UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

ESCUELA UNIVERSITARIA
DE
INGENIERIA TECNICA DE TELECOMUNICACION

Titulo: Sistema de proceso de señal digital controlado desde PC.

Autor: Roberto Morera Bello.

Tutor: Roberto Dominguez Rodriguez.

Fecha: Noviembre 1990.

Fdo. Fdo.

Roberto Morera Roberto Dominguez

A mi familia y a mi novia, porque se lo han merecido.

Deseo expresar mi agradecimiento a:

Roberto Domínguez Rodriguez, tutor del proyecto.

Ignacio, el encargado del laboratorio de Telemática.

Jose Antonio, encargado del laboratorio de electrónica de industriales.

Y a todos aquellos a los que les haya dado la lata durante la realización del proyecto.

INDICE:

-Introducción 1

-Descripción de conexión al bus del PC 4

-Descripción del 8255 6

-Descripción del 8237 8

Operación DMA 13

Tabla de puertos 16

-Caracaterístecas del procesador TMS32010

Descripcción general 18

Arquitectura 21

Características aritméticas 23

Memoria de datos 29

Registros 31

Memoria de programas 33

Modos de operación 34

Resumen del juego de instrucciones 48

-Emulador XDS/22

1.-Qué es un emulador 65

2.-El emulador XDS/22 66

3.-Conexión a un PC 69

4.-Inicialización del sistema 71

5.-Juego de comandos del programa monitor 74

-Descripción del circuito 118

-Esquema de la placa principal 119

-Placa principal 120

Funcionamiento básico del sistema 121

Funciones del 8255 123

Conexión del TMS con su memoria externa 124

Acceso del PC a la memoria del sistema 127

Conexión TMS-PC vía Latchs 129

Direccionamiento de los puertos del sistema 130

Configuración del conector 131

-Esquema de la placa de convertidores 134

-Placa de convertidores

Descripción 135

Sincronización de conversión 138

-Proceso digital de señal 140

Conversión A/D y D/A 143

Representación de las señales discretas 148

Sistemas LTI 150

-Descripción del software de control

* Funciones básicas

Función lee 157

Función escribe 163

* Programa de control 168

-Sistema de análisis de espectro 186

Programa en ensamblador 187

Programa del PC 204

-Presupuesto aproximado 219

APENDICE A: FOTOCOPIAS DE CARACTERISTICAS

APENDICE B: PLACAS DE CIRCUITO IMPRESO

INTRODUCCION:

INTRODUCCION:

El microprocesador TMS32010, es un procesador creado por Texas Instruments con el objetivo de permitir la realización de diversos sistemas de proceso de señal digital como pueden ser filtros digitales, modems, analizadores de espectro, generadores de efectos de sonido, sintetizadores de voz, etc.
Para ello posee un reducido (sin llegar a ser RISC), pero poderoso juego de instrucciones orientadas a facilitar las funciones para las que fue creado; con la interesante particularidad de que la mayor parte de ellas tiene un tiempo de ejecución de tan sólo 200 ns, lo que significa que su velocidad de proceso es de prácticamente 5 MIPS (Millones de Instrucciones Por Segundo).

El objetivo de este proyecto consiste en conectar y controlar este procesador desde un ordenador compatible PC estándar, lo que permitirá monitorizar la señal recibida o procesada, si así se desea, o bien controlar el funcionamiento del TMS32010 desde teclado según los intereses del usuario.
Esto haría posible, por ejemplo, el desarrollo de un sistema en el que se pudiera controlar las características de un filtro desde teclado, o bien el desarrollo de un sintetizador de voz que pronuncie frases cargadas desde un fichero o teclado.

Para demostrar el correcto funcionamiento del sistema se desarrollará además, como parte del proyecto, una utilidad que permita ver por pantalla gráficamente la distribución frecuencial de una señal introducida, es decir, se realizará el análisis de espectro de dicha señal.

Esta, como se ha comentado, no es más que una de las muchas utilidades posibles del sistema, quedando el campo abierto para el desarrollo futuro de posteriores aplicaciones.

En lo que respecta a la parte software, también se incluye un programa, escrito en C (mediante el compilador de TURBO C), que permitirá una serie de opciones de control del procesador, como por ejemplo monitorizar el contenido de la memoria del mismo, o rellenar una zona de memoria con un valor determinado.

Como es evidente, aparte de la etapa de control propiamente dicha, más una placa de bufferización de las señales del PC, hará falta una etapa, con sus correspondientes convertidores, que realice la comunicación del sistema con el mundo exterior.

Para el desarrollo del sistema en cuestión, se ha contado con la ayuda del emulador XDS model 22, que permite la visualización y modificación de cada uno de los registros del procesador TMS32010, y cuenta con un amplio conjunto de utilidades que facilitan la depuración de errores.

El emulador se conecta al sistema mediante un conector que irá encajado en el zócalo correspondiente al TMS32010, sustituyéndolo.

Además se cuenta con la ayuda de las siguientes utilidades software para el PC:

-Un ensamblador para crear ficheros de código objeto para el TMS32010, estos programas se pueden cargar y ejecutar por medio del emulador XDS/22. Este ensamblador permite el uso de macros.

-Un Linkador, para crear el código definitivo en caso de que se hiciera el programa en código reubicable.

-Un simulador, que permite la simulación del TMS302010 para verificar el correcto funcionamiento de los progrmas desarrollados.

En esta memoria se comentarán con amplitud todos los elementos relativos al sistema, algunos de los cuales han sido esbozados en esta introducción.

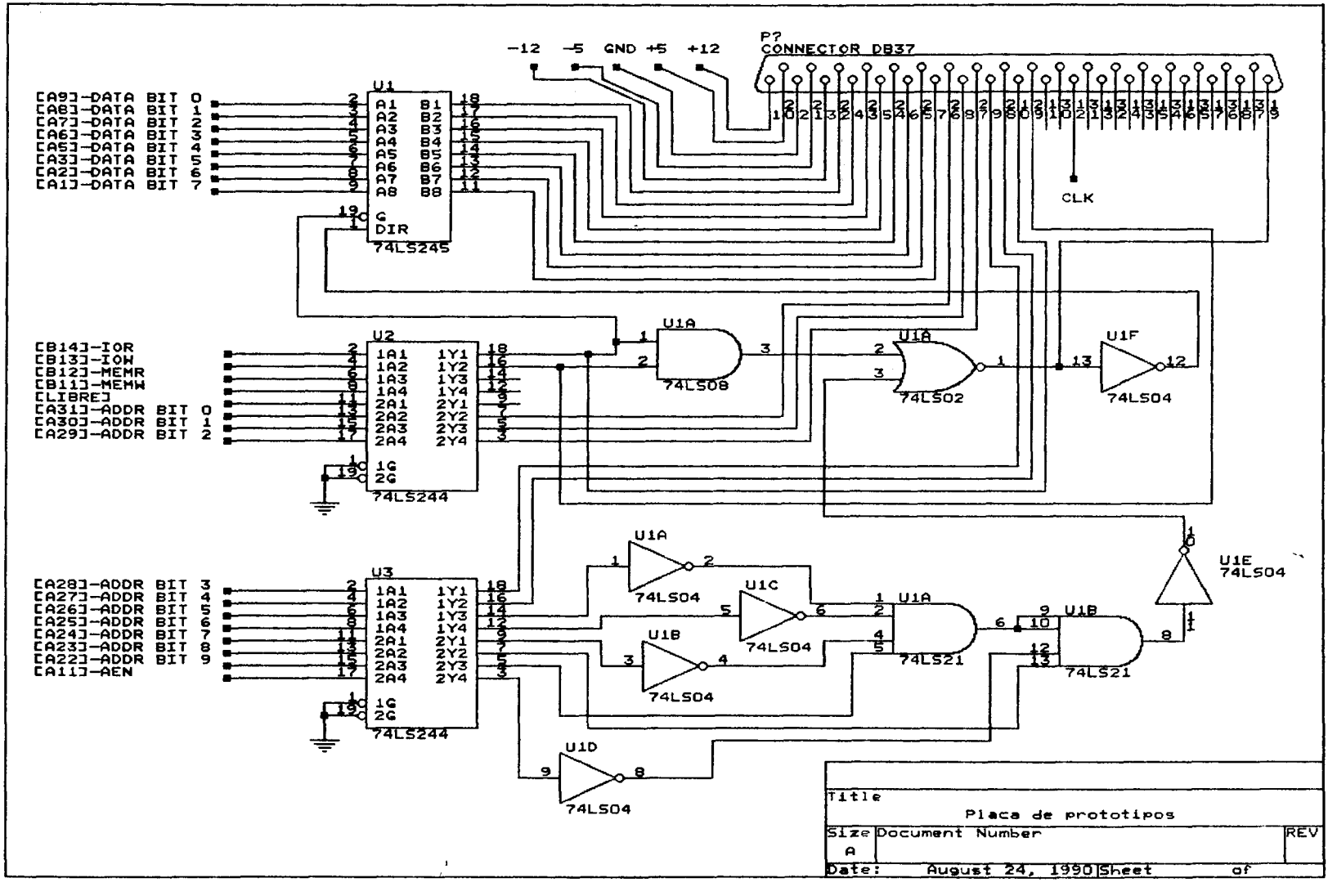
-DESCRIPCION DE LA ETAPA DE CONEXION AL BUS DEL PC:

La misión de esta etapa previa no es otra que la de bufferizar las señales del bus del PC, decodificando el bus de direcciones de tal forma que dicha bufferización no ocurra más que cuando se direccione alguno de los puertos que tiene asignado el sistema.

El esquema de esta etapa, que coincide con el de la tarjeta de prototipos descrito en el manual hardware del PC, aparece en la página siguiente.

En ella se bufferiza el bus de datos mediante el integrado **74LS245**, que es un buffer bidereccional, y el bus de direcciones (los diez bits de menor peso) y las señales de control (IOR, IOW, ..) mediante dos **74LS244**, que son dos simples buffers de ocho bits.

Además, en esta etapa se realizará la decodificación de las direcciones de puertos, para que el sistema sólo actúe ante las operaciones de E/S cuyas direcciones estén comprendidas en las direcciones **300H** a **31FH**, ambas inclusive.



A continuación se pasarán a describir dos integrados clave para el funcinamiento del sistema, como son el 8255 y el 8237:

-DESCRIPCCION DEL 8255:

El 8255A es un dispositivo diseñado para servir como interfaz programable de periféricos (PPI-Programmable Peripheral Interfase), cuya función es la de actuar como un componente de E/S de propósito general a la hora de conectar el equipo periférico al bus de datos del sistema microprocesador. La capacidad de ser programado desde el sistema microprocesador le confiere una gran flexibilidad.

Posee tres puertos de ocho bits A,B y C, que dependiendo del modo de funcionamiento pueden actuar aisladamente, o bien distribuirse los bits del puerto C entre el puerto A (formando el grupo A), y el puerto B (formando el grupo B).

Nosotros trabajaremos en el modo 0, modo en el cual los puertos A,B y C actuan como puertos independientes.

BUFFER DEL BUS DE DATOS:

La comunicación del 8255 con el bus de datos del sistema ,el del PC en nuestro caso, se realizará a través de un buffer 3-state de ocho bits, por medio del cual se transmitirán tanto los datos como la información de control y status.

La posibilidad de establecer esta comunicación estará controlada por el CS, activo a nivel bajo, que junto a las señales de IOR e IOW, y los bits de dirección A0 y A1 controlan las comunicaciones.

Las patillas A0 y A1 del 8255 se conectarán a los bits A0 y A1 respectivamente del bus de direcciones del PC,

A continuación indicamos en una tabla las operaciones básicas del 8255A, así como los puertos que les corresponden en nuestro sistema:

OPERACIONES BASICAS DEL 8255A:

A1	A0	RD	WR	CS	PUERTO	OPERACION DE ENTRADA(READ)
0	0	0	1	0	784	PUERTO A=>BUS DE DATOS
0	1	0	1	0	785	PUERTO B=>BUS DE DATOS
1	0	0	1	0	786	PUERTO C=>BUS DE DATOS
						OPERACION DE SALIDA(WRITE)
0	0	1	0	0	784	BUS DE DATOS=>PUERTO A
0	1	1	0	0	785	BUS DE DATOS=>PUERTO B
1	0	1	0	0	786	BUS DE DATOS=>PUERTO C
1	1	1	0	0	787	BUS DE DATOS=>CONTROL
						FUNCION DESHABILITADA
X	X	X	X	1		BUS DE DATOS=>3-STATE
1	1	0	1	0	787	CONDICION ILEGAL
X	X	1	1	0		BUS DE DATOS=>3-STATE

-DESCRIPCION DEL 8237:

El 8237 es un integrado cuya misión es la de controlar el acceso directo de un dispositivo externo a la memoria del sistema. Es un dispositivo programable, previsto para que proporcione un funcinamiento correcto en los más variados sistemas.

DESCRIPCION DE LAS PATILLAS:

Vcc: +5V alimentación.

Vss: Tierra.

CLK (Reloj, Entrada)
Esta patilla sirve como entrada de reloj, la frecuencia de entrada es de hasta 5Mz para el 8237-2.

CS (Chip Select, Entrada)
Esta patilla es activa a nivel bajo; durante el ciclo de reposo (idle cycle), permite las comunicaciones con la CPU.

RESET (Reset, Entrada)
Es una entrada asíncrona, activa a nivel alto, que resetea los registros del sistema.

READY (Ready, Entrada)
Es una entrada que permite extender los pulsos de lectura y escritura del 8237, con el fin de permitir la comunicación con periféricos o memorias más lentos.

HLDA (Hold Acknowledge, Entrada)

Mediante esta entrada, activa a nivel alto, le indica la CPU al 8237 que le cede el control del bus de datos del sistema, con lo cual queda permitida la transferencia que se desee realizar.

DREQ0-DREQ3 (DMA Request, Entrada)

Estas líneas son canales individuales asíncronos de entrada, usados por los periféricos para requerir los servicios de la DMA. La polaridad de estas líneas es programable, Reset las inicializa como activas a nivel alto.

Una petición de DMA se genera mediante la activación de una línea de DREQ, DACK (DMA Acknowledge) indicará el reconocimiento por parte del 8237 de la señal de DREQ. La señal de DREQ deberá de mantenerse hasta que la correspondiente DACK se active.

DB0-DB7 (Bus de Datos, Entrada/Salida)

Está constituido por ocho líneas bidireccionales triestadas, conectadas al bus de datos del sistema.

IOR (I/O Read, Entrada/Salida)

Es una línea bidireccional triestada, activa a nivel bajo.

En el ciclo de reposo, actúa como señal de entrada, y es usada por la CPU para leer los registros de control.

Durante el ciclo activo, funciona como salida, y es una señal de control usada por el 8237 para requerir un dato a un periférico durante una DMA Write transfer.

IOW (I/O Write, Entrada/Salida)

Es una línea bidireccional triestada, activa a nivel bajo.

En el ciclo de reposo, actúa como señal de entrada, y es usada por la CPU para cargar información en el 8237.

Durante el ciclo activo, funciona como salida, y es una señal de control usada por el 8237 para enviar un dato a un periférico durante una DMA Read transfer.

EOP (End Of Process, Entrada/Salida)

EOP es una señal bidireccional activa a nivel bajo. Puede ponerse a nivel bajo desde el exterior, con lo que finaliza un servicio activo de DMA, o desde el interior, lo que ocurrirá automáticamente al alcanzar un TC (terminal Count).

La recepción de esta señal, ya sea interna o externa, provocará la finalización de un servicio del 8237, reseteando la petición, y si la autoinicialización está permitida copiando el base register en el current register de ese canal.

A0-A3 (Address, Salida)

Las cuatro líneas menos significativas del bus de direcciones son señales bidireccionales triestadas.

Durante el ciclo de reposo funcionan como entradas, y las usa el 8237 para direccionar el registro de control que se quiera cargar o leer.

Durante el ciclo activo trabajan como salidas, y proveen los cuatro bits menos significativos de la dirección de salida.

A4-A7 (Address, Salida)

Estas cuatro líneas de direcciones más significativas son salidas triestadas y proveen cuatro bits de direccionamiento.
Estas líneas se activan únicamente durante el servicio de DMA.

HRQ (Hold Request, Salida)

Esta línea la emplea el 8237 para solicitarle a la CPU el control del bus del sistema.
Si el mask bit correspondiente está activo, la presencia de cualquier DREQ válido causa que el 8237 evíe esta señal. Después de que se active el HRQ, tinene que transcurrir al menos un ciclo de reloj antes de que la HLDA se active.

DACK0-DACK3 (DMA Acnowledge, Salida)

Esta señal tiene como misión notificar a los periféricos individuales cuando se les ha concedido un ciclo de DMA. El sentido de estas líneas es programable, siendo su valor inicial el de activo a nivel bajo.

AEN (Address Enable, Salida)

Esta salida permite al latch de ocho bits, que contiene los ocho bits de mayor peso del bus de direcciones, volcar su contenido en el bus de direcciones del sistema.

ADSTB (Address Strobe, Salida)

Esta salida, activa a nivel alto, se emplea para cargar el byte superior de direcciones en el latch externo.

MEMR (Memory Read, Salida)

Esta salida, activa a nivel alto y triestada, se emplea para acceder a la posición de memoria seleccionada durante una operación de lectura de DMA o una transferencia de memoria a memoria.

MEMW (Memory Write, Salida)

Esta salida, activa a nivel alto y triestada, se emplea para escribir en la posición de memoria seleccionada durante una operación de escritura de DMA o una transferencia de memoria a memoria.

LISTA DE REGISTROS:

NOMBRE	BITS	CANTIDAD
BASE ADDRES REGISTERS	16	4
BASE WORD COUNT REGISTERS	16	4
CURRENT ADDRESS REGISTERS	16	4
TEMPORARY ADDRESS REGISTERS	16	4
TEMPORARY WORD COUNT REGISTER	16	1
STATUS REGISTERS	8	1
COMMAND REGISTER	8	1
TEMPORARY REGISTER	8	1
MODE REGISTER	6	4
MASK REGISTER	4	1
REQUEST REGISTER	4	1

OPERACION DE LA DMA:

El 8237 funciona con dos modos principales de trabajo, que son: El active cycle, o ciclo activo, y el idle cycle , o ciclo de reposo.

IDLE CYCLE:

El 8237 permenacerá en este ciclo mientras que ningún canal requiera servicio.

La CPU puede ahora leer o escribir los registros internos del 8237. Las líneas de direcciones A0-A3 funcionan ahora como entradas, y permiten seleccionar el registro que se desea inspeccionar o modificar. Las líneas IOR e IOW se usan para seleccionar y sincronizar las operaciones de lectura y escritura. Debido al número y estructura de los registros internos, se hace uso de un flip-flop que trabaja como un bit adicional de dirección, seleccionando el byte superior o inferior de los registros de 16 bits "Address register" y "Word Count register". Este registro se resetea mediante un master clear, un reset, o bien mediante un comando especial.

El 8237 puede ejecutar comandos especiales estando en este modo, no haciendo para ello uso del bus de direcciones, ya que se especifican dichos comandos mediante la decodificación del bus de direcciones.

ACTIVE CICLE:

Cuando el 8237 está en el Idle cycle (ciclo de reposo), y un canal requiere un servicio de la DMA, el 8237 envía una señal de HRQ a la CPU y entra en el ciclo activo, en el que se produce la transferencia de DMA propiamente dicha, la cual puede realizarse mediante uno de los siguientes cuatro modos:

Single Transfer Mode: En este modo el sistema se programa para transferir un dato cada vez. El registro de Word Count se decrementa y el Registro Address se incrementa o decrementa después de cada transferencia.

Block Transfer Mode: En este mode el sistema se activa mediante un DRQ, y continúa haciendo transferencias hasta que el registro Word Count llega a cero, momento en el que se envía un TC (Terminal Count), o bien hasta que se produce una señal de EOP externa. DREQ deberá mantenerse activo hasta que se active la patilla de DACK.

Demand Transfer Mode: En este modo el sistema se programa de tal forma que sigue haciendo transferencias hasta que se encuentre un TC o un EOP, o bien hasta que DREQ se vuelva inactivo. Unicamente un EOP puede provocar una Autoinicialización del sistema, y puede generarse mediante un TC o mediante una señal externa.

Cascade Mode: Este modo se emplea para colocar en cascada varios 8237, lo que permite una sencilla expansión del sistema.

En los tres primeros modos, se producirá una Autoinicialización al finalizar un servicio si el canal está programado para ello. Una Autoinicialización provocará que se copien los Base Registers en los Current Registers.

TIPOS DE TRANSFERENCIA:

Son tres: Read, Write y Verify.

El tipo Verify es una pseudo-transferencia, el 8237 opera como en las transferencias de lectura y escritura, generando direcciones, respondiendo al EOP, etc, pero las líneas de control de E/S y de memoria permanecen inactivas.

GENERACION DE LAS DIRECCIONES:

El 8237 multiplexa los 16 bits de direcciones de tal forma que los ocho bits de mayor peso los almacena en un latch externo, desde el cual se colocan en el bus de direcciones. El flanco de bajada del ADSTB se emplea para cargar estos bits de las líneas de datos al latch. Los bits de menor peso de las direcciones los genera el 8237 directamente, para lo cual deberán conectarse al bus de direcciones las líneas A0-A7. El 8237 se encargará de actualizar la información del latch cuando proceda.

COMANDOS SOFTWARE:

Clear First/Last Flip-Flop: Inicializa el flip-flop de tal forma que en el siguiente acceso a un registro se direccionarán los bits superior e inferior en el orden correcto.

MASTER CLEAR: Tiene el mismo efecto que un hardware reset.

TABLA DE PUERTOS:

A3	A2	A1	A0	CS	IOR	IOW	CANAL	PUERTO	REGISTRO	OPERACION
0	0	0	0	0	1	0	0	768	BASE&CURRENT ADDR	WRITE
0	0	0	0	0	0	1	0	768	CURRENT ADDRESS	READ
0	0	0	1	0	1	0	0	769	BASE&CURRENT WORD	WRITE
0	0	0	1	0	0	1	0	769	CURRENT WORD COUNT	READ
0	0	1	0	0	1	0	1	770	BASE&CURRENT ADDR	WRITE
0	0	1	0	0	0	1	1	770	CURRENT ADDRESS	READ
0	0	1	1	0	1	0	1	771	BASE&CURRENT WORD	WRITE
0	0	1	1	0	0	1	1	771	CURRENT WORD COUNT	READ
0	1	0	0	0	1	0	2	772	BASE&CURRENT ADDR	WRITE
0	1	0	0	0	0	1	2	772	CURRENT ADDRESS	READ
0	1	0	1	0	1	0	2	773	BASE&CURRENT WORD	WRITE
0	1	0	1	0	0	1	2	773	CURRENT WORD COUNT	READ
0	1	1	0	0	1	0	3	774	BASE&CURRENT ADDR	WRITE
0	1	1	0	0	0	1	3	774	CURRENT ADDRESS	READ
0	1	1	1	0	1	0	3	775	BASE&CURRENT WORD	WRITE
0	1	1	1	0	0	1	3	775	CURRENT WORD COUNT	READ

A3	A2	A1	A0	IOR	IOW	PUERTO	OPERACION
1	0	0	0	1	0	776	WRITE COMMAND REGISTER
1	0	0	0	0	1	776	READ STATUS REGISTER
1	0	0	1	1	0	777	WRITE REQUEST REGISTER
1	0	0	1	0	1	777	ILEGAL
1	0	1	0	1	0	778	WRITE SINGLE MASK REGISTER BIT
1	0	1	0	0	1	778	ILEGAL
1	0	1	1	1	0	779	WRITE MODE REGISTER
1	0	1	1	0	1	779	ILEGAL
1	1	0	0	1	0	780	CLEAR BYTE POINTER FLIP-FLOP
1	1	0	0	0	1	780	ILEGAL
1	1	0	1	1	0	781	MASTER CLEAR
1	1	0	1	0	1	781	READ TEMPORARY REGISTER
1	1	1	0	1	0	782	ILEGAL
1	1	1	0	0	1	782	ILEGAL
1	1	1	1	1	0	783	WRITE ALL MASK REGISTER BITS
1	1	1	1	0	1	783	ILEGAL

PROCESADOR TMS32010:

-Características del procesador de señal digital TMS32010:

-Descripción general:

El TMS32010 es un procesador cuyas características están especialmente orientadas a facilitar el proceso de señal digital. Esquemáticamente, sus características más destacables son las siguientes:

-Ciclo de instrucción de 200 ns.

-288 bytes (144 palaras) de RAM destinados al almacenamiento de datos en el propio chip.

-Existen dos versiones:

La versión sin ROM.- TMS32010

y

la versión con 3K bytes de ROM integrados en el propio chip.- TMS320M10

-Posibilidad de trabajar a máxima velocidad con memoria externa hasta un total de 8K bytes (4K palabras).

-Palabra de instrucción/datos de 16 bits.

-Acumulador y ALU de 32 bits.

-Multiplicación de 16*16 bits en 200 ns.

-Desplazador(shifter) de 0 a 15 bits.

-Ocho canales de entrada y ocho canales de salida.

-Bus de datos de 16 bits con 40 megabits por segundo de velocidad de transferencia.

-Interrupción con almacenamiento completo del contexto.

-Aritmética en complemento a dos con signo y trabajando en punto fijo.

-Tecnología NMOS de 2.7 micron.

-Trabaja con alimentación de 5v.

-Integrado de 40 patillas tipo DIP.

La única diferencia entre el TMS320M10 y el TMS32010 consiste en que el primero posee una ROM en el propio integrado, mientras que el segundo utiliza memoria externa.

-Glosario de elementos hardware, sus correspondientes símbolos y la función que realizan:

-Acumulador(ACC): Acumulador de 32 bits.

-Unidad Aritmético-lógica(ALU): Unidad aritmético-lógica de 32 bits.

-Registros auxiliares (AR0 y AR1): Dos registros de 16 bits para el direccionamiento indirecto de la memoria de datos y control de de la cuenta de bucles. Los nueve bits menos significativos de cada registro están configurados como contadores bidireccionales.

-Registro puntero auxiliar (ARP): Contiene un único bit que apunta a uno de los dos registros auxiliares.

-Bus de datos (D Bus): Bus de datos de 16 bits para el acceso directo a memoria.

-Puntero de la página de memoria de datos (DP):
La memoria de datos interna del procesador está dividida en dos páginas: La primera contiene las 128 primeras posiciones de memoria y la segunda las 16 restantes; este registro de un bit apunta a la página con la que se esté trabajando en ese momento.

-RAM de datos: 144x16-bit palabras de memoria de acceso aleatorio para el almacenamiento de datos.

-Registro de bandera de interrupción (INTF):
Registro de bandera de un solo bit que indica que ha ocurrido un requerimiento de interrupción (petición pendiente).

-Registro de modo de interrupción (INTM):
Registro de mode de un solo bit que actúa como máscara respecto a la bandera de interrupción.

-Multiplicador: Multiplicador hardware de 16x16 bits en paralelo.

-Registro de bandera de overflow (OV): Registro de un solo bit que indica un overflow en las operaciones aritméticas.

-Registro de modo de overflow (OVM): Registro de modo de un solo bit que define el modo de saturación o no saturación en las operaciones aritméticas.

-Registro P (P): Registro de 32 bits donde se almacena el resultado de la operación de multiplicación.

-Bus de programa (P Bus): Bus de 16 bits para la ruta de instrucciones desde la memoria de programa.

-Contador de Programa (PC): Registro de 12 bits que contiene la dirección de la memoria de programa.

-ROM de programa: ROM de 1536x16-bit palabras que contienen el código del programa (En el caso del TMS320M10 únicamente).

-Desplazadores (Shifsters): Posee dos, un desplazador aritmético que mueve datos de la RAM a la ALU con un desplazamiento hacia la izquierda de 0 a 15 bits, y otro que actúa en el acumulador cuando almacena los datos en la DATA RAM; puede desplazar hacia la izquierda en una cantidad de 0,1 o 4 bits.

-Pila: Cuatro registros de 12 bits para el almacenamiento del contador de programa en el caso de llamadas a subrutinas e interrupciones.

-Registro T (T): Registro de 16 bits que contiene el multiplicando durante las operaciones de multiplicación.

-Arquitectura:

-Arquitectura Harvard:

En una arquitectura Harvard, la memoria de programa y de datos se estructuran en dos espacios separados, permitiendo traslaparse totalmente los ciclos de búsqueda y ejecución de la instrucción.

El procesador TMS32010 trabaja con una arquitectura Harvard modificada, en la cual la memoria de datos y la de programas yacen en dos espacios separados, pero en la que se permite la transferencia de información entre ambos espacios, obteniéndose con ello un aumento de la flexibilidad a la hora del desarrollo de aplicaciones.

Esta modificación permite por ejemplo el traspaso de coeficientes almacenados en la memoria de programa a la memoria de datos, con lo que se elimina la necesidad de una ROM separada para el almacenamiento de los coeficientes.

La memoria de programa puede venir incorporada en el propio chip, en la forma de una ROM de 1536x16-palabras, o ser externa.

La máxima cantidad de memoria de programa que puede ser direccionada directamente por el bus de direcciones es de 4Kx16-bits, al ser el bus de direcciones de 12 bits; este espacio de direcciones , auque bastante pequeño, resulta suficiente dada la orientación de este dispositivo hacia el campo del proceso de señal, para el que normalmente se emplean programas relativamente cortos en los que prima la optimización para obtener la máxima velocidad.

La ejecución de instrucciones, trabajando con memoria externa, puede realizarse a la velocidad máxima; para ello se requieren memorias rápidas con tiempos de acceso por debajo de los 100ns. Este tipo de memorias son difíciles de conseguir, y muy caras.

La memoria interna de datos es de 144x16-bits. Los operandos indicados por las instrucciones se extraen de esta RAM, no existiendo la posibilidad de que se obtengan directamente de la memoria de programa.

En cualquier caso, los datos pueden introducirse en la memoria de datos desde un periférico mediante el uso de la instrucción IN, o pueden leerse desde la memoria de programa mediante el uso de la instrucción TBLR (table read).

La instrucción OUT escribirá una palabra desde la RAM de datos a un periférico, mientras que la instrucción TBLW escribirá una palabra de la memoria de datos a la de instrucciones.

-CARACTERISTICAS ARITMETICAS:

El TMS32010 posee cuatro elementos aritméticos básicos:
La ALU, el acumulador, el multiplicador, y los desplazadores.
Todas las operaciones aritméticas se realizan mediante la aritmética del complemento a dos.
La mayor parte de las operaciones aritméticas accederán a una palabra de la memoria de datos, tanto directa como indirectamente, y la pasarán a través del desplazador (barrel shifter). Este desplazador pude desplazar el dato hacia la izquierda de 0 a 15 bits dependiendo del valor especificado por la instrucción. El operando es introducido entonces en la ALU, donde será añadido, sustraido o cargado en el acumulador.

Una vez obtenido el resultado en el acumulador, éste puede almacenarse en la memoria de datos; como el resultado es de 32 bits, habrá de almacenarse como dos palabras de 16 bits, aunque cabe la posibilidad de que no sea necesaria mantener la precisión de 32 bits y baste con almacenar el resultado en una única palabra de 16 bits.

A la salida del acumulador, existe un segundo desplazador en paralelo, que permite escalar los resultados mientras que se almacenan en la memoria de datos.

-ALU:

La ALU es una unidad aritmético-lógica de propósito general que trabaja con palabras de 32 bits.
Esta unidad suma, resta, y realiza operaciones lógicas.

El acumulador es siempre el destino y el operando primario.
Las operaciones lógicas se realizan como siguen:

Operación XOR:
Bits 31-16 del ACC=(Cero)(OR EXCLUSIVE)(bits 31-16 del ACC)
Bits 15-0 del ACC=(Valor)(OR EXCLUSIVE)(bits 15-0 del ACC)

Operación AND:
Bits 31-16 del ACC=(Cero).(bits 31-16 del ACC)
Bits 15-0 del ACC=(Valor del dato).(bits 15-0 del ACC)

Operación OR:

Bits 31-16 del ACC=(Cero)+(bits 31-16 del ACC)

Bits 15-0 del ACC=(Dato de memoria)+(bits 15-0 del ACC)

Modo de overflow (OVM):

El registro OVM se controla directamente por software, poniéndose a UNO mediante la instrucción SOVM y reseteándose mediante la instrucción ROVM.

Si ocurre un overflow cuando el registro OVM está puesto a uno, se cargará según sea el caso el valor más positivo, o el valor más negativo en el acumulador. Este valor quedará determinado por el signo del overflow.

-ACUMULADOR:

EL acumulador almacena los resultados obtenidos por la ALU, y sirve con frecuencia de operando de entrada en la propia ALU.

Trabaja con palabras de 32 bits y está dividido en dos palabras de 16 bits: la de mayor orden, que comprende los bits del 31 al 16, y la de menor orden, que comprende los bits del 15 al 0.

EL juego de instrucciones provee las instrucciones SACL (STORE ACCUMULATOR LOW) y SACH (STORE ACCUMULATOR HIGH) para almacenar ambas palabras por separado.

Estado del acumulador:

El estado de overflow del acumulador viene indicado por el registro bandera de overflow del acumulador (OV).

Este registro se pondrá a UNO al ocurrir un overflow en el acumulador. Como el registro OV es una parte del registro de estado, podrá ser almacenado en la memoria de datos.

Cuando la bandera de overflow es puesta a UNO, únicamente la ejecución de la instrucción de salto en overflow (BV) o la modificación directa del registro de estado pueden resetearla. Esta característica, permite el examinar si ha habido o no overflow fuera de bucles cuya ejecución es crítica en el tiempo.

Existe otra serie de instrucciones de salto condicional, en las que la ejecución o no de dicho salto dependerá del estado del acumulador.

Estas instrucciones son:

BLZ, que salta si ACC<0

BLEZ, que salta si ACC<=0

BGZ, que salta si ACC>0

BGEZ, que salta si ACC>=0

BNZ, que salta si ACC<>0

BZ, que salta si ACC=0

-EL MULTIPLICADOR:

El multiplicador de 16x16 bits en paralelo, consta de tres unidades: el registro T, el registro P, y el multiplicador propiamente dicho.

El registro T es un registro de 16 bits que almacena el multiplicando, mientras que el registro P es un registro de 32 bits que almacena el resultado de la multiplicación.

A fin de realizar la multiplicación, habrá que cargar primero el multiplicando en el registro T mediante una de las instrucciones LT, LTA ó LTD; a continuación se ejecutará la instrucción MPY o la instrucción MPYK.

La instrucción MPY multiplica el registro T por un valor de 16 bits de la memoria de datos, direccionado en dicha instrucción.

La instrucción MPYK multiplicará al registro T por una constante de 13 bits, que se indica como operando de dicha instrucción. Esta constante está justificada a la derecha, y con extensión de signo.

El producto obtenido, que se almacena en el registro P, puede añadirse, sustraerse o cargarse en el acumulador mediante la ejecución de una de las siguientes instrucciones: PAC, APAC, SPAC, LTA ó LTD.

El hardware del TMS32010, está preparado para inhibir una interrupción hasta que la instrucción siguiente a la de multiplicación se haya ejecutado, con el fin de evitar que el resultado de dicha operación, almacenado en el registro P, se pierda. Por ello es conveniente que la instrucción de multiplicación se vea siempre seguida inmediatamente por otra de las vistas anteriormente que almacenan el resultado.

-DESPLAZADORES (SHIFTERS):

El TMS32010 posee dos desplazadores para la manipulación de los datos: un desplazador (el barrel shifter) que actúa al

cargar un dato de la RAM a la ALU, y un desplazador en paralelo que actúa al pasar el resultado de la ALU a la RAM.

Barrel Shifter:

Realiza un desplazamiento de 0 a 15 bits hacia la izquierda en todos los valores de la memoria que se sumen, resten o carguen en el acumulador merced a las intrucciones ADD, SUB O LAC.

Este desplazador rellena con ceros los bits menos significativos, y realiza la extensión de signo de la palabra de memoria de 16 bits a la palabra de 32 bits del acumulador, lo que se conoce como desplazamiento aritmético hacia la izquierda. Esto significa que los bits a la izquierda del MSB del dato en cuestión se rellenarán con unos si el MSB es uno, o con ceros si el MSB es cero.

Se diferencia del desplazamiento lógico en que este último rellena siempre de ceros los bits a la izquierda del más significativo.

Resulta fácil el efectuar desplazamientos hacia la derecha, ya sean aritméticos o lógicos, a partir de este desplazador, pudiéndose implementar con pocas instrucciones.

Desplazador paralelo (Parallel shifter):

Este desplazador se activa únicamente mediante la instrucción SACH (Store ACcumulator High), y actúa desplazando hacia la izquierda la palabra completa de 32 bits del acumulador el número de bits indicado, para luego almacenar los 16 bits de mayor peso resultantes en la memoria de datos.

Esto implica la pérdida de los bits de mayor peso en el dato de memoria resultante.

Este desplazador puede realizar desplazamientos de 0, 1 o 4 bits únicamente, no existiendo la posibilidad de realizar desplazamientos hacia la derecha.

El valor del acumulador permanece igual después de efectuar esta operación.

-MEMORIA DE DATOS:

La memoria de datos consiste en 144 palabras de 16 bits de RAM pertenecientes al propio TMS32010. Cualquier referencia a algún operando que no sea inmediato, lo será a alguna posición de esta memoria.

Con el fin de permitir el trasvase de información entre ésta memoria y la de programas existen dos instrucciones:

La instrucción TBLR (Table Read), que transfiere valores de la memoria de programa a la de datos, y la TBLW (Table Write), que transfiere valores de la memoria de datos a la de programas. Ambas instrucciones emplean tres ciclos en su ejecución.

Otra forma de introducir o extraer información en esta RAM, esta vez desde o hacia un periférico, es mediante las instrucciones IN y OUT, de entrada y salida respectivamente.

Si el periférico en cuestión consiste en memoria externa estructurada en forma de pila, las instrucciones IN y OUT permitirán el acceso a una cantidad masiva de memoria.

Para ello habrá que proveer al sistema del hardware adicional que permita el control de esta información.
Este método tiene la ventaja de que las intrucciones IN y OUT requieren sólo dos ciclos para su ejecución, mientras que las instrucciones TBLR y TBLW requerían tres.

-Direccionamiento de la memoria de datos:

Existen dos formas de direccionar esta memoria: El direccionamiento directo, y el direccionamiento indirecto (mediante el uso de los registros auxiliares).

-Direccionamiento indirecto:

Este método de direccionamiento emplea los 8 bits menos significativos de los registros auxiliares para obtener la dirección de la memoria de datos, lo cual es más que suficiente para direccionar las 144 palabras de que consta. No es necesaria la paginación con el método de direccionamiento indirecto (recordar que la memoria de datos está paginada en una página de 128 palabras y la otra de 16). El registro auxiliar del que se obtiene la información de la dirección, se indica mediante el puntero de registro auxiliar (ARP). Existe la posibilidad de que el registro auxiliar usado por una instrucción de direccionamiento indirecto se incremente o decremente tras la ejecución de la instrucción, con el consiguiente ahorro de código y tiempo si por ejemplo se trabaja con tablas en memoria.

-Direccionamiento directo:

En el direccionamiento directo se emplean siete bits de la instrucción, junto con el puntero de página de memoria (DP) para formar la dirección de la memoria de datos.

El direccionamiento directo usa el siguiente esquema de paginación:

DP	POSICIONES DE MEMORIA
0	0-127
1	128-244

Normalmente, en la segunda página de memoria se almacenan los valores a los que se accede con menos frecuencia, para no tener que estar variando el puntero de página.

El puntero de página (DP) es parte del registro de estado, y como tal puede almacenarse en la memoria de estado.

-Registros:

-Registros auxiliares:

Son dos registros de 16 bits, que se denominan AR0 y AR1, y pueden usarse con las siguientes finalidades:

-Almacenamiento temporal.

-Direccionamiento de memoria indirecto: Para lo que se emplean los 8 bits menos significativos.

-Control de bucles: La instrucción BANZ (Branch on Auxiliar register Non Zero), que realiza el salto si el registro

auxiliar es distinto de cero, permite utilizar estos registros como contadores de lazo.

BANZ comprueba si el registro auxiliar es cero, en caso de que no lo sea, lo decrementa y salta.

Un ejemplo de bucle es el siguiente:

```
    LARP AR0          Se carga el registro ARP con 0
    LARK AR0,8        Se carga AR0 con 8
BL  IN *,PA0          Lee dato por el puerto cero
    BANZ BL
```

Este programa lee nueve veces el puerto de entrada mediante la instrucción IN y almacena los valores obtenidos en las posiciones de memoria de la 8 a la cero.

Cuando los registros auxiliares son incrementados o decrementados por una instrucción de acceso indirecto a memoria o por la instrucción BANZ, esta variación afecta a los nueve bits de menor peso, en lugar de los ocho bits que se usan para el direccionamiento indirecto. Esto es así porque se usan los ocho bits menos significativos cuando se realiza un direccionamiento indirecto, y los nueve menos significativos si el registro auxiliar en cuestión se va a usar como contador. Esta porción contadora de un registro auxiliar trabaja como un contador circular, en el que al decrementar el valor 000000000 se obtiene el valor 111111111, y al incrementar el valor 111111111 se obtiene el 000000000,

sin que se vean afectados ninguno de los bits que no pertenecen al contador.

Evidentemente, los valores que puede adoptar este contador oscilan entre el 0 y el 512

El valor de los registros auxiliares se puede guardar en la memoria de datos mediante la instrucción SAR, y cargar de ella mediante la instrucción LAR. Ambas instrucciones trabajan con los 16 bits completos de los registros auxiliares, aunque para el direccionamiento indirecto y control de bucles tan solo se emplee una porción de ellos.

-Puntero de registro auxiliar (ARP):

Este puntero consiste en un único bit que es parte del registro de estado, y su función es la de indicar con cual de ambos registros se trabaja por defecto.

Si su valor es cero, indica que el registro actual es el cero, y si es uno indica que es el uno.

Como parte del registro de estado, el ARP puede almacenarse en la memoria de datos.

-MEMORIA DE PROGRAMAS:

La memoria de programas consiste en hasta 4K palabras de 16 bits. El TMS320M10 tiene incluida una ROM de 1536 palabras, mientras que el TMS32010, que es el que realmente nos ocupa, viene sin ROM. Ambos modos de operación se seleccionan mediante la patilla MC/MP.

-Modos de operación:

Son el modo de microcomputador y el modo de microprocesador. El primero se selecciona poniendo la patilla MC/MP a nivel alto, y el segundo poniéndola a nivel bajo.

Tan solo el TMS320M10 permite el elegir entre ambos modos, puesto que el TMS32010 no posee la posibilidad de actuar en modo microcomputador.

Modo microcomputador (TMS320M10):

Este modo trabaja con una ROM de 1536 palabras de 16 bits, aunque únicamente las posiciones de la 0 a la 1523 son accesibles al programa de usuario, al estar las posiciones 1524 a la 1535 reservadas por Texas Instruments con propósitos de verificación.

La arquitectura del sistema permite el uso de 2560 posiciones adicionales de memoria externa.

Modo microprocesador (TMS32010 y TMS320M10):

Queda indicado por un cero en la patilla MC/MP. El conjunto de las 4K palbras de memoria son externas en este modo.

Las posiciones 0 y 1 de memoria contienen el vector de reset. Después de un reset el Contador de Programa (PC) se inicializa a cero. Normalmente, en estas posiciones 0 y 1 se sitúa una instrucción de salto a la rutina de tratamiento del reset.

Las posiciones 2 y 3 de la memoria de programa contienen el vector de interrupción. Tras una interrupción, el PC apuntará

a la posición 2 de memoria, a partir de la cual se efectuará el tratamiento de la misma. Queda a elección de el programador el ejecutar un salto a una rutina de tratamiento de interrupción, o el realizar dicho tratamiento inmediatamente. Evidentemente, el TMS32010 no precisa de una tabla de vectores de interrupción propiamente dicha, puesto que posee una única interrupción.

-El uso de memoria externa:

El TMS32010 posee 12 patillas (de la A11 a la A0) para el direccionamiento de la memoria externa. En ellas se bufferiza el contenido del PC, o la dirección del puerto de E/S si se trabaja con puertos.

Para leer la siguiente instrucción de la memoria externa, la patilla MEN (Memory ENable) se pone a nivel bajo, lo que permite que la RAM externa abandone el estado de alta impedancia y sitúe en el bus de datos el código de la siguiente instrucción, que es leído por el TMS32010.

