Recomendaciones para programar en ensamblador del MSP430

Las instrucciones de salto condicional del MSP430 se resumen en la tabla 1. Cuando una misma instrucción tiene más de un nemónico, el nemónico alternativo se muestra entre parentesis.

Nemónico	*Descripción*
JC (JHS)	Salta si hay acarreo (Salta si es mayor o igual para números sin signo, después de una comparación)
JZ (JEQ)	Salta si es cero (Salta si son iguales, tanto para números con signo como para números sin signo, después de una comparación)
JGE	Salta si es mayor o igual para números con signo, después de una comparación.
JL	Salta si es menor para números con signo, después de una comparación
JN	Salta si es negativo
JNC (JLO)	Salta si no hay acarreo (Salta si es menor, para números sin signo, después de una comparación)
JNZ (JNE)	Salta si no es cero (Salta si no son iguales, tanto para números con signo como para números sin signo, después de una comparación)

Estas instrucciones pueden clasificarse en dos grupos, las que solo revisan una bandera y las que se usan despues de una instrucción de comparación.

Los saltos condicionales que solo revisan una bandera son JZ, JNZ, JC, JNC y JN. Si la condición que revisan se cumple, entonces la ejecución se transfiere a la instrucción marcada con la etiqueta que se tiene como openando, si no, se continua con la siguiente instrucción. Ejemplo: `Jz etq1` se debe leer como: si la bandera Z=1, entonces brinca a la instrucción marcada con la etiqueta `etq1`, si no, ejecuta la siguiente instrucción.

Los salto condicionales que se usan después de una comparación deben ser interpretados como si el operador condicional se encontrara en medio de los dos operandos de la comparación, pero con el orden inverso. Por ejemplo, la secuencia de instrucciones

```
cmp R4,R5
jge etiqueta
```

debe interpretarse como:

si R5 es mayor igual que R4 (interpretados como números con signo), entonces brinca a Etiqueta sino, ejecuta la instrucción siguiente.

Observe que no existen instrucciones para saltar si es mayor que o si es menor igual. Esto es debido a que para comprobar estas condiciones seria necesario revisar el estado de dos banderas a la vez. De cualquier forma, las condiciones existentes pueden ser utilizadas para emular dichas condiciones de forma similar a los siguientes ejemplos:

```
cmp R4,R5
jl    etiqueta
jeq   etiqueta
```

que debe interpretarse como:

si R5 es menor igual que R4 (interpretados como números con signo), entonces brinca a Etiqueta sino, ejecuta la instrucción siguiente.

```
        cmp R4,R5
        jlo   sigue
        jeq   sigue
       jmp  etiqueta
sigue:
```

que debe interpretarse como:

si R5 es mayor que R4 (interpretados como números con signo), entonces brinca a Etiqueta sino, ejecuta la instrucción siguiente.

Observe que en este caso se utilizo la condición contraria y solo se utilizo para brincar a la etiqueta sigue y que el salto a la etiqueta deseada lo realizo un salto incondicional.

Una observación importante es que en el MSP430 la instrucción JMP solo puede saltar 512 palabras hacia adelante o hacia atrás, lo cual es más que suficiente para la mayoría de las aplicaciones. Sin embargo, en algunos caso en que se tiene que saltar a una etiqueta fuera de ese rango, entonces hay que usar la instrucción BRA, La cual mueve una constante de 16 bits al PC, y por lo tanto, puede saltar a cualquier dirección de memoria.

Finalmente, al tratar de convertir un algoritmo estructurado en un programa en ensamblador, hay que recordar que en las estructuras if y while se debe utilizar la condición contraria a la del algoritmo. Esto, aunado a que solo existen instrucciones para algunos operadores relacionales, puede complicar la traducción, por lo que se recomienda guiarse con la siguiente tabla.

Estructura en C	== / != / >= / <

`if (a op b) {`
`  **Bloque1;**`
`}`
`...`
	`  cmp b,a`
	`  jne/jeq/jlo/jhs FINSI`
	`  **Bloque1**`
	`FINSI: ...`

`if (a op b) {`
`  **Bloque1;**`
`}else{`
`  **Bloque2;**`
`}`
`...`
	`  cmp b,a`
	`  jne/jeq/jlo/jhs ELSE`
	`  **Bloque1**`
	`  jmp FINSI`
	`ELSE:  **Bloque2**`
	`FINSI: ...`

`while (a op b) {`
`  Bloque1;`
`}`
`...`
	`WHILE: cmp  b,a`
	`  jne/jeq/jlo/jhs FINW`
	`  **Bloque1**`
	`  jmp WHILE`
	`FINW: ...`

`do  {`
`  Bloque1`
`}while (a op M);`
`...`
	`DO:  **Bloque1**`
	`  cmp b,a`
	`  jeq/jne/jhs/jlo DO`
	`  ...`

En la tabla anterior se presenta la implementación en ensamblador de las estructuras más comunes con las condiciones para las que existen instrucciones. Note que en todas se utiliza la condición

contraria, con excepción de la estructura do-while. La implementación de las condiciones para las que no existe una estructura directa se muestra en la siguiente tabla.

Estructura en C	>	<=

`if (a op b) {`
`  **Bloque1;**`
`}`
`...`
	`  cmp b,a`
	`  jlo FINSI`
	`  jeq FINSI`
	`  **Bloque1**`
	`FINSI: ...`
		`  cmp b,a`
		`  jlo ETQ1`
		`  jne FINSI`
		`ETQ1:  **Bloque1**`
		`FINSI: ...`

`if (a op b) {`
`  **Bloque1;**`
`}else{`
`  **Bloque2;**`
`}`
`...`
	`  cmp b,a`
	`  jlo ELSE`
	`  jeq ELSE`
	`  **Bloque1**`
	`  jmp FINSI`
	`ELSE:  **Bloque2**`
	`FINSI: ...`
		`  cmp b,a`
		`  jlo ETQ1`
		`  jne ELSE`
		`ETQ1:  **Bloque1**`
		`  jmp FINSI`
		`ELSE:  **Bloque2**`
		`FINSI: ...`

`while (a op b) {`
`  Bloque1;`
`}`
`...`
	`WHILE: cmp  b,a`
	`  jlo FINW`
	`  jeq FINW`
	`  **Bloque1**`
	`  jmp WHILE`
	`FINW: ...`
		`WHILE: cmp b,a`
		`  jlo ETQ1`
		`  jne FINW`
		`ETQ1:  **Bloque1**`
		`  jmp WHILE`
		`FINW: ...`

`do  {`
`  Bloque1`
`}while (a op M);`
`...`
	`DO:  **Bloque1**`
	`  cmp b,a`
	`  jlo FINDO`
	`  jne DO`
	`FINDO: ...`
		`DO:  **Bloque1**`
		`  cmp b,a`
		`  jlo DO`
		`  jeq DO`
		`  ...`

Finalmente, los programadores novatos suelen confundir los diferentes modos de direccionamiento y usar la dirección de una localidad de memoria en vez de su contenido o viceversa, por lo que es muy importante que se familiarice con dichos modos de direccionamiento y se asegure de usar el que hace lo que desea hacer. En particular revise los modos indexado, absoluto e inmediato, que son los que suelen causar más confusión.