Tema 3
Arquitectura de los Procesadores Digitales de Señal

3.1. Introducción

Los Procesadores Digitales de Señal son microprocesadores diseñados para implementar algoritmos típicos del procesado digital de señales, especialmente en aplicaciones de tiempo real. En procesado en tiempo real, el parámetro principal es la cantidad de cálculo que puede realizarse en el tiempo intermuestral, si se trata de un algoritmo que produzca una muestra de salida para cada muestra de entrada (por ejemplo, un filtro digital), o en el tiempo de procesado del *buffer* para algoritmos que trabajan con grupos de muestras (por ejemplo, el cálculo de la FFT).

Respecto al tipo de operaciones, suelen implicar una gran cantidad de acceso a datos, son altamente repetitivas, y usualmente incluyen multiplicaciones. La 'primitiva' más común en Procesado Digital es:

$$Y = A \cdot X + B$$

que se utiliza en filtrado digital, convolución, transformada rápida de Fourier, etc. Esta ecuación implica el acceso a dos datos, una multiplicación y una suma. Por tanto, en procesado digital en tiempo real no sólo es importante la velocidad de ejecución de la operación, sino que un eficiente flujo de datos es crítico.

Los primeros sistemas DSP utilizaban componentes estándar para construir desplazadores, sumadores y multiplicadores. El diseño de estos últimos avanzó con la utilización de técnicas *pipe-line*, y los primeros multiplicadores que operaban en un único ciclo de reloj se implementaron en los primeros ‘70. con componentes ECL estándar.

En 1971, Lincon Laboratories construyó el FDP (Fast Digital Processor), con un ciclo de multiplicación de 600 ns pero que necesitaba 10.000 circuitos integrados. Tenía también los problemas inherentes a la utilización de una arquitectura von Neumann. Su sucesor, el Lincoln LSP/2, que ya utilizaba una arquitectura Harvard, era 4 veces más rápido con la tercera parte de circuitos integrados que su predecesor. Hacia mitad de los ‘70, otros grupos habían logrado

computadores para procesado digital de señal con ciclos de multiplicación del orden de 200 nseg, pero tan caros que no resultaban comercialmente viables. Se disponía, pues, de la estructura básica, pero sólo cuando la tecnología de fabricación de integrados se hubo desarrollado lo suficiente comenzaron a aparecer dispositivos operativos.

A principios de los 80 se presentaron los primeros dispositivos DSP. Todos ellos utilizaban arquitectura Harvard para separar memoria de datos y programa. Esto permitía que una instrucción y un dato pudieran direccionarse y accederse al mismo tiempo. No obstante, la verdadera aportación de la arquitectura Harvard era permitir el acceso a dos espacios de memoria simultáneamente, pudiendo así obtener los dos términos del producto incluidos en la multiplicación de la ecuación anterior. Por ello, los DSP utilizan una arquitectura Harvard modificada, en la cual el espacio de memoria de programa contiene también datos (por ejemplo, en el caso de un filtro digital, los coeficientes del filtro suelen localizarse en memoria de programa, mientras que las muestras de la señal se localizan en memoria de datos).

Los primeros DSP utilizaban un formato de representación de datos en coma fija (en una sección posterior se comentarán los formatos de representación). A partir de la segunda mitad de la década de los '80 aparecen los DSP de coma flotante. En la actualidad, existe una gran cantidad de familias de procesadores DSP desarrolladas por diversos fabricantes, algunos de los cuales se muestran en el Apéndice A.2.

El objetivo de ejecución de algoritmos de Procesado Digital en tiempo real condiciona la concepción de los DSP, intentando optimizar dos características básicas: precisión y velocidad. Algunos factores que influyen en la precisión son:

- La anchura del bus, que está relacionada con la precisión de los datos y resultados en memoria. En el caso de coma fija, suelen ser de 16 *bit*s, mientras que para coma flotante, la anchura es de 32 *bit*s.

- La utilización de aritmética expandida, para evitar desbordamientos en los cálculos intermedios y finales.

- El uso de escalado previo de los datos, que permite modificar el rango dinámico de la señal a tratar evitando también desbordamientos.

Respecto a la velocidad, los factores más influyentes son:

- Cálculo eficiente de productos y sumas.

- Debido al gran flujo de datos implicado en las operaciones a procesar, es deseable disponer de un generador de direcciones de datos que permita acceder a los mismos sin utilizar tiempo de la ALU de cálculo.

- También es interesante, para no utilizar tiempo de las unidades de computación, disponer de una unidad que determine automáticamente la secuencia de ejecución de instrucciones, encargándose del control de saltos y bucles.

La inclusión de todos estos factores en la definición de la arquitectura DSP lleva a la utilización de diversas unidades funcionales, que deben poder trabajar con un elevado grado de paralelismo. En una sección posterior se comentarán estos componentes de la arquitectura genérica de un DSP.

Además de las unidades básicas apuntadas anteriormente, es frecuente que los DSP incluyan en el mismo chip periféricos tales como puertos serie o paralelo, temporizadores, módulos de memoria, etc., motivado por la gran interactividad que suelen presentar los algoritmos de procesado digital (adquisición, almacenamiento y transmisión de datos). La inclusión de estos módulos mejora la velocidad de acceso del procesador a la vez que reduce costes.

3.2. Tipos de procesadores DSP

Existen diversas alternativas a la implementación de algoritmos de procesado digital. Las más usuales son:

- Microprocesadores de aplicación general. En la medida en que este tipo de procesadores ha ido adquiriendo mayor potencia, se han convertido en una opción para la implementación de aplicaciones de baja o moderada carga computacional. Como alternativa resultan interesantes ya que muchos productos incorporan microprocesadores, por lo que ejecutar *software* de procesado digital implica ventajas añadidas sin necesidad de modificar el *hardware* del sistema. Por otra parte, en la medida en que las aplicaciones del procesado digital se van generalizando (comunicaciones, multimedia, etc.), los fabricantes de procesadores de uso general han ido incorporando en su arquitectura elementos propios de los DSP, que los hacen más adecuados para este tipo de procesado.

- La utilización de computadores equipados con microprocesadores de aplicación general para procesado digital es una extensión de la aproximación anterior. Los soportes más utilizados son PC o *workstations,* compartiendo los campos de aplicación y limitaciones ya comentados.

- El desarrollo de *hardware* específico para una determinada aplicación es potencialmente la aproximación más eficiente, ya que se utiliza un diseño totalmente orientado que optimiza el comportamiento del circuito, eliminando aquellas partes presentes en dispositivos genéricos (tanto DSPs como otros microprocesadores) no utilizadas. Además, para volúmenes de producción altos, los circuitos a medida resultan más baratos que los genéricos. Por el contrario, los gastos y tiempos de desarrollo en este tipo de circuitos suele ser más elevado.

- Por último, los DSP representan una alternativa a medio camino entre la implementación de aplicaciones sobre *hardware* no optimizado para procesado digital, como puede ser el caso de microprocesadores o computadores de propósito general, y el *hardware* altamente especializado que representan los circuitos de aplicación específica. Actualmente cubren un segmento de aplicaciones con media o alta complejidad y frecuencia de muestreo. Cuando ambos parámetros son bajos, se puede optar por una aproximación no especializada, mientras que para valores muy altos puede ser necesario el desarrollo de un circuito específico.

Centrándonos en la aproximación DSP, existen diversos formatos en los que podemos encontrar este tipo de procesadores, dependiendo de la aplicación para la que vayan a ser utilizados:

- El más habitual es el procesador encapsulado en un chip, que aparece incorporado a diseños sobre placas de circuito impreso.

- Los módulos multichip (MCM) incorporan en el mismo encapsulado varios circuitos integrados, consiguiéndose así una especie de *superchip* que permite una mayor densidad de integración del sistema total. Los MCM pueden incluir varios procesadores (DSP y de otro tipo), así como memoria, etc.

- Una generalización de esto es integrar varios procesadores en el mismo circuito integrado, mejorando las características del sistema y reduciendo el consumo.

- Otra estrategia opuesta a las comentadas es distribuir el procesador en varios encapsulados (*chip sets*). Esto puede ser necesario si el procesador es muy complejo o se

necesita un gran número de *pines* de entrada/salida, proporcionándose además una flexibilidad suplementaria en la combinación de unidades funcionales del procesador.

- Los núcleos DSP (*DSP core*) son procesadores prediseñados que los fabricantes proporcionan para ser integrados en chips específicos junto al resto de circuito de desarrollo propio. Permite una solución intermedia entre la utilización de un DSP convencional y un circuito específico, mejorando los costes y tiempos de desarrollo del último y aprovechando la flexibilidad del primero. Se obtienen así dispositivos basados en DSP estándar orientados a aplicaciones específicas, como comunicaciones, multimedia, etc. Estos núcleos, dependiendo del fabricante, pueden incorporar sólo el procesador o también memoria y periféricos, y pueden utilizarse insertándolos en el diseño del ASIC como un módulo más, sin posibilidades de modificación, o pudiendo cambiar determinadas características del mismo orientadas a la aplicación, por ejemplo añadiendo una nueva unidad funcional. En este último caso, se denominan procesadores DSP a medida (*customizable DSP processor*).

- Por último, para aplicaciones en las que se requiera una elevada potencia de cálculo, suele ser necesaria una aproximación basada en multiprocesadores, combinando varios DSP que compartan las tareas del programa total. Los fabricantes suelen disponer de familias especialmente orientadas a trabajo en modo multiproceso, con características específicas tales como múltiples *buses* externos, lógica de control de *bus* o puertos específicos de comunicaciones entre procesadores.

3.3. Aritmética y representaciones numéricas

Como se ha comentado, el formato de representación de los datos en memoria o durante los cálculos intermedios influye en la precisión de los resultados obtenidos por el procesador. Por tanto, en esta sección haremos una breve revisión de los conceptos implicados y del impacto que tienen en la arquitectura del DSP.

Formatos de coma fija y coma flotante

En coma fija, la posición del punto decimal está predeterminada. El formato viene definido por una función F(b,c), donde b y c corresponden al número de dígitos que representan la parte entera y la parte decimal del número.

Por ejemplo, podemos decidir que el formato sea entero (sin parte decimal: c=0). En este caso, no obstante, si queremos operar con valores que sí tengan parte decimal, nos veremos obligados a pre-redondear antes de operar. Por ejemplo, si queremos sumar:

$$2.30 + 1.45 = 3.75$$

en formato entero tendríamos: 2 + 1 = 3, por lo que el error de cuantización, es decir, el error introducido al convertir un número de un formato de representación más largo a otro más corto, es en este caso de 0.30, 0.45 y 0.75 respectivamente.

Si en lugar de reservar todos los dígitos de la palabra de datos para la parte entera, dedicamos algunos para el campo fraccionario, podremos obtener mejor precisión. Por ejemplo, para una longitud de palabra de 3 dígitos, podemos reservar 1 para la parte entera y 2 para la decimal. De este modo, sí que podríamos representar los valores propuestos en el ejemplo anterior, y la precisión del resultado sería mayor que si pre-redondeamos para operación entera, con el mismo número de dígitos por palabra. Por tanto, la elección del formato incide en la precisión de los cálculos.

Un formato típico en la realización de filtros digitales en aritmética de coma fija, que requiere que los coeficientes y variables se representen en un rango fijo, es el Q15, que es una representación en complemento a dos para palabras de 16 *bit*s de longitud, en el que el *bit* más significativo (MSB) es el *bit* de signo (0:+, 1:-). Los 15 *bit*s restantes representan a un número fraccionario entre +0.9999... y -1.0000... La siguiente figura muestra esta representación, donde se supone que la posición del punto decimal es fijo y situado inmediatamente a continuación del *bit* de signo.

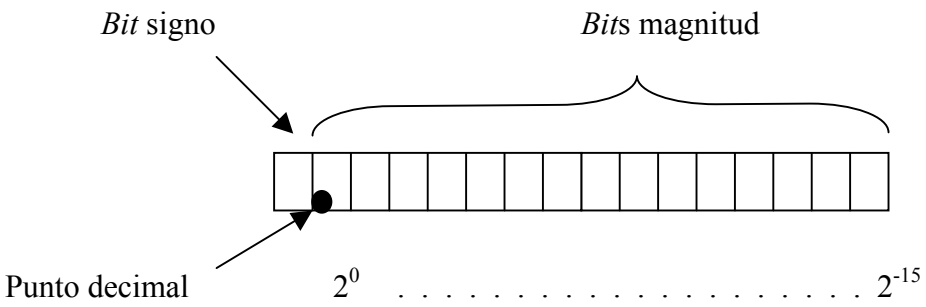


Figura 3.1. Representación en formato Q15.

La precisión máxima, en *bit*s, viene dada por la expresión:

$$\Pr ecision\_max = \log_2 \frac{|valor\_max|}{|error\_cuantizac\_max|}$$

Para el formato Q15, el máximo error de cuantización es 2^{-15}, y el valor máximo que puede representarse es –1. No obstante, cuando representamos un número menor que 1, la precisión disminuye, ya que el error de cuantización permanece constante. Para mantenerla constante, es necesario escalar el dato.

Cuando se utiliza formato Q15, por ejemplo en la implementación de un filtro digital, puede darse el caso de que el valor de algún coeficiente sea mayor que la unidad. Si así ocurre, pueden convertirse según dos convenciones:

1. Representarlo como un valor entero más uno fraccionario, y sumar ambos (por ejemplo, 1.4x=1x+0.4x).
2. Dividirlo por potencias de dos y posteriormente multiplicar (1.4x = (0.7x) 2). Las multiplicaciones por potencias de dos se reducen a desplazamientos a izquierdas.

La notación en coma flotante, conocida también como notación científica, representa:

$$y = m \cdot b^E$$

donde b es la base (10 para decimal, 2 para binario, etc), E es el exponente y m la mantisa.

Los números en coma flotante se almacenan en formato binario con algunos *bit*s para el exponente y el resto para la mantisa. En general, un número en coma flotante tendrá N *bit*s asignados al exponente y M *bit*s asignados a la mantisa, escribiéndose como una palabra de (N,M) *bit*s (el exponente primero). Por ejemplo, para una palabra de 16 *bit*s, podría ser:

(exponente: 4 *bit*s) (mantisa: 12 *bit*s) ≡ (4E,12M)

Para el caso anterior, tendríamos pues un número de *bit*s, con un rango dinámico (valor máximo menos valor mínimo representables) de:

	MANTISA	EXPONENTE	
Mayor:	1111 1111 1111	1111	$= (2^{12} - 1) * 2^{15}$
Menor:	0000 0000 0001	0000	$= 1 * 2^0$

siendo $E_{máx} = 2^{N-1} = 2^{(1111)-1} = 2^{16-1} = 2^{15}$.

Aproximadamente, el rango dinámico lo fija el número de *bit*s asignados al exponente, y la precisión los asignados a la mantisa. La precisión es menor que en el caso de coma fija, pero el rango dinámico es apreciablemente mayor. Ambos parámetros varían con la longitud de la palabra:

LONGITUD DE PALABRA	16	32	64
COMA FIJA			
Rango dinámico	$6.6x10^4$	$4.3x10^9$	$1.8x10^{19}$
Precisión	>4 díg.	>9 díg.	>19 díg.
COMA FLOTANTE	(4E,12M)	(8E,24M)	(11E,53M)
Rango dinámico	$3.3x10^4$	$5.8x10^{76}$	10^{1419}
Precisión	>3 díg.	>7 díg.	>15 díg.

Por ejemplo, para el caso de 32 *bit*s, en coma flotante se pierden 2 dígitos de precisión pero se gana 10^{67} de rango dinámico respecto del caso de coma fija. Con palabras de 16 *bit*s, se obtiene más rango dinámico y precisión con coma fija, por lo que el formato de coma flotante se utiliza para palabras de 32 o más *bit*s.

Las proporciones de rango dinámico y precisión dependen del reparto de *bit*s entre exponente y mantisa. Para una palabra de 32 *bit*s, tendremos:

Formato	Rango Dinamico	Precision
(8E,24M)	$2^{255} = 5x10^{76}$	>7 dígitos
(11E,21M)	$2^{2047} = 10^{1419}$	>6 dígitos

La precisión no ha cambiado apenas mientras que se ha incrementado de forma importante el rango dinámico. Existen estándares para la elección del formato, como el IEEE-754, desarrollado por el Instituto de Ingenieros Eléctricos y Electrónicos americano (IEEE) en 1985, que fija la representación de datos en coma flotante así como normas para la aritmética utilizada. Algunos procesadores DSP (Motorola DSP96002, Analog Devices ADSP-210xx) incorporan soporte *hardware* para este estándar.

