UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

DISEÑO Y CONSTRUCCION DE UN SISTEMA DIGITAL DE DESARROLLO BASADO EN EL MICROPROCESADOR 8088 Y EN UNA MICROCOMPUTADORA TIPO PC (Personal Computer) O PS (Personal System) (IBM compatible)

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO MECANICO ELECTRICISTA

(AREA ELECTRONICA)

P R E S E N T A :

AGUSTIN EDUARDO ALVAREZ VACA

DIRECTOR DE TESIS: ING. ANTONIO SALVA CALLEJA

MEXICO, D. F. 1989

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

C O N T E N I D O

P R O L O G O

Con la creación del Microprocesador en 1971 (desarrollado por Intel Corp.), nació simultáneamente la Era de las Microcomputadoras.

El amplio espectro de aplicación de los Microprocesadores es posible debido a su pequeño tamaño y bajo costo. Hoy, el Microprocesador afecta de forma importante nuestra vida cotidiana de varias maneras, haciendo posible el diseño e introducción en el mercado de productos sofisticados a precios razonables; es usado en automóviles, hornos de microondas, aparatos de sonido y video, calculadoras, computadoras personales, supercomputadoras, instrumentos de medición y control, juguetes, aplicaciones de biónica, robótica y bioingeniería, instrumentación médica, etc.

Gracias al Microprocesador es posible incrementar la productividad y reducir los costos en una industria, hacer crecer nuevos mercados y negocios, así como liberar al ser humano de la realización de tareas repetitivas y minuciosas (como en la automatización de una planta armadora X) o bien tareas que entrañan un alto riesgo y requieren gran precisión (como en la supervisión y control de procesos industriales o control automático de misiones espaciales guiadas a distancia); el Microprocesador facilita la organización, el acceso y el manejo de información (p.ej. a través de las computadoras), favorece el acercamiento entre la gente del planeta mediante la implantación de sistemas confiables de comunicación, contribuye a una más alta calidad de productos y, en general, a un mejor nivel de vida.

El control de procesos es un campo que en la actualidad se ve fuertemente apoyado por arquitecturas con base en microprocesadores, que realizan en forma automática, la adquisición y relación de las variables del proceso y el control del mismo. En la práctica, a un proceso específico le corresponde para su control, una solución particular, de aquí la necesidad de diseñar arquitecturas básicas apoyadas en microprocesadores (microcomputadoras) que consten de componentes electrónicos de fácil adquisición en el mercado y en cuya elección se busque contar con lo más avanzado posible (según el presupuesto disponible), para efectuar el control del proceso.

Para determinar la factibilidad de un proyecto (sistema o instrumento prototipo), para optimizar sus requerimientos de hardware y para crear y depurar el software manejador-administrador de sus componentes, es de gran ayuda para el diseñador el contar con un "sistema BASE" a partir del cual pueda desarrollar sus aplicaciones particulares bajo ciertos requerimientos de operación y para la solución experimental de los problemas que se vayan presentando en el proceso de diseño y construcción. Así, al menos en una etapa inicial de la elaboración del proyecto de propósito específico, éste puede funcionar como una extensión o "hardware satélite" del "sistema microcomputadora-base" (mientras se determinan los componentes que conformen su arquitectura mínima).

El OBJETIVO de este trabajo es :
diseñar y construir un Sistema de Desarrollo, que provea al diseñador de una arquitectura fundamental de Microcomputadora (CPU (Microprocesador), Memoria RAM y EPROM, y Puertos I/O (Entrada/Salida)), ya probada y funcionando, para que a partir de su conocimiento y empleo, pueda fácilmente manejar y añadir, mediante la lógica de decodificación adecuada, los periféricos

requeridos por su diseño : (temporizadores (timers), controladores jerárquicos de interrupción, controladores de comunicación avanzados, coprocesadores numéricos, sintetizadores digitales de voz o imagen, convertidores A/D y D/A, memoria dinámica o más memoria RAM estática, EPROM o EEPROM y/u otra circuitería digital y/o analógica de interface), además de implantar en *software*, las herramientas necesarias para que el usuario pueda probar y depurar las secciones de *hardware* que vaya agregando sobre la infraestructura base del sistema, (el sistema debe ofrecer facilidades para que el usuario pueda escribir y ejecutar módulos de *software* que manejen o actúen sobre los módulos de *hardware*).