Si se trabaja en el modo microcomputador, el procesador seleccionará internamente las 1536 posiciones de memoria de la ROM interna. En éstas circunstancias, tanto la patilla MEN como el bus de direcciones seguirán trabajando igual, pero la instrucción se leerá de la memoria interna.

Incluso cuando se trabaja con memoria externa, el ciclo de instrucción del procesador sigue siendo de 200ns, por lo que dichas memorias externas habrán de poseer como máximo un

tiempo de acceso de 100ns. Es preferible que dicho tiempo de acceso sea lo más reducido posible, con el fin de permitir el añadir una cierta lógica externa para el acceso de esta memoria externa.

La forma más sencilla de trabajar con la memoria externa, es conectando la patilla MEN a la patilla OUTPUT ENABLE de la RAM, la patilla WE a la Write Enable de la memoria y conectando el Chip Select de la memoria a un AND de las patillas MEN y WE del procesador.

Este método adolecerá del defecto de no distinguir las instrucciones de acceso a memoria de las de Entrada/Salida, con lo que si por ejemplo se ejecuta una instrucción IN de entrada por el puerto cero, el valor que se lea se almacenará en la posición cero de la memoria. Para evitar este problema habrá que decodificar el Bus de direcciones, con lo que se complicará la lógica del sistema.

-CONTADOR DE PROGRAMA Y PILA:

El contador de programa y la pila permiten al usuario efectuar saltos, llamadas a subrutinas e interrupciones, y ejecutar las instrucciones de lectura de tabla (TBLR) y escritura de tabla (TBLW).

-Contador de programa (PC):

El contador de programa (PC), es un registro de 12 bits que contiene la dirección de la memoria de programa de la próxima instrucción a ejecutar. Cuando el sistema va a buscar una

instrucción de la memoria de programa, obtine la dirección de este registro PC, y a continuación lo incrementa, con el fin de que apunte a la próxima instrucción.

El PC se inicializa a cero tras haber aplicado un reset (poniendo la patilla de reset a nivel bajo).

Con el fin de direccionar la memoria externa, el contenido del PC se bufferiza a los pines que comprenden el bus de direcciones, del A11 al A0.

Los tres bits menos significativos del bus de direcciones (A2..A0), están multiplexados con las patillas de diteccionamiento de los puertos de Entrada/Salida, PA2 a PA0. Estas patillas son también usadas por las instrucciones IN y OUT.

Al ejecutar una instrucción de salto, lo que ésta hace es cambiar el contenido del PC por la posición de memoria a la que vaya a saltar. Si la instrucción de salto es condicional, y la condición para dicho salto no se cumple, lo que hará entonces es aumentar en una unidad el contenido del PC.

-La pila:

La pila posee 12 bits de ancho y cuatro niveles de profundidad.

La instrucción PUSH empuja los doce bits menos significativos de el acumulador a la cima de la pila (TOS- Top Of Stack).

La instrucción POP saca de la pila el valor situado en su cima y lo carga en los 12 bits menos significativos del acumulador.

La cima de la pila puede cargarse en la memoria de datos mediante el proceso de ejecutar la instrucción POP seguido de la instrucción SACL, que almacena la palabra de menor orden del acumulador en la memoria de datos. Esto permite la expansión de la pila en la memoria de datos, y de ésta es posible expandirla a la memoria de programas mediante el uso de la instrucción TBLW.
De esta forma es posible expandir la pila, que de otra forma se quedaría muy corta para ciertas aplicaciones.
Si se trabaja con el emulador XDS/320, uno de los niveles de la pila queda reservado para uso del propio emulador, por lo que el número de niveles disponibles se reduce a tres.

-Overflow de la pila:
La pila puede permitir hasta cuatro subrutinas anidadas o interrupciones sin que exista overflow, siempre y cuando no se utilicen las instrucciones TBLR ó TBLW, que al utilizar un nivel de pila reducen los niveles antes mencionados de 4 a 3.
Si existe un overflow en la pila, la información almacenada en el nivel más bajo se perderá.
Si se continuan extrayendo valores de la pila, una vez que ésta está vacía, el valor situado en el fondo de la pila se irá copiando en los niveles superiores hasta llenar la pila.

Para trabajar con subrutinas e interrupciones de mucho mayor nivel de anidamiento, habrá que dedicar parte de la RAM, ya sea la externa o la de datos, a la gestión de la pila.

En este caso, la cima de la pila habrá de ser sacada inmediatamente al empezar una subrutina o una rutina de interrupción y almacenada en la RAM

Al finalizar la subrutina o la rutina de interrupción, el valor de la pila almacenada en la RAM se devuelve a la cima de la pila, antes de devolver el control a la rutina principal.

REGISTRO DE ESTADO:

Está constituido por cinco bits de estado.
Estos bits pueden alterarse individualmente mediante el uso de instrucciones adecuadas. Existe además la posibilidad de almacenar este registro en la memoria de datos, mediante el uso de la instrucción SST, así como la de cargar un nuevo valor merced a la instrucción LST, con excepción de la bandera de interrupción, el bit INTM.
Dicho bit no se ve afectado por el uso de la instrucción LST, pudiendo ser cambiado únicamente mediante la ejecución de las instrucciones EINT (Enable INTerruption) y DINT (Disable INTerruption).

Su estructura es la siguiente:

OV	OVM	INTM	ARP	DP

El significado de cada uno de los bits es el siguiente:

Puntero de página de memoria (DP-Data Memory Page Pointer).-
Si tiene el valor cero apunta a las primeras 128 palabras, si su valor es uno selecciona las 16 palabras de memoria restantes.
Su valor pude cambiarse además de con la instrucción SST, mediante la ejecución de las instrucciones LDP y LDPK.

Puntero de registro auxiliar (ARP-Auxiliary Register Pointer):
El valor cero selecciona el registro AR0 y el valor uno selecciona el registro AR1.
Su valor pude cambiarse además de con la instrucción SST, mediante las instrucciones MAR (Modify Auxiliary Register) y LARP (Load Auxiliary Register Pointer), así como mediante la ejecución de instrucciones que permitan el direccionamiento indirecto.

Bit de máscara de interrupción (INTM-Interrupt Mask Bit):
Si tiene el valor cero indica que las interrupciones están permitidas, mientras que el valor uno las inhibe.
La instrucción EINT coloca el valor de INTM a cero, mientras que la instrucción DINT lo pone a UNO.
Al ejecutarse una interrupción, el registro INTM se pone automáticamente a uno, con lo que se evita el que ocurra otra interrupción mientras que no se vuelva a habilitar mediante EINT.
El valor de INTM únicamente se puede alterar mediante EINT y DINT.

Bit de modo de overflow (OVM-OVERFLOW MODE BIT):

Si tiene el valor UNO indica que el modo de overflow está permitido, y adopta este valor mediante la ejecución de la instrucción SOVM (Set OVerflow Mode).
El valor cero inhabilita dicho modo, y se obtine mediante la instrucción ROVM (Reset OVerflow Mode).

Registro bandera de overflow del acumulador (OV):

Si tiene el valor uno indica que ha habido un overflow en el acumulador; el valor cero por tanto indica que no ha habido overflow.
La instrucción BV (Branch on oVerflow) reseteará este bit, y ejecutará un salto si su valor es uno.

-Almacenamiento del registro de estado:

El contenido del registro de estado puede cargarse en la memoria de datos mediante la ejecución de la instrucción SST.
Si la instrucción SST se ejecuta utilizando el modo de direccionamiento directo, el sistema seleccionará automáticamente la segunda página de memoria, con lo cual tan solo podrá ser almacenado en una de las 16 posiciones de memoria que pertenecen a dicha página.
Si se emplea el direccionamiento indirecto no existe este problema, y el contenido del registro podrá almanarse en cualquier posición de memoria.

La instrucción SST no modifica el contenido del registro de estado.

Al cargarse en la RAM, los bits de estado aparecen según la estructura siguiente:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OV	OVM	INTM	1	1	1	1	ARP	1	1	1	1	1	1	X	DP

La instrucción LST podrá ejecutarse para cargar el registro de estado. A diferencia de la SST, la instrucción LST no asume que el registro de estado esté en la página uno (segunda página), por lo que si dicho registro está almacenado en la página uno, habrá que poner previamente el bit de DP a uno. Todos los bits de estado, con la excepción del INTM, pueden ser cambiados mediante esta instrucción.

El bus de datos bidireccional externo, estará siempre en el modo de alta impedancia, excepto cuando la patilla Write Enable se ponga a nivel bajo. Dicha patilla es activada únicamente por las instrucciones OUT y TBLW.

-OPERACIONES DE ENTRADA/SALIDA:

-Las instrucciones IN y OUT:

La entrada de datos desde los periféricos se realiza mediante la ejecución de la instrucción IN.

Para ello se direcciona el puerto mediante las patillas PA2,PA1 y PA0, que están multiplexadas con los tres bits menos significativos del bus de direcciones A2,A1 y A0, se pone la patilla DEN a nivel bajo, y se lee por el bus de datos de 16 bits el valor en cuestión.

La salida de datos hacia los periféricos se hace mediante la instrucción OUT. El proceso es análogo al de la entrada de datos:
Se direcciona el puerto mediante las patillas PA2,PA1 y PA0, se bufferiza por el bus de datos el dato de salida, y se pone a nivel bajo la patilla Write Enable.
El hecho de que hayan 3 patillas (PA2,PA1 y PA0) para la selección de los puertos implica que existen ocho puertos de entrada y ocho de salida.

Las patillas del bus de direcciones que no están multiplexadas con las del bus de datos, es decir, de la A11 a la A3, se ponen a nivel bajo durante la ejecución de las instrucciones IN y OUT.
Esto permite distinguir las instrucciones OUT y TBLW, mediante la decodificación del bus de direcciones, ya que por lo demás actúan de forma similar al activar la patilla WE cuando van a sacar un dato por el bus de datos.
Este sistema tiene el inconveniente de no permitir escribir un dato en la memoria de programa en las ocho primeras posiciones de memoria mediante la instrucción TBLW, ya que si se detecta

una dirección perteneciente a ese rango, se sobreentiende que corresponde a un puerto de Entrada/Salida.

-Las instrucciones TBLR y TBLW:

Estas instrucciones permiten la transferencia de palabras entre las memorias de datos y de programas.

TBLR (Table Read) lee palabras de la memoria de programas y las transfiere a la memoria de datos.

TBLW (Table Write) lee palabras de la memoria de datos y las transfiere a la de programas.

Al ejecutar la instrucción TBLR, se activa la patilla MEN, que provoca que el contenido de la posición de memoria indicada aparezca en el bus de datos.

Al ejecutar la instrucción TBLW, se activa la patilla WE, patilla que es activada igualmente por la ejecución de la instrucción OUT.

Hay que hacer notar, que aunque se trabaje en el modo microcomputador y se realice la operación de TBLW para direcciones que pertenecen al rango de la ROM, el TMS seguirá activando la patilla WE y sacando por el bus de direcciones la dirección en cuestión.

-La patilla BIO:

Esta patilla trabaja en conjunción con la instrucción BIOZ, de salto condicional, que provoca el salto si dicha patilla está a

nivel bajo. Es pues un pin de entrada.

Resulta muy útil para controlar o coordinar sucesos externos.

El ejemplo más común es el de generar un bucle de espera hasta que esta patilla BIO se ponga a nivel alto, lo que permite por ejemplo el esperar hasta que un convertidor Analógico/Digital haya terminado la siguiente conversión, momento en que merced a su patilla End Of Conversion ponga el pin BIO a nivel alto y el procesador lea el dato.

Veamos un programa ejemplo que ralice el proceso arriba descito:

```
0000  ESP  BIOZ ESP    ;ESPERA SIGUIENTE VALOR
0001       IN   0,PA0  ;LEE VALOR POR PUERTO 0
....       .........  ;       PROCESA
....       .........  ;         EL
....       .........  ;        DATO
XXXX       B    ESP   ;PASA A LEER EL  SIGUIENTE
                       ;DATO
```

-INTERRUPCIONES:

La interrupción se activa tanto por un flanco de bajada en la patilla INT como manteniendo dicha patilla a nivel bajo.

Las interrupciones quedan deshabilitadas, bien mediante la instrucción DINT o bien mediante un RESET, mientras que la instrucción EINT las habilita.

-RESET:

El procesador entra en estado RESET cuando se pone la patilla RS a nivel bajo durante un mínimo de 5 ciclos de reloj.
Una vez en estado RESET las líneas DEN,MEN y WE se ponen a nivel alto, adquiriendo el bus de datos el estado de alta impedancia.

Un CLEAR es aplicado al PC y al bus de direcciones en el siguiente ciclo completo de reloj tras el flanco de bajada en la patilla RS.
La patilla RS además deshabilita las interrupciones, hace un clear en el regitro bandera de interrupción, y deja igual el registro de modo de overflow.
El TMS32010 puede mantenerse en estado RESET indefinidamente.

-RELOJ/OSCILADOR:

El TMS32010 puede tanto hacer uso de su oscilador interno como trabajar con una fuente de reloj externa.
Para hacer uso del oscilador interno hay que conectar un cristal entre las patillas X1 y X2/CLKIN. La frecuencia de CLKOUT y el ciclo de reloj del TMS32010 es un cuarto de la frecuencia fundamental del cristal.

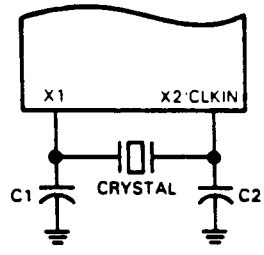


FIGURE 2-14 - INTERNAL CLOCK

Se puede usar una fuente de reloj externa, conectándola directamente a la patilla X2/CLKIN, y dejando sin conectar la patilla X1.

En caso de trabajar con fuente externa, será necesario el uso de una resistencia de pull-up, puesto que la tensión de nivel alto que habrá que entregar a la entrada de CLKIN tendrá que ser de un mínimo de 2.8 voltios, mientras que un nivel estándar TTL puede ser de un mínimo de 2.4 voltios.

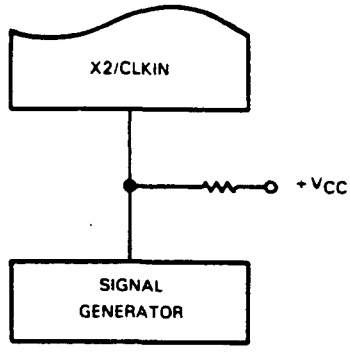


FIGURE 2-15 – EXTERNAL FREQUENCY SOURCE

El valor de la resistencia de pull-up se definirá en función de factores como la tensión y corriente a nivel alto de la fuente de frecuencia, así como el número de otros sistemas que estén conectados a dicha fuente de frecuencia.

Habrá que intentar adoptar un valor de resistencia lo mayor posible dentro de el rango de valores para los que cumpla los requisitos de la entrada CLKIN.

El tiempo de retraso entre CLKIN y CLKOUT queda sin especificar, puesto que puede variar tanto como un ciclo de CLKOUT y es muy dependiente de la temperatura.

RESUMEN DELJUEGO DE INSTRUCCIONES:

El juego de instrucciones del TMS32010 es, de forma escueta y por orden alfabético, el siguiente:

-ABS(ABsolute value of Accumulator):

Si el acumulador es mayor que cero, entonces no se ve afectado por la ejecución de la instrucción. Si es menor que cero, entoces su contido se cambiará por el complemento a dos de su valor.

Existe una situación excepcional si el acumulador tiene como contenido el valor hexadecimal >80000000:

Si el modo de overflow está activo, la ejecución de esta instrucción cambiará el contenido del acumulador de >80000000 a >7FFFFFFF.

Si el modo de overflow no está activo, el acumulador no se verá afectado.

-ADD(ADD to accumulator with shift):

El contenido de la posición de la memoria de datos direccionada es desplazado hacia la izquierda y sumado al acumulador.

El resultado se almacena en el acumulador.

El direccionamiento puede ser directo o indirecto.

El desplazamiento puede ser de 0 a 15 bits.

-ADDH(ADD to High-order accumulator):

Suma el contenido de la posición de memoria de datos direccionada a la mitad superior del acumulador (bits 31 al 16).

El direccionamiento puede ser directo o indirecto.

-ADDS(ADD to low accumulator with sign-extension suppressed):

Suma el contenido de la posición de memoria direccionada al acumulador, tratando dicho contenido como un número entero de 16 bits en lugar de como un entero en complemento a dos.

-AND(AND with low-order bits of accumulator):

Se realiza un AND lógico entre los 16 bits de menor peso del acumulador y el contenido de la dirección de memoria especificada. Los 16 bits de mayor peso se colocan a cero.

-APAC(Add P register to ACcumulator):

El contenido del registro P (resultado de la mutiplicación) se suma al acumulador, y el resultado se almacena en el propia acumulador.

-B(Branch unconditionally):

Provoca un salto a la posición de memoria especificada como operando de la instrucción. Dicha especificación puede ser bien mediante un valor simbólico,o bien mediante un valor numérico.
Al ejecutarse carga el PC con la dirección indicada.

-BANZ(Branch on Auxiliary register Not Zero):

Si el registro auxiliar en cuestión es distinto de cero, entonces dicho registro es decrementado y se salta a la dirección especificada como operando. Si el registro auxiliar es cero, se decrementa y se pasa a ejecutar la siguiente instrucción.
Esta instrucción puede usarse para el control de bucles, donde el registro auxiliar con el que se esté trabajando funciona como contador.

-BGEZ(Branch if accumulator Greater Than or Equal to Zero):

Si el contenido del acumulador es mayor o igual que cero, salta a la dirección especificada. La dirección se especifica como operando de la instrucción y pude ser tanto numérica como simbólica.

-BGZ(Branch if accumulator Greater than Zero):

Si el contenido del acumulador es mayor que cero, salta a la localización de memoria especificada.

-BIOZ(Branch on I/O status equal to Zero):

Saltará a la dirección de memoria especificada si la patilla BIO está a nivel bajo, en caso contrario se incrementará el PC.
Esta instrucción puede usarse en conjución con la patilla BIO para comprobar el estado de un periférico. Este tipo de interrupción es preferible cuando se está trabajando con bucles de ejecución crítica en el tiempo.

-BLEZ(Branch if accumulator Less than or Equal to Zero):

Se producirá el salto si el contenido del acumulador es menor o igual a cero.

-BLZ(Branch if accumulator Less than Zero):

Se producirá el salto si el contenido del acumulador es menor que cero.

-BNZ(Branch if accumulator Not equal to Zero):

Se producirá el salto si el contenido del acumulador es distinto de cero.

-BV(Branch on oVerflow):

Si la bandera de overflow está activada, entonces se produce un salto a la dirección especificada y se hace un clear a la bandera de overflow. En caso contrario se pasará a ejecutar la siguiente instrucción.

-BZ(Branch if accumulator equals Zero):

Se producirá el salto si el contenido del acumulador es igual a cero.

-CALA(Call Subroutine Indirect):

El PC se incrementa y se almacena en la cima de la pila.
A continuación se salta a la dirección indicada por los 12 bits menos significativos del acumulador.

-CALL(CALL subroutine direct):

El PC se incrementa y se almacena en la cima de la pila.
La dirección de memoria de programa, que se indica como operador de la instrucción, se carga en el PC.

-DMOV(Data MOVe in memory):

El contenido de la posición de la memoria de datos direccionada es copiado en la posición de memoria inmediatamente superior.
El direccionamiento puede ser directo o indirecto.
Esta instrucción, tiene la misión de proporcionar una forma rápida y cómoda de desplazar una zona de memoria a una posición superior, lo que se necesita frecuentemente en proceso de señales.

-DINT(Disable INTerrupt):

El bit de bandera de modo de interrupción (INTM), que pertenece al registro de estado, se pone a uno, lo que indica que cualquier interrupción posterior queda inhabilitada.

-EINT (Enable INTerrupt):

El bit de bandera de modo de interrupción (INTM), que pertenece al registro de estado, se pone a cero, lo que indica que las interrupciones están habilitadas.

-IN (INput data from port):

Esta instrucción lee un dato desde un periférico, y lo almacena en la memoria de datos. Es una instrucción de dos ciclos.
Durante el primer ciclo, se envía la dirección del puerto a las patillas PA0,PA1 Y PA2; también se pone a nivel bajo la patilla DEN, indicando al periférico que ponga el valor en el bus de datos.
Durante el segundo ciclo es cuando se lee el valor del bus de datos.
El direccionamiento de la posición de memoria puede ser directo o indirecto, añadiendo además como operando el puerto por el que se ha de leer el dato, separado por una coma.

Ejemplo:

IN 5,PA7,0 --> Leer del puerto 7, almacenando el valor obtenido en la dirección 5, y cargando el registro ARP con 0.

-LAC (Load ACcumulator with shift):

El contenido de la posición de la memoria de datos direccionada es desplazado hacia la izquierda, y cargado en el acumulador. Durante el desplazamiento, los bits de menor peso son rellenados con ceros, mientras que se procede a la extensión de signo con los de mayor peso.

El direccionamiento puede ser directo o indirecto.

-LACK (Load ACumulator with eight-bit constant):

La constante de ocho bits especificada como operando se carga en el acumulador. Los 24 bits superiores del acumulador se cargan con ceros, lo que significa que se suprime la extensión de signo.

-LAR (Load Auxiliary Register):

El contenido de la posición de la memoria de datos direccionada se carga en el registro auxiliar que haya sido especificado.

El direccionamiento puede ser directo o indirecto.

Ejemplo:

```
LARP AR0
LAR  AR0,*-
```

En este ejemplo, si p.e AR0 es cero, y el contenido de la posición cero de memoria es 10, entonces se cargará AR0 con el valor 10. AR0 no resulta decrementado después de ejecutar la instrucción LAR; de forma general, se puede decir que cuando se utiliza direccionamiento indirecto con autodecrementación al

emplear la instrucción LAR para cargar el registro auxiliar actual (el mismo apuntado por ARP), el nuevo valor del registro auxiliar no se decrementa como resultado de la ejecución de la instrucción.

-LARK (Load Auxiliary Register with eight-bit constant):
La constante positiva de ocho bits se carga en el registro auxiliar especificado. Los ocho bits de mayor peso se rellenan con ceros.

-LARP (Load Auxiliary Register Pointer inmediate):
Carga el ARP (Auxiliary Register Pointer) con una constante de un bit, indicando el registro auxiliar deseado. Lógicamente, la constante de un bit tendrá que valer o cero, o uno, que es lo que se corresponde con ambos registro auxiliares.

-LDP (Load Data Page pointer):
Esta instrucción carga el registro DP (Data memory Page pointer), con el bit menos significativo del contenido de la posición de la memoria de datos especificada, ignorándose el resto de los bits.
DP=0 define la página cero, que contiene las palabras 0 a 127.
DP=1 define la página uno, que contine las palabras 128 a 143.

-LDPK (Load Data Memory Page Pointer with one-bit constant):
La constante de un bit especificada como operando se carga en el registro DP (Data memory Page pointer).
DP=0 define la página cero, que contiene las palabras 0 a 127.
DP=1 define la página uno, que contine las palabras 128 a 143.

-LST (Load STatus from data memory):

Restaura el contenido del registro de estado, almacenado mediante la instrucción SST (Store STatus) cargandolo de una palabra de la memoria de datos. Esta instrucción se emplea para cargar los bits del registro de estado después de una interrupción, o de una llamada a subrutina.

El direccionamiento puede ser directo o indirecto.

El bit de la máscara de interrupción no puede modificarse mediante el uso de esta instrucción.

Los bits del registro de estado almacenados en la memoria de datos son los siguientes:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OV	OVM	INTM	1	1	1	1	ARP	1	1	1	1	1	1	1	DP

-LT (Load T register):

Esta intrucción carga el registro T con el contenido de la posición de la memoria de datos especificada.

El direccionamiento puede ser directo o indirecto.

-LTA (Load T register and Accumulate previos result):

El contenido de la dirección especificada de la memoria de datos se almacena en el registro T; además, el registro P se añade al acumulador, almacenando el resultado de dicha suma en el acumulador.

El direccionamiento puede ser directo o indirecto.

-LTD (Load T register, add P register to Acc, and shift memory):

El contenido de la dirección especificada de la memoria de datos se almacena en el registro T; además, el registro P se añade al acumulador, y por último se copia el contenido de la dirección especificada de la memoria de datos en la posición de memoria inmediatamente superior.

El direccionamieto puede ser directo o indirecto.

-MAR (Modify Auxiliary Register):

El registro auxiliar apuntado por el registro ARP (Auxiliary Register Pointer) se incrementa, decrementa, o permanece igual. El puntero de registro auxiliar se carga con el siguiente valor, indicado como operando en la instrucción, o permanece igual.

En el modo de direccionamiento directo, MAR equivale a una instrucción NOP. Además, la instrucción LARP es un subconjunto de la instrucción MAR. Por ejemplo, MAR *,0 equivale a LARP 0.

-MPY (Multiply):

Se multiplica el contenido del registro T por el contenido de la posición de memoria de datos especificada, almacenando el resultado en el registro P.

-MPYK (Multiply immediate):

El contenido del registro T se multiplica por la constante de 13 bits con signo especificada como operando, guardando el resultado en el registro P.

-*NOP (No Operation)*:

No realiza ninguna operación. Su utilidad puede estar en servir como un retardo de 200 ns, o bien como hueco o separador por motivos de comodidad.

-OR (OR with low-order bits of accumulator):

Se efectúa un OR lógico entre el contenido de la posición de la memoria de datos especificada, y el contenido del acumulador. Los 16 bits de mayor peso del acumulador no resultan modificados.

-OUT (OUTput data to port):

Esta instrucción transfiere un dato desde la memoria de datos a un periférico externo. Es una instrucción de dos ciclos.

El primer ciclo de la instrucción sitúa la dirección del puerto en las líneas PA2 a PA0; durante este mismo ciclo, la patilla WE se pone a nivel bajo y la palabra se coloca en el bus de datos.

Durante el siguiente ciclo es cuando el periférico accede al valor.

El direccionamiento puede ser directo o indirecto.

-PAC (Load Accumulator with P register):

El resultado de la multiplicación, almacenado en el registro P, se carga en el acumulador. No tiene ningún operando.

-POP (POP top of stack to accumulator):

El contenido de la cima de la pila se carga en el acumulador. El siguiente elemento de la pila pasa a ser la cima de la misma.

No tiene ningún operando.

Los 12 bits de la pila se cargan en los bits 11 a 0 del acumulador, mientras que los bits 31 a 12 se rellenan con ceros. No se detecta si se produce un stack underflow.

-PUSH (PUSH accumulator onto stack):

El contenido del acumulador se almacena en la cima de la pila hardware. Si se produce un overflow de la pila, se perderá el contenido del fondo de la pila al ejecutar esta instrucción.
Esta instrucción no tiene ningún operando.

-RET (RETurn from subroutine):

Se extrae el elemento de la cima de la pila y se almacena en el Program Counter. Esta instrucción se usa en conjunción con CALL y CALA para la creación de subrutinas.
Esta instrucción no tiene ningún operando.

-ROVM (Reset OVerflow Mode register):

Esta instrucción reseteará el bit de registro de modo de overflow(OVM) , poniéndolo a cero.
Esta instrucción no tiene ningún operando.

-SACH (Store ACcumulator High with shift):

Esta instrucción almacena la mitad superior del acumulador en la memoria de datos con desplazamiento. El desplazamiento únicamente puede ser de 0,1 o 4 bits.
El direccionamiento puede ser directo o indirecto.

-SACL (Store Accumulator):

Esta instrucción almacena los bits de menor peso del acumulador en la memoria de datos.

No posee la posibilidad de desplazar el valor a almacenar.

El direccionamiento puede ser directo o indirecto.

-SAR (Store Auxiliary Register):

El contenido del registro auxiliar especificado, se almacena en la localización indicada de la memoria de datos. Dicha localización se puede especificar mediante direccionamiento directo o indirecto.

-SOVM (Set OVerflow Mode register):

Esta instrucción pone a uno el registro de modo de overflow (OVM), activando dicho modo.

Cuando este modo está activo, el acumulador adoptará su mayor valor positivo o negativo si ocurre un overflow o underflow respectivamente.

El mayor valor positivo es >7FFFFFFF.

El menor valor negativo es >80000000.

Esta instrucción no tiene operandos.

-SPAC (Substract P register from ACcumulator):

El contenido del registro P se le resta al acumulador, almacenando el resultado de dicha sustracción en el acumulador.

Esta instrucción no tiene operandos.

-SST (Store STatus):

Los bits de estado se almacenan en la dirección especidicada de la memoria de datos de la página 1.

Los bits del registro de estado almacenados en la memoria de datos son los siguientes:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OV	OVM	INTM	1	1	1	1	ARP	1	1	1	1	1	1	1	DP

-SUB (SUBtract from accumulator with shift):

El contenido de la posición de la memoria de datos direccionada es desplazado hacia la izquierda, y restado al acumulador. Durante el desplazamiento, los bits de menor peso son rellenados con ceros, mientras que se procede a la extensión de signo con los de mayor peso. El resultado se almacena en el propio acumulador.

El direccionamiento puede ser directo o indirecto.

-SUBC (Conditional Subtract):

Esta instrucción realiza una substracción condicional, que puede utilizarse en los algoritmos de división.

El algoritmo del funcionamiento de esta instrucción es el siguiente:

C=Contenido de la posición de memoria especificada.

$$VALOR=(ACC)-C*2^{15}.$$

```
SI (Bits de mayor peso de VALOR)>0
   ENTONCES
        ACC=VALOR*2+1
   SI NO
        ACC=(ACC)*2.
```

El direccionamiento de la posición de memoria donde se almacena V puede ser directo o indirecto.

-SUBH (SUBtract from High-order accumulator):

Esta instrucción le resta a la mitad superior del acumulador el contenido de la localización especificada de la memoria de datos, almacenando el resultado en el acumulador.

El direccionamiento puede ser directo o indirecto.

-SUBS (Substract from Low Accumulator with Sign-Extension Suppressed):

Esta instrucción le resta al acumulador el valor contenido en la localización especificada de la memoria de datos, con la extensión de signo de dicho valor suprimida.

El dato se trata como un entero de 16 bits positivo en lugar de un entero en complemento a dos.

El direccionamiento del dato puede ser directo o indirecto.

-TBLR (TaBLe Read):

Esta instrucción transfiere una palabra de cualquier lugar de la memora de programa a la localización de la memoria de datos que se especifique. El direccionamiento puede ser directo o indirecto.

Es una instrucción de tres ciclos.

El funcionamiento de esta instrucción es el siguiente:

En primer lugar se incrementa el contador de programa (PC), y se almacena en la cima de la pila.

A continuación se carga el contenido del acumulador en el PC, para luego sacar dicho valor por las líneas del bus de direcciones (A11..A0).

Luego se carga el valor leído en las líneas D15 a D0 del bus de datos en la posición de la memoria de datos especificada como operando de la instrucción.

Por último, se extrae el contenido de la cima de la pila hardware, y se carga en el PC.

-TBLW (TaBLe Read):

Esta instrucción transfiere una palabra de la localización de la memoria de datos que se especifique, a una posición de la memoria externa de programa RAM.

Es una instrucción de tres ciclos.

El funcionamiento de esta instrucción es el siguiente:

En primer lugar se incrementa el contador de programa (PC), y se almacena en la cima de la pila.

A continuación se carga el contenido del acumulador en el PC, para luego sacar dicho valor por las líneas del bus de direcciones (A11..A0).
Simultáneamente se saca el contenido de la posición de la memoria de datos direccionada como operando de la instrucción por el bus de datos del TMS32010, líneas D15 a D0.
El direccionamiento de la memoria de datos puede ser directo o indirecto.

Por último, se extrae el contenido de la cima de la pila hardware, y se carga en el PC.

-XOR (Exclusive-OR with Low-Order Bits of Accumulator):
Se efectúa un OR exclusivo entre el contenido de la posición de la memoria de datos especificada, y los 16 bits de menor peso del acumulador. Los 16 bits de mayor peso del acumulador no resultan modificados.
La posición de la memoria de datos en cuestión puede especificarse mediante direccionamiento directo o indirecto.

-ZAC (Zero the accumulator):
Los 32 bits del acumulador son puestos a cero.
Esta instrucción no tiene ningún operando.

-ZALH (Zero Accumulator and Load High):

Esta instrucción limpia el contenido del acumulador (lo pone a cero), y carga el contenido de la posición de la memoria de datos especificada en la mitad superior del acumulador. La mitad inferior del acumulador permanece a cero.

La posición de la memoria de datos en cuestión se puede especificar mediante direccionamiento directo o indirecto.

-ZALS (Zero Accumulator and load Low with Sign-extension suppressed):

Esta instrucción limpia el contenido del acumulador (lo pone a cero), y carga el contenido de la posición de la memoria de datos especificada en la mitad inferior del acumulador.

El dato se trata como un entero positivo de 16 bits, en lugar de como un número en complemento a dos. No se realiza extensión de signo.

EMULADOR XDS/22

EMULADOR XDS/22

1- Qué es un emulador.-

Cuando se diseña un sistema controlado por un microprocesador, las tareas de depuración y verificación tanto de la parte hardware como de la parte software del sistema pueden resultar harto complicadas. El uso de un emulador de dicho microprocesador se convierte en una ayuda inestimable a la hora de realizar dicho proceso.

Un emulador trabaja mediante la sustitución del microprocesador del sistema en desarrollo por uno conectado al sistema de emulación, esto permite la visualización y control de los parámetros de interés del sistema en desarrollo.

Los elementos más característicos que normalmente se podrán visualizar y controlar son los siguientes:

-El acumulador.
-El contador de programa(PC).
-Los registros básicos del sistema.
-El contenido de la memoria.

Dependiendo del sistema emulador, existen otra serie de facilidades que se pueden aportar, como pueden ser:

-El uso de un ensamblador incorporado, para la creación de los programas.
-El uso de un desensamblador, para examinar los programas en memoria, así como observar en que posiciones de memoria están situados.

-La realización de una traza de la ejecución del software del sistema.

- La posibilidad de programar PROMS y EPROMS.

La posibilidad de trabajar con todas estas facilidades, así como otras que pueda aportar un sitema emulador en particular resulta de una ayuda considerable, pudiendo reducir en varias veces el tiempo de desarrollo del sistema en cuestión.

2- El Emulador XDS Model 22 del TMS32010.-

Consiste en una sola unidad que provee las siguientes facilidades:

-Utiliza los recursos del ordenador del usuario.-

Posee una gran flexibilidad para adaptarse a varios tipos de ordenador (entre ellos el VAX o el IBM PC), con los que se puede crear, editar, linkar y ensamblar los programas antes de empezar la sesión de emulación para la depuración final del sistema.

-Se comunica mediante el interface EIA RS-232-C.-

Cada unidad XDS está equipada con cuatro puertos RS-232-C para la comunicación con otros sistemas.

El puerto A se conecta al terminal del usuario, es el que usaremos siempre en nuestro caso en particular.

El puerto B está reservado para futuras expansiones y no está disponible.

El puerto C puede conectarse con una impresora o con y sistema de programación de EPROMS.

El puerto D permite conectarse a una "host computer" (host significa huésped) para el desarrollo del software mediante recursos más sofisticados. En nuestro caso no usaremos este puerto puesto que este sistema requiere el uso de un terminal adicional para controlar el XDS.

-Posee un simple pero poderoso programa monitor.-

El monitor del emulador provee de un simple pero poderoso juego de comandos, que permiten un control completo de las funciones del emulador, así como del sistema en desarrollo.

El usuario especifica las condiciones de prueba para la depuración de los sofisticados programas necesarios para las aplicaciones de proceso digital de señal.

Los registros son fácilmente accesibles mediante el uso de los nombres asignados a dichos registros.

Por ejemplo, si se quiere que el contenido del acumulador sea 22, basta con asignar el valor mediante:

ACC=22

El conjunto de comandos está constituido por más de sesenta comandos de monitor que permiten una gran flexibilidad a la hora de definir las condiciones de test en las sesiones de emulación. Estos comandos se pueden combinar como breves procedimientos, lo que permite que varios comandos se ejecuten secuencialmente.

Existe también la posibilidad de repetir las funciones, lo que permite que comandos individuales o procedimientos se ejecuten indefinidamente hasta que sean detenidos por el usuario o por alguna condición de ruptura definida por el usuario.

-Permite definir puntos de ruptura (Breakpoints) tanto de tipo software como hardware.-

El usuario puede definir hasta diez puntos de ruptura software para detener la ejecución del programa cuando se ejecute alguna de las instrucciones situada en alguna de las posiciones de memoria especificadas.
El módulo Breakpoint/Trace expande la capacidad del sistema, por lo que será posible definir los puntos de ruptura en función de la actividad del sistema a nivel hardware.
Estos puntos de ruptura hardware detienen la ejecucuón del programa tras la satisfacción de un conjunto de condiciones especificadas (Por ejemplo: Lectura de memoria, escritura de memoria, adquisición de una instrucción, lectura E/S, escritura E/S, rangos de direcciones o rangos de valor de los datos).

-Permite efuctuar trazas lógicas.-
Una traza consiste en un registro de la actividad del microprocesador durante la ejecución del programa.
Las muestras de las trazas son cualificadas independientemente siguiendo los mismos criterios que en los puntos de ruptura hardware.

Las direcciones y los bits de datos están controlados mediante máscaras y comparadores. Las máscaras permiten al usuario seleccionar bits específicos de las direcciones o los datos bajo comparación para que sean chequeados o ignorados como cualificadores para los sucesos de breakpoint o de muestras de traza.

-Se puede conectar con otros emuladores XDS/22 para la emulación de sistemas multiproceso.-

El usuario puede conectar hasta nueve unidades XDS, una a continuación de otra.

Cuando se trabaja en modo multiproceso, se pueden seleccionar varias combinaciones de condiciones de start y stop sincronizadas para el control global del sistema multi-emulador.

El uso de comandos especiales de emulador permiten al usuario seguir complejas operaciones multiproceso con relativa facilidad.

3- Conexión del emulador XDS/22 a un PC.-

La conexión del emulador al PC se efectuará conectando el puerto serie del PC con el puerto A del emulador.

El emulador del TMS32010 emplea un sistema de detección automática de baudios, que se activa al encender el sistema y apretar por dos veces la tecla ENTER del teminal.

El número de baudios a que trabaje el sistema se fijará determinando la velocidad usada por el terminal conectado al puerto A del emulador.

Las velocidades de los puertos C y D se fijarán entonces a la misma velocidad empleada por el puerto A.

Las velocidades de dichos puertos podrán reprogramarse mediante el uso del comando IPORT.

Texas Instruments recomienda emplear un terminal que trabaje a 9600 bps para obtener un rendimiento óptimo.

En nuestro caso, el terminal en cuestión será nuestro PC, que se comunicará con el emulador mediante el uso de algún programa de comunicaciones de los muchos que hay en el mercado, como el xtalk o el impersonator.

Para ello habrá que configurar el programa para que trabaje con las siguientes características:

VELOCIDAD:	19.2K,9.6K,4.8K,2.4K,1.2K,600,300 ó 100
PARIDAD:	PAR (EVEN)
BITS DE STOP:	2
LONGITUD DEL CARACTER:	7 Bits
MODO DE TRANSMISION:	FULL DUPLEX

El cable de conexión deberá poseer la siguiente configuración:

PC		XSD	
PATILLA	NOMBRE	PATILLA	NOMBRE
2	RX	2	TX
3	TX	3	RX
1	PE	1	PE
7	SG	7	SG

4- INICIALIZACION DEL SISTEMA.-

Una vez conectados el PC y el emulador mediante el cable apropiado, configurado según las conexiones descritas en el apartado anterior, y después de cargar en el PC el programa de conexión (xtalk, impersonator, etc) configurado apropiadamente, se pasará a encender el emulador.

Tras encenderlo, se procederá a apretar el botón de reset del emulador durante unos 5 segundos para a continuación pulsar por dos veces la tecla ENTER del PC; este proceso permite al emulador detectar la velocidad de transmisión de nuestro ordenador, y una vez fijada automáticamente la velocidad empieza la comunicación.

Lo primero que hace el emulador es enviar un encabezado con la lista de comandos disponibles para el control del sistema, y una vez enviada esta lista aparece un interrogante en pantalla indicando que está listo para recibir los comandos del usuario.

Nota: Aunque el manual aconseja apretar el botón de reset después de encender el emulador, si se omite este paso el sistema funcionará normalmente sin problemas.

-LUCES INDICADORAS DE ESTADO:

La unidad XDS/22 tiene cuatro luces indicadoras de estado que están controladas por el emulador TMS32010. Las funciones de estas luces son las que siguen:

1. INDICADOR R1- BRANCH(Salto)-

Este indicador se enciende cuando el TMS32010 está ejecutando una

instrucción de salto, cuyo código de operación es el F900. Para que esta luz opere tendrá que estar instalada la placa de "brakpoint/trace". Este indicador de salto se apaga una vez que se ha acabado la ejecución de la instrucción de salto.

2. Indicador Ŗ2- TARGET BIO ON-
Este indicador se enciende cuando la patilla BIO del TMS32010 que está siendo emulado recibe un nivel bajo, y permanecerá encendido mientras que no vuelva a recibirse un nivel alto en dicha patilla.

3. Indicador Ŗ3 -INTERRUPTUS ENABLED(Interrupciones permitidas) -
Este indicador se enciende cuando las interrupciones están permitidas en el sistema emulador, y se apaga en el caso contrario.

4. Indicador Ŗ4 -INTERRUPT PENDING (Interrupción pendiente) -
Este indicador estará encendido siempre que la patilla de interrupción del TMS32010 esté activada, es decir, a nivel bajo. La luz permanecerá encendida hasta que el TMS32010 atienda la interrupción, momento en el que el PC pasa a apuntar a la localización 2 y se pasa a ejecutar la rutina de tratamiento de la interrupción.

-CONECTOR DEL SISTEMA (TARGET CONNECTOR):
El conector del sistema se instala en el sistema en desarrollo en sustitución del procesador que va a ser emulado.
Este conector debe manejarse con gran cuidado en todo momento, debido a la extrema fragilidad de sus patillas.

Texas Instruments recomienda que el sistema a depurar y el XDS esten apagados a la hora de conector o desconectar el conector del sitema.

El emulador puede no trabajar correctamente cuando el conector del sistema (target connector) está conectado a un sistema que no está encendido.

Cuando el target connector está desconectado, deberá estar protegido con algun tipo de cubierta aislante, con el fin de evitar cualquer posible daño físico. Hay que hacer notar que si la cubierta protectora fuera conductora el sistema XDS no funcionaría correctamente

Características del conector del sistema:

-La patilla uno del conector está indicada por un punto amarillo situado en la esquina a la que pertenece dicha patilla.

-Existen unos pines de test situados en la parte superior del conector que se corresponden uno a uno con los pines que encajan el zócalo del sistema en cuestión. Estos pines facilitan la comprobación del estado de cada una de las patillas del TMS32010.

-En cada lado del target connector existe un agujero de conexión a tiérra, en cada uno de los cuales enrosca el cable de conexión a tierra que viene con el emulador. Este cable deberá ser conectado a la tierra del sistema en desarrollo.

5 - JUEGO DE COMANDOS DEL PROGRAMA MONITOR.-

A continuación se decriben por orden alfabético los comandos que controlan el funcionamiento del emulador. Se ignoran aquellos cuya función es contolar el modo multiprocesador.

***BGND (Background mode)-**

Comando para uso en modo multiprocesador.

***BP (Hardware Breakpoint)-**

Objetivo: Configura las condiciones de los puntos de ruptura hardware.

PARAMETRO	VALORES DE LOS PARAMETROS	VALOR INICIAL
EVENT COUNT	0-7FFF	0
DELAY COUNT	0-7FF	0
ADDRESS MASK		
NUMBER OF EXTENDED ADDRESS BITS=0	0-FFF	FFF
NUMBER OF EXTENDED ADDRES BITS=8	0-FFFFF	FFFFF

DESCRIPCION DE LOS PARAMETROS:

EVENT COUNT (Cuenta de eventos):

Un evento se define como la satisfacción de todas las condiciones para la existencia de un hardware breakpoint.

La cuenta de eventos se decrementa cada vez que se encuentra un evento de ruptura.

El valor cero define una cuenta infinita.

DELAY COUNT (Cuenta de retraso):

Fija el número de pasos que se ejecutan tras haber llegado a cero la cuenta de eventos. El retraso consiste en realidad en obtener una traza y requiere que al menos uno de los trace qualifiers esté en ON; en caso contrario la cuenta de eventos queda inhabilitada.

ADDRESS MASK (ONES ENABLE):

Si se quiere fijar cualquier bit para que se seleccione la dirección independientemente de su estado, habrá que poner el bit correspondiente de la máscara a cero.