En general, el formato de coma flotante facilita la programación, ya que es menos probable que se produzca desbordamiento, y no suelen escalarse los datos como ocurre en coma

fija. Por contra, los procesadores de coma flotante suelen ser más caros y en ocasiones menos rápidos que los de coma fija. Como conclusión de las consideraciones expuestas en este apartado, existen procesadores DSP comerciales tanto en formato de representación de coma fija como flotante. En coma fija, la longitud de palabra más usual es la de 16 *bit*s, aunque algunos fabricantes tienen dispositivos de 20 o 24 *bit*s. En coma flotante, con longitud de palabra de 32 *bit*s, podemos distinguir procesadores que implementan el estándar IEEE-754 u otros tipos de representación.

Aritmética con y sin signo

Las operaciones aritméticas implementadas por la ALU pueden realizarse entre operandos con o sin signo, e incluso mixtos (uno con y otro sin signo). El signo suele codificarse en el *bit* más significativo del operando (0: positivo, 1: negativo). En función del tipo de instrucción a ejecutar, la Unidad de Control supervisa el funcionamiento del operador para tener en cuenta el tipo de aritmética.

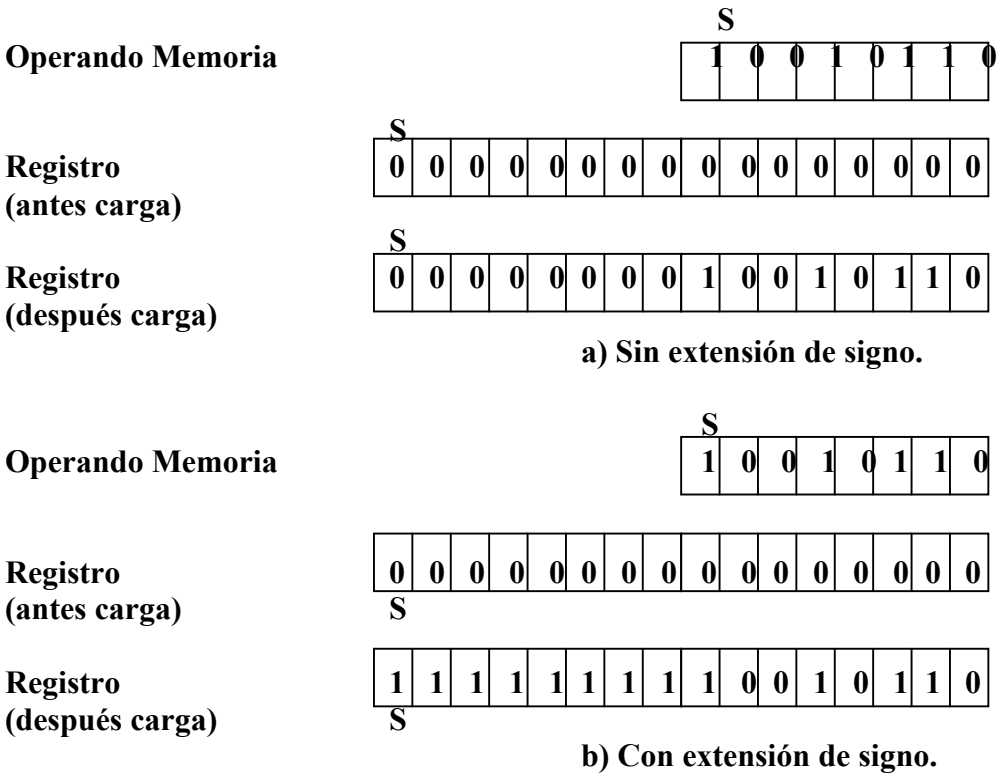


Figura 3.2. Modos sin extensión (a) y con extensión (b) de signo.

En el caso de aritmética con signo, es frecuente que un dato de m *bit*s se cargue en un registro de n *bit*s, donde m<n. Es, por ejemplo, el caso de un dato en memoria (supongámosla de 16 *bit*s) cargado en un registro interno de la CPU (de 32 *bit*s). En este caso, para mantener la consistencia de signo, se realiza la carga con **extensión de signo**, copiándose el *bit* de signo original hasta alcanzar el *bit* más significativo del nuevo registro. Sin extensión, el signo del operando cambia durante el proceso de carga, tal y como se muestra en la Figura 3.2.

Desbordamiento

El desbordamiento ocurre cuando un número excede el máximo valor que puede representarse con esa longitud de palabra, y produce un error importante puesto que el valor máximo súbitamente pasa a cero (aritmética sin signo) o cambia de signo (complemento a dos):

(11..11) + 1 = (00..00) + OVFL (sin signo)
(01..11) + 1 = (11..11) (complemento a 2)

Las técnicas más usuales para evitar el desbordamiento son escalar los datos o utilizar aritmética saturada. En el primer caso, el programador debe escalar los operandos antes de realizar operaciones que puedan producir desbordamiento. Esto reduce la precisión con que se representa la señal, pudiendo incidir en el resultado.

La opción de aritmética saturada simula la condición de máximo de escala en circuitos electrónicos. En este caso, cuando se produce desbordamiento, el valor en el registro adopta el valor máximo posible manteniendo el signo. Por ejemplo, para una palabra de 9 *bits* (1 bit de signo; magnitud máxima =255), supongamos la siguiente operación: 240+52=292.

VALOR DECIMAL	VALOR BINARIO		
	Signo	Dato	
292	+	100100101	resultado
-37	1	00100101	desbordamiento
292/2=146	0	10010010	escalado
255	0	11111111	aritmética saturada

Puesto que el registro sólo dispone de 8 *bits* para representar la magnitud, el resultado produciría desbordamiento, cambiando el signo. La opción de escalado mantiene el signo, pero produce un mayor error con el resultado real (292-146) que el valor que proporciona la aritmética saturada (293-255), manteniéndose también en este caso el signo del resultado.

Tamaño de los registros de calculo

El tamaño de los datos con que trabaja un procesador, es decir, el número de *bit*s que contiene el dato u operando, está relacionado con la anchura del bus de datos y de las posiciones de memoria. Esto se conoce como anchura nativa de la palabra de datos. La elección de este tamaño incide en el coste del sistema, ya que el encapsulado, *pines*, tamaño de la memoria externa y *buses* está directamente relacionada con dicha anchura. Por tanto, en un diseño se suele utilizar el dispositivo con tamaño de dato más pequeño que la aplicación puede tolerar.

No obstante, la longitud de los registros que contienen los resultados parciales o finales de las operaciones debe incrementarse durante los cálculos aritméticos si no quiere perderse precisión por desbordamiento (‘*overflow’*).

La suma de dos números en coma fija puede producir desbordamiento. La norma es: "Para N sumas consecutivas, prever $\log_2$ N *bit*s extra (peor caso) para garantizar que no habrá desbordamiento". Estos *bit*s extra se suelen denominar bits de guarda, y su inclusión en los registros de cálculo da lugar a cálculos con precisión extendida.

La multiplicación en coma fija impone requerimientos más severos, puesto que el producto de dos números de N *bits* da un resultado de 2N *bits*. El resultado puede ser truncado o redondeado, pero se pueden acumular errores importantes si la truncación se realiza en cada paso. La utilización de registros de cálculo de tamaño doble al de los datos da lugar a doble precisión.

En coma flotante también pueden encontrarse situaciones en las que se produzcan efectos de desbordamiento. Para una longitud finita del acumulador, la desnormalización de números de pequeño exponente implica pérdida de resolución en la mantisa, pudiendo producirse desbordamiento para valores muy pequeños (*underflow)*. Por ejemplo, para sumar números de coma flotante, se debe desnormalizar previamente, al igual que ocurre en la suma de número decimales en notación científica:

NORMALIZADO	DESNORMALIZADO (Exponente común)
9.8765×10^{-10}	0.0987×10^{-8}
7.0345×10^{-13}	0.0000×10^{-8}
$+ \quad 4.3210 \times 10^{-8}$	$+ \quad 4.3210 \times 10^{-8}$
	4.4197×10^{-8}

Técnicas de redondeo

En general, el resultado final de una operación tiene más *bits* que los operandos debido al uso de aritmética extendida, pero los resultados finales deben ajustarse al formato de datos en memoria . Cuando se almacena el resultado en memoria, cuyas posiciones tienen un número de *bits* correspondiente a precisión simple, puede optarse por partir el resultado en varias palabras. Por ejemplo, si las posiciones de memoria son de 16 *bits* y el tamaño del registro resultado es de 32, se puede dividir éste en una palabra más significativa (MSW), que contiene el bit de signo y los *bits* más significativos de la magnitud, y otra palabra menos significativa (LSW), que suelen guardarse en posiciones consecutivas de memoria. Las opciones más usuales son:

- **Truncación**: Se eliminan los *bits* menos significativos del resultado hasta ajustar el tamaño del operando. Su principal inconveniente es que el error introducido es siempre por defecto.

- **Redondeo**: Se toma el valor más próximo en cada caso (por defecto o por exceso), con lo que los errores tienden a compensarse. Su principal inconveniente es una mayor necesidad de cálculo, ya que suele implicar una operación de truncación más un incremento del valor (suma).

- **Redondeo convergente**: Una alternativa al redondeo cuando el valor de los números está en la mitad (lo que hace que se redondeen siempre al valor mayor) es el redondeo convergente, que redondea por defecto y por exceso al 50%.

  Si sólo se va a guardar la parte más significativa del resultado final, por ejemplo:

Numero	Truncar	Error	Redondear	Error
1.44	1.4	0.04	1.4	0.04
1.49	1.4	0.09	1.5	0.01

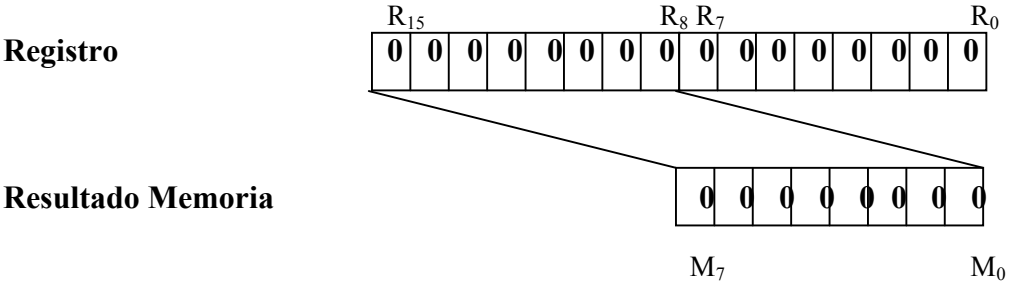
En general, la truncación debe evitarse puesto que sistemáticamente subestima el valor. Por ejemplo, supongamos el redondeo de un número de 6 dígitos, 3 de los cuales ocupan la MSW y los otros 3 la LSW. En primer lugar, sea el número 1.28325, cuyo valor redondeado al MSW (es decir, a los 3 dígitos más significativos), sería 1.28. Después supongamos el 1.28725, cuyo valor redondeado es 1.29. El redondeo se implementará sumando la mitad de escala a LSW (es decir, 500 para base decimal; 10...0 para base binaria) y quedándose con MSW:

	MSH	LSH		MSH	LSH
	1.28	325		1.28	725
+	0.00	500	+	0.00	500
	1.28	825		1.29	225

Por tanto, el redondeo se implementa en binario sumando 1 al bit más significativo de LSW (mitad menos significativa) antes de truncar. Si LSW está por debajo de la mitad de la escala completa, su bit más significativo estará a cero, y al sumar no se acarreará. Si está por encima, el bit más significativo será '1' y habrá acarreo a MSW.

El problema del redondeo es que siempre redondea los valores intermedios entre dos enteros al entero superior (1.5→2, 2.5→3 ...). Esto añade un sesgo a los valores redondeados. Para evitarlo, se puede utilizar la técnica de redondeo convergente. En este caso, los valores intermedios no siempre se redondean por exceso, sino que se redondearán por exceso o por defecto en función del bit menos significativo del resultado final. La Figura 3.3 muestra el esquema de redondeo y de redondeo convergente.

En general, el error introducido por la truncación o el redondeo se suele modelar como ruido de cuantización. En todos los casos comentados, el ruido generado tiene la misma potencia. La diferencia entre métodos radica en el sesgo introducido, mayor para la truncación y menor para el redondeo convergente. La elección del método está fijado para algunas aplicaciones de procesado por los estándares técnicos.



Redondeo:

$M_7:M_0 = R_{15}:R_8+R_7$

Redondeo convergente:

Si $R_7:R_0 = 10..00$

$M_7:M_0 = R_{15}:R_8+R_8$

Si no

$M_7:M_0 = R_{15}:R_8+R_7$

Figura 3.3. Técnicas de redondeo.

3.4. Elementos básicos de la arquitectura

Existe una gran cantidad de procesadores DSP en el mercado, pero aunque se den algunas diferencias entre ellos, podemos encontrar unidades funcionales y características comunes a todos ellos. Estos elementos básicos configuran la arquitectura genérica de un procesador digital de señal. En esta sección, se realizará una breve descripción de dichos elementos.

3.4.1. Multiplicadores

El multiplicador *hardware* es un elemento esencial en la arquitectura DSP. Si tenemos en cuenta la operación básica de procesado digital que comentábamos al principio del presente capítulo, es fácil comprender que, en muchas aplicaciones, la mitad o más de las operaciones a realizar sean multiplicaciones.

El multiplicador de un DSP realiza una multiplicación en paralelo de dos entradas de n *bits* en un único ciclo de reloj, permitiendo diversos formatos. Las características deseables en un multiplicador serían:

- Operación totalmente paralela, con multiplicación de dos entradas en un solo ciclo.
- Debe aceptar diversos formatos de entrada: complemento a dos o sin signo, fraccional o entero, modo mixto (con o sin signo).

- Debe proporcionar diversos formatos de salida: desplazamiento a derecha (entero) o a izquierda (fraccional).
- Opción de precisión extendida o doble.
- Multiplicación en coma fija y/o coma flotante.
- Disponibilidad de registros de entrada y salida configurables para *latch* (modo *pipelining*) o transparencia (mínimo retraso).

Formatos de entrada

En general, los multiplicadores permiten controlar el formato de la operación, adaptándola al tipo de operandos de entrada. Las opciones posibles serían:

X	Y	RESULTADO
con signo	con signo	con signo
sin signo	sin signo	sin signo
con signo	sin signo	con signo (mixto)
sin signo	con signo	con signo (mixto)

El modo mixto permite operaciones en doble precisión. En la siguiente figura, los productos que implican partes altas y bajas de los operandos de entrada deben implementarse como mixtos, mientras que XM·YM será un producto con signo y XL·YL sin signo.

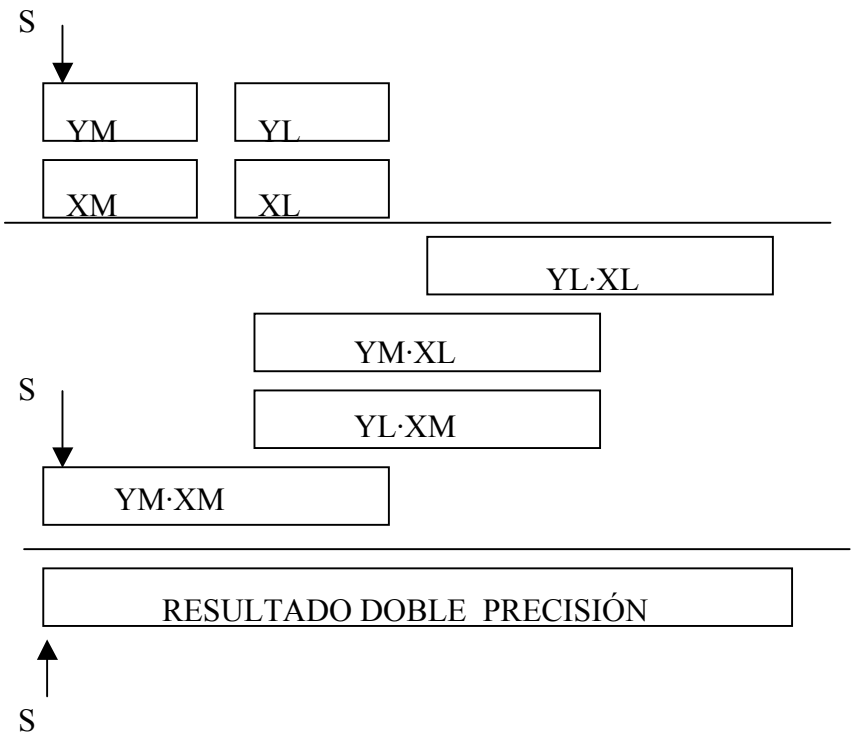


Figura 3.4. Signo y control de formato en doble precisión.

