TEMA 3. FAMILIA TMS320C3x. DESCRIPCIÓN DEL SOFTWARE.

1. Introducción.
2. Formato de datos.
3. Modos de direccionamiento.
4. Pila.
5. Sumario de instrucciones.

---

1. INTRODUCCIÓN

Hasta este punto, se ha abordado el estudio del hardware de los DSPs de la familia TMS320C3x de Texas Instruments, haciendo hincapié en las características del TMS320C30 y destacando algunas peculiaridades del TMS320C31. Se hace, a continuación, una descripción del software asociado a esta familia, entendiendo por software el conjunto de operaciones básicas que son capaces de realizar estos microprocesadores (instrucciones en ensamblador en concreto), formato de los datos tanto enteros como flotantes (recuérdese que este tipo de microprocesadores es especial por permitir operaciones con datos en formato flotante, al contrario que otros microcontroladores ya estudiados, véase Motorola MC68HC11) y finalmente los modos de direccionamiento y de acceso a los datos.

2. FORMATO DE DATOS

La familia de DSPs TMS320C3x trabaja con dos tipos de datos, enteros y flotantes de coma variable.

1. Datos **enteros**. Pueden ser, a su vez, de dos tipos, enteros de 16 bits (enteros cortos) o enteros de 32 bits (enteros de precisión simple), en ambos casos con o sin signo. En el caso en que se trabaje con enteros de 16 bits, el dato está recogido en los 16 LSBs del registro en el que se almacena (recuérdese que el bus de datos del DSP es de 32 bits lo que implica que los datos, salvo una excepción que luego comentaremos, siempre van a ocupar 32 bits de los cuales no todos tienen porqué ser parte del dato, pueden sobrar en definitiva). En el caso de datos enteros de 16 bits, *sobrarían* los 16 MSBs del registro donde se almacena el dato, figura 3.1. En el caso en que se trabaje con enteros de 32 bits el dato está recogido en los 32 bits del registro en el que se almacena, figura 3.1.

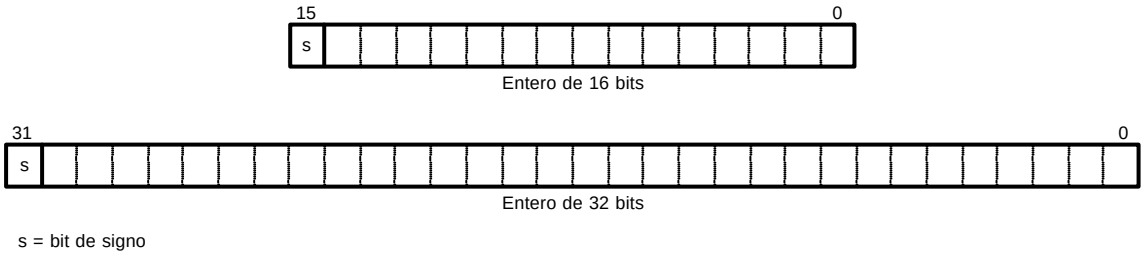


Figura 3.1. Representación de datos enteros en la familia TMS320C3x.

2. Datos **flotantes**. Pueden ser, en este caso, de tres tipos dependiendo del número de bits con los que se representa el dato: de 16 bits (flotantes cortos), de 32 bits (flotantes de precisión simple) ó de 40 bits (flotantes de precisión extendida, sólo posible con los registros R0 a R7, que son los únicos que tienen 40 bits en el DSP). En el caso de datos flotantes de 16 bits, *sobrarían* los 16 MSBs del registro donde se almacena el dato, figura 3.2. En el caso en que se trabaje con flotantes de 32 bits, el dato está recogido en los 32 bits del registro en el que se almacena, figura 3.2. Finalmente, existe la posibilidad de emplear flotantes de precisión superior a la simple, en los que el dato se representa en los 40 bits de que disponen los registros R0 a R7, internos de la CPU, figura 3.2.

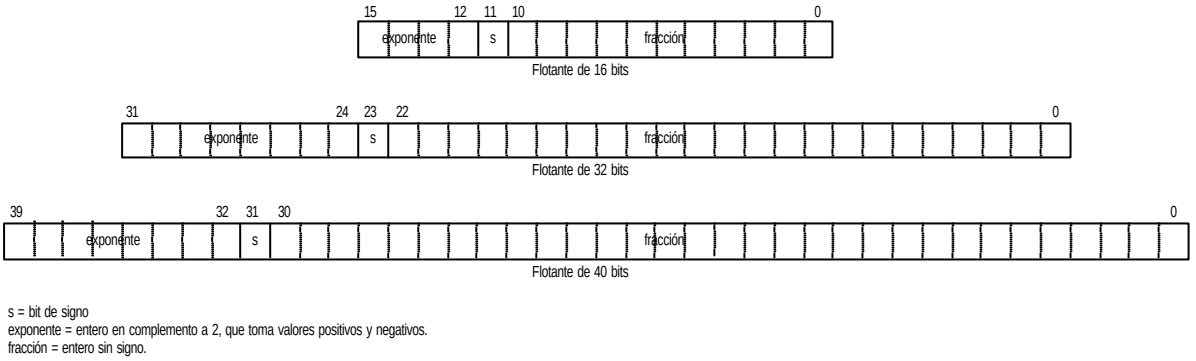


Figura 3.2. Representación de datos flotantes en la familia TMS320C3x. Se denomina *mantisa* a la parte formada por el bit de signo más la fracción.

Los números enteros con signo se representan en el conocido formato de complemento a dos, por lo que el margen de variación del entero depende del tipo de precisión que tenga (*N=16* ó *N=32* bits). En cualquier caso, el número varía entre -2^{N-1} y $2^{N-1}-1$. En el caso en que el dato represente un entero sin signo, varía entre 0 y $2^{N}-1$.

La función de transferencia que convierte en el número flotante *F* un registro de *N* (16, 32 ó 40) bits del DSP, con un conjunto de bits que constituyen el exponente, entero con signo denominado *e*, otro conjunto de bits que conforman la fracción, entero sin signo denominada *f*, más un bit de signo denotado por *s* es:

- Si el exponente *e* es el mayor número negativo posible, $F = 0$.
- Si $s = 0$ (número positivo), $F = (1 + \frac{fraccion}{2^M}) \cdot 2^e$. Donde *M* representa el número de bits de la fracción.
- Si $s = 1$ (número negativo), $F = -(2 - \frac{fraccion}{2^M}) \cdot 2^e$.

En la tabla 3-1 se muestran varios ejemplos de cálculo del flotante que representaría un registro cualquiera del DSP. En la tabla 3-2 se muestran los valores entre los que se encuentra comprendido el flotante representado con precisión corta, simple o extendida así como el paso mínimo, entre dos flotantes consecutivos, que es capaz de recoger la codificación empleada.

Codificación en el DSP	Número flotante equivalente, *F*
00B94467	-1.5526
1A651CF0	$1.2012122 \cdot 10^{8}$
459C9DCB	$-1.0486219 \cdot 10^{21}$

Tabla 3-1. Ejemplos de conversión de números en coma flotante, familia TMS320C3x.

Precisión	Menor Negativo	Mayor Negativo	Menor Positivo	Mayor Positivo
Corta (16 bits)	$-7.8163 \cdot 10^{-3}$	-256	$7.8125 \cdot 10^{-3}$	$2.5594 \cdot 10^{2}$
Simple (32 bits)	$-5.8774724 \cdot 10^{-39}$	$-3.4028236 \cdot 10^{38}$	$5.8774717 \cdot 10^{-39}$	$3.4028234 \cdot 10^{38}$
Extendida (40 bits)	$-5.8774717569 \cdot 10^{-39}$	$-3.4028236691 \cdot 10^{38}$	$5.8774717541 \cdot 10^{-39}$	$3.4028236683 \cdot 10^{38}$

Tabla 3-2. Valores entre los que está comprendido el flotante representado con precisión corta, simple o extendida, así como paso mínimo entre dos flotantes consecutivos.

La existencia de datos que pueden ser de tipo flotante o entero obliga al fabricante a facilitar la conversión entre ellos. Aparecen una serie de instrucciones básicas dedicadas a la conversión de entero a flotante y viceversa, ver las instrucciones FIX (*float to integer conversion*) y FLOAT (*integer to float conversion*) en el sumario de instrucciones. Aparecen, además, instrucciones dedicadas a operaciones con datos enteros y otras dedicadas a operaciones con datos flotantes.

Obsérvese que, en cualquier caso, el dato (ya sea entero o flotante) puede estar almacenado en cualquier registro interno del DSP con lo que recae en el usuario el manejo correcto del mismo: La CPU entenderá el dato como entero si se usa como operando de una instrucción de manejo de enteros y lo entenderá como flotante si se utiliza como operando en una instrucción de manejo de flotantes. Esto es muy importante pues los errores más comunes –no detectados en la compilación– que suelen aparecer al ejecutarse programas escritos en ensamblador están relacionados con el manejo erróneo de los datos (es entero pero se usa como flotante sin realizar una conversión previa o es flotante y se usa como entero sin realizar la conversión de formato correspondiente).