Un valor de 000 provoca que cualquier dirección sea considerada como dirección de breakpoint.

Un valor de FFF únicamente permite que las direcciones especificadas directamente actúen como cualificadoras de un breakpoint.

***BPIO (Breakpoint INPUT/OUTPUT) -**

Objetivo: Configurar los puntos de ruptura hardware en función de los aspectos de Entrada/Salida.

DESCRIPCION DE LOS PARAMETROS:

QUALIFIER[OFF,IOA,IOR,IOW]

Este parámetro define el criterio de cualificación primario par los hardware breakpoints.

El valor OFF inhibe cualquier acceso I/O como suceso de breakpoint.

El valor IOA permite que cualquier acceso I/O al puerto de Entrada/Salida especificado se cualifique como breakpoint.

El valor IOR selecciona únicamente las operaciones de lectura, mientras que el valor IOW selecciona únicamente las de escritura por el puerto especificado.

BREAKPOINT ADDRESS #1:

Selecciona un puerto específico de E/S como cualificador para un suceso de breakpoint. Si el RANGE INDICATOR está a OFF, tan sólo se cualificará al puerto especificado; si está a ON este valor actúa como un límite de rango.

BREAKPOINT ADDRESS #2:

Selecciona un segundo puerto de E/S como cualificador para un suceso de breakpoint. Si el RANGE INDICATOR está a OFF, tan sólo se cualificará al puerto especificado; si está a ON este valor actúa como un límite de rango.

RANGE INDICATOR[0=NO, 1=YES]

Cuando se selecciona YES, el RANGE INDICATOR está a ON y las direcciones seleccionadas arriba indican los límites entre los cuales los puertos E/S se cualifican como eventos de breakpoint.

Si ADDRESS #1 es mayor que ADDRS #2, el rango de valores comprendido entre las direcciones (ambas inclusive) queda excluído como cualificador de un evento de breakpoint.

En este caso los puertos no comprendido en dicho rango serán los que actúen como cualificadores.

EMU DATA COMPARE WORD:

El valor que aquí se especifique se compará con el valor existente en el bus de datos del emulador. En caso de coincidir se culaificará como un evento de breakpoint.

EMU DATA MASK (ONES ENABLE):

Este valor hexadecimal se usa como máscara para el valor definido en EMU DATA COMPARE WORD. Cuando un bit de la máscara tiene el valor cero, el bit correspondiente de la EMU DATA COMPARE WORD puede tomar indistintamente el valor cero o el uno. Si tiene el valor uno provoca que el bit correspondiente de la EMU DATA COMPARE WORD haya de coincidir con el bit del dato en cuestión para que éste pueda cualificarse como suceso de breakpoint (la comparación se efectuará bit a bit).

EXT DATA COMPARE BYTE:

El valor especificado aquí se compara con el dato obtenido mediante el CABLE EXTENDIDO DE TRAZA (EXTENDED TRACE CABLE) de la tarjeta de BREAKPOINT/TRACE.

Una coincidencia cualifica un suceso de breakpoint.

EXT DATA MASK (ONES ENABLE):

Este valor hexadecimal se usa como máscara con el EXT DATA COMPARE BYTE. Cuando aparece un bit cero en el DATA MASK

se considera como un bit "don't care".

Un valor FF causará que tan sólo funcione como cualificador el EXT DATA COMPARE BYTE especificado.

***BPM (Breakpoint Memory) -**

Objetivo: Configurar los puntos de ruptura hardware en función de los aspectos de direccionamiento de memoria.

DESCRIPCION DE LOS PARAMETROS:

QUILIFIER[OFF,MA,MW,MR,IAQ]

Este parámetro define el criterio de cualificación primario par los hardware breakpoints.

El valor OFF inhibe cualquier acceso a memoria como suceso de breakpoint.

El valor MA permite que cualquier acceso a alguna de las localizaciones de memoria especificadas se cualifique como un breakpoint.

El valor MR selecciona únicamente las operaciones de lectura en las localizaciones de memoria especificadas como breakpoints.

El valor MW hace lo mismo con las operaciones de escritura.

El valor IAQ cualifica la adquisición de instrucciones como breakpoints.

BREAKPOINT ADDRESS #1:

Selecciona una dirección específica de memoria como cualificador para un suceso de breakpoint. Si el RANGE INDICATOR está a OFF, tan sólo se cualificará la posición

de memoria especificada; si está a ON este valor actúa como un límite de rango.

El parámetro NUMBER OF EXTENDED ADDRES BITS cambia el valor máximo dependiendo del número de bits extendidos especificado.

BREAKPOINT ADDRESS #2:

Selecciona una segunda posición de memoria como cualificadora para un suceso de breakpoint. Si el RANGE INDICATOR está a OFF, tan sólo se cualificará la posición de memoria especificada; si está a ON este valor actúa como un límite de rango.

RANGE INDICATOR[0=NO, 1=YES]

Cuando se selecciona YES, el RANGE INDICATOR está a ON y las direcciones seleccionadas arriba indican los límites entre los cuales las posiciones de memoria se cualifican como eventos de breakpoint.

Si ADDRESS #1 es mayor que ADDRS #2, el rango de valores comprendido entre las direcciones (ambas inclusive) queda excluído como cualificador de un evento de breakpoint. En este caso las direcciones no comprendidas en dicho rango serán las que actúen como cualificadoras.

EMU DATA COMPARE WORD:

El valor que aquí se especifique se compará con el valor existente en el bus de datos del emulador. En caso de coincidir se cualificará como un evento de breakpoint.

EMU DATA MASK (ONES ENABLE):

Este valor hexadecimal se usa como máscara para el valor definido en EMU DATA COMPARE WORD. Cuando un bit de la máscara tiene el valor cero, el bit correspondiente de la EMU DATA COMPARE WORD puede tomar indistintamente el valor cero o el uno. Si tiene el valor uno provoca que el bit correspondiente de la EMU DATA COMPARE WORD haya de coincidir con el bit del dato en cuestión para que éste pueda cualificarse como suceso de breakpoint (la comparación se efectuará bit a bit).

EXT DATA COMPARE BYTE:

El valor especificado aquí se compara con el dato obtenido mediante el CABLE EXTENDIDO DE TRAZA (EXTENDED TRACE CABLE) de la tarjeta de BREAKPOINT/TRACE.

Una coincidencia cualifica un suceso de breakpoint.

EXT DATA MASK (ONES ENABLE):

Este valor hexadecimal se usa como máscara con el EXT DATA COMPARE BYTE. Cuando aparece un bit cero en el DATA MASK se considera como un bit "don't care".

Un valor FF causará que tan sólo funcione como cualificador el EXT DATA COMPARE BYTE especificado.

***CASB (Clear All Software Breakoints) -**

Objetivo: Borrar todos los puntos de ruptura software.

Sin parámetros.

***COPY (Copy Memory) -**

Objetivo: Realiza la copia de los 4K de memoria del sistema en depuracón, en la memoria de programa del emulador. Como consecuencia se borrará cualquier dato de la memoria del emulador.

***CRUN (Continue Run)-**

Objetivo: Continúa la ejecución de un programa después de haber sido detenido, con las mismas condiciones que tenía previamente a dicha detención.

Si la memoria de traza está llena, o si son cero el "breakpoint event" y el "delay count", esta instrucción iniciará un estado indefinido de RUN que podrá detenerse mediante la presión de cualquier tecla.

***CSB (Clear Software Breakpoint)-**

Borra el punto de ruptura software indicado en la instrución.

DESCRIPCION DE LOS PARAMETROS:

BREAKPOINT ADDRES:

Este parámetro se usa para declarar la dirección del breakpoint a borrar. Este parámetro lo comparte la instrucción SSB (Set Software Breakpoint) y puede modificarse con dicho comando.

***DDM (Display Data Memory)-**

Objetivo: Mostrar por pantalla el contenido de las posiciones de memoria indicadas.

DESCRIPCION DE LOS PARAMETROS:

START ADDRESS:

Indica la primera posición de memoria a mostrar.

END ADDRESS:

Indica la última posición de memoria a mostrart. Si START ADDRESS es mayor que END ADDRESS, sólo se mostrará la posición de memoria correspondiente a START ADDRESS.

***DHS (Display Halt Status)-**

Objetivo: Saca por pantalla un código que indica el motivo de la última detención (halt). Puede salir más de un código por pantalla.

***DI (Disable Interrupts)-**

Objetivo: Inhabilita las interrupciones del emulador del TMS32010 durante la ejecución del programa. Este comando pone la "interrupt mode flag" a uno, con lo que se impede la ejecución de interrupciones posteriores.

***DIAG (Diagnostic Mode)-**

Este comando se emplea para efectuar un dianóstico a distancia del emulador. No tiene interés fuera de EEUU.

*DIO (Display I/O Port)-

Objetivo: Permite al usuario conocer el valor de un puerto a especificar.

DESCRIPCION DE LOS PARAMETROS:

PORT ADDRESS:

Especifica la dirección del puerto de E/S del que se desea leer el valor.

*DL (Down Load)-

Objetivo: Configurar al emulador para recibir información de una fuente externa.

DESCRIPCION DE LOS PARAMETROS:

LOAD OFFSET:

En este parámetro se indica la dirección inicial a partir de la cual se cargará la información a recibir.

Sólo valdrán valores distintos de cero para el formato Texas Instruments (TI) con ficheros de código objeto relocalizables.

FORMAT (0=TI,1=TEK):

Este parámetro especifica el formato para los ficheros de código objeto.

VALORES:

0=Código ASCII o Formato Comprimido TI.

1=Formato Tektronix Hexadecimal.

DESTINATION (0=PROGRAM, 1=PROM, 2=DATA, 3=ASM)

Especifica el destino de la información que se recibe. Si la cantidad de información recibida supera a la cantidad de memoria disponible, se ignorará la información que no quepa.

Los posibles destinos son los siguientes:

0=La memoria normal de programa.

1=El programador de EPROMS o el aparato conectado al puerto C.

2=Memoria de datos.

3=Ensamblador Línea a Línea.

PROTOCOL (0=NONE,1=TEK,2=ASR,3=VAX)

Especifica el protocolo usado durante la transferencia de datos. Permite la selección de los caracteres de control usados para indicar el comienzo y final de la transmisión de carga o descarga y el carácter de pass-throug.

0=No se emplea protocolo pre-definido.

Los carateres de control se definen mediante el comando IHC. El comando IHC debe ejecutarse antes de usar el comando DL si se elige esta opción.

1=Se emplea el protocolo Tektronix.

ASCII "0"-Registro transferido sin error.

ASCII "7"-Se ha detectado error. Repetir transmisión.

Los caracteres de control específico se definen mediante el uso del comando IHC, por ello debe usarse antes de la instrucción de DL.

2=Se emplea el protocolo de terminal ASR.

CTRL-R- Comenzar Download.

CTRL-S-Finalizar Download.

CTRL-Q-Comenzar Upload.

CTRL-@-Finalizar Upload.

CTRL-P-Ignorar siguiente carácter de control.

3=Se emplea protocolo de Vax o PDP-11.

CTRL-A-Comenzar Upload.

CTRL-Z-Finalizar Upload.

CTRL-V-Comenzar Download.

CTRL-W-Finalizar Download.

CTRL-P-Ignorar siguiente carácter de control.

SOURCE (0=HOST,1=USER):

Permite al usuario definir la fuente de la información que se va a recibir.

0=La "Host computer" será la fuente de los datos que se carguen en el emulador. Esto significa que se leerá por el puerto D.

1=Se declarará al terminal de usuario como la fuente de los datos a cargar en el emulador. Esta opción está prevista para el caso en que se trabaje con ordenadores personales usados como terminales. La transmisión se hará por el puerto A, que es por donde se conectan este tipo de terminales.

***DPM (Display Program Memory)-**

Objetivo: Mostrar por pantalla el contenido de una zona de memoria de programa.

DESCRIPCION DE LOS PARAMETROS:

START ADDRESS:

Este parámetro especifica la dirección de la primera posición de memoria a mostrar.

END ADDRESS:

Este parámetro especifica la dirección de la última posicón de memoria a mostrar. En el caso de que la dirección de la primera posición sea superior a la de la última, sólo aparecerá en pantalla el contenido de la posición de memoria indicada por START ADDRESS.

***DPS (Display Processor Status)-**

Objetivo: Sacar por pantalla el contenido actual de todos los registros del emulador.

SIN PARAMETROS.

Los registros que muestra son los siguientes:

PC,ACC,ST,AR0,AR1,TPS,MOS,BOS,P,T.

Así como el contenido de los bits del registro de estado.

***DR (Display Registers)-**

Objetivos: Sacar por pantalla el contenido de los registros del emulador. La única diferencia con respecto a la instrucción DPR consiste en que DR no muestra el contenido de los bits del registro de estado.

***DSB (Display All Software Breakpoints) -**

Objetivos: Mostrar por pantalla el conjunto actual de puntos de ruptura software.

SIN PARAMETROS.

***DT (Display Trace)-**

Objetivo: Mostrar el contenido de las muestras de la traza especificadas por el usuario.

DESCRIPCION DE LOS PARAMETROS:

FIRST SAMPLE(0-7FE,0=OLDEST SAMPLE):

Aquí se especifica la primera muestra a representar. Si el valor introducido es 0, se representarán las muestras más antiguas.

Si el valor introducido es el 7FF, entonces siempre se representarán las muestras más recientes.

THE NUMBER OF SAMPLES:

Especifica el número de muestras a sacar por pantalla. Si el número de muestras es superior a la cantidad que se puede representar en una pantalla, se efectuará un deslizamiento (scroll) hacia arriba de la información de la pantalla. Si el número de muestras disponibles es inferior al especificado, únicamente se representarán las muestras disponibles.

*DTS (Display Trace Status)-

Objetivo: Mostrar el estado actual de la traza, así como el "trace count",el "breakpoint event" y el "delay counts".

SIN PARAMETROS.

*DV (Display Value) -

Objetivo: Efectúa la adición y substracción de dos númereos hexadecimales.

DESCRIPCION DE LOS PARAMETROS:

VALUE: Se introduce un número hexadecimal sin signo de hasta 24 bits de longitud.

VALUE: Se introduce un número hexadecimal sin signo de hasta 24 bits de longitud.

SALIDA:

SUM>000000 DIFFERENCE>000000

En SUM se representa un número de 24 bits sin signo en hexadecimal con el resultado de la suma de los dos valores introducidos.

En DIFFERENCE se representa un número con las mismas características que el anterior y es el resultado de restarle al valor mayor de los introducidos el valor menor, independientemente del orden en que se introdujeran.

***EI (Enable Interrupts)-**

Objetivo: Permite al emulador del TMS32010 aceptar interrupciones desde el código objeto almacenado en la memoria de programa.

SIN PARAMETROS.

***FILL (Fill Memory)-**

Objetivo: Permite al usuario rellenar una zona de memoria con un valor específico.

DESCRIPCION DE LOS PARAMETROS:

START ADDRESS:

Especifica la primera dirección que se rellenará con el dato.

END ADDRESS:

Especifica la última dirección a rellenar.

DATA:

Es el valor con el que se rellenarán las posiciones de memoria comprendidas dentro del rango indicado.

DESTINATION[0=PROGRAM,1=DATA]

Especifica el segmento de memoria a rellenar:

0=El destino será la memoria de programa.

1=El destino será la memoria de datos. Si la dirección superior excede las 8F posiciones de la memoria de datos, se rellenará toda la memoria de datos.

***FIND (Find Data)-**

Objetivo: Buscar en la memoria el valor especificado y sacar por pantalla todas las direcciones que lo contengan.

DESCRIPCION DE LOS PARAMETROS:

START ADDRESS: Especifica la dirección a partir de la cual empezará a buscar el valor indicado.

END ADDRESS: Especifica la dirección hasta la cual se realizará la búsqueda del valor indicado.

DATA: Especifica el valor a buscar.

DATA MASK (ONES EMABLE)

Establece una máscara bit a bit respecto al valor especificado.

Un 1 en un bit significa que el bit correspondinte en el dato a buscar debe coincidir con el del valor a comparar.

Un 0 en un bit significa que el bit correspondinte en el dato a buscar no cuenta a la hora de efectuar la comparación.

SOURCE[0=PROGRAM, 1=DATA]

Especifica el segmento de memoria a buscar:

0=Memoria de programa.

1=memoria de datos. Si el valor de END ADDRESS excede el valor 8F, la búsqueda se efectuará en toda la memoria de datos.

*GHALT (GROUP HALT)-

Comando usado en operaciones multi-procesador.

*GRUN (GROUP RUN)-

Comando usado en operaciones multi-procesador.

*HELP-

Objetivo: Mostrar una lista de todos los comandos.

SIN PARAMETROS.

*HOST (Terminal Mode)-

Objetivo: Colocar al emulador en el modo de terminal para comunicar con un sistema "host computer".

SIN PARAMETROS.

*ICC (Initialize Cursor Control)-

Objetivo: Inicializar las teclas para el control del cursor. Este comando es necesario debido a que no todos los terminales generan los mismos códigos ASCII al pulsar los cursores.

Nota: Los caracteres ASCII asociados a cada cursor han de ser de tipo simple, por lo que al usar de terminal un PC no será de utilidad este comando.

*ID (Identification)-

Identifica al emulador mediante la representación en pantalla del logotipo. En modo multiprocesador, tampién se muestra el número de identificación.

*IDM (Inspect Data Memory)-

Objetivo: Sacar por pantalla y modificar el contenido de una zona de la de memoria de datos, mostrando las localizaciones de una en una.

DESCRIPCION DE LOS PARAMETROS:

ADDRESS: Indica la dirección de memoria cuyo contenido se va a mostrar y, opcionalmente a modificar.

Controles:

<+>,<SPACE>, o <CR>: Permiten avanzar a la siguiente dirección.

<->: Permite retroceder a la dirección previa.
<Q><CR> o <ESC>: Para finalizar la ejecución de la instrucción IDM.

*IHC (Iniatialize Host Control)-

Objetivo: Permite definir las secuencias de caracteres de control, que informarán al monitor del emulador que un proceso de carga o descarga, ha comenzado o finalizado, o que un carácter de control ha de dejarse pasar sin realizar ninguna operación. Esta última característica permite transmitir caracteres de control sin que el XDS actúe sobre ellos.

PARAMETROS:

Al ejecutar esta instrucción se requerirán los siguientes parámetros:

DEPRESS KEY FOR DOWNLOAD START:

Introducir la tecla de control.

El valor inicial es CTRL-V.

DEPRESS KEY FOR DOWNLOAD END:

Introducir la tecla de control.

El valor inicial es CTRL-W.

DEPRESS KEY FOR UPLOAD START:

Introducir la tecla de control.

El valor inicial es CTRL-A.

DEPRESS KEY FOR UPLOAD END:

Introducir la tecla de control.

El valor inicial es CTRL-Z.

DEPRESS KEY FOR PASS THROUGH CHARACTER:

Introducir la tecla de control.

El valor inicial es CTRL-P.

***IMD (Initialize mode)-**

Objetivo: Inicializa el modo de operación de los emuladores trabajando en multi-proceso.

***IMP (Initialize Multi-Processor Mode)-**

Objetivo: Inicializa el modo de operación multi-procesador.

*INIT (Initialize Emulador)-

Objetivo: Inicializa el emulador TMS32010.

DESCRIPCION DE LOS PARAMETROS:

EMU:CLOCK SOURCE[0=INTERNAL, 1=SOURCE]

Este parámetro indica al emulador con qué fuente de reloj trabajar. Por defecto se asume que la fuente de reloj sea la propia del emulador.

El sistema a emular puede proveer el generador de frecuencia. La fuente también podrá obtenerse mediante un cable conectado del generador que se desee usar, al conector BNC1 situado en el chasis del XDS.

0=Fuente interna del emulador.

1=Cristal externo perteneciente al sistema en emulación.

MEMORY[0=EMULATOR, 1=TARGET, 2=MICROCOM]

0=EMULATOR. Los 4K pertenecerán al emulador.

1=TARGET. Los 4K pertenecerán a la memoria del sistema en emulación.

2=MICROCOM. Los primeros 1.5 K bytes pertenecerán al TMS32010, y los restantes 2.5 K bytes estarán situados en el sistema en emulación

BP:NUMBER OF EXTENDED ADDRESS BITS (0-8)

Este parámetro configura las señales en el EXTENDED TRACE CABLE perteneciente al "breakpoint/trace board". El valor inicial de cero indica que las 8 líneas del cable se configuran únicamente para datos.

El valor introducido en este parámetro reducirá el número de líneas de datos por el número seleccionado, es decir, el número de EXTENDED ADDRESS BITS, y el de DATA BITS sumarán siempre 8.

***IPM (Inspect Program Memory)-**

Objetivo: Mostrar y modificar el contenido de una zona de memoria a modificar, trabajando con las posiciones de memoria una a una.

DESCRIPCION DE LOS PARAMETROS:

ADDRESS: Especifica la posición de memoria a inspeccionar y opcionalmente a modificar.

Controles:

<+>,<SPACE>, o <CR>: Permiten avanzar a la siguiente dirección.

<->: Permite retroceder a la dirección previa.

<Q><CR> o <ESC>: Para finalizar la ejecución de la instrucción IPM.

***IPORT (Initialize EIA Port)-**

Objetivo: Inicializa el puerto serie EIA RS-232-C y configura la velocidad de transmisión, la paridad, el número de bits de stop, y el número de bits por carácter, con el fin de permitir la comunicación con el sistema conectado al puerto en cuestión.

DESCRIPCION DE LOS PARAMETROS:

PORT[0=D(HOST),1=C(LOG/PROM)]:

Indica qué puerto se va a configurar, el C o el D.

BAUD(19.2K,9.6K,4.8K,2.4K,1.2K,600,300,110):

Este parámetro permite fijar la velocidad de trasmisión. Cuando el XDS se enciende, el puerto A determina automáticamente la velocidad del terminal que tiene conectado (el terminal de usuario) y se adapta a dicha velocidad. El resto de los puertos igualmente adquieren dicho valor inicial.

Valores: (0=19.200bps,1=9600bps,....,6=300bps,7=110bps)

PARITY[0=OFF,1=ODD,2=EVEN]: Indica el tipo de paridad de la transmisión.

STOP BITS[0=2,1=1]: Configura el número de bits de stop en la señal.

BITS PER CHARACTER[0=7,1=8]: Configura el número de bits por carácter. El campo de datos puede consistir en siete u ocho bits.

*IPRM (Initialize Parameters)-

Objetivo: Inicializa todos los parámetros a los valores asumidos por defecto al encenderse el sistema.

registros del emulador, y permitir al usuario cambiar dichos contenidos directamente.

Al ejecutar este comando, irá mostrandose secuencialmente el contenido de los registros: PC,ACC,AR0,AR1,TOS,MOS,BOS. con la opción en cada caso de modificar su contenido.

La ejecución del código objeto podrá cambiar el contenido de estos registros, con lo cual la ejecución posterior de esta instrucción podrá mostrar nuevos valores en dichos registros.

*IT (Inspect Trace)-

Objetivo: Sacar por pantalla las muestras de la traza.

DESCRIPCION DE LOS PARAMETROS:

FIRST SAMPLE(0-7FE,0=OLDEST SAMPLE):

Con este parámetro se especifica cuál es la primera muestra a sacar. Se sacarán tantas muestras como quepan en una pantalla.

Controles:

<ESC>,<Q>: Aborta la ejecución de este comando.

+nnn: Indica que cada retorno de carro salta a las próximas nnn muestras y saca por pantalla una página de traza.

-nnn: Indica que cada retorno de carro salta a las previas nnn muestras y saca por pantalla una página de traza.

<CR>: Actúa como se indica anteriormente según el valor +nnn ó -nnn introducido.

Nota: nnn será un valor hexadecimal de rango 0-FFF. Para valores superiores a 7FF, se tomará como valor 7FF cuando se proceda a trabajar con él.

***LOG (Logging Device)-**

Objetivo: Activa o desactiva la función de logging (conexión). El aparato a conectar normalmente es una impresora.

DESCRIPCION DE LOS PARAMETROS:

LOG DEVICE[0=OFF,1=ON]:

Este parámetro activa o desactiva el aparato conectado al puerto C.

LINE FEED WITH BACKSPACE[0=OFF,1=ON]:

Cuando esta característica está a ON, el emulador enviará un linefeed con cada bakspace.

INTERACCION CON OTROS COMANDOS:

COMANDO: DL.

PARAMETRO CON EL QUE INTERACTUA: DESTINATION(PROM).

PARAMETRO AFECTADO: LOG DEVICE (PUESTO A CERO).

***MDM (MODIFY DARA MEMORY)-**

Objetivo: Cambiar el contenido de una posición específica de la memoria de datos.

DESCRIPCION DE LOS PARAMETROS:

ADDRESS: Especifica una dirección individual de la memoria de datos, que será modificada con la entrada de un nuevo valor.

DATA: Aquí se indica el nuevo valor.

***MESG (Exchange Diagnostic Message)-**

Objetivo: Intercambiar mensajes con el servicio técnico remoto cuando se está en el modo de diagnóstico. Sin utilidad fuera del radio de acción del servicio técnico de texas instruments.

***MIO (Modify I/O Port)-**

Objetivo: Permite al usuario sacar un determinado valor por el puerto que se indique.

DESCRIPCION DE LOS PARAMETROS:

PORT ADDRESS:

Especifica el puerto E/S por el que se va a sacar el dato.

DATA: Aqí se introduce el valor que se desea sacar por el puerto especificado.

***MPM (Modify Program Memory)-**

Objetivo: Cambiar el contenido de una posición específica de la memoria de programa.

DESCRIPCION DE LOS PARAMETROS:

ADDRESS: Especifica una dirección individual de la memoria de programa, que será modificada con la entrada de un nuevo valor.

DATA: Aquí se indica el nuevo valor.

***MR (Modify Registers)-**

Objetivo: Sacar por pantalla los valores de cada uno de los registros del emulador, y permitir al usuario cambiar dichos valores.

PARAMETROS:

PC,ACC,AR0,AR1,TOS,MOS,BOS.

***QDIAG (QUIT DIAGNOSTIC MODE)-**

Objetivo: Finalizar el modo de diagnóstico.

***RCC (Reset Cursor Control)-**

Objetivo: Resetear las funciones de control del cursor a los valores de encendido. Este comando se usa para restituir los valores de control del cursor modificados por el comando ICC.

***RES (Reset)-**

Objetivo: Resetea el procesador TMS32010 para resincronizar su operación con el procesador de control 9996. RES no cambia los valores de los parámetros.

*RESTART (Restart Emulator)-

Objetivo: Resetea el emulador a los valores que tiene tras el encendido. Esto elimina la necesidad de apagar y encender el emulador, con lo que se prolonga la vida de los componentes electrónicos.

PARAMETRO:

ARE YOU SURE[0=NO, 1=YES]

Su objetivo es el de confirmar las intenciones del usuario de reinicializar el sistema, no sea que lo fuera a reinicializar por error.

*RTR (Run on Target Reset)-

Objetivo: Inicia un comando RUN en el emulador cuando el sistema en emulación se recibe un reset. Cuando se ejecuta este comando, el TMS32010 del emulador entra en un bucle infinito hasta que se activa la patilla RESET del emulador.

*RUN (Execute Program)-

Objetivo: Ejecutar el programa que resida en ese momento en la memoria de datos. Antes de ejecutar este comando habrá que inicializar los registros a los valores adecuados, especialmente el registro PC, que indicará la dirección de la primera dirección a ejecutar.

SIN PARAMETROS.

*SNAP (Fix Display)-

Objetivos: Definir un formato de pantalla fija en lugar del formato en pantalla deslizante (scroll) asumido por defecto. Este comando se usa sólo como el primero de un procedimiento (sucesión de comandos separados por comas, o por puntos y comas).

*SOR (Set Opcode Range)-

Objetivo: Define el rango de los valores de código usados en TRIX (Trace On Instruccion Acquisition Extended). Este comando deberá ejecutarse previamente al comando TRIX si el parámetro OPCODE QUALIFIERS está a uno.

DESCRIPCION DE LOS PARAMETROS:

LOWER OPCODE BOUND(LS NIBBLE IGNORED):

La representación en hexadecimal de los códigos de las instrucciones en ensamblador pueden incluirse como muestras válidas de la traza. El usuario puede definir un rango de estos valores a incluir. En este parámetro se define el límite inferior de dicho rango. Los últimos cuatro bits del código hexadecimal son ignorados.

UPPER OPCODE BOUND(LS NIBBLE IGNORED):

Este valor define el límite superior del rango de códigos. los últimos cuatro bits del código hexadecimal son ignorados.

RANGO[0=DELETE,1=ADD]:

Este parámetro permite definir como muestras válidas uno o más rangos de códigos de instrucción. Si se selecciona la opción 1, entonces el rango de valores delimitado por los parámetros LOWER y UPPER OPCODE BOUND se añaden a los rangos previamente definidos. Si selecciona la opción 0, entonces el rango especificado se borra del conjunto de muestras de traza válidas.

***SS (Single Step)-**

Objetivo: Permite ejecutar un programa almacenado en la memoria de programa paso a paso. Con cada paso ejecutado, aparece en pantalla los registros PC, ACC, y el registro de estado, así como el código en ensamblador de la instrucción ejecutada.

***SSB (Set Software Breakpoint)-**

Objetivo: Declarar los puntos de ruptura software (breakpoints). Se pueden declarar un máximo de 10 breakpoints.

DESCRIPCION DE LOS PARAMETROS:

BREAKPOINT ADDRESS: Este parámetro, especifica una dirección de la memoria de programa que será definida como un software breakpoint. Cuando un programa en ejecución alcanza esta dirección, se detiene la ejecución. El usuario puede especificar hasta 10 breakpoints de este tipo.

*THALT (Total Halt)-

Objetivo: Detiene la operación de todos los emuladores en modo multi-proceso.

SIN PARAMETROS.

*TR (Trace)-

Objetivo: Configura las condiciones de traza. Para el uso de esta instrucción se requiere que esté instalada la placa "breakpoint-trace board".

DESCRIPCION DE LOS PARAMETROS:

TRACE COUNT(1-7FF,0=INFINITE):

Este parámetro especifica el número de muestras a contar en una traza. Por cada muestra válida se decrementa en una unidad la cuenta, y cuando la cuenta alcanza el valor cero, se detiene la ejecución del programa y aparece en pantalla el código TMF, que indica la causa de dicha detención (Trace Memory is Full).

ADDRES MASK(ONES ENABLE)-

El valor hexadecimal introducido en este parámetro determina un rango (o rangos) de direcciones que se cualificarán como muestras de traza. Cado bit cero en la palabra hexadecimal indicará que dicho bit será ignorado cuando se comparen las direcciones válidas.

TRACE MODE[0=TRM&TRIO,1=TRIX]

Este parámetro especifica el modo de traza a usar.

VALORES:

0=Traza de memoria o ciclo de E/S.

1=Realiza la traza al efectuar la adquisición de la instrucción (IAQ), para extender al exterior (memoria externa) el rango de direcciones especificado.

*TRIO (TRACE INPUT/OTPUT)-

Objetivo: Configurar las condiciones de traza basadas en los aspectos de Entrada/Salida.

DESCRIPCION DE LOS PARAMETROS:

QUALIFIER[OFF,IOA,IOR,IOW]:

Este parámetro define el criterio de cualificación primario de las trazas. El valor OFF descalifica cualquier acceso de E/S como muestra de traza. El valor IOA permite que cualquier acceso a los puertos especificados de E/S se clasifiquen como muestras de traza. La opción IOR selecciona únicamente las operaciones de lectura, mientras que la opción IOW hace lo mismo con las de escritura.

TRACE ADDRESS #1: Selecciona un puerto específico de E/S como una muestra de traza válida.

TRACE ADDRESS #2: Selecciona un segundo puerto de E/S como muestra de traza válida.

RANGE INDICATOR[0=NO,1=YES]:

Cuando el RANGE INDICATOR está a ON, las direcciones

declaradas anteriormente actuarán como límites de un rango de puertos E/S que se clasifican como muestras de traza. Si ADDRES#2 es menor que ADDRESS#1, el rango de valores comprendidos entre ambos extremos se excluyen como muestras de traza válidas. Los puertos fuera de dicho rango, serán en dicho caso los que actúen como validadores.

EMU DATA COMPARE WORD:

El valor que aquí se especifique se compara con el valor actual en el bus de datos del emulador. En caso de coincidir se cualificará como una muestra de traza.

EMU DATA MASK(ONES ENABLE):

Este valor hexadecimal se usa como máscara de la EMU DATA COMPARE WORD. Los bits 0 de la DATA MASK indican que su correspondiente de la EMU DATA COMPARE WORD serán ignorados. Un valor de FFFF implica que únicamente la propia DATA COMPARE WORD se cualifique como muestra de traza.

EXT DATA COMPARE BYTE:

El valor que aquí se especifique se comparará al valor leído por el EXTENDED TRACE CABLE. En caso de coincidir se clasifica como muestra de traza.

EXT DATA MASK(ONES ENABLE):

Este valor hexadecimal se usa como máscara de la EXT DATA COMPARE BYTE. Los bits 0 de la DATA MASK indican que su

correspondiente de la EXT DATA COMPARE BYTE serán ignorados. Un valor de FF implica que únicamente la propia DATA COMPARE BYTE se cualifique como muestra de traza.

***TRIX (Trace on Instruction Acquisition Extended)-**

Objetivo: Efectuar la traza durante la fase de adquisición de instrucción del ciclo de instrucción que se extiende por encima del rango de direcciones especificado.

DESCRIPCION DE LOS PARAMETROS:

OPCODE QUALIFIERS[0=NO,1=YES]:

Este comando permite usar los códigos de operaciones definidos mediante el comando SOR como cualificadores de muestras de traza cuando se ejecuta el comando TRIX.

QUALIFIER[OFF,MA,MW,MR]:

Este parámetro define el criterio de cualificación primario a la hora de efectuar una traza de las operaciones de memoria. El valor OFF descalifica cualquier acceso a memoria como una muestra de traza. El valor MA califica cualquier acceso a memoria como una muestra de traza. El valor MR califica únicamente a las operaciones de lectura, mientras que MW hace lo propio con las de escritura.

QUALIFIER[OFF,IOA,IOR,IOW]:

Este parámetro define el criterio de cualificación primario a la hora de efectuar la traza. El valor OFF descalifica cualquier operación de E/S como una muestra de traza. El valor IOA califica cualquier acceso a puerto como una muestra de traza. El valor IOR califica únicamente a las operaciones de lectura, mientras que IOW hace lo propio con las de escritura del puerto especificado.

TRACE ADDRESS #1: Selecciona una dirección de memoria específica como cualificadora de muestra de traza.

TRACE ADDRESS #2: Selecciona una segunda dirección de memoria como cualificadora de muestra de traza.

RANGE INDICATOR[0=NO,1=YES]:

Cuando el RANGE INDICATOR está a ON, las direcciones declaradas anteriormente actuarán como límites de un rango de direcciones de memoria que se clasifican como muestras de traza. Si ADDRES#2 es menor que ADDRESS#1, el rango de valores comprendidos entre ambos extremos se excluyen como muestras de traza válidas. Las posiciones de memoria fuera de dicho rango, serán en dicho caso los que actúen como validadores.

EXT DATA COMPARE BYTE:

El valor que aquí se especifique se comparará al valor leído por el EXTENDED TRACE CABLE. En caso de coincidir se clasifica como muestra de traza.

EXT DATA MASK(ONES ENABLE):

Este valor hexadecimal se usa como máscara de la EXT DATA COMPARE BYTE. Los bits 0 de la DATA MASK indican que su correspondiente de la EXT DATA COMPARE BYTE serán ignorados. Un valor de FF implica que únicamente la propia DATA COMPARE BYTE se cualifique como muestra de traza.

*TRM (Trace Memory)-

Objetivo: Configurar las condiciones de traza basadas en los aspectos de direccionamiento.

DESCRIPCION DE LOS PARAMETROS:

QUALIFIER[OFF,MA,MW,MR,IAQ]:

Este parámetro define el criterio de cualificación primario a la hora de efectuar una traza de las operaciones de memoria. El valor OFF descalifica cualquier acceso a memoria como una muestra de traza. El valor MA califica cualquier acceso a memoria como una muestra de traza. El valor MR califica únicamente a las operaciones de lectura, mientras que MW hace lo propio con las de escritura. El valor IAQ selecciona únicamente la adquisición de instrucciones como cualificador para el muestreo de trazas.

TRACE ADDRESS #1: Selecciona una dirección de memoria específica como cualificadora de muestra de traza.

TRACE ADDRESS #2: Selecciona una segunda dirección de memoria como cualificadora de muestra de traza.

RANGE INDICATOR[0=NO, 1=YES]:

Cuando se selecciona YES, el RANGE INDICATOR está a ON y las direcciones seleccionadas arriba indican los límites entre los cuales las posiciones de memoria se cualifican como eventos de breakpoint.

Si ADDRESS #1 es mayor que ADDRS #2, el rango de valores comprendido entre las direcciones (ambas inclusive) queda excluído como cualificador de un evento de breakpoint. En este caso las direcciones no comprendidas en dicho rango serán las que actúen como cualificadoras.

EMU DATA COMPARE WORD:

El valor que aquí se especifique se compará con el valor existente en el bus de datos del emulador. En caso de coincidir se culaificará como un evento de breakpoint.

EMU DATA MASK (ONES ENABLE):

Este valor hexadecimal se usa como máscara para el valor definido en EMU DATA COMPARE WORD. Cuando un bit de la máscara tiene el valor cero, el bit correspondiente de la EMU DATA COMPARE WORD puede tomar indistintamente el valor cero o el uno. Si tiene el valor uno provoca que el bit

correspondiente de la EMU DATA COMPARE WORD haya de coincidir con el bit del dato en cuestión para que éste pueda cualificarse como suceso de breakpoint (la comparación se efectuará bit a bit).

EXT DATA COMPARE BYTE:

El valor especificado aquí se compara con el dato obtenido mediante el CABLE EXTENDIDO DE TRAZA (EXTENDED TRACE CABLE) de la tarjeta de BREAKPOINT/TRACE.

Una coincidencia cualifica un suceso de breakpoint.

EXT DATA MASK (ONES ENABLE):

Este valor hexadecimal se usa como máscara con el EXT DATA COMPARE BYTE. Cuando aparece un bit cero en el DATA MASK se considera como un bit "don't care".

Un valor FF causará que tan sólo funcione como cualificador el DATA COMPARE BYTE especificado.

***TRUN (Total Run)-**

Objetivo: Indicar a todos los emuladores interconectados que empiecen a correr.

***UL (Upload)-**

Objetivo: Selecciona los parámetros para cargar el código objeto desde el emulador al sistema externo. Se emplea básicamente para almacenar en algún sistema de disco externo la información del emulador.

DESCRIPCCION DE LOS PARAMETROS:

START ADDRESS: Este parámetro especifica la dirección a partir de la cual se comenzará a almacenar la información en el sistema externo.

END ADDRESS: Este parámetro especifica la dirección hasta la cual se almacenará la información en el sistema externo. Si el rango de direcciones comprendido entre START ADDRESS y END ADDRES es superior a la longitud real del fichero, las posiciones de más se rellenarán con ceros.

OBJ FORM[0=TI,1=TICOM,2=TEK,3=INTEL,4=MR]:

Mediante este parámetro se especifica el formato de los ficheros de código objeto.

0=TI NORMAL ASCII.

1=TI FORMATO COMPRIMIDO.

2=FORMATO TEKTRONIX HEXADECIMAL.

3=FORMATO MOTOROLA EXORCISOR.

DATA FORMAT[0=WORD,1=HI BYTE,2=LOW BYTE]:

Este parámetro se usa en conjunción los ficheros cuyo formato sea distinto del TI-normal o el TI-comprimido. Esta opción permite al usuario cargar un byte de la palabra de datos por vez, de tal forma que si el sistema usa distintas direcciones de memoria para los bytes superior e inferior de cada palabra, éstos puedan ser transmitidos individualmente.

SOURCE[0=PROGRAM,1=DATA]:

Especifica el origen de los datos a trasmitir, indicando si provienen de la memoria de programas o la de datos:

0=MEMORIA DE PROGRAMA.

1=MEMORIA DE DATOS.

DEST[0=HOST,1=PROM,2=USER]:

Permite definir al usuario el destino de la información transmitida.

0=La información se trasmite a través del puerto D, al que se conecta el ordenador "HOST" (huésped).

1=La información se transmite a través del puerto C a el programador de EPROMS, o al dispositivo que se haya conectado a dicho puerto.

2=La información se transmite por el puerto A al terminal de usuario. Esta opción se ofrece considerando la posibilidad de que se esté trabajando con un terminal inteligente o con un ordenador personal como terminal de usuario.

PROTOCOL (0=NONE,1=TEK,2=ASR,3=VAX)

Especifica el protocolo usado durante la transferencia de datos. Permite la selección de los caracteres de control usados para indicar el comienzo y final de la transmisión de carga o descarga y el carácter de pass-throug.

0=No se emplea protocolo pre-definido.

Los carateres de control se definen mediante el comando IHC. El comando IHC debe ejecutarse antes de usar el comando DL si se elige esta opción.

1=Se emplea el protocolo Tektronix.

ASCII "0"-Registro transferido sin error.

ASCII "7"-Se ha detectado error. Repetir transmisión.

Los caracteres de control específico se definen mediante el uso del comando IHC, por ello debe usarse antes de la instrucción de DL.

2=Se emplea el protocolo de terminal ASR.

CTRL-R- Comenzar Download.

CTRL-S-Finalizar Download.

CTRL-Q-Comenzar Upload.

CTRL-@-Finalizar Upload.

CTRL-P-Ignorar siguiente carácter de control.

3=Se emplea protocolo de Vax o PDP-11.

CTRL-A-Comenzar Upload.

CTRL-Z-Finalizar Upload.

CTRL-V-Comenzar Download.

CTRL-W-Finalizar Download.

CTRL-P-Ignorar siguiente carácter de control.

***XA (Execute Assembles)-**

Objetivo: Producir y ensamblar programas en ensamblador desde el teclado.

DESCRIPCION DE LOS PARAMETROS:

START ADDRESS: Este parámetro, especifica la dirección de comienzo a partir de la cual se comenzará a almacenar el código objeto del programa que se esté ensamblando.

USE OLD SYMBOL TABLE[0=NO,1=YES]:

Este parámetro determina si se permite o no el uso de valores previamente definidos en la tabla de variables durante ejecuciones previas del comando XA.

***XRA (Execute Reverse Assembler)-**

Objetivo: Permite al usuario inspeccionar cualquier posición de la memoria de programas y ver la representación nemónica de su contenido. El monitor saca por pantalla un listado en el que aparecen: la dirección de memoria, su contenido, el nemónico que le corresponde y los operandos.

DESCRIPCCION DE LOS PARAMETROS:

START ADDRESS: Este parámetro especifica la dirección de comienzo a partir de la cual se comienza a desensamblar su contenido.

LINES OF OTPUT: Este parámetro especifica el número de líneas del programa objeto a sacar por pantalla.

VARIABLES:

Objetivo: Las variables son unas instrucciones especializadas que permiten al usuario un fácil acceso a los registros del emulador. Esto siplifica el proceso de ver o cambiar el contenido de una variable mediante el uso del nombre que la representa.

NOMBRES DE LAS VARIABLES:

VARIABLE	REGISTRO	VALOR MAXIMO	VALOR INICIAL
ACC	ACUMULADOR	FFFFFFFF	0
AR0	REG.AUX.0	FFFF	0
AR1	REG.AUX.1	FFFF	0
PC	CONTADOR PROGRAMA	FFF	0
ST	REFISTRO ESTADO	FFFF	3EFC
TOS	CIMA DE PILA	FFF	0
MOS	MITAD DE PILA	FFF	0
BOS	FONDO DE PILA	FFF	0
P	REG. PRODUCTO	FFFFFFFF	0
T	REG. TEMPORAL	FFFF	0

VARIABLES PARA LOS BITS DEL REGISTRO DE ESTADO:

OV (Overflow flag): Indica Overflow en operaciones aritméticas.

OVM (OVerflow Mode): Define el modo de overflow.

INTM (INTerrupt Mask): Bit de máscara para la bandera de interrupciones.

ARP (Auxiliari Register Pointer): Contiene la dirección del registro auxiliar actual.

DP (Data memory Page pointer): Puntero de página de datos de dos bits. El primer bit indica cuál de las dos páginas de memoria de datos utilizar, el segundo se reserva para futuras expansiones a cuatro páginas.

