Facultad de Ingeniería

UNAM

Manual del Laboratorio de Procesamiento Digital de Señales

Por
Larry Escobar Salguero

Enero del 2000

Índice general

Índice de figuras

Introducción

En la actualidad con los avances tecnológicos, 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. Los fundamentos del PDS están dados por la teoría de señales y sistemas, sin embargo, las arquitecturas para PDS conforma una parte muy importante en el PDS en el sentido de verificación algorítmica software-hardware, en la solución a un problema concreto 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 elementales 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 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.

Estas notas esán organizadas de la siguiente forma: en la *primera parte* de estos apuntes 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 parte* se presenta una descripción breve de la mayoría del conjunto de instrucciones del DSP TMS320C50. En la *tercera parte* se presenta la forma de implementar un filtro digital en un DSP, específicamente con las instrucciones del TMS320C50. En la *cuarta parte* se presenta 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 a la vez que aprenda a utilizar el ambiente. Estos programas empiezan desde muy elementales hasta algunos mas complicados que corren en tiempo real. La *quinta parte* consiste en programas propuestos que normalmente son similares a los proyectos que los alumnos desarrollan en clase con la asesoría del profesor.

El autor agradece al alumno José Miguel Ronquillo la colaboración en estas notas y prueba de los programas.

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

Capítulo 1

El starter kit del TMS320C50

El DSP starter kit del TMS320C50 (DSK) es una tarjeta conectada al puerto serie de una computadora personal (PC) y es utilizada para aplicaciones simples. La memoria disponible de esta tarjeta es únicamente la memoria interna del TMS320C50, 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, esta memoria no puede accesarse después, 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.

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.

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 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

[1]Ver configuración y mapa de memoria en el manual de usuario o notas sobre el TMS320C50 del mismo autor.

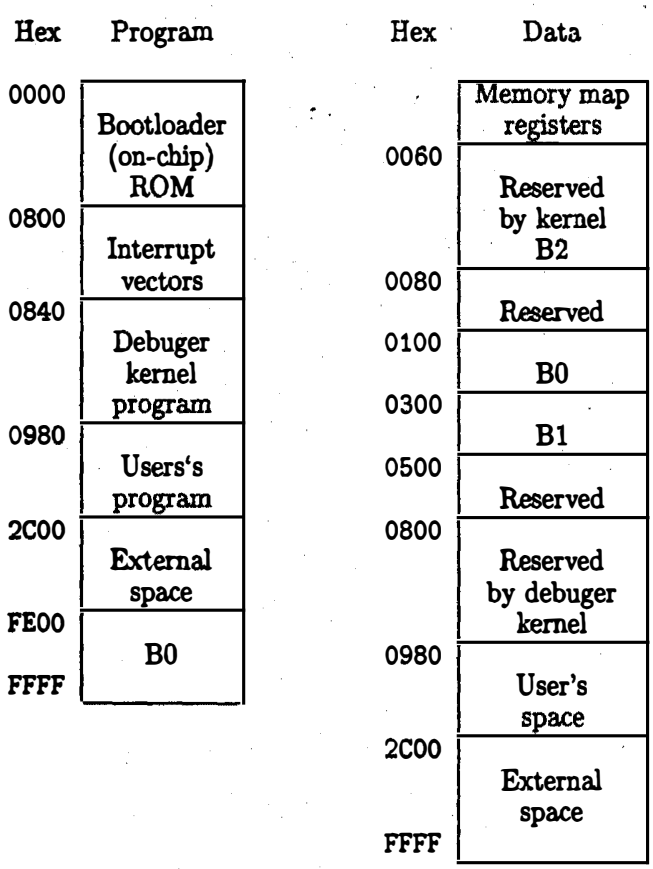


Figura 1.1: Mapas de memoria

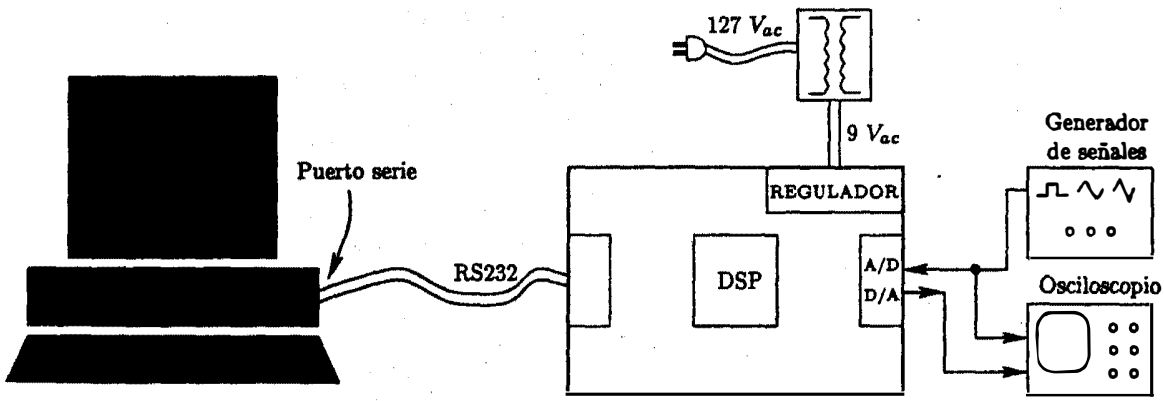


Figura 1.2: Conexión PC–TMS320C50

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 `.asm`, a través del ensamblador se genera un archivo `.lst` (opción -l), el cual contiene el archivo editado más una columna de direcciones y una de códigos, además se genera el archivo `.dsk` el cual contiene las direcciones con sus correspondientes códigos de operación en código hexadecimal.

