FACULTAD DE INGENIERÍA UNAM
DIVISIÓN DE EDUCACIÓN CONTINUA

DIPLOMADO EN MICROCONTROLADORES (SISTEMAS EMBEBIDOS)

CA 209 MÓDULO III PROCESADORES DIGITALES DE SEÑALES

TEMA:

INTRODUCCIÓN

AUTOR: M. I. LARRY ESCOBAR
INSTRUCTOR: ING. MIGUEL ANGEL MOLERO ARMENTA

DEL 11 AL 15 DE OCTUBRE DE 2004
PALACIO DE MINERÍA

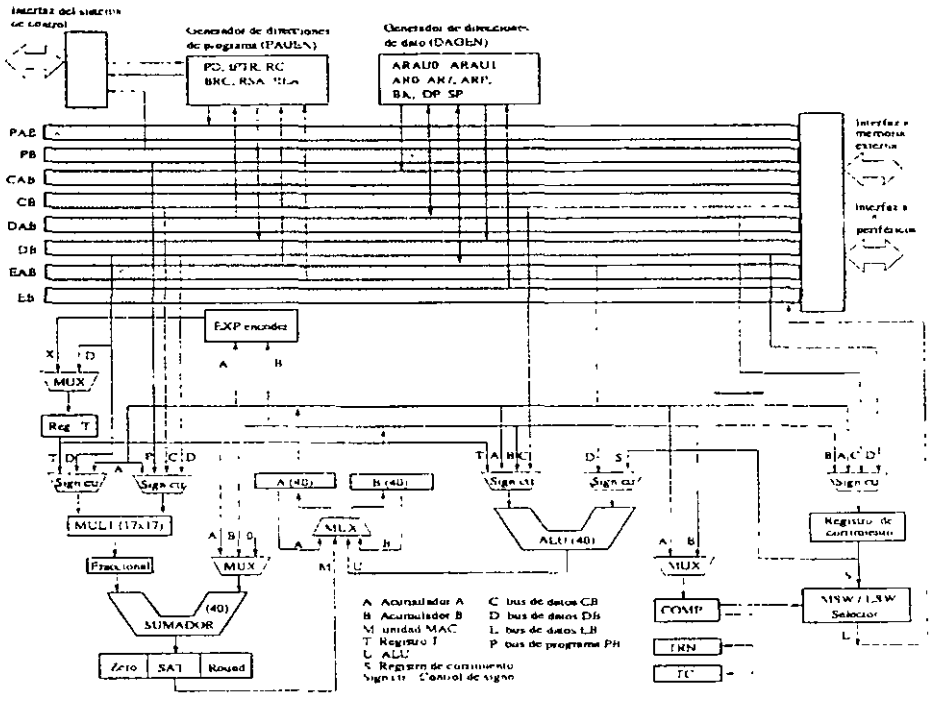
PALACIO DE MINERIA

Programa 2004

Facultad de Ingeniería

UNAM

Laboratorio de DSPs familias TMS320C5x y TMS320C54x



Por

M.I. Larry Escobar Salguero

Marzo del 2002

PRESENTACIÓN

La Facultad de Ingeniería ha decidido realizar una serie de ediciones provisionales de obras recientemente elaboradas por académicos de la institución, como material de apoyo para sus clases, de manera que puedan ser aprovechadas de inmediato por alumnos y profesores. Tal es el caso de la obra *Laboratorio de DSPs, familias TMS320C5x y TMS320C54x*, del M. en I. Larry Escobar Salguero.

Se invita a los estudiantes y profesores a que comuniquen a los autores las observaciones y sugerencias que mejoren el contenido de la obra, con el fin de que se incorporen en una futura edición definitiva.

ARQUITECTURAS DE DSP's

FAMILIA TMS320 Y EL TMS320C50

Por: Larry Escobar Salguero.

Facultad de Ingeniería

UNAM

Junio del 2000

EL PROCESAMIENTO DIGITAL DE SEÑALES

Y SUS APLICACIONES

Por: Larry Escobar Salguero.

Facultad de Ingeniería

UNAM

Junio del 2000

INTRODUCCION

Las presentes notas constituyen una breve introducción a las arquitecturas de Procesadores de Señales Digitales (DSP) de la familia TMS320 de la compañía Texas Instruments y en particular al DSP TMS320C50 de aritmética en punto entero.

El objetivo de este trabajo es aportar a la comunidad de la Facultad de Ingeniería UNAM un material de apoyo que de los elementos mínimos necesario de los DSPs y además motive a los alumnos de la materia Procesamiento Digital de Señales, alumnos de servicio social, tesistas y todo aquel interesado en esta área.

Es de hacer notar que este documento es una condensación de los manuales de usuario y de aplicaciones editados por Texas Instruments, así como un pequeño aporte de autor en su elaboración y uso de estos dispositivos, algunos ejemplos y aplicaciones con DSPs se pueden encontrar en el Manual del Laboratorio de Procesamiento Digital de Señales del mismo autor y editado por la FI, UNAM.

Es importante señalar que para entender estas notas y recibir el curso correspondiente es necesarios tener claro los conceptos de análisis de señales y sistemas, procesamiento digital de señales, filtros digitales, microprocesadores y lenguaje ensamblador.

Larry Escobar.
Profesor Asociado.
Facultad de Ingeniería, UNAM.
Junio del 2000.

Índice general

Índice de figuras

Introducción

Con los avances tecnológicos actuales, el procesamiento digital de señales (PDS) se ha convertido en una alternativa de solución a problemas de medición, control, filtrado, análisis de señales, análisis espectral, comunicaciones, etc. Aunque los fundamentos del PDS están dados por la teoría de señales y sistemas, las arquitecturas para PDS conforma una parte muy importante en el PDS en el sentido de verificación algorítmica software-hardware, en la solución a problemas reales y la realización de un prototipo final.

Desde hace varios años en la Facultad de Ingeniería, UNAM, se imparte la materia de Procesamiento Digital de Señales y se ha equipado un laboratorio con procesadadores de señales digitales (DSP) donde se desarrollan prácticas relacionadas con el PDS, con esa experiencia previa y la de otros cursos impartidos por el autor, se han elaborado las presentes notas con el objeto de que exista una guía básica para la experimentación en el laboratorio y desarrollo de proyectos mínimos por parte de los alumnos que reciben la materia. Por otro lado, estos apuntes pueden servir como una guía para los ayudantes del laboratarorio.

En esta *nueva versión del manual de prácticas de laboratorio de DSPs* [8], se han corregido algunos detalles y se han agregado más ejemplos de programas para el TMS320C50 (C50), se ha incluído un nuevo capítulo de la arquitectura del DSP TMS320C54x (C54x) con algunos ejemplos de programas. Este nuevo aporte se hace con la intención de que las nuevas generaciones de ingenieros tengan un conocimiento más profundo del estado del arte de los DSPs, ya que la punta de lanza de los DSPs de Texas Instruments (TI) son las familias TMS320C5000 (C5000) y TM320C6000 (C6000). Según la experiencia del autor, partiendo en forma ascendente desde el DSP C50, se puede migrar al C54x y la familia C5000 que sólo ha agregado una gran potencialidad en cuanto a periféricos y comunicación, pero la arquitectura base sigue siendo la del C54x. Además se pretende que en un futuro cercano se estén haciendo prácticas con estos últimos DSPs.

Este documento está conformado de seis partes, en la *primera* se hace una presentación de la conexión entre el Starter Kit del TMS320C50 (DSK) y la computadora personal (PC), la estructura básica de la realización de un programa, el ensamblado, directivas de ensamblado, como utilizar el ambiente de depuración y corrida de un programa. En la *segunda* se presenta una descripción breve del conjunto de instrucciones del DSP TMS320C50. En la *tercera* se presenta la forma de implementar un filtro digital en un DSP, específicamente con las instrucciones del TMS320C50. En la *cuarta* se presentan programas elementales para que el alumno los ensamble, los corra en el

ambiente del DSK observando la evolución de las variables, el estado del DSP y que a su vez aprenda a utilizar el ambiente. Estos programas empiezan con instrucciones elementales hasta instrucciones mas completas para llegar a elaborar programas más complicados que corren en tiempo real. La *quinta* consiste en programas propuestos que normalmente son similares a los proyectos que los alumnos desarrollan en clase con la asesoría del profesor, en la *sexta* se introduce la arquitectura del C54x y se muestran algunos programas.

Es importante señalar que el alumno que recibe este laboratorio debe haber cursado la parte teórica de la materia PDS, tener conocimiento de microprocesadores y lenguaje ensamblador para aprovechar al máximo el laboratorio.

Se agradece a la Facultad de Ingeniería de la UNAM y al departamento de publicaciones su apoyo a esta obra, así como las sugerencias hechas por algunos alumnos para ir depurando este material.

M.I. Larry Escobar Salguero.
Profesor Asociado
Facultad de Ingeniería, UNAM
e-mail larrye@servidor.unam.mx
http://electronica.fi-b.unam.mx/LARRY/LHES.htm
Marzo del 2002.

Capítulo 1

El starter kit del DSP TMS320C50

El starter kit del DSP TMS320C50 (DSK) es una tarjeta que se puede conectar al puerto serie de una computadora personal (PC) y se utiliza para la depuración de aplicaciones que ocupan poca memoria. La memoria disponible de esta tarjeta es únicamente la memoria RAM interna del TMS320C50 (C50 o C5x), es decir 10 K palabras. Un programa kernel está contenido en una memoria PROM de 32 K bytes, la cual se utiliza sólo para arrancar al TMS320C50 y crear el ambiente, en el mapa de memoria se reserva de la localidad 840h a 97Fh para el kernel del programa depurador y las localidades 000h a 07FFh son utilizadas para levantar al TMS320C50. Los vectores de interrupción se localizan en las direcciones 800h a 83Fh, ver figura 1.1.

El bloque B2 de memoria DARAM es reservado por el kernel como un buffer de registros de estado. Como el programa kernel está almacenado en SRAM, la memoria no puede ser configurada como memoria datos (bit RAM = 0 en el registro PMST)[1].

El circuito TLC32040 es un convertidor A/D y D/A de 14 bits con frecuencia de muestreo variable y comunicación vía puerto serial, por lo tanto está conectado directamente al puerto serie del TMS320C50. Además la tarjeta dispone de todos los pines externos del TMS32050 en su bus de expansión para conectar diseños externos propios del usuario. Para su funcionamiento basta con conectar la tarjeta a la PC a través de un cable RS232 con entrada DB9 y alimentarla con un transformador de 9 Vac a 250 mA, ver figura 1.2.

1.1. Creación del código fuente en lenguaje ensamblador

Para la creación del Código Fuente empleado en la operación del TMS320C50, se cuenta con el software que provee el fabricante, el DSK Ensamblador TMS320C50 que permite hacer

[1] Ver configuración y mapa de memoria en el manual de usuario [18], [19] o notas sobre la arquitectura del TMS320C50 del mismo autor [6], [8].

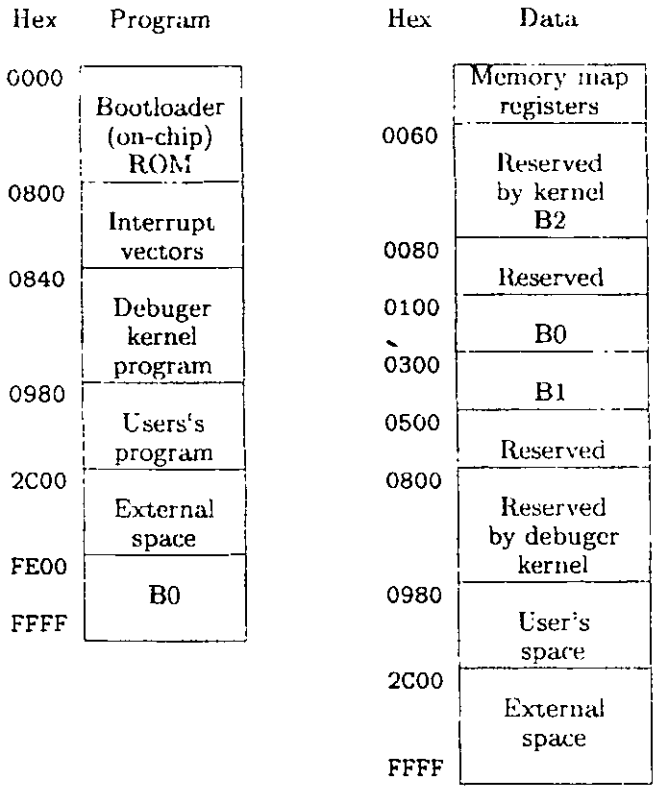


Figura 1.1: Mapa de memoria del DSK

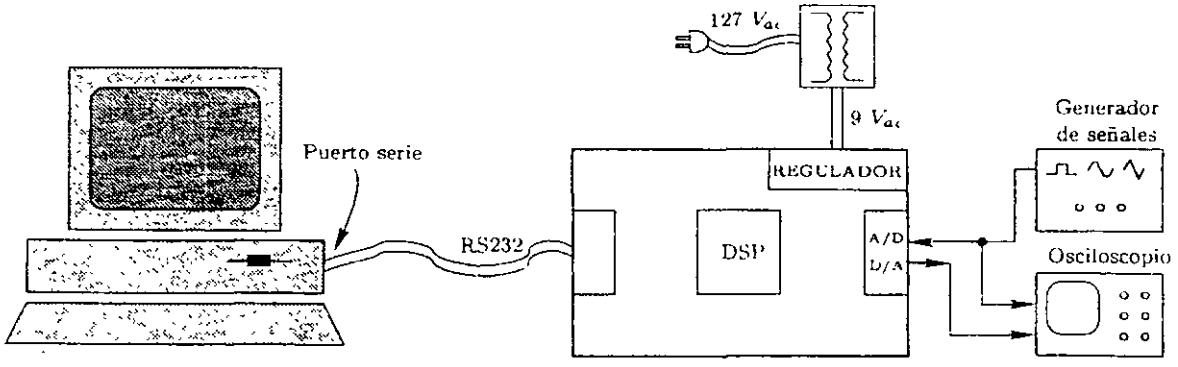


Figura 1.2: Conexión PC TMS320C50

la conversión de las instrucciones (*mnemónicos*) y directivas específicas en código fuente a código objeto ejecutable, este ensamblador no tiene necesidad de pasar por un proceso de ligado para la creación de código y para programas grandes tiene la opción de concatenar archivos utilizando las directivas `.include` y `.copy`.

Este software al generar el programa en código objeto, permite efectuar una simulación del mismo y efectuar la depuración del programa bajo un ambiente de ventanas.El lenguaje ensamblador del TMS320C50 consiste de códigos de operación llamados *mnemónicos*, que corresponden a las directivas de las instrucciones del lenguaje binario de máquina. Un programa en lenguaje ensamblador es llamado Programa Fuente y antes de poder ser ejecutado por el procesador, el programa fuente debe ser ensamblado para obtener el programa en código de máquina.

Cuando se edita un programa, se escribe un archivo fuente `.asm`, a través del programa ensamblador se genera el archivo `.dsk`, el cual contiene las direcciones con su correspondiente código de operación en formato hexadecimal. Con la opción de ensamblado -l se genera un archivo `.lst`, este contiene el archivo editado más una columna de direcciones y una de códigos que permite al usuario hacer una primera verificación de ensamblado, o posteriormente hacer una revisión más profunda.

1.2. El programa ensamblador

El proceso de ensamblado se realiza en dos lecturas del archivo fuente, en la primera el ensamblador mantiene un contador de localización, el cual define la dirección de memoria programa asignada a la palabra resultante en código objeto, la segunda, el ensamblador produce el código objeto que corresponde al código de operación asignándole su respectiva palabra.

Un programa fuente en lenguaje ensamblador consiste de expresiones que pueden contener directivas, instrucciones de máquina o comentarios. Estas expresiones pueden constar de cuatro campos:

— Etiquetas

— Comandos

— Operandos

— Comentarios

Cada campo es separado por uno o más espacios blancos. Las expresiones que contienen un asterisco (*) o un punto y coma (;) en la primera columna, corresponden a comentarios

y no afectan el programa. Una línea de una expresión fuente, puede ser tan larga como el formato fuente lo permita, sin embargo, el ensamblador trunca las líneas a 80 caracteres, por tanto los comentarios sólo pueden extenderse hasta la columna 80 sin ocasionar error. El ensamblador utiliza caracteres de entrada ASCII, es decir, que el programa fuente se puede editar en cualquier editor tipo ASCII.

Sintaxis de una expresión fuente:

[<ETIQUETA>][:] <MNEMÓNICO>[<OPERANDO>] ; [<COMENTARIOS>]

Donde los campos entre los paréntesis cuadrados pueden ser opcionales o no necesarios.

1.2.1. Campo de la etiqueta

Inicia en la primera columna de la expresión y puede contener hasta 6 caracteres, de los cuales el primero debe ser una letra, el resto puede ser alfanumérico. La etiqueta es opcional y se utiliza frecuentemente para indicar hacia donde debe transferirse el control del programa bajo cierta decisión. Cuando no se utiliza etiqueta, al menos la primera columna debe dejarse vacía.

Una expresión que consiste de sólo una etiqueta es válida, ya que a través de esta posibilidad, a la etiqueta se le asigna una constante numérica, por ejemplo:

```
etiqueta:   .set   número
```

donde el número puede representar el valor de localización del contador, permitiendo definir algunas variables a utilizar dentro del programa. La etiqueta se vuelve un símbolo que adquiere el valor que se le asigna, para utilizarlo dentro del programa. Los dos puntos pueden omitirse.

1.2.2. Campo del comando

Inicia un espacio en blanco después del campo de la etiqueta. En el caso de no existir etiqueta el comando inicia un espacio en blanco después de la primera columna. El campo del comando es terminado por uno o más espacios en blanco o tabuladores y no debe extenderse más allá del margen derecho.

El campo del comando puede contener:

- Mnemónicos del ensamblador o una instrucción de máquina.
- Macromnemónicos.
- Directivas del ensamblador.

1.2.3. Campo de operandos

Inicia un espacio en blanco después del campo del comando, puede contener:

Constantes precedidas del símbolo #.

- Una cadena de caracteres.
- Expresiones.

Los símbolos utilizados en el campo de operandos deben ser definidos en el ensamblador, usualmente por etiquetas. Cuando la instrucción tiene varios operandos éstos se separan por comas.

1.2.4. Campo de comentarios

Inicia después de un espacio en blanco al terminar el campo del operando o al terminar el campo del comando. Este campo puede extenderse hasta el final de la expresión fuente, los comentarios no tienen efectos en el ensamblado. Usualmente se escribe un punto y coma (;) antes del comentario de una expresión fuente, esto permite diferenciar donde empiezan los comentarios de la expresión. Los comentarios pueden contener todo tipo de caracteres y espacios. Una línea de sólo comentarios inicia en la columna uno con un asterisco o un punto y coma (;).

1.3. Ensamblado

Una vez editado el programa fuente (extensión asm) es necesario efectuar la conversión a un archivo con código ejecutable, este proceso se realiza con el ensamblador DSK. Cambiándose al directorio correspondiente (si el archivo `dsk5a.exe` no está en el PATH)

Sintaxis:

```
dsk5a [archivo(s)] [opciones]
dsk5a archivo asm"expresión"[asm"expresión"]
```

```
\..\>dsk5a  fuente     ; genera un archivo ejecutable extensión dsk si no existen errores
\..\>dsk5a  fuente -l  ; genera archivo dsk y un archivo de salida con extensión .lst
\..\>dsk5a  fuente -k  ; genera un archivo de salida no importando si existen errores
```

La opción asm permite definir expresiones en línea durante el ensamblado, esta es una posibilidad dado que el ensamblador del DSK no tiene un ligador.

Ejemplo de un archivo de salida .lst

```
00045 ---- 0001  UNO       .set      1
00046 ---- 041a  TOT       .set      1050   ; Total de Datos
00047 ---- ----  ************************************************
00048 ---- 0a00           .ps     00A00h
00049 ---- ----           .entry
>>>>> ENTRY POINT SET TO 0a00
00050 0a00 be41           SETC  INTM       ; des-habil. int. mask.
00051 0a01 be47           SETC    SXM      ;
00052 0a02 bf00           SPM     0        ; sin corrimiento de Preg
00053 0a03 5d07           OPL     #0034h,PMST ; Ram =1 OVLY = 1,
      0a04 0034
00054 ---- ----  *
00055 0a05 bf0d           LAR     AR5,#01850h  ; AR5 apunta a localidad 1850h
      0a06 1850
00056 0a07 bf0e           LAR     AR6,#TOT     ;
```

Donde las columnas significan:

Primera	número de línea de entrada en programa fuente
Segunda y tercera	código de la instrucción o valor de una definición
Cuarta	etiqueta
Quinta	instrucción (mnemónico)
Sexta	operandos
Séptima	comentarios

La directiva .set asigna un valor a una etiqueta. La directiva .entry le indica al ensamblador la dirección donde inicia el programa (donde lo debe de cargar en el depurador DSK).

Se pueden utilizar directivas de ensamblador para inicializar palabras en alguna localidad de memoria, ejemplo:

```
        .ps     0A00h       ; localidad donde apunta el contador de programa
INICIO  .word   0BCh,3,0FFh
```

Cuando una instrucción o un directiva es referenciada, la etiqueta es reemplazada con la dirección de la etiqueta localizada en memoria.

1.3.1. Constantes

El ensamblador puede manejar cuatro tipos de constantes:

Decimales enteras	Ejem. 1000 , −32768, 25
Hexadecimal entera	Ejem. 3E8h, 0FFh,8000h
Binaria entera	Ejem. 01100b, 10101b, −0101b
Character	Son caracteres ASCII encerrados entre comillas.

1.4. Directivas del ensamblador

Las directivas del ensamblador son comandos que proveen al proceso de ensamblado datos y control para ayudar al proceso de ensamblado y hacer más fácil la programación, esta directivas no generan código de programa. A continuación se explican brevemente cada una agrupándolas de acuerdo a su función.

1.4.1. Símbolos

Los símbolos pueden fijarse (directiva `.set`) con el valor de una constante y utilizarse dentro del programa, o indicar la localización de una variable en memoria dato (directivas `.word`, `.int`, etc) [19].

Ejemplo:

```
N       .set    512
N1      .set    10          ; N=10 , no ocupa memoria
        LAR     AR0,#N      ; carga en el registro auxiliar AR0 el valor de N
        LACC    #N1         ; carga en el acumulador el valor de N1
```

1.4.2. Directivas que definen secciones

`.data`	Ensambla en memoria dato. La memoria dato usualmente contiene datos iniciales.
`.ds [add]`	Ensambla en memoria dato (inicializando una dirección).
`.entry [add]`	Inicializa la dirección de contador de programa cuando se carga el archivo.
`.ps [add]`	Ensambla en memoria programa (inicializando una dirección [add]).
`.text`	Ensambla en memoria programa. La sección `text` usualmente contiene el código ejecutable.

1.4.3. Directivas que referencían a otros archivos

`.copy "archivo"`	Incluye expresiones de otro archivo en el archivo actual.
`.include "archivo"`	incluye expresiones de otro archivo en el archivo actual.

El archivo puede incluir su path completo, de lo contrario el ensamblador sólo buscará estos archivos en el directorio actual.

1.4.4. Directivas condicionales

Permiten efectuar un ensamblado condicional de acuerdo a la evaluación de una expresión La directiva `if` no es anidable [19].

`.if`	Inicia el bloque de ensamblado condicional si la expresión de if es verdadera.
`.else`	Inicio de bloque condicional si la expresión de if es falsa.
`.endif`	Final del ensamblado condicional.

1.4.5. Directivas que inicializan constantes (Datos y memoria)

Cada una de esta directivas a excepción de `.byte` y `.string` alinean el dato a los límites de las palabras de 16 bits en memoria [19], todas esta directivas si ocupan memoria y siempre deben de estar dentro del segmento de datos (`.ds`).

`.bfloat val1,...,valn`	Inicializa uno o más valores (valn) en punto flotante, 16 bits de exponente y 32 bits de mantisa en complemento a dos.
`.byte val1,...,valn`	Inicializa una o más palabras sucesivas de 8 bits en la sección actual.
`.double val1,...,valn`	Inicializa valores en punto flotante de doble precisión IEEE (64 bits).
`.efloat val1,...,valn`	Inicializa uno o más valores en punto flotante con 16 bits de exponente y 16 bits de mantisa en complemento a dos.
`.float val1,...,valn`	Inicializa uno o más valores en punto flotante en precisión simple IEEE (32 bits).
`.int val1,...,valn`	Inicializa uno o más enteros de 16 bits.
`.long val1,...,valn`	Inicializa uno o más enteros de 32 bits.
`.lqxx val1,...,valn`	Inicializa uno o más valores enteros de 32 bits en dos complemento con el punto decimal desplazado xx bits del LSB.
`.qxx val1,...,valn`	Inicializa uno o más valores enteros de 16 bits en dos complemento con el punto hipotético desplazado xx bits del LSB (formato Qxx).
`.space` tamaño en bits	Reserva el tamaño de bits en la sección actual.
`.string "string1",..."stringn"`	Inicializa una o más cadenas.
`.tfloat val1,...,valn`	Inicializa uno o más valores de 32 bits de exponente y 64 bits de mantisa en dos complemento.
`.word val1,...,valn`	Inicializa uno o más enteros de 16 bits.

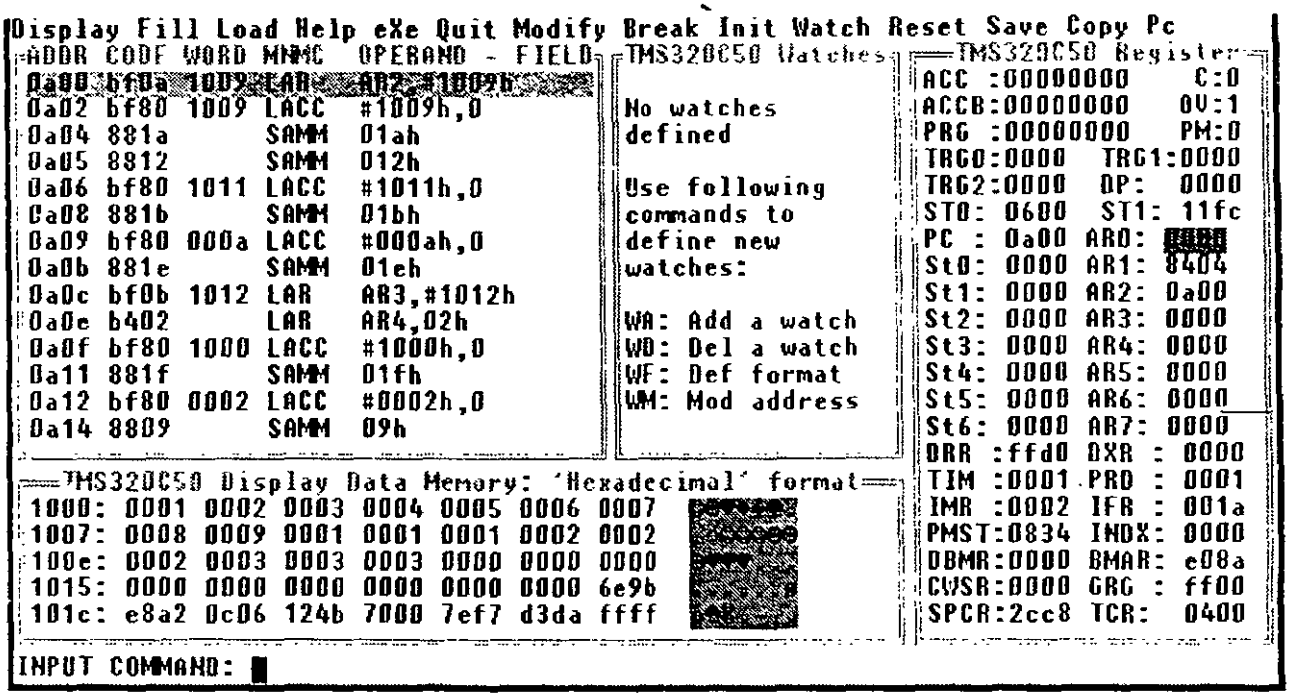
1.4.6. Otras Directivas

`.end`	Fin del programa.
`.listoff`	Anula la opción del archivo .lst de salida al efectuar el ensamblado.
`.liston`	Reinicia la opción del archivo .lst de salida.
`.set`	Iguala un símbolo a un valor.
`.mmregs`	Define los nombres simbólicos de los registros mapeados para utilizarlos dentro del programa. Pone los registros mapeados en una tabla de símbolos.

1.5. Depurador y ensamblador

El DSK ensamblador y el depurador son interfaces que ayudan al usuario a desarrollar, probar y depurar un programa en lenguaje ensamblador

Para invocar el depurador, cambiarse al directorio (si este no está en el PATH) donde está instalado el software e invocar el programa **dsk5d -c1** (donde -c1 indica que se comunica con la PC a través del puerto serie uno, la opción -c2 indicaría el puerto serie dos, al omitir la opción -cx, se asume que es el puerto serie uno), si no hubo ningún problema aparecerá en pantalla:



Con las ventanas:

— Código desensamblado (superior izquierda).
- Primera columna: dirección en memoria programa.
- Segunda y tercera: código de instrucción.
- Cuarta: Mnemónico de instrucción.
- Quinta: Operandos de la instrucción.

— Watch (centro).
En esta ventana se puede seleccionar una dirección de memoria y observar los cam-

bios durante la ejecución del programa. Se puede cambiar el formato numérico de las variables.

— Registros (derecha).
Muestra el contenido de los registros principales del C5x en formato hexadecimal.

— Memoria dato (inferior).
Muestra un bloque de memoria dato. En esta ventana se puede visualizar cualquier otro segmento de memoria dato, modificar la memoria, cambiar el formato de presentación (entero, hexadecimal, flotante, Q15, etc.)

En la parte superior se tiene una *barra del menú* que se activa al presionar la tecla iluminada correspondiente apareciendo un submenú, en la parte inferior aparece la secuencia de comandos de entrada. Para la utilización de este ambiente, el usuario puede ir al menú de ayuda para ver todas las opciones disponibles.

El comando `dsk5d` puede presentar información adicional al proveerle algunas opciones en la línea:

`? o H`	Despliega una lista de las opciones disponibles.
`bxxx`	Selecciona la razón de baudaje (`xxx` = 4800, 9600, 19200, 38400, 57600).
`com1, com2, c1, c2`	Selecciona el puerto serie de comunicación con la PC.
`eadd`	Define el punto de entrada del programa.
`i`	Selecciona el nivel lógico para el reset de DTR (data terminal ready).
`l`	Selecciona el tamaño de la pantalla EGA/VGA (default: 43 líneas × 80 caracteres).
`s`	Selecciona la longitud de pantalla (default: 25 líneas × 80 caracteres).

Nota:
La utilización del ambiente de depuración y sus comandos es explicado en el laboratorio por el profesor con el objeto de que los comandos se aprendan en la práctica. También existe en el menú principal una opción de ayuda que nos define todos los comandos del ambiente.

Capítulo 2

Lista de instrucciones del DSP TMS320C50

En esta sección se describen en forma breve la mayoría de las instrucciones del TMS320C50 en el sentido que el lector tenga una idea de estas instrucciones y pueda iniciarse en la realización de las prácticas y programas propuestos. Para mayor detalle se debe referir al manual de usuario del TMS320C50 [19] ya que muchas instrucciones pueden utilizar direccionamiento inmediato, indirecto, directo o presentan muchas posibilidades de utilización.

Algunas abreviaturas a utilizar:

ACC	Acumulador de 32 bits.
ACCB	Acumulador buffer de 32 bits.
ACCL	Parte baja del acumulador (0-15 bits).
ACCH	Parte alta del acumulador (16-31 bits).
ALU	Unidad aritmética lógica.
ARAU	Unidad aritmética de registro auxiliar.
AR	Registro auxiliar.
ARP	Registro apuntador de registros auxiliares (de 3 bits).
BMAR	Registro de direccionamiento dinámico.
C	Bit de carry.
CNF	Bit de control de configuración del bloque B0 de RAM interna.
dma	Dirección de memoria dato.
DP	Registro apuntador de página (de 9 bits), este registro está incluido en el registro de estado cero (ST0) bits 0 al 8.
dst	Operando destino.
HM	Bit de Modo Hold.

INTM	Bit de modo de interrupción, 0 = habilitado y 1 = deshabilitado.
LSB	Bit menos significativo.
MSB	Bit más significativo.
OVM	Bit de modo de sobreflujo.
PC	Contador de programa.
PLU	Unidad lógica paralela.
pma	Dirección de memoria programa.
src	Operando fuente.
SXM	Bit de extensión de signo.
TC	Bit de bandera de control de prueba.
TOS	Localidad más alta del stack.
XF	Bit de estado del pin externo XF.

El valor del código de bit correspondiente a la localidad de memoria especifica (utilizado con instrucción BIT y BITT):

	Bit	Código
(LSB)	0	1111
	1	1110
	2	1101
	3	1100
	4	1011
	5	1010
	6	1001
	7	1000
	8	0111
	9	0110
	10	0101
	11	0100
	12	0011
	13	0010
	14	0001
(MSB)	15	0000

Instrucción	Descripción
ABS	Da el valor absoluto del acumulador. Si el acumulador es menor que cero, este será reemplazado por el complemento a dos del valor anterior. En caso contrario, esta instrucción no afecta al acumulador. No utiliza operadores.
ADCB	El contenido del acumulador buffer (ACCB) y el valor del bit de acarreo (C) son sumados al acumulador. El bit de acarreo se pone en uno si el resultado de la suma genera un acarreo. No utiliza operadores.
ADD	Suma al acumulador el contenido de una dirección de memoria. El resultado se almacena en el acumulador.
ADDC	El contenido del acumulador buffer (ACCB) es sumado al acumulador. El bit C se pone en uno si el resultado de la suma genera acarreo. No utiliza operadores.
ADDS	El contenido de la localidad de memoria y el valor del bit de acarreo son sumados al acumulador con extensión de signo suprimido. El bit de acarreo es afectado de manera normal.
ADDS	Suma a la parte baja del acumulador el contenido de una dirección de memoria sin tomar en cuenta el signo. El resultado se almacena en el acumulador.
ADDT	El valor de la memoria de dato es corrido hacia la izquierda de 0 a 15 bits especificado por TREG1 y sumado al acumulador, reemplazando el contenido del acumulador. El dato es tratado como un número no signado de 16 bits. C es afectado de forma normal.
ADRK	El valor de una constante inmediata de 8 bits es sumada, justificado a la derecha, en el actual registro auxiliar (como se especifica por el actual ARP) reemplazando el contenido del registro auxiliar con el resultado. La suma se lleva a cabo en la unidad ARAU con el valor inmediato tratado como un entero positivo de 8 bits. Observe que todas las operaciones en el registro auxiliar son signadas.
AND	Realiza una operación lógica AND de la parte baja del acumulador con el contenido de una dirección especificada de memoria o una constante de 16 bits. La parte alta del acumulador se llena de ceros.
ANDB	Se realiza una operación lógica AND entre el valor del acumulador y el acumulador buffer(ACCB). El resultado es colocado en el acumulador mientras el acumulador buffer no se ve afectado. No utiliza operadores.

APAC	Suma al acumulador el contenido del registro producto P. El resultado se almacena en el acumulador. El registro P es corrido antes de la suma tal como se especifica en el registro PM. No utiliza operadores.
APL	Si es especificada una constante inmediata larga, se hace un AND de ésta con el valor de memoria de dato dma. De otra manera, se aplica un AND al valor de la memoria de dato y al contenido del registro de manipulación dinámico de bits (DBMR). En los dos casos, el resultado es escrito de nuevo en la localidad de memoria dato direccionada, mientras el contenido del acumulador no se ve afectado. Si el resultado de la operación lógica AND es 0, entonces el bit TC se pone en 1, si sucede lo contrario, el bit TC se pone en cero.
B	Salta incondicionalmente a una localidad especificada del programa.

Ejemplo:

```
B     191,*+,AR1
```

El valor 191 es cargado en el contador de programa, y el programa continúa ejecutándose desde esta localidad. El registro auxiliar actual es incrementado por 1, y el ARP apunta al registro auxiliar AR1

BACC	El control es transferido a la dirección especificada por la parte baja del acumulador. Corresponde a un goto computado.
BACCD	Salto con retardo a una localidad especificada por el ACC. Una instrucción de dos palabras o dos instrucciones de una palabra seguidas de la instrucción de salto son buscadas desde la memoria de programa y ejecutadas antes de que el salto se lleve a cabo.

Ejemplo:

```
BACC            ; (considere que el acumulador ACC
                ;  contiene el valor 191, por ejemplo)
```

El valor 191 es cargado en el contador de programa (PC), es decir, que el programa salta a la localidad 191.

Ejemplo: Salto con retardo

```
BACCD           ; (considere que el acumulador ACC
                ;  contiene el valor 191, por ejemplo)
MAR     *+,AR1
LPD     #5
```

Después que el actual ARi, ARP, y DP son modificados como se especifica las instrucciones, la ejecución del programa continua desde la localidad 191. Es decir, que antes de efectuar el salto se ejecutan las dos instrucciones siguientes (el retardo).

BANZ	Salta a la localidad especificada si el contenido del registro auxiliar en uso no es cero. Cuando se ejecuta esta instrucción, el registro auxiliar es decrementado en una unidad. Si el registro auxiliar en uso es cero el programa ejecuta la siguiente instrucción.
BCND	Se realiza un salto a la dirección de memoria de programa "pma" si se cumplen las condiciones especificadas. Condiciones: EQ ACC=0 LT ACC<0 GT ACC>0 C C=1 OV OV=1 BIO BIO=0 NTC TC=0 NEQ ACC≠0 LEQ ACC≤0 GEQ ACC≥0 NC C=0 NOV OV=0 TC TC=1 UNC Sin condición

Ejemplo:

```
BCND    PGM50,LEQ,C
```

Si el contenido del acumulador es menor o igual que cero y el bit de acarreo está encendido, la dirección de programa PGM50 es cargada en el contador de programa, y el programa continúa ejecutándose desde esa localidad. Si ambas condiciones no se cumplen, la ejecución continúa desde la localidad PC+2. No todas las combinaciones de condiciones tienen sentido.

Ejemplo:

```
BCNDD   PGM200,OV  ; salto condicional con retardo
MAR     *+,AR1
LDP     #5
```

Después que el actual ARi, ARP, y DP son modificados como se especifica, la ejecución del programa continúa desde la localidad PGM200 si la bandera de sobreflujo (OV) en el registro de estado STO está encendida. Si la bandera no lo está, la ejecución continúa en la siguiente instrucción a LDP.

BIT	La instrucción BIT copia el bit especificado del valor de memoria de dato al bit del registro de estado ST1.
BITT	Esta instrucción es igual que la anterior (BIT), sólo que el bit de prueba está especificado por el registro TREG2.
BLDD	Una palabra en memoria dato apuntada por el *src* (fuente) es copiada a un espacio de memoria dato que es apuntado por el *dst* (destino).
BLDP	Una palabra en la memoria de dato es copiada a una palabra en el espacio de memoria de programa que apunta el registro BMAR.
BSAR	Ejecuta un corrimiento aritmético a la derecha de 1 a 16 bits en el acumulador en un solo ciclo.
CALA	La instrucción CALA se utiliza para realizar llamadas computadas a subrutinas. El PC es incrementado y almacenado en el stack. Luego el contenido de los 16 bits menos significativos del acumulador se cargan en el PC.
CALL	Llamada a subrutina. El contador del programa es incrementado y almacenado en el stack. Entonces la dirección especificada por "pma" es cargada en el PC.

CC	Llamada condicional a subrutina. El control se transfiere al "pma" si se cumplen las condiciones especificadas. Se utilizan las mismas condiciones de la instrucción BNCD.

Ejemplo:

```
CC      PGM100,LEQ,C
```

Si el contenido del acumulador es menor o igual que cero y el bit de acarreo está encendido, la dirección PMG100 es cargada en el contador de programa, y el programa continúa ejecutándose desde esa localidad. Si las condiciones no se cumplen, la ejecución continúa a partir de la siguiente instrucción de CC.

Ejemplo:

```
CCD     PGM100,LEQ,C    ; llamada condicional con retardo
MAR     *+,AR1
LDP     #5
```

El actual ARi, ARP, y DP son modificados como se especifica. Si el contenido del acumulador es menor o igual que cero y el bit de acarreo está encendido, la dirección de la instrucción siguiente a LDP es cargada en el apuntador de stack y la ejecución del programa continúa a partir de la localidad PMG100. Si las condiciones no se cumplen, la ejecución continúa desde la instrucción siguiente a LDP.

CLRC	El bit de control especificado es puesto a cero lógico. Observe que la instrucción LST puede ser usada para cargar ST0 y ST1. Los bits de control son : C, CNF, HM, INTM, OVM, SXM, TC y XF.

Ejemplo:

```
CLRC    TC      ; Pone en cero el bit TC
```

CMPL	Complemento del acumulador. El contenido del acumulador es reemplazado con una inversión lógica (complemento a uno). El bit de acarreo no se ve afectado. No utiliza operadores.

CPL	Compara una constante de 16 bits con una localidad de memoria. Si las dos cantidades involucradas en una comparación son iguales, el bit TC se pone en 1; en cualquier otro caso, el bit TC se pone en cero.
CRGT	El contenido del acumulador (ACC) es comparado con el contenido del acumulador buffer (ACCB). El valor mayor (signado) es cargado en ambos registros. Si el contenido del acumulador es mayor o igual que el contenido del acumulador buffer, el bit de acarreo se pone en 1. El bit de acarreo se pone en cero en cualquier otro caso. No utiliza operadores.
CRLT	El contenido del acumulador (ACC) es comparado con el contenido del acumulador buffer (ACCB). El valor menor (signado) es cargado en ambos registros. Si el contenido del acumulador es menor que el contenido del acumulador buffer, el bit de acarreo se pone en 1. El bit de acarreo se pone en cero en cualquier otro caso. No utiliza operadores.
DMOV	El contenido de la dirección de memoria especificada es copiada en el contenido de la siguiente dirección más alta. Esta instrucción opera solo sobre memoria RAM interna. Esta instrucción puede utilizarse en la implementación de retardos (Z^{-1}).
EXAR	El contenido del acumulador es cambiado (switchado) con el contenido del acumulador buffer (ACCB) y viceversa. No utiliza operadores.
IDLE	La instrucción obliga a que el programa permanezca en estado de espera hasta que ocurra una interrupción no mascarable o se presente un reset. El contador de programa PC es incrementado sólo una vez, y el dispositivo permanece en estado de espera hasta que se presenta la interrupción. El puerto serial y el timer permanecen activos. No utiliza operadores.

Ejemplo:

```
IDLE    ; El procesador permanece en estado de espera
        ; hasta presentarse un reset o una interrupción mascarable.
```

IDLE2	Similar a IDLE, solo que el puerto serial y el timer no están activos. No utiliza operadores.
IN	La instrucción IN lee datos de un periférico y coloca estos en memoria de datos, esto toma 2 ciclos de instrucción.

Ejemplo:

```
IN      DAT7,PA5
```

Lee una palabra de un periférico en la dirección del puerto PA5 y la almacena en la localidad de memoria DAT7.

Ejemplo:

```
IN      *,PA0
```

Lee una palabra de un periférico en la dirección del puerto PA0 y la almacena en la localidad de memoria especificada por el actual registro auxiliar.

INTR	Es una interrupción por software que transfiere el control del programa a la dirección de memoria de programa especificada por el vector de interrupción "k".
LACB	El acumulador es cargado con el contenido del acumulador buffer ACCB. No utiliza operadores.
LACC	Carga al ACC con el contenido de la dirección de memoria dato o una constante de 16 bits y efectúa el número de corrimientos especificados a la izquierda. Durante el corrimiento los bits de orden bajo son llenados con ceros y los bits de orden alto son signados si el bit SXM = 1.
LACL	El contenido de una localidad de memoria o una constante de 8 bits es cargado en el ACCL, el ACCH es llenado con ceros. Esta instruccion no utiliza la extensión de signo (no importa el valor del bit SXM).
LAMM	La parte baja del acumulador (ACCL) es cargada con el contenido del registro de memoria mapeada. La parte alta del acumulador (ACCH) es cargada con ceros. Los 9 bits más significativos de la dirección de memoria son llenados con ceros independientemente del valor del registro apuntador de página (DP).
LAR	Carga un registro auxiliar con el contenido de una localidad de memoria o una constante de 16 bits.
LDP	Carga el registro apuntador de página con una constante de 9 bits ó el contenido de una localidad de memoria
LMMR	El registro de memoria mapeada es apuntado por los 7 bits más bajos del valor de la dirección de memoria de dato, es cargado con el contenido de la localidad de memoria de datos direccionada por una dirección de 16 bits.

LPH	Los bits de mayor orden (31-16) del registro P son cargados con el contenido de la memoria dato. Los bits de menor orden no son alterados.
LST	El contenido de un dato en memoria es cargado en un registro estado (ST0 o ST1). La instrucción opuesta sería SST, que almacena el contenido de un registro estado (ST0 o ST1) en una localidad de memoria.
LT	LT carga al registro TREG0 con el contenido de la localidad de memoria dato. El dato cargado en TREG0 constituye un factor de una multiplicación a realizar.
LTA	El contenido de la dirección de datos en memoria especificado es cargado en el registro TREG0 y el registro P que contiene el producto previo, es sumado al acumulador.
LTD	El registro TREG0 es cargado con el contenido de la dirección de datos especificada, el contenido del registro P es sumado al acumulador y el contenido de la dirección especificada de memoria es copiado a la próxima dirección de datos de memoria.
LTP	TREG0 es cargado con el contenido de la localidad de dirección de memoria de dato y el registro producto corrido como lo define PM es almacenado en el acumulador.
LTS	TREG0 es cargado con el contenido de la localidad de la memoria de dato. El registro producto corrido como lo define PM, es restado del acumulador. El resultado es colocado en el acumulador.
MAC	Multiplica el valor de memoria de dato (especificada por el dma) por el valor de la memoria de programa (especificado por pma). También es sumado el producto previo corrido como lo define PM al acumulador.
MACD	Multiplica un valor de memoria de dato (especificado por el "dma") por un valor de memoria de programa (especificado por el pma). También suma el producto anterior con un corrimiento especificado por PM al acumulador. El contenido de memoria dato especificado es copiado en la próxima dirección de memoria dato.
MADD	La instrucción multiplica un valor de memoria de dato (especificado por el dma) por un valor de memoria de programa (especificado por el pma). La dirección de memoria de programa está contenida en el registro de memoria BMAR. Esto facilita un direccionamiento dinámico de tablas de coeficientes. También suma el producto anterior con un corrimiento especificado por PM al acumulador. Además el contenido de memoria dato especificado es copiado en la próxima dirección de memoria dato.