DESCRIPCION DEL CIRCUITO

DESCRIPCION DEL CIRCUITO:

El sistema global está constituido por tres placas conectadas entre sí. Existe una primera placa de bufferización que permite la conexión de nuestro sistema al PC, es la que hemos denominado placa de prototipos y hemos descrito con anterioridad.

Esta placa se conecta al bus del PC mediante un conector que se inserta en uno de los slots del PC, alimentándose del propio PC.

La segunda placa, que es la placa principal, está constituida por los elementos del sistema de proceso propiamente dichos.
En ella se incluyen el TMS32010, los 8K bytes de memoria del procesador, el 8255 y el 8237 como elementos que permiten el acceso a esos 8K de memoria desde el PC, dos latchs de ocho bits que permiten enviar un valor directamente del TMS32010 al PC, varios buffers con el fin de evitar conflictos de señales en los buses del sistema, y el conjunto de la lógica del sistema que permite que éste funcione correctamente.
Esta placa se conecta con la de prototipos por medio de dos conectores de 36 patillas de los que corresponden evidentemente uno a cada placa y están unidos a ella mediante un cable plano.

La tercera placa, que es la de convertidores, es la encargada de recibir las señales externas a procesar.
La conexión a la placa del sistema se realiza mediante conectores tipo DB37.

SISTEMA DE PROCESO

Title: PLACA PRINCIPAL

Size: D | Document Number: | REV:

Date: Noviembre 12, 1991 | Sheet of

PLACA PRINCIPAL:

En esta placa es donde recae el peso del funcionamiento del sistema.
Como se ha dicho antes, su conexión con el PC es a través de la placa de prototipos.

Las señales de entrada que recibe son las siguientes:

-Recibe las conexiones de alimentación del PC al completo, que son: +12, +5, GND, -5, y -12 voltios.
Estas alimentaciones están disponibles caso de que fuera necesario, pero en un principio no se hará uso de ellas.

-El bus de datos del PC: DB7 a DB0.

-Los cinco bits de menor peso del bus de direcciones del PC, que permiten el direccionamiento de los puertos del sistema. No se trabajará mediante puertos mapeados en memoria.
Estos bits están numerados como A4 a A0.

-Las señales de IOR y IOW, que permiten diferenciar una operación de lectura de puerto de una operación de escritura.

-La señal de RESET del PC, que permite resetear al 8255 y al 8237 a la misma vez que se resetea el PC.

-La señal de reloj del PC.

-La señal de Chip Select de la placa, que sirve para indicarle a la placa que se ha direccionado un puerto perteneciente al rango de puertos del sistema.

ALIMENTACION DE LA PLACA:

La totalidad de los integrados de esta placa funcionan con una lógica del tipo TTL y sus alimentaciones son del tipo 5v y tierra.
Es evidente pues, que para alimentar esta placa no se necesita más que una entrada de alimentación de 5v y otra de tierra.
Esta alimentación la tomaremos preferiblemente de una fuente de alimentación externa.
Aunque existe la posibilidad de tomar la alimentación del propio PC, no es recomendable esta solución, debido principalmente al elevado consumo de la placa.
La tierra de esta placa está conectada intermamente a la del PC.

FUNCIONAMIENTO BASICO DEL SISTEMA:

El TMS32010 trabajará con 4K palabras de 16 bits de memoria externa. Esta memoria externa es la denominada memoria de programa del TMS, ya que es en ella donde residen las instrucciones del programa a ejecutar, junto con los datos del programa que se quieran almacenar aquí.

El peso de la conexión del procesador con el PC recaerá en la utilización conjunta de esta memoria, ya que se podrá acceder a ella también desde el PC, lo que hará posible el enviar el programa junto con los datos necesarios desde el PC para que el TMS32010 lo ejecute, con lo cual se controla el funcionamiento del procesador. También existe la posibilidad de que el procesador almacene los resultados en una zona de memoria determinada, con lo cual el PC podrá capturarlos mediante la lectura de la información de dicha zona.

Para acceder a esta memoria sin que haya ningún tipo de conflicto con el TMS, el PC deberá colocar previamente al TMS en estado de RESET, estado en el cual puede permanecer indefinidamente. Para ello el PC dispone del puerto C del 8255, que está programado como puerto de salida, y en el que estarán depositadas las señales de control del sistema, incluido el RESET. Posteriormente se ampliará información sobre estas señales de control.

Mientras que el TMS permanezca en estado RESET, el PC tendrá vía libre para leer o escribir cualquier posición de esta memoria externa.
Una vez que el PC termine su acceso a esta memoria, deberá resetear la señal de RESET, con el fin de permitir que el TMS32010 continúe con la ejecución de su programa.

Para ello hay que tener muy en cuenta que el TMS después de un RESET comenzará la ejecución del programa a partir de la posición cero de memoria, donde estará incluido el vector de RESET.

Con el fin de permitir el envío de datos individuales del TMS al PC, se han incluido en el sistema un par de latchs de ocho bits, que recibirán la información del TMS mediante un OUT por el puerto 7, y podrán leerse desde el PC como dos puertos individuales de ocho bits.

FUNCIONES QUE DESEMPEÑA EL 8255 EN EL SISTEMA:

Con el 8255 se trabajará en modo 0, con los puertos A y B funcionando como puertos de entrada o de salida según corresponda, y con el puerto C funcionando permanentemente como puerto de salida.

El 8255 tendrá dos tareas a realizar:

-La primera de ellas consistirá en permitir, por medio de los puerto A y B, la lectura y escritura de valores en la memoria del sistema. Más adelante se ampliará la información concerniente a este tema.

-La segunda tarea consistirá en controlar las señales de control del sistema, por medio del puerto C que trabajará permanentemente como puerto de salida.

Las señales a controlar son las siguientes:

PC0 (BIT0): DRQ0 (Entrada del 8237)

PC1 (BIT1): HLDA (Entrada del 8237)

PC2 (BIT2): READY (Entrada del 8237)

PC3 (BIT3): RESET (Entrada del TMS32010)

PC4 (BIT4): INT (Entrada del TMS32010)

PC5 (BIT5): Libre

PC6 (BIT6): Libre

PC7 (BIT7): Libre

CONEXION DEL TMS32010 CON SU MEMORIA EXTERNA:

Pasemos ahora a describir la parte del sistema destinada a permitir el acceso del TMS32010 a la memoria de programa.

-En primer lugar, el bus de datos del TMS se conectará directamente al bus de datos de las memorias: los ocho bits de menor peso a uno de los integrados, y los ocho bits de mayor peso al otro. Esto es así porque cuando el TMS32010 entra en RESET el bus de datos pasa al estado de alta impedancia, con lo que no crea conflictos con los intentos de acceso por parte del PC.

-El bus de direcciones necesitará ir conectado a las memorias mediante algún integrado que lo aisle de éstas, debido a que el bus de direcciones del TMS no pasa al estado de alta impedancia durante el RESET del TMS. Para ello se emplearán dos integrados 74LS244, que son buffers unidireccionales triestados.

Como entrada recibirán las líneas del bus de direcciones del TMS y las salidas correspondientes se conectarán a las entradas de direcciones de las memorias. Así, cuando el TMS esté ejecutando un programa las memorias recibirán correctemente las direcciones pertinentes, mientras que cuando el TMS entre en estado RESET los buffers pasarán al estado de alta impedancia, con lo cual se logra el deseado aislamiento. Para que esto último se cumpla, se conectará la entrada de Chip Select de los buffers a la línea de RESET, previa inversión de ésta, con lo cual la lógica resultante hará que durante un RESET los buffers estén en alta impedancia y en caso contrario funcionen con normalidad.

-La selección de las operacines de memoria se realizará en función de las entradas de las memorias Chip Select 1 (CS1), Write Enable (WE) y Output Enable (OE), todas ellas activas a nivel bajo. Para permitir el acceso a estas entradas tanto desde el PC (por medio del 8237), como desde el TMS32010, vendrán precedidas cada una de ellas por su respectiva puerta AND de dos entradas, correspondiendo cada una de las entradas a las dos posibles vías de acceso antes mencionadas (PC o TMS).

Veamos ahora cuándo selecciona el TMS cada una de éstas entradas:

CS1: Esta entrada tendrá que activarse tanto para las operaciones de lectura como para las de escritura. Para ello se conecta a la salida de una puerta AND cuyas entradas sean las patillas del TMS MEN y WE, que son las que indican que se realiza una operación de lectura y escritura respectivamente.

OE: Esta entrada habrá de activarse cuando se desee proceder a una operación de lectura. Para ello bastará conectarla a la patilla MEN del TMS.

WE: Esta entrada habrá de activarse cuando se desee proceder a una operación de escritura. Debido a que la señal de WE del TMS se activa para las operaciones de escritura tanto de memoria (TBLW) como de puertos (OUT), habrá que emplear algún método para discernir entre ambas operaciones, ya que en caso contrario resultaría que al realizar una operación de salida por un puerto se escribiría el valor a sacar por dicho puerto en la posición de memoria correspondiente al número del puerto, además de sacar el valor efectivamente por dicho puerto. También ocurrirá que al hacer una operación de TBLW se saque un valor por un puerto.

Para evitar este error nos basaremos en el hecho de que para una operación de OUT todos los bits de direcciones, menos los tres menos significativos (A0/PA0..A2/PA2), se ponen a cero. Por ello bastará que exista al menos un bit del conjunto (A3..A11) distinto de cero para que se autorice la operación de TBLW y se inhabilite la operación OUT. En caso contrario, es decir, que todos los bits sean cero, se inhabilitará la operación de TBLW y se autorizará la de OUT.
Esta solución tiene como consecuencia negativa el hecho de que no se podrán ejecuatar operaciones de TBLW que afecten a alguna de las ocho primeras posiciones de memoria, ya que en caso de hacerlo no se produciría ninguna operación de escritura efectiva.

ACCESO DEL PC A LA MEMORIA DEL SISTEMA:

Los integrados sobre los que recae el peso de la comunicación entre el PC y la memoria del sistema son el 8255 y el 8237.

-El 8237 se encargará de genererar las direcciones de las posiciones de memoria a las que se desea acceder desde el PC, así como de generar las señales pertinentes de lectura (MEMR) o escritura (MEMW). Su funcionamiento estará controlado desde el PC, desde donde se programarán los registros de modo, dirección, etc, así como desde el 8255, que tendrá a su cargo la generación de las señales de control DRQ0, HLDA y READY.

Para la generación de las direcciones requiere de un latch complementario, en nuestro caso un 74LS373, donde almacenar los ocho bits de mayor peso (de los cuales en nuestro caso sólo emplearemos cuatro) del bus de direcciones. El propio 8237 se encarga de la actualización y control del latch según resulte pertinente.

Los cuatro bits de menor peso del bus de direcciones sirven de entrada, con el fin de direccionar los puertos del 8237, y de salida, para direccionar la memoria externa.

También las patillas IOR e IOW actúan como entradas y como salidas.

Para evitar conflictos con el PC, se emplea un buffer 74LS244 que permite el acceso a las señales del PC por parte del 8237 y para aislarlas cuando funcionan en el sentido contrario.

Este buffer únicamente abandonará el estado de alta impedancia cuando se active el Chip Select del 8237, momento en el cual el 8237 podrá recibir estas seis señales desde el PC.

NOTA: Al desarrollar la placa se cometió el error de conectarle al latch de direcciones (74LS373) los cuatro bits de mayor peso del bus de datos , en lugar de conectar los cuatro bits de menor peso, que hubiera sido lo correcto. Este error se soluciona fácilmente mediante una corrección software del direccionamiento, con el único inconveniente de que ahora podrá ocurrir que el 8237 no actualice la información del latch cuando corresponda. Estas circunstancias habrá que tenerlas en cuenta a la hora de desarrollar aplicaciones del sistema.

-La misión del 8255, como ya se ha dicho con anterioridad, es doble:

*El sistema se servirá de los puertos A y B para transmitir la información desde el PC a la memoria y viceversa. El puerto A se conecta al bus de datos de ocho bits de uno de los integrados de memoria y el puerto B al otro. Esto permite convertir el formato de palabra de 16 bits del TMS al formato de 8 bits del PC, mediante la lectura o escritura sucesiva de ambos puertos.
Cuando se termine la comunicación se deberán dejar programados estos dos puertos como puertos de entrada, con lo que se situarán en estado de alta impedancia y no interferirán el acceso a la memoria por parte del TMS32010.

*El puerto C se encargará de mantener las señales de control del 8237 (DRQ0, HLDA y READY) y del TMS (RESET e INT).

CONEXION VIA LATCH DEL TMS32010 CON EL PC:

Con la intención de crear un canal adicional de comunicación del TMS32010 con el PC se han añadido un par de latchs de ocho bits (74LS373) que permiten escribir un dato desde el TMS y leerlo desde el PC.

Uno de los latchs tendrá sus líneas de entrada de datos conectadas a los ocho bits de menor peso del bus de datos del TMS y el otro latch las tendrá conectadas a los ocho bits de mayor peso.
La escritura en dichos latchs se hará mediante una operación de OUT por el puerto 7 del TMS. Para ello tendrán que coincidir las circunstancias de que los tres bits de menor peso del bus de direcciones sean UNO ,el resto de los bits sean CERO y se active la señal de Write Enable. Evidentemente la escritura de ambos latchs será simultánea (a diferencia de la lectura, por ser el bus del PC de ocho bits).

Las líneas de salida de ambos latchs estarán conectadas al bus de datos del PC. El PC podrá acceder al valor retenido en cualquiera de ambos latchs mediante una operación de lectura de puertos. Los valores de puertos que tienen asignados son el 788 y el 789 en decimal.

DIRECCIONAMIENTO DE LOS PUERTOS DEL SISTEMA:

Los rangos de los puertos que tienen asignados los distintos elementos del sistema son los siguientes:

El conjunto total de puertos abarca del 300H al 31FH (768 a 799 en decimal).

Para los valores de puertos de 300H a 30FH (768 a 783) el elemento seleccionado será el 8237, y dicha selección se efectuará mediante un par de puertas lógicas que tendrán en consideración el Chip Select general de la placa y el bit 4 del bus de direcciones (A4). En la sección dedicada al 8237 se precisan con exactitud a qué funciones corresponden estas direcciones.

La selección del resto de los puertos (310H a 31FH) se hará por medio del integrado 74LS138, que es un multiplexor de tres entradas de selección y por tanto ocho canales.

Para los valores de puertos compredidos entre el 310H y el 313H (784-787) el elemento direccionado será el 8255. En la sección dedicada al 8255 se precisan con exactitud a qué funciones corresponden estas direcciones.

Para los valores de puertos 314H y 315H (788 y 789) se seleccionarán los latchs de comunicación entre el TMS y el PC.

El resto de los puertos (790 a 799) se quedan sin utilizar.

CONFIGURACION DEL CONECTOR CON LA PLACA DE CONVERTIDORES:

Para que el sistema de proceso de señal sea realmente útil, necesita recibir de alguna forma las señales a procesar.
En este sistema de proceso en cuestión tendremos todos los circuitos destinados a esta conexión con el mundo exterior en otra placa, que hemos llamado "Placa de convertidores", con la que se comunicará la placa principal mediante conectores tipo DB37.

En el conector correspondiente a la placa principal se han intentado situar todas las señales de interés, de cara a la transmisión de datos desde el TMS32010, y también de cara a una futura expansión del sistema.

Las señales que se incorporan a este conector son las siguientes:

-El bus de datos del TMS32010:

Patilla 1:	D15	Patilla 20:	D14
Patilla 2:	D13	Patilla 21:	D12
Patilla 3:	D11	Patilla 22:	D10
Patilla 4:	D9	Patilla 23:	D8
Patilla 5:	D7	Patilla 24:	D6
Patilla 6:	D5	Patilla 25:	D4
Patilla 7:	D3	Patilla 26:	D2
Patilla 8:	D1	Patilla 27:	D0

-Las señales del TMS MEN, DEN y WE:

Patilla 9: MEN

Patilla 28: DEN

Patilla 10: WE

-Las líneas de direccionamiento de puertos:

Patilla 29: PA0/A0

Patilla 11: PA1/A1

Patilla 30: PA2/A2

-La línea que permite distinguir entre OUT y TBLW, mediante un AND de A3 a A11:

Patilla 12: A3(AND)A4(AND)...A11

-La patilla BIO del procesador, que permite realizar un salto condicional mediante la instrucción BIOZ:

Patilla 31: BIO

-La salida de reloj del TMS32010:

Patilla 13: CLKOUT

-Las entradas INT y RESET del TMS, por si se necesita algún tipo de sincronización con estas señales:

Patilla 32: INT

Patilla 14: RESET

-Las líneas de alimentación del PC:

Patilla 33: -12V

Patilla 15: -5V

Patilla 34: GND

Patilla 16: +5V

Patilla 35: +12V

-Las líneas de entrada de señal de reloj, para permitir la posibilidad de añadirle un cristal o una señal de reloj externa al TMS:

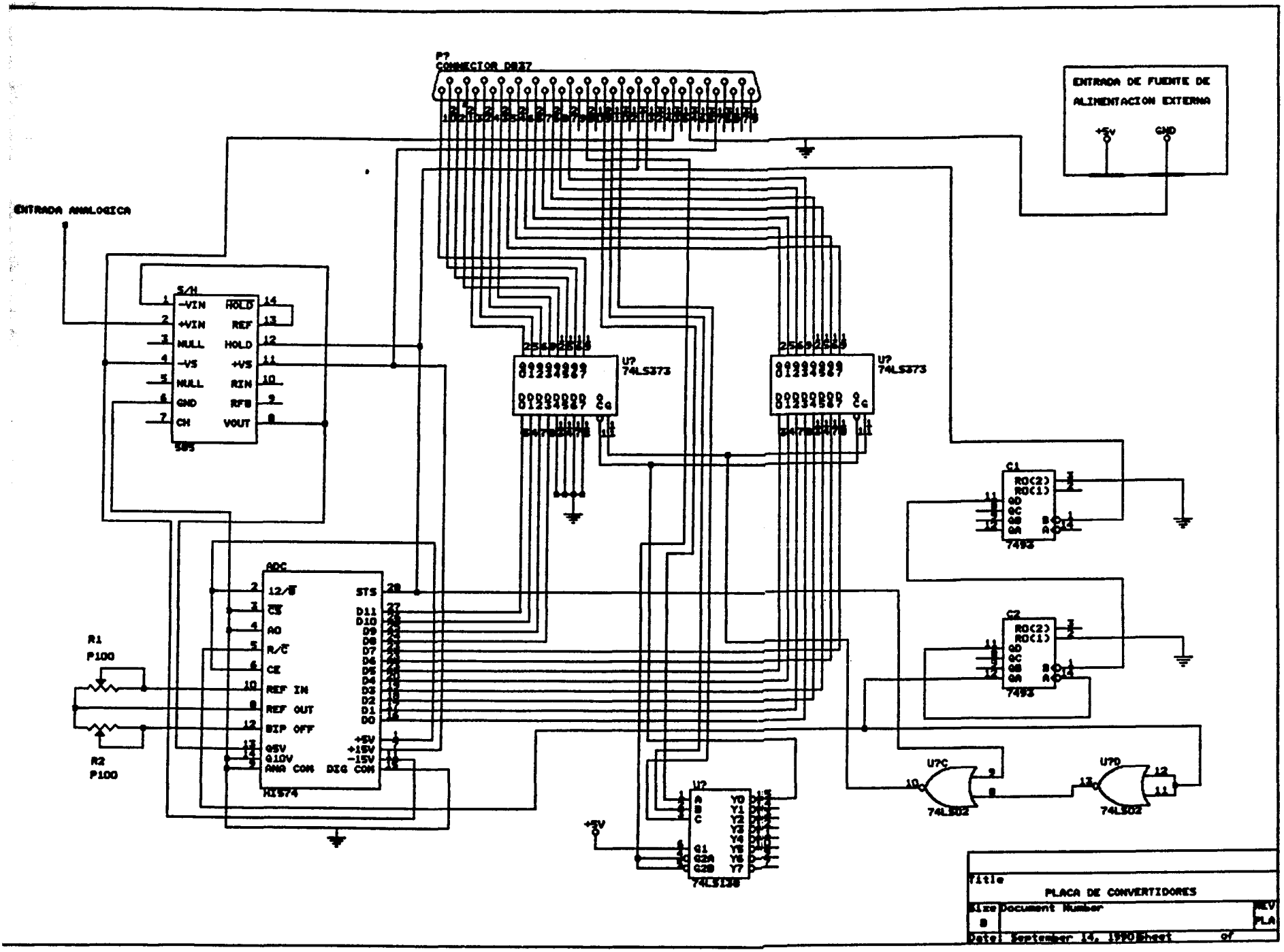
Patilla 17: X2/CLKIN

Patilla 36: X1

-La entrada de alimentación de la placa (+5V). Se supone que si tiene conectada la placa de convertidores, la alimentación de 5V se aplicará a dicha placa, y en caso contrario se aplicará directamente a esta entrada:

Patilla 37: +VCC (+5V)

-Las patillas 18 y 19 quedan libres.



PLACA DE CONVERTIDORES:

En su diseño se ha buscado la máxima sencillez. En un principio se tenía pensado incluir un convertidor digital analógico, y más concretamente un DAC10, pero debido a la filosofía de máxima sencillez ya comentada y debido a que no era imprescindible para el desarrollo del sistema, se ha optado finalmente por su no inclusión. La utilización de un 74LS138 para la lógica de selección de puertos permite una fácil inclusión a partir del diseño original de la placa de convertidores, teniendo en cuanta que el puerto de salida número 7 ya está siendo utilizado por los latchs de comunicaciones con el PC.

Los elementos de que consta esta placa son los siguientes:

-AD585(S&H): Es un integrado para el muestreo y retención de la señal analógica de entrada. Escalona la señal analógica de entrada, manteniéndola a un valor constante durante el tiempo de conversión del convertidor Analógico/Digital.

-HI574: Es el convertidor Analógico/Digital. Puede trabajar con valores de salida de 8 o 12 bits, y su tiempo de conversión para 12 bits es de 20 microsegundos. El tipo de conversión que emplea es por aproximaciones sucesivas, y posee un reloj interno para la sincronización en la conversión (No hay que confundir esta señal de reloj con la que emplea para indicar el principio de cada conversión).

En nuestro caso funcionará en el modo de 12 bits, y con una frecuencia de conversión de 39 Khz, obteniendo la señal de reloj que determina esta frecuencia mediante la división por 128 de la señal de reloj del TMS32010, que es de 5Mhz.

El HI574 puede trabajar con los siguientes rangos de tensiones de entrada:

*Entre -5V y +5V.

*Entre 0V y 10V.

*Entre -10V y +10V.

*Entre 0V y 20V.

En el diseño de esta placa se ha optado por el rango de valores de entrada comprendido entre -5V y +5V.
En este caso, la correspondencia entre valores analógicos de entrada y valores digitales de salida es la siguiente:

Valor Analogico:	Valor Digital:
-5V	000H
0V	800H
+5V	FFFH

La resolución, siendo la amplitud total de 10V y disponiendo de 12 bits para representarlos, es de 2.44 milivoltios, aunque hay que tener en cuenta que en las hojas de características se especifica un error de un LSB.

Para el ajuste de los valores de salida se dipone de dos potenciómetros de 100Ω, aunque en la práctica no se ha notado ninguna variación efectiva ante la regulación de dichos potenciómetros.

-74LS373 (Latchs, dos integrados): Debido a que la salida digital del HI574 tarda excesivo tiempo para las necesidades del TMS32010 en abandonar el estado de alta impedancia, se hace necesario el uso de estos latchs triestados de 8 bits, que reciben la salida del convertidor al final de cada conversión, y la vuelcan en el bus de datos del TMS32010 cuando este lo requiera.
La sincronización del funcionamiento del TMS32010 con el ciclo de conversión del HI574, mediante la activación de la patilla BIO del TMS32010, evita posibles errores tales como el intentar leer la información del lacth cuando está siendo actualizado, o leer dos veces el mismo valor.

La diferencia de bits que existe entre los 16 pertenecientes al bus de datos del TMS, y los 12 pertenecientes a la salida del convertidor, se compensa conectando las entradas correspondientes a los cuatro bits más significativos (pertencientes a uno de los latchs) a tierra.
Esto hace que los valores leídos por el TMS oscilen entre 000H y 0FFFH.

-7493 (Contadores, dos integrados): Su misión consiste en dividir la frecuencia de 5MHz obtenida del TMS32010 por 128, con lo que se obtiene la frecuencia de muestreo empleada por el convertidor.

-74LS02 (4 puertas NOR): Dos de sus puertas son empleadas por la lógica de sincronización.

-Un cristal de 20MHZ: Que permite obtener la señal de reloj del TMS32010, mediante su conexión a las patillas X1 y X2/CLKIN de dicho procesador. La frecuencia de salida es un cuarto de la frecuencia fundamental del cristal.

Sincronización de la conversión:

La frecuencia de 39 Khz obtenida a partir de la señal de reloj del TMS se conecta a la patilla de Read/Convert (R/C) del HI574, con lo que con cada ciclo de reloj se le indica un inicio de conversión del convertidor.
Este convertidor posee una salida de Status(STS), que permite saber cuándo está en proceso de conversión y cuándo ha acabado. Esta salida permite sincronizar el funcionamiento del Sample&Holder con el del convertidor, mediante su conexión a la patilla HOLD de este último.
También se conecta a la patilla BIO del TMS, con el fin de indicarle el final de cada conversión, y se usa junto con la señal de reloj para la actualización de la información de los latchs.

Alimentación: En esta placa está situada la entrada de alimentación de +5V, alimentacón que también abastece a la placa principal.

Los integrados AD585 y HI574 requieren además de la tensión de 5V, las tensiones de alimentación de -12V y de +12V. Estas tensiones las obtenemos de la fuente de alimentación del PC, ya que debido a su bajo consumo no presentan ningún problema. En la placa disponemos de dos interruptores para desconectarlas si fuera conveniente.

PROCESO DIGITAL:

PROCESO DE SEÑAL DIGITAL:

Pasaremos en esta sección a tratar sobre diversos elementos relativos al procesamiento de la señal digital. Empezaremos por lo más elemental, que es definir qué se entiende por señal, y más particularmente por señal digital.

Sin ir muy lejos, en un diccionario encontramos:

Señal analógica: Magnitud física variable en el tiempo de forma continua, que informa acerca de la evolución de un proceso en cada instante.

Señal digital: La que sólo puede tomar determinados valores o símbolos y que expresa la evolución de un proceso en sucesivos intervalos de tiempo.

La magnitud física que consideraremos por defecto en el caso de la señal analógica será la de tensión.

La obtención de ambas señales (analógicas y digitales) puede ser bien mediante la captación de algún fenómeno físico del mundo exterior por medio de sensores o bien por generación interna del sistema. (Ya sea, por ejemplo, por medio del uso de osciladores en el caso de señales analógicas o también por ejemplo mediante alguna función matemática en el caso de señales digitales.)

Cuando deseamos obtener una señal digital a partir de un fenómeno físico externo nos encontramos con que los sensores nos ofrecen una señal analógica.
Al contrario, cuando se obtiene una señal digital como resultado de un procesamiento de la misma y se desea actuar con ella sobre algún sistema físico externo necesitaremos de un proceso intermedio que nos convierta esta señal digital en analógica.
Esto nos lleva a la necesidad de efectuar unos procesos de conversión de señal analógica en digital y viceversa, de los que se hablará más adelante.

Ahora lo que intentaremos es resolver la siguiente cuestión: Si resulta que normalmente para trabajar con señales digitales tenenemos que partir de señales analógicas, ¿ qué ganamos con el procesamiento de la señal digital, en lugar de efectuar un proceso analógico en el que nos ahorraríamos las etapas de conversión, teniendo además en cuenta que la tecnología analógica está desarrollada desde bastante tiempo antes que la digital ?.

Las ventajas principales de la tecnología digital sobre la analógica son las siguientes:

-Para un determinado grupo de aplicaciones ofrece una mayor calidad en el resultado final, con mejor relación señal/ruido, mejor estabilidad frente a variaciones de temperatura, etc.

-Puede trabajar en una serie de campos vedados para la tecnolgía exclusivamente analógica, como por ejemplo el de los efectos visuales en los sistemas de vídeo.

-Tiene también sus ventajas a la hora del almacenamiento de la señal. Si por ejemplo se almacena una grabación de sonido en un sistema analógico, con el tiempo la calidad se deteriorará. Si el almacenamiento es digital tal deterioro no tiene por qué producirse.

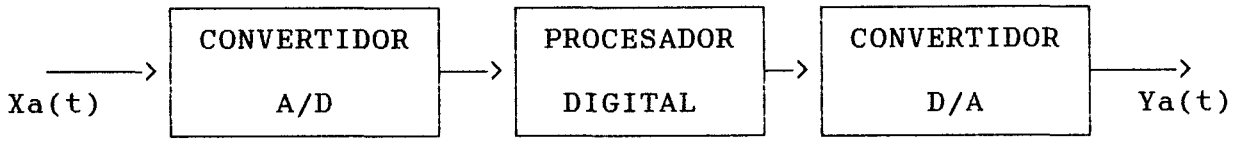
-El desarrollo de las comunicaciones con fibra óptica trae consigo una ampliación muy importante del caudal de la información a transmitir, además de una gran ventaja en lo que respecta a la inmunidad frente al ruido. La fibra óptica está diseñada para transmitir únicamente señales digitales.

-En el aspecto económico, hasta hace unos años un sistema digital que ofreciera unas prestaciones y funciones equivalentes a las de un sistema analógico era mucho más caro, por lo que normalmente se limitaba su uso a campos en los que, o bien sí podía competir en el aspecto económico, o bien quedaban vedados para los sistemas analógicos. En la actualidad, debido al enorme desarrollo de las tecnologías digitales, la relación calidad/precio se inclina cada vez más hacia el lado digital, en algunos casos de forma aplastante, por lo que es normal que sistemas que antes se desarrollaban con tecnología exclusivamente analógica ahora incluyan tecnología digital (Por ejemplo: los osciloscopios).

CONVERSION ANALOGICA/DIGITAL Y DIGITAL/ANALOGICA:

Ya hablamos en la introducción de este tema sobre la necesidad de la conversión A/D y D/A. En este apartado no se tratará de los elementos físicos que permiten esta conversión, sino del efecto que dicha conversión tiene sobre la señal a considerar.

Esquemáticamente, el proceso es el siguiente:



El primer paso en una conversión A/D consiste en muestrear la señal, obteniendo una secuencia X[n] mediante el muestreo periódico de la señal analógica.

Las muestras son:

$$X[n]=Xa(nT), \qquad -\infty<n<+\infty$$

Donde T es el periodo de muestreo, n es un entero, y 1/T es la frequencia de muestreo.

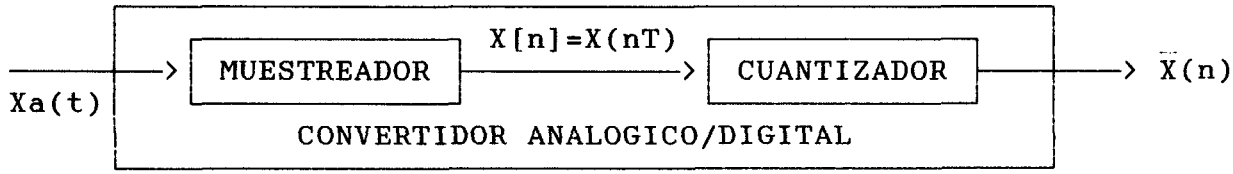
Como estas muestras se van a representar mediante números binarios con precisión finita, se tendrá que proceder a efectuar una cuantización de las mismas.

Así pues, la secuencia de las muestras cuantizadas será:

$$\bar{X}[n]=Q[X[n]]$$

Donde Q[] es una transformación no-lineal, que consiste en un proceso de redondeo o truncado hacia el valor de amplitud más próximo permitido.

PROCESO DE CONVERSION ANALOGICO-DIGITAL



A la hora de digitalizar una señal hay que tener en cuenta:

-El muestreo debe efectuarse a una frecuencia como mínimo dos veces superior a la mayor frecuencia de la señal que se está digitalizando (Teorema del muestreo).
Esto es así ya que partiendo de un determinado conjunto de muestras, existe un conjunto infinito de señales que muestreadas con un determinado período nos ofrecen un conjunto de muestras coincidente, y la única forma de establecer la unicidad necesaria para la correcta reconstrucción de la señal original es estableciendo esta limitación.

-El proceso de cuantización de las muestras añade forzosamente ruido.
Este proceso consiste en asignar un número binario a cada nivel de entrada.

Auque la asignación de códigos binarios a los niveles de entrada es arbitraria, resulta mucho más adecuado efectuar la asignación de códigos binarios según un esquema que permita una implementación optimizada de las operaciones aritméticas a realizar sobre las muestras (Por Ejemplo: En complemento a dos). Una vez fijado el número de niveles de cuantización (usualmente comprendido entre 2^8 y 2^{16}), la representación numérica binaria se relaciona con la amplitud de la señal analógica mediante el escalón de cuantización α.

La elección de α depende de la amplitud pico a pico de la señal. Si se usa un código de n bits, entonces α debe elegirse de tal forma que se cumpla la relación:

$$\alpha.2^n = \text{Amplitud de la señal pico a pico.}$$

En estas circunstancias, el máximo error en una muestra será de $\pm\alpha/2$, con lo cual el error de cuantización medio será proporcional a α.

Cuando se trabaja de esta forma, con un escalón o paso fijo entre distintos valores, ocurrirá que el error relativo aumentará según disminuya la amplitud de las muestras , y si resulta que se está trabajando con rangos dinámicos muy amplios serán necesarios un gran número de niveles de cuantización para mantener el error relativo de cuantización dentro de unos límites tolerables.

Aclaremos esto un poco mediante un ejemplo:

Si por ejemplo disponemos de 256 niveles de cuantización y se

prevee que la máxima señal de entrada va a ser de 10 voltios pico a pico, estableceremos un escalón de aproximadamente 78 milivoltios de separación entre un valor digital y otro.
Si resulta que en un momento dado la señal de entrada pasa a ser de 1 voltio pico a pico, el rango de valores digitales que la representan queda reducido a una décima parte (en nuestro caso serían unos 25 o 26), y por tanto el error relativo se decuplica.

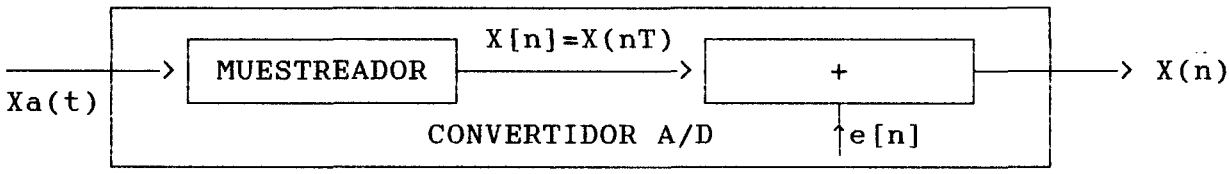
Para evitar este problema, existen como soluciones alternativas o bien usar un conjunto no uniforme de niveles de cuantización (por ejemplo aumentando el escalón a medida que aumenta el valor de las muestras) o bien adaptar el escalón de cuantización de forma dinámica a la amplitud de la señal de entrada.

En el caso de trabajar con un escalón uniforme no adaptativo, resulta de utilidad representar la señal cuantizada de la siguiente forma:

$$\bar{X}[n]=X[n]+e[n]$$

donde e[n] es por definición el error de cuantización, y resulta que: $-\alpha/2 <= e[n] < +\alpha/2$.

Esto se representa esquemáticamente:



El error cuadrático medio es proporcional a α, el cual es en cambio inversamente proporcional a 2^{n}, donde n es el número de bits del código binario de las muestras.

La relación entre la señal a cuantizar y el ruido de cuantización (SIGNAL/NOISE RATIO), definida como:

$$SNR=10.\log_{10}(\text{potencia de señal/potencia de ruido}).$$

disminuye en 6 decibelios por cada bit que se le añada al tamaño del número binario empleado para la representación de las muestras.

Otro elemento importante a tener en cuenta, es el de que desde el punto de vista estadístico, el ruido añadido a la secuencia de muestras parece estar uniformemente distribuido en amplitud y sin que exista una correlación entre muestra y muestra siempre que el número de niveles de cuantización sea lo suficientemente amplio. Es por tanto que el modelo de convertidor analógico/digital representado en la página anterior, se puede considerar como un muestreador ideal, a cuya salida se le añade un ruido blanco cuya potencia aumenta exponencialmente a medida que el número de bits por muestra disminuye.

REPRESENTACION DE LAS SEÑALES DISCRETAS:

Las señales de tiempo discreto son aquellas que únicamente están definidas para unos valores de tiempo determinados:
t=nT, donde n=0,1,2,..,N-1. y T es el perido de muestreo.

Estas señales se representan matemáticamente mediante series de números cuya amplitud puede variar de forma continua.

Una serie puede obtenerse de varias maneras:

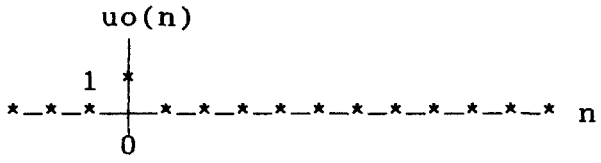
-Una de ellas consiste en generar un conjunto de números y ordenarlos en una sucesión.
Por ejemplo:
0,2,4,8,..,2*(N-1) que forman la sucesión en rampa h(n)=2*n para n comprendida entre 0 y (N-1); 0<=n<=N-1.

-También se puede obtener una serie mediante una relación recursiva, como por ejemplo: h(n)=h(n-1)*2 para 0<n<N y h(0)=7.

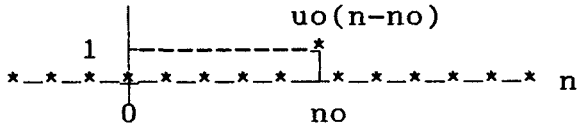
-Otra forma, que va a ser la que más nos va a interesar consistirá en muestrear una señal continua y usar las amplitudes de las muestras para formar una secuencia, que es lo que habíamos visto en el apartado anterior como conversión analógico-digital.

Algunas señales de interés son las siguientes:

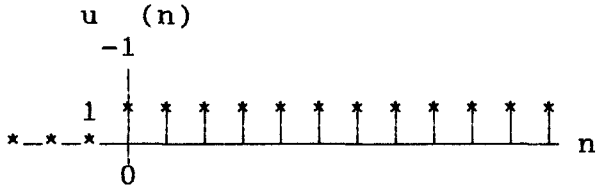
a)



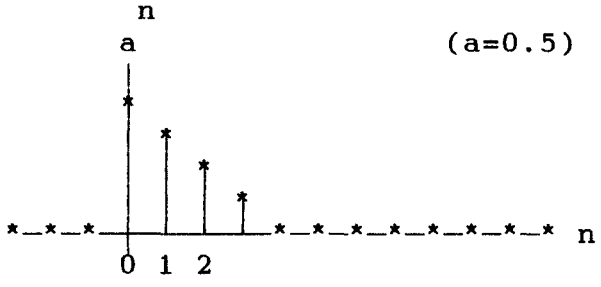
b)



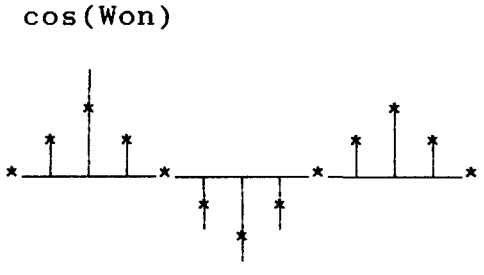
c)



d)



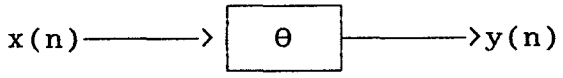
e)



SISTEMAS LINEALES INVARIANTES EN EL TIEMPO:

Un sistema de tiempo discreto es esencialmente un algoritmo para convertir una secuencia, la secuencia de entrada, en otra secuencia, que será la secuencia de salida.

Esto se representa gráficamente así:



Si llamo x(n) a la entrada e y(n) a la salida, la relación existente entre ambas quedará determinada por: $y(n)=\Theta[x(n)]$, donde $\Theta()$ dependerá de cada sistema específico.

Pasemos a definir una serie de conceptos fundamentales:

SISTEMA LINEAL (LINEAR SYSTEM):

Sean x1(n) y x2(n) dos entradas determinadas a un sistema específico, y sean y1(n) e y2(n) sus salidas respectivas.
Dicho sistema será lineal si se cumple que para la secuencia de entrada a*x1(n)+b*x2(n) le corresponde como salida la secuencia a*y1(n)+b*y2(n), donde a y b son constantes arbitrarias.

SISTEMA INVARIANTE EN EL TIEMPO (TIME INVARIANT SYSTEM):

En un sistema invariante en el tiempo ocurrirá que si para una secuencia de entrada x(n) obtenemos una secuencia de salida y(n),

entonces para una secuencia de entrada x(n-no) obtendremos como salida y(n-no) para todo valor de no.
Es decir, para una entrada x(n) se produce una salida y(n) independientemente del momento en que se produzca la entrada.

CONVOLUCION:

En un sistema lineal invariante en el tiempo, que llamaremos sistema LTI (Linear Time-Invariant) para abreviar, existe una relación de convolución entre la entrada y la salida.

Esta relación de convolución se define de la siguiente forma:

Sea h(n) la respuesta a un impulso unitario uo(n), llamada respuesta a impulso o respuesta a muestra unitaria.
Se puede escribir x(n) como:

$$x(n) = \sum_{m=-\infty}^{\infty} x(m).uo(n-m)$$

Como h(n) es la respuesta a uo(n), debido a que el sistema es invariante en el tiempo podemos decir que h(n-m) es respuesta a uo(n-m).
Además, debido a la linealidad tendremos que la respuesta a la secuencia x(m).uo(n-m) ha de ser x(m).h(n-m)
Como consecuencia tenemos que la respuesta a x(n) será:

$$y(n) = \sum_{m=-\infty}^{\infty} x(m).h(n-m)$$

La secuencia h(n) caracteriza completamente a un sistema LTI.

CAUSALIDAD:

Se dice que un sistema LTI es causal o realizable, si la salida para n=no es función únicamente de los valores de entrada para n<=no, es decir, la respuesta es función únicamente de los valores de entrada pasados y nunca de los valores de entrada futuros, lo cual no tendría sentido.

Hay que tener en cuenta además, que en un sistema LTI h(n) ha de ser cero para n<0.

Algunos sistemas importantes no cumplen esta condición de causalidad, como son el filtro paso bajo ideal o el diferenciador ideal, por lo que su implementación no podrá ser mas que mediante sistemas aproximados.

ESTABILIDAD:

La condición para que un sistema LTI sea estable, consiste en que para toda entrada acotada dentro de unos límites, produzca una salida igualmente acotada, aunque evidentemente no necesariamente dentro de los mismos límites.

Una condición necesaria y suficiente para que un sistema sea estable consiste en que la suma de sus respuestas a un tren de impulsos sea finita:

$$\sum_{n=-\infty}^{\infty} |h(n)| < \infty$$

ECUACIONES DE DIFERENCIA:

Una parte, un subconjunto, de los sistemas lineales invariantes en el tiempo (LTI) pueden representarse mediante una ecuación de diferencias lineal con coeficientes constantes que relacione la secuencia de entrada x(n) con la secuencia de salida y(n).
Mediante la representación de un sistema de esta forma se consigue un planteamiento mucho más intuitivo del mismo, lo que normalmente facilita el encontrar formas eficientes de realización de dicho sistema.

La representación más general de un sistema lineal causal de orden M, mediante ecuaciones de diferencia con coeficientes constantes es la siguiente:

$$Y(n) = \sum_{i=0}^{M} (b_i)*X(n-i) - \sum_{i=1}^{M} (a_i)*Y(n-i) \qquad \text{para } n>=0$$

Donde a_i y b_i caracterizan al sistema apropiado y $a_M <>0$.

TRANSFORMADA DISCRETA DE FOURIER:

La transformada discreta de fourier es una valiosa herramienta en lo que respecta a la teoría y diseño de sistemas discretos.

