

CAPITULO 2: Introducción al DSP

TMS320C30

2.1. INTRODUCCIÓN AL DSP TMS320C30

La estructura interna de la familia TMS320C3x incorpora una arquitectura microprocesadora clásica de tipo Harvard (ofrece buses de datos y direcciones independientes que permiten indistintamente el acceso a la memoria de programa y datos). La estructura de la CPU de este DSP, le permite, en principio, realizar, simultáneamente y en un único ciclo de máquina, operaciones de multiplicación y suma con enteros o flotantes de coma variable. Los accesos al exterior se realizan, sin embargo, utilizando una arquitectura microprocesadora de tipo Von-Neuman (selección y acceso a los datos e instrucciones por los mismos buses de dirección y datos).

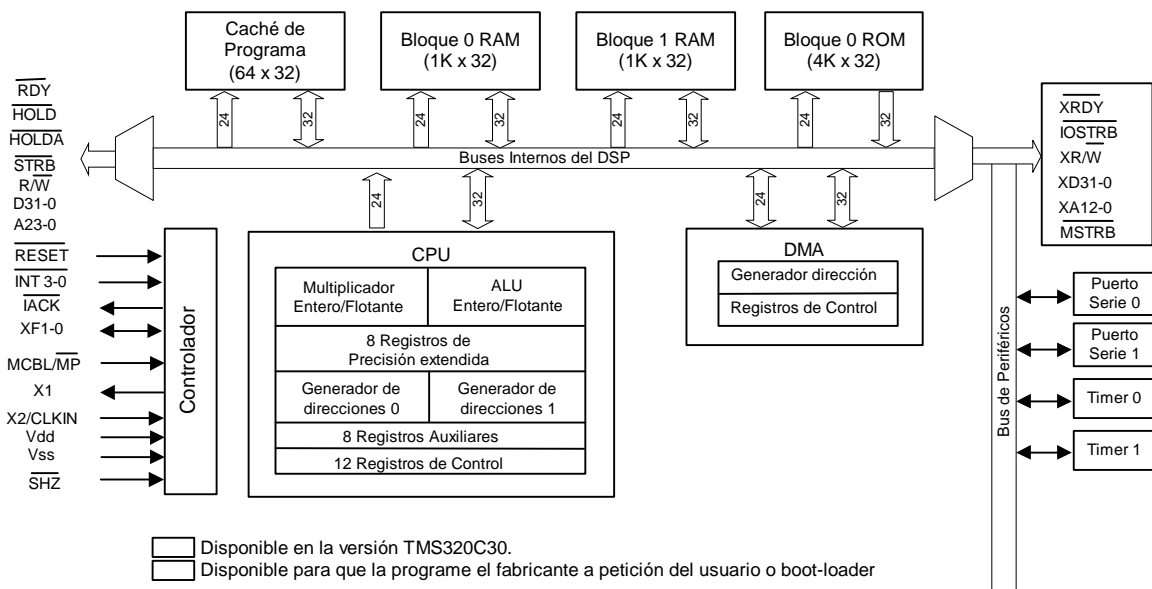


Figura 2.1 Diagrama de bloques de la estructura interna del DSP TMS320C3x

La CPU dispone de una serie de registros internos, R0-R7, que hacen las veces de registros acumuladores de propósito general. Su función es la de ser el operando de las diferentes operaciones aritméticas que puede realizar la CPU. Junto con estos registros

aparecen otros en la CPU, los denominados ARi, que se emplean como operandos de unas unidades aritméticas auxiliares.

El bus de datos de este sistema microprocesador es de 32 bits, siendo el bus de direcciones de 24 bits, lo que permite una capacidad de direccionamiento total del dispositivo de 16 *Mwords* (obsérvese que la *word* para este sistema es de 32 bits).

Se observa, además, que esta familia de DSP dispone de una serie de periféricos internos:

- Memoria interna: Memoria RAM (dos bloques de 1Kw cada uno) y ROM (4Kw de tipo OTP-programable una única vez por el fabricante). El tiempo de acceso a estos periféricos es de un ciclo de máquina.
- Periférico de memoria caché de programa: El tiempo de acceso a estos periféricos es de un ciclo de máquina. El usuario lo emplea por ejemplo, para disminuir los accesos a instrucciones ubicadas en memoria externa, con tiempos de acceso más elevados de los que tienen las instrucciones que se ubican en memoria interna (de tipo RAM o ROM, de 64K palabras de 32 bits).
- Periférico DMA interno.
- Puertos serie con protocolo síncrono: Dos puertos serie disponibles en el TMS320C30, únicamente uno en el TMS320C31 y TMS320C32.
- Dos periféricos temporizadores programables de 32 bits (*Timers*).
- Periférico de control y gestión de interrupciones que hace este proceso transparente a la CPU.
- Periféricos de control de acceso a memoria. Se encargan de gestionar los accesos a posiciones de memoria ubicadas en el interior o exterior del integrado. El acceso al exterior se realiza mediante dos buses independientes entre sí (para el caso del TMS320C30), el bus principal y el de expansión, cada uno formado por un bus de datos, otro de direcciones y, finalmente, un bus de control. Podemos, por tanto, controlar el acceso al exterior del dispositivo por dos canales independientes entre sí, cada uno con su protocolo, sus tiempos característicos, etc. Las versiones TMS320C31 y TMS320C32 únicamente disponen del bus principal para gestionar los accesos al exterior.

El DSP, mediante una unidad multiacumuladora (*MAC*), puede ir preparando la próxima instrucción a ejecutar mientras procesa (opera con) dos o más datos, en el mismo ciclo de reloj (estructura interna de tipo Harvard).

Uno de los componentes básicos de la familia TMS320 es el TMS320C30 con versiones que funcionan a diferentes frecuencias máximas de reloj externo (oscilador o cristal de cuarzo): 40 y 50 MHz. Por ejemplo, el de 40 MHz de frecuencia máxima tendría, funcionando con un cristal de cuarzo de dicha frecuencia conectado al DSP, 50ns de tiempo de ejecución de una instrucción (el doble del periodo del oscilador) y la posibilidad de realizar hasta 60 MFLOPS (millones de operaciones de coma flotante por segundo). En adelante, se analizará con detenimiento las características del TMS320C30.

2.2. ARQUITECTURA

La arquitectura del sistema responde a las necesidades de los algoritmos para optimizar las soluciones software y hardware. Para ello se ha optimizado el rendimiento, la precisión y el rango dinámico de la unidad del punto flotante, además de la capacidad de memoria, el grado de paralelismo y el controlador DMA que incorpora. En la figura 2.2 se muestra el diagrama de bloques del TMS320C30.

2.2.1. LA UNIDAD CENTRAL DE PROCESO O CPU

La arquitectura de la CPU está basada en registros y consta de los siguientes componentes:

- Multiplicador de números en punto flotante o enteros
- Unidad aritmético-lógica. (ALU)
- Desplazador de barril de 32 bits.
- Buses internos.
- Unidad aritmética de registros auxiliares. (ARAU)
- Lista de registros de la CPU.

2.2.1.1 Multiplicador de números en formato punto flotante o entero

El multiplicador realiza multiplicaciones con enteros de 24 bits y con números en punto flotante de 32 bits. La implementación de la aritmética de punto flotante permite operaciones de punto fijo y punto flotante a la velocidad de 33ns por ciclo de instrucción. Para conseguir un mayor rendimiento se pueden usar instrucciones paralelas con el objeto de realizar una multiplicación y una operación de la ALU en un solo ciclo.

Cuando el multiplicador realiza multiplicaciones en punto flotante, su entrada son números de 32 bits, y la salida es de 40 bits, mientras que cuando se trata de multiplicaciones de enteros, la entrada son números de 24 bits, y la salida es siempre un entero de 32 bits.

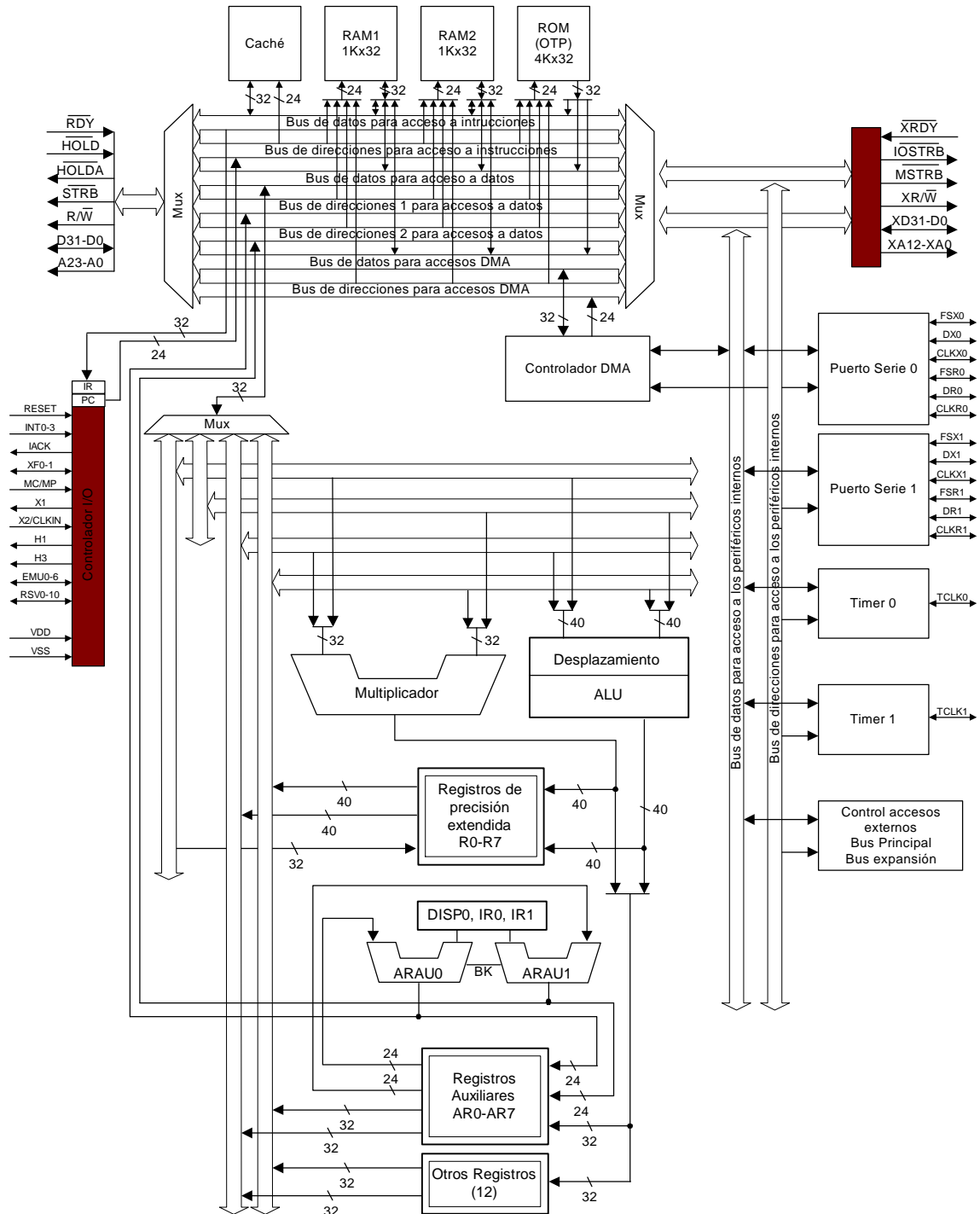


Figura 2.2. Diagrama de bloques del DSP TMS320C30

2.2.1.2 Unidad aritmético-lógica y buses internos

La ALU realiza en un solo ciclo operaciones con enteros de 32 bits, en punto flotante de 40 bits y operaciones lógicas de 32 bits. La ALU posee un desplazador de barril que se usa para desplazar hasta 32 bits a la izquierda o a la derecha el contenido de un registro en un solo ciclo de reloj.

Existen cuatro buses internos (CPU1, CPU2, REG1 y REG2) por los que circulan los dos operandos de memoria y los dos de los registros. De esta forma, se permite en un solo ciclo la multiplicación en paralelo y una suma o resta sobre cuatro operandos en formato entero o punto flotante.

2.2.1.3. Unidad aritmética de registros auxiliares

Las dos unidades aritméticas de registros auxiliares (ARAU0 y ARAU1) pueden generar dos direcciones de memoria en un solo ciclo de reloj. La ARAU opera en paralelo con el multiplicador y la ALU, soportando direccionamiento con desplazamiento, registro de índice (IR0 y IR1) simple, circular y bit invertido.

2.2.2. CONJUNTO DE REGISTROS PRIMARIOS DE LA CPU

Existen 28 registros primarios agrupados en la CPU:

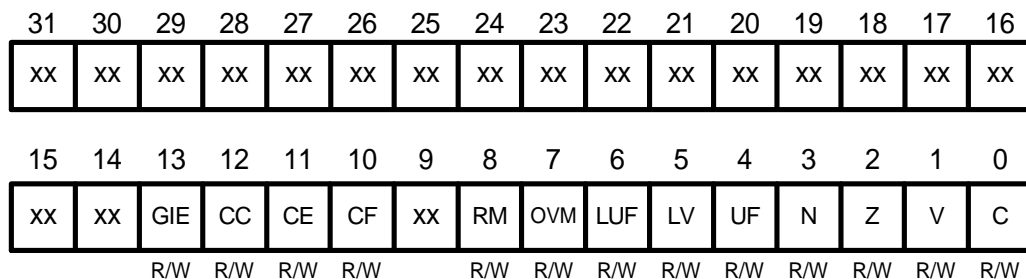
- Registros de precisión extendida (R0-R7): Son registros de propósito general, empleados por el multiplicador y la ALU de la CPU. Su tamaño es de 40 bits (obsérvese que el bus de datos del sistema es de 32 bits, de ahí el nombre de registro de precisión extendida, con 40 bits se obtiene mayor precisión que con 32 bits). Estos registros son los únicos, dentro de la CPU, que pueden ser empleados como operandos flotantes (si un usuario desea trabajar con números flotantes debe almacenar dichos números en estos registros antes de operar con ellos). Si se opera con alguno de estos registros en modo flotante se usarían los 40 bits del mismo. Si las operaciones a realizar son de enteros se usan los bits menos significativos (del 0 al 31), el resto (bits 32 al 39) permanecen inalterados (no se utilizan al operar con enteros). Figura 2.3.



Figura 2.3. Formato en punto flotante y entero (con o sin signo), de los registros de precisión extendida (R0-R7)

- Registros auxiliares (AR0-AR7): Pueden ser modificados por las ARAU, la ALU y el multiplicador. Son registros de propósito general y de 32 bits que pueden emplearse como operandos de tipo entero en operaciones aritméticas o como generadores de direcciones de 24 bits.
- Puntero a la página de datos (DP): Es un registro de 32 bits de los cuales sólo los 8 bits menos significativos son utilizados. El registro se usa, para acceder a operandos utilizando el modo de direccionamiento directo, como selector o puntero selector a una página de datos (definida con los 8 bits menos significativos del registro) a la que se está accediendo en un instante dado. Cada página ocupa 64K palabras de 32 bits de memoria existiendo un total de 256 páginas (la memoria tiene una capacidad de direccionamiento total de 16M palabras).
- Registros de índice (IR0 e IR1): Son registros de 32 bits empleados por las ARAU para calcular las direcciones de los datos en modo de direccionamiento indexado.
- Registro de dimensión de bloque (BK): Es un registro de 32 bits utilizado por las ARAU para fijar la dimensión del bloque de datos en el modo de direccionamiento denominado circular (se verá en el tema siguiente). Está especialmente diseñado para implementar colas mediante la técnica de matriz circular, donde BK determina el tamaño de la matriz. La implementación de colas circulares gestionadas automáticamente por hardware (de forma casi transparente al usuario) por el DSP es una de las características que ofrecen este tipo de microprocesadores, pensados para realizar, en tiempo real, algoritmos de tipo repetitivos (FFT, filtros digitales...)