Formatos de salida

El producto de dos número de n *bits* con signo (n+1 *bits*) necesita sólo 2n+1 *bits* (2n *bits* más signo). No obstante, el producto de dos número en complemento a dos produce un bit extra, es decir, 2(n+1) *bits*, que corresponde a un segundo bit de signo. Para el caso de multiplicaciones de número sin signo (n+1 *bits*), la salida también será de 2(n+1) *bits*. Para manejar las distintas posibilidades, los multiplicadores suelen disponer de un desplazador interno que permite al usuario seleccionar qué hacer con el signo redundante en operaciones en complemento a dos.

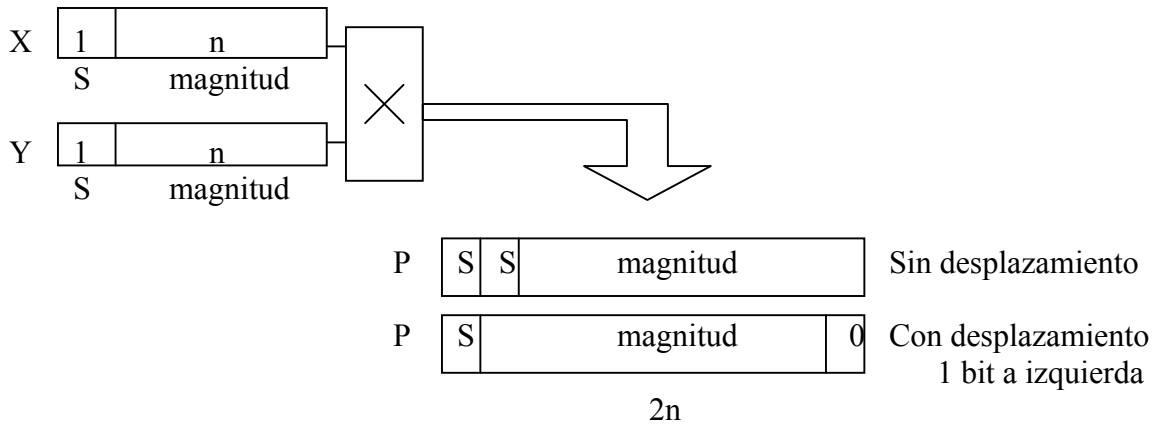


Figura 2.5. Alternativas para el signo redundante en el producto.

En modo sin desplazamiento, se trabaja como en aritmética entera sin signo. En el modo de desplazamiento a la izquierda, se justifica el valor como en artimética fraccionaria. Se puede obtener una mejora en la precisión con un factor de 2 si una de las entradas no tiene signo, por lo que la magnitud de salida, de 2n+1 *bits*, no tiene redundancia de signo.

Los multiplicadores de procesadores DSP de coma flotante realizan productos de dos operandos, pero no suelen producir un resultado de doble precisión (para un tamaño del dato de 32 *bits*, y una mantisa de 24 *bits*, el producto debería ser de 48 *bits*). En lugar de esto, utilizan precisión extendida con algunos *bits* de guarda en el registro desalida.

Multiplicadores con *pipeline*

Un multiplicador que disponga de estructura segmentada (*pipeline*) puede doblar su razón de proceso almacenando un resultado en un registro intermedio y aceptado el siguiente par de datos de entrada mientras la salida se está formateando, desplazando, etc. Un conjunto de registros intermedios, por tanto, facilitan cálculos repetitivos, pero también puede añadir retrasos al comienzo y al final del bucle, cuando la *pipeline* se llena o vacía. Lo ideal es disponer de la opción de modo *pipeline* de trabajo, funcionando el multiplicador en modo normal durante cálculos no repetitivos.

3.4.2. Multiplicadores-Acumuladores

La combinación de un sumador y un multiplicador (MAC) es deseable puesto que permite implementar funciones del tipo:

$$\Sigma\, X \cdot Y$$

utilizadas en operaciones de cálculo vectorial, tales como filtros digitales, correlación y transformada de Fourier. Las características deseables en un MAC son:

- Multiplicación y suma en un solo ciclo.
- *Bits* de guarda en el acumulador.
- Detección previa de desbordamiento.
- Redondeo.
- Aritmética saturada.
- Líneas de realimentación internas que permitan optimizar el control de bucles.
- Modo *pipeline*.

La estructura básica de un MAC se muestra en la Figura 3.6. La salida del multiplicador, de 2N *bits*, se conecta a un sumador/restador. La línea de realimentación desde su salida hasta la segunda entrada permite realizar la operación:

$$R(n+1) = X(n+1) \cdot Y(n+1) \pm R(n)$$

La salida del sumador se amplía en M *bits* de guarda, lo que permite 2^M sumas repetitivas sin desbordamiento. Algunos procesadores no disponen de *bits* de guarda, por lo que se debe escalar la salida del multiplicador para evitar desbordamiento. Para ello disponen de un desplazador entre dicha salida y la entrada correspondiente del sumador.

Las diversas opciones de funcionamiento del MAC (tales como redondeo o aritmética saturada) se realizan mediante un conjunto de registros y una lógica de control asociada. La

Figura 3.7 muestra una estructura ampliada, con un bloque MAC que incluiría el multiplicador y el sumador/restador comentados anteriormente.

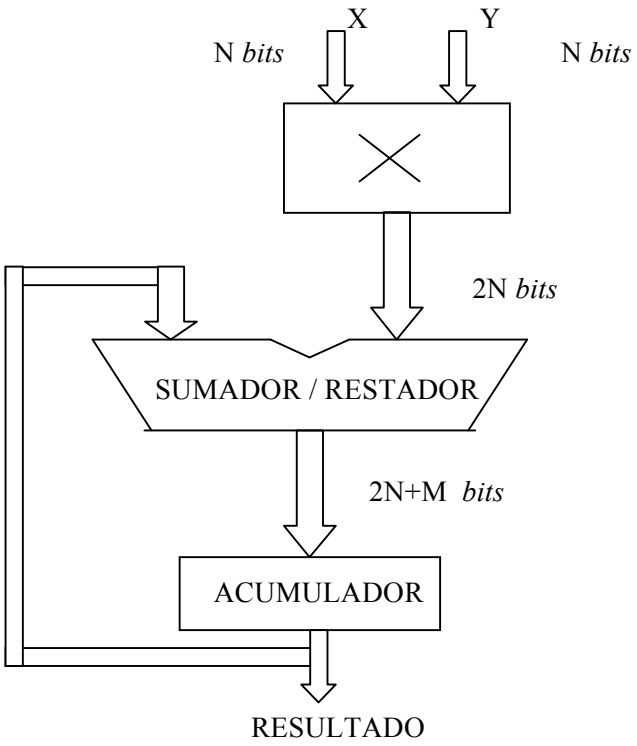


Figura 3.6. Estructura básica de un MAC.

El registro acumulador, ACC, está dividido en dos palabras de anchura igual a la de los operandos, MSW y LSW, más un campo EX (extensión) que contiene los *bits* de guarda. Examinando EX puede realizarse una detección previa de desbordamiento.

El registro de desplazamiento tiene una doble misión. El desplazamiento a la izquierda permite eliminar el bit de signo redundante en una multiplicación de complemento a dos. A la derecha realiza un escalado de los resultados. Puesto que el desplazamiento también puede producir desbordamiento, existe una entrada en la lógica de detección asociada al desplazador.

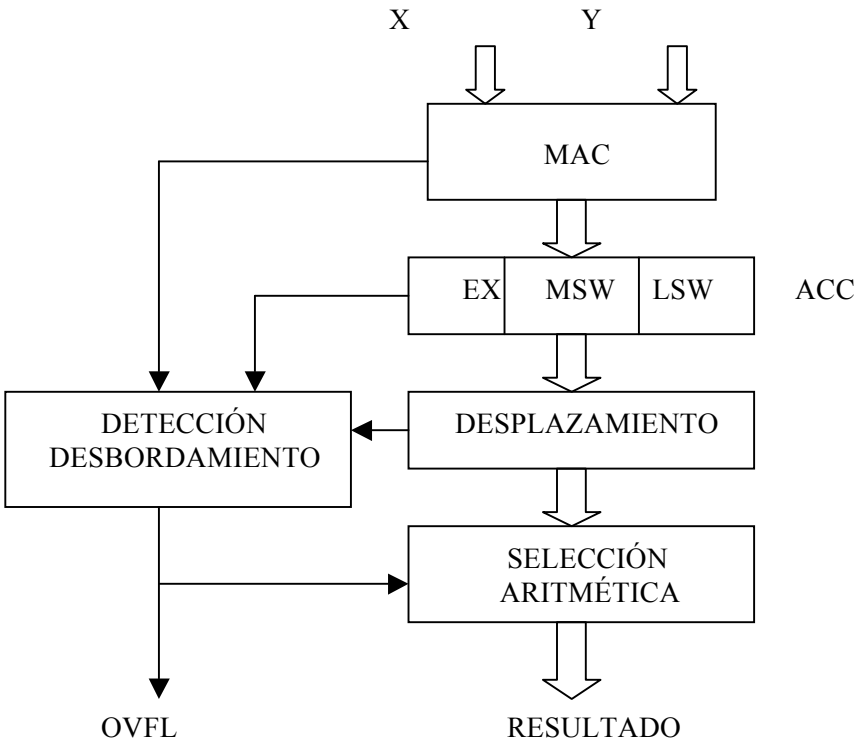


Figura 3.7. Estructura ampliada de un MAC.

El bloque final implementa las opciones aritméticas sobre el resultado. El redondeo es una opción importante en un MAC si va a guardarse sólo el MSW del resultado. La mayor precisión se obtiene redondeando sólo el resultado final. En este bloque es interesante la opción de aritmética saturada, ya que permite que desbordamientos producidos en resultados parciales, y posteriormente compensados, no produzcan escalados innecesarios. Esta opción puede responder a los *bits* de desbordamiento del sumador o del desplazador, los cuales dependen a su vez del contenido y bit de signo de EX. Por ejemplo:

OVFL	Signo (EX)	Resultado
0	0 (+)	Sin cambios
0	1 (-)	Sin cambios
1	0 (+)	01..11 11..11
1	1 (-)	10..00 00..00

Este modo proporciona una salida con el valor mayor más positivo o más negativo.

La salida del MAC debe proporcionar también signo extendido para resultados en complemento a dos, rellenando con ceros para datos sin signo. Por último, el resultado final se divide en MSW o LSW.

3.4.3. ALU

La unidad aritmético-lógica (ALU) de un DSP es similar a la de un microprocesador de propósito general, soportando el conjunto usual de operaciones aritmético-lógicas. Las diferencias vienen dadas por sus características añadidas, que permiten ejecución de instrucciones en un solo ciclo, y *pipeline* opcional, además de estar frecuentemente asociada a elementos de escalado, como desplazadores, que permiten mantener el mayor rango dinámico posible en las operaciones. En algunos procesadores DSP que no disponen de un sumador específico en el MAC, la ALU se utiliza para realizar la operación acumulación.

Las características deseables en una ALU de DSP son:

- Ejecución de funciones aritmeticas: suma, resta, suma con acarreo, módulo, división ...
- Ejecución de funciones logicas: AND, OR, OR exclusiva, negación lógica.
- Operaciones condicionales: por ejemplo, *Suma+(Desplazamiento si condición*). Distinguir con/sin signo.
- Capacidad para doble precisión, que permite menor pérdida de tiempo.
- Movimiento de datos y cálculo eficientes: datos más aritmética en el mismo ciclo y dos operandos cargado en la ALU en cada ciclo.
- Opción *pipeline*.

- Lazos de realimentación: permiten usar resultados como entradas en el siguiente ciclo de reloj, sin acceder al bus. Tipos: salida a entrada (acumulaciones); salida a entrada del desplazador o salida del desplazador a entrada de ALU.
- Registros: 1) Conjunto dual para conmutación de contexto en interrupciones; 2) Fichero de registros para acceso y almacenamiento rápidos de resultados intermedios.

La Figura 3.8 muestra la estructura básica de una ALU DSP. Los operandos de entrada pueden obtenerse directamente del bus de datos, pero resulta más eficiente disponer de ficheros de registros para evitar accesos a memoria. Por otra parte, aunque no aparece en la figura, algunos procesadores disponen de desplazadores en las entradas de la ALU, pudiendo así desplazar los datos durante la transferencia de los mismos y mejorando por tanto la velocidad de proceso.

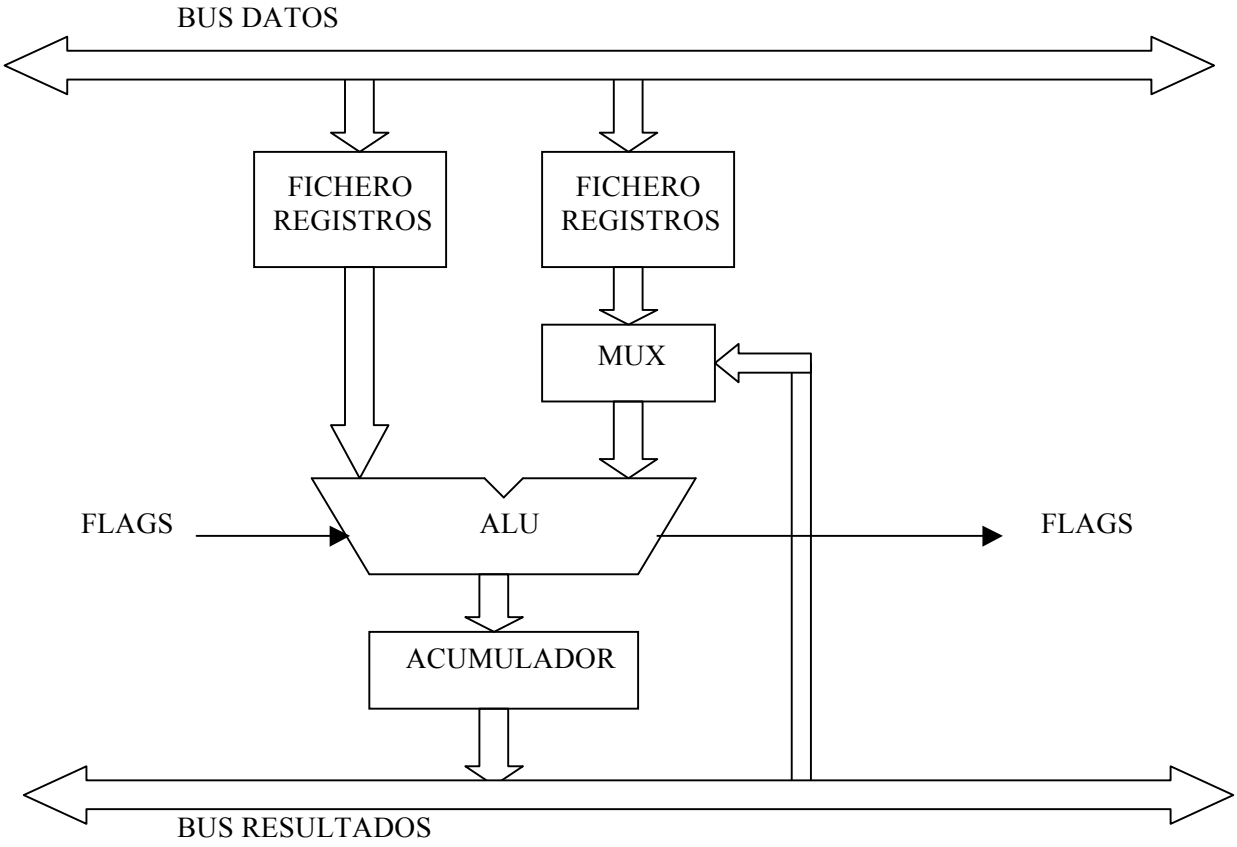


Figura 3.8. Estructura de una ALU-DSP.

3.4.4. Desplazadores

La operación de escalado se utiliza en un gran número de aplicaciones con DSP de coma fija. Por ejemplo, supongamos un filtro digital de ganancia G=10. Si utilizamos un formato de coma fija como el Q15 para representar los coeficientes y los datos de entrada y salida, cada operando puede tener un valor máximo de 1. No obstante, al aplicar la ganancia del filtro, la salida podría alcanzar un valor de 10. Puesto que dicho valor no puede representarse en el formato utilizado, se produciría desbordamiento. La solución consiste en encontrar un factor de escalado tal que coeficientes y datos estén previamente divididos por ese factor, aunque a cambio de perder resolución en los cálculos.

