PROCESAMIENTO DIGITAL DE SEÑALES

SEGUNDA PARTE

MICROCONTROLADORES Y REALIZACIÓN DE LOS FILTROS CON TMS320CXX

BOHUMIL PŠENIČKA

LARRY ESCOBAR

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

DEPARTAMENTO DE TELECOMUNICACIONES

MÉXICO 1997

PŠENIČKA, Bohumil y Larry Escobar. *Procesamiento digital de señales. Microcontroladores y realización de los filtros con TMS320CXX, 2ª parte*. México, UNAM, Facultad de Ingeniería, 1997, 178 pp., ils.

Procesamiento digital de señales. Microcontroladores y realización de los filtros con TMS320CXX, 2ª parte.

Ciudad Universitaria, México, D.F.

Primera edición, septiembre 1997.

ISBN 968-36-6473-3
Impreso en México.

Prólogo

Con el avance de la tecnología, la computadora se ha convertido en una herramienta fundamental en la simulación y el procesamiento digital de señales (PDS), por lo cual constituye una solución real para los problemas de medición, control, filtrado, análisis espectral, sistemas de comunicaciones, etc.

Entre las áreas de interés que conforman el PDS están el análisis de señales y sistemas, el análisis y síntesis de filtros digitales, la identificación de sistemas, la ingeniería de software, la arquitectura de microcomputadoras y el diseño con circuitos con muy alta escala de integración (VLSI). Las tareas del PDS se realizan eficiente y sistemáticamente a través de sistemas que combinan la tecnología de los circuitos VLSI con una gran variedad de algoritmos matemáticos y con el software necesario para su implementación.

Uno de los objetivos de un sistema de PDS es proveer una mejor aproximación al análisis o estimación contenido de la información. El análisis de señales es el proceso de definir y cuantificar las características de una señal o para una aplicación. Con el desarrollo de las comunicaciones, el PDS y los procesadores de señales digitales (DSP), que se realizan mediante los microcontroladores, se han convertido en una herramienta necesaria para cualquier sistema digital, por lo que la tendencia al uso de estos sistemas es cada día mayor.

A fin de fortalecer la asignatura de Procesamiento digital de señales que cursan los estudiantes de las carreras de electrónica, telecomunicaciones y computación, las cuales se imparten en la Facultad de Ingeniería de la UNAM, y contribuir a la enseñanza y aprendizaje de los DSP, los autores han tenido el empeño de elaborar este libro con objeto de explicar cómo se realizan los filtros digitales mediante los microprocesadores de la familia de DSP de TMS320 de la compañía Texas Instruments, principalmente el TMS320C25 de aritmética entera y el TMS320C30 de aritmética flotante, así como describir sus características en hardware y software, sus formas de programarlos, dar ejemplos y algunas aplicaciones.

El hecho de elegir esta familia es porque son los DSP que más se han difundido en el mercado y los que un ingeniero encontraría en la práctica profesional, a pesar de este enfoque, también se proporciona a los alumnos las bases de lo que es un DSP y las formas de presentar algoritmos propios del PDS, lo que da lugar a que cualquier DSP diferente a los tratados no sea difícil de utilizar.

El libro está dividido en 9 capítulos. El primer capítulo fue escrito por el Ing. Larry Escobar y los demás por el Dr Bohumil Pšenička. En los primeros tres se presenta la descripción y arquitectura del TMS320C25 con las instrucciones para escribir los progra-

mas en ensamblador y los ejemplos para la realización de los filtros digitales en cascada, en paralelo, de estado y en la forma de cruz. En los capítulos cuatro, cinco y seis se explica la arquitectura del TMS320C30, herramientas de desarrollo, la EVM y realización de filtros digitales. En los capítulos siete y ocho se describen la arquitectura y el depurador de TMS320C50. En el último capítulo los estudiantes encuentran algunas instrucciones útiles para realizar y analizar los filtros digitales mediante el paquete MATLAB.

Los autores esperan que este libro sea una contribución sólida para los desafíos tecnológicos del futuro que tendrán que enfrentar los ingenieros en telecomunicaciones, electrónica y en computación. Asimismo, agradecen el apoyo a la Facultad de Ingeniería de la UNAM y a las personas que colaboraron en el proceso de edición para que esta obra llegara a publicarse.

Dr. Bohumil Pšenička
Ing. Larry Escobar

Profesores de la Facultad
de Ingeniería de la UNAM

Contenido

Índice de abreviaturas

ACC- Acumulador.
ACCH- La parte alta del acumulador.
ACCL- La parte baja del acumulador.
ARAU- Unidad aritmético lógica de los registros auxiliares.
ARB- Buffer de los registros auxiliares.
ARP- Apuntador de los registros auxiliares.
ARx- Registros auxiliares
CALU- Unidad aritmética lógica central.
CNFD- Configura bloque B0 en la memoria de datos.
CNFP- Configura bloque B0 en la memoria de programa.
COFF- El archivo ejecutable para EVM de formato común.
DAB- Bus de datos.
DP- Apuntador de página.
DR- Pin del puerto de serie para recepción.
DRR- Registro para recepción en la forma serial.
DSP- Procesamiento digital de señales.
DSK- Archivo ejecutable para starter-kit.
DX- Pin del puerto de serie para transmisión.
DXR- Registro para transmisión en la forma serial.
EVM- Módulo de evaluación.
FIR- Filtros con respuesta finita.
GREG- Registro de memoria global.
IE- Registro de habilitación.
IF- Registro de banderas.
IFR- Registro de bandera para interrupción.
IMR- Registro interrupción mascarable.
IIR- Filtros con respuesta infinita.
IR0- Registro índice 0.
IR1- Registro índice 1.
MIPS- Millones de instrucciones por segundo.
MUX- Multiplexor.
PAB- Bus de programa.
PFC- Registro para direccionar B0.
PC- Contador de programa.
PR- Registro P.
Rx- Registro de precisión extendida.
RE- Registro de dir. final de repetición.
RPTC- Registro de repetición.
RS- Registro de dir. inicial de repetición.
RSR- Registro de corrimiento para recepción serial.
ST0- Registro de estado 0.
ST1- Registro de estado 1.
TBC- Controlador de canal de pruebas.
TR- Registro T.
VLSI- Muy alta escala de integración.

Capítulo 1

Descripción y arquitectura del TMS320C25

La familia de procesadores TMS320 de 16/32 bits de Texas Instruments, empleada para el procesamiento digital de señales (DSP), combina la flexibilidad de un controlador de alta velocidad con la capacidad numérica de un procesador y ofrece una opción para la tecnología VLSI (muy alta escala de integración) y el multiprocesamiento de señales.

Dichas características hacen que esta familia presente una alta capacidad de ejecución, arquitectura de gran funcionalidad y relación costo/efectividad ideal para solucionar diversos problemas en telecomunicaciones, computación, comercio, industria y aplicaciones militares, entre otras.

1.1 Descripción general

La combinación de la arquitectura tipo Harvard de la familia TMS320 (separación de los buses de datos y datos de programa) y su conjunto de instrucciones especiales para el DSP proveen una alta velocidad de ejecución y gran flexibilidad. De esta forma, se produce una familia de procesadores capaces de ejecutar 10 MIPS (millones de instrucciones por segundo), ya que se realizan funciones en hardware, que en comparación con otros procesadores, recurren al software o microcódigos para generarlas.

La segunda generación de la familia TMS320 está compuesta de dos procesadores: el TMS32020 y el TMS320C25. La arquitectura de ambos procesadores se basa en la del TMS32010. El TMS32020 trabaja a 20 MHz y ejecuta el doble de instrucciones del TMS32010, mientras que el TMS320C25 incrementa la funcionalidad de la arquitectura en relación al TMS32020.

1.2 Características

- Ciclo de instrucción 100 ns.
- 544-palabras de datos programables en memoria RAM interna.
- 4k-palabras de programa en memoria ROM interna.

- 128k-palabras de espacio total de memoria DATOS y PROGRAMA.
- ALU/acumulador de 32 bits.
- Multiplicador paralelo de 16 × 16 bits con producto de 32 bits.
- Instrucciones de multiplicación/acumulación ejecutadas en un ciclo simple de instrucción.
- Instrucciones de repetición para uso eficiente del espacio de programa y mejor ejecución.
- Movimiento de bloques para el manejo de DATOS/PROGRAMA.
- Timer integrado para operaciones de control.
- Ocho registros auxiliares con una propia unidad aritmética.
- Un stack por hardware de ocho niveles.
- Dieciséis canales de entrada o salida.
- Un registro de corrimiento paralelo de 16 bits.
- Posibilidad de generar tiempos de espera para comunicación a periféricos o memorias de respuesta lenta.
- Un puerto serie para interfaz directa a "codecs"(codificador-decodificador).
- Entrada de sincronización para sincronía en configuraciones de múltiples procesadores.
- Interfaz de memoria de datos global.
- Compatibilidad de códigos fuente del TMS32010.
- Concurrencia con DMA usando una operación de espera extendida.
- Instrucciones para realizar filtros, la transformada rápida de Fourier y aritmética de precisión extendida.
- Generador de reloj interno.
- Suministro de potencia de 5 volts.

1.3 Arquitectura

La destacada funcionalidad del TMS320C25 (TMS) para el procesamiento digital de señales (DSP) se debe a su arquitectura tipo Harvard, que maximiza el procesamiento mediante dos estructuras de buses de memoria separadas (memorias de programa y datos) para incrementar la velocidad de ejecución, contando con instrucciones que realizan la transferencia de datos entre ambos espacios.

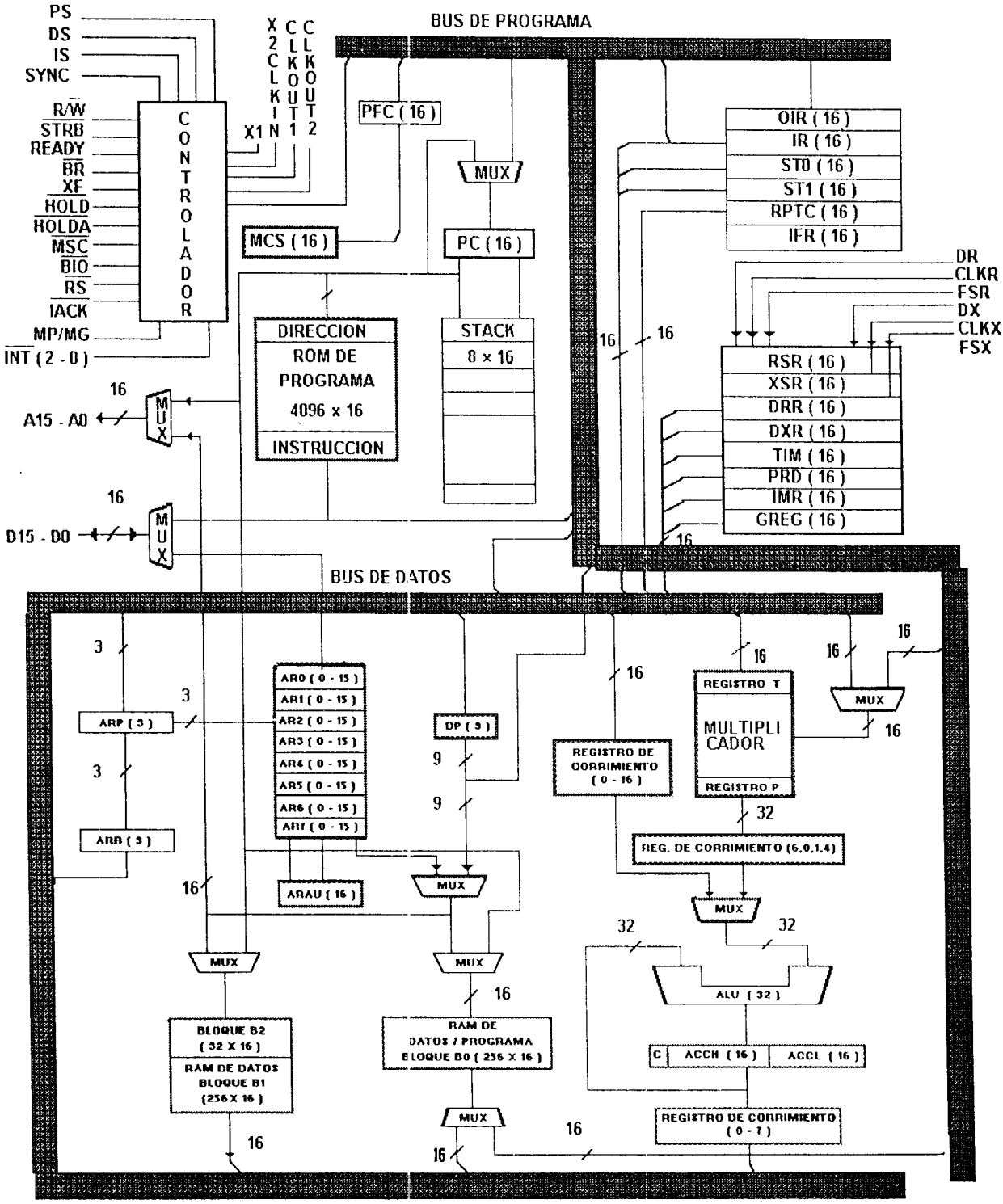


Figura 1.1. Arquitectura del TMS325C25

Externamente, las memorias de programa y datos son multiplexadas sobre el mismo bus para maximizar el intervalo de direccionamiento entre dichos espacios, mientras se minimizan los pines del dispositivo.

La flexibilidad en el diseño del sistema se incrementa con tres bloques de datos internos en RAM (un total de 544 palabras), donde uno de ellos puede ser configurado como memoria programa o memoria dato. El TMS es capaz de direccionar externamente 64k-palabras en un espacio de memoria dato para facilitar la realización de algoritmos para DSP.

La memoria interna ROM mascarable de 4k-palabras puede ser usada para reducir el costo de sistemas. Los programas de 4k-palabras pueden ser mascarables en la memoria ROM interna y, de esta forma, ejecutados a alta velocidad desde este espacio de memoria. Externamente el espacio de memoria de programa direccionable es de 64k-palabras.

El TMS funciona con una aritmética en modo dos complemento, empleando una ALU y un acumulador de 32 bits. La ALU es una unidad aritmética de propósito general que opera con palabras de 16 bits provenientes de la RAM de datos, o derivadas de instrucciones inmediatas, o por el empleo del registro resultado (producto) del multiplicador de 32 bits. La ALU puede efectuar operaciones booleanas. El acumulador almacena los resultados de la ALU y a su vez es una segunda entrada a la ALU. La longitud total del acumulador es de 32 bits, la cual está dividida en dos partes, una alta (bits 31 al 16) y otra baja (bits 15 al 0), y para el manejo de datos se tiene instrucciones de almacenamiento para cada una de las partes (alta y baja) del acumulador.

El multiplicador realiza operaciones de 16×16 bits en modo dos complemento con un resultado de 32 bits en un solo ciclo de instrucción. El multiplicador está compuesto de tres elementos: el registro T (de 16 bits) para almacenar temporalmente el multiplicado, el registro P (de 32 bits) para almacenar el producto, y un arreglo multiplicador. Los valores del multiplicador provienen de la memoria dato, memoria programa (cuando se emplean las instrucciones MAC/MACD), o son derivados de una instrucción inmediata MPYK (multiplicación inmediata). La rapidez del multiplicador integrado permite la ejecución de operaciones fundamentales en el DSP como la convolución, correlación y filtrado.

El registro de corrimiento del TMS tiene una entrada de 16 bits y está conectado al bus de datos, mientras que su salida de 32 bits está conectada a la ALU. Este registro proporciona corrimientos a la izquierda de 0 a 16 bits sobre los datos de entrada y son programados mediante las instrucciones por medio del "shift"(corrimiento). Adicionalmente, se tiene la capacidad de que el procesador funcione con escalamiento numérico, extracción de bits, aritmética extendida y prevención de sobreflujo.

La interfaz local de la memoria del TMS consta de un bus de datos paralelo de 16 bits (D15-D0), un bus de direcciones de 16 bits (A15-A0), tres pines para selección de memoria dato/programa o espacio de entrada/salida (/DS, /PS e /IS) y varias señales de control del sistema. La señal R/w controla el sentido de transferencia del dato, y la señal /STRB provee un tiempo válido para la transferencia de información. Cuando se emplean las memorias ROM, RAM internas o memoria de programa externa de alta velocidad, el TMS ejecuta instrucciones a la máxima velocidad sin tiempos de espera. De otro modo, se utiliza la señal READY para generar tiempos de espera para comunicarse con memorias externas lentas.

El stack por hardware de ocho niveles almacena el contenido del contador de programa durante interrupciones y llamadas de subrutinas. Las instrucciones PUSH y POP permiten salvar y recuperar información contenida en el stack. Las interrupciones empleadas en el dispositivo son mascarables.

Las operaciones de control son proporcionadas en el TMS por un timer interno de 16 bits mapeado en memoria, un contador de repetición, tres interrupciones externas mascarables, y una interrupción interna generada por el puerto serie o el timer.

Un puerto serie interno full-duplex provee comunicación directa con dispositivos seriales como codecs, convertidores seriales A/D, y otros dispositivos seriales. Los dos registros del puerto serial mapeados en la memoria (registros de transmisión/recepción de dato) pueden operar en modo byte (8 bits) o modo word (16 bits). Cada registro tiene una entrada de reloj externa, una entrada de sincronia y registros de corrimiento. La comunicación serial puede ser usada entre procesadores en aplicaciones de multiprocesos.

El TMS para aplicaciones de múltiples procesadores tiene la capacidad de distribuir el espacio global de memoria de dato y de comunicación mediante las señales /BR (requerimiento de bus) y READY. El registro de distribución de la memoria global de dato (GREG) de 8 bits especifica hasta 32k-palabras de memoria dato como memoria global externa. El contenido del registro determina el tamaño del espacio global de memoria. Si la instrucción actual direcciona un operando dentro de este espacio, /BR se activa para requerir el control del bus. La extensión del ciclo de lectura/escritura de memoria se controla con la línea READY.

El procesador TMS soporta acceso directo a memoria (DMA) para su memoria externa dato/programa, para lo cual utiliza las señales /HOLD y /HOLDA. Otro procesador puede tomar por completo el control de la memoria externa del TMS por medio de la señal /HOLD (activa baja). Esta señal provoca que el TMS mantenga en estado de alta impedancia sus líneas de direccionamiento, datos y control. Sin embargo, de manera concurrente al modo DMA, el TMS continuará ejecutando su programa si éste opera con la RAM y ROM internas.

1.4 Diagrama de bloques funcional

La arquitectura del TMS320C25 está construida alrededor de dos buses: de programa y de datos. El bus de programa proporciona el código de instrucción y operandos inmediatos de la memoria de programa. El bus de datos interconecta varios elementos, como la unidad aritmética lógica central (CALU) y los registros auxiliares a los datos RAM. Tanto el bus de programa como el de datos pueden transferir datos de memoria dato RAM interna y de la memoria programa interna o externa al multiplicador en un ciclo de instrucción para realizar operaciones de multiplicación/acumulación.

El TMS320C25 tiene un alto grado de paralelismo, es decir, que mientras está operando sobre la CALU puede realizar operaciones aritméticas en la unidad aritmética de registros auxiliares (ARAU). Tal paralelismo resulta en un poderoso conjunto de operaciones aritméticas, lógicas y manipulación de bits que se ejecutan en un solo ciclo de máquina.

1.5 Organización de la memoria

El TMS320C25 posee un total de 544 palabras de 16 bits de RAM interna, de las cuales 288 palabras son siempre memoria de dato y las 256 restantes pueden ser configuradas como memoria de programa o dato. También provee 4k-palabras mascarables en programa ROM.

Las 544 palabras de RAM interna se dividen en tres bloques separados: B0, B1 y B2. El bloque B0 de 256 palabras es configurado como memoria de programa o dato por medio de instrucciones (CNFP o CNFD). Como memoria de dato dato, B0 reside en las páginas 4 y 5 del mapa de memoria de datos, y como memoria de programa de $> FF00$ a $> FFFF$, pudiéndose utilizar la instrucción BLKP (movimiento de bloque de memoria de programa a memoria de dato) para cargar la información del programa al bloque B0 cuando éste es configurado como RAM de datos. Las 288 palabras (bloques B1 y B2) siempre son configuradas como memoria de dato. B1 está localizado en las páginas 6 y 7, y abarca 256 localidades, mientras que B2 está en las 32 localidades superiores de la página 0.

La memoria de dato interna y las localidades reservadas para registros son mapeadas dentro de las primeras 1024 palabras del espacio total de memoria de dato (configuración con la instrucción CNFD).

Las 4k-palabras de ROM interna pueden ser un programa mascarable. Esta área de memoria permite la ejecución de programas a máxima velocidad sin necesidad de memorias externas veloces para almacenar el programa. El mapeo de estas 4k-palabras se realiza por medio del pin de selección MP/MC (microprocesador/microcomputadora). Manteniendo MP/MC en alto, el TMS mapea este bloque de memoria como externo, y MP//MC en bajo lo mapea en ROM interna (Ver mapa de memoria en la figura 1.2).

El TMS provee tres espacios de direccionamiento para memoria de programa, memoria de dato e I/O. Estos espacios son distinguidos externamente por medio de las señales /PS, /DS e /IS (programa, dato e I/O). Las señales /PS, /DS, /IS y /STRB son activadas sólo cuando un espacio externo de memoria se direcciona. Durante un direccionamiento interno, estas señales permanecen en estado inactivo alto para prevenir conflictos de direccionamiento cuando el B0 es configurado como memoria de programa.

Registros mapeados en memoria

En el mapa de memoria, los registros pueden ser accesados de igual manera como un dato localizado en la memoria de dato, excepto el movimiento de bloques que usa la instrucción BLKD, la cual no puede ser ejecutada en las localidades de memoria de los registros.

Registros auxiliares

El TMS posee 8 registros auxiliares seleccionados por un apuntador de registros auxiliares de 3 bits (ARP), cargándose con los valores de 0 a 7 para designar desde AR0 a AR7, respectivamente. Dichos registros pueden ser utilizados para direccionamiento indirecto de la memoria de dato, o almacenamiento temporal de datos, como se observa en las figuras 1.3 y 1.4. Además, permiten posicionarse en una dirección de memoria de dato de un operando de una instrucción.

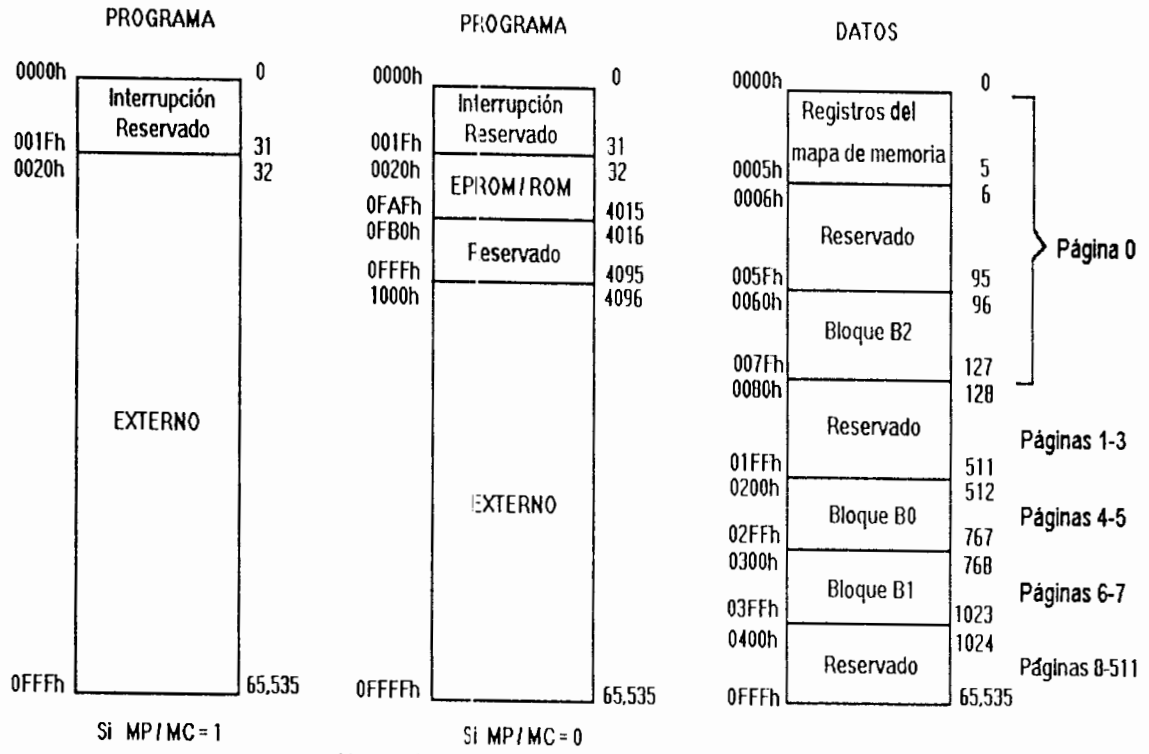


Figura 1.2. Mapa de la memoria

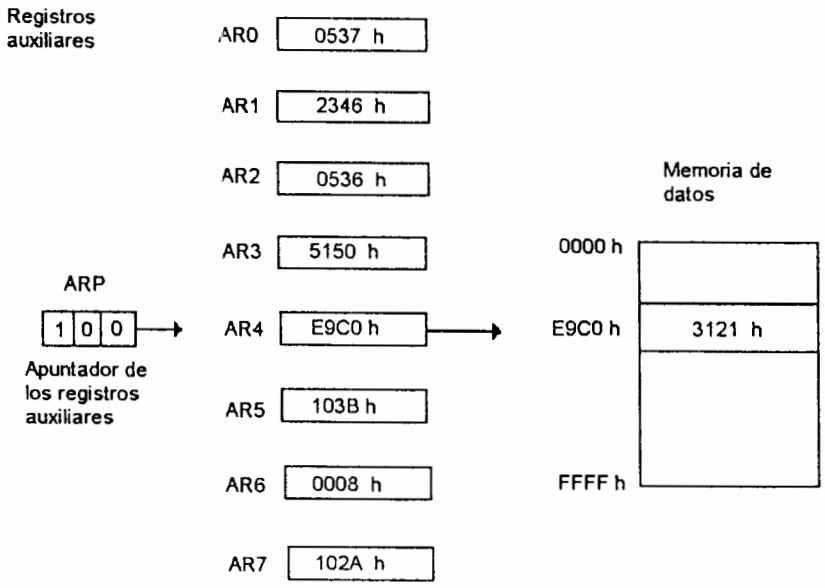


Figura 1.3. Registros en la memoria

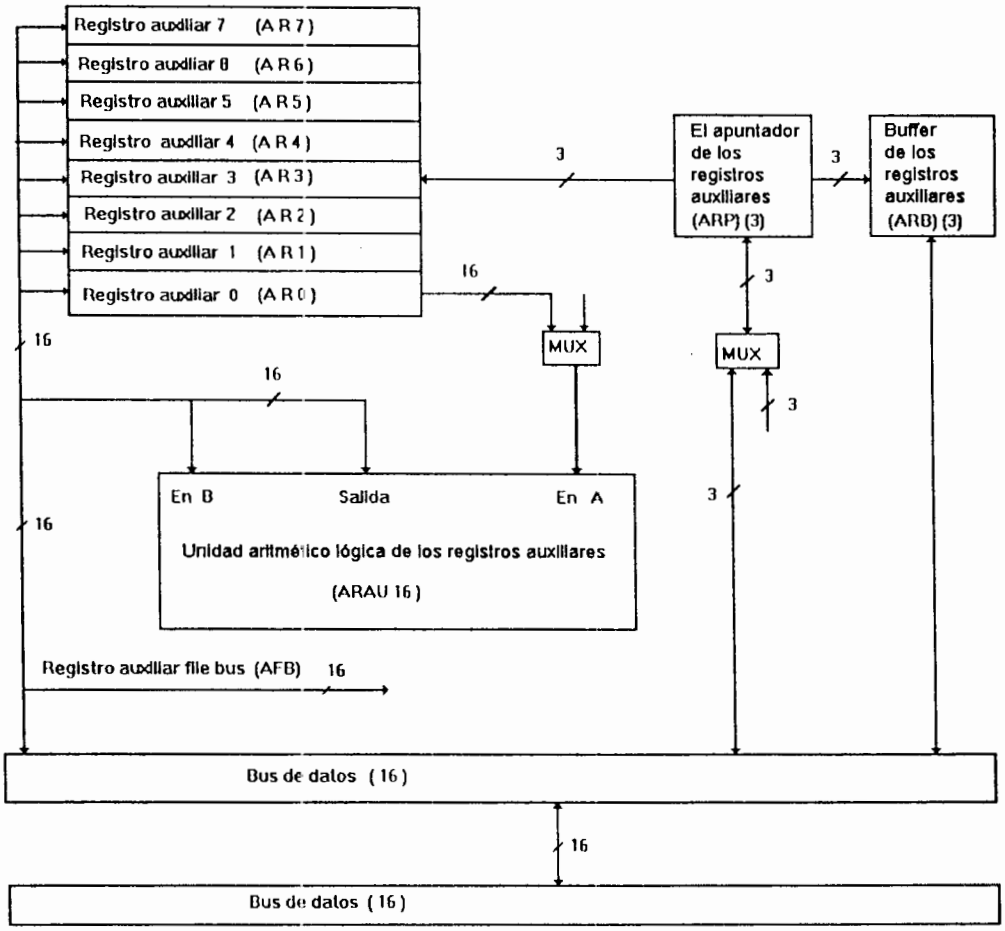


Figura 1.4. Registros auxiliares AR0-AR7

Los registros auxiliares están conectados a una unidad aritmética de registros auxiliares (ARAU). Esta unidad puede autoindexar el registro actual, mientras la localización del dato está siendo direccionada. Como resultado, la CALU no accesa a tablas de información para manipuleo de direcciones, de este modo realiza libremente otras operaciones.

ARAU efectúa las siguientes operaciones:

AR(ARP) + AR0 $\Longrightarrow$ AR(ARP).
AR(ARP) - AR0 $\Longrightarrow$ AR(ARP).
AR(ARP) + 1 $\Longrightarrow$ AR(ARP).
AR(ARP) - 1 $\Longrightarrow$ AR(ARP).
AR(ARP) $\Longrightarrow$ AR(ARP) AR(ARP) sin cambios.
AR(ARP) + IR(7-0) $\Longrightarrow$ AR(ARP) suma 8 bits inmediatos a AR(ARP).
AR(ARP) - IR(7-0) $\Longrightarrow$ AR(ARP) resta 8 bits inmediatos a AR(ARP).
AR(ARP) + rcARO $\Longrightarrow$ AR(ARP) suma AR0 con la propagación de carry reverso.
AR(ARP) - rcARO $\Longrightarrow$ AR(ARP) resta AR0 con la propagación de carry reverso.

Aunque la unidad ARAU es útil para la manipulación de direccionamiento en paralelo con otras operaciones, ésta puede servir como una unidad aritmética adicional de propósito general, ya que los registros auxiliares pueden comunicarse directamente con la memoria de datos. La unidad ARAU utiliza aritmética no signada de 16 bits, mientras que con la unidad CALU, aritmética modo dos complemento de 32 bits.

Modos de direccionamiento de memoria

El TMS puede direccionar un total de 64k-palabras de memoria de programa y 64k-palabras de memoria de datos, como puede observarse en la figura 1.5. Los 16 bits del bus de direcciones (DAB) direcciona la memoria de datos de dos formas:

1. Direccionamiento directo de bus usando modo de direccionamiento directo.
2. Por los registros auxiliares usando el modo de direccionamiento indirecto.

Los operandos también son direccionados por el contenido del contador de programa en el modo de direccionamiento inmediato.

a) Modo de direccionamiento directo

Utiliza 9 bits del apuntador de página DP, el cual puede apuntar 512 páginas de 128 palabras cada una. El dato de memoria direccionado es especificado por los 7 bits menos significativos de la instrucción para apuntar la palabra deseada dentro de la página.

b) Modo de direccionamiento indirecto

Los 16 bits de la dirección son seleccionados por el registro auxiliar en uso, direccionando el dato de memoria a través del bus de registro auxiliar file (AFB).

Cuando un operando inmediato es empleado, la palabra de la instrucción contiene el operando; en el caso de operandos inmediatos de 16 bits, éstos constituyen la siguiente palabra del código de instrucción.

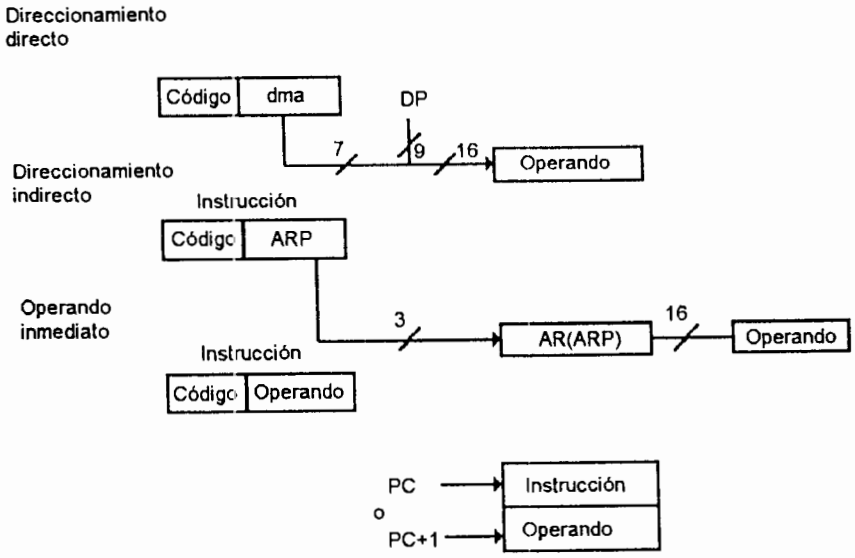


Figura 1.5. Modos de direccionamiento

Movimiento de memoria a memoria

Las instrucciones que permiten el movimiento de datos y programa determinan que se utilice eficazmente la configuración de la RAM interna. La instrucción BLKD mueve un bloque dentro de la memoria dato y BLKP un bloque de programa a memoria de datos. Para el empleo eficiente de estas instrucciones, se utilizan las de repetición RPT y RPTK.

La instrucción DMOV copia una palabra de la dirección actual en la memoria de dato en RAM interna a la próxima localización alta, mientras el dato de la dirección de localización está siendo operado en el mismo ciclo por CALU. Una operación de la ARAU también puede ejecutarse en el mismo ciclo cuando se usa el modo de direccionamiento indirecto.

La función de DMOV es útil en los algoritmos donde se utilizan operadores de retardo Z^{-1}, tales como convolución y filtrado digital donde el dato es pasado a través de una ventana de tiempo. La función de movimiento de dato puede ser utilizada dentro de los bloques B0, B1 y B2. Las instrucciones TBLR y TBLW (tabla de lectura y escritura) transfieren palabras entre el espacio de programa y de dato.

Unidad aritmética lógica central (CALU)

Una unidad aritmética lógica central contiene un registro de corrimiento de 16 bits, un multiplicador paralelo de 16 × 16 bits, una unidad aritmética lógica (ALU) de 32 bits, un acumulador de 32 bits (ACC), el cual está dividido en parte alta ACCH y parte baja ACCL, corrimientos de salida para el acumulador y el multiplicador. Las instrucciones SFL y SFR indican los corrimientos a la izquierda y derecha respectivamente.

Pasos en la ejecución de una instrucción típica en la CALU:

1) El dato se busca RAM a través del bus de datos.

2) El dato se pasa a través del registro de corrimiento y en la CALU se ejecuta la operación.

3) El resultado es movido dentro del acumulador.

Una entrada a la ALU es siempre dada por el acumulador y la otra puede ser transferida del registro producto (PR) del multiplicador o del registro de corrimiento que es cargado de la memoria dato. Después de ejecutar la operación aritmética o lógica, la ALU almacena el resultado en el acumulador.

El TMS soporta operaciones de punto flotante requeridas para grandes rangos dinámicos. La instrucción NORM (normalizar) es utilizada para normalizar el punto fijo del número en el acumulador por ejecución de corrimientos a la izquierda. La instrucción LACT desnormaliza un número de punto flotante por corrimiento aritmético a la izquierda de la mantisa a través de la entrada del registro de corrimiento.

El modo de saturación de sobreflujo en el acumulador puede ser programado a través de las instrucciones SOVM y ROVM. Cuando el acumulador está en modo de saturación de sobreflujo y un sobreflujo ocurre, la bandera de sobreflujo es fijada y el acumulador es cargado con cada uno de los números más positivos o negativos, dependiendo de la dirección de sobreflujo ($> 7FFFFFFF$ para positivo, y > 80000000 para negativo).

Se pueden implantar una variedad de instrucciones de salto a una dirección específica dependiendo del estado de la ALU y del acumulador. El acumulador tiene asociado un acarreo (carry), el cual puede ser puesto en uno (set) o cero (reset), según las operaciones que se realizan. También existen instrucciones para corrimientos y rotamientos del acumulador a la izquierda o a la derecha.

Registro de corrimiento

Puede producir corrimientos de 0 a 16 bits a un dato de entrada según sea programado en la instrucción. Los bits menos significativos se llenan de ceros y los más significativos pueden ser llenados con ceros o signo extendido, dependiendo del estado programado en el modo de signo extendido SXM.

Multiplicador, registros T y P

El TMS utiliza un hardware capaz de efectuar multiplicaciones de 16×16 bits en un ciclo de máquina. Todas las instrucciones de multiplicación, excepto MPYU (multiplicación no signada), realizan operaciones de multiplicación signada en el multiplicador. Es decir, que dos números pueden ser multiplicados como números en modo dos complemento y el resultado es un número de 32 bits complementado a dos. Los registros asociados a la multiplicación son:

- TR: registro temporal de 16 bits que mantiene uno de los operandos por multiplicar.
- PR: registro producto de 32 bits que mantiene el producto.

La salida del registro producto puede ser corrida a la izquierda 1 o 4 bits, esto es útil para la implantación de aritmética fraccionaria o justificación de productos fraccionales. La salida PR también puede ser corrida a la derecha en 6 bits, lo cual posibilita la ejecución de 128 multiplicaciones/acumulación sucesivas sin sobreflujo. El registro T normalmente es cargado con la instrucción LT, la cual le provee uno de los operandos (del bus de datos), y la instrucción MPY (multiplicación) proporciona el segundo operando también del bus de

datos. Una multiplicación también puede ser ejecutada con un operando inmediato, con la instrucción MPYK. En los dos casos, el producto es obtenido cada dos ciclos.

Las instrucciones de multiplicación/acumulación (MAC y MACD) emplean totalmente el ancho de cálculo de la multiplicación, lo cual permite que ambos operandos sean procesados simultáneamente.

Las instrucciones SQRA (cuadrado/suma) y SQRS (cuadrado/resta) pasan el mismo valor a ambas entradas de la multiplicación para elevar al cuadrado un valor de la memoria dato.

La instrucción MPYU permite multiplicaciones no signadas que facilitan la precisión de operaciones aritméticas.

Hay disponibles cuatro modos de corrimiento a través del registro producto (PR), los cuales son útiles cuando se ejecutan operaciones de multiplicación/acumulación, aritmética fraccional o justificación fraccional de productos. El registro PM ocupa 2 bits en el registro de estado ST1, que especifica el modo del producto (PM), como se indica en la tabla 1.1.

PM	*PM Modo de corrimiento*
00	No existe corrimiento.
01	Corrimiento de 1 bit a la izquierda.
10	Corrimiento de 4 bits a la izquierda.
11	Corrimiento de 6 bits a la derecha.

Tabla 1.1. Registro PM modo de corrimiento

1.6 Sistema de control

Está dado por el contador de programa, el stack, la señal de reset externo, las interrupciones, los registros de estado, el timer interno y el contador de repetición.

El contador de programa y el stack

El contador de programa (PC) consta de 16 bits, lo que permite direccionar hasta 64k-direcciones de memoria programa externas o internas en la búsqueda de las instrucciones. El stack consta de 8 localidades de 16 bits para almacenar el PC durante las interrupciones o las transferencias a subrutinas.

El PC direcciona la memoria programa, vía el bus de dirección de programa (PAB). A través del PAB, se busca una instrucción de la memoria programa y se carga en el registro de instrucción (IR). Cuando el IR ha sido cargado, el PC está listo para iniciar un nuevo ciclo de búsqueda. El PC puede direccionar la RAM interna del bloque B0 cuando éste ha sido configurado como memoria de programa o la ROM interna del TMS. El PC también direcciona la memoria de programa cuando es memoria externa a través del bus externo de direcciones A15-A0 y el bus externo de datos D15-D0. Al inicio del nuevo ciclo de búsqueda,

el PC es cargado con PC+1 o con una dirección de salto. En el caso de no tomar el salto, el PC es incrementado una vez más desde la localización de la instrucción de salto.

También el TMS tiene una característica que le permite ejecutar en el próximo ciclo simple N+1 veces una instrucción, donde N es definido por la carga de 8 bits en el contador de repeticiones RPTC con la instrucción RPTK.

El stack es accesado a través de las instrucciones PUSH y POP. Las instrucciones adicionales de PSHD y POPD permiten utilizar el stack para el almacenamiento temporal de la memoria de datos.

Operación de pipeline o encauzamiento

Una instrucción de pipeline consiste en una secuencia de operaciones externas de bus que ocurren durante la ejecución interna de la instrucción. El pipeline de prebúsqueda-decodificación-ejecución es esencialmente transparente al usuario, excepto en algunos casos donde el pipeline debe ser interrumpido con instrucciones, tales como saltos dentro del programa (B, BNZ, etc.). En la operación de pipeline, las operaciones de prebúsqueda, decodificación y ejecución son independientes. Durante algún ciclo dado, dos o tres diferentes instrucciones pueden ser activadas cada una a un estado distinto de realización.

Los diferentes niveles en pipeline no necesariamente afectan la velocidad de ejecución, pero sí la secuencia búsqueda/decodificación. Más instrucciones son ejecutadas en igual número de ciclos.

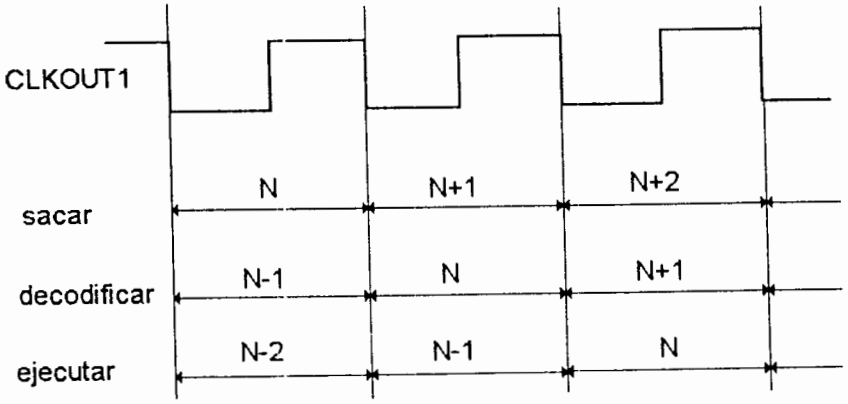


Figura 1.6. Operación de pipeline a tres niveles

En los tres niveles de pipeline, un contador de prebúsqueda contiene la dirección de la próxima instrucción que será prebuscada. Una vez que una instrucción ha sido prebuscada, entonces se carga en un registro de instrucción (IR), a menos que IR contenga la instrucción en ejecución, en cuyo caso la instrucción prebuscada se carga en un registro de instrucción a la cola (QIR). El contador de prebúsqueda es incrementado y después que la instrucción actual completa la ejecución, la instrucción en QIR se carga en IR para ser ejecutada.

El PC contiene la dirección de la próxima instrucción que va a ejecutarse sin ser usada directamente en las operaciones de búsqueda, pero sirve como un apuntador de referencia para la posición actual dentro del programa. El PC es incrementado cada vez que una instrucción se ejecuta. Cuando ocurre una interrupción o una llamada de subrutina, el

contenido del PC se guarda en el stack para preservarlo al regreso de una interrupción o subrutina.

La prebúsqueda, decodificación y ejecución de la operación de pipeline son independientes, lo cual permite la ejecución de operaciones traslapadas. Durante algún ciclo, tres instrucciones distintas pueden ser activadas cada una a diferente estado de realización.

La operación básica de pipeline es de 3.25 ciclos de duración, donde la secuencia en un dispositivo sobre algún ciclo dado será la siguiente: buscar una tercera instrucción, decodifica una segunda y ejecutar una primera. Si la instrucción se busca en la RAM interna, el pipeline es acortado a 2.5 ciclos, dado que TMS puede buscar la instrucción en la mitad del ciclo, contrario a lo que ocurre cuando se busca en la ROM externa, que requiere de un ciclo completo para la búsqueda.

Cuando existen estados de espera, sólo retardan directamente la operación de pipeline. Cada estado de espera, insertado durante la operación de búsqueda, agrega más ciclos de máquina en la ejecución de pipeline de la instrucción.

Acceso externo de memoria de programa o datos

La visibilidad de pipeline requiere el monitoreo de señales /MSC, /STRB, /PS y /DS. La señal /MSC indicada en la caída de CLKUOT2, ya sea o no el ciclo de una nueva instrucción de búsqueda, es decir, que /MSC activa señala la terminación de una instrucción y la adquisición de otra. /PS indica que el bus de datos está siendo utilizado para la búsqueda de una instrucción (aunque no necesariamente la activación de esta señal significa la búsqueda de una instrucción). El monitoreo de la búsqueda de memoria de dato debe hacerse seleccionando conjuntamente /DS y /STRB.

Reset (/RS)

Es una interrupción externa no mascarable, con la más alta prioridad sobre las demás interrupciones. Puede ser usada en cualquier instante y es aplicada típicamente en el encendido cuando los estados de máquina son aleatorios.

La señal /RS causa que el TMS termine la ejecución y fuerce al contador de programa a ir a la dirección cero, lo cual afecta varios registros y los bits de estado. Para la correcta operación del sistema, la señal de reset debe ser insertada baja, por lo menos tres ciclos de reloj. El bus de datos es puesto en alta impedancia y las señales de control se van a alto mientras dura el reset.

Registros de estado

Los registros de estado ST0 y ST1 contienen el estado de varias condiciones y modos de operación del TMS, figura 1.7. Los registros de estado pueden ser almacenados dentro de la memoria de dato y cargados de memoria dato, lo cual permite que el estado de la máquina sea salvado y restaurado para interrupciones y subrutinas. Todos los bits de estado pueden ser escritos y leídos con las instrucciones LST/LST1 y SST/SST1 (excepto INTM, que se carga con LST).