1.2 El programa ensamblador

El proceso de ensamblado se realiza en dos pasadas, 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 pasada, 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>] [<COMENTARIO>]

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 dsk5d.exe no está en el PATH).

sintaxis:

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

```
\..\>dsk5a  fuente[2]     ; genera un archivo ejecutable extensión dsk si no existen errores
\..\>dsk5a  fuente -l     ; además del archivo dsk genera una archivo de salida 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
```

[2] Archivo fuente, extensión `.asm`.

```
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 sustituida 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.

1.3.2 Símbolos

Los símbolos pueden fijarse (directiva .set) con el valor de una constante y utilizarse dentro del programa.

Ejemplo:

```
N     .set   512
N1    .set   10
      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 Directivas del ensamblador

Las directivas del ensamblador proveen al proceso de ensamblado datos y control para hacer más fácil la programación. A continuación se explican brevemente cada una agrupándolas de acuerdo a su función.

1.4.1 Directivas que definen secciones

`.data`[3]	Ensambla en memoria dato. La memoria dato usualmente contiene dato 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).
`.text`	Ensambla en memoria programa. La sección text usualmente contiene el código ejecutable.

1.4.2 Directivas que referencia 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.

[3] add significa dirección (de address).

1.4.3 Directivas condicionales

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

`.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.

1.4.4 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.

`.bfloat val1,...,valn`[4]	Inicializa una constante en punto flotante 16 bits de exponente y 32 bits de mantisa en dos complemento.
`.byte val1,...,valn`	Inicializa una o mas palabras sucesivas de 8 bits en la sección actual.
`.double val1,...,valn`	Inicializa constantes en punto flotante de doble precisión IEEE (64 bits).
`.efloat val1,...,valn`	Inicializa una constante en punto flotante con 16 bits de exponente y 16 bits de mantisa en dos complemento.
`.float val1,...,valn`	Inicializa una constante 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 una contante entera de 32 bits en dos complemento con el punto decimal desplazado xx bits del LSB.
`.qxx val1,...,valn`	Inicializa una contante entera de 16 bits en dos complemento con el punto hipotético desplazado xx bits del LSB.
`.space` tamaño en bits	Reserva el tamaño de bits en la sección actual.
`.string "string1",..."stringn"`	Inicializa una o mas cadenas.
`.tfloat val1,...,valn`	Inicializa una constante de 32 bits de exponente y 64 bits de mantisa en dos complemento.
`.word val1,...,valn`	Inicializa uno o más enteros de 16 bits.

[4]Significa valores de las constantes.