MADS	La instrucción multiplica un valor de memoria de dato (especificado por el dma) por un valor de memoria de programa (especificado por el pma). También suma el producto anterior con un corrimiento especificado por PM del acumulador. El pma es especificado por el contenido del registro BMAR, en vez de una constante inmediata larga. Esto permite el direccionamiento dinámico de tablas de coeficientes.
MAR	Esta instrucción utiliza direccionamiento en modo indirecto para incrementar/decrementar el registro auxiliar en uso y cambiar el apuntador de registros auxiliares.
MPY	El contenido de el registro TREG0 es multiplicado por el contenido de la dirección de datos en memoria, el resultado es almacenado en el registro P. El direccionamiento inmediato corto multiplica el registro por una constante de 13 bits con signo.
MPYA	El contenido de TREG0 es multiplicado por el contenido de la localidad de memoria de dato. El resultado se almacena en el registro P. El producto previo corrido como lo define PM es sumado al acumulador.
MPYS	El contenido de TREG0 es multiplicado por el contenido de la localidad de memoria de dato. El resultado se almacena en el registro P. El producto previo corrido como lo define PM es restado al acumulador y el resultado se almacena en el acumulador.
MPYU	El contenido sin signo del registro TREG0 es multiplicado por el contenido sin signo de la dirección de localidad de memoria. El resultado se almacena en el registro P. El multiplicador actúa como un multiplicador signado de 17 por 17 bits, con el MBS de ambos operandos forzado a cero. Esta instrucción es particularmente útil para cálculos de productos de precisión múltiple.
NEG	El contenido del acumulador es reemplazado por el complemento aritmético a dos. El bit OV se enciende cuando se realiza la negación de 80000000h. Si OVM = 1, el contenido del acumulador es reemplazado por 7FFFFFFFh. Si OVM = 0, el resultado es 80000000h. El bit de acarreo C es reiniciado a cero por esta instrucción para todos los valores diferentes de cero del acumulador, y es puesto a uno si el acumulador es igual a cero.
NMI	Este operador forza al contador de programa a un vector de interrupción no enmascarable localizado de memoria programa. La instrucción tiene el mismo efecto que el de una interrupción no enmascarable por hardware. No utiliza operadores.

Ejemplo:

```
NMI
```

El control es transferido a la localidad de memoria de programa 24h y se coloca el valor PC+1 en el apuntador de pila (STACK).

NOP	No se realiza ninguna operación. La instrucción sólo afecta al contador de programa (PC), esta instrucción es muy útil para crear pipeline y retardos de ejecución. No utiliza operadores. **Nota:** NOP es utilizado como un "pad" o instrucción temporal durante el desarrollo del programa.
OPL	Si una constante inmediata es especificada, se realiza la función lógica OR con el valor de la dirección de memoria de dato. Si no se especifica la constante, el segundo operando a la operación OR es el contenido del registro de manipulación dinámica de bits (DBMR). El resultado de la operación siempre es escrito de nuevo en la localidad de memoria especificada. El contenido del acumulador no es afectado. Si el resultado de la operación OR es cero, entonces el bit TC es puesto a uno; en cualquier otro caso, el bit TC es puesto a cero.
OR	El ACCL efectúa una operación lógica OR con una constante de 16 bits con corrimiento o con el contenido de el dato especificado en memoria. El resultado es almacenado en el acumulador.
ORB	Se realiza una operación lógica OR entre el contenido del acumulador y el acumulador buffer ACCB, el resultado se almacena en el acumulador. No utiliza operadores.
OUT	La instrucción OUT transfiere los datos de memoria de un periférico externo (puerto).
PAC	Carga el acumulador con el registro P con corrimiento especificado por PM. No utiliza operadores.
POP	El contenido de la parte alta de stack (TOS) es cargado en la parte baja del acumulador (ACCL). El siguiente elemento del stack se sube a la parte alta del stack.
POPD	El contenido de la parte alta del stack es transferido en la localidad de memoria de dato especificada por la instrucción.
PUSHD	El valor de la localidad de memoria de dato especificada es transferido a la parte alta del stack.

PUSH	El contenido de la parte baja del acumulador (ACCL) es agregado en el TOS del stack. Si el stack está lleno, el contenido de la parte inferior del stack se pierde.
RET	Retorno de subrutina. El contenido de la parte alta del stack se copia al contador de programa. Entonces el stack se corre un nivel. Es usada con CALA, CALL y CC para subrutinas. Con retardo (RETD) las dos instrucciones de una sola palabra o una instrucción de dos palabras que siguen a la instrucción RETD son ejecutadas antes de efectuar el retorno.
RETC	Retorno condicional de subrutina. Se ejecuta un retorno estándar (RET) si se cumplen las condiciones especificadas. Con retardo (RETCD), las dos instrucciones de una sola palabra o una instrucción de dos palabras que siguen a la instrucción RETCD son ejecutadas antes de efectuar el retorno. Si la instrucción es especificada con retardo, las siguientes dos palabras de instrucción a RETCD no tienen efecto sobre las condiciones de prueba.
RETE	Retorno de interrupción y habilita interrupciones mascarables. El contenido de la parte alta del stack se copia al contador de programa. Entonces el stack se corre una posición hacia arriba. RETE limpia automáticamente la interrupción global poniendo el bit a cero (INTM en STO) y hace una copia de los valores del registro de sombra.
RETI	Retorno de interrupción. El contenido de la parte alta del stack se copia al contador de programa. La instrucción también hace una copia de los valores del registro de sombra (almacenados cuando una interrupción se ha llevado a cabo) en sus correspondientes registros estratégicos. Los siguientes registros son recuperados: ACC, ACCB, PREG, ST0, ST1, PMST, ARCR, INDX, TREG0, TREG1 y TREG2.
ROL	Rota el acumulador a la izquierda. La instrucción rota el contenido del acumulador a la izquierda en un bit. El bit más significativo es corrido al bit de acarreo, y el valor del bit de acarreo es corrido al bit menos significativo. No es afectada por el bit SXM.
ROLB	La instrucción origina una rotación de 65 bits. Los contenidos del acumulador ACC y del acumulador buffer ACCB son rotados un bit hacia la izquierda. El bit más significativo del acumulador se corre a la posición de acarreo. El valor original del bit de acarreo C se corre a la posición del bit menos significativo en el acumulador buffer, y el bit más significativo del acumulador buffer se corre a la posición del bit menos significativo del acumulador ACC.

ROR	Rota un bit hacia la derecha el contenido del acumulador. El bit menos significativo es corrido hacia el bit de acarreo, y el valor del bit de acarreo es corrido a la posición más significativa desde antes de la ejecución de la instrucción. No es afectada por el bit SXM.
RORB	La instrucción origina una rotación de 65 bits. Los contenidos del acumulador ACC y del acumulador buffer ACCB son rotados un bit hacia la derecha. El bit menos significativo del acumulador se corre al bit más significativo del acumulador buffer. El valor original del bit de acarreo C se corre a la posición del bit más significativo del acumulador, y el bit menos significativo del acumulador buffer se corre a la posición del bit de acarreo.
RPT	Repite $n + 1$ veces la siguiente instrucción. El contador de repetición (RPTC) es cargado con la localidad de memoria de dato si es usado el modo de direccionamiento directo o el modo indirecto, un valor inmediato de 8 bits es usado si se usa direccionamiento inmediato corto, o un valor inmediato de 16 bits es usado si se usa direccionamiento inmediato largo. La instrucción siguiente a RPT es repetida $n + 1$ veces, donde n es la constante especificada en la instrucción. La instrucción RPT no es interrumpible.
RPTB	Permite que un bloque de instrucciones pueda repetirse un número de veces especificado por el registro contador de repetición de bloques (BRCR).
RPTZ	Limpia el acumulador y el registro de productos y repite la instrucción que le sigue $n + 1$ veces.

Ejemplo:

```
RPTZ    #7FFh    ; Carga con ceros el registro de productos
                 ; y el acumulador antes de repetir la
                 ; siguiente instrucción.
MACD    pma,*+   ; Repite MACD 2048 (7FFh +1h) veces.
```

SACB	El contenido del acumulador es copiado al acumulador buffer. No utiliza operadores.
SACH	Almacena la parte alta del acumulador en memoria dato. Con corrimiento a la izquierda (0-7) especificado en la instrucción. Durante el corrimiento los bits altos del ACC son perdidos.

SACL	Almacena la parte baja del acumulador en memoria dato. Con corrimiento a la izquierda (0-7) especificado en la instrucción. Durante el corrimiento los bits bajos del ACC son llenados con ceros.
SAMM	La parte baja del acumulador (ACCL) es copiada a un registro de memoria mapeada.
SAR	Guarda el registro auxiliar especificado en una localidad de memoria.
SBB	El contenido del acumulador buffer (ACCB) es restado del contenido del acumulador. El resultado es almacenado en el acumulador, y el acumulador buffer no se ve afectado. El bit de acarreo es puesto a cero si el resultado de la sustracción genera un préstamo.
SBBB	El contenido del acumulador buffer (ACCB) y la inversión lógica del bit de acarreo son restados del acumulador (ACC). Los resultados son almacenados en el acumulador, y el acumulador buffer no se ve afectado. El bit de acarreo es puesto a cero si el resultado genera préstamo.
SBRK	El valor inmediato de una constante de 8 bits es restado, justificado a la derecha, al actual registro auxiliar en uso. La sustracción se realiza en el ARAU, tratando la constante inmediata como un entero de 8 bits.
SETC	Un bit de control especificado es puesto a uno. Los bits de control pueden ser: C, CNF, INTM, OVM, TX y XF.
SFL	El contenido del ACC es corrido un bit a la izquierda, el MSB del ACC es corrido al bit de carry, los bits LSB de ACC son llenados con ceros. No es afectado por el SXM.
SFR	El contenido del ACC es corrido un bit a la derecha. El tipo de corrimiento es determinado por el bit SXM. Si SXM = 0, entonces el corrimiento es lógico, el MSB del ACC es llenado con ceros, el LSB del ACC es corrido al bit C. Si SXM = 1, entonces se produce un corrimiento aritmético, el MSB no cambia y es copiado al bit 30 del ACC. El LSB se copia en el bit C.
SFRB	El ACC y el ACCB son corridos un bit a la derecha. El LSB del ACC se copia en al MSB del ACCB, el LSB del ACCB se copia en el bit C, el MSB del ACC es corrido dependiendo del valor del SXM de manera similar que en la instrucción SFR.
SMMR	El valor del registro de memoria mapeada es almacenado en la localidad de memoria dato especificada.
SPAC	El contenido del registro P corrido como lo especifica PM se resta al acumulador, y el resultado se almacena en el mismo acumulador.

SPH	Los bits de mayor orden del registro P son almacenados en memoria dato. Afectado por PM.
SPL	Los bits de menor orden del registro P son almacenados en memoria dato. Afectado por PM.
SPLK	Permite almacenar una constante de 16 bits en cualquier localidad de memoria.
SPM	Una constante de dos bits es copiado al campo PM del registro de estado ST1. Define el corrimiento de salida del multiplicador.
SQRA	El contenido del registro TREG0 es elevado al cuadrado y el contenido del registro P es sumado al acumulador con corrimiento especificado por PM.
SQRS	El contenido del registro TREG0 es elevado al cuadrado y el contenido previo del registro P es restado al acumulador con corrimiento especificado por PM.
SST	Los registros de estado (ST0 o ST1) son almacenados en la dirección especificada de memoria dato.
SUB	El contenido de la dirección específica de memoria dato o una constante corrida a la izquierda se resta al acumulador. Durante el corrimiento, los bits de orden bajo se llenan con ceros y se coloca el signo en el bit más significativo si SXM = 1.
SUBB	El contenido de la localidad de memoria dato y la inversión lógica del bit de acarreo es restada del acumulador con extensión de signo suprimido.
SUBS	El contenido especificado de memoria dato se resta al acumulador sin considerar el bit de signo. El contenido de la dirección especificada de memoria dato se considera como un entero positivo de 16 bits.
SUBT	El valor de la dirección especificada de memoria dato se corre a la izquierda de 0 a 15 bits especificado por TREG1 y es restado del acumulador.
TBLR	Transfiere una palabra desde cualquier lugar en memoria de programa a la localidad especificada de memoria dato. La dirección pma es especificada por la parte baja del ACC.
TBLW	Transfiere una palabra desde la localidad especificada de memoria dato a la localidad de la RAM externa del programa. La dirección pma es especificada por la parte baja del ACC.
TRAP	Interrupción por software que transfiere el control a la localidad de memoria dato 22h.

XC	Las siguientes dos instrucciones de una sola palabra o la siguiente instrucción de dos palabras se ejecutan si se cumplen las condiciones. El número de palabras (instrucción de una palabra) a ejecutar se especifican en la instrucción. Utilizan las misma condiciones que BCND.
XOR	Efectúa el OR exclusivo del contenido de la localidad memoria de dato especificada o una constante de 16 bits con corrimiento con la parte baja del acumulador ACCL.
XORB	Se realiza la operación OR exclusivo entre el contenido del acumulador buffer y el contenido del acumulador. El resultado se escribe en el ACC y el ACCB no se ve afectado. No utiliza operandos.
XPL	XOR entre el valor de memoria de datos y la constante larga especificada. El resultado siempre es escrito de nuevo en la localidad de memoria especificada. Si no se especifica la constante, el segundo operando a la operación XOR es el contenido del registro de manipulación dinámica de bits (DBMR). El contenido del acumulador no es afectado. Si el resultado de la operación OR es cero, entonces el bit TC es puesto a uno; en cualquier otro caso, el bit TC es puesto a cero.
ZALR	Cargar el valor de la memoria de dato en la parte alta del acumulador, la instrucción redondea el resultado en el ACC, es decir, suma un uno al bit 15 en el ACC y llena con ceros los bits 0-14 del ACC.
ZAP	El acumulador y el registro de producto se cargan con ceros. No utiliza operandos.
ZPR	El registro P se carga con ceros. No utiliza operandos.

Para detalles más específicos de programación se sugiere consultar el manual de usuario de TI [18].

Capítulo 3

Implementación de filtros digitales en el DSP TMS320C50

En esta parte se considera la forma de implementar las ecuaciones y estructuras de sistemas lineales y filtros digitales en la arquitectura del C5x [1]. Se hace la observación que los DSP se han desarrollado para efectuar cálculos eficientes de algoritmos matemáticos utilizados en sistemas discretos.

Un sistema lineal e invariante en el tiempo discreto puede ser descrito (SLITD) por una ecuación en diferencias:

$$y(n) + \sum_{k=1}^{p} a_k y(n-k) = \sum_{m=0}^{q} b_m x(n-m) \quad \therefore a_0 = 1 \tag{3.1}$$

con función de transferencia:

$$H(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_q z^{-q}}{1 + a_1 z^{-1} + \cdots + a_p z^{-p}} \tag{3.2}$$

La respuesta al impulso unitario del sistema, $h(n)$ es la transformada inversa Z de la función de transferencia $H(z)$. Un filtro digital es un sistema lineal y también se puede caracterizar por una ecuación en diferencias, su función de transferencia o su respuesta al impulso.

Un filtro digital de respuesta finita al impulso (FIR), es aquel cuya respuesta al impulso es de duración finita. De la ecuación general (3.1) los coeficientes a_k son cero para $k \geq 1$, entonces, un filtro FIR queda descrito por:

[1] Se asume que el lector conoce la teoría básica del diseño de filtros digitales y el cálculo de los coeficientes. Para mayor información de diseño de filtros consultar [9], [14] y [12].

$$y(n) = \sum_{m=0}^{q} b_m x(n-m) \quad \therefore b_0 = 1 \tag{3.3}$$

y su función de transferencia:

$$H(z) = b_0 + b_1 z^{-1} + \cdots + b_q z^{-q} \tag{3.4}$$

donde se puede observar que la salida $y(n)$ del sistema FIR es una suma ponderada de los coeficientes b_m por la entrada actual $x(n)$ y las muestras retardadas, además un filtro FIR se caracteriza por ser siempre estable y de fase lineal.

La sumatoria que permite efectuar un filtro FIR es la operación de convolución de los coeficientes por una ventana temporal de una señal como se observa en la figura 3.1.

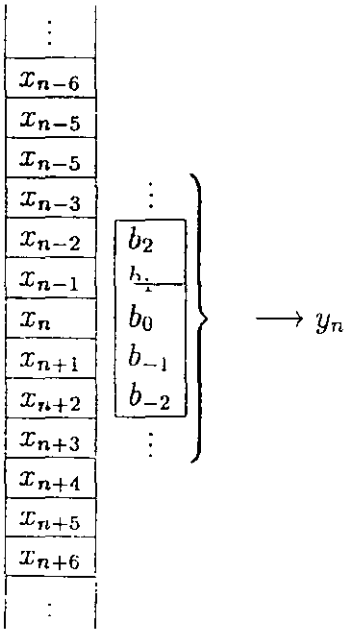


Figura 3.1: Convolución de una señal con los coeficientes del filtro

3.1. Implementación de líneas de retardo

En esencia un filtro FIR es una suma de productos de los coeficientes b_k por las muestras retrasadas $x(n-i)$ de la señal de entrada. Para la implementación de estos filtros eficiente-

mente es necesario tener la posibilidad de generar una línea de retardos.

3.1.1. Buffer lineal

La forma más simple de implementar una línea de retardo en un microprocesador es vía un buffer lineal, donde un filtro de N taps opera sobre las N muestras más recientes. Cada vez que se efectúa la sumatoria del filtro FIR se adquiere un nuevo dato y es agregado a la parte superior del buffer y el dato inferior es descartado. Es decir, que una vez calculada la salida, un dato es movido a la localización siguiente para realizar el retardo.

En un procesador convencional una línea de retardo involucraría el acceso a dos localidades de memorias continuas y un almacenamiento temporal del dato (por lo menos tres instrucciones por cada dato desplazado).

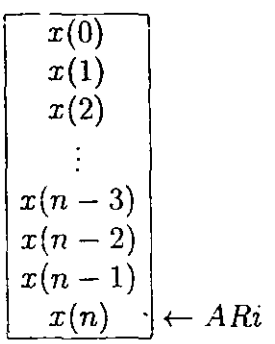


Figura 3.2: Buffer lineal

El TMS320C50 y la familia TMS320 pueden generar eficientemente las líneas de retardo con instrucciones orientadas para estos fines. La instrucción DMOV efectúa una copia del dato direccionado a la siguiente localidad en memoria dato sin afectar al acumulador u otro registro, esto es válido sólo en memoria interna.

```
DMOV  <dma>  ; puede ser direccionamiento directo e indirecto
```

El proceso anterior puede ser más eficiente si se combina simultáneamente con un carga de un dato al registro TREG0 y la acumulación del producto anterior, todo esto lo efectúa la instrucción LTD:

```
LTD    <dma> ; Carga el registro TREG0 con dma, mueve el contenido de dma
             ; a localidad dma+1 y suma al acumulador el producto anterior
```

Las líneas de retardo para la implementación de filtros son aún más eficientes utilizando la instrucción MACD. Se hace notar que el movimiento de bloque sólo funciona para memoria RAM interna para los bloque B0, B1 y B2 mapeados en memoria dato.

3.1.2. Implementación de un filtro FIR utilizando LTD y MPY

Direccionamiento directo			Direccionamiento indirecto		
	LDP	#X0		LDP	#X0
				MAR	*,AR1
			LOOP	LAR	AR2,#A4
				LAR	AR1,#X4
START	ZAP			ZAP	
	LT	X4		LT	*-,AR2
	MPY	A4		MPY	*-,AR1
	LTD	X3		LTD	*-,AR2
	MPY	A3		MPY	*-,AR1
	LTD	X2		LTD	*-,AR2
	MPY	A2		MPY	*-,AR1
	LTD	X1		LTD	*-,AR2
	MPY	A1		MPY	*-,AR1
	LTD	X0		LTD	*,AR2
	MPY	A0		MPY	*,AR1
	APAC			APAC	
	SACH	Y,1		SACH	Y,1
	OUT	Y,PA0		OUT	Y,PA0
	IN	X0,PA1		IN	X0,PA1
	B	START		B	LOOP

Figura 3.3: Implementación de un filtro FIR con LTD y MPY

El programa de direccionamiento indirecto puede hacerse más eficiente si se utiliza la instrucción de repetición de bloque. Ver la figura 3.3.

3.1.3. Filtro FIR utilizando la instrucción MACD

En sí la instrucción MACD es la operación en conjunto de las instrucciones MAC y DMOV, como la instrucción DMOV siempre escribe el dato actual en la siguiente localidad,

entonces las muestras de la señal de entrada se escriben a memoria dato y los coeficientes a memoria programa, ver figura 3.4.

```
        LAR     AR1,#x
        MAR     *,AR1
        LAR     AR0,#99
FIR:    RPTZ    #99
        MACD    coeff,*-
        APAC
        SACH    *,1
        OUT     *+,1
        BD      FIR
        IN      *0+,2
```

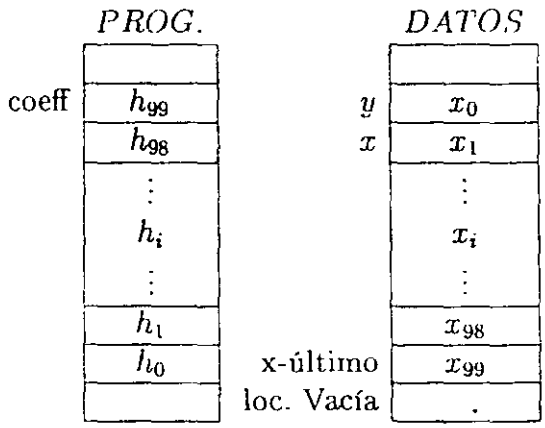


Figura 3.4: Filtro FIR usando MACD

3.1.4. Filtro FIR utilizando instrucción MADS y MADD

La instrucción MADS permite a la instrucción MAC operar sobre una localidad de memoria programa direccionada por el registro BMAR en lugar de un valor fijo en la instrucción MAC. La instrucción MADD es la operación que combina las instrucciones MADS y DMOV. Otra forma de implementar el ejemplo anterior es:

```
LACC    #coeff
SAMM    BMAR  ; carga la dirección del coeff  al registro BMAR
.
.
MADD    *-    ; multiplica, acumula y mueve dato a siguiente loc.
```

3.2. Filtros de respuesta infinita al impulso (IIR)

Una característica de un filtro IIR que lo diferencia de un filtro FIR es que retroalimenta la señal de salida y que puede llegar a ser inestable. En un filtro IIR los valores de los coeficientes a_m son diferentes de cero y también pueden existir todos los coeficientes b_m.

La implementación de un filtro IIR se puede ver como la convolución de los coeficientes b_m con la señal de entrada menos la convolución de la señal de salida retardada con los coeficientes a_m, esto se aprecia en la figura 3.5.

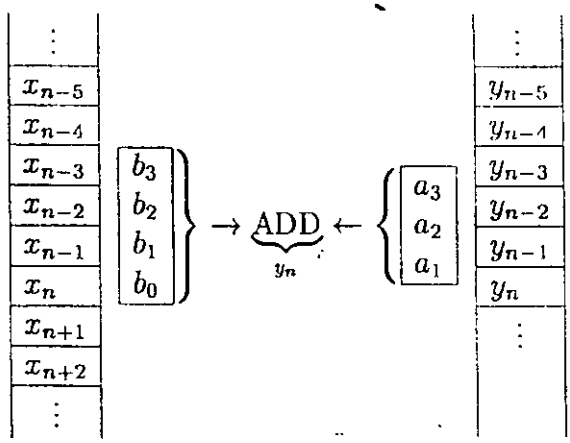


Figura 3.5: Líneas de retardo de un filtro IIR.

3.2.1. Estructuras de un filtro IIR

De la ecuación en diferencias general existen dos forma de implementar un filtro IIR: la forma directa uno donde existe donde existen $2p$ (si $p = q$) elementos de retardo mientras que el orden del filtro es p. En la segunda forma o canónica donde el número de retardos es igual al orden del filtro. Ver figuras 3.6 y 3.7.

3.2.2. Separación en estructuras de segundo orden

De la teoría del procesamiento digital de señales [9], [12], [14], [15], [16], para la implementación de un filtro digital IIR es conveniente implementarlo en estructuras de segundo orden para evitar la acumulación de errores cuando se opera en aritmética de punto entero.

3.2.3. Estructuras de filtros digitales IIR

La ecuación en diferencias que se utiliza para implementar un filtro IIR es:

$$y(n) = \sum_{i=0}^{q} b(i)x(n-i) - \sum_{i=1}^{p} a(i)y(n-i) \tag{3.5}$$

Aplicando la Transformada Zeta (TZ) a esta ecuación, se obtiene la función de transferencia H(z) del filtro:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{B(z)}{A(z)} = \frac{\sum_{i=0}^{q} b(i)z^{-i}}{\sum_{i=1}^{p} a(i)z^{-i}} = \frac{b_0 + b_1 z^{-1} + ... + b_q z^{-q}}{1 + a_1 z^{-1} + ... + a_p z^{-p}}, \qquad a_0 = 1 \tag{3.6}$$

Donde p es el número de polos, q es el número de ceros de $H(z)$, a_i y b_i son los coeficientes del filtro. Dependiendo como se traten las dos ecuaciones anteriores se pueden obtener tres formas o estructuras más comunes de implementación [2]:

- Forma directa
- Forma cascada
- Forma paralela

Filtro IIR Forma directa

En esta forma la ecuación (3.5) es implementada directamente. En este filtro existen dos partes denominadas movimento promedio (MA) y la parte recursiva (AR). A la vez esta implementación conduce a dos formas de implementación: forma directa I y forma directa II.

Forma directa I

Si la ecuación (3.5) se desarrolla, entonces se tiene:

$$y(n) = b_0 x(n) + b_1 x(1) + ...b_q x(n-q) - a_1 y(n-1) - a_2 x(n-2) - ... - a_p y(n-p) \tag{3.7}$$

Se observa que existen dos líneas de retardo, una para la señal de entrada $x(n)$ y la otra para la señal de salida $y(n)$, es decir, que se tendrían $p+q$ bloques de retardo.

Forma directa II o forma canónica

Las líneas de retardo se pueden convertir a una sola, lo que conduce a la forma directa II de un filtro IIR o *estructura canónica*, es decir, que a nivel de hardware se ahorrarían localidades de memoria.

[2] Otras estructuras como Lattice, Lattice-Ladder y Filtros de espacio de estado se pueden encontrar en [14] y [10] respectivamente

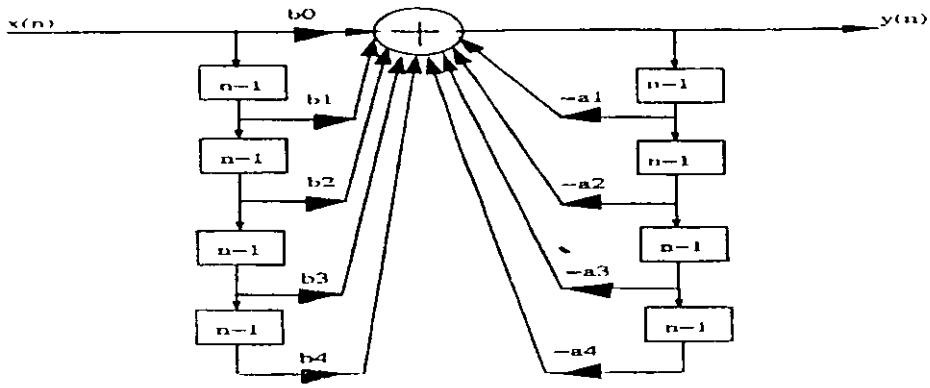


Figura 3.6: Filtro IIR, forma directa I

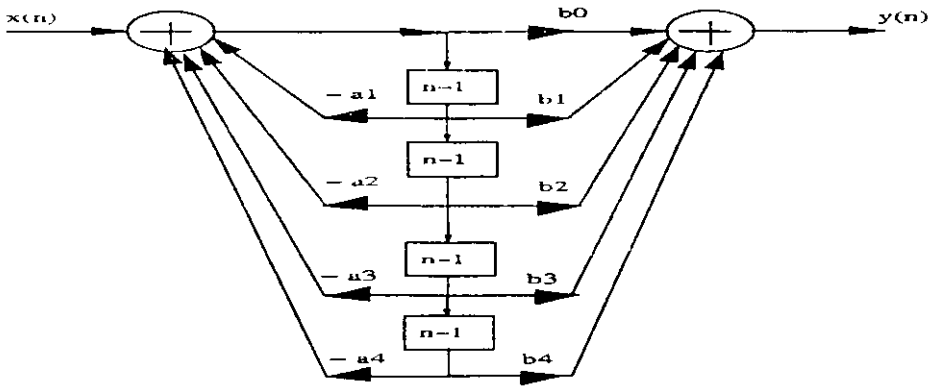


Figura 3.7: Filtro IIR, forma directa II

Filtro IIR Forma cascada

En esta forma, la ecuación (3.8) es factorizada en secciones de segundo orden, llamadas secciones *biquads* [18]. La función de transferencia del sistema es representada como el producto de funciones biquads. Cada función biquad es implementada en forma directa o canónica y la función completa del sistema como secciones *biquad* en cascada. Es decir, que se factorizan el numerador y denominador de la ecuación (3.6) para obtener factores de segundo orden con coeficientes reales, teniendo la ecuación a implementar:

$$H(z) = b_0 \prod_{k=1}^{K} \frac{1 + B_{k,1} z^{-1} + B_{k,2} z^{-2}}{1 + A_{k,1} z^{-1} + A_{k,2} z^{-2}} \tag{3.8}$$

Donde $K = \frac{N}{2}$, y $B_{k,1}$, $B_{k,2}$, $A_{k,1}$ y $A_{k,2}$ son números reales que representan los coeficientes de las secciones de segundo orden. Cada sección biquad se implementa en forma directa o canónica, y la entrada de la sección k-ésima es la salida de la sección $(k-1)$. Esta forma de implementación permite disminuir la acumulación de errores numéricos cuando los filtros se programan en aritmética de punto fijo.

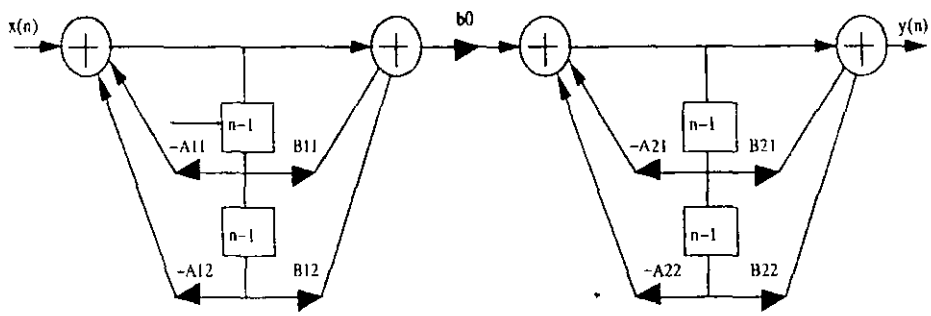


Figura 3.8: Filtro IIR en cascada

Forma paralela

Es similar a la forma cascada, pero con la diferencia que la ecuación (3.6) se representa como una suma de fracciones parciales de segundo orden. De nuevo cada sección es implementada en forma directa y la función total del sistema como una red paralela de las secciones. Desarrollando la ecuación (3.6) en fracciones parciales:

$$H(z) = \frac{\hat{b}_0 + \hat{b}_1 z^{-1} + \ldots + \hat{b}_q z^{-q}}{1 + a_1 z^{-1} + \ldots + a_p z^{-p}} + \sum_{0}^{q-p} C_k z^{-k} \qquad q \geq p \tag{3.9}$$

$$H(z) = \sum_{k=1}^{K} \frac{B_{k,0} + B_{k,1}z^{-1}}{1 + A_{k,1}z^{-1} + A_{k,2}z^{-2}} + \sum_{0}^{q-p} C_k z^{-k} \qquad q \geq p \tag{3.10}$$

Donde $K = \frac{N}{2}$, y $B_{k,0}$, $B_{k,1}$, $A_{k,1}$ y $A_{k,2}$ son números reales que representan los coeficientes de las secciones de segundo orden. La entrada $x(n)$ del filtro está disponible para toda sección biquad, la salida de cada sección es sumada para formar la salida $y(n)$ del filtro.

3.2.4. Filtro digital de IIR de segundo orden

Un filtro IIR se puede analizar en dos secciones, una de retroalimentación y una similar a un filtro FIR con entrada X_0, como se ve en la figura 3.9.

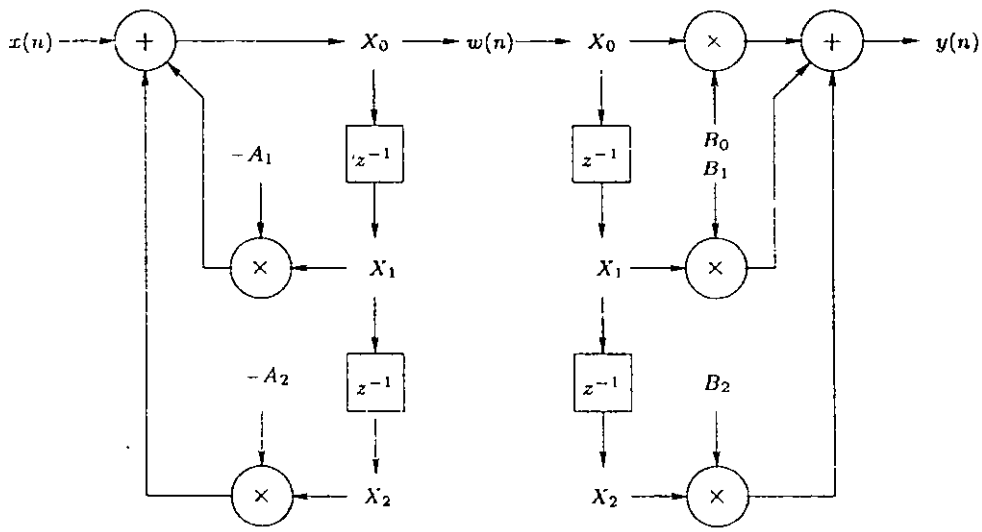


Figura 3.9: Implementación de un filtro IIR de segundo orden

La señal X_0 es una señal que es retardada y alimenta al filtro FIR y a la vez es retroalimentada al primer sumador. Ver figura 3.10.

```
        LDP     #X
        SPM     0               ; No existe corrimiento en registro PR al
                                ; transferirse al ACC.
IIR     IN      X,PA1           ; obtiene una muestra de entrada del puerto PA1
        LACC    X,15
        LT      X1              ; T = X1
        MPY     A1              ; PR = X1*A1
        LTA     X2              ; T = X2      ACC = Xin + X1*A1
        MPY     A2              ; PR = X2*A2
        APAC                    ; ACC = Xin + X1*A1 + X2*A2  (Q30)
        SACH X0,1               ; guarda en X0 parte alta del ACC en Q15
        LACC    #0              ; ACC = 0
        MPY     B2              ; PR = X2*B2 ( en T estaba B2 )
        LTD     X1              ; T = X1, ACC = X2*B2, X1 se mueve a X2
        MPY     B1              ; PR = X1*B1
        LTD     X0              ; T = X0, ACC = X2*B2 + X1*B1, X0 se mueve a X1
        MPY     B0              ; PR = X0*B0
        APAC                    ; AC =  X2*B2 + X1*B1 + X0*A0
        SACH    Y,1             ; Y localidad de salida temporal
        B       IIR             ; Regresa a al filtro IIR.
```

Figura 3.10: Filtro IIR de segundo orden

Capítulo 4

Prácticas

En este capítulo se presentan 21 programas para que el lector se introduzca en el mundo de los DSPs, conozca las instrucciones, la arquitectura del C5x y posteriormente pueda elaborar sus propios programas. Se propone que el alumno teclee los siguientes programas, los ensamble, los corra en el ambiente depurador y observe como se van realizando las operaciones y cómo cambian algunos registros y localidades de memoria.

4.1. Sugerencias

— Para correr los programas paso a paso utilizar tecla F8

— Cambiar datos de entrada y ver de nuevo como operan los programas

— Escribir una línea final:

```
FIN     NOP
        NOP
        B   FIN    ; ciclo infinito para fin de programa
```

y correr el programa con F5

— Para bloques de texto o código que se repitan en los programas, escribirlos en otro archivo y utilizar las directivas .include o .copy para utilizarlos en los programas.

4.2. Suma de cinco números con diferentes posibilidades

Los programas siguientes tienen como objetivo que el lector empiece a entender la programación del DSP C50 e ingresar al ambiente de depuración. Los programas del uno al cinco

tienen como objetivo que el lector entienda y distinga los modos de direccionamiento, del seis al 13 se introducen operaciones matemáticas para llegar a realizar de modo eficiente la convolución. Al final de este capítulo se agrega otro conjunto de programas más elaborados. Se hace notar que todos los programas han sido escritos por el autor y el funcionamiento de cada uno ha sido probado, otros programas de aplicación se pueden encontrar en [1], [2], [3], [4], [5], [6], [18].

Programa 1

Efectuar una suma de cinco constantes en modo de direccionamiento inmediato, dejar el resultado en el acumulador.

```
        .mmregs
        .ds 0f00h               ; Segmento de datos en memoria dato
                                ; localidad 0F00h.
D1      .set    1               ; declara constante D1
D2      .set    2               ; declara constante D2
D3      .set    3
D4      .set    4
D5      .set    5

        .ps                     ; Segmento de programa.
        .entry 0a00h            ; Localidad donde inicia ejecución del programa.

        LACL    #D1             ; ACC = D1 = 1
        ADD     #D2             ; ACC = D1+D2 = 1+2 = 3
        ADD     #D3             ; ACC = D1+D2+D3 = 1+2+3 = 6
        ADD     #D4             ; ACC = D1+D2+D3+D4 = 1+2+3+4 = 10
        ADD     #D5             ; ACC = D1+D2+D3+D4+D5 = 1+2+3+4+5 = 15
                                ; Observar el resultado en el ACC en hexadecimal.
*
FIN     NOP
        B   FIN                 ; ciclo infinito para fin de programa
*
        .end
```

Programa 2

Efectuar una suma de cinco constantes en modo de direccionamiento directo, dejar el resultado en localidad llamada total.

```
        .mmregs
        .ds 0f00h           ; Segmento de datos
dat1    .word   5           ; reserva memoria para dat1 y escribe el valor 5
dat2    .word   10          ; reserva memoria para dat2 y escribe el valor 10
dat3    .word   15
dat4    .word   2
dat5    .word   3
total   .word   0
        .ps
        .entry 0a00h

        LDP     #dat1       ; Carga en el reg. apuntador de página DP
                            ; la página donde está dat1
        ZAP                 ; Cero al acumulador y al registro de producto
        LACL    dat1        ; Pone en la parte baja del acumulador dat1
        ADD     dat2        ; Suma dat2
        ADD     dat3        ; Suma dat3
        ADD     dat4        ; Suma dat4
        ADD     dat5        ; Suma dat5
        SACL    total       ; Pone el resultado de la suma en total
*
FIN     NOP
        B   FIN             ; ciclo infinito para fin de programa
*
        .end
```

Programa 3

Efectuar una suma de cinco constantes en modo de direccionamiento indirecto, sin usar instrucción de repetición.

```
        .mmregs
        .ds 0f00h           ; Segmento de datos
dat1    .word   5
dat2    .word   10
dat3    .word   15
dat4    .word   2
dat5    .word   3
total   .word   0
        .ps
        .entry 0a00h
```

```
        LDP     #dat1       ; Carga en el reg. apuntador de página DP.
                            ; la página donde está dat1.
        ZAP                 ; Cero al acumulador y al registro de producto.
        MAR     *,AR0       ; Elige el reg. AR0 como registro auxiliar
                            ; actual en uso.
        LAR     AR0,#dat1   ; Pone la dirección de dat1 en el reg. auxiliar AR0.
        ADD     *+          ; Suma dat1 al acumulador y post-incrementa en uno a AR0.
        ADD     *+          ; Suma dat2 al acumulador y post-incrementa en uno a AR0.
        ADD     *+          ; Suma dat3 al acumulador y post-incrementa en uno a AR0.
        ADD     *+          ; Suma dat4 al acumulador y post-incrementa en uno a AR0.
        ADD     *+          ; Suma dat5 al acumulador y post-incrementa en uno a AR0.
        SACL    *           ; Pone el resultado de la suma en localida total

*
FIN     NOP
        B   FIN             ; ciclo infinito para fin de programa
*

          .end
```

Programa 4

Efectuar una suma de cinco constantes en modo de direccionamiento indirecto, con instrucción de repetición.

```
        .mmregs
        .ds 0f00h
        .data               ; Segmento de datos
dat1    .word   5
dat2    .word   10
dat3    .word   15
dat4    .word   2
dat5    .word   3
total   .word   0
        .ps
        .entry 0a00h

        LDP     #dat1       ; Carga en el reg. apuntador de página
                            ; la página donde está dat1.
        ZAP                 ; Cero al acumulador y al registro de producto.
```

```
        MAR     *,AR0       ; Elige el reg. AR0 como registro auxiliar
                            ; actual en uso.
        LAR     AR0,#dat1   ; Pone en AR0 la dirección de dat1 .
        RPT     #4          ; Repite 5 veces la siguiente instrucción. Recordar
                            ; que la instrucción RPT no es interrumpible, por lo
                            ; tanto no se observará en el depurador las sumas
                            ; parciales sino el resultado total.
        ADD     *+          ; Suma los 5 números al acumulador.
        SACL    *           ; Pone el resultado de la suma en loc. total.

*
FIN     NOP
        B   FIN             ; ciclo infinito para fin de programa
*

        .end
```

Se deja como ejercicio al lector que utilice la instrucción RPTB para observar las sumas parciales en el ciclo.

Programa 5

Efectuar una suma de cinco constantes en modo de direccionamiento indirecto y su resultado almacenarlo en direccionamiento directo.

```
        .mmregs                 ; Segmento de datos
        .ds 0f00h

dat1    .word   5
dat2    .word   10
dat3    .word   15
dat4    .word   2
dat5    .word   3
total   .word   0
        .ps
        .entry 0a00h

        ZAP                     ; Cero al acumulador y al registro de producto
        MAR     *,AR0           ; Elige el reg. AR0 como registro auxiliar
                                ; actual en uso
        LAR     AR0,#dat1       ; Pone en AR0 la dirección de dat1
```

```
        RPT     #4              ; Repite 5 veces la siguiente instrucción
        ADD     *+              ; Suma los 5 números al acumulador
        LDP     #total          ; Carga en el reg. apuntador de página
                                ; la página donde está total
        SACL    total           ; Pone el resultado de la suma en loc. total

*
FIN     NOP
        B   FIN                 ; ciclo infinito para fin de programa
*

        .end
```

Programa 6

Efectuar una suma de cinco constantes en modo de direccionamiento indirecto y su resultado almacenarlo en direccionamiento indirecto usando otro ARi además utilizar instrucción de repetición de bloque RPTB.

```
        .mmregs
        .ds 01000h          ; Segmento de datos

dat1    .word   5,10,15,2,3 ; reserva memoria para el vector dat1 y le
                            ; escribe valores
total   .word   0
        .ps
        .entry 0a00h

        ZAP                 ; Cero a ACC y a P
        LAR     AR0,#dat1   ; Pone la dirección de X1 en AR0
        MAR     *,AR0       ; Selecciona el registro auxiliar AR0

        LACL    #4          ; Pone en la parte baja de ACC un 4
        SAMM    BRCR        ; Pone lo que esta en la parte baja de ACC en el
                            ; registro de repetición de bloque BRCR
        ZAP                 ; Cero a ACC y a P
*
        RPTB    SUMA        ; Repetición del bloque SUMA de 4+1 veces
          ADD     *+        ; añade dat1..dat5 al acumulador
          NOP
SUMA      NOP               ; Observar el retorno de repetición de bloque y el
```

```
                                ; decremento del registro BRCR
*
        MAR     *,AR1           ; Selecciona el registro auxiliar AR1
        LAR     AR1,#total      ; Pone la dirección de total en AR1
        SACL    *               ; Pone lo que está en la parte baja de ACC en total

*
FIN     NOP
        B   FIN                 ; ciclo infinito para fin de programa
*

        .end
```

Suma de productos de 5 números con varias posibilidades

Se supone que el lector ya conoce lo suficiente la arquitectura y el conjunto de instrucciones del C50, por lo tanto a continuación se utilizan menos comentarios.