•

Así, el diseñador deberá poder obtener más rápidamente un prototipo armado y funcionando para ver si el instrumento de aplicación especifica que busca desarrollar es "factible" de realizarse. Dicho instrumento podrá ser dependiente del sistema de desarrollo para su operación y control, o bien, podrá eventualmente independizarse para constituir una aplicación autónoma.

•

El Sistema de Desarrollo, que a partir de ahora se designará como **SDM88-PC**, está basado en el microprocesador 8088 y en una microcomputadora tipo PC ó PS, IBM compatible. El sistema constará pues, de dos microcomputadoras trabajando en paralelo : una Microcomputadora basada en el µP 8088 (µC-Interface) y una Microcomputadora PC ó PS (µC-PC (PS)).

•

La µC-Interface es manejada por la µC-PC (PS) a través del puerto serie de la misma y el Software de control del SDM88-PC, mediante despliegues en forma de Menúes, proporciona al usuario distintas alternativas para visualizar, introducir, editar, manejar, organizar, almacenar y accesar información, y para operar directamente sobre los componentes de la µC-Interface (circuitos

integrados programables de soporte controlados por el microprocesador 8088 externo).

La Microcomputadora PC (PS) interviene activamente como administradora del *Hardware* externo (μC-Interface y/o circuitos de aplicación), canaliza comandos para efectuar acciones especificas y arbitra la comunicación entre μC-Interface y μC-PC (PS).

▪

El usuario puede guardar en disco o leer de disco programas en lenguaje de máquina del μP de la μC-Interface (en este caso 8088) mediante la μC-PC (PS), y de esta manera, cuenta con un medio de almacenamiento permanente y versátil para dichos programas, ya sea en disco flexible o duro, facilitándose así el desarrollo del *software* que fuese necesario para la μC-Interface en alguna aplicación especifica del sistema. Es decir, en general se aprovechan los recursos que ofrece la Microcomputadora PC (PS) : {despliegue en pantalla, interface de teclado, gran capacidad de almacenamiento (RAM y unidades de disco flexible y/o duro), extenso soporte de *software*, *hardware* y periféricos, etc.}.

▪

El SDM88-PC es pues, un sistema muy poderoso a costo relativamente bajo, considerando las ventajas que ofrece en cuanto a versatilidad, vigencia y facilidad de manejo como herramienta auxiliar en el desarrollo de instrumentos prototipo controlados digitalmente por μP, y que también puede aplicarse directamente, él mismo, en actividades de instrumentación, control y/o adquisición y procesamiento de datos, como parte constitutiva de un sistema mayor en la industria, en campo, en el laboratorio, etc. Una vez conociendo el principio de funcionamiento del sistema SDM88-PC, se puede crear y/o modificar el software de control, tanto del lado de la μC-Interface, como del lado de la μC-PC (PS), a fin de adaptarlo a los requerimientos del proyecto global de aplicación especifica.

▪

En el capítulo 1 se hace una descripción de las características y funciones del microprocesador 8088 de Intel Corp., ya que es parte esencial dentro de la configuración (Hardware) del Sistema Digital de Desarrollo SDM88-PC , (es el elemento principal de la μC-Interface). Se analizan pues, tópicos como la arquitectura interna del microprocesador 8088, sus especificaciones técnicas, la descripción de sus señales, organización de memoria, registros, modos de operación y de direccionamiento, manera de procesar interrupciones y conjunto de instrucciones.

En el capítulo 2 se describe la configuración en *Hardware* del Sistema Digital de Desarrollo SDM88-PC (Sistema de Desarrollo basado en el Microprocesador 8088 y en una microcomputadora tipo PC (o PS)) . Se muestra la simbiosis existente entre la "Microcomputadora PC (PS)" y la "Microcomputadora-Interface" basada en un microprocesador 8088 externo que administra su propia memoria y controla sus propios periféricos I/O. Se señala cómo el usuario puede operar sobre los componentes de la μC-Interface (circuitos integrados programables) , a través de la μC-PC (PS), haciendo uso de los recursos disponibles de este *Hardware* externo para el control de procesos o para la creación de instrumentos-prototipo concebidos para dar solución a una necesidad específica, y cómo estos sistemas de aplicación desarrollados a partir del SDM88-PC son controlados lógicamente, por programas (*Software*) creados por el usuario en la μC-PC (PS) y que posteriormente pueden ser transferidos (cargados) a RAM de la μC-Interface, para actuar sobre el *Hardware* de la aplicación a través del *Hardware* de la interface (con o sin la supervisión constante de la PC, según se requiera).