- **Puntero a la pila (SP):** Como en cualquier microprocesador, este DSP dispone de una pila. El registro SP es un registro de 32 bits que contiene la dirección de la cima de la pila (apunta al último elemento introducido en la pila). Este registro es alterado por interrupciones, excepciones de tipo software, llamadas y retornos de subrutinas e instrucciones de tipo PUSH y POP (almacena datos en la pila o recoge datos de la pila).
- **Registro de estado (ST):** Contiene información global sobre el estado de la CPU en cada instante. Su estructura se muestra en la figura 2.4. En la tabla 2.1 se definen cada uno de los bits de este registro. En este registro aparecen 8 bits, del bit 0 o bit menos significativo (de aquí en adelante LSB, *Least Significant Bit*) al bit 7, activados por hardware por la parte de manejo de datos de la CPU. Cada vez que se realiza alguna operación aritmético-lógica, en la que intervengan la ALU o el multiplicador, se activa o no alguno de estos bits. El resto son bits de control del periférico caché (bits CF, CE y CC), del modo de repetición de instrucciones (bit RM) y para la habilitación global de las interrupciones (bit GIE).



xx = Bit Reservado

R = Bit accesible en lectura

W = Bit accesible en escritura

Figura 2.4. Registro de Estado (ST)

A continuación se hará una descripción de los bits del registro de estado:

Bit	Nombre	Función que realiza
0	C	<u>Bandera de acarreo (<i>Carry flag</i>):</u> a) Cuando se realiza una operación aritmética de suma con manejo de enteros, C=1 si se produce acarreo en el bit más significativo (de aquí en adelante MSB, <i>Most Singnificant Bit</i>) del resultado. Si se trata de una resta,

		<p>C=1 si es necesario introducir un bit en el MSB del resultado.</p> <p>b) Cuando se realiza una operación con flotantes, el acarreo no se ve afectado.</p> <p>c) Cuando se realizan operaciones de desplazamiento sobre un registro, el bit de carry se actualiza con el valor desplazado fuera del registro.</p>
1	V	<u>Bandera de desbordamiento (<i>Overflow flag</i>)</u> : Si se produce desbordamiento, V=1. El desbordamiento de entero implica que el resultado de la operación de enteros es menor que o mayor que. En el caso de flotante, se produce desbordamiento cuando el resultado de una operación de flotantes tiene un exponente mayor que 127.
2	Z	<u>Bandera de cero (<i>Zero flag</i>)</u> : Si el resultado de una operación es cero, Z=1.
3	N	<u>Bandera de negativo (<i>Negative flag</i>)</u> : Si el resultado de una operación es negativo, N=1.
4	UF	<u>Bandera de desbordamiento flotante por debajo (<i>Underflow flag</i>)</u> : Sólo para operaciones con flotantes. Se produce desbordamiento por debajo cuando el resultado de una operación de flotantes tiene un exponente menor o igual que -128 y en ese caso, UF=1.
5	LV	<u>Bandera de desbordamiento latcheado (<i>Latched overflow flag</i>)</u> . Si se ha producido alguna vez un desbordamiento de tipo V, LV=1. Sólo puede borrarlo el usuario escribiendo un 0.
6	LUF	<u>Bandera de desbordamiento por debajo fijo (<i>Latched floating-point underflow flag</i>)</u> . Si se ha producido alguna vez un desbordamiento de tipo UF, LUF=1. Sólo puede borrarlo el usuario escribiendo un 0.

7	OVM	<u>Bandera de modo de desbordamiento (<i>Overflow Mode flag</i>)</u> . Esta bandera afecta exclusivamente a las operaciones enteras. Si OVM=0, el entero que resulte del desbordamiento no es tratado de manera especial. Si es OVM=1: <ul style="list-style-type: none"> a) Los enteros que desborden en dirección positiva se saturan al mayor número positivo en complemento a dos (7FFFFFFh). b) Los enteros que desborden en dirección negativa se saturan al número más negativo en complemento a dos (80000000h). Nota: V y LV son independientes del valor de OVM.
8	RM	<u>Bandera de modo de repetición (<i>Repeat Mode Flag</i>)</u> . Si RM=1, el contador de programa se está modificando en los modos de repetición, tanto de bloque como simple.
9	Reservado	Se lee un simple cero
10	CF	<u>Fija la Caché (<i>Caché Freeze</i>)</u> . Cuando CF=1, se impide su modificación, esté (se permite el acceso a recoger instrucciones pero no que se actualice) o no habilitada. Puede hacerse un Caché Clear si CF=1.
11	CE	<u>Habilitación de Caché (<i>Cache Enable</i>)</u> .Habilita la Caché cuando CE=1. Se puede realizar un Caché Clear si CE=0.
12	CC	<u>Limpieza de la Caché (<i>Cache Clear</i>)</u> . Limpia la Caché. Si CC=1, se reinicializa la Caché, aunque ésta se encuentre congelada (CF=1).
13	GIE	<u>Habilitación global de interrupciones (<i>Global Interrupt Enable</i>)</u> .
14-15	Reservado	Se leen siempre ceros.
16-31	Reservado	Valores indefinidos.

Tabla 2.1. Funciones de los bits del registro de estado

- Registro de habilitación de interrupciones de CPU y de programación del sincronismo de la DMA (IE): Cuando un bit del registro se pone a cero, se inhabilita la interrupción de CPU asociada a él o se impide la sincronización con dicha interrupción de los eventos (escritura y/o lectura) del periférico interno DMA. Obsérvese que todas las **interrupciones** de tipo hardware que tiene disponibles el usuario en este microprocesador son **enmascarables**. (Fig 2.4)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
xx	xx	xx	xx	xx	EDINT (DMA)	ETINT1 (DMA)	ETINT0 (DMA)	ERINT1 (DMA)	EXINT1 (DMA)	ERINT0 (DMA)	EXINT0 (DMA)	EINT3 (DMA)	EINT2 (DMA)	EINT1 (DMA)	EINT0 (DMA)
					R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
xx	xx	xx	xx	xx	EDINT (CPU)	ETINT1 (CPU)	ETINT0 (CPU)	ERINT1 (CPU)	EXINT1 (CPU)	ERINT0 (CPU)	EXINT0 (CPU)	EINT3 (CPU)	EINT2 (CPU)	EINT1 (CPU)	EINT0 (CPU)
					R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

xx = Bit Reservado
R = Bit accesible en lectura
W = Bit accesible en escritura

Figura 2.5. Registro de habilitación de interrupciones (IE)

- Registro de las banderas de petición de las interrupciones (IF): Cuando alguno de sus bits se pone a uno, es porque se ha producido la interrupción correspondiente. Mientras la CPU no atienda la rutina de servicio de dicha interrupción, la bandera asociada a la misma en el registro IF permanece activa. En cuanto se acceda a la rutina de servicio, la bandera correspondiente se pone a cero, indicando que no se encuentra pendiente la ejecución de la rutina de servicio de la interrupción.

(Fig. 2.6.)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
xx	xx	xx	xx	xx	DINT	TINT1	TINT0	RINT1	XINT1	RINT0	XINT0	INT3	INT2	INT1	INT0
					R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

xx = Bit Reservado
R = Bit accesible en lectura
W = Bit accesible en escritura

Figura 2.6. Registro de petición de servicio de rutinas de interrupción (IF)

- Registro de control de las entradas y salidas de propósito general (IOF): Controla los pines XF0 y XF1 del DSP (obsérvese que el TMS320C30, en principio, tan sólo

dispone de dos líneas digitales de entrada y/o salida de propósito general). El usuario utiliza este registro para indicar la función de los pines XF0 y XF1 (si son líneas de entrada o salida al ASIC) y para leer el valor digital asociado a las dos líneas o escribir en ellas.

2.2.3. MEMORIA

El espacio total de memoria que es capaz de direccionar la familia TMS320C3x de Texas Instruments, es de 16M (unos 16 millones) de palabras de 32 bits. Dentro de este espacio total que es capaz de direccionar el DSP se incluyen las zonas de memoria internas de las que dispone el sistema (memoria RAM interna, ROM interna y registros de configuración de periféricos internos). En la figura 2.7, se muestran los buses y la estructura de acceso a memoria del sistema.

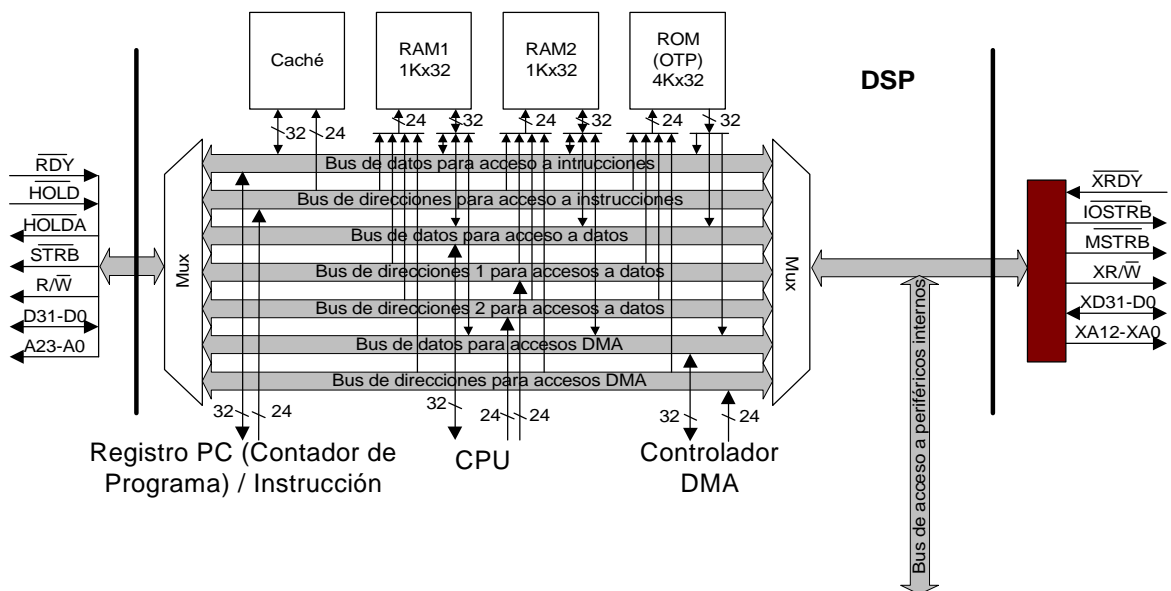


Figura 2.7. Acceso a memoria: Buses del sistema.

El sistema dispone de dos conjuntos de líneas para acceso al exterior, el denominado **bus principal** (formado por un bus de datos de 32 bits, un bus de direcciones de 24 bits y un bus de control de 5 bits) y el **bus de expansión** (formado por un bus de datos de 32 bits, un bus de direcciones de 13 bits y un bus de control de 4 bits). En la figura 2.8 se muestran los mapas de memoria del TMS320C30 y del TMS320C31. Se observa que cada DSP tiene dos posibles mapas de memoria. La diferencia estriba en la posibilidad de acceso o no a la

ROM interna del sistema. La forma de seleccionar uno u otro mapa de memoria es imponiendo un nivel alto o bajo en el pin de entrada a DSP.

Obsérvese que el TMS320C31 no dispone de bus de expansión, en el mapa de memoria no hay zonas asignadas al bus de expansión, y que la ROM ya viene programada de fábrica conteniendo un programa que se conoce con la denominación de *Boot-Loader*.

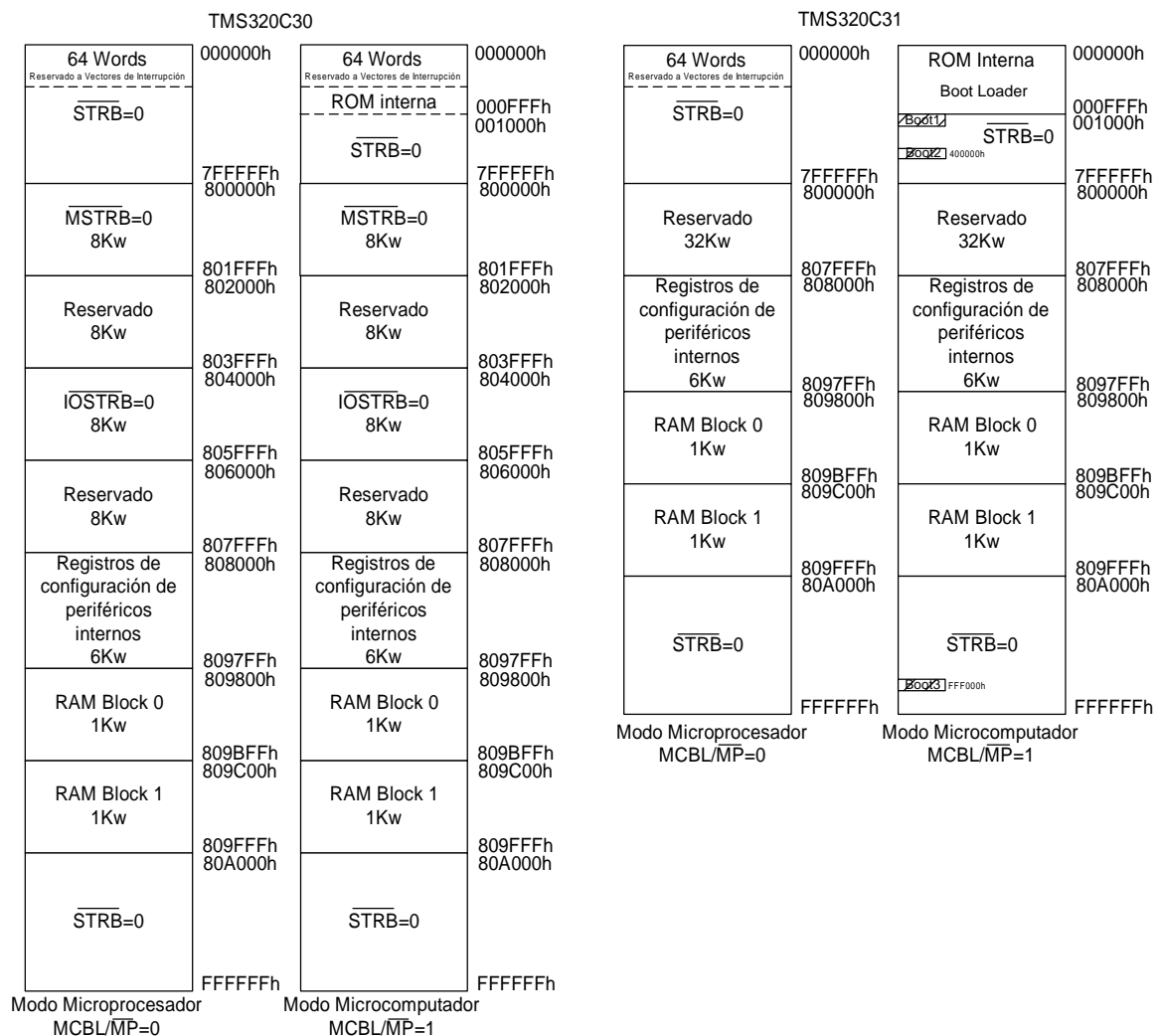


Figura 2.8. Mapas de memoria de la familia TMS320C3x. Obsérvese que la única diferencia entre el modo microcomputador y microprocesador es la habilitación o inhabilitación, en la zona baja del mapa de memoria, del periférico interno de memoria no volátil (ROM).

Como conclusión, destacar que el DSP posee dos modos de operación:

1. Microprocesador: El sistema arranca autovectorizando desde la posición 000000H, que debe coincidir en este caso con dispositivos externos de memoria dispuestos (rutados y

colocados en el mapa de memoria del sistema microprocesador) y programados por el usuario.

2. Microcomputador: El sistema arranca autovectorizando desde la posición 000000H que en este caso coincide con la ROM interna del sistema.

En la figura 2.9 se detalla la zona del mapa de memoria en la que se ubican los distintos periféricos internos que tienen los diferentes miembros de la familia TMS320C3x.

808000h	Registros del Controlador de DMA (16 Words)
808010h	Reservado (16 Words)
808020h	Registros del Timer0 (16 Words)
808030h	Registros del Timer1 (16 Words)
808040h	Registros del Puerto Serie 0 (16 Words)
808050h	Registros del Puerto Serie 1 (16 Words)
808060h	Reservado-TMS320C31 Registros de control del Bus Principal y de Expansión (16 Words)
808070h	Reservado
8097FFh	

Figura 2.9. Mapas de memoria de los registros de control de los periféricos internos de los DSP de la familia TMS320C3x

2.2.4. VECTORES DE RESET, INTERRUPCIONES HARDWARE Y SOFTWARE

Independientemente del modo de funcionamiento del DSP, la secuencia de operaciones siguientes a un reset hardware del sistema (aplicación de un 0 lógico en la entrada *Reset* del circuito integrado, reset externo no-enmascarable) es siempre la misma. El sistema recoge el vector de reset (posición 000000h en el mapa de memoria del DSP) y copia el valor que existe en ese registro (que se ubica en un periférico de memoria externa

suponiendo que el sistema funciona en modo microprocesador o en la ROM interna, si el sistema funciona en modo microcomputador) en el registro de contador de programa o PC de la CPU. Se dice por tanto que el sistema tiene el reset autovectorizado a la posición absoluta 000000h del mapa de memoria del DSP, dado que cuando aparece el reset, el sistema accede a la posición 000000h para recoger la dirección absoluta de la primera instrucción a ejecutar. Al conectar la alimentación al DSP, el estado de los registros es desconocido. La función fundamental del reset, es llevar los registros del sistema a un estado inicial conocido, para lo cual es preciso mantener a nivel bajo (0 lógico) el pin de entrada *RESET* al circuito integrado durante al menos 10 ciclos de reloj del sistema. Un reset genera la siguiente reacción de los registros y líneas de entrada y salida del circuito integrado:

- Accesos al exterior: Las líneas de los buses de dirección y de datos se ponen en alta impedancia, así como las líneas de control del tipo de acceso al exterior (*R/W* Y *XR/R*). Las líneas *STRB*, *MSTRB* y *IOSTRB* se ponen a 1. Los registros de control asociados al bus principal y de expansión se inicializan.
- Las líneas de los periféricos internos (temporizadores y puertos serie) así como las entradas o salidas digitales, *XF0* Y *XF1* se ponen en alta impedancia. Los periféricos internos al DSP y sus registros de control se inicializan.
- Los registros internos de la CPU se inicializan: *IE*=0, *ST*=0, *IF*=0, *IOF*=0, etc.