Programa 7

Usando instrucción LT, MPY y APAC.

```
        .mmregs
        .ds     0f00h
A1      .word   10,20,30,40,50,60
*
X1      .word   1,2,1,2,1,2
*
Y0      .word   0
*
N1      .set    5

        .ps     0a00h
        .entry

        LDP     #Y0
        SETC    SXM
        LAR     AR1,#X1
        LAR     AR2,#A1
        MAR     *,AR1
```

```
        LACL    #N1
        SAMM    BRCR
        ZAP
*
        RPTB    FIN_1
          LT      *+,AR2
          MPY     *+,AR1
          APAC
FIN_1     NOP
*
        SACL    YO
*
FIN     NOP
        B   FIN
*
        .end
```

Programa 8

Usando instrucción LTA y MPY.

```
        .mmregs
        .ds     0f00h
A1      .word   1,2,3,4,5,6
*
X1      .word   1,2,1,2,1,2
*
YO      .word   0
*
N1      .set    5

        .ps     0a00h
        .entry

        LDP     #YO
        SETC    SXM
        LAR     AR1,#X1
        LAR     AR2,#A1
        MAR     *,AR1
        LACL    #N1
        SAMM    BRCR
```

```
        ZAP

        RPTB    FIN_1
        LTA     *+,AR2
        MPY     *+,AR1
FIN_1   NOP
        APAC
        SACL    YO
*
FIN     NOP
        B   FIN
*

        .end
```

Programa 9

Usando instrucción LT y MPYA.

```
        .mmregs
        .ds     0f00h
A1      .word   010,20,30,40,50,60
*
X1      .word   1,2,3,3,2,1,
*
YO      .word   0
*
N1      .set    5

        .ps     0a00h
        .entry

        LDP     #YO
        SETC    SXM
        LAR     AR1,#X1
        LAR     AR2,#A1
        MAR     *,AR1
        LACL    #N1
        SAMM    BRCR
        ZAP
```

```
        RPTB    FIN_1
        LT      *+,AR2
        MPYA    *+,AR1
FIN_1   NOP
        APAC
        SACL    Y0
*
FIN     NOP
        B   FIN
*
        .end
```

4.3. Multiplicación acumulación (con instrucción MAC)

Programa 10

```
        .mmregs
        .ds     0f00h
A1      .word   1,2,3,4,5,6
*
X1      .word   1,2,1,2,1,2
*
Y0      .word   0
*
N       .set    5

        .ps     0A00h
        .entry

        LDP     #Y0
        LAR     AR0,#A1
        MAR     *,AR0

        ZAP
         RPT     #N
         MAC     #X1,*+

        APAC
        SACL    Y0
*
```

```
FIN     NOP
        B   FIN
*
        .end
```

Programa 11

Observar la diferencia respecto al programa 10

```
        .mmregs
        .ds     01000h
A1      .word   1,2,3,4,5,6
*
X1      .word   1,2,3,3,1,2
*
YO      .word   0
*
N       .set    5

        .ps     0A00h
        .entry

        LDP     #YO
        LAR     AR0,#X6
        MAR     *,AR0

        ZAP
          RPT     #N
          MAC     #A1,*-

        APAC
        SACL    YO
*
FIN     NOP
        B   FIN
*
        .end
```

Movimiento de bloques de datos

Programa 12

```
        .mmregs
        .ds     0f00h
X1      .word   001h
X2      .word   002h
X3      .word   003h
X4      .word   004h
X5      .word   005h
X6      .word   006h
*
Y       .word   0,0,0,0,0,0,0,0,0,0
*
*
N       .set    5

        .ps     0a00h
        .entry

        LDP     #Y
        LAR     AR0,#X1
        MAR     *,AR0

        LACC    #00

        RPT     #N
        BLDD    *+,#Y

*
FIN     NOP
        B   FIN
*
        .end
```

4.4. Suma de productos usando instrucción MADS

Programa 13

```
        .mmregs
```

```
        .ds     0f00h
A1      .word   1,2,3,4,5,6
*
X1      .word   1,2,1,2,1,2
*
Y0      .word   0
*
N       .set    5

        .ps     0a00h
        .entry

        LACC    #A1
        SAMM    BMAR
        LAR     AR0,#X1
        MAR     *,AR0
        ZAP

        RPT     #N
        MADS    *+
        APAC

        LDP     #Y0
        SACL    Y0
*
FIN     NOP
        B   FIN
*
        .end
```

4.5. Otros ejemplos de programas

4.5.1. Decimación

Efectúa la decimación de 16 números utilizando direccionamiento en carry inverso.

```
*
*       decima.asm
*       @larry
*       Efectúa el moviento de 16 datos con
*       decimacion radix 2
```

```
*
        .mmregs
        .ds  01000h
DAT     .word 0,1,2,3,4,5,6,7        ; Datos ordenados
DAT1    .word 8,9,10,11,12,13,14,15
DEC     .set  01010h                 ; Loc. incio datos decimados
N       .set  16
N1      .set  15                     ; N-1
N2      .set  8                      ; N/2
*
        .ps 0A00h
        .entry
        LACL  #N2
        SAMM  INDX                   ; INDX=N/2
        LAR   AR1,#DAT
        MAR   *,AR1
          RPT #N1
          BLDD *BRO+,#DEC

FIN     NOP
        B    FIN
        .end
```

4.5.2. Multiplicación de una matriz por un vector

```
*       MULTIPLICA MATRIX-POR VECTOR
*
*       mat_vec.asm
*       ©larry
        .mmregs
        .ds    01000h
A       .word  1,2,3,4,5,6,7,8,9
V       .word  1,2,3
C       .word  0,0,0
N       .set    2

        .ps 0a00h
        .entry

        LAR  AR1,#A    ; Matriz
        LAR  AR3,#C    ; Resultado
```

```
        LAR  AR4,#N    ; Contador de loop BANZ

        LACC  #V
        SAMM  CBSR1    ;  Inicio de Buf. circ. 1
        SAMM  AR2      ;  AR2 direcciona en forma circular
        ADD   #N
        SAMM  CBER1    ;  Fin de Buf. circ. 1
        LACL  #0Ah     ;
        SAMM  CBCR     ;  Activa Buf. circ. 1 con AR2


        MAR  *,AR1
UNO     LACL #N
        SAMM   BRCR

         ZAP
         RPTB  FIN_B
           LT   *+,AR2
           MPY   *+,AR1
FIN_B    APAC
           MAR   *,AR3      ;  AR3 direc. de resultado
           SACL  *+,0,AR4   ;  Salva C(i) , Ar4 contador de loop UNO
           BANZ  UNO,*-,AR1
*
FIN     NOP
        NOP
        B       FIN
        .end
```

4.5.3. Multiplicación de matriz por matriz

```
*       MULTIPLICA MATRIX-POR MATRIZ
*       UTILIZA DIRECCIONAMIENTO DINAMICO, REG. BMAR
*       INSTRUCCION     MADS  EN LUGAR DE MAC
*       mat_mat.asm
*       ©larry
*
        .mmregs
        .ds     01000h
A       .word   1,2,3,4,5,6,7,8,9  ; A Ordenada por filas
B       .word   1,1,1,2,2,2,3,3    ; B Ordenada por columnas
```

```
BF      .word  3
C       .word  0,0,0,0,0,0,0,0,0  ; Resultado Columnas [14  32  50]
N       .set   2
D       .set   3                  ; Salto de renglón

        .ps 0a00h
        .entry

        LAR  AR2,#B   ; Dir.  MATRIZ  B, opera circularmente
        LACC  #B
        SAMM  CBSR1   ; Inicio de Buf. circ. 1
        SAMM  AR2     ;
        LACC  #BF
        SAMM  CBER1   ; Fin.  Buff. circ.  1
        LACC  #0Ah
        SAMM  CBCR    ; Habilita buf. circ. 1

        LAR  AR3,#C   ; Dir.  Matriz  C  (Resultado)
        LAR  AR4,#2   ; Contador de loop BANZ

        LACC  #A
        SAMM  BMAR    ; Mat. A. Es apuntada dinámicamente por BMAR y PFC


MATRIZ        LACC  #N
              SAMM  BRCR        ;
              RPTB  FIN_B       ; Bloque que calcula C(fila,j)

              MAR *,AR2         ;
                ZAP
                RPT    #N       ; Loop para relizar producto punto
                MADS   *+
                APAC
                MAR    *,AR3    ;
FIN_B           SACL   *+       ; Salva C(fila,j)

            LAMM  BMAR          ; Recupera valor de BMAR
            ADD   #D            ; Desplaza D localidades
            SAMM  BMAR
            MAR   *,AR4         ; Loop Externo
```

```
          BANZ   MATRIZ,*-,AR2
*
FIN       NOP
          NOP
          B    FIN
          .end
```

4.5.4. Encuentra máximo, mínimo y máximo absoluto

```
*
*    maxmin.asm
*    @larry
*    Halla el Máximo, Mínimo
*    y máximo absoluto de una señal
*
         .mmregs
         .ds  0F00h
MAX      .word   0
MIN      .word   0
MAXA     .word   0
N        .set   25       ; Número de datos
N1       .set   01000h   ; Loc. inicio datos
CERO     .set   00
*
         .ps  0A00h
         .entry
         SETC SXM
         LDP  #MAX
         LACC #N
         SAMM BRCR
         LAR  AR1,#N1
         MAR  *,AR1
*
         LACC  #CERO
         SACB
           RPTB  FIN_M    ; <--
            LACC  *+
            CRGT          ; Halla máximo
FIN_M       NOP           ; --->
         SACL  MAX
*
```

```
        LACC #N
        SAMM BRCR
        LACC  #CERO
        SACB
          RPTB  FIN_M2  ; <--
           LACC  *-
           CRLT         ; Halla mínimo
FIN_M2     NOP          ; --->
        SACL  MIN
        ABS
        SACB
        LACC MAX
        CRGT
        SACL MAXA       ; Halla máximo absoluto
*
FIN     NOP
        B   FIN
        .end
```

4.5.5. Rectifica una señal

```
*
*   recti.asm
*    @larry
*    Rectifica una señal senoidal
*    La seal esta en loc. 1000h
       .mmregs
N      .set   1499
N1     .set   01000h
CERO   .set   00
*
       .ps  0A00h
       .entry
      SETC SXM
       LACC #N
       SAMM BRCR
       LAR AR1,#N1
       MAR *,AR1
       RPTB FIN_B     ;  <---
       LACC *
       BCND NEGA,LT   ; Salta si señal negativa
```

```
        B     FIN_B      ; Señal positiva
NEGA    LACL #CERO       ; Hace cero la señal
        SACL *
FIN_B   MAR *+,AR1       ;  --->

FIN     NOP
        B     FIN
        .end
```

4.5.6. Ordena en forma descendente un conjunto de datos

```
*       ordena5.asm
*        @larry
*        Ordena un conjunto de datos
*        Utiliza método de la burbuja
*        !!!!!
           .mmregs
           .ds  0F00h
TEMP       .word  0              ; Localidad temporal
N          .set  25
N1         .set  24
N2         .set  23
*
           .ds  01000h           ; datos    X(i)
X          .word  20,15,14,1,8,21,3,5,6,11,18,2,10
X1         .word  25,17,4,7,9,24,12,23,13,22,19,16
*
           .ps  0A00h
           .entry
           LDP  #TEMP
           LAR  AR3,#N2          ; N2=23
           MAR  *,AR1
UNO         LAR  AR1,#X          ; localidad inicio <----
            LACC #N              ; N2=23    , indice i
            SAMM BRCR
               RPTB FIN_0        ; <------
               LACC *+           ; X(i)
               SACL TEMP
               SACB
               LACC *            ; X(i+1)
               CRGT              ; X(i+1) > X(i)
```

```
              BCND MAYOR,C
NO            B   FIN_0
MAYOR         MAR *-,AR1        ; Intercambia X(i+1)-->X(i)
              SACL  *+
              LACC  TEMP
              SACL  *
FIN_0         NOP               ; ----> a RPTB
          MAR   *,AR3
          BANZ  UNO,*-,AR1     ;------>  a UNO
*
FIN       NOP
          B  FIN
          .end
```

4.5.7. Obtiene la raíz cuadrada

Este programa está basado en un algoritmo rápido de obtención de raíces cuadradas [13] que funciona en forma similar a un convertidor de aproximaciones sucesivas.

```
*
*       RAIZ CUADRADA DE UN NUMERO a 16 bits
*       Qent = número de bits para representar la parte entera
*       Raiz_02.asm
*       @Larry
*       !!!!!!!!!!!
        .mmregs
        .ds     01000h
Y       .word   0       ;  Resultado en Qent/2
TEMP    .word   08000h  ;  Valor inicial de Y
*X       .word   30000  ;  Valor de X=Y*Y  en Q0
X       .q8     130.55  ;  X en Q8 , Y en Q 12
MASK    .word   07FFFh  ;  Máscara para operación AND
MASKF   .word   0FFFFh  ;
N       .set     15     ;  N+1 pruebas
*
        .ps
        .entry 0A00h
        LDP     #TEMP
        SPM     #0          ; Shift(m) = 0
        CLRC    SXM         ; modo no signado
*
        LACL    #N
```

```
        SAMM    BRCR
*
          RPTB   FIN_R
           LACL   Y
           ADD    TEMP
           SACL   Y
           SAMM   TREGO
*
           LACC   X,16          ; ACCH = X
           MPYU   Y
           SPAC                 ; ACC = Y*Y
           BCND   FIN_ME,GT     ; Si Y*Y < X deja el bit
MENOR      LACL   Y
           AND    MASK          ; Limpia bit de prueba
           SACL   Y
*
FIN_ME     LACC   MASKF,16
           ADD    MASK
           ROR                  ; Corre MASK para AND
           SACL   MASK
           LACL   TEMP
           ROR                  ; Corre bit de prueba
           SACL   TEMP
FIN_R      NOP
*
FIN        NOP
           B    FIN
          .end
```

4.5.8. Filtro promediador tipo FIR

```
*          FILTRO PROMEDIADOR
*          Dada una señal de 1500 pts.
*          En dirección   1000h
            .mmregs
            .ds  0f00h
N          .set     1499            ; número de datos
N8         .set     7               ; N-1, longitud  del filtro - 1
x          .word    0               ; entrada datos
xn         .word    0,0,0,0,0,0     ; Memoria para el filtro
xf         .word    0,0             ; dato final + una loc.
```

```
EN      .word   1,1,1,1,1,1,1,1
y       .set    02000h          ; Dir. inic. salida
*
        .ps  0A00h
        .entry
*
        LDP  #x
        SETC SXM                ; Modo signado
        LAR AR1,#01000h         ; entrada de datos
        LAR AR4,#y              ; salida  de datos


        MAR *,AR1
        LACC  #N
        SAMM  BRCR
*************************************
        RPTB  FIN_B
        LAR AR2,#xf
        LACC *+,0,AR2           ; lee muestra x(n) de entrada
        SACL x
            ZAP
               RPT #N8
               MACD #HN,*-
               APAC
            BSAR 8
            MAR *,AR4
           SACL *+,0,AR1        ; Escribe salida
FIN_B   NOP
*********************************************
FIN     NOP
        NOP
        B    FIN
        .end
```

4.5.9. Algoritmo de la media de los mínimos cuadrados (LMS)

Para mayor detalle ver [4]

```
*****************************************************
*       ALGORITMO DE FILTRADO ADAPTABLE
*       DE LA MEDIA DE LOS MINIMOS CUADRADOS LMS
*       Archivo :   lms_0.asm
```

```
*       Entrada x(k) en loc.   1000h
*       Salida  x'(k) en loc   1500h , valor estimado
*********************************************************
*
        .mmregs                     ; incluye registros mapeados
        .ds     0F00h
*
*       Vectores del algoritmo
*
YS      .word    0                  ;   Salida
ER      .word    0                  ;   error
ER1     .word    0                  ;   e*u
U       .q15     0.0125
*
YK    .word    0
YK1   .word    0,0,0,0,0,0,0,0,0,0  ; Y(K-1) Aparta 10 loc. de memoria
YKN   .word    0                    ; YN muestra penúltima
*
AK    .word    0,0,0,0,0,0,0,0,0,0  ; A(K-1)
AN    .word    0                    ; Ultima A
*
N         .set     10               ; N
N1        .set      9               ; N-1
UNO       .set      1
TOT       .set      512             ; Total de Datos
*****************************************************************
        .ps     00A00h
        .entry
        SETC  INTM        ; des-habilita interrupciones mascarables
        SETC    SXM
        SPM     0         ; sin corrimiento de Preg al cargarse a ACC
*********

          LAR     AR5,#01500h     ; Apunta a Datos de Salida
          LAR     AR6,#TOT        ; Total de datos 256
          LAR     AR7,#01000h     ; Apunta a Datos Entrada

INICIO
          LDP     #YK
          MAR     *,AR7
          LACC    *+,12           ; Dato en Q15+12 = Q27
```

```
            SACH    YK              ; Dato entrada en Q11
*
            LAR     AR0,#YKN        ; primer retraso de señal
            MAR     *,AR0
            ZAP
            RPT     #N
CIC1        MACD    #AK,*-          ; obtiene y^(n) = Yki*Aki*2^23
            APAC
            ADD     #UNO,11
*
            MAR     *,AR5
            SACH    *+,4            ; YS salida en Q11
            SACH    YS,4
*
            LACL    YK
            SUB     YS
            SACL    ER              ; ER en Q11
*
            LT      U               ; ER en Q11
            MPY     ER              ; U  en Q15
            PAC                     ; U*ER*2^26
            SACH    ER1,3           ; ER1 Q13
*
            LACL    #N1
            SAMM    BRCR
            LAR     AR0,#AK
            LAR     AR1,#YKN
            MAR     *,AR0
            LT      ER1

            RPTB    CIC2
            LACC    *,12,AR1        ; ACC  = AK*2^24
            MPY     *-,AR0          ; PREG = ER1*YK1*2^24
            APAC
            ADD     #UNO,11
CIC2        SACH    *+,4
*
            MAR *,AR6
            BANZ INICIO,*-

FIN         NOP
```

```
NOP
B    FIN
.end
```

Capítulo 5

Ejemplos de programas en tiempo real

Los programas que se presentan a continuación se ejecutan en tiempo real. De la figura 5.1, en un canal del osciloscopio se puede observar la señal de entrada $x(n)$ que proviene del generador de señales y en el otro canal la salida $y(n)$ procesada por el DSP C5x instalado en la tarjeta DSK. Para correr estos programas, una vez ensamblados sin errores, se puede utilizar el programa cargador `dsk51.exe`. Se sugiere al lector modificar estos programas y agregarles algunos cambios para efectuar procesos diferentes sobre las señales.

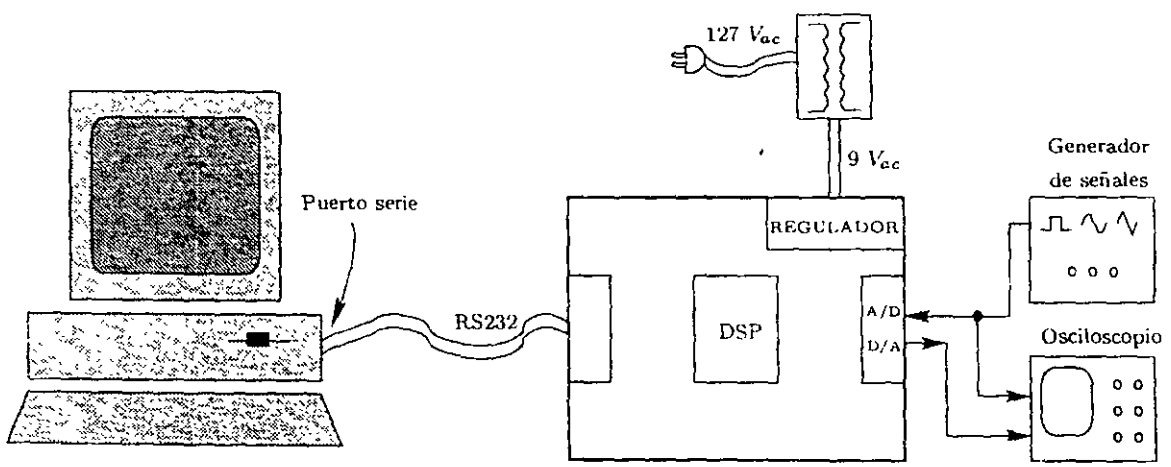


Figura 5.1: Conexión PC–DSK–Osciloscopio–Generador

5.1. Adquisición y despliegue de señales

```
*       LECTURA Y ESCRITURA DEL CONVERTIDOR ANALÓGICO DIGITAL
*       LEE-ESCRIBE DEL A/D y DEL D/A
*       Utiliza interrupciones con el TMS320C50
*
*       Frecuencias de muestreo
*        Fs = CLKOUT1/2/TA/TB       Para transmisión
*        Fs = CLKOUT1/2/RA/RB       Para recepción
*        Máximo valor de TA,TB,RA y RB, es de 3 a 63  ( 6 bits )
*
        .mmregs                 ; incluye registros mapeados

         .ds      0F00h
TA       .word     5
RA       .word     5
TB       .word     15
RB       .word     15
;
AIC_CTR  .word     08h

MASK     .set     00080h

        .ps      0080ah
         B        RINT          ; Fija vector de recepción de Interrup.


        .ps      00a00h
        .entry                  ; Inicia dirección de PC
        SETC     INTM           ; deshabilita Int. Mask
        LDP      #00            ; carga pag. cero para DXR
        OPL      #0800,PMST     ; reubica Vectores de INt.

        SPLK     #022h,IMR      ; habilita In. de Transmision
                                ; resetear puerto serie del c50
        CALL     AICINIT        ; --->

        LAMM     IMR            ; carga ACCL con reg. mapeado
        OR       #32h           ; Habilita int de tran/Recep ( XINT / RINT )
        SAMM     IMR            ; Salva ACCL en registro mapeado IMR
        LAMM     SPC
```

```
        AND     #0FFF0h         ; Modo continuo
        SAMM    SPC
        CLRC    INTM

ESPERA: NOP
        NOP
        B       ESPERA          ; Espera interrupción

* Subrutina de atención de interrupción, dato recibido.
RINT:                           ;
        LAMM    DRR             ; lee el dato nuevo en DRR
        AND     #0FFFCh         ; ceros a dos bits de control d0 y d1
*                               ; --- incluir algun proceso de la señal ---
        SACL    DXR             ; ACCL --->  DXR
        RETE

        .include ''ADC_INIT.asm''    ; incluye rutina de configuración del
                                     ; convertidor A/D

        .end
```

En la subrutina de atención de interrupción de recepción de dato, después de haber adquirido la muestra actual y haberle quitado los dos bits menos significativos (instrucción AND #0FFFCh , ya que el convertidor es 14 bits, y los dos bits LSB son de configuración), el lector puede incluir algún proceso, por ejemplo atenuar la señal, rectificarla, recortarla, saturarla, etc, luego correr el programa para observar de nuevo su procesamiento.

5.2. Filtro paso bajas

```
;                       FILTRO PASO BAJAS
;
;    PARA tms50
;    Archivo :       fir_pb.asm
;
;    N = 80, fc = 1 Khz, Ventana de Blackman
;
;    CONVERSIÓN, frecuencias de muestreo del A/D
;                    Fs = CLKOUT1/2/TA/TB  Para transmisión
;                    Fs = CLKOUT1/2/RA/RB  Para recepción
;                    Máximo valor de TA,TB,RA,RB, de 3 a 63  ( 6 bits )
```

```
; ------------------------------------------------------------------------

          .MMREGS
          .ds       0f00h
TA        .word     5
RA        .word     5


TB        .word     15
RB        .word     15
AIC_CTR   .word     08h


OUTPUT    .word     0
TEMP      .word     0
TEMP1     .word     0
*
*       COEFICIENTES  en Q15      fc= 1K   si    Fs = 10 K
*
h0      .word         0     ;  40        0.0000
h1      .word      -157     ;  39        -0.0048
h2      .word      -261     ;  38        -0.0080
h3      .word      -268     ;  37        -0.0082
h4      .word      -170     ;  36        -0.0052
h5      .word         0     ;  35        -0.0000
h6      .word       180     ;  34        0.0055
h7      .word       301     ;  33        0.0092
h8      .word       310     ;  32        0.0095
h9      .word       198     ;  31        0.0060
h10     .word         0     ;  30        0.0000
h11     .word      -211     ;  29        -0.0065
h12     .word      -354     ;  28        -0.0108
h13     .word      -367     ;  27        -0.0112
h14     .word      -236     ;  26        -0.0072
h15     .word         0     ;  25        -0.0000
h16     .word       255     ;  24        0.0078
h17     .word       431     ;  23        0.0132
h18     .word       451     ;  22        0.0138
h19     .word       292     ;  21        0.0089
h20     .word         0     ;  20        0.0000
h21     .word      -323     ;  19        -0.0098
```

```
h22     .word     -551    ;  18      -0.0168
h23     .word     -584    ;  17      -0.0178
h24     .word     -383    ;  16      -0.0117
h25     .word        0    ;  15      -0.0000
h26     .word      438    ;  14       0.0134
h27     .word      763    ;  13       0.0233
h28     .word      827    ;  12       0.0252
h29     .word      557    ;  11       0.0170
h30     .word        0    ;  10       0.0000
h31     .word     -681    ;   9      -0.0208
h32     .word    -1240    ;   8      -0.0378
h33     .word    -1417    ;   7      -0.0432
h34     .word    -1022    ;   6      -0.0312
h35     .word        0    ;   5      -0.0000
h36     .word     1533    ;   4       0.0468
h37     .word     3307    ;   3       0.1009
h38     .word     4960    ;   2       0.1514
h39     .word     6131    ;   1       0.1871
h40     .word     6131    ;   1       0.1871
h41     .word     4960    ;   2       0.1514
h42     .word     3307    ;   3       0.1009
h43     .word     1533    ;   4       0.0468
h44     .word        0    ;   5      -0.0000
h45     .word    -1022    ;   6      -0.0312
h46     .word    -1417    ;   7      -0.0432
h47     .word    -1240    ;   8      -0.0378
h48     .word     -681    ;   9      -0.0208
h49     .word        0    ;  10       0.0000
h50     .word      557    ;  11       0.0170
h51     .word      827    ;  12       0.0252
h52     .word      763    ;  13       0.0233
h53     .word      438    ;  14       0.0134
h54     .word        0    ;  15      -0.0000
h55     .word     -383    ;  16      -0.0117
h56     .word     -584    ;  17      -0.0178
h57     .word     -551    ;  18      -0.0168
h58     .word     -323    ;  19      -0.0098
h59     .word        0    ;  20       0.0000
h60     .word      292    ;  21       0.0089
h61     .word      451    ;  22       0.0138
h62     .word      431    ;  23       0.0132
```

```
h63     .word       255     ;  24        0.0078
h64     .word         0     ;  25       -0.0000
h65     .word      -236     ;  26       -0.0072
h66     .word      -367     ;  27       -0.0112
h67     .word      -354     ;  28       -0.0108
h68     .word      -211     ;  29       -0.0065
h69     .word         0     ;  30        0.0000
h70     .word       198     ;  31        0.0060
h71     .word       310     ;  32        0.0095
h72     .word       301     ;  33        0.0092
h73     .word       180     ;  34        0.0055
h74     .word         0     ;  35       -0.0000
h75     .word      -170     ;  36       -0.0052
h76     .word      -268     ;  37       -0.0082
h77     .word      -261     ;  38       -0.0080
h78     .word      -157     ;  39       -0.0048
h79     .word         0     ;  40        0.0000

*       LOCALIDADES PARA  ENTRADA XN y sus retardos
XN       .word     0,0,0,0,0,0,0,0,0,0    ; 80  localidades
XN1      .word     0,0,0,0,0,0,0,0,0,0
XN2      .word     0,0,0,0,0,0,0,0,0,0
XN3      .word     0,0,0,0,0,0,0,0,0,0
XN4      .word     0,0,0,0,0,0,0,0,0,0
XN5      .word     0,0,0,0,0,0,0,0,0,0
XN6      .word     0,0,0,0,0,0,0,0,0,0
XN7      .word     0,0,0,0,0,0,0,0,0
XNLAST   .word     0;

         .ps       0080Ah
rint:    B         RECIBE
xint:    B         TRANSMITE

         .ps       0A00h
         .entry
*-----------------------------------------------------------------

         SETC      INTM
         LDP       #0
         OPL       #0834h,PMST
         LACC      #0
```

```
        SAMM    CWSR
        SAMM    PDWSR
        SETC    SXM             ;  SXM = 1
        SPLK    #022h,IMR
                                ; resetear puerto serie
        CALL    AICINIT
        SPLK    #12h,IMR
        CLRC    OVM
        SPM     0
        CLRC    INTM

ESPERA
        B       ESPERA

* Rutina de atención de interrupción y realización del filtro FIR
RECIBE
        LDP     #XN

        LAMM    DRR
        AND     #0FFFCh
        SACL    XN

        LAR     AR0,#XNLAST
        ZAP
        MAR     *,AR0

        RPT     #79
        MACD    #h0,*-
        APAC
        SACH    OUTPUT
        LACC    OUTPUT

        SACH    OUTPUT,1
*         SFL

        AND     #0FFFCh
        SAMM    DXR
        RETE

TRANSMITE
        RETE                    ; no hace nada
```

```
        .include ''ADC_INIT.asm''     ; incluye rutina de configuración del
                                      ; convertidor A/D
        .end
```

5.3. Generador de Eco

Este programa trabaja en tiempo real, para una señal de audio de entrada genera una salida con eco. Para más información ver [4]

```
*
*       Pasos:
*
*       0) Limpia memoria de 300h-4ffh
*       1) Lee entrada x(n) y lo almacena
*       2) salida y(n) = x(n) +  x1(n-D)
*       3) Retroalimentación x1(n) = x(n) + G x1(n-D)
*       4) Efectua DMOV
*       @larry
        .mmregs          ; incluye registros mapeados

         .ds      0800h
TA       .word     5             ; 5 Fcut = 8 KHz
RA       .word     5             ; 5 Fcut = 8 KHz
TB       .word     15            ; 15  Fs = 2*Fcut
RB       .word     15            ; 15 Fs = 2*Fcut

XN       .word     0             ; Localidad de entrada
YN       .word     0             ; Localidad de salida

AIC_CTR  .word     08h

MASK     .set    00080h  ;
GOUT     .set    07FF0h  ; Ganancia de salida de ECO
GBACK    .set    00002h  ; Ganancia de Retroalimentación
C_INI    .set    01000h  ; Loc. Inicio de retardo X1(n)
C_FIN    .set    018FFh  ; Loc. Fin de retardo    X1(n-D)
DELAY    .set    008FFh
*
        .ps     0080ah
         B       RINT    ; Fija vector de recepción de interrupción.
```

```
*
        .ps     00a00h
        .entry

        SETC    SXM
        SETC    INTM            ; deshabilita Int. Mask
        LDP     #00             ; carga pag. cero para DXR
        OPL     #0834,PMST      ; reubica Vectores de Interrupción.
                                ; OVLY = 1 , RAM = 1, MP/mc = 1

        SPLK    #022h,IMR       ; habilita In. de Transmision
        CALL    AICINIT ; --->

        LAMM    IMR      ; carga ACCL con reg. mapeado
        OR      #30h     ; Habilita int de tran/Recep. ( XINT / RINT )
        SAMM    IMR      ; Salva ACCL en registro mapeado IMR
        LAMM    SPC
        AND     #0FFF0h     ; Modo continuo
        SAMM    SPC

        LAR     AR0,#C_INI ; Localidad de x1(d)
        LAR     AR2,#C_FIN ; Loc. de x(n-D)

        MAR     *,AR1

        ZAP
        RPT     #DELAY          ; Limpia memoria (No funciona RPTZ)
        SACL    *-

        CLRC    INTM

ESPERA:  NOP
         NOP
         B       ESPERA         ; Espera interrupción


RINT:
        LDP     #XN
        LAMM    DRR       ; lee el dato nuevo en DRR
        SACL    XN        ; Almacena entrada
```

```
        ADD     YN        ; y(n) = x(n) +  x1(n-D)
        AND     #0FFFCh   ; ceros a dos bits de control d0 y d1
        SAMM    DXR       ; ACCL --->  DXR

        LAR     AR1,#C_FIN ; localidad del retardo
        MAR     *,AR1

        LT      *         ; x1(n) = x(n) + G * x1(n-D)
        MPY     #GOUT
        PAC
        SACH    YN,0      ;  G * x1 (n - D)


        LT      *,AR0
        MPY     #GBACK
        LACC    XN,16
        APAC
        SACH    *,0,AR1
        SFL

        RPT     #DELAY    ; Efectua movimientos de Memoria
        DMOV    *-

        RETE

        .include ''ADC_INIT.asm''     ; incluye rutina de configuración del
                                      ; convertidor A/D
        .end
```

5.4. Rutina del convertior A/D y D/A TLC320C46

Esta subrutina le envía parámetros al convertidor A/D y D/A para su configuración.

```
*       ADC_INIT.asm
*****************************************************************
*  INICIALIZA AL CONVERTIDOR A/D y D/A TLC320C46          *
*****************************************************************
*
AICINIT: SPLK    #20h,TCR          ; Genera 10 MHz de Tout
         SPLK    #01h,PRD          ; Reloj maestro AIC
         MAR     *,AR0
```

```
        LACC    #0008h          ; Modo no continuo
        SACL    SPC             ; FSX  input
        LACC    #00c8h          ; 16 bit words
        SACL    SPC
        LACC    #080h           ; Resetea al AC
        SACH    DXR
        SACL    GREG
        LAR     AR0,#0FFFFh
        RPT     #10000          ; alto después de 10000 ciclos
        LACC    *,0,AR0         ; (.5ms at 50ns)
        SACH    GREG
        SETC    SXM
        ;-------------------------
        LDP     #TA
        LACC    TA,9            ; Inicializa registos TA y RA
        ADD     RA,2
        CALL    AIC_2ND
        ;-------------------------
        LDP     #TB
        LACC    TB,9            ; Inicializa  reg.  TB y RB
        ADD     RB,2            ;
        ADD     #02h            ;
        CALL    AIC_2ND         ;
        ;-------------------------
        LDP     #AIC_CTR
        LACC    AIC_CTR,2       ; Inicializa registro control
        ADD     #03             ; 4h SIN Auxin    53h Auxin gain=4
        CALL    AIC_2ND
        RET

AIC_2ND:
        LDP     #00
        SACH    DXR
        CLRC    INTM
        IDLE
        ADD     #6h,15          ; 0000 0000 0000 0011 XXXX XXXX XXXX XXXX b
        SACH    DXR
        IDLE
        SACL    DXR
        IDLE
        LACL    #0
```

```
        SACL    DXR             ; Asegura el envío de palabra
        IDLE
        SETC    INTM
        RET
*
        .end
```

Capítulo 6

Programas propuestos.

En esta sección se propone una lista de programas que el lector debe realizar para afianzar sus conocimientos sobre el TMS320C50. Eventualmente estos programas o similares pueden ser proyectos a realizar durante el curso de Procesamiento Digital de Señales.

Programa 1

Utilizando el TMS320C50 realizar el ordenamiento de las componentes de un vector

$$\mathbf{A} = [a_1\ a_2\ a_3\ a_4\ a_5]$$

dentro de una matriz:

a)
$$\begin{bmatrix} a_1 & a_2 & a_3 & a_4 & a_5 & a_4 & a_3 & a_2 & a_1 \\ 0 & a_1 & a_2 & a_3 & a_4 & a_3 & a_2 & a_1 & 0 \\ 0 & 0 & a_1 & a_2 & a_3 & a_2 & a_1 & 0 & 0 \\ 0 & 0 & 0 & a_1 & a_2 & a_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a_1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

b)
$$\begin{bmatrix} 0 & 0 & 0 & 0 & a_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & a_1 & a_2 & a_1 & 0 & 0 & 0 \\ 0 & 0 & a_1 & a_2 & a_3 & a_2 & a_1 & 0 & 0 \\ 0 & a_1 & a_2 & a_3 & a_4 & a_3 & a_2 & a_1 & 0 \\ a_1 & a_2 & a_3 & a_4 & a_5 & a_4 & a_3 & a_2 & a_1 \end{bmatrix}$$

c)
$$\begin{bmatrix} a_1 & 0 & 0 & 0 & 0 \\ a_2 & a_1 & 0 & 0 & 0 \\ a_3 & a_2 & a_1 & 0 & 0 \\ a_4 & a_3 & a_2 & a_1 & 0 \\ a_5 & a_4 & a_3 & a_2 & a_1 \\ a_4 & a_3 & a_2 & a_1 & 0 \\ a_3 & a_2 & a_1 & 0 & 0 \\ a_2 & a_1 & 0 & 0 & 0 \\ a_1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

d)
$$\begin{bmatrix} 0 & 0 & 0 & 0 & a_1 \\ 0 & 0 & 0 & a_1 & a_2 \\ 0 & 0 & a_1 & a_2 & a_3 \\ 0 & a_1 & a_2 & a_3 & a_4 \\ a_1 & a_2 & a_3 & a_4 & a_5 \\ 0 & a_1 & a_2 & a_3 & a_4 \\ 0 & 0 & a_1 & a_2 & a_3 \\ 0 & 0 & 0 & a_1 & a_2 \\ 0 & 0 & 0 & 0 & a_1 \end{bmatrix}$$

e)
$$\begin{bmatrix} a_5 & a_4 & a_3 & a_2 & a_1 \\ a_4 & a_3 & a_2 & a_1 & 0 \\ a_3 & a_2 & a_1 & 0 & 0 \\ a_2 & a_1 & 0 & 0 & 0 \\ a_1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Programa 2

Dados los siguientes polinomios, realiza un programa que encuentre las raíces de los mismos. Utilizar cualquier método numérico programado en el TMS320C50.

a) $x^5 - 10{,}5x^4 + 34{,}06x^3 - 35{,}1x^2 + 11{,}74x - 1{,}2$

b) $x^5 - 12{,}5x^4 + 37{,}99x^3 - 15{,}875x^2 - 0{,}38x + 0{,}16$

c) $x^5 - 15{,}5x^4 + 89{,}34x^3 - 228{,}58x^2 + 225{,}98x - 20{,}4$

d) $x^5 + 1{,}1x^4 - 73x^3 - 347{,}9x^2 - 348x + 270$

e) $x^5 + 0{,}9x^4 - 59{,}7x^3 + 132{,}1x^2 + 110{,}7x - 81$

f) $x^5 - 3{,}73x^4 - 8{,}415x^3 + 31{,}1566x^2 - 12{,}0903x + 1{,}2448$

g) $x^5 - 11{,}86x^4 + 14{,}604x^3 + 7{,}4704x^2 - 15{,}401x + 4{,}5926$

Programa 3

Proponiendo dos matrices A y B de 5×5 con números fraccionarios, realizar un programa que efectúe el producto $C = A \times B$.

Programa 4

Dada una señal senoidal en memoria (localidades 1000h) de longitud 1500 puntos, desarrollar un programa que efectúe:

a) Un rectificado de onda completa (positivo).

b) Un rectificado de onda completa (negativo).

c) Un rectificado de media onda (positivo).

c) Un rectificado de media onda (negativo).

d) Un recortado de la señal $a \pm \frac{1}{2}$ amplitud máxima.

Programa 5

Dados treinta y dos puntos de una señal, efectuar un programa que copie estos puntos en otro bloque de memoria en forma decimada (usar direccionamiento en carry reverso).

Programa 6

Diseñar y realizar los siguientes filtros tipo IIR en tiempo real.

a) Filtro paso bajas:
$fc = 1$ khz a 3 db
$fs = 1{,}5$ khz a 15 db

b) Filtro paso altas:
$fc = 2$ khz a 3 db
$fs = 1{,}5$ khz a 15 db

c) Filtro paso
$f_0 = 4{,}5$ khz $BW = 50$hz

d) Filtro supresor de banda
$f_0 = 4$ khz $BW = 50$hz

Programa 7

Realizar el programa 4 en tiempo real.

Programa 8

Realizar un programa en tiempo real que module en amplitud (A.M.) a una señal de entrada, donde la señal moduladora debe generarse por medio de un oscilador senoidal.

— Especificar el ancho de banda de la señal a modular.

— Especificar la frecuencia de la señal moduladora.

Programa 9

Realizar un programa en tiempo real cuya salida genere varias señales senoidales de igual amplitud pero que para diferentes intervalos se observen diferentes frecuencias y se repita indefinidamente. Ejemplo:

Intervalo de 1 segundo $f = 500$ hz
Intervalo de 2 segundos $f = 1000$ hz
Intervalo de 3 segundos $f = 5000$ hz

Programa 10

Realizar un programa en tiempo real que module en frecuencia F.M. Especificar el ancho de banda de la señal a modular.

Programa 11

Habiendo realizado la modulación A.M. y usando la salida como entrada de otro starter kit, desarrollar un programa que efectúe la demodulación y obtener la señal original.

Programa 12

Dadas dos secuencias propuestas $x(n)$ longitud 500 y $h(n)$ longitud 20 a 32 bits en formato Q28, obtener y(n)=x(n)*h(n). En este caso hay que implementar un multiplicador a 32x32 bits.

Programa 13

Dado el vector r=[0.999 0.889 0.455 0.332 0.223 0.111 0.0234], realizar un programa que lo ordene en una matriz tipo Toeplitz.

Programa 14

Dada una señal senoidal de N puntos, realizar un programa que sólo deje pasar la primera mitad del ciclo positivo y la segunda mitad del ciclo negativo.

Programa 15

Filtrado de señales: Dada una señal senoidal con ruido agregado (seno-r.dat en formato Q0) diseñar y programar un filtro paso bajas FIR.

Programa 16

Filtrado de señales: Dada una señal senoidal con ruido agregado (seno-r.dat en formato Q0) diseñar y programar un filtro paso bajas IIR.

Programa 17

Dadas dos señales de longitud mínima 500 puntos, realizar un programa que calcule la autocorrelación normalizada.

Programa 18

Proponer una señal cuadrada de N puntos de entrada y realizar un programa que la module en FSK.

Programa 19

Generar una señal PWM de 5000 puntos y especificarla.

Programa 20

Diseñar y programar un VCO con especificaciones.

Programa 21

Dada una señal de N=32 puntos, realizar un programa que encuentre su espectro discreto.

Programa 22

Realizar un programa que adquiera una señal real del convertidor A/D del Starter Kit del DSP C5x y almacene 5,000 datos en memoria RAM, posteriormente salvar en un archivo esta información y desplegarla con algún programa de graficación.

Programa 23

Realizar un algoritmo para el control de ganancia adaptivo (CGA) el cual usa un estimado de la potencia de la señal de entrada. Considérese el siguiente sistema con entrada $x(n)$ y salida $y(n)$.

Se requiere que la salida $y(n)$, sea una versión escalada de $x(n)$, pero con una potencia constante.

$$y(n) = G(n)x(n) \tag{6.1}$$

donde $G(n)$ es una ganancia variante en el tiempo que se ajuste a sí misma (se adapta) a la potencia estimada de la señal de entrada.

Si la potencia estimada de la señal de entrada usando las L muestras más reciente de $x(n)$

$$P_x(n) = \frac{1}{L}\sum_{i=n-L+1}^{n} x^2(i) \tag{6.2}$$

Por simplicidad limitaremos $G(n)$ a un número entero de potencia de 2:

$$G(n) = 2^{P(n)}$$

donde $P(n)$ es un entero. De esta forma para alterar $G(n)$ simplemente se puede hacer con corrimientos al lado derecho o izquierdo, pero al hacer esto estamos introduciendo saltos indeseables en la función de ganancia.

El problema es escoger una $P(n)$ a cada instante de n, tal que el promedio de la potencia de salida $y(n)$ esté dentro de rango:

$$\frac{P_0}{4} \leq P_y(n) < P_0; \quad \frac{P_0}{4} \leq P_x(n)G^2(n) < P_0 \tag{6.3}$$

$$\frac{P_0}{2^{2p(n)+2}} \leq P_x(n) < \frac{P_0}{2^{2p(n)}} \tag{6.4}$$

donde P_0 es la potencia de salida deseada.

La dependencia de $G(n)$ y $P_x(n)$ es resumida en la siguiente tabla

$P_x(n)$	$G(n)$	$P(n)$
$4P_0 \leq P_x(n)$	$\frac{1}{4}$	-2
$P_0 < P_x(n) \leq 4P_0$	$\frac{1}{2}$	-1
$\frac{P_0}{4} < P_x(n) \leq P_0$	1	0
$\frac{P_0}{16} < P_x(n) \leq \frac{P_0}{4}$	2	1
$\frac{P_0}{64} < P_x(n) \leq \frac{P_0}{16}$	4	2
$\frac{P_0}{64} < P_x(n)$	8	3

Realizar el sistema CGA descrito usando el TMS320C50.

La ecuación (6.2) puede ser reescrita en la siguiente forma

$$P_x(n) = \frac{1}{L}\sum_{i=n-L+1}^{n} x^2(i) = \frac{1}{L}\left[x^2(n) + \sum_{i=n-L}^{n-1} x^2(i) - x^2(n-L)\right] = P_x(n-1) + \frac{x^2(n)}{L} - \frac{x^2(n-} \tag{6.5}$$

Capítulo 7

El DSP TMS320C54x

7.1. Introducción

El DSP TMS320C54x (C54x) de Texas Instruments (TI) es un procesador digital de señales orientado a aplicaciones incrustadas de tiempo real en áreas tales como comunicaciones, este DSP está basado en un hardware mejorado del TMS320C50, una arquitectura tipo Harvard modificada, alto grado de paralelismo, bajo consumo de potencia, modos de direccionamiento versátiles y un conjunto de instrucciones que mejoran el desempeño. La familia TMS320C5000 (C5000) es igual al C54x en cuanto a la arquitectura y la programación, a excepción que el C5000 contiene otros periféricos internos más poderosos para comunicaciones.

En este material se da una breve introducción al DSP C54x, ya que el autor considera que cuando una persona conoce la filosofía de los DSPs de la familia TMS320 de Texas Instruments (TI) o que haya programado algunos de ellos, es capaz de migrar hacia otro DSP en muy poco tiempo.

7.2. Características generales

— CPU:
- Arquitectura multibuses: uno para programa, tres para datos y cuatro para direcciones.
- Unidad ALU de 40 bits, con dos acumuladores independientes de 40 bits.
- Multiplicador en paralelo de 17x17 bits acoplado a un sumador dedicado (MAC).
- Unidad de comparación, selección y almacenamiento (CSSU) para codificación Viterbi.
- Codificador de exponente para calcular el exponente de un acumulador de 40 bits.

- Dos generadores de dirección, es decir, dos unidades ARAU y ocho registros auxiliares mapeados (AR0-AR7).

— Memoria: 192 K-words en el espacio de memoria
- 64 K-words para memoria programa, 64 K-words para datos y 64 K-words para puertos I/O. El C548 puede direccionar hasta 8 M-words (divididos en 128 páginas de 64 K) y trae un registro extra para apuntar a estas páginas y seis instrucciones extras para direccionar el espacio de programa extendido.

— Instrucciones:
- Para repetición de una instrucción y repetición de bloques de instrucciones.
- Para movimiento de bloques de datos y programa.
- Con operandos de 32 bits.
- Para lectura de dos o tres operandos simultáneos.
- Aritméticas con almacenamiento y carga paralela.
- Almacenamiento condicional.
- Lenguaje de programación algebraico.

— Periféricos internos:
- Estados de espera programables por software.
- Generador de una malla de fase amarrada (PLL).
- Control externo de bus para deshabilitación de buses externos.
- Bus de datos con características de retención (Holder).
- Timer programable.
- Tres puertos seriales y dependiendo de la versión pueden ser tipo host, serial o puerto multiplexado por división de tiempo (TDM).

— Potencia:
- Opera en modo de bajo consumo con instrucciones IDLE1, IDLE2 e IDLE3.
- Emula el estandar 1149.1 de IEEE.
- Velocidades: 25/20/12.5/10 ns en un ciclo de instrucción (40/50/66/80/100 MIPS).
- Maneja seis niveles de pipeline: prebúsqueda, búsqueda, decodificación, acceso, lectura y ejecución.

Arquitectura

Su arquitectura tipo Harvard de buses separados para datos y programa permite el acceso simultáneo a instrucciones y datos proveyendo un alto grado de paralelismo, el C54x puede

ejecutar tres lecturas y una escritura en un ciclo simple .

Buses

La arquitectura del C54x está construída sobre ocho buses de 16 bits (figura 7.1):

— Bus de programa: transportan el código de instrucción y los operandos inmediatos de memoria programa.

— Buses de datos: Interconectan varios elementos tales como el CPU, generador de direcciones de datos y programa, periféricos y memoria dato.
- Los buses CB y DB transportan los operandos que son leídos de memoria dato.
- El bus EB transporta los datos para ser escritos en memoria.

— Buses de direcciones: PAB, CAB, DAB y EAB.

Organización de la memoria

La memoria del C54x está organizada en tres espacios: 64 K-words para memoria programa, 64 K-words para datos y 64 K-words para puertos I/O, estos espacios de memoria se pueden seleccionar por medio de las señales /PS, /DS e /IS respectivamente (pines externos). La memoria puede ser del tipo RAM o ROM, la memoria RAM puede ser SARAM (de simple acceso por ciclo de máquina) y DARAM (de doble acceso por ciclo de máquina).

El DSP C54x trae al menos una memoria ROM de 2 K-words mapeada en la dirección F800h a FFFFh que contiene:
- Un programa bootloader que levanta desde puerto serie, memoria externa, puertos I/O o interface de host.
- Una tabla de la Ley μ de 256 valores.
- Una tabla de la Ley A de 256 valores.
- Una tabla de senos de 256 valores para realizar la FFT.
- Una tabla de vectores de interrupción, el vector de reset está mapeado en localidad FF80h.
- 26 registros mapeados de acceso en un ciclo (0000h a 001Eh). Dentro de éstos están los registros de estado ST0, ST1 y PMST. Las tres partes de los acumuladores A y B también son registros mapeados (AG, AH, AL, BG, BH y BL)
- Registros de control y datos de periféricos (0020h a 005Fh).