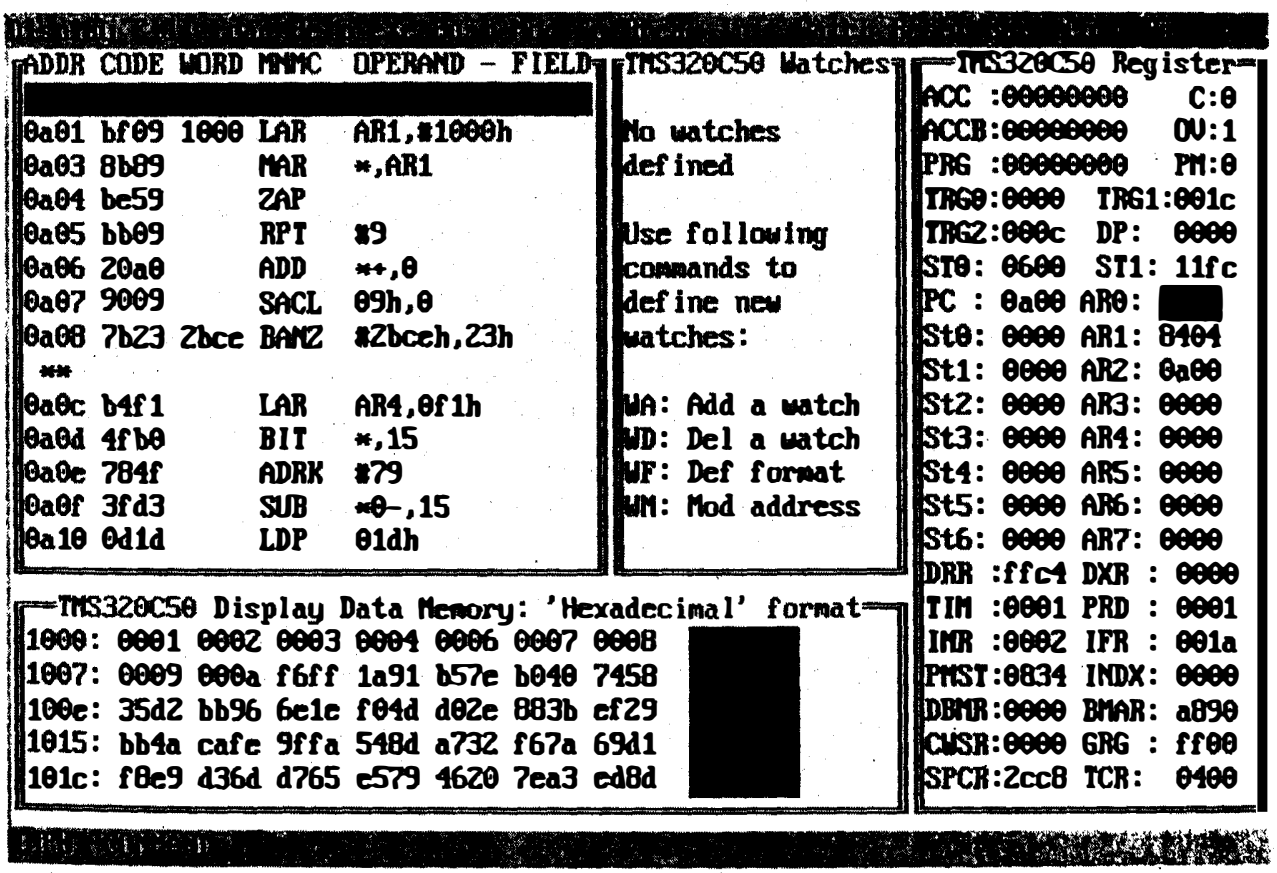
1.4.5 Otras Directivas

`.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.
- Registros.
- Watch.
- Memoria.

La 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.

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 se da en el laboratorio por el profesor con el objeto de que los comandos se aprendan en la práctica.

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 [TI 94] 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.
ADDB	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.
ADDC	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 bit (DBMR). En los dos casos, el resultado es escrito de nuevo en la localidad de memoria de dato, 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.
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, y el programa continúa ejecutándose de la localidad.

Ejemplo:

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

Después que el actual AR, ARP, y DP son modificados como se especifica, la ejecución del programa continua desde la localidad 191.

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    PGM191,LEQ,C
```

Si el contenido del acumulador es menor o igual que cero y el bit de acarreo está encendido, la dirección de programa 191 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:

```
BCND    PGM191,OV
MAR     *+,AR1
LDP     #5
```

Después que el actual AR, ARP, y DP son modificados como se especifica, la ejecución del programa continúa desde la localidad 191 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 al 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      PGM191,LEQ,C
```

Si el contenido del acumulador es menor o igual que cero y el bit de acarreo está encendido, la dirección PMG191 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     PGM191,LEQ,C
MAR     *+,AR1
LDP     #5
```

El actual AR, 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 PMG191. 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.
```

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 a al izquierda especificados. 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 instrucción 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 es 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 me 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 de el programa.

OPL	Si una constante inmediata es especificada, se realiza lo 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 bit (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 interior del stack se pierde.
RET	Retorno de subrutina. El contenido de la parte alta del 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 las dos instrucciones de una sola palabra o una instrucción de dos palabras que siguen a la instrucción RETD son alcanzadas y ejecutadas antes de la ejecución del retorno.

RETC	Retorno condicional de subrutina. Se ejecuta un retorno estándar (RET) si se cumplen las condiciones especificadas. Con retardo las dos instrucciones de una sola palabra o una instrucción de sos palabras que siguen a la instrucción RETCD son alcanzadas y ejecutadas antes de la ejecución del retorno. Si la instrucción con retardo es especificada, las dos siguientes 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 al 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.
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 constantes 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 de 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 cumple cierta condición. El número de palabras (instrucción de una pàlabra) 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 es escrito de nuevo en la localidad de memoria especificada.
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.

Capítulo 3

Implementación de filtros digitales en el DSP TMS320C50

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

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

con función de transferencia:

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

La respuesta al impulso unitario del sistema, $h(n)$ es la transformada inversa Z de la función de transferencia $H(z)$. Si los coeficientes a_k y b_m son reales, $h(n)$ es real, entonces el sistema sería real. 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[1].

Un filtro digital de respuesta finita al impulso (FIR) es aquel cuya respuesta al impulso es de duración finita. De la ecuación general las coeficientes a_k son cero, entonces, un filtro FIR queda descrito por:

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

y su función de transferencia:

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

[1]Se asume que el lector conoce la teoría básica del diseño de filtros digitales y el cálculo de los coeficientes.

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.

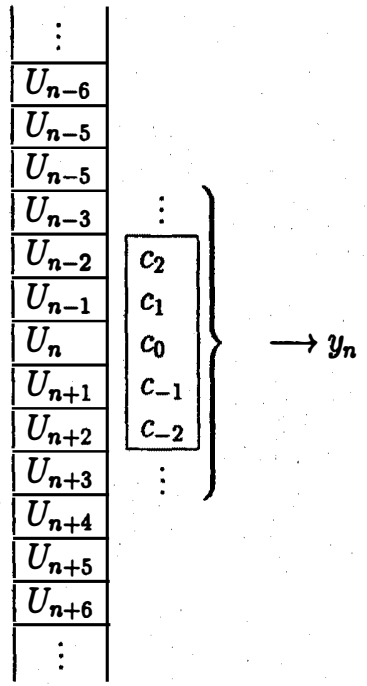


Figura 3.1: Convolución de una señal con los coeficientes del filtro

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 eficientemente 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).