El desplazador realiza la operación de escalado, entre otras funciones. Es frecuente encontrar desplazadores no sólo en las entradas de operandos, sino también entre la salida del multiplicador y la entrada del sumador, o a la salida del resultado. Las operaciones que implementa son:

- Desplazamientos aritméticos y lógicos de n *bits*.
- Rotación de palabras.
- Test de *bits*/ Set bit/ *Reset* bit.
- Normalización y desnormalización.

Desplazamientos aritmético-lógicos y rotación.

El desplazador realiza desplazamientos aritméticos y lógicos para escalado de operandos. Los desplazamientos de *bits* a derecha o izquierda implican divisiones o multiplicaciones, respectivamente, por potencias de 2 (2^n). Los desplazamientos aritméticos a derechas efectúan una extensión de signo, rellenando los *bits* desplazados con el bit de signo S, para preservar el valor de los número positivos o negativos. Mientras que los desplazadores de un solo bit guardan el bit desplazado (por ejemplo, en el indicador de acarreo), desplazamientos de *bits* múltiples generalmente truncan el número. Desplazamientos aritméticos a la izquierda rellenan con ceros los *bits* menos significativos. Los desplazamientos lógicos suponen números sin signo, por lo que ambos sentidos de desplazamiento se rellenan con ceros.

Por contraposición con los desplazadores de los microprocesadores de propósito general, que desplazan un bit por ciclo, algunos desplazadores utilizados en DSPs (denominados *barrel shifter*) son capaces de desplazar varios *bits* de una palabra en un solo ciclo para preservar la

velocidad de ejecución. La arquitectura *barrel shifter* permite también rotación de 1 a n *bits* en sólo 2 ciclos.

Normalización.

La normalización (conversión de coma fija a flotante) se realiza en dos pasos (ver Figura 3.9): en primer lugar, un detector de exponente calcula el valor del desplazamiento necesario, y proporciona este valor al control del desplazamiento. En el caso de la desnormalización (conversión de coma flotante a coma fija) el exponente, cargado del bus de datos, constituye el código de control para el desplazador. El signo es extendido por el desplazador.

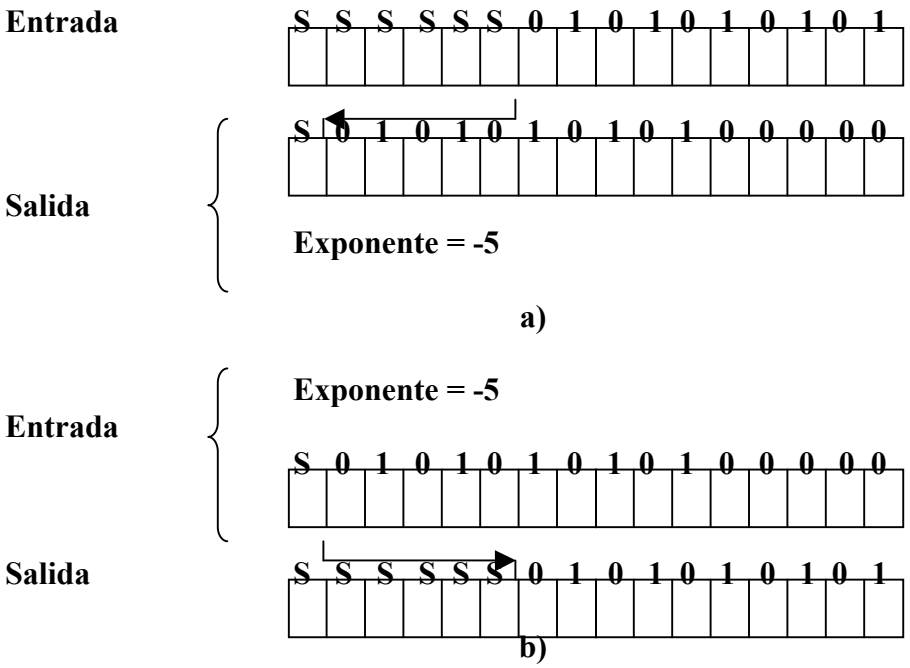


Figura 3.9. Funciones de normalización (a) y desnormalización (b).

3.4.5. Generador de direcciones

Una unidad generadora de direcciones (Address Generator Unit: AGU) tiene uno o más registros de salida que contienen direcciones de memoria de datos. Para evitar utilizar la ALU de computación en el cálculo de direcciones, la AGU debe disponer de su propia ALU para aritmética entera, proporcionando la dirección en paralelo con la ejecución de operaciones.

Las características deseables en una AGU son:

- Proporcionar una dirección precalculada a la memoria de datos y modificar la dirección según un *offset*.
- Control de *buffers*.
- Implementación de *buffer* circular.
- Revertir el orden de los *bits* (esto permite ordenar automáticamente los armónicos durante el cálculo de la FFT, que típicamente aparecen con un patrón de *bit* inverso a los datos de entrada).

Buffer circular

En muchos algoritmos de procesado digital se utilizan *buffers* para almacenar temporalmente las muestras de señal. Una estructura usual es la FIFO, que implica al menos la utilización de un puntero de acceso. Después de cada operación de lectura o escritura en el *buffer*, debe actualizarse el puntero comprobando si ha llegado al límite del *buffer*, en cuyo caso debe inicializarse al comienzo. La utilización de instrucciones explícitas para efectuar esta comprobación implica un consumo de tiempo y, por tanto, una pérdida de velocidad de proceso.

Algunas AGU permiten implementar *buffers* circulares, en los que la detección del final del *buffer* se realiza a nivel *hardware*, utilizando una aritmética de módulo. Para ello, las AGU combinan diversas funciones en una única instrucción:

$$R = R + M \; ; \; \text{Si} \; R >= FIN \; \text{entonces} \; R = COMIENZO.$$

donde R es la dirección calculada, M el *offset* y COMIENZO y FIN especifican los límites del *buffer*.

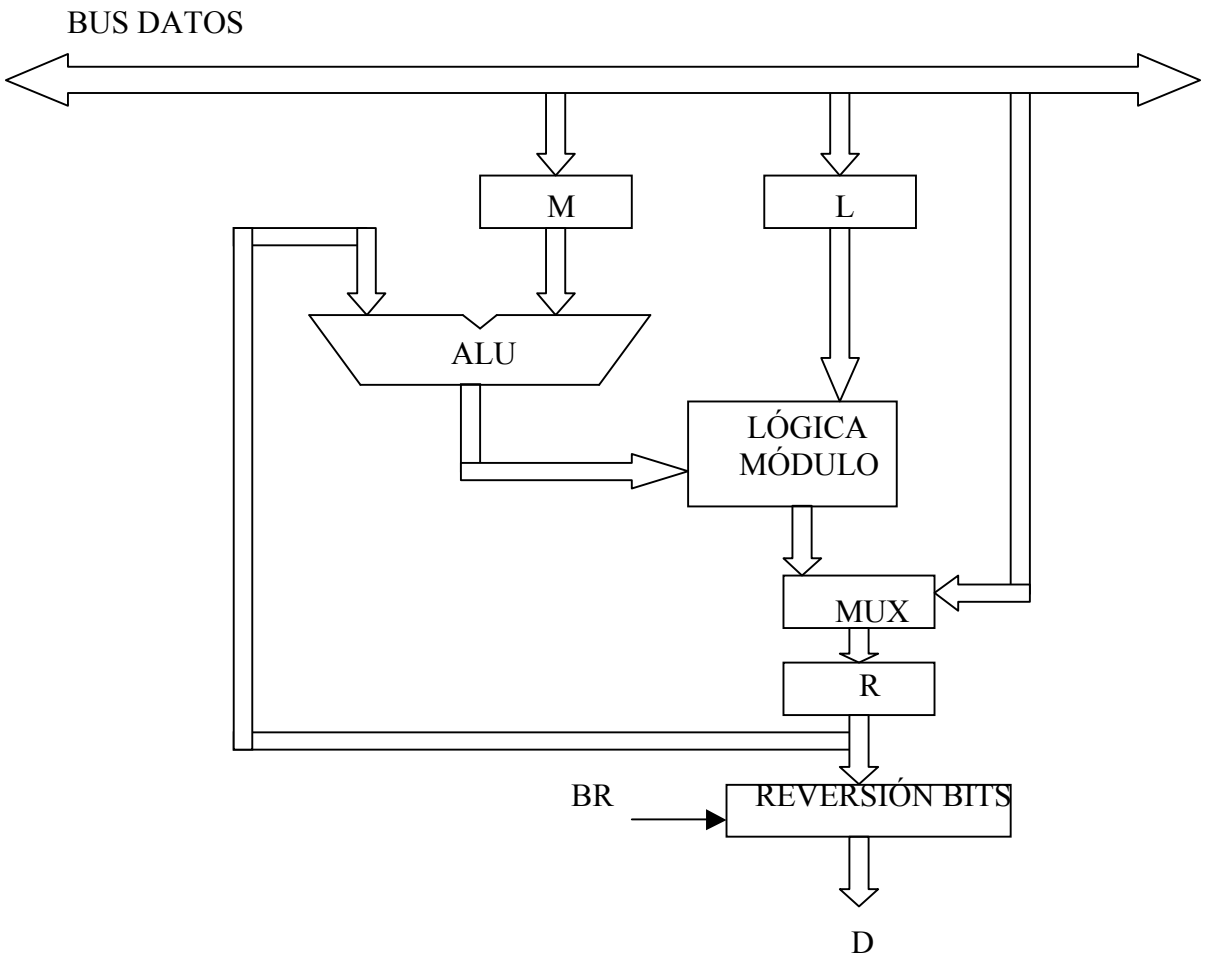


Figura 3.10. Esquema general de una AGU: M (modificación), L (límite del *buffer*), R (resultado), D (dirección de salida). La entrada al multiplexor desde el bus de datos permite hacer una carga directa de dirección.

Muchos programas necesitan modificar el valor del *offset* M durante la ejecución para efectuar accesos a posiciones no contíguas. La AGU debe poder implementar:

$$R = R + M \; ; \text{ modificar M.}$$

donde modificar M puede implicar un incremento fijo, y ';' indica concurrencia, es decir, ambas operaciones se ejecutan en el mismo ciclo. La ejecución de la parte derecha se realiza cuando R se ha calculado.

Una estructura básica de una AGU con soporte para *buffer* circular se muestra en la figura 3.10.

3.4.6. Secuenciador de programa

Un secuenciador de programa (SEQ) controla el flujo del programa determinando la siguiente instrucción a ejecutar. Incluye los siguientes componentes:

- Contador de Programa.
- Lógica de control de saltos condicionales.
- Control de bucles.
- Control de interrupciones.
- Pila.

Control de bucles

Los algoritmos de procesado digital suelen contener, al igual que ocurre con otros tipos de programas, bucles de instrucciones, es decir, varias instrucciones que se repiten un cierto número de veces. La aproximación más usual a la gestión de bucles es utilizar instrucciones de salto condicional, que permiten repetir el bucle en función del valor de una variable que contiene el número de veces que se debe repetir la ejecución y que actúa como contador *software*. No obstante, las instrucciones de salto condicional suelen necesitar más de un ciclo para su ejecución, además de implicar el vaciado de la *pipeline*. Por otra parte, la variable contador debe incrementarse o decrementarse en cada iteración, comprobando también si ha alcanzado el valor límite. Todo esto introduce un tiempo añadido de cálculo a la ejecución de las instrucciones del bucle y, por tanto, disminuyendo la velocidad de proceso.

Los procesadores DSP suelen evitar este problema mediante la utilización de contadores *hardware* de bucles, que permiten ejecutar las instrucciones implementando por *hardware* la comprobación de condición, sin añadir así penalizaciones de tiempo por gestión del bucle.

Básicamente hay dos tipos de contadores de bucles en DSPs: los que gestionan bucles de una única instrucción y los de varias instrucciones.

Los contadores de bucle de una instrucción ejecutan repetidamente la misma instrucción por lo que, después de la primera fase de búsqueda, no vuelve a accederse a memoria para buscar nueva instrucción hasta que el bucle termine. Por tanto, se dispone de ciclos de memoria adicionales que permiten obtener, por ejemplo, dos operandos en el mismo ciclo de instrucción. En el caso de contadores de bucles multi-instrucción, no se dispone de dichos ciclos de memoria adicionales.

Algunos procesadores DSP desactivan las interrupciones durante la ejecución de bucles de una instrucción. Esto puede eventualmente producir retrasos (latencias) en el servicio de interrupciones que deben valorarse cuidadosamente en ciertas aplicaciones.

Gestión de interrupciones

Se llama interrupción a la ocurrencia de un evento externo al procesador que detiene la ejecución del programa en curso y fuerza la ejecución de una rutina específica, denominada rutina de servicio de interrupción. Esta rutina depende del evento que produjo la interrupción, por lo que el procesador dispone de tantas rutinas de servicio como tipos de interrupciones acepta (por ejemplo, la rutina de servicio de transmisión en un puerto de comunicaciones leería un dato del *buffer* de salida y lo escribiría en el registro correspondiente del puerto). Después de ejecutada, el procesador retoma el programa anteriormente en curso en el punto en que lo había dejado.

El método de interrupciones constituye la técnica más importante en los DSPs para comunicación con sus periféricos. Las fuentes más usuales de interrupción son:

- Periféricos internos, incluidos en el procesador: temporizadores, puertos serie, etc.
- Líneas de interrupción externas: entradas procedentes de periféricos externos.
- Interrupciones *software*: generadas por la ejecución de instrucciones especiales o por errores de proceso como desbordamiento, división por cero, etc. También se denominan excepciones.

La dirección de la instrucción a la que se salta tras una interrupción no es la de comienzo de la rutina de servicio. En lugar de esto, el procesador salta a una posición determinada (distinta para cada interrupción) de una tabla donde se localizan punteros que contienen las direcciones reales de comienzo de las rutinas de servicio. Estos punteros se denominan vectores de interrupción, y permiten localizar las rutinas en cualquier posición del mapa de memoria, ganando así en flexibilidad.

Una cuestión importante asociada a la gestión de interrupciones es la priorización. Consiste en decidir cuál es el orden en el que se servirán varias interrupciones simultáneas. Generalmente el orden de prioridad está prefijado en el procesador, y una interrupción de mayor prioridad puede interrumpir la ejecución de una de menor prioridad, produciéndose un anidamiento automático. Utilizándose instrucciones de activación o desactivación global de interrupciones puede impedirse este efecto de anidamiento.

Otro aspecto asociado a la utilización de interrupciones, y especialmente crítico cuando se desarrollan aplicaciones para trabajo en tiempo real, es la latencia o retraso existente entre el momento en que se produce la interrupción y el acceso al vector correspondiente. Este tiempo se mide en ciclos de instrucción, y tiene en cuenta la periodicidad con que el procesador comprueba la existencia de interrupciones (generalmente en la última fase de cada ciclo de instrucción, aunque algunos casos como la ejecución de bucles *hardware* puede aumentar significativamente este tiempo), la sincronización con otras señales internas de la unidad de control, los efectos de la *pipeline*, etc.

Pilas

Las pilas se utilizan para almacenar temporalmente direcciones de retorno de subrutinas e interrupciones. En este último caso también guardan información de estado. En procesadores DSP suele existir diversos tipos:

- Registros ocultos, que duplican en contenido de registros del procesador para preservar el contexto durante interrupciones. Permite conmutaciones rápidas de contexto, pero su tamaño es limitado.

- Pila *hardware*, utilizada para direcciones de retorno en subrutinas e interrupciones. Constituida por posiciones de memoria interna y de tamaño generalmente pequeño, por lo que puede producirse desbordamiento de la pila si el grado de anidamiento es alto.

- Pila *software* convencional, localizada generalmente en memoria principal. Permite una programación del tamaño dependiendo de la demanda de la aplicación.

3.4.7. Estructuras de memoria

Las estructuras de memoria están relacionadas con los modos de interconexión con el procesador. La más básica, tal y como se vio en el capítulo 1, es la arquitectura Von Neumann, que se muestra en la figura 3.1.

Debido a la existencia de un único *bus* de datos y otro de direcciones, esta arquitectura sólo permite un único acceso a memoria por ciclo. Si consideramos la operación básica de procesado digital para la que se optimizó la arquitectura DSP, el procesador necesitaría al menos tres ciclos para completarla (un acceso para buscar la instrucción y dos para obtener los operandos). Por tanto, aunque el procesador esté optimizado para realizar dicha operación en un ciclo, la estructura de memoria lo impediría.

En sistemas DSP, se han adoptado otras alternativas a esta estructura, encaminadas a permitir realizar las operaciones en un solo ciclo. Las más usuales son la arquitectura Harvard y las memorias de acceso múltiple.

