件ジャンプです			
jmp	pc+W	pc+=W+1	1
テーブルの参照によく使います			
jmp	W	pc = W	1
jnb	bit,addr9	if(bit==0) goto addr9	2
jnc	addr9	if(cf==0) goto addr9	2
jnz	addr9	if(zf==0) goto addr9	2
jz	addr9	if(zf==1) goto addr9	2
lcall	addr11	(*addr11)()	3
PA0,1 をセットしたあとコールします			
ljmp	addr11	goto addr11	1
PA0,1 をセットしたあとジャンプする			
lset	addr11		3
PA0,1 をセットします			
mov	fr,#lite	fr = lite	2
mov	fr1,fr2	fr1 = fr2	2
mov	fr,W	fr = W	1
mov	option,#lite	option = lite	2
mov	option,fr	option = fr	2
mov	option,W	option = W	1

```

mov    !port_fr,#lite    tris_fr = lite    2    port_fr の入出力方向
をセットします
mov    !port_fr,fr      tris_fr = fr      2
mov    !port_fr,W      tris_fr = W      1
mov    W,#lite        W = lite        1
mov    W,fr          W = fr          1
mov    W,/fr         W = ~fr         1
mov    W,fr-W       W = fr - W       1
mov    W,++fr       W = fr + 1       1
mov    W,--fr       W = fr - 1       1
mov    W,<<fr       W = fr<<1       1
mov    W,>>fr       W = fr>>1       1
mov    W,<>fr       W = swap(fr)      1    fr の上位 4bit 下位
4bit を入れ替えます
movb   bit1,bit2     bit1 = bit2      4
movb   bit1,bit2     bit1 = !bit2     4
movsz  W,++fr       if(fr+1==0) skip  1
movsz  W,--fr       if(fr-1==0) skip  1
neg    fr          fr = -fr          2    2の補数をとります
nop    ;            ;                1    1クロック時間を消費します
not    fr          fr = ~fr          1    全ビット反転します
not    W           W = ~W            1    全ビット反転します
or     fr,#lite    fr|=lite         2
or     fr1,fr2     fr1|=fr2         2
or     fr,W        fr|=W            1
or     W,#lite     W|=lite          1
or     W,fr        W|=fr            1
ret    return      1    サブルーチンからもどります
retw   lite[!,lite...] return lite  1    Wに lite をセットして、
return します
rl     fr          fr<<=1           1
rr     fr          fr>>=1           1
sb     bit         if(bit==1) skip  1
sc     bit         if(cf==1) skip    1
setb   bit         bit = 1          1
skip   skip        1    次の命令を飛ばして実行しま
す。
sleep  sleep()     1    スリープモードに入ります
snb    bit         if(bit==0) skip  1
snc    bit         if(cf==0) skip    1
snz    bit         if(zf==0) skip    1
stc    bit         cf = 1            1
stz    bit         zf = 1            1
sub    fr,#lite    fr-=lite         2
sub    fr1,fr2     fr1-=fr2         2
sub    fr,W        fr-=W            1
subb   fr,bit      fr-=(bit==1) ? 1:0 2
swap   fr          fr = swap(fr)    1    fr の上位 4bit 下位 4bit を
入れ替えます
sz     bit         if(zf==1) skip    1
test   fr          fr = fr          1    ゼロフラグのみ変化します
xor    fr,#lite    fr = xor(lite)    1    排他的論理和をとります(C
にはない！)
xor    fr1,fr2     fr1 = xor(fr2)    2
xor    fr,W        fr = xor(W)       1
xor    W,#lite     W = xor(lite)     1
xor    W,fr        W = xor(fr)       1

```

▲上記命令で例えば jmp addr9 と goto addr9 は同じコードになりますので、

どちらの

書き方でOKです。

▲必要ワード数は展開された時のコードの大きさです。全ての複合命令が

実行クロック数=必要ワード数 とはなりませんので注意してください。

▲スキップする命令について

スキップする命令(csa,cse,decsz 等)は次の命令に必ず1ワードで終わる

インストラクショ

ンを置いてください。そうしないと、スキップした際に暴走します。(アセンブル時に警告されます)

♪初心者が割と引っ掛かりやすい落とし穴

◆PIC16F(C)84のRA4ピンはオープンドレインです。他のI/Oピンとは異なる

ります。電流を吸い込むことしかできません。他のピンと同じように使うには4.7kΩ

の抵抗でプルアップすればいいでしょう。(使い方にもよりますが...)

