Page d'aide assembleur

Préambule

La difficulté de la programmation en assembleur PIC se résume en deux points:

- *seulement 35 instructions*,
  mais pas de structures de contrôle, pas de fonctions...
- *et quelques centaines d'octets de mémoire*,
  mais pas de typage, pas de pointeurs, pas de fichiers...

La pauvreté du langage entraine des difficultés pour exprimer des choses aussi simples qu'un test IF-THEN-ELSE. Le but de cette page est de vous donner une aide sur la bonne façon d'écrire de l'assembleur PIC efficacement. Il n'y a aucune utilisation des périphériques internes ou externes du PIC.

Le code que vous trouverez ici est issu de nombreuses sources à la fois sur internet et dans les livres. Les auteurs (ou sources) sont cités quand ils sont connus.

liste des instructions pic 16

Instructions sur les registres octet

Mnémotique	arguments	Commentaire	drapeaux	cycles
addwf	*reg*, *dst*	add w to *reg*, result in *dst*	z, dc, c	1

andwf	*reg*, *dst*	and w and *reg*, result in *dst*	z	1
clrf	*reg*	write zero to *reg*	z	1
clrw		write zero to w i	Z	1
comf	*reg*, *dst*	complement *reg*, result in *dst*	Z	1
decf	*reg*, *dst*	decrement *reg*, result in *dst* Z	Z	1
incf	*reg*, *dst*	Increment *reg*, result in *dst* Z	Z	1
iorwf	*reg*, *dst*	Or W and *reg*, result in *dst*	Z	1
movf	*reg*, *dst*	Move *reg* to *dst*	Z	1
movwf	*reg*	Move W to *reg*	none	1
nop		No operation	none	1
rlf	*reg*, *dst*	Rotate *reg* left, result in *dst*	C	1
rrf	*reg*, *dst*	Rotate *reg* right, result in *dst*	C	1
subwf	*reg*, *dst*	Subtract W from *reg*, result in *dst*	Z, DC, C	1
swapf	*reg*, *dst*	Swap nibbles of *reg*, result in *dst*	none	1
xorwf	*reg*, *dst*	Xor W and *reg*, result in *dst*	Z	1
decfsz	*reg*, *dst*	Decrement *reg*, result in *dst*, skip IF zero	none	1 (2 if skip)
incfsz	*reg*, *dst*	Increment *reg*, result in *dst*, skip IF zero	none	1 (2 if skip)

Instructions sur les bits

Mnémotique	arguments	Commentaire	drapeaux	cycles
bcf	*reg*, *bit*	Clear *bit* of *reg*	none	1
bsf	*reg*, *bit*	Set *bit* of *reg*	none	1
btfsc	*reg*, *bit*	Skip next instruction IF *bit* of *reg* is clear	none	1 (2 if skip)
btfss	*reg*, *bit*	Skip next instruction IF *bit* of *reg* is set	none	1 (2 if skip)

Instructions sur les immediats

Mnémotique	arguments	Commentaire	drapeaux	cycles
addlw	*imm*	Add immediate to W	Z, DC, C	1
andlw	*imm*	And immediate to W	Z	1
iorlw	*imm*	Inclusive OR littéral to W	Z	1
movlw	*imm*	Move littéral to W	none	1
retlw	*imm*	Load W with immediate and return	none	2
sublw	*imm*	Subtract W from littéral	Z, DC, C	1
xorlw	*imm*	Xor W and littéral	Z	1

Instructions de controle

Mnémotique	arguments	Commentaire	drapeaux	cycles
clrwdt		Reset watchdog timer	TO, PD	1
call	*addr*	Call subroutine	none	2
goto	*addr*	Go to *addr*	none	2
retfie		Return from interrupt	GIE	2
return		Return from subroutine	none	2

sleep	Enter sleep mode	TO, PD	1

Structures de contrôle

Branchements conditionnels

Comme vous le savez en assembleur, pour faire une rupture de séquence on ne sait faire qu'une chose: tester une condition et si cette condition est vraie ou fausse alors sauter à une adresse, c'est peu. En outre, en assembleur PIC, il y a une difficulté supplémentaire: si la condition est vraie on se contente de sauter une instruction.