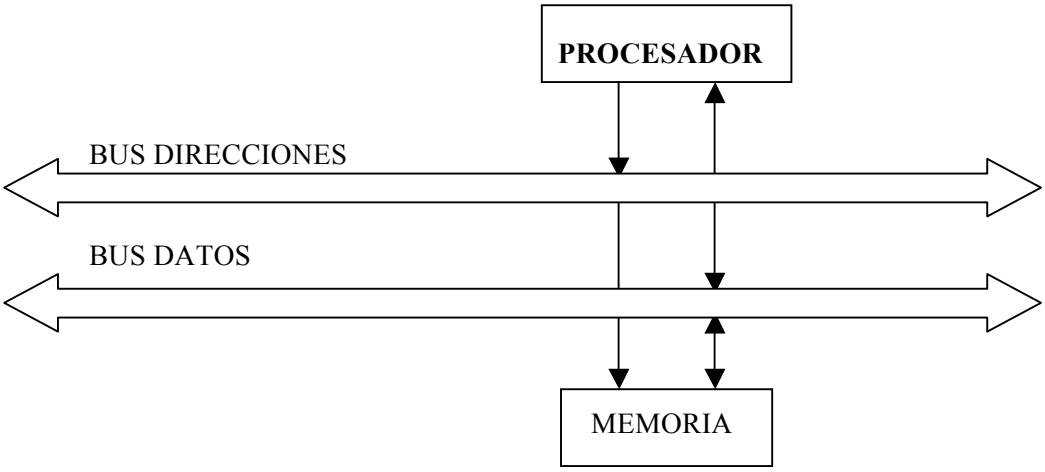


FIGURA 3.1. Arquitectura Von Neumann.

Arquitectura Harvard

En la arquitectura Harvard, el procesador se conecta a dos módulos independientes de memoria, conteniendo uno de ellos instrucciones (memoria de programa) y el otro datos (memoria de datos). Utiliza por tanto dos *buses* de direcciones y dos de datos que permiten obtener una instrucción y un dato en el mismo ciclo. Sin embargo, en el caso de sistemas DSP se suele utilizar esta arquitectura localizando en memoria de programa no sólo instrucciones sino también datos. Esto permite acceder simultáneamente a los dos operandos necesarios en la operación de multiplicación. A esta variante se le denomina arquitectura Harvard modificada. En la figura 3.2 se muestra su estructura.

Esta estructura puede generalizarse a más de dos bloques de memoria. Algunas familias de DSP disponen de tres memorias independientes, una para contener instrucciones y dos para datos, incrementando así la velocidad de proceso.

No obstante, aunque muchos DSP utilizan arquitectura Harvard, ésta suele restringirse a la memoria interna (memoria integrada en el mismo encapsulado que el procesador). Para minimizar costes, el *interface* con memoria externa se realiza generalmente mediante un único *bus* de datos y otro de direcciones, aunque puede distinguirse entre accesos a memoria de programa o de datos a través de *pines* de selección. Esto obviamente penaliza el funcionamiento del procesador cuando trabaja con memoria externa.

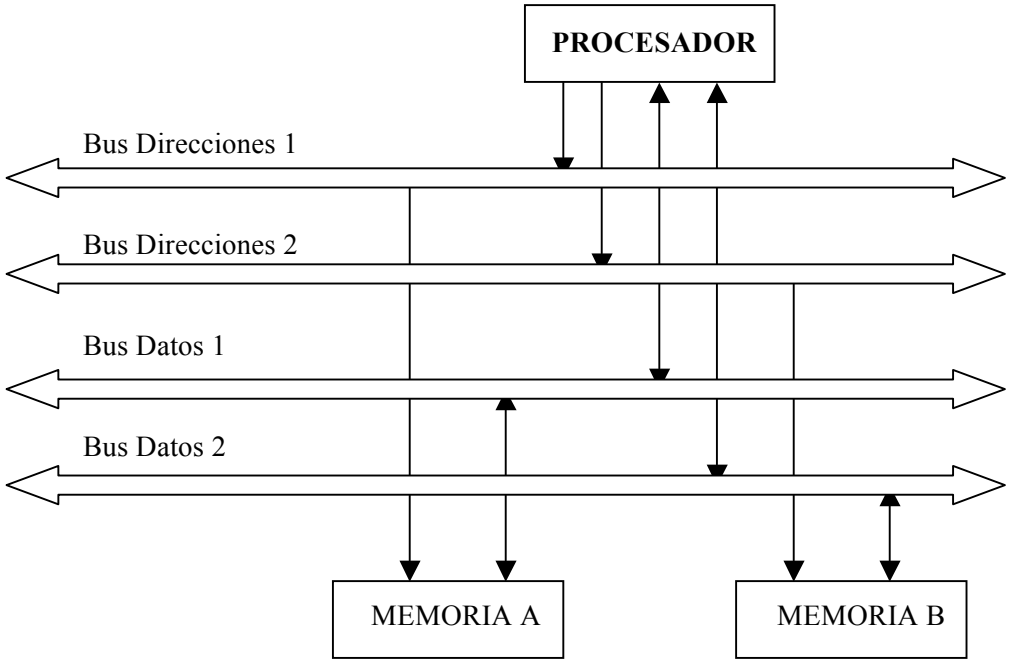


FIGURA 3.2. Arquitectura Harvard.

Memorias de acceso múltiple

Las memorias de acceso múltiple permiten también varios accesos por ciclo, al igual que la arquitectura Harvard comentada. Podemos distinguir dos clases de este tipo de memorias: las rápidas y las multipuerto.

Las memorias rápidas tienen un tiempo de acceso menor o igual a la mitad del ciclo de instrucción, por lo que soportan varios accesos secuenciales por ciclo utilizando un único *bus*. Además, estas memorias pueden combinarse con arquitecturas Harvard, incrementando el número de acceso. Por ejemplo, en el caso de una arquitectura con dos memorias rápidas, podrían realizarse hasta 4 accesos en un ciclo de instrucción (este valor máximo puede ser menor en caso de que se necesiten 3 operandos localizados en la misma memoria, con lo que automáticamente se prolongaría el tiempo de ejecución de la instrucción a dos ciclos).

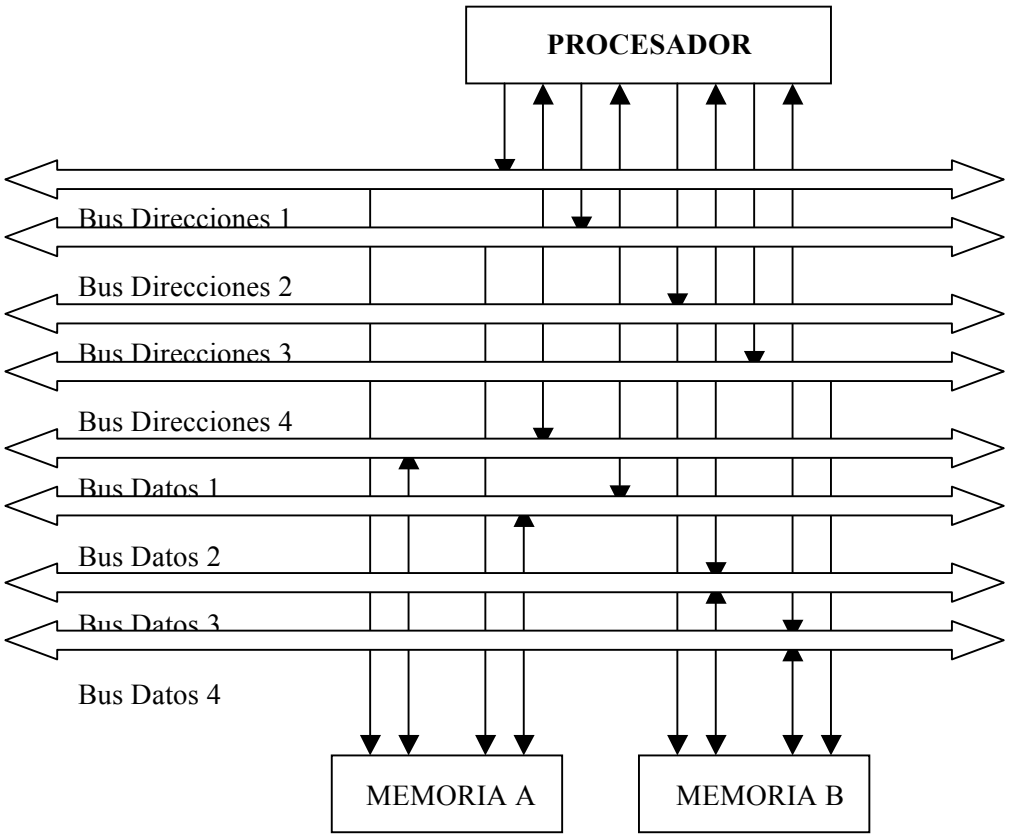


FIGURA 3.3. Arquitectura Harvard con memorias bipuerto.

Las memorias multipuerto permiten varios accesos simultáneos a la misma memoria, aunque a posiciones diferentes, utilizando dos o mas *buses* de datos y direcciones accediendo en paralelo. La estructura más común es la bipuerto, aunque existen también memorias de tres y cuatro puertos. El incremento del número de puertos permite que una mayor cantidad de datos pueda accederse por ciclo, pero también incrementa el coste del hardware, no sólo de la memoria, que debe multiplicar los *pines* externos y los mecanismos de direccionamiento y control de acceso, sino de los buses externos. La siguiente figura muestra una arquitectura Harvard utilizando memorias multipuerto.

3.4.8. Periféricos integrados

Puertos Serie

Los puertos serie transmiten o reciben los datos bit a bit, por lo que requieren muchos menos *pines* que un puerto paralelo, disminuyendo el coste de implementación. Sus aplicaciones más usuales en sistemas DSP son:

- Conexión con conversores A/D y D/A, y *codecs*.
- Conexión con otros microprocesadores o DSPs.
- Comunicación con otros periféricos externos.

Podemos distinguir dos tipos de puertos serie: síncronos y asíncronos. En el primer caso, se utiliza una señal de reloj, que se transmite junto con los datos, para temporizar en el receptor la duración de cada bit. En el segundo, no se utiliza señal de reloj, por lo que el receptor genera la temporización a partir de los datos, lo que limita la velocidad de transmisión.

Las características más importantes de un puerto serie son:

- Datos: hay dos aspectos que definen los datos a transmitir. En primer lugar, el número de bits que tiene el dato (valores comunes son 8 y 16 bits). En segundo lugar, el orden de transmisión: puede comenzarse por el bit menos significativo (LSB) o el más significativo (MSB), dependiendo del procesador.

- Sincronización: en puertos síncronos, el estado de los bits cambia de forma sincronizada con el flanco (ascendente o descendente) de la señal de reloj, que fija por tanto la duración del bit. Además, estos puertos suelen disponer de otra señal de sincronización asociada a cada palabra de datos (*frame sync*, o *word sync*). Esta señal puede tener una duración igual al primer bit de la palabra o a la palabra completa (*bit length* o *word length*). La siguiente figura muestra un ejemplo de este tipo de sincronización. Por último, algunos puertos serie utilizan una señal de sincronización para más de una palabra, como ocurre en el caso de conexión con conversores de audio estéreos, en los que se transmite una muestra de cada canal por vez.

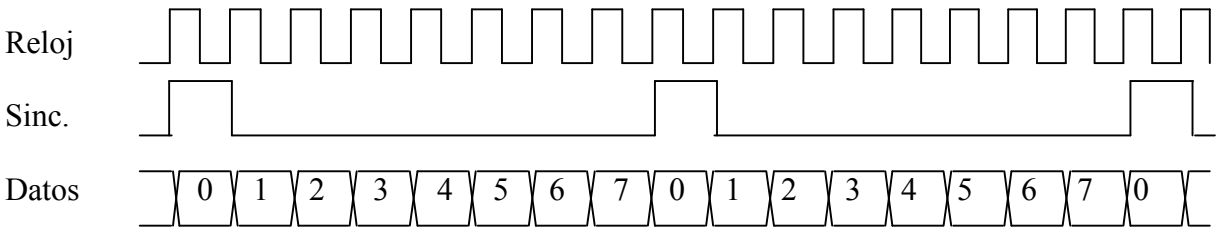


Figura 3.9. Ejemplo de señales de reloj, sincronismo y datos en un *bus* serie síncrono.

- Secciones de recepción y transmisión independientes: En caso de darse, permite una comunicación en los dos sentidos simultáneamente (*full duplex*). Si sólo hay una sección, la transmisión y recepción deben producirse secuencialmente.

- Generación de la señal de reloj: dicha señal se comparte por los dos dispositivos conectados, por lo que uno de ellos (o un tercer dispositivo externo) debe disponer de la lógica necesaria para generar dicha señal. Usualmente se obtiene a partir de la señal de reloj del procesador utilizando contadores que dividen la frecuencia hasta adaptarla a la velocidad de transmisión deseada.

- Soporte para funcionamiento en multiplexado por división temporal (TDM: *time division multiplexing*): Este método permite conectar más de dos procesadores a un puerto serie síncrono. Para ello, se divide el tiempo de transmisión en porciones (*slots*), cada una de las cuales está asociada al tiempo en que un determinado procesador puede transmitir, mientras el resto escuchan. La estructura del *bus* serie es similar a la comentada: una línea de reloj, otra de datos y otra de sincronismo. Uno de los procesadores (o un circuito externo) es el encargado de generar las señales de reloj y sincronismo para el resto. Esta última señal se activa al comienzo de cada ciclo, en el tiempo asignado al primer DSP. En ese momento, el DSP puede transmitir por el *bus* una palabra, formada por los bits de datos más algunos que contienen información de destino (por ejemplo, si hay conectados 4 procesadores, utilizará 2 bits para codificar el destino). En algunos casos, este información se envía por una línea accesoria. En cualquier caso, cada procesador debe recibir durante su tiempo de escucha toda la información enviada por el resto, y determinar mediante los bits de dirección de destino cuál le está destinada.

- Soporte para *companding*: se trata del proceso de compresión de los datos en el emisor y su posterior expansión en el receptor. Esta técnica se utiliza frecuentemente en

transmisión de voz, ya que incrementa la cantidad de información que puede transmitirse por el canal. Los puertos serie de algunos procesadores DSP permiten su conexión con *codecs* externos, conversores A/D y D/A utilizados para telefonía y audio de baja fidelidad en transmisión *companding*, implementando la ley de compresión (*μ-law* o *A-law*, según se trate del estándar americano o europeo) y evitando así su ejecución *software*.

Temporizadores

Prácticamente todos los procesadores DSP incluyen temporizadores como periféricos internos. Los temporizadores se utilizan preferentemente para generar secuencias de eventos (por ejemplo, la señal de muestreo en sistemas de tiempo real). La estructura de un temporizador se muestra en la siguiente figura. Básicamente consta de una fuente de la señal de reloj, generalmente el propio reloj del procesador (*master clock*), un escalador para reducir la frecuencia de entrada según un valor programado, y un contador descendente. Cuando este contador alcanza el valor cero, produce una interrupción que fuerza al procesador a saltar a la rutina de servicio correspondiente. Posteriormente, el contador se recarga con el valor de cuenta programado y comienza un nuevo ciclo.

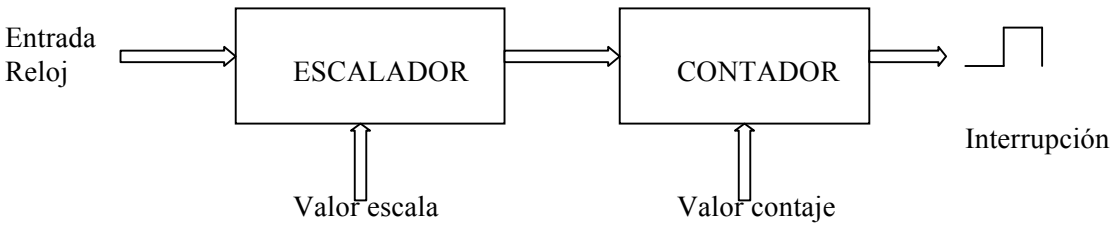


FIGURA 3.10. Estructura genérica de un temporizador.