◆即値の代入について

例えば cn に即値10を代入する場合

```
mov    cn,#10        ; cn←10
```

と記述しますが、もし10の前の#がないと

```
mov    cn,10        ; cn←PCLATH !?
```

ファイルレジスタ10(この場合PCLATH)の内容がcnに代入されます。

この様に全然違う値がcnに入るため、プログラムがおかしくなることがあります。

アセンブラはこのミスをチェックできません。

◆sub 命令のキャリーフラグ

sub 命令のみボロー(桁下がり)が発生すると、CF(キャリーフラグ)は0になります。

発生しないときは1です。(一般的なCPUとは論理が逆です)

◆PIC16C71等のADコンバータ内蔵品種

パワーオンリセット時I/OピンがAD入力ピンになっています。そのままTRISを

入力にただけではポートを読めません。I/OピンをDigitalに変更してから

入力してください。

◆16C5Xシリーズのプログラム領域

16C57は000~7FF番地までありますが、PCのビット8が1のアドレス(100-1FF, 300-3FF, 500-5FF, 700-7FF)には call 命令、

jmp

PC+W 等のPCを変化させる命令ではコール/分岐できません。ビット8は常に0になりま

す。この間のアドレスにジャンプできるのは goto 命令だけです。PA0,PA1も同時に指定しないと、変なバンクに飛んでしまいます。

■このソフトを使用した際に発生した、損失・損害等については作者及び当社では一切責任を負いません。予めご了承ください。
■無断で上記のソフトを販売・配布・通信などのネットワーク上に流すことを禁止します。

●付録 ・拡張インストラクションの詳細

```
add    fr,#lite

    movlw    lite
    addwf   fr,1

add    fr1,fr2

    movf    fr2,0
    addwf   fr1,1

add    fr,W

    addwf   fr,1

add    W,fr

    addwf   fr,0

addb   fr,bit

    btfsc   bit
    incf   fr,1

and    fr,#lite

    movlw    lite
    andwf   fr,1

and    fr1,fr2

    movf    fr2,0
    andwf   fr1,1

and    fr,W

    and     fr,1

and    w,#lite

    andlw   lite
```

```
and    W,fr

    and     fr,0

cja    fr,#lite,addr9

    movlw   lite ^ Offh
    addwf   fr,0
    btfsc   3,0
    goto    addr9

cja    fr1,fr2,addr9

    movf    fr1,0
    subwf   fr2,0
    btfss   3,0
    goto    addr9

cjae   fr,#lite,addr9

    movlw   lite
    subwf   fr,0
    btfsc   3,0
    goto    addr9

cjae   fr1,fr2,addr9

    movf    fr2,0
    subwf   fr1,0
    btfsc   3,0
    goto    addr9

cjb    fr,#lite,addr9

    movlw   lite
    subwf   fr,0
    btfss   3,0
    goto    addr9

cjb    fr1,fr2,addr9

    movf    fr2,0
    subwf   fr1,0
    btfss   3,0
    goto    addr9

cjbe   fr,#lite,addr9

    movlw   lite ^ Offh
    addwf   fr,0
    btfss   3,0
    goto    addr9

cjbe   fr1,fr2,addr9

    movf    fr1,0
    subwf   fr2,0
    btfsc   3,0
```

goto	addr9		
cje	fr,#lite,addr9	movf	fr1,0
		subwf	fr2,0
		btfsc	3.0
movlw	lite	csae	fr,#lite
subwf	fr,0		
btfsc	3.2	movlw	lite
goto	addr9	subwf	fr,0
cje	fr1,fr2,addr9	btfss	3.0
		csae	fr1,fr2
movf	fr2,0		
subwf	fr1,0	movf	fr2,0
btfsc	3.2	subwf	fr1,0
goto	addr9	btfss	3.0
cjne	fr,#lite,addr9		
		csb	fr,#lite
movlw	lite		
subwf	fr,0	movlw	lite
btfss	3.2	subwf	fr,0
goto	addr9	btfsc	3.0
cjne	fr1,fr2,addr9		
		csb	fr1,fr2
movf	fr2,0		
subwf	fr1,0	movf	fr2,0
btfss	3.2	subwf	fr1,0
goto	addr9	btfsc	3.0
clc			
bcf	3.0	csbe	fr,#lite
clr	fr		
		movlw	lite ^ Offh
clrf	fr	addwf	fr,0
		btfsc	3.0
clr	w		
		csbe	fr1,fr2
clrw			
		movf	fr1,0
clr	wdt	subwf	fr2,0
		btfss	3.0
clrwdt			
		cse	fr,#lite
clrb	bit		
		movlw	lite
bcf	bit	subwf	fr,0
		btfss	3.2
clz			
bcf	3.2	cse	fr1,fr2
csa	fr,#lite		
		movf	fr2,0
movlw	lite ^ Offh	subwf	fr1,0
addwf	fr,0	btfss	3.2
btfss	3.0		
csa	fr1,fr2	csne	fr,#lite
		movlw	lite
		subwf	fr,0
		btfsc	3.2

```

csne    fr1,fr2

        movf    fr2,0
        subwf   fr1,0
        btfsc   3,2

dec     fr

        decf    fr,1

decsz   fr

        decfsz  fr,1

djnz    fr,addr9

        decfsz  fr,1
        goto    addr9

ijnz    fr,addr9

        incfsz  fr,1
        goto    addr9

inc     fr

        incf    fr,1

incsz   fr

        incsz   fr,1

jb      bit,addr9

        btfsc   bit
        goto    addr9

jc      addr9

        btfsc   3,0
        goto    addr9

jmp     addr9

        goto    addr9

jmp     pc+w

        addrwf  2,1

jmp     w

        movwf   2

jnb     bit,addr9

        btfss   bit
        goto    addr9

