LABORATORIO DE SISTEMAS ELECTRONICOS PARA EL TRATAMIENTO DE LA INFORMACION.

PROFESOR: Dr. Juan F. Guerrero Martínez.
TIPO: Troncal.
CREDITOS: 4.5
PERIODO: 4º cuatrimestre.

PRACTICA 1.

Introducción a las herramientas de desarrollo del TMS 320: Manejo de las herramientas software: compilador, enlazador, simulador. BIBLIOGRAFIA: [3][13].

PRACTICA 2.

Filtrado digital IIR: Programación en ensamblador del TMS 320C25. BIBLIOGRAFIA: [3][5][6][7][8][13].

PRACTICA 3.

Generador de onda cuadrada, senoidal y triangular: Programación en ensamblador del TMS 320C25. BIBLIOGRAFIA: [3][5][6][7][8][13].

PRACTICA 4.

Filtrado digital FIR: Programación en ensamblador del TMS 320C25. BIBLIOGRAFIA: [3][5][6][7][8][13].

PRACTICA 5.

Modulador PWM: Programación en ensamblador del TMS 320C25. BIBLIOGRAFIA: [3][5][6][7][8][13].

PRACTICA 6.

Filtro adaptativo: Programación en ensamblador del TMS 320C25. BIBLIOGRAFIA: [3][5][6][7][8][13].

PRACTICA 7.

Generador DTFM: Programación en ensamblador del TMS 320C25. BIBLIOGRAFIA: [3][5][6][7][8][13].

PRACTICA 8.

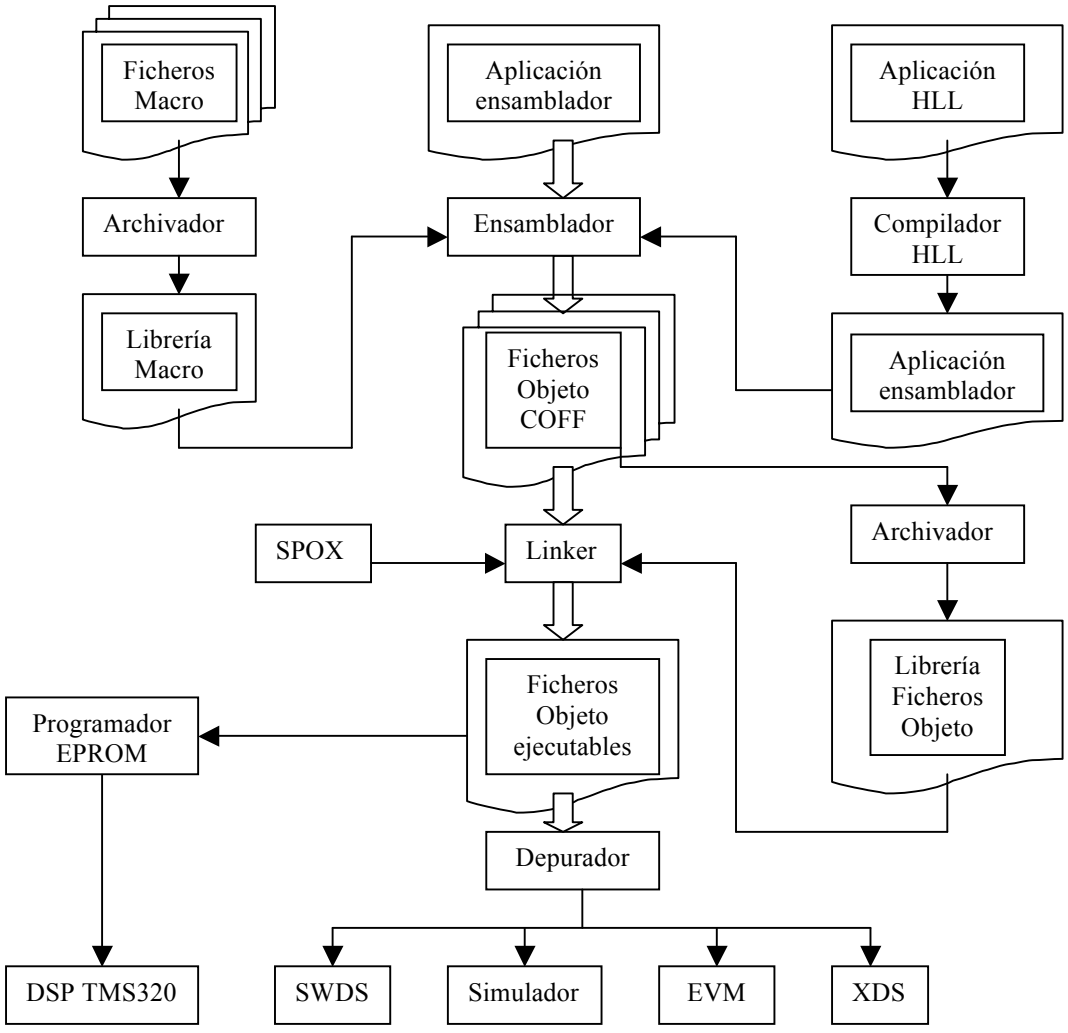
Modulación DPCM: Programación en ensamblador del TMS 320C25. BIBLIOGRAFIA: [3][5][6][7][8][13].

