¿Cómo realizar comparaciones en un PIC si no existe la instrucción “Compara”? De eso se trata este documento.

Adaptado de varios artículos y extractos de información de diversas fuentes por Sergio Fco. Hernández Machuca [sfhm\_clases@hotmail.com]

La instrucción fundamental para la mayoría de las comparaciones es la resta, después de restar A – B y conocer la naturaleza del resultado (¿fue igual a cero?, ¿cupo el resultado en la dimensión de los operandos?, es decir, ¿se generaron valores verdaderos para las banderas de “Cero” y de “Acarreo”? ó qué pasó) podremos decir cómo son entre si A y B, por ejemplo: A = B; A > B; A < B; A distinta a B; A >= B; A <= B.

La instrucción fundamental que emplea el PIC para la resta es: **subwf** **Y**,**W**

Después de la ejecución de esta instrucción el estado de las banderas del registro STATUS (“**C”**, la cual se ajusta a “1” si ocurrió un Acarreo o *Carry*; “**Z”**, la cual se ajusta a “1” cuando el resultado de la operación fue igual a Cero) refleja si **Y** fue Mayor, Menor o Idéntico al valor de **W.**

Dentro del código que se esté construyendo es responsabilidad del programador colocar en el registro “**W**” el valor adecuado, así como reemplazar a “**Y**” con el nombre del registro que contenga el otro valor.

La instrucción **subwf** **Y**,**W,** por sí misma, compara dos cantidades de 8 bits sin signo. Para extender la comparación a cantidades de 16 bits habrá que emplear algún código más complejo, después se resolverá ese caso, lo mismo sucederá para cuando se requiere comparar cantidades consigo.

Para nuestro caso, en donde requerimos sólo conocer la naturaleza de la relación entre **Y** y **W**, al emplear la instrucción anterior tendremos como un efecto lateral el resultado de la resta. Para fines de la siguiente explicación se debe considerar lo siguiente:

- Se denominará “w” al valor que se carga en el registro W ***antes*** de efectuar la operación **subwf**.

- Se denominará “wnew” al valor que se en el registro W ***después*** de efectuar la operación **subwf**.

Si se está familiarizado con el hecho de que

-------------------------------------------------------------------------------------------------

The subwf instruction, all by itself, compares 2 unsigned 8 bit values. Later we'll see how to compare 16 bit values and signed values.

As a side effect, it also performs subtraction. I'll call the value that you load into w before the subwf ``w''; I'll call the value the PIC puts into w after the subwf ``wnew''.

If you're already familiar with how "addwf" sets the carry register, you might reflect on the fact that

subwf Y,w

gives exactly the same results (in wnew, Carry, and Zero) as the final add in this sequence:

; w(intermediate) := -w (two's complement)

xorlw 0xff

addlw 1

; wnew = Y + w(intermediate) = Y-w

addwf Y

There's only 3 possible results:

if w < Y: Status,Z = 0. Status,C = 1. wnew = Y-w.

if w == Y: Status,Z = 1. Status,C = 1. wnew = 0.

if Y < w: Status,Z = 0. Status,C = 0. wnew = Y-w + 0x100.

You can combine those possibilites in pairs to get these other commonly used operators:

if Y <= w: Status,Z = Status,C. (i.e., either both are 1, or both are 0).

if w != Y: Status,Z = 0. Status,C = (could be either).

if w <= Y: Status,Z = (could be either). Status,C = 1.

Hotshot PIC assembly-language programmers try to force their comparisons to be (w <= Y) or (Y < w), because then they only need to check Status,C -- they can ignore Status,Z.

-- with help from Robin Abbott - robin.abbott@dial.pipex.com

If you're used to programming in assembly language on a non-PIC chip, check this out:

Comparisons to 0 -- if( Y <= 0 ) or if( 0 < Y ), where Y is a signed (8 or 16 bit) number -- are the fastest comparisons on the PIC. They are even faster than comparing ``if( w == Y )'' or even ``if( w == 0 )''.

**8 bit compares**

So how do you actually use this nifty instruction ?

I'm going to use ``RAMx'' and ``RAMy'' to indicate values in a RAM register, ``K'' to indicate some fixed constant value (often defined using the ``EQU'' assembler directive). (If you want to compare 2 constant values, use the assembler directive ``#if ... #else ... #endif''. The grid below \*should\* cover all other combinations. ).