En el capítulo 3 se describe el *Software* de Control ó SISTEMA OPERATIVO del Sistema Digital de Desarrollo SDM88-PC .
Debido a que el sistema SDM88-PC se compone de dos microcomputadoras trabajando conjuntamente, la Microcomputadora-PC (PS) y la Microcomputadora-Interface (*Hardware*-8088-externo), en donde la primera controla a la segunda, se precisa de la existencia de dos programas de control ejecutándose en paralelo : uno residente en la memoria de la μC-PC (PS) { disco -▷ RAM } y otro en la memoria { EPROM } de la μC-Interface.
El *Hardware* conjunto del sistema es administrado por el *Software* de Control que, en unos casos actúa exclusivamente sobre los recursos de la μC-PC , y en otros casos establece un protocolo de comunicación entre la μC-PC (maestra) y la μC-Interface (esclava) para transferencia de comandos, código y datos a través de un canal de enlace serie. El Software de control del SDM88-PC está diseñado de tal manera que la parte residente en RAM de la μC-PC(PS) (después de ser cargada de disco), proporciona al usuario distintas alternativas para el control del sistema y para la operación directa sobre los componentes de la μC-Interface, a través de despliegues autoexplicativos en forma de Menúes, y empleando los recursos que ofrece la Microcomputadora PC (PS).
En este capítulo se describe pues, la estructura del Software contenido en la EPROM de la μC-Interface, los requerimientos y justificaciones en el diseño y desarrollo del programa de control en la μC-PC(PS) y se explican los distintos comandos (opciones de control, análisis y edición) que el usuario tiene a su disposición desde la μC-PC (PS).

En el capítulo 4 se describen las herramientas comunes involucradas en el desarrollo de programas en lenguaje Ensamblador, que pueden usarse en un sistema basado en μP (Microcomputadora). Se indica cómo pueden usarse programas

auxiliares para desarrollo de Software, desde la μC-PC (PS) para elaborar programas destinados a operar sobre el Hardware de la μC-Interface y de sus aplicaciones, en el sistema SDM88-PC.
Así, a través del conjunto de comandos con que cuenta el sistema **SDM88-PC** para operar sobre cualquier tipo de archivos binarios generados desde el sistema operativo MS-DOS de la μC-PC (PS) a través de alguno de estos programas de soporte para desarrollo de Software, comercialmente asequibles (Macro_Ensambladores, Debuggeres, Grabadores de EPROMs, etc.), el usuario podrá crear e introducir sus programas de aplicación de una manera más fácil y eficiente (mediante un editor de texto, empleando mnemónicos en vez de código hexadecimal para las instrucciones de μP), y de forma más ordenada y documentada, y minimizando la ocurrencia de errores en la generación e introducción de código (hex) para el Microprocesador. También se mostrará el Algoritmo o Ciclo de Desarrollo de un Programa, se establecerán las diferencias entre un archivo binario en formato {*.EXE} y en formato {*.COM}, y se incluirán tres ejemplos de programas-tipo para operar sobre el Hardware de la μC-Interface, indicando el proceso de generación del archivo binario adecuado, desde la μC-PC (PS), empleando productos comerciales ("Microsoft").

En el capítulo 5 se describe el uso de las microcomputadoras y sistemas de desarrollo basados en microprocesador, para el control de procesos o para la creación de instrumentos-prototipo de propósito específico. Se situará al SDM88-PC dentro del contexto de sistemas auxiliares para el diseño en instrumentación y para el control de procesos, indicando comparativamente sus características con respecto a equipos equivalentes disponibles en el mercado. Se hablará de las consideraciones básicas en el diseño de un sistema cualquiera, del balance *Hardware-Software* , y de algunos métodos o aproximaciones que pueden ayudar al desarrollo de un Prototipo, haciendo énfasis en la descripción de

las características y ventajas que para ello ofrece el Sistema de Desarrollo **SDM88-PC** . Finalmente se mostrará, como un ejemplo de aplicación concebido tomando como *Hardware* base o herramienta inicial de desarrollo al sistema **SDM88-PC** , la elaboración de un instrumento prototipo denominado :

" **Analizador Digital de Respuesta en Frecuencia** "

que permite conocer el comportamiento en Magnitud y Angulo de Fase de la Función de Transferencia de un sistema (circuito analógico) cualquiera, dentro de un rango de operación.