En PIC, il existe 4 instructions de sauts conditionnels: *decfsz*, *incfsz*, *btfss*, *btfsc*. Les deux première instructions sont surtout utilisés pour les boucles. Nous allons ici, nous intéresser au comparaisons sur les nombres 8 bits signés (en complément à 2) et non signés, il y a 6 possibles:

- égalité (== ou BEQ)
- inégalité (!= ou BNE)
- infériorité stricte (< ou BLT ou BLTS)
- supériorité stricte (> ou BGT ou BGTS)
- infériorité ou égalité (<= ou BLE ou BLES)
- supériorité ou égalité (>= ou BGE ou BGES)

Les registres doivent être dans le même banc. Le tableau qui suit donne en commentaire (après les ;) une notation de type macro instruction. Les comparaisons sont dépendent du type des données qui peuvent être non signées (valeur de 0 à 255) ou signées (valeurs de -128 à +127). Il est parfois nécessaire d'utiliser un registres temporaire TMP

Égalité registre/registre et registre/immédiat

```
; si (REG1 == REG2) goto LAB
; BEQ REG1, REG2, goto, LAB
  movf    REG1, W
  xorwf   REG2, W
  btfsc   STATUS, Z
  goto    LAB

; si (REG1 != REG2) goto LAB
; BNE REG1, REG2, goto, LAB
  movf    REG1, W
  xorwf   REG2, W
  btfss   STATUS, Z
  goto    LAB

; si (REG == IMM) goto LAB
; BEQ REG, IMM, goto, LAB
  movlw   IMM
  xorwf   REG, W
  btfsc   STATUS, Z
  goto    LAB

; si (REG1 != IMM) goto LAB
; BNE REG1, IMM, goto, LAB
  movlw   IMM
  xorwf   REG, W
  btfss   STATUS, Z
  goto    LAB
```

Comparaisons registre/registre sur des nombres non signés

```
; si (REG1 < REG2) goto LAB
; BLT REG1, REG2, goto, LAB
```

```
  movf    REG2, W
  subwf   REG1, W
  btfss   STATUS, C
  goto    LAB

; si (REG1 > REG2) goto LAB
; BGT REG1, REG2, goto, LAB
  movf    REG1, W
  subwf   REG2, W
  btfss   STATUS, C
  goto    LAB

; si (REG1 <= REG2) goto LAB
; BLE REG1, REG2, goto, LAB
  movf    REG1, W
  subwf   REG2, W
  btfsc   STATUS, C
  goto    LAB

; si (REG1 >= REG2) goto LAB
; BGE REG1, REG2, goto, LAB
  movf    REG2, W
  subwf   REG1, W
  btfsc   STATUS, C
  goto    LAB
```

Comparaisons registre/immediat sur des nombres non signés

```
; si (REG > IMM) goto LAB
; BGT REG, IMM, goto, LAB
  movf    REG, W
  sublw   IMM
  btfss   STATUS, C
  goto    LAB

; si (REG < IMM) goto LAB
; BLT REG, IMM, goto, LAB
  movlw   IMM
  subwf   REG, W
  btfss   STATUS, C
  goto    LAB

; si (REG <= IMM) goto LAB
; BLE REG, IMM, goto, LAB
  movf    REG, W
  sublw   IMM
  btfsc   STATUS, C
  goto    LAB

; si (REG >= IMM) goto LAB
; BGE REG, IMM, goto, LAB
  movlw   IMM
  subwf   REG, W
  btfsc   STATUS, C
  goto    LAB
```