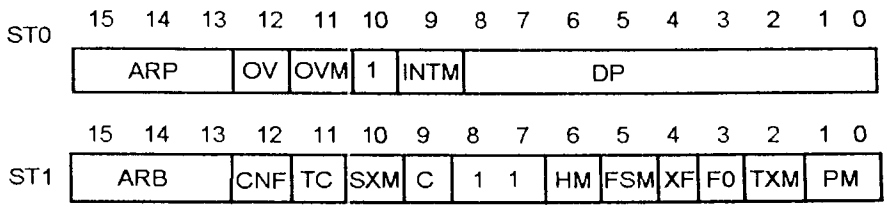


Figura 1.7. Registros de estado

Donde:

- ARB: buffer apuntador de registros auxiliares.
- ARP: apuntador de registros auxiliares.
- C: bit de carry.
- CNF: bit de control de configuración de RAM interna.
- DP: apuntador de página de memoria dato.
- FO: bit de formato para cada puerto serie.
- FSM: bit de modo de estructura de sincronización.
- HM: bit de modo hold.
- INTM: bit de modo de interrupción, 0 = habilitado, 1 = deshabilitado.
- OV: bit de bandera de sobreflujo.
- OVM: bit de modo de sobreflujo.
- PM: modo de corrimiento de producto.
- SXM: bit de modo de signo extendido.
- TC: bandera de control de prueba de bit.
- TMX: bit de modo de transmisión.
- XF: bit de estado del pin XF.

Operación del timer

El TMS posee un registro timer (TIM) y un registro período (PRD) de 16 bits. El timer cuenta descendentemente en la caída de CLKOUT1. Los registros TIM y PRD son fijados a su máximo valor > $FFFF$ en el reset. Una interrupción del timer (TINT) es generada cada vez que el TIM llega a cero, el TIM es recargado con el valor de PRD en el próximo ciclo. Esta característica es útil para operaciones de control y sincronización de muestreos o escritura de periféricos.

Si el timer no es utilizado, TINT debe ser mascarable o todas las interrupciones deben ser deshabilitadas por la instrucción DINT.

Contador de repetición (RPTC)

Es de 8 bits y cuando es cargado con un número N causa que la siguiente instrucción sea ejecutada N+1 veces. El RPTC puede ser cargado con un número de 0 a 255 mediante las instrucciones RPT o RPTK. Se utiliza en las instrucciones de multiplicación/acumulación, movimiento de bloques, transferencias de I/O y tablas de lectura/escritura.

Memoria externa e interfaz I/O

La interfaz local de memoria consta de los siguientes elementos:

- Bus de datos paralelos de 16 bit D15-D0.
- Bus de direcciones A15-A0.
- Señales de selección de espacio programa, dato e I/O, /PS, /DS y /IS, respectivamente.
- Varias señales del sistema de control.

La señal de lectura/escritura R/W provee la dirección de transferencia y la señal /STRB el tiempo de transferencia de control.

El microcontrolador tiene 16 puertos de entrada y 16 de salida, con 16 bits para datos de entrada o salida. Para entrada o salida se utilizan de manera respectiva las instrucciones de IN y OUT, que típicamente ocupan dos ciclos; sin embargo, con el contador de repeticiones la operación se convierte en un ciclo simple.

Combinación de memoria

La secuencia exacta de las operaciones ejecutadas de una instrucción depende de las áreas de memoria donde la instrucción y los operandos son localizados. Las seis posibles combinaciones de programa y dato, dada la información, pueden ser localizadas en memoria RAM interna, memoria ROM externa o interna:

- PI/DI: programa interno RAM/dato interno.
- PI/DE: programa interno RAM/dato externo.
- PE/DI: programa externo/dato interno.
- PE/DE: programa externo/dato externo.
- PR/DI: programa interno ROM/dato interno.
- PR/DE: programa interno ROM/dato externo.

Relaciones de tiempo del reloj interno

El cristal o la fuente de frecuencia externa es dividida para producir cuatro fases de reloj, definidas como CLKOUT1 y CLKOUT2.

Interrupciones

El TMS tiene tres interrupciones externas mascarables (/INT2, /INT1, /INT0), disponibles para dispositivos externos que interrumpan al microprocesador. Dos interrupciones internas generadas por el puerto serie (RINT y XINT), una por el timer (TINT) y otra más por software a través de la instrucción TRAP, aunque esta última no tiene prioridad.

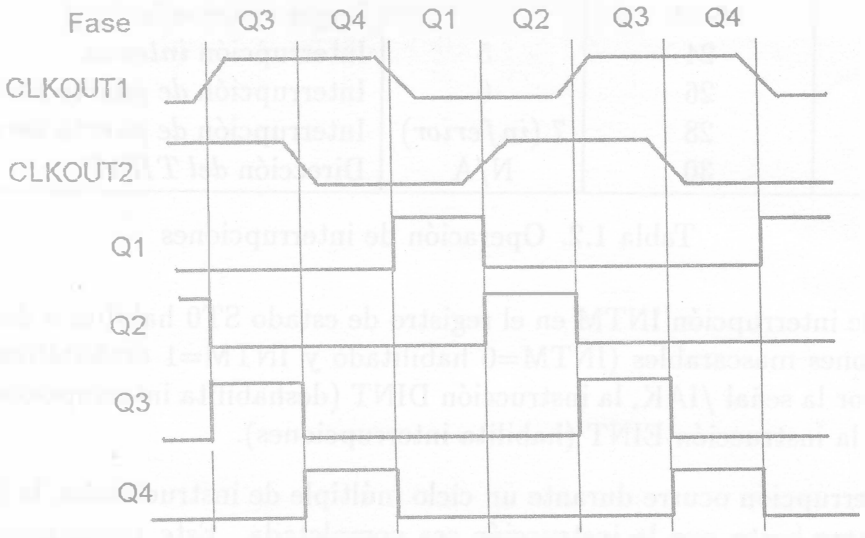


Figura 1.8. Reloj interno

Operación de las interrupciones

Cada dirección de interrupción ha sido espaciada cada dos localidades, de tal modo que se puedan acomodar instrucciones para realizar saltos cuando así se desea, tabla 1.2. Cuando una interrupción ocurre, ésta es almacenada en un registro de bandera de interrupción de 6 bits (IFR). Este registro es fijado por el uso de interrupciones externas /INT(2-0) y las interrupciones RINT, XINT y TINT. Cada interrupción es almacenada en el registro IFR hasta que es identificada y automáticamente borrada por el reconocimiento de interrupciones (/IACK) o el reset (/RS). /RS no es almacenado en el registro IFR. Ninguna instrucción puede leerse o escribirse en IFR.

El TMS tiene un mapa de memoria para registro de interrupciones mascarables (IMR) para mascarar las interrupciones internas y externas (figura 1.9).

El IMR, que habilita la correspondiente interrupción colocando un cero, es accesible para operaciones de lectura y escritura, excepto cuando se usa BLKD. IMR no es afectado por reset.

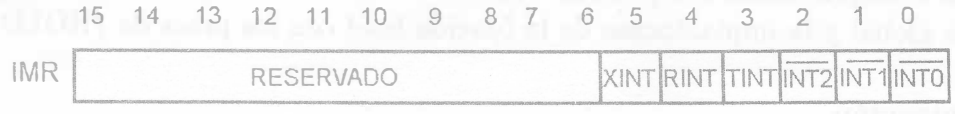


Figura 1.9. Mapa de memoria para registro de interrupciones

Nombre de interrupción	*Localización en la memoria*	*Prioridad*	*Función*
$\overline{RS}$	0	1 (*suprema*)	Reset *externo*
$\overline{INT0}$	2	2	Interrupción *externa* #0
$\overline{INT1}$	4	3	Interrupción *externa* #1
$\overline{INT2}$	6	4	Interrupción *externa* #2
	8-23		Lugar *reservado*
TINT	24	5	Interrupción *interna*
RINT	26	6	Interrupción *de puerto serial*
XINT	28	7 (*inferior*)	Interrupción *de puerto serial*
TRAP	30	N/A	Dirección *del TRAP*

Tabla 1.2. Operación de interrupciones

El modo de interrupción INTM en el registro de estado ST0 habilita o deshabilita todas las interrupciones mascarables (INTM=0 habilitado y INTM=1 deshabilitado). INTM es fijado a uno por la señal /IAK, la instrucción DINT (deshabilita interrupciones) y por reset, y en cero por la instrucción EINT (habilita interrupciones).

Si una interrupción ocurre durante un ciclo múltiple de instrucciones, la interrupción no puede procesarse hasta que la instrucción sea completada. Este mecanismo de protección también es aplicado a instrucciones de ciclo múltiple debido a la señal de READY. Tampoco se permite que se procesen interrupciones cuando una instrucción está siendo repetida por medio de RPTK, la interrupción es almacenada en el registro IFR hasta que el ciclo de repetición se termine, después la interrupción es procesada. Las interrupciones no pueden ser procesadas entre la instrucción EINT y la próxima instrucción en la secuencia del programa. También una interrupción no es reconocida cuando el /HOLD está activado.

Multiprocesamiento y acceso directo de memoria (DMA)

El TMS puede ser configurado para satisfacer amplios rangos de requerimiento de sistemas. Algunas configuraciones son:

- Sistema estándar (proceso simple).
- Multiprocesamiento con dispositivos en paralelo.
- Multiprocesos huésped/esclavo con espacio de memoria dato compartida.
- Proceso a periféricos interfaceados mediante señales de control de procesos para otros dispositivos.

Estas configuraciones son posibles con la señal externa de entrada /SYNC, la interfaz de memoria global y la implantación de la función hold con los pines de /HOLD y /HOLDA.

Sincronización

Esta señal externa de entrada puede ser utilizada con gran facilidad entre procesos, lo cual provoca que cada TMS en el sistema se sincronice con su reloj interno y el proceso corra

amarrado a los pasos de la operación. La transición negativa de /SYNC fija a cada proceso a un cuarto de fase interna (Q1). Esta transición debe ocurrir sincronizada con la bajada del reloj CLKIN. En el TMS hay dos ciclos de retardo CLKIN que permiten que la señal /SYNC vaya abajo, por lo tanto, ocurre la sincronización Q1.

Normalmente /SYNC es aplicado mientras /RS está activado para evitar situaciones no predecibles del procesador.

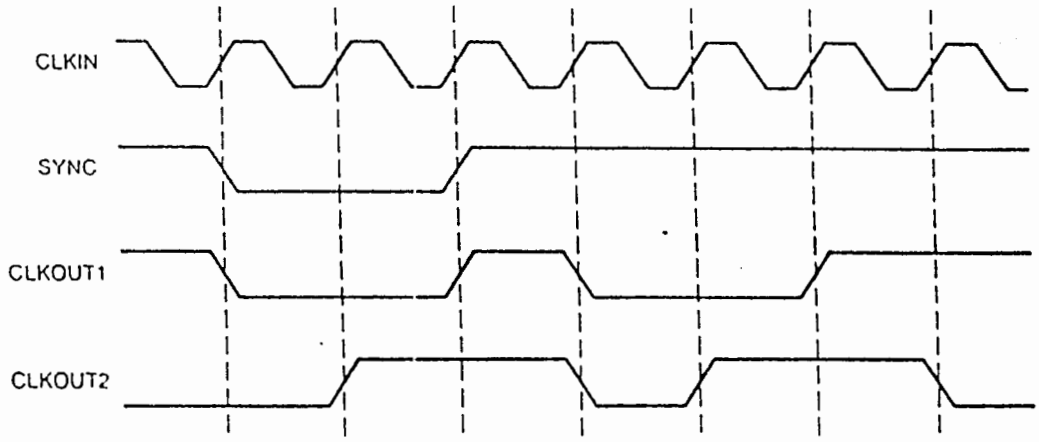


Figura 1.10. Sincronización

Memoria global

Para aplicaciones de multiprocesamiento, el TMS tiene la capacidad de localizar un espacio de memoria global de datos y comunicarse con éste a través de /BR (requerimiento de bus) y la señal de control READY.

La memoria global es compartida por más de un procesador, por lo tanto, su acceso debe ser arbitrado. Cuando se utiliza memoria global, el espacio de direccionamiento del procesador es dividido en secciones: local y global. La sección local es usada por el procesador para ejecutar funciones individuales y la sección global para comunicarse con otros procesadores.

El registro de mapeo de memoria global (GREG) especifica la parte de memoria de datos del TMS como memoria global externa. GREG es mapeado como memoria de datos en la localización 5, es un registro de 8 bits conectados con los 8 bits menos significativos del bus de datos.

El contenido de GREG determina el tamaño del espacio de memoria global. El valor de GREG y el correspondiente espacio se indican en la tabla 1.3.

Cuando un dato es direccionado directa o indirectamente de memoria global (definido por GREG), entonces la señal /BR es activada bajo mediante la señal /DS para indicar al procesador que se desea accesar de memoria global.

La lógica externa arbitra el control de memoria global con la señal READY.

Valor de GREG	*Memoria local*		*Memoria global*		
	Rango	*Palabras*	*Rango*		*Palabras*
000000xx	$0h - 0FFFFh$	65,536	$----$		0
10000000	$0h - 07FFFh$	32,768	$08000h$	$-0FFFFh$	32,768
11000000	$0h - 0BFFFh$	49,152	$0C000h$	$-0FFFFh$	16,384
11100000	$0h - 0DFFFh$	57,344	$0E000h$	$-0FFFFh$	8,192
11110000	$0h - 0EFFFh$	61,440	$0F000h$	$-0FFFFh$	4,096
11111000	$0h - 0F7FFh$	63,488	$0F800h$	$-0FFFFh$	2,048
11111100	$0h - 0FBFFh$	64,512	$0FC00h$	$-0FFFFh$	1,024
11111110	$0h - 07DFFh$	65,024	$0FE00h$	$-0FFFFh$	512
11111111	$0h - 07EFFh$	65,280	$0FF00h$	$-0FFFFh$	256

Tabla 1.3. Valor de GREG

Capítulo 2

Instrucciones del DSP TMS320C25

El TMS320C25 cuenta con un conjunto de 133 instrucciones, 97 de ellas son ejecutadas en un ciclo simple de instrucción, otras 36 requieren ciclos adicionales, 21 involucran saltos, llamadas de rutinas y retornos (las cuales rompen la ejecución de la operación pipeline), otras siete instrucciones son de dos palabras (instrucciones largas inmediatas). Las ocho restantes (IN, OUT, BLKD, BLKP, TBLR, TBLW, MAC y MACD) soportan transferencia de I/O y de datos entre espacio de memoria, o son utilizadas para operaciones en paralelo en el procesador. Adicionalmente estas ocho instrucciones se convierten en un ciclo simple cuando son utilizadas en conjunto con el contador de repeticiones.

Estas instrucciones utilizan los modos de direccionamiento del TMS. Además explotan el paralelismo del procesador, lo cual permite cálculos complejos que son desarrollados con pocas instrucciones.

Las instrucciones las podemos clasificar de la siguiente manera:

- aritmético-lógicas que trabajan con el acumulador,
- para multiplicación,
- para saltos del programa,
- para comunicación con los circuitos entrada/salida,
- para la modificación de direcciones de registros y
- dirigidas.

De la tabla 2.1 hasta la 2.6 se muestran las instrucciones del DSP, su sintaxis y características.

SINTAXIS EN ENSAMBLADOR	TÍTULO	DESCRIPCIÓN DE OPERACIONES
ADD $< dma >$ [, $< shift >$] **ADD** $< ind >$ [, $< shift >$ [, $< nextARP >$]]	Suma al acumulador con desplazamiento	$(ACC) + [(dma) \times 2^{shift}] \rightarrow ACC$
ADDC $< dma >$ **ADDC** $< ind >$ [, $< nextARP >$]	Suma al acumulador con carry	$(ACC) + (dma) + (C) \rightarrow ACC$
ADDH $< dma >$ **ADDH** $< ind >$ [, $< nextARP >$]	Suma a la parte alta del acumulador	$(ACC) + [(dma) \times 2^{16}] \rightarrow ACC$
ADDK $< constant >$	Suma valor corto inmediato al acumulador	$(ACC) + 8-$bit constant$\rightarrow ACC$
ADDS $< dma >$ **ADDS** $< ind >$ [, $< nextARP >$]	Suma a parte baja del acum. sin extensión del signo	$(ACC) + (dma) \rightarrow ACC$
ADDT $< dma >$ **ADDT** $< ind >$ [, $< nextARP >$]	Suma al acumulador con desplazamiento dado por T	$(ACC) + [(dma) \times 2^{(Treg)}] \rightarrow ACC$
ADLK $< constant >$ [, $< shift >$]	Suma constante con desplazamiento	$(ACC) + [16-$bit constant $\times 2^{shift}] \rightarrow ACC$
AND $< dma >$ **AND** $< ind >$ [, $< nextARP >$]	AND con el acumulador	$(ACC(15-0)).AND.(dma) \rightarrow ACC(15-0)$ $0 \rightarrow ACC(31-16)$
ANDK $< constant >$ [, $< shift >$]	AND constante con acumulador	$(ACC(30-0)).AND.[16-$bit constant $\times 2^{shift}] \rightarrow ACC(30-0)$ $0 \rightarrow ACC(31-0$ unoccupied)
CMPL	Complemento al acumulador	$(\overline{ACC}) \rightarrow ACC$
LAC $< dma >$ [, $< shift >$] **LAC** $< ind >$ [, $< shift >$ [, $< nextARP >$]]	Carga el acumulador con desplazamiento	$(dma) \times 2^{shift} \rightarrow ACC$
LACK $< constant >$	Carga valor inmediato al acumulador	$8-$bit constant $\rightarrow ACC$
LACT $< dma >$ **LACT** $< ind >$ [, $< nextARP >$]	Carga al acumulador con desplazamiento dado por T	$(dma) \times 2^{(Treg)} \rightarrow ACC$
LALK $< constant >$ [, $< shift >$]	Carga valor inmediato largo al acumulador con desplazamiento	$16-$bit constant $\times 2^{shift} \rightarrow ACC$

Tabla 2.1. Instrucciones del DSP TMS320C25 que trabajan con el acumulador

SINTAXIS EN ENSAMBLADOR	TÍTULO	DESCRIPCIÓN DE OPERACIONES
NORM < *ind* >	Normaliza el contenido del acumulador	por ejemplo para transferencia del punto fijo a punto flotante
OR < *dma* > **OR** < *ind* > [, < *nextARP* >]	OR entre contenido de dma y acumulador	$(ACC(15-0)).OR.(dma) \rightarrow ACC(15-0)$
ORK < *constant* > [, < *shift* >]	OR entre valor inmediato y acumulador con desplazamiento	$(ACC(30-0)).OR.[16\text{–bit constant} \times 2^{shift}] \rightarrow ACC(30-0)$
ROL	Rotamiento del acumulador a la izquierda	$(ACC(30-0)) \rightarrow ACC(31-1)$ $(C) \rightarrow ACC(0)$, $(ACC(31)) \rightarrow C$
ROR	Rotamiento del acumulador a la derecha	$(ACC(31-1)) \rightarrow ACC(30-0)$ $(C) \rightarrow ACC(31)$, $(ACC(0)) \rightarrow C$
SACH < *dma* > [, < *shift* >] **SACH** < *ind* > [, < *shift* > [, < *nextARP* >]]	Carga parte alta del acumulador con desplazamiento	$[(ACCH) \times 2^{shift}] \rightarrow dma$
SACL < *dma* > [, < *shift* >] **SACL** < *ind* > [, < *shift* > [, < *nextARP* >]]	Carga parte baja del acumulador con desplazamiento	$[(ACCL) \times 2^{shift}] \rightarrow dma$
SBLK < *constant* > [, < *shift* >]	Sustrae constante con desplazamiento	$(ACC) - [16\text{–bit constant} \times 2^{shift}] \rightarrow ACC$
SFL	Corrimiento del acumulador a la izquierda	$(ACC(30-0)) \rightarrow ACC(31-1)$ $0 \rightarrow ACC(0)$
SFR	Corrimiento del acumulador a la derecha	$(ACC(31-1)) \rightarrow ACC(30-0)$ if $SXM = 0$: then $0 \rightarrow ACC(31)$ if $SXM = 1$: then $ACC(31) \rightarrow ACC(31)$
SUB < *dma* > [, < *shift* >] **SUB** < *ind* > [, < *shift* > [, < *nextARP* >]]	Valor en dma se sustrae del acumulador con desplazamiento	$(ACC) - [(dma) \times 2^{shift}] \rightarrow ACC$
SUBB < *dma* > **SUBB** < *ind* > [, < *nextARP* >]	Valor en dma se sustrae del acumulador con préstamo	$(ACC) - (dma) - (\overline{C}) \rightarrow ACC$
SUBC < *dma* > **SUBC** < *ind* > [, < *nextARP* >]	Valor en dma se sustrae del acumulador condicionalmente	$ACC - [, (dma).2^{15}]alALU$
SUBH < *dma* > **SUBH** < *ind* > [, < *nextARP* >]	Valor en dma se sustrae de la parte alta del acumulador	$(ACC) - [(dma) \times 2^{16}] \rightarrow ACC$

Tabla 2.1 Instrucciones del DSP que trabajan con el acumulador - *Continuación*

SINTAXIS EN ENSAMBLADOR	TÍTULO	DESCRIPCIÓN DE OPERACIONES
SUBK < *constant* >	Valor inmediato se sustrae del acumulador	$(ACC) - 8-$bit constant $\rightarrow ACC$
SUBS < *dma* > **SUBS** < *ind* > [, < *nextARP* >]	Contenido del dma se sustrae del acumulador sin tomar en cuenta el signo	$(ACC) - (dma) \rightarrow ACC$
SUBT < *dma* > **SUBT** < *ind* > [, < *nextARP* >]	Valor en dma se sustrae del ACC con corrimiento especificado por T	$(ACC) - [(dma) \times 2^{(Treg)}] \rightarrow ACC$
XOR < *dma* > **XOR** < *ind* > [, < *nextARP* >]	XOR entre contenido de dma y el acumulador	$(ACC(15-0)).XOR.(dma) \rightarrow ACC(15-0)$
XORK < *constant* > [, < *shift* >]	XOR entre valor inmediato y el acumulador con desplazamiento	$(ACC(30-0)).XOR.[16-$bit constant $\times 2^{shift}] \rightarrow ACC(30-0)$
ZAC	Carga ceros al acumulador	$0 \rightarrow ACC$
ZALH < *dma* > **ZALH** < *ind* > [, < *nextARP* >]	Carga ceros al ACCL y contenido de dma al ACCH	$(dma) \times 2^{16} \rightarrow ACC$ $0 \rightarrow ACCL$
ZALR < *dma* > **ZALR** < *ind* > [, < *nextARP* >]	Carga ceros al ACCL y contenido de dma al ACCH con redondeo	$(dma) \times 2^{16} + > 8000 \rightarrow ACC$ $0 \rightarrow ACCL$
ZALS < *dma* > **ZALS** < *ind* > [, < *nextARP* >]	Carga ceros al ACCH y contenido de dma al ACCL sin tomar en cuente el signo	$(dma) \rightarrow ACCL$ $0 \rightarrow ACCH$

Tabla 2.1. Instrucciones del DSP que trabajan con el acumulador - *Continuación*

SINTAXIS EN ENSAMBLADOR	TÍTULO	DESCRIPCIÓN DE OPERACIONES
ADRK < *constant* >	Suma valor inmediato corto al registro auxiliar	$(ARn) + 8-$bit constant $\rightarrow ARn$
CMPR < *constant* >	Se compara el registro auxiliar con el AR0	if $AR(ARP).oper.AR0$ then $1 \rightarrow TC$ else $0 \rightarrow TC$
LAR < *AR* >, < *dma* > **LAR** < *AR* >, < *ind* > [, < *nextARP* >]	Valor en dma se carga al registro auxiliar	$(dma) \rightarrow ARn$
LARK < *AR* >, < *constant* >	Valor inmediato corto se carga al registro auxiliar	$8-$bit constant $\rightarrow ARn$
LARP < *constant* >	Valor inmediato se carga en el apuntador de registros AR	$3-$bit constant $\rightarrow ARP$, $(ARP) \rightarrow ARB$
LDP < *dma* > **LDP** < *ind* > [, < *nextARP* >]	Contenido en dma se guarda en DP	$(dma) \rightarrow DP$
LDPK < *constant* >	Valor inmediato se carga en DP	$9-$bit constant $\rightarrow DP$
LRLK < *AR* >, < *constant* >	Carga valor inmediato largo en el registro auxiliar	$16-$bit constant $\rightarrow ARn$
MAR < *dma* > **MAR** < *ind* > [, < *nextARP* >]	Se modifica el registro auxiliar AR(ARP)	haga posible variar contenido de AR(ARP) a ARP
SAR < *AR* >, < *dma* > **SAR** < *AR* >, < *ind* > [, < *nextARP* >]	Valor en dma se guarda en el registro auxiliar	$(ARn) \rightarrow dma$
SBRK < *constant* >	Valor inmediato corto se sustrae del registro auxiliar	$(ARn) - 8-$bit constant $\rightarrow ARn$

Tabla 2.2. Instrucciones del DSP TMS320C25 que trabajan con registros auxiliares

SINTAXIS EN ENSAMBLADOR	TÍTULO	DESCRIPCIÓN DE OPERACIONES
APAC	Suma el registro P al acumulador	$(ACC) + (shifted\ Preg) \rightarrow ACC$
LPH $< dma >$ **LPH** $< ind >$ [, $< nextARP >$]	Carga valor en dma en el registro P alto	$(dma) \rightarrow Preg(31-16)$
LT $< dma >$ **LT** $< ind >$ [, $< nextARP >$]	Carga valor en dma en el registro T	$(dma) \rightarrow Treg$
LTA $< dma >$ **LTA** $< ind >$ [, $< nextARP >$]	Carga valor en dma en T y se suma registro P al acumulador	$(dma) \rightarrow Treg$ $(ACC) + (shifted\ Preg) \rightarrow ACC$
LTD $< dma >$ **LTD** $< ind >$ [, $< nextARP >$]	Carga valor dma en T, suma P al ACC y corre los datos	$(dma) \rightarrow Treg$, $(dma) \rightarrow dma + 1$ $(ACC) + (shift\ Preg) \rightarrow ACC$
LTP $< dma >$ **LTP** $< ind >$ [, $< nextARP >$]	Carga valor en dma en T y contenido de P al acumulador	$(dma) \rightarrow Treg$ $(shifted\ Preg) \rightarrow ACC$
LTS $< dma >$ **LTS** $< ind >$ [, $< nextARP >$]	Carga valor dma en T y sustrae del acumulador	$(dma) \rightarrow Treg$ $(ACC) - (shifted\ Preg) \rightarrow ACC$
MAC $< pma >, < dma >$ **MAC** $< pma >, < ind >$ [, $< nextARP >$]	Multiplica y acumula	$(ACC) + (shifted\ Preg) \rightarrow ACC$ $(pma)x(dma) \rightarrow Preg$
MACD $< pma >, < dma >$ **MACD** $< pma >, < ind >$ [, $< nextARP >$]	Multiplica, suma y desplaza los datos	$(ACC) + (shifted\ Preg) \rightarrow ACC$ $(pma)x(dma) \rightarrow Preg$, $(dma) \rightarrow dma + 1$
MPY $< dma >$ **MPY** $< ind >$ [, $< nextARP >$]	Multiplica contenido de dma con T y el producto se guarda en T	$(Treg)x(dma) \rightarrow Preg$
MPYA $< dma >$ **MPYA** $< ind >$ [, $< nextARP >$]	Multiplica y acumula P con el acumulador	$(ACC) + (shifted\ Preg) \rightarrow ACC$ $(Treg)x(dma) \rightarrow Preg$
MPYK $< constant >$	Multiplica constante con T y se guarda en P	$(Treg)x13-bit\ constant \rightarrow Preg$

Tabla 2.3. Instrucciones del DSP TMS320C25 que trabajan con registros de multiplicador de hardware

SINTAXIS EN ENSAMBLADOR	TÍTULO	DESCRIPCIÓN DE OPERACIONES
MPYS < *dma* > **MPYS** < *ind* > [, < *nextARP* >]	Multiplica y sustrae el producto anterior	$(ACC) - (shifted\ Preg) \rightarrow ACC$ $(Treg)x(dma) \rightarrow Preg$
MPYU < *dma* > **MPYU** < *ind* > [, < *nextARP* >]	Multiplica sin tomar en cuenta el signo	$Usgn(Treg)xUsgn(dma) \rightarrow Preg$
PAC	Registro P desplazado se guarda en el acumulador	$(shifted\ Preg) \rightarrow ACC$
SPAC	Sustrae el registro P del acumulador	$(ACC) - (shifted\ Preg) \rightarrow ACC$
SPH < *dma* > **SPH** < *ind* > [, < *nextARP* >]	Registro P desplazado se guarda en dma	$(shifted\ Preg(31-16)) \rightarrow dma$
SPL < *dma* > **SPL** < *ind* > [, < *nextARP* >]	Carga parte baja de P en dma	$(shifted\ Preg(15-0)) \rightarrow dma$
SPM < *constant* >	Pone el registro P en la salida en modo de corrimiento	2–bit constant $\rightarrow PM$
SQRA < *dma* > **SQRA** < *ind* > [, < *nextARP* >]	Suma el acumulador con el registro P y el cuadrado del dma	$(ACC) + (shifted\ Preg) \rightarrow ACC$ $(dma)x(dma) \rightarrow Preg$
SQRS < *dma* > **SQRS** < *ind* > [, < *nextARP* >]	Cuadrado del contenido de dma y lo substrae del producto anterior	$(ACC) - (shifted\ Preg) \rightarrow ACC$ $(dma)x(dma) \rightarrow Preg$

Tabla 2.3. Instrucciones del DSP TMS320C25 que trabajan con los registros del multiplicador de hardware – *continuación*

SINTAXIS EN ENSAMBLADOR	TÍTULO	DESCRIPCIÓN DE OPERACIONES
B $< pma >$ [, $< ind >$ [, $< nextARP >$]]	Salto incondicional	$pma \rightarrow PC$
BACC	Salto a la dirección especificada por el acumulador	$(ACC(15-0)) \rightarrow PC$
BANZ $< pma >$ [, $< ind >$ [, $< nextARP >$]]	Salto a pma si el registro auxiliar no es cero	if $(AR(ARP)) \neq 0$: then $pma \rightarrow PC$ if $(AR(ARP)) = 0$: then $(PC)+2 \rightarrow PC$
BBNZ $< pma >$ [, $< ind >$ [, $< nextARP >$]]	Salto si el bit TC no es cero	if $(TC) = 1$: then $pma \rightarrow PC$ if $(TC) = 0$: then $(PC)+2 \rightarrow PC$
BBZ $< pma >$ [, $< ind >$ [, $< nextARP >$]]	Salto si el bit TC es cero	if $(TC) = 0$: then $pma \rightarrow PC$ if $(TC) = 1$: then $(PC)+2 \rightarrow PC$
BC $< pma >$ [, $< ind >$ [, $< nextARP >$]]	Salto si el carry está puesto	if $(C) = 1$: then $pma \rightarrow PC$ if $(C) = 0$: then $(PC)+2 \rightarrow PC$
BGEZ $< pma >$ [, $< ind >$ [, $< nextARP >$]]	Salto si el acumulador ≥ 0	if $(ACC) \geq 0$: then $pma \rightarrow PC$ if $(ACC) < 0$: then $(PC)+2 \rightarrow PC$
BGZ $< pma >$ [, $< ind >$ [, $< nextARP >$]]	Salto si el acumulador > 0	if $(ACC) > 0$: then $pma \rightarrow PC$ if $(ACC) \leq 0$: then $(PC)+2 \rightarrow PC$
BIOZ $< pma >$ [, $< ind >$ [, $< nextARP >$]]	Salto dependiente al bit de($overline BIO$)	if $(\overline{BIO}) = 0$: then $pma \rightarrow PC$ if $(\overline{BIO}) = 1$: then $(PC)+2 \rightarrow PC$
BLEZ $< pma >$ [, $< ind >$ [, $< nextARP >$]]	Salto si acumulador ≤ 0	if $(ACC) \leq 0$: then $pma \rightarrow PC$ if $(ACC) > 0$: then $(PC)+2 \rightarrow PC$
BLZ $< pma >$ [, $< ind >$ [, $< nextARP >$]]	Salto si acumulador < 0	if $(ACC) < 0$: then $pma \rightarrow PC$ if $(ACC) \geq 0$: then $(PC)+2 \rightarrow PC$
BNC $< pma >$ [, $< ind >$ [, $< nextARP >$]]	Salto si el carry no está puesto	if $(C) = 0$: then $pma \rightarrow PC$ if $(C) = 1$: then $(PC)+2 \rightarrow PC$
BNV $< pma >$ [, $< ind >$ [, $< nextARP >$]]	Salto si OV no está puesto	if $(OV) = 1$: then $pma \rightarrow PC$ if $(OV) = 0$: then $(PC)+2 \rightarrow PC$
BNZ $< pma >$ [, $< ind >$ [, $< nextARP >$]]	Salto si acumulador $\neq 0$	if $(ACC) \neq 0$: then $pma \rightarrow PC$ if $(ACC) = 0$: then $(PC)+2 \rightarrow PC$
BV $< pma >$ [, $< ind >$ [, $< nextARP >$]]	Salto si OV está puesto	if $(OV) = 0$: then $pma \rightarrow PC$ if $(OV) = 1$: then $(PC)+2 \rightarrow PC$

Tabla 2.4. Instrucciones de salto y llamada de las subrutinas

SINTAXIS EN ENSAMBLADOR	TÍTULO	DESCRIPCIÓN DE OPERACIONES
BZ $< pma >[,< ind >[,< nextARP >]]$	Salto si acumulador = 0	if $(ACC) = 0$: then $pma \rightarrow PC$ if $(ACC) \neq 0$: then $(PC) + 2 \rightarrow PC$
CALA	Se llama a subrutina indirectamente	$(ACC(15-0)) \rightarrow PC$, $(PC) + 1 \rightarrow TOS$
CALL $< pma >,< ind >[,< nextARP >]$	Llamado a subrutina	$(PC) + 2 \rightarrow TOS$, $pma \rightarrow PC$
RET	Regresa de la subrutina	$(TOS) \rightarrow PC$

Tabla 2.4 Instrucciones del salto y llamada de las subrutinas - *continuación*

SINTAXIS EN ENSAMBLADOR	TÍTULO	DESCRIPCIÓN DE OPERACIONES
BIT $< dma >,< bit\ code >$ **BIT** $< ind >,< bit\ code >[,< nextARP >]$	Se prueba bit	(dma bit at (15− bit code)) $\rightarrow TC$
BITT $< dma >$ **BITT** $< ind >[,< nextARP >]$	Bit de prueba especificado con el registro T	(dma bit at (15− Treg)) $\rightarrow TC$
CNFD	Configura el bloque como la memoria de datos	$0 \rightarrow CNF$
CNFP	Configura el bloque como la memoria de programa	$1 \rightarrow CNF$
DINT	Deshabilita interrupción	$1 \rightarrow INTM$
EINT	Habilita	$0 \rightarrow INTM$
IDLE	Esperar a la interupción	$(PC) + 1 \rightarrow PC$, *powerdown*
LST $< dma >[,< ind >[,< nextARP >]]$	Cargar el registro de estado ST0	$(dma) \rightarrow ST0$
LST1$< dma >[,< ind >[,< nextARP >]]$	Cargar el registro de estado ST1	$(dma) \rightarrow ST1$
NOP	Ninguna operación	$(PC) + 1 \rightarrow PC$
POP	La parte alta del stack al ACCL	$(TOS) \rightarrow ACC$
POPD$< dma >[,< ind >[,< nextARP >]]$	La parte alta del stack a la memoria de datos	$(TOS) \rightarrow dma$
PSHD$< dma >[,< ind >[,< nextARP >]]$	Cargar los datos de la memoria al stack	$(dma) \rightarrow TOS$

Tabla 2.5. Instrucciones de estado y para dirección

SINTAXIS EN ENSAMBLADOR	TÍTULO	DESCRIPCIÓN DE OPERACIONES
PUSH	Cargar la parte baja del ACC al stack	$(ACCL) \rightarrow TOS$
RC	Reset el bit de carry	$0 \rightarrow C$
RHM	Reset el modo de hold	$0 \rightarrow HM$
ROVM	Reset el modo de sobreflujo	$0 \rightarrow OVM$
RPT $< dma >$ [, $< ind >$ [, $< nextARP >$]]	Se repite la instrucción como está especificado con (dma)	$(dma) \rightarrow RPTC$
RPTK $< constant >$	Se repite la instrucción como está especificado con constante	8−bit constant $\rightarrow RPTC$
RSXM	Reset al modo de extensión al signo	$0 \rightarrow SXM$
RTC	Reset especificado con el bit en TC	$0 \rightarrow TC$
SC	Configurar el bit de carry	$1 \rightarrow C$
SHM	Configurar el modo de hold	$1 \rightarrow HM$
SOVM	Configurar el modo de sobreflujo	$1 \rightarrow OVM$
SST $< dma >$ [, $< ind >$ [, $< nextARP >$]]	Cargar el estado del registro ST0 al dma	$ST0 \rightarrow dma$
SST1$< dma >$ [, $< ind >$ [, $< nextARP >$]]	Cargar el estado del registro ST1 al dma	$ST1 \rightarrow dma$
SSXM	Configurar el modo de sign−extension	$1 \rightarrow SXM$
STC	Configurar la bandera de test/control	$1 \rightarrow TC$
TRAP	Interrupción con software	$(PC) + 1 \rightarrow TOS$, $30 \rightarrow PC$

Tabla 2.5. Instrucciones de estado y para dirección – *continuación*

SINTAXIS EN ENSAMBLADOR	TÍTULO	DESCRIPCIÓN DE OPERACIONES
BLKD $< dma1 >, < dma2 >$ **BLKD** $< dma1 >, < ind > [, < nextARP >]$	Carga los datos desde la memoria de datos a la memoria de datos	$(dma1,$ *addressed by* $PC) \rightarrow dma2$
BLKP $< pma >, < dma >$ **BLKP** $< pma >, < ind > [, < nextARP >]$	Carga los datos desde la memoria de programa a la memoria de datos	$(pma,$ *addressed by* $PC) \rightarrow dma$
DMOV $< dma >$ **DMOV** $< ind > [, < nextARP >]$	Mueve los datos dentro de la memoria de datos	$(dma) \rightarrow dma + 1$
FORT $< constant >$	Carga la constante en FO (Format serial port register)	$1-$bit constant $\rightarrow FO$
IN $< dma >, < PA >$ **IN** $< ind >, < PA > [, < nextARP >]$	Carga los datos desde el puerto al dma	$($*data bus*, *addressed by* $PA) \rightarrow dma$
OUT $< dma >, < PA >$ **OUT** $< ind >, < PA > [, < nextARP >]$	Carga los datos desde dma al puerto	$(dma) \rightarrow$ *data bus*, *addressed by* PA
RFSM	Configura el FSM para la sincronización del puerto serial	$0 \rightarrow FSM$
RTXM	Configura el TXM para el modo de transmisión	$0 \rightarrow TXM$
RXF	Configura la bandera externa	$0 \rightarrow XF$
SFSM	Configura el FSM para la sincronización del puerto serial para la transmisión	$1 \rightarrow FSM$
STXM	Configura el TXM para el modo de transmisión	$1 \rightarrow TXM$
SXF	Configura la bandera externa	$1 \rightarrow XF$
TBLR $< dma >$ **TBLR** $< ind > [, < nextARP >]$	Lee tabla de datos	$(pma,$ *addressed by* $ACC(15-0)) \rightarrow dma$
TBLW $< dma >$ **TBLW** $< ind > [, < nextARP >]$	Escribe una tabla de datos	$(dma) \rightarrow pma,$ *addressed by* $ACC(15-0))$

Tabla 2.6. Instrucciones de entrada/salida e instrucciones para el trabajo con la memoria de datos

Capítulo 3

Realización de filtros digitales con DSP TMS320C25

En este capítulo se muestran algunas instrucciones importantes para realizar filtros digitales y ejemplos de programas en lenguaje ensamblador. También se describe cómo se trabaja con el simulador del DSP TMS320C25.

3.1 Instrucciones del DSP TMS320C25

Los filtros digitales necesitan sumas, multiplicaciones y elementos de retraso. Por eso, para los DSP se crearon las operaciones de multiplicar, sumar y retardar las señales usando una instrucción simple. En este capítulo explicamos algunas instrucciones que se utilizan al escribir los programas en ensamblador para la realización de los filtros digitales.

Las instrucciones que retardan y transladan la señal de una dirección a otra son MACD, LTD, DMOV.

3.1.1 MACD (Multiply and Accumulate with Data Move)

Sintaxis

Directo:	MACD $< pma >, < dma >$
Indirecto:	MACD $< pma >$,{ind}[,$nextARP$]

Operandos

$0 \leq pma \leq 65535$
$0 \leq dma \leq 127$
$0 \leq nextARP \leq 7$

El operando $\{ind\}$ sirve para el direccionamiento indirecto y puede tomar la forma:

- $\boxed{*}$ El contenido del registro auxiliar AR(ARP) se emplea para el direccionamiento de la memoria.

- *− El contenido del registro auxiliar AR(ARP) se utiliza para el direccionamiento de la memoria y después de realizar la orden, el contenido de AR(ARP) se disminuye en uno.

- *+ El contenido del registro auxiliar AR(ARP) sirve para el direccionamiento de la memoria de datos y después de realizar la orden, el contenido de AR(ARP) se incrementa en uno.

- *0− El contenido de AR(ARP) es la dirección de la memoria de datos y después de realizar la orden, el contenido de ARP se sustrae del contenido de AR0.

- *0+ El contenido de AR(ARP) es la dirección de la memoria de datos y después de realizar la operación, el contenido de AR(ARP) se agrega al contenido de AR0.

Secuencia de operaciones que realiza la instrucción MACD:

(PC)+2 → PC
(PFC)→ MCS
(pma)→ PFC
Si el contador $\neq 0$:
(ACC)+(el registro translado P)→ ACC
(dma)→ T registro
(dma)×(pma direccionado PFC)→ P registro
Modificación AR(ARP) y ARP según exigencia
(PFC)+1→ PFC
(contador)-1→ contador
De otro modo (ACC)+(el registro P traslado)→ ACC
(dma)→ T registro
(dma)×(pma direccionada de PFC)→ P registro
Modificación AR(ARP) y ARP según la exigencia
(MCS)→ PFC

Ejecución

Con el comando MACD se puede multiplicar el valor guardado en la memoria de datos (con dirección dma) con el número guardado en la memoria de datos (con la dirección pma), se suma el resultado anterior al contenido del acumulador. El bloque B0 debe ser especificado como la memoria de programa con la dirección $< FF00$ hasta $< FFFF$, desde 65280 hasta 65535.

El comando MACD hace lo mismo que el comando MAC, y además corre los datos en el bloque B0, B1 o B2. También se utiliza, principalmente, cuando calculamos la convolución de dos señales o si realizamos filtros transversales. Este comando se puede utilizar con los comandos de repetición RPT o RPTK, lo cual optimiza y hace más potente un programa.

Ejemplo

```
SPM    0          ; Ningún aplazamiento en la salida
SOVM              ; El modo de saturación
CNFP              ; Configura el bloque B0 como la memoria de programa
LARP   AR3        ; Actualización del registro AR3 para el
                  ; direccionamiento del bloque B1
LRLK   AR3,1023   ; Guarda la dirección 1023 en AR3
RPTK   255        ; Se repite el siguiente comando 256 veces
MACD   >FF00,*-   ; Multiplica, suma, desplaza los datos en el bloque B1
                  ; y decrementa el número en AR3
```

En la figura 3.1 se muestra el procedimiento y cómo cambian los registros si se ejecuta el comando MACD.

El contenido del registro AR1 es $3FD_H = 1021_D$. En la dirección 1021 se guarda el número $23_H = 35_D$. Después de la ejecución del comando MACD, el número 23_H se traslada de la memoria de datos 1021 a la memoria de datos 1022, con esto se simula el elemento de retraso en el circuito. En el registro de 16 bits PFC está el número $FF02_H = 65282_D$ y en la dirección 65282_D está el número por el cual se multiplica el número guardado en la dirección 1021.

En el registro RPT está el número $FD_H = 253_D$, que se decrementa después de la ejecución del comando y sirve como contador, además determina cuántas veces se repite el comando.

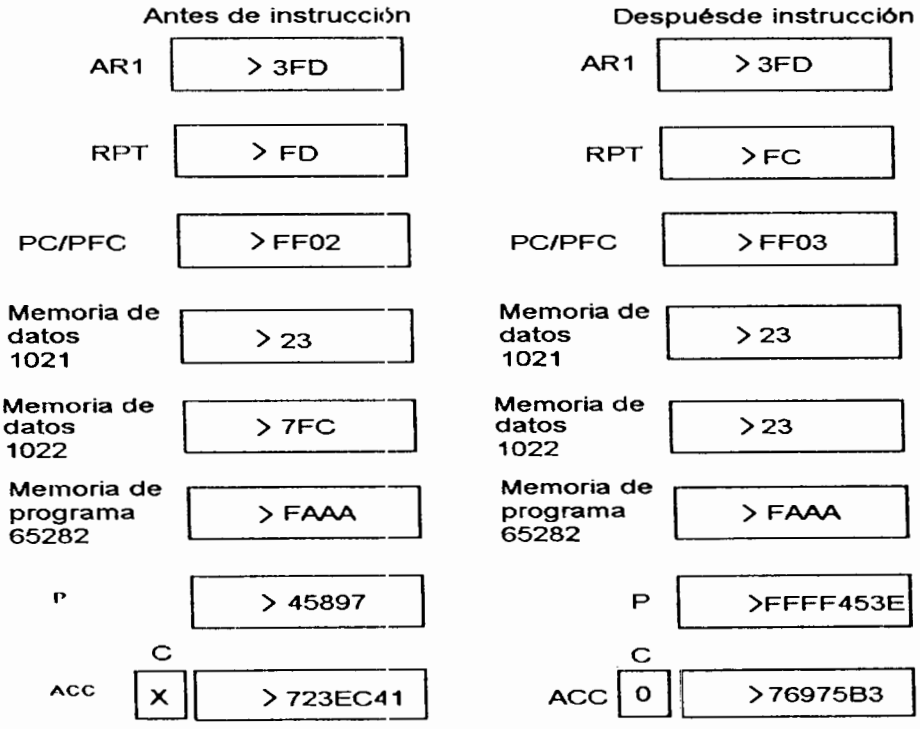


Figura 3.1. Cambio de los valores en la memoria y en los registros antes y después del MACD

Entonces, después de la instrucción estará en la segunda fila el número $FC_{H} = 252_{D}$.