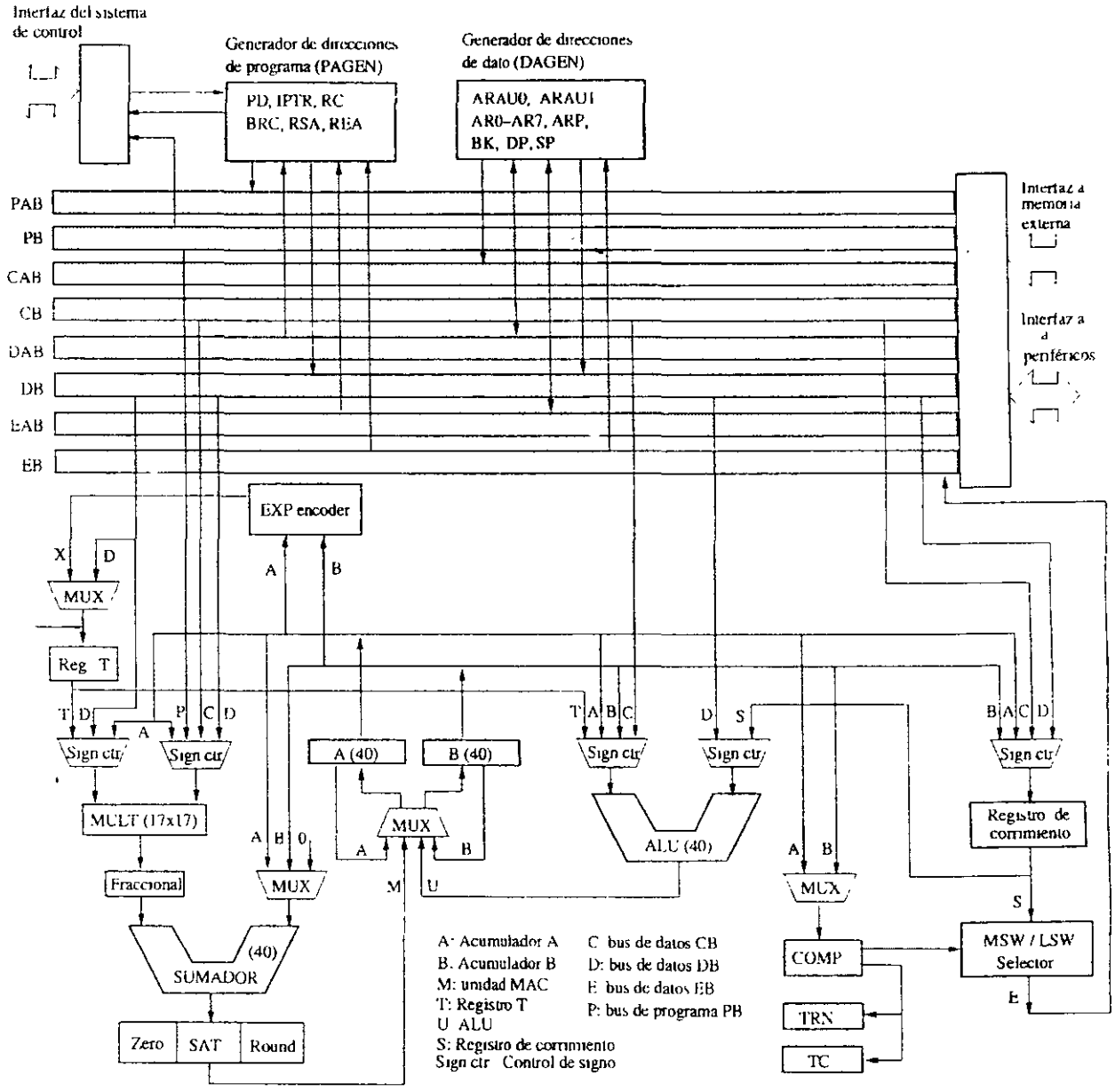


Figura 7.1: Arquitectura del DSP TMS320C54x

Memoria Programa			Memoria Dato	
OVLY = 1	0000h-007Fh	Reservado	0000h-005Fh	Registros Mapeados
	0080h-27FFh	DARAM Interna	0060h-007Fh	DARAM pad
OVLY = 0	0000h-27FFh	Externa	0080h-27FFh	DARAM Interna
	2800h-EFFFh	Externa	2800h	
MP/mc = 0	F000h-F7FFh	Reservado		
	F800h-FF7Fh	ROM Interna		
	FF80h-FFFFh	Vectores de Interrupción		Externa
MP/mc = 1	F000h-FF7Fh	Externo		
	FF80h-FFFFh	Vectores de Interrupción	FFFFh	

Cuadro 7.1: Mapa de memoria del DSP C542 y C543

Unidad central de proceso (CPU)

Esta unidad está compuesta de:

- La unidad aritmético lógica (ALU) de 40 bits.
- Dos acumuladores (A y B) de 40 bits, éstos almacenan la salida de la ALU o del bloque multiplicador/sumador. Cada acumulador está dividido en tres partes:
  - Bits de guarda (bits 39-32, AG o BG)
  - Bits parte alta (bits 31-16, AH o BH)
  - Bits parte baja (bits 15-0, AL o BL)
- Registro para corrimientos de 0-31 a la izquierda (+) 0-16 a la derecha (-). Este registro junto con el codificador de exponente normalizan el acumulador en un ciclo.
- Multiplicador de 17x17 bits en complemento a dos.
- Sumador de 40 bits.
- Unidad de comparación, selección y almacenamiento (CSSU).
- Unidad generadora de dirección de datos.
- Unidad generadora de dirección de programa.

Periféricos internos

- Pines de propósito general (/BIO de entrada y XF de salida) para manejo de dispositivos externos.
- Generador de estados de espera por software.
- Banco de switches lógicos programables.
- Interfaz de puerto huésped (HPI) de 8 bits paralelos que provee una interfaz a un procesador huésped.
- Timer por hardware de 16 bits con cuatro bits preescaladores.
- Generador de reloj que consiste de un oscilador interno y un circuito PLL.
- Puertos serie: puerto serial síncrono, puerto serial buffereado y puerto TDM (dependen de la versión "x" del C54x).

Modos de direccionamiento

- Inmediato: una constante viene en el código de instrucción.
- Absoluto: la instrucción codifica una dirección fija.
- de Acumulador: utiliza el acumulador A para accesar una localidad en memoria programa como dato.
- Directo: similar al C5x, divide la memoria dato en páginas apuntadas por el apuntador de página (DP) o el apuntador de pila (SP).
- Direccionamiento indirecto: similar al C5x, utiliza registros ARi y dos unidades ARAU.
- Registros mapeados.
- De stack: agrega o remueve datos del sistema de stack.

Dirección	Nombre	Descripción
0	IMR	Registro de máscara de interrupción
1	IFR	Registro de banderas de interrupción
2-5	—	Reservado
6	ST0	Registro de estado 0
7	ST1	Registro de estado 1
8	AL	Acumulador A parte baja (bits 15-0)
9	AH	Acumulador A parte alta (bits 31-16)
A	AG	Acumulador A bits de guarda (bits 39-32)
B	BL	Acumulador B parte baja (bits 15-0)
C	BH	Acumulador B parte alta (bits 31-16)
D	BG	Acumulador B bits de guarda (bits 39-32)
E	T	Registro temporal
F	TRN	Registro de transición
10	AR0	Registro auxiliar 0
11	AR1	Registro auxiliar 1
12	AR2	Registro auxiliar 2
13	AR3	Registro auxiliar 3
14	AR4	Registro auxiliar 4
15	AR5	Registro auxiliar 5
16	AR6	Registro auxiliar 6
17	AR7	Registro auxiliar 7
18	SP	Apuntador de pila
19	BK	Registro del tamaño del buffer circular
1A	BRC	Contador de repetición de bloque
1B	RSA	Dirección inicial de repetición de bloque
1C	REA	Dirección final de repetición de bloque
1D	PMST	Registro de estado, para modos del procesador
1E	XPC	Registro de extensión del contador de programa sólo en el C548
1E-1F	—	Reservado

Cuadro 7.2: Registros mapeados del DSP C54x

7.3. La unidad central de proceso CPU

7.3.1. Registros de Estado

De forma similar al DSP C5x, el C54x tiene tres registros de estado mapeados que son: ST0, ST1 y PMST, estos registros contienen algunos estados, ciertas condiciones y modos de operación del DSP. Cada registro contiene conjuntos de bits o bits con diferente función que también son similares al C5x:

Bits	Nombre	Función
15-13	ARP	Apuntador de registros auxiliares
12	TC	Bandera de prueba o control
11	C	Acarreo
10	OVA	Sobreflujo para acumulador A
9	OVB	Sobreflujo para acumulador B
8-0	DP	Apuntador de página en memoria dato

Cuadro 7.3: Registro de estado ST0

7.3.2. Unidad Aritmética Lógica ALU

La unidad ALU efectúa operaciones aritméticas y lógicas a 40 bits casi siempre en un ciclo de instrucción, el resultado de las operaciones se escribe en los acumuladores A o B.

Con fuentes de entradas X (figura 7.2):
- Del registro de corrimiento (operando de 32 o 16 bits de memoria o acumuladores).
- Dato de memoria del bus DB.

entradas Y:
- Valores de los acumuladores A o B.
- Dato de memoria del bus CB.
- Valor en el registro temporal T.

Como se observa de la figura 7.2, dependiendo de las instrucciones la unidad ALU manipula las banderas de estado de entrada OVM, C16 y C, y también afecta las banderas de estado de salida C, OVA/OVB, ZA/ZB y TC .

Bits	Nombre	Descripción
15	BRAF	"1" Repetición de bloque activa
14	CPL	"0" DP, apuntador de página para modo directo "1" SP, para direccionamiento directo relativo al SP
13	XF	Estado del pin externo de salida XF
12	HM	Modo Hold "0" DSP en ejecución interna, buses externos en alta impedancia "1" DSP detiene la ejecución interna
11	INTM	"0" Habilita todas las interrupciones mascarables "1" Deshabilita todas las interrupciones mascarables
10	—	Siempre "0"
9	OVM	Modo de saturación si OVM = "1"
8	SXM	Modo de extensión de signo si SXM = "1"
7	C16	Modo de doble precisión aritmética de la ALU si C16 = "1" si C16 = "0" opera en aritmética dual a 16 bits
6	FRCT	Modo fraccional. Corre un bit a la izquierda de la salida del multiplicador
5	CMPT	Modo de compatibilidad para ARP
4-0	ASM	Modo de corrimiento del Acumulador (de -16 a 15) utilizado por instrucciones STH, STL, ADD, SUB y LD.

Cuadro 7.4: Registro de estado ST1

Bits	Nombre	Descripción
15-7	IPRT	Apuntador de vectores de interrupción a páginas de 128 words
6	MP/mc	Modo Microprocesador / Microcomputadora
5	OVLY	"0" RAM en dato, "1" RAM en dato y programa
4	AVIS	"1" visualiza direcciones internas en los buses externos
3	DROM	"1" ROM mapeada en memoria dato
2	CLKOFF	"1" Deshabilita salida CLKOUT, la fija en nivel alto
1	SMUL	"1" Saturación de la multiplicación antes de una MAC
0	SST	"1" Saturación en el almacenamiento de datos

Cuadro 7.5: Registro de estado PMST

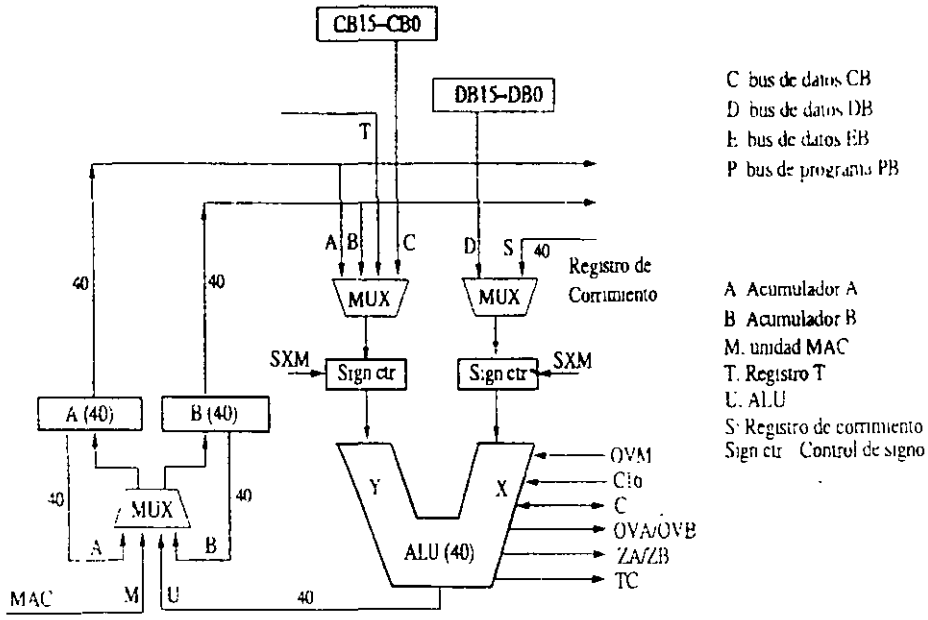


Figura 7.2: Unidad ALU del DSP TMS320C54x

7.3.3. Registro de corrimiento

El registro de corrimiento es utilizado para escalar operaciones tales como:
- Preescalar operandos de entrada de la memoria o un acumulador antes de operar en la unidad ALU.
- Efectuar corrimientos lógicos o aritméticos del valor del acumulador.
- Normalizar el acumulador para implementar operaciones en punto flotante.
- Post-escalamiento de un acumulador antes de almacenarlo en memoria.

Estas operaciones de corrimiento se observan en la figura 7.3.

7.3.4. Unidad Multiplicador/Sumador

Esta unidad consta de un multiplicador en paralelo de 17x17 bits acoplado a un sumador dedicado, lo que conforma la unidad de multiplicación acumulación (MAC) de 40 bits (figura 7.4). El multiplicador puede efectuar multiplicaciones signadas, no signadas y signada/no signada con las limitaciones:
- En multiplicación no signada, cada operando de memoria es asumido como una palabra de 17 bits con signo extendido.

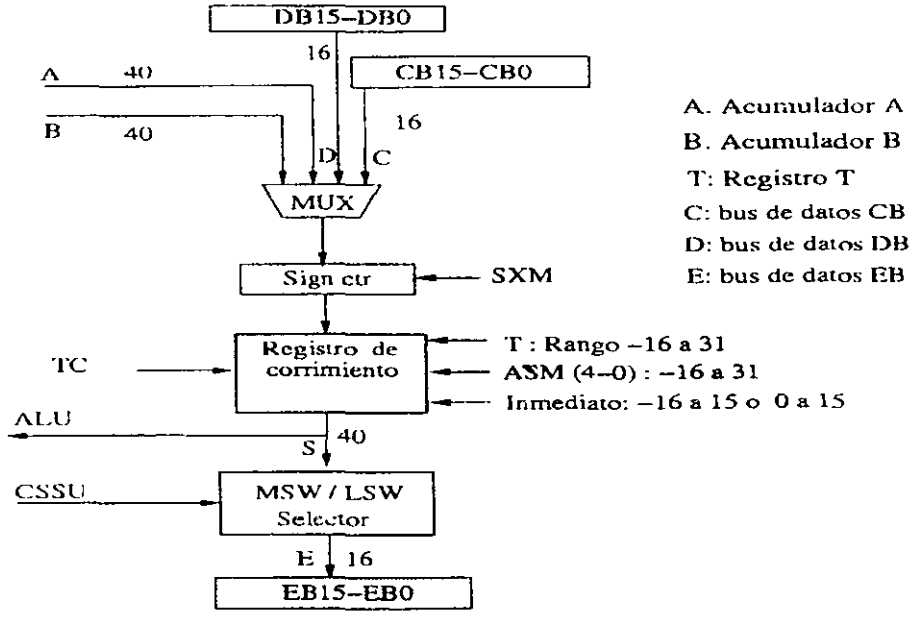


Figura 7.3: Registro de corrimiento del DSP TMS320C54x

- En multiplicación signada, se agrega un 0 al bit 16 (MSB) a cada operando de entrada (operandos de 17 bits).
- En multiplicación signada/no signada, uno de los operandos es de signo extendido y al otro se le agrega un 0 al bit 16 (MSB).

La salida del multiplicador se puede correr un bit a la izquierda para compensar el bit de signo extra generado por el multiplicador, esto se efectúa fijando el bit FRCT en "1".

La unidad MAC contiene a la salida un detector de cero, un modo de redondeo y una lógica de saturación. El redondeo consiste en sumar 2^{15} al resultado (MAC) y limpiar los 16 bits bajos.

La fuente de datos viene seleccionado en las instrucciones, la fuente XM puede ser alguno de los valores (como se ven en la figura 7.4):
- El registro temporal T.
- Datos de memoria por el bus DB.
- El acumulador A parte alta, bits 32-16.

Para la fuente YM:
- Datos de memoria por el bus DB.

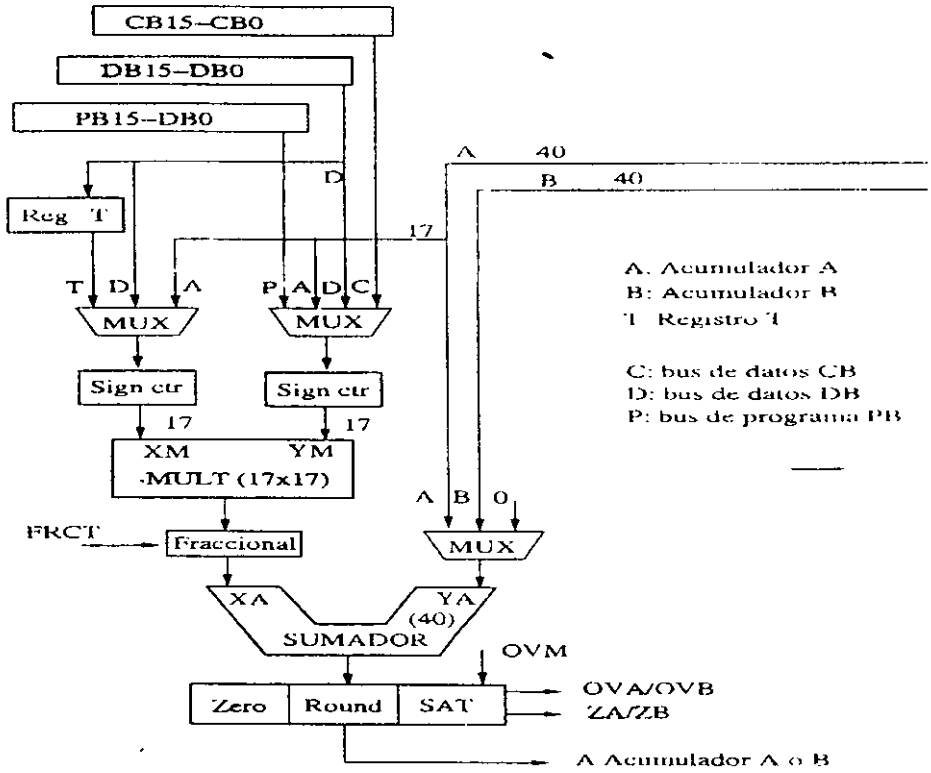


Figura 7.4: Unidad Multiplicador/Sumador del DSP TMS320C54x

- Datos de memoria por el bus CB.
- Operando inmediato de memoria progama por el bus PB.
- El acumulador A parte alta, bits 32-16.

En instrucciones que utilizan al registro T como una entrada, la segunda entrada puede ser obtenida de memoria dato por el bus DB o del acumulador A.
En instrucciones que usan un operando simple de memoria dato, un operando es alimentado al multiplicador por el bus DB, y el segundo operando puede venir del registro T, o como valor inmediato de memoria programa por PB o del acumulador A.
En instrucciones que usan direccionamiento doble de operandos de memoria dato, los buses DB y CB llevan los datos al multiplicador.

7.3.5. Unidad de comparación, selección y almacenamiento

La unidad de comparación, selección y almacenamiento (CSSU) es una unidad de aplicación específica de hardware dedicada a la operación suma/comparación/selección (ACS) del operador Viterbi. La unidad CSSU del C54x soporta varias formas del algoritmo Viterbi en la igualación y decodificación de canal (figura 7.5).
El operador Vieterbi es ejecutado en la ALU, esta función consiste en la doble adición, esta adición es completada en un ciclo de máquina si la ALU está configurada para la suma dual fijando el bit C16 en el registro de estado ST1, las palabras de 32 bits se convierten en aritméticas de 16 bits. Para ejecutar este tipo de operación el C54x tiene instrucciones especiales.

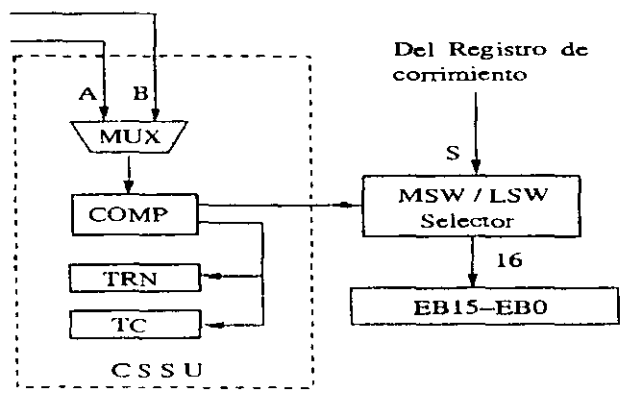


Figura 7.5: Unidad CSSU del DSP TMS320C54x

7.3.6. Codificador de Exponente

Es una aplicación específica de hardware dedicada a ejecutar la instrucción EXP en un ciclo de instrucción. Esta instrucción extrae el valor del exponente del acumulador y le resta ocho, este valor es almacenado en el registro T en complemento a dos con un rango de -8 a +31. Este número corresponde a la cantidad de corrimientos (+ a la izquierda, - a derecha) requeridos en el acumulador (40 bits) para eliminar los bits menos significativos a excepción del bit de signo, es decir, que extrae el exponente para representar al ACC en formato Qi máximo para representar su mantisa (Q31). Las instrucciones EXP y NORM son utilizadas para codificar el exponente y normalizar el contenido del acumulador eficientemente. La instrucción NORM extrae la mantisa del ACC, haciendo los corrimientos de acuerdo al exponente almacenado en el registro T.

7.4. Modos de direccionamiento

Inmediato

En este modo una constante específica es el operando de la instrucción, y la constante misma se codifica en un campo del código de la instrucción. Para este modo el símblo # precede a la constante o un símbolo. Una constante corta puede ser de 3,5,8 y 9 bits y una constante larga es 16 de bits, es decir, que en este último caso el código de instrucción puede ser de dos words.

Absoluto

En este modo la instrucción codifica una dirección fija en el mapa de memoria. Existen cuatro posibilidades:

- Direccionamiento a memoria dato (dmad), direcciona un valor específico en memoria dato.
- Direccionamiento a memoria programa (pmad), direcciona un valor específico en memoria dato.
- Direccionamiento a puerto I/O, direcciona un valor específico en un puerto, utiliza instrucciones PORTR y PORTW
- *(lk), utiliza un valor para especificar una dirección en el espacio de dato. La sintáxis *(lk) utiliza un símbolo o un número para indicar una dirección en el espacio de dato. Este direccionamiento no se puede utilizar con instrucciones RPT y RPTZ.

De Acumulador

Utiliza el acumulador A para accesar una localidad en memoria programa como dato. Específicamente se utiliza en dos instrucciones: READA Smem, que transfiere una palabra de memoria programa especificada por el ACC A parte baja a una localidad de memoria dato especificada por Smem (memoria dato de simple acceso) y WRITA Smem, que transfiere un word de memoria dato especificada por Smem a una localización de memoria programa especificada por el ACC A.

Direccionamiento directo

Este modo es similar al C5x, divide la memoria dato en páginas apuntadas por el apuntador de página (DP) o el apuntador de pila (SP). La instrucción contiene el offset o desplazamiento de 7 bits de la dirección del dato a transferir. Con este modo de direccionamiento se pueden accesar a 128 localidades en memoria RAM sin cambiar DP o SP.

La selección de DP o SP se hace:

— Si el bit de modo compile (CPL) del registro ST1 es cero, se selecciona el apuntador de página DP, en este caso la memoria dato se divide en 512 páginas de 128 localidades cada una. Una dirección efectiva de 16 bits se forma de los 9 bits de la página con los 7 bits del offset.

— Si CPL=1, se selecciona el contenido del stack pointer (SP) como dirección de referencia. En este caso la dirección efectiva es la suma de SP más el offset, donde el SP apunta a una dirección en memoria, se pueden accesar a 128 localidades tomando como dirección base SP.

Direccionamiento indirecto

Es similar al C5x, cualquier dirección de 16 bits contenida en un registro ARi puede ser accesada. Este modo permite accesar dos datos en un ciclo de instrucción. Diferente al C5x, en el C54x se indica directamente el ARi a utilizar y su modificador.

Este modo utiliza los registros auxiliares ARi (AR0-AR7) y las unidades ARAU1 y ARAU2 que efectúan aritmética no signada a 16 bits sobre los registros ARi. Con los registros auxiliares pueden efectuarse las operaciones:
- Cargarse con un valor inmediato utilizando instrucción STM.
- Cargarse por el bus de datos.
- Modificarse utilizando modificadores en direccionamiento indirecto.
- Modificarse por instrucción MAR (modifica registro auxiliar).

- Utilizarse como contador de ciclo utilizando instrucción BANZ[D].

Operando	Función	Descripción
*ARi	dir=ARi	ARi contiene la dirección de memoria dato
*ARi-	dir=ARi ARi=ARi-1	Post-decrementa ARi después de accesar el dato
*ARi+	dir=ARi ARi=ARi+1	Post-incrementa ARi después de accesar el dato
*+ARi	dir=ARi ARi=ARi+1	incrementa ARi antes de accesar dato
*ARi-0B	dir=ARi ARi=B(ARi-AR0)	después de accesar el dato, al ARi se le resta AR0 en acarreo inverso
*ARi-0	dir=ARi ARi=ARi-AR0	después de accesar el dato, al ARi se le resta AR0
*ARi+0	dir=ARi ARi=ARi+AR0	después de accesar el dato, al ARi se le suma AR0
*ARi+0B	dir=ARi ARi=B(ARi+AR0)	después de accesar el dato, a ARi se le suma AR0 en acarreo inverso
*ARi-%	dir=ARi ARi=circ(ARi-1)	Post-decrementa ARi después de accesar el dato en buffer circular
*ARi-0%	dir=ARi ARi=circ(ARi-AR0)	Post-decrementa ARi en AR0 después de accesar el dato en buffer circular
*ARi+%	dir=ARi ARi=circ(ARi+1)	Post-incrementa ARi después de accesar el dato en buffer circular
*ARi+0%	dir=ARi ARi=circ(ARi+AR0)	Post-incrementa ARi en AR0 después de accesar el dato en buffer circular
*ARi(lk)	ARi=ARi+lk dir=ARi	la dirección del dato es ARi más una constante de 16 bits, ARi no cambia
*+ARi(lk)	dir=ARi+lk ARi=ARi+lk	la dirección del dato es ARi más una constante de 16 bits, ARi se actualiza
*+ARi(lk)%	dir=circ(ARi+lk) ARi=circ(ARi+lk)	la dirección del dato es ARi más una constante de 16 bits en direc. circular, ARi se actualiza
*(lk)	dir=lk	Una constante de 16 bits es utilizada como dirección absoluta

Cuadro 7.6: Modificadores de direccionamiento indirecto

El C54x utiliza como desplazamiento el registro AR0, este registro también se carga con el valor de N/2 para direccionamiento en acarreo inverso.

El **direccionamiento circular** es un modificador del direccionamiento indirecto, en este caso el registro mapeado BK contiene el tamaño del buffer circular. Todo buffer circular de tamaño R debe de empezar en límites (fronteras) de N bits, es decir, los N bits LSB de la dirección base del buffer circular deben ser cero, donde N es un número entero menor que satisface $2^N > R$. El valor de R es cargado en BK.

Para el uso de buffer circular tomar en cuenta las reglas:
- Poner la dirección (parte baja) del buffer circular en una frontera 2^N, donde 2^N es mayor que el buffer circular.
- Utilizar un paso menor o igual al buffer circular.
- La primera vez que se utiliza el buffer circular, el ARi a usar debe estar dentro del buffer.

El algoritmo de direccionamiento de buffer circular es:

```
if ( 0 <= ARi + paso < BK )
  ARi = ARi + paso
  elseif ( ARi + paso >= BK )
   ARi = ARi + paso - BK
   elseif (ARi + paso < 0)
     ARi = ARi + paso + BK
```

Direccionamiento de Registros mapeados

Es similar al C5x, este modo de direccionamiento se utiliza para modificar los registros mapeados (página 0) sin modificar el contenido de DP o SP. Este modo utiliza los siete bits menos significativos de una dirección para seleccionar un dato en memoria. En direccionamiento indirecto utiliza nada más los siete bits LSB del registro ARi para acceder a la localidad en memoria.

Para el C54x se tiene un ambiente de depuración más iteractivo que el C50, este ambiente se ejecuta en Windows para versiones superiores a la 3.11, en la figura 7.6 se puede observar este ambiente que también presenta la facilidad de graficar bloques de memoria dato y el espectro discreto. Además se tiene una amplia ayuda en línea.

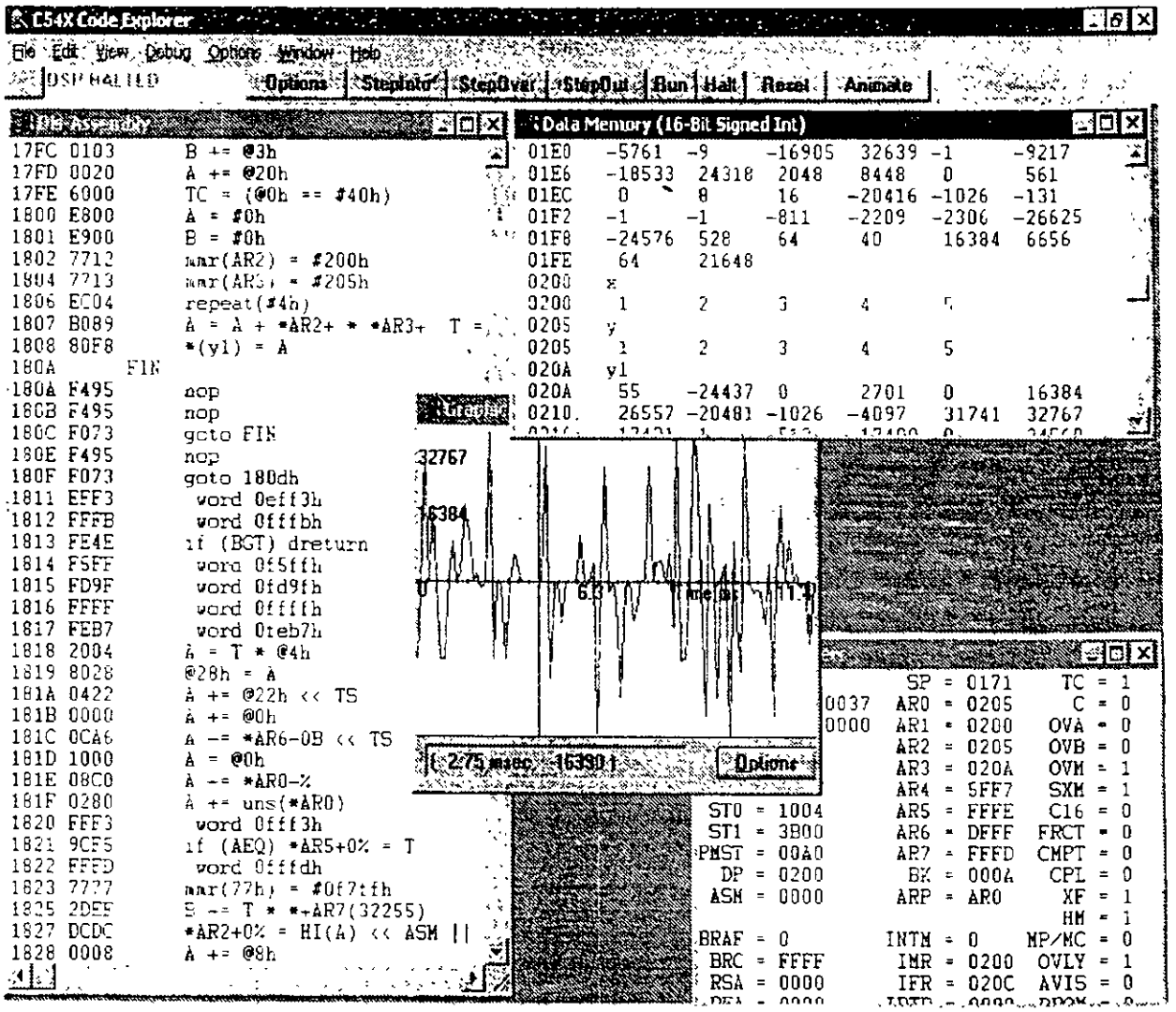


Figura 7.6: Ambiente de depuración del DSP TMS320C54x

7.5. Ejemplos de programas

En esta parte se agregan algunos ejemplos para ilustrar la operación del C54x, los programas desarrollados se han realizado en lenguaje algebraico combinado con instrucciones de ensamblador. Los primeros ejemplos son similares a los programas iniciales elaborados para el C5x, los siguientes son un poco más complejos. Estos ejemplos pueden ser útiles como una breve introducción a la programación y utilización del C54x. Se hace notar que una vez que el lector comprenda el funcionamiento del C50 y haya elaborado sus propios programas, el entendimiento y la utilización del C54x le será muy fácil.

7.5.1. Suma utilizando direccionamiento inmediato

```
*
*       Suma cinco constantes en modo inmediato
*       ©larry
*       !!!!!!    C54x
        .title   "Suma cinco constantes'' ; Título
        .mmregs                    ; Reconoce abreviaturas de
                                   ; registros mapeados y nombres de bits

        .width  80                 ; Ancho de la página editada
        .length 55                 ; Número de líneas de la página editada
        .setsect ".text",0x1800,0  ; Ubica en memoria programa el código
        .setsect ".data",0x0200,1  ; Ubica en memoria dato inicio de datos

        .sect ".data"              ; datos
D1 .set 1
D2 .set 2
D3 .set 3
D4 .set 4
D5 .set 5
total   .word 0
*
        .sect".text"               ; Código
        A = #0
        A = A+#D1
        A = A+#D2
        A = A+#D3
        A = A+#D4
        A = A+#D5
        ©total = A                 ; Salva la suma en total
```

```
FIN     NOP
        NOP
        GOTO FIN
          .end
```

7.5.2. Suma utilizando direccionamiento indirecto

```
*
*       Suma cinco números en modo indirecto
*       Salva resultado en modo directo
*       @larry
*       !!!!!     c54x
          .mmregs
          .width   80
          .length  55
          .setsect ".text",0x1800,0
          .setsect ".data",0x0200,1

          .sect ".data"                ; Datos
D1      .word 1
D2  .word 2
D3  .word 3
D4  .word 4
D5  .word 5
total     .word 0

          .sect".text"                 ; Código

          DP = #D1
          A = #0
          AR0 = #D1
          repeat(#4)
            A = A + *AR0+
          @total = A                   ; La suma se almacena en loc. total

FIN       NOP
          GOTO FIN
          .end
```

7.5.3. Suma utilizando buffer circular

```
*
*  suma de 5 números utilizando repeat(#num)
*  y direccionamiento indirecto utilizando
*  buffer circular en el C54x
*  @larry  !!!!!!!

        .mmregs
        .width   80
        .length  55
        .setsect ".text",0x1800,0
        .setsect ".data",0x0200,1

        .sect ".data"
x .word 1,2,3,4,5,5,4,3,2,1
y       .word 0         ; resultado
N1      .set 10         ; tamaño del buffer circular
N2      .set 9          ; diez sumas con buffer circular
*
        .sect".text"
        A = #0          ; inicialización de A a cero
        BK = #N1        ; inicialización del tamaño del buffer circular
        AR1 = #x        ; dirección donde comienza el buffer circular
        repeat(#N2)     ; repetición de la siguiente instrucción N2 +1
         ; veces
            A = A + *AR1+%    ; AR1 en uso incrementandose en 1 en
                              ; forma de buffer circular
        @y = A          ; se asigna el valor de A en la dirección
                        ; que apunta y

FIN     NOP
        NOP
        GOTO FIN
        .end
```

7.5.4. Multiplicación de dos vectores

```
*
*       Multiplicación de dos vectores con suma de
*       productos utilizando dos registros auxiliares
```

```
*       C54x
*       @larry  !!!!!

        .width   80
        .length  55
        .mmregs
        .setsect ".text",0x1800,0
        .setsect ".data",0x0200,1

        .sect ".data"           ; datos
x .word 1,2,3,4,5
y       .word 1,2,3,4,5
y1      .word 0
N2      .set 4

        .sect ".text"           ; Código

        A = #0                  ; inicialización de A a cero
        B = #0
        AR2 = #x                ; AR2 apunta a valores de x
        AR3 = #y                ; Ar3 apunta a valores de y
        repeat(#N2)             ; repetición siguiente instrucción N2 +1
                                ; veces
           A = A + *AR2+ * *AR3+ ; suma de productos

        *(y1) = A               ; Salva el valor de AL en la dirección
                                ; que apunta 'y'
*
FIN     NOP
        NOP
        GOTO FIN
        .end
```

7.5.5. Multiplicación de una matriz por un vector

```
*
*
*  @larry
*
         .title   "Multiplicación de Matriz por vector"
         .mmregs
```

```
         .width   80
         .length  55
*
*      Ubica segmentos:  Código, datos y vectores de interrupción
*
     .setsect ".text",0x1800,0    ; Sección de código, mapa 0
     .setsect ".data",0x0200,1    ; Sección de datos, mapa 1
     .setsect "vectors",0x0180,0 ; Vectores de Interrupción
*
*    Sección de Datos y variables
*
        .sect ".data"
*
XA      .word    1,2,3,4,5,6,7,8,9     ; Matriz A
V       .word    1,2,1                 ; Vector V
Y       .word    0,0,0,0,0,0,0,0,0,0   ; Matriz C=A.V
N1      .set     2
N       .set     2
*
        .sect".text"

        dp = #XA    ; PAGINA DE DATOS DE SUMA

        AR1=#XA     ; AR1 apunta a vector x
        AR2=#Y      ; Ar2 salida de datos AxV

        brc = #N1
         blockrepeat(FIN_B)
           a = #0               ; acc A = 0
           repeat(#N)
           macp(*AR1+,V,A)

           *AR2+ = A

FIN_B   nop

FIN     nop
        goto  FIN
        .end
```

7.5.6. Multiplicación de una matriz por una matriz

```
*
*  @larry
*                    10_feb_002
*  !!!!!

          .title   "Multiplicación de Matrices"
          .mmregs
          .width   90
          .length  55
*
*       Ubica segmentos:  Código, datos y vectores de interrupción
*
          .setsect ".text",0x1800,0
          .setsect ".data",0x0200,1
          .setsect "vectors",0x0180,0
*
*        Sección de Datos y variables
*
.sect ".data"

X         .word 1,2,3,4,5    ; Matriz X(5x5), puede ser de NxN
X1        .word 6,7,8,9,10
X2        .word 1,2,3,4,5
X3        .word 1,2,3,2,1
X4        .word 2,2,2,2,2
V         .word 1,1,1,1,1    ; Matriz V(5x5), puede ser de NxN
V1        .word 2,2,2,2,2
V2        .word 3,3,3,3,3
V3        .word 4,4,4,4,4
V4        .word 5,5,5,5,5
Y         .word 0,0,0,0,0,0,0,0,0,0,0 ; Matriz Y=AxV
Y1        .word 0,0,0,0,0,0,0,0,0,0,0 ; Matriz Y=AxV
Y2        .word 0,0,0,0,0             ; Matriz Y=AxV

TEM       .word 0 ; Salva AR1
CON       .word 0 ; Contador
*            Aquí se puede cambiar el orden de las matrices NxN
CONTOT    .word 5 ; Orden N
N5        .set  5 ; Orden N
```

```
N1      .set  4 ; Orden N-1
*
        .sect".text"
        dp = #TEM       ; Página de datos
        sp = #OFFAh     ; Ubica el Stack Pointer
        AR1 = #X        ; AR1 apunta a vector
        AR2 = #V
        AR3 = #Y
        A = MMR(AR1) ; A = contenido de lo que apunta AR1
        @TEM = A ; Salva dir. inicio de AR1

MISMAL
        CALL MULVV
        B = @CON
        B = B +  #1
        @CON = B
        B = B - #N5         ; Ve cuantos vectores ha multiplicado
        if (BEQ) goto OTRAL ; Va a otra linea de X, V regresa
          A = @TEM      ; Lo contrario
          MMR(AR1) = A      ; AR1 regresa a inicio línea
          goto MISMAL
OTRAL   AR2 = #V      ; AR2 regresa a inicio matriz V
        B =#0
        @CON = B            ; Reincia contador Vec x Vec = 0
        B = @CONTOT
        B = B - #1
        @CONTOT = B
        if (BEQ) goto FIN   ;--->
        A = @TEM
        A = A + #N5         ; Actualiza TEMP para sig. línea
        @TEM = A
        goto MISMAL     ; ---->

FIN     NOP
        NOP
        goto FIN

**================================================
MULVV   NOP ; subrutina Vector x Vector
        AR5 = #N1       ; N-1
        A = #0
```

```
SIGUE   T = *AR1+
        A = A + T**AR2+
        if (*AR5- != 0) goto SIGUE
        *AR3+ = A
        RETURN
*
       .end
```

7.5.7. Ordenamiento de un vector en forma descendente

```
*
*      Ordena 25 números: en orden descendente
*      Por el método de la burbuja
*      @ larry
*      10_feb_002
*     !! !!!

    .title   "Ordena números de mayor a menor"
          .mmregs
          .width   90
          .length  55
*
*    Ubica segmentos: Código, datos y vectorer de interrupción
*
          .setsect ".text",0x1800,0
          .setsect ".data",0x0200,1
          .setsect "vectors",0x0180,0
*
*        Sección de Datos y variables
*
.sect ".data"

MA       .word 1,2,3,4,15,6,7,8,9,10
MA1      .word 11,20,13,24,22,19,23,12,17
MA2      .word 21,14,18,25,16
*
*
N25      .set  25                ; N
N24      .set  24                ; N-1
```

```
N23     .set  23              ; N-2
*
              .sect".text"

              sp = #0FFAh       ; Ubica el Stack Pointer
              AR3 = #N23        ; Para ciclo externo

CICLO_N       AR1 = #MA         ; AR1 apunta a inicio de datos
              BRC = #23         ; repeticin de bloque
              BLOCKREPEAT(FIN_B)
              A = *AR1+         ; A = MA(i)
              B = *AR1          ; B = MA(i+1)
              B = B - A         ; MA(i+1) - MA(i)
              if(BGT) goto CAMBIA
*
              goto FIN_B
CAMBIA        B = *AR1-         ; mar(*AR1-)
              *AR1+ = B         ; Intercambia
              *AR1 = A
FIN_B         NOP
*
              if (*AR3- != 0) goto CICLO_N   ; Ciclo externo
*
FIN           NOP
              NOP
              goto FIN
```

7.5.8. División de dos números

Este programa está basado en un algoritmo rápido de obtención de división y raíces cuadradas [13] que funciona en forma similar a un convertidor de aproximaciones sucesivas.

```
*
*     División = Numerador (32b) / Denominador (16 b)
*     El Numerdor y denominador son positivos
*     Numerador > Denominador
*       @ larry
*       23_feb_002
*     div_540.asm
```

```
*       !! !!!
          .title   "División"
          .mmregs
          .width   90
          .length  55
*
*       Ubica segmentos:  Código, datos y vectores de interrupción
*
          .setsect ".text",0x1800,0
          .setsect ".data",0x0200,1
          .setsect "vectors",0x0180,0
*
*         Seccin de Datos y variables
*
.sect ".data"
NUMH      .word 0064h       ; Numerador H
NUML      .word 0AB90h      ; Numerador L
*DEN       .word 095Ah      ; Denominador (q8)
DEN       .word 055Ch       ; Denominador (q8)
TEM       .word 8000h       ; Bit de prueba
COC       .word 0           ; Cociente de división
MASK      .word 07FFFh      ; Máscara para operación AND
MASKH     .word 07FFFh      ; Parte alta de la máscara
N         .set  15          ; N-1 pruebas
*
          .sect".text"

          SXM = #0          ; Suprime extensión de signo
          DP = #TEM         ; P gina de datos
          SP = #0FFAh       ; Ubica el Stack Pointer
          BRC = #N          ; Para repetición de bloque

          blockrepeat(FIN_B)
          A = @TEM          ;
          A = A + @COC      ; Pone bit de prueba
          @COC = A          ; COC = COC + TEM
          T = A             ;
          B = T*uns(@DEN)   ; B = COC*DEN (no signado)
          A = @NUMH << 16   ;
```

```
        A = A + @NUML     ;
        B = B - A         ; B = C*DEN - NUM
         if (BLT) goto MUEVE_M ;--->
           A = @COC
           A = A & @MASK        ; Borra bit de prueba
           @COC = A
*
MUEVE_M    A = @MASKH << 16
           A = A + @MASK
           A = A >> 1
           @MASK = A
*
           A = @TEM >> 1
FIN_B      @TEM = A

FIN       NOP
          NOP
          goto FIN
*
         .end
```

Bibliografía

[1] ALCÁNTARA S. R. & ESCOBAR S. L. *Dynamic Range and Scaling Evaluation in Adaptive Filtering Algorithms for DSP fixed-point implementation.* International Symposium on Information Theory and its Applications (ISITA98). México, octuber 14-16, 1998.

[2] ALCÁNTARA S. R. & ESCOBAR S. L. *A comparative TMS DSP Fixed-Point Implementation of FRLS Adaptive Filtering Algorithms Family.* The International Conference on Signal Processing Applications and Technology (ICSPAT). November 1-4, 1999. Orlando, Florida USA.

[3] ESCOBAR S. L. & RODRÍGUEZ S. R. *Diseño y construcción de Arquitectura para Procesamiento Digital de Imágenes.* F.I. UNAM. 1992. Tesis de Licenciatura.

[4] ESCOBAR S. L. *Algoritmos de Filtrado Adaptable: Implementación, Evaluación, Comparación y Aplicaciones en Telecomunicaciones.* F.I. UNAM. 1997. Tesis de Maestría.

[5] ESCOBAR S. L. & PASALAGUAS A. *Síntesis de voz en tiempo real utilizando una arquitectura de punto flotante.* Congreso de Electrónica. Instituto Tecnológico de Chihuahua. Chihuahua, México, octubre de 1997.