CAPITULO I

EL MICROPROCESADOR 8088

INTRODUCCION

En este capitulo se hace una descripción de las capacidades y funciones del microprocesador 8088 de Intel Corp., ya que es parte esencial dentro de la configuración (Hardware) del Sistema Digital de Desarrollo SDM88-PC .

La familia de microprocesadores 8086/88 de Intel se basa en una arquitectura (interna) de 16 bits y sus µP's miembros están diseñados para funcionar como el CPU (Central Processing Unit) en una Microcomputadora. El término "16-bits" significa que su Unidad Aritmética y Lógica (ALU), sus registros internos y gran parte de sus instrucciones están diseñados para trabajar con palabras binarias de 16 bits.

El microprocesador (µP) 8088 es una variante del µP 8086 de Intel, siendo la anchura del canal de datos (Data bus), la principal diferencia entre ambos. El µP 8086 tiene tanto un canal de datos interno de 16 bits, como uno externo también de 16 bits para interface con memoria y puertos I/O; mientras que, el µP 8088 tiene un canal de datos interno de 16 bits, pero un canal de datos externo de 8 bits. Ambos tienen la misma ALU, los mismos registros internos y el mismo conjunto de instrucciones. Debido a que el µP 8086 y el 8088 tienen la misma estructura interna, pueden utilizar los mismos componentes de soporte (periféricos) y el código escrito para cualquiera de los dos procesadores, se ejecutará en el otro sin cambios (son compatibles en *Software*). Esto permite un intercambio completo de programas entre sistemas de 8 y 16 bits.

La razón de la existencia en el mercado del µP 8088 con ese canal de datos externo (8 bits), es el establecimiento de una continuidad entre el µP 8086 con sus 16 bits en el Bus de datos

(estructura externa) y los antiguos procesadores de 8 bits fabricados por Intel : 8080 y 8085 . Esta continuidad es especialmente importante para aquellos que han desarrollado sistemas basados en este tipo de productos. El µP 8088, teniendo el mismo tamaño de canal de datos, puede remplazar a uno de esos primeros procesadores de 8 bits en un sistema ya existente, sin necesidad de hacer grandes cambios en él, aumentando tanto su capacidad y rendimiento en cuanto al número de operaciones realizadas (velocidad de procesamiento) y al conjunto de instrucciones que posee, así como su capacidad de almacenamiento de datos, a un costo muy bajo.

La diferencia en anchura del Bus de Datos entre el 8086 y el 8088, no implica que el primero sea dos veces más rápido que el segundo. Puesto que el µP 8088 ejecuta sus instrucciones obteniéndolas de una "Cola interna de instrucciones (bytes de código)" que es llenada tan pronto como exista una localidad vacía, es posible "traslapar" ciclos de ejecución (execution cycles) con ciclos de obtención-de-código-de-instrucción (instruction-fetch cycles). Ya que las instrucciones son ejecutadas tan pronto como son "extraídas" de la "Cola de instrucciones" y no tienen que esperar un ciclo de acceso a bus para ser leídas por el µP, se ejecutan tan rápido como lo harían en un µP 8086. Sin embargo, si los ciclos-de-dato son de 16 bits de longitud, en un 8088 tomarán dos ciclos de lectura a memoria, mientras que en un 8086 tomarían sólo uno; es decir, dado que el 8086 tiene un bus de datos (externo) de 16 bits, puede "leer datos de" o "escribir datos en" memoria y puertos I/O ya sea en bloques de 16-bits o de 8-bits en una sola operación, mientras que el 8088, por tener un bus de datos (externo) de 8 bits, sólo puede leer o escribir datos de/en memoria y puertos I/O, en bloques de 8 bits por vez; así, por ejemplo, para leer una palabra de 16 bits de dos localidades de memoria sucesivas, el 8088 tendrá siempre que realizar dos operaciones de lectura.