PRACTICA 9.

Demodulación DPCM: Programación en ensamblador del TMS 320C25. BIBLIOGRAFIA: [3][5][6][7][8][13].

Introducción a las herramientas de desarrollo del TMS320C25

Las herramientas de desarrollo proporcionadas por Texas Instruments para su familia de DSP TMS320 se muestran en el diagrama de flujo de la siguiente figura: ensamblador, enlazador, compilador HLL, depurador, simulador, distintas opciones de emulación (SWDS, EVM, XDS), sistema operativo en tiempo real (SPOX) y archivador para crear librerías propias.



De las distintas herramientas, vamos a utilizar el ensamblador y el enlazador, para generar el código ejecutable por el DSP, y el simulador para depurar y ejecutar ese código.

ENSAMBLADOR

El macro-ensamblador traduce ficheros fuente en lenguaje ensamblador (*.asm) a ficheros objeto (*.obj) en lenguaje máquina. Los ficheros fuente pueden contener instrucciones, directivas del ensamblador y directivas macro. Las directivas del ensamblador controlan varios aspectos del proceso de ensamblaje, tales como el formato del listado fuente, definición de símbolos, y la forma en que el código fuente se localiza en secciones.

Para ejecutar el ensamblador, se utiliza el siguiente comando:

dspa [fich_entrada [fich_objeto [fich_listado]]] [-opciones]

Fich_entrada es el fichero fuente en ensamblador. Por defecto, se toma como extensión '.asm'. Fich_objeto es el creado por el ensamblador. Por defecto tiene extensión '.obj', y si no se proporciona el nombre se asume el mismo del anterior. Fich_listado es opcional, y utiliza la extensión '.lst' por defecto.

Las opciones más usuales son:

-l: produce un fichero listado.

FORMATO COFF

Los ficheros COFF contienen dos tipos básicos de secciones. Las inicializadas contiene datos o código. Las no inicializadas simplemente reservan espacio en el mapa de memoria. Por defecto, se generan tres secciones, las dos primeras inicializadas, y la tercera no inicializada :

.text: generalmente contiene código ejecutable.
.data: generalmente contiene datos inicializados (tablas, etc.)
.bss: reserva espacio para variables no inicializadas.

El ensamblador dispone de una serie de directivas que le permiten ensamblar el programa dentro de la correspondiente sección. Cada vez que encuentra una de estas directivas, cambia de sección de ensamblaje. Por defecto, y en caso de no usar ninguna, ensambla todo el código en la sección .text.

Como secciones no inicializadas, que sólo reservan espacio, están:

.bss símbolo, tamaño
.usect "nombre sección", tamaño

donde 'símbolo' apunta a la primera palabra reservada, 'tamaño' es el número de palabras consecutivas reservadas, y 'nombre sección' define el nombre utilizado para la zona reservada. Estas directivas cambian la sección sólo para el código que está definido a continuación, no como en el caso de las restantes, que cambian la sección hasta encontrar una nueva directiva.

Como secciones inicializadas, que contienen programa o datos, están:

.text
.data
.sect "nombre sección"

.asect "nombre sección",dirección

.sect trabaja de forma similar a las dos primeras, pero con nombre definido por el programador.

.asect es una directiva que define secciones cuya dirección es absoluta en memoria y viene dada por el parámetro 'dirección'.