Al hablar de interrupciones en un microprocesador se distingue entre fuente que genera la interrupción y servicio que provoca:

- Fuentes de interrupción. La forma de atención de las interrupciones es análoga al proceso de reset. Los DSP de la familia TMS320C3x disponen de cuatro fuentes de interrupciones externas (*INT0*, *INT3*), dos fuentes de interrupción asociadas a los periféricos internos puertos serie síncronos, una fuente asignada a la transmisión y otra a la recepción (*XINT0*, *RINT0* para el puerto serie 0 y *XINT1*, *RINT1* para el puerto serie 1), así como una fuente de interrupción asociada a cada uno de los periféricos internos restantes, temporizadores y controlador de DMA (*TINT0*, *TINT1* y *DINT*).

- Servicio de interrupciones. Cuando una interrupción es detectada, el bit correspondiente a dicha interrupción en el registro IF se pone a 1.
 - Si la interrupción tiene habilitada la generación de eventos CPU (el bit correspondiente del registro IE está a 1 y además el bit de habilitación global de interrupciones está también a 1) la instrucción que, cuando se detecta la interrupción, está siendo recogida no se ejecuta. La dirección de esta instrucción, registro PC, es almacenada en la dirección apuntada por el registro SP, puntero a la cima de la pila. Inmediatamente después de almacenar en la pila el PC, se pone a cero el bit GIE del registro ST (se desactiva el bit de habilitación global de las interrupciones) y se recoge el vector de interrupción asociado a la interrupción detectada que se almacena en PC. Se empieza, por tanto, a ejecutar la rutina de servicio de la interrupción. Al final de la rutina de servicio de la interrupción deberá realizarse el retorno del contador de programa a la dirección de la instrucción que no se ejecutó. El retorno normal de la rutina de servicio lo realiza la instrucción RETI, que almacena en el registro PC el dato que aparece en la cima de la pila (deberá coincidir con el valor del registro PC que se almacenó en la pila al comenzar el proceso de atención de la interrupción) y que reactiva el bit GIE.
 - Si la interrupción tiene habilitada la generación de eventos DMA, está a 1 el bit correspondiente del registro IE y además se permite la sincronización de eventos DMA por interrupciones, sirve para lanzar las transferencias de tipo DMA programadas.

Obsérvese, por tanto, que una interrupción puede generar un evento de tipo CPU (ruptura de la secuencia normal de ejecución en un programa) y otro de DMA. Ambos procesos son totalmente independientes y se ejecutan en paralelo.

Comentarios relacionados con el proceso de atención de interrupciones en la familia TMS320C3x:

1. El flag GIE (del registro ST de la CPU) se pone a 0 al comenzar el proceso de atención de una interrupción. Por tanto, y en principio, no se produce anidamiento entre interrupciones. No es posible que, mientras se esté ejecutando la rutina de servicio de una interrupción, al producirse una nueva interrupción, ésta última se atienda antes de

terminar de ejecutarse la rutina de servicio de la primera interrupción que apareció. Al terminar de ejecutar la rutina de servicio de la interrupción habrá que devolver el registro PC al valor que tenía antes de la llegada de la interrupción y, si se desea permitir la generación de nuevas interrupciones habrá que volver a activar el bit GIE (instrucciones RETI o RETS). Obsérvese que, dentro de la propia rutina de servicio de las interrupciones, el usuario puede poner, vía software, el bit GIE a 1 permitiendo el anidamiento de las interrupciones.

2. Los accesos en escritura al registro IF pueden causar fallos en la generación de las interrupciones (si se ha generado una interrupción y está lista para ser latcheada en IF a la par que se escribe en el mismo registro IF, el proceso de escritura tiene más prioridad, perdiéndose la interrupción generada, lo que es especialmente grave si la interrupción generada lo fue por un periférico interno como la DMA).

3. La señal que genere una interrupción externa debe tener un 0 lógico (permanecer activa) un tiempo comprendido de entre 2 y 4 ciclos de reloj del sistema para garantizar que no se produzca más de una interrupción externa en el sistema (el flags del registro IF correspondiente a la interrupción externa lo limpia automáticamente la CPU generando una señal de reset asíncrono al entrar en la rutina de servicio de la interrupción).

4. La atención de las interrupciones provoca que no se ejecute la instrucción que entre en el momento de la aparición de la interrupción en su fase de recogida (Fetch en la estructura de pipeline) siempre que ésta se encuentre en su primer ciclo máquina de recogida. Si la instrucción está en su fase de recogida pero en un ciclo máquina superior al primero (debido, por ejemplo, a la aparición de conflictos en la estructura *pipeline*) se termina de ejecutar antes de atenderse la rutina de servicio de la interrupción. Todo esto lo gestiona la CPU de forma transparente al usuario.

5. Posibilidad de generar interrupciones de tipo software (excepciones). Aparte de las interrupciones antes mencionadas. Se pueden generar hasta 32 interrupciones software empleando la instrucción “TRAPcond n”, con $0 < n < 31$, que generaría una ruptura programada de la secuencia normal de control de forma análoga a como sucede en el caso de interrupciones de tipo hardware. Este tipo de interrupciones se generan independientemente del valor (0 ó 1) de GIE.

En la figura 2.10 se muestra el mapa de memoria de los vectores de reset, interrupciones y excepciones software en la familia TMS320C3x de Texas Instruments.

00h	Reset
01h	INT0
02h	INT1
03h	INT2
04h	INT3
05h	XINT0
06h	RINT0
07h	XINT1 (Reservado TMS320C31)
08h	RINT1 (Reservado TMS320C31)
09h	TINT0
0Ah	TINT1
0Bh	DINT
0Ch-1Fh	Reservado
20h	TRAP0
21h	TRAP1
22h	⋮
3Ah	⋮
3Bh	TRAP27
3Ch	TRAP28 (Reservado)
3Dh	TRAP29 (Reservado)
3Eh	TRAP30 (Reservado)
3Fh	TRAP31 (Reservado)

Figura 2.10. Mapas de memoria de los vectores de reset, interrupciones y excepciones de software.

2.2.5. FORMATO DE DATOS

Los datos en el TMS320C30 están organizados en tres tipos fundamentales: enteros, enteros sin signo y punto flotante.

2.2.5.1. Números enteros

El C30 soporta dos tipos de formatos enteros: el formato entero corto (16 bits) y otro de precisión normal (32 bits).

- Para enteros con signo:
 - El formato corto de 16 bits utiliza un formato en complemento a dos para los operandos inmediatos enteros. El rango está comprendido entre -2^{15} y $2^{15}-1$.
 - En el formato de precisión normal el número se representa mediante 32 bits en complemento a dos. El rango para este formato está comprendido entre -2^{31} y $2^{31}-1$.
- Para enteros sin signo:

- El formato corto de 16 bits para operandos enteros sin signo. El rango está comprendido entre 0 y 2^{16} .
- En el formato de precisión normal, el número está representado como un valor de 32 bits. El rango para este formato está comprendido entre 0 y 2^{32} .

2.2.5.2. Números enteros sin signo

El C30 soporta dos formatos de enteros sin signo: uno corto (16 bits) y otro de precisión normal (32 bits).

El corto de 16 bits es utilizado para operandos enteros sin signo inmediato. El rango de este formato está comprendido entre 0 y 2^{16} .

En el formato de precisión normal el número está representado mediante un valor de 32 bits, y su rango está comprendido entre 0 y 2^{32} .

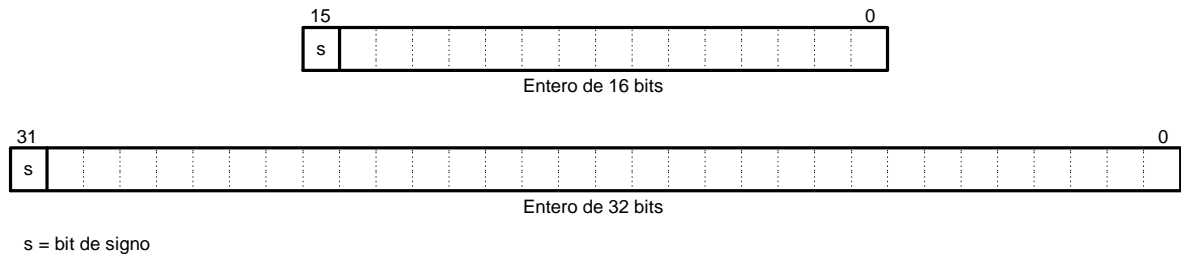


Figura 2.11. Representación de datos enteros en el DSP TMS320C30

2.2.5.3. Números en punto flotante

El C30 soporta tres formatos en punto flotante:

- Corto, para operandos en punto flotante inmediatos, de 16 bits.
- Precisión normal, de 32 bits.
- Precisión extendida, de 40 bits.

Todos los formatos en punto flotante constan de tres partes: un campo de exponente e , un campo de 1 bit de signo s y un campo fracción f . Los campos signo y fracción pueden ser considerados como una unidad llamada mantisa (*man*).

La ecuación general para calcular el valor del número en punto flotante es:

$$x = s * \bar{s} . f_2 * 2^e$$

Ecuación 2.1

En la ecuación 2.1, s es el valor del bit de signo, \bar{s} es el inverso del valor del bit de signo, f es el valor binario del campo fracción y e es el equivalente decimal del campo correspondiente.

En el formato corto, el campo exponente tiene 4 bits y está representado en complemento a dos. La mantisa, tiene 12 bits, de los cuales el más significativo es el bit de signo.

En el formato de precisión normal, el número se representa mediante 8 bits de exponente y 24 bits de mantisa en complemento a dos.

En el formato de precisión extendida, los números en punto flotante están representados por 8 bits de exponente y 32 de mantisa.

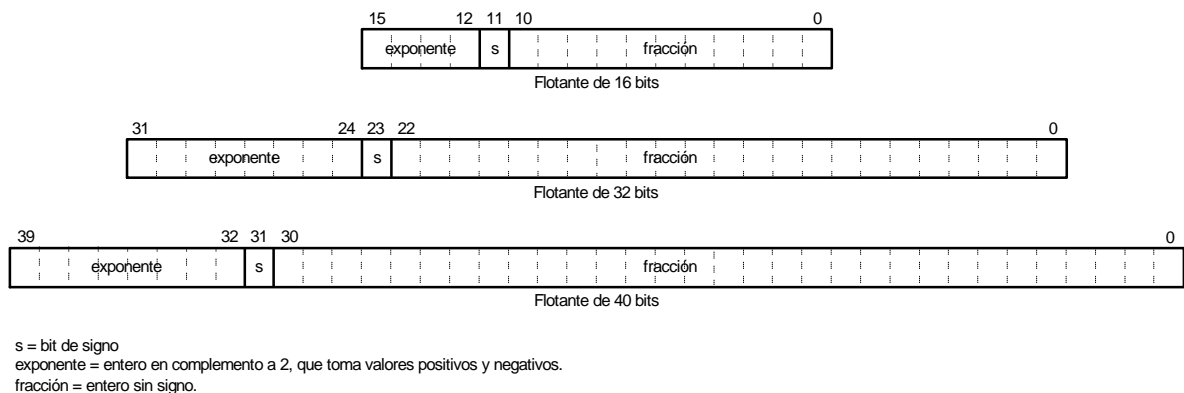


Figura 2.12. Representación de datos en punto flotante

2.2.6. 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. El DSP TMS320C30 dispone de seis modos de direccionamiento:

- **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.

- **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 bits menos significativos o LSB (*Least Significant Bit*) 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 bits más significativos o MSB (*Most Significant Bit*) de la dirección donde se encuentra el dato. Los 16 LSB que faltan de la dirección del dato se indican en la instrucción.

Ejemplo:

ADDI @0BCDEh,R7 ; $R7 = R7 + \text{dato}$

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

; Si $DP=0x8A$ se encuentra en la dirección $0x8ABCDE$.

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

Tabla 2.2. Resultado del ejemplo anterior

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.

- **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 LSB que resultan de operar aritméticamente (suma), mediante las ARAU, el registro ARi empleado y el desplazamiento definido en la instrucción. Ejemplos:

1) 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.

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

Tabla 2.3. Resultado del ejemplo anterior

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

2) 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.

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

Tabla 2.4. Resultado del ejemplo anterior

3) 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.

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

Tabla 2.5. Resultado del ejemplo anterior

A continuación se muestran los diferentes modos de direccionamiento indirecto según se emplee el desplazamiento fijo de 8 bits definido por el usuario, 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, FFT). 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*.

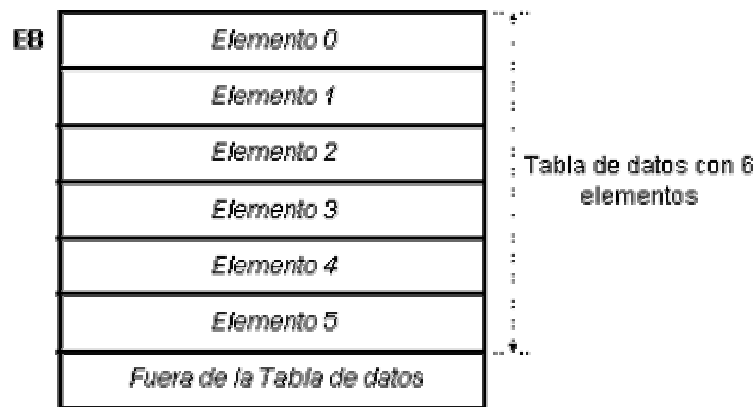


Figura 2.13. Ilustración del ejemplo anterior

En la figura 2.13 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 LSB a cero y los MSB coinciden con los MSB de AR_n . 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).

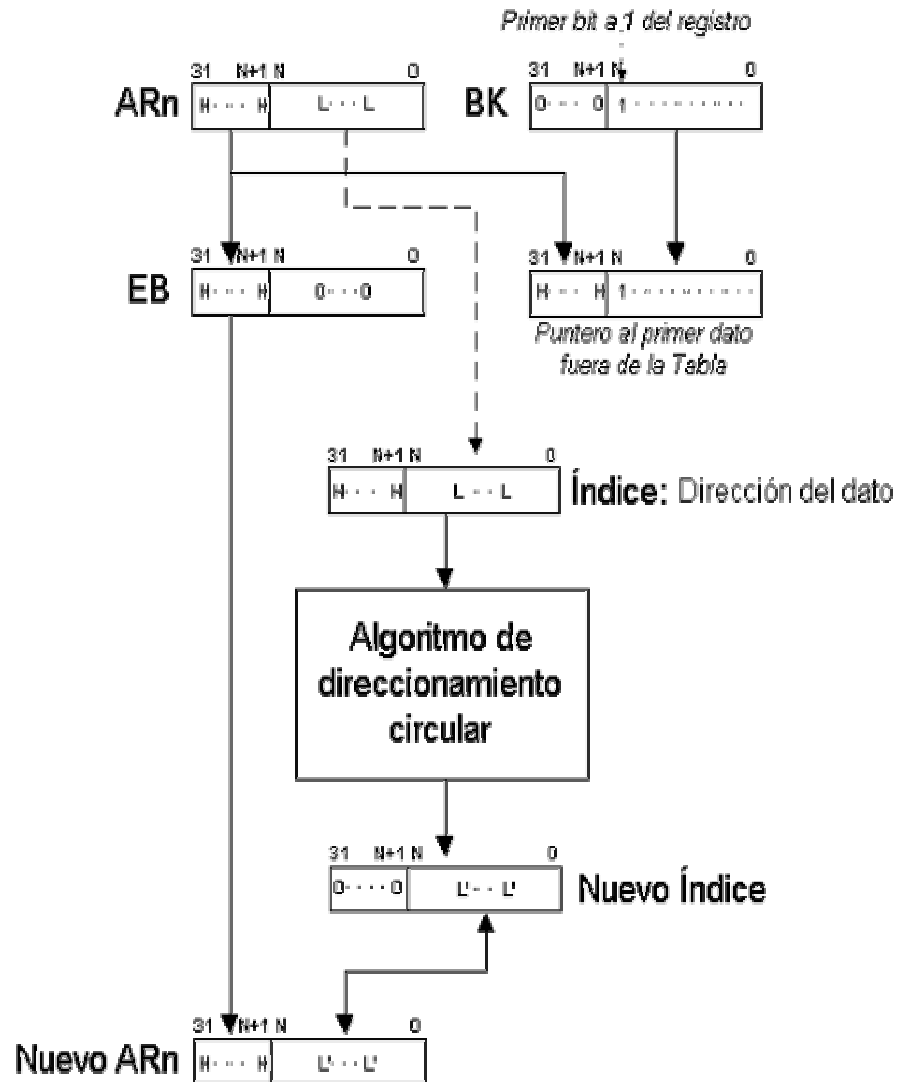


Figura 2.14. Direccionamiento indirecto circular

- Direccionamiento **indirecto a bit invertido** (*bit-reversed*). Particularmente interesante en la implementación de algoritmos como la transformada rápida de Fourier (FFT, *Fast Fourier Transform*) donde, si los datos iniciales que desean ser transformados están colocados 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 ; después, AR2 = 01100100b: *Elemento 4*.