[6] ESCOBAR S. L. *Arquitecturas de DSPs, familia TMS320 y el TMS20C50.* Facultad de Ingeniería, UNAM, México D. F., agosto del 2000.

[7] ESCOBAR S. L. & ALCÁNTARA S. R. *Fixed Point Arithmetic using Digital Signal Procesors.* International Conference on Telecommunications (ICT200), Acapulco, México 22-25 de mayo del 2000.

[8] ESCOBAR S. L. *Manual del laboratorio de Procesamiento Digital de Señales.* Facultad de Ingeniería, UNAM, México D. F., abril del 2000.

[9] HAMMING R. W. *Digital filters.* Prentice Hall, New Jersey 1983.

[10] LANDEROS A. S., PSENICKA B. & KARPF M. *Design and Applications of State-Space Digital Filter.* Revista: Ingeniería, Investigación y Tecnología, Vol. II No. 4., FI, UNAM, oct-dic. 2001, pg.149.

[11] MARTÍNEZ J. & ESCOBAR S. L. & RODRÍGUEZ S. R. *Arquitectura para Procesamiento digital de Imágenes. Congreso de Electrónica.* Instituto Tecnológico de Chihuahua, Chihuahua México, octubre de 1993.

[12] OPPENHEIM A. V. & SCHAFER R. W. *Digital Signal Processing.* Prentice-Hall, Englewood Cliffs, New Jersey 1975.

[13] PRADO J. & ALCÁNTARA S. R. *A Fast Square-Rooting Algorithm Using a Digital Signal Processor.* Proceedings of IEEE, USA, Vol. 75, No.2, pg. 262-264, Feb. 1987.

[14] PROAKIS J.G. & MANOLAKIS. *Digital Signal Processing.* Macmillan. Singapure 1992.

[15] PSENICKA B. *Apuntes de Procesamiento Digital de Señales.* Facultad de Ingeniería, UNAM, México D.F., 1996.

[16] PSENICKA B. Y ESCOBAR S. L *Procesamiento Digital de Señales, segunda parte, Micdrocontroladores y realización de los filtros digitales con TMS320Cxx.* Facultad de Ingeniería, UNAM, México D.F., julio de 1998.

[17] TEXAS INSTRUMENTS *Texas Instruments. Digital Signal Processing Applications with the TMS320 Family, Theory, Algorithms, and implementations.* Vol.1,2 y 3. USA 1990.

[18] TI. *TMS30C5x, User's Guide.* USA 1993.

[19] TI. *TMS320C5x DSP starter Kit User's Guide.* USA 1994

[20] TI. *TMS320C54x Reference Set, Vols 1: CPU and Peripherals, 2: Mnemonic Instruction Set, 3: Algebraic Instruction Set & 4: Applications Guide.* USA 1997.

Esta obra se terminó de imprimir
en junio de 2002
en el taller de imprenta del
Departamento de Publicaciones
de la Facultad de Ingeniería,
Ciudad Universitaria, México, D.F.
C.P. 04510

Secretaría de Servicios Académicos

El tiraje consta de 300 ejemplares
más sobrantes de reposición.

FACULTAD DE INGENIERÍA UNAM
DIVISIÓN DE EDUCACIÓN CONTINUA

DIPLOMADO EN MICROCONTROLADORES (SISTEMAS EMBEBIDOS)

CA 209 MÓDULO III PROCESADORES DIGITALES DE SEÑALES

TEMA:

EL PROCESAMIENTO DIGITAL DE SEÑALES Y SUS APLICACIONES

AUTOR: M. I. LARRY ESCOBAR
INSTRUCTOR: ING. MIGUEL ANGEL MOLERO ARMENTA

DEL 11 AL 15 DE OCTUBRE DE 2004
PALACIO DE MINERÍA

TEMARIO

EL PROCESAMIENTO DIGITAL DE SEÑALES Y SUS APLICACIONES

El ser humano en sus actividades diarias realiza una gran variedad de tareas como reconocer, clasificar, comparar, agrupar, abstraer, comprimir, almacenar, cuantificar, inferir, predecir etc, todo este tipo de tareas las realiza sobre información que percibe del mundo real representada en imágenes, sonidos, fenómenos naturales, ondas electromagnéticas y acústicas que en general pueden ser caracterizadas como señales.

En la actualidad 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) siendo una alternativa real para la solución de problemas de medición, control, filtrado, análisis espectral, 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 de muy alta escala de integración (VLSI). El estudio y entendimiento de cada una de estas diferentes áreas, forman un conjunto de principios fundamentales que constituyen la disciplina del Procesamiento Digital de Señales.

Los principios fundamentales del PDS están dados por la teoría de señales y sistemas, 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 del análisis o estimación contenido de la información. El análisis de señales es el proceso de definir y cuantificar las características de una señal para una aplicación.

El PDS se puede concebir como un conjunto de operaciones básicas aplicadas sobre una señal de entrada para obtener una señal de salida, formalmente estas operaciones se describen como transformaciones matemáticas de una señal a otra por una regla predefinida.

Para su estudio el PDS se puede dividir en:

1) **La parte conceptual**, que involucra los fundamentos del PDS y la generación de nuevos algoritmos.

2) **La parte algoritmica**, que trata con la interpretación y la utilización de los algoritmos existentes, es decir, la proposición de soluciones a problemas concretos.

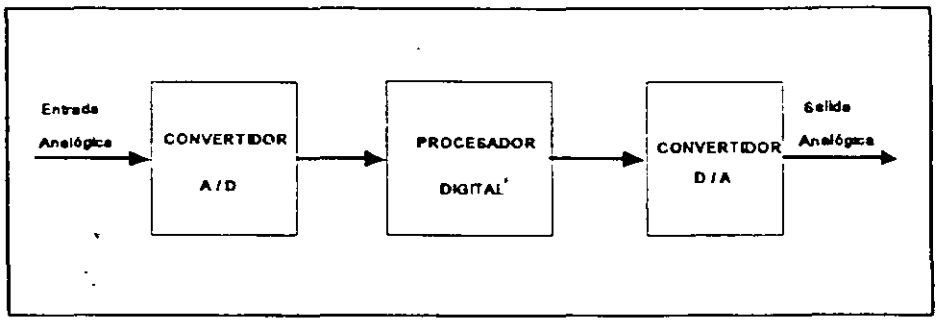
3) **La implantación de los algoritmos.**

- En esta parte se puede presentar alternativas de solución como la simulación de los algoritmos a través de lenguajes de alto nivel, la validación y verificación de los algoritmos.

- El traslado de los algoritmos a arquitecturas de procesamiento específicas, es decir la utilización de un lenguaje ensamblador para la instalación de una aplicación concreta. Esta etapa involucra la verificación algorítmica-software-hardware de la solución al problema, lo que constituiría un prototipo final.

ELEMENTOS BASICOS DE UN SISTEMA DE PROCESAMIENTO DIGITAL DE SEÑALES:

Para efectuar el PDS es necesario una interface entre la señal analógica y un procesador digital, y a la salida la señal procesada digitalmente debe convertirse a una señal analógica. El Procesador digital puede ser desde una computadora digital hasta un pequeño microprocesador cuyo programa efectúa las operaciones deseadas sobre la señal de entrada.



Sistema Básico de PDS

Con el advenimiento de los Procesadores de Señales Digitales (DSP), en la actualidad en la mayoría de aplicaciones orientadas a señales, el bloque del procesador digital está constituido por un DSP, por lo que en este trabajo se pretende conocer principalmente la arquitectura del DSP TMS320C50.

VENTAJAS DE PROCESAMIENTO DIGITAL SOBRE EL ANALOGICO:

- La programación de un sistema digital permite más flexibilidad en la reconfiguración del sistema, simplemente un cambio del programa permite cambiar el algoritmo o la aplicación. En un sistema analógico su reconfiguración implica un rediseño del hardware.
- El PDS provee un mejor control de los requerimientos de precisión. Debido a la tolerancia de los circuitos analógicos es muy difícil el control de precisión.
- Una señal digital es muy fácil almacenarla sin deterioro o pérdida de fidelidad, las señales almacenadas son fácilmente transportables para ser analizadas y/o procesadas remotamente.
- Una limitación del PDS es la velocidad de operación de un convertidor A/D. Las señales con un gran ancho de banda requieren velocidades de muestreo y conversión muy altas, sin embargo, con el avance de la tecnología actual esta limitición tiende a desaparecer.

CARACTERISTICAS DEL PROCESAMIENTO DIGITAL DE SEÑALES

El PDS cubre una gran cantidad de aplicaciones, las cuales tienen algunas características en común:

- Algoritmos matemáticos intensivos.
- Operaciones en tiempo real.
- Implementación de muestreo de datos.
- Sistemas flexibles.
- Movimiento de bloques de datos.
- Operaciones de multiplicación acumulación rápida.
- Direccionamiento eficiente de tablas.
- Operaciones en paralelo.
- Tamaño adecuado de palabra.
- Memoria RAM interna de alta velocidad.

APLICACIONES DE LOS DSP's[1]

Los DSP han sido exitosamente aplicados en voz y audio en productos de tiempo real tales como modems, canceladores de eco, codificación de voz y audio, síntesis de voz, reconocimiento de voz, digitalización de audio, síntesis de música, procesamiento de música y múltiples ampliaciones en telecomunicaciones. Además se han utilizado en aplicaciones en tiempo real tales como gráficos, imágenes médicas, simulación y procesamiento de imágenes.

[1] DSP Digital Signal Processig

Algunas aplicaciones y funciones que realizan los DSPs

Filtros digitales	**Procesamiento de señales**
De respuesta finita al impulso (FIR) De respuesta infinita al impulso (IIR) Lattice-Lader Correlación Transformada coseno Windowing Filtrado adaptable Igualación de canal Eliminación de ruido Cancelación de eco	Compresión Expansión Energía Procesamiento homomórfico Ley μ Ley A Conversión
Procesamiento de datos	**Modulación**
Encriptado Scrambling Codificación Decodificación	Amplitud Frecuencia Fase QAM
Procesamiento numérico	**Análisis Espectral**
Operaciones matriciales Funciones trascendentales Funciones no lineales generación de números aleatorios Aproximaciones numéricas	Transformada rápida de Fourier (FFT) Transformada discreta de Fourier (DFT) Modelo moving average (MA) Modelo autorregresivo (AR)

INTRODUCCION A LOS PROCESADORES DE SEÑALES DIGITALES

Inicialmente el procesamiento digital de señales se realizaba en grandes computadoras (maiframes) debido a sus velocidades de procesamiento. Posteriormente el procesamiento se hizo en arreglos de procesadores convirtiéndose en una herramienta flexible y veloz.

Los procesadores de señales digitales (DSP) son microprocesadores especializados para efectuar operaciones numéricas en tiempo real, en aplicaciones donde se requieren numerosas cálculos aritméticos en un tiempo limitado. Por sus aplicaciones específicas los DSPs tienen arquitecturas que son significante diferentes a los microprocesadores tradicionales.

En su arquitectura los DSPs incorporan unidades de multiplicación acumulación donde el tiempo de un ciclo de instrucción puede se igual al tiempo de un ciclo de la aritmética en hardware. En la actualidad existen DSPs que efectúan multiplicaciones de 32 bits en punto flotante en 60 u 80 ns, esto es de 25 a 30 millones de operaciones de punto flotante por segundo (MFLOPS). Los actuales DSPs utilizan extensivamente operaciones en pipeline, memorias independientes y unidades trabajando en paralelo.

Los DSPs lograron alcanzar desempeños que sólo se habían logrado con arreglos de procesadores. En 1982 la compañía Texas Instruments (TI) lanza al mercado su primer DSP el TMS32010, posteriormente introdujo varias generaciones de esta familia estando en la actualidad en la cuarta generación con el TMS320C40 y la más avanzada generación TMS320C80 con un hardware de alto paralelismo. En 1996-98 se dió a conocer la familia TMS320C6xx con arquitecturas de ocho unidades funcionales trabajando en paralelo, que pueden ejecutar ocho instrucciones por ciclo y alcanzar desempeños de hasta 5,000 MIPS (millones de instrucciones por segundo) y hasta un GFLOP. Estos últimos procesadores trabajan con la tecnología de palabra de instrucción muy larga (VLIW).

ALGUNAS CARACTERISTICAS GENERALES DE LOS DSPs

Los DSPs disponen de una arquitectura y un conjunto de instrucciones orientadas al Procesamiento Digital de Señales. La arquitectura de un DSP normalmente contiene buses separados para el direccionamiento simultáneo de dos operandos, un multiplicador, corrimientos, una unidad de direccionamiento, puertos seriales y temporizadores. Entre las instrucciones de grandes posibilidades están el movimiento de bloques, multiplicaciones en un ciclo de instrucción, multiplicación acumulación para la realización de operaciones de convolución.

Evolución de los DSPs de Texas Instruments

Año	DSP	Instrucción MAC (ns)	Bits en punto Fijo	Bits en punto Flotante	Ejecución de FFT 1024 (ms)
1982	TMS320C10	390	16/32		
1985	TMS320C20	195	16/32		14.2
1985	TMS320C25	100	16/32		5.7
1987	TMS320C30	60	24/32	32/40	3.8
1990	TMS320C40	50	32/64	32/40	1.55
1993	TMS320C50	35	16/32		
1995	TMS320C54xx	20	16/40		
1997	TMS320C6xx	5	32/64	32/64	

ARQUITECTURA BASICA DE LA FAMILIA

TMS320 DE TEXAS INSTRUMENTS

Por: Larry Escobar Salguero.

Facultad de Ingeniería

UNAM

Junio del 2000

ARQUITECTURA BASICA DE LA FAMILIA TMS320 de TEXAS INSTRUMENTS

La familia de procesadores TMS320 de 16/32 bits de Texas Instruments, empleada para el Procesamiento Digital de Señales (PDS) combina la flexibilidad de un controlador de alta velocidad con la capacidad numérica de un arreglo procesador. Estas características hacen que esta familia ofrezca alta capacidad de ejecución y una arquitectura de gran funcionalidad para solucionar diversos problemas en telecomunicaciones, computación, comercio, industria, en aplicaciones militares, entre otras.

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 implementan funciones en hardware, que en comparación a otros procesadores, recurren al software o microcódigos para generarlas.

Un DSP para cumplir con sus propósitos de realizar algoritmos para el procesamiento digital de señales debe de realizar operaciones aritméticas muy rápidamente y manejar algoritmos con matemáticas intensivas para su realización en tiempo real. La familia de procesadores TMS320 tiene algunos conceptos de diseño básicos que los caracteriza y que son comunes en sus diversas generaciones de DSPs:

- Arquitectura Harvard.
- Extensivo Pipeline.
- Un multiplicador dedicado en hardware.
- Instrucciones especiales para el PDS.
- Ciclos de instrucción rápidos.
- Procesamiento en paralelo.

PRIMERA GENERACION DE LA FAMILIA TMS320

El TMS32010 es el primer miembro de la familia TMS320 de Texas Instruments (TI), es un microprocesador capaz de efectuar multiplicaciones de 16 x 16 bits en un simple ciclo de instrucción de 200 ns. Contiene 144 palabras disponibles en memoria interna y hasta cuatro mil palabras de memoria externa para memoria de programa que pueden ejecutarse a velocidad máxima. También está disponible en la versión de microcomputadora con 1.5 K palabras de memoria interna ROM de programa y hasta 2.5 K palabras de memoria para programas externos, para un total de cuatro K palabras. Está fabricado con tecnología CMOS logrando una baja disipación de potencia (165 mW).

En esta familia existen algunas alternativas con características variadas dependiendo de la aplicación concreta del usuario.

El **TMS320C10-14**, es una versión del TMS320C10 a 14 Mhz, ofrece una alternativa para aplicaciones de DSP cuando no se requiere operación a máxima frecuencia, puede ejecutar 3.5 MIPS a un ciclo de instrucción de 280 ns.

El **TMS320C10-25**, es una versión del TMS320C10 a 25 Mhz, tiene un ciclo de instrucción de 160 ns. Es de baja potencia y alta velocidad, conveniente para aplicaciones de alto desempeño en el procesamiento de señales digitales (PSD).

El **TMS320C14** y el **TMS320E14** son DSP's con un ciclo de instrucción menor de 160 ns, 256 palabras en RAM interna y cuatro K palabras en ROM interna para programa (el 'C14) o EPROM ('E14) respectivamente.

El **TMS320C15** y el **TMS320E15** es compatible en código y pin a pin con el TMS32010. Cada uno ofrece una expansión de memoria RAM interna de 256 palabras y cuatro K palabras en ROM interna ('C15) o EPROM ('E15). Ambos están en tecnología CMOS y en la versión de 160 ns de ciclo de instrucción.

El **TMS320C17** y el **TMS320E17** son microcontroladores, cada uno de 256 palabras de RAM interna y cuatro K palabras en ROM o EPROM. El **TMS320C17/E17** ofrece un puerto serial de doble canal capaz de efectuar una comunicación serie full-duplex y una interfaz directa a codecs. Además provee una conexión directa con microcomputadoras de 4/8 bits y microprocesadores de 16/32 bits. El hardware interno permite comprimir expandir (compand) datos en formato ley μ y ley A).

CARACTERISTICAS PRINCIPALES DEL TMS32010

- Bus de datos bidireccional de 16 bits.
- Acumulador de la unidad aritmético lógica (ALU) de 32 bits.
- Multiplicaciones de 16 x 16 bits con un producto de 32 bits.
- Corrimiento de 0 16 bits.
- Generador interno de reloj.
- Ocho canales de entrada y ocho de salida.
- Puede ejecutar 5 millones de instrucciones por segundo (MIPS).
- Dos registros índices para direccionamiento indirecto.
- Ciclo de instrucción de 200 ns.

ARQUITECTURA

La arquitectura del TMS320C1x (TMS10) incrementa su desempeño al permitir la búsqueda del programa y solapar operaciones de datos. El TMS10 efectúa multiplicaciones en un ciclo de instrucción, su flexibilidad mejora con el conjunto de instrucciones que soporta instrucciones de propósito general o de aplicaciones de PDS.

La memoria programa y datos están en dos buses separados, permitiendo un total solapamiento de la búsqueda de instrucción y la ejecución. Las modificaciones de la arquitectura Harvard de la familia TMS320 permite la transferencia entre los espacios de dato y programa incrementando así la flexibilidad.

La Unidad Aritmética Lógica (ALU) y el acumulador soportan doble precisión en aritmética de dos complemento con palabras de 16 bits. **El acumulador** (ACC) de 32 bits almacena la salida de la unidad ALU y además es una entrada a la ALU. El acumulador está dividido en parte alta (bit 31 al 16) y parte baja (bit 15 al 0) para instrucciones de almacenamiento de palabras en la parte baja o alta del acumulador.

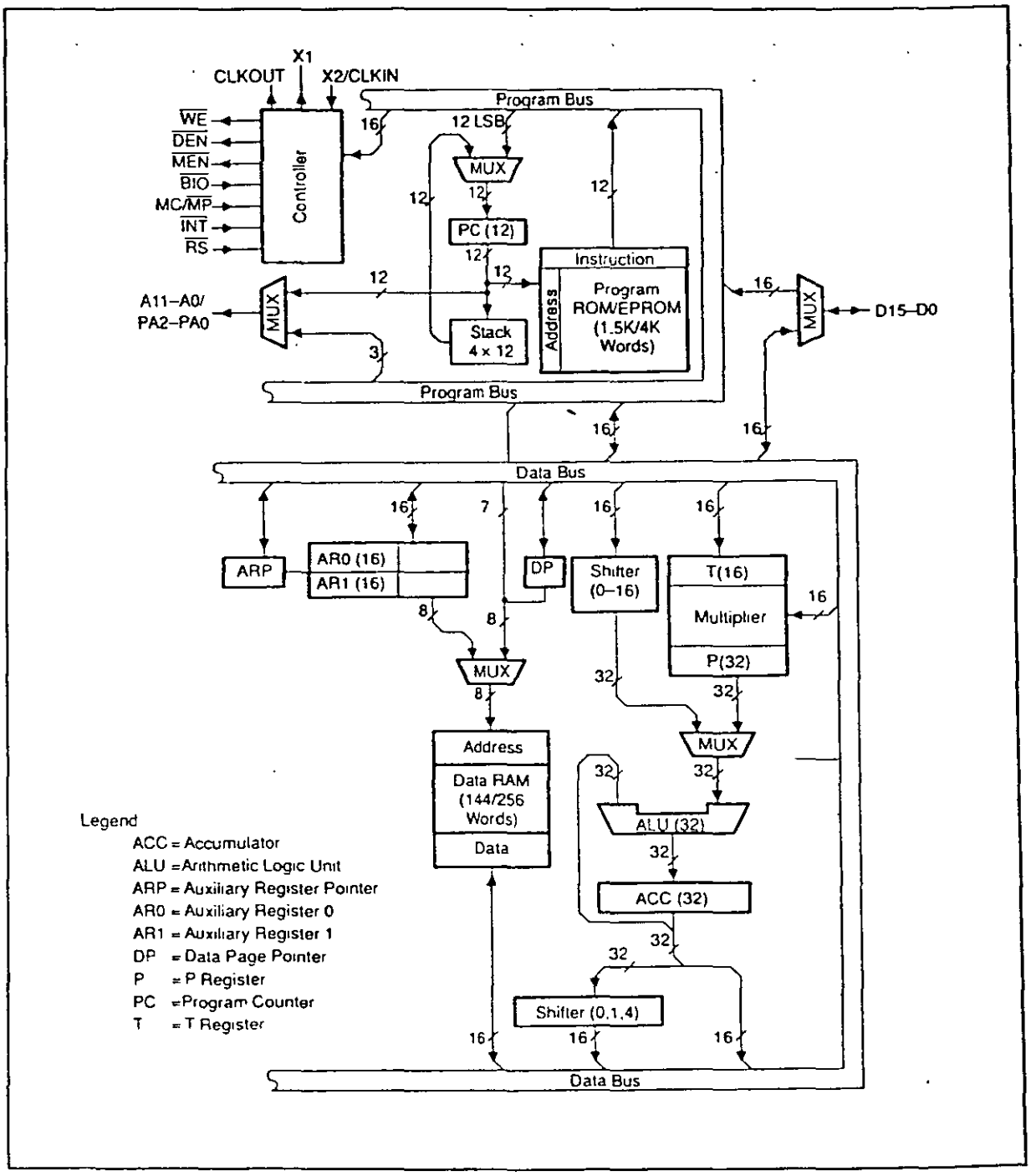
El multiplicador efectúa multiplicaciones en un ciclo de instrucción de 16 x 16 bits en dos complemento con resultado de 32 bits. El multiplicador consiste de tres elementos:

Registro **T**, de 16 bits para almacenamiento temporal del multiplicando.
Registro **P**, que almacena el producto de 32 bits.
Arreglo multiplicador.

Cada uno de los valores a multiplicar vienen de memoria dato o son derivados inmediatamente de la palabra de instrucción MPYK (multiplicación inmediata).

Existen dos posibilidades de **corrimiento** para datos:

- La ALU puede efectuar corrimientos de 0 a 16 bits a la izquierda para datos de memoria al cargarse a la ALU.
- El acumulador puede efectuar corrimientos de derecha a izquierda en 0,1 y cuatro posiciones al guardar su contenido en memoria. Estos corrimiento son útiles para el escalamiento y la extracción de bits.



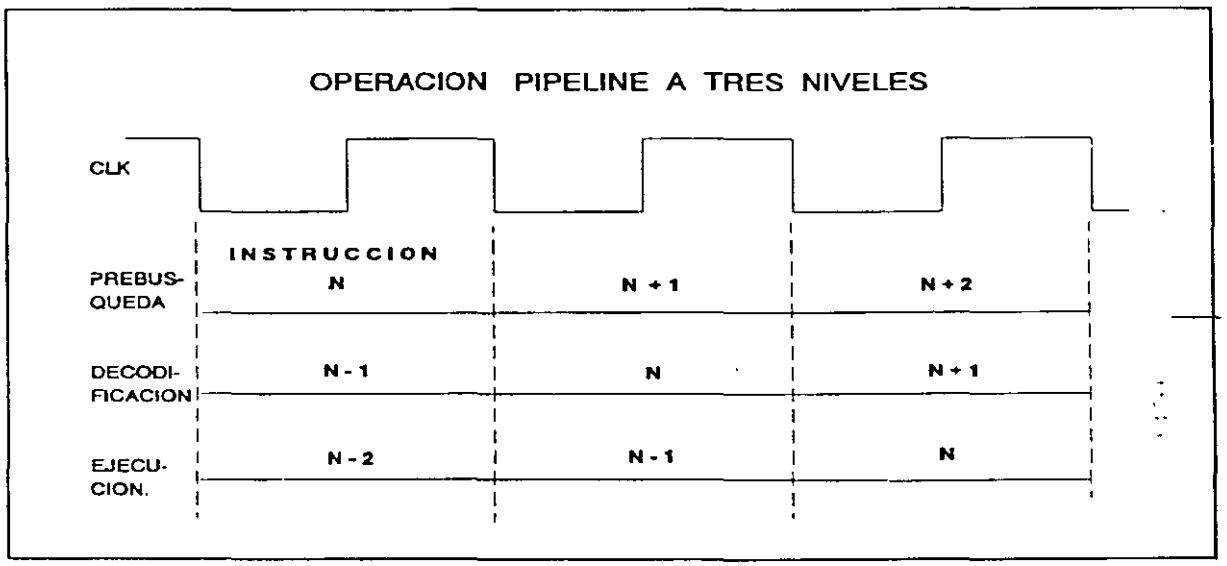
Arquitectura del TMS32C010 y TMS320C15

Contiene **cuatro niveles de pila** en hardware para salvar el contenido del contador de programa durante las interrupciones y llamadas de subrutinas, utilizando las instrucciones PUSH y POP. Los puertos de I/O son direccionados por los tres bits menos significativos (LSB) de las líneas de dirección.

LA OPERACION PIPELINE

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 las operaciones prebúsqueda, decodificación y 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 prebúsqueda, decodificación y ejecución son independientes.

Las arquitecturas de los DSPs tienen algunos rasgos en común: mientras una instrucción es buscada, el operando de la instrucción previa es buscado, esto puede verse como un flujo donde una instrucción es buscada, una decodificada y otra instrucción es ejecutada, es decir una forma de pipeline de tres estados.



En un programa típico para DSP una instrucción buscará dos operandos de memoria, los multiplicará, sumará el producto anterior al acumulador, escribirá el resultado en memoria e incrementará el contenido de los registros de dirección; en forma secuencial llevará varios ciclos de tiempo, mientras que en pipeline se optimizará el tiempo de las operaciones mencionadas. En esta arquitectura, las operaciones aritméticas son transferidas a estados sucesivos, donde unidades separadas de hardware proveen los cálculos a cada estado. Las computadoras que operan bajo este concepto son conocidas de procesamiento vectorial. La eficiencia de las funciones solo es posible si las operaciones aritméticas a ejecutar son vectorizables, es decir arregladas como un flujo continuo de datos. Una dificultad que se presenta con el pipeline son las ramificaciones, es decir que es difícil lograr eficientes saltos y ramificaciones debido a que el contador de programa debe de continuar en otra dirección del programa perdiéndose la secuencia de búsqueda y por lo tanto la operación de pipeline.

SEGUNDA GENERACION DE LA FAMILIA

TMS320

Por: **Larry Escobar Salguero.**

Facultad de Ingeniería

UNAM

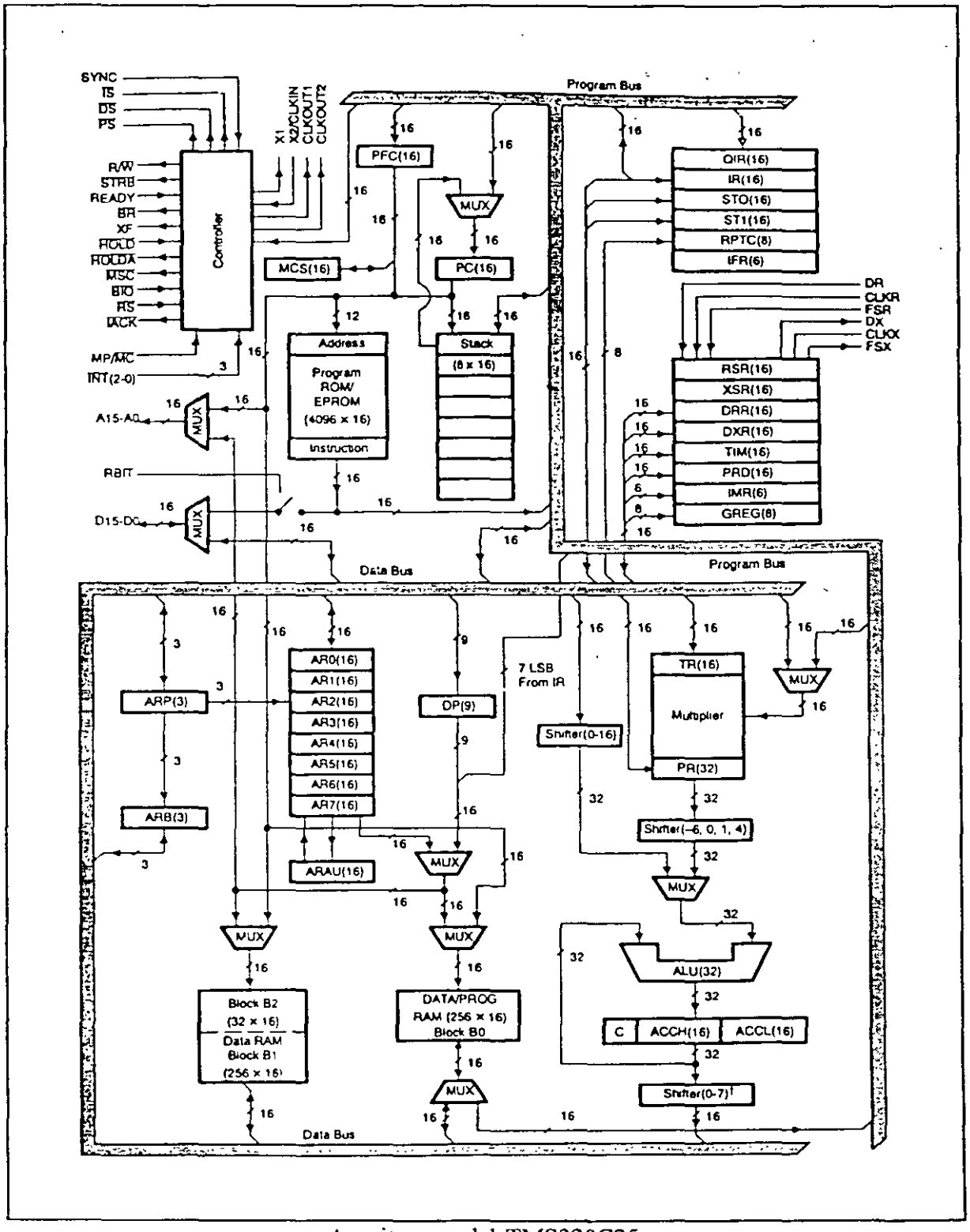
Junio del 2000

SEGUNDA GENERACION DE LA FAMILIA TMS320

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 el TMS23010. El TMS32020 trabaja a 20 MHz y ejecuta el doble de instrucciones del TMS32010, mientras el TMS320C25 incrementa la funcionalidad de la arquitectura en relación al TMS32020.

CARACTERISTICAS DEL TMS320C25 (TMS25)

- Ciclo de Instrucción 100 ns (10 MIPS).
- 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 16x16 con producto de 32 bits.
- Instrucciones multiplicación/acumulación ejecutadas en un ciclo de instrucción.
- Instrucciones de repetición para uso eficiente del espacio de programa.
- Movimiento de bloques para el manejo de DATOS/PROGRAMA.
- Timer integrado para operaciones de control.
- Ocho registros auxiliares con una unidad aritmética propia .
- Un stack por hardware de ocho niveles.
- Dieciseis puertos 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.
- Sincronía en configuraciones de múltiples procesadores.
- Interfaz de memoria de datos global.
- Compatibilidad de códigos fuente del TMS32010.
- Acceso directo a memoria (DMA).
- Instrucciones para implementar filtros, la Transformada Rápida de Fourier, y aritmética de precisión extendida.
- Generador de reloj interno.
- Suministro de potencia 5 Volts.
- Tres niveles de pipeline.



Arquitectura del TMS320C25

DESCRIPCION DE LA ARQUITECTURA

La arquitectura tipo Harvard del TMS25 maximiza el procesamiento mediante dos estructuras de buses de memoria separadas (memoria programa y memoria datos), para incrementar la velocidad de ejecución, contando con instrucciones de transferencia de datos entre ambos espacios. Externamente, las memorias de programa y datos son multiplexadas sobre el mismo bus externo para maximizar el rango de direccionamiento para dichos espacios mientras se minimizan los pines del dispositivo.

La flexibilidad en el diseño del sistema es incrementada al contar 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 TMS25 es capaz de direccionar externamente 64K-Palabras en un espacio de memoria dato, para facilitar la implementación de algoritmos para DSP.

Los programas de 4K-Palabras pueden ser mascarables en la memoria ROM interna y de esta forma pueden ser ejecutados a alta velocidad desde este espacio de memoria. Externamente el espacio de memoria de programa direccionable es de 64K-Palabras. El TMS25 funciona con una aritmética en modo dos complemento, empleando una ALU y un Acumulador de 32 bits.

El multiplicador realiza operaciones de 16x16 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, el registro P y un arreglo multiplicador.

El registro de corrimiento del TMS25 tiene una entrada de 16 bits y está conectado al bus de datos, mientras que su salida está conectada a la ALU 32 bits. Este registro proporciona corrimientos a la izquierda de 0 a 16 bits sobre el datos de entrada y son programados mediante instrucciones.

La interfaz local de la memoria del TMS25 consiste 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 de datos, y la señal /STRB provee un tiempo válido para la transferencia de información. La señal READY sirve para generar tiempos de espera para comunicarse con memorias externas lentas.

El stack por hardware de ocho niveles es empleado para almacenar el contenido de contador de programa durante interrupciones y llamadas de subrutinas. Las operaciones de control son proporcionadas en el TMS25 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.

El TMS25 soporta Acceso Directo a Memoria (DMA) para su memoria externa dato/programa empleando las señales /HOLD y /HOLDA. Otro procesador puede tomar por completo el control de la memoria externa. Sin embargo, de manera concurrente al modo DMA, el TMS50 continuará ejecutando su programa si éste opera con la RAM y ROM internas.

El TMS320C25 tiene un alto grado de paralelismo, es decir, que mientras está operando sobre la CALU, puede implementar operaciones aritméticas en la Unidad Aritmética de Registros Auxiliares (ARAU).

ORGANIZACIÓN DE LA MEMORIA

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 programa o dato por medio de instrucciones CNFP o CNPD respectivamente. Como memoria dato, B0 reside en las páginas 4 y 5 del mapa de memoria de datos, y como memoria de programa de FF00h a FFFFh, pudiéndose utilizar la instrucción BLKP (movimiento de bloque de memoria programa a memoria 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 dato. B1 está localizado en las páginas 6 y 7 abarcando 256 localidades mientras que B2 está en las treinta y dos localidades superiores de la página 0.

La memoria dato interna y las localidades reservadas para registros son mapeadas dentro de las primeras 1024-Palabras del espacio total de memoria dato.

CONJUNTO DE INSTRUCCIONES

El TMS25 cuenta con un conjunto de 133 instrucciones, 97 de ellas son ejecutadas en un ciclo, 36 requieren ciclos adicionales, 21 son para el control de flujo del programa, otras siete son de dos palabras (palabras largas inmediatas), las ocho restantes soportan transferencia I/O y de datos entre espacio de memoria. Además posee tres formas de direccionamiento: el indirecto, directo e inmediato.

QUINTA GENERACION DE LA FAMILIA

TMS320

EL DSP TMS320C50

Por: **Larry Escobar Salguero.**

Facultad de Ingeniería

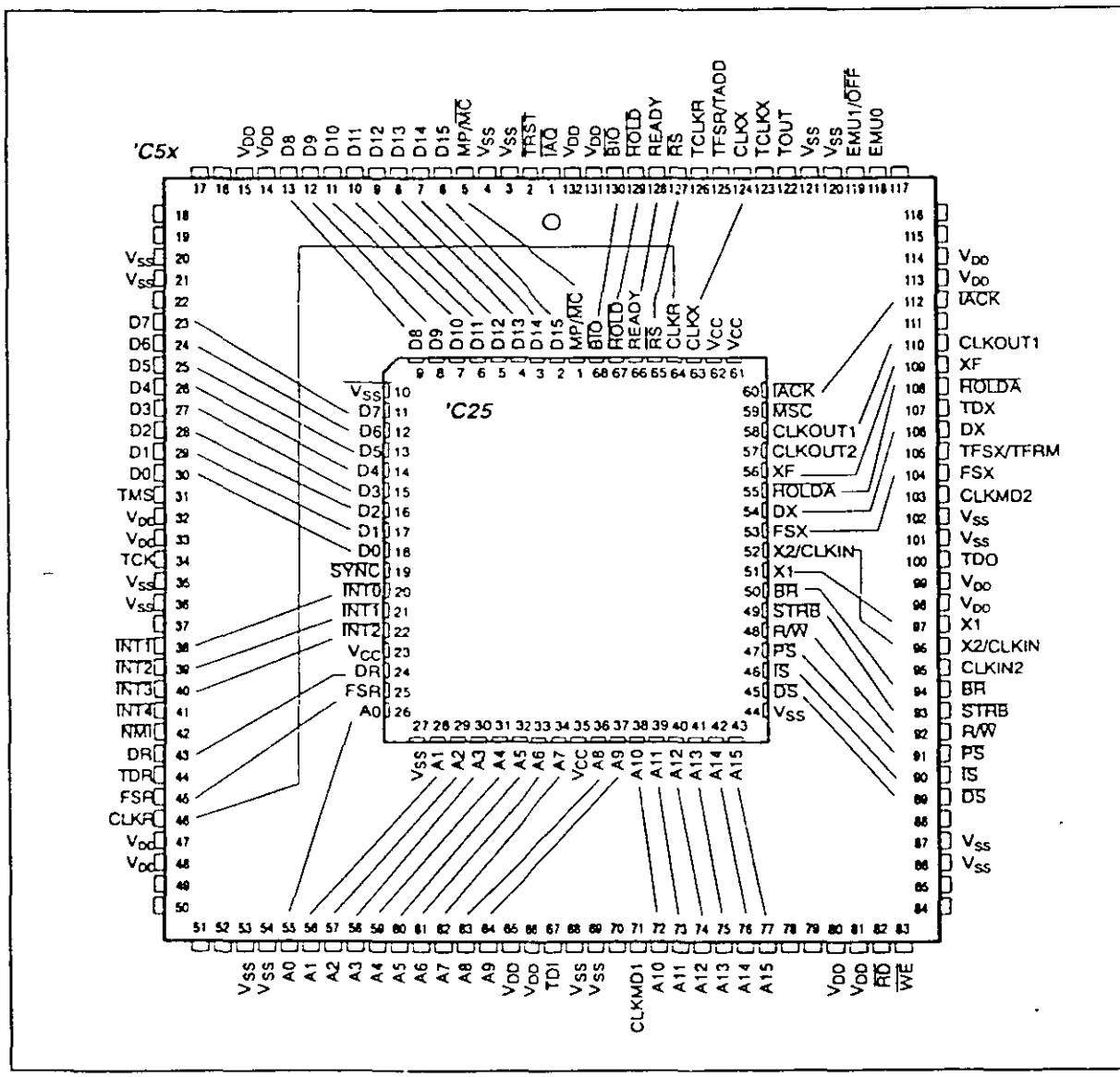
UNAM

Junio del 2000

QUINTA GENERACION DE LA FAMILIA TMS320
EL DSP TMS320C50

El TMS320C50 (TMS50) es un procesador de punto fijo de la familia TMS320 que está basado en el TMS320C25 con una arquitectura adicional que mejora el desempeño, entre estas caraterísticas están:

- Ciclo de instrucción más rápido
- Manipulación de bits vía la unidad paralela lógica.
- Gran cantidad de memoria interna.
- Estados de espera por software.
- Respuesta rápida a interrupciones.
- Bloques de repetición.
- Buffer circular.
- Bloques de repetición.
- Corrimientos de 0 a 16 bits en el acumulador.
- Registros shadow.



Relación del TMS25 al TMS50

Los dispositivos de la generación 'C5x son capaces de efectuar operaciones en dos veces la velocidad de la generación 'C2x y su código es compatible hacia arriba con las familias 'C1x y 'C2x. La familia 'C5x está diseñada para ejecutar 28 Millones de instrucciones por segundo (MIPS).

RASGOS PRINCIPALES DEL DSP TMS320C50

- 35/50 ns de ciclo de instrucción para instrucciones en punto entero (28.6/20 MIPS).
- 9K x 16 bits de memoria RAM interna, con acceso en un ciclo (SARAM).
- 2K x 16 bits en memoria ROM acceso en un ciclo.
- 1056 x 16 bits en memoria RAM interna, puede accesarse dos veces por ciclo de máquina (DARAM).
- 224k x 16 bits máximo espacio direccionable en memoria externa (64 K palabras[2] de programa, 64 k palabras de datos 64 k puertos I/O y 32 k palabras globales)
- Acumulador (ACC) de 32 bits y un acumulador buffer (ACCB) de 32 bits.
- Acumulador (ACCB) de 32 bits.
- Unidad Lógica Paralela (PLU) de 16 bits.
- Multiplicador paralelo de 16 x 16 con capacidad de productos de 32 bits.
- Instrucciones de multiplicación/acumulación en un ciclo simple.
- Ocho registros auxiliares con una unidad aritmética para direccionamiento indirecto.
- Ocho niveles de pila por hardware.
- 0 a 16 corrimientos a la izquierda.
- Dos buffer de direccionamiento indirecto para direccionamiento circular.
- Instrucciones de repetición de bloques.
- Instrucciones para el manejo de movimiento de bloques entre datos y programa.
- Puerto serial síncrono full-duplex para comunicación directa con otros 'C5x y otros dispositivos seriales.
- Puerto serial de múltiple acceso por división de tiempo (TDM).
- Temporizador
- 64k puertos I/O paralelos, con 16 puertos mapeados en memoria.
- 16 estados de espera programables por software, para programa, datos o espacio I/O.
- Operación de DMA.
- Cuatro niveles de Pipeline.
- Direccionamiento indexado.
- Direccionamiento de bit reverso para efectuar FFT's radix 2.
- Generador interno de reloj.
- Polarización de 5 volts, con modo power down.
- Empacado en 132 pines.
- 28 registros mapeados en memoria.
- Cuatro interrupciones externas y cinco internas una del timer y cuatro para puerto serie.
- 32 interrupciones por software.

[2] una palabra equivale a 16 bits

La familia TMS320C5x está disponible en diferentes versiones que se caracterizan por la distribución de la memoria interna:

	SARAM	ROM
TMS320C50	10K	2K
TMS320C51	2K	8K
TMS320C52	1K	4K
TMS320C53	4K	16K
TMS320C56	7K	32K

Algunas de estas versiones están disponibles para polarización de 5 y 3.3 volts.

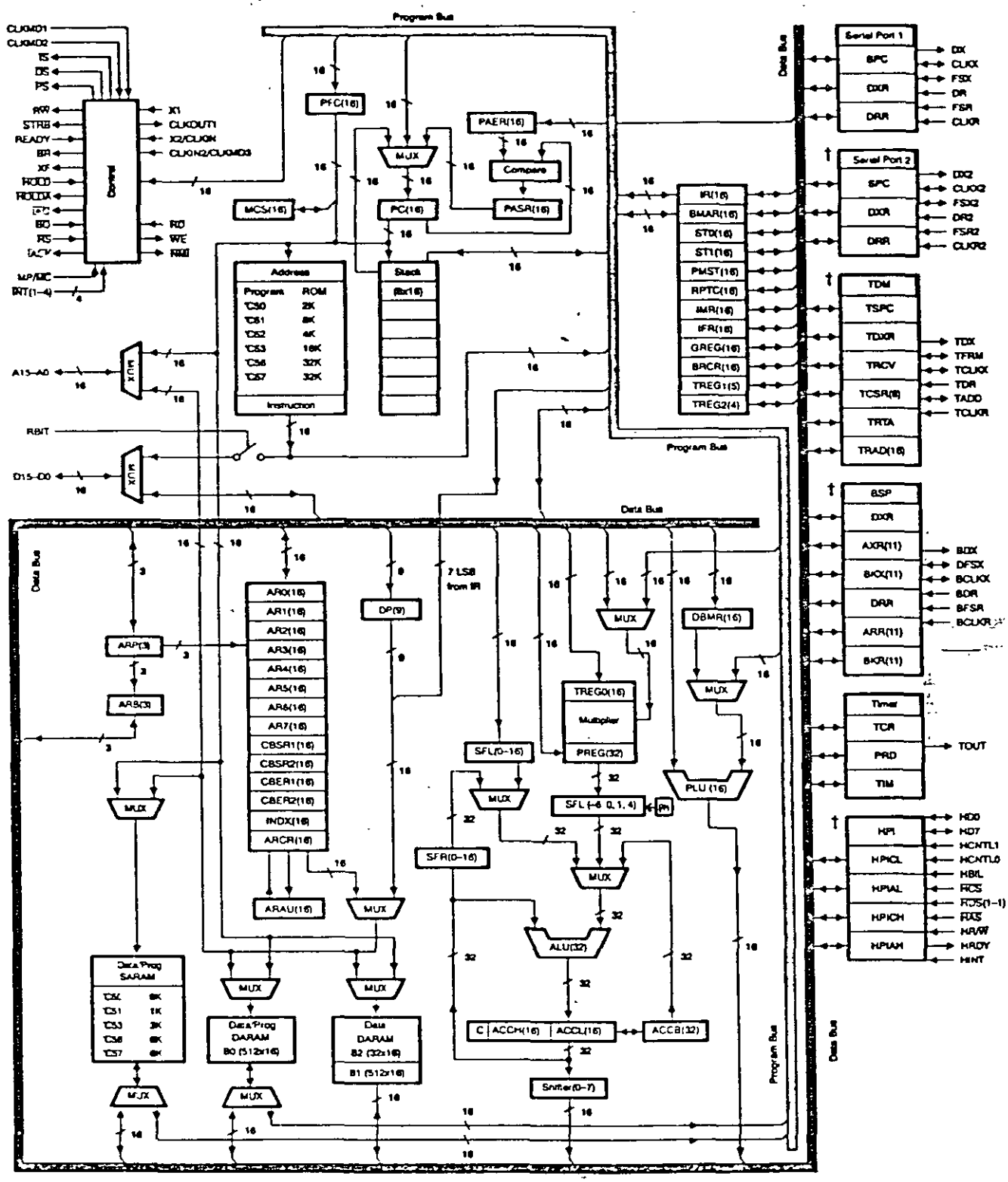
ARQUITECTURA

El TMS50 consiste de tres bloques básicos:

- Unidad Central de Proceso (CPU)
- Memoria (Unidad de direccionamiento)
- Interfaz a circuitos periféricos.
- Unidad de control

Su arquitectura utiliza dos buses separados, datos y programa, con instrucciones que aceptan transferencia de datos entres ambos espacios. El TMS50 efectúa aritmética de complemento a dos, utilizando la ALU y el acumulador de 32 bits. Contiene una Unidad Lógica Paralela (**PLU**), que ejecuta operaciones lógicas sobre la memoria de datos sin afectar el contenido del acumulador.