Si sumamos el número guardado en el registro P con el número en el acumulador ACC, obtenemos el valor que se guarda en ACC.

$$458972_{H} + 723EC41_{H} = 76975B3_{H}$$

Si estudiamos la figura 3.1 y el esquema del bloque del DSPS, comprenderemos también el comando MAC que es similar.

3.1.2 LT (Load T Register)

Sintaxis

Directo: LT < *dma* >
Indirecto: LT {ind}[, < *next ARP* >]

Operandos

$0 \leq dma \leq 127$
$0 \leq next\ ARP \leq 7$

Ejecución

(PC)+1 → PC
(dma) → T Registro

El contenido de la memoria de datos, que está destinado por la dirección dma, se guarda en el registro T.

Ejemplo

LT *DAT24* $(DP = 8)$
o
LT * Si en el registro auxiliar está el número 1048

En la figura 3.2 se representa el cambio de los números en el registro de la memoria de datos 1048 y en el registro T.

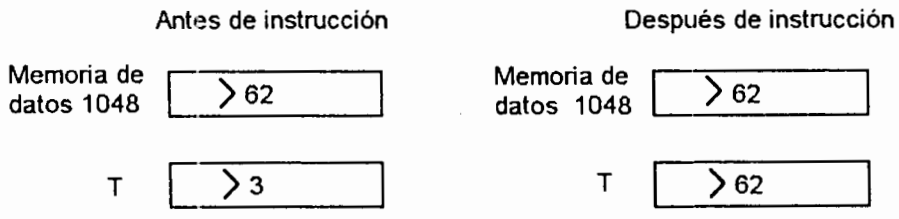


Figura 3.2. Cambio de los números en registros antes y después de la instrucción LT

3.1.3 LTA (Load T Register and Accumulate Previous Product and Move Data)

Sintaxis

Directo: LTA < *dma* >
Indirecto: LTA {ind}[, < *next ARP* >]

Operandos

$0 \leq dma \leq 127$
$0 \leq next\ ARP \leq 7$

Ejecución

(PC)+1 → PC
(dma) → T Registro
(ACC)+(P trasladado) → ACC

El contenido de la memoria de datos que, está destinado por la dirección dma, se guarda en el registro T, y al mismo tiempo en la ALU se suma el contenido del registro P con el contenido de ACC, y el resultado se guarda en el acumulador ACC.

Ejemplo

LTA *DAT36* $(DP = 6, PM = 0)$
o
LTA * Si en registro auxiliar está el número 804

En la figura 3.3 se pueden observar los cambios de los números en el registro de la memoria de datos 804 y en los registros T, P y en ACC.

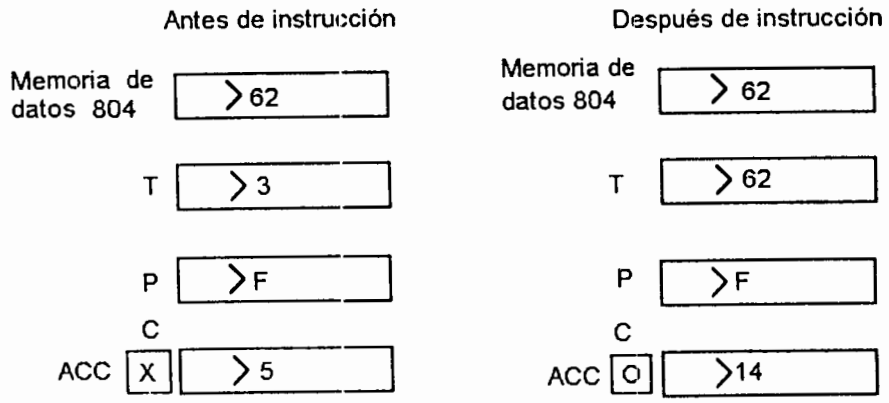


Figura 3.3. Cambio de los números en registros antes y después de la ejecución LTA

3.1.4 LTD (Load T Register, Accumulate Previous Product and Move Data)

Sintaxis

Directo: LTD $< dma >$
Indirecto: LTD {ind}[, $< next\ ARP >$]

Operandos

$0 \leq dma \leq 127$
$0 \leq next\ ARP \leq 7$

Ejecución

(PC)+1 → PC
(dma) → T Registro
(dma) → dma+1
(ACC)+(P) → ACC (el registro P trasladado)

En otras palabras, en el registro T se guarda el contenido de la memoria de datos apuntado por la dirección dma.

Ejemplo

$LTD \quad DAT126 \quad (DP = 7, PM = 0)$
o
$LTD \ *$ Si en registro auxiliar está el número 1022

En la figura 3.4 se muestran los cambios de los números en los registros de la memoria de datos 1022 y 1023, y en los registros T, P y en ACC.

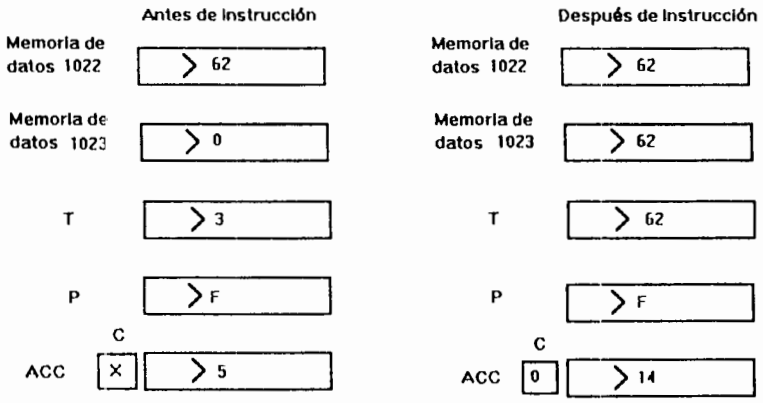


Figura 3.4. Cambio de los números en los registros antes y después del comando LTD

3.1.5 LTP (Load T Register and store P register in Accumulator)

Sintaxis

Directo: LTP $< dma >$
Indirecto: LTP {ind}[$<, next\ ARP >$]

Operandos

$0 \leq dma \leq 127$
$0 \leq nextARP \leq 7$

Ejecución

(PC)+1 → PC
(dma) → T Registro
(P registro trasladado) → ACC

El contenido de la memoria de datos, que está apuntado por la dirección dma, se guarda en el registro T y al mismo tiempo el contenido de registro P trasladado se guarda en el acumulador. El bit de estado PM controla la traslación del contenido del registro P.

Ejemplo

$LTP \quad DAT36 \quad (DP = 6, PM = 0)$
o
$LTP \ *$ Si en el registro auxiliar está el número 804

En la figura 3.5 se indican los cambios de los números en el registro de la memoria de datos 1048, en el registro T y en ACC.

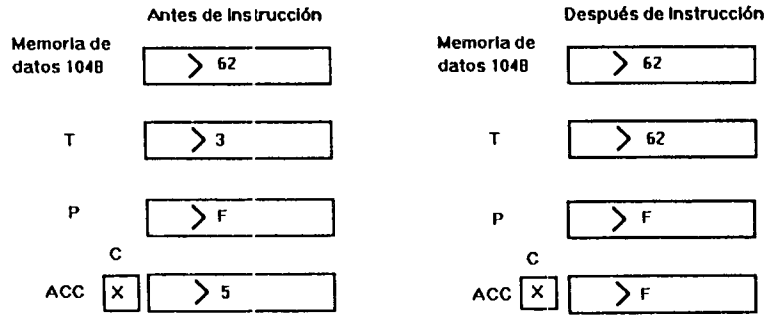


Figura 3.5. Cambio de los números en registros antes y después del comando LTP

3.1.6 PAC (Load Accumulador with P Register)

Sintaxis

Directo: PAC

Ejecución

(PC)+1 → PC
(P registro trasladado) → ACC

El contenido de registro P trasladado, como está especificado con los bits en el registro de estado PM, se guarda en ACC.

Ejemplo

$$PAC \quad (PM = 0)$$

En la figura 3.6 se muestran los cambios de los números en ACC.

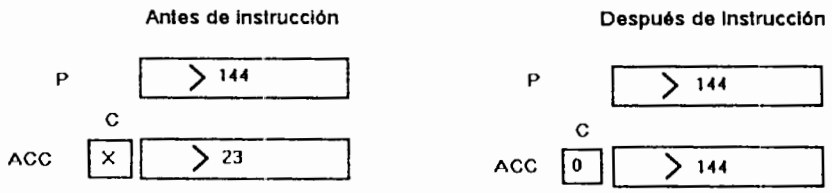


Figura 3.6. Cambio de los números en los registros antes y después de la instrucción PAC

3.1.7 LTS (Load T Register, Substract Previous Product)

Sintaxis

Directo: LTS $< dma >$
Indirecto: LTS /ind/[, $< next\ ARP >$]

Operandos

$0 \leq dma \leq 127$
$0 \leq next\ ARP \leq 7$

Ejecución

(PC)+1 → PC
(dma) → T Registro
(ACC)-(P registro trasladado) → ACC

El contenido de la memoria de datos, que está apuntado por la dirección dma, se guarda en el registro T; al mismo tiempo, el contenido del registro P trasladado se resta con el acumulador y el resultado se guarda en el acumulador. El bit de estado PM controla la traslación del contenido del registro P.

Ejemplo

$LTS \quad DAT36 \quad (DP = 6, PM = 0)$
o
$LTP \ *$ Si en registro auxiliar está el número 804

En la figura 3.7 se pueden observar los cambios de los números en el registro de la memoria de datos 804, en el registro T y en ACC.

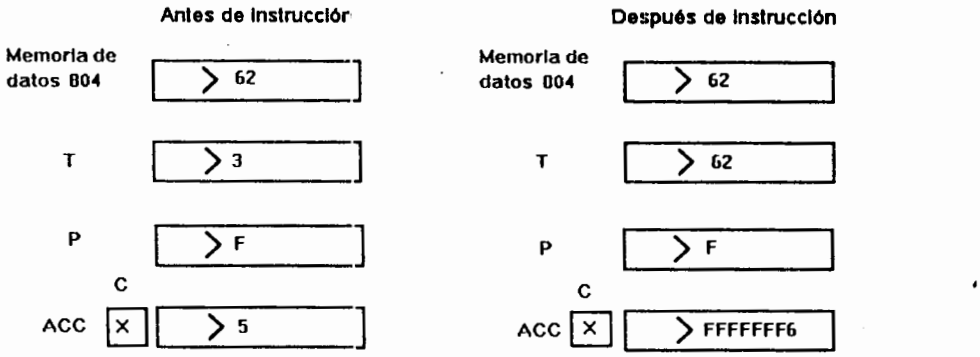


Figura 3.7. Cambio de los números en los registros antes y después de la ejecución de la instrucción LTS

3.1.8 MPY (Multiply)

Sintaxis

Directo: MPY $< dma >$
Indirecto: MPY {ind}[, $< next\ ARP >$]

Operandos

$0 \leq dma \leq 127$
$0 \leq next\ ARP \leq 7$

Ejecución

(PC)+1 → PC
(dma).(T registro) → P registro

El contenido de registro T se multiplica con el contenido de la dirección de la memoria de datos dma y el resultado se guarda en el registro P.

Ejemplo

$MPY \quad DAT13 \quad (DP = 8)$
o
$MPY \ *$ Si en el registro auxiliar está el número 1037

En la figura 3.8 se muestran los cambios de los números en el registro de la memoria de datos 1037 y en el registro P.

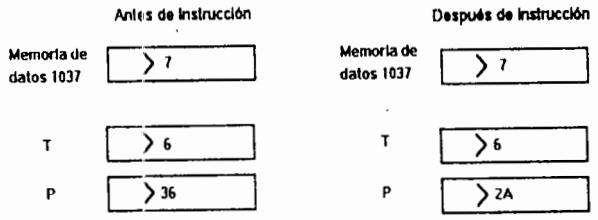


Figura 3.8. Cambio de los números en los registros antes y después de la ejecución de la instrucción MPY

3.1.9 MPYA (Multiply and Accumulate Previous Product)

Sintaxis

Directo: MPYA $< dma >$
Indirecto: MPYA {ind}$[, < next\ ARP >]$

Operandos

$0 \leq dma \leq 127$
$0 \leq next\ ARP \leq 7$

Ejecución

(PC)+1 → PC
(ACC)+(P registro recorido) → ACC registro
(T registro).(dma) → P registro

El contenido del registro T se multiplica con el contenido de la dirección de la memoria de datos dma y el resultado se guarda en el registro P. Al mismo tiempo, el producto anterior se suma con el producto del ACC.

Ejemplo

$MPYA \quad DAT13 \quad (DP = 6, PM = 0)$
o
$MPYA \ *$ Si en el registro auxiliar está el número 781

En la figura 3.9 se pueden observar los cambios de los números en el registro de la memoria de datos 781 en el registro P y en ACC.

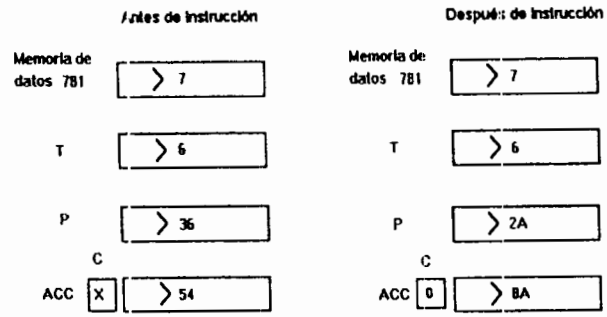


Figura 3.9. Cambio de los números en los registros antes y después de la instrucción MPYA

3.1.10 MPYS (Multiply and Subtract Previous Product)

Sintaxis

Directo: MPYS $< dma >$
Indirecto: MPYS {ind}[, $< next\ ARP >$]

Operandos

$0 \leq dma \leq 127$
$0 \leq next\ ARP \leq 7$

Ejecución

(PC)+1 → PC
(ACC)-(P registro recorrido) → ACC registro
(T registro).(dma) → P registro

El contenido de registro T se multiplica con el contenido de la dirección de la memoria de datos dma y el resultado se guarda en el registro P. Al mismo tiempo, el producto anterior guardado en el registro P se resta con el producto del ACC y el resultado se guarda en ACC.

Ejemplo

$MPYS \quad DAT13 \quad (DP = 6, PM = 0)$
o
$MPYS\ *$ Si en el registro auxiliar está el número 781

En la figura 3.10 se representan los cambios de los números en el registro P y en ACC.

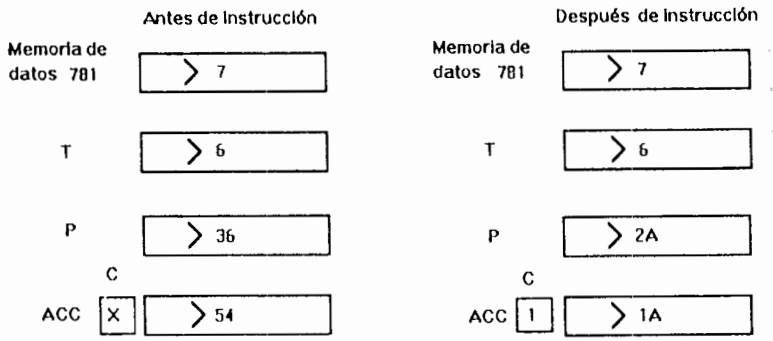


Figura 3.10. Cambio de los números en registros antes y después de la instrucción MPYS

3.1.11 BLKD (Block Move from Data to Data Memory)

Sintaxis

Directo: BLKD $< dma1 >, < dma2 >$
Indirecto: BLKD $< dma1 >$,{ind}[, $< next\ ARP >$]

Operandos

$0 \leq dma1 \leq 65535$
$0 \leq dma2 \leq 127$
$0 \leq next\ ARP \leq 7$

Ejecución

(PC)+→ PC
(PFC)→ MCS
dma1→ PFC
Si contador $\neq 0$:
después dma1 direccionada de PFC→ dma2
Modificación AR(ARP) y ARP según la especificación
(PFC)+1→ PFC
(contador)-1→ contador
o
(dma1,direccionada con PFC)→ dma2
Modificación AR(ARP) y (ARP) según la especificación
(MCS)→ PFC

Con la instrucción BLKD necesitamos utilizar la instrucción RPT o RPTK si queremos transferir más de un número desde la memoria de datos. El número total de los números que queremos transferir desde una memoria a otra debe ser sobre un número más grande que el guardado al final del ciclo en RPTC cero, y si utilizamos el direccionamiento indirecto en el registro auxiliar AR(ARP) debe estar guardada la dirección última del bloque determinado.

Ejemplo

$$RPTK \quad 2$$
$$BLKD \quad > F400, *+$$

Los cambios en los registros antes y después de la ejecución de la instrucción se presentan en la figura 3.11.

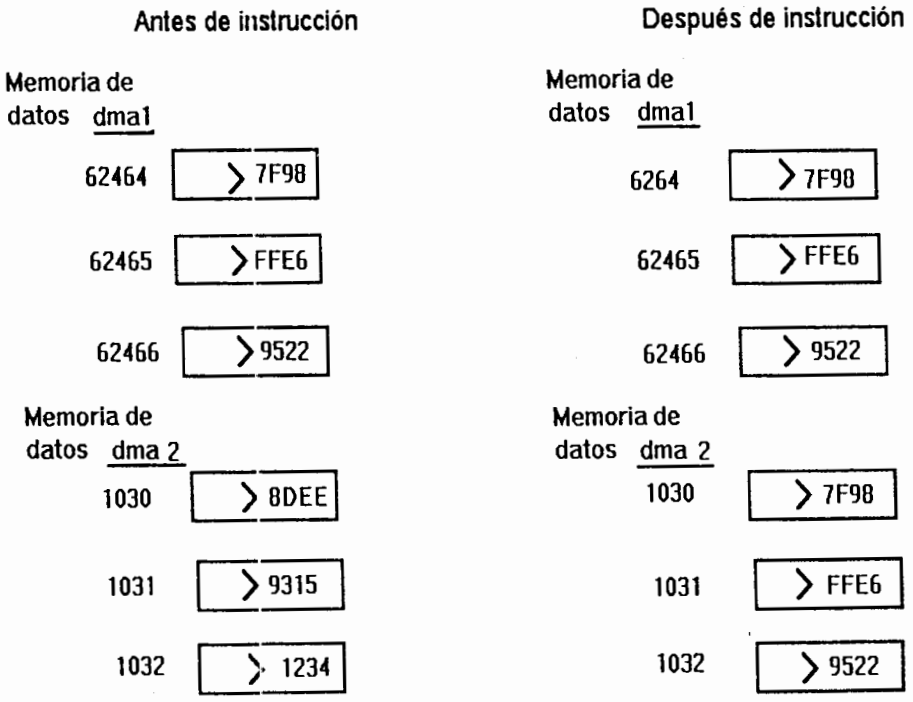


Figura 3.11. Contenido de los registros después de la ejecución de la instrucción BLKD

El ejemplo muestra cómo se utiliza el comando BKLD. Los datos $> 7F98$, $> FFE6$, > 9522 guardados en la memoria de datos dma1 son trasladados a las direcciones 1030, 1031, 1032 en la memoria de datos dma2.

3.1.12 BLKP (Block Move from Program to Data Memory)

Sintaxis

Directo: BLKP $< pma >, < dma >$
Indirecto: BLKP $< pma >$,{ind}[, $< next\ ARP >$]

Operandos

$0 \leq pma \leq 65535$
$0 \leq dma \leq 127$
$0 \leq next\ ARP \leq 7$

Ejecución

(PC)+2 → PC
(PFC) → MCS
(pma) → PFC
Si contador ≠ 0:
después
(pma direccionada de PFC) → dma
Modificación AR(ARP) y ARP según la especificación
(PFC)+1 → PFC
(contador)-1 → contador
o
(pma, d reccionada con PFC) → dma
Modificación AR(ARP) y (ARP) según la especificación
(MCS) → PFC

Con la instrucción BLKP necesitamos utilizar la instrucción RPT o RPTK si queremos transferir más de un número desde la memoria de programa. El número total de los números que queremos transferir desde una memoria a otra debe ser uno más grande que el número guardado al final del ciclo en RPTC cero y si utilizamos el direccionamiento indirecto en el registro auxiliar AR(ARP) debe estar guardada la dirección última del bloque determinado.

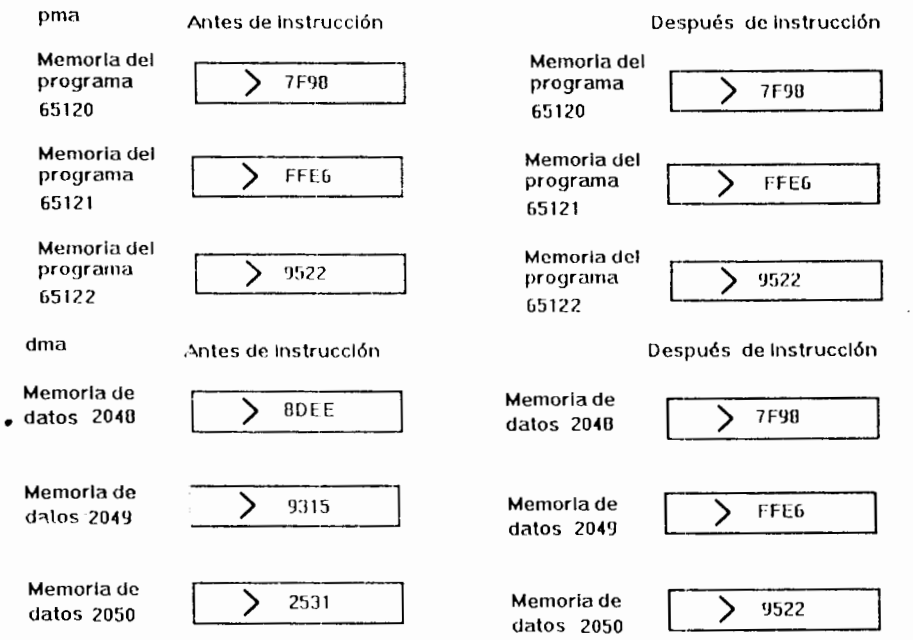


Figura 3.12. Contenido de los registros después de la ejecución de la instrucción BLKP

Ejemplo

RPTK 2
BLKD >65120, *+ ; el contenido del registro auxiliar es 2048

Los cambios en los registros antes y después de la ejecución de la instrucción se representan en la figura 3.12.

3.1.13 TBLR (Table Read)

Sintaxis

Directo: TBLR $< dma >$
Indirecto: TBLR {ind}[, $< next\ ARP >$]

Operandos

$0 \leq dma \leq 127$
$0 \leq next\ ARP \leq 7$

Ejecución

(PC)+1 → PC
(ACC(15-0)) → PFC

Si contador $\neq$ 0:
después
(pma direccionada de PFC)→ dma
Modificación AR(ARP) y ARP según la especificación
(PFC)+1→ PFC
(contador)-1→ contador
o
(pma, direccionada con PFC) → dma
Modificación AR(ARP) y (ARP) según la especificación
(MCS) → PFC

Con la instrucción TBLR se trasladan las palabras desde la memoria de programa a la memoria de datos. La dirección de la memoria de datos está especificada con la instrucción, y el número en acumulador ACCL (los bits menos significativos) indica la dirección en la memoria de programa.

Ejemplo

$TBLR\ *$;el contenido del registro auxiliar es 518

Los cambios en los registros antes y después de la ejecución de la instrucción se muestran en la figura 3.13.

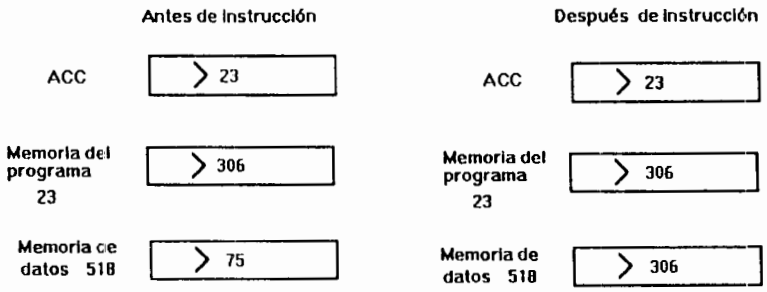


Figura 3.13. Contenido de los registros después de la ejecución de la instrucción TBLR

3.1.14 TBLW (Table Write)

Sintaxis

Directo: TBLW $< dma >$
Indirecto: TBLW {ind} [, $< next\ ARP >$]

Operandos

$0 \leq dma \leq 127$
$0 \leq next\ ARP \leq 7$

Ejecución

(PC)+1 → PC
(ACC(15-0)) → PFC
Si contador $\neq 0$:
después
(pma, direccionada con PFC)→ dma
Modificación AR(ARP) y ARP según la especificación
(PFC)+1→ PFC
(contador)-1→ contador
o
(dma, direccionada con PFC)→ pma
Modificación AR(ARP) y (ARP) según la especificación
(MCS)→ PFC

Con la instrucción TBLW se trasladan las palabras desde la memoria de datos a la memoria de programa. La dirección de la memoria de datos está especificada con la instrucción, y el número guardado en el acumulador ACCL (los bits menos significativos) especifica la dirección en la memoria de programa.

Ejemplo

$TBLW \quad *$;el contenido del registro auxiliar es 4101

Los cambios en los registros antes y después de la ejecución de la instrucción se pueden observar en la figura 3.14.

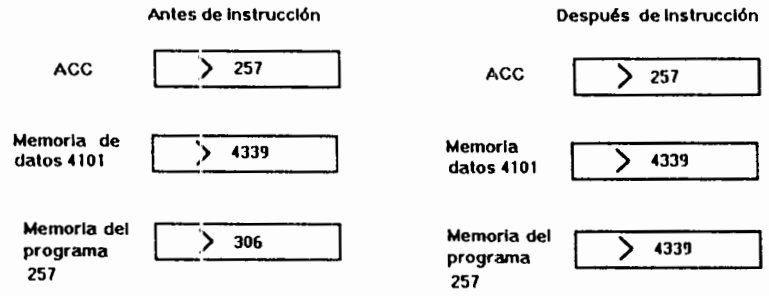


Figura 3.14. Contenido de los registros después de la ejecución de la instrucción TBLW

3.1.15 IN (Input Data from Port)

Sintaxis

Directo: IN $< dma >, < PA >$
Indirecto: IN {ind},$< PA >$ [, $< next\ ARP >$]

Operandos

$0 \leq dma \leq 127$
$0 \leq next\ ARP \leq 7$
$0 \leq dirección\ del\ puerto PA \leq 15$

Ejecución

(PC)+1 → PC
La dirección del puerto → dirección del bus A3-A0
0 → dirección del bus A15-A4
Los datos del bus D15-D0 → dma

La instrucción IN lee el valor de la constante expresada por 16 bits desde un puerto externo y la guarda en la memoria de datos del chip.

Ejemplo

$$IN \quad STAT, PA5$$

Desde un archivo PA en el puerto 5 se lee la palabra de 16 bits y se guarda en la memoria RAM en la variable STAT anteriormente declarada.

3.1.16 OUT (Output Data to Port)

Sintaxis

Directo: OUT $< dma >, < PA >$
Indirecto: OUT {ind},$< PA >$ [, $< nextARP >$]

Operandos

$0 \leq dma \leq 127$
$0 \leq next\ ARP \leq 7$
$0 \leq dirección\ del\ puerto\ PA\ \leq 15$

Ejecución

(PC)+1 → PC
La dirección del puerto PA → T
dirección del bus A3-A0
0 → a la dirección del bus A15-A4
(dma) → bus de datos D15-D0

La instrucción OUT guarda el valor de la memoria expresada por 16 bits en el puerto externo especificado.

Ejemplo

$$OUT \quad STAT, PA5$$

El valor de la variable STAT se guarda en archivo PA en el puerto 5.

3.1.17 B (Branch Unconditionally)

Sintaxis

Indirecto: OUT $< pma >$ [, $\{ind\}$[, $< next\ ARP >$]]

Operandos

$0 \leq pma \leq 65535$
$0 \leq next\ ARP \leq 7$

Ejecución

(pma) → PC Se modifica AR(ARP) como está especificado por ARP

Se modifica el registro auxiliar como está especificado y el programa se corre desde la dirección guardada en la memoria pma.

Ejemplo

$$B \quad PRG191$$

La constante 191 se guarda en el contador de programa y el programa se ejecuta desde esta dirección.

3.2 Programas para los filtros FIR

3.2.1 Programa del filtro FIR con los comandos LTD, MPY

A continuación se presenta el programa para el filtro FIR de orden n=4. La estructura se muestra en la figura 3.15.

```
DALSI  IN     XN, PA2  ; Entrada de la muestra del puerto PA2
       ZAC             ; Anulación del acumulador
       LT     D4       ;< D4 >→ T
       MPY    H4       ;< D4.H4 >→ P
       LTD    D3       ;< D3 >→ D4; < D3 >→ T
                       ;< ACC > + < P >→ ACC
       MPY    H3       ;< D3.H3 >→ P
       LTD    D2       ;< D2 >→ D3; < D2 >→ T
                       ;< H4.D4 + D3.H3 >→ ACC
       MPY    H2       :< D2.H2 >→ P
       LTD    D1       ;< D1 >→ D2; < D1 >→ T
                       ;< D2.H2 + D3.H3 + D4.H4 >→ ACC
       MPY    H1       ;< D1.H1 >→ P
       LTD    XN       ;< XN >→ D1; < XN >→ T;
                       ;< D1.H1 + D2.H2 + D3.H3 + D4.H4 >→ ACC
       MPY    H0       ;< D0.H0 >→ P
       APAC            ;< ACC > + < P >→ ACC
       SACH   YN, 1    ; El resultado se guarda en YN
       OUT    YN, PA2  ;< YN > −− > PA2
       B      DALSI    ; Iniciación del procesamiento de la nueva
                       ; muestra (etiqueta DALSI).
```

3.2.2 Programa del filtro FIR con la instrucción BANZ

Con el comando BANZ se describe el programa para el filtro FIR mostrado en la figura 3.15.

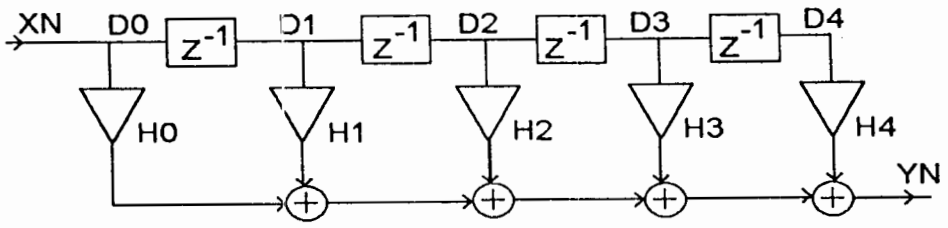


Figura 3.15. Filtro FIR del orden n=4 con cinco coeficientes de la respuesta al impulso

```
        LARP   AR0        ; Actualiza AR0
                          ; de registro auxiliar
DAL     IN     XN, PA1    ; Entrada de nueva muestra desde PA1
        LARK   AR0, D4    ; AR0 indica a la variable D4
        LARK   AR1, H4    ; AR1 indica a la variable H4
        ZAC               ; anulación del acumulador
        LT     *-, AR1    ; < H4 > → T
        MPY    *-, AR0    ; < D4.H4 > → P
SMY     LTD    *-, AR1    ; El contenido de dirección especificada
                          ; registro auxiliar AR1 → T
                          ; El contenido de la dirección especificada por AR1
                          ;  se decrementa sobre 1.
        MPY    *-, AR0    ; El contenido de la dirección especificada por registro auxiliar
                          ; AR0 se multiplica por el contenido de registro T
        BANZ   SMY        ; Si el contenido de la dirección AR0 no es cero
                          ; corre el programa desde la etiqueta SMY
        APAC              ;  Suma el contenido del registro P con ACC
        SACH   YN, 1      ; Guarda el resultado al YN
        OUT    YN, PA2    ; La salida hacia al puerto PA2
        B      DAL        ; Corre el programa nuevamente desde la etiqueta DAL
```

En el comando LARK AR0, D4 de este programa $D4$ es la dirección de la memoria, además en lugar de $D4$ puede ser escrito el número hexadecimal > 200.

3.2.3 Programa del filtro FIR usando MACD y RPTK

En este programa se puede observar que la instrucción MACD combina LTD y MPY en una sola instrucción.

```
        CNFP              ; El bloque B0 se configura como
                          ; la memoria del programa
DALSI   IN     XN, PA1    ;  Guarda la nueva muestra XN desde el archivo PA1
        LRLK   AR1, > 3FF , Inicia la dirección inferior del bloque B1
        LARP   AR1        ; Actualiza el registro
                          ;  auxiliar AR1
        MPYK   0          ; Anulación del registro P
        ZAC               ; Anulación del acumulador
        RPTK   MN1        ; Repite la instrucción N-1veces
        MACD   > 302, *-  ; Multiplica, suma y traslada
        APAC              ; Suma el registro P con el acumulador
        SACH   YN, 1      ; Guarda contenido de ACC en YN
        OUT    YN, PA2    ; Salida de las muestras en el archivo PA2
        B      DALSI      ; Corre el programa desde la etiqueta DALSI
```

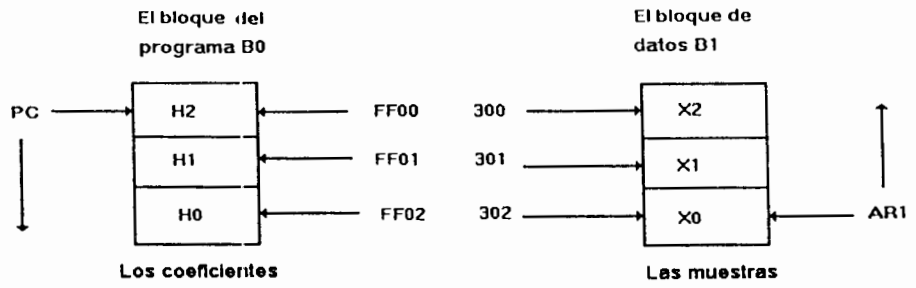


Figura 3.16. Distribución de las direcciones en los bloques B0 y B1

La figura 3.16 muestra la distribución de las direcciones en el bloque $B0$ y $B1$ para el direccionamiento indirecto.

Para el direccionamiento indirecto utilizamos el registro auxiliar AR1. Para los filtros que tienen el menor orden, la combinación de las instrucciones $MACD$ a $RPTK$ no es efectiva. Por eso usamos esta combinación para el diseño de los filtros FIR donde se utilizan los filtros de orden $n = 64, 128$ hasta 256.

La ventaja de la instrucción $MACD$ se muestra cuando tenemos las órdenes del filtro más grandes que $n = 4$. Para los filtros que tienen menor orden utilizamos mejor las instrucciones LTD,MPY y no las instucciones RPT,MACD. Por ejemplo, el filtro FIR con 256 coeficientes realizados con las instrucciones RPT,MACD necesita en la memoria de programa el mismo espacio que el programa para el filtro FIR con cuatro coeficientes.

3.3 Programas en ensamblador para los filtros IIR

A continuación se presentan las programas en ensamblador para el filtro digital cuya función de transferencia toma la forma

$$H(z) = \frac{0.18732 + 0.33601z^{-1} + 0.47753z^{-2} + 0.3359z^{-3} + 0.18713z^{-4}}{1 - 0.4396z^{-1} + 1.1724z^{-2} - 0.38597z^{-3} + 0.26753z^{-4}} \tag{3.1}$$

para diferentes estructuras:

- de cascada, directa, paralela,
- de filtro de estado y filtro de la forma cruz.

3.3.1 Estructura de cascada

Para este caso se tiene que dividir la función de transferencia 3.1 en dos funciones de transferencia parciales $H_1(z)$ y $H_2(z)$. Calculando obtenemos

$$H_1(z) = \frac{0.242342 + 0.339521z^{-1} + 0.242117z^{-2}}{1 - 0.447687z^{-1} + 0.308310z^{-2}}$$

$$H_2(z) = \frac{0.77299 + 0.303581z^{-1} + 0.772887z^{-2}}{1 + 0.00808z^{-1} + 0.867723z^{-2}}$$

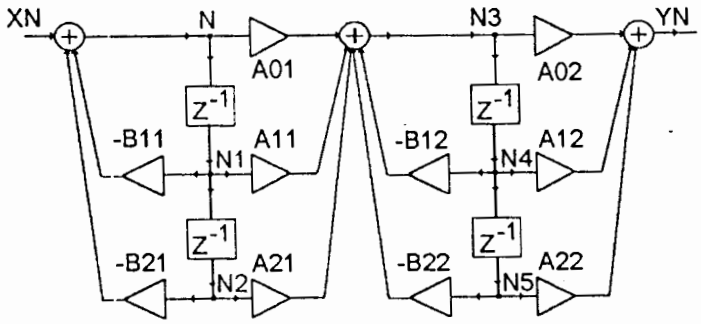


Figura 3.17. Realización de cascada de la función de transferencia de 4o orden

La estructura para la cual escribimos el programa en ensamblador se observa en la figura 3.17.

El programa con los coeficientes del filtro en la forma hexadecimal es

```
*               FILTRO DE CASCADA DE 4o ORDEN              *
*************************************************************
*                    -1      -2              -1      -2     *
*          A01+A11*z  +A21*z       A02+A12*z  +A22*z       *
* H(z)  = ------------------  x ------------------------    *
*                   -1      -2              -1      -2      *
*           1+B11*z  +B21*z         1+B12*z  +B22*z         *
*************************************************************
*
       AORG >0000
RESET  B    INI
       AORG >0020
*
* INICIALIZACION DEL MICROCONTROLADOR
*
INI SOVM             ;Trabaja con saturación
    LDPK 0           ;Trabaja con la página de memoria 0
    ZAC              ;Ceros en el acumulador
    LARP AR2         ;Se actualiza el registro auxiliar AR2
    LRLK AR2,>0060   ;Comienzo del bloque B2
    RPTK 31          ;El comando que sigue se repite 32 veces
    SACL *+          ;En la memoria del bloque B2 se guardan ceros
    LRLK AR2,>0060   ;Comienzo del bloque  B2
    RPTK 13          ;La instrucción que sigue se repite 14 veces
    BLKP COEF,*+     ;Los datos desde el archivo COEF se guardan en B2
    LACK 1           ;En el acumulador se guarda 1
    SACL ONE         ;En variable  ONE se guarda un bit de redondeo
```

```
*
* DECLARACION DE LAS VARIABLES
*
A01    EQU  >0060    ;\
A11    EQU  >0061    ; |
A21    EQU  >0062    ; |
B11    EQU  >0063    ; |
B21    EQU  >0064    ;  > Coeficientes de la función de transferencia
A02    EQU  >0065    ;  > representan los valores de los multiplicadores
A12    EQU  >0066    ; |
A22    EQU  >0067    ; |
B12    EQU  >0068    ; |
B22    EQU  >0069    ;/
MODE   EQU  >006A
CLOCK  EQU  >006B
MASK1  EQU  >006C
MASK2  EQU  >006D
YN     EQU  >006E    ; Muestras de salida del filtro
XN     EQU  >006F    ; Muestras de entrada del filtro
ONE    EQU  >0070    ; Constante de redondeo
N      EQU  >0071    ;\
N1     EQU  >0072    ; |
N2     EQU  >0073    ; | Variables internas de la estructura
N3     EQU  >0074    ; |
N4     EQU  >0075    ; |
N5     EQU  >0076    ;/
*
* INICIO DEL PROGRAMA
*
NXTP   IN   XN,PA1   ;<PA1> --> XN
       LAC XN,15     ;<XN> --> ACC
       LT  N1        ;<N1> --> T
       MPY B11       ;<B11*N1> --> P
       LTA N2        ;<N2> AL --> T, <B11*N1> --> ACC
       MPY B21       ;<B21*N2> --> P
       APAC          ;<B21*N2+B11*N1> --> ACC
       SACH N,1      ;<B21*N2+B11*N1> --> N
       ZAC           ;0 --> ACC
       MPY A21       ;<A21*N2> --> P
       LTD N1        ;<N1> --> T , <N1> --> N2
*                    ;<A21*N2> --> ACC
       MPY A11       ;<A11*N1>  --> P
       LTD N         ;<N> --> T , <N> --> N1
*                    ;<A11*N1+A21*N2> --> ACC
       MPY A01       ;<A01*N> --> P
       LTA N4        ;<N4> --> T
```

```
*                   ;<A01*N+A11*N1+A21*N2> --> ACC
      MPY B12       ;<B12*N4> --> P
      LTA N5        ;<N5> --> T
*                   ;<B12*N4+A01*N+A11*N1+A21*N2> --> ACC
      MPY B22       ;<B22*N5> --> P
      APAC          ;<B22*N5+B21*N4+A01*N+A11*N1+A21*N2> --> ACC
      SACH N3,1     ;<B22*N5+B21*N4+A01*N+A11*N1+A21*N2> --> N3
      ZAC           ;0 --> ACC
      MPY A22       ;<A22*N5> --> P
      LTD N4        ;<N4> --> T <N4> --> N5
*                    ;<A22*N5> --> ACC
      MPY A12       ;<A12*N4> --> P
      LTD N3        ;<N3> --> T , <N3> --> N4
*                   ;<A12*N4+A22*N5> --> ACC
      MPY A02       ;<N3*A02> --> P
      APAC          ;<N3*A02+A12*N4+A22*N5> --> ACC
      SACH YN,1     ;<N3*A02+A12*N4+A22*N5> --> YN
      OUT YN,PA2    ;<YN> --> PA2
      B NXTP        ;El programa se repite desde la etiqueta NXTP
*
* CONSTANTES DEL FILTRO EN HEXADECIMAL
*
COEF   DATA >1F05,>2B75,>1EFD,>394D,>D889,>62F1,>26DB
       DATA >62ED,>FEF7,>90EE,>000A,>0200,>7FFF,>8000
       END
```

3.3.2 Estructura directa del filtro IIR de 4o orden

En el caso del filtro directo, se realiza la función de transferencia 3.1 y la estructura se muestra en la figura 3.18.

Los coeficientes del filtro a_0 - a_4 y b_1 -b_4 se expresan en la forma hexadecimal. Si marcamos las variables internas para la realización del retardo como N_1 - N_4, el programa en ensamblador de TMS320C25 es el siguiente:

```
*------- ESTRUCTURA DIRECTA DE 4o ORDEN ----------*
*************************************************
*                    -1       -2     -3     -4    *
*           A0 + A1*z   + A2*z  +A3*z  +A4*z       *
* H(z)  =  ----------------------------------      *
*                    -1       -2     -3     -4    *
*            1 - B1*z   - B2*z  -B3*z  -B4*z       *
*************************************************
*
       AORG >0000
RESET  B    INIT
       AORG >0020
*
```

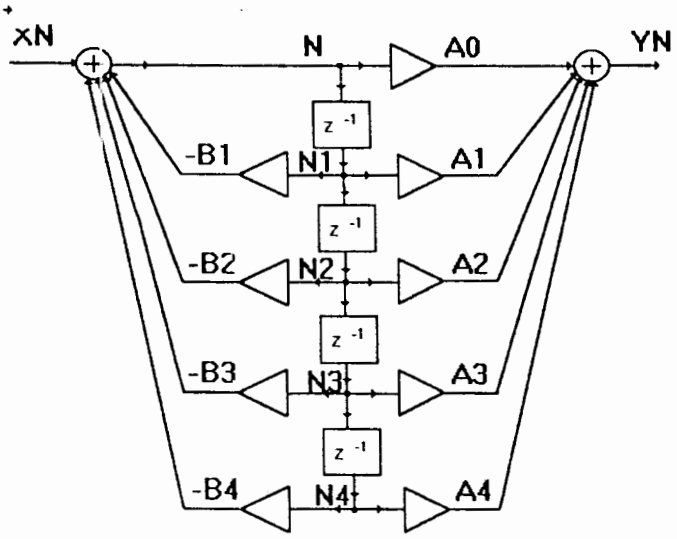


Figura 3.18. Estructura directa canónica del filtro digital de 4o orden

```
* INICIALIZACION DEL MICROCONTROLADOR
*
INIT    SOVM              ;Trabaja con modo de saturación
        LDPK 0            ;Trabaja con la página de memoria 0
        ZAC               ;0 --> ACC
        LARP AR2          ;Se activa registro auxiliar AR2
        LRLK AR2,>0060    ;Se marca inicio del bloque B2
        RPTK 31           ;La instrucción que sigue se repite 32 veces
        SACL *+           ;Se guardan los ceros en la memoria del bloque B2
        LRLK AR2,>0060    ;Se marca inicio del bloque B2
        RPTK 11           ;La instrucción que sigue se repite 13 veces
        BLKP COEF,*+      ;Los coeficientes del bloque marcados  COEF
*                         ;se transportan al bloque de memoria  B2
        LACK 1            ;Se guarda 1 en el acumulador
        SACL ONE          ;Un bit se deposita en la variable  ONE

B1      EQU  >0060        ;\
B2      EQU  >0061        ; |
B3      EQU  >0062        ; |
B4      EQU  >0063        ; |
A0      EQU  >0064        ;  > Coeficientes del filtro
A1      EQU  >0065        ; |
A2      EQU  >0066        ; |
A3      EQU  >0067        ; |
A4      EQU  >0068        ;/
MODE    EQU  >0069
MASK1   EQU  >006B
MASK2   EQU  >006C
YN      EQU  >006D        ;Muestras de entrada
```

```
XN      EQU  >006E        ;Muestras de salida
ONE     EQU  >006F        ;Bit de redondeo
N       EQU  >0070        ;\
N1      EQU  >0071        ; |
N2      EQU  >0072        ;  > Variables del filtro interno
N3      EQU  >0073        ; |
N4      EQU  >0074        ;/
*
* INICIO DEL PROGRAMA
NXTP    IN   XN,PA1       ;<PA1> --> XN
        LAC XN            ;<XN> --> ACC
        LT N1             ;<N1> --> T
        MPY B1            ;<B1*N1> --> P
        LTA N2            ;<N2> --> T , <B1*N1> --> ACC
        MPY B2            ;<B2*N2> --> P
        MPYA B2           ;<B2*N2+B1*N1>-->ACC , <B2*N2>-->P
        LTA N3            ;<N3> --> T , <B2*N2+B1*N1> --> ACC
        MPY B3            ;<B3*N3> --> P
        LTA N4            ;<N4> --> T
*                         ;<B3*N3+B2*N2+B1*N1> --> ACC
        MPY B4            ;<B4*N4> --> P
        APAC              ;<B4*N4+B3*N3+B2*N2+B1*N1> --> ACC
        SACH N,1          ;<B4*N4+B3*N3+B2*N2+B1*N1> --> N
        ZAC               ;0 --> ACC
        MPY A4            ;<N4*A4> --> P
        LTD N3            ;<N3> --> T  , <N3> --> N4
*                         ;<N4*A4> --> ACCH
        MPY A3            ;<A3*N3> --> P
        LTD N2            ;<N2> --> T  N<2> --> N3
*                         ;<A3*N3+A4*N4> --> ACC
        MPY A2            ;<N2*A2> --> P
        LTD N1            ;<N1> --> T  ,  <N1> --> N2
*                         ;<A2*N2+A3*N3+A4*N4> --> ACC
        MPY A1            ;<A1*N1> --> P
        LTD N             ;<N> --> T y <N> --> N1
*                         ;<A1*N1+A2*N2+A3*N3+A4*N4> --> ACC
        MPY A0            ;<A0*N> --> P
        APAC              ;<A0*N+A1*N1+A2*N2+A3*N3+A4*N4> --> ACC
        SACH YN,1         ;<A0*N+A1*N1+A2*N2+A3*N3+A4*N4> --> YN
        OUT YN,PA2        ;<YN> --> PA2
        B NXTP            ;Se repite el programa desde NXTP
*
* CONSTANTES DEL FILTRO
*
COEF DATA >3845,>B4F8,>3167,>DDC1,>17FA,>2B02,>3D1F
     DATA >2AFF,>17F3,>000A,>0200,>7FFF
     END
```

3.3.3 Estructura paralela del filtro IIR de 4o orden

En el caso de la estructura paralela es necesario dividir la función de transferencia H(z) en dos funciones de transferencia parciales $H_1(z)$ y $H_2(z)$ que cumplan $H(z) = H_1(z) + H_2(z)$ (ecuación 3.1), dichas funciones son de segundo orden. Si arreglamos la función de transferencia (3.1), obtenemos

$$H(z) = 0.699474 + \frac{-0.4428446 + 0.7435591z^{-1}}{1 - 0.447687z^{-1} - 0.308310z^{-2}} + \frac{-0.0693031 - 0.1274959z^{-1}}{1 + 0.00808z^{-1} + 0.867723z^{-2}}$$

La estructura para la cual escribimos el programa en ensamblador para el microcontrolador TMS320C25 se presenta en la figura 3.19.