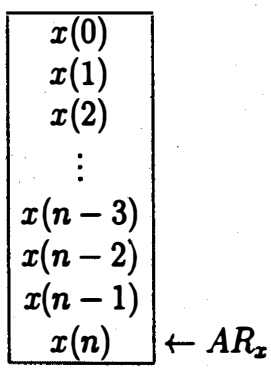


Figura 3.2: Buffer lineal

El TMS320C50 y la familia TMS320 pueden generar eficientemente las líneas de retardo con instrucciones orientadas para estos fines.

En el TMS320C50 (y la familia TMS320) una carga y almacenamiento de datos se combina en la instrucción DMOV:

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

esta instrucción mueve el valor en la localización <dma> y lo copia en la localización siguiente dma+1, sin afectar al acumulador u otro registro.

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.

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: Filtro FIR con LTD y MPY

3.1.2 Implementación de un filtro FIR utilizando LTD y MPY

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

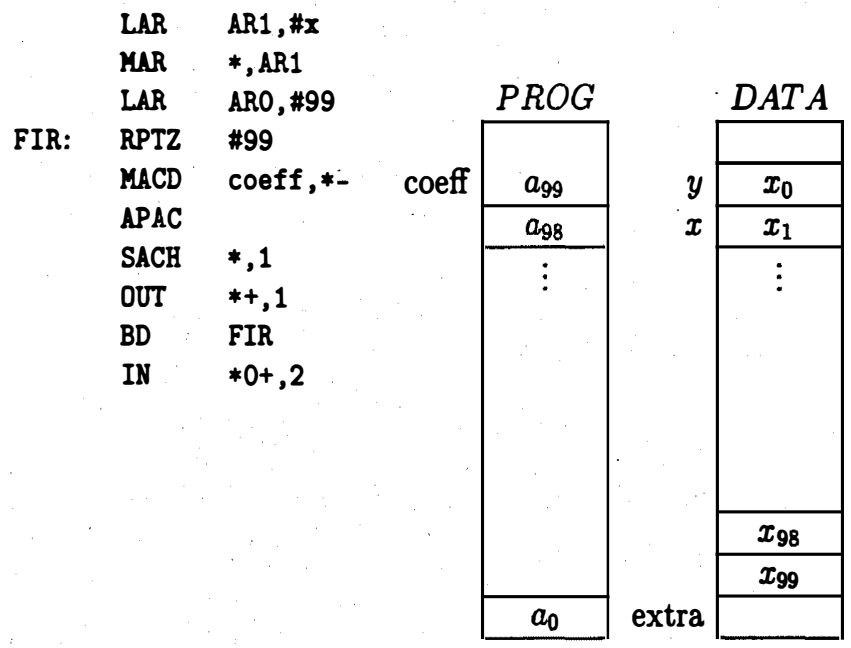


Figura 3.4: Filtro FIR usando MACD

3.1.4 Filtro FIR utilizando instrucción 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 en conjunto de las instrucciones MADS y DMOV. Otra forma de implementar el ejemplo anterior es:

```
LACC    #coeff
SAMM    BMAR        ; carga la dirección del coeficiente a0 al registro BMAR
.
.
.
MADD    *-          ; multiplica, acumula y mueve dato.
```

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.

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, 3.7.

3.2.2 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 3.8.

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 3.9.