Como ejemplo, un registro que tenga el valor 0x00000000 (cero en hexadecimal) si se maneja como entero el dato que representa dicho registro es el entero cero, si se maneja como flotante, y al ser el bit de signo cero, es $F = (1 + f \cdot 1.192092896 \cdot 10^{-7}) \cdot 2^{e}$, donde tanto la fracción, *f*, como el exponente, *e*, valen cero por lo que el dato es *1.0*.

3. MODOS DE DIRECCIONAMIENTO

Cuando se analiza el software de un sistema microprocesador uno de los aspectos más importantes es el modo de direccionamiento. Hace referencia a la forma en que la CPU accede a los datos que necesita. Los DSPs de la familia TMS320C3x disponen de seis modos de direccionamiento:

1. Direccionamiento **a registro**. Hace referencia a una posible forma de acceso a datos en la que el dato se encuentra almacenado en uno de los 28 registros internos de que dispone la CPU. Ejemplo:

```
ABSF    R1          ; R1 = |R1|
                    ; Si R1, antes de ejecutarse la instrucción era –6.3, luego vale 6.3.
```

   Obsérvese que el operando, en este caso, es R1, uno de los registros internos de la CPU.

2. Direccionamiento **directo**. Hace referencia a una posible forma de acceso a datos en la que el dato se encuentra almacenado en una posición del mapa de memoria del DSP determinada por los 8 LSBs del registro interno de la CPU denominado DP, puntero a una de las 256 posibles páginas de datos que existen en el mapa de memoria del DSP y que forman los 8 MSBs de la dirección donde se encuentra el dato. Los 16 LSBs que faltan de la dirección del dato se indican en la instrucción. Ejemplo:

```
ADDI    @0BCDEh,R7  ; R7 = R7 + dato
                    ; dato se obtiene en función del valor de DP.
                    ; Si DP=0x8A se encuentra en la dirección 0x8ABCDE.
```

Antes de ejecutar la instrucción	*Después de ejecutar la instrucción*
DP=08Ah Dato en la dirección 8ABCDEh=02h R7=1h	DP=08Ah Dato en la dirección 8ABCDEh=02h R7=3h

Obsérvese que aparecen dos operandos, @0BCDEh, que es el accedido empleando direccionamiento directo y R7, al que accede la CPU empleando direccionamiento a registro y que almacena el resultado de la operación.

3. Direccionamiento **indirecto**. Hace referencia a una posible forma de acceso a un dato en la que la dirección del mismo se genera a partir de uno de los registros auxiliares, *AR0*...*AR7*, y de un desplazamiento (entero de 8 bits sin signo, de 0 a 255, especificado directamente en la instrucción o uno de los dos registros índice *IR0* ó *IR1* que existen en la CPU). La dirección efectiva del dato la constituyen los 24 LSBs que resultan de operar aritméticamente (suma), mediante las ARAUs, el registro ARi empleado y el desplazamiento definido en la instrucción. Ejemplos:

```
ADDI    *+AR3(14),R7        ; R7 = R7 + dato
                            ; dato se obtiene en función del valor de AR3.
                            ; En este caso se encuentra en la dirección 0x8ABCDE.
```

Antes de ejecutar la instrucción	*Después de ejecutar la instrucción*
AR3=08ABCD0h 14=0x0E Dato en la dirección 8ABCDEh=02h R7=1h	AR3=08ABCD0h 14=0x0E Dato en la dirección 8ABCDEh=02h R7=3h

Obsérvese que aparecen dos operandos, *+AR3(14), que es accedido empleando direccionamiento indirecto, y R7, al que se accede empleando direccionamiento a registro.

ADDI *++AR3(IR0),R7 ; $R7 = R7 + dato$

; *dato* se obtiene en función del valor de AR3 y de IR0.

; En este caso se encuentra en la dirección 0x8ABCDE.

Antes de ejecutar la instrucción	*Después de ejecutar la instrucción*
AR3=08ABCD0h IR0=0Eh Dato en la dirección 8ABCDEh=02h R7=1h	AR3=08ABCDEh IR0=0Eh Dato en la dirección 8ABCDEh=02h R7=3h

ADDI *AR3++(IR1),R7 ; $R7 = R7 + dato$

; *dato* se obtiene en función del valor de AR3.

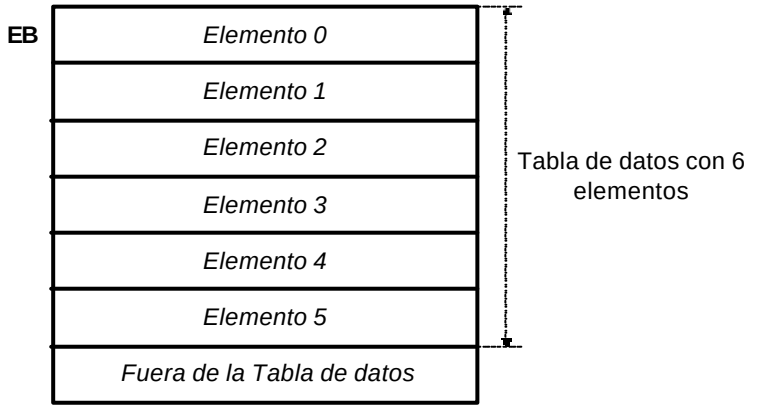
; En este caso se encuentra en la dirección 0x8ABCD0.

Antes de ejecutar la instrucción	*Después de ejecutar la instrucción*
AR3=08ABCD0h IR1=0Eh Dato en la dirección 8ABCD0h=02h R7=1h	AR3=08ABCDEh IR1=0Eh Dato en la dirección 8ABCD0h=02h R7=3h

En la tabla 3-3 se muestran los diferentes modos de direccionamiento indirecto según se emplee el desplazamiento fijo de 8 bits definido por el usuario, *disp*, que deberá valer entre 0 y 255, o uno de los dos registros de desplazamiento de la CPU: *IR0* o *IR1*. Se dispone, además, de dos formas especiales de realizar el acceso al dato empleando direccionamiento indirecto: Indirecto circular e indirecto bit-reversed. Estos direccionamientos son muy interesantes cuando se desea programar algoritmos matemáticos recursivos (convoluciones, correlaciones, FFTs). Veamos en qué consiste cada uno de estos modos especiales de direccionamiento indirecto.

- Direccionamiento **indirecto circular**. Particularmente interesante en la implementación de convoluciones y correlaciones. El cálculo de la dirección donde se ubica el dato requiere del registro interno de la CPU denominado BK. Dicho registro se emplea en la definición del tamaño de una *tabla de datos* de forma que el dato direccionado siempre se encuentre dentro de la tabla definida. Ejemplo de manejo de direccionamiento indirecto circular en el que el operando viene dado por el registro *AR0* con *BK=0110b* (tabla de datos de tamaño 6):

```
*AR0++(5)%      ; Inicialmente AR0 = Puntero al Elemento 0.
                ; Al final de la instrucción, AR0 = Puntero al Elemento 5.
*AR0++(2)%      ; Al final de la instrucción, AR0 = Puntero al Elemento 1.
*AR0—(3)%       ; Al final de la instrucción, AR0 = Puntero al Elemento 4.
*AR0++(6)%      ; Al final de la instrucción, AR0 = Puntero al Elemento 4.
*AR0--%         ; Al final de la instrucción, AR0 = Puntero al Elemento 3.
```



En la figura 3.3 se muestra cómo se calcula la dirección del dato direccionado en una *tabla de datos* definida. La dirección base de la tabla de datos, *EB*, tiene los *N+1* LSBs a cero y los MSBs coinciden con los MSBs de *ARn*. El bit *N* se determina con el registro *BK* (el bit más significativo de este registro a uno determina el valor de *N*). Obsérvese que, este tipo de direccionamiento indirecto impide que el puntero al dato exceda los límites inferior o superior de la tabla: Siempre se direcciona un dato dentro de la tabla de datos. La tabla de datos equivale, por tanto, a una memoria de datos circular (detrás del último dato aparece nuevamente el primero).