Su definición es la siguiente:

$$X(e^{jwT}) = \sum_{n=-\infty}^{\infty} X(n).e^{-jwnT}$$

Siendo la definición de la transformada inversa:

$$X[n] = \frac{T}{2\pi} \cdot \int_{-\pi}^{+\pi} X(e^{jwT}) \cdot e^{jwT} \cdot dw$$

TRANSFORMADA Z:

Es una de las técnicas más útiles y cómodas para la representación y manipulación de secuencias discretas.
Su definición es la siguiente:

Dada una secuencia X(n), definida para todo n, su transformada Z es:

$$X(Z) = \sum_{n=-\infty}^{\infty} X(n).Z^{-n}$$

Donde Z es una variable compleja. Esta función compleja estará definida únicamente para aquellos valores de Z para los que converja la serie de potencias .
Como normalmente se trabajará con secuencias causales, resulta útil definir la transformada Z de un lado (one sided Z transform) como:

$$X(Z) = \sum_{n=0}^{\infty} X(n).Z^{-n}$$

Por otra parte se define la transformada inversa de Z como:

$$X(n) = \frac{1}{2\pi j} \cdot \oint_{c1} X(Z).Z^{(n-1)} \cdot dZ$$

Comparando la transformada de fourier con la transformada Z se observa que:

$$X(e^{jwT}) = X(Z)\Big|_{z=e^{jwT}}$$

Una de las más importantes razones para el uso de la representación en el dominio de la frecuencia es el hecho de que si Y(n) es la salida de un sistema LTI, entonces su transformada Z, y por analogía la transformada de fourier, satisface la ecuación:

$$Y(z)=H(z).X(z)$$

Donde H(z) y X(z) son las transformadas Z de la respuesta del sistema a la entrada impulso unitario, y de la entrada al sistema respectivamente.

TRANSFORMADA RAPIDA DE FOURIER:

Si se desea evaluar directamente la transformada discreta de fourier en los N puntos de una señal discreta, a partir de la expresión que se había definido con anterioridad resultará que habremos de realizar $4N^2$ multiplicaciones reales y N(4N-2) sumas reales. Este número de operaciones lleva aparejado un tiempo de cálculo considerable, que hace impracticable la evaluación directa de la DFT en la gran mayoría de los casos.
Basandose en una serie de simetrías y periodicidades, existen una serie de algoritmos que optimizan el tiempo de cálculo,

permitiendo reducir el número de operaciones a $N\log_2 N$ sumas complejas y $0.5N\log_2 N$ multiplicaciones complejas.

Estos algoritmos se conocen como algoritmos de FFT (Fast Fourier Transform) y se basan en la descomposicón del cálculo de la DFT de una secuencia de N puntos en varias DFT de secuencias más pequeñas.

DESCRIPCION DEL SOFTWARE DE CONTROL

DESCRIPCION DEL SOFTWARE DE CONTROL:

Pasaremos ahora a describir el software que nos permite sacarle rendimiento a nuestro sistema. Empezaremos describiendo las funciones básicas.

FUNCIONES BASICAS:

Para acceder a la memoria de programa del TMS32010 desde nuestro PC, contamos con dos funciones básicas que nos permiten la lectura y escritura de una posición a especificar de dicha memoria. Ambas funciones, como el resto del software de control han sido desarrolladas mediante el lenguaje de programación C, más concretamente mediante el compilador de Turbo C, versión 2.0.

Función lee(pos,y):
pos: Número entero.
y: Puntero de número entero.

Objetivo: Esta función lee el contenido de la posición de memoria direccionada por la variable pos, que puede tomar un valor comprendido entre cero y 4095, y almacena el resultado en la posición de memoria del PC indicada por el puntero y.

Funcionamiento:
Los pasos que sigue en su ejecución esta función son los que siguen:

-En primer lugar saca por el puerto C, al que le corresponde la dirección 786 (en decimal), el valor 20 (00010100), con lo que se ponen todos los bits de control a cero, excepto el de READY e INT.

-Se efectúa un master-clear.

-Se programa el registro de comando con el valor cero, con el fin de evitar valores de encendido indeseados.
De esta forma se define DREQ como activo a nivel alto, se deshabilita la transferencia de memoria a memoria, se selecciona el modo de "normal timing", y otra serie de opciones que no vamos a comentar.

-Se programan los cuatro canales para:

* Single mode.
* Address increment.
* Auto initialization disable.
* Read transfer.

El motivo por el que se programan los cuantro canales, en vez de programar únicamente el canal cero, que es el único que emplearemos, se debe a que no es recomendable dejar ningún registro de control sin programar, para evitar estados indeseados.

-Se elimina el bit de máscara del canal cero, con el fin de habilitar la petición de DRQ0 de dicho canal.

-Se programan los registros de base y current address del canal cero:

Para ello se hace primero un clear del flip-flop interno, para después programar primero el byte de menor peso y por último el de mayor peso. Al byte de mayor peso lo multiplicamos por 16, con el fin de desplazar los cuatro bits de mayor peso del bus de direcciones a los cuatro bits de mayor peso correspondientes a este segundo byte, esto se hace para compensar el error hardware de las conexiones al latch externo asociado al 8237.

-Se programan los registros de base y current word count del canal cero:

Para ello se hace primero un clear del flip-flop interno, para después programar primero el byte de menor peso y por último el de mayor peso.

Por el byte de menor peso sacamos un uno, y por el de mayor peso un cero, con lo que se programa la cuenta de palabras a uno.

-Se solicita servicio a la DMA, mediante la activación de la línea de DRQ0, poniéndola a uno.

-Se autoriza la transferencia, poniendo la línea de HLDA a uno.
Se prolonga el ciclo de lectura poniendo la línea READY a cero.

-Se leen los valores de memoria:

* Se almacena el valor leído por el puerto A en la posición de memoria direccionada por y.
* Se almacena el valor leído por el puerto b en la posición de memoria siguiente a la direccionada por y.

-Se ponen los bits de READY e INT a uno, el resto a cero.

Aquí se acaba la ejecución de esta instrucción.

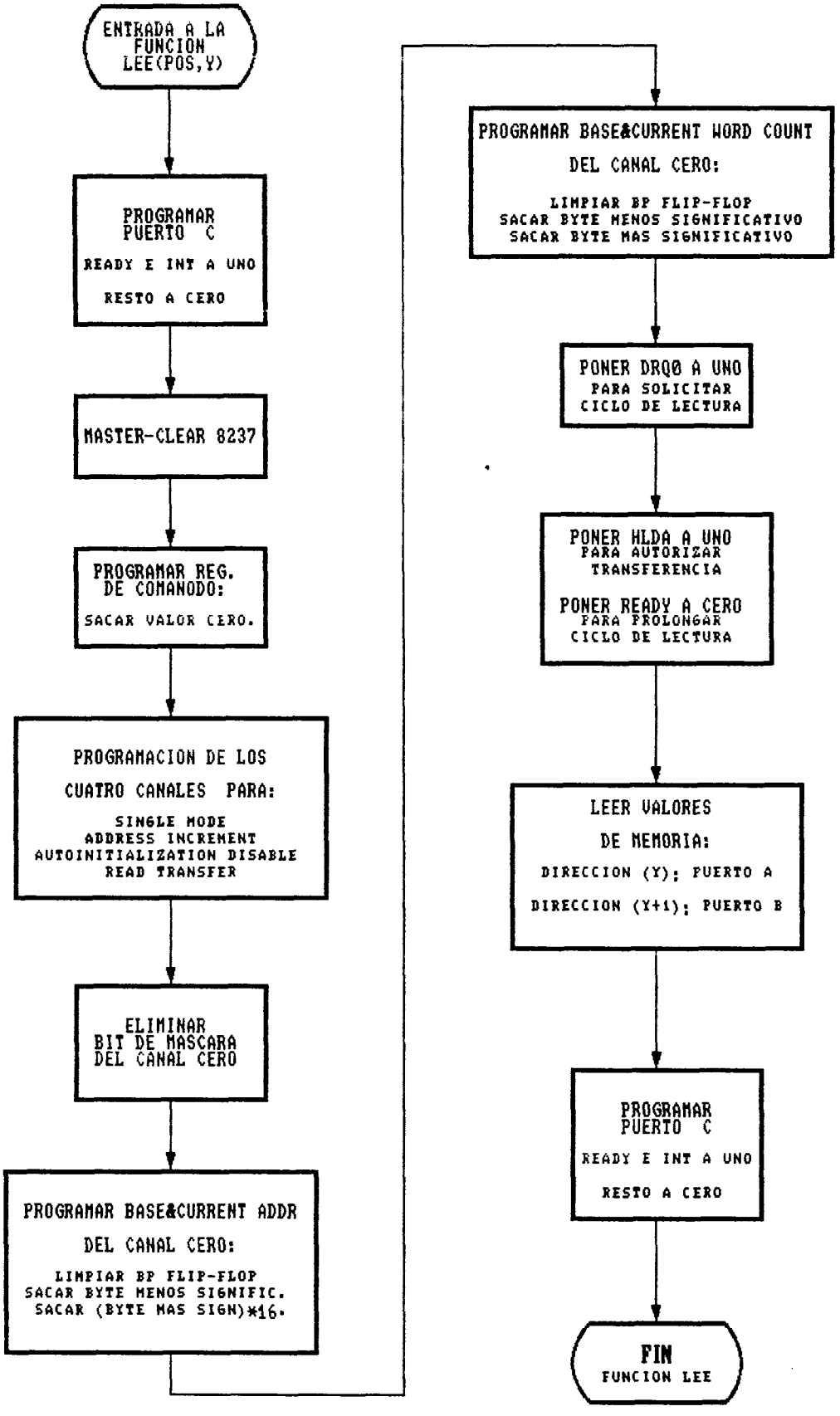
LISTADO DE LA FUNCION LEE:

```
/* LA SIGUIENTE FUNCION LEE LA POSICION DE MEMORIA
   DEL SISTEMA INDICADA POR POS */

void lee(pos,y)
int pos,*y;
{
   unsigned char c1,c2,*x;


/* Se programa el puerto C de la siguiente
   forma:                                                    */

   outportb(786,20);     /* C=00010100 BIN

                            READY e INT A 1 */

/* Master Clear para el 8237 */

   outportb(781,0);


/* Programación del registro de comando    */

   outportb(776,0);

/* Pragramación del canal cero para:

        SINGLE MODE
        ADDRESS INCREMENT
        AUTOINITIALIZATION DISABLE
        READ TRANSFER                              */

   outportb(779,72);  /* Canal 0 */
   outportb(779,73);  /* Canal 1 */
   outportb(779,74);  /* Canal 2 */
   outportb(779,75);  /* Canal 3 */

/* Eliminar mask bit para canal cero */

   outportb(778,0);
```

```
/* Programar BASE y CURRENT ADDRESS del canal cero */

   outportb(780,0);     /* Clear BP Flip-Flop */


/* programar current address en función de pos,
   para ello hay que obtener el byte menos significativo,
   y el más significativo en función de el valor de pos y
   compensando el error de direccionamiento hardware*/

   x=&pos;
   c1=*x;
   c2=(*(x+1))*16;
   outportb(768,c1);
   outportb(768,c2);

/* Programar Base y Current Word Count del canal cero */

   outportb(780,0);    /* Clear BP Flip-Flop */

   outportb(769,10);   /* Word count=1 */
   outportb(769,0);

/* Poner bit de DRQ0 a 1 */

   outportb(786,21);   /* Canal C=00010101
                          DRQ0, INT y READY a 1 */

/* Poner HLDA a 1 para autorizar la transferencia y
   READY a 0 para prolongar el ciclo de lectura     */

   outportb(786,19);   /* Canal C=00010011 */

/* LEER LOS VALORES DE MEMORIA */

   *y=inportb(784);
   *(y+1)=inportb(785);
   outportb(786,28);   /* Canal C=00011100 */
}
```

NOTA: Al empezarse a ejecutar esta instrucción, tiene que estar programado el 8255 en modo cero, con los puertos A y B definidos como puertos de entrada, y el puerto C definido como puerto de salida. Esto hay que tenerlo muy en cuenta a la hora de efectuar una llamada a esta instrucción.

FUNCION LEE(POS,Y)



Función escribe(pos,val):

pos: Número entero.

val: Puntero de número entero.

Objetivo: Esta función escribe en la posición de memoria direccionada por la variable pos, que puede tomar un valor comprendido entre cero y 4095, el valor contenido en la posición de memoria direccionada por el puntero val.

Funcionamiento:

Los pasos que sigue en su ejecución esta función son los que siguen:

-Para empezar, se programan los tres puertos del 8255 como puertos de salida.

-Se saca por el puerto C, al que le corresponde la dirección 786 (en decimal), el valor 20 (00010100), con lo que se ponen todos los bits de control a cero, excepto el de READY e INT.

-Se sacan por los puertos los valores a escribir:

* Por el puerto A el byte direccionado por el puntero val.
* Por el puerto B el byte correspondiente a la dirección (val+1)

-Se efectúa un master-clear.

-Se programan los cuatro canales para:

* Single mode.
* Address increment.

* Auto initialization disable.

* Write transfer.

El motivo por el que se programan los cuantro canales, en vez de programar únicamente el canal cero, que es el único que emplearemos, se debe a que no es recomendable dejar ningún registro de control sin programar, para evitar estados indeseados.

-Se elimina el bit de máscara del canal cero, con el fin de habilitar la petición de DRQ0 de dicho canal.

-Se programan los registros de base y current address del canal cero:
Para ello se hace primero un clear del flip-flop interno, para después programar primero el byte de menor peso y por último el de mayor peso. Al byte de mayor peso lo multiplicamos por 16, con el fin de desplazar los cuatro bits de mayor peso del bus de direcciones a los cuatro bits de mayor peso correspondientes a este segundo byte, esto se hace para compensar el error hardware de las conexiones al latch externo asociado al 8237.

-Se programan los registros de base y current word count del canal cero:
Para ello se hace primero un clear del flip-flop interno, para después programar primero el byte de menor peso y por último el de mayor peso.

Por el byte de menor peso sacamos un uno, y por el de mayor peso un cero, con lo que se programa la cuenta de palabras a uno.

-Se solicita servicio a la DMA, mediante la activación de la línea de DRQ0, poniéndola a uno.

-Se autoriza la transferencia, poniendo la línea de HLDA a uno. Se prolonga el ciclo de lectura poniendo la línea READY a cero.

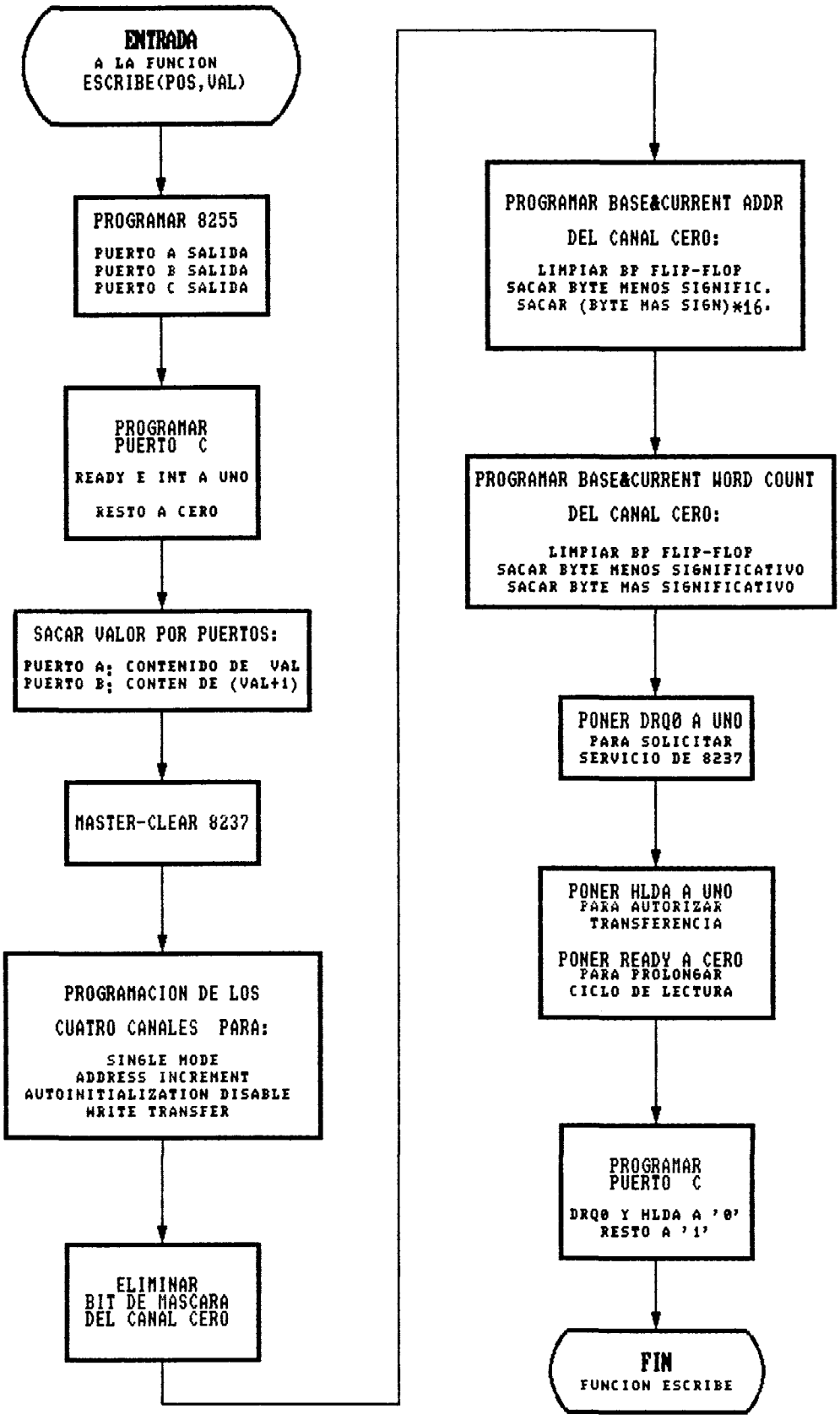
-Se ponen los bits de DRQ0 y HLDA a cero, el resto a uno. Aquí se acaba la ejecución de esta instrucción.

LISTADO DE LA FUNCION ESCRIBE:

```
void escribe(pos,val)
int pos,*val;
{
   unsigned char *x,c1,c2;

/* El primer paso consiste en programar el 8255 en mode 0,
   con los puertos A,B y C como salida       */

   outportb(787,128);

/* Se programa el puerto C de la siguiente
   forma:                                                  */

   outportb(786,20);     /* C=00010100 BIN

                            READY e INT A 1 */

/* Sacar por los puertos A y B los valores a escribir  */

   outportb(784,*val);
   outportb(785,*(val+1));

/* Master Clear para el 8237 */

   outportb(781,0);
```

```
/* Pragramación de los cuatro canales para:

        SINGLE MODE
        ADDRESS INCREMENT
        AUTOINITIALIZATION DISABLE
        WRITE TRANSFER                              */

   outportb(779,68);  /* Canal 0 */
   outportb(779,69);  /* Canal 1 */
   outportb(779,70);  /* Canal 2 */
   outportb(779,71);  /* Canal 3 */

/* Eliminar mask bit para canal cero */

   outportb(778,0);

/* Programar BASE y CURRENT ADDRESS del canal cero */

   outportb(780,0);    /* Clear BP Flip-Flop */

/* programar current address en función de pos,
   para ello hay que obtener el byte menos significativo,
   y el más significativo en función de el valor de pos y
   compensando el error de direccionamiento hardware*/

   x=&pos;
   c1=*x;
   c2=(*(x+1))*16;

   outportb(768,c1);
   outportb(768,c2);

/* Programar Base y Current Word Count del canal cero */

   outportb(778,0);   /* Clear BP Flip-Flop */

   outportb(769,1);  /* Word count=1 */
   outportb(769,0);

/* Poner bit de DRQ0 a 1 */

   outportb(786,21);  /* Canal C=00010101
                         DRQ0, INT y READY a 1 */

/* Poner HLDA a 1 para autorizar la transferencia  */

   outportb(786,23);  /* Canal C=00010011 */

/* Poner DRQ0 y HLDA a CERO, poner READY a uno */

   outportb(786,252);
}
```

FUNCION ESCRIBE(POS,VAL)



PROGRAMA DE COTROL:

Con el fin de permitir una serie de opciones de control del sistema, que nos posibiliten entre otras cosas modificar y almacenar en el PC programas desarrollados en el emulador, para su posterior carga directa, se ha desarrollado un programa de utilidad que nos permite las siguientes opciones:

-Mirar bloque de memoria: Esta opción nos representa de forma paginada en pantalla los diferentes valores leídos de la memoria externa del sistema. Los valores se representan en hexadecimal.

-Copiar bloque de memoria en fichero: Eligiendo esta opción, podemos almacenar la información correspondiente a una zona de la memoria externa en un fichero. Así podemos o bien guardar en el PC un programa creado en el emulador, o bien almacenar una zona de datos que nos interese, sea el caso de una serie de valores obtenidos por el convertidor analógico-digital.

-Cargar fichero en memoria: Esta opción es complementaria de la anterior, puesto que nos permite cargar la información de un fichero obtenido previamente en la memoria del sistema.

-Rellenar zona de memoria: Esta última opción tiene como misión el rellenar una zona de memoria especificada desde teclado con un valor igualmente especificado desde teclado.

A continuación tenemos el listado del programa:

PROGRAMA MEMO.C

```
#include<conio.h>
#include<ctype.h>
#include<stdio.h>

/* Definición de las variables globales    */

 char m1[4096];


/* LA SIGUIENTE FUNCION LEE LA POSICION DE MEMORIA
   DEL SISTEMA INDICADA POR POS */

void lee(pos,y)

int pos,*y;

{
   unsigned char c1,c2,*x;


/* Se programa el puerto C de la siguiente
   forma:                                                    */

   outportb(786,20);     /* C=00010100 BIN

                            READY e INT A 1 */

/* Master Clear para el 8237 */

   outportb(781,0);


/* Programación del registro de comando   */

   outportb(776,0);


/* Pragramación del canal cero para:

        SINGLE MODE
        ADDRESS INCREMENT
        AUTOINITIALIZATION DISABLE
        READ TRANSFER                          */
```

```
   outportb(779,72);  /* Canal 0 */
   outportb(779,73);  /* Canal 1 */
   outportb(779,74);  /* Canal 2 */
   outportb(779,75);  /* Canal 3 */

/* Eliminar mask bit para canal cero */

   outportb(778,0);

/* Programar BASE y CURRENT ADDRESS del canal cero */

   outportb(780,0);    /* Clear BP Flip-Flop */

/* programar current address en función de pos,para ello hay que
obtener el byte menos significativo,y el más significativo en
función de el valor de pos y compensando el error de
direccionamiento hardware*/

   x=&pos;
   c1=*x;
   c2=(*(x+1))*16;
   outportb(768,c1);
   outportb(768,c2);

/* Programar Base y Current Word Count del canal cero */

   outportb(780,0);   /* Clear BP Flip-Flop */

   outportb(769,10);  /* Word count=1 */
   outportb(769,0);

/* Poner bit de DRQ0 a 1 */

   outportb(786,21);  /* Canal C=00010101
                         DRQ0, INT y READY a 1 */

/*  Poner  HLDA a 1 para autorizar la transferencia y READY  a  0
para prolongar el ciclo de lectura    */

   outportb(786,19);  /* Canal C=00010011 */

/* LEER LOS VALORES DE MEMORIA */

   *y=inportb(784);
   *(y+1)=inportb(785);
   outportb(786,28);  /* Canal C=00011100 */
}
```

```
void escribe(pos,val)

int pos,*val;

{
   unsigned char *x,c1,c2;

/* El primer paso consiste en programar el 8255 en mode 0,con los

puertos A,B y C como salida       */

   outportb(787,128);

/* Se programa el puerto C de la siguiente

   forma:                                                    */

   outportb(786,20);     /* C=00010100 BIN

                            READY e INT A 1 */

/* Sacar por los puertos A y B los valores a escribir  */

   outportb(784,*val);
   outportb(785,*(val+1));

/* Master Clear para el 8237 */

   outportb(781,0);

/* Pragramación de los cuatro canales para:

        SINGLE MODE
        ADDRESS INCREMENT
        AUTOINITIALIZATION DISABLE
        WRITE TRANSFER                              */

   outportb(779,68);  /* Canal 0 */
   outportb(779,69);  /* Canal 1 */
   outportb(779,70);  /* Canal 2 */
   outportb(779,71);  /* Canal 3 */

/* Eliminar mask bit para canal cero */

   outportb(778,0);

/* Programar BASE y CURRENT ADDRESS del canal cero */

   outportb(780,0);     /* Clear BP Flip-Flop */
```

```
/* programar current address en función de pos, para ello hay que
obtener el byte menos significativo, y el más significativo en
función de el valor de pos y compensando el error de
direccionamiento hardware*/

   x=&pos;
   c1=*x;  c2=(*(x+1))*16;

   outportb(768,c1);   outportb(768,c2);

/* Programar Base y Current Word Count del canal cero */

   outportb(778,0);   /* Clear BP Flip-Flop */

   outportb(769,1); outportb(769,0);  /* Word count=1 */

/* Poner bit de DRQ0 a 1 */

   outportb(786,21);  /* Canal C=00010101
                         DRQ0, INT y READY a 1 */

/* Poner HLDA a 1 para autorizar la transferencia  */

   outportb(786,23);  /* Canal C=00010011 */

/* Poner DRQ0 y HLDA a CERO, poner READY a uno */

   outportb(786,252);
}

void borde(void)
{
 int a;
 gotoxy(1,1);putch('╔'); gotoxy(79,1);putch('╗');
 gotoxy(1,24);putch('╚'); gotoxy(79,24);putch('╝');

 for(a=2;a<13;a++)
   {
    gotoxy(a,1);putch('=');gotoxy((80-a),1);putch('=');
    gotoxy(1,a);putch('║');gotoxy(1,25-a);putch('║');
    gotoxy(79,a);putch('║');gotoxy(79,25-a);putch('║');
    gotoxy(a,24);putch('=');gotoxy((80-a),24);putch('=');
   }
   gotoxy(13,24);
   cprintf("═══════════════════════════════════════════════════
   ═══");
   gotoxy(13,1);
   cprintf("═══════════════════════════════════════════════════
═══");
}
```

```
int menu1(pun)
char *pun;
{
 int a;
 textmode(LASTMODE);
 puttext(1,1,80,25,pun);
 gotoxy(62,17);
 a=cual();
 return(a);
}

int menu(pun)
char *pun;
{
   int a;
   borde();
   gotoxy(35,2);cprintf("Opción:");
   gotoxy(34,1);putch('╤');
   gotoxy(42,1);putch('╤');
   gotoxy(34,2);putch('│');
   gotoxy(42,2);putch('│');
   gotoxy(34,3);putch('└');
   gotoxy(42,3);putch('┘');
   gotoxy(35,3);cprintf("═══════");
   gotoxy(21,9);
   cprintf("  1.- Mirar bloque de memoria");
   gotoxy(21,11);
   cprintf("  2.- Copiar bloque de memoria en fichero");
   gotoxy(21,13);
   cprintf("  3.- Cargar fichero en memoria");
   gotoxy(21,15);cprintf("  4.- Rellenar zona de memoria");
   gotoxy(21,17);
   cprintf("  5.- Finalizar la ejecución del programa");
   gettext(1,1,80,25,pun);
   a=cual();
   return(a);
}
int cual()
{
 int a;
 char b;
 b=bioskey(0);
 switch(b)
    {
     case '1':return(1);
     case '2':return(2);
     case '3':return(3);
     case '4':return(4);
     case '5':return(5);
     default:a=cual();return(a);
    }
}
```

```
int num(str)
char *str;
{
 int l,i;
 l=strlen(str);
 for (i=0;i<l;i++)
    if(!isdigit(*(str+i))) return(-1);
 return(0);
}

void pag(dir)
int *dir;
{
 int i,a,*c;
 int b[2];
 c=&b[0];
 clrscr();
 for(i=1;i<16;i+=2)
   {
        gotoxy(1,i);
        highvideo();
        cprintf("  %d: ",*dir);
        normvideo();
        for(a=12;a<55;a+=6) {
                             gotoxy(a,i); lee((*dir)++,c);
                             cprintf("%x%x ",b[0],b[1]);
                           }
   }
}

void mir(void)
{
 int dir,a,i;
 unsigned char b;
 char val[20];
 gotoxy(13,1);
 cprintf("═══════════════════════════════════════════════════
 ═══════");
 window(2,2,78,23);
 clrscr();
 gotoxy(19,9);cprintf("Dirección de memoria (entre 0 y 4095):");
 scanf("%s",&val[0]);
 dir=atoi(&val[0]);
 if (num(&val[0])<0) dir=-1;
 while ((dir<0)||(dir>4095))
     {
      clrscr();
      gotoxy(19,9);
      cprintf("Dirección de memoria (entre 0 y 4095):");
      scanf("%s",&val[0]);
      dir=atoi(&val[0]);
      if (num(&val[0])<0) dir=-1;
     }
```

```
 clrscr();
 dir=div(dir,64)*64;
 gotoxy(18,2);
 highvideo();
 cprintf("P=Página previa  S=Siguiente página  F=Fin");
 gotoxy(9,5); putch('┌');
 gotoxy(10,5);
 cprintf("──────────────────────────────────────────────────────────
 ──────");
 gotoxy(9,21); putch('└');
 gotoxy(70,5); putch('┐');
 gotoxy(70,21); putch('┘');
 gotoxy(10,21);
 cprintf("──────────────────────────────────────────────────────────
 ──────");
 for(i=6;i<21;i++)
   {
    gotoxy(9,i);putch('│');
    gotoxy(70,i);putch('│');
   }
 normvideo();
 window(11,7,70,21);

 /* El primer paso consiste en programar el 8255 en mode 0,
   con los puertos A y B como entrada y el C como salida       */

   outportb(787,146);

 pag(&dir);
 b=' ';
 while(b!='F')
      {
        b=bioskey(0);
        strupr(&b);
        switch(b)
            {
             case 'P': if (dir>64) dir-=128;
                       else dir-=64;
                       pag(&dir); break;
             case 'S': if (dir>=4096) dir-=64; pag(&dir); break;
            }
      }
}

void cmf(void)
{
 int dir,dirf,a,i,x[2],*y;
 unsigned char b;
 char val[20];
 FILE *pf;
```

```
 gotoxy(13,1);
 cprintf("═══════════════════════════════════════════════════════
 ═══════");
 window(2,2,78,23);
 clrscr();
 gotoxy(19,9);cprintf("Inicio de bloque (entre 0 y 4095):");
 scanf("%s",&val[0]);
 dir=atoi(&val[0]);
 if (num(&val[0])<0) dir=-1;
 while ((dir<0)||(dir>4095))
      {
       clrscr();
       gotoxy(19,9);
       cprintf("Inicio de bloque (entre 0 y 4095):");
       scanf("%s",&val[0]);
       dir=atoi(&val[0]);
       if (num(&val[0])<0) dir=-1;
      }
 gotoxy(19,11);cprintf("Fin de bloque (entre %d y 4095):",dir);
 scanf("%s",&val[0]);
 dirf=atoi(&val[0]);
 if ((num(&val[0])<0)||(dirf<dir)) dirf=-1;
 while ((dirf<0)||(dirf>4095)||(dirf<dir))
      {
       gotoxy(19,11);
       delline();
       cprintf("Fin de bloque (entre %d y 4095):",dir);
       scanf("%s",&val[0]);
       dirf=atoi(&val[0]);
       if (num(&val[0])<0) dirf=-1;
      }
 gotoxy(19,13);cprintf("Nombre del fichero: ");
 scanf("%s",&val[0]);
 pf=fopen(&val[0],"w");
 y=&x[0];

  /* programar el 8255 en mode 0, con los puertos A y B
     como entrada y el C como salida                        */

   outportb(787,146);

 for(i=dir;i<=dirf;i++)
  {
   lee(i,y);
   fprintf(pf,"%c%d%c%d",'|',x[0],'|',x[1]);
  }
 fclose(pf);
}
```

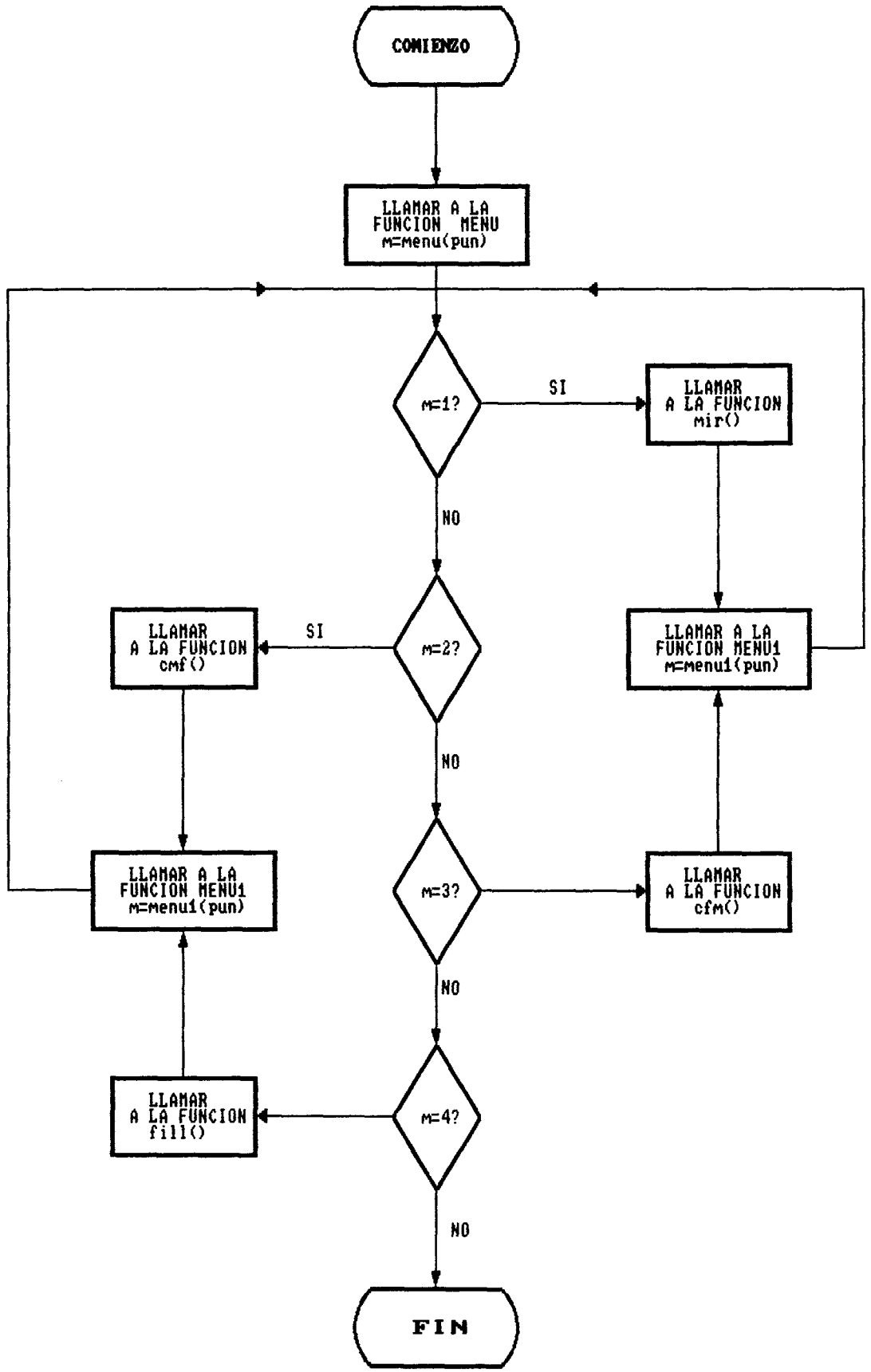
```
void cfm(void)
{
 int dir,dirf,a,i,x[2],*y;
 unsigned char b;
 char val[20];
 FILE *pf;
 gotoxy(13,1);
 cprintf("═══════════════════════════════════════════════════════════
 ═══");
 window(2,2,78,23); clrscr();
 gotoxy(19,9);cprintf("Inicio de bloque (entre 0 y 4095):");
 scanf("%s",&val[0]);
 dir=atoi(&val[0]);
 if (num(&val[0])<0) dir=-1;
 while ((dir<0)||(dir>4095))
     {
      clrscr(); gotoxy(19,9);
      cprintf("Inicio de bloque (entre 0 y 4095):");
      scanf("%s",&val[0]);
      dir=atoi(&val[0]);
      if (num(&val[0])<0) dir=-1;
     }
 gotoxy(19,11);cprintf("Nombre del fichero: ");
 scanf("%s",&val[0]);
 pf=fopen(&val[0],"r");

 if (pf==NULL)
   {
    gotoxy(25,14); highvideo();
    cprintf("Fichero no encontrado");
    normvideo(); bioskey(0);
    return;
   }

 y=&x[0];
 while((!feof(pf))&&(dir<4096))
  {
   fscanf(pf,"%c%d%c%d",&b,&x[0],&b,&x[1]);
   escribe(dir++,y);
  }
 fclose(pf);
}

void fill(void)
{
 int dir,dirf,a,c,i;
 unsigned char b;
 int dato[2],*data;
 char val[20];
 gotoxy(13,1);
 cprintf("═══════════════════════════════════════════════════════════
 ═══");
 window(2,2,78,23);
```

```
 clrscr();
 gotoxy(19,9);cprintf("Inicio de bloque (entre 0 y 4095):");
 scanf("%s",&val[0]);
 dir=atoi(&val[0]);
 if (num(&val[0])<0) dir=-1;
 while ((dir<0)||(dir>4095))
      {
       clrscr();
       gotoxy(19,9);
       cprintf("Inicio de bloque (entre 0 y 4095):");
       scanf("%s",&val[0]);
       dir=atoi(&val[0]);
       if (num(&val[0])<0) dir=-1;
      }
 gotoxy(19,11);cprintf("Fin de bloque (entre %d y 4095):",dir);
 scanf("%s",&val[0]);
 dirf=atoi(&val[0]);
 if ((num(&val[0])<0)||(dirf<dir)) dirf=-1;
 while ((dirf<0)||(dirf>4095)||(dirf<dir))
      {
       gotoxy(19,11);
       delline();
       cprintf("Fin de bloque (entre %d y 4095):",dir);
       scanf("%s",&val[0]);
       dirf=atoi(&val[0]);
       if (num(&val[0])<0) dirf=-1;
      }
 gotoxy(19,13);cprintf("Byte de menor peso: ");
 scanf("%s",&val[0]);
 a=atoi(&val[0]);
 gotoxy(19,15);cprintf("Byte de mayor peso: ");
 scanf("%s",&val[0]);
 c=atoi(&val[0]);
 dato[0]=a;dato[1]=c;
 data=&dato[0];

 for(i=dir;i<=dirf;i++) escribe(i,data);

}



main()
{
 unsigned char f[2],*g;
 char *pun;
 int m;
 pun=&m1[0];
 clrscr();
 m=menu(pun);
```

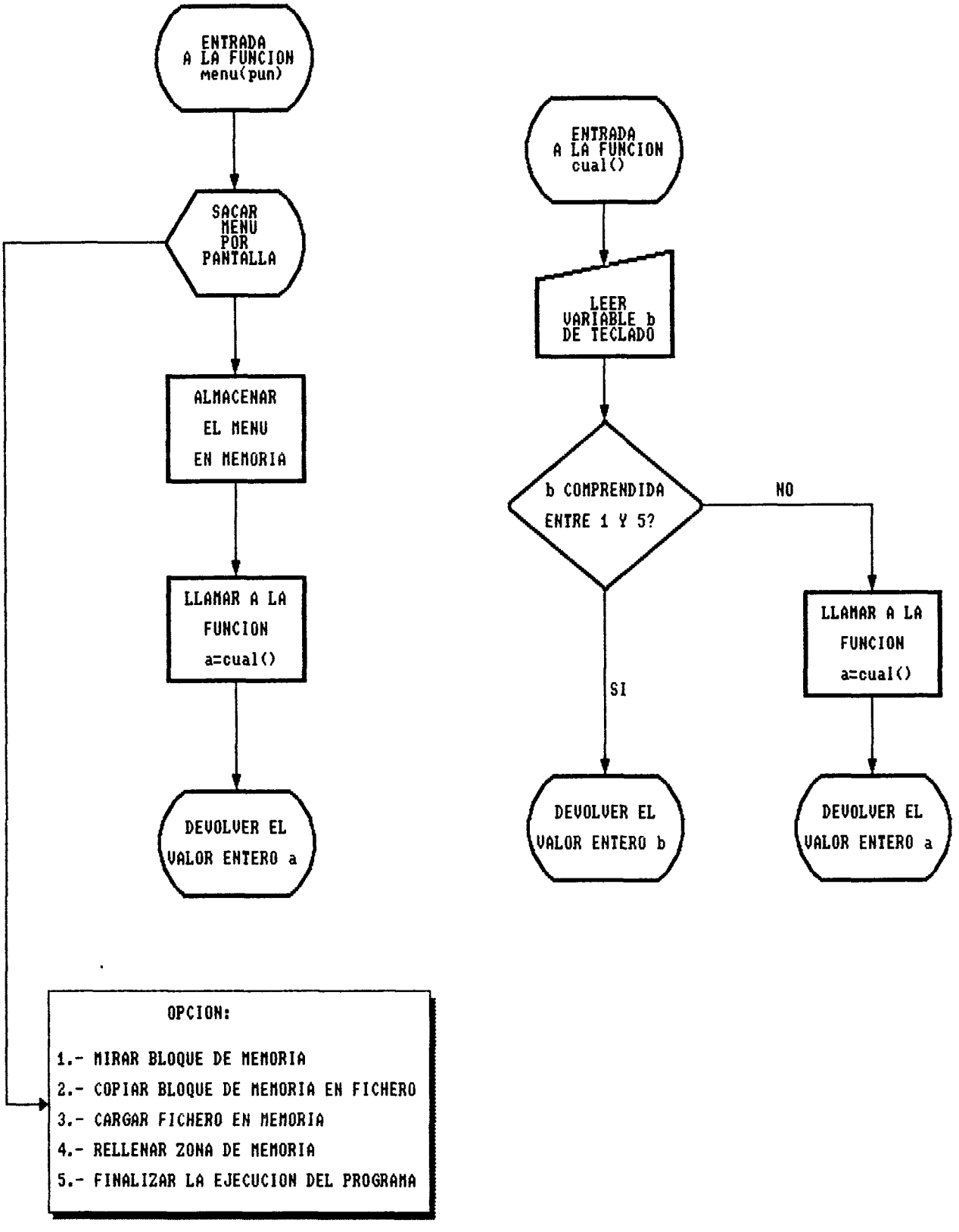
```
do
    {
       switch(m)
                {
                 case 1: mir();break;
                 case 2: cmf();break;
                 case 3: cfm();break;
                 case 4: fill();break;
                 case 5: clrscr();
                         outportb(787,146);
                         outportb(786,252);
                         exit();
                }
       m=menu1(pun);
    }
 while (10);
}
```

Con el fin de explicar el funcionamiento de este programa, vienen a continuación los diagramas de flujo correspondientes al mismo. Estos diagramas están hechos de forma concisa, y su objetivo es el de explicar y subrayar la estructura del programa:

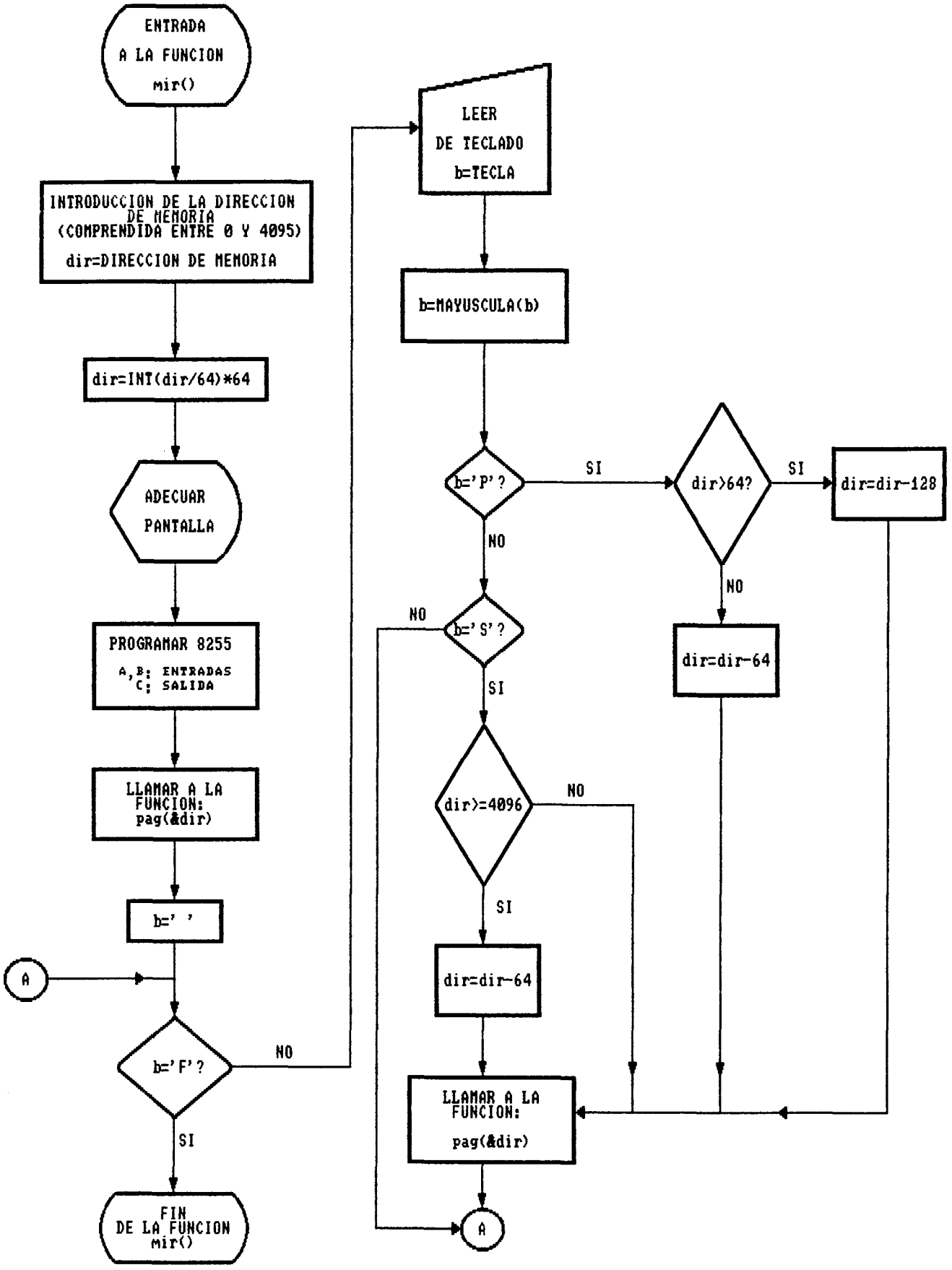
PROGRAMA MEM.C



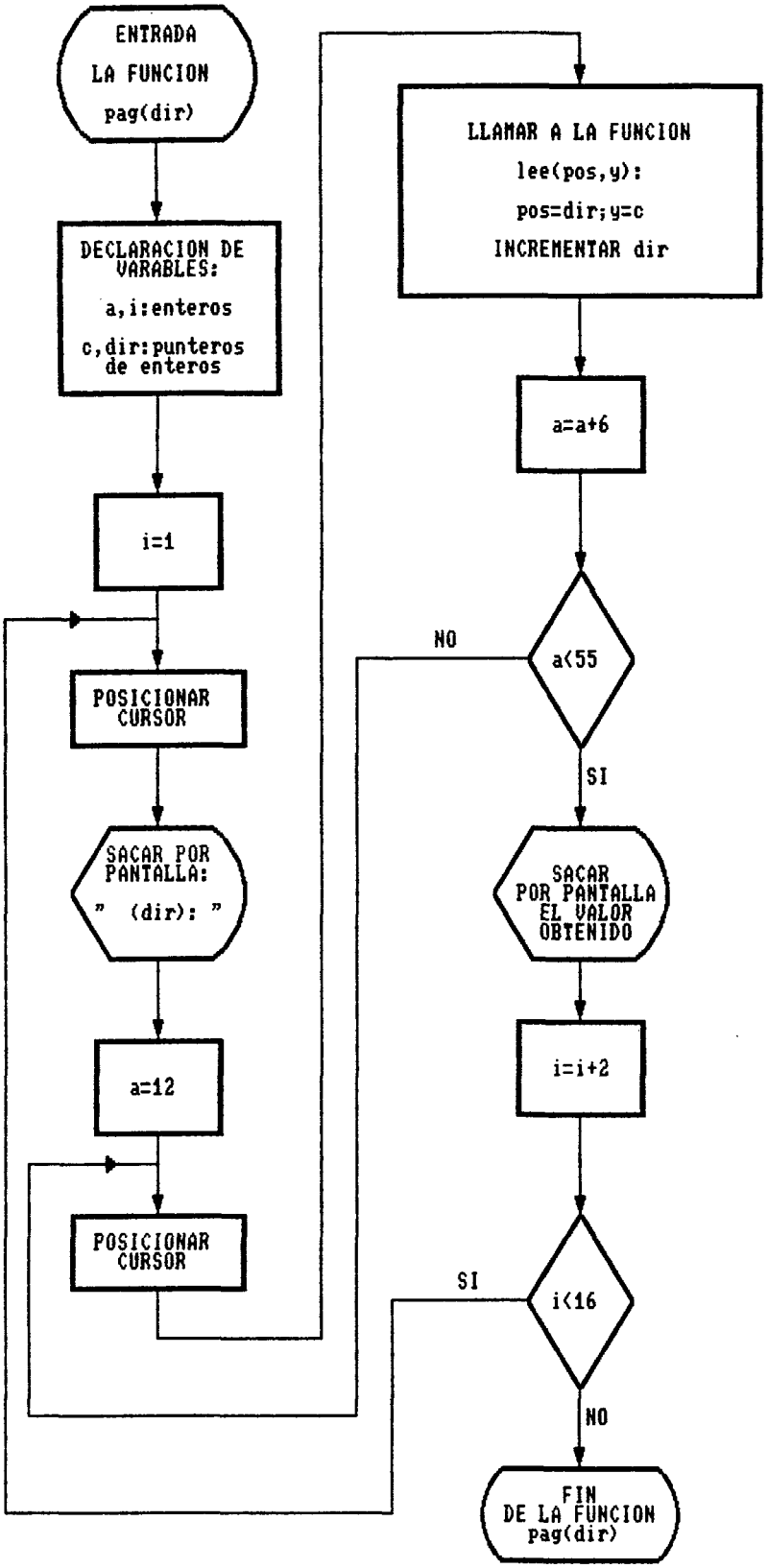
FUNCION menu(pun)



FUNCION mir()



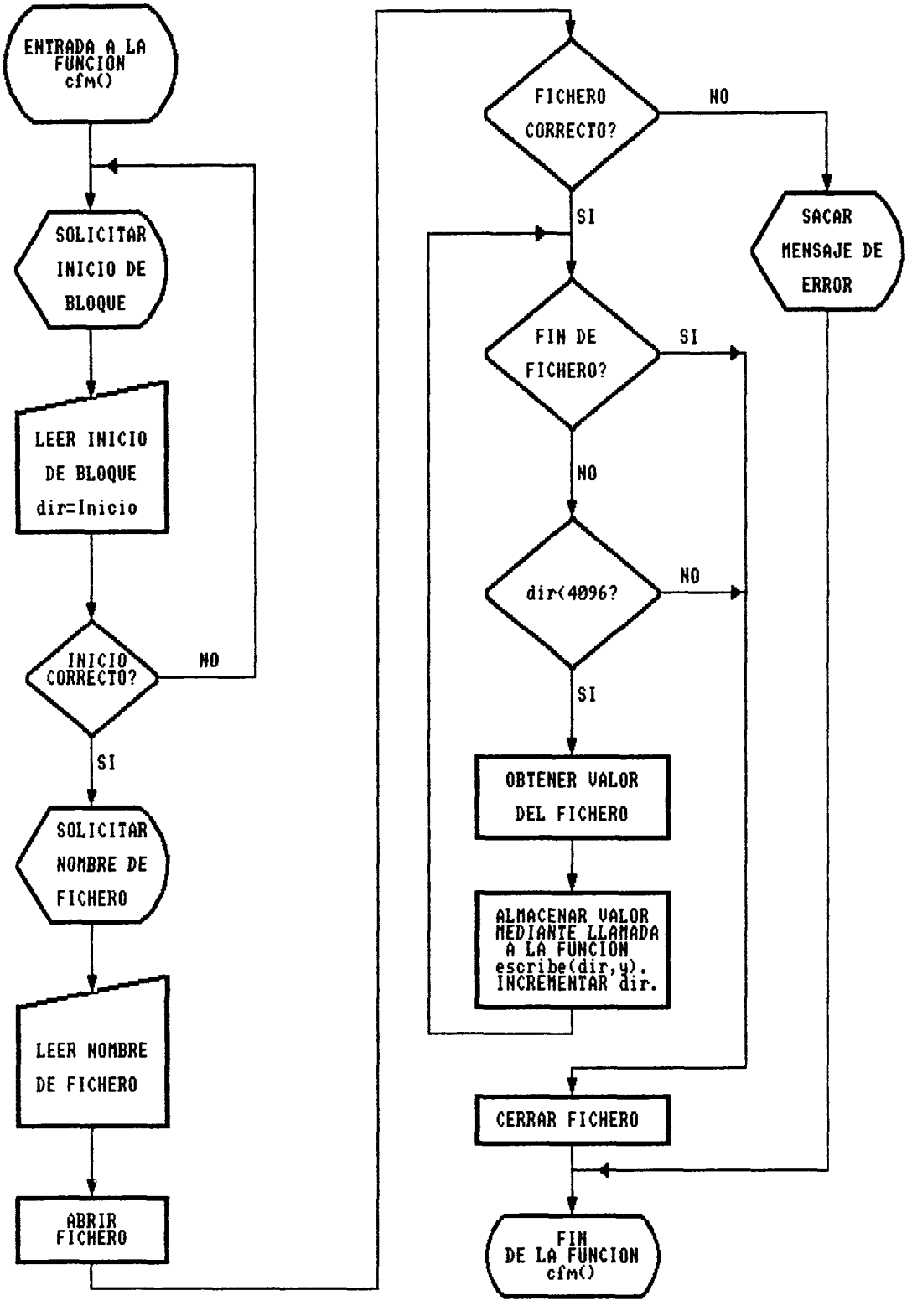
FUNCION pag(dir)



FUNCION menu1(pun)

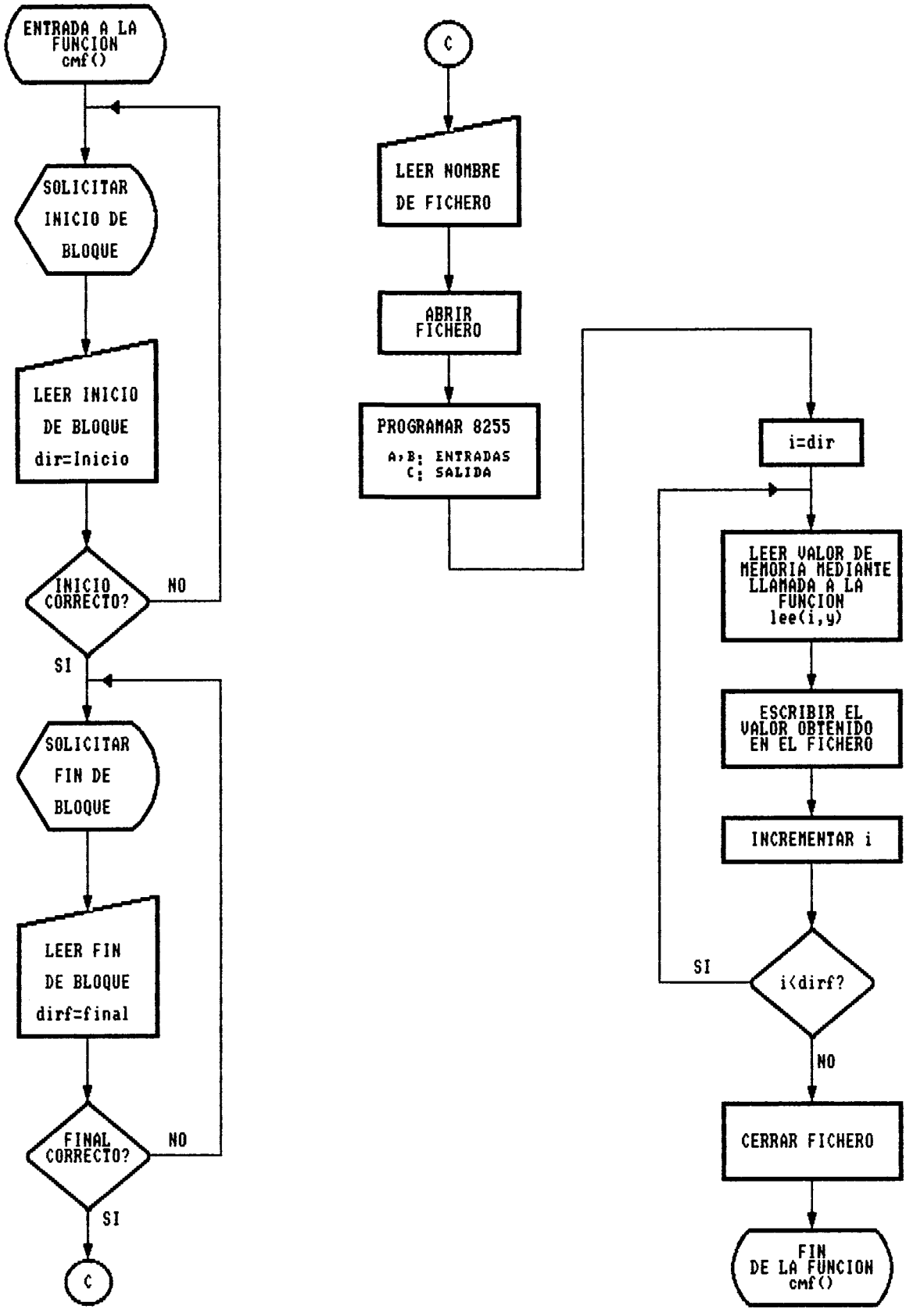
FUNCION cfm()

Descripcion: Esta funcion lee un fichero y lo almacena en la memoria del sistema.



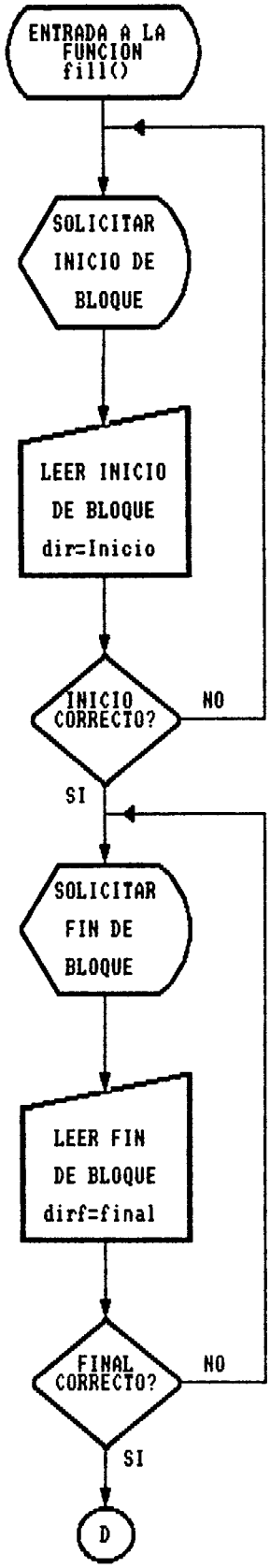
FUNCION cmf()

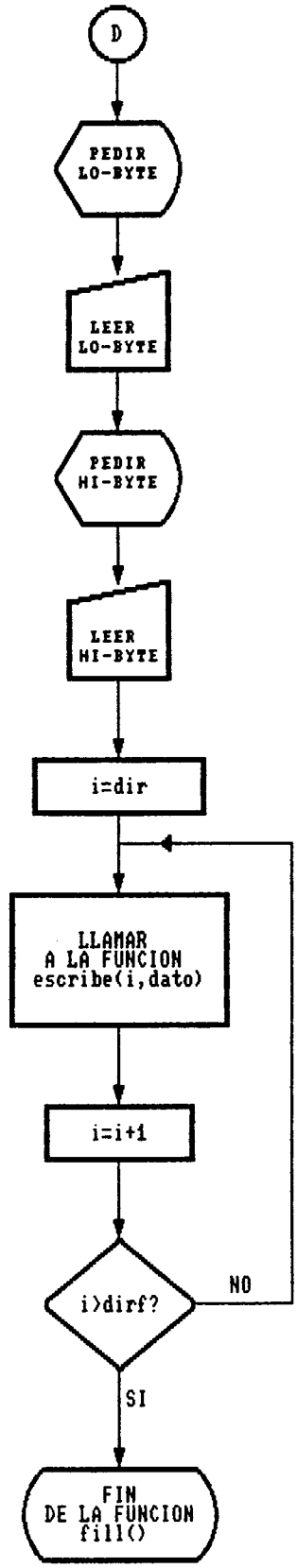
Descripcion: Esta funcion lee un bloque de memoria del sistema y lo almacena en un fichero.



FUNCION fill()

Descripcion: Esta funcion llena un bloque de memoria del sistema con un dato introducido por teclado.





Opción:

1.- Mirar bloque de memoria

2.- Copiar bloque de memoria en fichero

3.- Cargar fichero en memoria

4.- Rellenar zona de memoria

5.- Finalizar la ejecución del programa

P=Página previa S=Siguiente página F=Fin

0:	ffff	ffff	ffff	ffff	ffff	ffff	ffff	ffff
8:	ffff	ffff	ffff	ffff	ffff	ffff	ffff	ffff
16:	ffff	ffff	ffff	ffff	ffff	ffff	ffff	ffff
24:	ffff	ffff	ffff	ffff	ffff	ffff	ffff	ffff
32:	ffff	ffff	ffff	ffff	ffff	ffff	ffff	ffff
40:	ffff	ffff	ffff	ffff	ffff	ffff	ffff	ffff
48:	ffff	ffff	ffff	ffff	ffff	ffff	ffff	ffff
56:	ffff	ffff	ffff	ffff	ffff	ffff	ffff	ffff

SISTEMA DE ANALISIS DE ESPECTRO:

Con el propósito de demostrar la utilidad de nuestro sistema de proceso, se ha desarrollado una aplicación típica dentro de este tipo de sistemas. Esta aplicación consiste en analizar el espectro de potencia de las distintas bandas de frecuencia de una señal de entrada. Para ello se requiere una larga serie de sumas y multiplicaciones con números complejos que de ser realizadas directamente por el PC se tomarían un tiempo considerable.
En cambio, si el proceso lo realiza el TMS32010, el tiempo total de proceso de cada trama, estando compuesta cada trama de 512 muestras, es de varias decenas de milisegundos.

El proceso que se sigue, de una forma muy esquemática, es el siguiente:

En primer lugar, el ordenador carga el programa para el cálculo de la FFT (Transformada rápida de fourier) en la memoria del TMS32010.
Una vez cargado dicho programa, se entra en un ciclo en el que primero el TMS calcula el espectro de la señal y almacena el resultado en una zona determinada de la memoria, y a continuación el PC lee los valores almacenados en dicha zona y los representa gráficamente.
Cuando el TMS termina su ciclo de lectura y cálculo, entra en un bucle de espera del que sale después de que el ordenador haya leído los datos almacenados en la memoria, puesto que para ello

el PC sitúa al TMS en el estado de RESET, tras el cual el procesador comienza un nuevo ciclo al encontrarse con el registro de Program Counter a cero.

Pasaremos a continuación a describir más detalladamente tanto el programa en ensamblador del TMS32010, que como se ha dicho es el que analiza la señal, como el programa del PC, que es el que controla el funcionamiento global del sistema:

PROGRAMA EN ENSAMBLADOR:

Este programa se creó mediante un procesador de texto estándar para PC, en formato ASCII, y se compiló mediante el programa XASM3, que crea como resultado un fichero con la extensión .mpo que se puede cargar en la memoria del sistema a través del emulador XDS/22.

Para reconvertirlo al formato que usa el programa en C para cargarlo en la memoria del sistema se ha hecho uso del siguiente programa, que a partir del fichero con extensión .mpo crea el programa prog.fft:

PROGRAMA PROG.C:

```
#include<stdio.h>

 char *val[5];
 FILE *f,*g;

ini()
{
 char c;
 c=getc(f);
```

```
 while (c!='B')
        {
          if (!feof(f)) c=getc(f);
          else break;
        }
}

sig()
{
 int x;
 char c;
 for(x=0;x<5;x++) if (!feof(f)) c=getc(f);
}

int n(c)
char c;
{
 if (c<65) return(c-48);
 return(c-55);
}

cua()
{
 int a,b;
 char c,d;
 if (!feof(f)) c=getc(f);
 if (!feof(f)) d=getc(f);
 fprintf(g,"%c%d",'¦',(a=n(c)*16+n(d)));
 if (!feof(f)) c=getc(f);
 if (!feof(f)) d=getc(f);
 fprintf(g,"%c%d",'¦',(b=n(c)*16+n(d)));
}

main()
{
 char t;
 scanf("%s",val);
 f=fopen(val,"r");
  g=fopen("prog.fft","w");
 ini();
 cua();
 while(!feof(f))
 {
   while ((t=getc(f))=='B') cua();
   sig();
   if (feof(f)) break;
   ini();
   if (feof(f)) break;
   cua();
 }
   fcloseall();
}
```

Para verificar el correcto funcionamiento del programa que nos tiene que analizar el espectro de la señal, se ha empleado el programa sim.exe, que a partir del fichero .mpo obtiene el código del programa que se desea verificar y permite simular el funcionamiento de dicho programa.

Para efectuar la simulación, se obtienen los valores correspondientes a la supuesta señal de entrada a partir de un fichero creado a tal efecto, y se almacenan los valores de salida en un fichero de salida. A continuación se pasan a representar gráficamente dichos valores, comparando la gráfica con otra obtenida mediante el análisis directo del PC de la misma señal de entrada. Esto permitió verificar que el programa del TMS32010 funciona correctamente, al coincidir ambas gráficas obtenidas por diferentes caminos.

Algoritmo básico del programa:

La transformada rápida de fourier se obtiene mediante el algoritmo de Cooley-Tukey:

```
N=2^LN
NV2=N/2
NM1=N-1
J=1
Para I=1 hasta NM1 hacer
  Si I<J hacer
       T=F(J)
       F(J)=F(I)
       F(i)=T
  Fin Si
  K=NV2
  Mientras K<J hacer
       J=J-K
       K=K/2
  Fin Mientras
```

```
  J=J+K
Fin Para
Para L=1 hasta LN hacer
  LE=2^L
  LE1=LE/2
  U=(1.0,0.0)
  W=COS(PI/LE1)-j*SIN(PI/LE1)
  Para J=1 hasta LE1 hacer
       Para I=J hasta N paso LE hacer
            IP=I+LE1
            T=F(IP)*U
            F(IP)=F(I)-T
            F(I)=F(I)+T
       Fin Para
       U=U*W
  Fin Para
Fin Para
```

El vector F() es un vector de números complejos, que inicialmente almacena la trama de valores de entrada y que una vez realizado el proceso almacena el resultado. Para que los valores complejos obtenidos tengan realmente significado hay que obtener su módulo, que es lo que realmente nos da una expresión de la distribución de potencia de la señal.

En este algoritmo, el número de muestras a procesar a de ser resultado de la función:

Número de muestras=2^N

En nuestro caso calcularemos la transformada de 512 muestras.

De los 512 valores que en nuestro caso obtenemos como resultado de ejecutar este algoritmo, sólo tendrán significación los primeros 256, puesto que los 256 restantes son una copia simétrica de los primeros.

Cada uno de estos 256 valores nos representarán la potencia de una franja correspondiente a la 1/256 parte de la mitad de la frecuencia de muestreo.

El programa en ensamblador es el siguiente:

```
FFT          32010 FAMILY MACRO ASSEMBLER   PC2.1 84.107

 0001        0000  CERO    EQU     0
 0002        0001  ONE     EQU     1
 0003        0002  TWO     EQU     2
 0004        0003  T       EQU     3
 0005        0004  L       EQU     4
 0006        0005  LE      EQU     5
 0007        0006  LE1     EQU     6
 0008        0007  DATO    EQU     7
 0009        0008  CNT     EQU     8
 0010        0009  POS     EQU     9
 0011        000A  I       EQU     10
 0012        000B  J       EQU     11
 0013        000C  K       EQU     12
 0014        000D  X       EQU     13
 0015        000E  Y       EQU     14
 0016        000F  XRO     EQU     15    *XRO=DIRECCION TEMPORAL
 0017        0010  UR      EQU     16         POR DEFECTO
 0018        0011  UI      EQU     17
 0019        0012  WR      EQU     18
 0020        0013  WI      EQU     19
 0021        0014  VRI1    EQU     20
 0022        0015  VII1    EQU     21
 0023        0016  VRI2    EQU     22
 0024        0017  VII2    EQU     23
 0025        0018  IP      EQU     24
 0026        0019  TEMP    EQU     25
 0027        001A  N       EQU     26
 0028        7FFF  UNO     EQU     >7FFF
 0029              CONVER  $MACRO  W
 0030                      LAC     :W.S:
 0031                      SUB     ONE
 0032                      SACL    Y
 0033                      LAC     Y,1
 0034                      SACL    Y
 0035                      $END
 0036  0000
 0037  0000                MLIB    'a:'
 0038  0000
 0039                      IDT     'FFT'
 0040  0000                AORG    0
 0041  0000
 0042  0000  7F8B          SOVM
 0043  0001  7E00          LACK    0
 0044  0002  5000          SACL    CERO
 0045  0003  7E01          LACK    1          *
 0046  0004  5001          SACL    ONE        *
```

```
0047 0005 7E02          LACK    2          *
0048 0006 5002          SACL    TWO        *
0049 0007 7E04          LACK    >4         * INICIALIZAR VALORES
0050 0008 5009          SACL    POS        *
0051 0009 2809          LAC     POS,8      *
0052 000A 5009          SACL    POS        *
0053 000B 2802          LAC     TWO,8      *
0054 000C 501A          SACL    N
0055 000D 1001          SUB     ONE        *
0056 000E 5008          SACL    CNT        *
0057 000F
0058 000F 2A02          LAC     TWO,10
0059 0010 500D          SACL    X
0060 0011
0061 0011
0062 0011 F600  BENT    BIOZ    S          *    BENT=BUCLE DE
     0012 0015                                  ENTRADA DE DATO
0063 0013 F900          B       BENT
     0014 0011
0064 0015
0065 0015 4007  S       IN      DATO,PA0
0066 0016
0067 0016 2007          LAC     DATO       *        ELIMINO NIVEL
0068 0017 100D          SUB     X          *             DE
0069 0018 5007          SACL    DATO       *          CONTINUA.
0070 0019
0071 0019 2009          LAC     POS        *
0072 001A 7D07          TBLW    DATO       *
0073 001B 0001          ADD     ONE        *
0074 001C 7D00          TBLW    CERO       *
0075 001D 0001          ADD     ONE        *
0076 001E 5009          SACL    POS        *
0077 001F
0078 001F F600  W       BIOZ    W          *    ESPERAR SIGUIENTE
     0020 001F                                      DATO.
0079 0021
0080 0021 2008          LAC     CNT        *
0081 0022 FE00          BNZ     CB         *
     0023 0026
0082 0024 F900          B       SB         *
     0025 002A
0083 0026
0084 0026 1001  CB      SUB     ONE        *
0085 0027 5008          SACL    CNT        *    FIN DE BUCLE?
0086 0028 F900          B       BENT       *
     0029 0011
0087 002A
0088 002A 7F80  SB      NOP                *FIN DE LA ZONA DE
0089 002B                                       ENTRADA
0090            **********************************************
0091            *                                            *
```

```
0092 002B 7E04          LACK     >04                          *
0093 002C 5009          SACL     POS                          *
0094              *                                           *
0095 002D 7E01          LACK     01                           *
0096 002E 500A          SACL     I                            *
0097 002F 500B          SACL     J                            *
0098 0030
0099 0030 200A  U       LAC      I,0
0100 0031 100B          SUB      J,0                          *
0101 0032 FD00          BGEZ     B1
     0033 004D
0102 0034
0103                    CONVER   J
0001 0034 200B          LAC      J
0002 0035 1001          SUB      ONE
0003 0036 500E          SACL     Y
0004 0037 210E          LAC      Y,1
0005 0038 500E          SACL     Y
0104              *                                           *
0105 0039 2809          LAC      POS,8
0106 003A 000E          ADD      Y
0107 003B 6703          TBLR     T
0108 003C
0109                    CONVER   I                            *
0001 003C 200A          LAC      I
0002 003D 1001          SUB      ONE
0003 003E 500E          SACL     Y
0004 003F 210E          LAC      Y,1
0005 0040 500E          SACL     Y
0110 0041
0111 0041 2809          LAC      POS,8
0112 0042 000E          ADD      Y
0113 0043 670D          TBLR     X
0114 0044 7D03          TBLW     T
0115              *                                           *
0116                    CONVER   J
0001 0045 200B          LAC      J
0002 0046 1001          SUB      ONE
0003 0047 500E          SACL     Y
0004 0048 210E          LAC      Y,1
0005 0049 500E          SACL     Y
0117              *                                           *
0118 004A 2809          LAC      POS,8                    *
0119 004B 000E          ADD      Y
0120 004C 7D0D          TBLW     X
0121              *                                  *
0122 004D 2801  B1      LAC      ONE,8
0123 004E 500C          SACL     K                 *     ALGORITMO DE
0124 004F                                                  REORDENACION
0125 004F 200C  AQ      LAC      K,0                       *
0126 0050 100B          SUB      J,0
```

```
0127 0051 FD00          BGEZ    B2                          *
     0052 0059
0128 0053
0129 0053 7F88          ABS                                 *
0130 0054 500B          SACL    J
0131 0055
0132 0055 2F0C          LAC     K,15                        *
0133 0056 580C          SACH    K,0
0134 0057 F900          B       AQ
     0058 004F
0135 0059
0136 0059 200B  B2      LAC     J,0                         *
0137 005A 000C          ADD     K,0
0138 005B 500B          SACL    J
0139 005C 2802          LAC     TWO,8
0140 005D 1001          SUB     ONE                         *
0141 005E 100A          SUB     I,0
0142 005F FB00          BLEZ    V
     0060 0066
0143 0061
0144 0061 200A          LAC     I,0                         *
0145 0062 0001          ADD     ONE,0                       *
0146 0063 500A          SACL    I                           *

0147            *                                           *
0148 0064 F900          B       U                           *
     0065 0030
0149            ********************************************
0150 0066
0151 0066 7E01  V       LACK    1
0152 0067 5004          SACL    L       *HACER L=1 HASTA LN(=9);
0153 0068 7E09          LACK    9
0154 0069 5008          SACL    CNT
0155 006A
0156 006A 3804  PB1     LAR     AR0,L   *
0157 006B 6880          LARP    AR0     *
0158                    DEC     ,AR0    *     CALCULAR LE=2^L;
0001 006C 6880           LARP     AR0
0002 006D 6898           MAR      *-
0159 006E 7E01          LACK    1       *           Y
0160 006F 5006  PB2     SACL    LE1     *         LE1=LE/2
0161 0070 2106          LAC     LE1,1   *
0162 0071 F400          BANZ    PB2     *
     0072 006F
0163 0073 5005          SACL    LE      *
0164 0074
0165                    LCAC    UNO     *   HACER:  (U=(1.0,0)
0001 0074 F800           CALL    LDAC$       CARGAR AC CON
     0075 0128
0002 0076 7FFF           DATA    UNO                 UNO
0166 0077 5010          SACL    UR      *     UR=1  (7FFFH)
```

```
0167 0078 7F89          ZAC                 *      UI=0  (0000H)
0168 0079 5011          SACL    UI          *
0169 007A
0170 007A 7EEB          LACK    COS
0171                    DEC
0001 007B 1001           SUB       ONE,0
0172 007C 0004          ADD     L
0173 007D 6712          TBLR    WR
0174 007E
0175 007E 7EF4          LACK    SEN
0176                    DEC
0001 007F 1001           SUB       ONE,0
0177 0080 0004          ADD     L
0178 0081 6713          TBLR    WI
0179 0082
0180              *     PRINCIPIO DEL BUCLE J=1 HASTA LE1
0181 0082
0182 0082 2001          LAC     ONE
0183 0083 500B          SACL    J
0184 0084
0185              *     PRINCIPIO DEL BUCLE I=J HASTA N PASO LE
0186 0084
0187 0084 200B          LAC     J
0188 0085 500A          SACL    I
0189 0086
0190 0086 200A  PBIJN   LAC     I
0191 0087 0006          ADD     LE1
0192 0088 5018          SACL    IP
0193 0089
0194                    CONVER  I
0001 0089 200A          LAC     I
0002 008A 1001          SUB     ONE
0003 008B 500E          SACL    Y
0004 008C 210E          LAC     Y,1
0005 008D 500E          SACL    Y
0195 008E 2809          LAC      POS,8      *    CARGO  VR1 CON F(I)
0196 008F 000E          ADD      Y          *
0197 0090 6714          TBLR     VRI1       *    VRI1<-PARTE REAL
0198 0091
0199 0091 0001          ADD      ONE        *
0200 0092 6715          TBLR     VII1       *  VII1<-PARTE IMAGINARIA
0201 0093
0202                    CONVER  IP
0001 0093 2018          LAC     IP
0002 0094 1001          SUB     ONE
0003 0095 500E          SACL    Y
0004 0096 210E          LAC     Y,1
0005 0097 500E          SACL    Y
0203 0098 2809          LAC      POS,8      *   CARGO  VR1 CON F(IP)
0204 0099 000E          ADD      Y          *
```

```
0205 009A 6716          TBLR    VRI2      *       VRI1<-PARTE REAL
0206 009B
0207 009B 0001          ADD     ONE       *
0208 009C 6717          TBLR    VII2      *       VII1<-PARTE
0209 009D                                                IMAGINARIA
0210 009D
0211 009D
0212                    FFT     VRI1,VII1,VRI2,VII2,UR,UI *LLAMAR
                                                        *A LA  MACRO
0001 009D 6A16           LT        VRI2
0002 009E 6D10           MPY       UR
0003 009F 7F8E           PAC
0004 00A0 6A17           LT        VII2
0005 00A1 6D11           MPY       UI
0006 00A2 7F8F           APAC
0007 00A3 5919           SACH      TEMP,1
0008 00A4 6D10           MPY       UR
0009 00A5 7F8E           PAC
0010 00A6 6A16           LT        VRI2
0011 00A7 6D11           MPY       UI
0012 00A8 7F90           SPAC
0013 00A9 5917           SACH      VII2,1
0014 00AA 2F14           LAC       VRI1,15
0015 00AB 1F19           SUB       TEMP,15
0016 00AC 5816           SACH      VRI2
0017 00AD
0018 00AD 6019           ADDH      TEMP
0019 00AE 5814           SACH      VRI1
0020 00AF
0021 00AF 2F15           LAC       VII1,15
0022 00B0 0F17           ADD       VII2,15
0023 00B1 5815           SACH      VII1
0024 00B2
0025 00B2 6217           SUBH      VII2
0026 00B3 5817           SACH      VII2
0213 00B4
0214 00B4
0215                    CONVER  I

0001 00B4 200A          LAC     I
0002 00B5 1001          SUB     ONE
0003 00B6 500E          SACL    Y
0004 00B7 210E          LAC     Y,1
0005 00B8 500E          SACL    Y
0216 00B9 2809          LAC      POS,8     *   CARGO  F(I) CON VR1
0217 00BA 000E          ADD      Y         *
0218 00BB 7D14          TBLW     VRI1      *   VRI1->PARTE REAL
0219 00BC
0220 00BC 0001          ADD      ONE       *
0221 00BD 7D15          TBLW     VII1      *  VII1->PARTE IMAGINARIA
0222 00BE
```

```
0223                        CONVER  IP
0001  00BE  2018           LAC     IP
0002  00BF  1001           SUB     ONE
0003  00C0  500E           SACL    Y
0004  00C1  210E           LAC     Y,1
0005  00C2  500E           SACL    Y
0224  00C3  2809           LAC      POS,8     *  CARGO  F(IP) CON VR1
0225  00C4  000E           ADD      Y         *
0226  00C5  7D16           TBLW     VRI2      *        VRI1->PARTE REAL
0227  00C6
0228  00C6  0001           ADD      ONE       *
0229  00C7  7D17           TBLW     VII2      *        VII1->PARTE
0230  00C8                                                IMAGINARIA
0231  00C8
0232  00C8  200A           LAC      I
0233  00C9  0005           ADD      LE
0234  00CA  101A           SUB      N
0235  00CB  FC00           BGZ     SI1     * FIN DE LAZO
      00CC  00D2
0236  00CD
0237  00CD  200A           LAC      I        *    SI I>=N ENTONCES
0238  00CE  0005           ADD      LE       *        FIN DE LAZO
0239  00CF  500A           SACL     I        *        SI NO HACER I=I+LE
0240  00D0  F900           B        PBIJN    *      Y  HACER OTRA PASADA
      00D1  0086
0241  00D2
0242  00D2
0243  00D2
0244  00D2  6A10  SI1      LT       UR       *          HACER U=U*W
0245  00D3  6D12           MPY      WR       *
0246  00D4  7F8E           PAC               *
0247  00D5  6A11           LT       UI       *         TEMP<-UR*WR-UI*WI
0248  00D6  6D13           MPY      WI       *
0249  00D7  7F90           SPAC              *
0250  00D8  5919           SACH     TEMP,1   *
0251  00D9
0252  00D9  6D12           MPY      WR       *
0253  00DA  7F8E           PAC               *
0254  00DB  6A10           LT       UR       *          UI<- UI*WR+UR*WI
0255  00DC  6D13           MPY      WI       *
0256  00DD  7F8F           APAC              *
0257  00DE  5911           SACH     UI,1     *
0258  00DF
0259  00DF  2019           LAC      TEMP     *          UR<-TEMP
0260  00E0  5010           SACL     UR       *

0261  00E1
0262  00E1
0263  00E1  200B           LAC      J        *
0264  00E2  1006           SUB      LE1      *  COMPROBAR FIN DE BUCLE
```

```
0265 00E3 FD00          BGEZ    SI2     *
     00E4 00FD
0266 00E5
0267                     INC      J       *  SI NO  FIN  ENTONCES
0001 00E5 200B           LAC      J,0     *  INICIALIZAR BUCLE
0002 00E6 0001           ADD      ONE,0
0003 00E7 500B           SACL     J,0
0268 00E8 500A          SACL    I       *  I=J HASTA N PASO LE
0269 00E9 F900          B       PBIJN   *  Y SEGUIR
     00EA 0086
0270 00EB
0271 00EB
0272 00EB FFFF  COS     DATA    >FFFF,>0,>5A82,>7641,>7D8A
     00EC 0000
     00ED 5A82
     00EE 7641
     00EF 7D8A
0273 00F0 7F62          DATA    >7F62,>7FD8,>7FF6,>7FFD
     00F1 7FD8
     00F2 7FF6
     00F3 7FFD
0274 00F4 0000  SEN     DATA    >0,>7FFF,>5A82,>30FB,>18F8
     00F5 7FFF
     00F6 5A82
     00F7 30FB
     00F8 18F8
0275 00F9 0C8B          DATA    >0C8B,>0647,>0324,>0192
     00FA 0647
     00FB 0324
     00FC 0192
0276 00FD
0277 00FD
0278 00FD
0279            SI2     INC     L       *L=L+1    *   INCREMENTAR L,
0001 00FD 2004           LAC      L,0             *   Y COMPARAR
0002 00FE 0001           ADD      ONE,0           *   SI ES MENOR
0003 00FF 5004           SACL     L,0             *   O IGUAL
0280 0100 1008          SUB     CNT             *   QUE NUEVE
0281 0101 FB00          BLEZ    PB1             *
     0102 006A
0282 0103
0283 0103
0284 0103
0286 0103
0287 0103 7E01          LACK 1
0288 0104 500A          SACL I
0289 0105
0290 0105 2802          LAC  TWO,8
0291 0106 1001          SUB  ONE
```

```
0292 0107 5008          SACL CNT
0293 0108
0294              PRCLA  CONVER I
0001 0108 200A          LAC    I
0002 0109 1001          SUB    ONE
0003 010A 500E          SACL   Y
0004 010B 210E          LAC    Y,1
0005 010C 500E          SACL   Y
0295 010D 2809          LAC  POS,8
0296 010E 000E          ADD  Y
0297 010F 6714          TBLR VRI1
0298 0110
0299 0110 0001          ADD  ONE
0300 0111 6716          TBLR VRI2
0301 0112
0302 0112 6A14          LT   VRI1
0303 0113 6D14          MPY  VRI1
0304 0114 7F8E          PAC
0305 0115
0306 0115 6A16          LT   VRI2
0307 0116 6D16          MPY  VRI2
0308 0117 7F8F          APAC
0309 0118
0310 0118 5015          SACL VII1
0311 0119
0312 0119 200A          LAC  I
0313 011A 1001          SUB  ONE
0314 011B 500E          SACL Y
0315 011C 2809          LAC  POS,8
0316 011D 000E          ADD  Y
0317 011E 7D15          TBLW VII1
0318 011F
0319                    INC  I
0001 011F 200A           LAC        I,0
0002 0120 0001           ADD        ONE,0
0003 0121 500A           SACL       I,0
0320 0122 200A          LAC  I
0321 0123 1008          SUB  CNT
0322 0124 FA00          BLZ  PRCLA
     0125 0108
0323 0126
0324 0126 F900  LAZOF   B        LAZOF
     0127 0126
0325 0128
0326 0128
0327 0128 7F9D  LDAC$    POP
0328 0129 670F           TBLR   XRO
0329 012A 0001           ADD    ONE
0330 012B 7F9C           PUSH
0331 012C 200F           LAC    XRO
0332 012D 7F8D           RET
0333                   END
NO ERRORS, NO WARNINGS
```

Básicamente, la estructura general de este programa es la siguiente:

El programa comienza con una zona de iniziación de valores.

Pasamos a continuación al bucle de entrada de valores, donde leemos los valores obtenidos por el convertidor, y los adecuamos a la aritmética con la que vamos a operar mediante la resta del valor 2^11 (dos elevado a once). La sincronización con el convertidor se controla mediante al valor de la patilla BIO:

Primero esperamos a que la patilla BIO pase a nivel bajo. Cuando esto ocurre efectuamos la lectura del valor y se pasa a un bucle de espera que aguardará a que la patilla BIO regrese al nivel alto. Por último comprobamos si ya se han leído todos los valores, y caso de que no sea así volvemos al bucle de espera inicial.

Una vez obtenidos los valores se pasa a ejecutar la implementación del algoritmo de Cooley-Tukey antes descrito, que comienza con un bucle de reordenación previa de las muestras para pasar a continuación a efectuar las operaciones complejas correspondientes. Como trabajamos con números complejos, cada valor ocupará dos posiciones contiguas de memoria, correspondiendo la primera a la parte real y la segunda a la parte imaginaria. En la fase de lectura de valores, estos se colocan en el campo real, inicializando la parte imaginaria a cero.

Una vez obtenida la transformada rápida de estos valores, lo que se hace es elevar tanto la parte real como la imaginaria de los números complejos obtenidos y almacenar la suma de ambas en las primeras 256 posiciones consecutivas de memoria, siendo este el resultado final de todo el proceso.