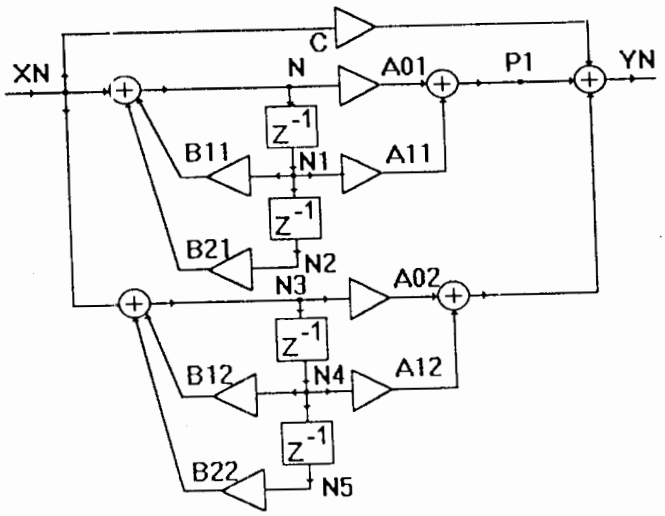


Figura 3.19. Estructura paralela del filtro digital de 4o orden

Transferimos los coeficientes a la forma hexadecimal. Marcamos los nodos internos del filtro N hasta $N5$ como se aprecia en la figura 3.19. El programa para la realización del filtro digital en la forma paralela es el siguiente:

```
*--------------FILTRO IIR PARALELO DE 4o ORDEN------------ *
*************************************************************
*                                -1                    -1   *
*                   A01 + A11*z              A02 + A12*z     *
* H(z) = C +  ------------------- + --------------------- *
*                        -1      -2              -1       -2 *
*               1 +B11*z  +B21*z       1 - B12*z   + B22*z  *
*************************************************************
*
        AORG >0000
RESET   B    INI
        AORG >0020
*
* INICIALIZACION DEL MICROCONTROLADOR
```

```
*
INI SOVM              ;Trabaja en modo de sobreflujo
    LDPK 0            ;Trabaja con la página 0
    ZAC               ;0 --> ACCH
    LARP AR2          ;Trabaja con registro auxiliar AR2
    LRLK AR2,>0060    ;Al AR2 se guarda la dirección >0060
    RPTK 31           ;La instrucción que sigue se repite 32 veces
    SACL *+           ;Se guardan ceros en la memoria del bloque B2
    LRLK AR2,>0060    ;Se marca inicio del bloque  B2
    RPTK 8            ;La instrucción que sigue se repite 13 veces
    BLKP COEF,*+      ;Los coeficientes del archivo COEF se guardan al B2
    LACK 1            ;1 se guarda en el acumulador
    SACL ONE          ;En la variable ONE se guarda un bit de redondeo
*
* DECLARACION DE LAS VARIABLES
*
A01   EQU  >0060   ;\
A11   EQU  >0061   ; |
B11   EQU  >0062   ; |
B21   EQU  >0063   ; |
A02   EQU  >0064   ;  >  Constantes del filtro
A12   EQU  >0065   ; |
B12   EQU  >0066   ; |
B22   EQU  >0067   ; |
C     EQU  >0068   ;/
XN    EQU  >0069   ;     En variable  XN están las muestras de entrada
YN    EQU  >006A   ;     En variable  YN están las muestras de salida
N     EQU  >006B   ;\
N1    EQU  >006C   ; |
N2    EQU  >006D   ; |
N3    EQU  >006E   ;  >  Las variables internas del filtro
N4    EQU  >006F   ; |
N5    EQU  >0070   ; |
P1    EQU  >0071   ;/
ONE   EQU  >0072   ;      Variable para redondeo
*
* COMIENZO DEL PROGRAMA
*
NXTP  IN   XN,PA1   ;<PA1> --> XN
      LAC  XN       ;<XN> --> ACC, Nueva muestra en ACC
      SACL XN       ;<XN> --> T
      LAC XN,15     ;<XN> -->ACC
      LT  N2        ;<N2> --> T
      MPY B21       ;<N2*B21> --> P
      LTD N1        ;<N1> --> T , <N1> --> N2
*                   ;<XN+B21*N2> --> ACC
      MPY B11       ;<B11*N1> --> P
```

```
      APAC          ;<B21*N2+XN+B11*N1> --> ACC
      SACH N,1      ;<B21*N2+XN+B11*N1> --> N
      ZAC           ;0 --> ACC
      MPY A11       ;<A11*N1> --> P
      LTD N         ;<N> --> T , <N> --> N1
*                   ;<A11*N1> --> ACC
      MPY A01       ;<A01*N> --> P
      APAC          ;<A01*N+A11*N1> --> ACC
      SACH P1,1     ;<A01*N+A11*N1> --> P1
*     ZAC           ;0 --> ACCH
*AQUI TERMINA EL PROGRAMA PARA LA PRIMERA SECCION Y EMPIEZA
*EL PROGRAMA PARA LA SEGUNDA SECCION
      LAC XN,15     ;<XN> --> ACC
      LT  N5        ;<N5> --> T
      MPY B22       ;<B22*N5> --> P
      LTD N4        ;<N4> --> T , <N4> --> N5
*                   ;<B22*N5+XN> --> ACC
      MPY B12       ;<B12*N4> --> P
      APAC          ;<B12*N4+B22*N5+XN> --> ACC
      SACH N3,1     ;<B22*N5+B12*N4+XN> --> N3
      LAC P1,15     ;<P1>  --> ACC
      MPY A12       ;<A12*N4> --> P
      LTD N3        ;<N3> --> T, <N3> --> N4
*                   ;<A12*N4> --> ACC
      MPY A02       ;<A02*N3> --> P
      LTA C         ;<C> --> T, <A02*N3+A12*N4> --> ACC
      MPY XN        ;<C*XN> --> P
      APAC          ;<C*XN+A02*N3+A12*N4> --> ACC
      SACH YN,1     ;<A02*N3+A12*N4+A22*N5> --> YN
*     ZAC           ;0 --> ACCH
      OUT YN,PA2    ;<YN> -->  PA2
      B NXTP        ;Salto a la etiqueta NXTP para leer otra muestra

*
* LAS CONSTANTES DEL FILTRO
*
COEF   DATA >C750,>5F2C,>394D,>D889,>F721,>EFAE,>FEF7
       DATA >90EE,>5988
       END
```

3.3.4 Realización de dos filtros de estado en cascada

El filtro digital se puede diseñar como cascada de dos filtros de estado de segundo orden. Por eso dividimos la función de transferencia (3.1) en el producto de dos funciones de transferencia de segundo orden y obtenemos

$$H_1(z) = \frac{0.242342 + 0.3395217z^{-1} + 0.242117z^{-2}}{1 - 0.447687z^{-1} + 0.308310z^{-2}}$$

$$H_2(z) = \frac{0.77299 + 0.303581z^{-1} + 0.772887z^{-2}}{1 + 0.00808z^{-1} + 0.867723z^{-2}}$$

En la figura 3.20 se muestra la estructura del filtro como cascada de dos filtros de estado de segundo orden.

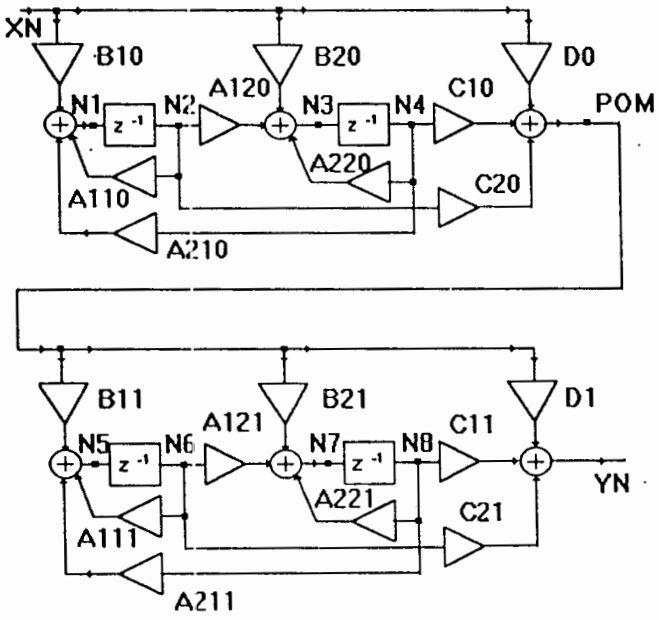


Figura 3.20. Filtro de estado de 4o orden

Primeramente tenemos que calcular los coeficientes de la función de transferencia $H_1(z)$ a $H_2(z)$ usando las ecuaciones:

$$\begin{aligned}
&d_0 = a_0 = 0,242342 && > 1F05\\
&a_{110} = a_{220} = -B_1/2 = 0.2238435 && > 1CA6\\
&a_{120} = \sqrt{|B_2 - B_1^2/4|} = 0.5081379 && > 410A\\
&a_{210} = -0.5081379 && > BEF6\\
&\alpha_{10} = A_1 - A_0.B_1 = 0.4480144\\
&\beta_{10} = A_2 - A_0.B_2 = 0.1674005\\
&\epsilon = (\beta_1 + \alpha_1.a_{220})^2/4a_{120}^2 = 0.0693788\\
&c_{20} = \sqrt{(\sqrt{\alpha_1^2 + 4.\epsilon_1} - \alpha_1)/2} = 0.3489476 && > 2CAA\\
&b_{10} = c_{10} = -\sqrt{\alpha_1 + c_2^2} = -0.754837 && > 9F62\\
&b_{20} = -c_{20} = -0.3489476 && > D356
\end{aligned}$$

Para la función de transferencia $H_2(z)$ calculamos los coeficientes del segundo filtro de estado

$$\begin{aligned}
&d_1 = 0.77299 => 62F1 && a_{111} = -0.00404 => FF76 && a_{221} = 0.00404 => FF76\\
&a_{121} = 0.931507 => 773B && a_{211} = -0.931507 => 88C5 && c_{11} = -0.5540037 => BB17\\
&b_{11} = -0.5540037 => BB17 && c_{21} = 0.0979025 => 0C88 && b_{21} = -0.0979025 => F378
\end{aligned}$$

Si marcamos como XN las muestras de entrada, YN como la variable para las muestras de salida y las variables internas del filtro N_1 hasta N_8, el programa en ensamblador para el simulador toma la forma

```
* ----------> FILTRO DE ESTADO DE 4o ORDEN <----------  *
*************************************************************
*                      -1       -2           -1       -2  *
*           A00 + A10z   + A20z    A01 +A11z   + A21z     *
* H(z)  =  ---------------------- x ---------------------- *
*                     -1       -2           -1       -2   *
*            1 - B10z   - B20z     1 - B11z   + B21z      *
*************************************************************
*
       AORG >0000
RESET  B    INIT
       AORG >0020
*
* INICIALIZACION DEL MICROCONTROLADOR
*
INIT   SOVM              ;Trabaja en modo sobreflujo
       LDPK 0            ;Trabaja con la página 0 de memoria
       ZAC               ;Se guardan ceros en acumulador
       LARP AR2          ;Se activa AR2
       LRLK AR2,>0060    ;Principio del bloque B2
       RPTK 31           ;La instrucción que sigue se repite 32 veces
       SACL *+           ;En la memoria de datos B2 se guardan ceros
       LRLK AR2,>0060    ;Principio del bloque  B2
       RPTK 18           ;La instrucción que sigue se repite 15 veces
       BLKP COEF,*+      ;Los coeficientes desde el archivo COEF se guardan
                         ;en el bloque B2
       LACK 1            ;En el acumulador se guarda un bit para redondeo
       SACL ONE          ;Un bit para redondear se guarda en la variable ONE
*
* DECLARACION DE LAS VARIABLES
*
A110    EQU  >0060       ;\
A220    EQU  >0061       ; |
A120    EQU  >0062       ; |
A210    EQU  >0063       ; |
C10     EQU  >0064       ; |> Coeficientes del filtro
C20     EQU  >0065       ; |> de la primera sección
B10     EQU  >0066       ; |
B20     EQU  >0067       ; |
D0      EQU  >0068       ; |
ONE     EQU  >0069       ;    Variable para redondear
A111    EQU  >006A       ;\
A221    EQU  >006B       ; |
```

```
A121    EQU  >006C        ; |
A211    EQU  >006D        ; |> Coeficientes del filtro
C11     EQU  >006E        ; |> de la segunda sección
C21     EQU  >006F        ; |
B11     EQU  >0070        ; |
B21     EQU  >0071        ; |
D1      EQU  >0072        ; /
YN      EQU  >0073        ;    Muestras de salida
XN      EQU  >0074        ;    Muestras de entrada
N1      EQU  >0075        ;\
N2      EQU  >0076        ; |
N3      EQU  >0077        ; |
N4      EQU  >0078        ; |> Variables externas
N5      EQU  >0079        ; |> del filtro
N6      EQU  >007A        ; |
N7      EQU  >007B        ; |
N8      EQU  >007C        ; |
POM     EQU  >007D        ;/
*
* INICIO DEL PROGRAMA
*
ZACAT  IN   XN,PA1       ;<PA1> --> XN
       LT XN             ;<XN> --> T
       MPY B10           ;<XN*B10> --> P
       LTA N4            ;<N4> --> T y <XN*B10> --> ACC
       MPY A120          ;<A120*N4> --> P
       LTA N2            ;<N2> --> T
*                        ;<A120*N4+XN*B10> --> ACC
       MPY A110          ;<A110*N2> --> P
       APAC              ;<A110*N2+A120*N4+XN*B10> --> ACC
       SACH N1,1         ;<A210*N2+A120*N4+XN*B10> --> N1
       ZAC               ;0 --> ACC
       LAC  ONE,14       ;Bit de redondeo --> ACC
       MPY  A210         ;<N2*A210> --> P
       LTA  XN           ;<XN> --> T , <N2*A210> --> ACC
       MPY  B20          ;<XN*B20> --> P
       LTA  N4           ;<N4> --> T
*                        ;<XN*B20+N2*A210> --> ACCH
       MPY  A220         ;<A220*N4> --> P
       APAC              ;<A220*N4+XN*B20+N2*A210> --> ACC
       SACH N3,1         ;<A220*N4+XN*B20+N2*A210> --> N3
       ZAC               ;0 --> ACCH
       LAC ONE,14        ;Bit de redondeo --> ACC
       LT N4             ;<N4> --> T
       MPY C20           ;<N4*C20> --> P
       LTD N3            ;<N3> --> T , <N3> --> N4
*                        ;<N4*C20> --> ACCH
```

```
        LT XN               ;<XN> --> T
        MPY D0              ;<D0*XN> --> P
        LTA N2              ;<N2> --> T y <D0*XN+N4*C20> --> ACC
        MPY C10             ;<C10*N2> --> P
        LTD N1              ;<N1> --> T , <N1> --> N2
*                           ;<C10*N2+D0*XN+N4*C20> --> ACC
        SACH POM,1          ;<C10*N2+D0*XN+C20*N4> --> YN
        ZAC                 ;0 --> ACCH
        LAC ONE,14          ;Bit de redondeo --> ACC
        LT   POM            ;<POM> --> T
        MPY  B11            ;<B11*POM> --> P
        LTA  N8             ;<N8> --> T, <B11*POM> --> ACCH
        MPY  A121           ;<A121*N8> --> P
        LTA  N6             ;<N6> --> T , <B11*POM+A121*N8> --> ACC
        MPY  A111           ;<A111*N6> --> P
        APAC                ;<A111*N6+B11*POM+A121*N8> --> ACC
        SACH N5,1           ;<A111*N6+B11*POM+A121*N8> --> N5
        ZAC                 ;0 --> ACC
        LAC  ONE,14         ;Bit de redondeo --> ACC
        MPY  A211           ;<A211*N6> --> P
        LTA  POM            ;<POM> --> T , <A211*N6> --> ACC
        MPY  B21            ;<POM*B21> --> P
        LTA  N8             ;<N8> --> T , <POM*B21+A211*N6> --> ACC
        MPY  A221           ;<A221*N8> --> P
        APAC                ;<A221*N8+POM*B21+A211*N6> --> ACC
        SACH N7,1           ;<A221*N8+POM*B21+A211*N6> --> N7
        ZAC                 ;0 --> ACCH
        LAC  ONE,14         ;Bit de redondeo --> ACC
        LT   N8             ;<N8> --> T
        MPY  C21            ;<C21*N8> --> P
        LTD  N7             ;<N7> --> T , <N7> --> N8
*                           ;<C21*N8> --> ACC
        LT   POM            ;<POM> --> T
        MPY  D1             ;<D1*POM> --> P
        LTA  N6             ;<N6> --> T, <D1*POM+C21*N8> --> ACC
        MPY  C11            ;<C11*N6> --> P
        LTD  N5             ;<N5> --> T, <N5> --> N6
*                           ;<C11*N6+D1*POM+C21*N8> --> ACC
        SACH YN,1           ;El resultado (la respuesta) se guarda en YN
        OUT  YN,PA2         ;El resultado se guarda en el archivo PA2
        ZAC                 ;0 --> ACCH
        LAC ONE,14          ;Bit de redondeo se guarda en ACC
        B    ZACAT          ;Se salta a la etiqueta ZACAT
*
* CONSTANTES DEL FILTRO
*
COEF DATA >1CA6,>1CA6,>410A,>BEF6,>9F62,>2CAA
```

```
DATA >9F62,>D356,>1F05,>0001,>FF7C,>FF7C
DATA >773B,>88C5,>BB17,>0C88,>BB17,>F378,>62F1
END
```

3.3.5 Diseño del filtro digital en forma de cruz

Se puede realizar y diseñar con un DSP TMS320C25 el filtro digital en forma de cruz (figura 3.21), si se conoce la función de transferencia H(z).

$$H(z) = \frac{0.242 + 0.339z^{-1} + 0.242z^{-2}}{1 - 0.447z^{-1} + 0.308z^{-2}} = \frac{B_0 + B_1z^{-1} + B_2z^{-2}}{1 + A_1z^{-1} + A_2z^{-2}} \tag{3.2}$$

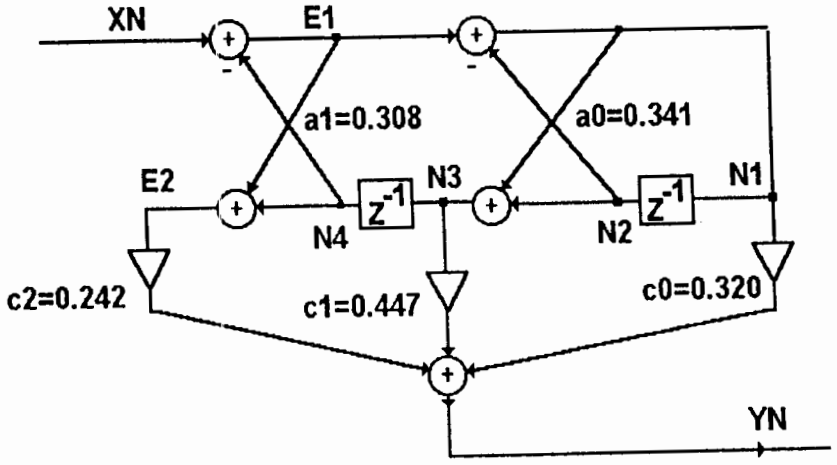


Figura 3.21. El filtro en cruz de segundo orden

Los coeficientes del filtro cruz de segundo orden se calculan mediante las ecuaciones:

$$a_1 = A_2 \quad c_2 = B_2$$

$$a_0 = \frac{A_1}{1 + A_2}$$

$$c_1 = B_1 - B_2A_1$$

$$c_0 = B_0 - B_2A_2 + \frac{B_2A_1^2 - B_1A_1}{1 + A_2}$$

Sustituyendo por $A_1 = -0.447$, $A_2 = 0.308$, $B_0 = B_2 = 0.242$ y $B_1 = 0.339$, obtenemos

$$a_0 = -0.3417431 \quad a_1 = 0.308 \quad c_0 = 0.3208 \quad c_1 = 0.447174 \quad c_2 = 0.242$$

El filtro digital en forma de cruz se observa en la figura 3.21, de su estructura podemos escribir las siguientes ecuaciones:

$$E_1 = XN - N4.a_1$$
$$N_1 = E1 - N2.a_0$$
$$N3 = N1.a_0 + N4$$
$$E2 = E1.a_0 + N4$$
$$YN = E2.c_2 + N3.c_1 + N1.c_0$$

El programa para el filtro en ensamblador es el siguiente:

```
*    FILTRO IIR DE 2o ORDEN                *
*******************************************
*                                          *
*                     -1       -2          *
*           A0 + A1*z   + A2*z             *
* H(z)  =  ---------------------------     *
*                     -1       -2          *
*            1 - B1*z   - B2*z             *
*******************************************
*
       AORG >0000
RESET  B    INIT
*
       AORG >0020
*
* INICIO DEL MICROCONTROLADOR
*
INIT   SOVM              ;Trabaja en saturación
       LDPK 0            ;Trabaja con bandera cero
       ZAC               ;Anulación del acumulador
       LARP AR2          ;Actualiza el registro auxilar AR2
       LRLK AR2,>0060    ;Inicializa bloque B2
       RPTK 31           ;Repite la instrucción que sigue 32 veces
       SACL *+           ;Anulación del bloque B2
       LRLK AR2,>0060    ;Inicializa el bloque B2 desde la dirección 0060
       RPTK 11           ;Repite la instrucción que sigue 5 veces
       BLKP COEF,*+      ;Guarda los coeficientes del archivo COEF al
                         ;bloque B2
*
* DECLARACION DE LAS VARIABLES
*
C2     EQU  >0060        ;\
C1     EQU  >0061        ; \
C0     EQU  >0062        ;  > Coeficientes del filtro
A1     EQU  >0063        ; /
A0     EQU  >0064        ;/
E1     EQU  >0065        ;    Muestras de entrada
```

```
E2      EQU  >0066          ;    Muestras de salida
N1      EQU  >0067          ;    Bit para redondear
N2      EQU  >0068          ;\
N3      EQU  >0069          ; >  Variables internas del filtro
N4      EQU  >006A          ;/
XN      EQU  >006B
YN      EQU  >006C
*
* EMPIEZA EL PROGRAMA EN ENSAMBLADOR
*
ZACAT   IN   XN,PA1     ;Entrada de la muestra
        LT   A1         ;A1 --> TR
        MPY  N4         ;<A1*N4> -->PR
        LTA  A0         ;<A0>-->TR <A1*N4>-->ACC
        NEG             ;<-A1*N4>-->ACC
        ADD  XN,15      ;<-A1*N4 + XN>-->ACC
        SACH E1,1       ;<-A1*N4+XN>-->E1
*---------------------------------------
        OUT E1,PA3
        ZAC             ;0-->ACC
        MPY  N2         ;<N2*A0>-->PR
        APAC            ;<N2*A0>-->ACC
        NEG             ;<-N2*A0>-->ACC
        ADD  E1,15      ;<E1-N2*A0>-->ACC
        SACH N1,1       ;<E1-N2*A01>-->N1
*---------------------------------------
        OUT N1,PA3
        ZAC             ;0-->ACC
        MPY  N1         ;<A0*E1>-->PR
        APAC            ;<N1*A0>-->ACC
        ADD N2,15       ;<A0*N1+N2>-->ACC
        SACH N3,1       ;<A0*N1+N2>-->N3
*---------------------------------------
        OUT N3,PA3
        ZAC             ;0-->ACC
        LT A1           ;<A1>-->TR
        MPY E1          ;<E1*A1>-->PR
        APAC            ;<E1*A1>-->ACC
        ADD N4,15       ;<N4+E1*A1>-->ACC
        SACH E2,1       ;<N4+E1*A1>-->E2
*---------------------------------------
        OUT E2,PA3
        ZAC             ;0-->ACC
        LT C2           ;<C2>-->TR
        MPY E2          ;<E2*C2>-->PR
        LT C1           ;<C1>-->TR
        MPYA N3         ;<C1*N3>-->PR <C2*E2>-->ACC
```

```
       LT C0          ;<C0>-->TR <C2*E2>-->ACC
       MPYA N1         ;<C0*N1>-->PR <C1*N3+C2*E2>-->ACCH
       APAC            ;<C2*E2+C1*N3+C0*N1>-->ACC
       SACH YN,1       ;<C2*E2+C1*N3+C0*N1>-->YN
*-----------------------------------------
       OUT YN,PA2
       LTD N1          ;<N1>-->N2
       LTD N3          ;<N3>-->N4
       ZAC             ;0-->ACC
       B    ZACAT
*
* DEFINICION DE LAS CONSTANTES
*
COEF   DATA >1EF9,>393C,>28FE,>276C,>D442
       END
```

En la figura 3.22 se muestran las respuestas del filtro en el dominio del tiempo y la frecuencia - espectro. La respuesta en el dominio del tiempo se obtuvo mediante el simulador TMS320C25 y el programa VIEW. El espectro se obtuvo mediante la transformada de Fourier rápida, utilizando el programa FR. En la figura 3.22 obtenemos el espectro utilizando sólo 20 muestras recibidas mediante el simulador de TMS320C25.

3.4 Simulador del PSD TMS320C25

El programa para la simulación se compone de los siguientes archivos ejecutables:

$$XAS25.EXE$$

$$SIM25.EXE$$

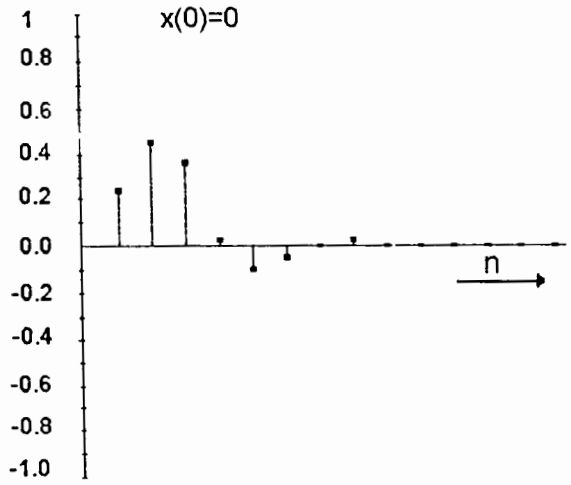
$$LINK25.EXE$$

Antes de simular un filtro digital, se tiene que escribir con un editor de texto en ASCII el archivo input.dat que contiene las muestras de la señal en la entrada. Esa señal puede ser, por ejemplo, un impulso de Dirac o una secuencia de impulsos unitarios. Los números deben ser escritos en una columna en forma hexadecimal.

Después, con cualquier editor de texto, se escribe el programa en ensamblador que realizará el circuito digital. Este programa se puede llamar por ejemplo: IIR.ASM. Con la instrucción XAS25 IIR.ASM creamos los programas con la extensión .LST y .MPO. Si el programa en ensamblador está escrito sin errores, podemos ver en su directorio los programas IIR.LST y IIR.MPO, y utilizar el programa IIR.MPO para la simulación de nuestro filtro propuesto.

La simulación se realiza con el comando SIM25. Después de teclear enter y el número 1, obtenemos en la pantalla la estructura del simulador mostrada en la figura 3.23.

Las instrucciones que se ejecutan y que se van a realizar las podemos observar en la parte izquierda superior de la pantalla.



1	0.2419
2	0.4471
3	0.3672
4	0.0264
5	-0.1013
6	-0.0535
7	0.007263
8	0.01965
9	0.0065
10	-0.003235
11	-0.00351
12	-0.0006409
13	0.0007629
14	0.0004578
15	-9.155c-05
16	-0.0002441
17	-0.0001526
18	-3.052c-05
19	0
20	-3.052e-05

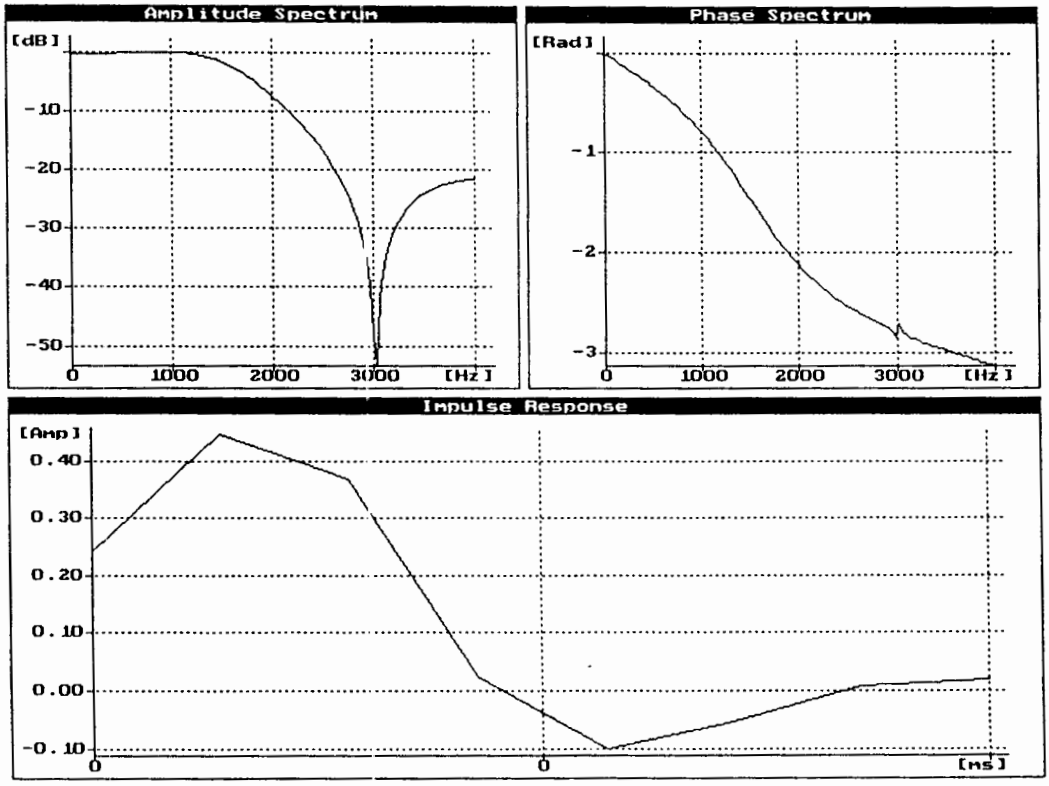


Figura 3.22. Respuesta del filtro al impulso unitario

```
PC :0000 ADD  >0200          +-------------C25 MC-----------------------
-1 :0000 ADD  >0200          ¦IFR :000000   ¦ST0:0604  ST1 :07F0  BIO:1
-2 :0000 ADD  >0200          ¦IMR :000000   ¦ARB:0     ARP :0      CNFD
-3 :0000 ADD  >0200          ¦-----MMRS------¦CRY:1     DP  :04     FO :0
--------------------------STACK----¦DRR :0000   ¦FSM:1     INTM:1      OV :0
=>AR0:  0000      ¦SK0: 0000 ¦DXR :0000   ¦OVM:0     PM  :0      SXM:1
 AR1:  0000       ¦SK1: 0000 ¦TIM :FFFF   ¦TC :0     TXM :0      XF :1
 AR2:  0000       ¦SK2: 0000 ¦PRD :FFFF   ¦OUTP:0000
 AR3:  0000       ¦SK3: 0000 ¦GREG:0000   ¦RPTC:  0  CLK:      0
 AR4:  0000       ¦SK4: 0000 ¦-----------------------------------------
 AR5:  0000       ¦SK5: 0000 ¦   TREG:    0000
 AR6:  0000       ¦SK6: 0000 ¦   PREG:  00000000
 AR7:  0000       ¦SK7: 0000 ¦   ACC :  00000000
-------------------------------------------------------------------------
 TH    : Trace help        UTLH  : Utilities help
 BH    : Breakpoint help
 EH    : Execution help
 MH    : Memory help
 RH    : Register/Flags help
 DH    : Display help
 IOH   : I/O help
 TICH  : Interrupt/Timing help
 STH   : Status register/pin help
Command:
```

Figura 3.23. Representación de algunas funciones del simulador

En el siguiente bloque, la flecha indica qué registro auxiliar está activo en un momento determinado.

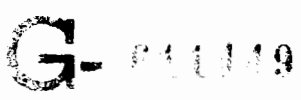
En la columna que se llama STACK se puede leer la dirección en que se van a ejecutar las instrucciones después de saltar a un subprograma.

En las registros TREG y PREG de la figura, abajo y a la derecha, se puede leer el número guardado en el registro T o P, y el ACC el resultado después de la multiplicación.

Si se teclea enter, en la pantalla podemos observar una columna con todas las instrucciones que se pueden usar y que indican cómo manejar el simulador. Si se teclea EH enter, se obtienen los comandos que están en la tabla 3.1.

Instrucción	*Descripción*
C	Sigue en la simulación.
EX	Hace la simulación del programa especificado.
L	Graba el programa con la extensión .TMP para la simulación.
LC	Graba el archivo con los coeficientes.
NB	Teclea el número de los comandos que ejecuta el simulador.
Q	Salta del programa de simulación.
R	Hace la simulación.
RS	Interrumpe la simulación.
SS	Simula el programa paso a paso.

Tabla 3.1. Instrucciones para la simulación



Primeramente se tiene que cargar el programa para la simulación con la extensión .MPO. Eso se cumple con la instrucción L FIR.MPO. Las instrucciones que siguen son las de salida

y entrada. Si se teclea la instrucción IOH se puede ver en la pantalla las instrucciones SI y SO. Después de la instrucción SI enter, la computadora pregunta el número del archivo donde está guardada la señal de entrada. No se puede elegir cualquier número de puerto, sólo el número que está en la instrucción IN XN PA1. En nuestro caso es el número 1. Después, la computadora pregunta el nombre del archivo donde están las muestras de entrada, para lo cual la instrucción es INPUT.DAT enter. En INPUT.DAT tenemos un impulso unitario.

Si se ejecuta la instrucción SO enter el programa pregunta en qué número de puerto debe guardar la respuesta al impulso Se tiene que escribir el número que está en el programa en ensamblador junto con la instrucción OUT YN,PA2. En nuestro caso es el número 2. Después, el programa quiere conocer el nombre del archivo en dónde guardar la respuesta al impulso. Se teclea entonces OUT.DAT enter.

Otra instrucción es NB y el número que dice al simulador cuántas instrucciones debe ejecutar antes de parar la simulación. Se elige el número 5000 y tecleamos enter. El programa entonces ejecuta 5000 instrucciones. Para correr la simulación tecleamos R enter. Si se quiere simular este ejemplo paso a paso se necesita teclear la instrucción SS enter en lugar de R enter. De esta manera se puede ver en los registros los resultados parciales y corregir el programa si se calculó mal.

El resultado lo vemos en el archivo OUT.DAT. Los números en el archivo OUT.DAT son organizados en columna en la forma hexadecimal.

La instrucción MH Memory Help muestra los comandos que podemos utilizar para ver los números localizados en la memoria de datos. El comando UTLII ofrece las instrucciones que sirven para crear un archivo con la señal senoidal en la forma discreta.

Capítulo 4

Microprocesador TMS320C30

4.1 Introducción

El microprocesador TMS320C30 forma parte de la tercera generación de la familia TMS320 de procesadores digitales de señales (DSP) desarrollados con tecnología 1μm CMOS, capaces de manejar operaciones con punto flotante de 32 bits. Se fabrica en un encapsulado de tipo PGA (*Pin Grid Array*) de 181 pins.

El ciclo de bus es de 60 ns, palabra de datos y programa de 32 bits, con bus de direcciones de 24 bits, lo que resulta en un espacio de memoria de 2^{24} = 16.7 megapalabras. Ejecuta instrucciones a una tasa de 33.3 MFLOPS [1] y 16.7 MIPS [2]. Funciona con un reloj externo de 66 MHz. Otra característica importante de un DSP con respecto a los demás microprocesadores es la multiplicación en paralelo y carga al acumulador, todo esto en un solo ciclo (instrucción tipo MAC, *Multiply and Accumulate*).

Este microprocesador posee un conjunto de registros, algunos de propósito general denominados archivos de registros, que son utilizados por el CPU para llevar el control de sus operaciones. Tiene un caché de 64 palabras de longitud empleado por la memoria de programa. Para realizar operaciones aritméticas, usa dos unidades aritméticas que trabajan en paralelo, cada una de las cuales posee un registro auxiliar asociado (ARAU0 y ARAU1). Los bloques de memoria internos son de acceso dual (pueden accesar dos operandos en un ciclo de máquina). Existe un canal destinado para DMA (acceso directo a memoria) que disminuye la carga al CPU, el cual soporta operaciones de entrada/salida concurrente.

En cuanto a los periféricos desarrollados dentro del circuito integrado, se consideran dos temporizadores y dos puertos seriales independientes. Debido a su arquitectura basada en registros, facilitan la implantación de compiladores de alto nivel, como es el caso del lenguaje C. El bloque de memoria ROM es de 4k-palabras y existen además dos bloques de memoria RAM de 1k. La unidad aritmética lógica y el multiplicador son de 40 bits para punto flotante y de 32 bits para números enteros. El CPU tiene un registro de corrimiento de hasta 32 bits, ocho registros o acumuladores de precisión extendida, así como dos generadores de direcciones con ocho registros auxiliares y dos registros auxiliares de las unidades aritméticas. Puede ejecutar instrucciones de dos o tres operandos. Posee dos banderas externas de propósito

[1] MFLOPS = millones de operaciones de punto flotante por segundo.

[2] MIPS = millones de instrucciones por segundo.

general y cuatro interrupciones externas. El controlador de DMA se utiliza para leer o escribir en la memoria sin que se interrumpa al CPU.

4.2 Arquitectura

La arquitectura interna del TMS320C30 se estructuró con la finalidad de poder ejecutar grandes volúmenes de operaciones aritméticas, utilizando tanto hardware como software. Posee una extensa memoria interna y alto grado de paralelismo en las operaciones, lográndose así una reducción significativa en el tiempo de ejecución de un bloque de instrucciones.

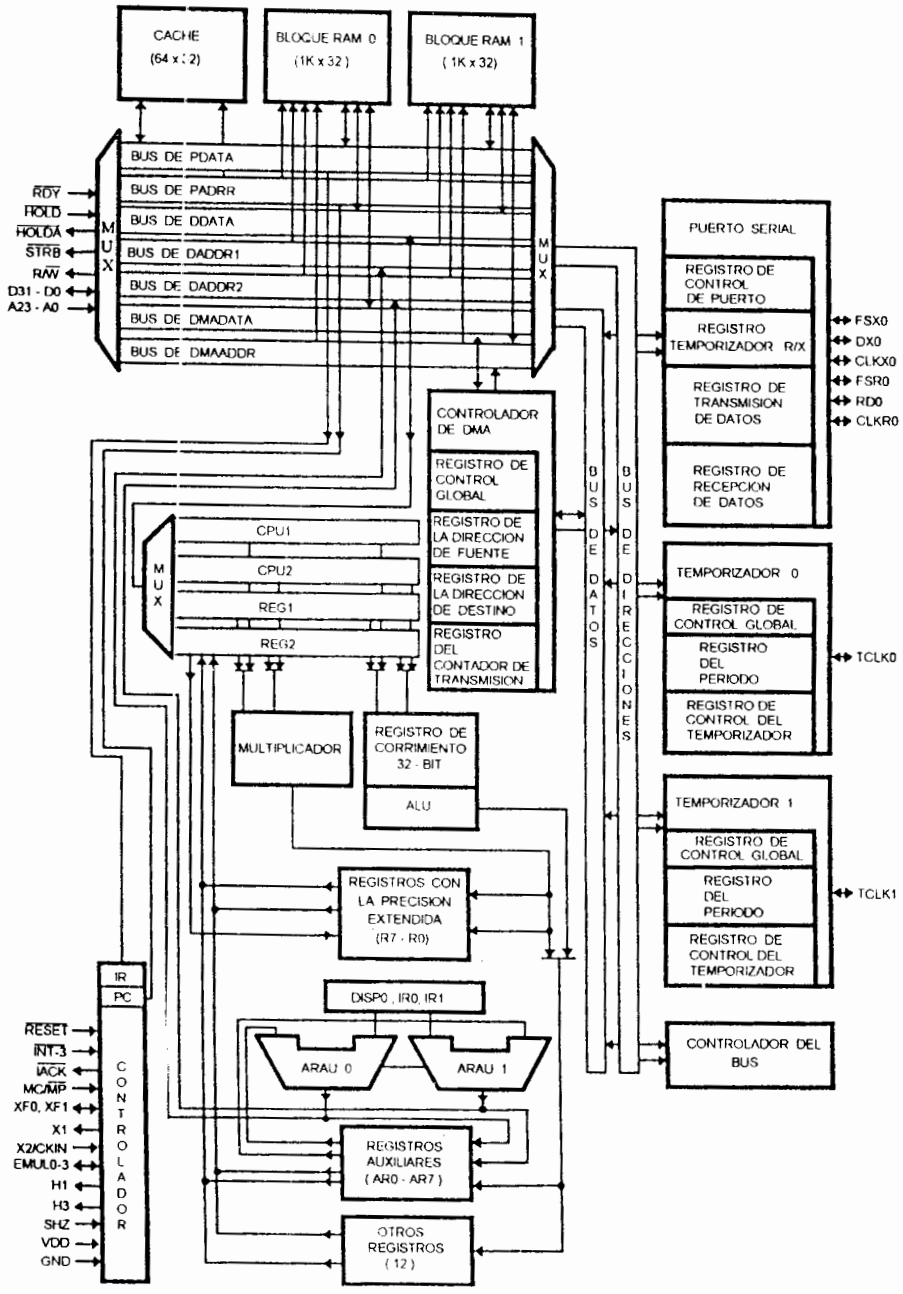


Figura 4.1. Arquitectura del microprocesador TMS320C30

4.3 Unidad central de proceso (CPU)

La administración de los recursos del CPU se realiza a través de registros agrupados en un archivo de registros que contiene los siguientes componentes:

1. Multiplicador de enteros y números de punto flotante.
2. Unidad aritmética lógica (ALU), la cual ejecuta operaciones de punto flotante, de enteros y operaciones lógicas.
3. Registro de corrimiento de hasta 32 bits.
4. Buses internos (CPU1/CPU2 y REG1/REG2).
5. Unidades aritméticas con registros auxiliares asociados (ARAU).
6. Conjunto de 28 registros con diversos propósitos (register file).

Multiplicador para enteros y punto flotante

Ejecuta instrucciones de un ciclo de máquina de duración de la siguiente manera:

- Multiplicación entera de 24 bits con resultado de 32 bits.
- Multiplicación de punto flotante de 32 bits con resultado de 40 bits.

Unidad aritmética lógica (ALU)

Aquí se ejecutan operaciones de duración de un ciclo de máquina para datos enteros y lógicos de 32 bits, y datos de punto flotante de 40 bits. La extensión de la palabra de datos resultado tiene la misma extensión que los operandos.

Registro de corrimiento de 32 bits

Es utilizado para realizar desplazamientos de hasta 32 bits hacia la izquierda o hacia la derecha en un solo ciclo.

Buses internos CPU1/CPU2 y REG1/REG2

La concepción separada de estos buses permite extraer dos operandos de memoria y dos operandos desde los registros para que se realicen multiplicaciones paralelas, así como sumas y restas de cuatro operandos en un solo ciclo de máquina.

Unidades aritméticas con registros auxiliares asociados ARAU0 y ARAU1

Es posible generar dos direcciones en un solo ciclo que opera en paralelo con el multiplicador y la ALU, esto permite usar modo de direccionamiento indexado, circular y de inversión de bit.

4.3.1 Archivo de registros del CPU

Éstos se pueden utilizar como registros de propósito general, aunque también cada uno de ellos realiza una función específica; por ejemplo, se tienen ocho registros de precisión extendida, ocho registros auxiliares, entre otros que se presentan a continuación.

Registros de precisión extendida R0-R7

Almacenan operandos y realizan operaciones con enteros de 32 bits y números de punto flotante de 40 bits. Se asume que para operaciones con enteros, usan los 32 bits menos significativos y en operaciones de punto flotante 40 bits.

Registros auxiliares AR0-AR7

Se accesan por el CPU y por la ARAU. Su función principal es la generación de direcciones de 24 bits, pero se pueden usar también como contadores de cuenta cerrada o como registros de propósito general.