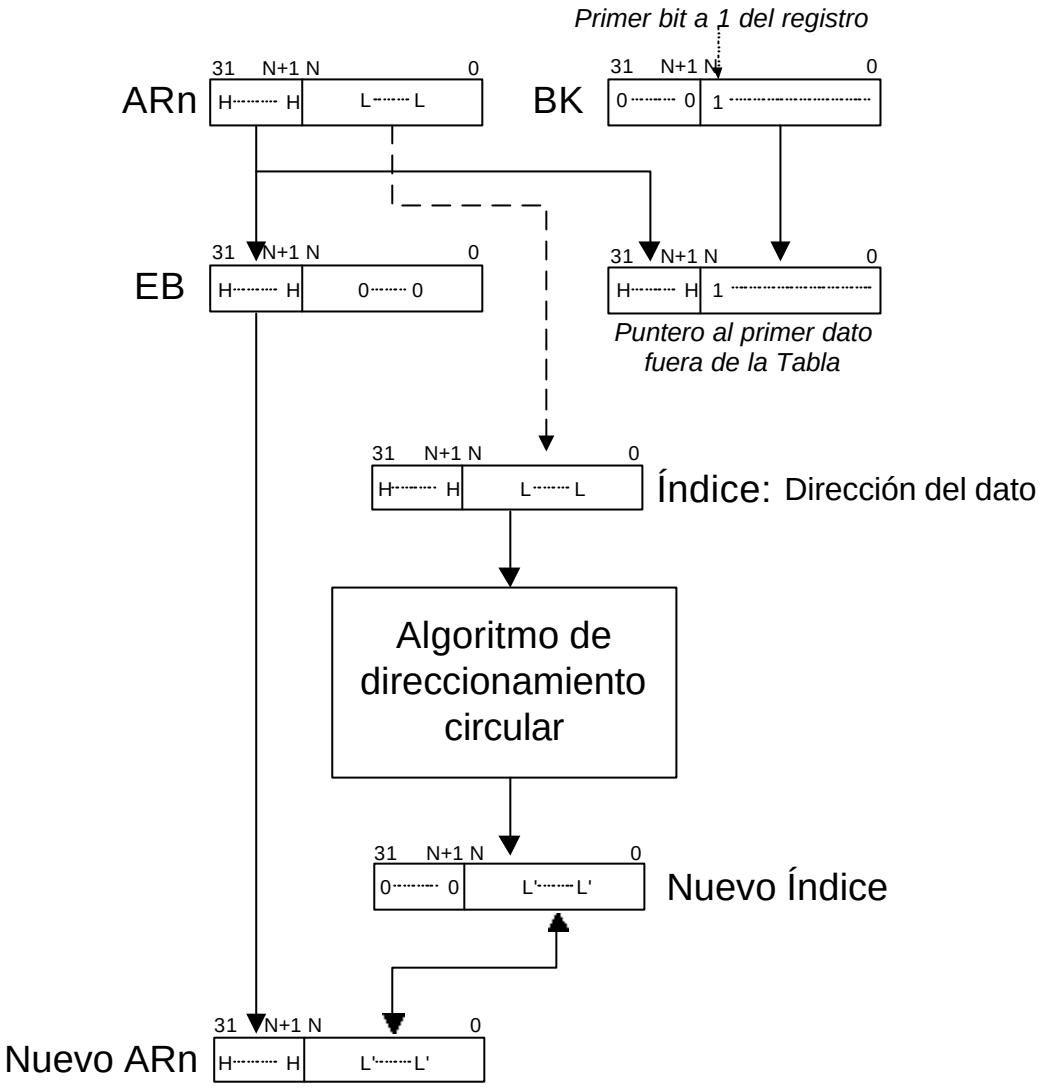


Figura 3.3. Direccionamiento Indirecto Circular.

- Direccionamiento **indirecto bit-reversed**. Particularmente interesante en la implementación de algoritmos FFTs (*Fast Fourier Transforms*) donde, si los datos iniciales que desean ser transformados están ordenados en el orden correcto, el resultado final del algoritmo óptimo implementado en este DSP los reordena en orden *bit-reversed*. El cálculo de la dirección donde se ubica el dato requiere de uno de los registros índice de la CPU, *IR0*, que se emplea, en este caso, en la definición de la mitad del tamaño de la *tabla de datos*. Al igual que en el direccionamiento indirecto de tipo circular, el puntero al dato no puede abandonar la tabla. Ejemplo de manejo y de acceso a los datos de una tabla de 16 valores, empleando direccionamiento

indirecto de tipo *bit-reversed*, en el que el operando lo determina *AR2*, con *IR0=1000b* (tabla de datos de tamaño 16):

```
*AR2++(IR0)B        ; Inicialmente AR2 = 01100000b. Elemento 0.
                    ; Al final de la instrucción, AR2 = 01101000b: Elemento 8.
*AR2++(IR0)B        ; Al final de la instrucción, AR2 = 01100100b: Elemento 4.
*AR2++(IR0)B        ; Al final de la instrucción, AR2 = 01101100b: Elemento 12.
*AR2++(IR0)B        ; Al final de la instrucción, AR2 = 01100010b: Elemento 2.
*AR2++(IR0)B        ; Al final de la instrucción, AR2 = 01101010b: Elemento 10.
*AR2++(IR0)B        ; Al final de la instrucción, AR2 = 01100110b: Elemento 6.
*AR2++(IR0)B        ; Al final de la instrucción, AR2 = 01101110b: Elemento 14.
*AR2++(IR0)B        ; Al final de la instrucción, AR2 = 01100001b: Elemento 1.
*AR2++(IR0)B        ; Al final de la instrucción, AR2 = 01101001b: Elemento 9.
*AR2++(IR0)B        ; Al final de la instrucción, AR2 = 01100101b: Elemento 5.
*AR2++(IR0)B        ; Al final de la instrucción, AR2 = 01101101b: Elemento 13.
*AR2++(IR0)B        ; Al final de la instrucción, AR2 = 01100011b: Elemento 3.
*AR2++(IR0)B        ; Al final de la instrucción, AR2 = 01101011b: Elemento 11.
*AR2++(IR0)B        ; Al final de la instrucción, AR2 = 01100111b: Elemento 7.
*AR2++(IR0)B        ; Al final de la instrucción, AR2 = 01101111b: Elemento 15.
*AR2++(IR0)B        ; Al final de la instrucción, AR2 = 01100000b: De nuevo elemento 0.
```

En la tabla 3-3 se muestran todas las posibles formas de acceso al dato que emplean el modo de direccionamiento indirecto.

Operando	Operación	Descripción
Direccionamiento Indirecto con desplazamiento fijo de 8 bits, *disp* $disp \in N$, $disp \subset [0,255]$		
*+ARn(disp)	Addr=ARn+disp	Desplazamiento positivo, previo el acceso, de la dirección del dato.
*-ARn(disp)	Addr=ARn-disp	Desplazamiento negativo, previo el acceso, de la dirección del dato.
*++ARn(disp)	Addr=ARn+disp ARn=ARn+disp	Desplazamiento positivo, previo el acceso, de la dirección del dato. Modifica ARn.

*--ARn(disp)	Addr=ARn-disp ARn=ARn-disp	Desplazamiento negativo, previo el acceso, de la dirección del dato. Modifica ARn.
*ARn++(disp)	Addr=ARn ARn=ARn+disp	Desplazamiento positivo, posterior al acceso, de la dirección del dato. Modifica ARn.

*ARn--(disp)	Addr=ARn ARn=ARn-disp	Desplazamiento negativo, posterior al acceso, de la dirección del dato. Modifica ARn.
*ARn++(disp)%	Addr=ARn ARn=circ(ARn+disp)	Desplazamiento positivo, posterior al acceso, de la dirección del dato. Modifica ARn empleando direccionamiento circular.
*ARn--(disp)%	Addr=ARn ARn=circ(ARn-disp)	Desplazamiento negativo, posterior al acceso, de la dirección del dato. Modifica ARn empleando direccionamiento circular.
Direccionamiento Indirecto con desplazamiento dado por IR0 ó IR1		
*+ARn(IRi)	Addr=ARn+IRi	Desplazamiento positivo, previo el acceso, de la dirección del dato.
*-ARn(IRi)	Addr=ARn-IRi	Desplazamiento negativo, previo el acceso, de la dirección del dato.
*++ARn(IRi)	Addr=ARn+IRi ARn=ARn+IRi	Desplazamiento positivo, previo el acceso, de la dirección del dato. Modifica ARn.
*--ARn(IRi)	Addr=ARn-IRi ARn=ARn-IRi	Desplazamiento negativo, previo el acceso, de la dirección del dato. Modifica ARn.
*ARn++(IRi)	Addr=ARn ARn=ARn+IRi	Desplazamiento positivo, posterior al acceso, de la dirección del dato. Modifica ARn.
*ARn--(IRi)	Addr=ARn ARn=ARn-IRi	Desplazamiento negativo, posterior al acceso, de la dirección del dato. Modifica ARn.
*ARn++(IRi)%	Addr=ARn ARn=circ(ARn+IRi)	Desplazamiento positivo, posterior al acceso, de la dirección del dato. Modifica ARn empleando direccionamiento circular.
*ARn--(IRi)%	Addr=ARn ARn=circ(ARn-IRi)	Desplazamiento negativo, posterior al acceso, de la dirección del dato. Modifica ARn empleando direccionamiento circular.
Casos especiales de direccionamiento indirecto		
*Arn	Addr=ARn	Direccionamiento Indirecto clásico.
*ARn++(IR0)B	Addr=ARn ARn=bitrev(ARn+IR0)	Desplazamiento positivo, posterior al acceso, de la dirección del dato. Modifica ARn empleando direccionamiento *bit-reversed*.

Tabla 3-3. Modos de direccionamiento indirecto, DSP TMS320C3x.