*AR2++(IR0)B ; después, AR2 = 01101100b: *Elemento 12*.

*AR2++(IR0)B ; después, AR2 = 01100010b: *Elemento 2*.

*AR2++(IR0)B ; después, AR2 = 01101010b: *Elemento 10*.

*AR2++(IR0)B ; después, AR2 = 01100110b: *Elemento 6*.

*AR2++(IR0)B ; después, AR2 = 01101110b: *Elemento 14*.

*AR2++(IR0)B ; después, AR2 = 01100001b: *Elemento 1*.

*AR2++(IR0)B ; después, AR2 = 01101001b: *Elemento 9*.

*AR2++(IR0)B ; después, AR2 = 01100101b: *Elemento 5*.

*AR2++(IR0)B ; después, AR2 = 01101101b: *Elemento 13*.

*AR2++(IR0)B; después, AR2 = 01100011b: *Elemento 3*.

*AR2++(IR0)B ; después, AR2 = 01101011b: *Elemento 11*.

*AR2++(IR0)B ; después, AR2 = 01100111b: *Elemento 7*.

*AR2++(IR0)B ; después, AR2 = 01101111b: *Elemento 15*.

*AR2++(IR0)B ; después, AR2 = 01100000b: De nuevo ***elemento 0***.

- **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.

- 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.

- Direccionamiento **relativo al PC**. 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.

2.2.6.1. Manejo de la pila del sistema y del usuario

El C30 posee un registro de puntero de pila del sistema SP para construir pilas en memoria. Además, los registros auxiliares pueden ser usados para construir un conjunto de listas lineales como:

- Pilas: los datos se insertan y se sacan del final de la lista.
- Colas: los datos se insertan al final de la lista, mientras que se sacan del principio.

1) Puntero de pila del sistema

El registro de puntero de pila del sistema o SP, es un registro de 32 bits que contiene la dirección de la cima de la pila del sistema.

La pila del sistema se llena de las direcciones bajas de memoria a las más altas. El SP siempre apunta al último elemento almacenado en la pila. Un almacenamiento realiza un preincremento, y una descarga de pila realiza un postdecremento del puntero de pila.

El contador de programa es almacenado en la pila del sistema en las llamadas a subrutinas, interrupciones hardware y software. Es descargado de la pila del sistema al retornar de tales bloques de código.

Las instrucciones que se utilizan para trabajar con la pila son PUSH, POP, PUSHF y POPF.

En la figura 2.15 se muestra un diagrama con la configuración de la pila del sistema:

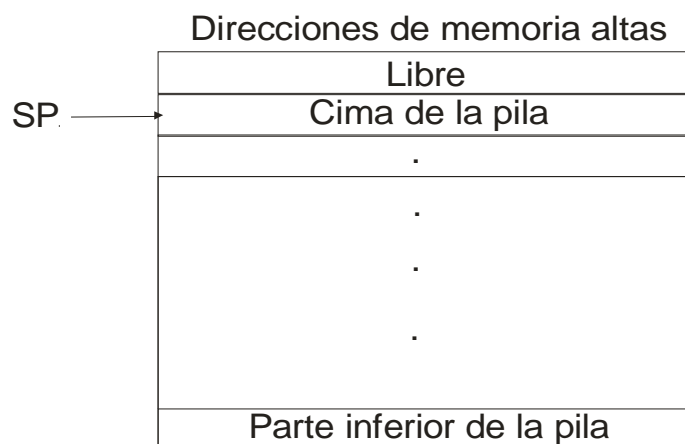


Figura 2.15. Configuración de la pila del sistema

2) Pilas:

Las pilas pueden ser construidas de zonas de memoria baja a alta o al contrario, de alta a baja. Para ello existen dos alternativas para la modificación de los registros auxiliares AR, en el caso de que la pila crezca hacia las direcciones bajas de memoria:

- Caso 1: Utilizando $*--ARn$ para almacenar en la pila datos de memoria y $*ARn++$ para descargar datos de la pila.
- Caso 2: Utilizando $*ARn--$ para almacenar en la pila los datos y $*++ARn$ para descargar los datos de la misma.

En la figura 2.16, se muestra un diagrama donde se pueden observar las diferencias entre los dos casos posibles en los que la pila va de las direcciones altas a las bajas de memoria:

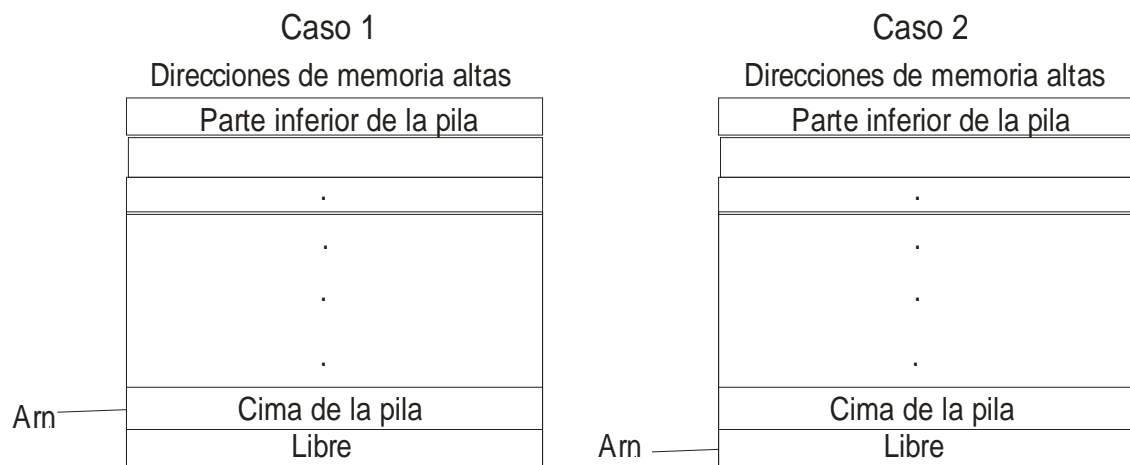


Figura 2.16. Pila con crecimiento hacia las direcciones bajas de memoria

Existen otras dos configuraciones posibles creciendo la pila en este caso de direcciones bajas a altas de memoria:

- Caso 3: Almacenando datos en la pila usando $*++ARn$ y descargándolos usando $*ARn--$.
- Caso 4: Almacenando datos en la pila usando $*ARn++$ y descargándolos usando $*--ARn$.

Estos dos casos se muestran en la figura 2.17:

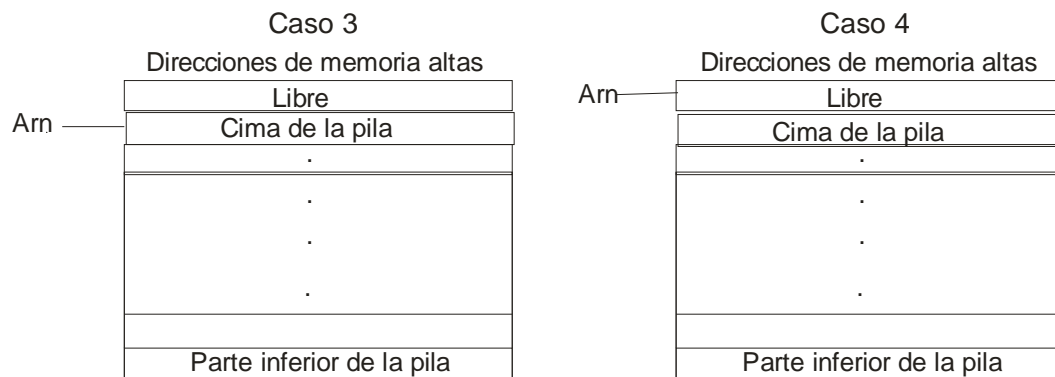


Figura 2.17. Pila con crecimiento hacia las direcciones altas de memoria

3) Colas:

Una cola es como una estructura FIFO (*First In First Out*). La implementación de la cola está basada en la manipulación de los registros auxiliares.

Se deben utilizar dos registros auxiliares: uno señala el principio de la cola que es por donde la cola decrece y el otro señala el final de la misma que es por donde se añaden elementos a la cola.

Con el correcto uso de los registros la cola puede ser incluso circular, utilizando el direccionamiento circular comentado anteriormente.

2.2.7. CONTROL DE FLUJO DE PROGRAMA

El C30 tiene un conjunto de construcciones para facilitar el control software y hardware del flujo del programa.

El control software incluye repeticiones, saltos, llamadas y retornos de las mismas. El control hardware incluye la operación de reset, interrupciones y modos de regulación de energía.

El C30 proporciona un mecanismo para compartir memoria global entre varios procesadores de forma coherente. Como en el sistema de desarrollo no es posible trabajar con varios DSP simultáneamente de forma fácil, no será comentada esta característica.

2.2.7.1. Modos de repetición

Los modos de repetición pueden implementar bucles hardware con 0 ciclos de sobrecarga. Para algunos algoritmos, el mayor tiempo de ejecución es empleado en los bucles. Usando los modos de repetición se puede reducir significativamente el tiempo de cómputo en algunas ocasiones.

El C30 tiene dos instrucciones para soportar bucles con 0 ciclos de sobrecarga:

- RPTB (*RePeaT a Block of code*), que repite la ejecución de un bloque de código un número determinado de veces.
- RPTS (*RePeaT a Single instruction*), que busca una sola instrucción y repite su ejecución un número especificado de veces. Como la instrucción es buscada solamente una vez, se reduce la carga del bus.

RPTS y RPTB son instrucciones de 4 ciclos de reloj. Estos 4 ciclos de sobrecarga ocurren durante la ejecución inicial del bucle, mientras que las restantes ejecuciones del bucle no tienen ninguna sobrecarga.

Tres registros controlan el cambio del PC cuando el C30 está en un modo de repetición:

- RS: Este registro contiene la dirección de la primera instrucción del bloque de código que va a ser repetido.
- RE: Este registro contiene la dirección de la última instrucción del bloque de código a ser repetido.
- RC: Contiene una unidad menos que el número de repeticiones del bucle.

1) Bits de control del modo repetición:

Existen dos bits importantes para la operación en este modo de trabajo del DSP:

- Bits RM: Este indicador del registro de estado especifica cuando el procesador está en el modo de repetición. En caso de valer 1, la búsqueda de la instrucción es hecha en el modo de repetición.
- Bit S: El bit S no puede ser programado ya que es interno al procesador, pero es necesario para describir el modo de repetición.

§ Si RM=1 y S=0, RPTB está ejecutándose. Las búsquedas de las instrucciones del programa ocurren de memoria.

§ Si $RM=1$ y $S=1$, RPTS está ejecutándose. Después de la primera búsqueda de instrucción, la búsqueda ocurre del registro de instrucción.

2) Modo de operación de repetición:

La información en los registros del modo de repetición y sus bits de control asociados controlan la modificación del PC en los modos de repetición durante la fase de búsqueda de la instrucción en el *pipeline*.

Los modos de repetición comparan el contenido del registro RE con el PC después de la ejecución de cada instrucción. Si coincide y el registro RC no es negativo, el RC es decrementado; el PC es cargado con la dirección de comienzo del bloque a repetir y el proceso continúa. En la etapa de búsqueda de la instrucción el bit de estado apropiado es modificado si es necesario.

El RC nunca se modifica en caso de que el bit RM sea 0.

El contador de repetición RC debe ser cargado con uno menos del número de repeticiones a realizar del bloque. El máximo número de repeticiones posibles ocurre cuando se carga 80000000h en el RC, por lo que el número de repeticiones es 800000001h.

3) La instrucción RPTB:

La instrucción RPTB repite un bloque de código un número especificado de veces. Debido a que esta instrucción no carga el RC, éste debe ser actualizado antes de su ejecución.

Para configurar y ejecutar esta instrucción se suele seguir la siguiente estructura:

LDI 15,RC

RPTB FIN_BUCLE

INICIO_BUCLE

.

.

FIN_BUCLE

En la cual se repetirá 16 veces el código comprendido entre los identificadores 'INICIO_BUCLE' y 'FIN_BUCLE'.

La inicialización de la instrucción puede ser interrumpida, ya que esta realiza 4 acciones diferenciadas:

- 1- Carga la dirección de comienzo del bloque en el registro RS. Es decir: $RS \leftarrow PC$ de $RPTB+1$.
- 2- Carga la dirección final del bloque en el registro RE.
- 3- Configura el registro de estado para indicar el modo de repetición $RM \leftarrow 1$.
- 4- Indica la operación del modo de repetición limpiando el bit S. $S \leftarrow 0$.

Es posible detener la repetición del bloque poniendo RC a 0, o escribiendo un 0 en el bit RM del registro de estado.

4) La instrucción RPTS:

La instrucción RPTS repite la siguiente instrucción una vez más del número de veces especificado en el registro RC. Esta instrucción, al contrario que la anterior, no puede ser interrumpida, ya que la búsqueda de la instrucción solamente se realiza una vez y se almacena para reutilizarla tantas veces como sea necesaria. Una interrupción en este caso puede hacer que se pierda la palabra de instrucción.

Esta instrucción reduce los accesos a memoria, ya que no es necesario buscar la instrucción cada vez. En el caso de que se deba permitir que el proceso se interrumpa, se puede utilizar la instrucción RPTB con un bloque de una instrucción.

En este caso la instrucción realiza las siguientes tareas internamente:

- 1- $RS \leftarrow PC+1$.
- 2- $RE \leftarrow PC+1$.
- 3- Bit RM del registro de estado $\leftarrow 1$.
- 4- Bit S $\leftarrow 1$.
- 5- $RC \leftarrow$ Operando.

En esta instrucción se carga automáticamente el valor de RC, por lo que no es necesario cargar su valor antes de la instrucción.

5) Restricciones:

Debido a que en los modos de repetición se modifica el PC, ninguna otra instrucción puede modificar el PC simultáneamente. Así se puede formular dos reglas:

- 1- La última instrucción del bloque no puede ser Bcond, BR, DBcond, CALL, CALLcond, TRAPcond, RETIcond, IDLE, RPTB o RPTS.
- 2- Las 4 últimas instrucciones de bloque no pueden ser BcondD, BRD o DbcondD.

6) Salto retardados:

El C30 posee tres tipos principales de saltos: estándar, retardados y condicionales retardados.

En los saltos estándar se espera a que el *pipeline* esté vacío antes de realizar el salto, asegurando el correcto uso del PC. De esta forma, el salto se hace efectivo en 4 ciclos de instrucción. En esta clase se encuentran incluidos las repeticiones, las llamadas, los retornos y las trampas.

En los saltos retardados no se espera a que el pipeline esté vacío, por lo que se ejecutarán tres instrucciones antes de que sea efectivo el cambio en el PC. Al poder ejecutar 3 instrucciones antes de efectuar el salto, hace que el salto sólo ocupe un ciclo de instrucción, optimizando de esta forma el código. El único problema de este tipo de salto es encontrar estas tres instrucciones que se puedan ejecutar en ese instante.

Cada salto estándar tiene como contrapartida su salto retardado. Los tipos de saltos retardados son: BcondD, BRD, y DBcondD.

Los saltos retardados condicionales usan las condiciones que existen al final de la instrucción inmediatamente precedente al salto retardado. Los indicadores de condición son modificados por la instrucción anterior sólo cuando el registro destino es uno de los registros de precisión extendida o cuando es una instrucción de comparación.

En los saltos retardados se garantiza que las siguientes 3 instrucciones se ejecutarán a pesar de los conflictos en el *pipeline*.

7) Funciones e interrupciones software:

Las llamadas a funciones e interrupciones software permiten invocar bloques concretos de código, pudiendo posteriormente volver a la rutina invocante en el punto por donde comenzó la ejecución del bloque de código.

La instrucción CALL (4 ciclos), CALLcond (5 ciclos) y TRAPcond (5 ciclos) almacena el valor del PC en la pila antes de cambiar al contenido del mismo para producir un salto al bloque de código adecuado. Las instrucciones RETScond o RETIcond usan el valor almacenado en la pila para retomar la ejecución del programa por el punto en que se invocó la llamada a la función o la interrupción software.

La instrucción CALL almacena el valor del PC en la pila y modifica el PC poniéndolo al valor que se pasa como operando. El operando es un valor inmediato de 24 bits.

La instrucción CALLcond es como la anterior excepto por dos diferencias:

- Se ejecuta sólo si la condición específica es cierta.
- El operando es un desplazamiento relativo al PC o un registro de direccionamiento.

Los indicadores de condición son establecidos por la instrucción anterior solamente en el caso de que el registro de destino de la instrucción sea uno de los registros de precisión extendida (R0-R7) o cuando la instrucción es una comparación.