Como ejemplo de la utilización de directivas de secciones, se muestra el listado de un fichero COFF, que tiene 4 campos: 1) Contador de línea del código fuente; 2) Contador de sección de programa; 3) Código objeto; 4) Instrucciones fuente.

```
0001                    ************************************************
0002                    *       Ensambla una tabla inicicializada en .data     *
0003                    ************************************************
0004    0000                    .data
0005    0000    0011    coeff   .word   11h,22h,33h     ; inicializa 3 coeficientes
        0001    0022
        0002    0033
0006
0007                    ************************************************
0008                    *       Reserva espacio en .bss para 2 variables       *
0009                    ************************************************
0010    0000                    .bss    var1,1          ; reserva 1 palabra en memoria
0011    0001                    .bss    buffer,10       ; reserva 10 palabras
0012
0013                    ************************************************
0014                    *               Continua en .data                      *
0015                    ************************************************
0016    0003    0123    ptr     .word   123h            ; inicializa 1 coeficiente
0017
0018
0019                    ************************************************
0020                    *               Ensambla código en .text               *
0021                    ************************************************
0022    0000                    .text
0023    0000    200f    add:    LAC     0Fh
0024    0001    d003    aloop:  SBLK    1
        0002    0001
0025    0003    f280            BLEZ    aloop
        0004    0001'
0026    0005    6000-           SACL    var1,0
0027
0028                    ************************************************
0029                    *       Ensambla otra tabla inicicializada en .data    *
0030                    ************************************************
0031    0004                    .data
0032    0004    00aa    ivals   .word   0AAh,0BBh,0CCh      ; inicializa 3 coeficientes
        0005    00bb
        0006    00cc
0033
0034
0035                    ************************************************
0036                    *       Define la sección newvars para más variables   *
0037                    ************************************************
```

```
0038    0007            var2    .usect  "newvars",1     ; reserva 1 palabra en memoria
0039    0007            inbuf   .usect  "newvars",7     ; reserva 7 palabras en memoria
0040
0041


0042                    ************************************************
0043                    *               Ensambla más código en .text           *
0044                    ************************************************
0045    0006                    .text
0046    0006    d001    mpy:    LALK    0Ah
        0007    000a
0047    0008    a00a    mloop:  MPYK    0Ah
0048    0009    f780            BNV     mloop
        000a    0008'
0049    000b    6001-           SACL    buffer,0
0050
0051                    ************************************************
0052                    *       Define sección para vectores de interrupción     *
0053                    ************************************************
0054    0000                    .sect   "vectors"
0055    0000    0000            .word   0h,6h           ; inicializa 2 vectores
0056    0001    0006
```

En este caso, se han creado 5 secciones:

.text:	contiene 12 palabras de código objeto.
.data:	contiene 7 palabras de código objeto.
vectors:	es una sección creada por una directiva .sect; contiene 2 palabras de datos inicializados.
.bss:	reserva 11 palabras.
newvars:	es una sección creada por una directiva .usect; reserva 8 palabras.

Además de las directivas que definen secciones, otro grupo usual es el de directivas que inicializan memoria:

.byte:	coloca palabras de 8 bits en posiciones consecutivas dentro de la sección actual.
.word:	coloca palabras de 16 bits en posiciones consecutivas dentro de la sección actual.

Otra directiva frecuentemente utilizada es:

símbolo .set valor

asigna un valor a un símbolo, y el ensamblador lo sustituye en tiempo de ensamblaje en el resto del texto. Funciona de forma similar al #define del preprocesador del C.

Por último, .end termina el ensamblado, y tiene el mismo efecto que un fin de fichero.

ENLAZADOR

El enlazador combina ficheros objeto en un único módulo objeto ejecutable. Al crear este módulo, realiza operaciones de relocalización y resuelve referencias externas. El enlazador acepta ficheros objeto COFF (Common Object File Format) creados por el ensamblador como entrada. Puede aceptar también miembros de la librería creada por el archivador y módulos creados por enlazados previos. Las directivas del enlazador permiten combinar secciones de ficheros objeto, definir direcciones específicas para dichas secciones y definir/redefinir símbolos globales.

Para ejecutar el enlazador se utiliza el siguiente comando:

dsplnk [-opciones] $fichero_1$ ... $fichero_n$

Los ficheros pueden ser ficheros objeto, ficheros comando del enlazador o ficheros de librerías. El fichero de salida por defecto es a.out. Las opciones más usuales son:

-a: produce un módulo ejecutable con direcciones absolutas. Es la opción por defecto.
-o fich_sal: define un nombre para el fichero de salida distinto de a.out.