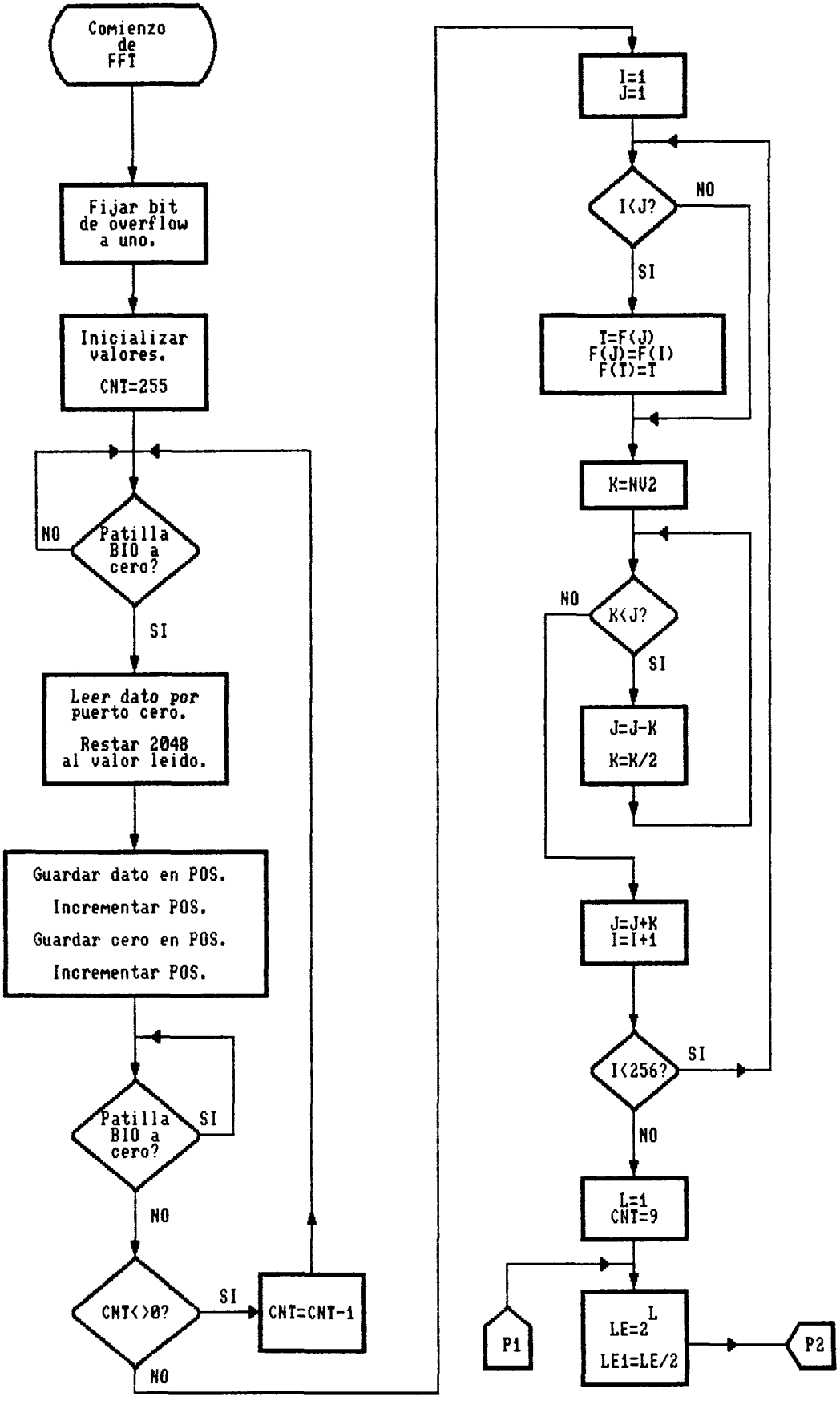
Por último el programa entra en un bucle de espera del que sólo saldrá después de un RESET, tras el cual vualve a comenzar su ejecución desde el principio.

A la hora de realizar este programa se hizo uso de algunas macros, las cuales aparecen expandidas en el listado arriba presentado. La macro más importante es la que efectúa la mariposa en la transfomada rápida de fourier, que es la que viene indicada como "FFT VRI1,VII1,VRI2,VII2,UR,UI ".

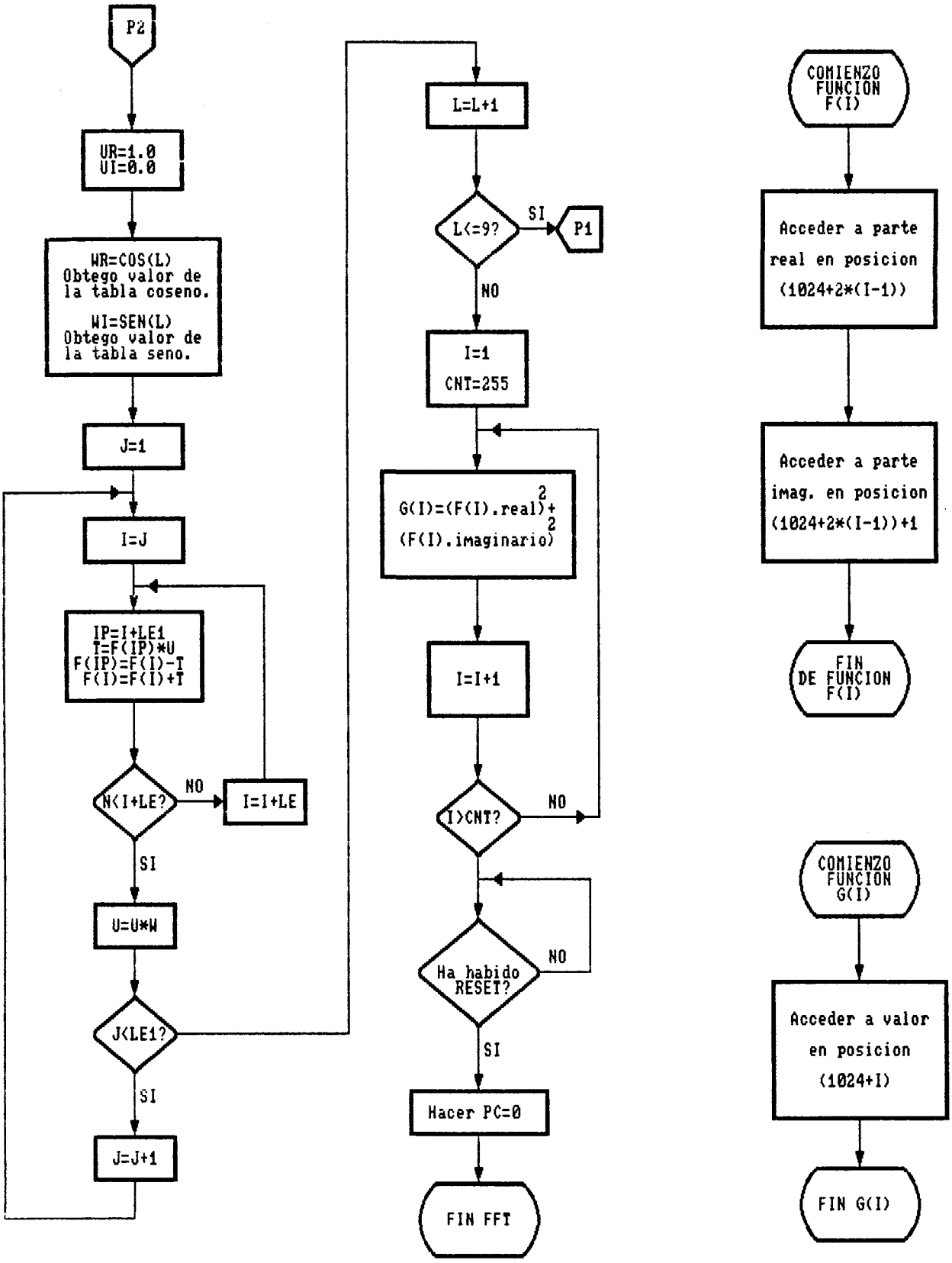
Para obtener los valores de las funciones seno y coseno empleados se crearon dos tablas de datos. Los valores almacenados se representan en formato Q15, que consiste en representar los valores con 15 dígitos fraccionarios y uno de signo. En este formato el máximo valor representable es un valor ligeramente inferior a la unidad.

Para idicar más claramente el funcionamiento del programa, ponemos a continuación su diagrama de flujo:

ALGORITMO PARA CALCULO DE FFT (1)



ALGORITMO PARA CALCULO DE FFT (2)



PROGRAMA DE CONTROL DEL PROCESO DESDE EL PC (REPFFT.C):

El control del proceso de carga del programa del TMS32010 EN la memoria de nuestro sistema, y de la toma y representación gráfica de los resultados obtenidos se realiza mediante el programa REPFFT, realizado en C como el resto de las aplicaciones desarrolladas para el PC.

Este programa, carga las instrucciones del programa en esamblador que nos calcula el espectro de señal en la memoria externa del TMS, obteniendo el código de dicho programa del fichero prog.fft. Una vez inicializado el sistema pasa a representar gráficamente los resultados obtenidos, con lo cual va apareciendo en pantalla periódicamente el espectro de la señal a analizar.
Para evitar que se vea el proceso de dibujo en pantalla de la grágica, lo que se hace es trabajar con dos páginas gráficas (numeradas como cero y uno), pudiédose dibujar sobre una de ellas mientras que en pantalla aparece la otra página.

Salvo que se trabaje con un PC muy rápido, el proceso de representación gráfica de los resultados obtenidos es más lento que el proceso de cálculo que realiza el TMS32010 sobre la señal de entrada.
En un PC/XT a 10MHz la velocidad de paginación del programa es de aproximadamente una por segundo, mientras que el límite impuesto por el tiempo de cálculo del TMS es de aproximadamente 10 por segundo.

Veamos ahora el listado del programa:

PROGRAMA REPFFT.C:

```
#include <graphics.h>
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
#include<bios.h>

typedef struct { int re,im;
                } complejo;
double pi=3.14159265358979;

/* LA SIGUIENTE FUNCION LEE LA POSICION DE MEMORIA
   DEL SISTEMA INDICADA POR POS */

void lee(pos,y)
int pos,*y;
{
   unsigned char c1,c2,*x;


/* Se programa el puerto C de la siguiente
   forma:                                                    */

   outportb(786,20);     /* C=00010100 BIN

                            READY e INT A 1 */

/* Master Clear para el 8237 */

   outportb(781,0);


/* Programación del registro de comando   */

   outportb(776,0);

/* Pragramación del canal cero para:

        SINGLE MODE
        ADDRESS INCREMENT
        AUTOINITIALIZATION DISABLE
        READ TRANSFER                              */

   outportb(779,72);  /* Canal 0 */
   outportb(779,73);  /* Canal 1 */
```

```
   outportb(779,74);  /* Canal 2 */
   outportb(779,75);  /* Canal 3 */

/* Eliminar mask bit para canal cero */

   outportb(778,0);

/* Programar BASE y CURRENT ADDRESS del canal cero */

   outportb(780,0);    /* Clear BP Flip-Flop */

/* programar current address en función de pos,
   para ello hay que obtener el byte menos significativo,
   y el más significativo en función de el valor de pos y
   compensando el error de direccionamiento hardware*/

   x=&pos;
   c1=*x;
   c2=(*(x+1))*16;
   outportb(768,c1);
   outportb(768,c2);

/* Programar Base y Current Word Count del canal cero */

   outportb(780,0);   /* Clear BP Flip-Flop */

   outportb(769,10);  /* Word count=1 */
   outportb(769,0);

/* Poner bit de DRQ0 a 1 */

   outportb(786,21);  /* Canal C=00010101
                         DRQ0, INT y READY a 1 */

/* Poner HLDA a 1 para autorizar la transferencia y
   READY a 0 para prolongar el ciclo de lectura    */

   outportb(786,19);  /* Canal C=00010011 */

/* LEER LOS VALORES DE MEMORIA */

   *y=inportb(784);
   *(y+1)=inportb(785);
   outportb(786,20);  /* Canal C=00010100 */
}

void escribe(pos,val)
int pos,*val;
{
   unsigned char *x,c1,c2;
```

```
/* El primer paso consiste en programar el 8255 en mode 0,
   con los puertos A,B y C como salida       */

   outportb(787,128);

/* Se programa el puerto C de la siguiente
   forma:                                                  */

   outportb(786,20);     /* C=00010100 BIN

                            READY e INT A 1 */

/* Sacar por los puertos A y B los valores a escribir  */

   outportb(784,*val);
   outportb(785,*(val+1));

/* Master Clear para el 8237 */

   outportb(781,0);

/* Pragramación de los cuatro canales para:

        SINGLE MODE
        ADDRESS INCREMENT
        AUTOINITIALIZATION DISABLE
        WRITE TRANSFER                          */

   outportb(779,68);  /* Canal 0 */
   outportb(779,69);  /* Canal 1 */
   outportb(779,70);  /* Canal 2 */
   outportb(779,71);  /* Canal 3 */

/* Eliminar mask bit para canal cero */

   outportb(778,0);

/* Programar BASE y CURRENT ADDRESS del canal cero */

   outportb(780,0);     /* Clear BP Flip-Flop */

/* programar current address en función de pos,
   para ello hay que obtener el byte menos significativo,
   y el más significativo en función de el valor de pos y
   compensando el error de direccionamiento hardware*/

   x=&pos;
   c1=*x;
   c2=(*(x+1))*16;

   outportb(768,c1);
   outportb(768,c2);
```

```
/* Programar Base y Current Word Count del canal cero */

   outportb(778,0);    /* Clear BP Flip-Flop */

   outportb(769,1);   /* Word count=1 */
   outportb(769,0);

/* Poner bit de DRQ0 a 1 */

   outportb(786,21);   /* Canal C=00010101
                          DRQ0, INT y READY a 1 */

/* Poner HLDA a 1 para autorizar la transferencia  */

   outportb(786,23);   /* Canal C=00010011 */

/* Poner DRQ0 y HLDA a CERO, poner READY a uno */

   outportb(786,252);
}

unsigned int fun(dir)
int dir;
{
 int b[2],*c;
 c=&b[0];
 lee(dir,c);
 return(b[0]*256+b[1]);
}

void set(x)
int x;
{
 if (x==0)
  {
   setactivepage(0);
   setvisualpage(1);
  }
 else
  {
   setactivepage(1);
   setvisualpage(0);
  }
}

void rep(void)
{
 int i,j,k,ln,n,dir;
 int t;
 long int ang,alg[1025];
 complejo tc;
```

```
 ln=9;
 n=pow(2,ln);
 setviewport(102,120,618,250,1);
 clearviewport();
    ang=0;
    for(i=1;i<=(n/2);i++)
     {
      alg[i]=fun(i+1023);
      if (alg[i]>ang) ang=alg[i];
     }

  /* Poner RESET a UNO  */

    outportb(786,28);

    tc.re=0;tc.im=103;
    if (ang==0) ang=1;;
    for(i=1;i<=(n/2+1);i++)
      {
        j=(alg[i]*100)/ang; k=i*2;
        putpixel(k,(103-j),1);line(tc.re,tc.im,k,(103-j));
        tc.re=k;tc.im=103-j;
      }
      line(tc.re,tc.im,tc.re,103);

}

void dibuja(void)
{
 int var2=68;
 int var1=96;
 setlinestyle(SOLID_LINE,0,NORM_WIDTH);
 rectangle(2,2,getmaxx()-2,getmaxy()-2);
 rectangle(5,5,getmaxx()-5,getmaxy()-5);
 setlinestyle(CENTER_LINE,0,NORM_WIDTH);
 rectangle(3,3,getmaxx()-3,getmaxy()-3);
 setlinestyle(DASHED_LINE,0,NORM_WIDTH);
 rectangle(4,4,getmaxx()-4,getmaxy()-4);
 setlinestyle(SOLID_LINE,0,NORM_WIDTH);
 rectangle(var1,var2,getmaxx()-var1,getmaxy()-var2);
 rectangle(var1+4,var2+4,getmaxx()-var1-4,getmaxy()-var2-4);
 setlinestyle(DOTTED_LINE,0,NORM_WIDTH);
 rectangle(var1+2,var2+2,getmaxx()-var1-2,getmaxy()-var2-2);

 settextstyle(3,0,5);
 outtextxy(165,20,"Analisis de espectro:");
 settextstyle(1,0,1);
 outtextxy(150,298,"P: Pausa");
 outtextxy(315,298,"F: Fin");
 outtextxy(425,298,"R: Reinicializar");
 setlinestyle(SOLID_LINE,0,NORM_WIDTH);
```

```
 for (var1=120;var1<616;var1+=16) line(var1,274,var1,266);
 setlinestyle(SOLID_LINE,0,THICK_WIDTH);
 line(360,274,360,262);
}

void inicializa(void)
{
 FILE *prog;
 int *y,x[2];
 unsigned char b;
 int dir=0;
 prog=fopen("prog.fft","r");
 if (prog==NULL)
   {
    printf("Falta el fichero 'prog.fft' con el programa en
    ensamblador.");
    exit(0);
   }
 y=&x[0];
 while((!feof(prog))&&(dir<4096))
   {
    fscanf(prog,"%c%d%c%d",&b,&x[0],&b,&x[1]);
    escribe(dir++,y);
   }
  fclose(prog);

  /* Programar 8255 en MODE 0, A y B como entrada,
    y C como salida                                    */

  outportb(787,146);


  /* Programar el puerto C del 8255 con los bits de interrupción,

  RESET y READY a uno, y los demás, incluyendo HLDA y DRQ0,

  a cero */

   outportb(786,28);  /* Canal C=00011100 */


}

void borra(fondo)
void *fondo;
{
  setviewport(280,120,400,220,1);
  clearviewport();
  setviewport(102,120,618,250,1);
  putimage(100,0,fondo,OR_PUT);
}
```

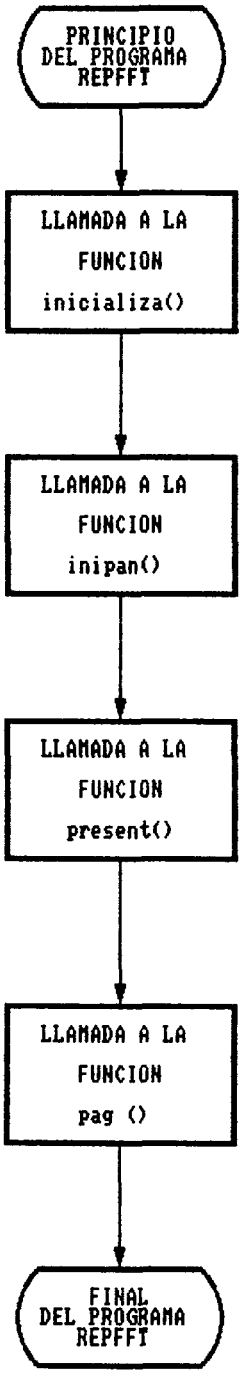
```
void fin(void)
{
 int ta,cox,coy,cord,nocual,i;
 void *dib,*fondo;
 cox=230;
 coy=180;
 ta=imagesize(cox,coy,cox+50,coy+30);
 dib=malloc(ta);
 getimage(cox,coy,cox+50,coy+30,dib);
 ta=imagesize(200,30,300,130);
 fondo=malloc(ta);
 getimage(100,0,400,150,fondo);
 putimage(cox,coy,dib,XOR_PUT);
 for(cord=coy;cord>50;cord--)
  {
   putimage(cox,cord,dib,XOR_PUT);
   delay(10);
   putimage(cox,cord,dib,XOR_PUT);
  }
 for (i=0;i<7;i++)
 {
  settextstyle(1,0,i+1);
  outtextxy(230-5*i,50-5*i,"Fin");
  borra(fondo);
 }
  settextstyle(1,0,i+1);
  outtextxy(230-5*i,50-5*i,"Fin");
  delay(3000);
}
void inipan(void)
{
 int gd,gm;
 detectgraph(&gd,&gm);
 initgraph(&gd,&gm,'');
}
void present()
{
 setvisualpage(0);setactivepage(0);
 cleardevice();
 setviewport(0,0,719,347,0);
 setlinestyle(SOLID_LINE,0,NORM_WIDTH);
 rectangle(2,2,getmaxx()-2,getmaxy()-2);
 rectangle(5,5,getmaxx()-5,getmaxy()-5);
 setlinestyle(CENTER_LINE,0,NORM_WIDTH);
 rectangle(3,3,getmaxx()-3,getmaxy()-3);
 setlinestyle(DASHED_LINE,0,NORM_WIDTH);
 rectangle(4,4,getmaxx()-4,getmaxy()-4);
 settextstyle(3,0,4);
 outtextxy(160,70,"ANALIZADOR DE ESPECTRO");
 outtextxy(330,130,"EN");
 outtextxy(255,190,"TIEMPO REAL");
```

```
 settextstyle(2,0,4);
 outtextxy(40,300,"Por: Roberto Morera Bello.");
 bioskey(0);
}

void pag(void)
{
 int cual=1;
 char c;
 cleardevice();
 set(1);
 dibuja();
 set(0);
 dibuja();
 setlinestyle(SOLID_LINE,0,NORM_WIDTH);
do
{
 set(cual);
 rep();
 cual=(cual==0)?1:0;
 while (bioskey(1)!=0)
  {
   c=getch();
   strupr(&c);
   switch(c)
    {
     case 'P': bioskey(0); break;
     case 'R':
     inicializa();present();pag(); break;
     case 'F':
     setvisualpage((cual==0)?1:0);fin();
     closegraph(); exit(0);
    }
  }
}
 while (bioskey(1)==0);
}

main()
{
 inicializa();
 inipan();
 present();
 pag();
}
```

Los diagramas de flujo correspondientes a este programa son los siguientes:

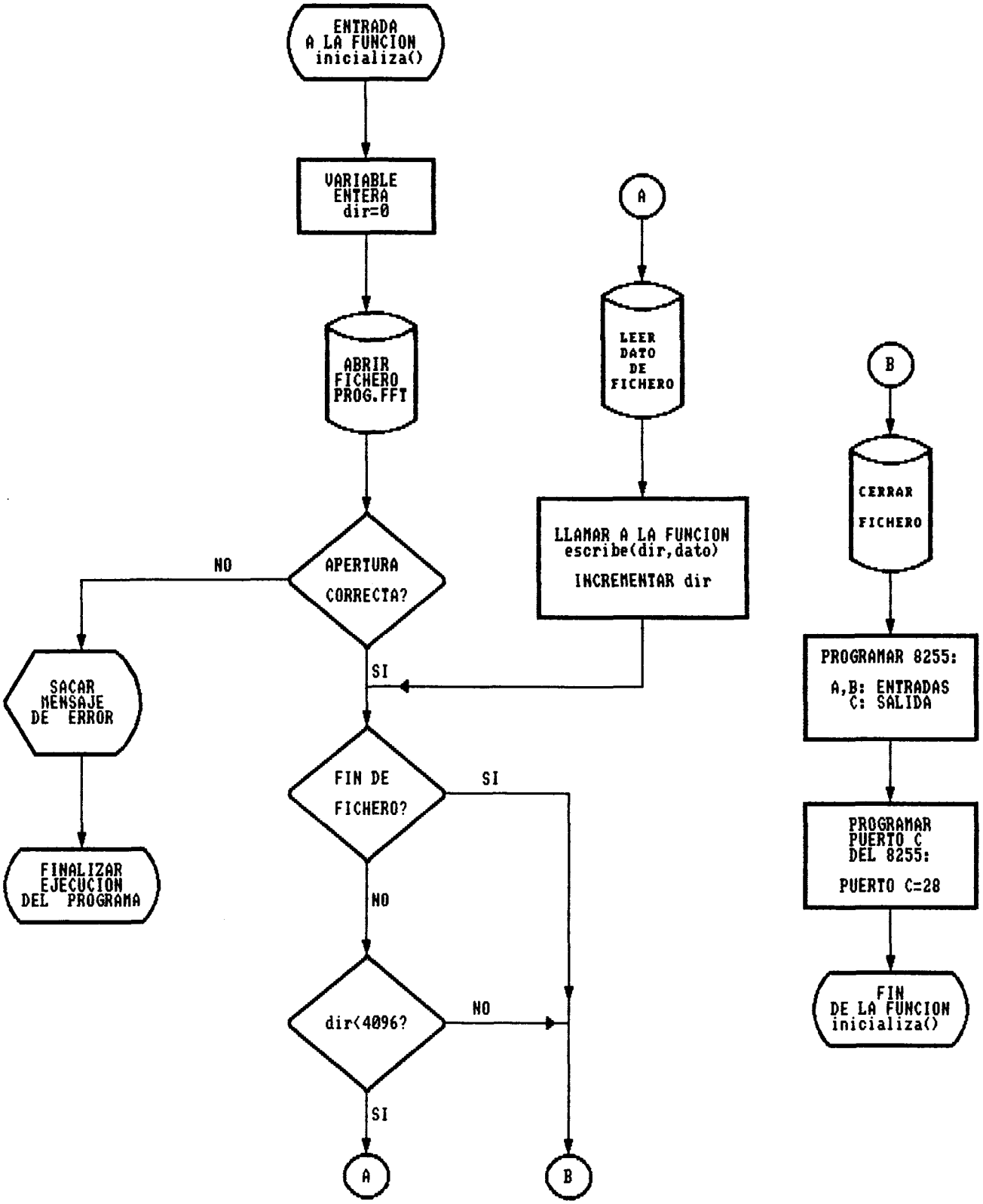
PROGRAMA REPFFT

Nota: El algoritmo que aqui se muestra, es el de la funcion main() de nuestro programa, ya que es por donde empieza a ejecutarse nuestro programa en C.

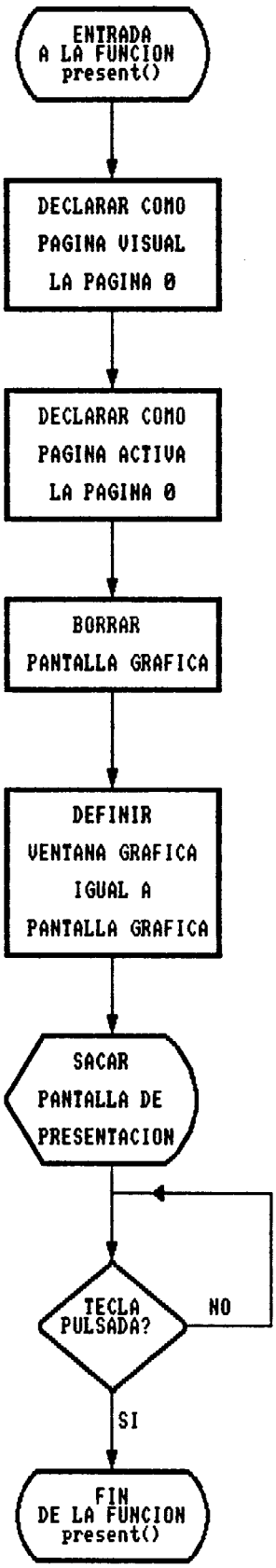


FUNCION inicializa()

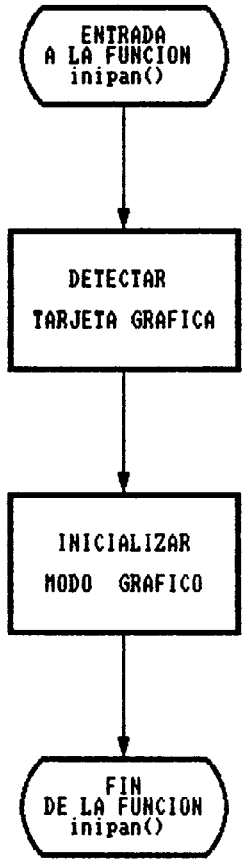
Descripcion: Carga el programa de control del TMS32010 en la memoria del sistema.



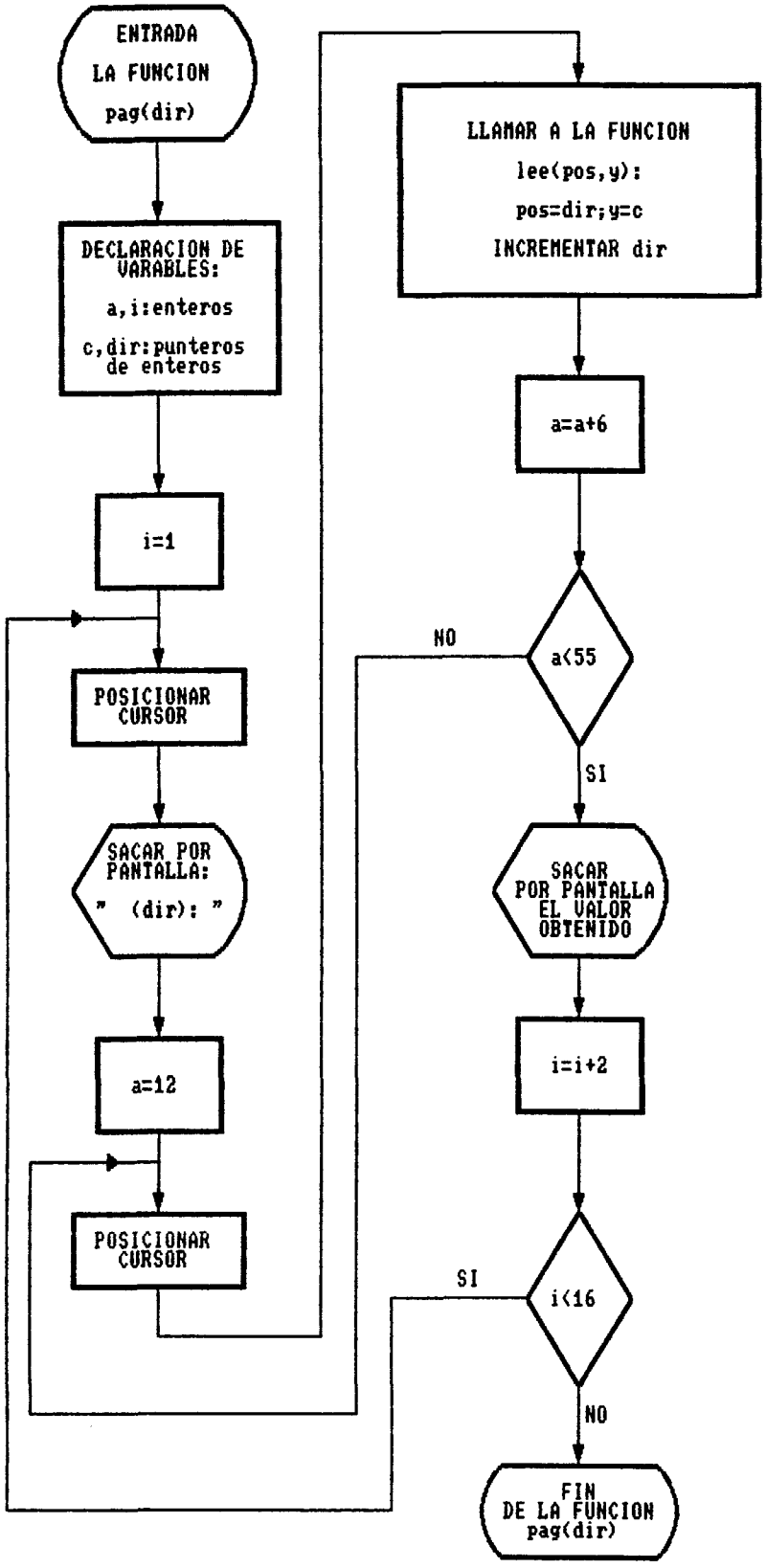
FUNCION present()



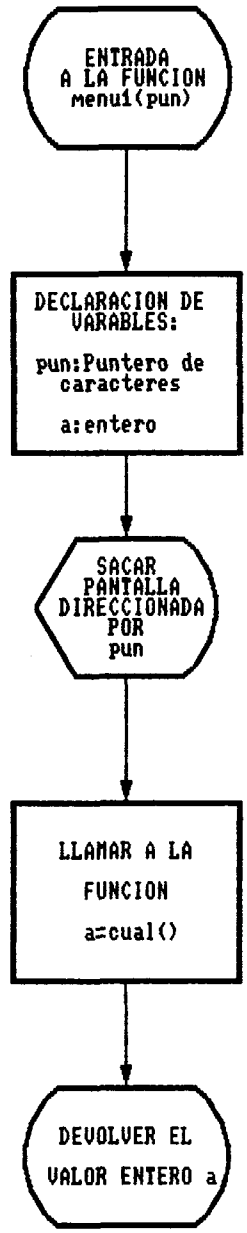
FUNCION inipan()



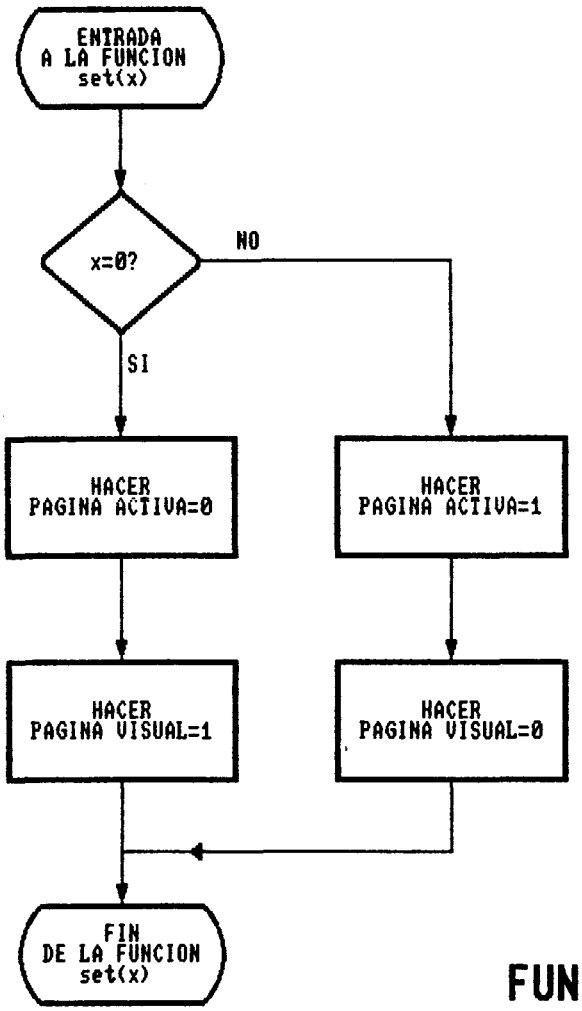
FUNCION pag(dir)



FUNCION menu1(pun)

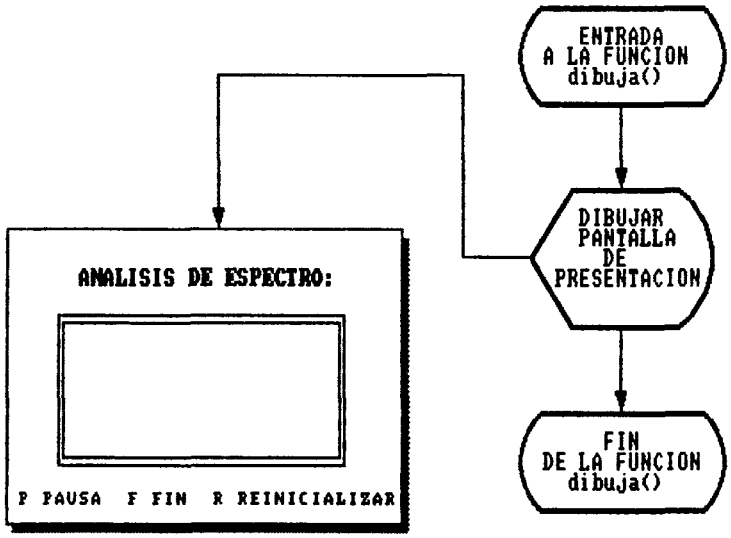


FUNCION set(x)

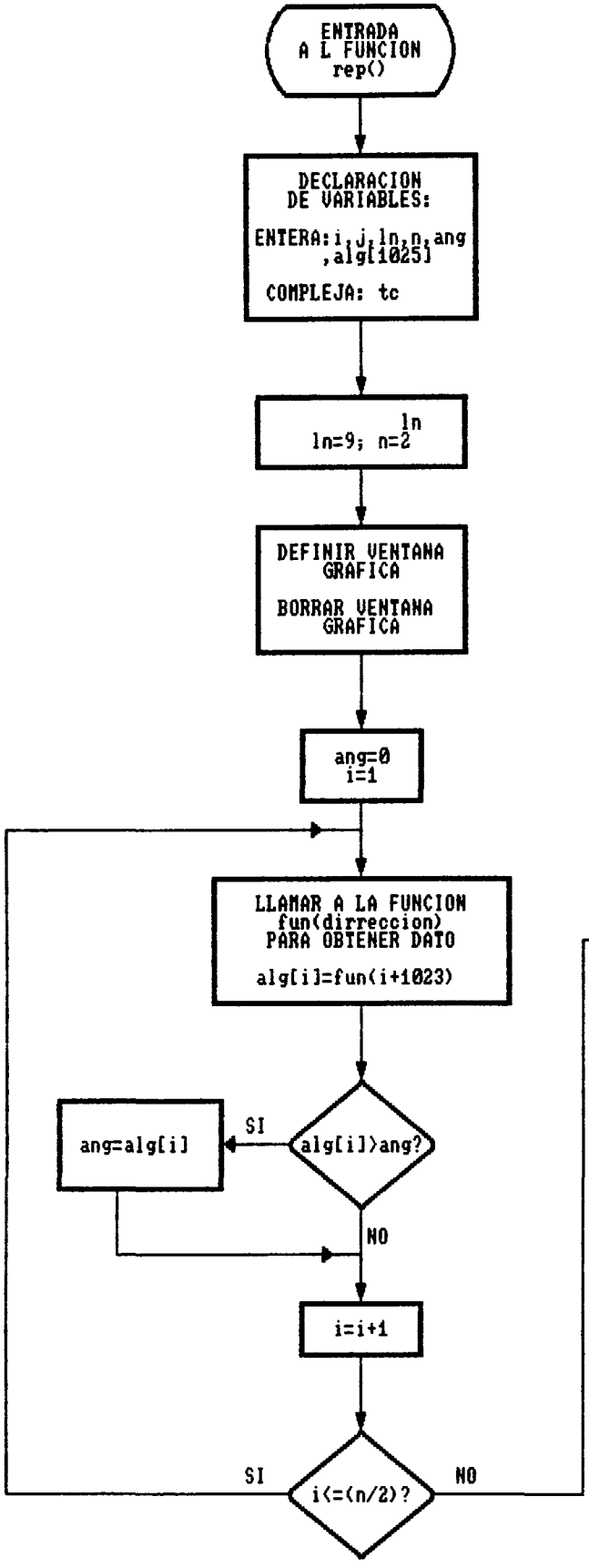


FUNCION fin()

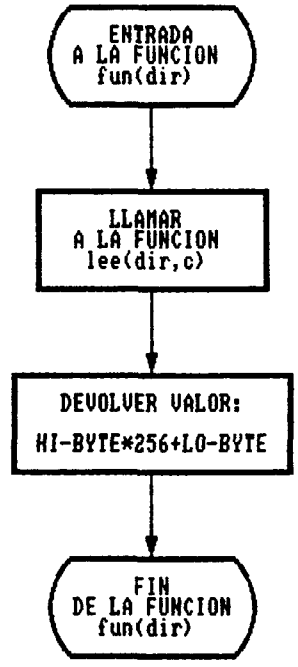
FUNCION dibuja()



FUNCION rep()



FUNCION fun(dir)



PRESUPUESTO APROXIMADO

PRESUPUESTO APROXIMADO:

CONCEPTO:	PRECIO P/U:	TOTAL PESETAS:
30 Zócalos profesionales	120	3.600
10 Integrados Puertas Lógicas	80	800
5 Integrados 74HC244	180	900
1 Integrado 74HC255	200	200
5 Integrados 74HC373	180	900
1 Integrado 8237	800	800
1 Integrado 8255	800	800
1 Integrado TMS32010	3.500	3.500
2 Integrados MT5C6408	7.000	14.000
1 Integrado ADQ585	7.000	7.000
1 Integrado HI574	15.000	15.000
2 Integrados 74LS93	180	360
2 Integrados 74LS138	180	360
Conectores	2.500	2.500
Cable plano	900	900
Placas de circuito impreso	3.500	3.500
Cajas	3.500	3.500
		58.620

APENDICE A:

FOTOCOPIAS DE CARACTERISTICAS

RS data

High speed, precision sample and hold circuit AD585

Stock number 631-446

The AD585 is a monolithic sample-and-hold circuit consisting of a high performance operational amplifier in series with an ultra-low leakage analogue switch and a FET input integrating amplifier. An internal holding capacitor and connections to the internal feedback resistors, completes the sample and hold.

With the analogue switch closed, the AD585 functions like a standard op amp; any feedback network may be connected around the device to control gain and frequency response. With the switch open, the capacitor holds the ouput at its previous level.

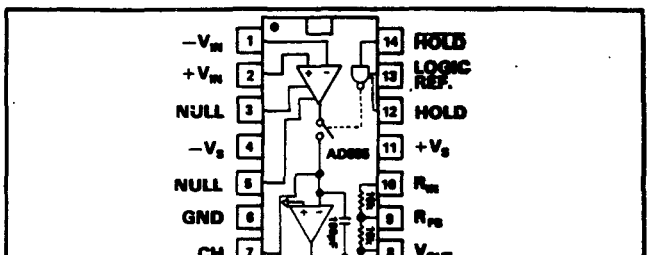
The AD585 offers performance previously unavailable in monolithic sample-and-hold amplifiers. The combination of a fast acquisition time (3.0μs to 0.01%) and low offset step (3mV) are suitable for high speed 12-bit data acquisition systems.

Features

- Fast 3.0μs acquisition time to ±0.01%
- Low droop rate: 1.0mV/ms
- Sample/Hold offset step: 3mV
- Aperture jitter: 0.5ns
- Internal hold capacitor.

Applications

- ▲ Data acquisition systems
- ▲ Data distribution systems
- ▲ Analogue delay and storage
- ▲ Peak amplitude measurements.



Absolute maximum ratings

Operating voltage	±18V
Differential input voltage	30V
Input voltage, either input	±Vs
Operating temperature range	−55°C to +125°C
Storage temperature range	−65°C to +150°C
Lead temperature (soldering 15s)	300°C

ATTENTION

OBSERVE PRECAUTIONS FOR HANDLING

ELECTROSTATIC SENSITIVE DEVICES

Electrical characteristics V_S = ± 15V, T_A = 25°C, C_H = Internal A = +1.

Parameter	Conditions	Min.	Typ.	Max.	Units
Sample/Hold characteristics					
Acquisition time	10V step to 0.01%	—	—	3.0	μs
	20V step to 0.01%	—	—	5.0	μs
Aperture time	20V p-p input, hold 0V	—	35	—	ns
Aperture jitter	20V p-p input, hold 0V	—	0.5	—	ns
Settling time	20V p-p input, hold 0V to 0.01%	—	0.5		μs
Droop rate		—	—	1.0	mV/ms
Droop rate	T_{min} to T_{max}		Doubles every 10°C		
Charge transfer		—	0.3	—	pC
Sample to hold offset		—	—	3.0	mV
Feedthrough	20V p-p, 10kHz input	—	0.5	—	mV p-p

Parameter	Conditions	Min.	Typ.	Max.	Units
Transfer characteristics					
Open loop gain	V_{OUT} = 20Vp-p, R_L = 2k	—	200	—	kV/V
Closed loop gain accuracy		—	0.05	—	%
Common mode rejection	V_{CM} = 20Vp-p, F = 50Hz	80	—	—	dB
Small signal gain bandwidth	V_{OUT} = 100mVp-p	—	2.0	—	MHz
Full power bandwidth	V_{OUT} = 20Vp-p	—	160	—	kHz
Slew rate	V_{OUT} = 20Vp-p	—	10	—	V/μs
Output resistance hold mode	I_{OUT} = ±10mA	—	0.05	—	Ω
Output short circuit current		—	—	50	mA
Analogue input characteristics					
Offset voltage		—	—	2.0	mV
Offset voltage	T_{min} to T_{max}	—	—	3.0	mV
Bias current		—	2.0	—	nA
Input capacitance	f = 1MHz	—	10	—	pF
Input resistance, sample or hold	20mVp-p input, A = +1	—	10^{12}	—	Ω
Digital input characteristics					
Input voltage with respect to pin 13	(TTL compatible)				
Logic reference		1.2	1.4	1.6	V
Hold mode, $\overline{HOLD}$ – logic reference					
or logic reference – $\overline{HOLD}$	T_{min} to T_{max}	0.4	—	—	V
Sample mode	T_{min} to T_{max}	–0.4	—	—	V
Logic input current	(Either input)	—	—	50	μA
Power supply characteristics					
Operating voltage range		+5/–12	±15	±18	V
Supply current	R_L = ∞	—	—	10	mA
Power supply rejection	Sample mode	70	—	—	dB

Typical characteristics

Figure 1 **Sample to hold offset vs. logic level ($\overline{HOLD}$ active)**

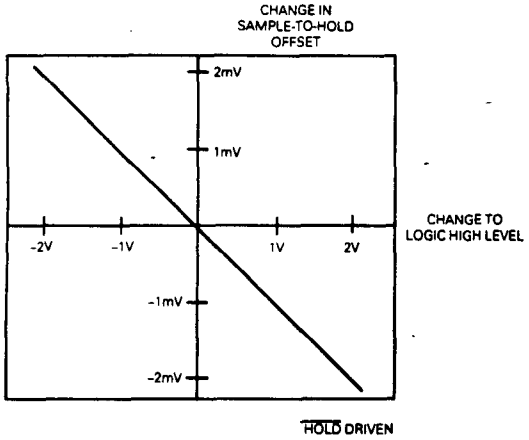


Figure 3 **Follower, sample mode (large signal)**

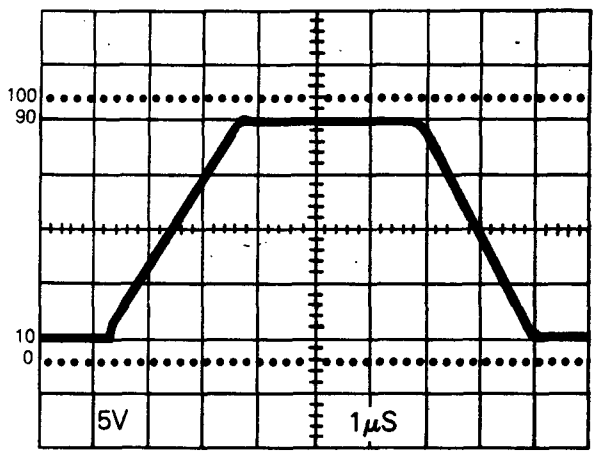


Figure 2 **Acquisition time vs. hold capacitance**

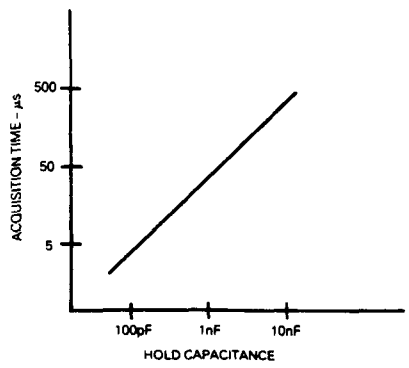
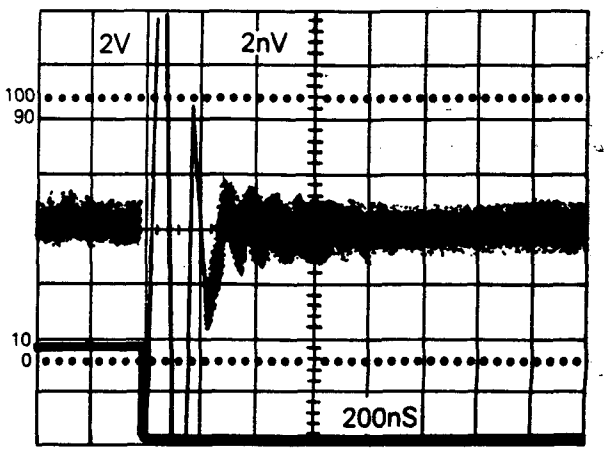


Figure 4 **Sample – to hold settling time ($\overline{HOLD}$ active)**



Sample data systems

In sample data systems there is a number of limiting factors in digitizing high frequency signals accurately. Figure 5 shows pictorially the sample-and-hold errors that are the limiting factors. In the following discussions of error sources the errors will be divided into the following groups: 1. Sample-to-Hold Transition; 2. Hold Mode and 3. Hold-to-Sample Transition.