La instrucción TRAPcond se ejecuta solamente en el caso de que la condición sea cierta. Cuando ésta se ejecuta se realizan las siguientes acciones:

- 1- Las interrupciones son deshabilitadas escribiendo un 0 en el bit GIE del registro ST.
- 2- El siguiente valor del PC es almacenado en la pila.
- 3- La dirección de la interrupción es obtenida de la tabla de vectores de interrupción y almacenada en el PC.

La instrucción RETScond devuelve el flujo del programa al punto donde se invocó a la subrutina, obteniendo el valor del PC descargándolo de la pila, donde anteriormente se había almacenado, en el caso de que la condición sea cierta.

La instrucción *RETI*cond devuelve el flujo del programa al punto donde se invocó la interrupción y se habilitan las interrupciones poniendo el bit GIE a 1.

2.2.8. SUMARIO DE INSTRUCCIONES

El formato o forma genérica de expresar las instrucciones en ensamblador del DSP TMS320C30 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. Todas las instrucciones se pueden ejecutar 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 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:

<i>MNEMÓNICO1</i>	<i>SRC1,SRC2,DST1</i>
// <i>MNEMÓNICO2</i>	<i>SRC3,SRC4,DST2</i>

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 dicha 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 DSP, 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.

En la tabla 2.6 se enumeran las instrucciones de almacenamiento y carga de datos de que dispone el DSP; en la tabla 2.7 se enumeran las instrucciones que realizan operaciones

aritméticas simples; en la tabla 2.8 se enumeran las instrucciones que realizan operaciones lógicas; en la tabla 2.9 se enumeran las instrucciones de control del formato del dato; en la tabla 2.10 se enumeran las instrucciones de control del contador de programa; en la tabla 2.11 se enumeran las instrucciones interbloqueadas y, finalmente, en la tabla 2.12 se enumeran las instrucciones que realizan dos operaciones en paralelo.

2.2.8.1. Operaciones de carga y almacenamiento

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 2.6. Operaciones básicas de carga y almacenamiento de datos del TMS320C30

2.2.8.2. Operaciones aritméticas básicas

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 <i>cond</i>	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 2.7. Operaciones aritméticas simples básicas, TMS320C30

2.2.8.3. Operaciones lógicas básicas

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 2.8. Operaciones lógicas básicas, TMS320C30

2.2.8.4. Operaciones básicas de control del formato de los datos

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

Tabla 2.9. Operaciones básicas de control del formato de los datos, TMS320C30

2.2.8.5. Operaciones básicas de control del contador de programa

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 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. (Esta instrucción no debe ser utilizada por un usuario).
TRAPcond	Llamada condicionada a vector TRAP.

Tabla 2.10. Operaciones básicas de control del contador de programa, TMS320C30

2.2.8.6. Operaciones básicas con interbloqueo

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 2.11. Operaciones básicas con interbloqueo, TMS320C30

2.2.8.7. Operaciones de ejecución paralela

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 2.12. Operaciones en paralelo, TMS320C30

2.2.8.8. Descripción del juego de instrucciones

En la tabla 2.13 se describen, por orden alfabético, las instrucciones de que dispone esta familia de microprocesadores de Texas Instruments. Además, en la tabla 2.14 se describen las instrucciones de ejecución paralela. Las instrucciones condicionadas, disponibles en esta familia de DSP, precisan de un código que indica el tipo de condición que debe de

cumplirse para que la instrucción se ejecute. En la tabla 2.15 se suministran los códigos y tipos de condición que contempla el ensamblador de la familia TMS320C30.

Mnemónico	Operación	Descripción
Instrucciones simples		
<i>Tipo de operandos especificados en la columna Operación (modos de direccionamiento posibles al operando):</i> <ol style="list-style-type: none"> 1) Operando src, count: cualquier modo de direccionamiento es posible. 2) Operandos src1, src2 (direccionamiento a registro o indirecto con desplazamiento 0, 1, IR0 ó IR1). 3) Operando Csrc (direccionamiento a registro o relativo al contador de programa). 4) Operandos Dreg, Sreg: modo de direccionamiento a registro. 5) Operando Rn: modo de direccionamiento a registro R0-R7. 6) Operando ARn: modo de direccionamiento a registro AR0-AR7. 7) Operando addr: modo de direccionamiento inmediato largo. 		
ABSF	$ src \rightarrow Rn$	Calcula el valor absoluto de un número flotante.
ABSI	$ src \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 $count \geq 0 \Rightarrow$ $Dreg \ll count \rightarrow Dreg$ else $Dreg \gg count \rightarrow Dreg$	Desplazamiento aritmético (conserva el signo) <i>count</i> veces. Si <i>count</i> >0, desplazamiento a la izquierda, si <i>count</i> <0, a la derecha.
ASH3	if $count \geq 0 \Rightarrow$ $src \ll count \rightarrow Dreg$ else $src \gg count \rightarrow Dreg$	Desplazamiento aritmético (conserva el signo) <i>count</i> veces, de tres operandos. Si <i>count</i> >0, desplazamiento a la izquierda, si <i>count</i> <0, a la derecha.
Bcond	if <i>cond</i> es cierto if <i>Csrc</i> es un registro, $Csrc \rightarrow PC$ if <i>Csrc</i> es un valor, $Csrc + PC \rightarrow PC$	Salto condicional.

	else $PC + 1 \rightarrow PC$	
BcondD	if <i>cond</i> es cierto if <i>Csrc</i> es un registro, $Csrc \rightarrow PC$ if <i>Csrc</i> 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.
CALLcond	if <i>cond</i> es cierto $PC + 1 \rightarrow *++SP$ if <i>Csrc</i> es un registro, $Csrc \rightarrow PC$ if <i>Csrc</i> 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).
DBcond	$ARn - 1 \rightarrow ARn$ if <i>cond</i> es cierto y $ARn \geq 0$ if <i>Csrc</i> es un registro, $Csrc \rightarrow PC$ if <i>Csrc</i> es un valor, $Csrc + PC + 1 \rightarrow PC$ else $PC + 1 \rightarrow PC$	Decrementa y salta condicionalmente.
DBcondD	$ARn - 1 \rightarrow ARn$ if <i>cond</i> es cierto y $ARn \geq 0$ if <i>Csrc</i> es un registro, $Csrc \rightarrow PC$ if <i>Csrc</i> es un valor, $Csrc + PC + 3 \rightarrow PC$ else $PC + 1 \rightarrow PC$	Decrementa y salta condicionalmente y con retardo.

FIX	$(\text{int})(\text{src}) \rightarrow \text{Dreg}$	Convierte un valor flotante en entero.
FLOAT	$(\text{float})(\text{src}) \rightarrow \text{Rn}$	Convierte un valor entero en flotante.
IACK	<i>IACK</i> se pone a 0 y luego a 1 Lectura sin utilidad de <i>src</i>	Reconocimiento de interrupción.
IDLE	$\text{PC} + 1 \rightarrow \text{PC}$ Idle hasta la próxima interrupción	Parada de la CPU hasta la llegada de una interrupción.
LDE	$\text{exp}(\text{src}) \rightarrow \text{exp}(\text{Rn})$	Carga en el exponente de <i>Rn</i> el de otro valor flotante.
LDF	$\text{src} \rightarrow \text{Rn}$	Carga en <i>Rn</i> un valor flotante.
LDFcond	if <i>cond</i> es cierto $\text{src} \rightarrow \text{Rn}$ else no se actualiza <i>Rn</i>	Carga condicionalmente en <i>Rn</i> un valor flotante.
LDFI	$\text{src} \rightarrow \text{Rn}$, interbloqueado	Pone $\text{XF0}=0$ y comienza un ciclo de lectura del flotante <i>src</i> . Ejecuta un LDF y extiende el ciclo de lectura hasta que no se ponga $\text{XF1}=0$. Termina el acceso y deja $\text{XF0}=0$.
LDI	$\text{src} \rightarrow \text{Dreg}$	Carga un entero.
LDIcond	if <i>cond</i> es cierto $\text{src} \rightarrow \text{Dreg}$ else no se actualiza <i>Dreg</i>	Carga un entero condicionalmente.
LDII	$\text{src} \rightarrow \text{Dreg}$, interbloqueado	Pone $\text{XF0}=0$ y comienza un ciclo de lectura del entero <i>Dreg</i> . Ejecuta un LDI y extiende el ciclo de lectura hasta que no se ponga $\text{XF1}=0$. Termina el acceso y deja $\text{XF0}=0$.
LDM	$\text{mantisa}(\text{src}) \rightarrow \text{mantisa}(\text{Rn})$	Carga en la mantisa de <i>Rn</i> la de otro valor flotante
LSH	if $\text{count} \geq 0 \Rightarrow$ $\text{Dreg} \ll \text{count} \rightarrow \text{Dreg}$ else $\text{Dreg} \gg \text{count} \rightarrow \text{Dreg}$	Desplazamiento lógico (no conserva el signo) <i>count</i> veces. Si $\text{count}>0$, desplazamiento a la izquierda, si $\text{count}<0$, a la derecha.
LSH3	if $\text{count} \geq 0 \Rightarrow$ $\text{src} \ll \text{count} \rightarrow \text{Dreg}$ else $\text{src} \gg \text{count} \rightarrow \text{Dreg}$	Desplazamiento aritmético (conserva el signo) <i>count</i> veces, de tres operandos. Si $\text{count}>0$, desplazamiento a la izquierda, si $\text{count}<0$, a la derecha.
MPYF	$\text{src} \times \text{Rn} \rightarrow \text{Rn}$	Multiplica valores flotantes.
MPY3	$\text{src1} \times \text{src2} \rightarrow \text{Rn}$	Multiplica valores flotantes (tres operandos).
MPYI	$\text{src} \times \text{Dreg} \rightarrow \text{Dreg}$	Multiplica valores enteros.
MPYI3	$\text{src1} \times \text{src2} \rightarrow \text{Dreg}$	Multiplica valores enteros (tres operandos).
NEGB	$0 - \text{src} - \text{C} \rightarrow \text{Dreg}$	Entero cambiado de signo menos bit de acarreo.
NEGF	$0 - \text{src} \rightarrow \text{Rn}$	Flotante cambiado de signo.
NEGI	$0 - \text{src} \rightarrow \text{Dreg}$	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.
RETIcond	if <i>cond</i> 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.
RETScond	if <i>cond</i> 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	$src \rightarrow RE$ $1 \rightarrow ST(RM)$ <i>Próximo</i> $PC \rightarrow RS$	Ejecuta varias veces (RC+1 veces) un bloque de instrucciones.
RPTS	$src \rightarrow RC$ $1 \rightarrow ST(RM)$ <i>Próximo</i> $PC \rightarrow RS$ <i>Próximo</i> $PC \rightarrow RE$	Ejecuta varias veces (RC+1 veces) una instrucción.
SIGI	<i>Maneja señales de Interbloqueo</i>	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	$Rn \rightarrow Daddr$, <i>interbloqueado</i>	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	$Sreg \rightarrow Daddr$, <i>interbloqueado</i>	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 <i>borrow</i> de dos operandos.
SUBB3	$src1 - src2 - C \rightarrow Dreg$	Operación de resta enteros con <i>borrow</i> de tres operandos.
SUBcond	if $Dreg - src \geq 0$ $[(Dreg - src) \ll 1] \oplus 1 \rightarrow Dreg$ else $Dreg \ll 1 \rightarrow Dreg$	Operación de resta enteros condicionada.
SUBF	$Rn - src \rightarrow Rn$	Operación de resta de flotantes, de dos operandos.
SUBF3	$src1 - src2 \rightarrow Rn$	Operación de resta de flotantes, de tres operandos.
SUBI	$Dreg - src \rightarrow Dreg$	Operación de resta de enteros, de dos operandos.
SUBI3	$src1 - src2 \rightarrow Dreg$	Operación de resta de enteros, de tres operandos.
SUBRB	$src - Dreg - C \rightarrow Dreg$	Operación de resta de entero cambiado de signo con <i>borrow</i> .
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 prohíbe su uso.
TRAPcond	if <i>cond</i> es cierto o no existe $PC + 1 \rightarrow *++SP$ (<i>Top Pila</i>) $Vector\ Trap^N \rightarrow PC$ $0 \rightarrow ST(GIE)$ 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).

Tabla 2.13. Sumario de instrucciones del DSP TMS320C30

Instrucciones en paralelo		
<p><i>Tipos de operando (modos de direccionamiento posibles al operando):</i></p> <ol style="list-style-type: none"> 1) operandos src1, src3, dst1: Cualquier modo de direccionamiento a registro R0-R7 2) operando op3: direccionamiento a registro R0 ó R1 3) operando op6: direccionamiento a registro R2 ó R3 4) operandos src2, src4, dst2: modo de direccionamiento indirecto con desplazamiento 0, 1, IR0 ó IR1 5) 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	$ src2 \rightarrow dst1$ $src3 \rightarrow dst2$	Calcula el valor absoluto de un número flotante. En paralelo, almacena valor flotante.
ABSI STI	$ src2 \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 $count \geq 0 \Rightarrow$ $src2 \ll count \rightarrow dst1$ $src3 \rightarrow dst2$ else $src2 \gg count \rightarrow dst1$ $src3 \rightarrow dst2$	Desplazamiento aritmético (conserva el signo) <i>count</i> veces, de tres operandos. Si $count > 0$, desplazamiento a la izquierda, si $count < 0$, a la derecha. En paralelo, almacena valor entero.
FIX STI	$(int)(src2) \rightarrow dst1$ $src3 \rightarrow dst2$	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	$src2 \rightarrow dst1$	Carga dos enteros.

	$src4 \rightarrow dst2$	
LDI STI	$src2 \rightarrow dst1$ $src3 \rightarrow dst2$	Carga un entero. En paralelo, almacena otro entero.
LSH3 STI	if $count \geq 0 \Rightarrow$ $src2 \ll count \rightarrow dst1$ $src3 \rightarrow dst2$ else $src2 \gg count \rightarrow dst1$ $src3 \rightarrow dst2$	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	$op1 \times op2 \rightarrow op3$ $op4 - op5 \rightarrow op6$	Multiplica dos enteros. 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 2.14. Sumario de instrucciones de ejecución paralela del DSP TMS320C30

<i>cond</i>	Descripción
	<i>Compara incondicionalmente</i>
U	Sin condición.

<i>Compara sin signo</i>	
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).
<i>Compara con signo</i>	
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).
<i>Compara con cero</i>	
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).
<i>Compara con otros códigos de condición del registro ST de la CPU</i>	
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 2.15. Códigos y tipos de condición del DSP TMS320C30

2.2.9. PERIFÉRICOS

El C30 posee dos temporizadores, dos puertos serie y un controlador DMA. Estos periféricos están controlados por registros mapeados en memoria accesibles a través de un bus con dedicación exclusiva.

- **Temporizadores:** El C30 tiene dos temporizadores de propósito general de 32 bits. Cada temporizador tiene dos modos de señalización, pudiendo ser el reloj que controla externo o interno. Se puede usar cada temporizador para mandar señales al C30, para generar una señal externa con un intervalo predefinido o para contar eventos externos. No se profundizará en el estudio de los temporizadores ya que no han sido implementados en el simulador.
- **Puertos Serie:** El C30 posee dos puertos serie bidireccionales e idénticos y totalmente independientes entre sí. Cada puerto serie puede realizar transferencias de datos de 8, 16, 24 o 32 bits por palabra. La señal de reloj para cada puerto serie se puede generar internamente, a través del temporizador del puerto serie, o a través de un reloj suministrado de forma externa.
- **Controlador de DMA:** El controlador de DMA es un periférico programable, que transfiere bloques de datos a cualquier localización del mapa de memoria, sin interferir con el funcionamiento de la CPU. Las características más relevantes de este periférico son:
 - Transferencia desde y hacia cualquier zona de memoria del C30.
 - Operación concurrente del controlador DMA y la CPU, gracias a los buses separados de datos y direcciones.
 - Registros de dirección fuente y destino con modificación automática.
 - Sincronización de datos transferidos desde el exterior mediante interrupciones.

No se profundizará en el estudio de este periférico ya que no ha sido implementado en el simulador.

2.2.9.1. Puertos Serie

Como ya se ha comentado, el C30 posee dos puertos serie bidireccionales e idénticos y totalmente independientes entre sí. La señal de reloj para cada puerto serie se puede generar internamente o externamente: en el caso de que esté generado internamente, el reloj generado es un divisor de frecuencia de reloj del C30 $f(H1)$.