Puertos paralelo

Los puertos paralelo transmiten múltiples *bits* simultáneamente, permitiendo, por tanto, mayores velocidades de comunicación que los serie, pero necesitan un mayor número de *pines* externos (valores típicos son 8 y 16 *bits*). Además es frecuente la existencia de *pines* adicionales de control (*handshake* o *strobe*), que transmiten información sobre el momento en que el procesador o el periférico externo han enviado nuevos datos. Dependiendo de su uso, podemos distinguir varios tipos de puertos paralelos en DSPs:

- Puertos de datos: la transmisión puede hacerse mediante *pines* dedicados, lo que simplifica la conexión con los periféricos pero implica aumentar el encapsulado del procesador y, por tanto, el precio. Otra opción más común es utilizar los *pines* del *bus* de datos externo. Cuando las direcciones de los puertos están mapeadas en memoria, es decir, ocupan posiciones del mapa de memoria del sistema, leer o escribir un dato en un puerto es totalmente equivalente a hacerlo en una posición de memoria, y por tanto la utilización de los buses está totalmente compartida. Cuando el procesador tiene un espacio de direccionamiento independiente para los puertos, mapeado E/S, puede utilizar también el *bus* de datos, pero necesita alguna línea del *bus* de control interno que identifique esa dirección como de un puerto.

- Puertos de bits. En este caso, cada bit de la palabra transmitida tiene un significado propio. Esto es frecuente en aplicaciones de control, donde los bits individuales gestionan entradas o salidas independientes, y aunque los DSP orientados a control suelen tener puertos de bits más sofisticados, en general sólo disponen de unas pocas líneas dedicadas (un valor usual suele ser 2).

- Puertos *Host*. Estos puertos permiten la comunicación entre el DSP y otro microprocesador de propósito general u otro DSP. Sus aplicaciones incluyen la transferencia de datos y, en algunos casos, también para control del DSP por parte del otro procesador, forzando la ejecución de determinadas instrucciones o rutinas de servicio de interrupción, así como modificación de registros o carga de programas.

- Puertos de comunicaciones. Se trata de puertos paralelos específicos para comunicaciones entre DSPs en funcionamiento multiprocesador. Las principales diferencias con los puertos *host* son la conexión entre procesadores iguales o similares, y el hecho de no incluir generalmente posibilidades de control.

Conversores A/D y D/A

Algunos DSPs, orientados específicamente al procesado de voz (por ejemplo, en el caso de telefonía móvil digital), disponen de codecs internos. Las características más importantes de este tipo de conversores son:

- Resolución (en bits): Un valor usual es 16 bits.
- Frecuencia de muestreo: Generalmente se muestrea a 8 kmuestras/seg., pero algunos procesadores alcanzan las 24 kmuestras/seg.
- Relación señal/(ruido+distorsión): Equivalente a la relación señal/ruido usual, pero teniendo en cuenta la distorsión que introduce el codec en el proceso de *companding*.
- Número de canales analógicos de entrada.
- Ganancia programable de salida.

Controladores DMA

(Ver tema 2).

3.5. DSP TMS320C2x

Los DSP de coma fija suelen utilizarse en campos de aplicación del procesado digital en los que el coste de implementación es un factor importante. Algunas de las aplicaciones más usuales de este tipo de procesadores son: telefonía móvil, *modems*, videoconferencia, audio digital, codificación de voz, multimedia, control digital, controladores de discos, compresión de datos, etc.

Actualmente existen diversas familias de distintos fabricantes. En esta sección se estudiará la estructura interna de una familia DSP de coma fija, la TMS320C2x, que no presenta una gran complejidad y por tanto nos proporcionará un ejemplo asequible de implementación de la arquitectura de este tipo de procesadores. Sobre esta base, se comentará por último las novedades presentadas por otros DSP más avanzados.

La familia TMS320C2x de Texas Instruments utiliza una arquitectura de tipo Harvard, manteniendo dos estructuras de *bus* independientes para datos y programa. Alcanza hasta 12.8

MIPS (millones de instrucciones por segundo). Opera con aritmética entera de complemento a dos, disponiendo de una ALU y acumulador de 32 *bits*. El multiplicador realiza productos entre palabras de 16 *bits* codificadas como enteros con signo. Los registros de desplazamiento permiten al procesador implementar escalado numérico, extracción de *bits*, aritmética extendida y prevención de desbordamiento.

El procesador dispone de pila interna, contador de repetición de instrucciones, tres interrupciones externas del usuario (enmascarables), e interrupciones internas generadas por los periféricos internos: un puerto serie *full-duplex* y un temporizador.

Incluye bloques internos de memoria ROM y RAM, de los cuales algunos pueden ser configurado como memoria de datos o programa. El *interface* con la memoria local consta de *buses* únicos tanto de datos como de direcciones. El *bus* de control permite seleccionar memoria de datos o programa y espacio de E/S. Además dispone de capacidad para funcionamiento multiproceso o utilización de DMA.

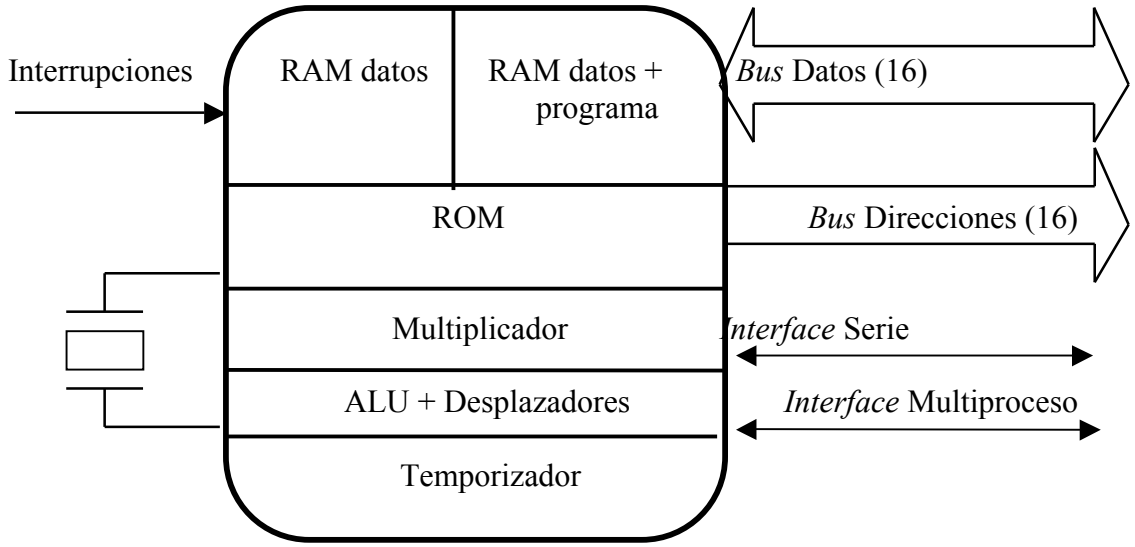


Figura 3.1. Diagrama de bloques simplificado del TMS320C2x.

Esta familia dispone de varios procesadores, cuyas diferencias consisten básicamente en las diversas velocidades de reloj, cantidades y configuraciones de memoria, así como el tipo y número de periféricos de que disponen. Esto, obviamente, implica diferencia de precios, por lo

que el diseñador puede elegir aquel DSP que más se ajuste a su aplicación. Los elementos que conforman la familia TMS320C2x son:

- TMS320C25: versiones con reloj de 33 MHz, 40 MHz y 50 MHz, 544x16 bits RAM, 4kx16 bits ROM.
- TMS320C26: reloj de 40 MHz, 1568x16 bits RAM, 256x16 bits ROM.
- TMS320C28: reloj de 40 MHz, 544x16 bits RAM, 8kx16 bits ROM.

A continuación describiremos la arquitectura del TMS320C25, como elemento genérico de la familia, comentando también algunas particularidades del TMS320C26 ya que es el que se incluye en la placa hardware cuya programación se realizará en el capítulo siguiente.

Hardware interno

El diagrama de bloques funcional del TMS320C25 se muestra en la Figura 3.2. Podemos distinguir las siguientes estructuras básicas:

- Estructura de *buses*: Utiliza arquitectura Harvard modificada con separación de *bus* de memoria de programa (*program bus*) y de memoria de datos (*data bus*). La memoria de programa puede contener datos además de instrucciones. Esto le va a permitir acceder a dos operandos simultáneamente a través de los dos *buses* para realizar productos.
- Unidad central de cálculo**:** Está compuesta por el multiplicador *hardware* y sus registros asociados, la ALU, el Acumulador y los desplazadores.
- Generador de direcciones**:** Está compuesta por la ARAU (Unidad aritmética de los registros auxiliares), el fichero de registros auxiliares (generalmente utilizados como punteros de direcciones de datos) y otros registros asociados (ARB y ARP).
- Control: compuesto por el controlador y los registros asociados para secuenciar el programa (PC, PFC), la pila interna, y otros registros de control (QIR, IR, ST0, ST1, RPTC, IFR, IMR y GREG).
- Periféricos integrados: controlados a través de registros: Puerto serie (RSR, XSR, DRR, DXR); Timer (TIM, PRD).

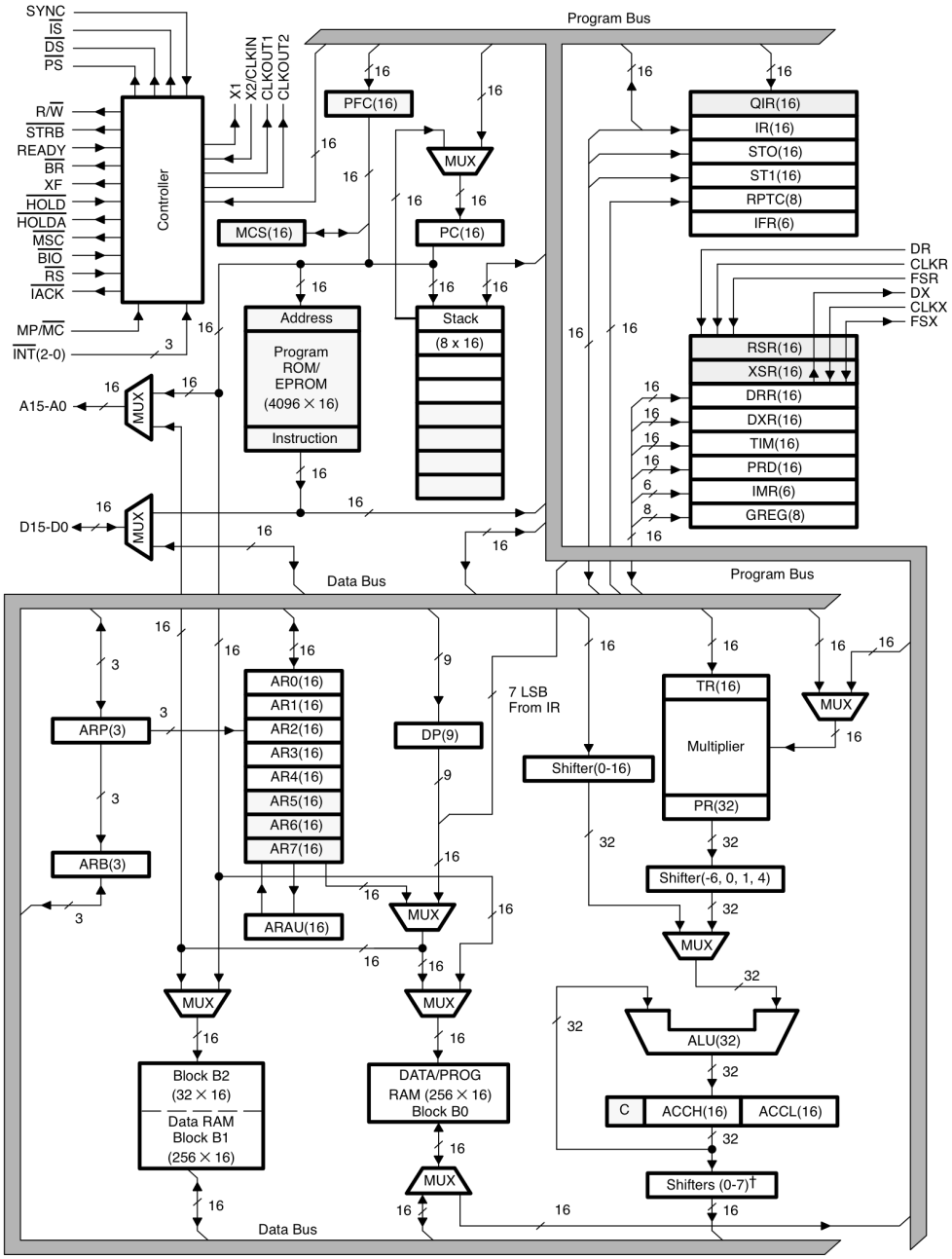


Figura 4.2. Diagrama funcional del TMS320C25 (SMJ320C25DSP. Texas Instruments).

- Memoria interna: Memoria EPROM (programa), memoria RAM de datos (B1 y B2) y memoria RAM configurable como datos o programa (B0).

- *Interface* con memoria local**:** 1) *Bus* de datos de 16 *bits* (D15-D0); 2) *Bus* de direcciones de 16 *bits* (A15-A0); 3) *Bus* de control: DS, PS, IS, R/W, STROBE, READY. El espacio de E/S consiste en 16 puertos de entrada y 16 de salida. Los puertos correspondientes a periféricos internos están tratados como memoria mapeada (accesibles como posiciones de memoria interna).

3.5.1. Organización de memoria

Tamaño

Los TMS320C2x utilizan *buses* de direcciones de 16 *bits*, por lo que pueden direccionar un total de 64k palabras de memoria de programa y 64k palabras de memoria de datos. La memoria interna (RAM/ROM/EPROM), está localizada en las direcciones más bajas del espacio de memoria, y es una memoria rápida que permite accesos en un solo ciclo de reloj. La expansión hasta 64k palabras externa se puede realizar con memorias lentas utilizando la señal de control READY. Los *buses* externos permanecen inactivos durante accesos a la memoria interna para evitar conflictos de direccionamiento.

El TMS320C25 dispone de 544 palabras de 16 *bits* de memoria interna RAM, divididas en tres bloques (B0, B1, y B2), de los cuales B0 puede configurarse como memoria de programa o datos. Dispone también de 4 kpalabras de memoria ROM.

El TMS320C26 dispone de 1568 palabras de 16 *bits* de memoria interna RAM, divididas en cuatro bloques (B0, B1, B2 y B3), de las cuales 32 palabras son siempre memoria de datos (B2), mientras que las restantes pueden configurarse como memoria de programa o datos. El TMS320C26 dispone también de 256 words de memoria ROM que contiene un programa cargador.

Direccionamiento de la memoria

El direccionamiento se realiza mediante diversos *buses* de direcciones asociados a cada espacio de memoria:

- Memoria de programa: mediante el PAB (*bus* de direcciones de la memoria de programa) generado por el PC.

- Memoria de datos: mediante el DAB (*bus* de direcciones de la memoria de datos) direcciona la memoria de datos de dos maneras distintas:

    1. Mediante el *bus* de direcciones directo (DRB) utilizando modo de direccionamiento directo. En este caso, se utiliza el registro puntero de página de memoria de datos (DP), que divide la memoria en 2^9=512 páginas de 2^7=128 palabras.

    2. Mediante el *bus* del fichero de registros auxiliares (AFB), usando el modo de direccionamiento indirecto. Utiliza el generador de direcciones del procesador, que consta de:

        - Un fichero de 8 registros auxiliares de 16 *bits* (AR0-AR7). Generalmente contienen direcciones, pero pueden ser utilizados también para almacenamiento temporal de datos.
        - Una ALU para datos enteros que permite calcular direcciones utilizando los registros auxiliares (ARAU).
        - ARP: Puntero de 3 *bits* (2^3=8) que direcciona el registro auxiliar actual (activo).
        - ARB (ARP buffer) para almacenar el ARP durante llamadas a subrutinas e interrupciones.

Los registros auxiliares están conectados a una de las entradas de la ARAU, que puede autoindexar el AR actual mientras la posición de memoria está siendo direccionada. El indexado puede modificarse por +1, -1 o según un cierto valor, en función de la otra entrada, que puede ser AR0 o los 8 *bits* bajos del Registro de Instrucciones (IR).