Comparaisons immediat/registre sur des nombres non signés

```
; si (IMM < REG) goto LAB
; BLT IMM, REG, goto, LAB
  movf    REG,W
  sublw   IMM
  btfss   STATUS, C
  goto    LAB
```

```
; si (IMM > REG) goto LAB
; BGT IMM, REG, goto, LAB
  movlw   IMM
  subwf   REG,W
  btfss   STATUS, C
  goto    LAB

; si (IMM <= REG) goto LAB
; BLE IMM, REG, goto, LAB
  movlw   IMM
  subwf   REG,W
  btfsc   STATUS, C
  goto    LAB


; si (IMM >= REG) goto LAB
; BGE IMM, REG, goto, LAB
  movf    REG,W
  sublw   IMM
  btfsc   STATUS, C
  goto    LAB
```

comparaisons registre/registre sur des nombres signés

```
; si (REG1 < REG2) goto LAB
; BLTS REG1, REG2, goto, LAB
  movf    REG1, W
  addlw   0x80
  movwf   TMP
  movf    REG2, W
  addlw   0x80
  subwf   TMP, W
  btfss   STATUS, C
  goto    LAB

; si (REG1 > REG2) goto LAB
; BGTS REG1, REG2, goto, LAB
  movf    REG2, W
  addlw   0x80
  movwf   TMP
  movf    REG1, W
  addlw   0x80
  subwf   TMP, W
  btfss   STATUS, C
  goto    LAB

; si (REG1 <= REG2) goto LAB
; BLES REG1, REG2, goto, LAB
  movf    REG2, W
  addlw   0x80
  movwf   TMP
  movf    REG1, W
  addlw   0x80
  subwf   TMP, W
  btfsc   STATUS, C
  goto    LAB

; si (REG1 >= REG2) goto LAB
; BGES REG1, REG2, goto, LAB
  movf    REG1, W
  addlw   0x80
  movwf   TMP
  movf    REG2, W
  addlw   0x80
```

```
  subwf   TMP, W
  btfsc   STATUS, C
  goto    LAB
```

Comparaisons registre/immediat sur des nombres signés

```
; si (REG < IMM) goto LAB
; BLTS REG, IMM, goto, LAB
  movf    REG, W
  addlw   0x80
  movwf   TMP
  movlw   0x80 | IMM
  subwf   TMP, W
  btfss   STATUS, C
  goto    LAB

; si (REG > IMM) goto LAB
; BGTS REG, IMM, goto, LAB
  movf    REG, W
  addlw   0x80
  sublw   0x80 | IMM
  btfss   STATUS, C
  goto    LAB

; si (REG <= IMM) goto LAB
; BLES REG, IMM, goto, LAB
  movf    REG, W
  addlw   0x80
  sublw   0x80 | IMM
  btfsc   STATUS, C
  goto    LAB

; si (REG >= IMM) goto LAB
; BGES REG, IMM, goto, LAB
  movf    REG, W
  addlw   0x80
  sublw   0x80 | IMM
  btfss   STATUS,Z
  btfss   STATUS, C
  goto    LAB
```

Comparaisons immediat/registre sur des nombres signés

```
; si (IMM < REG) goto LAB
; BLTS IMM, REG, goto, LAB
  movf    REG,W
  addlw   0x80
  sublw   0x80 | IMM
  btfss   STATUS, C
  goto    LAB


; si (IMM > REG) goto LAB
; BGTS IMM, REG, goto, LAB
  movf    REG,W
  addlw   0x80
  sublw   0x80 | IMM
  btfsc   STATUS,Z
  goto    $+3
  btfsc   STATUS,C
  goto    LAB

; si (IMM <= REG) goto LAB
; BLES IMM, REG, goto, LAB
  movf    REG,W
  addlw   0x80
  movwf   TMP
  movlw   0x80 | IMM
```

```
        subwf   TMP,W
        btfsc   STATUS, C
        goto    LAB


    ; si (IMM >= REG) goto LAB
    ; BGES IMM, REG, goto, LAB
        movf    REG,W
        addlw   0x80
        sublw   0x80 | IMM
        btfsc   STATUS, C
        goto    LAB
```