Existen ocho registros mapeados en memoria para cada uno de los puertos serie:

Cada periférico puerto serie posee ocho registros de 32 bits, ubicados en una zona determinada del mapa de memoria del DSP, que pueden ser accedidos por software en lectura o escritura. En la tabla 2.16 se muestra la posición, en el mapa de memoria, de estos registros que son:

- Registro de control global del periférico: Ubicado en la posición 808040H (Puerto Serie0) y en la posición 808050H (Puerto Serie1). Determina el modo de operación del periférico y monitoriza el estado del mismo.
- Registro de control de FSX, DX y CLKX: Ubicado en la posición 808042H (Puerto Serie0) y en la posición 808052H (Puerto Serie1). Controla las funciones de las líneas de entrada/salida asociadas a la parte de transmisión del puerto serie.
- Registro de control de FSR, DR y CLKR: Ubicado en la posición 808043H (Puerto Serie0) y en la posición 808053H (Puerto Serie1). Controla las funciones de las líneas de entrada/salida asociadas a la parte de recepción del puerto serie.
- Registro de control de los dos temporizadores de 16 bits: Ubicado en la posición 808044H (Puerto Serie0) y en la posición 808054H (Puerto Serie1). Es un registro de 32 bits en el que los 12 bits menos significativos se corresponden con el registro de control del temporizador asociado a la parte de transmisión del puerto serie, y los 12 bits que le siguen constituyen el registro de control del temporizador asociado a la parte de recepción del puerto serie.
- Registro contador de los temporizadores de 16 bits: Ubicado en la posición 808045H (Puerto Serie0) y en la posición 808055H (Puerto Serie1). Es un registro de 32 bits constituido por los 16 bits del registro de cuenta del temporizador asociado a la parte de recepción del puerto serie (16 MSB), y los 16 bits del registro de cuenta del temporizador asociado a la parte de transmisión del puerto serie (16 LSB).
- Registro periodo de los temporizadores de 16 bits: Ubicado en la posición 808046H (Puerto Serie0) y en la posición 808056H (Puerto Serie1). Determina el valor de fin de cuenta del registro contador asociado a la recepción (16 MSB) y a la transmisión (16 LSB) del puerto serie.
- Registro de datos en transmisión, DXR: Ubicado en la posición 808048H (Puerto Serie0) y en la posición 808058H (Puerto Serie1). Registro de 32 bits. El usuario

escribe en este registro los datos que desea que transmita el periférico puerto serie. El puerto serie dispone de un doble *buffer* en transmisión: El contenido del registro DXR se vuelca, al iniciar la transmisión del dato por la línea serie, sobre otro registro que se denomina XSR –registro de desplazamiento en transmisión–. El registro XSR es el que se encarga de generar la salida de los bits del dato por la línea *DX*.

- **Registro de datos en recepción, DRR:** Ubicado en la posición 80804CH (Puerto Serie0) y en la posición 80805CH (Puerto Serie1). Registro de 32 bits. El usuario lee en este registro los datos que ha recibido el periférico. El puerto serie dispone de un doble *buffer* en recepción: Dispone de un registro de desplazamiento en recepción, denominado RSR, que se encarga de capturar los bits del dato que le llega al periférico por la línea *DR*. Posteriormente, el contenido de RSR se vuelca sobre el registro DRR que es el registro desde el cual el usuario recoge, por software, el dato recibido.

Dirección		Descripción
Puerto serie0	Puerto Serie1	
808040h	808050h	Registro Global de Control
808041h	808051h	Reservado
808042h	808052h	Registro de Control de FSX, DX, CLKX
808043h	808053h	Registro de Control de FSR, DR, CLKR
808044h	808054h	Registro de Control de los Timers (RxD, TxD)
808045h	808055h	Registro Contador de los Timers (RxD, TxD)
808046h	808056h	Registro Periodo de los Timers (RxD, TxD)
808047h	808057h	Reservado
808048h	808058h	Registro de DATOS en transmisión
808049h	808059h	Reservado
80804Ah	80805Ah	Reservado
80804Bh	80805Bh	Reservado
80804Ch	80805Ch	Registro de DATOS en recepción
80804Dh	80805Dh	Reservado
80804Eh	80805Eh	Reservado
80804Fh	80805Fh	Reservado

Tabla 2.16. Registro del periférico puerto serie del DSP TMS320C30

2.2.9.2. Registro de transmisión de datos o DXR

Cuando se escribe en el registro de transmisión de datos o DXR, el transmisor carga la palabra en el registro de desplazamiento de transmisión XSR, y los bits son desplazados a través del puerto. La palabra no se carga en el registro de desplazamiento hasta que éste está vacío. Cuando DXR es copiado en XSR, el bit XRDY se pone a uno, indicando que el *buffer* está preparado para recibir otra palabra.

El registro de desplazamiento del transmisor está dividido en cuatro secciones, que se corresponden con los cuatro posibles tamaños de palabra de los datos: 32 bits, 24 bits, 16 bits y 8 bits.

El desplazamiento se produce hacia la izquierda, es decir, del bit menos significativo al más significativo, siendo el bit que se transmite por el puerto el más significativo de la palabra a transmitir, teniendo en cuenta que la longitud de la palabra varía.

2.2.9.3. Registro de recepción de datos o DRR

Cuando se recibe un dato por el puerto serie, el receptor desplaza los bits en el registro de desplazamiento RSR. Cuando el número de bits especificado es recibido, el registro de datos DRR se carga con la palabra recibida y el bit RRDÍ se pone a uno.

Si el DRR no ha sido leído y el RSR está lleno, el receptor está saturado. En este caso, los nuevos datos que llegan por el pin DR son ignorados. Se deben leer datos para proseguir la recepción.

Los datos son desplazados de las posiciones menos a las más significativas.

2.2.9.4. Temporización del puerto serie

La frecuencia del reloj interno del puerto serie depende del modo de trabajo del temporizador:

$$f(\text{ModoPulsos}) = \frac{f(\text{RelojTemporizador})}{\text{RegistroPeriodo}}$$

Ecuación 2.2

$$f(\text{ModoReloj}) = \frac{f(\text{RelojTemporizador})}{2 * \text{RegistroPeriodo}}$$

Ecuación 2.3

El reloj generado internamente tiene una frecuencia máxima de $f(H1)/2$, mientras que el reloj generado externamente puede tener hasta $f(H1)/2.6$, siendo $f(H1)$ la frecuencia del reloj interno del C30.

Todos los datos son transmitidos y recibidos comenzando por el bit más significativo. En caso de que se reciban menos de 32 bits, después de la operación, los datos recibidos se justifican a la derecha del *buffer* de recepción.

2.2.10. EL MÓDULO DE EVALUACIÓN (EVM)

El módulo de evaluación EVM (*Evaluation Module*) es una herramienta de desarrollo que permite ejecutar y depurar programas de aplicación.

El EVM permite que el sistema de desarrollo se encuentre dentro de una tarjeta de expansión del PC, al poseer los siguientes elementos:

- Un TMS320C30.
- 16K palabras de SRAM con 0 estados de espera en el bus primario.
- Adquisición de datos analógicos con calidad de voz a través del TLC32044 (*Analog Interface Circuit* o AIC).
- Conexión de un puerto serie al exterior.
- Puerto de comunicaciones con el PC bidireccional de 16 bits.
- Soporte de emulación embebida gracias al controlador de bus 74ACT8990.

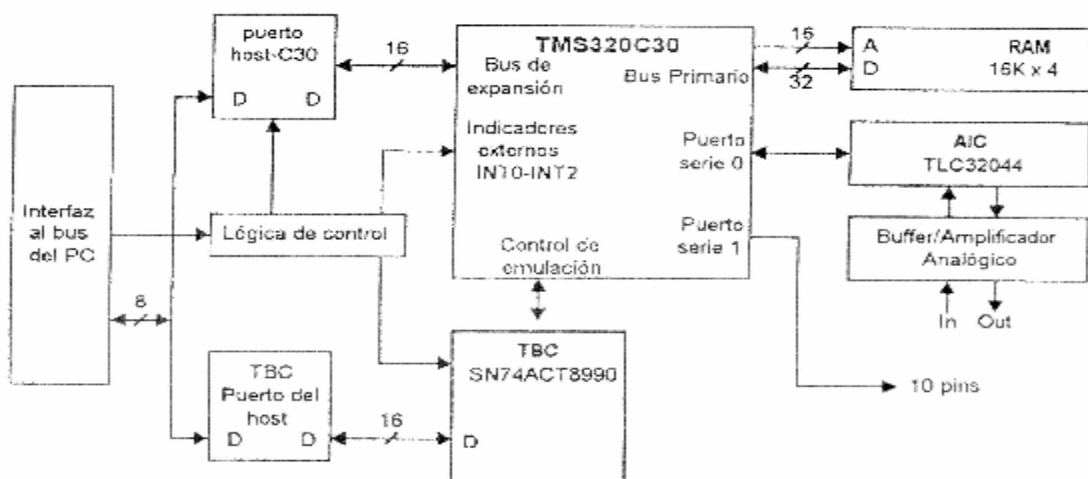


Figura 2.18. Diagrama de bloques del TMS320C30 EVM

Todo el código es cargado a través del puerto de emulación del C30, por lo que no es necesario tener memoria ROM o EPROM con un programa cargador para iniciar el sistema.

El interfaz entre el host y el C30 es muy simple. Está basado en registros y proporciona un ancho de banda moderado de 200Kbytes por segundo.

La sección analógica del EVM está formada por un circuito interfaz analógico denominado AIC TLC32044. En la salida se encuentra acoplado con un amplificador de bajo nivel de ruido (LM286) y otro en la entrada (TL072).

La ganancia de entrada y salida es fija, soportando niveles de voltaje de cualquier línea estándar de audio. Para la conexión con estos elementos se han utilizado jacks RCA externos.

El interfaz entre el AIC y el C30 se realiza a través del puerto serie 0, mientras que el puerto serie 1 está libre y hay un conector externo para poder utilizarlo.

El C30 tiene conectado directamente a través del bus primario 16K palabras con acceso a memoria sin estados de espera del tipo CY7C164-35VC.

El soporte de emulación embebida se produce gracias al controlador de bus (*Test Bus Controller* o TBC) SN74ACT8990 y al puerto de emulación del C30.

2.2.10.1. El mapa de memoria del C30

El mapa de memoria del C30 es el que se muestra en la figura 2.19:

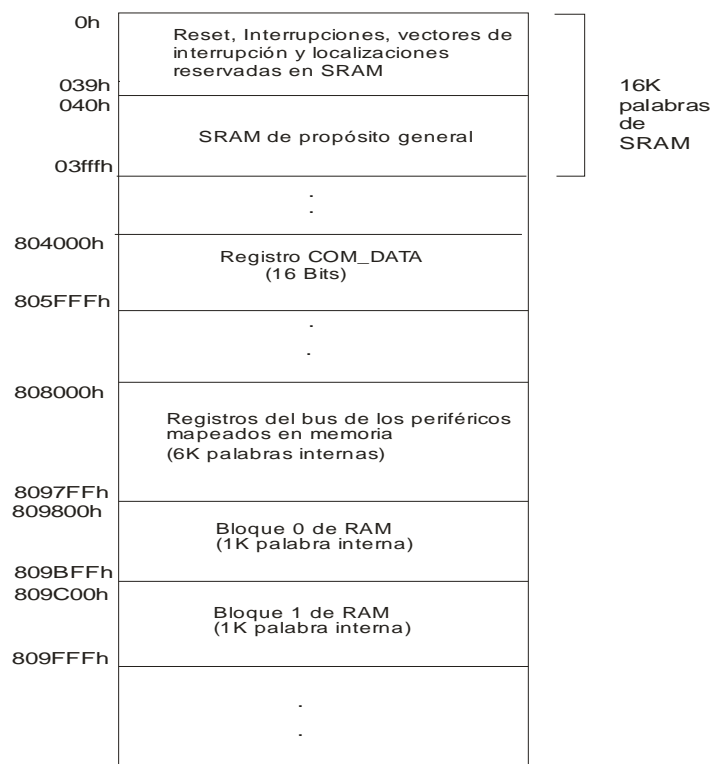


Figura 2.19. Mapa de memoria útil en el TMS320C30 EVM

Se puede observar que existe un registro llamado COM_DATA al que se puede acceder con cualquier dirección comprendida entre 804000h y 805FFFh, que se corresponde con una parte del bus de expansión.

Para poder acceder a este registro hay que configurar directamente el registro de control del bus de expansión que se encuentra mapeado en la dirección de memoria 808060h del C30. Hay que inicializarlo a 0 para que la señal de espera se genere desde el exterior, controlando de esta forma el número de estados de espera en cada acceso.

2.2.10.2. Interfaz entre el AIC y el C30

El EVM posee un conversor analógico-digital (*ADC, Analog Digital Converter*) y otro digital-analógico (*DAC, Digital Analog Converter*) que permiten obtener muestras a partir de una señal de entrada o generar una señal analógica de salida a partir de las muestras generadas por el C30.

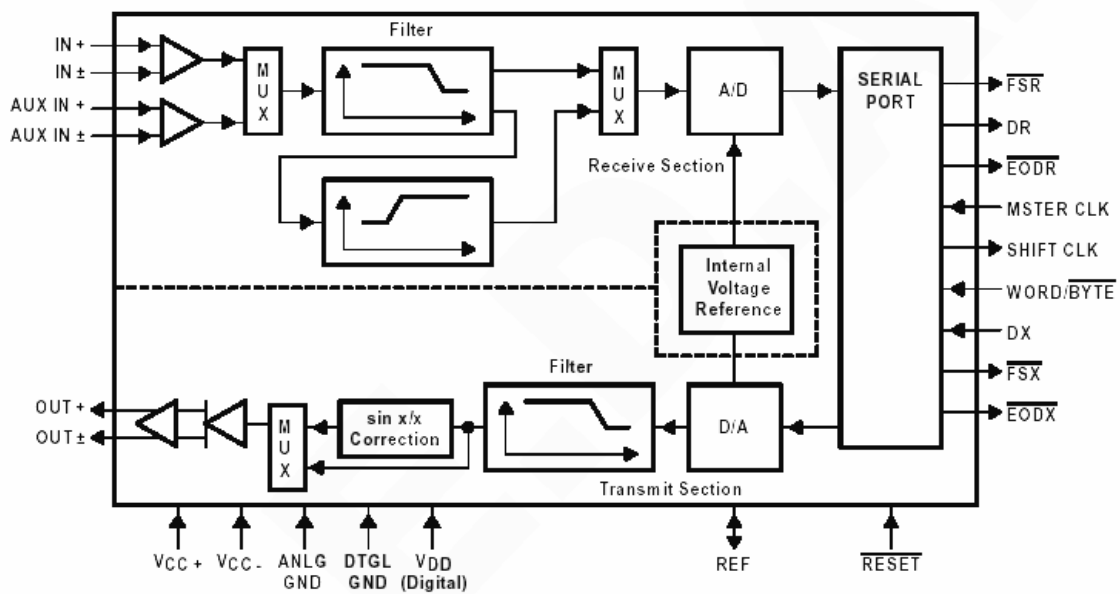


Figura 2.20. Comunicación desde AIC al puerto serie del C30

Los dos conversores están integrados en un solo chip TLC32044 denominado en nomenclatura AIC. El AIC representa para el C30 una alternativa barata para realizar la conversión entre la realidad analógica y el mundo discreto con el que los procesadores son capaces de trabajar.

La frecuencia máxima de muestreo para la señal de entrada y para la de salida es de 19200 muestras por segundo en teoría, incluyendo el ADC un filtro paso bajo (antialiasing) y el DAC un paso alto con el fin de suavizar los desperfectos de la señal analógica resultante.

La resolución del AIC tanto para el ADC como el DAC es de 14 bits, haciendo que sea un interfaz ideal para trabajar con señales analógicas con calidad vocal.

2.2.10.3. Control del AIC

El AIC posee un puerto serie con dos registros de transmisión de datos: el registro de transmisión de datos o DXR y el registro de recepción de datos o DRR.

Los dos bits menos significativos en la transmisión de datos, son utilizados para realizar funciones de control de la comunicación. Para realizar una transmisión normal de datos desde el C30, los dos bits menos significativos han de estar puestos a 0. En cambio, cuando estos dos bits se transmiten a 1, se inicia una transmisión secundaria. En este modo de transmisión secundaria, el AIC espera una segunda palabra de control, que cambia el modo de funcionamiento modificando posteriormente los registros de control del AIC. Después de la transmisión de esta palabra de control el AIC restaura el modo de funcionamiento normal.

El AIC posee internamente cinco registros de control, que modifican la frecuencia de muestreo del DAC y del ADC, así como la frecuencia del filtro paso bajo de entrada, el filtro paso alto de salida y el modo de funcionamiento general. Los registros son los siguientes:

- TA: Registro de cinco bits que controla la frecuencia del filtro paso alto del transmisor.
- RA: Registro de cinco bits que controla la frecuencia del filtro paso bajo del receptor.
- TB: Registro de seis bits que controla la frecuencia de muestreo del DAC.
- RB: Registro de seis bits que controla la frecuencia de muestreo del ADC.
- CONTROL: Registro que controla el modo de funcionamiento del AIC.

La información que se transmite entre el C30 y el AIC, se envía a través del puerto serie 0 del C30. El AIC utiliza un reloj para sincronizarse con el C30, que en el caso del EVM lo

genera el temporizador 0 del C30. Este reloj será llamado *reloj maestro* del AIC, y controlará la frecuencia de muestreo tanto del DAC como del ADC.