El enlazador puede ejecutarse incluyendo en la línea de comando todas las opciones y ficheros asociados, o invocarlo sólo con el nombre de un fichero de comando, lo cual hace que el enlazador busque en éste las especificaciones. Los ficheros comando pueden contener, además de las opciones y nombres de los ficheros de entrada y salida, las directivas de memoria comentadas anteriormente.

Dispone de dos directivas para combinar las secciones generadas por el ensamblador y proporcionar las direcciones definitivas en función de la memoria específica del sistema:

* MEMORY: permite definir el mapa de memoria del sistema, nombrando porciones de memoria y especificando su dirección de comienzo y su longitud.

*SECTIONS: indica al enlazador cómo combinar las secciones y dónde localizarlas en memoria.

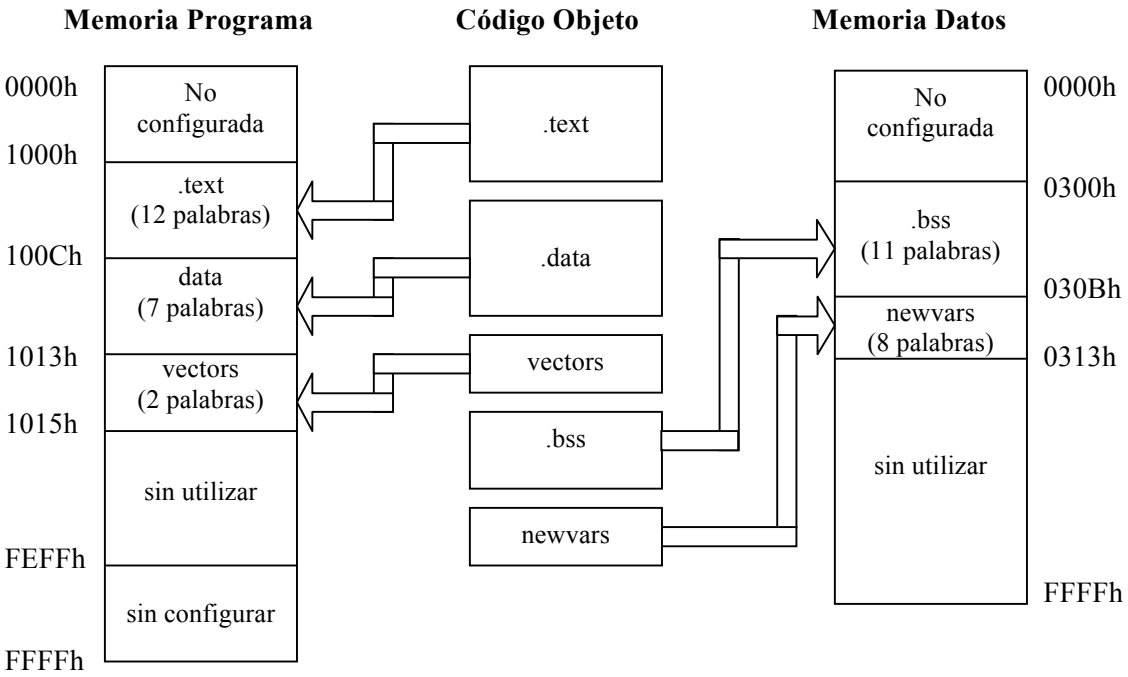
En caso de no especificar estas directivas, el enlazador utiliza un **modelo por defecto**:

- Asume que el DSP es el TMS320C25, con:

  - Memoria programa: externa (1000h-0FEFFh).
  - Memoria de datos: interna (B1: 300h-3FFh).
    externa (400h-0FFFFh).

  - Localiza .text en memoria de programa, a partir de 1000h.
  - Localiza .data en memoria de programa, a continuación de .text.
  - Localiza cualquier sección inicializada en memoria de programa a continuación de .data, y en el orden encontrado.
  - Localiza .bss en memoria de datos a partir de 300h.
  - Localiza cualquier sección no inicializada a continuación de .bss.

La siguiente figura muestra la localización por defecto del ejemplo. Si hay más de un programa a linkar, se localizan consecutivamente dentro de cada sección.