Apuntador a página de datos DP

Registro de 32 bits, donde los ocho menos significativos se utilizan para direccionamiento directo como un apuntador a la página de datos direccionada. Las páginas son de 64 k-palabras, teniéndose un total de 256 páginas.

Registros índice IR0, IR1

Contienen el valor usado por la correspondiente ARAU para calcular una dirección indexada.

Registros de tamaño de bloque BK

Son utilizados por la ARAU para especificar el tamaño del bloque en el direccionamiento circular.

Apuntador al stack del sistema SP

Registro de 32 bits que contiene la dirección guardada en la localidad más alta del stack. Este registro siempre apunta al último elemento que entra. El stack es intervenido por interrupciones, llamadas y retornos de subrutinas, y por las instrucciones PUSH y POP.

Registro de estado ST

Guarda información global sobre el estado del CPU, es decir, contiene las banderas que indican si el resultado de una operación fue cero o negativo. Es posible mediante software salvar y restaurar este registro.

Nombre del registro	*Función*
R0	Registro de precisión extendida 0
R1	Registro de precisión extendida 1
R2	Registro de precisión extendida 2
R3	Registro de precisión extendida 3
R4	Registro de precisión extendida 4
R5	Registro de precisión extendida 5
R6	Registro de precisión extendida 6
R7	Registro de precisión extendida 7
AR0	Registro auxiliar 0
AR1	Registro auxiliar 1
AR2	Registro auxiliar 2
AR3	Registro auxiliar 3
AR4	Registro auxiliar 4
AR5	Registro auxiliar 5
AR6	Registro auxiliar 6
AR7	Registro auxiliar 7
DP	Apuntador a página de datos
IR0	Registro índice 0
IR1	Registro índice 1
BK	Registro de tamaño de bloque
SP	Apuntador al stack
ST	Registro de estado
IE	Registro de habilitación de int. CPU/DMA
IF	Registro de banderas de int. del CPU
IOF	Registro de banderas de E/S
RS	Registro de dir. inicial de repetición
RE	Registro de dir. final de repetición
RC	Contador de repetición
PC	Contador de programa

Tabla 4.1. Registros del TMS320C30

Registro de habilitación de interrupciones de CPU/DMA IE

Registro de 32 bits donde un "1"en la posición adecuada habilita la interrupción correspondiente. Las interrupciones para DMA están en los bits 26 a 16 de este registro. Un 0 deshabilita la interrupción.

Registro de banderas de interrupciones del CPU IF

Registro de 32 bits en el cual, de la misma manera que el anterior, con un "1"habilita y con un "0" deshabilita la interrupción correspondiente.

Registro de banderas de E/S IOF

Controla los pines externos XF0 y XF1, los cuales pueden ser configurados como entradas o salidas.

Registro de dirección inicial de repetición RS

Cuando el procesador se encuentra operando en el modo de repetición, este registro contiene la dirección inicial del bloque de instrucciones que van a ser repetidas.

Registro de direccionamiento final de repetición RE

Contiene la última dirección del bloque de programa que va a ser repetido.

Contador de repetición RC

Registro de 32 bits que especifica el número de veces que un bloque de código se repetirá.

Contador de programa PC

Registro de 32 bits que contiene la dirección de la siguiente instrucción que será accesada por el CPU. Aunque el PC no es parte de los registros del CPU, sí es un registro que puede ser modificado por instrucciones que alteran el flujo del programa.

4.4 Organización de la memoria

El espacio total de memoria del procesador definido por un bus de direcciones de 24 bits da por resultado 16M-palabras. El programa, datos y espacio de E/S están contenidos dentro de este espacio de 16M-palabras, lo cual permite que tablas, coeficientes, código de programa o datos puedan estar guardados ya sea en RAM o en ROM.

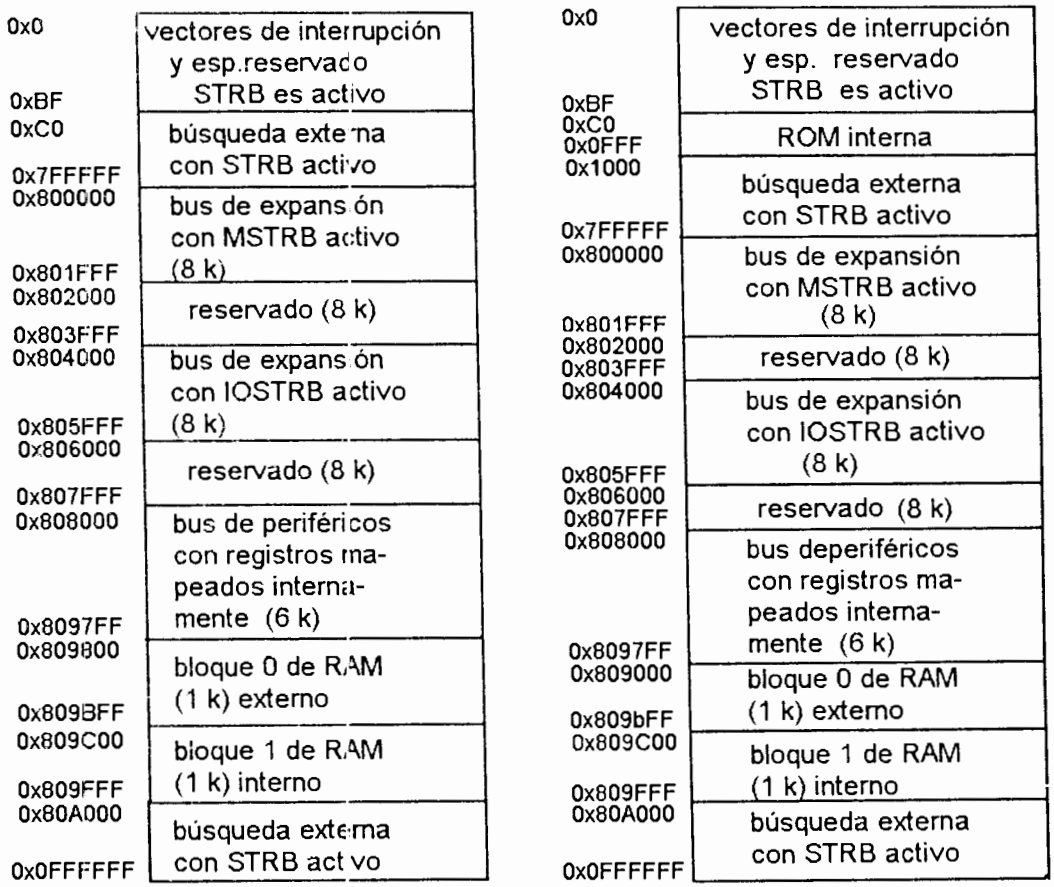
Memorias RAM, ROM y Caché

El procesador tiene dos bloques de memoria RAM (Bloque 0 y Bloque 1), cada uno de ellos es de 1k-palabras, y un bloque de ROM de 4k-palabras. Cada bloque permite dos accesos en un solo ciclo de máquina.

Dado que los buses de programa (PADDR, PDATA), de datos (DADDR1, DADDR2) y de DMA (DMAADDR, DMADATA) están separados, pueden realizarse paralelamente lecturas y escrituras de datos, acceso al código del programa y operaciones de DMA. Existe un caché de instrucciones de 64 palabras que almacena las secciones de código más utilizadas, reduciendo así el número de accesos necesario.

Mapa de memoria

La configuración del mapa de memoria depende del estado del pin $MC/\overline{MP}$.



a) Modo de microprocesador b) Modo de microcontrolador

Figura 4.2. Mapa de memoria

Modo de microcomputadora ($MC/\overline{MP} = 1$). El espacio en ROM de 4k-palabras está mapeado de las localidades 0h a la 0FFFh; las primeras 192 localidades se destinan a

vectores de interrupción, vectores de "trampa", y a un espacio reservado. Las localidades de la 1000h a la 7FFFFFh son para memoria externa, cuando está activo el pin $\overline{STRB}$.

En el modo de microprocesador ($MC/\overline{MP} = 0$) no existe el bloque de 4k de ROM. Las primeras 192 localidades están destinadas a interrupciones, trampas y algunas localidades reservadas. En este espacio se accesa a memoria externa cuando está activo el pin $\overline{STRB}$. A partir de la localidad C0h, es decir, el espacio de memoria restante corresponde a memoria externa.

Para ambos modos de operación, microcomputadora y microprocesador, las localidades 800000h a la 801FFFh están mapeadas al bus de expansión cuando se activa el pin $\overline{MSTRB}$. Los registros de control de los periféricos se encuentran entre las localidades 808000h a la 8097FFh para ambos modos de operación. El bloque 0 de RAM está mapeado de la 809800h a la 809BFFh y el bloque 1 de la 809C00h a la 809FFFh, y las localidades 80A000h a la 0FFFFFFh están destinadas a memoria externa.

Modos de direccionamiento

Este microprocesador soporta seis tipos de direccionamiento:

- *Por registro:* el operando es siempre un registro del CPU, pudiendo ser un registro de precisión extendida.
- *Corto-inmediato:* el operando es un valor inmediato de 16 bits.
- *Largo-inmediato:* el operando es un valor inmediato de 24 bits.
- *Directo:* el operando es el contenido de una dirección de 16 bits.
- *Indirecto:* un registro auxiliar indica la dirección del operando.
- *Relativo al PC:* se suma al PC un desplazamiento signado de 16 bits.

4.5 Operación del bus interno

El contador de programa (PC) se encuentra conectado al bus de direcciones de programa, y el registro de instrucción (IR) al bus de datos de programa; ambos buses pueden accesar sólo una palabra cada ciclo de máquina. En cuanto al bus de direcciones de datos, éste permite dos accesos en un ciclo de máquina.

El controlador de DMA posee sus propios buses de direcciones (DMAADDR) y de datos (DMADATA), lo cual permite al controlador accesar a memoria en paralelo con los accesos a memoria de los buses de programa y datos.

4.6 Operación del bus externo

Está constituido por un bus primario de direcciones de 24 bits y un bus de expansión de 13 bits. Este bus accesa memoria de programa, datos y espacio de E/S. Se utiliza una señal

externa $\overline{RDY}$ para generar estados de espera, cuando se accesan memorias o dispositivos lentos.

Interrupciones externas

El CPU soporta cuatro interrupciones externas ($\overline{INT3} - \overline{INT0}$). Maneja también interrupciones internas y una señal externa de $\overline{RESET}$ no mascarable. Estas señales pueden interrumpir, ya sea al CPU o al controlador de DMA. Cuando el CPU responde a la interrupción, el pin $\overline{IACK}$ puede utilizarse como una contraseña externa a la interrupción.

Señalización para ejecución de instrucciones sincronizadas

EL CPU utiliza dos banderas externas XF0 y XF1 configurables mediante software como entradas o salidas. Se utilizan por las operaciones de sincronización del procesador, las cuales soportan comunicación entre microprocesadores, así como también es posible establecer tiempos de espera, entre otras aplicaciones.

4.7 Control de periféricos

Los periféricos incluidos en el circuito integrado son dos puertos seriales y dos temporizadores. El control de ellos se realiza a través de registros mapeados en memoria, para lo cual se utiliza un bus periférico específico. Dichos registros de control se encuentran en las localidades 808000h hasta la 8097FFh.

Temporizadores

Los dos módulos de temporización de 32 bits pueden funcionar como temporizadores, o bien, como contadores de eventos con dos modos de señalización y una señal de reloj generada, ya sea interna o externamente. Asociado a cada módulo, hay un pin que puede ser configurado como entrada de reloj para el temporizador o como una salida controlada por el mismo procesador.

Puertos seriales

Ambos puertos son completamente independientes uno del otro, pero idénticos en su configuración de registros y también configurables para operar con palabras de datos de 8, 16, 24 o 32 bits.

La señal de reloj para cada puerto serial puede ser generada en forma interna o externa, contándose con un divisor de frecuencia.

4.8 Acceso directo a memoria (DMA)

El controlador de DMA, como ya se mencionó, puede leer o escribir en la memoria sin interferir con la operación del CPU. Esto es de gran utilidad cuando se interactúa con

dispositivos externos de baja velocidad, evitando disminuir el desempeño del CPU. El controlador de DMA contiene sus propios generadores de direcciones, registro fuente y destino, y contadores de transferencia. Dado que el controlador posee sus propios buses de datos y direcciones, se minimizan los conflictos con el CPU. Una operación de DMA consiste en la transferencia de una palabra o un bloque de datos desde o hacia la memoria.

4.9 Operaciones con pipeline

El microprocesador, al accesar una instrucción de memoria y proceder a ejecutarla, realiza básicamente cuatro operaciones, las cuales son ejecutadas en el modo pipeline como sigue:

Unidad de búsqueda (FETCH) (F)

Obtiene las palabras de instrucción de memoria y actualiza el contador de programa (PC).

Unidad de decodificación (D)

Realiza la decodificación de la instrucción y genera una dirección, también controla las modificaciones realizadas a los registros auxiliares y al apuntador del stack.

Unidad de lectura (R)

En caso de ser necesario, realiza lectura de operandos.

Unidad de ejecución (E)

De requerirse, lee los operandos del archivo de registros, ejecuta la operación necesaria y actualiza el archivo de registros. En ocasiones también escribe resultados previos a memoria.

Cuando estas cuatro operaciones se superponen perfectamente en el mismo ciclo de pipeline, operando en paralelo, la potencialidad del pipeline se utiliza plenamente (figura 4.3). Para el programador, la ejecución de instrucciones en pipeline es transparente, sin embargo, éste puede buscar que el acomodo de las instrucciones favorezcan que se realice óptimamente.

Durante la ejecución de operaciones en pipeline pueden generarse conflictos debidos a los diferentes accesos que realiza una instrucción en particular, o bien, al uso del stack que pueda realizar, entre otras causas.

Prioridad de las unidades de pipeline

Se asigna una prioridad a las operaciones, esto es, el orden en que se llevan a cabo:

- Ejecución (E)
- Lectura (R)
- Decodificación (D)

- Búsqueda (F)
- Canal de DMA (DMA) (en caso de presentarse)

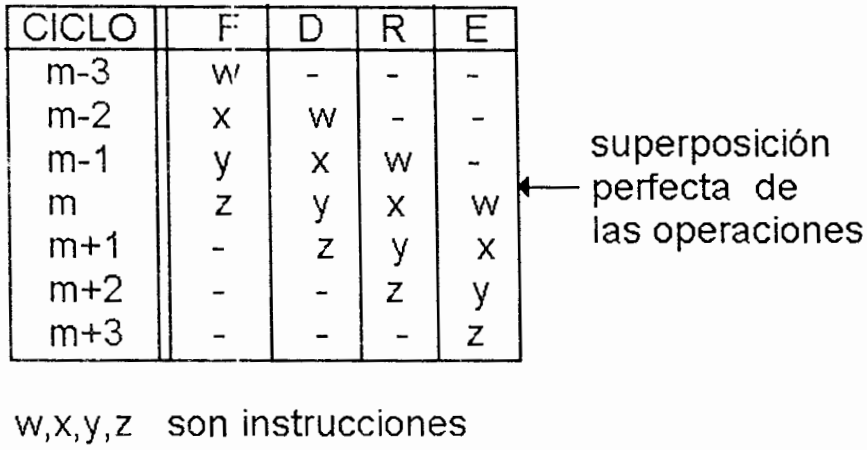


Figura 4.3. Estructura de las operaciones en pipeline

Cuando una instrucción está lista para entrar al siguiente nivel de pipeline, pero ese nivel no está listo para aceptar a dicha instrucción se presenta un conflicto. En este caso, la unidad de más baja prioridad espera hasta que la siguiente unidad en prioridad complete su ejecución.

Los conflictos entre las operaciones de DMA y la estructura de pipeline se minimizan debido a que el controlador de DMA posee sus propios buses de datos y direcciones.

Conflictos en pipeline

Pueden agruparse en tres categorías principales:

1. *Conflictos de saltos.* Se generan cuando alguna instrucción lee o modifica el contador de programa (PC).
2. *Conflictos con registros.* Involucran retardos generados al leer o escribir los registros cuando sean usados para generar direcciones.
3. *Conflictos con memoria.* Ocurren cuando las diferentes unidades del microprocesador compiten por los recursos de memoria.

4.10 Uso de los recursos del sistema

El microprocesador soporta aritmética entera y de punto flotante, lo que permite al programador concentrarse en el algoritmo que debe programar y preocuparse lo menos posible de problemas como escalamiento, rango dinámico o sobreflujos.

Al efectuarse un RESET, el microprocesador finaliza la ejecución del programa en curso y coloca el valor del vector de reset en el contador de programa (PC). El reset por hardware (pin externo) también inicializa varios registros y bits de estado.

Después del reset deben inicializarse modos de operación, apuntadores de memoria, interrupciones y especificaciones propias de la aplicación. Dentro de la configuración inicial debe incluirse la asignación de valores iniciales a los registros mapeados en memoria y en la estructura de interrupciones.

Control de flujo dentro de un programa

El modelo de programación de este microprocesador posee diversas estructuras que facilitan la programación de algoritmos de procesamiento digital de señales, como son:

- Llamadas a subrutinas (llamadas, trampas y retornos).
- Uso del stack por software.
- Saltos retardados.
- Operaciones en modo multiprocesador.
- Interrupciones.
- Modo de repetición.

Llamadas a subrutinas (llamadas, trampas y retornos)

Una vez diseñada la subrutina correspondiente, se ejecuta mediante las instrucciones CALL, CALLcond y TRAPcond. A través de las instrucciones RETScond y RETIcond se regresa de la subrutina invocada, lo que restaura automáticamente el stack.

Uso del stack por software

El microprocesador tiene un stack controlado por software cuya localización se determina por el contenido del apuntador al stack (SP). Dicho stack se accesa no solamente por las instrucciones CALL y RETS, sino también como una forma de almacenamiento temporal con las instrucciones PUSH y POP para números enteros, y PUSHF y POPF para números de punto flotante. El apuntador del stack no se inicializa por hardware durante el reset, es entonces importante asignarle un valor con una dirección de memoria predeterminada.

Saltos retardados

Este tipo de saltos funcionan como los saltos normales, sin embargo, no omiten el uso de la estructura de pipeline, por lo que es conveniente procurar utilizarlos. Cuando un salto retardado es encontrado dentro de un programa, las tres instrucciones siguientes dentro de un programa son ejecutadas. Las restricciones en este caso son que ninguna de estas tres instrucciones sea un salto, llamada a subrutina, retorno de subrutina o de interrupción, instrucción de repetición, instrucción trampa o instrucción de espera.

Operación en modo multiprocesador

Estas operaciones, auxiliadas mediante señalización externa, garantizan la integridad de la comunicación e incrementan la velocidad de ejecución.

Para mantener el acceso a una memoria global y compartir datos de una manera coherente es necesario un protocolo para arbitrar este tipo de procesamiento. Este es el propósito de un pequeño conjunto de instrucciones que ayudan a la sincronización (LDFI, LDII, SIGI, STFI, STII).

Interrupciones

Se encuentran en forma vectorizada y existe un orden de prioridad entre ellas. Cuando una interrupción ocurre, se activa el correspondiente bit en el registro de banderas de interrupción (IF). Si en el registro de habilitación de interrupciones (IE), el bit correspondiente se encuentra activo, y las interrupciones a su vez se han habilitado por el bit GIE en el registro de estado, se ejecuta la rutina de interrupción respectiva. Existe también la posibilidad de escribir en el IE y de esta manera forzar la interrupción por software, o bien, deshabilitarla para que no se lleve a cabo.

Exceptuando rutinas de interrupción muy simples, es importante asegurar que el contexto del procesador (es decir, el archivo de registros) se haya respaldado previamente al dar el salto a la subrutina. Este conjunto de registros debe almacenarse en el stack con anterioridad al brinco de subrutina y recuperarse de manera inversa (el stack puede verse como una memoria LIFO), conservándose entonces el contexto original.

Modo de repetición

Se consideran dos modos de repetición: repetición de un bloque de código (RPTB) y repetición de una sola instrucción (RPTS). El control del número de repeticiones, así como las direcciones inicial y final del bloque que se van a repetir, se realiza manipulando los registros RS, RE y RC.

4.11 Conjunto de instrucciones

Con un total de 113 instrucciones, el microprocesador TMS320C30 provee un versátil juego de instrucciones, tanto de propósito general como las que están especialmente diseñadas para cálculo intensivo.

A su vez el conjunto se subdivide en grupos de instrucciones especializadas en carga-almacenamiento, operaciones lógicas o aritméticas de dos o tres operandos, operaciones paralelas -justificando plenamente la existencia de dos ALU-, operaciones de control de programa -donde sobresalen las instrucciones para repetición de instrucciones o bloques de ellas-, así como cinco instrucciones previstas especialmente para ejecución compartida con otros microprocesadores C30. Todas las instrucciones ocupan al codificarse una palabra completa de 32 bits, y algunas de ellas se ejecutan en un solo ciclo. La eficiencia en la ejecución depende también de la optimación del código que favorezca las operaciones en pipeline.

Instrucción	*Descripción*	*Ciclos*
LDE	Carga exponente de punto flotante.	1
LDF	Carga valor de punto flotante.	1
LDFcond	Carga v. de p. f. condicionalmente.	1
LDI	Carga un entero.	1
LDIcond	Carga un entero condicionalmente.	1
LDM	Carga mantisa de punto flotante.	1
POP	Extrae un entero del stack.	1
POPF	Extrae un valor de p. f. del stack.	1
PUSH	Deposita un entero en el stack.	1
PUSHF	Deposita un valor p. f. al stack.	1
STF	Almacena un valor de p.f.	1
STI	Almacena un entero.	1

Tabla 4.2. Instrucciones de carga y almacenamiento

Instrucciones de carga y almacenamiento

Se consideran 12 instrucciones de este tipo para realizar cargas de memoria hacia un registro, cargas de un registro hacia memoria, o manipulación de datos en el stack del sistema. Dos de estas instrucciones pueden efectuar cargas condicionales, tabla 4.2.

Instrucciones de dos operandos

Son 35 instrucciones de este tipo, donde el operando fuente puede ser una localidad de memoria, un registro, o parte de la palabra de instrucción. El operando destino es siempre un registro del CPU. Se realizan operaciones con enteros, punto flotante, lógicas y de aritmética de precisión múltiple, tabla 4.3.

Operaciones con tres operandos

Estas operaciones son un beneficio directo de la duplicación de los buses internos en el CPU.

En este caso, el CPU toma dos operandos fuente, ya sea de registros o memoria, o bien, un operando y un contador, y deposita el resultado de la operación en un registro. Las 14 instrucciones de tres operandos se distinguen por terminar siempre en número 3, tabla 4.4.

Instrucciones para control de flujo

Este grupo consta de 15 instrucciones utilizadas para realizar saltos, llamar a subrutinas, ejecutar repeticiones de segmentos de código, efectuar ciclos. Cabe hacer notar la existencia de saltos retardados, en los cuales las siguientes tres instrucciones después del salto son cargadas sin incrementar el PC, dando por resultado un salto en un solo ciclo, lo que a su vez agiliza la ejecución en pipeline, tabla 4.5.

Instrucción	*Descripción*	*Ciclos*
ABSF	Valor absoluto de un valor de p.f.	1
ABSI	Valor absoluto de un entero.	1
ADDC	Suma de enteros con acarreo.	1
ADDF	Suma de valores de p.f.	1
ADDI	Suma de enteros.	1
AND	Operación AND a nivel de bits.	1
ANDN	AND al nivel de bits con complemento.	1
	de uno de los operandos.	1
ASH	Corrimento aritmético.	1
CMPF	Compara valores de p. f.	1
CMPI	Compara enteros.	1
FIX	Convierte valor de p.f. a entero.	1
FLOAT	Convierte entero a p.f.	1
LSH	Corrimiento lógico.	1
MPYF	Multiplica valores de p.f.	1
MPYI	Multiplica enteros.	1
NEGB	Negación entera con préstamo.	1
NEGF	Negación de p.f.	1
NEGI	Negación entera.	1
NORM	Normalización de un valor de p.f.	1
NOT	Complemento lógico al nivel de bits.	1
OR	Operación OR al nivel de bits.	1
RND	Redondeo de un valor p.f.	1
ROL	Rotación a la izquierda de un bit.	1
ROLC	Rotación a la izquierda de un bit,	1
	incluyendo al bit de acarreo.	1
ROR	Rotación a la derecha de un bit.	1
RORC	Rotación a la derecha de un bit,	1
	incluyendo al bit de acarreo.	1
SUBB	Sustracción de enteros con préstamo.	1
SUBC	Sustracción condicional de enteros.	1
SUBF	Sustracción de valores de p.f.	1
SUBI	Sustracción de enteros.	1
SUBRB	Sustracción inversa de enteros con	1
	préstamo.	1
SUBRF	Sustracción inversa de valores	1
	de p.f.	1
SUBRI	Sustracción inversa de enteros.	1
TSTB	AND al nivel de bits entre dos ente-	1
	ros, pero sin sustitución de regs.	1
XOR	Operación OR exclusiva.	1

Tabla 4.3. Instrucciones de dos operandos

Instrucción	*Descripción*	*Ciclos*
ADDC3	Suma con acarreo.	1
ADDF3	Suma de valores de p.f.	1
ADDI3	Suma de enteros.	1
CMPF3	Compara valores de p. flotante.	1
CMPI3	Compara enteros.	1
LSH3	Corrimiento lógico.	1
MPYF3	Multiplicación de valores p.f.	1
MPYI3	Multiplicación de enteros.	1
OR3	Operación OR al nivel de bits.	1
SUBB3	Sustracción de enteros con préstamo.	1 1
SUBF3	Sustracción de valores de p.f.	1
SUBI3	Sustracción de enteros.	1
TSTB3	Operación AND al nivel de bits, en versión de 3 operandos.	1 1
XOR3	Operación OR exclusiva, en versión de 3 operandos.	1 1

Tabla 4.4. Operaciones con tres operandos

Instrucción	*Descripción*	*Ciclos*
Bcond	Salto condicional (estándar).	4
BcondD	Salto condicional (retardado).	1
BR	Salto incondicional (estándar).	4
BRD	Salto incondicional (retardado).	1
CALL	Llamada a subrutina.	4
CALLcond	Llamada condicional a subrutina.	5
DBcond	Decremento y salto condicional (estándar).	4
DBcondD	Decremento y salto condicional (retardado).	1
IDLE	Desocupado hasta interrupción.	1
NOP	No realiza ninguna operación.	1
RETIcond	Regreso de interrupción condicionado.	4
RETScond	Regreso de subrutina condicionado.	4
RPTB	Repite bloque de instrucciones.	4
RPTS	Repite una sola instrucción.	4
SWI	Ejecuta una interrupción del emulador (instr. reservada).	4

Tabla 4.5. Instrucciones para control de flujo

Instrucción	*Descripción*	*Ciclos*
LDFI	Carga un valor de punto flotante.	1*
LDII	Carga un entero.	1*
SIGI	Señalización externa a través de los pines XF0 y XF1.	1 1
STFI	Almacena un valor de punto flotante.	1
STII	Almacena un entero.	1

Tabla 4.6. Instrucciones para operar sincronizadamente

* La duración de estas instrucciones es de un ciclo, siempre y cuando la bandera $XF = 0$.

Instrucciones para operar sincronizadamente

El microprocesador TMS320C30 posee cinco instrucciones diseñadas especialmente para permitir la comunicación entre procesadores y el manejo de señales externas de sincronización.

Instrucciones para operaciones en paralelo

Como consecuencia de la duplicación interna de buses, cierto número de operaciones de carga, aritméticas y lógicas están permitidas para su ejecución de manera paralela. Al ser escrito el código fuente, se indica la operación paralela con una doble barra (||) entre ambas operaciones. A continuación se listan los pares de instrucciones permitidos. Es claro que se debe buscar el paralelismo durante la programación, puesto que todas las operaciones paralelas permitidas se ejecutan en un solo ciclo.

ABSF	|| STF	NEGI	|| STI
ABSI	|| STI	NOT3	|| STI
ADDF3	|| STF	OR3	|| STI
ADDI3	|| STI	STF	|| STF
FIX	|| STI	SUB3	|| STI
FLOAT	|| STF	XOR3	|| STI
LDF	|| STF	LDF	|| LDF
LDI	|| STI	LDI	|| LDI
LSH3	|| STI	MPYF3	|| ADDF3
MPYF3	|| STF	MPYF3	|| SUBF3
MPYI3	|| STI	MPYI3	|| ADDI3
NEGF	|| STF	MPYI3	|| SUBI3

Tabla 4.7. Instrucciones para operaciones en paralelo

Capítulo 5

Herramienta de desarrollo EVM

5.1 Introducción

En este capítulo se describe brevemente la herramienta de desarrollo para ejecutar algunos de los algoritmos que se proponen más adelante. El módulo de evaluación (EVM) es una tarjeta diseñada para usarse con el software depurador de código (*C Source Debugger*), el cual ofrece al usuario la posibilidad de realizar una emulación en pantalla, a la vez que los algoritmos se ejecutan en la tarjeta. La intervención del controlador de bus de prueba (TBC) permite realizar la emulación directa en la tarjeta, parar la ejecución del programa, y visualizar registros y memoria. Mediante el uso de un puerto de emulación MPSD y el TBC se carga el programa en la EVM.

Se define un protocolo de comunicación entre el host (una computadora IBM PC/AT compatible) y la tarjeta objetivo, es decir, la EVM. Dicho protocolo basa su operación en el intercambio de información entre registros mapeados en memoria, lo que permite desarrollar software de propósito específico para una aplicación dada, utilizando un lenguaje de alto nivel que permita el acceso a memoria, como lo es el lenguaje C.

El programa debe estar previamente formateado según la especificación COFF (common object file format), para generar código objeto a partir de segmentos de programa escritos, tanto en ensamblador como en ANSI C. De este código objeto se genera código ejecutable en un archivo `.out`, que debe cargarse en la tarjeta ya sea mediante el programa `evmload.exe`, o bien, con el ambiente del depurador.

5.2 Características generales

Interfaz con el bus anfitrión

La tarjeta emuladora tiene las especificaciones necesarias para insertarse en una ranura libre del bus IBM PC/AT de tarjetas de 8 bits. En la figura 5.1 se esquematiza la tarjeta EVM y sus componentes principales.

La diferencia más notoria sobre otros emuladores existentes consiste en lo que sus diseñadores denominan emulación en el sistema mismo (*embedded in-system emulation*). Esto significa que la emulación se efectúa dentro del sistema objetivo (la EVM), realizándose la interfaz con el host mediante un circuito destinado a ello: el 74ACT8990, *Test Bus Controller*,

(TBC). Dicho circuito controla el intercambio de información entre el TMS320C30 y el bus de la computadora anfitriona.

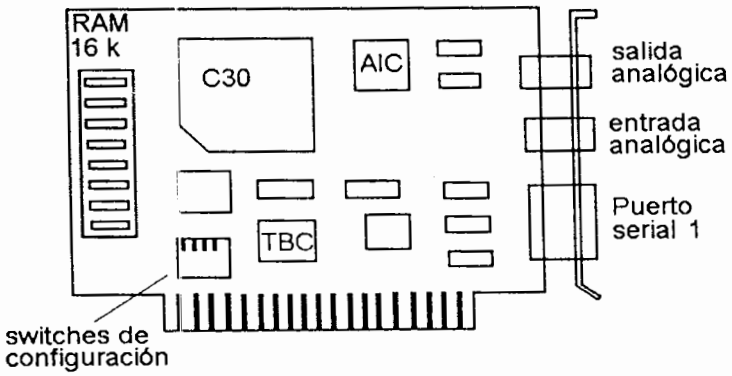


Figura 5.1. Módulo de evaluación (EVM) para el TMS320C30

Un programa que se ejecuta desde la memoria de la computadora personal, se comunica en realidad con el TBC a través del poleo (monitoreo) de seis direcciones de 16 bits. El TBC interactuará con el TMS320C30 mediante interrupciones por hardware por medio de los pines INT0, INT1 e INT2. Esto quiere decir que la comunicación desde un programa anfitrión en la computadora opera con poleo, mientras que el TMS320C30 se comunica con el TBC a través de interrupciones de hardware, lo que le proporciona a este último total autonomía del anfitrión mientras no se realice una requisición de intercambio de información.

La configuración del sistema, como se ha referido, provee una tasa moderada de intercambio de información (alrededor de 200 kbytes por segundo), sin embargo, garantiza la sincronización de dicha transferencia, ya que si un dato tiene que ser escrito o leído por el host (la PC) es necesario hacer un poleo y borrado de banderas en los registros mapeados en memoria. En ciertas condiciones, es posible evitar dicho poleo, que eleva el *throughput* [1] hasta 400 kbytes por segundo.

Como hardware adicional en la generación de señales de protocolo (*glue logic*) se incluyen un par de PAL (dispositivos lógicos programables de uso generalizado), y dos tranceivers 74ALS652. El formato de interfaz entre el TBC y el puerto de emulación del TMS320C30 sigue las especificaciones MPSD (*modular port scan device*) de Texas Instruments.

Memoria externa

El módulo de evaluación posee un banco de memoria estática SRAM de 16k-palabras con cero estados de espera y se halla directamente conectado al espacio de direcciones del

[1] ***Throughput*** es un término generalmente aceptado para referir la cantidad de información digital que puede obtenerse como salida de un sistema, se utiliza como un parámetro de desempeño.

bus primario. Las memorias utilizadas son de la marca Cypress, modelo CY7C164-35VC, con un tiempo de acceso de 35 ns, lo que requiere un reloj de 30 MHz como base de tiempo en el 'C30 para satisfacer los requerimientos de la SRAM. Este banco de memoria dota al módulo de cierta autonomía en cuanto a capacidad de almacenamiento se refiere. Es posible almacenar aproximadamente dos segundos de una señal muestreada a 8 kHz sin ser necesario el intercambio de información con el host.

Interfaz analógica

De los dos puertos seriales de que dispone el TMS320C30, el primero de ellos (puerto 0) se ha utilizado para comunicarse con un controlador de interfaz analógica: el TLC32044 *analog interface controller* (AIC). Dicha interfaz necesita de una base de tiempo, por lo que se utilizó para ello el primer temporizador, además del pin XF0 como $\overline{RESET}$ del AIC. Este controlador provee de una interfaz A/D y D/A con 14 bits de resolución, tasa de muestreo y filtrado programables, que cubren satisfactoriamente los requerimientos de una arquitectura para procesar señales de voz.

La entrada analógica acepta voltajes en un rango de ± 1.5 V y de ± 3 V. Se incluye también una etapa de acondicionamiento de la señal proveniente del exterior que consta de un seguidor y un amplificador con ganancia 2, basados en amplificadores TL072 con entrada JFET. La salida analógica acepta una carga mínima de salida de 300Ω, sin embargo, para poder ser manejada por un equipo amplificador comercial, es necesario acoplarla para cargas de hasta 4Ω. Con una salida típica de 1.5 Vpp, el AIC se conecta a una etapa de atenuación con una ganancia de x0.2 para posteriormente ser amplificado por un factor de x20 en el amplificador de potencia LM386, de manera que la ganancia de voltaje resultante sea de x4. Por ejemplo, para una carga de 8Ω con una salida inicial de 1.5 Vpp, el voltaje resultante a la salida del LM386 será de 6 Vpp.

Puerto serial externo

El puerto serial 1, completamente independiente de aquél que se usa como interfaz A/D y D/A, se ha previsto como un medio de comunicación al exterior de la EVM. Se incluye además en el conector de salida de 10 pins la señal XF1. Los pins disponibles en dicho conector se presentan en la tabla 5.1.

Pin	*Señal*	*Pin*	*Señal*
1	GND	2	FSR1
3	CLKX1	4	DR1
5	DX1	6	TCLK1
7	FSX1	8	XF1
9	CLKR1	10	GND

Tabla 5.1. Señales de puerto serial

Los usos que pueden darse a dicho conector son diversos: conexión con otras EVM para

ejecutar procesamiento distribuido, conexión a otro dispositivo serial para recibir información, como puede ser el backplane de un PBX, etc.

5.3 Configuración y generación de código

Configuración del mapa de memoria de la tarjeta

Es posible configurar algunas características de la tarjeta mediante microinterruptores. Como se muestra en la tabla 5.2, con los interruptores marcados SW1, SW2, se selecciona la dirección base de los registros utilizados por el programa de aplicación en el host para la comunicación hacia la EVM. La primera dirección base debe ser la opción por default, a menos que exista otro periférico (por ejemplo, un cursor) en ese espacio de memoria.

SW1	SW2	*Dirección base*
ON	ON	$0 \times 0240 - 0 \times 025F$
ON	OFF	$0 \times 0280 - 0 \times 029F$
OFF	ON	$0 \times 0320 - 0 \times 033F$
OFF	OFF	$0 \times 0340 - 0 \times 035F$

Tabla 5.2. Microinterruptores en EVM

Los interruptores SW3 y SW4, por lo general, se mantienen en ON para permitir modo de microprocesador y la salida SBM en alta impedancia, respectivamente.

Configuración interna del TMS320C30

Al inicio de cualquier programa que sea cargado en la memoria de la tarjeta, es absolutamente necesario realizar una inicialización de algunas variables y banderas internas. A continuación se muestra un segmento de programa que efectúa dicha inicialización, habilitando sólo las interrupciones que son necesarias, como aquéllas utilizadas para comunicarse con el AIC. Esta secuencia de instrucciones puede variar un poco, dependiendo principalmente de los periféricos que se utilicen y de las interrupciones que se manejen.

```
elstack     .usect  "stack",100h
   .sect    "vectors"
PARMS:
reset       .word   sysinit ; al arrancar ejecuta sysinit
int0        .word   interr0
int1        .word   interr1
int2        .word   interr2
int3        .word   interr3
xint0       .word   transmit0
rint0       .word   receive0 ; RINT0 es causada por el AIC al mandar un dato,
```

```
xint1       .word   transmit1 ; y es atendida por receive0
rint1       .word   recieve1
tint0       .word   timer0
tint1       .word   timer1
dint0       .word   dmadone

   .sect   "comdata"
stack_addr .word   elstack      ; dirección del stack
dma_ctl    .word   000808000h   ; registro de control global del dma
mcntlr0    .word   000808064h   ; registro de control del bus primario
mcntlr1    .word   000808060h   ; registro de control del bus secundario
t0_ctladdr .word   000808020h   ; reg. control global timer 0
t1_ctladdr .word   000808030h   ; reg. control global timer 1
p0_addr    .word   000808040h   ; reg. control global puerto serial 0
enbl_sp0_r .word   000000020h   ; palabra para habilitar int. por Rx en p.s. 0
t0_ctlinit .word   0C00002C1h   ; habilita timer 0 como salida de reloj
                                ; a (H)1/2 (reloj interno), el timer
                                ; continúa funcionando aun cuando el CPU esté
                                ; en emulación
p0_global  .word   00e970300h   ; palabra para configurar el p.s. 0
CACHE              .set    1800h ; palabra para habilitar caché
ENBL_GIE           .set    2000h ; palabra para habilitar interrupciones
ENBL_XINT0         .set    0010h ; habilita la interrupción de Tx del p.s. 0
ENBL_RINT0         .set    0020h ; habilita la interrupción de Rx del p.s. 0

   .text
sysinit:
   xor     ie,ie           ; borra IE y deshabilita todas las ints.
   xor     if,if           ; también borra todas las banderas de int. en IF
   ldp     PARMS           ; carga DATA PAGE POINTER con los MSBs de PARMS
                                   ; para poder usar direccionamiento directo
   ldi     @stack_addr,sp ; carga un 100h en el STACK POINTER
   ldi     CACHE,st        ; habilita el bit de caché en el STATUS REGISTER
   ldi     0,r0            ; carga 0 en R0
   ldi     @mcntlr0,ar0    ; carga AR0 con la dirección del PRIMARY BUS
                                   ; CONTROL REGISTER
   sti     r0,*ar0         ; pone el bus I/O en READY
   ldi     @mcntlr1,ar0    ; carga AR0 con la dirección del EXPANSION BUS
                                   ; CONTROL REGISTER
   sti     r0,*ar0         ; también lo pone en READY
   or      80h,ST          ; habilita overflow mode en STATUS REGISTER


   or      ENBL_GIE,st ; habilita bit GLOBAL INT, que es el 13 de ST

interr0:   reti
interr1:   reti
```

```
interr2:   reti
interr3:   reti
transmit0: reti
transmit1: reti
recieve1:  reti
timer0:    reti
timer1:    reti
dmadone:   reti

*** como ejemplo, se propone una rutina de atención de interrupción, ******
*** que atienda a aquélla que se genera cuando se recibe un dato en p.s. 0 **

receive0:  push    st                  ; salva en el stack el valor de
           push    r0                  ; algunos registros
           push    R3
           push    ar0
           push    dp
           ldp     PARMS

**** instrucciones propias de la rutina ****

           pop     dp                  ; saca del stack el valor de los
           pop     ar0                 ; registros salvados
           pop     R3
           pop     r0
           pop     st
           reti
           .end
```

Generación de código

Una vez que se ha compilado o ensamblado cualquier programa para el TMS320C30 se genera un archivo `.obj`, el cual contiene ya el código de máquina en formato COFF (*Common Object File Format*) que es posible ejecutar en un sistema basado en este microprocesador.

Ensamblado

Si se trata de un programa escrito exclusivamente en lenguaje ensamblador, se utiliza el programa `asm30.exe` como sigue:

```
asm30 [ archivo_de_entrada [ archivo_objeto [archivo_de_lista ]]] [-opciones]
```

Las opciones más comunes son:

-v especifica la versión, puede ser -v30 para el TMS320C30 o -v40 para el TMS320C40.
-l produce un archivo de listado (.LST).
-i especifica un directorio para búsqueda de archivos especificados en las directivas `.copy`, `.include` o `.mlib`.

-c realiza la compilación con sensibilidad a la mayúsculas.
-x produce una tabla de referencias cruzadas al final del archivo de listado.

Una opción es invocar al programa sin ningún argumento, lo que ocasiona que el programa pida entonces los argumentos al usuario.

Ligado

Es necesaria una última etapa de ligado en la cual se asignan las direcciones definitivas para que el código ejecutable sea localizado dentro del mapa de memoria del sistema objetivo. En este caso, dicho sistema objetivo es la EVM, y es necesario realizar esta etapa de ligado con información adicional sobre la localización de los datos, del código de programa, memorias RAM y ROM, etc. Tal acción se ejecuta con el programa `lnk30.exe`, cuya sintaxis es la siguiente:

```
lnk30 [-opciones] archivo_1 ... archivo_n
```

Las opciones más usuales son:

-m produce un archivo de mapeo (.MAP)
-o especifica a continuación el archivo de salida (por default A.OUT)
-q al momento de ligar no aparece el rótulo de TI
-x fuerza a que relean las librerías y encuentre referencias no encontradas en la primera pasada
-a produce código con direcciones absolutas
-r produce código con direcciones relocalizables

Un ejemplo común de ligado es el siguiente:

```
lnk30 archivo  archivo_de_comandos -o archivo_de_salida
```

El ligador también tiene la opción de ser invocado sin argumentos, lo cual ocasiona que dicho programa pida los datos necesarios al usuario.

Archivos de comandos

Para que el ligador conozca la localización de la memoria dentro de un sistema objetivo, es necesario especificarle tal información en un archivo de texto con extensión `.cmd` (archivo de comandos). En dicho archivo de comandos se especifica la naturaleza de los bloques de memoria usados (lectura y escritura, sólo lectura) así como su localización y extensión dentro del mapa de memoria. A cada bloque se le da un nombre y a su vez se localizan en ellos las diferentes secciones del código. La división en secciones del código al momento de ligar obedece a que tal código puede tener diferentes características, siendo en algunas ocasiones código de programa, una tabla de datos fija o una tabla de datos variable.

En el formato `.coff` se definen tres secciones por default:

sección .text contiene código ejecutable.
sección .data contiene datos inicializados e invariables.

sección .bss es un espacio reservado para variables no inicializadas (por ejemplo, los estados internos de un filtro).

Además, el ligador permite que el usuario pueda crear otras secciones con características similares, y básicamente las clasifica en dos categorías: inicializadas y no inicializadas.

Secciones inicializadas

Contienen datos o código ejecutable, las secciones .text y .data son secciones inicializadas. El usuario puede crear nuevas secciones inicializadas con las directivas .sect y .asect.

Secciones no inicializadas

Se utilizan para reservar espacio en memoria para datos no inicializados. La sección .bss es de este tipo y el usuario puede definir nuevas secciones con la directiva .usect. Pueden encontrarse varios ejemplos de secciones no inicializadas creadas por el usuario con los nombres "stack", "vectors" y "comdata".

A continuación se presenta el contenido de un archivo de comandos utilizado para ligar el código de inicialización listado en la sección anterior.

```
MEMORY
{
INT_V : org = 0x000000, len = 0x40
SRAM  : org = 0x000040, len = 0x1FC0
a     : org = 0x002000, len = 0x2000
RAM0  : org = 0x809800, len = 0x400
RAM1  : org = 0x809C00, len = 0x400
}
SECTIONS
{
 vectors: {} > INT_V
 comdata: {} > SRAM
.text   : {} > SRAM
 buffer : {} > a
 stack  : {} > RAM1
}
```

5.4 Comunicación entre el host y la EVM

La microcomputadora anfitriona, es decir, el host, tiene comunicación con la EVM dentro de un esquema maestro-esclavo, donde obviamente el host ejecuta el papel de maestro. Como ya se ha mencionado, la interfaz entre ambos la realiza un circuito denominado *Test Bus Controller* (TBC), el cual es direccionado desde el bus AT mediante los PAL que actúan como decodificadores de memoria, y que también se utilizan entre el TBC y el 'C30 para sincronizar adecuadamente la producción de las interrupciones con la base de tiempo propia del microprocesador 'C30. Un esquema de la interconexión del bus PC/AT, el TBC y el TMS320C30 se muestra en la figura 5.2.

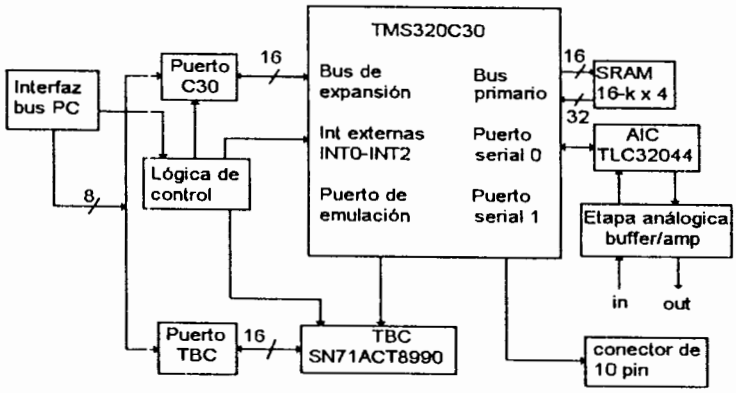


Figura 5.2. Interconexión entre el bus anfitrión, el TBC y el TMS320C30

Operaciones de lectura/escritura desde el host hacia el TBC

Se direccionan 6 registros desde el bus PC/AT, los cuales se localizan física y lógicamente en el TBC, y a través de los cuales un programa de aplicación interactúa con el TMS320C30. Aunque la mayor parte son de 16 bits, uno de ellos en realidad sólo existe como una dirección y al realizarse una escritura en él se ejecuta un reset por software (registro SOFT RESET). En la tabla 5.3 se presentan los desplazamientos a partir de una dirección base (ver configuración) de los registros mencionados, así como su nemónico asociado y su tamaño en bits.