Figure 5 **Sequence diagram showing various sample and hold characteristics**

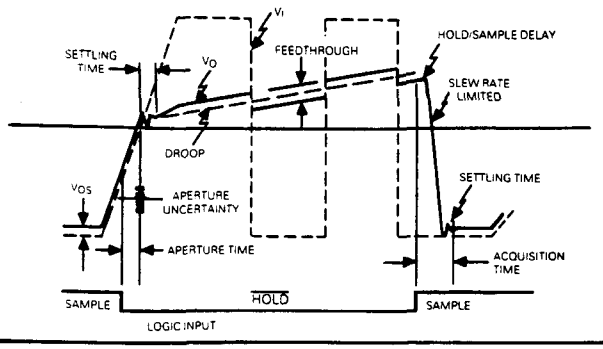
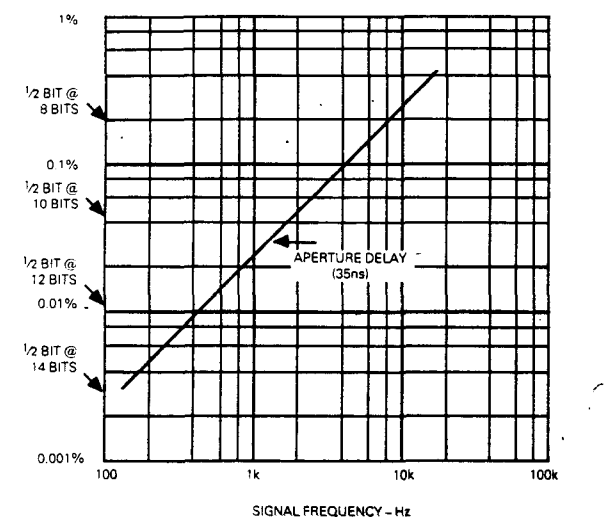


Figure 6 **Aperture delay error vs. frequency**



Sample-to-hold transition

The aperture time is the time required for the sample-and-hold amplifier to switch from sample to hold. Since this is effectively a constant then it may be tuned out. If however, the aperture delay time is not accounted for then errors of the magnitude as shown in Figure 6 will result.

To eliminate the aperture delay as an error source the sample-to-hold command may be advanced with respect to the input signal.

Once the aperture time has been eliminated as an error source then the aperture jitter which is the variation in aperture time from sample-to-sample remains. The aperture jitter is a true error source and must be considered. The aperture jitter is a result of noise within the switching network which modulates the phase of the hold command and is manifested in the variations in the value of the analogue input that has been held. The aperture error which results from this jitter is directly related to the dV/dT of the analogue input.

The error due to aperture jitter is easily calculated as shown below. The error calculation takes into account the desired accuracy corresponding to the resolution of the A/D converter.

$$F_{max} = \frac{2^{-(N+1)}}{\pi \text{ (Aperture jitter)}}$$

For an application with a 10-bit A/D converter with a 10V full scale to a ½LSB error maximum.

$$F_{max} = \frac{2^{-(10+1)}}{\pi\ (0.5 \times 10^{-9})}$$

$F_{max} = 310.8\,kHz.$

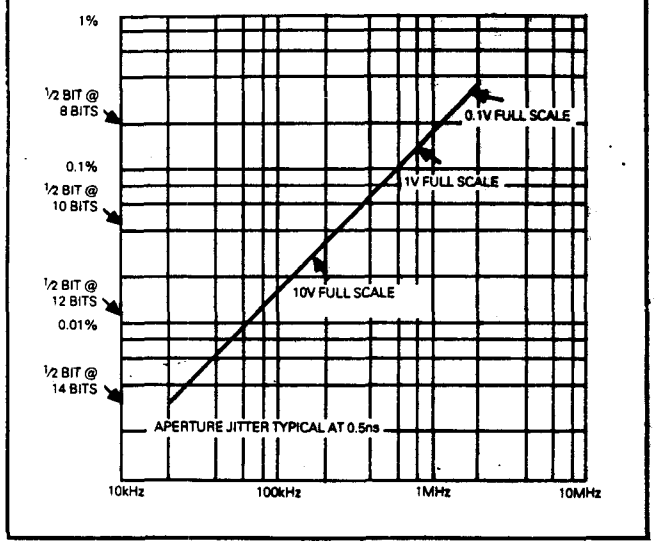
For an application with a 12-bit A/D converter with a 10V full scale to a ½LSB error maximum.

$$F_{max} = \frac{2^{-(12+1)}}{\pi\ (0.5 \times 10^{-9})}$$

$F_{max} = 77.7\,kHz.$

Figure 7 shows the entire range of errors induced by aperture jitter with respect to the input signal frequency.

Figure 7 **Aperture jitter error vs. frequency**



Sample-to-hold offset is caused by the transfer of charge to the holding capacitor via the gate capacitance of the switch when switching in to hold. Since the gate capacitance couples the switch control voltage applied to the gate on to the hold capacitor, the resulting sample-to-hold offset is a function of the logic level as shown in Figure 1.

The sample-to-hold offset can be reduced by adding capacitance to the internal 100pF capacitor. This may be easily accomplished by adding an external capacitor between Pins 7 and 8. The sample-to-hold offset is then governed by the relationship.

$$\text{S/H Offset (V)} = \frac{\text{Charge (pC)}}{C_H \text{ Total (pF)}}$$

For the AD585 in particular it becomes:

$$\text{S/H Offset (V)} = \frac{0.3\,pC}{100\,pF + (C_{EXT})}$$

The addition of an external hold capacitor also affects the acquisition time of the AD585. The change in acquisition time with respect to the C_{EXT} is shown graphically in Figure 2.

Hold mode

In the hold mode there are two important specifications that must be considered, feedthrough and the droop rate. Feedthrough errors appear as an attenuated version of the input at the output while in the hold mode, hold mode feedthrough varies with frequency, increasing at higher frequencies. Feedthrough is an important specification when a sample and hold follows an analogue multiplexer that switches between many different channels.

Hold mode droop rate is the change in output voltage per unit of time while in the hold mode. Hold mode droop originates as leakage from the hold capacitor, of which the major leakage current contributors are switch leakage current and bias current. The rate of voltage current on the capacitor dV/dT is the ratio of the total leakage current I_L to the hold capacitance C_H.

$$\text{Droop Rate} = \frac{dV_{OUT}}{dT}\ (\text{Volts/Sec}) = \frac{I_L\ (pA)}{C_H\ (pA)}$$

For the AD585 in particular,

$$\text{Droop Rate} = \frac{100\,pA}{100\,pF + (C_{EXT})}$$

Additionally, the leakage current doubles for every 10°C increase in temperature; therefore, the hold mode droop rate characteristic will also double in the same fashion. The hold mode droop rate can be traded-off with acquisition time to provide the best combination of droop error and acquisition time. The tradeoff is easily accomplished by varying the value of C_{EXT}.

Since a sample and hold is used typically in combination with an A/D converter, then the total droop in the output voltage has to be less than ½LSB during the period of a conversion. The maximum allowable signal change on the input of an A/D converter is:

$$\Delta V\ \text{max} = \frac{\text{Full Scale Voltage}}{2^{(N+1)}}$$

Once the maximum ΔV is determined then the conversion time of the A/D converter (T_{CONV}) is required to calculate the maximum allowable dV/dT.

$$\frac{V\ \text{max}}{dt} = \frac{\Delta V\ \text{max}}{T_{CONV}}$$

The maximum $\frac{dV\ \text{max}}{dT}$ as shown by the previous equation is the limit not only at 25°C but at the maximum expected operating temperature range. Therefore, over the operating temperature range the following criteria must be met ($T_{OPERATION} - 25°C) = \Delta T$.

$$\frac{dV\ 25°C}{dT} \times 2\ \frac{(\Delta T°C)}{10°C} \leqslant \frac{dV\ \text{max}}{dT}$$

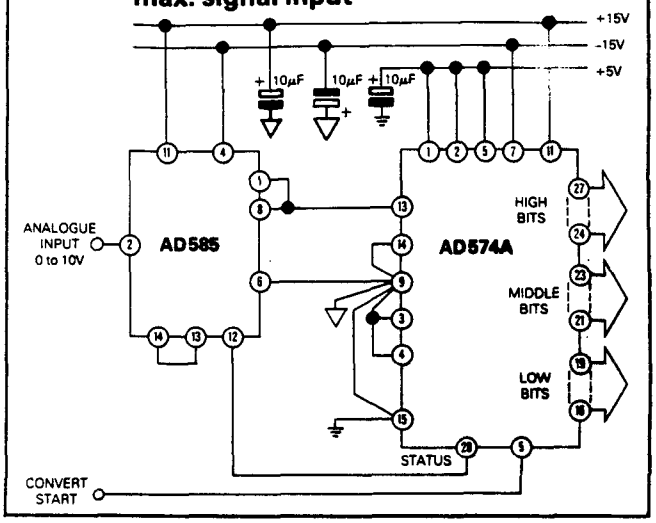
Hold-to-sample transition

The Nyquist theorem states that a band-limited signal which is sampled at a rate at least twice the maximum signal frequency can be reconstructed without loss of information. This means that a sampled data system must sample, convert and acquire the next point at a rate equal to twice the signal frequency. Thus the maximum input frequency is equal to

$$\frac{1}{2\,(T_{ACQ} + T_{CONV} + T_{AP})}$$

Where T_{ACQ} is the acquisition of the sample-to-hold amplifier, T_{AP} is the maximum aperture time (small enough to be ignored) and T_{CONV} is the conversion time of the A/D converter.

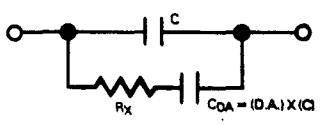
Figure 8 **12-bit A to D conversion system, 26.3kHz throughput rate, 13.1kHz max. signal input**



Optional capacitor selection

If an additional capacitor is going to be used in conjunction with the internal 100pF capacitor it must have a low dielectric absorption. Dielectric absorption is just that; it is the charge absorbed into the dielectric that is not immediately added to or removed from the capacitor when rapidly charged or discharged. The capacitor with dielectric absorption is modelled in Figure 9.

Figure 9 **Capacitor model with dielectric absorption**



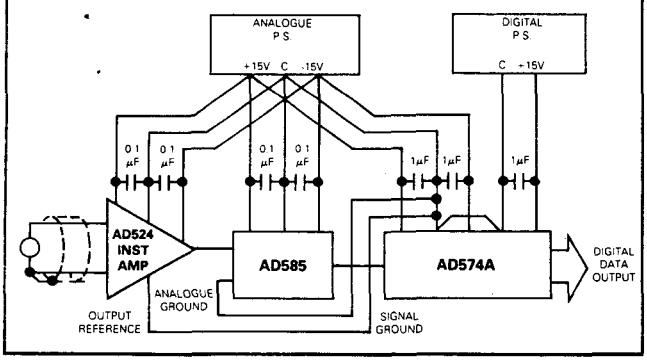
If the capacitor is charged slowly, C_{DA} will eventually charge to the same value as C. But unfortunately, good dielectrics have very high resistances, so while C_{DA} may be small, R_X is large and the time constant $R_X\,C_{DA}$ typically runs into the millisecond range. In fast-charge, fast-discharge situations the effect of dielectric absorption resembles 'memory'. In a data acquisition system where many channels with widely varying data are being sampled the effect is to have an ever changing offset which appears as a very nonlinear sample-to-hold offset since the difference between the voltage being

measured and the voltage previously measured determines the fraction by which the dielectric absorption figure is multiplied. It is impossible to readily correct for this error source. The only solution is to use a capacitor with dielectric absorption less than the maximum tolerable error. Capacitor types such as polystyrene, polypropylene or Teflon are recommended.

Grounding

Many data-acquisition components have two or more ground pins which are not connected together within the device. These 'grounds' are usually referred to as the Logic Power Return, Analogue, Common (Analogue Power Return), and Analogue Signal Ground. These grounds must be tied together at one point, usually at the system power-supply ground. Ideally, a single solid ground would be desirable. However, since current flows through the ground wires and etch strips of the circuit cards, and since these paths have resistance and inductance, hundreds of millivolts can be generated between the system ground point and the ground pin of the AD585. Separate ground returns should be provided to minimise the current flow in the path from sensitive points to the system ground point. In this way supply currents and logic-gate return currents are not summed into the same return path as analogue signals where they would cause measurement errors.

Figure 10 **Basic grounding practise**



Typical applications

Figure 11 **Connection diagram, gain = +1, $\overline{\text{HOLD}}$ active**

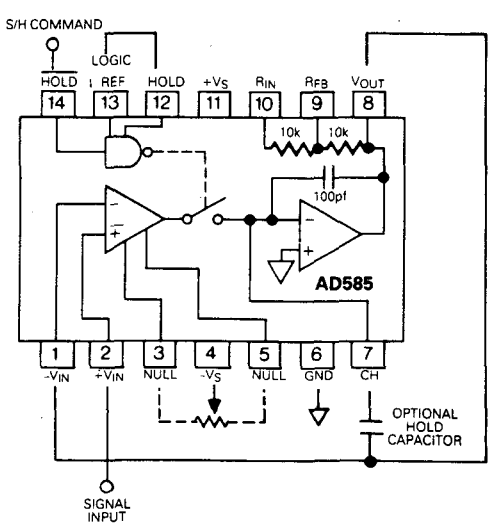


Figure 12 **Connection diagram, gain = +2, $\overline{\text{HOLD}}$ active**

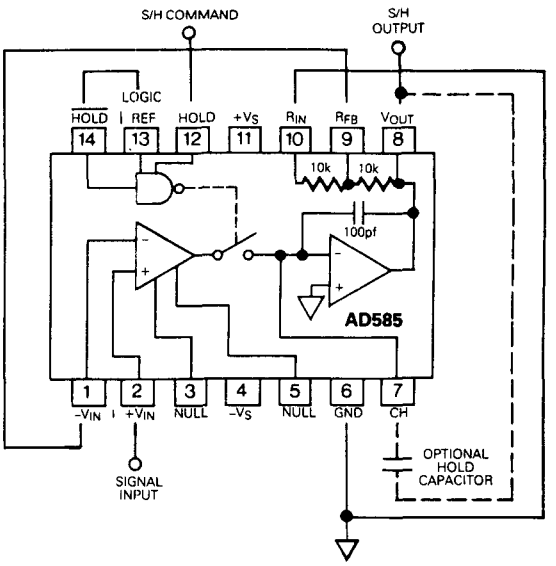
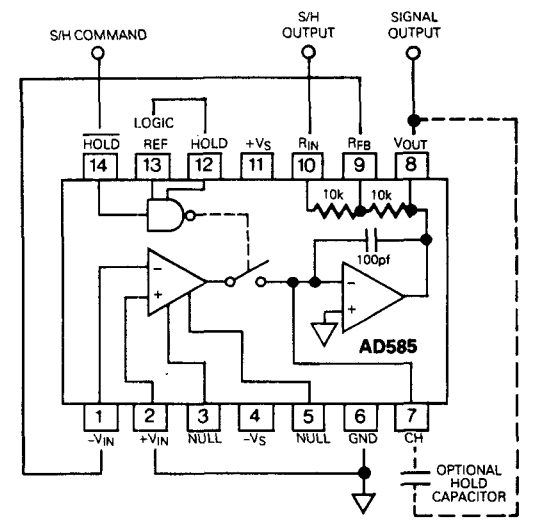


Figure 13 **Connection diagram, gain = –1, $\overline{\text{HOLD}}$ active**



6020

RS data

R.S. Components Ltd. PO Box 99, Corby, Northants, NN17 9RS
An Electrocomponents Group Company

Telephone: 0536 201234

2. −3.0V for pulse width < 20ns.

3. I_{CC} is dependent on output loading and cycle rates.

4. This parameter is sampled.

5. Test conditions as specified with the output loading as shown in Fig. 1 unless otherwise noted.

6. THZCE, THZWE and THZOE are specified with C_L = 5pF as in Fig. 2. Transition is measured ±500mV from steady state voltage.

7. At any given temperature and voltage condition, THZCE is less than TLZCE.

8. $\overline{WE}$ is high for READ cycle.

9. Device is continuously selected. All Chip Enables held in their active state.

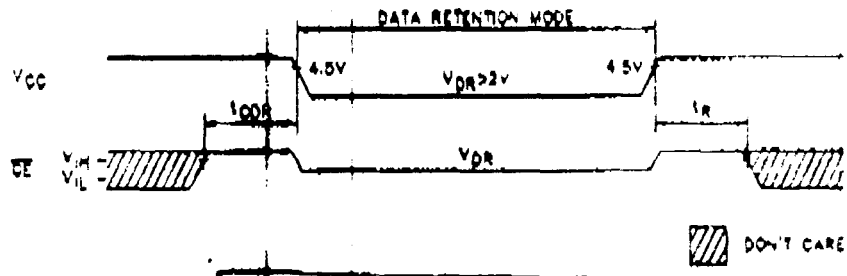
10. Address valid prior to or coincident with latest occurring Chip Enable.

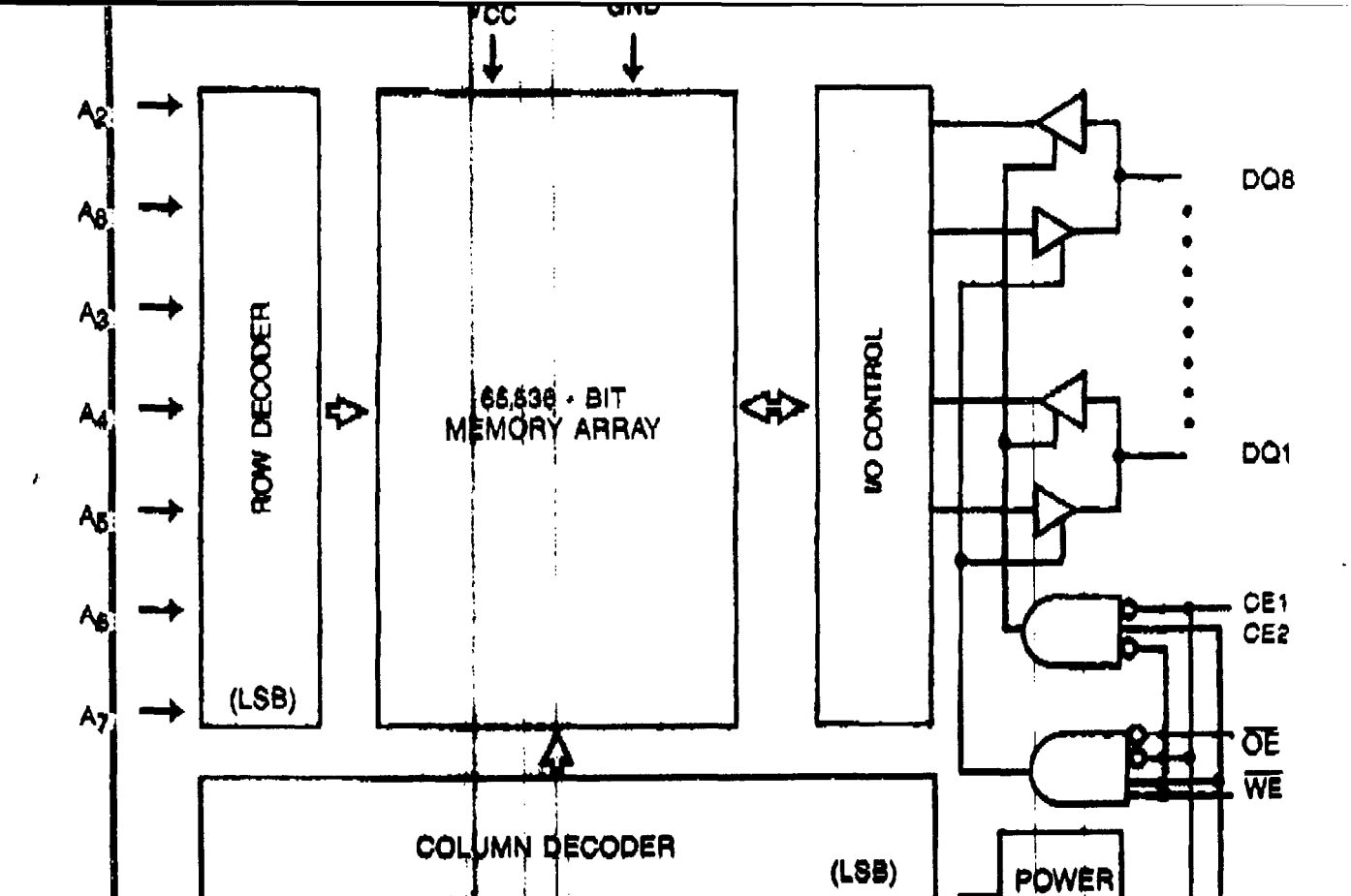
11. t_{RC} = Read Cycle Time. (Page 4)

DATA RETENTION ELECTRICAL CHARACTERISTICS (L Version Only)

SYMBOL	PARAMETER	TEST CONDITIONS		MIN	TYP	MAX	UNIT
V_{DR}	V_{CC} for Retention Data			2		—	V
I_{CCDR}	Data Retention Current	$\overline{CE} \geq (V_{CC} - 0.2V)$ $V_{IN} \geq (V_{CC} - 0.2V)$ or $\leq 0.2V$	V_{CC}=2V		95	500	µA
			V_{CC}=3V		350	750	µA
t_{CDR}[4]	Chip Deselect to Data Retention Time			0		—	ns
t_R[4]	Operation Recovery Time			t_{RC}[11]			ns

LOW V_{CC} DATA RETENTION WAVEFORM





TRUTH TABLE — MT5C6408

MODE	$\overline{CE1}$	CE2	$\overline{WE}$	$\overline{OE}$	DQ OPERATION	POWER
STANDBY	H	X	X	X	HIGH Z	STANDBY
STANDBY	X	L	X	X	HIGH Z	STANDBY
READ	L	H	H	L	DOUT	ACTIVE
READ	L	H	H	H	HIGH Z	ACTIVE
WRITE	L	H	L	X	DIN	ACTIVE

Voltage on Vcc supply relative to Vss	-1.0V to +7.0V
Storage Temperature (Ceramic)	-65°C to +150°C
Storage Temperature (Plastic)	-55°C to +150°C
Power Dissipation	1 Watt
Short Circuit Output Current	50mA

*Stresses greater than those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect reliability.

ELECTRICAL CHARACTERISTICS AND RECOMMENDED DC OPERATING CONDITIONS
($0°C \leq T_A \leq 70°C$, $V_{CC} = 5.0V \pm 10\%$)

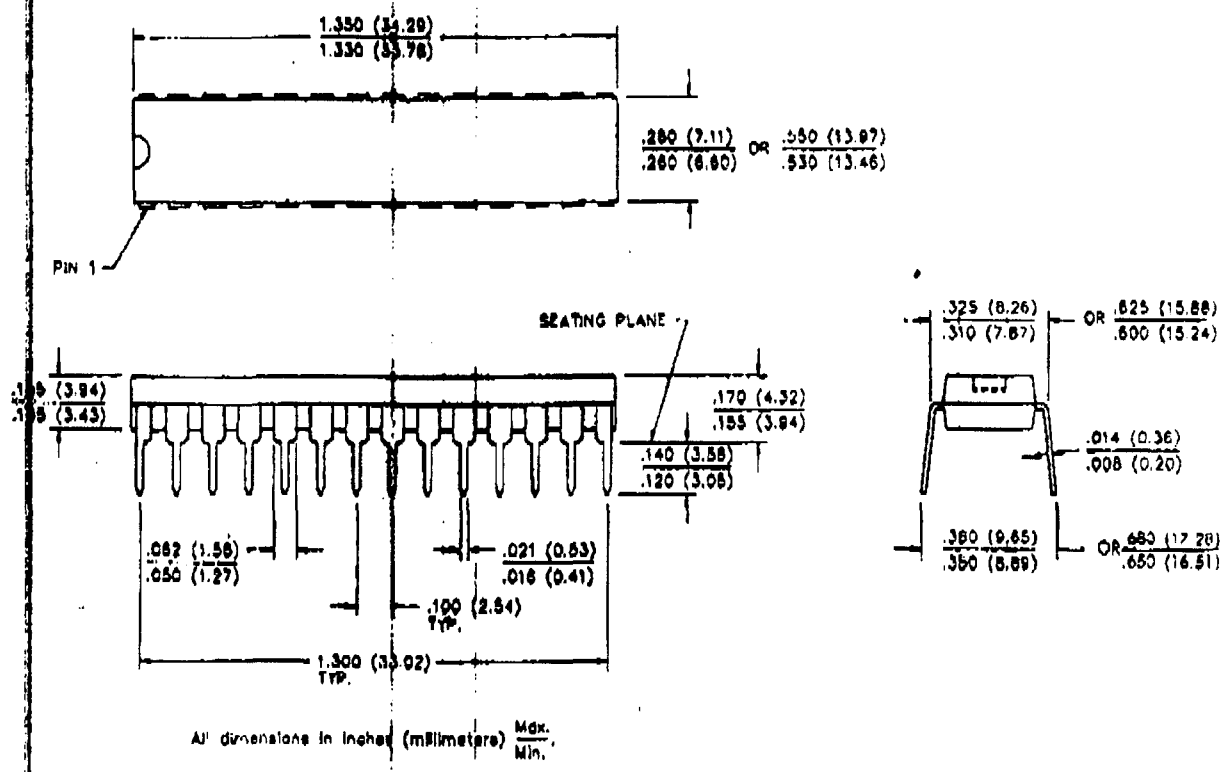
DESCRIPTION	CONDITIONS	SYMBOL	MIN	MAX	UNITS	NOTES
Input High (Logic 1) Voltage		V_{IH}	2.2	Vcc +1	V	1
Input Low (Logic 0) Voltage		V_{IL}	-0.5	0.8	V	1, 2
Input Leakage Current	$0V \leq V_{IN} \leq V_{CC}$	I_{LI}	-10	10	µA	
Output Leakage Current	Output(s) Disabled, $0V \leq V_{OUT} \leq V_{CC}$	I_{LO}	-10	10	µA	
Output High Voltage	$I_{OH} = -2.0mA$	V_{OH}	2.4		V	1
Output Low Voltage	$I_{OL} = 4.2mA$	V_{OL}		0.4	V	1

DESCRIPTION	CONDITIONS	SYMBOL	-12	-15	-20	-25	-30	-35	UNITS	NOTES
Power Supply Current: Operating	$\overline{CE} \leq V_{IL}$, V_{CC} = Max., Outputs Open	I_{CC}	140	130	120	110	100	100	mA	3
	$\overline{CE} \geq V_{IH}$, V_{CC} = Max.	I_{SB1}	60	50	45	40	40	40	mA	
Power Supply Current: Standby	$\overline{CE} \geq V_{CC} - 0.2$, V_{CC} = Max. $V_{IL} \leq V_{SS} + 0.2$, $V_{IH} \geq V_{CC} - 0.2$, f = 0	I_{SB2}	5	5	5	5	5	5	mA	

CAPACITANCE

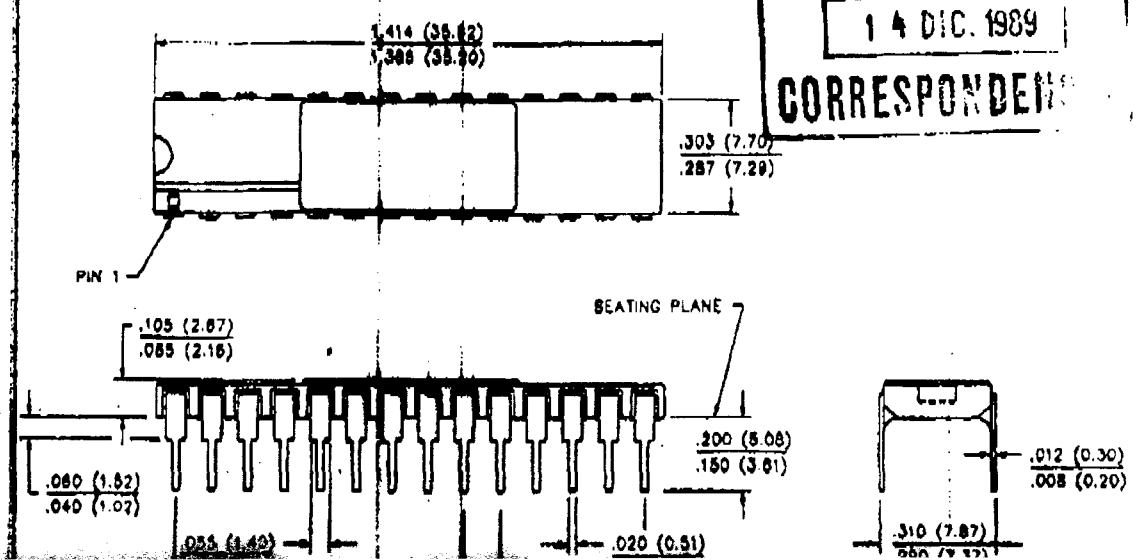
DESCRIPTION	CONDITIONS	SYMBOL	MIN	MAX	UNITS	NOTES
Input Capacitance	T_A = 25°C, f = 1MHz	C_I		7	pF	4
Output Capacitance	V_{CC} = 5V	C_O		7	pF	4

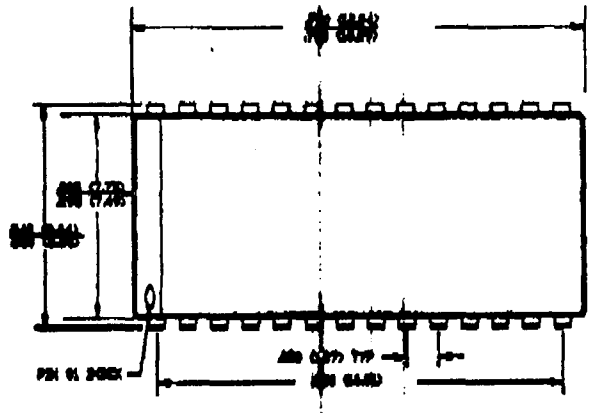
28 PIN PLASTIC DIP
300 OR 600 MIL



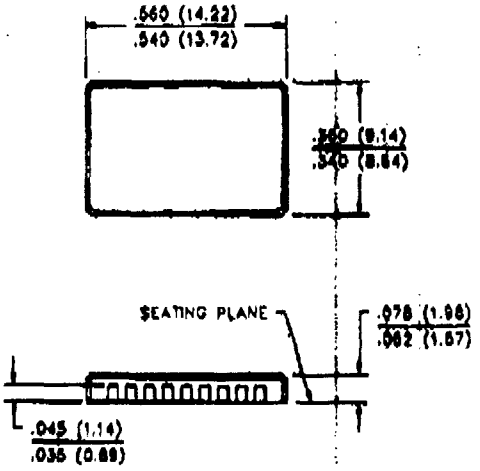
All dimensions in inches (millimeters) Max./Min.

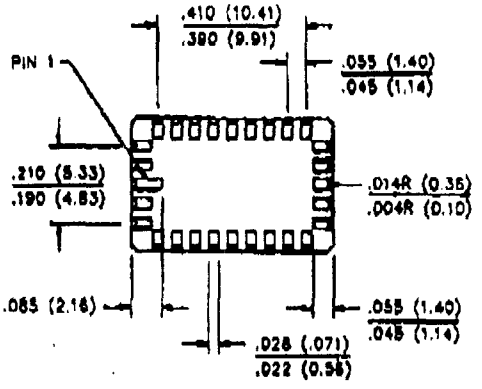
28 PIN CERAMIC DIP





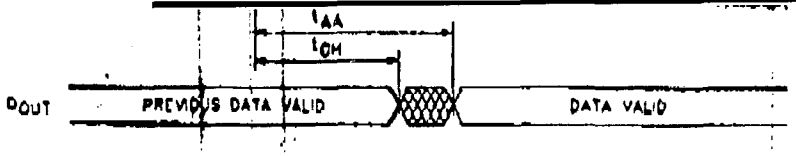
28 PIN LCC [1].



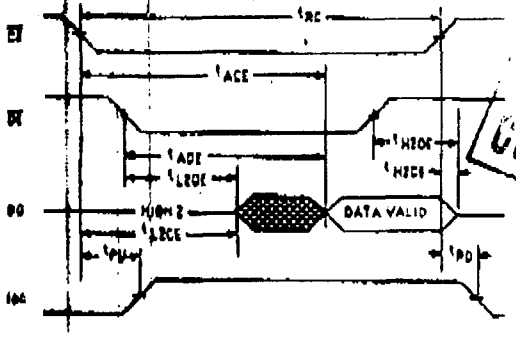


All dimensions in inches (millimeters) Metric or English.

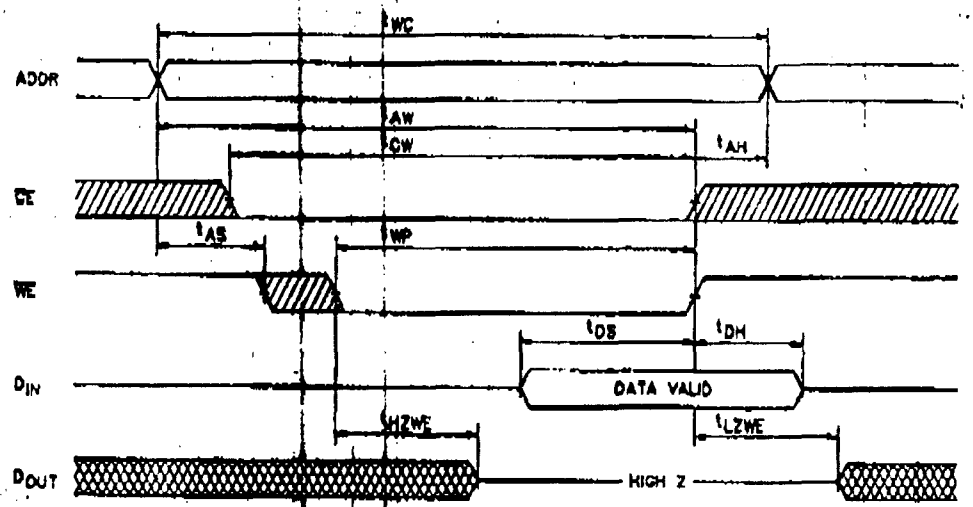
1. 32 Pin LCC also available.



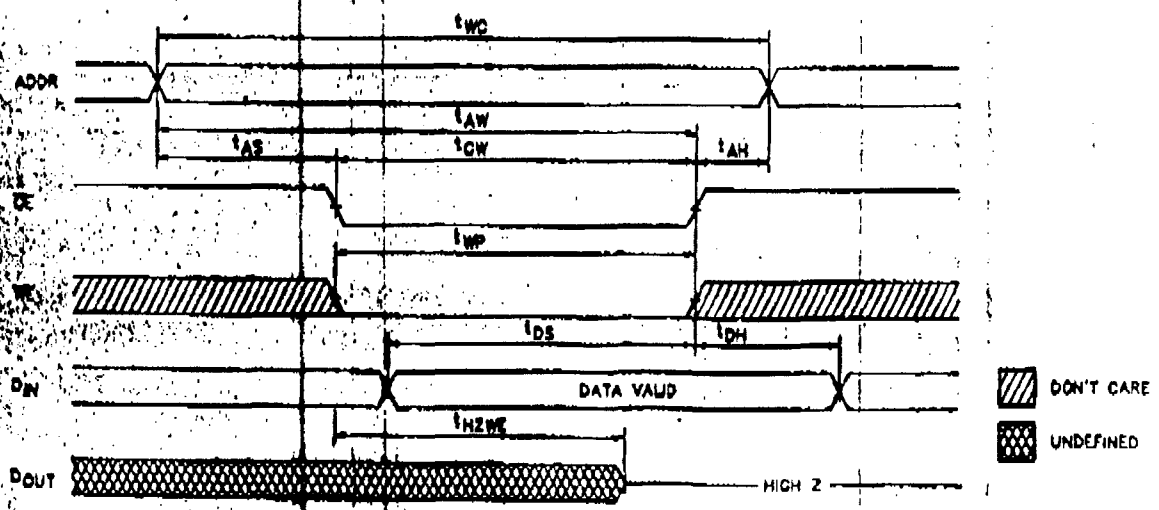
READ CYCLE NO. 2 (7, 8, 10)



WRITE CYCLE NO. 1
(Write Enable Controlled)



WRITE CYCLE NO. 2
(Chip Enable Controlled)



DESCRIPTION	SYM	-12 MIN	-12 MAX	-15 MIN	-15 MAX	-20 MIN	-20 MAX	-25 MIN	-25 MAX	-30 MIN	-30 MAX	-35 MIN	-35 MAX	UNITS	NOTES
READ Cycle															
READ cycle time	t_{RC}	12		15		20		25		30		35		ns	
Address access time	t_{AA}		12		15		20		25		30		35	ns	
Chip enable access time	t_{ACE}		12		15		20		25		30		35	ns	
Output hold from access change	t_{OH}	3		3		3		3		3		3		ns	
Chip enable to output in low Z	t_{LZCE}	5		5		5		5		5		5		ns	
Chip disable to output in high Z	t_{HZCE}		10		10		15		15		20		20	ns	6,7
Chip enable to power up time	t_{PU}	0		0		0		0		0		0		ns	
Chip disable to power down time	t_{PD}		12		15		20		25		30		35	ns	
Output enable access time	t_{AOE}		10		12		15		15		20		20	ns	
Output enable to output in low Z	t_{LZOE}	0		0		0		0		0		0		ns	
Output disable to output in high Z	t_{HZOE}		10		10		15		15		20		20	ns	6
WRITE Cycle															
WRITE cycle time	t_{WC}	12		15		20		25		30		35		ns	
Chip enable to end of write	t_{CW}	10		12		15		20		25		30		ns	
Address valid to end of write	t_{AW}	10		12		15		20		25		30		ns	
Address set-up time	t_{AS}	0		0		0		0		0		0		ns	
Address hold from end of write	t_{AH}	0		0		0		0		0		0		ns	
Write pulse width	t_{WP}	10		12		15		20		25		25		ns	
Data set-up time	t_{DS}	10		10		12		15		15		15		ns	
Data hold time	t_{DH}	0		0		0		0		0		0		ns	
Write disable to output in low Z	t_{LZWE}	0		0		0		0		0		0		ns	
Write enable to output in high Z	t_{HZWE}	0	10	0	10	0	15	0	15	0	15	0	15	ns	6

AC Test Conditions

Input pulse levels	Vss to 3.0V
Input rise and fall times	5ns
Intput timing reference levels	1.5V
Output reference levels	1.5V
Output load	See figures 1 and 2

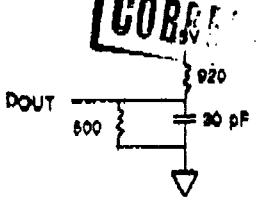


Fig. 1 OUTPUT LOAD EQUIVALENT

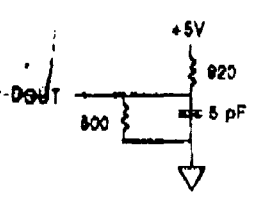
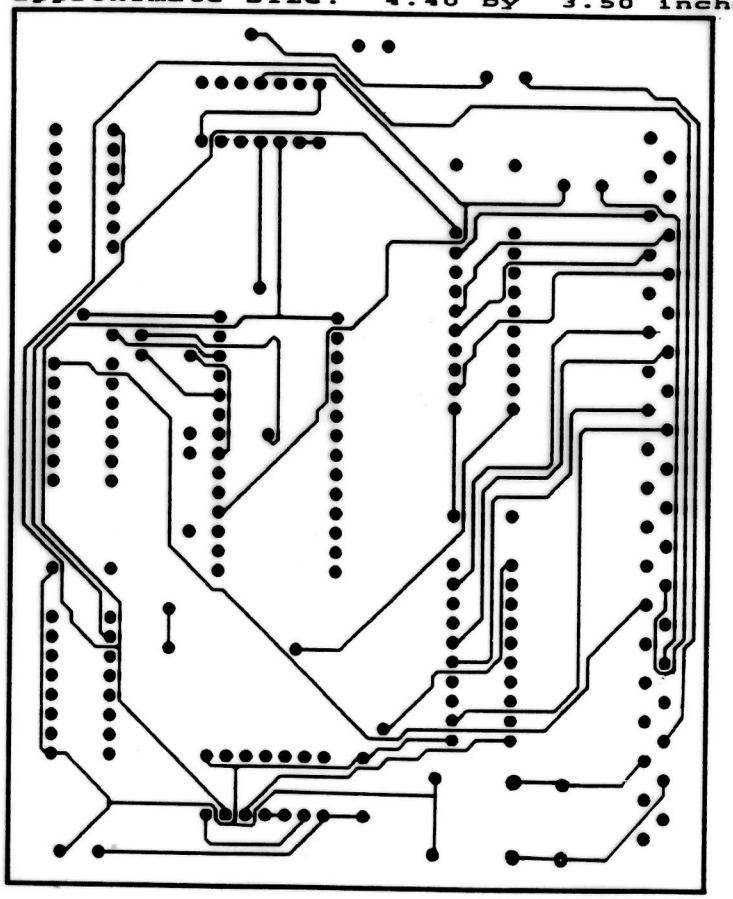


Fig. 2 OUTPUT LOAD EQUIVALENT

APENDICE B:

PLACAS DE CIRCUITO IMPRESO

2X artwork 22 Sep 1990 11:38:51
file: conv.12
v1.0 r2 holes: 208 upper layer
approximate size: 4.40 by 3.50 inches



2X artwork 22 Sep 1990 11:38:51
file: conv.12
v1.0 r2 holes: 208 lower layer
approximate size: 4.40 by 3.50 inches