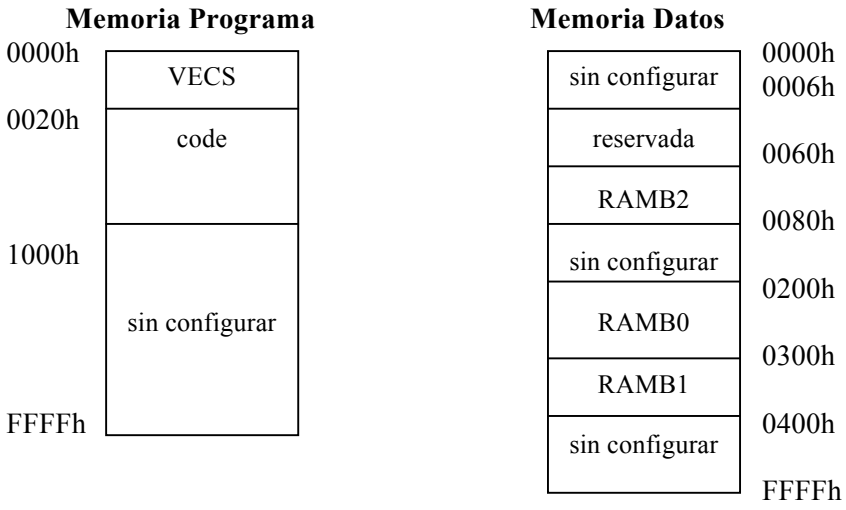
Mediante las directivas del *enlazador* se puede particularizar la relocalización de secciones al mapa de memoria del sistema. Estas directivas se localizan en un fichero de comando del *enlazador*. Siguiendo con el ejemplo anterior, la nueva especificación dada por el fichero de comando del *enlazador* sería:

```
MEMORY
{                                       /* Memoria programa (modo MC)*/
        PAGE 0 : VECS: origin = 000h,     length = 020h
                 CODE:        origin = 020h,     length = 0F90h

                                        /* Memoria datos  */
        PAGE 1 : RAMB2:      origin = 060h,     length = 020h
                 RAMB0:      origin = 200h,     length = 100h
                 RAMB1:      origin = 300h,     length = 100h
}

SECTIONS
{                                       /* Memoria programa */
        vectors     0000h       : { }                   PAGE 0
        .text                   : { }     > CODEPAGE 0

                                        /* Memoria datos  */
        .data                   : { }     > RAMB2       PAGE 1
        .bss                    : { }     > RAMB0       PAGE 1
        newvars                 : { }     > RAMB1       PAGE 1
}
```



SIMULADOR.

El simulador es un programa software que simula el DSP para verificación del software desarrollado en modo off-line. Permite depurar programas sin disponer de tarjeta hardware con el procesador. Puede asociar ficheros con puertos I/O, lo que permite simular datos de entrada/salida durante el test, y optimizar código con características de tiempo críticas mediante el contador de reloj que proporciona el número de ciclos que consume el programa.

Para ejecutar el simulador se utiliza el siguiente comando:

sim25

Al entrar, el simulador pide el modo de funcionamiento MC/MP. Por defecto, se toman los 4K de EPROM internos. A continuación, aparece la pantalla de trabajo. Los menús de ayuda están disponibles pulsando <CR>.

COMANDOS DE AYUDA.

Existen diversos menús de ayuda, que proporcionan información en línea sobre los comandos necesarios para introducir puntos de ruptura (BH), ejecutar programas (EH), definir entradas/salidas (IOH), modificar e inspeccionar memoria (MH) o registros (RH) o estado del procesador (STH), controlar la traza de ejecución (TH), interrupciones (TICH) y utilidades de registros (UTLH: guardar estados, etc.).

COMANDOS DE EJECUCION.

- LC: cargar un fichero ejecutable (formato COFF).
- R: comienzo de ejecución.
- SS: ejecución paso a paso.
- <ESC>: interrumpe ejecución.
- C: continúa ejecución.

- RS: reset simulador.
- EX: ejecuta comandos desde un fichero.
- Q: salir del simulador.

COMANDOS DE PUNTOS DE RUPTURA.

- BPP: ruptura cuando se lee un patrón desde progmem.
- BDP: ruptura cuando se lee/escr. un patrón desde datmem.
- BIAQ: ruptura cuando se lee una instrucción (dir.)
- DB: lista puntos de ruptura definidos.
- RB: borra un punto de ruptura.

COMANDOS DE ENTRADA/SALIDA.