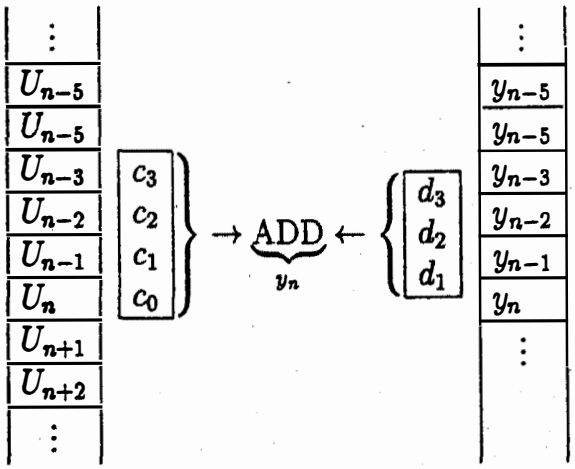


Figura 3.5: Líneas de retardo de un filtro IIR

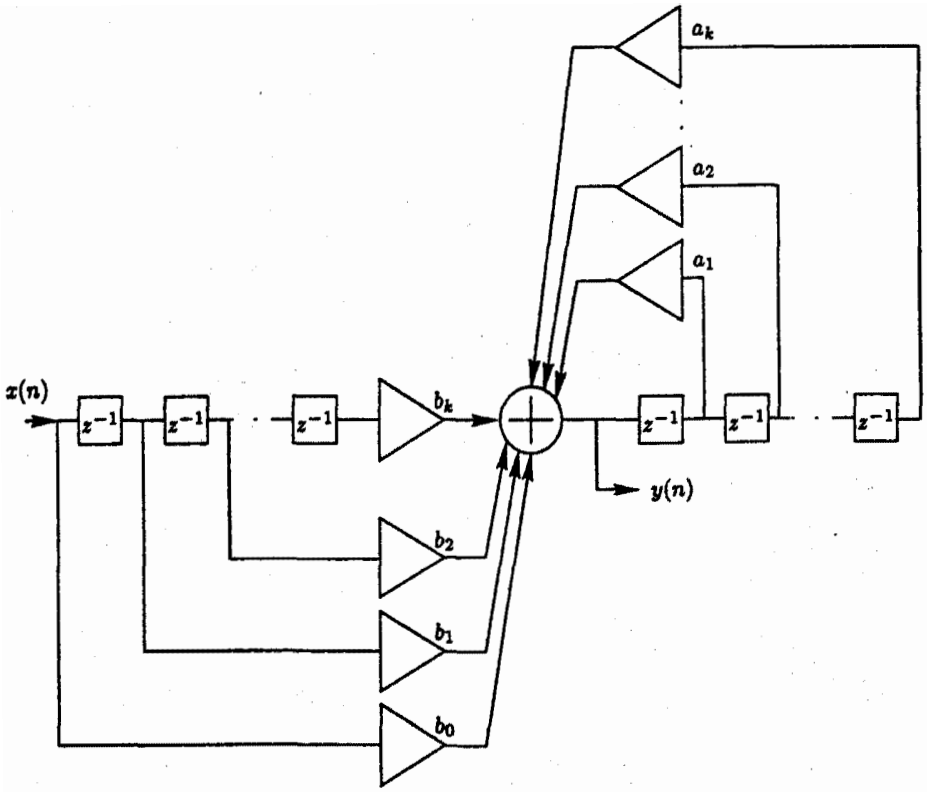


Figura 3.6: Forma directa de un filtro IIR

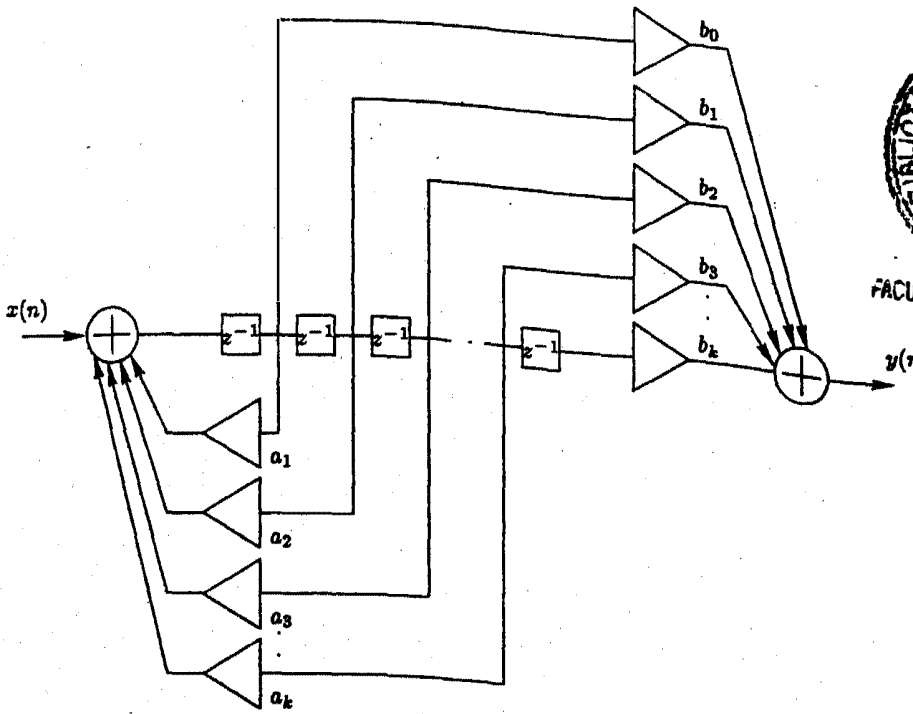


Figura 3.7: Forma canónica de un filtro IRR

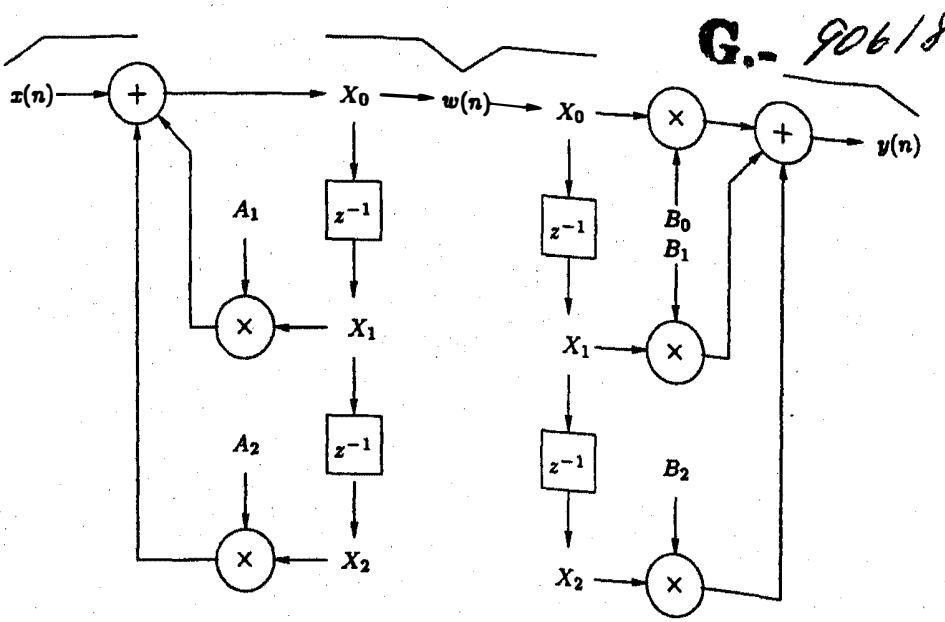


Figura 3.8: Filtro IIR de segundo orden

```
LDP    #X
SPM    0            ; No existe corrimiento en registro PR al
                    ; transferirse al ACC.
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*X1
MPY    A2           ; PR = X2*A2
APAC                ; ACC = Xin + X1*X1 + 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 3.9: Filtro IIR de segundo orden

Capítulo 4

Prácticas

Se propone a el alumno teclee los siguientes programas, los ensamble y los corra en el ambiente depurador y que observe como se van realizando las operaciones y como cambian algunos registros y localidades de memoria.

4.1 Sugerencias

- Cambiar datos de entrada y ver de nuevo como opera el programa.
- Escribir una línea final:

```
FIN     B       FIN
```

y correr el programa con F5

4.2 Suma de cinco números con diferentes posibilidades

4.2.1 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
D2      .set    2
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.
        .end
```

4.2.2 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
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
        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

        .end
```

4.2.3 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
                                ; 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 total.


        .end
```

4.2.4 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 la suma dato.
                                ; a dato sino el resultado total.
        ADD     *+              ; Suma los 5 números al acumulador.
        SACL    *               ; Pone el resultado de la suma en total.

        .end
```

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

4.2.5 Programa 5

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

```
        .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

        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 total

        .end
```

4.2.6 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 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

        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

        .end
```

4.3 Suma de productos de 5 números con varias posibilidades

Se supone que el lector ya conoce lo suficiente de la arquitectura y el conjunto de instrucciones del TMS320C50, por lo tanto, en seguida casi ni se utilizan comentarios.

4.3.1 Programa 7

Usando instrucción LT, MPY y APAC.