Registro	*Desplazamiento*	*Tamaño (bits)*	*Acceso*
RESERVADO	$0 \times 0000 - 0 \times 0008$	-	-
CONTROL5	$0 \times 000A$	16	L/E
RESERVADO	$0 \times 000C - 0 \times 0012$	-	-
MINOR CMD	0×0014	16	L/E
RESERVADO	$0 \times 0016 - 0 \times 001E$	-	-
STATUS 0	0×0400	16	L
RESERVADO	$0 \times 0402 - 0 \times 041F$	-	-
COM CMD	0×0800	8	L/E
COM DATA	0×0808	16	L/E
SOFT RESET	0×818	0	E

Tabla 5.3. Registros desde el bus PC/AT

Manejo de eventos a través del TBC

El TBC considera cuatro eventos provenientes del bus PC/AT, los cuales a su vez disparan las correspondientes interrupciones al TMS320C30. Se enumeran de EVT0 a EVT3 y su utilización se describe a continuación:

EVT0: Embedded Simulation Support

Evento no modificable, se utiliza como parte del protocolo entre TBC y el puerto de emulación del TMS320C30.

EVT1: Host Read Acknowledge

Se activa (EVT1=1) cuando el TMS320C30 escribe al registro de comunicaciones. Entonces el host puede leer el dato de dicho registro y poner en cero esta bandera para que una futura escritura del C30 pueda ser detectada.

EVT2: Host Write Acknowledge

Se activa (EVT2=1) cuando el TMS320C30 efectúa una lectura del registro de comunicaciones. Esto quiere decir que el C30 ha recogido el dato. Acto seguido, el host debe poner de nuevo en cero a esta bandera y puede escribir otra vez.

EVT3: TMS320C30 Reset Source

Mientras esta bandera sea mantenida en 1, el C30 permanece en estado de reset. Existe un registro cuyo direccionamiento provoca dicho estado.

Escritura de datos

Para realizar la escritura de un dato localizado en una variable en la memoria del host hacia el TMS320C30, se llevan a cabo una serie de pasos, los cuales se ejemplifican en el siguiente segmento de un programa en Microsoft C:

```
#define IOBASE      0x0240
#define MINOR_CMD   0x0014
#define COM_DATA    0x0808
#define STATUS0     0x0400
#define MASK1       0x0004
#define MASK2       0x6044

/* la variable 16_bit_data contiene el valor que va a ser escrito */
unsigned short 16_bit_data;

do {

/* 1. borra la bandera EVT2 localizada en el reg. MINOR_CMD */
```

```
outpw(IOBASE + MINOR_CMD, MASK1);

/* 2. escribe el dato en el registro de comunicaciones */

outpw(IOBASE + COM_DATA, 16_bit_data);

/* 3. actualiza el registro de estado del TBC */

outpw(IOBASE + MINOR_CMD, MASK2);

/* 4. verifica que el dato ha sido leído del lado del TMS320C30 */

}while( inpw(IOBASE + STATUS0) & MASK1);

/* actualiza el registro de estado */

outpw(IOBASE + MINOR_CMD, MASK2);

/* 5. una vez abandonado el ciclo anterior se puede empezar de
nuevo desde 1 */
```

Lectura de datos

Para recoger un dato del registro de comunicaciones se debe polear la bandera EVT2 hasta que se indique la presencia de un dato en el registro de comunicaciones, como se muestra en el siguiente segmento de código:

```
#define IOBASE      0x0240
#define MINOR_CMD   0x0014
#define COM_DATA    0x0808
#define STATUS0     0x0400
#define MASK0       0x0002
#define MASK2       0x6044

unsigned short 16_bit_data;

do {

/* actualiza el registro de estado del TBC */

outpw(IOBASE + MINOR_CMD, MASK2);

/* si la bandera EVT1 es activa, se borra, en otro caso
regresa a actualizar el registro de estado */
```

```
if (inpw(IOBASE + STATUS0) & MASK0)
outpw(IOBASE + MINOR_CMD, MASK0);

}while(!(inpw(IOBASE + STATUS0) & MASK0));

/* procede a leer el dato */

16_bit_data = inpw(IOBASE + COM_DATA);
```

Reinicialización por software

La inicialización del TMS320C30 se da de manera automática cuando se le aplica energía por primera vez. Sin embargo, el TBC provee la manera de aplicarle un reset por software, a través del registro CONTROL5, de la siguiente manera:

```
#define IOBASE      0x0240
#define CONTROL5    0x000A
#define RESET_ON    0x0808
#define RESET_OFF   0x0800

outpw(IOBASE + CONTROL5, RESET_ON);

outpw(IOBASE + CONTROL5, RESET_OFF);
```

Existe una tercera manera de reinicializar al TMS320C30 sin la intervención del TBC. Como se había mencionado, el registro SOFT RESET no existe físicamente, pero su direccionamiento provoca que una señal activa en bajo se genere en el PAL UA5, el cual aplica el reset al microprocesador.

```
#define IOBASE      0x0240
#define SOFT_RESET  0x0818

outpw(IOBASE + SOFT_RESET, 0);
```

5.5 Manejo de información desde el TMS320C30

En el caso del TMS320C30, el manejo del protocolo de comunicaciones se realiza por medio de interrupciones, a través de las cuales dicho dispositivo da servicio a los procesos de lectura/escritura por parte del host. Cuando el TMS320C30 desea efectuar una lectura o escritura, la lógica de direccionamiento utilizada en las PAL accesa los registros del TBC. El registro COM DATA, visto desde el espacio de direcciones del 'C30, se encuentra en todo el bus de expansión como un solo registro de 16 bits habilitado para lectura y escritura. Cualquier dirección contenida dentro del espacio $0 \times 804000 - 0 \times 805FFF$ corresponde a este registro.

Mapa de memoria de la tarjeta

En la tabla 5.4 se presenta el mapa de memoria del módulo de evaluación, donde se incluyen todos aquellos registros mapeados en memoria para el control de periféricos, así como el espacio estándar de datos y programa. La memoria puede utilizarse para almacenar programa, datos, y también los vectores de reset, de interrupción y de trampa.

Manejo de interrupciones en el TMS320C30

Como se ha visto, las interrupciones por hardware son el medio mediante el cual el TBC avisa al microprocesador que se realizan operaciones de lectura/escritura. En este caso, las interrupciones se detectan por nivel durante el flanco de bajada de la señal H1, el 'C30 puede aceptar dos interrupciones de la misma fuente cada dos ciclos de reloj de H1.

Los PAL encargados de generar las fuentes de interrupción para el 'C30 se desarrollaron como máquinas de 4 estados (con 2 flip-flops) con entrada asíncrona, dichos PAL generan las señales $\overline{CNTLINT}$, $\overline{WRINT}$ y $\overline{RDINT}$ que controlan las correspondientes entradas INT0, INT1 e INT2 en el TMS320C30. La cuarta entrada disponible para interrupción se deshabilita mediante una resistencia de pull-up a +5V.

Para asegurar que sólo sea generada una interrupción en cada operación de lectura o escritura por parte del host, después de una interrupción el generador de interrupciones espera a que se genere una señal de lectura o escritura y que ésta sea inactiva, para después pasar la correspondiente interrupción al TMS320C30 a través de los pines INT0 a INT2.

Cada una de estas entradas realiza una de las siguientes funciones en el protocolo de comunicaciones.

INT0: command interrupt

Su estado activo indica que el host depositó un comando en el registro de comandos. Se genera cuando el host escribe un dato de 8 bits, pero no cuando el host hace una lectura, lo que le posibilita a este último polear el registro de estado del 'C30 sin ocasionar cambios en algún registro.

INT1: data write interrupt

Su estado activo indica que el host realizó una escritura de 16 bits en COM DATA.

INT2: data read interrupt

Su estado activo indica que el host realizó una lectura de 16 bits de COM DATA.

El uso de interrupciones por hardware posibilita al controlador de DMA para dar servicio a la entrada y salida de datos.

Dirección	*Función*
bus primario	
$0 \times 000000 - 0 \times 003FFF$	16k-palabras de SRAM
bus de expansión	
0×804000	registro COM DATA hacia el host
0×808000	control global del DMA
0×808004	dirección fuente de DMA
0×808006	dirección destino de DMA
0×808008	contador de transferencia de DMA
0×808020	control global del timer 0
$0 \times 8080240 \times 808028$	periodo del timer 0
0×808030	control global del timer 1
0×808034	contador del timer 1
0×808038	periodo del timer 1
0×808040	control global del puerto serie 0
0×808042	puerto control Tx p. serie 0
0×808043	puerto control Rx p. serie 0
0×808044	control de timer Rx/Tx p. serie 0
0×808045	contador de timer Rx/Tx p. serie 0
0×808046	periodo de timer Tx/Rx p. serie 0
0×808048	dato de Tx puerto serie 0
$0 \times 80804C$	dato de Rx puerto serie 0
0×808050	control global del puerto serie 1
0×808052	puerto control Tx p. serie 1
0×808053	puerto control Rx p. serie 1
0×808054	control de timer Rx/Tx p. serie 1
0×808055	contador de timer Rx/Tx p. serie 1
0×808056	periodo de timer Tx/Rx p. serie 1
0×808058	dato de Tx puerto serie 1
$0 \times 80805C$	dato de Rx puerto serie 1
0×808060	control del bus de expansión
0×808064	control del bus primario
$0 \times 809800 - 809FFF$	2k-palabras de RAM interna, acceso dual

Tabla 5.4. Mapa de la memoria de la tarjeta

5.6 Manejo del controlador de interfaz analógica

El controlador de interfaz analógica (AIC) es el dispositivo que hace posible que los algoritmos diseñados puedan ser realizados en tiempo real con las siguientes limitaciones:

- El rango de voltajes que acepta es cuando más de $\pm$ 3 volts.
- La frecuencia de muestreo máxima configurable es de 19200 Hz.

Asimismo, son programables las características anteriores, más:

- Un filtro corrector $\sin(x)/x$ en la salida del convertidor D/A.
- Un filtro paso bajas antialiasing de entrada de frecuencia programable.

El AIC se configura mediante un protocolo serial, posee un pin de reset y necesita también una base de tiempo. Estas cuestiones fueron resueltas en la arquitectura de la tarjeta como sigue:

1. Se empleó el puerto serial 1 para comunicarse con el AIC.
2. El timer 0 se utilizó como base de tiempo.
3. El pin XF0 funciona como señal de reset para el AIC.

Para lograr una adecuada comunicación entre el AIC y el TMS320C30 se debe tomar en cuenta las siguientes características: el AIC es capaz de realizar el intercambio de información con otro dispositivo mediante un protocolo serial asíncrono (es decir, a una tasa variable de transmisión) y con una longitud de palabra de datos de 16 bits; además, el AIC maneja datos de tipo entero, mientras que el TMS320C30 es un dispositivo de punto flotante, por lo que debe realizarse la conversión adecuada de la información.

El temporizador, usado como base de tiempo, se configura a una frecuencia de $f_{MCLK}7.5MHz$, siendo la mitad de la señal interna $H1$ (es decir: 15MHz/2 = 7.5 MHz). El pin $XF0$ es programable modificando su estado en el registro IOF.

El AIC soporta dos tipos de transmisión:

1. *Transmisión primaria.* Cuando el dispositivo simplemente intercambia información desde el convertidor A/D o hacia el convertidor D/A, los datos de 14 bits deben estar acomodados en los bits más significativos de la palabra de transmisión, asignando dos ceros a los bits menos significativos.
2. *Transmisión secundaria.* Este tipo de transmisión se inicia cuando se envían al AIC palabras con los dos bits menos significativos con valores de uno. A continuación del inicio de la transmisión secundaria, se envían palabras que configuran tanto la frecuencia de muestreo de entrada como la de salida, la frecuencia de corte de filtro antialiasing, así como la ventana de voltaje de los convertidores.

Tanto para Tx como para Rx deben configurarse dos parámetros, denominados A y B, que establecen las frecuencias de muestreo como sigue:

$$f_{SCF} = \frac{f_{MCLK}}{2A} \tag{5.1}$$

$$f_c = \frac{f_{MCLK}}{2AB} \tag{5.2}$$

Donde f_{SCF} es la frecuencia del filtro de capacitores conmutados de entrada (filtro paso-bajas) y f_c es la frecuencia de muestreo. Para ello deben calcularse los números TA, TB para la transmisión, RA y RB para la recepción, que serán enviados posteriormente al inicio de una transmisión secundaria, como puede apreciarse en la figura 5.3.

Los números TA y RA se utilizan para realizar un ajuste fino de las frecuencias de muestreo, y su valor se resta al contenido original de TA y RA. El parámetro A tiene un rango de valores entre 1 y 31, mientras que B, con un bit más de resolución, puede variar entre 1 y 63.

Además, existe un registro de control, accesible al colocar unos en los dos bits menos significativos de la palabra transmitida después del inicio de la transmisión secundaria. En dicha palabra se encuentran los bits $d2$ a $d9$ con los siguientes significados:

Bit	*Función*
d2	Habilita filtro paso-altas en A/D
d3	Habilita "loopback"
d4	Selecciona entrada analógica primaria(0) o secundaria (1)
d5	Tx y Rx síncrona
d7,d6	Selecciona entrada ± 1.5V (1,0) o entrada ± 3.0V (1,1)
d9	Habilita filtro corrector (sen x)/x en D/A

Tabla 5.5. Parámetros del registro de control del TLC32044

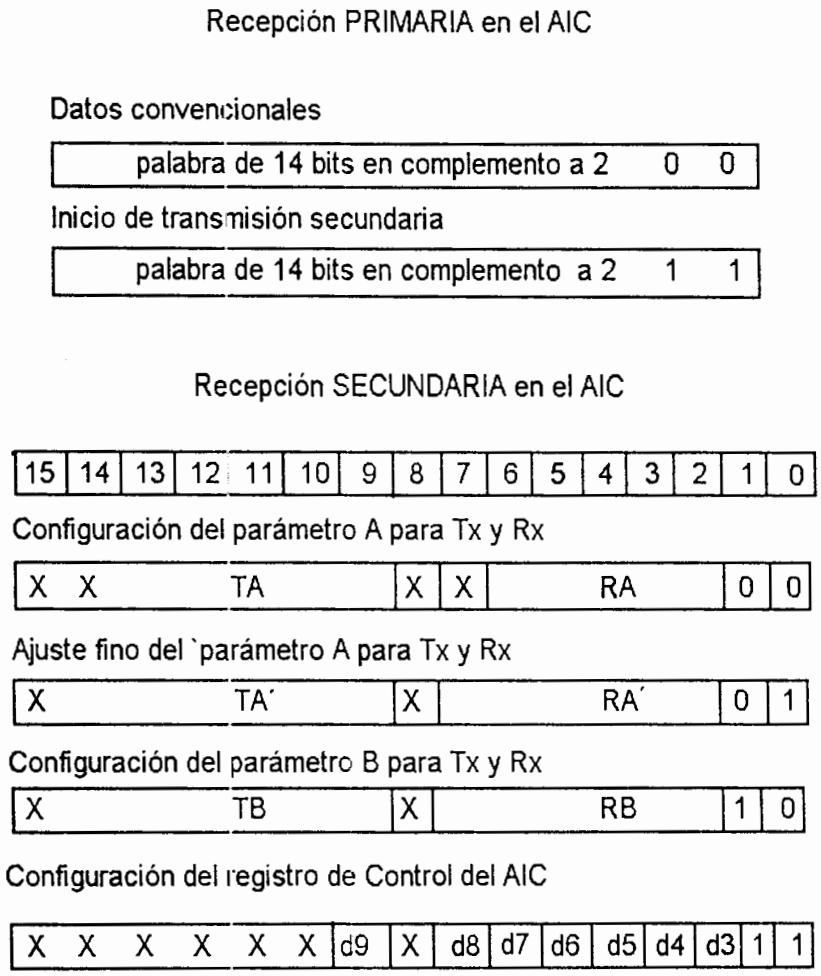


Figura 5.3. Formatos de transmisión de datos hacia el AIC

A continuación se muestra la secuencia de inicialización del AIC para configurarlo a una tasa de muestreo de 8 kHz:

```
aicreset:
   ldi     2,iof              ; configura XF0 para que sea salida con 0,
                              ; con lo que se pone en reset al AIC
   ldi     @t0_ctladdr,ar0    ; carga AR0 con la dirección del TIMER 0
                              ; GLOBAL CONTROL REG
   ldi     1,r1               ; con 1, tclk0 será (H)1/2
   sti     r1,*+ar0(8)        ; pone 1 en el TIMER 0
   ldi     @t0_ctlinit,r1     ; carga R1 con la configuración del TIMER 0
   sti     r1,*ar0            ; pone la configuración en TIMER GLOBAL
                              ; CONTROL REG
```

```
ldi     @p0_addr,ar0        ; carga en AR0 la dir. del GLOBAL CONTROL
                            ; REG del puerto serial 0
ldi     111h,r1             ; carga en R1 la configuración del p.s. 0
sti     r1,*+ar0(2)         ; carga la configuración en S.P. 0 Tx PORT
                            ; CONTROL
sti     r1,*+ar0(3)         ; carga la config. en S.P. 0 Rx PORT CONTROL
ldi     @p0_global,r1       ; carga R1 con la config. para p.s. 0
sti     r1,*ar0             ; escribe configuración en el GLOBAL CONTROL
                            ; REG del p.s. 0
xor     r1,r1               ; borra el contenido de R1
sti     r1,*+ar0(8)         ; pone un 0 en el reg. de Tx del p.s. 0
rpts    99                  ; espera a que hayan ocurrido 50 ciclos
nop                         ; del timer š para que se estabilice ?
ldi     6,iof               ; pone a XF0 en 1, saca del reset al AIC
                            ; y comienza a enviarle su configuración



call    wait_transmit_0     ; polea a ver si se ha transmitido algo
ldi     3,r1                ; guarda un 3 en R1
sti     r1,*+ar0(8)         ; escribe el 3 en el S.P 0 Tx DATA
call    wait_transmit_0     ; espera a que se haya transmitido
ldi     1a34h,r1            ; carga la palabra de config. del reg de
                            ; control del AIC
sti     r1,*+ar0(8)         ; envía dicha palabra al AIC
ldi     *+ar0(12),r1        ; lee lo que haya en el S.P. 0 Rx DATA
call    wait_transmit_0     ; espera a que esté listo el puerto
ldi     3,r1                ; ahora, siguiendo el mismo protocolo
sti     r1,*+ar0(8)         ; vuelve a mandar un 3
call    wait_transmit_0     ; espera a que se vacíe el puerto
ldi     2a7h,r1             ; carga la config. de la frec. de muestreo
sti     r1,*+ar0(8)         ; la envía
ldi     *+ar0(12),r1        ; lee lo que haya en la recepción
xor     if,if               ; borra todas las banderas de interrupción
or      @enbl_sp0_r,ie      ; habilita el P.S. 0
rets


wait_transmit_0:
          xor     if,if             ; borra todas las banderas de interrupc
wloop:    tstb    10h,if            ; revisa la bandera de Tx efectuada
          bz      wloop
          rets
```

5.7 Software asociado: el depurador de código

El depurador de código es una poderosa herramienta de software que aprovecha la característica de emulación que posee la tarjeta. Mediante él es posible realizar una emulación en la cual, a diferencia de una simulación, se realiza la ejecución de las instrucciones realmente en el interior del microprocesador, a la vez que se obtienen las ventajas típicas de los simuladores, como son:

- Realizar la ejecución paso a paso, o utilizando *breakpoints.*
- Observar los cambios en diferentes variables a la vez.
- Poder modificar aleatoriamente el contenido de las variables durante la ejecución.
- Utilizar archivos como entrada o salida de datos.
- Definir "macros"con los comandos más usados.
- Realizar el rastreo de variables en un sistema de múltiples ventanas.

Al igual que el microprocesador al cual apoya, este programa está orientado hacia la programación en lenguaje C, por lo que sus instrucciones suelen tener similitudes principalmente en la sintaxis con la forma de expresar operaciones y direccionamientos en dicho lenguaje.

El depurador se invoca al ejecutar el comando `evm30.exe` y por default se inicializa con la información contenida en `evminit.cmd`, si se desea se puede editar este archivo, o bien, invocar el programa con otro archivo de comandos mediante la sentencia

```
evm30 -t archivo.cmd
```

5.7.1 Comandos del depurador de código

A continuación se resumen las instrucciones más usuales dentro del ambiente del depurador de código.

Instrucción	*Función*	*Ejemplo*
asm	Rastrear programa en ensamblador	asm
c	Rastreo en c (modo por default)	c
mix	Modo mixto	mix

Tabla 5.6. Instrucciones para cambiar de modo de programación

Tecla	*Función*
$< F2 >$	Último comando
$< F3 >$	Extiende DISSASEMBLY y CPU ocultando FILE y CALLS
$< F4 >$	Cerrar la ventana actual de una estructura
$< F5 >$	Equivale a RUN
$< F6 >$	Brinca entre ventanas
$< F8 >$	Ejecuta paso a paso
$< F9 >$	Entrar a una subestructura (ver manipular datos)
$< F10 >$	Siguiente instrucción
$< TAB >$	Recorre el buffer de comandos
$< ESC >$	Salir

Tabla 5.7. Funciones de algunas teclas

Instrucción	*Función*	*Ejemplo*
zoom	Maximiza la ventana actual	zoom
move	Mueve la ventana actual	move
win	Cambia de ventana	win CPU, win FIL, win CAL win MEM, win COM, win DIS
size	Cambia dimensiones de la ventana	size

Tabla 5.8. Instrucciones para manejo de ventanas

Instrucción	*Función*	*Ejemplo*
system	Llamadas al DOS en modo SHELL	system, system "dir/w "[,0][,1] con 1 espera un ENTER con 0 regresa de inmediato
exit	Salir del modo SHELL	exit
take	Toma un archivc de comandos (.cmd)	take arch[.cmd][,0] el 0 inhibe el eco de los comandos
cd/chdir	Cambia el directorio de trabajo	cd user
cls	Borra la ventana de la ventana COM	cls
alias	Definir un macro	alias macro1,"comando1;comando2"
	Ejecutar el macro	macro1
	Mostrar macros existentes	alias
unalias	Cancelar el macro	unalias macro1
dir	Listar el directorio actual	dir
use	Nombrar directorios de fuentes adicionales	use otrodir
reset	Resetear el sistema (la EVM)	reset
restart	Continuar ejecución del programa	restart
quit	Salir del depurador	quit

Tabla 5.9. Instrucciones para ejecutar tareas del sistema

Instrucción	*Función*	*Ejemplo*
load	Cargar archivo ejecutable (.out)	load c:file[.out]
file	Cargar un archivo ASCII en FILE	file myfile.ext
func	Mover el apuntador de la ventana FILE a una función específica	func función
dasm	Desplegar código a partir de una función específica	dasm función
	Desplegar código a partir de una función con fuente en C o ASM	dasm 0x100200
addr	Desplegar código a partir de una función con fuente en C	addr[funcion][direcc]
reload	Cargar archivo .out sin su tabla de símbolos	reload file1[.out]
sload	Cargar sólo la tabla de símbolos de un archivo .out	sload file1[.out]
	Modificar una instrucción en lenguaje ensamblador	patch 0x100200[,instrucción]
calls	Reabrir la ventana CALLS	calls

Tabla 5.10. Instrucciones para cargar y desplegar programas

Instrucción	*Función*	*Ejemplo*
wa	Abrir ventanas para rastrear variables	wa variable[,etiqueta] (aparece la ventana watch)
	Observar el contenido de una localidad	wa *0x100300
	Observar una variable	wa variable [,etiqueta][o,x,i,etc.]
	Observar como entero	wa i
	Observar como float	wa f
	Observar como double	wa d
	Observar con etiqueta	wa i,NEW i
wd	Borrar una variable de la ventana	wd N (N es el renglón)
wr	Borrar toda la ventana WATCH	wr
setf	Cambiar la forma de desplegar tipo de datos	setf int, x (los enteros se verán como hexa)
	Reestablecer el despliegue normal	setf*
	Listar los tipos de datos y su modo de despliegue actual	setf
disp	mostrar en la ventana un dato:	
	Mostrar una estructura	disp estructura
	Mostrar en detalle un elemento de la estructura	hacer <click> en el elemento
	Cerrar la ventana actual	presionar $< F4 >$
	Entrar a una "subestructura"	presionar $< F9 >$
	Conmutar entre "subestructuras" "hijas"	$< CTRL >< PgUp >$ $< CTRL >< PgDn >$
?	Evaluar y desplegar el resultado de una expresión	
	Evaluar una loc. de memoria	?*0x100200
	Evaluar loc. de memoria, forzando el despliegue como un tipo de datos	disp *(float *)0x100200
	Evaluar el contenido de una variable	? variable[c,f,o,etc.]
mem	Desplegar a partir de una localidad en la ventana MEMORY	mem 0x100200, pc [*,c,d,e,f,x,o,p,u]
	Saber el valor de un símbolo	mem &símbolo
eval	Evaluar una expresión	eval -R3
whatis	Averiguar el tipo de una variable o una función	whatis variable, whatis función

Tabla 5.11. Instrucciones para desplegar y manipular datos y variables

Instrucción	*Función*	*Ejemplo*
ba	Marcar un punto de prueba	ba [dirección,función o etiqueta]
br	Desmarcar todos los puntos	br
bd	Desmarcar un punto	bd 0x100200
bl	Listar los puntos marcados	bl
clk	Examinar el contenido de CLK, que cuenta ciclos de reloj entre breakpoints	? clk

Tabla 5.12. Instrucciones para utilizar puntos de prueba (breakpoints)

Instrucción	*Función*	*Ejemplo*
sconfig	Cambiar archivo de configuración de la pantalla	sconfig archivo[.clr]
prompt	Cambiar el prompt de la línea de comandos	prompt miprompt
ssave	Salvar en archivo la configuración	ssave archivo.ext
scolor	Modificar atributos de ventana	scolor menu-bar, yellow

Tabla 5.13. Instrucciones para personalizar la pantalla

Instrucción	*Función*	*Ejemplo*
ma	Dar de alta un bloque de memoria	ma 0x100200,0x0400,RAM/ROM
mc	Conectar el bloque a un archivo de entrada	mc 0x100200,filein.ext,READ
mc	Conectar el bloque a un archivo de salida	mc 0x100200,fileout.ext,WRITE
md	Dar de baja un bloque de memoria	md 0x100300
mi	Desconectar un puerto de memoria	mi 0x100200,READ
LDI	Leer de un puerto en ensamblador	LDI@0x100200,R0
ml	Desplegar configuración actual de memoria	ml
map	Habilitar/deshabilitar el mapeo en memoria	map ON/OFF
mr	Reinicializar todo el mapa de memoria	mr
fill	Llenar un bloque de memoria con un valor	fill 0x100200,0x100,0xffffffff
ms	Salvar un segmento de código como un programa .coff	ms dir-de-inicio, longitud, archivo.ext

Tabla 5.14. Instrucciones para manejo de memoria

Sólo se permite asociar a un puerto un archivo de entrada y otro de salida. Un archivo puede tener asociados múltiples puertos.

Instrucción	*Función*	*Ejemplo*
step	Recorrer N sentencias en ensamblador	step N
cstep	Recorrer N sentencias en lenguaje C	cstep N
next	Recorrer N sentencias en ensamblador pero sin entrar a las subrutinas	next N
cnext	Recorrer N sentencias en C, pero sin entrar a las subrrutinas	cnext N
run	Ejecutar un programa	run
runf	Ejecutar un programa sin la intervención del puerto emulador (la EVM ejecuta libremente)	runf
halt	"Conecta"de nuevo el puerto de emulación a la EVM	halt
go	Ejecutar una subrutina	go subrrutina
runb	Ejecutar hasta el siguiente breakpoint contando ciclos de CPU	runb (benchmarking)
ret	Ejecutar el código de la función actual en C y regresar cuando termina	ret

Tabla 5.15. Instrucciones para depuración de programas

Capítulo 6

Realización de filtros digitales con TMS320C30

Para la realización de filtros digitales con TMS320C30 necesitamos algunas instrucciones que nos permitan multiplicar, sumar dos números, retardar la señal, etc. En este capítulo se explican algunas de ellas y al final se presenta un programa completo en ensamblador del filtro FIR.

6.1 Instrucciones del TMS320C30

6.1.1 MPYF3-Multiplicación en punto flotante

Sintaxis

Directo: MPYF3 $src2, src1, dst$
Indirecto: MPYF3 $*+AR2(IR0), src1, dst$

Ejecución

$$src1 \times src2 - - > dst$$

El contenido del registro src1 se multiplica por el contenido de src2 y el resultado se guarda en dst.

Ejemplo 1

MPYF3 $R0, R7, R1$

Antes del comando:	Después del comando:
R0=057B400000h=6.281250e+01	R0=057B400000h=6.281250e+01
R7=0733C00000h=1.79750e+02	R7=0733C00000h=1.79750e+02
R1=0	R0=0D306A3000h=1.12905469e+04

Ejemplo 2

MPYF3 $*+AR2(IR0), R7, R2$

Antes del comando:	Después del comando:
AR2=809800h	AR2=809800h
IR0=12AH	IR0=12AH
R7=057B400000h=6.281250e+01	R7=057B400000h=6.281250e+01
R2=0h	R0=0D09E4A000h=8.82515625e+03
Los datos en 80992Ah=1.4050e+02	Los datos en 80992Ah=1.4050e+02

6.1.2 Instrucciones en paralelo MPYF3||ADDF3

En ensamblador existen instrucciones en paralelo que se realizan en un ciclo de reloj. El símbolo para las instrucciones que se ejecutan en paralelo es ||. Se pueden ejecutar estas instrucciones para multiplicación y guardar el resultado en un registro, o multiplicar y sumar en un ciclo de reloj.

Sintaxis

Directo: MPYF3 $srcA, srcB, dst1$
|| ADDF3 $srcC, srcD, dst2$

Indirecto: MPYF3 $*AR2(1)++, *--AR1(IR0), R0$
|| ADDF3 $R5, R7, R3$

Ejecución

$srcA \times srcB -- > dst1$
$||\ srcC + srcD -- > dst2$

El contenido del registro srcA se multiplica por el contenido de srcB y el resultado se guarda en dst1. Al mismo tiempo se suma el contenido de srcC con el de srcD y el resultado se guarda en el registro dst2.

Ejemplo

```
    MPYF3  *AR5++(1),*-(IR0),R0
||  ADDF3  R5,R7,R3
```

Antes del comando:	Después del comando:
AR5=8098C5h	AR5=8098C6h
AR1=8098A8h	AR1=8098A4h
IR0=4h	IR0=4h
R0=0h	R0=0467180000h=2.88867188e+01
R5=0733C00000h=1.79750e+02	R5=0733C00000h=1.79750e+02
R7=070C800000h=1.4050e+02	R7=070C800000h=1.4050e+02
R3=0h	R3=0820200000h=3.20250e+02
Los datos en 8098C5h=1.2750e+02	Los datos en 8098C5h=1.2750e+01
Los datos en 8098A4h=2.2500e+00	Los datos en 8098A4h=2.2500e+00

6.1.3 Multiplicar y cargar en paralelo MPYF3||STF

Sintaxis

Directo:	MPYF3	$src2, src1, dst$
||	STF	$src3, dst2$
Indirecto:	MPYF3	$*--AR2(1), R7, R0$
||	STF	$*AR0----(IR0)$

Ejecución

$src1 \times src2 --> dst1$
$\|\| \; src3 --> dst2$

El contenido del registro src1 se multiplica por el contenido src2 y el resultado se guarda en dst1. Al mismo tiempo el contenido src3 se guarda en el registro dst2.

Ejemplo

```
      MPYF3  *-AR2(1),R7,R0
||    STF    R3,*AR0- -(IR0)
```

Antes del comando:	Después del comando:
AR2=80982Bh	AR2=80982Bh
R7=057B400000h=6.281250e+01	R7=057B400000h=6.281250e+01
R0=0h	R0=0D09E4A000h=8.82515625e+03
R3=086B280000h=4.7031250e+02	R3=086B280000h=4.7031250e+02
AR0=809860h	AR0=809858h
IR0=8h	IR0=8h
Los datos en 80982Ah=1.4050e+02	Los datos en 80982Ah=1.4050e+02
Los datos en 809860h=0h	Los datos en 809860h=4.703125e+02

6.1.4 Multiplicar y restar en paralelo MPYF3||SUBF3

Sintaxis

$$\begin{array}{rll} \text{Directo:} & \text{MPYF3} & srcA, srcB, dst1 \\ || & \text{SUBF3} & srcC, srcD, dst2 \end{array}$$

$$\begin{array}{rll} \text{Indirecto:} & \text{MPYF3} & R5, *++AR7(IR1), R0 \\ || & \text{SUBF3} & R7, *AR3----(1), R2 \end{array}$$

Ejecución

$$\begin{array}{l} srcA \times srcB -- -- > dst1 \\ || \; srcD - srcC - - > dst2 \end{array}$$

El contenido del registro srcA se multiplica por el contenido del registro srcB y el resultado se guarda en dst1. Al mismo tiempo el contenido del registro srcC se sustrae del contenido del registro srcD y el resultado se guarda en el registro dst2.

Ejemplo

$$\begin{array}{rll} & \text{MPYF3} & *++AR7(IR1), R5, R0 \\ || & \text{SUBF3} & R7, *AR3--(1), R2 \end{array}$$

Antes del comando:	Después del comando:
R5=034C000000h=1.2750e+01	R5=034C000000h=1.2750e+01
AR7=809904h	AR7=80090Ch
IR1=8h	IR1=8h+01
R0=0h	R0=0467180000h=2.88867188e+01
R7=0733C00000h=1.79750e+02	R7=0733C00000h=1.79750e+02
AR3=8098B2h	AR3=8098B1h
R2=0h	R2=05E3000000h=-3.9250e+01
Los datos en 80990Ch=2.50e+00	Los datos en 80990Ch=2.250e+00
datos en 8098B2h=1.4050e+02	Los datos en 8098B2h=1.4050e+02

6.1.5 Cargar en paralelo LDF||LDF

Sintaxis

$$\begin{array}{rll} \text{Directo:} & \text{LDF} & src2, dst2 \\ || & \text{LDF} & src1, dst1 \end{array}$$

$$\begin{array}{rll} \text{Indirecto:} & \text{LDF} & *--AR1(IR0), R7 \\ || & \text{LDF} & *AR7++(1), R3 \end{array}$$

Ejecución

$$src2 - - > dst2$$
$$|| \; src1 - - > dst1$$

El contenido del registro src2 se guarda en dst2 y al mismo tiempo el contenido de scr1 se guarda en el dst1.

Ejemplo

$$\begin{array}{ll} & \text{LDF} \quad *--AR1(IR0), R7 \\ || & \text{LDF} \quad *AR7++(1), R3 \end{array}$$

Antes del comando:	Después del comando:
AR1=80985Fh	AR1=809857h
IR0=8h	IR0=8h
R7=0h	R7=070C800000h=1.4050e+02
AR7=80988Ah	AR7=80988Bh
R3=0h	R3=6.281250e+01
Los datos en 809857h=1.4050e+02	Los datos en 809857h=1.4050e+02
Los datos en 80988Ah=6.281250e+01	Los datos en 80988Ah=6.281250e+01

6.1.6 Instrucciones para entrada y salida

Entrada del número

La muestra en la entrada puede ser obtenida desde el convertidor analógico digital (ADC) mediante las siguientes instrucciones:

$$\begin{array}{ll} \text{LDI} & @IN\_ADDR, AR2 \\ \text{FLOAT} & *AR2, R3 \end{array}$$

La dirección de la entrada (IN_ADDR) está cargada en el registro AR2. El número tipo entero obtenido desde ADC se convierte en el número flotante y éste es guardado en el registro extendido R3.

Salida del número

El número puede mandarse al convertidor digital-analógico (DAC) con las instrucciones:

```
LDI  @OUT_ADDR,AR3
FIX  R0,R1
STI  R1,*AR3
```

La dirección de salida está cargada en el registro AR3. El valor R0 tipo punto flotante es convertido en número entero y es cargado en R1. Este número se guarda en la dirección de salida que está determinada mediante el registro AR3.

6.1.7 Instrucción de repetición

El comando RPTS permite repetir la instrucción que sigue y también el bloque de las instrucciones que podemos repetir con la instrucción RPTB. Utilizando la instrucción RPTB, la dirección inicial se guarda en el registro de repetición RS (*Repeat Start register*). La última dirección que se debe repetir se guarda en un registro especial RE (*Repeat End address register*). En el registro especial RC (*Repeat Counter register*) se guarda el número de repeticiones que se desean.

Ejemplo 1

```
                LDI     9,RC
                RPTB    END_ADDR
                CALL    FILTER
                FIX     R0
END_ADDR        STI     R0,*AR3
```

El contador de la repetición está cargado con el número 9, eso significa que el bloque de las instrucciones se repetirá 10 veces, con las cuales se llama al subprograma FILTER. La dirección donde empieza el bloque se guarda en el registro RS y la última dirección se carga en el registro RE. El contenido de R0 que calcula la subrutina FILTER se convierte en un número tipo entero, el cual se guarda después en la dirección que señala el registro AR3.

Ejemplo 2

```
        LDF     0,R0
        LDI     29,AR2
        RPTS    AR2
        MPYF    *AR0++,*AR1++,R0
||      ADDF    R0,R2,R2
        ADDF    R0,R2
```

En este ejemplo se ejecutan en paralelo las instrucciones MPYF y ADDF. A consecuencia del trabajo en paralelo, el microcontrolador TMS320C25 realiza 33 millones de operaciones por segundo. El símbolo paralelo || junto con la instrucción ADDF indican que se realizan al mismo tiempo las instrucciones MPYF y ADDF.

Primeramente, se carga al registro R0 con cero y el número tipo entero (LDI) se carga al registro auxiliar AR2. En el registro AR2 se almacena el número 29. Entonces 29+1 veces se repiten las instrucciones MPYF y ADDF. Los números en los registros AR1 y AR2 se incrementan después de la multiplicación de los números guardados en las direcciones que señalan los registros auxiliares AR1 y AR0, y el resultado se carga al registro R0. Al mismo tiempo se suman los números guardados en los registros R0 y R2, y el resultado se guarda en el registro R2. Este cálculo se repite 30 veces hasta que el contenido en el registro auxiliar AR2 es igual a cero y se cumple la última instrucción. En esta instrucción se suma el contenido de R0 y R2, y el resultado se guarda en el registro R2. La última instrucción se cumple sólo una vez y se puede escribir también en la forma ADDF R0, R2, R2.

6.1.8 Multiplicación con buffer circular

La multiplicación con buffer circular se utiliza muy a menudo si queremos, por ejemplo, calcular la convolución, correlación o realizar los filtros con la respuesta finita. El símbolo para buffer circular es %.

Ejemplo

```
LENGTH  .SET   30
        LDI    LENGTH,BK
        LDF    0,R0
        RPTS   LENGTH-1
        MPYF   *AR0++%,*AR1++%,R0
||      ADDF   R0,R2,R2
        ADDF   R0,R2,R2
```

En el programa, primeramente, el número 30 se guarda en la variable LENGTH. Esta variable señala la longitud del bufer circular. Después, la longitud del bufer se guarda en un registro especial BK. En el registro R0 se guarda cero. Las instrucciones MPYF y ADDF se repiten treinta veces LENGTH-1=29. Las direcciones guardadas en los registros auxiliares AR0 y AR1 se incrementan siempre en uno hasta alcanzar el tope del bufer circular. En este ejemplo utilizamos dos bufers circulares para AR0 y AR1 que se marcan si guardamos en el registro auxiliar AR0 la dirección de la variable X y en el registro auxiliar AR1 la dirección de variable Y. Después de 30 repeticiones tenemos en el registro auxiliar AR0 la dirección X+29 y en registro auxiliar AR1 la dirección Y+29.

Después de terminar la multiplicación en paralelo con la adición en los registros AR0 y AR1, se guarda la dirección X y Y.

6.2 Representación de los números en TMS320C30

En los registros de TMS320C30 podemos guardar los números de punto flotante con la precisión simple o extendida. Los registros para los números de punto flotante con precisión simple y precisión extendida se presentan en la figura 6.1.

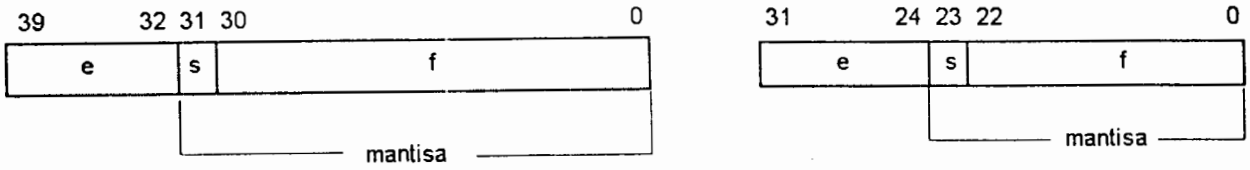


Figura 6.1. Formatos de los números con punto flotante: a) Precisión simple, b) Precisión extendida

El número que nos da el convertidor ADC es tipo entero y lo convertimos con la instrucción FLOAT en el formato con punto flotante. Antes de mandar el número al convertidor DAC es necesario convertirlo de punto flotante a tipo entero. La conversión desde la precisión extendida a la precisión simple se realiza automáticamente. El número expresado en

la forma extendida se puede guardar con 40 bits en los registros R0-R7. El resultado después de la multiplicación está expresado con la precisión extendida. El formato de número con precisión extendida está representado con un exponente (E) de 8 bits (bits 32-39), un bit de signo, (S, bit 31) y la parte fraccionaria (F, bits 0-30). En la figura 6.1 se muestran los formatos para los números expresados con la precisión simple y extendida. El número para el formato con punto flotante está en el rango $=-2^{128}$ hasta 2^{128}.

Donde el número N está representado en la forma:

$$N := \begin{cases} 01.F \times 2^E, & si\ el\ bit\ S = 0 \\ \\ 10.F \times 2^E, & si\ el\ bit\ S = 1 \end{cases} \tag{6.1}$$

Ejemplo 1

Convertir el número 07F38000h expresado con 32 bits a número decimal:

```
N=07F38000h = 0000 0111 1111 0011 1000 0000 0000 0000b
              ----E---- S-------------F-------------
```

El bit S=1 y por eso, mediante la ecuación 6.1, podemos escribir

```
N=10.111 0011 1000 0000 ...exp+07
N=10 111 0011. 1000 0000 ...exp+00
```

El bit más significativo de F es uno. Por lo que el número es negativo. Se debe complementar a dos el número N, es decir, sumar al complemento de N un bit.

```
N= 0 1 0 0 0 1 1 0 0, 0  1  1  1 ...
                                1
-------------------------------
N= 0 1 0 0 0 1 1 0 0, 1  0  0  0
  (8 7 6 5 4 3 2 1 0 -1 -2 -3 -4)
```

$$N_d = 2^{-1} + 2^2 + 2^3 + 2^7 = -140.5exp + 00 = -1.405e^{02}$$

Ejemplo 2

Convertir el número FC200000 hexadecimal expresado con 32 bits a número decimal:

```
N=FC200000h = 1111 1100 0010 0000 0000 0000 0000 0000b
              ----E---- S-------------F-------------
```

El bit S=0 y por eso, mediante la ecuación (6.1), podemos escribir

```
N=01.010 exp-04
```

El exponente es negativo porque el bit más significativo es uno. Entonces, el exponente E se calcula si los ceros son convertidos en uno y los unos en ceros, y sumamos con uno. Se tiene

```
exp = 0000 0011
                1
     ----------
exp = 0000 0100 =4
```

Entonces obtenemos

$$N = .0001\ \ 0100\ \ 0000... = 2^{-4} + 2^{-6} = \frac{5}{64} = 0.0781$$

Ejemplo 3

Convertir número hexadecimal 070C8000 expresado con 32 bits a número decimal:

```
N=07|0C8000h = 0000 0111 0000 1100 1000 0000 0000 0000b
  E    F       ---E------S--------------F-------------
```

El bit más significativo de E y F es cero, por lo tanto el número y el exponente son positivos. El bit S=0 y por eso, mediante la ecuación (6.1), podemos escribir

5mm

```
N=0 1. 0 0 0 1 1 0 0  1 0 0 0 exp+07
N=0 1  0 0 0 1 1 0 0. 1 exp+00
   (7  6 5 4 3 2 1 0  -1)
```

Calculando el número N en la forma decimal, obtenemos

$$N_d = 2^{-1} + 2^2 + 2^3 + 2^7 = 140.5\ e^{+00} = 1.405\ e^{+02}$$

Ejemplo 4

Convertir el número hexadecimal 0733C000 expresado con 32 bits a número decimal:

```
N=07|33C000h = 0000 0111 0011 0011 1100 0000 0000 0000b
  E    F        ---E------S--------------F-------------
```

El bit más significativo de E y F es cero, por lo tanto el número y el exponente son positivos. El bit S=0 y por eso, mediante la ecuación (6.1), podemos escribir

```
N=0 1. 0 1 1 0 0 1 1  1 1 0 0 exp+07
N=0 1  0 1 1 0 0 1 1. 1 1 exp+00
   (7  6 5 4 3 2 1 0 -1 -2)
```

Calculando el número N en la forma decimal, obtenemos

$$N_d = 2^{-1} + 2^{-2} + 2^0 + 2^1 + 2^4 + 2^5 + 2^7 = 179,75\ e^{+00} = 1.7975\ e^{+02}$$

Ejemplo 5

Convertir el número hexadecimal 05E30000 expresado con 32 bits a número decimal:

```
N=05|E30000h = 0000 0101 1110 0011 0000 0000 0000 0000b
  E    F        ---E------S--------------F-------------
```

El bit más significativo de E es cero, por lo tanto, el exponente es positivo. El bit más significativo de F es uno, por lo que el número N es negativo. Bit S=0 y por eso, mediante la ecuación (6.1), podemos escribir

```
N=1 0. 1 1 0 0 0  1  1  exp+05
N=1 0  1 1 0 0 0. 1  1  exp+00
( 6 5  4 3 2 1 0 -1 -2)
```

Para obtener el número negativo calculamos el complemento del número N y sumamos con uno. Calculando obtenemos:

```
N=0 1 0 0 1 1 1 . 0 0
                  1
----------------------
N=0 1 0 0 1 1 1 . 0 1
  (  5 4 3 2 1 0  -1 -2)
```

Y el número N en la forma decimal tiene la forma

$$N_d = 2^{-2} + 2^0 + 2^1 + 2^2 + 2^5 = -39.25\ e^{+00}$$

6.3 Programas en ensamblador

En esta parte presentamos los programas para multiplicación de dos series de números y un programa para multiplicar una matriz con un vector.