Nota: Los datos de entrada pueden introducirse por teclado o desde un fichero de entrada. Análogamente, la salida puede guardarse en otro fichero. Los siguientes comandos permiten definir estas opciones.

- SI: selecciona un puerto de entrada.
- SO: selecciona un puerto de salida.
- LI: lista los puertos de entrada definidos.
- LO: lista los puertos de salida definidos.

COMANDOS DE INSPECCION/MODIFICACION DE MEMORIA.

- RAMI: visualiza contenido de memoria RAM en dec.
- RAMH: visualiza contenido de memoria RAM en hex.
- ROMI: visualiza contenido de memoria ROM en dec.
- ROMH: visualiza contenido de memoria ROM en hex.

OTROS COMANDOS.

- Z: pone a cero el contador de reloj.
- ZRAM: pone a cero el contenido de la RAM especificada.
- JF: Selecciona un fichero diario.

Nota: El fichero diario permite crear un fichero con todos los comandos ejecutados durante una sesión. Este fichero puede ejecutarse posteriormente con el comando EX. Es particularmente útil cuando se está en fase de depuración, en la que es frecuente salir del simulador para realizar cambios y posteriormente ejecutar de nuevo. JF crea un fichero y lo llena con todos los comandos introducidos por teclado hasta que se pulsa Q. Otro modo de ejecutar comandos es editarlos en un fichero texto y posteriormente ejecutarlos con EX.

LABORATORIO DE SISTEMAS ELECTRONICOS PARA EL TRATAMIENTO DE LA INFORMACION.

PRACTICA 1. INTRODUCCIÓN A LAS HERRAMIENTAS DE DESARROLLO DEL TMS320.

1. Ensamblar el fichero **seti_p1.asm** generando un fichero listado **seti_p1.lst** y uno objeto **seti_p1.obj**. Este programa calcula la serie:

$$Z = \sum_{i=1}^{10} X(i)\, Y(i)$$

2. Linkar el módulo objeto obtenido según el fichero comando **link.cmd** y generar un fichero ejecutable **seti_p1.out**.

3. Cargar en el simulador el fichero ejecutable. Ejecutar con entrada de datos desde teclado (modo SS para comprobar la correcta carga). Comprobar los resultados.

4. Después de los primeros 5 términos de la serie, interrumpir la ejecución (mediante ESC) y comprobar el contenido de la memoria asociada (El espacio para datos está situado en B1: 10 palabras para Y(i) y una para Z).

5. Continuar con la ejecución. Verificar el correcto fin del programa.

6. Introducir punto de ruptura para terminar el programa y ejecutar de nuevo.

7. Modificar **seti_p1.asm** para que acepte entrada desde un fichero de 100 datos entrada (seti_p1.dat: onda seno ±15000. Los primeros 10 datos son los valores de X(i) y permanecen constantes. Los siguientes 90 son valores para Y(i)). Repetir los puntos 1 y 2.

8. Crear un fichero de comandos del simulador. Definir PA1: Fichero entrada (seti_p1.dat) y PA2: Fichero salida (seti_p1.sal: contiene los 9 cálculos de la serie).

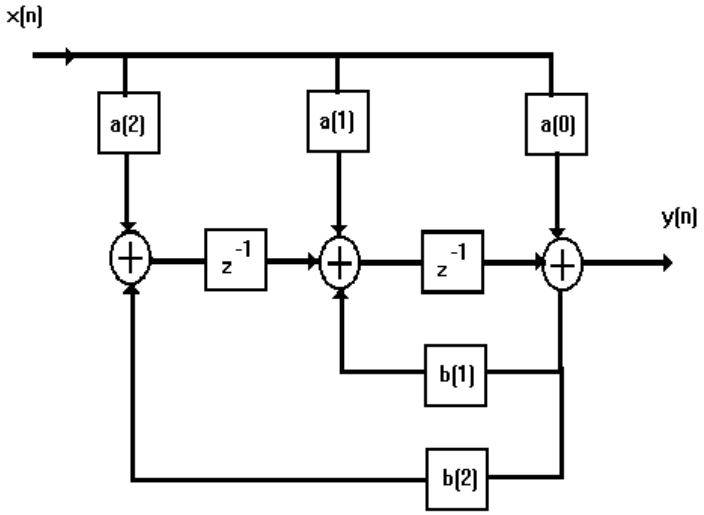
9. Ejecutar el programa modificado mediante el fichero de comandos. Medir el tiempo de ejecución de cálculo de una serie suponiendo un T_{CLK} de 80 ns.