```
        .mmregs
        .ds     0f00h
A1      .word   001h
A2      .word   002h
A3      .word   003h
A4      .word   004h
A5      .word   005h
A6      .word   006h
*
X1      .word   001h
X2      .word   002h
X3      .word   001h
X4      .word   002h
X5      .word   001h
X6      .word   002h
*
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    Y0
        .end
```

4.3.2 Programa 8

Usando instrucción LTA y MPY.

```
        .mmregs
        .ds     0f00h
A1      .word   001h
A2      .word   002h
A3      .word   003h
A4      .word   004h
A5      .word   005h
A6      .word   006h
*
X1      .word   001h
X2      .word   002h
X3      .word   001h
X4      .word   002h
X5      .word   001h
X6      .word   002h
*
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
        LTA     *+,AR2
        MPY     *+,AR1
FIN_1   NOP
        APAC
        SACL    Y0
        .end
```

4.3.3 Programa 9

Usando instrucción LT y MPYA.

```
        .mmregs
        .ds     0f00h
A1      .word   001h
A2      .word   002h
A3      .word   003h
A4      .word   004h
A5      .word   005h
A6      .word   006h
*
X1      .word   001h
X2      .word   002h
X3      .word   001h
X4      .word   002h
X5      .word   001h
X6      .word   002h
*
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
        MPYA    *+,AR1
FIN_1   NOP
        APAC
        SACL    Y0
        .end
```

4.4 Multiplicación acumulación (usando la instrucción MAC)

4.4.1 Programa 10

```
        .mmregs
        .ds     0f00h
A1      .word   001h
A2      .word   002h
A3      .word   003h
A4      .word   004h
A5      .word   005h
A6      .word   006h
*
X1      .word   001h
X2      .word   002h
X3      .word   001h
X4      .word   002h
X5      .word   001h
X6      .word   002h
*
Y0      .word   0
XC      .word   0,0,0,0,0,0,0,0
*
```

```
N       .set    5

        .ps     0A00h
        .entry

        LDP     #Y0
        LAR     AR1,#X1
        LAR     AR0,#A1
        MAR     *,AR0

        LACC    #00

        RPT     #N
        MAC     #X1,*+

        APAC
        SACL    Y0

        .end
```

4.4.2 Programa 11

```
        .mmregs
        .ds     0f00h
A1      .word   001h
A2      .word   002h
A3      .word   003h
A4      .word   004h
A5      .word   005h
A6      .word   006h
*
X1      .word   001h
X2      .word   002h
X3      .word   001h
X4      .word   002h
X5      .word   001h
X6      .word   002h
*
Y0      .word   0
XC      .word   0,0,0,0,0,0,0,0
*
N       .set    5
```

```
        .ps     0A00h
        .entry

        LDP     #Y0
        LAR     AR1,#X1
        LAR     AR0,#X6
        MAR     *,AR0

        LACC    #00

        RPT     #N
        MAC     #A1,*-

        APAC
        SACL    Y0

        .end
```