4. Direccionamiento **inmediato corto**. Hace referencia a una posible forma de acceso a datos en la que el operando es un valor de 16 bits que se incluye en la propia instrucción. El dato puede ser un entero o un flotante dependiendo del tipo de instrucción que lo requiera como operando (instrucción de manejo de enteros o de flotantes). En ambos casos se trata de un operando de precisión simple (16 bits). Ejemplo:

```
SUBI    1,R7        ; R7 = R7 – 1
                    ; Antes de la instrucción, R7=6d. Luego, R7=5d.
```

5. Direccionamiento **inmediato largo**. Hace referencia a una posible forma de acceso a datos en la que el operando es un valor de 24 bits que se incluye en la propia instrucción. Este tipo de direccionamiento se emplea en instrucciones que provocan un cambio en el valor del registro PC (contador de programa del DSP). Ejemplo:

```
BR      8000h       ; Antes de la instrucción, PC=100h. Luego, PC=8000h.
```

6. Direccionamiento **relativo al contador de programa**. Hace referencia a una posible forma de acceso a datos en la que el operando es un desplazamiento relativo al valor, justo antes de la ejecución de la instrucción que emplea este modo de direccionamiento, del registro contador de programa. Este tipo de direccionamiento se emplea en instrucciones que provocan un cambio en el valor del registro PC (contador de programa del DSP) pero, al contrario que en el caso anterior, el valor a almacenar en el registro *PC* lo determina el propio ensamblador de forma transparente al programador. Ejemplos:

```
BU      NEWPC       ; Antes de la instrucción, PC=1000h y NEWPC es una etiqueta que
                    ; apunta a la dirección 1005h.
                    ; Después de la instrucción, PC=1005h. El desplazamiento producido
                    ; en el PC (el dato u operando) es 5h (NEWPC-PC).
BUD     NEWPC       ; Antes de la instrucción, PC=1000h y NEWPC es una etiqueta que
                    ; apunta a la dirección 1005h.
                    ; Después de la instrucción, PC=1005h. El desplazamiento producido
                    ; en el PC (el dato u operando) es 2h (NEWPC-(PC+3)), no 5h, debido
                    ; a que, en las instrucciones de salto con retardo, el salto efectivo no
se
                    ; produce hasta 3 instrucciones después de la instrucción de salto.
```

4. PILA

La ruptura de la secuencia normal de ejecución de un programa, al generarse llamadas definidas por el usuario (salto a subrutinas) o al atenderse una petición de interrupción

habilitada (salto a la rutina de interrupción correspondiente), requiere del empleo de una *pila* o zona de almacenamiento de datos de tipo FILO (entrada de datos a la pila con incremento asociado del puntero a la misma y salida de datos con decremento asociado del puntero a la pila).

Los DSPs de la familia TMS320C3x emplean, como pila, una tabla creciente[1] de datos en la que el puntero a los datos de la pila es el registro interno de la CPU denominado *SP* (*Stack pointer*). El registro *SP* apunta al último elemento introducido en la pila. El uso de la pila requiere de la inicialización, antes de ser usada, de la zona de memoria del DSP reservada a la pila. Para el manejo directo de la pila se suministran cuatro instrucciones en ensamblador, PUSH, POP, PUSHF y POPF (ver sumario de instrucciones). En la figura 3.4 se muestra la estructura básica de la pila de este DSP.

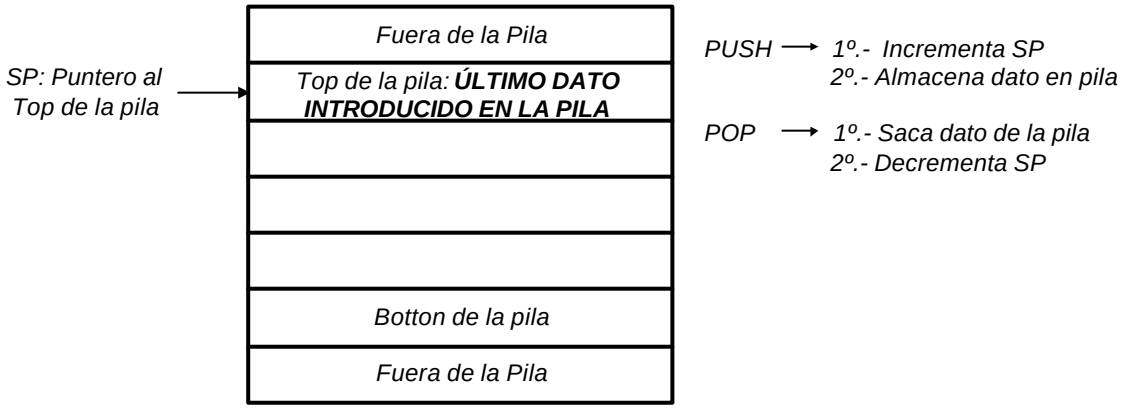


Figura 3.4. Estructura de la Pila en los DSPs de la familia TMS320C3x.

5. SUMARIO DE INSTRUCCIONES

El formato o forma genérica de expresar las instrucciones en ensamblador de un DSP de la familia TMS320C3x de Texas Instruments es como sigue:

MNEMÓNICO SRC1,SRC2,DST

Donde *MNEMÓNICO* hace referencia a una instrucción en particular, por ejemplo el mnemónico ADDI hace referencia a una instrucción que realiza una operación de suma con operandos enteros, *SRC1* y *SRC2* hacen referencia a los operandos fuente y *DST* al operando destino de la instrucción.

Obsérvese que hay instrucciones que no requieren de operandos (por ejemplo, la instrucción *NOP*, que no realiza operación alguna). Otras, sólo requieren un operando (el destino, por ejemplo, la instrucción *PUSH AR0*, que almacena el registro *AR0* en la pila del DSP), o dos operandos (uno fuente y otro que hace las veces de destino o de fuente y destino, por ejemplo, *ADDI R0,R7*, que suma el contenido del registro *R0* al contenido del registro *R7* almacenando el resultado en el registro *R7*). Por último, existen instrucciones que requieren tres operandos (dos fuentes y un destino como, por ejemplo, *ADDI R0,R2,R7*, que suma el contenido del registro *R0* al contenido del registro *R2* almacenando el resultado en el registro *R7*).

Destacar, asociado a estos microprocesadores, las siguientes características especiales del *software*:

1. Las instrucciones se pueden ejecutar todas en un único ciclo máquina salvo instrucciones de ruptura, controlada por el usuario, de la secuencia de ejecución del programa (instrucciones de salto) que tardan, al no poder aprovechar la estructura de ejecución en paralelo de microinstrucciones del DSP (estructura *pipeline* vista anteriormente), cuatro ciclos máquinas en hacerse efectivas. Para posibilitar la optimización de la ejecución de las instrucciones de salto se definen unas instrucciones de salto con retardo, *BRD*. Estas instrucciones se ejecutan en un único ciclo máquina puesto que el salto efectivo se

[1] Recuérdese que en el Motorola MC68HC11 la pila era decreciente, el registro puntero a la pila se decrementaba antes de introducir un nuevo valor y se incrementaba luego de sacar de ella un dato.

produce con el aprovechamiento de la estructura *pipeline* (las tres instrucciones que siguen al salto se ejecutan antes de producirse el salto efectivo).

2. Posibilidad de ejecutar dos operaciones en paralelo. Gracias a la existencia interna de múltiples buses de datos así como de operadores aritméticos independientes se pueden realizar dos operaciones, simultáneamente, en la CPU del DSP. Las instrucciones que permiten este tipo de operaciones tienen la forma genérica:

   MNEMÓNICO1 ***SRC1,SRC2,DST1***

   || ***MNEMÓNICO2*** ***SRC3,SRC4,DST2***

3. Posibilidad de ejecutar múltiples veces una instrucción (*RPTS*) o varias (*RPTB*, repetición de un bloque de instrucciones) sin que ello suponga algún tipo de penalización por la ejecución de bucles. En el caso de ejecución de una única instrucción múltiples veces, *RPTS*, la fase FETCH de recogida de la instrucción es posible realizarla desde el registro IR de la CPU, con lo que se evita cualquier penalización (en tiempo de ejecución) en el acceso a memoria para recoger la instrucción.

4. Existencia de instrucciones que facilitan el acceso de dos procesadores a una única memoria (*instrucciones de interbloqueo*). Se definen una serie de instrucciones que permiten la implementación sencilla de un semáforo y que, de manera coherente, dos microprocesadores compartan una zona común de datos generando, para ello, el arbitraje necesario que evite los conflictos en el acceso a los datos comunes. La familia TMS320C3x dispone de cinco instrucciones de *interbloqueo* (*LDFI*, *LDII*, *SIGI*, *STFI*, *STII*).

   Estas instrucciones le ofrecen al usuario una potente herramienta de sincronización en el procesamiento en paralelo de DSPs, garantizando la integridad y la velocidad de la

comunicación. Emplean los puertos de entrada y salida de propósito general, XF0 y XF1. XF0 debe configurarse como línea de salida y XF1 como línea de entrada de forma que, XF0 genere la señal de petición de bloqueo y XF1 funcione como línea de reconocimiento del bloqueo solicitado. Ambas señales son activas a nivel bajo. Los accesos en lectura y escritura (*LDFI, LDII, STFI, STII*), generados por las instrucciones *interbloqueadas*, son idénticos a los normales salvo que, además, usan las líneas XF0 y XF1. Los operandos de esas instrucciones deben ser siempre accedidos por direccionamiento directo o indirecto.

Al emplear las instrucciones *LDFI* y *LDII*, XF0 se pone a cero si y sólo si el operando fuente se ubica en **memoria externa** al DSP ($\overline{STRB}$, $\overline{MSTRB}$ ó $\overline{IOSTRB}$ activos) o en la zona de **memoria interna de periféricos**. En caso contrario, acceso a memoria interna de tipo RAM por ejemplo, las operaciones de lectura *interbloqueadas* no activan XF0 y son idénticas a las operaciones de lectura normales que realiza el bus. Cuando se ejecuta una instrucción de tipo *LDFI* ó *LDII*, la CPU realiza las siguientes operaciones:

1. Comienza el acceso en lectura al operando: Se impone en el bus la dirección del dato y, simultáneamente, se activa XF0 ($XF0 = 0$).
2. Se extiende, en el tiempo, el acceso en lectura al dato hasta que no se activen XF1 ($XF1 = 0$) y la variable $\overline{RDY_{\text{int}}}$ (acceso con $\overline{STRB}$ activo) ó $\overline{XRDY_{\text{int}}}$ (acceso con $\overline{MSTRB}$ ó $\overline{IOSTRB}$ activos).