10. Visualizar el fichero de entrada mediante Matlab, y listar el fichero de salida.

LABORATORIO DE SISTEMAS ELECTRONICOS PARA EL TRATAMIENTO DE LA INFORMACION.

PRACTICA 2. FILTRO DIGITAL IIR.

1. Programar el filtro IIR en ensamblador del TMS320C25 con estructura transpuesta dada por:



2. Los coeficientes originales son:
   a_0= 2.0585 a_1= 0.4262 a_2= -1.6324
   b_0= -0.8524 b_1= 0.7047

3. Los factores de escala a aplicar son: 2 para coeficientes y 8 para datos.

4. El programa adquiere los datos y genera las muestras de salida por el método de interrupciones (frecuencia de muestreo igual a 10 kHz).

5. Filtrar el fichero de datos (IIRT.dat) y visualizar mediante Matlab. Comparar los espectros de la señal original y filtrada.

6. Obtener el tamaño de memoria ocupado por el programa y calcular la máxima frecuencia de muestreo a la que podría trabajar el filtro.

LABORATORIO DE SISTEMAS ELECTRONICOS PARA EL TRATAMIENTO DE LA INFORMACION.

PRACTICA 3: GENERACION DE ONDA CUADRADA, SENOIDAL Y TRIANGULAR.

Implementar un generador de onda cuadrada, senoidal y triangular en ensamblador del TMS320C25 con las siguientes características:

1. Onda cuadrada: Amplitud de salida: ± 2000. Frecuencia: 1 kHz.

2. Onda triangular: se obtiene por integración de la onda cuadrada. Amplitud de salida: ± 2000. Frecuencia: 1 kHz.

3. Onda seno: se obtiene por filtrado del armónico principal de la onda cuadrada. Frecuencia: 1 kHz.

4. Se generan 3 ficheros de salida, correspondientes a las tres ondas: 'gen_c.dat', 'gen_s.dat' y 'gen_t.dat'. Tamaño de los ficheros: 2000 datos.

5. El programa adquiere los datos y genera las muestras de salida por el método de interrupciones (frecuencia de muestreo igual a 10 kHz).

6. Visualizar los ficheros mediante Matlab. Comparar los espectros de las señales obtenidas.

LABORATORIO DE SISTEMAS ELECTRONICOS PARA EL TRATAMIENTO DE LA INFORMACION.

PRACTICA 4. Filtrado digital FIR.

Implementar un banco de filtros FIR en ensamblador del TMS320C25 con las siguientes características:

1. Obtener el espectro del fichero de entrada (firc25.dat) mediante Matlab y determinar las frecuencias discretas que lo componen.

2. El programa debe implementar un banco de filtros, generando tantos ficheros de salida como componentes frecuenciales tenga la señal de entrada (fir_f#.sal).

3. El programa adquiere los datos y genera las muestras de salida por el método de interrupciones (frecuencia de muestreo igual a 20 kHz).

4. Diseño de los filtros: creación de los ficheros de coeficientes.

5. Filtrar el fichero de datos y visualizar entrada y salidas mediante Matlab. Comparar los espectros de la señal original y filtrada.

LABORATORIO DE SISTEMAS ELECTRONICOS PARA EL TRATAMIENTO DE LA INFORMACION.

PRACTICA 5: MODULACION PWM.

Implementar un modulador y demodulador PWM en ensamblador del TMS320C25 con las siguientes características:

1. Entrada moduladora: fichero con onda seno. Frecuencia: 10 Hz. Fm(A/D): 1KHz. Amplitud: ± 170. Nº datos: 500.

2. Salida PWM modulada: Frec. portadora: 1KHz. Fm(D/A): 20 KHz. Amplitud: ± 2000. Nº datos: 2000.

3. El generador PWM tiene un ciclo de trabajo del 50% para Vin=0. El ciclo de trabajo varía entre el 10% y el 90% para Vin=± 170.

4. El programa adquiere los datos y saca los resultados por el método de interrupciones.

5. La demodulación se implementará mediante filtrado pasa-baja.

6. Generar fichero de entrada moduladora: 'pwm_mod.dat'.

7. Generar fichero de salida modulada: 'pwm_sal.dat'.

8. Generar fichero de salida demodulada: 'pwm_dem.dat'.

9. Visualizar los ficheros mediante Matlab. Comparar los espectros de las señales obtenidas.

LABORATORIO DE SISTEMAS ELECTRONICOS PARA EL TRATAMIENTO DE LA INFORMACION.