6.3.1 Multiplicación de dos series de números

El programa en ensamblador para TMS320C30 se compone de un programa principal que se llama MULT.ASM y la subrutina MULTSUB.ASM.

```
/* MULT.ASM - MULTIPLICACION DE DOS SERIES */

          .TITLE "MULT.ASM"          ;El título del programa
          .GLOBAL BEGIN,MULT         ;Los símbolos REF/DEF
          .DATA                      ;Se ensambla a la sección de datos
H_ADDR    .WORD  HN                  ;La primera dirección del vector HN
X_ADDR    .WORD  XN                  ;La primera dirección del vector YN
IO_OUT    .WORD  804001H
HN        .FLOAT  1,2,3,4            ;Los números del vector HN
XN        .FLOAT  2,3,4,5            ;Los números del vector XN
          .TEXT                      ;Se ensambla a la sección de texto
BEGIN      LDP        H_ADDR         ;Se inicializa la página de datos
           LDI   @H_ADDR,AR0         ;La dirección de HN en AR0
           LDI   @X_ADDR,AR1         ;La dirección de XN en AR1
           LDI   @IO_OUT,AR2         ;AR2 se configura como la salida
           LDI   3,RC                ;El contador de repetición RC=3
           CALL  MULT                ;Salto a subprograma MULT
           STI   R2,*AR2             ;El resultado a la salida IO_OUT
WAIT       BR    WAIT                ;Espera
```

Cada serie HN y XN tiene cuatro números. La serie HN tiene los números 1, 2, 3, 4 y la serie XN tiene los números 2, 3, 4, 5. El programa principal MULT.ASM llama una subrutina MULT en el programa MULTSUB.ASM.

```
/* MULTISUB.ASM - SUBRUTINA PARA MULTIPLICACION /*

          .TITLE    "MULTISUB.ASM"     ;El título
          .GLOBAL    MULT              ;El símbolo REF/DEF
          .TEXT                        ;Ensambla a la sección de texto
MULT       LDF       0,R0              ;R0=0
           LDF       0,R2              ;R2=0
           RPTS      RC                ;Se ejecuta instrucción 3 veces
           MPYF      *AR0++,*AR1++,R0  ;(AR0)*(AR1)->R0
        ||  ADDF     R0,R2             ;El resultado se suma con R2
```

```
        ADDF    R0,R2           ;La última suma se guarda al R2
        RETS                    ;Se regresa desde el subprograma
        .END                    ;Fin del programa
```

Se necesita escribir el programa MULT.CMD para enlazador (linker) con el propósito de enlazar los programas MULT.ASM, MULTSUB.ASM, así como configurar la memoria y las direcciones de entrada y de salida.

```
/* MULT.CMD EL PROGRAMA PARA ENLAZADOR */
/* CON PROPOSITO DE ENLAZAR LOS PROGRAMAS */

        -E  BEGIN               /*Se especifica el inicio  */
        MULT.OBJ                /*El programa principal MULT*/
        MULTSUB.OBJ             /*La subrutina MULTSUB      */
        -O MULT.OUT             /*Instrucción para ligador  */
MEMORY
{
VECS:  org=0           len=0x40
SRAM:  org=0x40        len=0x3FC0       /*BUS de (16K)     */
RAM:   org=0x809800    len=0x800        /*RAM interna de 2K*/
IO:    org=0x804000    len=0x2000       /*IOSTRB de 8K     */
}
SECTIONS
{
 .data:   {} >  SRAM              /*Sección de datos en SRAM    */
 .text:   {} >  SRAM              /*Sección del texto en SRAM   */
 .cinit:  {} >  SRAM              /*Inicialización de las tablas */
 .stack:  {} >  RAM               /*Sistema del STACK       */
 .bss:    {} >  RAM               /*Sección de BSS in RAM   */
 vecs:    {} >  VECS              /*Los vectores de RESET e INTERRUPT /*
IN_PORT      804000h   : {} > IO   /*La dirección de entrada */
OUT_PORT     804002h   : {} > IO   /*La dirección de salida   */
XN_BUFFER    ALIGN(64) : {} > RAM  /*El bufer circular en RAM */
}
```

Ahora bien, si vamos a trabajar con el simulador es necesario crear los módulos con extensión .obj y .out, entonces en nuestro caso son MULT.OBJ, MULTSUB.OBJ y MULT.OUT.

Para ensamblar es necesario teclear ASM30 MULT.ASM y ASM30 MULTSUB.ASM. En el archivo se crean los módulos MULT.OBJ y MULTSUB.OBJ. En este momento podemos llamar al enlazador con el propósito de obtener el módulo MULT.OBJ que ya podemos cargar en el simulador o si trabajamos con las señales reales, en el módulo de evaluación EVM. Para obtener el módulo MULT.OUT es necesario teclear LNK30 MULT.CMD.

Si tecleamos SIM3x MULT.OUT cargamos en el simulador MULT.OUT. En una ventana también podemos cargar el programa en ensamblador si tecleamos ALT+L después FILE

y al final MULT.ASM. En la pantalla se muestran las ventanas que están en la figura 6.2. Podemos correr paso a paso el programa, si se teclea F8 o también teclear la instrucción STEP 5000 o F5.

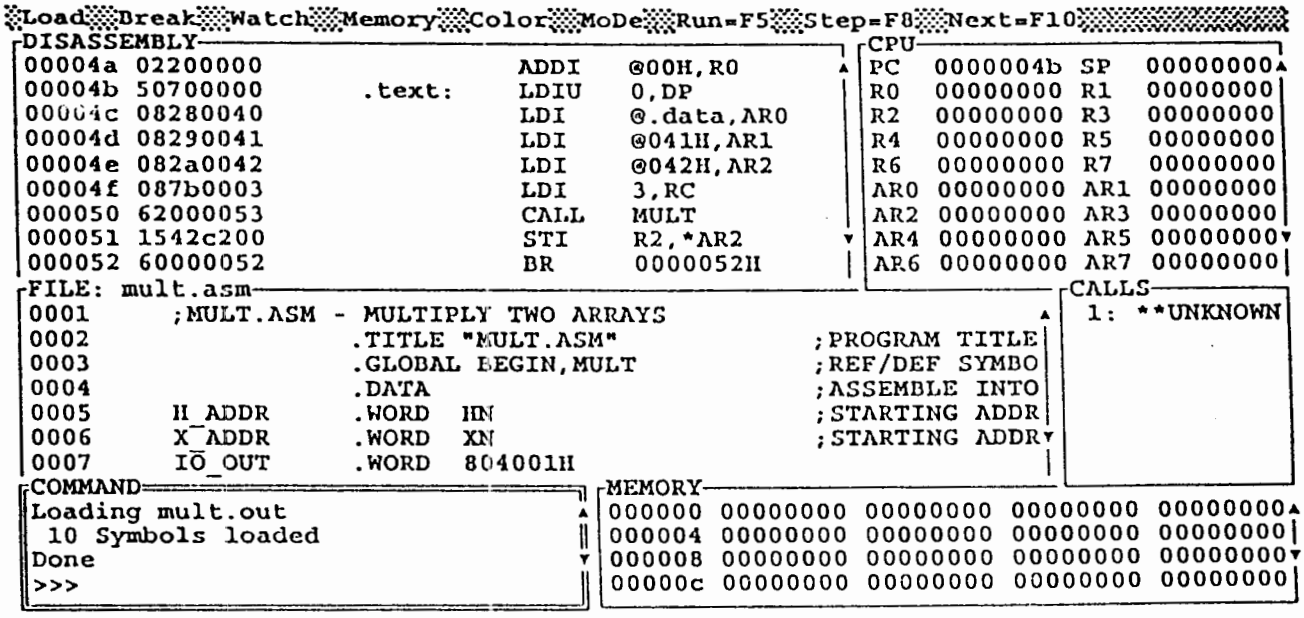


Figura 6.2. Pantalla del simulador SIM3x

6.3.2 Multiplicación de las matrices

Con cualquier editor de texto se puede escribir un programa para la multiplicación de una matriz con un vector.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \times \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 14 \\ 32 \\ 50 \end{bmatrix}$$

El programa en ensamblador se muestra a continuación.

```
;MATRIX.ASM-MULTIPLICA LA MATRIZ CON UN VECTOR EN ENSAMBLADOR
            .GLOBAL BEGIN
            .DATA                          ;Se ensambla en el sección de datos
A           .FLOAT 1,2,3,4,5,6,7,8,9       ;Valores del matriz  A
B           .FLOAT 1,2,3                   ;Valores del matriz  B
A_ADDR      .WORD A                        ;Dirección de los valores A
B_ADDR      .WORD B                        ;Dirección de los valores B
IO_OUT      .WORD 804001H                  ;Dirección del puerto de salida
```

```
            .TEXT                          ;Ensamblar a la sección de texto
BEGIN       LDI  @A_ADDR,AR0               ;AR0=Dirección de matriz A
            LDI  @B_ADDR,AR1               ;AR1=Dirección del vector B
            LDI  @IO_OUT,AR2               ;AR2=El puerto de salida
            LDI  3,R4                      ;3 -->R4
LOOPI       LDF  0,R0                      ;0 --> R0
            LDI  2,AR4                     ;2 --> AR4
LOOPJ       MPYF *AR0++,*AR1++,R1          ;A[I,J]*B[J] --> R1
            ADDF R1,R0                     ;En R0 se guarda el resultado
            DB   AR4,LOOPJ                 ;Decrementa AR4 y si AR4<0 salto
            FIX  R0                        ;Se convierte R0 al número entero
            STI  R0,*AR2                   ;El resultado al IO_OUT
            LDI  @B_ADDR,AR1               ;AR1 --> B_ADDR
            SUBI 1,R4                      ;Se decrementa R4
            BNZ  LOOPI                     ;Salto si R4<>0
WAIT        BR   WAIT                      ;Espera
```

En los registros auxiliares AR0, AR1 y AR2 se cargan las direcciones de las variables A, B y la dirección de puerto de salida. R4 se utiliza como contador de repetición. Tenemos R4=3, eso significa que multiplicamos tres números de la primera fila A con tres números en la columna de B y el resultado lo sumamos en R0. Las direcciones de la memoria de AR0 y AR1 se incrementan. El asterisco nos indica que se utiliza direccionamiento indirecto. La instrucción SUBI 1,R4 decrementa la variable R4. En caso de que R4=0 se salta a la etiqueta LOOPI. En el registro auxiliar AR1 se guarda la dirección donde principia el arreglo A. El resultado se guarda en el archivo OUTPUT.DAT. En el archivo OUTPUT.DAT deben aparecer los números 0000000e 00000020 00000032. Si a estos números los convertimos en decimales obtenemos 14, 32, 50. El archivo de comandos MATRIX.CMD o MULT.CMD lo podemos cambiar y utilizarlo para otros programas.

```
/*MATRIX.CMD              Archivo de los comandos              */
-E BEGIN                /*Se especifica el inicio              */
MATRIX.OBJ              /*Se crea archivo MATRIX.OBJ           */
-O MATRIX.OUT           /*Instrucción para crear MATRIX.OUT    */
```

6.3.3 Programa con el buffer circular

En este ejemplo se realiza un filtro con la respuesta finita FIR. El programa FIRFIL.ASM se presenta a continuación y el programa de comandos en el archivo FIRFIL.CMD.

```
; NOMBRE DE ARCHIVO: FIRFIL.ASM
;
; A continuación se observa
; cómo son organizados los datos en la memoria
; y cómo se cambia la memoria donde son guardadas las
; muestras de la señal x(n). Las dos últimas columnas
```

```
; muestran cómo funciona el buffering circular en el
; registro  AR1.
;
;     la respuesta      las muestras de      las muestras de
;      al impulso         la señal       la señal después buffering
;      +--------+       +------------+  +------------+
;      | h(N-1) |       | x[n-(N-1)] |  |    x(n)    |
;      +--------+       +------------+  +------------+
;      | h(N-2) |       | x[n-(N-2)] |  | x[n-(N-1)] |
;      +--------+       +------------+  +------------+
;          :                  :               :
;      +--------+       +------------+  +------------+
;      | h(1)   |       |   x(n-1)   |  |   x(n-2)   |
;      +--------+       +------------+  +------------+
;      | h(0)   |       |    x(n)    |  |   x(n-1)   +
;      +--------+       +------------+  +------------+
;
;       Si la subrutina filter es llamada por  primera vez,
;       la dirección de h(N-1) se guarda en el registro
;       auxiliar AR0,  y la dirección de x[n-(N-1)] en el
;       registro AR1.
;
        .global _main,fir
        .page
;
; Primeramente necesitamos definir el lugar para I/O en la memoria.
; Utilizamos el LINKER para definir las direcciones  de las secciones
; "in_port" y "out_port" en el mapa de la memoria.
;
in      .usect  "in_port",1  ;A/D la dirección de "input port"
out     .usect  "out_port",1 ;D/A la dirección  de "output port"
;
        .data
in_addr  .word  in           ;guardar 24-bit A/D dirección
out_addr .word  out          ;guardar 24-bit D/A dirección

x_n_addr .word  x_n+length-1 ;la dirección de x(n)
h_n_addr .word  h_n          ;la dirección de h(N-1)

scaler   .float  6.62        ;scaler for D/A output
;
                             ; Los coeficientes para el filtro FIR.
                             ; Podemos calcular en este ejemplo
                             ; cualquier tamaño de filtro hasta
                             ; N=1024.
;
h_n     .float   2.2579136E-03  ; h(41)
```

```
        .float   4.1378370E-04  ; h(40)
        .float  -6.3593970E-03  ; h(39)
        .float  -4.3735630E-03  ; h(38)
        .float   9.0539033E-03  ; h(37)
        .float   1.0372631E-02  ; h(36)
        .float  -6.4953700E-03  ; h(35)
        .float  -1.1626530E-02  ; h(34)
        .float   7.7737306E-04  ; h(33)
        .float   6.4344249E-04  ; h(32)
        .float  -2.8973380E-03  ; h(31)
        .float   2.1137551E-02  ; h(30)
        .float   2.6027328E-02  ; h(29)
        .float  -3.8419060E-02  ; h(28)
        .float  -7.2032840E-02  ; h(27)
        .float   2.9972621E-02  ; h(26)
        .float   1.2341397E-01  ; h(25)
        .float   1.3980616E-02  ; h(24)
        .float  -1.5107940E-01  ; h(23)
        .float  -7.9517490E-02  ; h(22)
        .float   1.3472794E-01  ; h(21)
        .float   1.3472794E-01  ; h(20)
        .float  -7.9517490E-02  ; h(19)
        .float  -1.5107940E-01  ; h(18)
        .float   1.3980616E-02  ; h(17)
        .float   1.2341397E-01  ; h(16)
        .float   2.9972621E-02  ; h(15)
        .float  -7.2032840E-02  ; h(14)
        .float  -3.8419060E-02  ; h(13)
        .float   2.6027328E-02  ; h(12)
        .float   2.1137551E-02  ; h(11)
        .float  -2.8973380E-03  ; h(10)
        .float   6.4344249E-04  ; h(9)
        .float   7.7737306E-04  ; h(8)
        .float  -1.1626530E-02  ; h(7)
        .float  -6.4953700E-03  ; h(6)
        .float   1.0372631E-02  ; h(5)
        .float   9.0539033E-03  ; h(4)
        .float  -4.3735630E-03  ; h(3)
        .float  -6.3593970E-03  ; h(2)
        .float   4.1378370E-04  ; h(1)
h_0     .float   2.2579136E-03  ; h(0)
;
length  .set    (h_0-h_n)+1     ; length = 42 (02Ah)
;
                                ; Al final es necesario reservar en la memori
                                ; RAM el lugar para los datos de las
                                ; muestras de entrada  x(n) hasta x[n-(N-1)]
```

```
;
        .bss    offset,3
x_n     .usect "x_n_buff",length
;
        .page
        .text
;
_main   LDP     in_addr
        LDI     @in_addr,AR5  ;A/D dirección -> AR5
        LDI     @out_addr,AR6 ;D/A dirección -> AR6
        LDI     length,BK     ;el tamaño (longitud) del filtro -> BK
        LDI     @x_n_addr,AR1 ;x(n) dirección -> AR1
        LDF     0.0,R0        ;0.0 -> R0
        RPTS    length-1
        STF     R0,*AR1--(1)% ;R0 -> AR1 (dirección en AR1 se decrementa)
;
loop    FLOAT   *AR5,R3       ;se lee nueva muestra de x(n)
        STF     R3,*AR1++(1)% ;R3 se guarda en el bufer circular
                              ;AR1 es ahora en x(n-(N-1))
        LDI     length-2,RC   ;length-2 se guarda en contador de repetición
        LDI     @h_n_addr,AR0 ;la dirección de h(n) -> AR0
        CALL    fir           ;se calcula el valor de salida
        BUD     loop
        MPYF    @scaler,R0
        FIX     R0,R1         ;R1 es la forma entera para la salida
        STI     R1,*AR6
        BR      loop          ;se salta a la etiqueta loop
;
; La subrutina FIR
; Las ecuaciones que se calculan
; y(n) = h(0)*x(n) + h(1)*x(n-1) +...+ h(N-1)*x(n-(N-1))
;
; LOS REGISTROS UTILIZADOS EN EL PROGRAMA:
;
;       carga   AR0     addr of h(N-1)
;       carga   AR1     addr of x(N-1)
;       carga   RC      filter length - 2 (N-2)
;       carga   BK      filter length (N)
;       call    fir
;
; REGISTROS UTILIZADOS COMO ENTRADA: AR0, AR1, RC, BK
; REGISTROS QUE SE MODIFICAN : R0, R2, AR0, AR1, RC
; REGISTRO QUE CONTIENE EL RESULTADO: R0
; LOS CICLOS DE EJECUCION : 11 + (N-1)
;
        .global fir
        .text
```

```
;
fir     mpyf3   *AR0++(1),*AR1++(1)%,R0
;                               ; R0 = h(N-1)*x(n-(N-1))
        ldf     0.0,R2          ; se inicializa R2
        rpts    RC              ; se inicializa el contador de la repetición
        mpyf3   *AR0++(1),*AR1++(1)%,R0
    ||  addf3   R0,R2,R2        ; la instrucción paralela
                                ; multiplica y suma
                                ; h(N-1-i)*x(n-(N-1-i))
        addf    R0,R2,R0        ; se suma el último producto
        retsu

/*************************************************************************/
/* FILFIR.CMD -  EL ARCHIVO DE COMANDOS PARA ENLAZADOR                   */
/*                                                                       */
/* Las instrucciones: lnk30 <obj archivo...>  -o <out archivo            */
/*                    -m <map archivo>  FILFIR.cmd                       */
/*                                                                       */
/*    Descripción: Este archivo trae las instrucciones para ligador y    */
/*                 compilador de TMS320C30 C                             */
/*************************************************************************/
-c                                      /* LINK UTILIZANDO C            */
-i c:\c30tools                          /* DIRECTORIO DE LIBRERIA       */
-l rts.lib                                                              */
-l bus.lib                              /* UTILITARIO DE BUS LIBRERIA   */
-l serial.lib                           /* SERIAL PUERTO LIBRERIA       */
-l timers.lib                                                           */
-l dma.lib                              /* UTILITARIO DE DMA LIBRERIA   */
                                                                        */
vectors.obj                             /* LOS VECTORES DE INTERRUPCION */
ints.obj                                /* LA RUTINA C DE INTERRUPCION  */

/* MAPA DE LA MEMORIA */

MEMORY
{
   VECS:   org = 0         len = 0x40      /* EL VECTOR DE INTERRUPCIONES*/
   ROM:    org = 0x40      len = 0x3fc0    /* C30 EVM 16K RAM            */
   RAM0:   org = 0x809300 len = 0x400      /* EL BLOQUE DE RAM 0         */
   RAM1:   org = 0x809c00 len = 0x400      /* EL BLOQUE DE RAM 1         */

   IOM:    org = 0800000h len = 02000h     /* -MSTRB I/O                 */
   IO:     org = 0804000h len = 02000h     /* -IOSTRB I/O                */

}

/* COLOCACION DE LAS SECCIONES O BLOQUES EN LA MEMORIA */
```

```
SECTIONS
{
   vectors: {} > VECS            /* EL VECTOR DEL INTERRUPCION          */
   .text:   {} > ROM             /* EL CODE                             */
   .cinit:  {} > ROM             /* LAS TABLAS DE INICIALIZACION        */
   .data:   {} > ROM             /* DATOS INICIALIZADOS SOLO DE .ASM    */
   .stack:  {} > RAM0            /* LA SISTEMA DE STACK                 */
   .bss:    {} > RAM1
   x_n_buff  align(64) : {} > RAM1   /* circ.buffer en RAM0             */
   in_port   0804000h  : {} > IO     /* A/D "input port" dirección      */
   out_port  0804001h  : {} > IO     /* D/A "output port" dirección     */
}
```

Capítulo 7

Descripción y arquitectura del TMS320C50

7.1 Introducción

Como ya se mencionó en el capítulo 1, el procesador TMS320C50 pertenece a la familia de procesadores TMS320 de 16/32 bits de Texas Instruments, de aritmética de punto entero, y combina un alto grado de paralelismo. Su especializado conjunto de instrucciones para procesamiento digital de señales provee rapidez y flexibilidad para producir microprocesadores capaces de ejecutar operaciones en tiempo real.

Esta generación de DSP está compuesta por el C50, el C51 y el C53. Estos procesadores son capaces de ejecutar el doble de instrucciones que la familia C2x y su código es compatible con las generaciones anteriores de punto fijo.

La generación de los C50 consta de los siguientes dispositivos:

- C50 es un procesador digital de señales CMOS con 10k en RAM y 2k en ROM.
- C51 es un DSP CMOS con 2k en RAM y 8k en ROM.
- C53 es un DSP CMOS con 4k en RAM y 16k en ROM.

7.2 Características

Enseguida se listan los características de este procesador:

- Ciclo de instrucción de punto fijo de 35/50 ns (28.6/20 MIPS).
- Código fuente compatible con los de las familias C1x yC2x.
- Operación basada en memoria RAM (C50).
- Operación basada en memoria ROM (C51).
- RAM de 9k×16 bits programa/datos (C50).

- RAM de 1k×16 bits programa/datos (C51).
- RAM de 3k×16 bits programas/datos (C53).
- Memoria ROM:
  - interna de 2k×16 bits (C50),
  - de programas dentro del micro de 8k×16 bits (C51),
  - de programa dentro del micro de 16k×16 bits (C53).

Máximo espacio direccionable externo de memoria 64k de programa, 64k de datos, 64k de entrada y salida, y 32k de uso global.

Unidad aritmetica-lógica de 32 bits. Acumulador de 32 bits. Buffer de 32 bits. Una unidad lógica paralela de 16 bits. Multiplicador paralelo de 16 por 16 bits con capacidad de 32 bits en el producto. Instrucción de multiplicación y acumulación en un solo ciclo. Ocho registros auxiliares con una unidad aritmética dedicada para direccionamiento. Once registros para guardar el contenido del CPU durante las interrupciones. Ocho niveles de stack en hardware. Corrimiento de cero a 16 bits a la izquierda o a la derecha. Dos bufers de direcionamiento circular indirecto. Instrucciones de repetición de bloques o instruciones. Instrucción de movimiento de un bloque de memoria. Puerto serial síncrono para comunicación con otro dispositivo. Puerto serial de acceso múltiple por división en el tiempo. Contador de intervalos con periodo, control y un contador de registros para inicializar y parar el dispositivo por software. Puerto paralelo de entrada y salida de 64k. Operación de sujeción para direccionamiento externo, 4 niveles en el pipeline. Modo indexado de direccionamiento. Modo indexado de direccionamiento en modo inverso sobre bits para la transformada rápida de Fourier FFT. Generador de reloj interno.

7.3 Arquitectura

La arquitectura del DSP TMS320C50 consta de tres segmentos básicos, como se observa en la figura 7.1:

- Unidad central de procesamiento.
- Memoria.
- Interfaz hacia periféricos.

El CPU es capaz de ejecutar operaciones aritméticas a una alta velocidad dentro de un corto ciclo de instrucciones por medio de un diseño arquitectural con un alto paralelismo. El alto desempeño de estos DSP está basado en una arquitectura tipo Harvard, la cual minimiza el procesamiento debido a la existencia de dos buses separados, uno para datos y otro para programas.

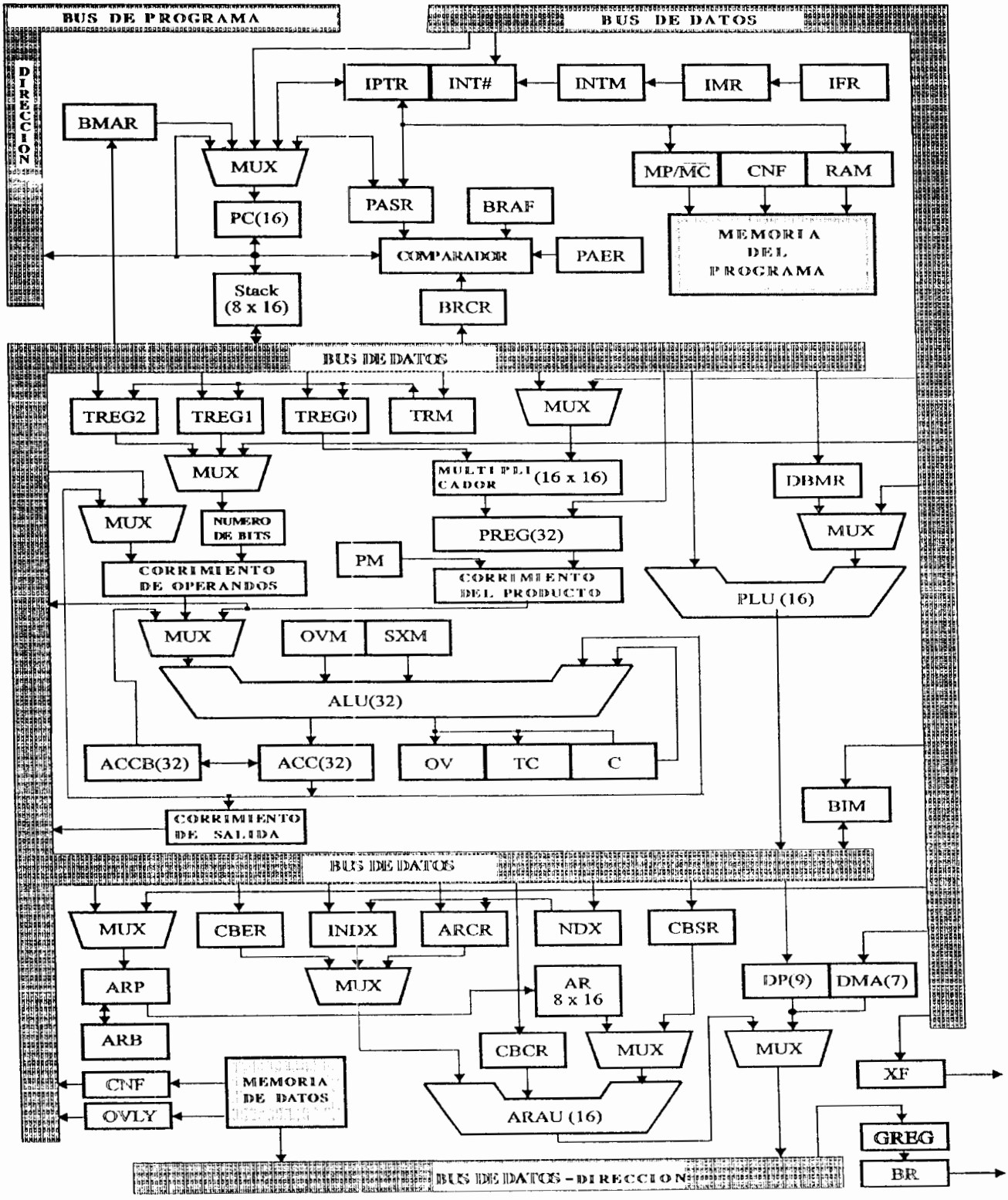


Figura 7.1. Arquitectura del TMS320C50

El C50 realiza aritmética en modo dos complemento mediante el uso de una ALU de 32 bits y un acumulador. La ALU es una unidad aritmética de proposito general que puede realizar operaciones booleanas; utiliza palabras de 16 bits de la memoria o derivadas de la instrucción anterior. El acumulador guarda el resultado de salida de la ALU y puede ser una segunda entrada para ésta. El acumulador es de 32 bits y está dividido en dos palabras, existen instrucciones para accesar a estas palabras por separado en memoria.

En adición a la ALU, existe una unidad lógica paralela PLU, que efectúa operaciones lógicas sin afectar al acumulador; provee de la manipulación necesaria para un control de alta velocidad.

El multiplicador realiza multiplicaciones de 16 por 16 bits en modo dos complemento con un resultado de 32 bits en un solo ciclo de máquina.

El registro de corrimiento del C50 tiene una entrada de 16 bits conectada al bus de datos y una salida de 32 bits conectada a la ALU. El desplazamiento puede ser de 0 a 16 bits programable con instrucciones o por el registro TREG1. Los espacios se rellenan con ceros en las posiciones menos significativas y las más significativas dependen del bit SXM en el registro ST1.

Tiene 8 niveles de stack en hardware donde se guarda el contenido del contador de programa durante las interrupciones y llamadas a subrutinas. Durante las interrupciones los registros importantes son guardados en el stack. La arquitectura del C50 está construida alrededor de los dos buses mayores "programa y datos". El bus de programa lleva el código de las instrucciones y los operandos inmediatos desde la memoria de programa. El bus de datos interconecta varios elementos como la CALU y el archivo de registros auxiliares a la memoria de datos. Ambos buses llevan los datos de la memoria interna y de la de programa al multiplicador en un ciclo de máquina para la operación de multiplicación y acumulación.

Hardware interno

El hardware interno del C50 ejecuta funciones que en otros procesadores se establecen por software o mediante código; se pueden realizar multiplicaciones de 16 por 16 bits en un ciclo de máquina, corrimiento de datos y manipulación de direcciones.

Modos de direccionamiento de memoria

El C50 puede direccionar 64k de memoria de programa y 96k de memoria de datos. Tiene 8 formas de direccionamiento:

- Modo directo mediante el bus de direccionamiento directo, relativo al apuntador de página en la memoria de datos.
- Modo de direccionamiento mediante un mapeo de la memoria.
- Modo indirecto.
- Modo indirecto corto.
- Modo inmediato largo.

- Modo de acceso por registro.
- Modo de direccionamiento inmediato largo.
- Modo de direccionamiento por bloques en los registros de memoria.

Las instrucciones, que son diferentes de las instrucciones de los DSP TMS320C25, se presentan en las siguientes tablas.

Mnemónico	*Descripción*	*Palabras*	*Ciclos*
ADCB	Suma ACCB al ACC con acarreo	1	1
ADDB	Suma ACCB al ACC	1	1
ANDB	AND ACCB con ACC	1	1
BSAR	Shift de ACC a la derecha	1	1
CRGT	Prueba si ACC ¿ ACCB	1	1
CRLT	Prueba si ACC ¡ ACCB	1	1
EXAR	Cambia ACCB con ACC	1	1
LACB	Carga ACC con ACCB	1	1
LAMM	Carga ACC con contenido de memoria mapeada por registro	1	1
ORB	OR de ACCB con ACC	1	1
ROLB	Giro de ACCB y ACC a la izq.	1	1
RORB	Giro de ACCB y ACC a la der.	1	1
SACB	Guarda ACC en ACCB	1	1
SAMM	Guarda ACC a memoria mapeada por registro	1	1
SATH	Desplazamiento de ACC a la derecha 0 o 16 bits especificada por el TREG1	1	1
SATL	Desplazamiento de ACC a la derecha 0 o 15 bits especificada por el TREG1	1	1
SBB	Resta de ACCB de ACC	1	1
SBBB	Resta de ACCB de ACC con préstamo	1	1
SFLB	Desplazamiento de ACCB y ACC a la izq.	1	1
SFRB	Desplazamiento de ACCB y ACC a la der.	1	1
XORB	OR exclusiva de ACCB y ACC	1	1

Tabla 7.1. Instrucciones del DSP TMS320C50 que trabajan con el acumulador

Mnemónico	*Descripción*	*Palabras*	*Ciclos*
MADD	Multiplica y acumula con dato apuntado por BMAR	1	3
MADS	Multiplica y acumula con dato apuntado por BMAR y con dato generado	1	3
ZPR	Cero se guarda en registro de producto	1	1

Tabla 7.2. Instrucciones del DSP TMS320C50 para multiplicación

Mnemónico	*Descripción*	*Palabras*	*Ciclos*
APL	AND de DBMR con una constante o dato de memoria	1/2	1 o 2
CPL	Compara DBMR o una constante con un valor de memoria	1/2	1 o 2
OPL	OR de DBMR o una constante con un valor de memoria	1/2	1 o 2
SPLK	Guarda un dato largo a una locación de memoria	2	2
XPL	XOR de DBMR o una constante con un valor de memoria	1/2	1 o 2

Tabla 7.3. Instrucciones del DSP TMS320C50 que trabajan con registros auxiliares

Mnemónico	*Descripción*	*Palabras*	*Ciclos*
B	Salto incondicional	2	4 o 2
BACC	Salto a dirección especificada por ACC	2	4 o 2
BANZ	Salto cuando ARN es diferente de cero	2	4 o 2
BCND	Salto condicional	2	4 o 2
CALA	LLamado indirecto a subrutina	1	4 o 2
CALL	LLamado a subrutina	2	4 o 2
CC	LLamado condicional	2	4 o 2
INTR	Interrupción suave	1	4
RETC	Retorno o regreso de subrutina	1	4 o 2
RETE	Retorno global de interrupción	1	4
RETI	Retorno con intercambio	1	4
XC	Ejecuta la siguente operación condicionada	1	1

Tabla 7.4. Instrucciones de salto del DSP TMS320C50

Mnemónico	*Descripción*	*Palabras*	*Ciclos*
BLPD	Mover el bloque de datos de la memoria de datos a la de programa	1	2 o 2
LMMR	Carga en memoria mapeada por registro	2	2 o 3
SMMR	Guarda en memoria mapeada por registro	2	2 o 3
CLRC	Limpia bit de control	1	1
IDLE	Permanece sin trabajar hasta la interrupción	1	1
IDLE2	Permanece sin trabajar hasta la interrupción por baja potencia	1	1
RPTB	Repite el bloque	2	2
RPT2	Repite la siguiente instrucción y limpia ACC y PREG	2	2
SETC	Habilita o pone el bit de control	1	1

Tabla 7.5. Instrucciones de memoria, I/O y de control del DSP TMS320C50

7.4 Memoria

El rango total de memoria direccionable por la familia del C50 es de 224k en palabras de 16 bits. El espacio en memoria está dividido en cuatro segmentos:

- 64k de programa.
- 64k de datos locales.
- 32k de datos globales.
- 64k para puertos I/O.

La arquitectura en paralelo del C50 permite al DSP realizar tres operaciones sobre memoria en un ciclo de máquina:

- Buscar una instrucción.
- Leer un operando.
- Escribir un operando.

Debido a la arquitectura tipo Harvard es posible accesar simultáneamente a la memoria de datos y a la memoria de programa.

Los tres buses paralelos son: el de programa de lectura y escritura (PAB), el de lectura de datos (DAB1), y el de escritura de datos (DAB2). Cada bus accesa a diferentes espacios de memoria.

La memoria del C50 está organizada en cuatro espacios individuales: programa, datos locales, datos globales, y puertos de entrada y salida.

El espacio de memoria de programa contiene las instrucciones que van a ser ejecutadas, así como las tablas utilizadas en la ejecución. El espacio de datos local guarda los datos usados por las instrucciones. El espacio global de datos puede intercambiarlos con otros procesos dentro del sistema, o puede servir como un espacio adicional de datos.

El C50 incluye una considerable cantidad de memoria dentro del dispositivo para lograr alto rendimiento e integración del DSP. El C50 incluye 2k en palabras de 16 bits para el arranque en ROM, 96k para programas y datos de acceso simple en RAM (SARAM), y 1056 palabras de acceso doble en RAM (DARAM). El sistema de arranque reside en la memoria de programa a partir de la dirección 0h, incluye una prueba para el dispositivo y el código de arranque.

El bloque de 9k de acceso simple en RAM puede ser mapeado como programa o espacio de datos y reside a partir de la dirección 0800h; este espacio de memoria requiere de un ciclo completo de máquina para realizar una escritura o lectura. El espacio de acceso doble puede ser leído y escrito en el mismo ciclo de máquina. Las 1056 palabras de memoria de acceso doble están configuradas en tres bloques: bloque 0 (B0) es de 512 palabras y se direcciona a partir de 0100h hasta la 02FFh en la memoria local de datos, o en la dirección 0FE00h-0FFFFh en el espacio de programa. El bloque 1 (B1) es de 512 palabras y está localizado en

la memoria de datos local en las direcciones 0300h-04FFh. El bloque 2 (B2) de 32 palabras también se encuentra en la memoria de datos local a partir de la dirección 060h.

El C51 no tiene la ROM de 2k arranque en el espacio de memoria de programa; en cambio, tiene un espacio de 8k de palabras de 16 bits de memoria de programa o de datos de acceso simple en RAM. También contiene un bloque de 8k de palabras de 16 bits de memoria ROM mascarable. La ROM está localizada de la dirección 0h y hasta la dirección 1FFFh en el espacio de programa. Se cuenta con memoria mapeada RAM de datos de acceso simple de 1k palabras de 16 bits, con las direcciones a partir de 800h y hasta la 0BFFh. El espacio del programa se localiza a partir de la dirección 2000h y hasta la 23FFh.

El bloque de la RAM de acceso doble en el C51 está mapeado en las mismas direcciones que en el C50.

El C53 tiene memoria mascarable ROM de 16 k palabras de 16 bits en el dispositivo y 3k palabras de 16 bits de RAM de acceso simple. La ROM está localizada a partir de la dirección 0h y hasta la 3FFFh en el espacio de programa. La memoria RAM de acceso simple es de 3k mapeada en el espacio de datos en las direcciones de la 800h a la 13FFh. La memoria RAM de doble acceso es la misma en todos los DSP de la familia de C50. La descripción anterior puede verse en las gráficas de memoria, figura 7.2.

La configuración del mapa de memoria se puede hacer mediante software para ubicarla dentro o fuera del dispositivo.

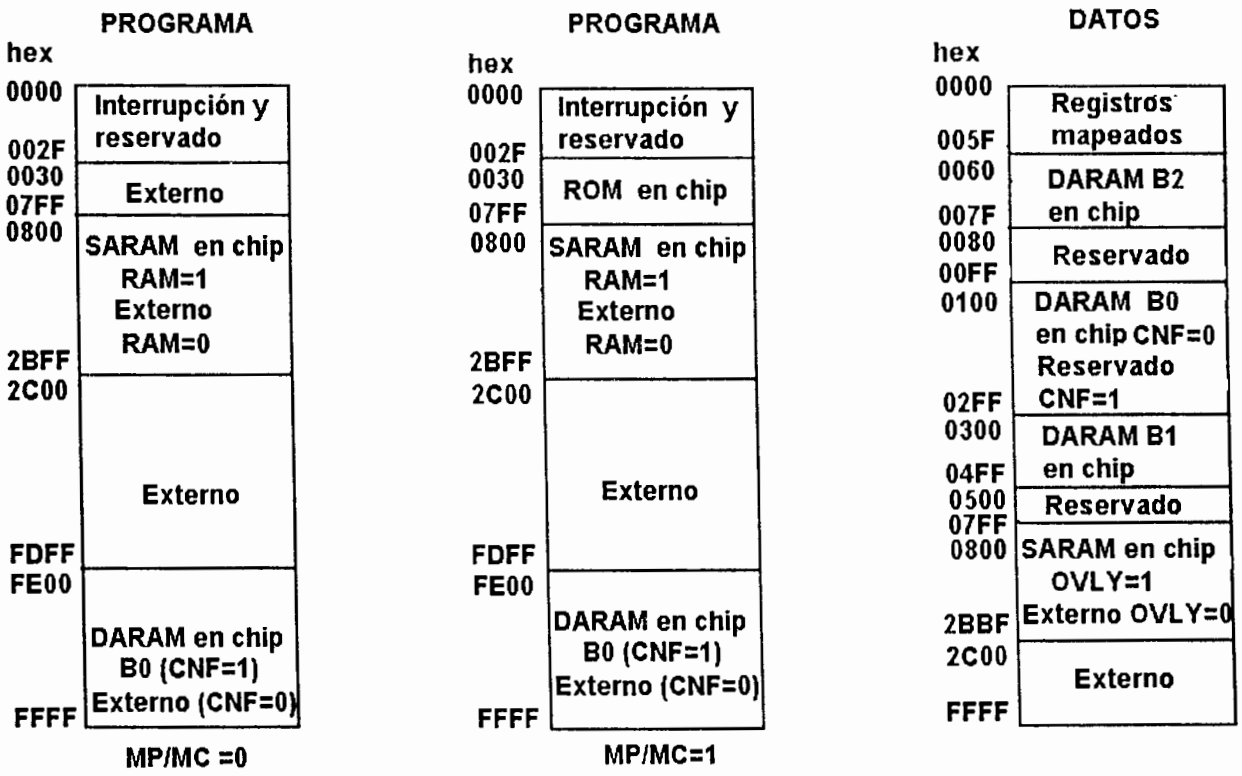


Figura 7.2. Mapa de memoria de la familia de C50

Las ventajas de la operación con memoria interna son:

- Alto desempeño, ya que no se requiere de espera en acceso a memorias externas.
- Menor costo que las memorias externas.
- Se requiere de menor potencia para la alimentación.

La memoria del programa puede residir dentro o fuera del chip. Después de la inicialización, la configuración de la memoria se establece por el nivel en el pin $MP/\overline{MC}$. Si este pin tiene un nivel alto, el dispositivo es configurado como un microprocesador y la memoria ROM interna no es direccionada. Si este pin tiene un nivel bajo, el dispositivo es configurado como una microcomputadora y la ROM interna está disponible. El C50 busca el vector de inicialización en la localidad cero de la memoria de programa, de tal manera que si el dispositivo opera como una microcomputadora, se inicializa desde la ROM interna. De otra manera, se inicializa de memoria externa. Una vez que el programa está corriendo, se puede cambiar la configuracion del pin $MP/\overline{MC}$, poniendo o limpiando del bit $MP/\overline{MC}$ en el registro PMST.

7.5 Ejemplos de un filtro de respuesta infinita al impulso IIR

Los filtros IIR son usados frecuentamente en procesamiento digital de señales. La función de transferencia de uno de estos filtros está dada por:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + ... + b_m z^{-m}}{1 + a_1 z^{-1} + ... + a_n z^{-n}}$$

En la figura 7.3 se muestra un diagrama de bloques de un filtro IIR de orden N.

En el dominio del tiempo, un filtro IIR de orden N está representado por las siguientes ecuaciones diferenciales:

$$d(n) = x(n) - d(n-1)a_1 - ... - d(n-N+1)a_{N-1}$$

$$y(n) = d(n)b_0 + d(n-1)b_1 + ... + d(n-N+1)b_{N-1}$$

donde

$x(n)$ es la muestra actual de entrada.

$y(n)$ es la salida del filtro.

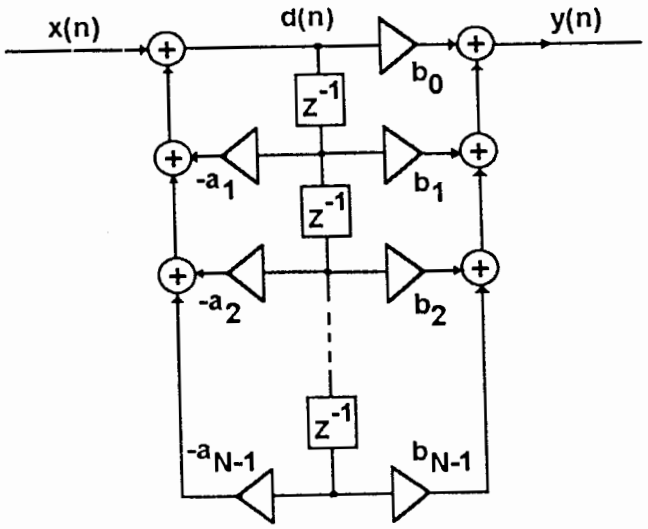


Figura 7.3. Filtro IIR de orden N

Las dos ecuaciones anteriores se pueden realizar fácilmente en el C50 con la instrucción de multiplicación-acumulación (MAC, MACD, MADS, MADD). Obsérvese que en la segunda ecuación se requiere mover los datos para actualizar la secuencia de la variable d. En este ejemplo utilizamos las instrucciones de repetición simple (RPT) y multiplicamos y acumulamos (MAC, MACD).

```
        .title"Filtro IIR tipo II de orden N"
        .mmregs
IIR-N:  ZPR               ;limpia el registro P
        LACC  *,15,AR1    ;carga la entrada
        RPT   #(N-2)      ;desde i=1, i<=N,++i
        AC    COEFFB,*-   ;ACC+=-a(N-i)*d(n-N+i)
        APAC              ;acumulación final
        SACH  *,1         ;salva d(n)
        ADRK  N-1         ;AR1 --> d(n-N+1)
        RPTZ  #(N-1)      ;desde i=1,i<=N,++i
        MACD  COEFFA,*-   ;ACC+=b(N-i)*d(n-N+i)
        LTA   *,AR2       ;acumulación final
        SACH  *,1         ;salva  y(n)
```

Este programa requiere que las variables de estado se guarden en la memoria de datos y los coeficientes en la memoria de programas con las siguientes condiciones de entrada:

```
AR0 --> entrada
AR1 --> d(n-N+1)
AR2 --> Salida
COEFFA --> -a(N-1)
COEFFB --> b(N-1)
ARP = AR0
```

Debido a la naturaleza recursiva de un filtro IIR, la cuantización (*quantization*) de los coeficientes del filtro puede causar una variación significante en la respuesta en frecuencia. Para evitar este problema, la función de transferencia puede ser dividida en secciones de orden menor que son puestas en cascada con las otras. El siguiente ejemplo muestra un programa de un filtro IIR en cascada de segundo orden. Los coeficientes del filtro y las variables de estado son guardados en la memoria de datos. Obsérvese el uso de las instrucciones LTD y MPYA para la multiplicación-acumulación y para mover datos.

```
        .title "Filtro IIR en cascada de orden N en Biquad"
        .mmregs
BIQUAD:
        ZPR                 ;limpia el registro P
        LACC  *,15,AR1      ;carga valores de entrada
        SPLK  #2,INDX       ;Habilita el registros de índice
        SPLK  #N-1,BRCR     ;habilita el contador
        RPTB  ELOOP-1       ;se repite por N i biquads
LOOP:
        LT    *-,AR2        ;T=d(n-2)
        MPYA  *+,AR1        ;ACC=x(n), P=-d(n-2)*a2
        LTA   *-,AR2        ;ACC+=-d(n-2)*a2, T=d(n-1)
        MPY   *+            ;P=-d(n-1)*a1
        LTA   *+,AR1        ;ACC+=-d(n-1)*a1, T=b2
        SACH  *0+,1         ;salva d(n)
        MPY   *-            ;P=d(n-2)*b2
        LACL  #0            ;ACC=0
        LTD   *-,AR2        ;T=d(n-1), d(n-2)=d(n-1)
        MPY   *+,AR1        ;ACC+=d(n-2)*b2, P=d(n-1)*b1
        LTD   *-,AR2        ;T=d(n), d(n-1)=d(n)
        MPY   *+,AR1        ;ACC+=d(n-1)*b1, P=d(n)*b0
ELOOP:
        LTA   *,AR4         ;acumulación final
        SACH  *,1           ;salva a salida
```

Capítulo 8

Descripción del ensamblador y depurador DSK

En la figura 8.1 se muestra la forma en que el ensamblador convierte el programa fuente (miprogr.asm) al código para procesador TMS320C5x, esto crea el archivo ejecutable que se puede correr en la tarjeta "DSP starter kit"(DSK). Asimismo se puede utilizar el depurador para corregir el programa.