La estructura de la CPU permite la atención de una interrupción activa que aparezca cuando se esté ejecutando una instrucción *LDFI* ó *LDII* siempre que se haya activado la señal $\overline{RDY_{\text{int}}}$ al DSP y aunque no se haya reconocido el bloqueo (aunque no se haya activado XF1). Al final de la ejecución de la rutina de servicio de la interrupción se vuelve a la instrucción siguiente a la *interbloqueada*. Ésta es la única forma de evitar, en el caso de que no llegue nunca el reconocimiento externo del bloqueo ($XF1 = 0$), que la CPU quede bloqueada al ejecutar las instrucciones *LDFI* ó *LDII*.

Al emplear las instrucciones *STFI* y *STII*, XF0 se pone a uno si y sólo si el operando destino se ubica en **memoria externa** al DSP ($\overline{STRB}$, $\overline{MSTRB}$ ó $\overline{IOSTRB}$ activos) o en la zona de **memoria interna de periféricos**. En caso contrario, acceso a memoria interna de tipo RAM por ejemplo, las operaciones de escritura *interbloqueadas* no desactivan XF0 y son idénticas a las operaciones de escritura normales que realiza el bus. Cuando se ejecuta una instrucción de tipo *STFI* ó *STII*, la CPU realiza las siguientes operaciones:

1. Comienza el acceso en escritura al operando: Se impone en el bus la dirección del dato y, simultáneamente, se desactiva XF0 ($XF0 = 1$).
2. Se extiende, en el tiempo, el acceso en escritura del dato hasta que no se active la variable $\overline{RDY_{\text{int}}}$ (acceso con $\overline{STRB}$ activo) ó $\overline{XRDY_{\text{int}}}$ (acceso con $\overline{MSTRB}$ ó $\overline{IOSTRB}$ activos).

Al emplear la instrucción *SIGI*, que no dispone de operandos, la CPU realiza las siguientes operaciones:

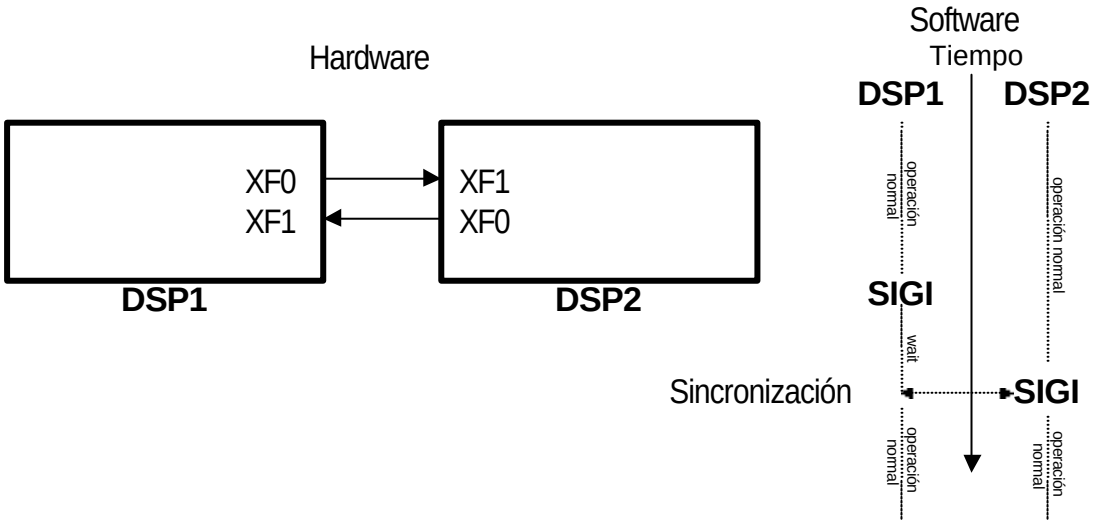
1. Se activa XF0 ($XF0 = 0$).
2. La CPU se queda en estado de espera hasta que detecte la activación de XF1 ($XF1 = 0$).
3. Termina de ejecutar la instrucción desactivando XF0 ($XF0 = 1$).

Ejemplo1: Multiproceso en el que DSP1 espera, en el bucle L1, hasta que DSP2 ponga el operando indicado por LOCK a cero. Sólo se abandona el bucle L1 si el DSP2 escribe un valor cero en la dirección @LOCK (lo debe hacer cuando XF0 del DSP valga 0) y, antes de abandonar el bucle, el DSP1 vuelve a activar el semáforo (pone la dirección @LOCK a uno).

```
    LDI     1,R0                ; R0=1
L1: LDII    @LOCK,R1            ; Comienza el interbloqueo: R1=[LOCK], XF0=0. Afecta a
```

```
                                ; las banderas del registro ST.
STII    R0,@LOCK                ; [LOCK]=1, XF0=1; Fin del interbloqueo. No afecta a las
                                ; banderas del registro ST, no afecta al salto.
BNZ     L1                      ; Salta a L1 mientras la condición para el salto no se haya
                                ; activado con la instrucción LDII (R1!=0). La única forma de
                                ; salir del bucle es que un dispositivo externo escriba en
                                ; @LOCK un 0 cuando XF0 vale 0.
```

Ejemplo2: Multiproceso en el que DSP1 ejecuta el programa en su secuencia normal hasta que llega a la instrucción *SIGI*. Entonces, espera hasta que DSP2 quite el bloqueo (ejecute *SIGI*). Para que el proceso de ejecución esté sincronizado, es preciso conectar XF0 del DSP1 (configurado como línea de salida del microprocesador) a XF1 del DSP2 (configurado como línea de entrada) y XF1 del DSP1 (configurado como línea de entrada) a XF0 del DSP2 (configurado como línea de salida).



En la tabla 3-4 se enumeran las instrucciones de almacenamiento y carga de datos de que dispone el DSP. En la tabla 3-5 se enumeran las instrucciones que realizan operaciones aritméticas simples. En la tabla 3-6 se enumeran las instrucciones que realizan operaciones lógicas. En la tabla 3-7 se enumeran las instrucciones de control del formato del dato. En la tabla 3-8 se enumeran las instrucciones de control del contador de programa. En la tabla 3-9

se enumeran las instrucciones que introducen al DSP en el modo de funcionamiento de bajo consumo. En la tabla 3-10 se enumeran las instrucciones interbloqueadas y, finalmente, en la tabla 3-11 se enumeran las instrucciones que realizan dos operaciones en paralelo.

Mnemónico	Descripción breve
LDE	Carga el exponente de un número flotante.
LDF	Carga un número flotante.
LDFcond	Carga un número flotante si se cumple la condición definida.
LDI	Carga un número entero.
LDIcond	Carga un número entero si se cumple la condición definida.
LDM	Carga la mantisa de un número flotante.
LDP	Carga el puntero a la página de datos (direccionamiento directo).
POP	Recoge entero de la pila.
POPF	Recoge flotante de la pila.
PUSH	Almacena entero en la pila.
PUSHF	Almacena flotante en la pila.
STF	Almacena flotante.
STI	Almacena entero.

Tabla 3-4. Operaciones básicas de carga y almacenamiento de datos, TMS320C3x.

Mnemónico	Descripción breve
ABSF	Calcula el valor absoluto de un flotante.
ABSI	Calcula el valor absoluto de un entero.
ADDC	Suma enteros (al resultado le suma el valor del bit de acarreo).
ADDC3	Suma enteros (al resultado le suma el valor del bit de acarreo).
ADDF	Suma flotantes.
ADDF3	Suma flotantes.
ADDI	Suma enteros.
ADDI3	Suma enteros.
CMPF	Compara dos flotantes.
CMPF3	Compara dos flotantes.
CMPI	Compara dos enteros.
CMPI	Compara dos enteros.
MPYF	Multiplica flotantes.
MPYF3	Multiplica flotantes.
MPYI	Multiplica enteros.
MPYI3	Multiplica enteros.
NEGB	Calcula el entero cambiado de signo (al resultado le resta el valor del bit de acarreo).
NEGF	Calcula el flotante cambiado de signo.
NEGI	Calcula el entero cambiado de signo.
SUBB	Resta enteros (al resultado le resta el valor del bit de acarreo).
SUBB3	Resta enteros (al resultado le resta el valor del bit de acarreo).
SUB*cond*	Resta enteros, si se cumple una determinada condición.
SUBF	Resta flotantes.
SUBF3	Resta flotantes.
SUBI	Resta enteros.
SUBI3	Resta enteros.
SUBRB	Resta enteros (al resultado le resta el valor del bit de acarreo). Similar a SUBB.
SUBRF	Resta flotantes. La operación de resta es la contraria que en SUBF.
SUBRI	Resta enteros. La operación de resta es la contraria que en SUBI.

Tabla 3-5. Operaciones aritméticas simples básicas, TMS320C3x.