Q: Hey ! you left out all the ``greater than'' operators !

A: Whoops. Well, if you need ``A > B'', you can always use ``B < A''.

Most of these are from Tony Nixon.

I'm going to go on the assumption that you want to execute a chunk of code only if the conditional is \*true\*. (This is what ``if()'' means in C and other decent high-level languages. Don't let those BASIC programmers confuse you.)

Put that chunk of code (or a CALL to it) immediately after these blocks of code, and follow that chunk with some unique label. Then replace the word "Endif" with the name of that label.

If that chunk of code is \*exactly\* one instruction long (say ``CALL''), and you're very clever, you can optimize this code even more.

( [swstmt.htm](http://www.piclist.com/techref/microchip/swstmt.htm) describes how to code ``if - then - else - endif'' blocks).

Q: Is there a \*reason\* Tony Nixon uses ``addlw'' and ``addwf'' rather than ``sublw'' and ``subwf'' ? All PICs support subwf, right ? And the ones that don't support sublw don't support ``addlw'' either, right ?

If you want to use a ``12 bit core PIC'' such as the 12C509, note that it does not have the SUBLW, ADDLW instructions. See the [PIC Microcontroller Instruction Set Quick Reference and Core Comparison Matrix](http://www.piclist.com/techref/member/DW--RA4/PICISM.HTM) . If you're using one of those chips, then you can use MOVLW then SUBWF (or ADDWF) ...

;**if RAMx <= K**

;Tony Nixon

movf RAMx,w

addlw 255 - K ; eg if RAMx > 5 ... addlw d'250'

skpnc

goto Endif

;**if K <= RAMx**

;Tony Nixon

movf RAMx,w

addlw 255 - K + 1 ; eg if RAMx < 5 ... addlw d'251'

skpc

goto Endif

;**if RAMx < K**

; Tony Nixon

movf RAMx,w

addlw 255 - K + 1 ; eg if RAMx >= 5 ... addlw d'251'

skpnc

goto Endif

;**if K < RAMx**

; Tony Nixon

movf RAMx,w

addlw 255 - K ; eg if RAMx <= 5 ... addlw d'250'

skpc

goto Endif

;**if RAMx <= RAMy**

; Scott Dattalo

movf RAMx,w

subwf RAMy,w

skpc

goto Endif

;**if RAMy < RAMx**

; Scott Dattalo

movf RAMx,w

subwf RAMy,w

skpnc

goto Endif

;**if RAMx < RAMy** /\* obsolete \*/

; unknown -- perhaps James Newton ?

movf RAMx, w

subwf RAMy, w

skpz ;for the case that RAMx=RAMy where C will be 1

skpc

goto Endif

;**if RAMx <= RAMy**

; better, but only works on 18cxxx chips !

; Scott Dattalo

movf RAMx,w ;wreg = RAMx

subwf RAMy,w ;wreg = RAMy - RAMx

bn Endif ;Branch if negative only available on 18cxxx

; (You could use the bnc [branch if no carry] to achieve the same effect.

; The negative bit has a clearer meaning in this context).

;The N bit will be cleared if RAMx == RAMy or

;RAMy > RAMx, and will be set if RAMy < RAMx

;18cxxx 3\*16 = 48 bits of program memory

;16cxxx 5\*14 = 70 bits of program memory

;**if RAMy < RAMx**

; better, but only works on 18cxxx chips !

; Scott Dattalo

movf RAMx,w ;wreg = RAMx

subwf RAMy,w ;wreg = RAMy - RAMx

bnn Endif ;branch if not negative

;The N bit will be cleared if RAMx == RAMy or

;RAMy > RAMx, and will be set if RAMx < RAMy

; Again, you could use the bc (branch on carry) instruction too.

Regulus Berdin says: For readability, one could write a jump if equal macro:

JIFEQ MACRO register,literal,address

movlw literal

xorwf register,w

skpnz

goto address

ENDM

switch:

JIFEQ data,'A',Process\_A

JIFEQ data,'B',Process\_B

JIFEQ data,'C',Process\_C

.

.

.

process\_A:

...

return

process\_B:

...

return

process\_C:

...

return

.

.

.