If condition then else

Pour pouvoir faire une alternative, il faut savoir faire un branchement conditionnel. Nous supposons que nous savons le faire. Dans une alternative *si-alors-sinon*, nous allons mettre la séquence *alors* avant la séquence *sinon*. Ce choix permet, a priori, une plus grande lisibilité du code, car il est plus proche du code écrit dans les langages évolués de type C ou Pascal. Pour respecter cet ordre, il est souvent nécessaire d'inverser le test. En effet, si la condition est fausse, il faut sauter à la séquence *sinon* et donc simplement continuer si la condition est vraie.

Quand les conditions sont plus complexes, qu'elles sont le résultat d'une expression Booléenne, il faut faire des séquences de test. Le tableau suivant contient quelques exemples permettant de comprendre la méthode à suivre. Ce n'est pas une méthode générale

SI test ALORS sequence_alors SINON sequence_sinon FIN

Cas général:

```
        IFNOT test GOTO sinon
alors   sequence_alors
        goto  fin
sinon   sequence_sinon
fin
```

Exemple: si (reg1==reg2) alors sequence_alors SINON sequence_sinon FIN

```
si      ; BNE reg1, reg2, goto, sinon
        movf  reg1,w
        xorwf reg2,w
        btfss STATUS, Z
        goto  sinon

alors   sequence_alors
        goto  fin

sinon   sequence_sinon
fin
```

SI test1 ET test2 ALORS sequence_alors SINON sequence_sinon FIN

```
si      IFNOT test1 GOTO sinon
        IFNOT test2 GOTO sinon
alors   sequence_alors
        goto  fin
sinon   sequence_sinon
fin
```

SI test1 OU test2 ALORS sequence_alors SINON sequence_sinon FIN

```
si      IF test1 GOTO alors
        IFNOT test2 GOTO sinon
```

```
alors  sequence_alors
       goto  fin
sinon  sequence_sinon
fin
```

SI (test1 OU test2) ET (test3 OU test4) ALORS sequence_alors SINON sequence_sinon FIN

```
si     IF test1 GOTO si2
       IFNOT test2 GOTO sinon
si2    IF test3 GOTO alors
       IFNOT test4 GOTO sinon
alors  sequence_alors
       goto  fin
sinon  sequence_sinon
fin
```

SI (test1 ET test2) OU (test3 ET test4) ALORS sequence_alors SINON sequence_sinon

```
si     IFNOT test1 GOTO si2
       IF test2 GOTO alors
si2    IFNOT test3 GOTO sinon
       IFNOT test4 GOTO sinon
alors  sequence_alors
       goto  fin
sinon  sequence_sinon
fin
```

Faire n fois

On traite deux cas, si N est sur 8 bits ou si N est sur 16 bits. Dans le cas 8 bits, la valeur 0 est en fait 256. Dans le as 16 bits, la valeur 0 c'est vraiment 0 (c'est-à-dire qu'on n'exécute pas la séquence_corps).

Faire N fois (8 bits) sequence_corps FIN

```
       ; Si N est une valeur immédiate.
       movlw   N
       movwf   REG
corps  sequence_corps
       decfsz  REG
       goto    corps

       ; Si N est dans un registre.
       movf    REG_N, W
       movwf   REG
corps  sequence_corps
       decfsz  REG
       goto    corps
```

Faire N fois (16 bits) sequence_corps FIN

```
       ; Si N est une valeur immédiate
       movlw   (HIGH N) + 1
       movwf   REG+1
       movlw   LOW N
       movwf   REG
       movf    REG, F
       btfsc   STATUS, Z
       goto    testh
corps  sequence_corps
       decfsz  REG
       goto    corps
testh  decfsz  REG+1
       goto    corps
fin

       ; Si N est dans une paire de registres successifs::
       incf    REG_N+1, W
       movwf   REG+1
```

```
        movf    REG_N, W
        movwf   REG
        btfsc   STATUS, Z
        goto    testh
corps   sequence_corps
        decfsz  REG
        goto    corps
testh   decfsz  REG+1
        goto    corps
fin
```