Su **Multiplicador** efectúa multiplicaciones de 16 x 16 bits en complemento a 2 con resultado de 32 bits en un simple ciclo de instrucción. El multiplicador consiste de tres elementos: un arreglo multiplicador, un registro producto (PREG almacena el producto de 32 bits) y un registro temporal (TREG0 que almacena un multiplicando).



Arquitectura del TMS320C50

Existen dos bloque de **corrimientos**, uno a la entrada de los datos de 16 bits y otro a la salida de datos de la ALU. A la entrada de los datos puede producir corrimientos de 0 a 16 bits a la izquierda, según se programe en la instrucción o definido en el registro contador de corrimiento TREG1.

Contienen una **Pila** de 8 niveles de profundidad, que salva el contenido del apuntador de programa (PC) durante las interrupciones o llamados de subrutinas. En las interrupciones los registros estratégicos (ACC, ACCB, ARCR, INDX, PMST, PREG, STO, ST1 y TREGs) son guardados en los registros shadow y recuperados en el retorno de interrupción.

El TMS50 posee alto grado de paralelismo, ya que mientras se están operando datos en la Unidad Central Aritmético Lógica (CALU), se pueden estar ejecutando operaciones aritméticas en la Unidad Aritmética de Registros Auxiliares (ARAU), estas operaciones simultáneas se efectúan en un ciclo de instrucción.

REGISTROS

El TMS50 consta de 28 registros mapeados en la parte baja de la memoria dato, en las localidades 04h a 5Fh, estos se listan a continuación:

Registro	Dirección (hex.)	Descripción
-	0-3	Reservado
IMR	4	Registro de mascara de interrupción.
GREG	5	Reg. para localización global de memoria.
IFR	6	Reg. de banderas de Interrupción.
PMST	7	Reg. de modo estado del procesador.
RPTC	8	Reg. contador de repetición.
BRCR	9	Contador de repetición de bloque.
PASR	A	Dirección inicial de programa para bloque de repetición.
PAER	B	Dirección final de programa para bloque de repetición
TREG0	C	Reg. temporal para multiplicando.
TREG1	D	Reg. temporal para contador de corrimiento dinámico.
TREG2	E	Reg. temporal usado como apuntador de bit en prueba dinámica de bit.
DBMR	F	Manipulación dinámica de bit.
AR0-AR7	10-17	Registros auxiliares.
INDX	18	Indice.
ARCR	19	Reg. Auxiliar de comparación.
CBSR1	1A	Dirección inicial de buffer circular 1.
CBER1	1B	Dirección final de buffer circular 1.
CBSR2	1C	Dirección inicial de buffer circular 2.

CBER2	1D	Dirección final de buffer circular 2.
CBCR	1E	Registro de control de buffer circular.
BMAR	1F	Registro de direccionamiento dinámico.
	20-35h	Periféricos mapeados en memoria.
	36-4Fh	Reservado.
	50-5F	Puertos mapeados en memoria PA0-PA15.

Estos registros pueden cargarse al acumulador en direccionamiento indirecto o directo a través de la instrucción LAMM (carga el acumulador con registro mapeado en memoria) sin necesidad de hacer cambio de página. Para cambiar el contenido de estos registros se utiliza el acumulador salvándolo con la instrucción SAMM (salva el acumulador en registro mapeado).

Ejemplo:

```
 LACL #1Ah      ; ACC = 1Ah, carga el ACC con la constante 1Ah.
SAMM BRCR       ; Carga el ACC en el registro contador de bloque.
LAMM RPTC       ; ACC = RPTC, carga el ACC con el contenido del
                ; registro RPTC.
```

DESCRIPCION GENERAL DEL TMS320C50

La destacada funcionalidad del TMS320C50 (TMS50 o C50) para el PDS, se debe a su tipo de arquitectura Harvard que maximiza el procesamiento mediante el establecimiento de dos estructuras de buses de memoria separadas (memoria programa y memoria datos) para incrementar la velocidad de ejecución, contando con instrucciones para realizar transferencia de datos entre ambos espacios. Externamente, las memorias de programa y datos son multiplexadas sobre el mismo bus externo para maximizar el rango de direccionamiento para dichos espacios mientras se minimizan los pines del dispositivo.

La flexibilidad en el diseño del sistema es incrementada al contar con tres bloques de datos internos en memoria RAM de doble acceso DRAM (un total de 1056-palabras), donde uno de ellos puede ser configurado como memoria programa o memoria dato y 9 k-palabra de simple acceso (SRAM), además el TMS50 es capaz de direccionar externamente 64 k-palabras en un espacio de memoria dato, para facilitar la implementación de algoritmos para DSP. La memoria interna ROM de 2 k-palabras puede ser usada para reducir el costo de sistemas. Los programas en memoria ROM interna pueden ejecutarse a alta velocidad desde este espacio de memoria. Externamente el espacio de memoria de programa direccionable es de 64 k-palabras.

El TMS50 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 producto (PR) 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 es dividida en parte alta (bits 31 al 16 ACCH) y parte baja (bits 15 al 0 ACCL), 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 16x16 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 TREG0 (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 la instrucción MPY #constante (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 de entrada está conectado al bus de datos y su salida de 32 bits está conectada a la ALU. Este registro proporciona corrimientos a la izquierda de 0 a 16 bits sobre el datos de entrada y son programados mediante las instrucciones. 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 consiste 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, además contiene las señales de salida /RD y /WE que se pueden conectar directamente a memorias RAM con pines de entrada /OE y /WE. Cuando emplea las memorias ROM, RAM internas o memoria de programa externa de alta velocidad, el TMS50 ejecuta instrucciones a la máxima velocidad sin tiempos de espera. Para direccionar memorias externas lentas emplea de la señal READY para generar tiempos de espera suficientes para la transferencia de información.

El stack (o pila por hardware de ocho niveles) es empleado para almacenar el contenido de contador de programa durante interrupciones y llamadas de subrutinas. Las instrucciones PUSH y POP permiten salvar y recuperar el acumulador parte baja (ACCL) contenida en el stack. Las interrupciones empleadas en el dispositivo son mascarables.

Las operaciones de control son proporcionadas en el TMS50 por: un timer interno de 16 bits mapeado en memoria, un contador de repetición, cuatro 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 registros del puerto serial mapeados en la memoria (registros de transmisión/recepción de dato) pueden operar en modo byte (ocho bits) o modo word (16 bits). Cada registro tiene una entrada de reloj externa, una entrada de sincronía y registros de corrimiento. Contiene un puerto serial (TDM, time división multiplexed) para aplicaciones de múltiples procesadores.

El TMS50 para aplicaciones de múltiples procesadores tiene la capacidad de distribuir el espacio global de memoria de dato y de comunicación con este espacio mediante las señales /BR (requerimiento de bus) y READY. El registro de distribución de la memoria global de dato (GREG) de ocho 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 es insertado para requerir el control del bus. La extensión del ciclo de lectura/escritura de memoria es controlado con la línea READY.

El procesador TMS50 soporta Acceso Directo a Memoria (DMA) para su memoria externa dato/programa empleando 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 TMS50 continuará ejecutando su programa si éste opera con la RAM y ROM internas.

La arquitectura del TMS50 está construida alrededor de dos buses: el de programa y el 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 lo son 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 operaciones de multiplicación/acumulación.

El TMS50 tiene un alto grado de paralelismo, es decir, que mientras está operando sobre la CALU, puede implementar 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.

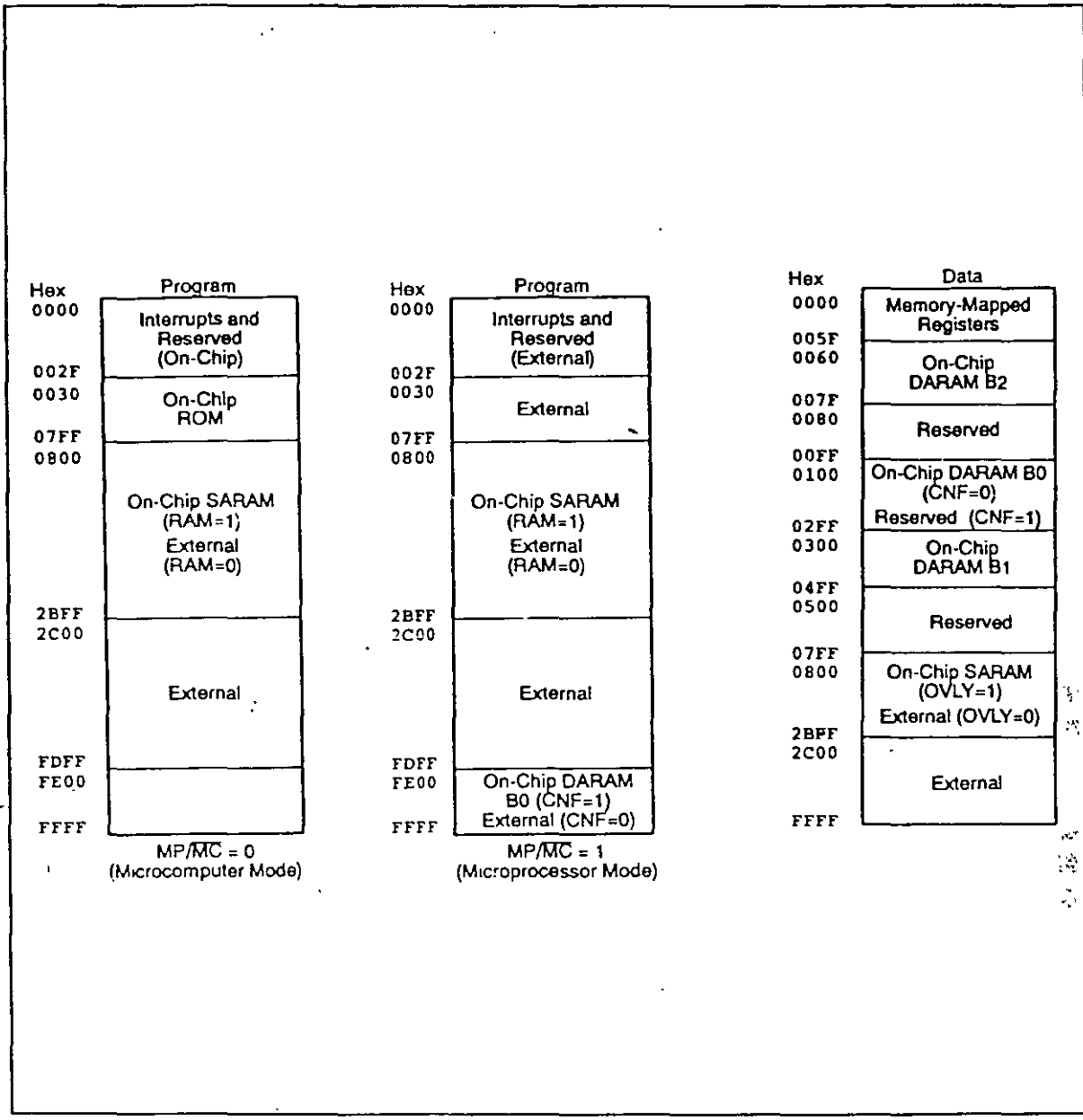
ORGANIZACIÓN DE LA MEMORIA DEL TMS50

El TMS50 posee un total de 1056-Palabras de 16 bit de DARAM interna, de las cuales 544-Palabras son siempre memoria dato y las 512 restantes pueden ser configuradas como memoria programa o dato. También provee 2k-Palabras ROM.

Las 1056 Palabras en DARAM interna se dividen en tres bloques separados: B0, B1 y B2. El bloque B0 de 512-Palabras es configurado como memoria programa (por instrucción SETC CNF) o dato (por medio de instrucciones CLRC CNF). Como memoria dato, B0 reside en las páginas 2 a 5 del mapa de memoria de datos (0100h-02FFh), y como memoria de programa de localidades FE00h a FFFFh. Las 544 Palabras (bloques B1 y B2) siempre son configuradas como memoria dato. B1 está localizado en las páginas 6 a 9 (300h-4FFh) abarcando 512 localidades mientras que B2 está en las treinta y dos localidades superiores de la página 0 (60h-7Fh), cada página consta de 128 palabras, es decir que el mapa completo tiene 512 páginas.

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 es hecho por medio del pin de selección MP/mc (Microprocesador/microcomputadora) Manteniendo MP/mc en alto mapea este bloque de memoria como externo, y MP/mc en bajo lo mapea en ROM interna.

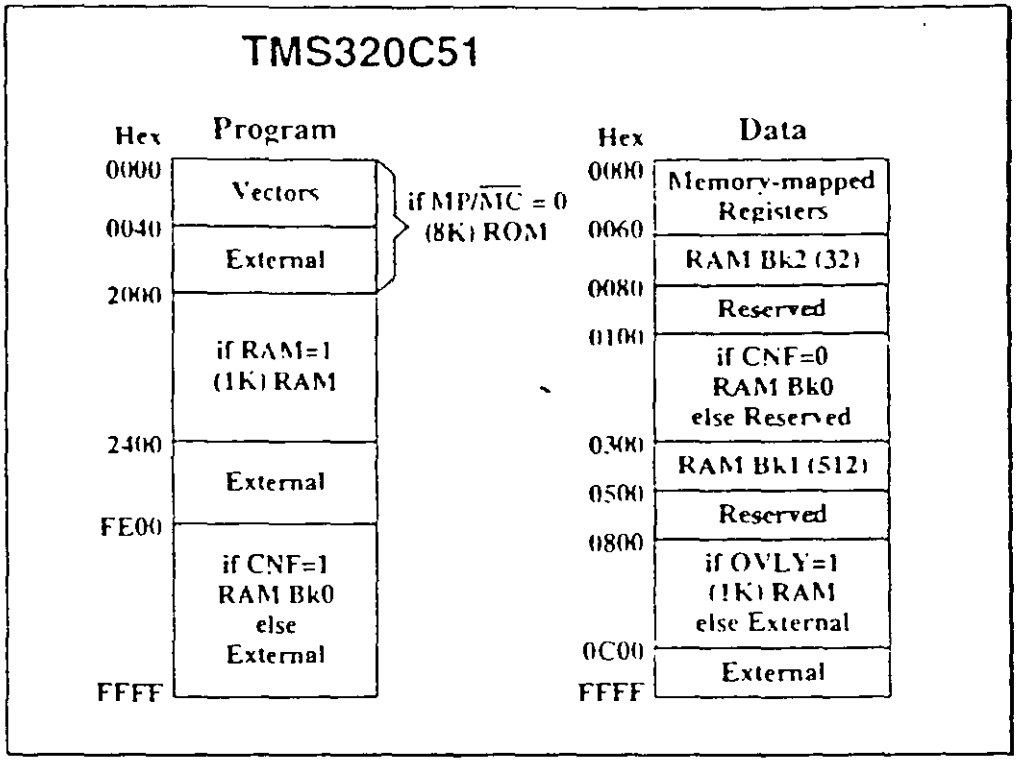
El TMS50 provee tres espacios de direccionamiento, para memoria programa, memoria dato e I/O. Estos espacios son distinguidos externamente por medio de las señales /PS (programa), /DS (dato) e /IS (puertos I/O). Las señales /PS, /DS, /IS Y /STRB son activadas sólo cuando un espacio externo de memoria está siendo direccionado.

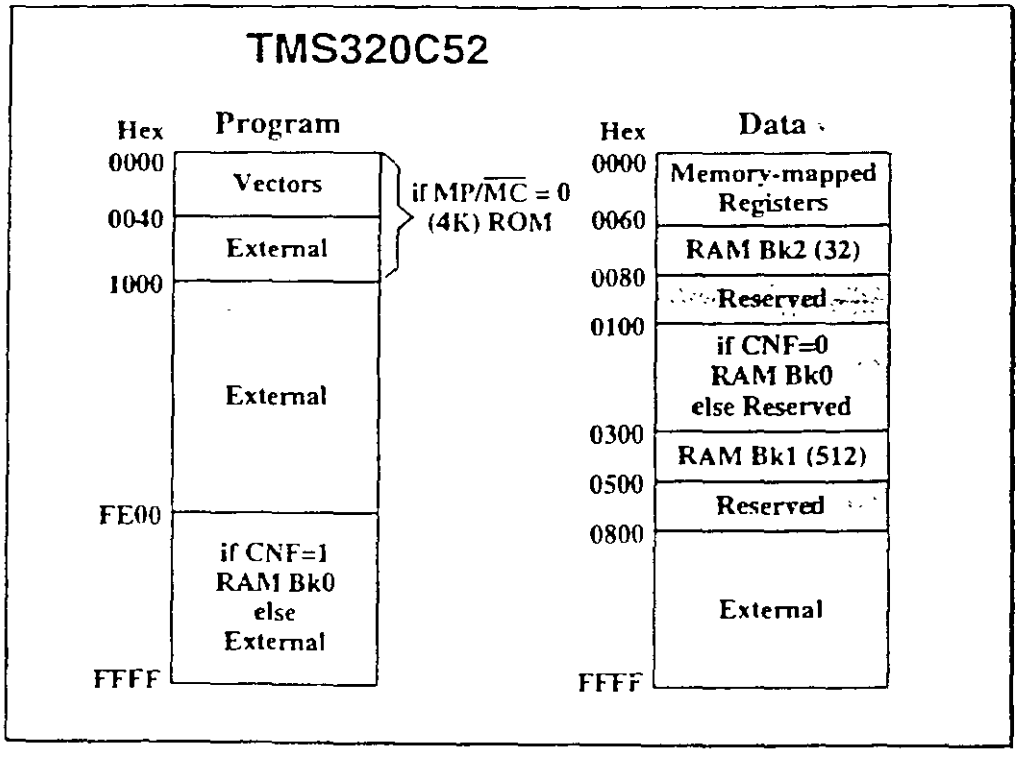


Mapa de memoria del TMS320C50

Durante un direccionamiento interno estas señales permanecen en estado inactivo alto para prevenir conflictos de direccionamiento cuando el B0 es configurado como memoria programa. La señal R/w determina el sentido del flujo de los datos, es decir, lectura (alto) y escritura (bajo).

MAPA DE MEMORIA DEL TMS320C51 y TMS320C52





MODOS DE DIRECCIONAMIENTO DE MEMORIA

Por: Larry Escobar Salguero.

Facultad de Ingeniería

UNAM

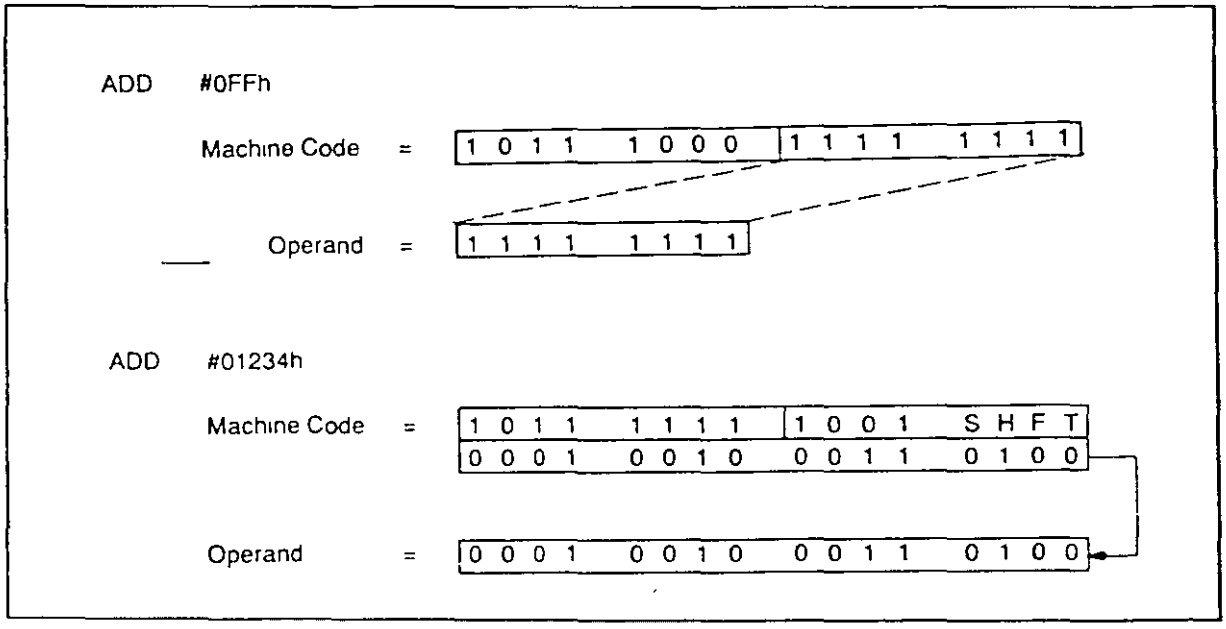
Junio del 2000

MODOS DE DIRECCIONAMIENTO DE MEMORIA

El TMS50 puede direccionar un total de 64k-palabras de memoria de programa y 64k-palabras d memoria de datos. Los 16 bits del bus de direcciones (DAB) direccionan la memoria de datos de do formas: modo de direccionamiento directo e indirecto.

MODO INMEDIATO CORTO Y LARGO

Cuando un operando inmediato es empleado, el operando está contenido en la instrucción, operando corto es una constante de 1 a 13 bits, en este caso el código de la instrucción es de 16 bit Para el caso de operandos inmediatos de 16 bits permite hace corrimientos a la izquierda sobre operando de hasta 15 bits, el código de instrucción es de 32 bits, el corrimiento está en la parte baj de la primera palabra y el operando queda contenido en la segunda palabra de la instrucción, es dec que su ejecución es más lenta.



Direccionamiento inmediato corto y largo

La diferencia entre un operando corto o largo es el tamaño del operando y si existe corrimiento operando es de tipo largo.
Ejemplos.

Operando corto

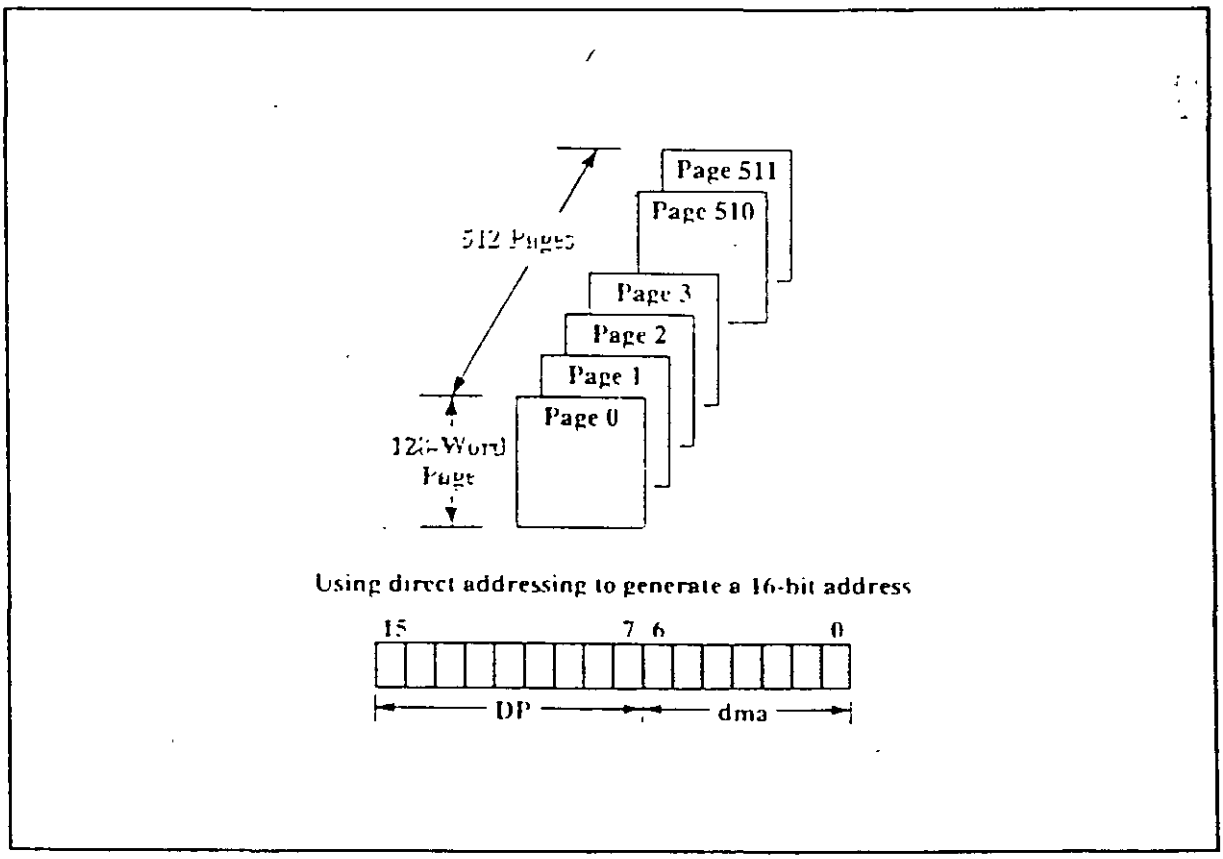
```
ADD  #0C1h ; suma al ACC una constante corta 0Ch
MPY  #03Ah ; multiplica el registro TREG0 por la constante 3A
```

Operando largo

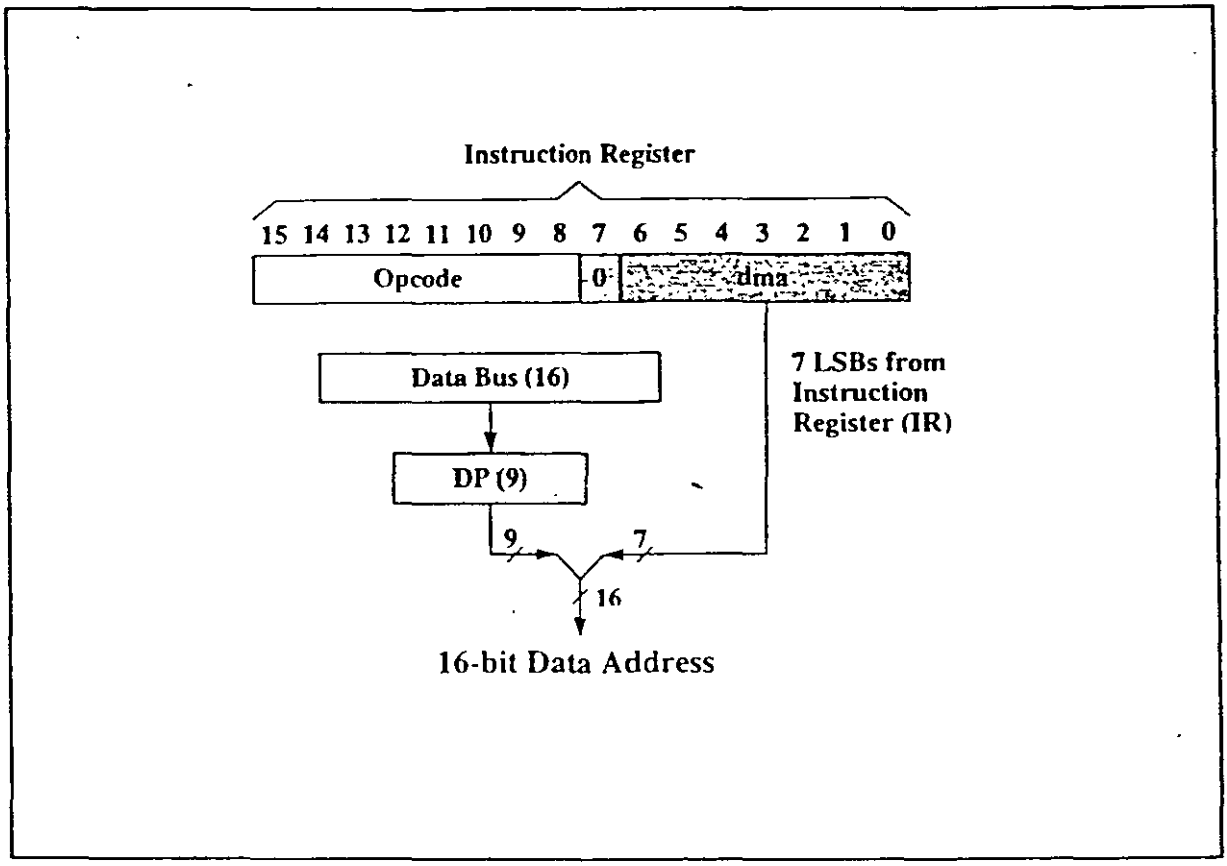
```
ADD   #111Fh,1   ;suma al ACC una constante larga con
                 ;corrimiento de un bit a la izquierda
MPY   #0124Ah    ;multiplica el registro TREG0 por la constante
                 ;124Ah
```

MODO DE DIRECCIONAMIENTO DIRECTO

A este modo se le llama directo ya que en el código de la instrucción está contenida una parte de la dirección del dato a operar. En este modo la memoria total de datos está dividida en páginas. Los 9 bits del apuntador de página DP pueden apuntar 512 páginas de 128 palabras cada una (64 k palabras). 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 (opera como una especie de offset sobre la página). Antes de usar este modo es necesario cargar el número de página en el registro DP; esto se hace con la instrucción LDP #pag.



Memoria dato en páginas



Modo de direccionamiento directo

Ejemplos de modo de direccionamiento directo:

```
LACL  dir_mem        ; carga el ACCL con el dato en dir_mem, ACCH=0
LACC dir_mem,shift   ; carga un dato al ACC con corrimiento (shift)
ADD  dir_mem,shift ; suma al ACC un dato en memoria co
                     ; corrimiento
SUB dir_mem,shift    ; resta del ACC un dato en memoria
                     ; con corrimiento
```

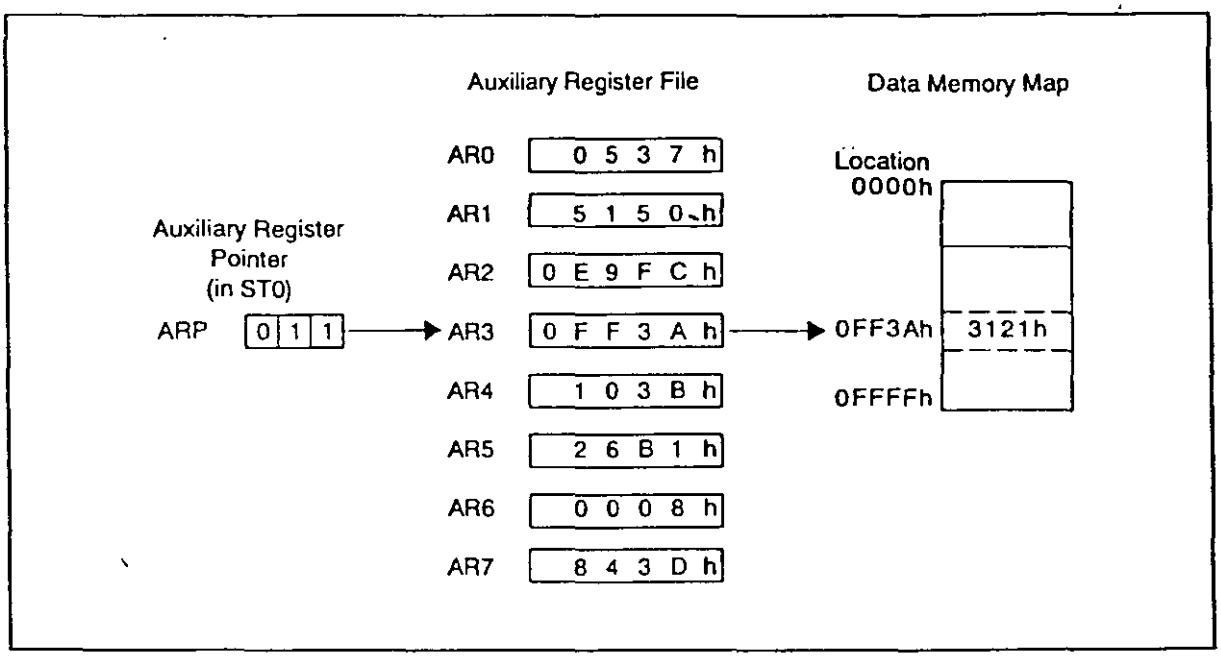
MODO DE DIRECCIONAMIENTO INDIRECTO

En este modo este modo, la dirección del dato a operar está en otro registro. Este modo es muy eficiente para el direccionamiento de tablas y arreglos, en la instrucción no se expresa ninguna dirección de memoria. 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 buffer (AFB).

Un operando * (asterisco) en una instrucción, implico que se está empleando este modo.

Registros Auxiliares

El TMS50 posee 8 Registros Auxiliares: AR0-AR7, los cuales pueden ser utilizados para direccionamiento indirecto de memoria dato (como se observa en la figura) o almacenamiento temporal de datos. Estos registros son seleccionados por un apuntador de Registros Auxiliares de tres bits (ARP), cargándose con los valores de 0 a 7 para designar desde AR0 a AR7 respectivamente.

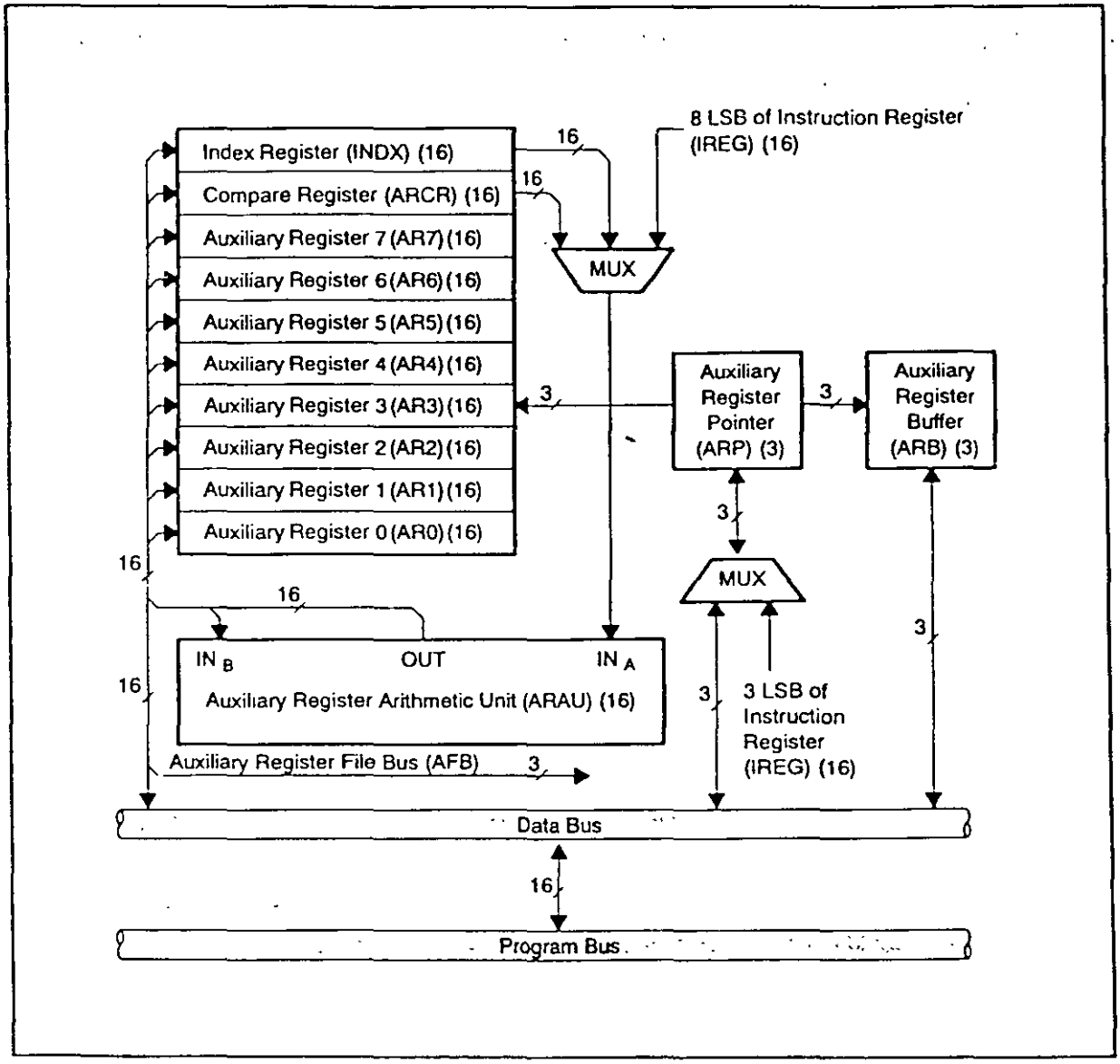


Direccionamiento indirecto

Los registros auxiliares están conectados a una Unidad Aritmética de Registros Auxiliares (ARAU). Esta unidad puede autoindexar el registro actual mientras que la localización del dato está siendo direccionada. Como resultado la CALU no accesa a tablas de información para manipuleo de direcciones, de este modo efectúa libremente otras operaciones. Los registros auxiliares deben de inicializarse y seleccionarse antes de ser utilizados, para cargarlos se utiliza la instrucción LAR y se seleccionan con la instrucción MAR.

Ejemplo:

```
LAR AR0,#100   ; Carga al registro AR0 con la dirección 100 d.
LAR AR1,#060h  ; Carga al registro AR1 con la dirección 60 h.
MAR *,AR1      ; Selecciona el registro AR1,
               ; no modifica al anterior ARi.
```



Registros auxiliares

Operaciones que efectúa la unidad aritmética de registros auxiliares ARAU

Operacion de ARAU	Comentario
AR(ARP) + INDX => AR(ARP)	al registro ARi actual en uso se le post-suma el contenido del registro INDX, ejem. ADD *0+
AR(ARP) - INDX ==> AR(ARP)	al registro ARi actual en uso se le post-resta el contenido del registro INDX, ejem. ADD *0-
AR(ARP) + 1 ==> AR(ARP)	el registro ARi actual en uso se postincrementa en uno, ejem. ADD *+
AR(ARP) - 1 ==> AR(ARP)	el registro ARi actual en uso se postdecrementa en uno, ejem. ADD *-
AR(ARP) ==> AR(ARP)	ARi sin cambios, ejem. SUB *
AR(ARP)+IR(7-0) => AR(ARP)	suma una constante de 8 bits inmediatos al ARi, ejem. ADRK #03ah
AR(ARP)-IR(7-0) => AR(ARP)	resta una constante de 8 bits inmediatos al AR(ARP), ejem. SBRK #06Bh
AR(ARP)+rc(INDX)=>AR(ARP)	le suma el registro INDX al ARi en uso con la propagación de carry inverso.
AR(ARP)-rc(INDX)=>AR(ARP)	le resta el registro INDX al ARi en uso con la propagación de carry inverso.

En la comparación del registro actual ARi con el registro ARCR, si la condición es cierta el bit TC del registro de estado ST1 se fija a uno, si es falsa se limpia. Con la instrucción CMPR #CM (donde CM puede valer 0,1,2,3) se pueden comprobar las condiciones respectivas al valor de CM:

CM	Comparación
00	(AR(ARP)) = (ARCR)
01	(AR(ARP)) < (ARCR)
10	(AR(ARP)) > (ARCR)
11	(AR(ARP)) ≠ (ARCR)

EJEMPLOS DE DIRECCIONAMIENTO INDIRECTO

1) Desarrollar un programa que realice la siguiente suma, suponer que los datos de x están almacenados a partir de la localidad "x".

$$y=\sum_{n=1}^{100} X(n)$$

```
LAR  AR1,#x   ; carga el registro AR1 con la dirección de x
MAR  *,AR1 ; selecciona al registro AR1 como registro
           ;auxiliar en uso, no modifica el anterior ARi
ZAP        ; ACC=0,    PR=0
RPT  #99   ; repite la siguiente instrucción 100 veces (n+1vez)
ADD  *+    ; suma al ACC lo que apunta AR1,  y AR1=AR1+1
SACL *     ; salva el ACCL en localidad apuntada por AR1
```

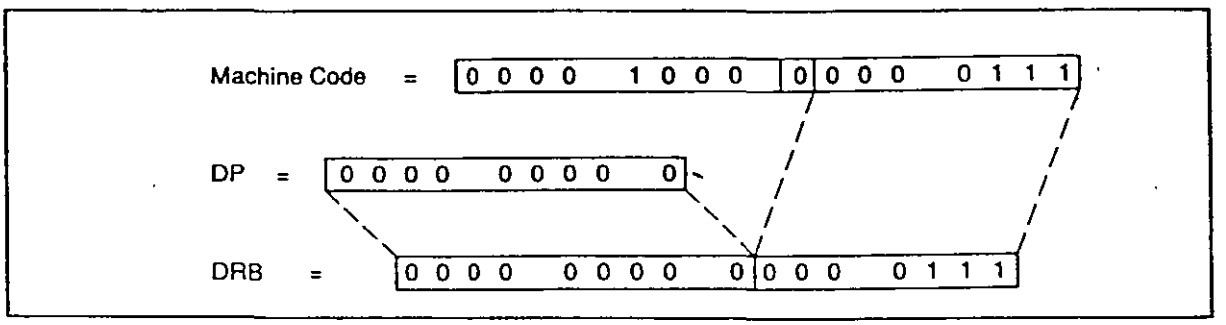
2) Desarrollar un programa que realice la siguiente suma de los valores el localidades pares de x , suponer que los datos de x están almacenados a partir de la localidad "x".

$$y=\sum_{n=0}^{49} X(2n)$$

```
LDP   #y   ; selecciona página donde está la variable y
LACL #2    ; ACC=2
SAMM  INDX ; Registro INDX = 2
LAR  AR1,#x ; carga el registro AR1 con la dirección de x
MAR  *,AR1 ; selecciona al registro AR1 como registro
           ;auxiliar en uso, no modifica el anterior ARi
ZAP        ; ACC=0,    PR=0
RPT  #49   ; repite la siguiente instrucción 100 veces (n+1vez)
ADD  *0+   ; suma al ACC lo que apunta AR1, y AR1=AR1+INDX
SACL y     ; salva el ACC en localidad de y en modo directo
```

DIRECCIONAMIENTO DE REGISTROS MAPEADOS

Este direccionamiento permite un acceso rápido a los registro mapeados en memoria. Opera similar al direccionamiento directo y se utiliza para acceder a los registros mapeados, en este caso se forza a los 9 bits más significativos de la dirección sean cero independiente del valor que tenga el registro de página, esto permite hacer un acceso directo a estos registros sin necesidad de hacer un cambio de página. La instrucciones utilizadas para este direccionamiento son LAMM y SAMM.



Direccionamiento de registros mapeados

Algunas instrucciones:

```
LAMM  dir_mem  ; el ACCL es cargado con el contenido de memoria direccionado(modo directo).
LAMM  *,ARi    ; el ACCL es cargado con el contenido de memoria direccionado(modo indirecto).
                 ; solo toma los siete bits menos significativos del ARi
SAMM  dir_mem  ; el ACCL es copiado en el registro mapeado (modo directo)
SAMM  *,ARi    ; el ACCL es cargado en memoria direccionada(en modo indirecto).
LMMR reg_map,#dir_mem ; el registro mapeado es cargado con el contenido de dir_mem
                 ; (modo directo)
LMMR *,#dir_mem,ARi  ; el registo mapeado y direcionado por los 7 bits LSB del ARi en uso, es
                 ; cargado con el contenido de dir_mem
SMMR reg_map,#dir_mem  ; el registro mapeado es guardado en  dir_mem (modo directo)
SMMR *,#dir_mem,ARi ; el registo mapeado y direcionado por los 7 bits LSB del ARi en uso, es
                 ; almacenado en de dir_mem
```

DIRECCIONAMIENTO CIRCULAR

El direccionamiento circular es utilizado para direcci onar eficientemente una ventana de datos que se se están procesando repetidamente, los siguientes registros son utilizados para el direccionamiento circular:

CBSR1	1A	Dirección inicial de buffer circular 1.
CBER1	1B	Dirección final de buffer circular 1.
CBSR2	1C	Dirección inicial de buffer circular 2.
CBER2	1D	Dirección final de buffer circular 2.
CBCR	1E	Registro de control de buffer circular.

El registro CBCR es de 8 bits que se definen:

Bit	Nombre	Función
0-2	CAR1	Identifica que registro auxiliar es utilizado para el buffer circular 1.
3	CENB1	Habilita buffer circular 1 con un uno, con cero lo deshabilita.
4-6	CAR2	Identifica que registro auxiliar es utilizado para el buffer circular 2.
7	CENB1	Habilita buffer circular 2 con un uno, con cero lo deshabilita.

Antes de utilizar el direccionamiento circular hay que cargar los registros de inicio y final de buffer y escribir al registro de control CBCR. Estos registros se pueden cargar con las instrucciones:

```
LACC #0100h
SAMM CBSR1
LACC #010Ah
SAMM CBER1
```

Estos registros también se pueden cargar con la instrucción SPLK que significa carga en paralelo una constante larga inmediata (ver unidad PLU), ejemplo:

```
SPLK #100h,CBSR1
SPLK #10Ah,CBER1
```

El control del buffer circular lo efectúa la unidad ARAU haciendo la comparación:

```
Si (AR(ARP))=(CBER)     entonces AR(ARP)=CBSR
                        de lo contrario AR(ARP) = AR(ARP) + PASO
```

significa, que si el buffer circular ha terminado el registro índice en uso se recarga con la dirección inicial del direccionamiento circular.

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 implementa aritmética no signada de 16 bits en comparación a la unidad central aritmético lógica (CALU) que implementa aritmética en modo dos complemento de 32 bits.

MODO DE DIRECCIONAMIENTO A REGISTROS

Este es un tipo de direccionamiento especial que opera con la unidad lógica paralela (PLU), la cual efectúa operaciones lógicas directamente sobre cualquier localidad de memoria dato sin afectar el contenido del ACC, permite direccionamiento directo ê indirecto y operaciones con el registro de manipulación dinámica de bits (DBMR). Esto se verá cuando se estudie la unidad PLU.

MOVIMIENTO DE MEMORIA A MEMORIA

Son instrucciones que para el movimiento de datos y programa permiten utilizar eficazmente la configuración de la RAM interna. La instrucción BLDP mueve un bloque de datos de memoria dato a memoria programa y BLPD mueve un bloque de programa a memoria de datos. Para el empleo eficiente de estas instrucciones, se utilizan las instrucciones de repetición RPT y RPTZ.

La instrucción DMOV permite mover una palabra de la dirección actual en la memoria dato en RAM interna a la próxima localización alta mientras que 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 la implementación de 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) permiten transferir palabras entre el espacio de programa y de dato.