PRACTICA 6. FILTRO ADAPTATIVO.

Implementar un filtro adaptativo en ensamblador del TMS320C25 con las siguientes características:

1. Algoritmo adaptación: LMS. Orden filtro FIR: 70.
2. El programa adquiere los datos y genera las muestras de salida por el método de interrupciones (frecuencia de muestreo igual a 20 kHz).
3. Entrada señal+ruido: fichero f_adap2.d. Nº datos: 2000.
4. Entrada ruido: fichero f_adap2.r. Nº datos: 2000.
5. Filtrar el fichero de datos y visualizar entradas y salida mediante Matlab. Comparar los espectros de la señal original y filtrada.

**LABORATORIO DE SISTEMAS ELECTRONICOS
PARA EL TRATAMIENTO DE LA INFORMACION.**

PRACTICA 7. Generador DTFM.

Implementar un generador DTFM en ensamblador del TMS320C25 con las siguientes características:

1. El programa lee de un puerto de entrada los dígitos de marcado (0...9) y genera el tono correspondiente (dos ondas seno) según la siguiente tabla:

DIGITO	FREC_1	FREC_2
0	941	1336
1	697	1209
2	697	1336
3	697	1477
4	770	1209
5	770	1336
6	770	1477
7	852	1209
8	852	1336
9	852	1477

2. El programa genera un fichero dtfm.sal con 500 muestras del tono correspondiente por cada dígito de entrada (frecuencia de muestreo: 8 kHz).

3. Visualizar fichero de salida con Matlab. Comparar los espectros de la señal para cada tramo de 500 muestras y comprobar la frecuencia correcta de los tonos generados.

LABORATORIO DE SISTEMAS ELECTRONICOS PARA EL TRATAMIENTO DE LA INFORMACION.

PRACTICA 8: MODULACION DPCM.

Implementar un modulador DPCM en ensamblador del TMS320C25 con las siguientes características:

1. Utiliza un polinomio predictor para obtener el valor estimado de la muestra dado por la ecuación:

$$y_{est}(n) = 2 \cdot y_{orig}(n-1) - y_{orig}(n-2)$$

donde y_{orig} es la muestra original del fichero de entrada y y_{est} la estimada por el predictor.

2. Codifica la diferencia entre y_{orig} y y_{est} en 4 bits (1 para signo y 3 para magnitud). En caso de superar la máxima magnitud, aplica aritmética saturada.

3. Cada 4 datos de entrada son comprimidos en una única palabra que contiene las correspondientes diferencias (formato: 4ª - 3ª - 2ª - 1ª) y enviados al fichero de salida. Los dos primeros datos de entrada se repiten sin modificación en el fichero de salida.

4. El programa adquiere los datos y saca los resultados por el método de interrupciones (f_m = 44 kHz).

5. Procesar el fichero de entrada (dpcm_m.dat) y generar el fichero de salida (dpcm_m.sal). Repetir para (dpcm_m1.dat) y generar (dpcm_m1.sal).

LABORATORIO DE SISTEMAS ELECTRONICOS PARA EL TRATAMIENTO DE LA INFORMACION.

PRACTICA 9: DEMODULACION DPCM.

Implementar un demodulador DPCM en ensamblador del TMS320C25 con las siguientes características:

1. Utiliza como fichero de entrada el generado por el modulador DPCM de la práctica anterior. Cada dato contiene la diferencia entre y_{orig} y y_{est} en 4 bits (1 para signo y 3 para magnitud), correspondientes a 4 muestras de la señal original (formato: 4ª - 3ª - 2ª - 1ª), generada por el polinomio predictor. Los dos primeros datos de entrada contienen los dos primeros valores de la señal original.

2. El programa adquiere los datos y saca los resultados por el método de interrupciones (f_m = 44 kHz).

3. Generar la señal reconstruida (dpcm_d.res) y la diferencia (dpcm_d.dif). Comparar con Matlab las señales original y reconstruida (dpcm_m.dat y dpcm_d.res), y sus correspondientes espectros.

4. Repetir para (dpcm_m1.sal) y (dpcm_d1.res). Comparar señales y espectros. Razonar diferencias con los resultados obtenidos en el punto anterior.