Branchement relatif au PC

On veut faire PC = PC+1+registre c'est-à-dire goto PC+1+reg

```
BRPC macro _reg_
        movlw   high($+7)
        movwf   PCLATH
        movlw   $+5
        addwf   _reg_,w
        btfsc   STATUS,C
        incf    PCLATH,f
        movwf   PCL
     endm
```

Switch case

Le switch case est un test à choix multiples. Il existe dans tous les langages évolués et permet entre autre de décrire les automates d'états finis particulièrement important dans la programmation en assembleur.

La forme générale du switch case est

```
switch reg
case CST1 : sequence_cst1; break;
case CST2 : sequence_cst2; break;
case CST3 : sequence_cst3; break;
default   : sequence_default
end
```

Les valeurs CST peuvent être quelconque. Dans ce cas, il n'y a d'autres choix que de faire les tests les uns après les autres en séquence. Dans le code qui suit, on suppose que les constantes sont bien des valeurs immédiates mais en fait il peut s'agir de registres. On peut même faire des tests différents. C'est très général mais évidemment ce n'est pas très effice et tous les tests ne sont pas équivalent vis à vis du temps de traitement. Le premier est favorisé par rapport au dernier.

```
; BEQ reg, CST1, goto, label_cst1
movlw  CST1
xorwf  reg,  W
btfsc  STATUS, Z
goto   label_cst1

; BEQ reg, CST2, goto, label_cst2
movlw  CST2
xorwf  reg,  W
btfsc  STATUS, Z
goto   label_cst2

; BEQ reg, CST3, goto, label_cst3
movlw  CST3
xorwf  reg,  W
btfsc  STATUS, Z
```

```
    goto   label_cst3

    ; BEQ reg, CST4, goto, label_cst4
    movf   CST4
    xorwf  reg,  W
    btfsc  STATUS, Z
    goto   label_cst4

    ; puis a la suite des tests
    ; les sequences a executer
    sequence_default
label_cst1
    sequence_cst1
    goto  fin
label_cst2
    sequence_cst2
    goto  fin
label_cst3
    sequence_cst3
    goto  fin
label_cst4
    sequence_cst4
    goto  fin
fin
```

Pour être efficace, il est souhaitable de se contraindre dans le choix des valeurs. Le plus simple est alors de choisir les constantes consécutives à partir de 0, on peut alors faire une table de sauts.

```
cst_A   EQU  0
cst_B   EQU  1
cst_C   EQU  2
    ; SWITCH reg
    movlw   high($+7)
    movwf   PCLATH
    movlw   $+5
    addwf   _reg_,w
    btfsc   STATUS,C
    incf    PCLATH,f
    movwf   PCL
    goto    label_cst_A
    goto    label_cst_B
    goto    label_cst_C
label_cst_A
    sequence_cst1
    goto  fin
label_cst_B
    sequence_cst2
    goto  fin
label_cst_C
    sequence_cst3
    goto  fin
fin
```

Tantque condition faire

Arithmétiques

Entière 8 bits

- Addition

- Soustraction
- Multiplication
- Division

Entière 16 bits

Virgule fixe 16 bits

Flottante 16 bits

Expressions

Boolèennes

Arithmétiques

Structures de données

Tableaux de données

- Dans les registres
- Dans le programme

Liste chainée

Pile

Macroinstructions

Automates

Routines diverses (snippets)

Attentes actives

- Attente 10ms

```
wait10ms   movlw  D'66' ; il n'est pas utile d'initialiser le registre 0x70
           movwf  0x71  ; attente d'environ 3*256*66*.2 = 10ms
           decfsz 0x70
           goto   $-1
           decfsz 0x71
           goto   $-3
           return
```

Mathématiques

- Parité
- Moyenne glissante
- PGCD