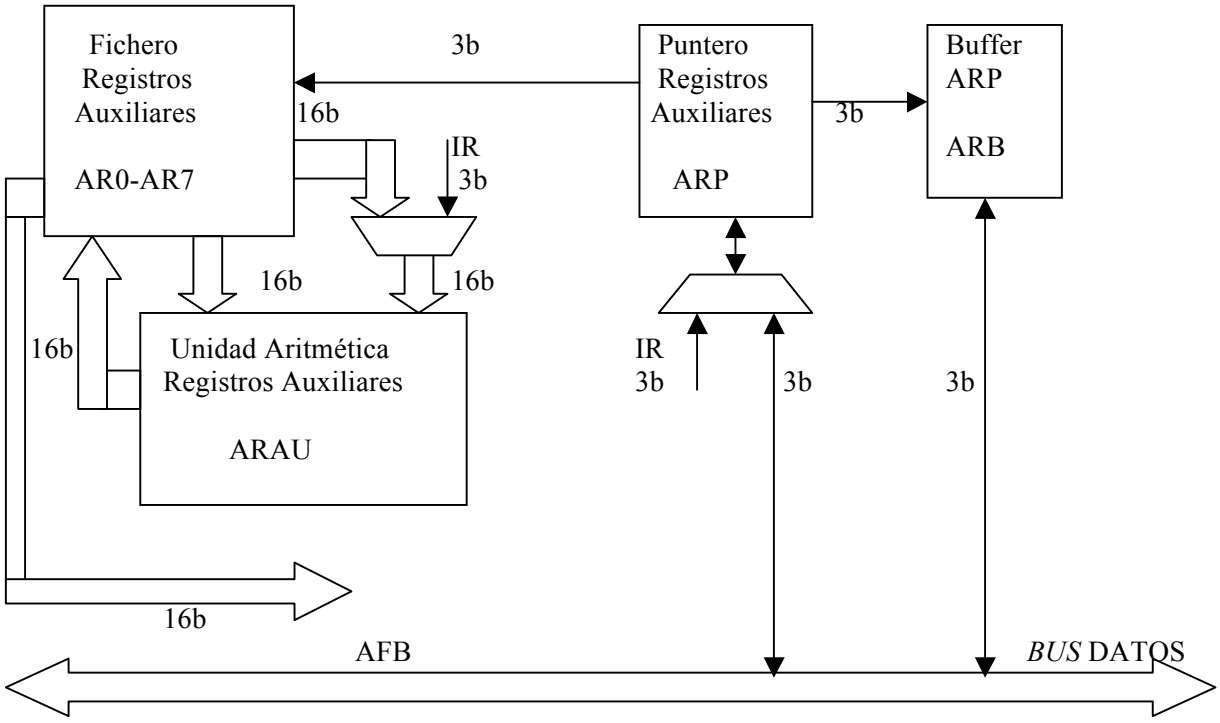


Figura 3.6. ARAU y fichero de registros auxiliares.

Mapas de memoria y registros mapeados

Debido a la separación de memoria de programa y datos, los mapas de memoria están duplicados. Además, existen varios factores que modifican la distribución de los mapas de memoria en el DSP:

- Señal de control MP/MC (microprocesador/microcomputador): Controla el mapa de memoria haciendo que utilice EPROM externa o interna, respectivamente.
- B0 puede configurarse como memoria de datos o programa a través de las instrucciones CNFD/CNFP (TMS320C25).
- B0, B1 y B3 pueden configurarse como memoria de datos o programa a través de las instrucciones CNFD#/CNFP# (TMS320C26).

Se obtienen, pues, diversas combinaciones en función de estas opciones, tales como se muestran en las siguientes figuras. Cabe destacar:

- Las posiciones bajas de la memoria de programa están reservadas a los vectores de interrupción.

- Las posiciones bajas de la memoria de la memoria de datos están ocupadas por los registros internos del procesador, y pueden ser direccionados como posiciones de memoria convencionales (registros mapeados). Sus direcciones son:

REGISTRO	DIREC.	DEFINICION
DRR	0	Reg. recepción
DXR	1	Reg. transmisión
TIM	2	Reg. temporizador
PRD	3	Reg. Periodo
IMR	4	Reg. Máscara interrupción
GREG	5	Reg. Memoria global

- GREG (*global register*) permite definir la zona de memoria externa (direcciones altas) dedicada a memoria global cuando el DSP trabaja en multiproceso.

- Las posiciones reservadas son utilizadas por el procesador, y no deben ser usadas, proporcionando un contenido imprevisible cuando son leídas.

- Las direcciones de la memoria de datos se muestran como direcciones absolutas (izquierda), correspondiendo al caso de direccionamiento indirecto, y como páginas, para el caso de direccionamiento directo.

- B0 reside en las páginas 4 y 5 del mapa de memoria de datos cuando es configurado como memoria de datos, y en las posiciones FF00h-FFFFh cuando se configura como programa (en este caso, las páginas 4 y 5 no son direccionables) (TMS320C25).

- B0, B1 y B3 residen en las páginas páginas 4-7 (para B0), 8-11 (para B1), 12-15 (para B3) del mapa de memoria de datos cuando son configurados como memoria de datos, y en las posiciones FA00h a FFFFh cuando se configuran como programa (TMS320C26).

- La memoria externa del TMS320C26 comienza en la dirección 1000h para mantener compatibilidad con la ROM del TMS320C25.

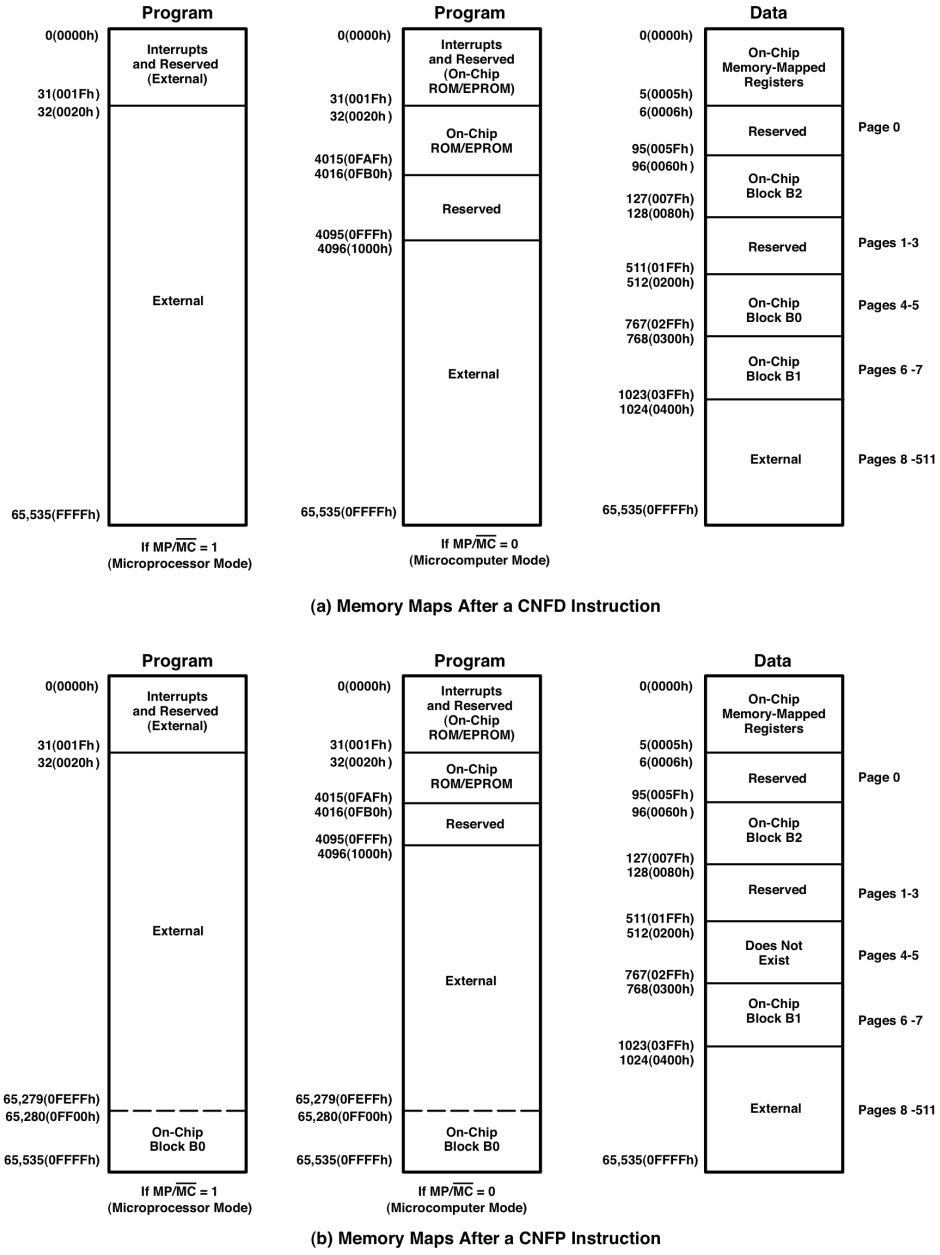


Figura 3.7. Mapas de memoria del TMS320C25 (SMJ320C25DSP. Texas Instruments).

3.5.2. Unidad Central Aritmético-Lógica (CALU)

La CALU del TMS320C2x consta de:

- **Multiplicador:** implementa productos entre dos número de coma fija sin signo y en complemento a dos, dando un resultado de 32 *bits* que está también en formato de complemento a dos. El registro TR (16 *bits*) contiene uno de los operandos de entrada y la salida se almacena en PR (32 *bits*). Admite opciones de signo en los operandos de entrada.

- **ALU**: utiliza operandos de 16 *bits* e implementa operaciones aritméticas y lógicas sobre ellos, produciendo una salida de 32 *bits* que es almacenada en el acumulador. Una entrada de la ALU es siempre el ACC, y la otra puede ser transferida desde el Registro Producto (PR) del multiplicador o desde el desplazador de escalado, previamente cargado con un dato de memoria.

- **Acumulador:** tiene 32 *bits* y puede utilizarse en dos partes de 16 *bits*, ACCL y ACCH.

    - El acumulador soporta operaciones de desplazamiento y rotación. Cuando el propio ACCH es desplazado a izquierdas, los MSB del ACCH se pierden y los LSB se transfieren desde el ACCL. Cuando ACCL es desplazado a izquierdas, los LSB se rellenan con ceros y los MSB se pierden.
    - Soporta modo de extensión de signo. Cuando se activa este modo, el desplazamiento del acumulador a derechas mantiene el signo del dato en ACC. Si no, implementa un desplazamiento lógico a derechas, introduciendo un '0' en MSB.
    - No soporta opción de redondeo, por lo que ésta debe realizarse por *software*.
    - Soporta aritmética saturada, cargándose con el número más grande positivo o negativo, dependiendo del sentido de la saturación (7FFFFFFFh o 80000000h), en caso de desbordamiento.

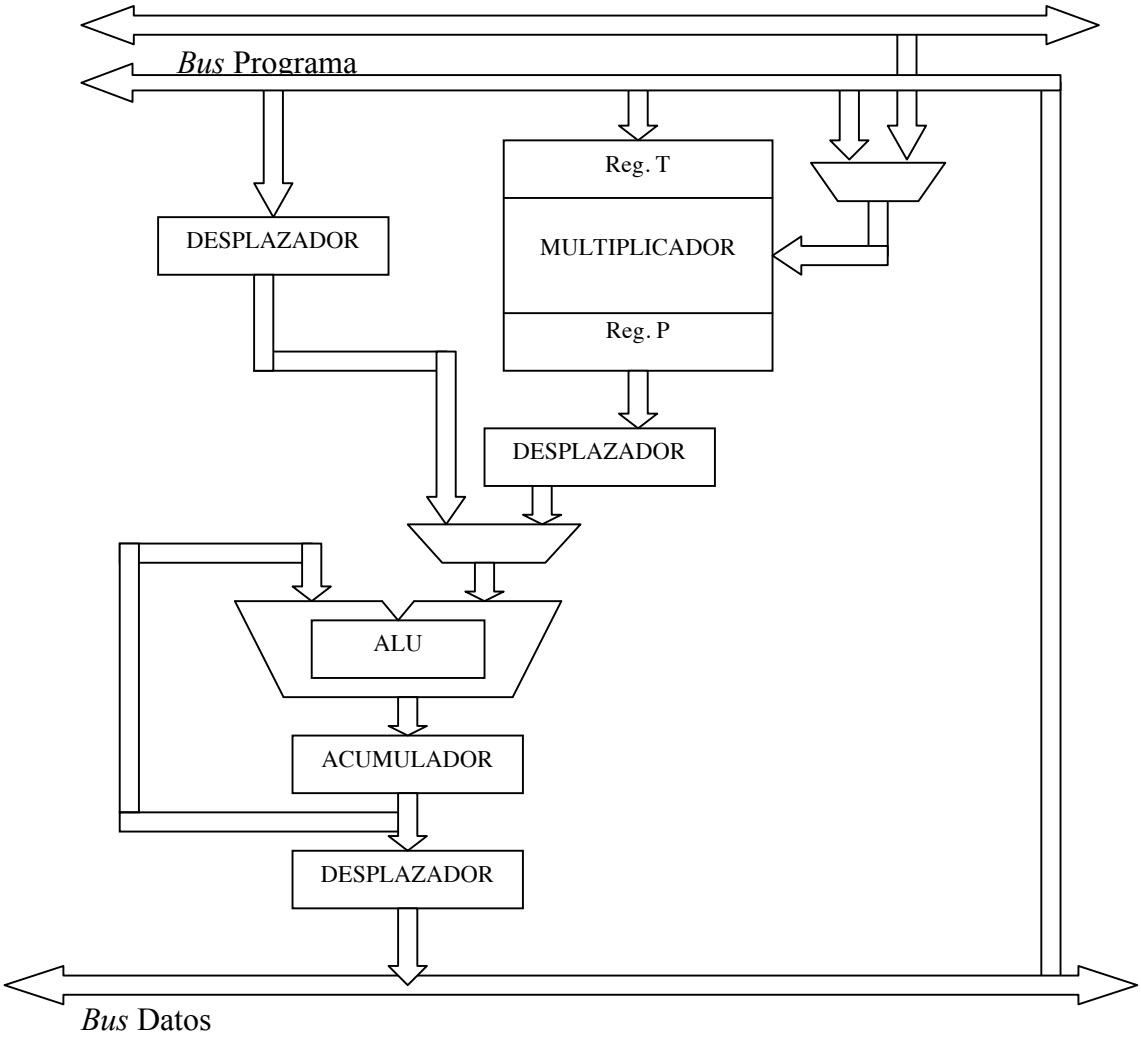


Figura 3.9. Unidad Central Aritmético-Lógica (CALU).

* **Desplazadores:** La CALU dispone de 3 desplazadores:

  * Desplazador de escalado: constituye una de las entradas de la ALU y permite implementar funciones de escalado de los datos durante la transferencia desde memoria, sin utilizar, por tanto, ciclos de reloj. Puede programarse desplazamientos a izquierdas de 0 a 15 *bits* del dato de entrada. Los LSB se rellenan con ceros para ajustar el dato de 16 a 32 *bits*.

* Desplazador del acumulador: desplazamientos a la izquierda de 0 a 7 *bits* en el proceso de almacenamiento del ACCH o ACCL en memoria, por lo que el contenido de éste permanece inalterado.
* Desplazador del multiplicador: permite desplazamiento a izquierdas 1 ó 4 *bits*, para eliminar signo redundante o justificar datos, y a derechas 6 *bits* para permitir la ejecución de hasta 128 multiplicaciones/sumas consecutivas sin desbordamiento.

3.5.3. Control del sistema

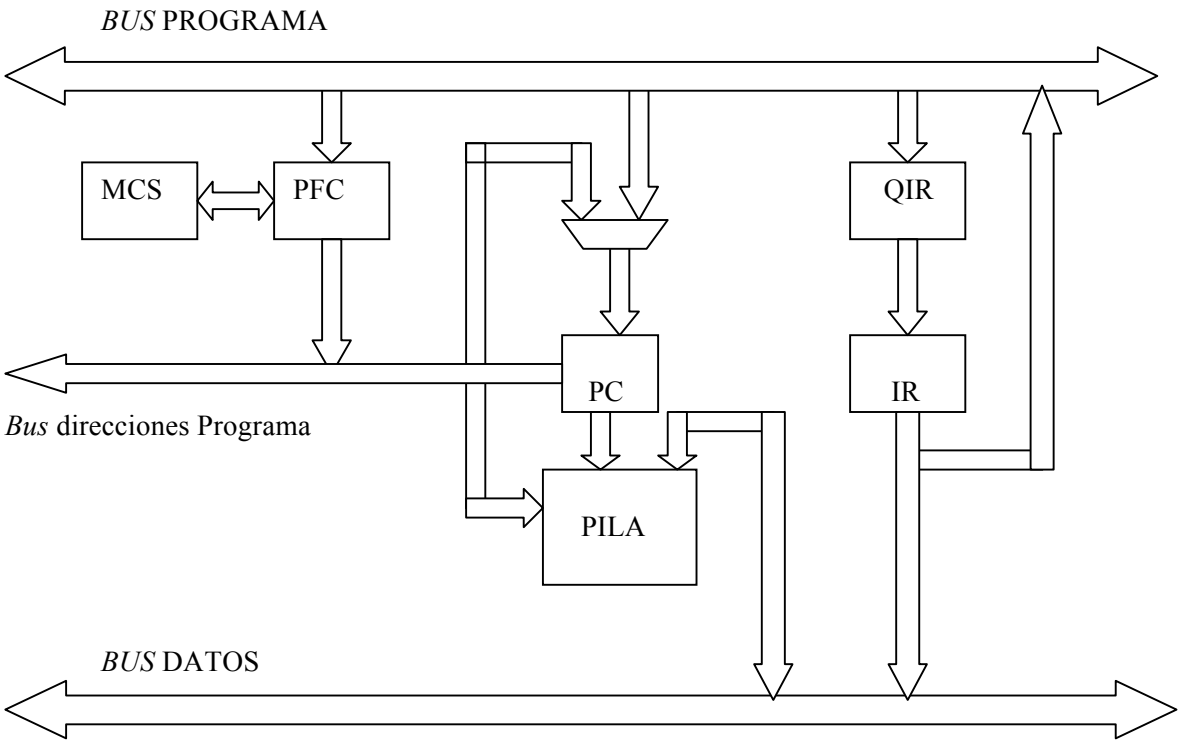


Figura 3.10. PC, pila y *hardware* asociado.