Como se expone posteriormente, el temporizador 0 generará una señal de reloj de 7.5 MHz, por lo que los contenidos de los registros se pueden calcular en función de la frecuencia de muestreo que se desee que tengan al ADC y el DAC.

La frecuencia de muestreo del DAC y del ADC puede ser diferente, aunque no suele ser así. Los registros TA y TB son los que influyen en la frecuencia del DAC y se calculan de esta forma:

$$TA = \frac{\text{RelojMaestroDelAIC}}{2 * SCF} = \frac{\text{RelojMaestroDelAIC}}{160 * A}$$

Ecuación 2.4

$$TB = \frac{SCF}{Fs} = \frac{\text{RelojMaestroDelAIC}}{2 * A * B}$$

Ecuación 2.5

En este caso, el reloj maestro del AIC valdrá 7.5 MHz, el SCF vale siempre 288KHz, Fs es la frecuencia de muestro del DAC, y A y B son valores tabulados.

Para el caso del ADC es exactamente igual, con la diferencia de que los registros implicados son en este caso el RA y el RB. Así las fórmulas serán:

$$RA = \frac{\text{RelojMaestroDelAIC}}{2 * SCF}$$

Ecuación 2.6

$$RB = \frac{SCF}{Fs}$$

Ecuación 2.7

De estas fórmulas obtenemos las distintas frecuencias de trabajo del ADC y DAC, según los valores tabulados A y B:

Frecuencia de muestreo	A	B
17361Hz	6	36
11574Hz	9	36
8013Hz	13	36
4006Hz	26	36
3360Hz	31	36

Tabla 2.17. Frecuencias de muestreo del ADC y DAC.

La transmisión se realiza mediante palabras de 16 bits. Es los dos sentidos de la comunicación los dos bits menos significativos realizan una distinción del tipo de transferencia a realizar.

En el caso en el que el C30 deba recibir una muestra desde el conversor analógico-digital, los 14 bits más significativos de la palabra representan la muestra convertida, mientras que los dos bits menos significativos no se utilizan. A continuación, se pueden observar los bits de esta palabra:

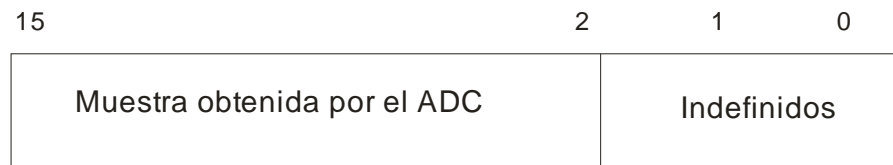


Figura 2.21. Palabra enviada por el AIC desde el C30

Desde el C30 se pueden enviar palabras de 16 bits, con diversos significados: palabras para que el DAC reconstruya la señal analógica deseada, una palabra para cambiar el registro RA y TA, otra para cambiar el valor del registro RB y TB, y por último una palabra de control.

En el caso de que lo que se deba enviar sea una palabra que incluya una nueva muestra para el DAC, el valor de la muestra se debe situar en los 14 bits más significativos y los dos bits inferiores se deben de poner a 0 como se muestra en el siguiente diagrama:

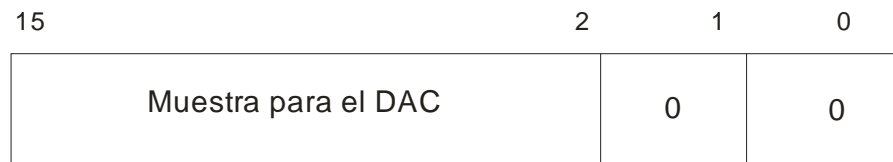


Figura 2.22. Palabra con la muestra para el DAC

2.3. ESTUDIO DEL DEPURADOR

El depurador del C30 es un interfaz avanzado para la programación, que ayuda al programador a desarrollar, testear y refinar programas en C y en ensamblador.

El interfaz de esta herramienta aumenta la productividad permitiendo al programador depurar el programa en el lenguaje que fue escrito: C, ensamblador o ambos. Las facilidades de depuración de alto nivel de código, están disponibles tanto en la depuración

de código ensamblador como en la del C. Este programa es de fácil aprendizaje y uso, gracias al uso del ratón y teclado conjuntamente.

Las características del depurador son:

- Permite trabajar con código C o ensamblador de forma simultánea o aislada.
- Interfaz orientado a ventanas completamente configurables.
- Las ventanas para la visualización de datos son fáciles de comprender y muy versátiles, permitiendo la edición de los mismos.
- Edición de datos pulsando con el ratón sobre cualquier dato.

2.3.1. DESCRIPCIÓN DE LAS VENTANAS EXISTENTES

El depurador del C30 tiene un interfaz en modo texto orientado a ventanas. Estas ventanas se pueden mover, cambiar de sitio y en algunos casos cerrarlas o crearlas. Estas acciones se pueden hacer tecleando comandos o bien mediante el ratón, que es mucho más intuitivo, ya que sigue el patrón de conducta de las ventanas de Windows.

A continuación se realizará una breve descripción de las ventanas existentes en el programa.

2.3.1.1. Ventana de comandos

Es la ventana donde se teclean los comandos y se visualiza información como mensajes de progreso, de error o los comandos de salida.

Esta ventana siempre está disponible en cualquiera de los modos de trabajo.

2.3.1.2. Ventana de visualización de código

Existen tres tipos de ventanas que hacen que se visualice el código de diferentes formas:

- Ventana de desensamblado: *'DISASSEMBLY'*, en la que se visualiza el contenido de la memoria del EVM desensamblada, es decir, el código correspondiente en ensamblador. Esta ventana aparece de forma automática cuando se ha de visualizar el código ensamblador.
- La ventana de fichero *'FILE'* muestra el texto de cualquier fichero de texto que se desee visualizar. Es una ventana de propósito general, aunque se utiliza siempre que se trabaje con el código C de los programas. Para crear una de estas ventanas, se pueden

seguir varios pasos, pero quizás el más simple sea teclear en la ventana de comandos: *'file ruta_fichero\nombre_fichero'*.

- La ventana de llamadas '*CALLS*', identifica el rastro de funciones invocadas antes de llegar hasta la función que se ejecuta actualmente. Esta ventana sólo aparece cuando se está depurando un programa escrito en C.

2.3.1.3. Ventana de visualización de datos

Son cuatro tipos de ventanas diferentes que se utilizan para observar y modificar algunos tipos de datos.

Los cuatro tipos de ventanas se pueden observar a continuación:

- La ventana de memoria '*MEMORY*', muestra el contenido de un rango de memoria. Pueden existir hasta cuatro ventanas de este tipo simultáneamente. Esta ventana muestra las direcciones de memoria mostradas junto con el contenido. La primera ventana de memoria se crea automáticamente y las siguientes escribiendo el comando '*MEM#*', donde # representa el número de ventana.
- La ventana CPU, muestra el contenido de los registros de la CPU. Esta ventana se crea automáticamente mientras se está monitorizando código en ensamblador.
- La ventana de visualización '*DISP*', muestra el contenido de algunos tipos de datos y estructuras definidas en el código. Muestra valores de miembros individuales.

2.3.2. MODOS DE VISUALIZACIÓN

El depurador permite tres modos para monitorizar el código de un programa, que se cambian en el menú del programa llamado '*Mode*'.

- Modo Automático: en este modo se reconoce automáticamente si el código que se está utilizando fue programado en C o en ensamblador, visualizándose en el lenguaje fuente. El tipo de ventana cambia según el lenguaje fuente, ya que algunas ventanas no son útiles cuando se depura un lenguaje de programación.

- Modo Ensamblador: sea cual sea el lenguaje de programación utilizando el código será representado en ensamblador. De esta forma si el código originalmente fue programado en C, lo que aparece es el código ensamblador que el compilador de C genera.
- Modo Mixto: se ven todas las ventanas de otros modos simultáneamente, como la del código C y el código ensamblador. En caso de utilizar código programado en ensamblador, la ventana de código C aparece vacía.

2.4. EL ENSAMBLADOR

Las utilidades para trabajar con el código ensamblador crean y usan ficheros objeto en un formato que TI ha llamado COFF (*Common Object File Format*). Los ficheros objeto contienen bloques separados (llamados secciones) de código y datos que se pueden cargar en diferentes zonas de memoria del C30.

2.4.1. DESCRIPCIÓN DE LAS UTILIDADES

El propósito principal de las utilidades de ensamblador es crear un módulo que pueda ser ejecutado en el TMS320C30 EVM.

A continuación se hará una breve introducción a las utilidades:

- El compilador de C: traduce el código fuente escrito en C en código ensamblador para el TMS320C30.
- El ensamblador, traduce el código fuente escrito en lenguaje ensamblador a ficheros objeto en lenguaje máquina (formato COFF). Los ficheros fuente pueden contener instrucciones, directivas de ensamblador y directivas de macros. Se pueden utilizar directivas para controlar varios de los aspectos del proceso de ensamblado.
- El linker, combina los ficheros objeto en un solo módulo objeto ejecutable. Al crear el módulo objeto, permite la relocalización y resuelve referencias externas. El linker utiliza ficheros objeto con formato COFF creados por el ensamblador, además de aceptar archivos de librerías y módulos objeto creados previamente. Las directivas permiten combinar secciones de ficheros objeto, asignar secciones o símbolos a una dirección o rango de direcciones y definir o redefinir símbolos globales.

- El archivador: permite recoger un grupo de ficheros en uno. Se pueden agrupar en una sola librería macros, ficheros objeto o grupos de ficheros objeto.
- El programa que hace listados absolutos: acepta ficheros objeto como entrada y crea un fichero .abs como salida. Cuando se ensambla el fichero .abs produce un listado con la dirección absoluta en lugar de tener direcciones relativas.
- La utilidad de conversión hexadecimal: convierte los ficheros objeto (en formato COFF) en formatos para la programación de EPROM.

2.4.2. SECCIONES

El ensamblador y el linker crean ficheros objeto que pueden ser ejecutados por el TMS320C30 EVM. El formato en el que están estos ficheros objeto es el llamado COFF.

El COFF hace la programación modular más fácil, ya que hace pensar en términos de bloques de código y datos llamados secciones. Una sección es un bloque de código o datos que ocupa un espacio contiguo, en el mapa de memoria del C30. Cada sección de un fichero objeto está separada de las otras secciones del mismo fichero.

En cada sección existe un contador de sección de programa o SPC que cuenta cuantas palabras hay almacenadas.

Los ficheros objeto COFF siempre contienen tres secciones por defecto:

- .text : Usualmente contiene código ejecutable.
- .data : Usualmente contiene datos inicializados
- .bss: Usualmente reserva espacio para variables inicializadas

Además, el ensamblador y el linker permiten crear, nombrar y unir secciones con nombre propio, que se usan de forma similar a las secciones mencionadas anteriormente.

Se ha de distinguir las secciones inicializadas, que contienen código o datos con un valor definido, de las no inicializadas, que reservan espacio en memoria para datos que no tienen un valor definido.

El ensamblador especifica la sección en la que los datos o el código se encontrarán alojados, mediante el uso de una serie de directivas, que son diferentes para las secciones inicializadas y las no inicializadas.

El ensamblador agrupa los datos y código en secciones, siendo el linker el encargado de decidir en qué localización de memoria se mapeará cada una de las secciones.

2.4.2.1. Secciones no inicializadas

Las secciones no inicializadas reservan espacio en memoria del C30, sin definir el contenido actual en el fichero objeto, simplemente el espacio se reserva hasta que el C30 use este espacio durante la ejecución del programa.

Se utilizan dos directivas para reservar espacio en secciones no inicializadas: `.bss` y `.usect`.

La primera de las directivas `.bss`, reserva espacio en la sección `.bss` del mismo nombre. Su sintaxis es la siguiente:

`.bss` símbolo, tamaño

El símbolo es el nombre de la variable para la que se está reservando espacio (apunta a la primera palabra del conjunto de las reservadas). El tamaño representa el número de palabras que se va a reservar en la sección `.bss`.

La segunda de las directivas que se puede utilizar tiene la siguiente sintaxis:

Símbolo **`.usect`** “nombre_de_la_sección”, tamaño

Símbolo y tamaño representan la misma función que en la directiva anterior. El nombre de la sección es el nombre que el ensamblador asignará a la nueva sección definida.

Cuando el compilador encuentra una de estas directivas de compilación, reserva el espacio en la sección adecuada, pero no cambia en qué sección están los datos o el código siguientes. Es decir, estas directivas no marcan el final de una sección.

2.4.2.2. Secciones inicializadas

Las secciones inicializadas contienen código ejecutable o datos inicializados. El contenido de estas secciones es almacenado en el fichero objeto para posteriormente cargarlo en memoria del C30.

Cada sección inicializada es ubicable por separado y puede referenciar a símbolos definidos en otras secciones. El linker es el encargado de resolver las referencias a símbolos entre las diferentes secciones.

Cuando el compilador encuentra una directiva para inicializar una sección, automáticamente hace que todo el código y los datos siguientes se agrupen en orden en esta sección, finalizando por tanto la sección anterior. Se puede volver a introducir más instrucciones o datos en una sección ya utilizada, para lo cual basta con volver a poner la directiva de la nueva sección.

Existen tres directivas para las secciones inicializadas. La primera y más sencilla tiene la siguiente sintaxis:

.text

Esta directiva, hace que todo el código y los datos inicializados que aparezcan después se agrupen en la sección .text, que se mencionó anteriormente. Si no se especifica ninguna sección en el programa, esta es la sección que se toma por defecto. En la sección .text normalmente solo se incluye código de programa.

La segunda de las directivas hace que todos los datos y código siguientes se agrupen entorno a la sección .data. Normalmente en esta sección solamente se incluyen variables inicializadas. La sintaxis es la siguiente:

.data

La tercera directiva permite crear secciones con un nombre propio, pudiendo hacer el número de secciones que el programador crea oportuno. La sintaxis de la directiva es:

.sect “nombre”

El campo *nombre*, debe ir entre comillas.

2.4.3. ENSAMBLADOR

El ensamblador traduce ficheros fuente escritos en lenguaje ensamblador en ficheros objeto en formato COFF.

Los ficheros fuente contienen directivas del ensamblador, instrucciones del lenguaje ensamblador y directivas de macros.

2.4.3.1. Invocando al ensamblador

Para invocar al ensamblador después de haber ejecutado el fichero INIT, basta con ejecutar lo siguiente:

asm30 [*fichero_entrada*][*fichero_objeto*][*fichero_listado*]][*-opciones*]

Fichero_entrada, es el nombre del fichero fuente en lenguaje ensamblador. Si el nombre del fichero no incluye la extensión, por defecto se asume .asm. En caso de no proporcionar el nombre del fichero al ensamblador, este preguntará por un nombre.

Fichero_objeto, es el nombre del fichero objeto que crea el ensamblador. Si no se especifica la extensión se asume .obj. Si no se especifica el nombre del fichero objeto de salida, se utiliza el nombre del fichero de entrada con extensión .obj.

Fichero_listado, es un fichero opcional que el ensamblador puede crear. Si no se suministra el nombre, el ensamblador no crea este fichero a menos que se introduzca la opción -l, en cuyo caso el nombre del fichero será el mismo que el fichero de entrada con la extensión .lst.

2.4.3.2. Formato de una línea en ensamblador

El programa fuente en lenguaje ensamblador consta de una serie de líneas que contienen directivas de ensamblador, instrucciones, directivas de macros y comentarios.

Las líneas pueden ser tan largas como se desee, pero el ensamblador solamente procesa hasta los 200 primeros caracteres, ignorando el resto tras dar un mensaje de peligro.

El formato de la línea es el siguiente:

[*etiqueta*][:] *nemotécnico* [*lista de operandos*] [*;comentarios*]

Las líneas deben seguir las siguientes reglas:

- Todas las líneas deben comenzar con una etiqueta, un espacio en blanco, un asterisco o un punto y coma.
- Las etiquetas son opcionales, pero si se usan deben comenzar en la primera columna.
- Cada uno de los campos debe estar separado entre sí por uno o más espacios en blanco. El carácter tabulador es considerado como un espacio.

- Los comentarios son opcionales. Los que comienzan en la primera columna pueden comenzar por un asterisco o un punto y coma, pero los comentarios que comiencen en otra columna deben comenzar con un punto y coma.

Las etiquetas pueden contener hasta 32 caracteres alfanuméricos (A-Z,a-z,0-9, _ y \$). En las etiquetas se distingue entre mayúsculas y minúsculas. El primer carácter no puede ser un número. Las etiquetas pueden aparecer seguidas del carácter dos puntos ‘:’. Este carácter no es tratado como parte de la etiqueta.

El campo nemotécnico no puede comenzar en la primera columna, ya que sino sería interpretado como una etiqueta. Este campo puede contener:

- Instrucciones máquina.
- Directivas del ensamblador.
- Directivas de macros.
- Invocaciones de macros.

En caso de que el campo *lista de operandos* tenga más de un operando, éstos se separan entre sí por comas.