UNIDAD CENTRAL ARITMETICA LOGICA

(CALU)

Por: Larry Escobar Salguero.

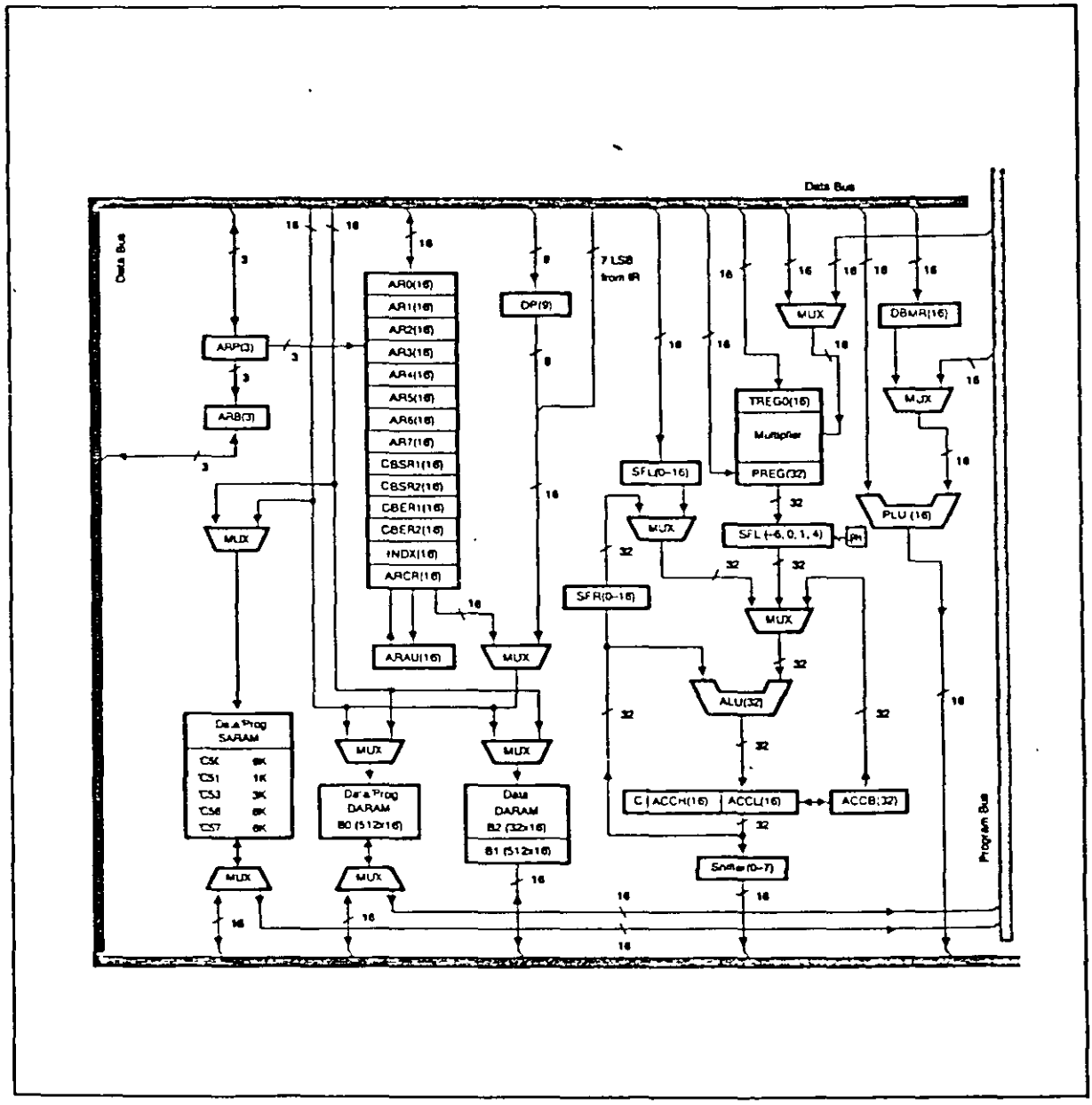
Facultad de Ingeniería

UNAM

Junio del 2000

UNIDAD CENTRAL ARITMETICA LOGICA

La Unidad Central Aritmética Lógica (CALU) del C50 contiene un registro de corrimiento de 16 bits, un multiplicador paralelo de 16x16 bits, una Unidad Aritmética Lógica de 32 bits, un acumulador de 32 bit (ACC) el cual está dividido en parte alta ACCH y parte baja ACCL, un acumulador buffer (ACCB) de 32 bits, un registro de corrimientos de salida para el acumulador y el multiplicador. Las instrucciones SFL y SFR realizan los corrimientos a la izquierda y derecha respectivamente del ACC.



Unidad central aritmética lógica

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

1) El dato es buscado de RAM a través del bus de datos.
2) El dato es pasado a través del registro de corrimiento y en la ALU 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 o del ACCB. Después de ejecutar la operación aritmética o lógica, la ALU almacena el resultado en el acumulador.

El TMS50 soporta operaciones de punto flotante requeridas para grandes rangos dinámicos. La instrucción NORM (normalizar) es utilizada para normalizar un número signado (en puntos fijo) contenido 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 donde el exponente está en los cuatro bits menos significativos del registro TREG1.

El modo de saturación de sobreflujo en el acumulador puede ser programado a través de la instrucciones SETC OVM (fija el modo) y CLRC OVM (quita el modo). Cuando el TMS50 está en modo de saturación y si un sobreflujo ocurre entonces 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. (7FFFFFFFh para positivo, y 80000000h para negativo).

Se pueden implementar 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) dependiendo de las operaciones a implementar. También existen instrucciones que permiten hacer corrimientos y rotaciones 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 o en el registro TREG1. 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.

Unidad Aritmético Lógica

Esta unidad es la encargada de efectuar sumas, substracciones y operaciones boolenas, el resultado de la ALU siempre queda en el registro ACC. Una entrada a la ALU siempre viene del ACC y la otra entrada puede venir de el ACCB, o del registro producto o del bus de datos o programa. Para efectuar sus operaciones utiliza los modos de direccionamiento inmediato, directo e indirecto.

Los cuatro del registro TREG1 se utilizan para indicar un corrimiento sobre un dato de entra a operarse en la ALU con las instrucciones ADDT, LACT y SUBT

Las instrucciones SFL y SFR efectúan un corrimiento del ACC un bit a la izquierda o derecha respectivamente. Si el modo signado está puesto (SXM = 1) la instrucción SFR efectúa un corrimiento aritmético a la derecha, es decir mantiene el signo del acumulador, cuando SXM = 0 la instrucción SFR efectúa un corrimiento lógico, el modo SXM no afecta a la instrucción SFL. Las instrucciones ROL (rotación a la izquierda) y ROR (rotación a la derecha) efectúa rotaciones del acumulador a través del bit de acarreo.

Las instrucciones SFLB, SFRB, RORB y ROLB pueden efectuar corrimiento y rotaciones de 65 bits al utilizar el bit de carry, el ACC y el ACCB.

El ACC puede ser corrido a la derecha en 0 a 31 bit en dos ciclos de instrucción o de 1 a 16 bits en un ciclo con la instrucción BSAR.

Cuando se obtiene un resultado de alguna operación aritmétoca o lógica éste se almacena en el ACC, si es necesario guardarlo en algún lugar de memoria se utilizan las instrucciones SACH (salva la parte alta del acumulador) y SACL (salva la parte baja del acumulador) con corrimientos de 0 a 7 bits a la izquierda, estos corrimientos se efectúan durante la transferencia de los datos sin afectar al contenido del ACC.

```
SACH VAR1,1     ; salva la parte alta del ACCH en VAR1 con corrimiento de uno.
```

Acumulador Buffer

Este registro de 32 bits acompaña al acumulador para efectuar operaciones de 32 bits, intercambio de datos y comparación de datos de una tabla, entre las instrucciones que utilizan:

```
LACB    ; carga al ACC con el valor del ACCB
SACB    ; salva el ACC en el registro ACCB
EXAR    ; intercambia el contenido del ACC con el ACCB
ADDB    ; ACC  = ACC + ACCB
ANDB    ; ACC  = ACC and ACCB
SBB     ; ACC = ACC - ACCB
ORB     ; ACC = ACC or ACCB
CRLT    ; ACC = ACCB con el valor menor que contenían ambos.
CRGT    ; ACC = ACCB con el valor mayor que contenían ambos.
XORB    ; ACC = ACC (XOR) ACCB
```

Estas instrucciones no utilizan operandos y se ejecutan en un ciclo de instrucción.

Multiplicador y Registros T y P

El TMS50 utiliza un hardware capaz de efectuar multiplicaciones de 16x16 bits en un ciclo de máquina. Todas las instrucciones de multiplicación exceptuando MPYU (multiplicación no signada) efectúan operaciones de multiplicación signada en el multiplicador. Es decir, que dos números pueden ser multiplicados siendo tratados 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:

- TREG0 registro temporal de 16 bits, que mantiene uno de los operandos a ser multiplicado.
- PR registro producto de 32 bits, que mantiene el resultado del producto.

La salida del registro producto puede ser corrida a la izquierda 1 ó 4 bits, esto es útil para la implementación de aritmética fraccionaria o justificación de productos fraccionales. La salida PR también puede ser corrida a la derecha en 6 bits habilitando la posibilidad de ejecución de 128 multiplicaciones/acumulación sucesivas sin sobreflujo. El registro TREG0 normalmente es es cargado con la instrucción LT, la cual le provee uno de los operandos (del bus de datos), y la instrucción MPY (multiplicación) provee el segundo operando también del bus de datos. Una multiplicación también puede ser ejecutada con un operando inmediato (de 13 bits) utilizando la instrucción MPY. En cada uno de los dos casos el producto es obtenido cada dos ciclos.

Las instrucciones de multiplicación/acumulación (MAC y MACD) utilizan totalmente el ancho de banda de cálculo de la multiplicación permitiendo que dos operandos sean procesados simultáneamente.

Las instrucciones SQRA (eleva al cuadrado y acumula un producto previo) y SQRS (eleva al cuadrado y resta del ACC el producto previo) 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, las cuales facilitan grandemente la precisión de operaciones aritméticas.
Algunas instrucciones para efectuar operaciones de multiplicación:

LT x	; Carga en modo directo el registro TREG0 con el contenido del dato x
LT *,AR2	; Carga el contenido de memoria apuntada por el reg. AR en uso al registro TREG0 y cambia al registro auxiliar AR2
LTA x	; Carga el registro TREG0 con el dato x, y acumula el producto previo ACC = ACC + PR
LTD x	; Carga el registro TREG0 con el dato x, acumula el producto previo y mueve el dato x a la siguiente localidad de memoria. El movimiento de dato sólo es válido en memoria RAM interna.

LTP x ; Carga el registro TREG0 con el dato x, y almacena el producto previo al ACC.
LTS x ; Carga el registro TREG0 con el dato x, y resta del ACC el producto anterior.
MAC 0FF00h,x ; Multiplica el contenido de FF00h en memoria programa con x, y acumula el producto previo.
MACD 0FF00h,x ; Multiplica el contenido de FF00h en memoria programa con x, y acumula el producto previo, y mueve el dato x a la localidad siguiente en memoria..
MPY x ; Multiplica el contenido de TREG0 con el contenido de x, el resultado lo deja en el registro PR. Esta instrucción se puede utilizar en direccionamiento directo, indirecto, inmediato corto y largo (sin corrimiento)
MPYA x ; Multiplica el contenido de TREG0 con el contenido de x, el resultado lo deja en el registro PR, y el producto previo es sumado al acumulador. Esta instrucción se puede utilizar en direccionamiento directo e indirecto.
MPYS x ; Multiplica el contenido de TREG0 con èl contenido de x, el resultado lo deja en el registro PR, y el producto previo es restado al acumulador. Esta instrucción se puede utilizar en direccionamiento directo e indirecto.
MPYU x ; Multiplica el contenido de TREG0 con el contenido no signado de x , el resultado lo deja en el registro PR, puede utilizarse en direccionamiento directo e indirecto.

En la operaciones anteriores cuando el registro PR opera con el ACC, el contenido del registro PR es corrido tal como se establece en registro de modo de corrimiento de PR. Se disponen de cuatro modos de corrimiento a través dèl Registro Producto (PR), los cuales son útiles cuando se ejecutan operaciones de multiplicación/acumulación, aritmética fraccional o justificación fraccional de productos. PM ocupa dos bits en el registro estado ST1 especificando el modo del producto (PM) como se indica:

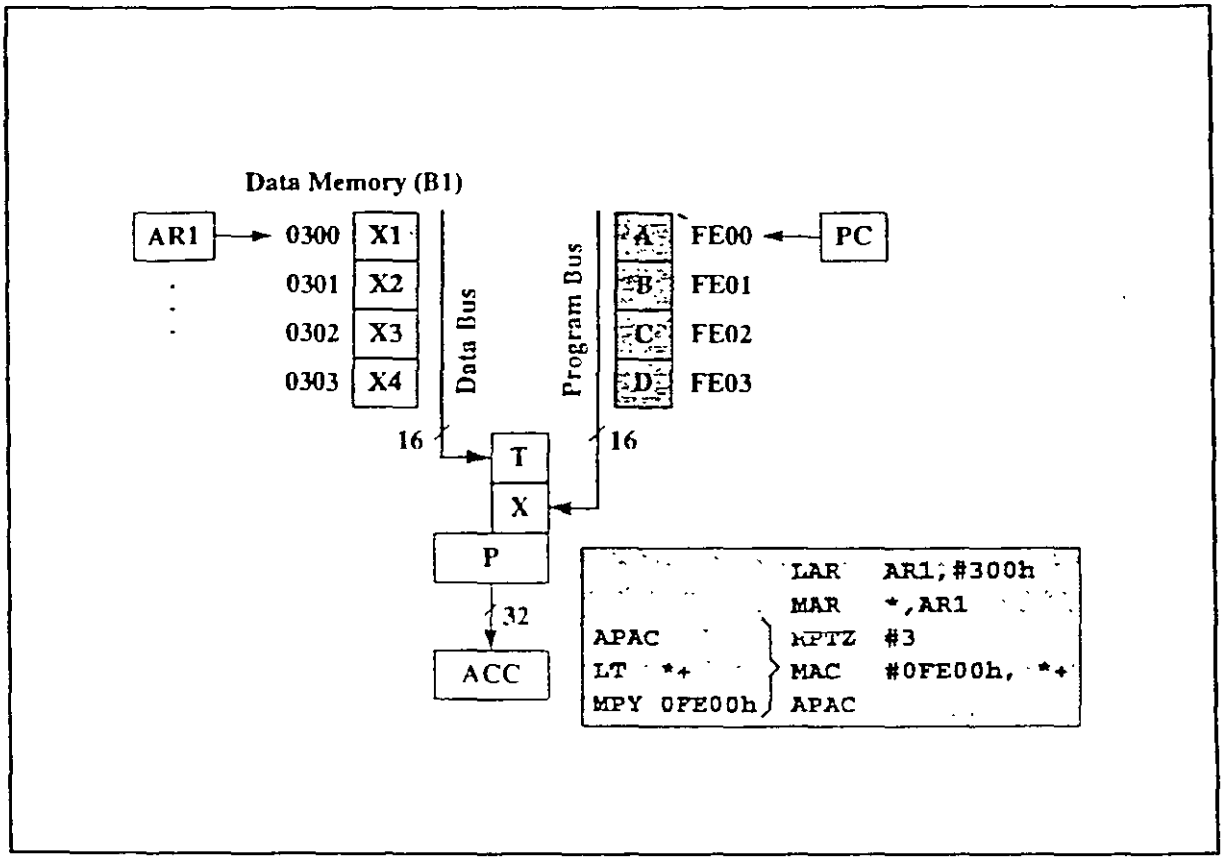
PM MODO DE CORRIMIENTO

PM	RESULTADO
00	No hay corrimiento
01	Corrimiento a la izquierda de 1 bit
10	Corrimiento a la izquierda de 4 bit
11	Corrimiento a la derecha de 6 bit

El resultado de una multiplicación (en el registro PR) puede transferirse al acumulador de varias formas por diferentes instrucciones (sin operandos):

PAC ; El contenido del registro PR corrido como se especifica en PM es cargado al ACC.
APAC ; El contenido del registro PR corrido como se especifica en PM es sumado al ACC.
SPAC ; El contenido del registro PR corrido como se especifica en PM es restado al ACC.

La unidad CALU del TMS50 alcanza su máximo desempeño cuando las instrucciones de multiplicación acumulación (MAC) o multiplicación acumulación y movimiento de datos (MACD) son utilizadas en conjunto con las instrucciones de repetición (RPT Y RPTZ) para efectuar operaciones de convolución.



Suma de productos con instrucción MAC

SISTEMA DE CONTROL

Por: Larry Escobar Salguero.

Facultad de Ingeniería

UNAM

Junio del 2000

SISTEMA DE CONTROL

El sistema de control del TMS50 está dado por el contador de programa (PC), el stack, la señal de reset externo, las interrupciones, los registros de estado y el contador de repetición (RPTC).

El Contador de Programa y el Stack

El contador de programa 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 una instrucción es buscada de la memoria programa y cargada en el Registro de Instrucción (IR). Cuando el IR es 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. El PC también direcciona la memoria de programa cuando está en 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 de la localización de la instrucción de salto.

El TMS50 permite la ejecución de GOTO computados al efectuar saltos en la dirección del acumulador con la instrucción BACC, o saltos al subrutina con la instrucción CALA a una dirección dada por el ACC.

EL TMS50 permite el movimiento de datos en memoria dato a través de la instrucción BLDD, entre memoria dato y programa por medio de la instrucción BLDP y de programa a datos por medio de la instrucción BLPD. Además también se pueden utilizar las instrucciones de movimiento de tablas TBLR y TBLW.

El TMS50 permite efectuar saltos condicionales con la instrucción BCND donde puede probar hasta 4 condiciones o llamadas condicional a subrutina CC, una por cada grupo de las siguientes posibilidades :

Grupo 1	Grupo 2	Grupo 3	Grupo 4
EQ ⟹ ACC = 0 NEQ ⟹ ACC ≠ 0 LT ⟹ ACC < 0 LEQ ⟹ ACC ≤ 0 GT ⟹ ACC > 0 GEQ ⟹ ACC ≥ 0	OV ⟹ OV = 1 NOV ⟹ OV = 0	C ⟹ C = 1 NC ⟹ C = 0	BIO ⟹ /BIO = 0 TC ⟹ TC = 1 NTC ⟹ TC = 0

UNC ==> Sin condición

Para que el salto sea válido se deben de cumplir todas las condiciones de la instrucción.
Ejemplo:

```
BCND  BANDERA,LT,OV  ; Salta a BANDERA si ACC < 0  y si existe overflow.
CC    SUB1,EQ        ; Salta a la subrutina SUB1 si el ACC = 0
```

Hay que notar que no todas la combinaciones de condiciones son válidas, por lo que debe de probarse sólo una de cada grupo si es necesario, ya que sólo es necesario llenar un campo de las condiciones.

Con la instrucción RPT #N el TMS50 puede ejecutar en el próximo ciclo N+1 veces la siguiente instrucción, donde la constante N es cargada en el contador de repeticiones RPTC (la constante N puede ser de 8 o 16 bits, la instrucción RPT también puede operar en modo directo e indirecto).

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.

OPERACION DE PIPELINE A CUATRO NIVELES

El TMS50 contiene cuatro niveles de Pipeline (búsqueda, decodificación, operando y ejecución), donde un contador de búsqueda contiene la dirección de la próxima instrucción a ser prebuscada. Una vez que una instrucción es buscada, entonces es cargada en un Registro de Instrucción (IR), al menos que IR contenga la instrucción en ejecución.

El PC contiene la dirección de la próxima instrucción a ser ejecutada, 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 es guardado en el stack para preservar su contenido al regreso de una interrupción o subrutina.

La prebúsqueda, decodificación y ejecución de la operación de pipeline son independientes, permitiendo así la ejecución de operaciones solapadas. Durante algún ciclo, tres instrucciones diferentes pueden ser activadas cada una a diferente estado de completez.

Cuando existen estados de espera, sólo retardan directamente la operación de pipeline. Cada estado de espera insertado durante la operación de búsqueda contribuye a agregar ciclos adicionales de máquina en la ejecución de pipeline de la instrucción.

INSTRUCCIONES DE SALTO Y SUBRUTINAS

Salto incondicional:

B ETIQUETA ; salta incondiconalmente a la dirección de programa donde esta ETIQUETA

Salto condicional:

BCND ETIQUETA, Condición1,Condición2,Condición3,Condición4 ;
; salta a ETIQUETA si se cumplen las cuatro dondiciones.

Salto computado o GOTO computado:

BACC ; el PC salta incondicionalmente a la dirección que contiene el ACCL,
; obviamente, antes se ha cargado una dirección en el ACCL.

Salto incondicional a subrutina:

CALL SUBRUTINA ; va a ejecutar la SUBRUTINA y retorna a la siguiente instrucción.

Llamada computado a subrutina o subrutina computada:

CALA ; salta incondicionalmente a una dirección contenida en el ACCL, el programa
; retorna a la siguiente instrucción después de haber ejecutado la subrutina.

Llamada condicional a subrutina:

CC ETIQUETA ; salta a ETIQUETA si se cumplen las cuatro dondiciones, después de
; ejecutar la subrutina el programa regresa a la siguiente instrucción.

Retorno de subrutina:

RET ; esta instrucción debe estar al final de toda subrutina para el programa regrese
; a dónde fue llamada la subrutina.

Ejecución condicional:

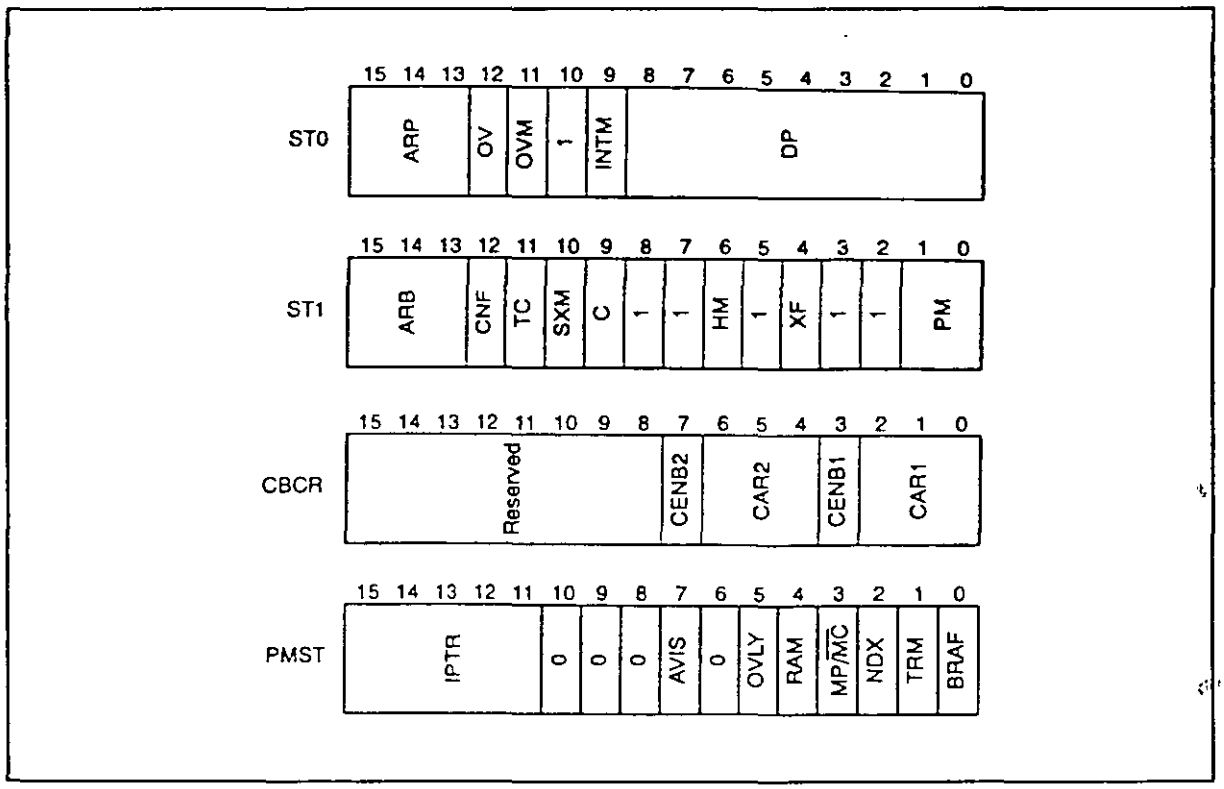
XC K,Condición1,Condición2,Condición3,Condición4 ;

Donde K=1,2 , el programa ejecuta las siguientes K instrucciones si se cumplen las condiciones, de lo contrario se salta esas instrucciones Hay que hacer notar que si K=1, la siguiente instrucción debe ser de una palabra y si K=2 la instrucción a ejecutar puede ser de 2 palabras o 2 instrucciones de una palabra.

De las instrucciones anteriores la mayoría se pueden ejecutar con retardo para la recuperación de pipeline, sin embargo, estas posibilidades son para programación avanzada.

REGISTROS DE ESTADO

El TMS50 contiene cuatro registros de estado, los registros de estado ST0 y ST1, contiene el estado de varias condiciones y modos de operación del TMS50 y los registros PMST y CBCR contienen estados extras y control de la información.



Registros de estado

Los registros de estado pueden ser almacenados dentro de la memoria dato y cargados de memoria dato, permitiendo así que el estado de la máquina sea salvado y restaurado para interrupciones y subrutinas Todos los bits de estado pueden ser cargados y salvados utilizando las instrucciones LST #(0-1) y SST #(0-1) respectivamente en direccionamiento directo e indirecto.

Los registros ST0, ST1 y PMST tiene asociado un registro sombra para salvarse automáticamente cuando una interrupción ocurre, en la rutina de atención de interrupción cualquiera de la instrucciones RETI o RETE le devuelve a los registros de estado sus valores originales antes de la interrupción.

Significado de los bits de registro de estado:

ARB Buffer Apuntador de Registros Auxiliares.
ARP Apuntador de Registros Auxiliares.
AVIS En uno permite la visualización de las líneas de dirección externas cuando el programa efectúa direccionamientos internos.
BRAF Bandera de repetición de bloque, en uno indica que se está repitiendo un bloque
C Bit de acarreo para una suma o préstamo para una resta.
CAR1 Tres bits que identifican que registro auxiliar AR se utiliza en el buffer circulara uno.
CAR2 Tres bits que identifican que registro auxiliar AR se utiliza en el buffer circulara dos.
CENB1 En uno habilita el buffer circulara uno.
CENB2 En uno habilita el buffer circulara dos.
CNF Bit de control de configuración de RAM interna.
DP Apuntador de página de Memoria Dato.
HM Bit de Modo Hold.
INTM Habilita interrupciones mascarables, 0 = habilitado,1 = deshabilitado.
IPTR Apuntador de vector de interrupción. Cinco bits que apuntan a 2000 localidades donde reside el vector de interrupción. Permite el remapeo de los vectores de interrupción.
MP/mc Bit de modo microprocesador (1) /microcomputadora (0).
NDX Habilita el registro índice.
OV Bit de bandera de sobreflujo.
OVLY Habilita el acceso de programa en memoria RAM, si es uno, el bloque de memoria es mapeado en el espacio de dato, si es cero, el bloque de memoria no es direccionable en memoria dato. Fijado a cero en el reset.
OVM Bit de modo de sobreflujo.
PM Modo de corrimiento del registro producto al cargarse en el ACC.
RAM Habilita el programa en memoria RAM. Habilita la SARAM mapearse en el espacio de memoria programa. Fijado a cero en el reset.
SXM Bit de modo de signo extendido y extensión de signo.
TC Bandera de control de prueba de bit.
TRM Habilita múltiples TREGs, En uno habilita el uso de los registro TREG1 y TREG2.
XF Bit de estado del pin externo XF.

Configuración de la memoria SARAM

OVLY	RAM	Configuración de SARAM
O	O	Deshabilitada
0	1	Mapeada en el espacio de programa.
I	0	Mapeada en el espacio de datos.
1	1	Mapeada en ambos espacios, datos y programa.

Instrucciones asociadas don los registros de estado y los bits individuales:

```
SETC  BIT          ; el BIT de control especificado es puesto a 1.
CLRC  BIT          ; el BIT de control especificado es limpiado.

LST   n,dir_mem    ; el registro de estado STn es cargado en modo directo con dato en
                   ; dir_mem.
LST   n,*,ARi      ; el registro de estado STn es cargado en modo indirecto con el dato
                   ; apuntado por el ARi en uso.
SST   n,dir_mem    ; el registro de estado STn es almacenado en modo directo en dir_mem.
SST   n,*,ARi      ; el registro de estado STn es almacenado en modo indirecto en
                   ; dirección apuntada por el ARi en uso.
```

JERARQUIA DE CICLOS

En múltiples aplicaciones es necesario anidar varios ciclos, por lo que es necesario conservar cierta jerarquía para obtener un buen desempeño:

```
        LAR   ARi,#N1
ETIQ1   INSTRUCCION
        INSTRUCCION
        ...
        ...
        LACC #NB
        SAMM BRCR
             RPTB  FINB
             INSTRUCCION
             INSTRUCCION
               ...
               RPT  #N
               INSTRUCCION ; ciclo de una instrucción del  RPT
             INSTRUCCION
               .....
FINB         INSTRUCCION      ; fin del ciclo del bloque de instrucciones.
        INSTRUCCION
        INSTRUCCION
        ....
        MAR *,ARi            ; asegura al ARi para que sirva de contador
        BANZ  ETIQ1,*-;ARi   ; salta a ETIQ1 si ARi en uso es diferente de cero.
```

CONTADOR DE REPETICION (RPTC)

RPTC es un registro de 16 bits, que cuando es cargado con un número N en la instrucción RPT, causa que la siguiente instrucción sea ejecutada N+1 veces. El RPTC puede ser cargado con un número e 0 a 65,536 usando las instrucciones RPT #N o RPTZ #N. Es utilizado en las instrucciones de multiplicación/acumulación, movimiento de bloques, transferencias de I/O y tablas de lectura/escritura. La instruccción RPTZ limpia el acumulador y el registro producto (PR) antes de empezar la repetición de la siguiente instrucción. Cuando la instrucción de repetición (RPT o RPTZ) es decodificada todas las interrupciones mascarables son deshabilitadas, sin embargo el TMS5O responde a señales de HOLD mientras efectúa un ciclo de repetición. Las instrucciones de repetición se utilizan en conjunto con las instrucciónes MAC, MACD, BLDD, BLPD, TBLR y TBLW.

Ejemplo:

Desarrollar un programa que realice la siguiente suma de productos entre 50 puntos de la señal X(n) y los coeficientes h(n), suponer que los datos de X(n) están almacenados a partir de la localidad "x" y los coeficientes h(n) están almacenados a partir de la localidad "h", el resltado lo salva en y.

$$y=\sum_{n=0}^{49} X(n)\,h(n)$$

```
LDP  #y    ; carga página donde está y
LAR  AR1,#x ; carga el registro AR1 con la dirección de x
MAR  *,AR1 ; selecciona al registro AR1 como registro
           ;auxiliar en uso, no modifica el anterior ARi
ZAP        ; ACC=0,   PR=0
  RPT  #49   ; repite la siguiente instrucción 50 veces
  MAC  #h,*+ ; suma al ACC lo que apunta AR1,  y AR1=AR1+1
APAC         ; rescata el último producto
SACL y     ; salva el ACCL en localidad "y"
```

Hay que hacer notar que notar que no todas las instrucciones se pueden utilizar con las instrucciones de repetición de ciclo y algunas no tienen significado (Ver el manual de usuario pag. 3-42).

REPETICION DE BLOQUES

El TMS50 permite la implementación de ciclos DO y FOR. Para su ejecución utiliza los registros PASR, PAER, BRCR y el bit BRAF en el rgistro PMST.

El registro contador de repetición de bloque BRCR es cargado con la cuenta del ciclo desde 0 a 65,536. La instrucción de repetición de bloque RPTB es la que inicia el bloque y carga la dirección de la siguiente instrucción en el registro de inicio de bloque PASR y la dirección del última instrucción del bloque la carga en el registro PAER. Cuando se inicia la ejecución de bloque el bit BRAF se pone en uno. La dirección de PAER es comparada con el PC, si son iguales entonces el contenido de BRCR es comparado a cero, si el registro BRCR es igual a cero el ciclo termina, de lo contrario éste se decrementa y el PC es cargado con la dirección de PASR.

Ejemplo:

Desarrollar un programa que realice la misma suma que se hizo con RPT, pero ahora con RPTB

$$y=\sum_{n=0}^{49} X(n)\,h(n)$$

```
      LDP   #y    ; selecciona página donde está la variable y
      LAR  AR1,#x ; carga el registro AR1 con la dirección de x
      LAR  AR2,#h ; carga el registro AR2 con la dirección de h
      MAR  *,AR1 ; selecciona al registro AR1 como registro
                 ;auxiliar en uso, no modifica el anterior ARi
      LACC  #49   ; ACC=49
      SAMM  BRCR ; el registro contador de bloque BRCR = 49
      ZAP        ; ACC=0,    PR=0

      RPT  FINB ; repite las instrucciones de blque 50 veces
       LT  *+,AR2 ; TREG0 = X(n)
       MPY *+,AR1 ; PR = X(n)*h(n)
FINB   APAC        ; ACC=ACC+PR
      SACL y     ; salva el ACC en localidad de y en modo directo
```

Algunas consideraciones en la repetición de bloques:

- RPTB si es interrumpible.
- El ciclo RPT puede anidarse dentro del ciclo RPTB
- El anidamiento de ciclos RPTB no es recomendado, ya que hay que estar salvando y restaurando registros ocasionando más consumo de tiempo.
- Las interrupciones y llamadas a subrutina son permitidos dentro de un bloque de repetición.
- Un bloque debe tener al menos tres instrucciones de longitud.

MODO POWER-DOWN

Este modo le permite al TMS50 entrar al modo de consumo de baja potencia. Este modo se invoca con las instrucciones IDLE e IDLE2 o por la entrada /HOLD en bajo cuando el bit de estado HM está en uno.

En el modo Power-down detiene la ejecución del TMS50 manteniendo el contenido de todos sus registros internos. Si este modo se invocó con la instrucciones IDLE o IDLE2 entonces el TMS50 sale de este modo al recibir alguna interrupción, si el bit INTM = 0 (interrupciones mascarables habilitadas) el TMS50 va a atender la rutina de interrupción y después continúa con el programa, si INTM = 1, entonces el TMS50 continúa con la intrucción seguida a IDLE o IDLE2.

UNIDAD LOGICA PARALELA

Por: Larry Escobar Salguero.

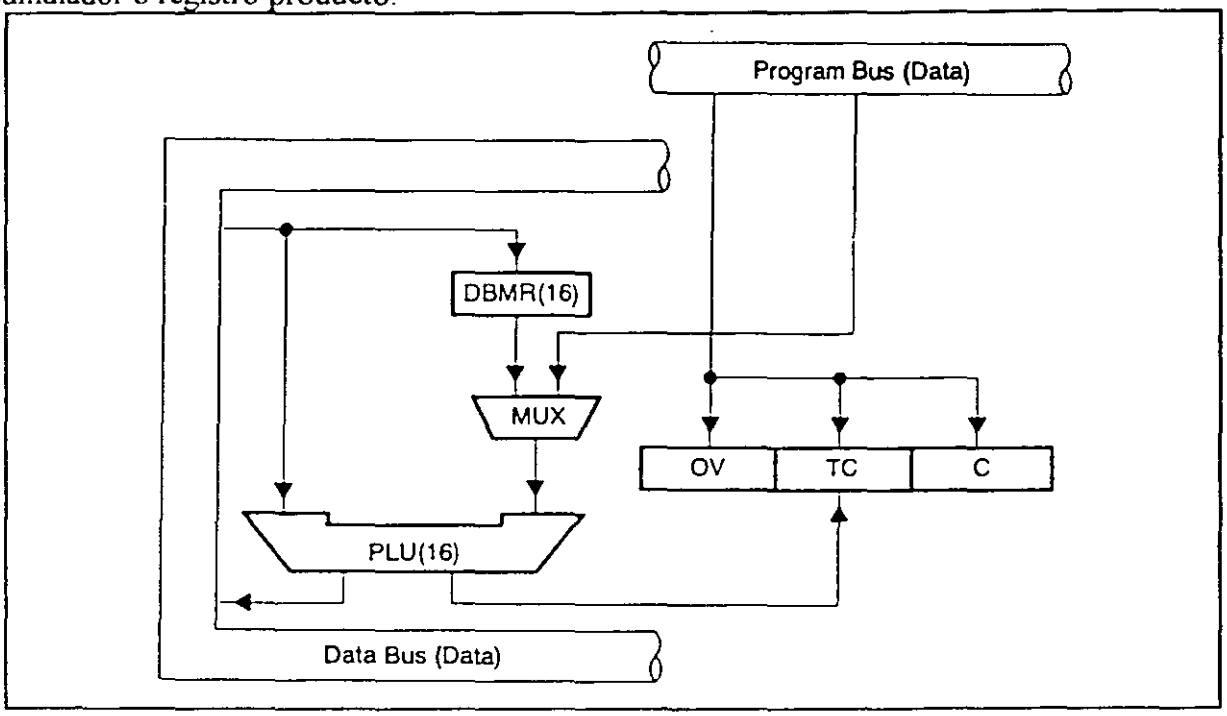
Facultad de Ingeniería

UNAM

Junio del 2000

UNIDAD LOGICA PARALELA

La unidad lógica paralela (PLU) efectúa operaciones lógicas independientemente de la unidad ALU y ARAU, es decir, que no utiliza el acumulador para efectuar las operaciones. Ésta puede fijar, limpiar o cambiar múltiples bits de los registros de control o de estado o de cualquier localidad en memoria dato. La unidad PLU efectúa operaciones lógicas directas sin afectar el contenido del acumulador o registro producto.



Unidad PLU

En las operaciones de la unidad PLU un operando es buscado en memoria dato y el otro proviene del operando inmediato en la instrucción, al efectuar la operación lógica entre los operando el resultado es escrito de nuevo en la localización de la memoria dato direccionado, normalmente todas estas operaciones se efectúan en un ciclo de instrucción

Esta unidad permite la prueba de bits individuales con la intrucción BIT, o comparación de una constante con una localidad de memoria o comparación dinámica utilizando el registro de manipulación dinámica (DBMR), si las cantidades a comparar son iguales entonces el bit de estado TC se pone a uno.

Ejemplos:

```
APL  #000Fh,x      ; efectúa operación AND entre la constante 000Fh y el dato x
OPL  #0FF0h,y      ; efectúa operación OR entre la constante 000Fh y el dato y
XPL  #01,*         ; cambia el bit 0 del dato apuntado por el reg. ARi en uso
SPLK #0ABh,dato    ; almacena la constante 0ABh en variable dato.
```

INTERRUPCIONES

Por: Larry Escobar Salguero.

Facultad de Ingeniería

UNAM

Junio del 2000

INTERRUPCIONES

Las interrupciones son un recurso para suspender la actividad del TMS50 (o un microprocesador) para atender algún requerimiento por hardware o software, esto evita la necesidad de estar encuestando por software un evento externo. Las interrupciones típicas son generadas por dispositivos externos que quieren transferir información al TMS50, tal el es el caso de convertidores A/D y D/A. El TMS50 tiene cuatro interrupciones externas mascarables (/INT4, /INT3, /INT2 e /INT1), disponibles para dispositivos externos que interrumpan al microprocesador. Dos interrupciones internas que son generadas por el puerto serie (RINT y XINT), una por el timer (TINT) y una por software a través de la instrucción TRAP. Aunque esta última no es priorizable incluye su vector de interrupción. Cada dirección de interrupcción consta de dos localidades para ubicar una instrucción de salto y la dirección de la subrutina de atención de interrupciones.

Tabla de vectores de interrupción

Las fuentes de interrupción del TMS50 son seis externas, cinco internas, una de TRAP y por software:

Interrupción	Dirección	Descripcción
RESET	0h	Reset externo
/INT1	2h	Interrupcción externa de usuario N. 1
/INT2	4h	Interrupcción externa de usuario N. 2
/INT3	6h	Interrupcción externa de usuario N. 3
TINT	8h	Interrupcción interna de timer
RINT	Ah	Interrupcción de recepción del puerto serie.
XINT	Ch	Interrupcción de transmisión del puerto serie.
TRNT	Eh	Interrupcción de recepción del puerto TDM.
TXNT	10h	Interrupcción de transmisión del puerto TDM.
/INT4	12h	Interrupcción externa de usuario N. 4
	14-21 h	Reservado.
TRAP	22h	Vector de instrucción TRAP
NMI	24h	Interrupcción externa no mascarable.

Las interrupcciones externas son activadas por una señal en nivel bajo al menos tres ciclos en el respectivo pin externo.

Las interrupciones internas son generadas por eventos internos del TMS50.

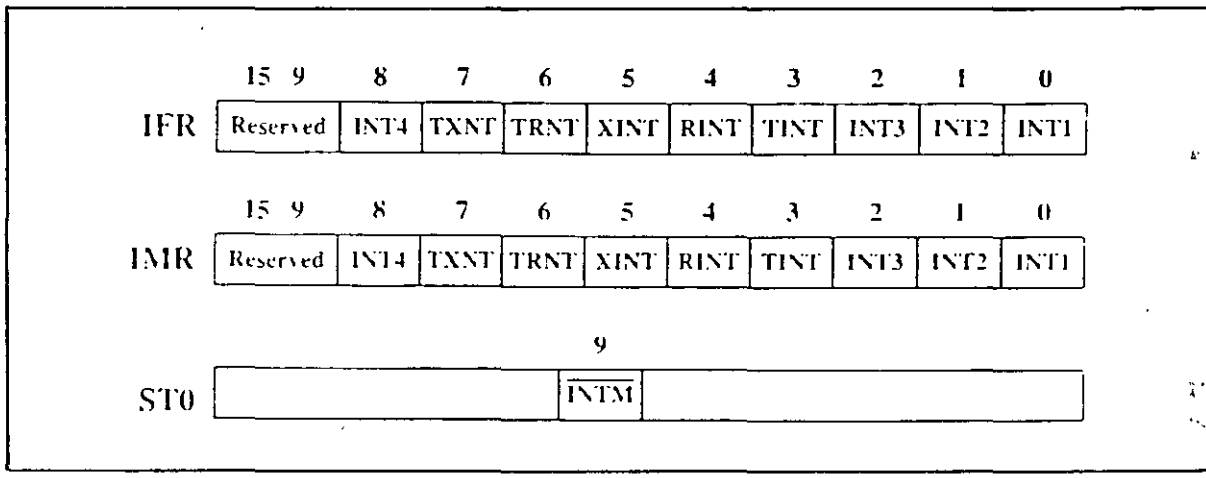
Reset (/RS)

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

La señal /RS causa que el TMS50 termine la ejecución y obligue al contador de programa ir a cero afectando varios registros y los bits de estado. Para la correcta operación del sistema, la señal de reset debe ser activada 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.

Operación de las interrupciones

Cuando una interrupción ocurre, ésta es almacenada en un Registro de Bandera de Interrupción de 16 bits (IFR). Este registro es fijado por el uso de interrupciones externas /INT(4-1) y las interrupciones RINT, XINT y TINT. Cada interrupción es almacenada en IFR hasta que ésta es reconocida y automáticamente borrada por el Reconocimiento de Interrupciones (/IACK), o el reset (/RS), o se escribe un uno en el bit correspondiente en el registro IFR. El reset no es almacenado en IFR.



Registro de banderas de interrupcion IFR e IMR

El TMS50 tiene una mapa de memoria para Registro de Interrupciones Mascarables (IMR) para enmascarar las interrupciones internas y externas. Un uno en cualquiera de los bits 0-15 del registro IMR habilita la correspondiente interrupción siempre que el bit de estado INTM esté en cero (habilita interrupciones mascarables) . Obviamente las interrupciones NMI y /RS no están incluidas en el registro IMR.

El bit /INTM en el registro de status ST0 habilita o deshabilita todas las interrupciones mascarables (INTM=0 habilitado y INTM=1 deshabilitado). INTM es fijado a uno por la señal /IACK.Si una interrupción ocurre durante un ciclo múltiple de instrucciones, la interrupción no puede ser procesada hasta que la instrucción es 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 RPT, la interrupción es almacenada en IFR hasta que el ciclo de repetición sea terminado, después la interrupción es procesada. Las interrupciones no pueden ser procesadas entre la instrucción CLRC INTM y la próxima instrucción en la secuencia del programa. También una interrupción no es reconocida cuando el /HOLD está activado.

La instrucción RETE permite terminar una rutina de atención de interrupción, reestablecer los registros principales salvados y habilitar interrupciones (INTM = 0) .

Salvado de registros

Cuando una interrupción es ejecutada ciertos registros estratégicos son salvados automáticamente, y en el retorno de la interrupción (con instrucciones RETE y RETI) estos registros son reestablecidos. En el TMS50 existe una región llamada de registros shadow donde se almacenan los registros: ACC, ACCB, PREG, STO, ST1, PMST, TREG0, TREG1, TREG2, INDX y ARCR. Esto permite salvar el constexto de registros que está trabajando el programa al llegar una interrupción.

Implementación de Stack por software

Cuando se anidan interrupciones o subrutinas más allá de los ocho niveles, entonces se puede implementar un stack por software por medio de la instrucciones POPD y PSHD, permintiendo a la parte alta del stack (TOS) sea salvado o recuperado de memoria dato. La intrucción POPD salva el contenido del PC en TOS en memoria dato y los siete valores bajos del stack son recorridos una localidad hacia arriba, la intrucción PSHD pone un valor de la memoria dato en el TOS del stack y los demás valores en el stack son recorridos hacia bajo una localidad de memoria (el valor más bajo del stack es perdido). Esto implica que el stack puede ser expandido en memoria dato.

Interrupciones por software

La instrucción de interrupciones por software (INTR #k) permite efectuar hasta 32 (k de 0 a 31) interrupciones por software con las mismas características de una interrupción por hardware. El programador sólo invoca la rutina de atención de interrupción de acuerdo a la tabla siguiente, lo importante de esta forma de interrupción es que el usuario puede definir sus propias rutinas de antección de interrupción.

Interrupciones por software

K	Interrupción	Localización	K	Interrupción	Localización
0	RESET	0h	16	Reservado	20h
1	/INT1	2h	17	TRAP	22h
2	/INT2	4h	18	NMI	24h
3	/INT3	6h	19	Reservado	26h
4	TINT	8h	20	De usuario	28h
5	RINT	Ah	21	De usuario	2Ah
6	XINT	Ch	22	De usuario	2Ch
7	TRNT	Eh	23	De usuario	2Eh
8	TXNT	10h	24	De usuario	30h
9	/INT4	12h	25	De usuario	32h
10	Reservado	14h	26	De usuario	34h
11	Reservado	16h	27	De usuario	36h
12	Reservado	18h	28	De usuario	38h
13	Reservado	1Ah	29	De usuario	3Ah
14	Reservado	1Ch	30	De usuario	3Ch
15	Reservado	1Eh	31	De usuario	3Eh

PERIFERICOS Y PUERTO SERIE

Por: Larry Escobar Salguero.