```

```

jnc     addr9

        btfss   3,0
        goto    addr9

jnz     addr9

        btfss   3,2
        goto    addr9

jz      addr9

        btfsc   3,2
        goto    addr9

lcall   addr11

        bcf/bsf 3,x
        bcf/bsf 3,x
        call    addr11

ljmp    addr11

        bcf/bsf 3,x
        bcf/bsf 3,x
        goto    addr11

lset    addr11

        bcf/bsf 3,x
        bcf/bsf 3,x

mov     fr,#lite

        movlw   lite
        movwf   fr

mov     fr1,fr2

        movf    fr2,0
        movwf   fr1

mov     fr,W

        movwf   fr

mov     option,#lte

        movlw   lite
        option

mov     option,fr

        movf    fr,0
        option

mov     option,W

        option

```

mov	!port_fr,#lite	btfsc	bit2
		bcf	bit1
movlw	lite	btfss	bit2
tris	port_fr	bsf	bit1
mov	!port_fr,fr	movsz	W,++fr
movf	fr,0	incfsz	fr,0
tris	port_fr	movsz	W,—fr
mov	!port_fr,W	decfsz	fr,0
tris	port_fr	neg	fr
mov	W,#lite	comf	fr,1
movlw	lite	incf	fr,1
mov	W,fr	not	fr
movf	fr,0	comf	fr,1
mov	W,/fr	not	W
comf	fr,0	xorlw	0ffh
mov	W,fr-W	or	fr,#lite
subwf	fr,0	movlw	lite
mov	W,++fr	iorwf	fr,1
incf	fr,0	or	fr1,fr2
mov	W,—fr	movf	fr2,0
decf	fr,0	iorwf	fr1,1
mov	W,<<fr	or	fr,W
rff	fr,0	iorwf	fr,1
mov	W,>>fr	or	W,#lite
rff	fr,0	iorlw	lite
mov	W,◊fr	or	W,fr
swapf	fr,0	iorwf	fr,0
movb	bit1,bit2	ret	
btfss	bit2	retlw	0
bcf	bit1	retw	lite1[,lite2,lite3...]
btfsc	bit2	retlw	lite1
bsf	bit1	retlw	lite2
movb	bit1,/bit2	retlw	lite3
		...	

rl	fr	decf	fr,1
	rff	swap	fr,1
rr	fr	swapf	fr,1
	rff	sz	
sb	bit	btfss	3,2
	btfss	test	fr
sc		movf	fr,1
	btfss	test	W
setb	bit	iorlw	0
	bsf	xor	fr,#lite
skip		movlw	lite
	btfss	xorwf	fr,1
snb	bit	xor	fr1,fr2
	btfsc	movf	fr2,0
snc		xorwf	fr1,1
	btfsc	xor	fr,W
snz		xorwf	fr,1
	btfsc	xor	W,#lite
stc		xorlw	lite
	bsf	xor	W,fr
stz		xorwf	fr,0
	bsf		3,2
sub	fr,#lite		
	movlw	movlw	lite
	subwf	subwf	fr,1
sub	fr1,fr2		
	movf	movf	fr2,0
	subwf	subwf	fr1,1
sub	fr,W		
	subwf	subwf	fr,1
subb	fr,bit		
	btfss	btfss	bit