2.4.3.3. Constantes

El ensamblador soporta siete tipos de constantes que se podrán utilizar en diferentes momentos y que internamente se representan en el ensamblador de 32 bits:

- Enteros binarios: Son conjuntos de 32 bits binarios (0 ó 1) seguidos del sufijo B o b.
- Enteros octales: Son un conjunto de 11 dígitos octales (0-7) seguidos del sufijo q o Q.
- Enteros decimales: Es un conjunto de dígitos decimales, cuyo rango está comprendido entre 4294967295 y 0 (enteros sin signo) y 2147482647 y -2147482647 (enteros con signo).
- Enteros hexadecimales: Es un conjunto de ocho dígitos hexadecimales seguidos del sufijo H o h. Una constante hexadecimal debe comenzar con un valor decimal.
- Caracteres: Una constante carácter es un conjunto de 1 a 4 caracteres encerrados entre comillas simples. Cada uno de los caracteres es representado mediante ocho bits en formato ASCII.

- Punto flotante: Es un conjunto de dígitos decimales, seguidos por un punto decimal opcional, una fracción y un exponente. La sintaxis es:

$$[+/-] [nnn] . [nnn[E/e[+/-] nnn]]$$

Donde nnn es una cadena de dígitos decimales. El exponente indica una potencia de 10.

- En tiempo de ensamblado: Se usan la directiva .set para asignar valores constantes a un símbolo en tiempo de ensamblado. Cuando una constante es entera o punto flotante no puede ser usada de otra forma. (Ejemplo: sim .set R0; LDF 10, sim; Esto cargará 10 en R0)

2.4.3.4. Símbolos

Los símbolos son usados como etiquetas, constantes y símbolos sustituidos. El nombre del símbolo es una cadena de hasta 32 caracteres alfanuméricos que pueden incluir los siguientes caracteres:

- Letras mayúsculas comprendidas entre la A y la Z.
- Letras minúsculas comprendidas entre la a y la z.
- Números del 0 al 9.
- Caracteres '\$' y '_'.

El primer carácter no puede ser un número.

Tipos de símbolos:

- Etiquetas: representan direcciones simbólicas que están asociadas a localizaciones del programa. Ejemplo:

etiqueta2 nop;

b etiqueta2;

- Constantes: a un símbolo puede asignársele valores constantes mediante la directiva .set. De esta forma se le asigna a un nombre con significado, un valor, para hacer más fácil de comprender el código ensamblador. Las constantes simbólicas no pueden ser redefinidas. Ejemplo:

K .set 1024 ; definición de una constante

.bss array, 10*K;

- Constantes simbólicas: el ensamblador tiene predefinidos algunos símbolos como estos:
 - El símbolo \$, representa el valor actual del contador de programa de la sección.
 - Se incluyen los siguientes símbolos de registros: AR0-AR7, BK, DP, IR0, IR1, PC, IE, RE, RC, RS, IOF, R0-R7, SP, ST.
 - Símbolos de versiones del hardware, que en caso de realizar desarrollos de aplicaciones para el EVM solamente no son necesarios.
 - Símbolos del modelo de compilación del C, que son constantes con valores 0 o 1.
- Símbolos sustituidos: a un símbolo se le puede asignar una cadena de caracteres, que permite que cuando el ensamblador encuentre una sustitución de un símbolo, el valor de la cadena sea cambiada por el nombre del símbolo. Los símbolos sí que pueden redefinirse. Ejemplo:

.asg "ar3",FP ; Puntero a la trama

.asg "*-FP(2)",PARAM1

LDI PARAM1, R0 ; La operación será LDI *-ar3(2), R0

2.4.3.5. Directivas del ensamblador

En las siguientes tablas se muestran las directivas con su sintaxis y una breve descripción. Para más información, se debe consultar [Texas, 2].

- Directivas que definen secciones:
 - 1- **.bss** *símbolo*, *tamaño* [, *indicador_bloqueo*]. Reserva el número de palabras especificado por el tamaño en la sección especificada .bss.
 - 2- **.data**. Ensambla en la sección de datos inicializados .data.
 - 3- **.sect** "*nombre*". Ensambla en una sección inicializada con el nombre especificado.
 - 4- **.text**. Ensambla en la sección de código .text.

- 5- símbolo **.usect** "*nombre*", *tamaño* [, *indicador_bloqueo*]. Reserva el número de palabras enumerado por el tamaño en la sección sin inicializar. El nombre de la sección también es especificado.
- Directivas que inicializan constantes (memoria y datos):
 - 1- **.byte** *valor1* [, ..., *valorn*]. Inicializa uno o más bytes consecutivos en la sección actual.
 - 2- **.double** *valor1* [, ..., *valorn*]. Inicializa uno o más constantes en punto flotante de precisión simple de 32 bits.
 - 3- **.field** *valor* [, *tamaño en bits*]. Inicializa un campo de longitud variable.
 - 4- **.float** *valor1* [, ..., *valorn*]. Inicializa uno o más constantes en punto flotante de precisión simple de 32 bits.
 - 5- **.hword** *valor1* [, ..., *valorn*]. Inicializa uno o más valores de 16 bits.
 - 6- **.ieee** *valor1* [, ..., *valorn*]. Inicializa una o más constantes punto flotante en formato IEEE de 32 bits.
 - 7- **.int** *valor1* [, ..., *valorn*]. Inicializa uno o más enteros de 32 bits.
 - 8- **.ldouble** *valor*. Inicializa una constante de punto flotante de precisión extendida (40 bits).
 - 9- **.long** *valor1* [, ..., *valorn*]. Inicializa uno o más enteros de 32 bits.
 - 10- **.space** *tamaño de bits*. Reserva los bits especificados en la directiva, haciendo que la etiqueta que lo acompaña apunte al inicio del espacio reservado.
 - 11- **.string** "*cadena1*" [, ..., "*cadena_n*"]. Inicializa una o más cadenas de texto.
 - 12- **.word** *valor1* [, ..., *valorn*]. Inicializa uno o más enteros de 16 bits.
 - Directivas que alinean el contador de la sección de programa (SPC):
 - 1- **.align**. Alinea el SPC en el comienzo de la página siguiente (32 palabras).
 - 2- **.even**. Alinea el SPC en el comienzo de la siguiente palabra.
 - 3- **.drlist**. Habilita el listado de todas las directivas de las líneas (por defecto).
 - 4- **.drnolist**. Inhibe el listado de ciertas líneas de directivas.

- 5- **.fclist**. Permite el que no se liste un bloque de código de forma condicional (por defecto).
 - 6- **.fcnolist**. Inhibe el listado de bloques de forma condicional.
 - 7- **.length** *longitud de la página*. Configura la longitud de la página del listado fuente.
 - 8- **.list**. Retoma el listado del fichero fuente.
 - 9- **.mlist**. Permite el listado de macros y bloques repetitivos (defecto).
 - 10- **.mnolist**. Inhibe el estado de macros y bloques repetitivos (defecto).
 - 11- **.nolist**. Detiene el listado del fichero fuente.
 - 12- **.option** {B/D/F/L/M/T/X}. Selecciona las opciones de listado del fichero de salida.
 - 13- **.page**. Hace que el listado siguiente comience en una nueva página.
 - 14- **.sslist**. Permite expandir el listado de los símbolos sustituidos.
 - 15- **.ssnolist**. Inhibe la expansión del listado de los símbolos sustituidos.
 - 16- **.title** "título". Imprime un título en la cabecera de las páginas del listado.
 - 17- **.width**. *ancho de programa*. Configura el ancho de página del fichero fuente.
- Directivas que referencian a otros ficheros:
 - 1- **.copy** ["*nombre del fichero*"]. Incluye el código de otro fichero.
 - 2- **.def** *símbolo1* [..., *símbolon*]. Identifica uno o más símbolos que están definidos en el módulo actual y usados en otros módulos.
 - 3- **.global** *símbolo1* [..., *símbolon*]. Identifica uno o más símbolos globales (externos).
 - 4- **.include** ["*nombre del fichero*"]. Incluye el código de otro fichero.
 - 5- **.mlib** ["*nombre del fichero*"]. Librería de definición de macros.
 - 6- **.ref** *símbolo1* [..., *símbolon*]. Identifica uno o más símbolos que están usados en el módulo actual pero definidos en otro módulo.

- Directivas condicionales de ensamblador:

- 1- **.break** [*expresión*]. Fin del ensamblado dentro del .loop si la condición es cierta.
- 2- **.else**. Ensambla el bloque de código si la condición del .if es falsa.
- 3- **.elseif** *expresión*. Ensambla el bloque de código si la condición del .if es falsa y la condición del .elseif es cierta.
- 4- **.endif**. Fin del bloque de código .if.
- 5- **.endloop**. Fin de bloque de código .loop.
- 6- **.if** *expresión*. Ensambla el bloque de código si la condición es cierta.
- 7- **.loop** [*expresión*]. Comienza el ensamblado de un bloque de código repetitivo.
- 8- **.asg** [*“cadena de caracteres”*], *símbolo sustituido*. Asigna una cadena de caracteres a un símbolo sustituido.
- 9- **.endstruct**. Fin de la definición de la estructura.
- 10- **.eval** *expresión*, *símbolo sustituido*. Realiza la sustitución sobre un símbolo numérico.
- 11- **.label** “*símbolo*”. Define una etiqueta durante la carga del programa en una sección.
- 12- **.set**. Iguala un valor a un símbolo.
- 13- **.struct**. Comienzo de la definición de la estructura.
- 14- **.tag**. Asigna los atributos de la estructura a la etiqueta.

- Directivas varias:

- 1- **.emsg** *cadena*. Envía un mensaje de error definido por el usuario a los dispositivos de salida.
- 2- **.end**. Fin del programa.
- 3- **.mmsg** *cadena*. Envía mensajes definidos por el usuario al dispositivo de salida.
- 4- **.regalias**. Usa el nombre de registros Fn en lugar de Rn.
- 5- **.version** *generación # número*. Configura la versión del procesador.
- 6- **.wmsg** *cadena*. Envía mensajes de peligro definidos por el usuario.

2.4.4. *EL LINKER*

El linker crea módulos ejecutables combinando ficheros objeto COFF. Cuando el linker une los ficheros objeto realiza las siguientes tareas:

- Configura la dirección de memoria en el EVM donde las diferentes secciones serán almacenadas.
- Relocaliza los símbolos y secciones para asignarles una dirección final.
- Resuelve las referencias externas indefinidas entre los ficheros de entrada.

El linker soporta un lenguaje de comandos que controla la configuración de memoria del EVM, la definición de las secciones y el tipo de memoria de las direcciones. El lenguaje soporta la asignación y evaluación de expresiones e incorpora dos poderosas directivas llamadas *MEMORY* y *SECTIONS*.

2.4.4.1. Invocando al linker

La sintaxis general para invocar al linker:

Lnk30 [-opciones] *archivo1..archivon*

Las opciones pueden aparecer en cualquier lugar de la línea de comandos o en un fichero de comandos.

Los archivos, pueden ser ficheros objeto, ficheros de comandos del linker o librerías, siendo la extensión que asume el linker por defecto *.obj*.

Al linker se le puede pasar toda la información en la línea de comandos, o interrogará por los parámetros que le falten.

Uno de los archivos que se le pasan al linker en la línea de comandos puede ser un archivo de comandos que normalmente tiene la extensión *.cmd*.

2.4.4.2. Ficheros de comando del linker

Los archivos de comando permiten al linker que se le especifique información en un fichero. Estos ficheros son útiles ya que permiten utilizar las directivas *MEMORY* y *SECTIONS* para personalizar la aplicación.

Los archivos de comando contienen uno o más de los siguientes campos:

- Archivos de entrada, que especifican ficheros objeto, librerías u otros ficheros de comando. En caso de invocar a otro fichero de comando, debe ser la última sentencia, ya que el linker no devuelve el control después de procesar este archivo de comando.
- Opciones del linker.
- Sentencias de asignación, que definen y asignan valores a símbolos globales.

Si el linker reconoce un fichero como fichero objeto, lo linka. En caso contrario asume que es un archivo de comandos a procesar.

A continuación se muestra el contenido de un fichero de comando de ejemplo para el EVM:

```
samp.obj                /* Ficheros de entrada */
sysinit.obj

-e start                /* Punto de entrada al código */
-o samp.out             /* Fichero de salida */

MEMORY                  /* Configura el mapa de memoria */
{
    INT_V : origin = 0x000000, length = 0x40
    SRAM  : origin = 0x000040, length = 0x3FC0
    RAM0  : origin = 0x809800, length = 0x400
    RAM1  : origin = 0x809C00, length = 0x400
}

SECTIONS                /* Especificación de la localización de las secciones de memoria */
{
    vectors: {} > INT_V
    comdata: {} > SRAM
    .text   : {} > SRAM
    .bss    : {} > RAM0
    stack   : {} > RAM1
}
```

- La directiva *MEMORY*:

El linker determina en qué posición de memoria debe mapearse cada sección. Para poder realizar esta tarea, el linker ha de tener alguna forma de conocer el mapa de memoria del sistema de desarrollo, en este caso el EVM. Para este fin se utiliza la

directiva *MEMORY*, que permite definir la memoria del sistema de desarrollo, así como el tipo de memoria que se encuentra en cada zona.

Esta directiva asigna un nombre a cada rango de memoria, definiendo su dirección de comienzo y su longitud, entre otros parámetros.

La sintaxis general de la directiva es:

MEMORY

```
{  
    [PAGE 0:] nombre1[(atributos)]:origin= constante, longitud= constante  
    [PAGE n:] nombren[(atributos)]:origin= constante, longitud= constante  
}
```

La palabra reservada *PAGE* identifica un espacio de direcciones completamente independiente de otro. Se pueden reservar hasta 255 espacios independientes. En caso de no especificarse, el linker asigna uno de forma automática.

El *nombre* de un espacio de memoria puede tener entre 1 y 8 caracteres (A-Z, a-z, \$, ., -). Estos nombres no tienen ningún significado especial, solamente identifican los rangos de memoria. En caso de que dos rangos de memoria tengan el mismo nombre, se distinguirán por el número del campo *PAGE*.

Los *atributos* especifican las características de la memoria a la que se asocian. En caso de que aparezcan, deben ponerse juntos sin separación entre ellos. En caso de no utilizarse no se ponen restricciones al uso de la memoria, suponiéndose que tiene los cuatro atributos. Los atributos posibles son:

- R, que especifica que se puede leer de la memoria.
- W, que especifica que se puede escribir en memoria.
- X, que especifica que la memoria puede contener código ejecutable.
- I, que especifica que la memoria puede ser inicializada.

El campo *origin*, especifica la dirección de comienzo del rango de memoria. En lugar de la palabra reservada *origin*, se puede utilizar la palabra *len* o *l*.

- La directiva *SECTIONS*:

La directiva *SECTIONS* especifica al linker cómo debe combinar las secciones de los ficheros de entrada, y en qué zona de memoria debe alojarlos. Así, esta directiva:

- Describe cómo se han de combinar las secciones de entrada en las secciones de salida.
- Define las secciones de salida en el programa ejecutable.
- Especifica en qué zona de memoria se a emplazar cada sección.
- Permiten renombrar las secciones de salida.

En caso de no utilizarse esta directiva, el linker tomará las decisiones pertinentes según un algoritmo diseñado por TI, que se puede estudiar con detalle en el libro: [Texas, 2].

La sintaxis de esta directiva es la siguiente:

SECTIONS

```
{  
    nombre:[propiedad, propiedad,...]  
    nombre:[propiedad, propiedad,...]  
    nombre:[propiedad, propiedad,...]  
}
```

Cada una de las secciones de entrada que se especifiquen corresponderán con las definidas en los ficheros fuente realizados en ensamblador, por lo que se deben definir cada una de las secciones creadas en los códigos fuente, a menos que se desee que el linker los mapee según su propio criterio.

La especificación de cada sección, que comienza con el nombre, define una sección de salida. Después del nombre de la sección aparece una lista de propiedades que definen el contenido de la sección y cómo es mapeada.

Las propiedades se pueden separar entre sí, de forma opcional, por comas. Las propiedades posibles son:

LOAD ALLOCATION. Define en qué posición de la memoria se almacenará la sección. La sintaxis de esta opción puede ser de tres formas diferentes:

load = *localización*

localización

>localización

RUN ALLOCATION. Define en qué posición de memoria se ejecutará el código de la sección. El código puede almacenarse en una zona determinada de memoria y ejecutarse en otra zona diferente. El objetivo es que el código se almacene en una zona de memoria lenta (más barata) y se ejecute en otra más rápida, que normalmente es más escasa. Existen dos opciones de sintaxis:

run = *localización*

run > *localización*

INPUT SECTIONS. Define el nombre de la sección de entrada, en caso de que el nombre de la entrada sea diferente del nombre de la sección de salida. La sintaxis es:

{nombres de las secciones de entrada}

SECTION TYPE. Define los indicadores para tipos de secciones espaciales. La sintaxis es:

type = **COPY**

type = **DSECT**

type = **NOLOAD**

FILL VALUE. Define el valor usado para rellenar los huecos sin inicializar. La sintaxis es:

fill = *valor*

nombre: ...{...} = **valor**