Facultad de Ingeniería

UNAM

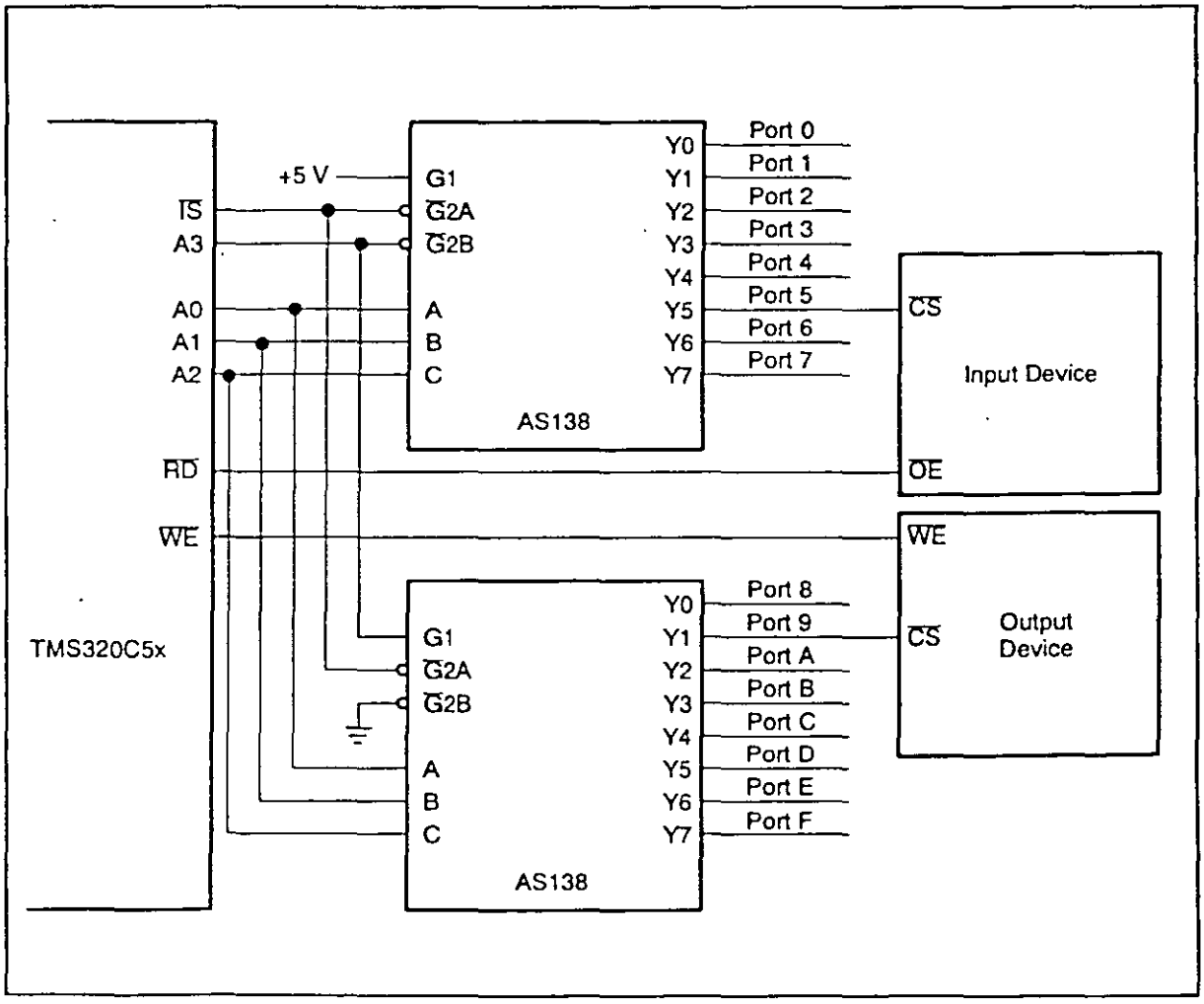
Junio del 2000

PERIFERICOS Y PUERTO SERIE

Los periféricos que maneja el TMS50 son controlados por algunos registros mapeados, para la inicialización de estos periféricos en sus registros deben de escribirse las palabras necesarias para que operen correctamente. Dentro de estos periféricos están el puerto serie (transmisión / recepción), el puerto serie multiplezado por división (TDM), 64 k-puertos paralelos y el timer.

PUERTOS PARALELOS DE ENTRADA SALIDA

El TMS50 puede direccionar hasta 64k puertos de entrada salida (I/O), éstos son seleccionados cuando la señal externa /IS es activada en bajo cuando el TMS50 ejecuta alguna instrucción IN u OUT. La señal /RD y /WE se pueden utilizar en conjunto con /IS para accesar a dispositivos externos tal como se ve en la figura siguiente:



Interfaz a puertos paralelos I/O

GENERACION DE ESTADOS DE ESPERA POR SOFTWARE

El TMS50 es un procesador de señales muy rápido, sin embargo, brinda la posibilidad de acceder a dispositivos lentos al generar hasta 7 estados de espera por medio de la línea externa READY, pero aún más estos estados de espera se pueden generar por software evitando hardware externo.

La programación por software de los estados de espera puede ser controlado por dos registros generadores de estado de espera (PDWSR para dato y programa y IOWSR para I/O) y un registro de control de 5 bits (CWSR).

Register	Bits	Space	Address Range	
PDWSR	0–1	Program	0000h–3FFFh	
	2–3		4000h–7FFFh	
	4–5		8000h–0BFFFh	
	6–7		0C000h–0FFFFh	
	8–9	Data	0000h–3FFFh	
	10–11		4000h–7FFFh	
	12–13		8000h–0BFFFh	
	14–15		0C000h–0FFFFh	
IOWSR		I/O	BIG = 0	BIG = 1
	0–1		Port 0/1, Port 10/11, etc.	0000h–1FFFh
	2–3		Port 2/3, Port 12/13, etc	2000h–3FFFh
	4–5		Port 4/5, Port 14/15, etc.	4000h–5FFFh
	6–7		Port 6/7, Port 16/17, etc	6000h–7FFFh
	8–9		Port 8/9, Port 18/19, etc.	8000h–9FFFh
	10–11		Port 0A/0B, Port 1A/1B, etc	0A000h–0BFFFh
	12–13		Port 0C/0D, Port 1C/1D, etc	0C000h–0DFFFh
	14–15		Port 0E/0F, Port 1E/1F, etc.	0E000h–0FFFFh

Generación de estados de espera

El mapa de memoria para dato y programa se divide en 8 bloques de 16k palabras y se le asocia en el registro PDWSR dos bits a cada bloque, es decir, que a cada bloque se le puede programar su propio estado de espera independientemente de los otros bloques. El espacio para I/O puede configurarse en dos formas dependiendo como se especifique el bit BIG en el registro CWSR. Si BIG=0, se mapean ocho pares de periféricos I/O por cada dos bits en IOWSR, si BIG=1, ocho bloques de 8 K palabras y a cada bloque se le puede programar su estado de espera independiente.

Cuatro bits en el registro CWSR permiten al usuario seleccionar uno de dos mapas de estados de espera y el número de estados de espera correspondientes (de 0 a 7 estados). Los valores de los bits se escriben tanto en los registros PDWSR, IOWSR y CWSR como se especifica a continuación:

Wait-State Field† of PDWSR or IOWSR (Binary Value)	No. of Wait States (CWSR Bit n = 0)	No. of Wait States (CWSR Bit n = 1)
00	0	0
01	1	1
10	2	3
11	3	7

n (Bit Position In CWSR)	Space
0	Program
1	Data
2	I/O (lower-half: Port 0–Port 7 if BIG=0, 0000h–7FFFh if BIG=1)
3	I/O (upper-half. Port 8–Port F if BIG=0, 8000h–0FFFFh if BIG=1)
4	BIG mode bit

Registros PDWSR, IOWSR y CWSR

El registro CWSR debe escribirse antes que los registros PDWSR e IOWSR.

PUERTO SERIE

El TMS50 consta de un puerto serie bidireccional full duplex, este puerto provee una comunicación directa a dispositivos tales como codecs, convertidores seriales A/D, otros sistemas seriales y para aplicaciones multiprocesos a través del puerto serial TDM.

Las señales del puerto serie son compatibles con codecs y otros dispositivos seriales. La máxima frecuencia de operación del puerto serie cuando utiliza el reloj interno es de 5 Mbits/s a 50 ns y 7.14 Mbits/s a 35 ns.

Para su operación el TMS50 contiene los pines externos:

CLKX	Señal de reloj de transmisión.
CLKR	Señal de reloj de recepción.
DX	Señal de transmisión de dato serial, envía el dato actual
DR	Señal de recepción de dato serial, recibe el dato actual.
FSX	Señal de sincronización de frame de transmisión, inicia la transferencia de un frame.
FSR	Señal de sincronización de frame de recepción.

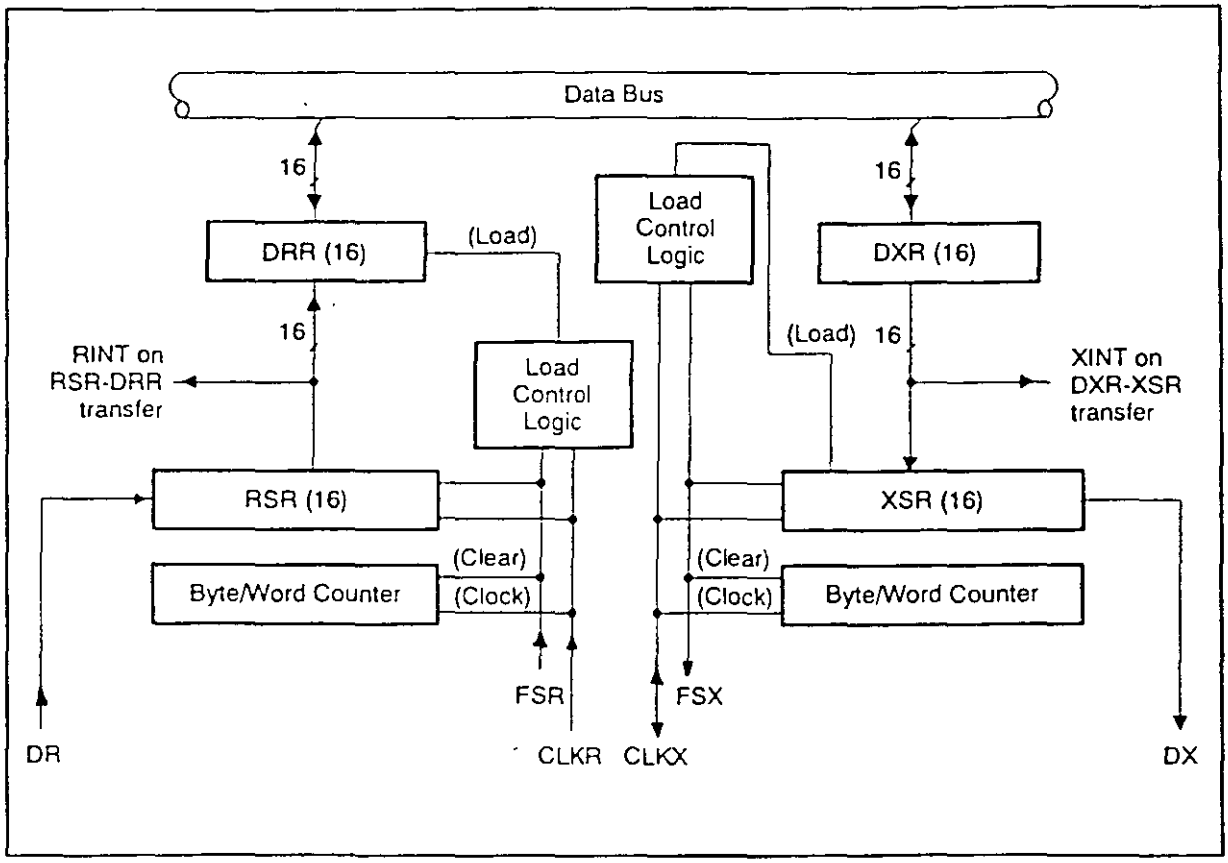


Diagrama de bloques del puerto serie

Para la operación del puerto serie se utilizan los registros:

SPC	Registro de control del puerto serie.
DXR	Registro de transmisión de datos.
DRR	Registro de recepción de datos.
XSR	Registro de corrimiento en la transmisión.
RSR	Registro de corrimiento en la recepción.

Un dato a transmitir es escrito en el registro DXR el cual copia el dato en el registro XSR que efectúa el corrimiento de bits para hacer la transmisión serial sobre el pin DX, tan pronto como se copia DXR a XSR se permite escribir otro dato en el registro DXR, a la vez ocurre una transición 0-1 en el bit de transmisión lista (XRDY bit 11) del registro de control SPC y se genera una interrupción de transmisión de puerto serie (XINT).

Para la recepción un dato recibido en el pin DR es corrido dentro del registro RSR, el cual es copiado en el registro de recepción de dato (DRR) y el dato puede ser leído. Una vez completada la copia de RSR a DRR existe una transición 0-1 en el bit de recepción lista (RRDY bit 10) del registro de control SPC y a la vez se genera una interrupción de recepción de puerto serie (RINT). El puerto serie es doblemente buffereado porque los datos puede ser transferidos de DXR o DRR mientras otra transmisión o recepción se efectúa.

El registro de control SPC es un registro mapeado y se describe a continuación:

Bit	Name	Description
0		Reserved
1	DLB	Digital Loopback
2	FO	Word length 0 = 16 bits 1 = 8 bits
3	FSM	Frame sync mode: 0 = continuous mode 1 = one sync pulse/word
4	MCM	CLKX source: 0 = CLKX 1 = CLKOUT1/4
5	TXM	0 = FSX is an input 1 = FSX is an output
6	XRST	0 = reset transmiter 1 = operate transmitter

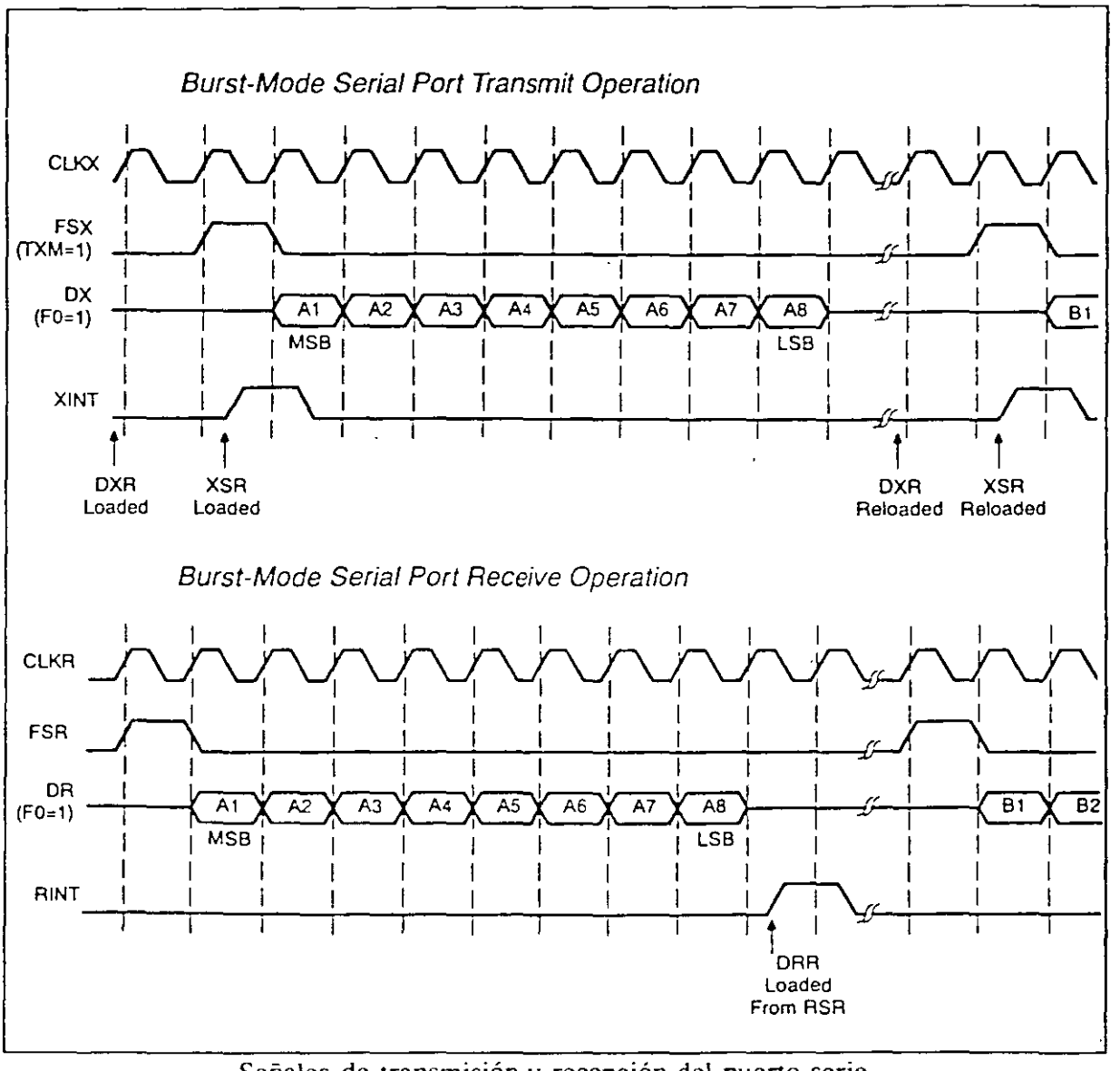
Bit	Name	Description
7	RRST	0 = reset receiver 1 = operate receiver
8	IN0	Input level at CLKR
9	IN1	Input level at CLKX
10	RRDY	1 = receive data ready
11	XRDY	1 = transmitter ready
12	XSREMPTY	0 = transmiter underrun
13	RSRFULL	1 = receiver overflow
14 15	SOFT FREE	Determine state of serial port when a breakpoint is encountered in the HLL Debugger

FREE	SOFT	Action
1	x	Free run
0	0	Immediate stop
0	1	Stop after completing word

Registro de control de puerto serie (SPC)

Para la configuración del puerto serie es necesario escribir a los bits 1-5 del registro SPC, sin embargo, antes de escribir a estos bits hay que efectuar un reset sobre los bits 6 y 7 (XRST y RRST), es decir, escribir ceros a estos bits y luego escribir a los bits 1-5, para que la configuración del puerto quede habilitada se ponen los bits 6 y 7 a unos. Los bits 10-13 del registros SPC son bits de estado del puerto serie.

En la siguiente figura se muestran los diagramas de tiempo de las señales del puerto serie, tanto para la operación de transmisión como de recepción.



Señales de transmisión y recepción del puerto serie

EL TEMPORIZADOR (TIMER)

El timer es un contador interno del TMS50 que puede utilizarse para generar interrupciones periódicas con tiempos exactos.

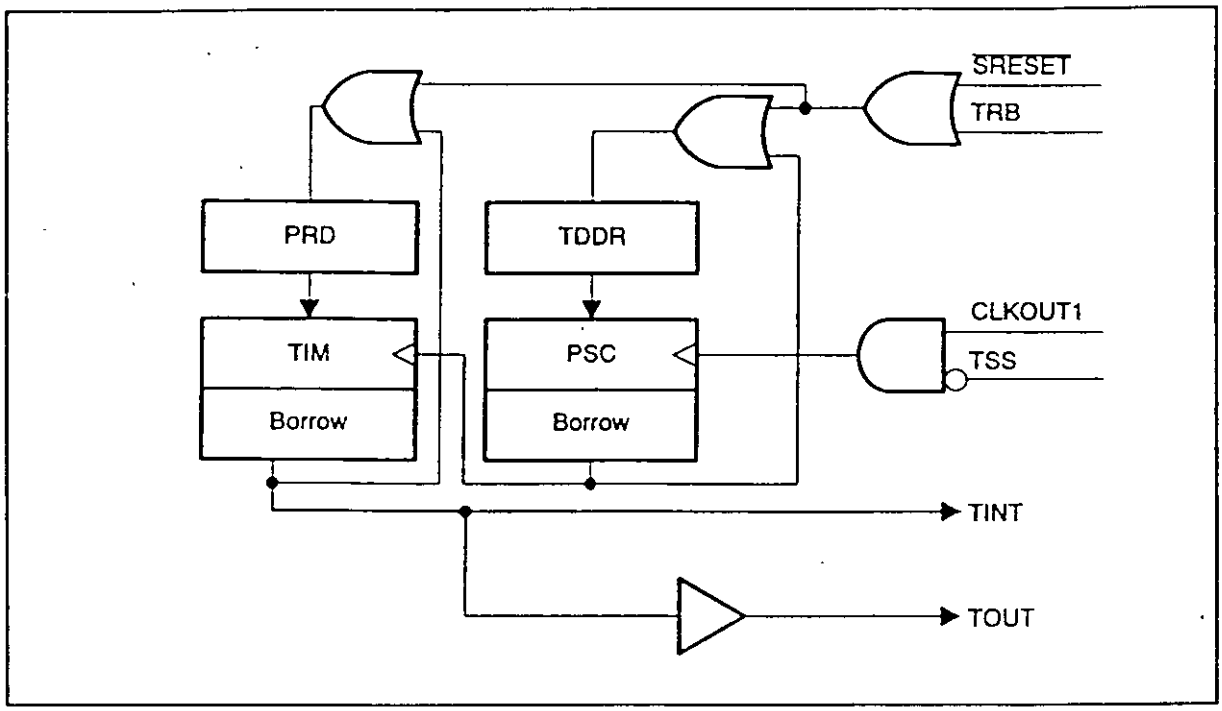


Diagrama de bloques de Timer

El TMS50 posee un Registro Timer (TIM) y un Registro Período (PRD) de 16 bits. El registro timer cuenta descendentemente en la caída de CLKOUT1. Los registros TIM y PRD son fijados a su máximo valor FFFFh en el reset. Una interrupción de 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 muestreo o escritura de periféricos.

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

La razón de tiempo de interrupción del timer está dada por:

$$f_{TINT} = \frac{1}{t_{c(C)}\,(TDDR + 1)\,(PDR + 1)}$$

donde:

f_{TINT} : razón de interrupción de timer
$t_{c(C)}$: período del reloj CLKOUT1
TDDR + 1 : escalamiento de 4 bits
PRD + 1 : escalamiento de 16 bits

El primer contador (descendente) TDDR se encuentra en el registro de control de timer (TCR bits 0-3), los bits 6-9 del registro TCR corresponden al contador preescalador. Cuando el registro TCR llega a cero es cargado con el valor del registro TDDR.

La operación del timer es controlada por el registro TCR, el bit TSS permite detener la operación del timer al escribirle un uno, el bit TRB permite cargar de nuevo el período.

Registro de control del Timer (TCR)

Bit	Nombre	Descripción
0-3	TDDR	Razón de división del Timer
4	TSS	Detiene al Timer =1, Reinicializa al Timer =0
5	TRB	Reinicia el Timer con el PRD y TDDR
6-9	PSC	Contador preescalador

15-12	11	10	9-6	5	4	3-0
Reservado	SOFT	FREE	PSC	TRB	TSS	TDDR

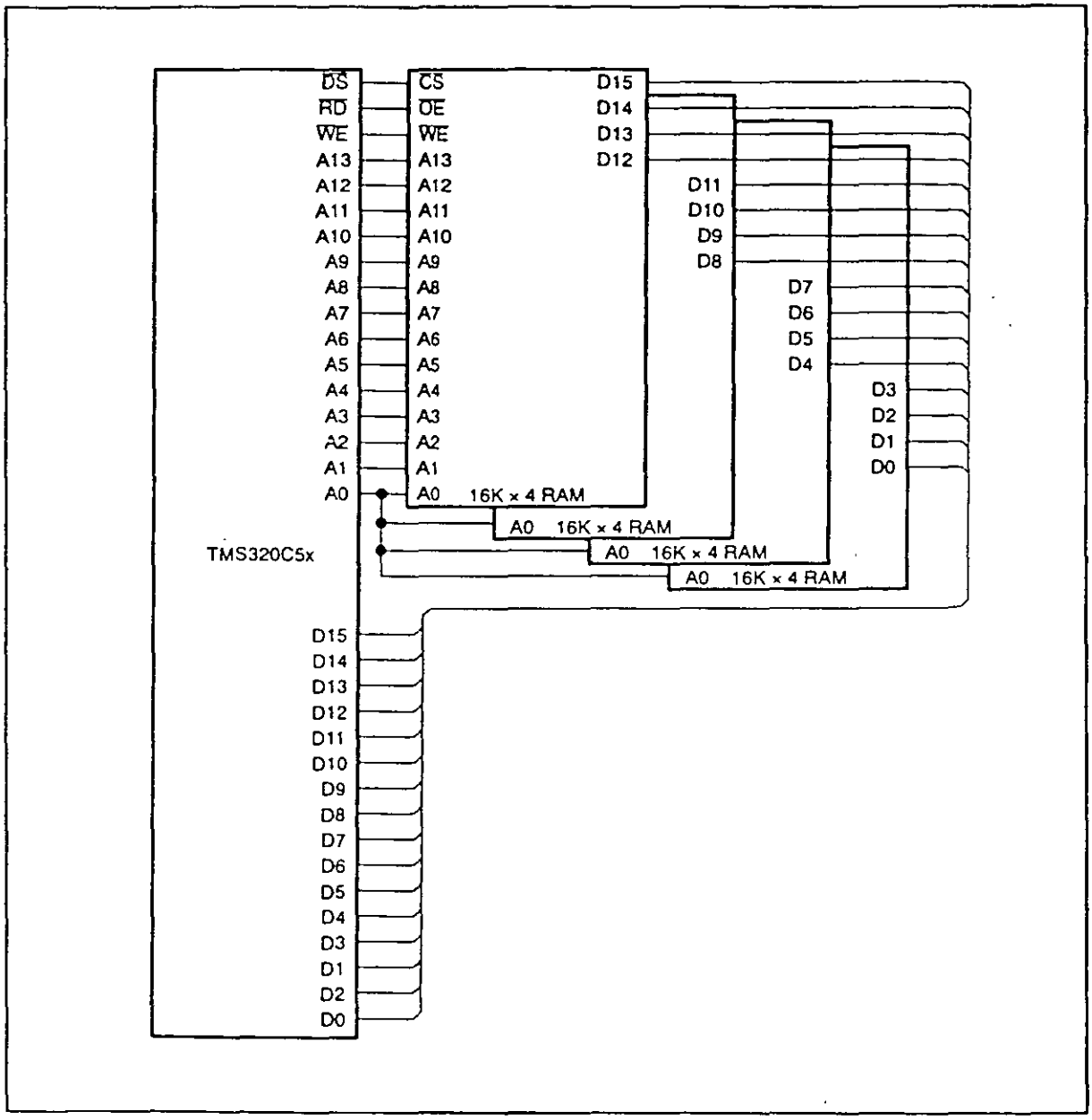
Los bits SOFT y FREE determinan el estado del timer cuando éste es detenido.

La secuencia para utilización del timer sería:

- escribir los valores necesarios al registro PRD y el TDDR al registro TCR .
- habilitar la interrupción del timer (desenmascararla) en el registro IMR.
- limpiar cualquier interrupción pendiente en el registro IFR.
- Habilitar interrupciones limpiando el bit de estado INTM.

INTERFACE A MEMORIA RAM EXTERNA

En la siguiente figura se observa la conexión directa del TMS50 a una memoria RAM, el TMS50 provee las señales necesarias para leer y escribir en la memoria RAM.



ESPACIO DE ENTRADA SALIDA (I/O)

El TMS50 puede direccionar hasta 64 k puertos I/O de 16 bits permitiendo el acceso a periféricos utilizados en aplicaciones para el PDS.

El acceso a puertos es multiplexado sobre el mismo bus de direcciones y datos utilizados para accesar a memoria programa y dato, el espacio I/O es distinguido por la activación de la señal externa /IS en bajo cuando el TMS50 ejecuta una instrucción de OUT o IN:

```
IN    DATO,01FFh ;     lee un dato del puerto 1FFh y lo escribe en la localidad de
                       memoria DATO
OUT   SALIDA,01F1h;    escribe un dato de la localidad e memoria SALIDA y lo
                       escribe en el puerto 1F1h.
```

De los 64 k puertos 16 son mapeados en el espacio de memoria dato (PA0 a PA15 de 50h a 5Fh), estos puertos son tratados como memoria, sin embargo externamente son distinguidos por la activación de la señal /IS. Estos puertos pueden accesarse con instrucciones que se utilizan para direccionar memoria:

```
SACL  52h          ; salva el acumulador al puerto externo 52h (DP = 0)
```

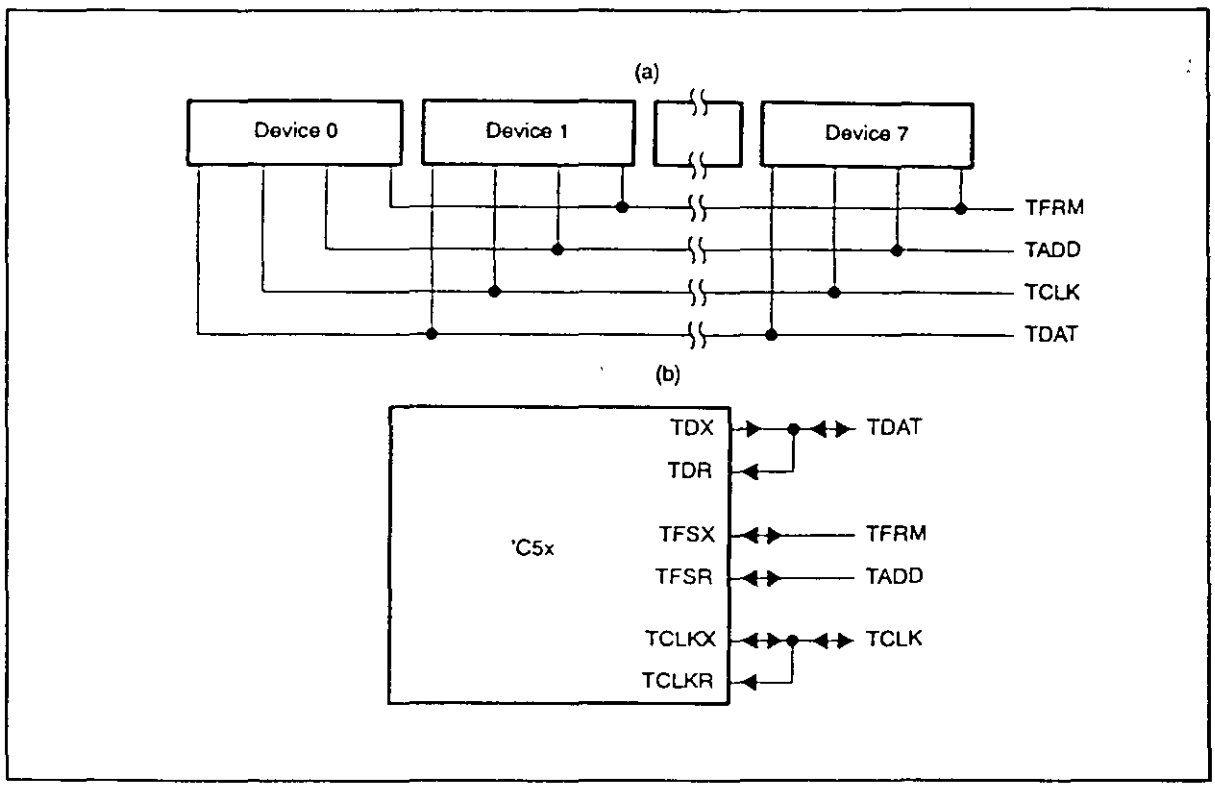
Las señales /RD y /WE pueden utilizarse para seleccionar un dispositivo como puerto de entrada o salida. También hay que recordar que a los puertos se les puede programar estados de espera por software.

PUERTO SERIE TDM

El TMS50 tiene un puerto serie multiplexado por división de tiempo (TDM) que le permite la comunicación serial con otros siete TMS50 proveyendo una poderosa interface para aplicaciones multiprocesos.

Para la operación del puerto serie TDM se pone un uno en el bit TDM del registro de configuración de puerto serie TDM (TSPC). Si el bit TDM = 0, el puerto TDM puede operar como otro puerto serie convencional del TMS50. El registro TSPC es similar al registro SPC del puerto serie a excepción de que el bit TDM indica la operación de este puerto.

El concepto de división de tiempo es que cada dispositivo toma una parte del tiempo disponible a compartir. El puerto TDM requiere juntar las líneas de transmisión de datos (TDX) y de recepción (TRD) en una línea simple llamada línea de datos (TDAT), para que los datos fluyan sobre esta línea. Similarmente los relojes se unen en una línea de reloj TDM. Una señal simple de frame es utilizada para indicar el inicio de un evento de 8 palabras TDM. Hasta ocho TMS50 pueden conectarse a cuatro líneas para la comunicación por el puerto TDM.



Conexión del TMS50 por puerto TDM

La línea TADD es manejada por un TMS50 particular para un tiempo particular y determina qué dispositivo de la conexión TDM puede efectuar una recepción válida en el tiempo de slot.

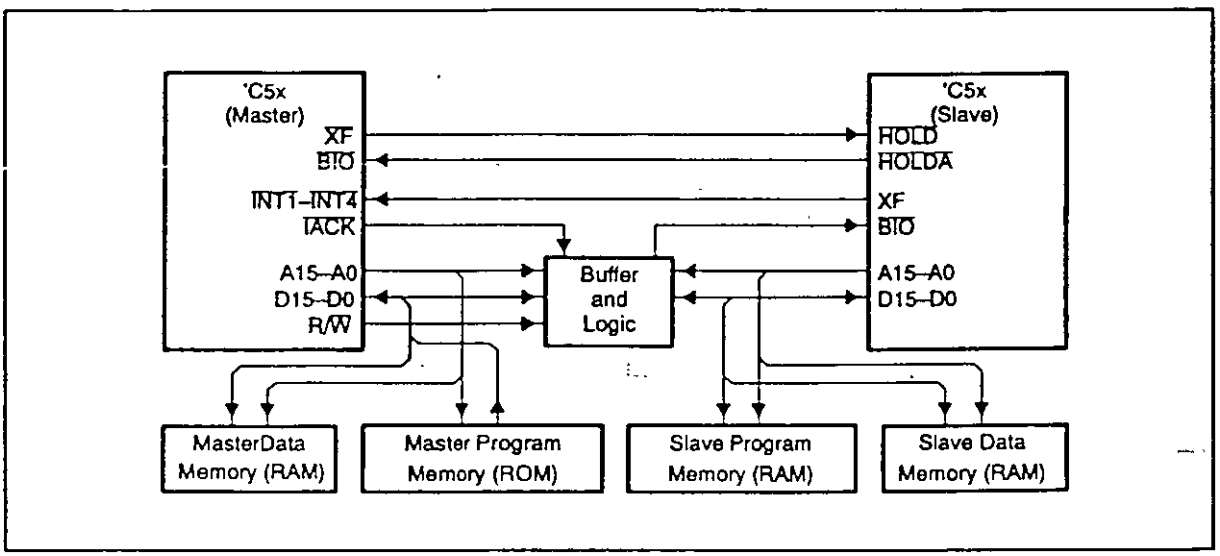
El puerto TDM tiene asociados seis registros mapeados en memoria. En una lectura válida TDM, el dato es transferido del registro TRSR al registro TRCV y una interrupción de recepción es generada indicando que el registro TRCV tiene un dato válido y que puede ser leído. Todos los puertos TDM operan sincronizados por las líneas TCLK y TFRM las cuales son generadas por cada dispositivo.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TRCV	Receive Data															
TDXR	Transmit Data															
TSPC	FREE	SOFT	X	X	XRDY	RRDY	IN1	IN0	RRST	XRST	TXM	MCM	FSM	FO	DLB	TDM
TCSR	X	X	X	X	X	X	X	X	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
TRTA	TA7	TA6	TA5	TA4	TA3	TA2	TA1	RA0	RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0
TRAD	X	X	X2	X1	X0	S2	S1	S0	A7	A6	A5	A4	A3	A2	A1	A0

Registros del puerto TDM

ACCESO DIRECTO A MEMORIA

El TMS50 soporta multiprocesamiento utilizando Acceso Directo de Memoria (DMA). La característica del DMA es que puede utilizarse para multiprocesamiento deteniendo temporalmente la ejecución de uno o más procesos que permita a otro proceso leer o escribir a la memoria externa o interna de otro TMS50.



Configuración Maestro-esclavo utilizando DMA

Las dos señales /HOLD y /HOLDA permiten a otros dispositivos tomar el control de los buses del TMS50 esclavo. Cuando el TMS50 recibe una señal de /HOLD de un dispositivo externo, el TMS50 envía una señal de reconocimiento /HOLDA (activa baja), entonces pone sus buses de direcciones, datos y las señales de control (/PS, /DS, /IS, R/w y /STRB) en alta impedancia. Los pines de puerto serie (DX y FSX) no son afectados. /HOLD es muestreado en el tercer cuarto de ciclo de reloj, si es encontrado activo, éste toma tres ciclos de máquina antes que los buses y señales se vayan a alta impedancia, permitiendo terminar la instrucción que está siendo ejecutada.

Esto significa que el TMS50 maestro toma el control completo de la memoria externa del TMS50 esclavo, la señal de /HOLDA es encuestada en el pin /BIO del TMS50 maestro. La señal externa de salida XF del esclavo puede utilizarse para interrumpir al maestro indicándole la finalización de alguna tarea o el requerimiento de información adicional para seguir trabajando.

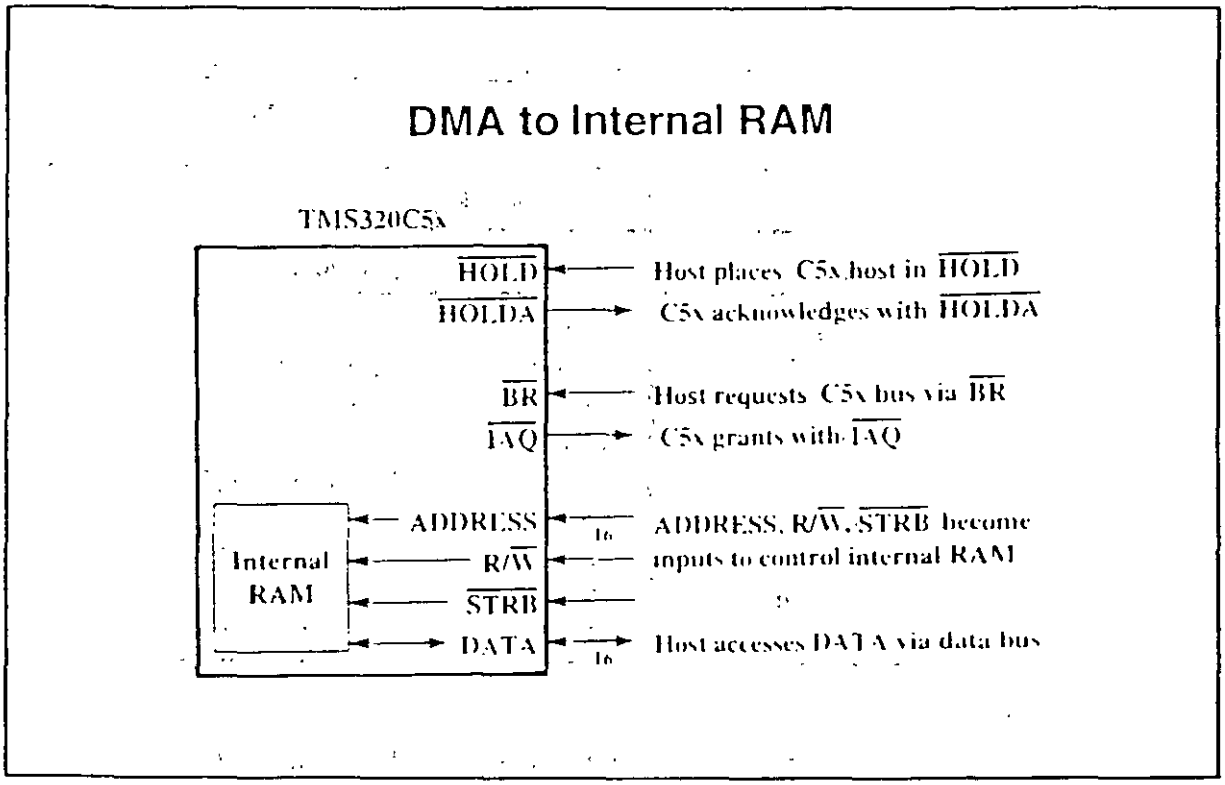
El modo de operación es seleccionada por HM (modo hold) en el registro de estado. Cuando HM=1, el TMS50 detiene la ejecución del programa e ingresa al estado de hold directamente. Cuando HM = 0 el procesador puede continuar la ejecución del programa en memoria interna pero pone la interface externa en alta impedancia. Si el programa en ejecución es de memoria interna y no se accesa dato externo de memoria, el procesador entra a estado externo de hold, pero el programa

sigue ejecutándose internamente (si HM = 0). Esto permite una operación más eficiente del sistema, dado que puede continuarse la ejecución mientras que una operación de DMA está siendo ejecutada.

Todas la interrupciones son deshabilitadas mientras que /HOLD es activada con HM = 1. Si HM=0 las interrupciones funcionan normalmente.

El dispositivo maestro puede accesar las localidades internas del TMS50 (esclavo), cuando el esclavo envía el reconocimiento de /HOLDA, el maestro puede enviar una señal baja a la entrada /BR del TMS50 esclavo, éste último responderá con la señal /IAQ reconociendo el acceso a su memoria interna. La dirección de transferencia es manejada por señal R/w ; por medio del maestro. Con la señal /STRB se selecciona el acceso a memoria, es decir que esta señal determina la duración del acceso a memoria.

La finalización de la transferencia DMA se da cuando se desactiva la señal /HOLD (se pone en alto) y en seguida también se desactiva la señal de /HOLDA.



Señales de DMA

MEMORIA GLOBAL

Para aplicaciones de multiprocesamiento, el TMS50 tiene la capacidad de localizar un espacio de memoria global de datos y comunicarse con este espacio a través de la señal /BR (requerimiento de bus) y la señal de control READY.

La memoria global es compartida por más de un procesador, por lo tanto para 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. A diferencia del DMA, la lectura o escritura de memoria global no requiere que uno de los procesadores esté detenido.

El Registro de Mapeo de Memoria Global (GREG) especifica la parte de memoria de datos del TMS50 como memoria global externa. El registro GREG es mapeado como memoria de datos en la localización 5h, es un registro de 8 bit 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 es indicado en la tabla de configuración de memoria global:

Valor de GREG	LOCALIZACION DE MEMORIA		MEMORIA GLOBAL	
	Intervalo	No. de Palbras	Invervalo	No. de Palabras
000000xx	0-FFFFh	65,536	–	O
10000000	0-7FFFh	32,768	8000-FFFFh	32,768
11000000	0-BFFFh	49,152	C000-FFFFh	16,384
11100000	0-DFFFh	57,344	E000-FFFFh	8,192
11110000	0-EFFFh	61,440	F000-FFFFh	4,096
11111000	0-F7FFh	63,488	F800-FFFFh	2,048
11111100	0-FBFFh	64,512	FC00-FFFFh	1,024
11111110	0-FDFFh	65,024	FE000-FFFFh	512
11111111	0-FEFFh	65,280	FF000-FFFFh	256

Se observa que se puede compartir memoria global desde 256 hasta 32,768 palabras.
Cuando un dato de memoria es direccionado en modo directa o indirectamente correspondiente a direccionamiento de memoria global (definido por GREG), /BR es acertado bajo con /DS para indicar al procesador que se desea hacer una acceso de memoria global. La lógica externa arbitra el control de memoria global acertando una señal READY controlando el tiempo de acceso a la memoria. También se pueden generar estados de espera por software como ya se especificó. La memoria global es compartida por varios procesadores pero es accesada individualmente sólo por un procesador. Las señales /BR y /DS son utilizadas para habilitar el espacio de memoria global.

BIBLIOGRAFIA

- Edward A. Lee. Arquitectura programable DSP. IEEE ASSP MAGAZINE, octubre de 1988 y enero 1989.
- Alcántara Silva Rogelio. Introducción al procesamiento digital de señales. Septiembre 1989.
- Texas Instruments. Digital Signal Processing Applications with the TMS320 Family, Theory, Algorithms, and implementations, vol.1,2 y 3. USA 1990.
- Texas Instruments. TMS30C1x, User's Guide. USA 1990.
- Texas Instruments. TMS30C2x, User's Guide. USA 1991.
- Texas Instruments. TMS30C3x, User's Guide. USA 1992.
- Texas Instruments. TMS320C5x, User's Guide. USA 1997.
- Texas Instruments. TMS320C5x, General-purpose applications user's guide. USA 1997.
- Texas Instruments. TMS320C5x DSP starter Kit. User's Guide. USA 1994.
- Escobar S. L. y Rodríguez R. Diseño y construcción de una Arquitectura para Procesamiento Digital de Imágenes. Facultad de Ingeniería, UNAM, 1990. Tesis de Licenciatura.
- Martínez J., Escobar S. L. y Rodríguez R. Arquitectura para Procesamiento Digital de Imágenes. Congreso de Electrónica. Instituto Tecnológico de Chihuahua. Chihuahua, México, octubre de 1993
- Psenicka B. y Escobar S. L. Procesamiento Digital de Señales, Segunda parte, Microcontroladores y realización de los filtros digitales con TMS320Cxx. Facultad de Ingeniería, UNAM, julio de 1998.
- Escobar S. Larry. Manual del laboratorio de procesamiento digital de señales. Facultad de Ingeniería, UNAM, abril del 2000.
- Alcántara S. R. & Escobar S. L. Dynamic Range and Scaling Evaluation in Adaptive Filtering Algorithms for DSP fixed-point implementation. International Symposium on Information Theory and its Applications (ISITA98). México, D.F., oct. 14-16, 1998.
- Alcántara S. R. & Escobar S. L. A comparative Filtering Algorithms TMS DSP Fixed-Point Implementation of FRLS Adaptive Filtering Algorithms Family. The International Conference on Signal Processing Applications and Technology (ICSPAT99). Nov. 1-4, 1999. Orlando, Florida, U.S.A.
- Escobar S. L. & Alcántara S. R. Fixed Point Arithmetic using Digital Signal Processors. International Conference on Telecommunications, ICT 2000. 22-25 may 2000, Acapulco, México
- Escobar S. L Algoritmos de filtrado adpatable: implementación, evaluación, comparación y aplicaciones en telecomunicaciones. Tesis de maestría en Ingeniería Eléctrica Facultad de Ingeniería, UNAM, nov. 1997.