4.5 Movimiento de bloques de datos

4.5.1 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     #Y0
        LAR     AR0,#X1
```

```
        MAR     *,AR0

        LACC    #00

        RPT     #N
        BLDD    *+,#Y

        .end
```

4.6 Suma de productos usando MADS

4.6.1 Programa 13

```
        .mmregs
        .ds     0f00h
A1      .word   001h
A2      .word   002h
A3      .word   003h
A4      .word   004h
A5      .word   005h
A6      .word   006h
*
X1      .word   001h
X2      .word   002h
X3      .word   001h
X4      .word   002h
X5      .word   001h
X6      .word   002h
*
Y0      .word   0
*
N       .set    5

        .ps     0a00h
        .entry

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

```
RPT     #N
MADS    *+
APAC

LDP     #YO
SACL    YO

.end
```

Capítulo 5

Ejemplos de programas en tiempo real

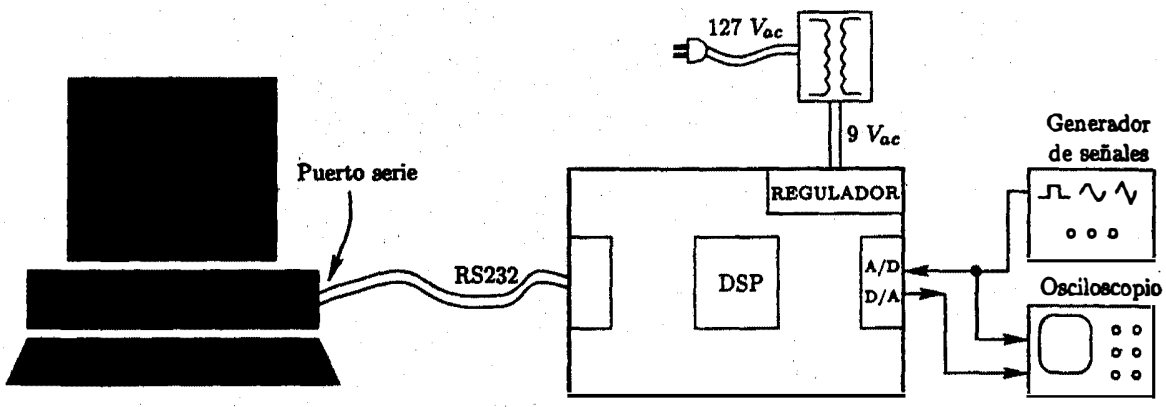


Figura 5.1: Conexión PC–DSK–Osciloscopio–Generador

```
*       LECTURA Y ESCRITURA DEL CONVERTIDOR ANALÓGICO DIGITAL
*       LEE-ESCRIBE DEL A/D y DEL D/A
*       Utiliza interrupciones con el TMS320C50
*
        .mmregs                 ; incluye registros mapeados

        .ds     0F00h
TA      .word    5              ; 5 Fcut = 8 KHz   para el A/D y D/A
RA      .word    5              ; 5 Fcut = 8 KHz   para el A/D y D/A
;
TB      .word    15             ; 15 Fs = 2*Fcut   para el A/D y D/A
RB      .word    15             ; 15 Fs = 2*Fcut   para el A/D y D/A
;
AIC_CTR .word    08h            ;                  para el A/D y D/A
```

```
MASK    .set    00080h  ;

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


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

        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 em registro mapeado IMR
        LAMM    SPC
        AND     #0FFF0h         ; Modo continuo
        SAMM    SPC
        CLRC    INTM

ESPERA: NOP
        NOP
         B       ESPERA         ; Espera interrupción

* Rutina 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
        SACL    DXR             ; ACCL --->  DXR
        RETE



*********************************************************************
*  Initicializa al  TLC320C46, convertidor A/D Y D/A
*********************************************************************
```

```
*
AICINIT: SPLK    #20h,TCR          ;
         SPLK    #01h,PRD          ;
         MAR     *,AR0
         LACC    #0008h            ;
         SACL    SPC               ;
         LACC    #00c8h            ;
         SACL    SPC               ;
         LACC    #080h             ;
         SACH    DXR               ;
         SACL    GREG              ;
         LAR     AR0,#0FFFFh       ;
         RPT     #10000            ;
         LACC    *,0,AR0           ;
         SACH    GREG              ;
         SETC    SXM               ;
         ;-------------------------
         LDP     #TA               ;
         LACC    TA,9              ;
         ADD     RA,2              ;
         CALL    AIC_2ND           ;
         ;-------------------------
         LDP     #TB               ;
         LACC    TB,9              ;
         ADD     RB,2              ;
         ADD     #02h              ;
         CALL    AIC_2ND           ;
         ;-------------------------
         LDP     #AIC_CTR          ;
         LACC    AIC_CTR,2         ;
         ADD     #03               ;
         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             ;
        IDLE                    ;
        SETC    INTM            ;
        RET
*
        .end
```

```
;                          FILTRO PASO BAJAS
;
;    PARA tms50
;    Archivo :       fir_pb.asm
;
;    N = 80, fc = 1 Khz, Ventana de Blacman
;
;    CONVERSIÓN
;    DSK a 40 M hz    Fs = 10,000,000/2/TA/TB  Para transmisión
;                     Fs = 10,000,000/2/RA/RB  Para recepción
;                     Máximo valor de TA,TB.RA,RB is 63 por los 6 bits
; ---------------------------------------------------------------------

        .MMREGS
        .ds     0f00h
TA      .word   5           ; 16
RA      .word   5           ; 16

                            ;
TB      .word   15          ; 31
RB      .word   15          ; 31
AIC_CTR .word   08h         ;


OUTPUT  .word   0
TEMP    .word   0           ;
TEMP1   .word   0           ;
*
*       COEFICIENTES     fc= 1K       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
;zero   .word     6554    ;   0       0.2000
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  localizaciones para 80
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        RECEIVE
xint:    B        TRANSMIT
```

```
        .ps     0a00h
        .entry                  ;
        ;---------------------------------------------------------

        SETC    INTM
        LDP     #0
        OPL     #0834h,PMST
        LACC    #0