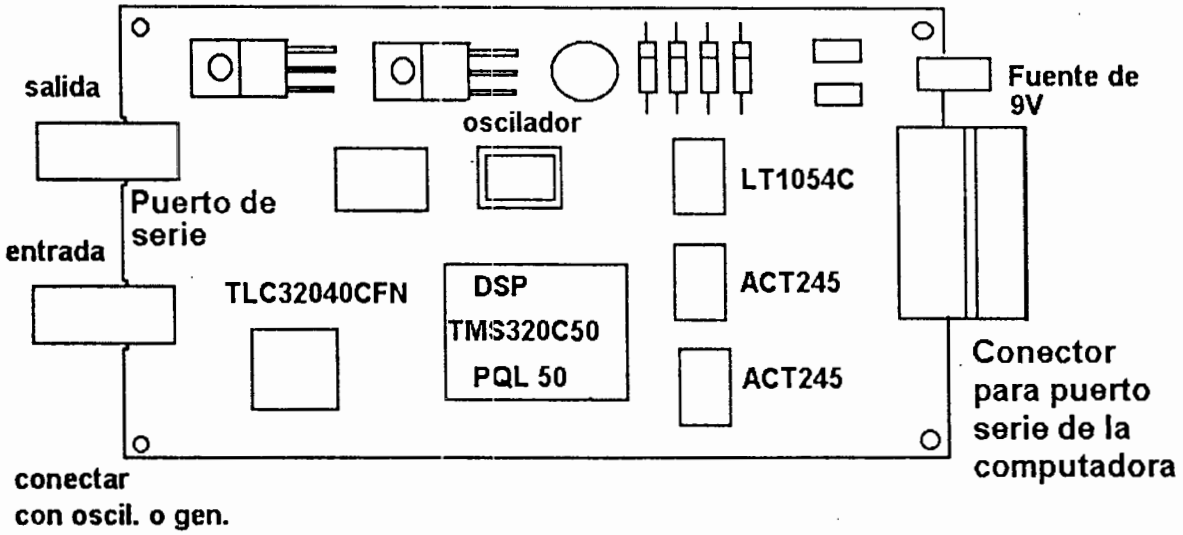


Figura 8.1. Sistema DSK (Digital Starter Kit)

Con la instrucción dsk5a miprogr.asm se genera el archivo ejecutable miprogr.dsk, que es útil también para el depurador.

Para obtener el listado del programa miprogr.asm con los errores y advertencias, es necesario teclear la instrucción dsk5a miprogr.asm -l. El ensamblador no crea sólo el archivo miprogr.dsk, sino también el archivo miprogr.lst donde está la descripción de los errores y advertencias. Si tenemos el archivo miprogr.dsk podemos invocar el depurador, para lo cual se utiliza la instrucción dsk5d.

La instrucción dsk5d despliega en la pantalla el depurador del TMS320C5x con las ventanas, como se observa en la figura 8.2. La ventana INPUT COMMAND se utiliza para cargar los programas en la pantalla con el propósito de corregir errores, etc.

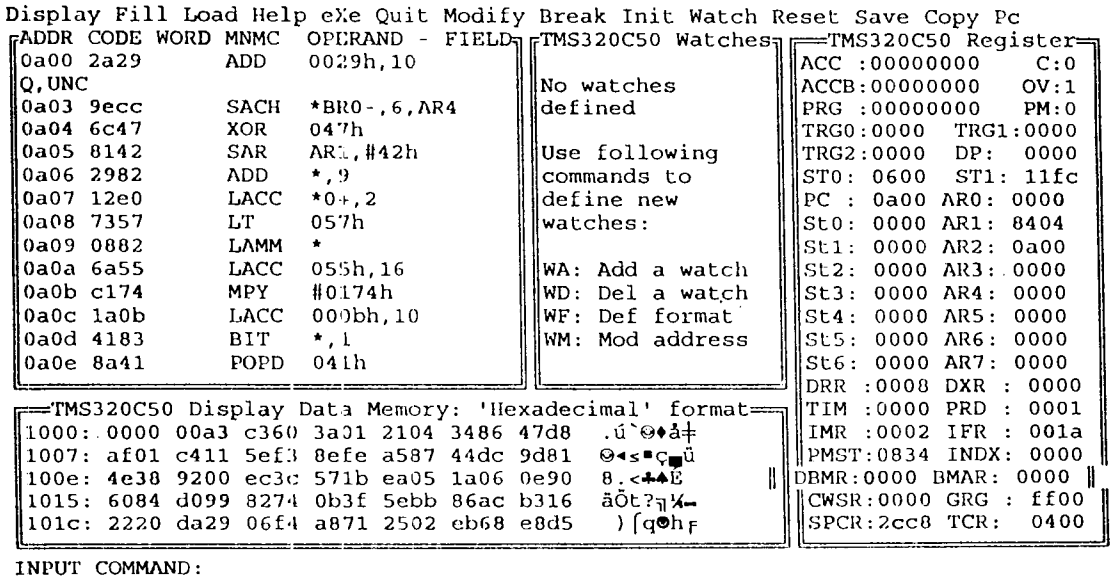


Figura 8.2. Ventanas de la pantalla del depurador

El depurador del DSK tiene más de 50 instrucciones. Todos los comandos se pueden invocar mediante el menú que nos lleva al submenú, como se aprecia en la figura 8.3. Si se oprime la tecla ESC se regresa del submenú al menú. Si se teclea F1 o H se obtiene ayuda, como se observa en la figura 8.4.

Se pueden utilizar las teclas de función para abreviar los comandos. Por ejemplo: F1 invoca la ayuda a la pantalla, F2 imprime el contenido de la pantalla a un archivo screen.srn, F5 ejecuta el programa hasta el próximo punto de interrupción señalado (*breakpoint*), F8 ejecuta el programa instrucción por instrucción.

Como ejemplo de cómo se utiliza el depurador y starter kit-DSK, realizamos un programa que genere una onda senoidal. Primeramente, con un editor de texto, se escribe el programa en ensamblador sen.asm.

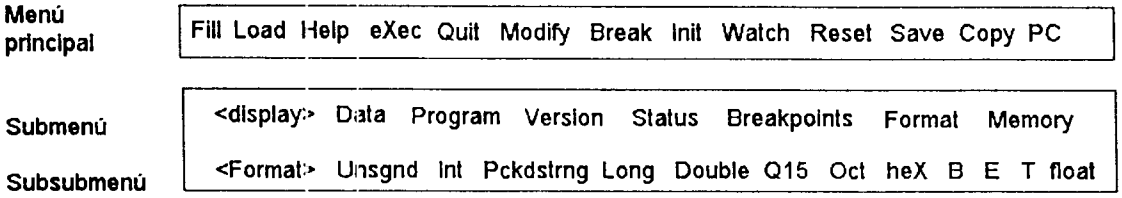


Figura 8.3. Menú principal y submenú del display

```
DSP STARTER KIT DEBUGGER
+ADD+---------- Help Window Display ----------+ter-+
¦0a0¦                                          ¦C:0¦
¦0a0¦ Usage of the program: dsk5d [[-|\][options]  ¦V:1¦
¦0a0¦ Options: Please note that options are not case sensitive ¦M:0¦
¦0a0¦ --------                                 ¦000¦
¦0a0¦ ? or H : this display                    ¦000¦
¦0a0¦ Bxxxxx : xxxxx selects the baudrates! Baud rate options: ¦1fc¦
¦0a0¦       4800,9600,19200[default],38400,57600 ¦00 ¦
¦NDE¦ ComX  : comport: x=1 [default] or x=2 optional 'C1' or 'C2' can ¦04 ¦
¦0a0¦       be used for each comport respectively. (comport 3 & 4 ¦00 ¦
¦0a0¦       are not supported)                 ¦00 ¦
¦0a0¦ Exxxx : defines entry point address in hexidecimal ¦00 ¦
¦0a0¦       format: c|E[0x]xxxx[h]             ¦00 ¦
¦0a0¦ I     : selects logic level for DTR->Reset (default->inverse) ¦00 ¦
¦0a0¦ L     : selects the EGA/VGA screen length (43 or 50) ¦00 ¦
+---¦ S     : selects the default screen length (25) ¦000¦
+--T¦                                          ¦001¦
¦100¦ Function Key definitions                 ¦01a¦
¦100¦ ------------------------                 ¦000¦
¦100¦                                          ¦000¦
¦101¦                                          ¦f00¦
¦101¦   PGUP PGDN HOME END newPage File Quit ESC   01/17  ¦400¦
+---+-------------------------------------------------------------+----+
Press any key to continue...
```

Figura 8.4. Ayuda en la pantalla

```
        .mmregs
        .ps         008ah
        B           RINT
        .ps         00a00h
        .entry
        LDP         DXR
        LAMM        IMR
        OR          #10h
        SAMM        IMR
LOOP:   ADD         #10
        SACL        DXR,3
        IDLE
        B           LOOP
PRINT:  RETE
```

Para correr el programa se necesita entrar en el directorio donde se tienen los archivos dsk5a y dsk5d. Al teclear dsk5a sen.asm se crea el módulo sen.dsk. Ahora conectamos la salida de la tarjeta al osciloscopio y en seguida tecleamos L, D, sen.dsk enter, enter, enter, X, G. En la pantalla del osciloscopio, si todo está bien conectado, aparece la senoidal.

Capítulo 9

Programa MATLAB para el procesamiento digital de señales

El programa MATLAB se utiliza para cálculos numéricos y para graficar las respuestas de los circuitos digitales y analógicos. Lo podemos utilizar para síntesis y análisis de los filtros digitales. Utilizando el paquete MATLAB se puede calcular fácilmente la respuesta de los filtros digitales a un impulso, al tren de impulsos o a cuaquier señal.

MATLAB trabaja sólo con los datos que son expresados en forma vectorial o matricial. Si trabajamos con escalares los expresamos como una matriz 1×1. Los vectores se expresan como matrices que tienen sólo una columna o una fila, entonces, los expresamos como matrices $1 \times n$ o $n \times 1$. Por esta razón toma el paquete el nombre MATLAB, es abreviación de las palabra "MATrix LABoratory".

Los datos en los vectores pueden representar, por ejemplo, una señal acústica – voz – o una señal visual – una imagen –.

9.1 Introducción de datos en MATLAB

Los elementos de la matriz se escriben dentro de los paréntesis cuadrados. Los elementos en las columnas son separados con un espacio y las filas son separadas con punto y coma. Por ejemplo:

$$>> A = [1\ \ 2\ \ 3; 4\ \ 5\ \ 6; 7\ \ 8\ \ 9]$$

Después de la tecla "entrar"obtenemos el resultado

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Los elementos de las matrices pueden ser también las siguientes expresiones. Por ejemplo, el vector

$$>> X = [1.2 \quad -sqrt(2) \quad (1+2+3)*2/5]$$

resulta un vector de la forma

$$X = 1.2000 \quad -1.4142 \quad 2.4000$$

Si queremos ampliar el vector X sobre un elemento de valor absoluto de $-sqrt(2)$, podemos escribir:

$$>> X(5) = abs(X(2))$$

y el nuevo vector X toma la forma:

$$X =$$
$$1.200 \quad -1.4141 \quad 2.4000 \quad 0.0000 \quad 1.4142$$

El vector es sustituido automáticamente sobre el valor X(5) y los elementos no definidos en nuestro caso X(4) se sustituyen por cero. Podemos en la matriz X agregar una fila. Por ejemplo, se desea agregar la cuarta fila en la matriz X que tiene los valores (10 11 12). Esto se efectúa con los comandos:

$$>> r = [10 \quad 11 \quad 12]$$
$$>> A = [A; r]$$

y la nueva matriz toma la forma:

$$A =$$
$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{matrix}$$

Si la matriz es muy grande, se pueden omitir algunas filas. Por ejemplo, la tercera y cuarta fila. Esto se logra con el comando:

$$>> A = A(1:2,:)$$

9.2 Números complejos y matrices complejas

En todos los cálculos, MATLAB permite utilizar los números complejos. La parte imaginaria de los números complejos se denota con i o j. Por ejemplo:

$$z = 3 + 4 * j \qquad\qquad 5 + 5 * i$$

La otra posibilidad de escribir el número complejo es

$$>> w = r * exp(i * theta)$$

Las matrices complejas se pueden escribir en dos modos:

$$>> a = [1 \quad 2; 3 \quad 4] + j * [5 \quad 6; 7 \quad 8] \qquad a = [1 + 5 * j \quad 2 + 6 * j; \quad 3 + 7 * j \quad 4 + 8 * j]$$

9.3 Formatos de salida

El formato de los números de salida se elige con la instrucción *format*. Supongamos que tenemos el vector:

$$>> x = [4/3 \;\; 1.2345e-5];$$

Los formatos que podemos elegir son los siguientes:

>>format short

$$1.3333 \qquad 0.0000$$

>>format short e

$$1.3333e+000 \qquad 1.2345e-6$$

>>format long

$$1.33333333333333 \qquad 0.00000123450000$$

>>format long e

$$1.333333333333333e-000 \qquad 0.0000012345000000e-006$$

Para los números en la forma hexadecimal, obtenemos

>>hex

$$3fff5555555555555 \qquad 3eb4b6231abfd271$$

9.4 Operaciones con las matrices

Para obtener la matriz transpuesta utilizamos el apóstrofo. Por ejemplo, la matriz transpuesta de la matriz A

$$>> A = [1\ 2\ 3;\ 4\ 5\ 6;\ 7\ 8\ 9]$$

es la matriz

$$>> B = A'$$

y el resultado es

$$B = \begin{matrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{matrix}$$

Los símbolos para las operaciones con matrices son iguales que para los números. Para la suma de las matrices utilizamos + y para la resta −. Las matrices deben tener el mismo tamaño: (m,n)+(m,n)=(m,n).

La suma de los matrices anteriores

$$>> C = A + B$$

resulta

$$C = \begin{matrix} 2 & 6 & 10 \\ 6 & 10 & 14 \\ 10 & 14 & 18 \end{matrix}$$

La suma es también definida si el operando es un escalar:

$$>> x = [-1 \quad 0 \quad 2]$$

$$>> y = x - 1$$

obtenemos como resultado

$$y = \begin{matrix} -2 \\ -1 \\ 1 \end{matrix}$$

La operación de multiplicación se realiza con *. Por ejemplo:

$$>> x' * y$$

nos da el resultado

$$ans = \begin{matrix} 4 \end{matrix}$$

Pero si escribimos $y * x'$, obtenemos

$$ans = \begin{matrix} 2 & 0 & -4 \\ 1 & 0 & -2 \\ -1 & 0 & 2 \end{matrix}$$

Se puede dividir la matriz A entre la matriz B. Si escribimos $C = A \quad B$, multiplicamos $inv(A) * B$; si escribimos C=B/A, multiplicamos $B * inv(A)$. Por ejemplo:

$$>> s = x \; y$$

nos da el resultado

$$s = \begin{matrix} 0.8000 \end{matrix}$$

	Instrucciones para matrices
expm	Exponencial de la matriz.
logm	Logaritmo de la matriz.
sqrtm	Raíz de la matriz.
poly	Polinomio característico.
det	Determinante.
trace	Trazo de la diagonal.
kron	Producto de Cronecker.

Tabla 9.1. Algunas operaciones matriciales

Resultado de división $>> S = y/x$ es la matriz:

$$S = \begin{matrix} 0.0000 & 0.0000 & -1.0000 \\ 0.0000 & 0.0000 & -0.5000 \\ 0.0000 & 0.0000 & 0.5000 \end{matrix}$$

Podemos también obtener la raíz de la matriz con la instrucción A^p, p es un escalar que significa la raíz y A es una matriz cuadrada. Es posible también elevar la matriz con la instrucción p^A. Si ambas son matrices, el programa marca el error. Un operando debe ser el escalar.

Las expresiones como *exp(A)* y *sqrt(a)* están comprendidas entre las operaciones con los elementos de las matrices. En la tabla 9.1 encontramos las operaciones que podemos utilizar para matrices.

9.5 Instrucciones matemáticas elementales

La instrucción *fix([a])* genera los números enteros en la matriz A. Por ejemplo, si tenemos la matriz $>> A = [1\ \ 2\ \ 3;\ \ 4\ \ 5\ \ 6]$; después de la instrucción

$$>> B = fix(pi * A)$$

obtenemos como resultado:

$$B = \begin{matrix} 3 & 6 & 9 \\ 12 & 15 & 18 \end{matrix}$$

Las instrucciones elementales se muestran en la tabla 9.2 y, y en la tabla 9.3, las instrucciones trigonométricas y las funciones especiales.

	Instrucciones elementales
abs	Valor absoluto del número complejo.
angle	Fase del número complejo.
sqrt	Raíz cuadrada.
real	Parte real del número complejo.
imag	Parte imaginaria del número complejo.
conj	Número conjugado.
round	Redondeo del número.
fix	Truncamiento.
sign	Signo del número.
rem	Resto.
exp	Función exponencial.
log	Logaritmo natural.
log10	Logaritmo de base 10.

Tabla 9.2. Operaciones elementales

	Instrucciones trigonométricas y especiales
sin	Función seno.
cos	Función coseno.
tan	Función tangente.
asin	Función arco seno.
acos	Arco coseno.
atan	Arco tangente.
atan2	Arco tangente en los cuatro cuadrantes.
sinh	Seno hiperbólico.
cosh	Coseno hiperbólico.
tanh	Tangente hiperbólica.
asinh	Argumento seno hiperbólico.
acosh	Argumento coseno hiperbólico.
atanh	Argumento tangente hiperbólica.
bessel	Función de Bessel.
gamma	Función gamma.
rat	Aproximación racional.
erf	Función de error.
ellipk	Integral elíptico.
ellipj	Jacobiano de la funcion elíptica.

Tabla 9.3. Operaciones elementales

9.6 Vectores y matrices

Si utilizamos dos puntos en la expresión

$$>> X = 1 : 5$$

obtenemos en una fila un vector:

$$X =$$
$$1 \quad 2 \quad 3 \quad 4 \quad 5$$

el vector puede tener el incremento $\pi/8$, si se escribe

$$>> y = 0 : \pi/8 : \pi$$

El incremento puede ser negativo. Por ejemplo:

$$z = 6 : -1 : 0$$

Con los dos puntos en la instrucción podemos crear una tabla. Por ejemplo, con las instrucciones

```
>> x = (0.0 : 0.2 : 1.0)
>> y = exp(-x). * sin(x)
>> [xy]
```

obtenemos la siguiente tabla:

```
ans =
    0.000   0.0000
    0.2000  0.1627
    0.4000  0.2610
    0.6000  0.3099
    0.8000  0.3223
    1.0000  0.3096
```

La otra posibilidad de crear los vectores es mediante la instrucción *linspace*. Por ejemplo, la instrucción

$$>> k = linspace(-\pi/2, \pi/4, 4)$$

crea un vector que tiene cuatro parámetros en los límites $-\pi/2$ y $\pi/2$. Obtenemos

```
k =
   -1.5708   -0.5236   0.5236   1.5708
```

Los datos del resultado son guardados en la matriz, para lo cual abrimos un archivo con la instrucción *save*. Por ejemplo, la instrucción

```
>> A = rand(4,3);
>> save temp.datA/ascii
```

guarda la matriz A en el archivo *temp.dat*.

9.7 Instrucciones para graficar

Las instrucciones para graficar se presentan en la tabla 9.4.

	Instrucciones para graficar
plot	Gráfica lineal x-y
loglog	Gráfica x-y con los ejes logarítmicos
semilogx	Gráfica x-y con eje x logarítmico
semilogy	Gráfica x-y con eje y logarítmico
polar	Gráfica en coordenadas polares
mesh	Gráfica de tres dimensiones
contour	Gráfica del contorno
bar	Gráfica de barras
stairs	Histograma
title	Título de la gráfica
xlabel	Señalización del eje x
ylabel	Señalización del eje y
text	Texto dentro de la imagen
grid	Cuadricular la imagen
axis	Escala de los ejes
hold	Congelar la gráfica en la pantalla
clg	Borrar la gráfica anterior
subplot	Para dividir la pantalla en partes
ginput	Entrada de los datos

Tabla 9.4. Operaciones elementales

Ejemplo

Las instrucciones

```
>> y = [0 0.40 0.84 1 0.91 0.6 0.14];
>> plot(y);
>> title('miprimerGraf');
>> xlabel('eje x');
>> ylabel('eje y');
>> grid
```

nos generan la gráfica de la figura 9.1.

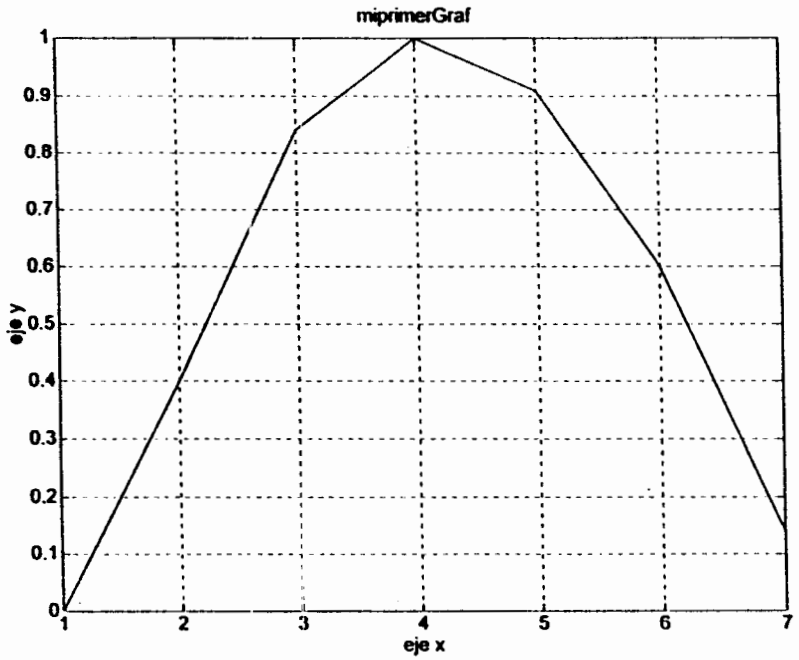


Figura 9.1. Gráfica de y

Gráfica con curvas múltiples

En una gráfica podemos representar más funciones. Para obtener un número mayor de curvas, en una gráfica utilizamos el comando:

$$>> plot(x1, y1, x2, y2, ..., xn, yn)$$

Los tipos de curva también pueden elegirse. Si la primera gráfica debe ser realizada con ":"y la otra marcada con "+", el comando para graficar será

$$>> plot(x1, y1, ':', x2, y2, '+')$$

También se puede seleccionar el color de la gráfica. Para el color rojo utilizamos r; para el color verde, g; para el color azul, b; y para el color blanco, w.

Los comandos

$$>> plot(x, y, 'r')$$
$$>> plot(x, y, '+g')$$

representan en la primera gráfica la curva roja y en la segunda gráfica la curva con la cruz verde +.

Gráfica de tres dimensiones

Si queremos representar la gráfica de tres dimensiones $Z = f(x, y)$ necesitamos crear los vectores X y Y que contienen los números de las filas y columnas de la matriz. Vamos a graficar la gráfica de tres dimensiones de la ecuación $sin(r)/r$ donde $r = X^2 + Y^2$.

$x = -8 : .5 : 8;$
$y = x';$
$X = ones(y) * x;$
$Y = y * ones(x);$
$R = sqrt(X.\hat{}2 + Y.\hat{}2) + eps;$
$Z = sin(R)./R;$
$mesh(Z)$

Las primeras dos instrucciones definen la zona de la gráfica. La tercera y cuarta instrucción sirven para crear los vectores X y Y. La matriz R contiene las distancias desde el origen. La gráfica de tres dimensiones se muestra en la figura 9.2

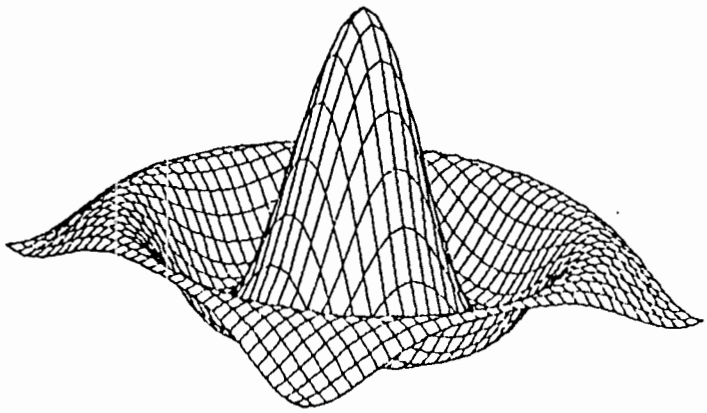


Figura 9.2. Gráfica de tres dimensiones

Cómo imprimir la gráfica

Para imprimir la gráfica sirven las instrucciones *prtsc*, *meta* y *gpp*. Antes de imprimir mediante prtsc, necesitamos configurar la computadora con la instrucción *graphics laserjet* si la impresora es láser. Después de obtener la gráfica en la pantalla oprimimos la tecla Print Screen. Otro método sería guardar la gráfica en un archivo, por ejemplo *nombre.met* con la instrucción meta nombre. Creamos el archivo *nombre.met* y lo podemos imprimir con la instrucción:

gpp %1.met /djet150 /op /fltp1

op significa imprimir la gráfica en la primera mitad de la página.

En el paquete MATLAB podemos programar también ciclos repetitivos. Por ejemplo, guardar los ceros en una variable $x(i)$ mediante la instrucción

$$for \;\; 1 : n, x(i) = 0, end$$

Si hay más ciclos, entonces, cada uno debe tener su propio *end*. Por ejemplo:

```
for  i = 1 : m
     for        j = 1 : n
                A(i, j) = 1/(i + j − 1);
     end
end
```

El punto y coma después de la instrucción hace que el resultado no se despligue en la pantalla, sólo el resultado se guarda en la matriz *A(i,j)*. Si queremos ver el resultado es necesario teclear A y enter. La instrucción *while* permite repetir la instrucción más veces hasta que la condición se cumpla.

```
n = 1;
while prod(1 : n) < 1.e10;
n = n + 1;
```

Con la instrucción *if* podemos hacer las decisiones en el programa. Por ejemplo, en el programa que sigue, la instrucción *if* crea tres salidas que dependen del signo n.

```
if   n < 0
     A =   negative(n)
           elseif       rem(n, 2) == 0
                        A = even(n)
           else
                        A = odd(n)
end
```

En MATLAB podemos crear programas con extensión .m, por ejemplo, miprogr.m, y después se pueden correr mediante la instrucción *miprogr*.

Diseño de los filtros

Las instrucciones para calcular los filtros de Butterworth, Chebychev, Chebychev inversa y los filtros de Cauer (filtros elípticos) se presentan en la tabla 9.5

El significado de las instrucciones es:

N	Orden del filtro
WN	Frecuencia del corte
WN1; *WN2*	Frecuencias del corte de paso banda
$[Z, P, K]$	El programa calcula los ceros, los polos y la constante
$[A, B, C, D]$	El programa calcula las matrices de estado
R y RP	Atenuación en la banda de paso
RS	Valor de atenuación en la banda supresora
high o stop	El programa calcula el filtro de paso alta o supresor de banda

$[B,A] = butter(N,Wn)$
$[B,A] = butter(N,WN1,WN2)$
$[B,A] = butter(N,WN,'high')$
$[B,A] = butter(N,WN1,WN2,'stop')$
$[Z,P,K] = butter(.....)$
$[A,B,C,D] = butter(.....)$
$[B,A] = cheby1(N,R,Wn)$
$[B,A] = cheby1(N,R,WN1,WN2)$
$[B,A] = cheby1(N,R,WN,'high')$
$[B,A] = cheby1(N,R,WN1,WN2,'stop')$
$[Z,P,K] = cheby1(.....)$
$[A,B,C,D] = cheby1(.....)$
$[B,A] = cheby2(N,R,Wn)$
$[B,A] = cheby2(N,R,WN1,WN2)$
$[B,A] = cheby2(N,R,WN,'high')$
$[B,A] = cheby2(N,R,WN1,WN2,'stop')$
$[Z,P,K] = cheby2(.....)$
$[A,B,C,D] = cheby2(.....)$
$[B,A] = ellip(N,RP,RS,Wn)$
$[B,A] = ellip(N,RP,RS,WN1,WN2)$
$[B,A] = ellip(N,RP,RS,WN,'high')$
$[B,A] = ellip(N,RP,RS,WN1,WN2,'stop')$
$[Z,P,K] = ellip(.....)$
$[A,B,C,D] = ellip(.....)$

Tabla 9.5. Instrucciones para calcular los filtros

9.8 Ejemplos

Ejemplo 1

A continuación se presenta un programa realizado con Matlab para diseñar el filtro paso bajas cuyas especificaciones se muestran en la figura 9.3. Utilizamos la aproximación Butterworth, Chebychev, Chebychev Inversa y la aproximación de Cauer.

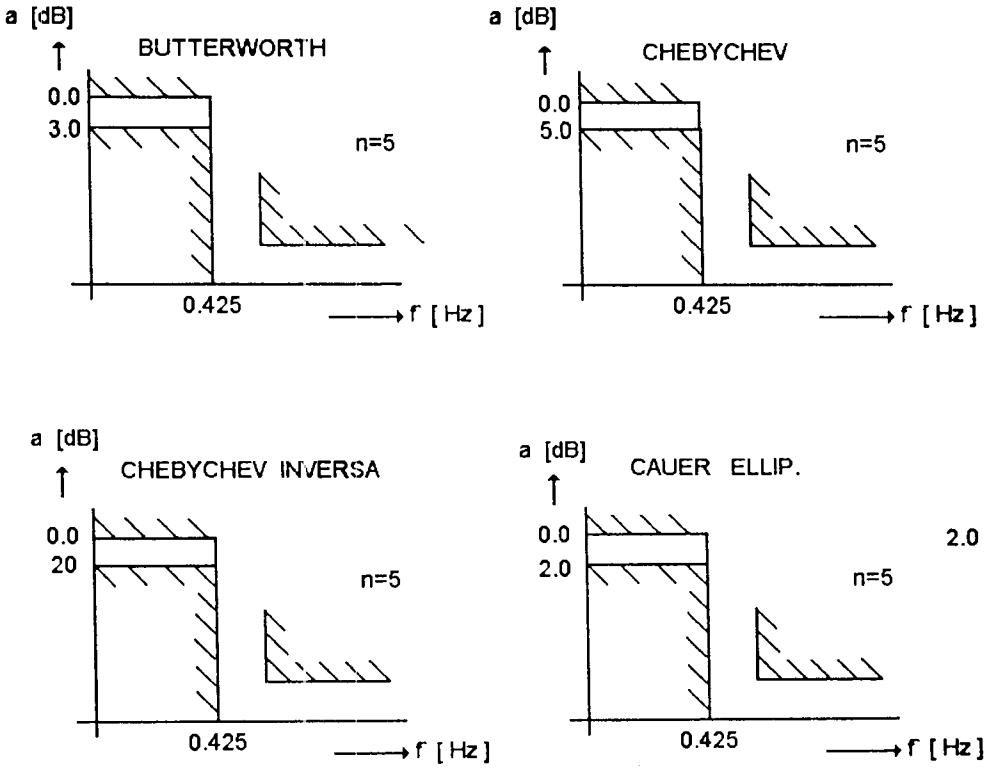


Figura 9.3. Especificaciones del filtro paso bajas

```
clg
[a,b]=butter(5,0.425);
[c,d]=cheby1(5,5,0.425);
[e,l]=cheby2(5,20,0.425);
[g,i]=ellip(5,2,15,0.425);
[h1,w]=freqz(a,b,100);
[h2,w]=freqz(c,d,100);
[h3,w]=freqz(e,l,100);
[h4,w]=freqz(g,i,100);
f=w*8000/pi;
subplot(221);
plot(f,20*log(abs(h1)));title('Filtro Butterworth');
xlabel('-->f');ylabel('20*log|h|');
```

```
plot(f,20*log(abs(h2)));title('Filtro Chebychev');
xlabel('-->f');ylabel('20*log|h|');
plot(f,20*log(abs(h3)));title('Filtro Chebychev inversa');
xlabel('-->f');ylabel('20*log|h|');
plot(f,20*log(abs(h4)));title('Filtro Cauer');
xlabel('-->f');ylabel('20*log|h|');
```

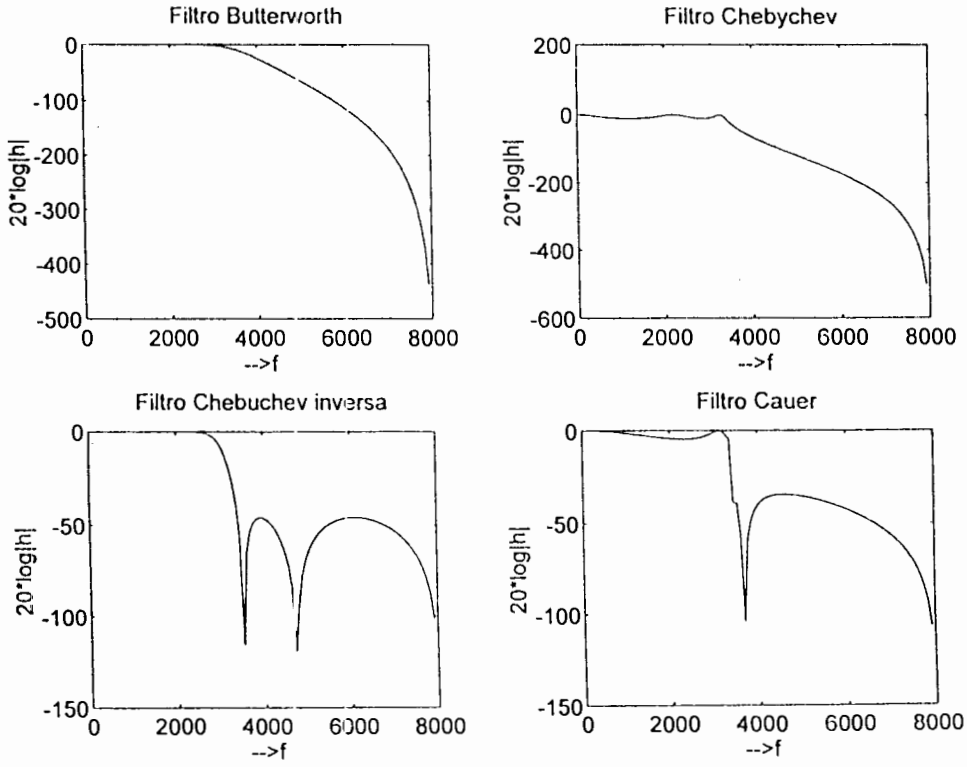


Figura 9.4. Análisis de los filtros paso bajas

En el programa, los coeficientes de la función de transferencia del filtro Butterworth son guardados en los vectores a y b, los del filtro Chebychev en c y d, los del filtro Chebychev inverso en e y l y los del filtro Cauer en g e i. Si teclamos a y después b, en la pantalla se escriben los coeficientes de H(z) de Butterworth. En los vectores $h1$, $h2$, $h3$, y $h4$ se guardan las partes imaginaria y real de la función de transferencia para cien frecuencias de w. Antes de graficar el resultado es necesario desnormalizar el eje w y obtenemos la frecuencia desnormalizada f. Con la instrucción plot obtenemos las atenuaciones de los filtros. El resultado se representa en la figura 9.4.

Ejemplo 2

En el siguiente programa se crea un vector de tiempo t que tiene 1000 puntos en el eje del tiempo y después se calcula un vector senoidal de f= 50 Hz con 1000 muestras. Los vectores $ur1$ y $ur2$ presentan un ruido que tiene una frecuencia de 200 Hz y 500 Hz. El vector yr es el vector de un señal senoidal $sen(2*pi*50*t)$ sumado con el ruido $ur1$ y $ur2$. En la gráfica de la señal yr no se puede ver la periodicidad de la señal original. Para identificarla necesitamos

quitar el ruido mediante los filtros. Ambos filtros en este ejemplo, son de Chebychev de orden 7. El primer filtro tiene la frecuencia del corte 200 Hz y el segundo 500 Hz. La séptima instrucción calcula los coeficientes b_i y a_i de la función de transferencia H(z) del primer filtro y la octava instrucción calcula los coeficientes c_i y d_i de la función de transferencia H(z) del segundo filtro. Los vectores z y v guardan la respuesta de las salidas de los filtros. Las últimas instrucciones son para graficar la respuesta. En la figura 9.5 podemos observar que el primer filtro no quita el ruido de frecuencia 200 Hz porque la frecuencia del corte del primer filtro es 250 Hz. Pero el segundo filtro cuya frecuencia del corte es 150 Hz nos da buen resultado. De la última gráfica, la señal se parece a la señal original, pero debido a los 7 elementos de retardo, la señal en la entrada del segundo filtro está retardada.

```
clg
t=0:0.001:1;
y=sin(2*pi*50*t);
ur1=5*sin(2*pi*200*t);
ur2=7*sin(2*pi*500*t);
yr=y+ur1+ur2;
[b,a]=cheby1(7,1,250/1000);
[c,d]=cheby1(7,1,150/1000);
z=filter(b,a,yr);
v=filter(c,d,yr);
subplot(221);
plot(t(1:50),y(1:50));title('la señal util ');
xlabel('tiempo');ylabel('la amplitud');
plot(t(1:50),yr(1:50));title('la señal con ruido');
xlabel('tiempo');ylabel('la amplitud');
plot(t(1:50),z(1:50));title('filtro Cheb. f0=250 Hz');
xlabel('tiempo');ylabel('la amplitud');
plot(t(1:50),v(1:50));title('filtro Cheb. f0=150 Hz');
xlabel('tiempo');ylabel('la amplitud');
```

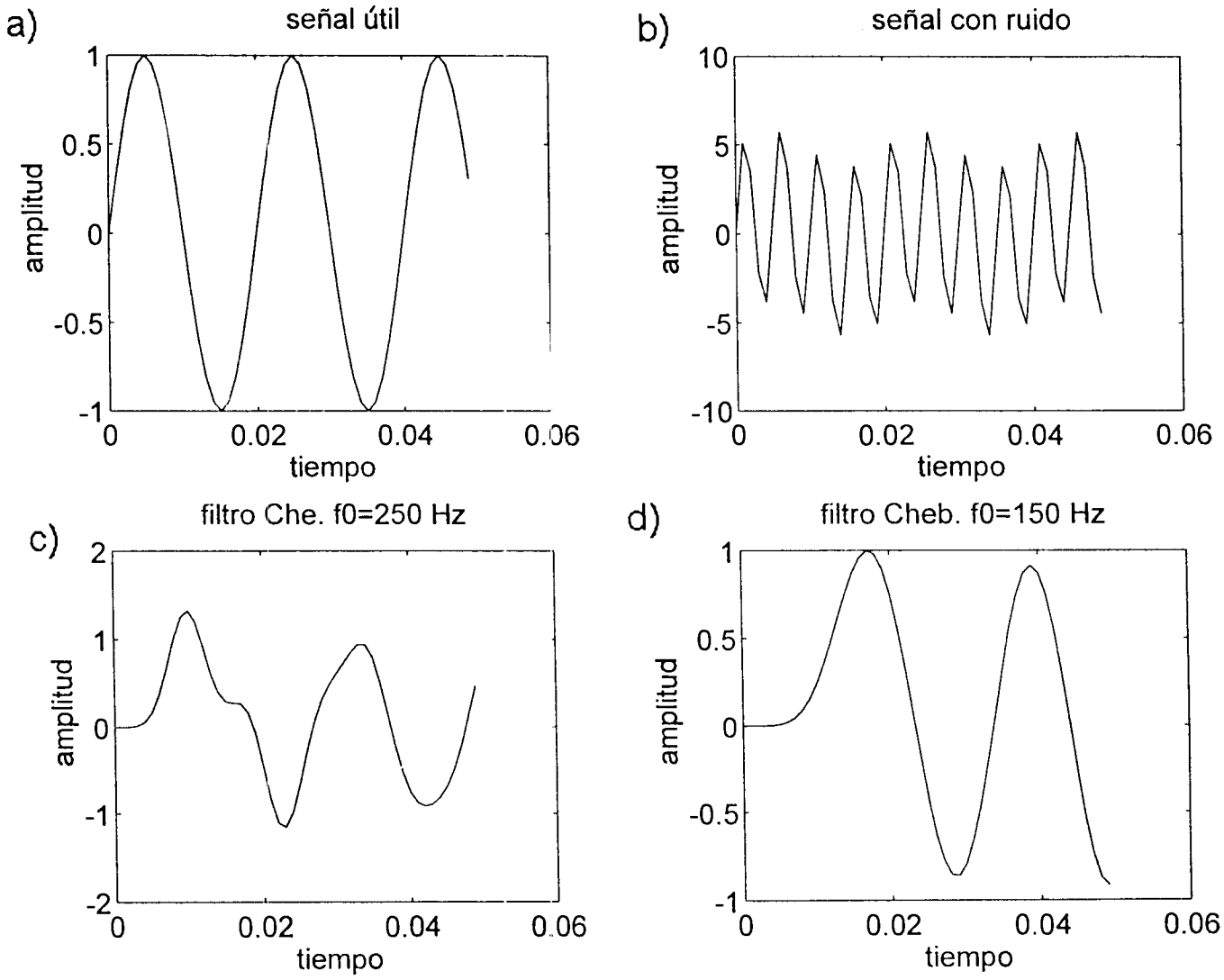


Figura 9.5. a) Señal original, b) Señal con ruido, c) Respuesta en la salida del primer filtro, d) Señal en la salida del segundo filtro

Bibliografía

[1] Texas Instruments : *User's Guide TMS320C5X*. U.S.A, Texas Instruments Incorporated, 1993.

[2] Texas Instruments : *User's Guide TMS320C3X*. U.S.A, Texas Instruments Incorporated, 1994.

[3] Texas Instruments : *User's Guide TMS320C2X*. U.S.A, Texas Instruments Incorporated, 1993.

[4] Texas Instruments : *Application Book - Telecommunication Application with the TMS320C5X DSPs*. U.S.A, Texas Instruments Incorporated, 1994.

[5] Texas Instruments : *Application Book - Digital Signal Processing Applications with the TMS320 Family*. U.S.A, Texas Instruments Corporation, 1989 Volumen 1.

[6] Texas Instruments : *Application Book - Digital Control Applications with the TMS320 Family*. U.S.A, Texas Instruments Corporation, 1991.

[7] Texas Instruments : *TMS320 Floating - Point DSP Assembly Language Tools*. U.S.A, Texas Instruments Corporation, 1995.

[8] Texas Instruments : *TMS320C5X DSP Starter Kit Users Guide*. U.S.A, Texas Instruments Corporation, 1994.

[9] Texas Instruments : *TMS320 Family Development Suport -Reference Guide*. U.S.A, Texas Instruments Corporation, 1994.

[10] Texas Instruments : *TMS320C32 Addendum to the TMS320C3X - Users Guide*. U.S.A, Texas Instruments Corporation, 1995.

[11] Texas Instruments : *TMS320C2X DSP Starter Kit Users Guide*. U.S.A, Texas Instruments Corporation, 1993.

[12] Rulph Chassaing. *Digital Signal Processing with C and the TMS320C25*. New York, John Wiley Sons, Inc., 1990.

[13] Rulph Chassaing. *Digital Signal Processing with C and the TMS320C30*. New York, John Wiley Sons, Inc., 1992.

[14] Parks, T.W - Burrus, C.S. *DFT - FFT and Convolution*. New Jersey, John Wiley Sons Inc., 1985.

[15] Parks, T.W - Burrus, C.S. *Digital Filter Design.* New Jersey, John Wiley Sons Inc., 1987.

[16] Parks, T.W - Hutchins, B.A. *A Digital Signal Processing Laboratory Using the TMS-320C25.* New Jersey, Prentice Hall and Texas Instruments, 1990.

[17] Graig, M. - Ewers, G. *A Simple Approach a Digital Signal Processing.* U.S A. Texas Instruments, 1994.

[18] Bateman, A. - Yates, W. *Digital Signal Processing Design.* New York, Computer Science Press Inc., 1989.

[19] Leon W, Couch. *Modern Communication Systems, Principles and Applications.* New Jersey, Prentice Hall., 1995.

[20] Sophocles J. Orfanidis. *Introduction to Signal Processing.* New Jersey, Prentice Hall., 1996.

Índice analítico