Contador de programa y pila

El contador de programa (PC) direcciona memoria interna (ROM interna y bloques RAM cuando están configurados como memoria de programa) y/o externa mediante el *bus* de direcciones de programa (PAB). Direcciona también la memoria de datos al ejecutar una instrucción BLKD.

La pila consiste en 8 niveles de 16 *bits*, utilizados en servicio de interrupciones. El TMS320C25 dispone de una pila suplementaria (MCS) para ciertas instrucciones que utilizan una posición de pila.

Estructura *pipeline*

La *pipeline* del TMS320C25 consta de tres fases: búsqueda-decodificación-ejecución (cuando se trabaja con memoria interna, puede reducirse a dos fases). Su implementación utiliza el contador de pre-búsqueda (PFC), la pila de micro-llamada (MCS), el registro de instrucciones (IR) y la cola del registro de instrucciones (QIR). El PFC contiene la dirección de la siguiente instrucción a ser pre-buscada. Esta instrucción será cargada en IR, excepto si IR contiene ya una instrucción previa en fase de ejecución, en cuyo caso se almacenará en QIR. El PFC es incrementado y, después de que la actual instrucción se haya ejecutado, IR se carga con el contenido de QIR. PC contiene la dirección de la siguiente instrucción a ser ejecutada, sirviendo como puntero de referencia de la posición actual en el programa.

Registros de estado

ST0 y ST1 contienen los estados de diversas condiciones y modos de funcionamiento. Los *bits* de estado se modifican con las instrucciones LST/LST1 y SST/SST1, excepto algunos que dispone de instrucciones específicas. Los *bits* reservados toman el valor '1' cuando se lee el registro. La siguiente figura muestra los campos en que se dividen estos registros.

Además de los flags usuales (C: acarreo, OV: desbordamiento), los registros de estado contienen los registros DP, ARP y ARB, así como otros flags que controlan distintos modos de funcionamiento del procesador. Los más importantes son:

- OVM: activa el modo de aritmética saturada.
- INTM: activa o desactiva globalmente las interrupciones enmascarables.

- CNF: configura el bloque B0 de RAM como memoria de programa o datos.
- SXM: activa el modo de extensión de signo.
- HM: controla el funcionamiento durante la activación de la entrada HOLD. Con valor '1', el procesador detiene su funcionamiento al activar dicha entrada. Con valor '0', puede seguir funcionando con su memoria interna, pero no puede acceder a los *buses* externos.
- FSM, FO, TXM: controlan el funcionamiento del puerto serie.
- TC: usado para operaciones sobre bits.
- XF: controla el puerto de bit externo XF.
- PM: controla el desplazador de salida del multiplicador. 0: sin desplazamiento; 1: 1 bit a la izquierda; 2: 4 bits a la izquierda; 3: 6 bits a la derecha.

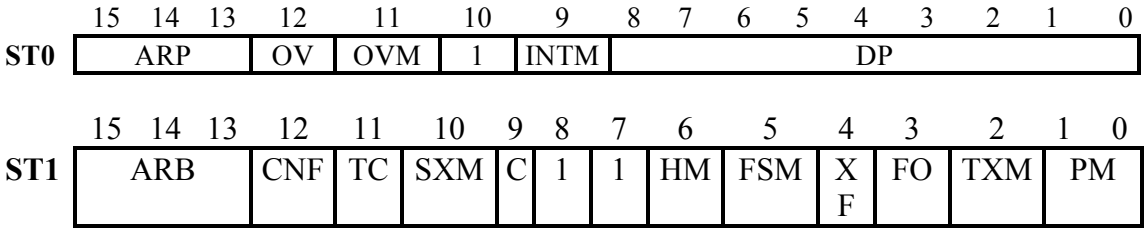


Figura 3.11. Organización de los registros de estado.

Contador de bucles

El contador de bucles (RPTC) permite implementar bucles de una única instrucción con cero estados de penalización. Utiliza un contador *hardware* descendente de 8 bits que permite repetir la siguiente instrucción del programa N+1 veces, siendo N un valor entre 0 y 255.

Modos de bajo consumo

Los dos modos de bajo consumo (*powerdown*) que soporta el TMS320C25 hacen entrar al dispositivo en un estado durante el cual consume aproximadamente la mitad que en el estado normal. El contenido de los registros y memorias internas se mantiene.

La entrada en estos modos se produce mediante la ejecución de la instrucción IDLE y activando el pin HOLD. Los *buses* externos permanecen en alta impedancia. Se sale de este modo desactivando HOLD o por una interrupción.

3.5.4. Interrupciones

El TMS320C25 soporta interrupciones vectorizadas. Los tipos disponibles son:

* 3 interrupciones externas (INT2-INT0): generadas por el usuario.
* Interrupciones internas: generadas por el puerto serie (RINT y XINT), por el timer (TINT) y por la instrucción de interrupción *software* (TRAP).
* RESET (no enmascarable).

INTERRUPCION	POSICION MEMORIA	PRIORIDAD
RS	0h	1 (mayor)
INT0	2h	2
INT1	4h	3
INT2	6h	4
Reservadas	8-17h	
TINT	18h	5
RINT	1Ah	6
XINT	1Ch	7 (menor)
TRAP	1Eh	sin prioridad

Figura 3.13. Vectores de interrupción y prioridades.

Las posiciones de los vectores y las prioridades de las interrupciones se muestran en la tabla (figura 3.13). La interrupción generada por TRAP no está priorizada, pero posee su propio vector de interrupción. Cada dirección de interrupción ocupa dos posiciones, por lo que puede sustituirse el vector por una instrucción de salto a una dirección de servicio.

Registros asociados

Cuando se produce una interrupción, se almacena en el IFR (Interrupt Flag Register). Permanece almacenada hasta que es reconocida, siendo entonces desactivada por IACK (interrupt acknowledge) o RS (reset). La señal RS no es almacenada en el IFR. El IFR no puede accederse por *software*.

El registro de máscara de interrupción (IMR: Interrupt Mask Register) permite enmascarar las interrupciones internas y externas. Un '1' en cualquier posición de B5 a B0 la activa, siempre que INTM=0. Cuando se lee IMR, los *bits* 6 a 15 aparecen a '1'.

B15 ..... B6	B5	B4	B3	B2	B1	B0
RESERVADO	XINT	RINT	TINT	INT2	INT1	INT0

Figura 3.14. Registro de máscara de interrupciones (IMR).

Activación de interrupciones

Las interrupciones enmascarables se desactivan globalmente con INTM=1, y se activan con INTM=0. INTM es puesto a '1' por IACK, la instrucción DINT o por un *reset*. Es puesta a cero por las instrucciones EINT y IDLE.

3.5.5. Periféricos internos

Temporizador

El temporizador se utiliza como reloj de muestreo o en acceso a periféricos. La señal de reloj de entrada, CLKOUT1, se genera a partir del reloj maestro (MASTERCLOCK) del procesador tras un escalado por 4. Sus registros asociados son:

* TIM: contador descendente de 16 bits conectado a CLKOUT1 en el TMS320C25. TIM contiene la cuenta actual del temporizador. A cada ciclo de reloj, TIM se decrementa, y produce una interrupción (TINT) cuando alcanza el cero.

* PRD: contiene el valor inicial, que se carga de nuevo en TIM cada final de cuenta. PRD puede programarse entre 1 y 65535.

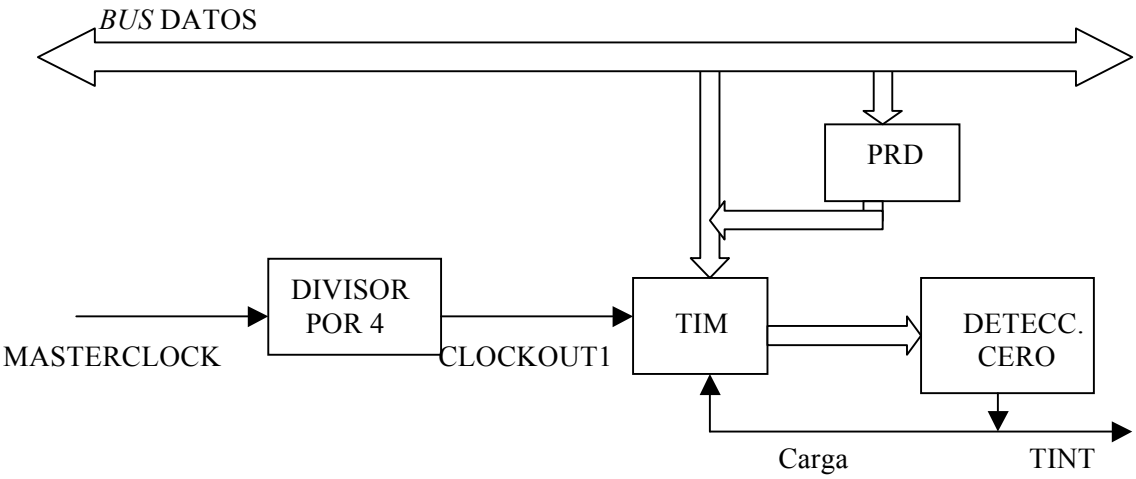


Figura 3.12. Diagrama de bloques del temporizador.

Puerto serie

El TMS320C2x dispone de un puerto serie integrado para comunicación *full-duplex*, que permite conectarlo directamente con otros dispositivos serie de E/S como *codecs*, conversores A/D, etc. Puede utilizarse también para intercomunicar diversos procesadores en aplicaciones multiproceso.

La estructura del puerto serie se muestra en la siguiente figura. Las secciones de transmisión y recepción están implementadas separadamente para permitir operaciones simultáneas. Pueden trabajar en modo *byte* (8 *bits*) o modo palabra (16 *bits*). Cada registro tiene un reloj externo, un patrón de pulsos de sincronización y registros de desplazamiento asociados. DRR y DXR están mapeados en las posiciones 0 y 1 del espacio de direcciones de datos. XSR y RSR, los desplazadores asociados, no son accesibles por *software*.

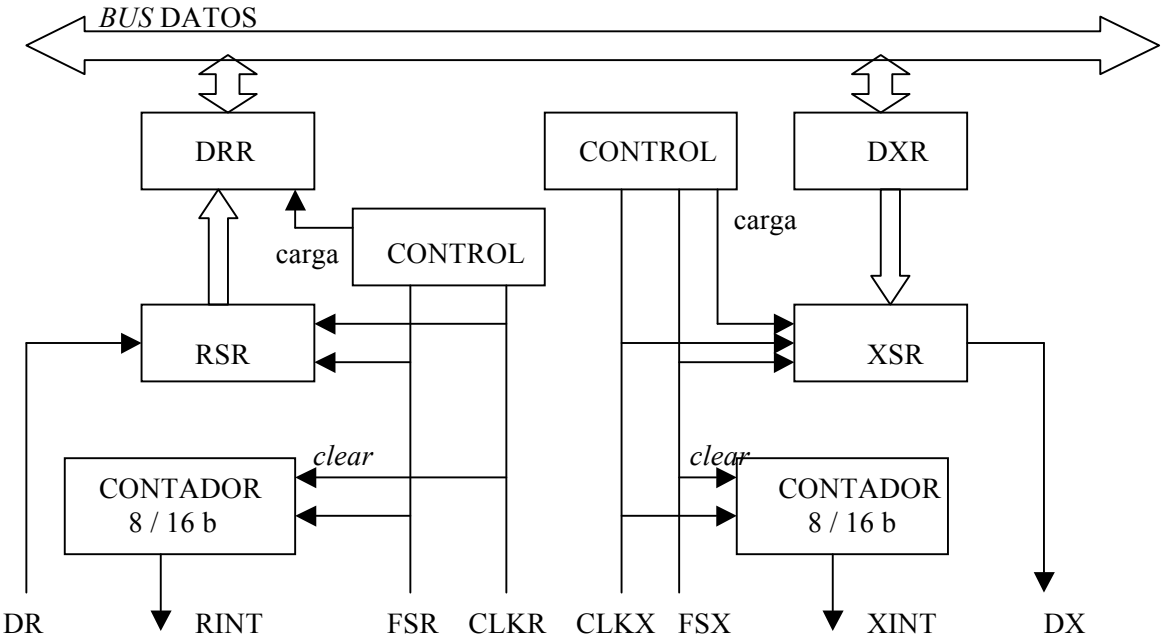


Figura 3.15. Diagrama de bloques del puerto serie.

Los flags de control del puerto serie, que forman parte del registro de estado ST1, son:

* FO: define el formato del dato a transmitir como byte (FO=1) o palabra (FO=0).

* FSM: define si se necesita patrón de sincronismo para la transmisión ('1'), por lo que un pulso en FSX hará que XSR se cargue con el contenido de DXR y comience la transmisión, o no ('0').

* TXM: determina si el patrón de sincronización se genera externa o internamente. Si TXM=1, FSX se configura como salida y se produce un pulso de sincronismo cada vez que DXR es cargado. Si TXM=0, FSX se configura como entrada y el DSP espera que llegue un pulso de sincronismo externo antes de comenzar la transmisión.

Modos de operación

La siguiente figura muestra el funcionamiento de la operación de transmisión del puerto serie (la operación de recepción funciona de forma similar). Las fases para la transmisión de un dato son:

1. Se escribe el dato en DXR.
2. Se copia DXR en XSR y se genera el pulso de sincronismo en FSX (en este caso, suponemos que se utiliza sincronismo interno TXM=1).
3. Se trasmite el dato.
4. Cuando todos los *bits* se han transmitido, se genera una interrupción interna XINT.

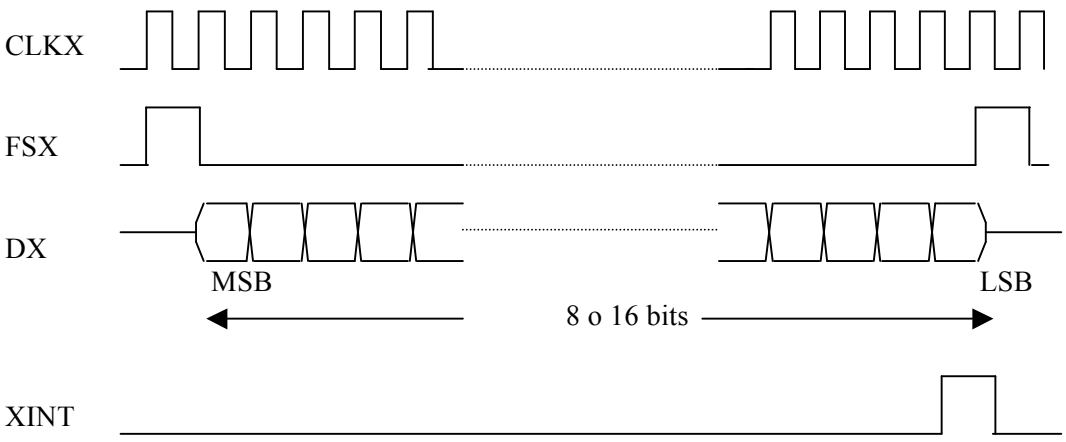


Figura 3.17. Diagrama de transmisión del puerto serie.

El puerto serie soporta dos modos de transmisión:

- Modo '*burst*': En este modo, las transferencias están separadas por periodos de inactividad del puerto. Utiliza patrón de sincronismo para la transmisión.

- Modo continuo: Permite transferencias continuas de datos, es decir, después de transferir el último bit del dato anterior se comienza con el primer bit del siguiente sin periodos de inactividad. Puede transferir con y sin patrón de sincronismo.