        SAMM    CWSR
        SAMM    PDWSR
        SETC    SXM             ;  SXM EN SET
        SPLK    #022h,IMR       ;

        CALL    AICINIT         ;
        SPLK    #12h,IMR
        CLRC    OVM
        SPM     0               ;
        CLRC    INTM            ;

WAIT:                           ;
        B       WAIT            ;

* Rutina de atención de interrupción y realización del filtro FIR
RECEIVE:
        LDP     #XN
        CLRC    INTM            ;

        LAMM DRR                ;
        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                    ;

TRANSMIT:
        RETE


*************************************************************************
*    INICIALIZA                                                         *
*    CONVERTIDOR TLC320C40's  (AIC) TA,RA,TB,RB                         *
*************************************************************************

AICINIT: SPLK   #20h,TCR        ;
         SPLK   #01h,PRD        ;
         MAR    *,AR0           ;
         LACC   #0008h          ;
         SACL   SPC             ;
         LACC   #00c8h          ;
         SACL   SPC             ;
         LACC   #080h           ;
         SACH   DXR             ;
         SACL   GREG            ;
         LAR    AR0,#0FFFFh     ;
         RPT    #10000          ;
         LACC   *,0,AR0         ;
         SACH   GREG            ;
         ;-----------------------
         LDP    #TA             ;
         SETC   SXM             ;
         LACC   TA,9            ;
         ADD    RA,2            ;
         CALL   AIC_2ND         ;
         ;-----------------------
         LDP    #TB             ;
         LACC   TB,9            ;
         ADD    RB,2            ;
         ADD    #02h            ;
         CALL   AIC_2ND         ;
         ;-----------------------
```

```
        LDP     #AIC_CTR        ;
        LACC    AIC_CTR,2       ;
        ADD     #03h            ;
        CALL    AIC_2ND         ;
        RET                     ;

AIC_2ND:
        LDP     #0              ;
        SACH    DXR             ;
        CLRC    INTM            ;
        IDLE                    ;
        ADD     #6h,15          ;
        SACH    DXR             ;
        IDLE                    ;
        SACL    DXR             ;
        IDLE                    ;
        LACL    #0              ;
        SACL    DXR             ;
        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 prácticas y conocimientos sobre el TMS320C50. Eventualmente estos programas o similares pueden ser proyectos a realizar durante el curso.

6.1 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)
$$\begin{bmatrix} a_5 & a_4 & a_3 & a_2 & a_1 \\ a_4 & a_3 & a_2 & a_1 & 0 \\ a_3 & a_2 & a_1 & 0 & 0 \\ a_2 & a_1 & 0 & 0 & 0 \\ a_1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

6.2 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) $x^5 - 11.86x^4 + 14.604x^3 + 7.4704x^2 - 15.401x + 4.5926$

6.3 Programa 3

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

6.4 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.

6.5 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).

6.6 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

6.7 Programa 7

Realizar el programa 4 en tiempo real.

6.8 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.

6.9 Programa 8

Realizar un programa que en tiempo real de una salida de varias señales senoidales de igual amplitud pero que en diferentes intervalos den 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	$f = 5250$ hz

6.10 Programa 9

Realizar un programa en tiempo real que module en frecuencia F.M.

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

6.11 Programa 10

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

6.12 Programa 11

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 si 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:

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

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

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

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

Bibliografía

[LEE 89] IEEE ASSP MAGAZINE, Arquitectura programable DSP, Edward A. Lee. octubre de 1988 y enero 1989.

[TIA 90] Texas Instruments. Digital Signal Processing Applications with the TMS320 Family, Theory, Algorithms, and implementations, vol.1,2 y 3. USA 1990.

[TI 90] Texas Instruments. TMS30C1x, User's Guide. USA 1990.

[TI 91] Texas Instruments. TMS30C2x, User's Guide. USA 1991.

[TI 92] Texas Instruments. TMS30C3x, User's Guide. USA 1992.

[TI 94] Texas Instruments. TMS320C5x DSP starter Kit. User's Guide. USA 1994.

[ALC 89] Alcántara Silva Rogelio. Introducción al procesamiento digital de señales. Septiembre 1989.

[HAM 83] Hamming R. W. Digital filters. Prentice Hall, New Jersey 1983.