Mnemónico	Descripción breve
AND	And lógico.
AND3	And lógico.
ANDN	And lógico con complemento.
ANDN3	And lógico con complemento.
ASH	Desplazamiento aritmético.
ASH3	Desplazamiento aritmético.
LSH	Desplazamiento lógico.
LSH3	Desplazamiento lógico.
NOT	Complemento a uno.
OR	Or lógico.
OR3	Or lógico.
ROL	Giro a la izquierda.
ROLC	Giro a la izquierda a través del bit de acarreo.
ROR	Giro a la derecha.
RORC	Giro a la derecha a través del bit de acarreo.
TSTB	Comprueba un campo de bits.
TSTB3	Comprueba un campo de bits.
XOR	Xor lógico.
XOR3	Xor lógico.

Tabla 3-6. Operaciones lógicas básicas, TMS320C3x.

Mnemónico	Descripción breve
FIX	Convierte de flotante a entero.
FLOAT	Convierte de entero a flotante.
NORM	Normaliza flotante.
RND	Redondea flotante.

Tabla 3-7. Operaciones básicas de control del formato de los datos, TMS320C3x.

Mnemónico	Descripción breve
Bcond	Salto condicionado.
BcondD	Salto condicionado con retardo.
BR	Salto incondicional.
BRD	Salto incondicional con retardo.
CALL	Llamada a subrutina.
CALLcond	Llamada a subrutina condicionada.
DBcond	Decrementa y salto condicionado.
DBcondD	Decrementa y salto condicionado con retardo.
IACK	Reconocimiento de interrupción.
IDLE	Espera hasta la aparición de una interrupción.
NOP	No operación.
RETIcond	Retorno condicionado de rutina de servicio de interrupción.
RETScond	Retorno condicionado de subrutina.
RPTB	Repite bloque de instrucciones.

RPTS	Repite una instrucción.
SWI	Interrupción software. El fabricante indica, en el manual del DSP, que esta instrucción no debe ser utilizada por un usuario.
TRAPcond	Llamada condicionada a vector TRAP.

Tabla 3-8. Operaciones básicas de control del contador de programa, TMS320C3x.

Mnemónico	Descripción breve
IDLE2	Para el contador de programa y elimina la entrada de reloj al sistema digital.
LOPOWER	Divide por 16 la fuente de reloj, H1=H1/16.
MAXSPEED	Restaura el reloj a su velocidad normal.

Tabla 3-9. Operaciones básicas de control del modo de funcionamiento de bajo consumo, sólo para el DSP TMS320LC31.

Mnemónico	Descripción breve
LDFI	Carga valor flotante con interbloqueo.
LDII	Carga valor entero con interbloqueo.
SIGI	Genera señales de interbloqueo.
STFI	Almacena valor flotante con interbloqueo.
STII	Almacena valor entero con interbloqueo.

Tabla 3-10. Operaciones básicas con interbloqueo, TMS320C3x.

Mnemónico	Descripción breve
ABSF || STF	Calcula el valor absoluto de un flotante y almacena otro flotante.
ABSI || STI	Calcula el valor absoluto de un entero y almacena otro entero.
ADDF3 || STF	Suma dos flotantes y almacena otro flotante.
ADDI3 || STI	Suma dos enteros y almacena otro entero.
AND3 || STI	AND lógica bit a bit de dos registros y almacena un entero.
ASH3 || STI	Desplazamiento aritmético y almacena un entero.
FIX || STI	Convierte a entero un flotante y almacena otro entero.
FLOAT || STF	Convierte a flotante un entero y almacena otro flotante.
LDF || LDF	Carga dos flotantes.
LDF || STF	Carga un flotante y almacena otro.
LDI || LDI	Carga dos enteros.
LDI || STI	Carga un entero y almacena otro.
LSH3 || STI	Desplazamiento lógico y almacena un entero.
MPYF3 || ADDF3	Multiplica dos flotantes y suma otros dos flotantes.
MPYF3 || STF	Multiplica dos flotantes y almacena otro flotante.
MPYF3 || SUBF3	Multiplica dos flotantes y resta otros dos flotantes.
MPYI3 || ADDI3	Multiplica dos enteros y suma otros dos enteros.
MPYI3 || STI	Multiplica dos enteros y almacena otro entero.
MPYI3 || SUBI3	Multiplica dos enteros y resta otros dos enteros.
NEGF || STF	Cambia de signo un flotante y almacena otro flotante.
NEGI || STI	Cambia de signo un entero y almacena otro entero.
NOT || STI	Complemento a uno de un registro y almacena un entero.
OR3 || STI	OR lógica bit a bit de dos registros y almacena un entero.
STF || STF	Almacena dos flotantes.
STI || STI	Almacena dos enteros.
SUBF3 || STF	Resta dos flotantes y almacena otro flotante.
SUBI3 || STI	Resta dos enteros y almacena otro entero.
XOR3 || STI	XOR lógico bit a bit de dos registros y almacena un entero.

Tabla 3-11. Operaciones en paralelo, TMS320C3x.

En la tabla 3-12 se describen, por orden alfabético, las instrucciones de que dispone esta familia de microprocesadores de Texas Instruments. Las instrucciones condicionadas, disponibles en esta familia de DSPs, precisan de un código que indica el tipo de condición que debe de cumplirse para que la instrucción se ejecute. En la tabla 3-13 se suministran los códigos y tipos de condición que contempla el ensamblador de la familia TMS320C3x.

Mnemónico	Operación	Descripción
Instrucciones simples		
Tipo de operando (modos de direccionamiento posibles al operando): Cualquier modo de direccionamiento es posible (operando ***src, count***), operandos ***src1***, ***src2*** (direccionamiento a registro o indirecto con desplazamiento 0, 1, IR0 ó IR1), operando ***Csrc*** (direccionamiento a registro o relativo al contador de programa), modo de direccionamiento a registro (operandos ***Dreg***, ***Sreg***), modo de direccionamiento a registro R0-R7 (operando ***Rn***), modo de direccionamiento a registro AR0-AR7 (operando ***ARn***), modo de direccionamiento inmediato largo (operando ***addr***). Códigos de condición, ***cond***, tabla 3-13.		
ABSF	$\lvert src \rvert \rightarrow Rn$	Calcula el valor absoluto de un número flotante.
ABSI	$\lvert src \rvert \rightarrow Dreg$	Calcula el valor absoluto de un número entero.
ADDC	$src + Dreg + C \rightarrow Dreg$	Suma enteros con acarreo.
ADDC3	$src1 + src2 + C \rightarrow Dreg$	Suma enteros con acarreo (tres operandos).
ADDF	$src + Rn \rightarrow Rn$	Suma números flotantes.
ADDF3	$src1 + src2 \rightarrow Rn$	Suma números flotantes (tres operandos).
ADDI	$src + Dreg \rightarrow Dreg$	Suma enteros sin acarreo.
ADDI3	$src1 + src2 \rightarrow Dreg$	Suma enteros sin acarreo (tres operandos).
AND	$Dreg \times src \rightarrow Dreg$	AND lógica.
AND3	$src1 \times src2 \rightarrow Dreg$	AND lógica (tres operandos).
ANDN	$Dreg \times \overline{src} \rightarrow Dreg$	AND lógica con complemento.
ANDN3	$src1 \times \overline{src2} \rightarrow Dreg$	AND lógica con complemento (tres operandos).
ASH	**if** $count \geq 0 \Rightarrow$ $Dreg << count \rightarrow Dreg$ **else** $Dreg >> count \rightarrow Dreg$	Desplazamiento aritmético (conserva el signo) *count* veces. Si *count*>0, desplazamiento a la izquierda, si *count*<0, a la derecha.
ASH3	**if** $count \geq 0 \Rightarrow$ $src << count \rightarrow Dreg$ **else** $src >> \lvert count \rvert \rightarrow Dreg$	Desplazamiento aritmético (conserva el signo) *count* veces, de tres operandos. Si *count*>0, desplazamiento a la izquierda, si *count*<0, a la derecha.
B*cond*	**if** *cond* es cierto **if** *Csrc* es un registro, $Csrc \rightarrow PC$ **if** *Csrc* es un valor, $Csrc + PC \rightarrow PC$ **else** $PC + 1 \rightarrow PC$	Salto condicional.

B*cond*D	**if** *cond* es cierto **if** *Csrc* es un registro, $Csrc \rightarrow PC$ **if** *Csrc* es un valor, $Csrc + PC + 3 \rightarrow PC$ **else** $PC + 1 \rightarrow PC$	Salto condicional con retardo.
BR	$Value \rightarrow PC$	Salto incondicional.
BRD	$Value \rightarrow PC$	Salto incondicional con retraso.

