¿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 la operación “addwf” ajusta la bandera de *Carry* (STATUS, C) así como la de *Cero* (STATUS, Z), entonces puede considerar en el hecho de que “subwf Y,W” da exactamente el mismo resultado (en “wnew”, la bandera de *Carry* y la bandera de *Cero*) que la ejecución de la siguiente secuencia de operaciones:

; w(intermedia) := -w (complemento a dos)

xorlw 0xFF ; Cambios de “0”s a “1”s y viceversa.

addlw 1 ; Se le agrega 1 al resultado. Se obtiene el *Complemento a Dos*

; wnew = Y + w(intermedia) = Y-W

addwf Y ; La suma de A + (Complemento a 2 de B) => A - B

Para lo anterior sólo existen tres posibles resultados:

Si W < Y 🡪 STATUS, Z = 0; STATUS, C = 1; wnew = Y – W

Si W = Y 🡪 STATUS, Z = 1; STATUS, C = 1; wnew = 0

Si Y < W 🡪 STATUS, Z = 0; STATUS, C = 0; wnew = Y – W + 0x100

Lo anterior nos resuelve las principales condiciones que se desean probar en programas en lenguaje ensamblador.

Se pueden combinar los resultados anteriores para que, en pares, se puedan determinar otras condiciones comúnmente usadas.

Si Y <= W 🡪 STATUS, Z = STATUS, C (siendo ambos “0” o ambos “1”)

Si W <> Y 🡪 STATUS, Z = 0; STATUS, C (cualquier valor)

Si W <= Y 🡪 STATUS, Z (cualquier valor); STATUS, C = 1

Sugerencias para implementar código en PICs:

- Tratar de forzar las comparaciones a que sean (W <= Y) ó (Y <W), dado que éstas sólo requieren probar la bandera STATUS, C, sin importar el valor que tuviera STATUS, Z.

Comparaciones de 8 bits.

Para lo que continúa, se emplea “RAMx” y “RAMy” para identificar valores que se encuentran en alguno de los registros, de propósito específico (periféricos o recursos, por ejemplo) o de propósito general (para uso del programador, por ejemplo, variables, contadores, apuntadores, datos), “K” indica un valor constante o literal (habitualmente definido por una directiva del lenguaje ensamblador “EQU”).

Si se requiere comparar dos valores constantes se pueden usar las directivas del lenguaje ensamblador “#if…#else…#endif”. Lo que se muestra a continuación debe cubrir todas las posibilidades restantes. Si, por ejemplo, se desea encontrar si “A > B”, entonces se puede emplear la evaluación de “B < A”.

Se asume que en cada caso se desea ejecutar una sección de código en particular en consecuencia de que la condición probada sea cierta, es decir, se evalúe como \*true\*. Esto es cierto para estructuras como las del lenguaje C, en donde la expresión

If (Condición) {Código\_a\_Ejecutar}.

Para emplear adecuadamente las estructuras que se proponen, se debe colocar la sección de código que se desea ejecutar en consecuencia de que la condición probada sea verdadera, inmediatamente después de probar la condición. Se puede hacer uso de una instrucción **call *sección***. A continuación del segmento de código (o instrucción **call**) se debe incluir una etiqueta que indique el inicio de la parte “EndIf” de la estructura. El nombre que lleve esta etiqueta es el que sustituirá al término “EndIf” en la estructura propuesta originalmente.

Si el código que se debe ejecutar es de tan sólo una instrucción (digamos, la instrucción **call** algún\_lugar) es posible optimizar el código.

Es interesante revisar la siguiente propuesta:

; ¿Es el resultado verdadero?

**btfss** byte\_prueba, bit\_prueba ; Prueba la ***condición***.

**goto** Parte\_else

Parte\_Then:

; El resultado fue VERDADERO (bit\_prueba=1), se ejecuta lo que sigue:

... ;

**goto** Parte\_endif

Parte\_else:

; El resultado fue FALSO, (bit\_prueba=0), so ejecuta desde aquí…

... ;

Parte\_endif:

; Continuación de procesar la ***condición***, con cualquiera de sus dos resultados

...;

Estructuras para pruebas de condiciones

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Salta Si** | **Condición**  **a probar** | **Código para Variables** | **Código para Literales** | **Condición**  **Bandera probada** |
| A == B | A – B = 0 | **movf A, W**  **subwf B, W**  **btfsc STATUS, Z**  **goto Label ; Salta si Z = 1** | **movlw A**  **sublw B**  **btfsc STATUS, Z**  **goto Label ; Salta si Z = 1** | Calcula A – B  Si Z = 1, Resultado = 0 |
| A ¡= B | A – B ¡= 0 | **movf A, W**  **subwf B, W**  **btfss STATUS, Z**  **goto Label ; Salta si Z = 0** | **movlw A**  **sublw B**  **btfss STATUS, Z**  **goto Label ; Salta si Z = 0** | Calcula A – B  Si Z = 1, Resultado ¡= 0 |
| A > B | B – A < 0 | **movf A, W**  **subwf B, W**  **btfss STATUS, C**  **goto Label ; Salta si C = 0** | **movlw A**  **sublw B**  **btfss STATUS, C**  **goto Label ; Salta si C = 0** | Calcula B – A  Si Cy = 0, Resultado < 0  Si Cy = 1, Resultado >= 0 |
| A >= B | A – B >= 0 | **movf B, W**  **subwf A, W**  **btfsc STATUS, C**  **goto Label ; Salta si C = 1** | **movlw B**  **sublw A**  **btfsc STATUS, C**  **goto Label ; Salta si C = 1** | Calcula A – B  Si Cy = 1, Resultado >= 0  Si Cy = 0, Resultado < 0 |
| A < B | A – B < 0 | **movf B, W**  **subwf A, W**  **btfss STATUS, C**  **goto Label ; Salta si C = 0** | **movlw B**  **sublw A**  **btfss STATUS, C**  **goto Label ; Salta si C = 0** | Calcula A – B  Si Cy = 0, Resultado < 0  Si Cy = 1, Resultado >= 0 |
| A <= B | B – A >= 0 | **movf A, W**  **subwf B, W**  **btfsc STATUS, C**  **goto Label ; Salta si C = 1** | **movlw A**  **movlw B**  **btfsc STATUS, C**  **goto Label ; Salta si C = 1** | Calcula B – A  Si Cy = 1, Resultado >= 0  Si Cy = 0, Resultado < 0 |

**Label** es la etiqueta a donde se salta en caso de que la prueba sea EXITOSA.