CALL	$PC+1 \rightarrow *++SP$ (Top Pila) $Value \rightarrow PC$	Llamada a subrutina.
CALL*cond*	**if** *cond* es cierto $PC+1 \rightarrow *++SP$ **if** *Csrc* es un registro, $Csrc \rightarrow PC$ **if** *Csrc* es un valor, $Csrc+PC \rightarrow PC$ **else** $PC+1 \rightarrow PC$	Llamada condicional a subrutina.
CMPF	Activa los **flags de operación** (Registro ST) como consecuencia de realizar $Rn-src$	Compara valores flotantes. Para realizar la comparación, resta los dos valores pero no almacena el resultado de la operación.
CMPF3	Activa los **flags de operación** (Registro ST) como consecuencia de realizar $src1-src2$	Compara valores flotantes. Para realizar la comparación, resta los dos valores pero no almacena el resultado de la operación (tres operandos).
CMPI	Activa los **flags de operación** (Registro ST) como consecuencia de realizar $Dreg-src$	Compara valores enteros. Para realizar la comparación, resta los dos valores pero no almacena el resultado de la operación.
CMPI3	Activa los **flags de operación** (Registro ST) como consecuencia de realizar $src1-src2$	Compara valores enteros. Para realizar la comparación, resta los dos valores pero no almacena el resultado de la operación (tres operandos).
DB*cond*	$ARn-1 \rightarrow ARn$ **if** *cond* es cierto y $ARn \geq 0$ **if** *Csrc* es un registro, $Csrc \rightarrow PC$ **if** *Csrc* es un valor, $Csrc+PC+1 \rightarrow PC$ **else** $PC+1 \rightarrow PC$	Decrementa y salta condicionalmente.
DB*cond*D	$ARn-1 \rightarrow ARn$ **if** *cond* es cierto y $ARn \geq 0$ **if** *Csrc* es un registro, $Csrc \rightarrow PC$ **if** *Csrc* es un valor, $Csrc+PC+3 \rightarrow PC$ **else** $PC+1 \rightarrow PC$	Decrementa y salta condicionalmente y con retardo.
FIX	$(\text{int})(src) \rightarrow Dreg$	Convierte un valor flotante en entero.
FLOAT	$(float)(src) \rightarrow Rn$	Convierte un valor entero en flotante.
IACK	$\overline{IACK}$ se pone a 0 y luego a 1 Lectura sin utilidad de *src*	Reconocimiento de interrupción.
IDLE	$PC+1 \rightarrow PC$ Idle hasta la próxima interrupción	Parada de la CPU hasta la llegada de una interrupción.
LDE	$\exp(src) \rightarrow \exp(Rn)$	Carga en el exponente de *Rn* el de otro valor flotante.
LDF	$src \rightarrow Rn$	Carga en *Rn* un valor flotante.
LDF*cond*	**if** *cond* es cierto $src \rightarrow Rn$ **else** no se actualiza *Rn*	Carga condicionalmente en *Rn* un valor flotante.
LDFI	$src \rightarrow Rn$, interbloqueado	Pone $XF0=0$ y comienza un ciclo de lectura del flotante *src*. Ejecuta un LDF y extiende el ciclo de lectura hasta que no se ponga $XF1=0$. Termina el acceso y deja $XF0=0$.
LDI	$src \rightarrow Dreg$	Carga un entero.
LDI*cond*	**if** *cond* es cierto $src \rightarrow Dreg$ **else** no se actualiza *Dreg*	Carga un entero condicionalmente.

LDII	$src \rightarrow Dreg$, interbloqueado	Pone $XF0=0$ y comienza un ciclo de lectura del entero *Dreg*. Ejecuta un LDI y extiende el ciclo de lectura hasta que no se ponga $XF1=0$. Termina el acceso y deja $XF0=0$.
LDM	$mantisa(src) \rightarrow mantisa(Rn)$	Carga en la mantisa de *Rn* la de otro valor flotante
LSH	**if** $count \geq 0 \Rightarrow$ $Dreg << count \rightarrow Dreg$ **else** $Dreg >> count \rightarrow Dreg$	Desplazamiento lógico (no conserva el signo) *count* veces. Si *count*>0, desplazamiento a la izquierda, si *count*<0, a la derecha.
LSH3	**if** $count \geq 0 \Rightarrow$ $src << count \rightarrow Dreg$ **else** $src >> \lvert count \rvert \rightarrow Dreg$	Desplazamiento aritmético (conserva el signo) *count* veces, de tres operandos. Si *count*>0, desplazamiento a la izquierda, si *count*<0, a la derecha.
MPYF	$src \times Rn \rightarrow Rn$	Multiplica valores flotantes.
MPY3	$src1 \times src2 \rightarrow Rn$	Multiplica valores flotantes (tres operandos).
MPYI	$src \times Dreg \rightarrow Dreg$	Multiplica valores enteros.
MPYI3	$src1 \times src2 \rightarrow Dreg$	Multiplica valores enteros (tres operandos).
NEGB	$0 - src - C \rightarrow Dreg$	Entero cambiado de signo menos bit de acarreo.
NEGF	$0 - src \rightarrow Rn$	Flotante cambiado de signo.
NEGI	$0 - src \rightarrow Dreg$	Entero cambiado de signo.
NOP	Modifica *ARn*, si se especifica en la instrucción	Consume un ciclo máquina sin realizar operación alguna.
NORM	$Normaliza(src) \rightarrow Rn$	Normaliza un valor flotante.
NOT	$\overline{src} \rightarrow Dreg$	NOT lógica.
OR	$Dreg + src \rightarrow Dreg$	OR lógica.
OR3	$src1 + src2 \rightarrow Dreg$	OR lógica (tres operandos).
POP	$*SP-- \rightarrow Dreg$	Recupera el último entero almacenado en la pila y actualiza (decrementa) el puntero a la pila.
POPF	$*SP-- \rightarrow Rn$	Recupera el último flotante almacenado en la pila y actualiza (decrementa) el puntero a la pila.
PUSH	$Sreg \rightarrow *++SP$	Actualiza (incrementa) el puntero de pila y almacena un entero en la pila.
PUSHF	$Rn \rightarrow *++SP$	Actualiza (incrementa) el puntero de pila y almacena un flotante en la pila.
RETI*cond*	**if** *cond* es cierto o no existe $*SP-- \rightarrow PC$ $1 \rightarrow ST(GIE)$ **else** $PC+1 \rightarrow PC$	Retorno condicional de la rutina de servicio de una interrupción.
RETS*cond*	**if** *cond* es cierto o no existe $*SP-- \rightarrow PC$ **else** $PC+1 \rightarrow PC$	Retorno condicional de una rutina.
RND	$redondeo(src) \rightarrow Rn$	Redondea un valor flotante.
ROL	**If** $i \neq 0$, $Dreg.i \rightarrow Dreg.(i+1)$ $Dreg.31 \rightarrow Dreg.0$ $Dreg.31 \rightarrow C$	Rotación de 1 bit hacia la izquierda.
ROLC	**If** $i \neq 0$, $Dreg.i \rightarrow Dreg.(i+1)$ $Dreg.31 \rightarrow C$ $C \rightarrow Dreg.0$	Rotación de 1 bit hacia la izquierda, a través del bit de acarreo.
ROR	**If** $i \neq 31$, $Dreg.i \rightarrow Dreg.(i-1)$ $Dreg.0 \rightarrow Dreg.31$ $Dreg.0 \rightarrow C$	Rotación de 1 bit hacia la derecha.

RORC	**If** $i \neq 31$, $Dreg.i \rightarrow Dreg.(i-1)$ $Dreg.0 \rightarrow C$ $C \rightarrow Dreg.31$	Rotación de 1 bit hacia la derecha, a través del bit de acarreo.
RPTB	$src \rightarrow RE$ $1 \rightarrow ST(RM)$ *Próximo* $PC \rightarrow RS$	Ejecuta varias veces (RC+1 veces) un bloque de instrucciones.
RPTS	$src \rightarrow RC$ $1 \rightarrow ST(RM)$ *Próximo* $PC \rightarrow RS$ *Próximo* $PC \rightarrow RE$	Ejecuta varias veces (RC+1 veces) una instrucción.
SIGI	*Maneja señales de Interbloqueo*	Activa *XF0*, $XF0 = 0$ (genera el interbloqueo). Espera hasta que no lea $XF1 = 0$ (espera hasta que se reconozca el interbloqueo). Termina poniendo $XF0 = 1$ (limpia el interbloqueo).
STF	$Rn \rightarrow Daddr$	Almacena valor flotante.
STFI	$Rn \rightarrow Daddr$, *interbloqueado*	Activa *XF0*, $XF0 = 1$ (genera el interbloqueo). Espera hasta que no lea $XF1 = 0$ (espera hasta que se reconozca el interbloqueo). Realiza la operación STF, poniendo $XF0 = 1$ (limpia el interbloqueo).
STI	$Sreg \rightarrow Daddr$	Almacena valor entero.
STII	$Sreg \rightarrow Daddr$, *interbloqueado*	Activa *XF0*, $XF0 = 0$ (genera el interbloqueo). Espera hasta que no lea $XF1 = 0$ (espera hasta que se reconozca el interbloqueo). Realiza la operación STI, poniendo $XF0 = 1$ (limpia el interbloqueo).
SUBB	$Dreg - src - C \rightarrow Dreg$	Operación de resta enteros con *borrow* de dos operandos.
SUBB3	$src1 - src2 - C \rightarrow Dreg$	Operación de resta enteros con *borrow* de tres operandos.
SUB*cond*	**if** $Dreg - src \geq 0$ $[(Dreg - src) << 1] \oplus 1 \rightarrow Dreg$ **else** $Dreg << 1 \rightarrow Dreg$	Operación de resta enteros condicionada.
SUBF	$Rn - src \rightarrow Rn$	Operación de resta flotantes, de dos operandos.
SUBF3	$src1 - src2 \rightarrow Rn$	Operación de resta flotantes, de tres operandos.
SUBI	$Dreg - src \rightarrow Dreg$	Operación de resta enteros, de dos operandos.
SUBI3	$src1 - src2 \rightarrow Dreg$	Operación de resta enteros, de tres operandos.
SUBRB	$src - Dreg - C \rightarrow Dreg$	Operación de resta de entero cambiado de signo con *borrow*.
SUBRF	$src - Rn \rightarrow Rn$	Operación de resta de flotante cambiado de signo.
SUBRI	$src - Dreg \rightarrow Dreg$	Operación de resta de entero cambiado de signo.
SWI	Emula la secuencia de eventos de las interrupciones	Emulación de las Interrupciones. El fabricante no dice como funciona esta instrucción y prohibe su uso.
TRAP*cond*	**if** *cond* es cierto o no existe $PC+1 \rightarrow *++SP$ (*Top Pila*) *Vector Trap N* $\rightarrow PC$ $0 \rightarrow ST(GIE)$ **else** $PC+1 \rightarrow PC$	Interrupción software de tipo TRAP. No necesita que se encuentre el bit GIE del registro ST habilitado para que se produzca la interrupción software.
TSTB	$Dreg \times src$	Comprueba un campo de bits, dos operandos.
TSTB3	$src1 \times src2$	Comprueba un campo de bits, tres operandos.
XOR	$Dreg \otimes src \rightarrow Dreg$	XOR lógico (dos operandos).
XOR3	$src1 \otimes src2 \rightarrow Dreg$	XOR lógico (tres operandos).

Instrucciones en paralelo		
Tipo de operando (modos de direccionamiento posibles al operando): Cualquier modo de direccionamiento a registro R0-R7 (operandos ***src1***, ***src3***, ***dst1***), direccionamiento a registro R0 ó R1 (operando ***op3***), direccionamiento a registro R2 ó R3 (operando ***op6***), modo de direccionamiento indirecto con desplazamiento 0, 1, IR0 ó IR1 (operandos ***src2***, ***src4***, ***dst2***), operandos ***op1***, ***op2***, ***op3*** y ***op4*** (dos de ellos deben accederse empleando direccionamiento a registro y los otros dos deben accederse empleando direccionamiento indirecto).		
ABSF || STF	$\lvert src2\rvert \rightarrow dst1$ $src3 \rightarrow dst2$	Calcula el valor absoluto de un número flotante. En paralelo, almacena valor flotante.
ABSI || STI	$\lvert src2\rvert \rightarrow dst1$ $src3 \rightarrow dst2$	Calcula el valor absoluto de un número entero. En paralelo, almacena valor entero.
ADDF3 || STF	$src1 + src2 \rightarrow dst1$ $src3 \rightarrow dst2$	Suma dos flotantes. En paralelo, almacena valor flotante.
ADDI3 || STI	$src1 + src2 \rightarrow dst1$ $src3 \rightarrow dst2$	Suma dos enteros. En paralelo, almacena valor entero.
AND3 || STI	$src1 \times src2 \rightarrow dst1$ $src3 \rightarrow dst2$	AND lógico bit a bit entre dos registros. En paralelo, almacena valor entero.
ASH3 || STI	**if** $count \geq 0 \Rightarrow$ $src2 << count \rightarrow dst1$ $src3 \rightarrow dst2$ **else** $src2 >> \lvert count\rvert \rightarrow dst1$ $src3 \rightarrow dst2$	Desplazamiento aritmético (conserva el signo) *count* veces, de tres operandos. Si *count*>0, desplazamiento a la izquierda, si *count*<0, a la derecha. En paralelo, almacena valor entero.
FIX || STI	$(\text{int})(src2) \rightarrow dst1$ $src3 \rightarrow dst2$	Convierte un valor entero en flotante. En paralelo, almacena valor entero.
FLOAT || STF	$(float)(src2) \rightarrow dst1$ $src3 \rightarrow dst2$	Convierte un valor flotante en entero. En paralelo, almacena valor flotante.
LDF || LDF	$src2 \rightarrow dst1$ $src4 \rightarrow dst2$	Carga dos flotantes.
LDF || STF	$src2 \rightarrow dst1$ $src3 \rightarrow dst2$	Carga un flotante. En paralelo almacena otro flotante.
LDI || LDI	$src2 \rightarrow dst1$ $src4 \rightarrow dst2$	Carga dos enteros.
LDI || STI	$src2 \rightarrow dst1$ $src3 \rightarrow dst2$	Carga un entero. En paralelo, almacena otro entero.
LSH3 || STI	**if** $count \geq 0 \Rightarrow$ $src2 << count \rightarrow dst1$ $src3 \rightarrow dst2$ **else** $src2 >> \lvert count\rvert \rightarrow dst1$ $src3 \rightarrow dst2$	Desplazamiento lógico (no conserva el signo) *count* veces, de tres operandos. Si *count*>0, desplazamiento a la izquierda, si *count*<0, a la derecha. En paralelo, almacena valor entero.
MPYF3 || ADDF3	$op1 \times op2 \rightarrow op3$ $op4 + op5 \rightarrow op6$	Multiplica dos flotantes. En paralelo suma otros dos flotantes.
MPYF3 || STF	$src1 \times src2 \rightarrow dst1$ $src3 \rightarrow dst2$	Multiplica dos flotantes. En paralelo almacena otro flotante.
MPYI3 || STI	$src1 \times src2 \rightarrow dst1$ $src3 \rightarrow dst2$	Multiplica dos enteros. En paralelo almacena otro entero.
MPYF3 || SUBF3	$op1 \times op2 \rightarrow op3$ $op4 - op5 \rightarrow op6$	Multiplica dos flotantes. En paralelo resta otros dos flotantes.
MPYI3 || ADDI3	$op1 \times op2 \rightarrow op3$ $op4 + op5 \rightarrow op6$	Multiplica dos enteros. En paralelo suma otros dos enteros.
MPYI3 || SUBI3	$op1 \times op2 \rightarrow op3$	Multiplica dos enteros.

	$op4 - op5 \rightarrow op6$	En paralelo resta otros dos enteros.
NEGF || STF	$-src2 \rightarrow dst1$ $src3 \rightarrow dst2$	Cambia de signo un flotante. En paralelo, almacena otro flotante.
NEGI || STI	$-src2 \rightarrow dst1$ $src3 \rightarrow dst2$	Cambia de signo un entero. En paralelo, almacena otro entero.
NOT || STI	$\overline{src1} \rightarrow dst1$ $src3 \rightarrow dst2$	Complemento a uno de un registro. En paralelo, almacena un entero.
OR3 || STI	$src1 + src2 \rightarrow dst1$ $src3 \rightarrow dst2$	OR bit a bit de dos registros. En paralelo, almacena un entero.
STF || STF	$src1 \rightarrow dst1$ $src3 \rightarrow dst2$	Almacena dos flotantes.
STI || STI	$src1 \rightarrow dst1$ $src3 \rightarrow dst2$	Almacena dos enteros.
SUBF3 || STF	$src1 - src2 \rightarrow dst1$ $src3 \rightarrow dst2$	Resta dos flotantes. En paralelo almacena otro flotante.
SUBI3 || STI	$src1 \otimes src2 \rightarrow dst1$ $src3 \rightarrow dst2$	Resta dos enteros. En paralelo almacena otro entero.
XOR3 || STI	$src1 - src2 \rightarrow dst1$ $src3 \rightarrow dst2$	XOR bit a bit de dos registros. En paralelo almacena otro entero.

Tabla 3-12. Sumario de Instrucciones, DSP TMS320C3x.

cond	**Descripción**
Compara incondicionalmente	
U	Sin condición.
Compara sin signo	
LO	Si es más pequeño que (C=1).
LS	Si es más pequeño o igual que (C=1 ó Z=1).
HI	Si es más grande que (C=0 y Z=0).
HS	Si es más grande o igual que (C=0).
EQ	Si es igual que (Z=1).
NE	Si no es igual que (Z=0).
Compara con signo	
LT	Si es menor que (N=1).
LE	Si es menor o igual que (N=1 ó Z=1).
GT	Si es mayor que (N=0 y Z=0).
GE	Si es mayor o igual que (N=0).
EQ	Si es igual que (Z=1).
NE	Si no es igual que (Z=0).
Compara con cero	
Z	Si es cero (Z=1).
NZ	Si no es cero (Z=0).
P	Si es positivo (N=0 y Z=0).
N	Si es negativo (N=1).
NN	Si no es negativo (N=0).
Compara con otros códigos de condición del registro ST de la CPU	
NV	Si no se ha producido overflow (V=0).
V	Si se ha producido overflow (V=1).
NUF	Si no se ha producido underflow (UF=0).
UF	Si se ha producido underflow (UF=1).
NC	Si no se ha producido el bit de acarreo (C=0).
C	Si se ha producido acarreo (C=1).
NLV	Si no se ha producido nunca overflow (LV=0).
LV	Si se ha producido alguna vez overflow (LV=1).

NLUF	Si no se ha producido nunca underflow en las operaciones con flotantes (LUF=0).
LUF	Si se ha producido alguna vez underflow en las operaciones con flotantes (LUF=1).
ZUF	Si es cero o es underflow de flotantes (Z=1 ó UF=1).

Tabla 3-13. Códigos de condición, TMS320C3x.