De esta manera, si una aplicación está marcadamente orientada a datos-byte, se desarrollará casi tan eficientemente con un μP 8088 como con un 8086; si la aplicación es de 16-bits por naturaleza, se realizará "menos bien" con un 8088, pero esto no implica que lo haga a la mitad de la velocidad de procesamiento que con un 8086.

El μP 8088 proporciona el apoyo del nivel de programación característico de un microprocesador de 16 bits y a la vez la ventaja de utilizar lógica de 8 bits en su electrónica (suficiente en la mayoría de los casos para aplicaciones de instrumentación, control y adquisición-procesamiento de datos).

En este capítulo se analizarán pues, tópicos como la arquitectura interna del microprocesador 8088, sus especificaciones técnicas, la descripción de sus señales, organización de memoria, registros, modos de operación y de direccionamiento, manera de procesar interrupciones y conjunto de instrucciones.

1.1 MICROPROCESADORES

Cuando una computadora es pequeña y barata, y se usa para el procesamiento de datos de acuerdo con programas realizados por el usuario, se le puede llamar "Microcomputadora" (μC). Cuando deja de ser un procesador de propósito general y se dedica al control de algún dispositivo, de acuerdo a un programa fijo, es mejor llamarlo un "Controlador". Otra importante distinción que conviene señalar, es que un controlador auxiliado por un Microprocesador (μP) es "dedicado", esto es, es programado para realizar una sola tarea, normalmente por el diseñador y no por el usuario, es decir, la programación está en forma permanente (*firmware*). Una Microcomputadora, sin embargo, es diseñada para ser programada por el usuario.

Un Microprocesador (μP) es un circuito integrado LSI o VLSI o un grupo de circuitos que realizan las operaciones aritméticas, lógicas y la lógica de control para las instrucciones y las secuencias de la ALU (Arithmetic-Logic Unit). El mismo μP, entonces, puede ser parte de una máquina maestra o instrumento y convertirse en su controlador, o puede ser un elemento de una Microcomputadora de propósito general.

El μP consta de una ALU, acumuladores y registros (que mantienen datos temporales para una rápida operación de los otros componentes del μP), y un Control Decodificador-Secuenciador (que interpreta los comandos (instrucciones o sentencias) del programa y envía órdenes detalladas, llamadas microinstrucciones, a cada uno de los componentes del μP, para que ellos realicen la operación deseada). En la mayoría de los casos el μP está contenido en un solo circuito, sin embargo, existen circuitos μP's

como los de la familia 2900 de *Advanced Micro Devices* (CPU's de 4 bits con ALU, multiplexores, secuenciadores, etc.), que pueden ser apilados juntos (conectados en paralelo) para formar procesadores de cualquier longitud de palabra (8,16,32,48,64 bits, etc.); estas unidades de 4 bits se conocen como Procesadores *Bit-Slice* y son circuitos que frecuentemente se usan para construir Minicomputadoras y no controladores, o para aplicaciones en que los μP's de propósito general (ej. Z-80, 6809, 8085, 8088/86, etc.) no son suficientemente rápidos o sus "sets" de instrucciones no se ajustan del todo a las necesidades del diseñador (el diseñador no sólo *custom*-diseña el hardware del CPU, sino que también *custom*-desarrolla el "set" de instrucciones para dicho hardware usando "microcódigo").

Algunas de las características distintivas de los μP son :

- Arquitectura
- Longitud de Palabra
- Conjunto ("set") de Instrucciones
- Organización de Memoria
- Tecnología
- Disipación de Potencia, encapsulado y otras consideraciones de hardware y software

De éstas, la Longitud de Palabra es la más obvia. Usualmente, la Longitud de Palabra determina la precisión de los datos, velocidad de cálculos (aunque no en forma directa) y costo de memoria, o los compromisos entre estos factores.

El primer μP disponible en el mercado (4004 de Intel), apareció en 1971, y tenía una ALU y Longitud de palabra de 4 bits que encajaba perfectamente en el formato BCD (así cada palabra puede representar un # decimal en un "display" numérico o en un teclado). Obviamente, el formato de 4 bits requiere de operaciones múltiples para todas las funciones, si se manejan

números más grandes que de un sólo dígito. Es posible para el μP de 4 bits lograr resoluciones más altas pero a expensas de repetir operaciones secuencialmente. Los μP de 8 bits son más apropiados para manipular datos alfanuméricos (como el código ASCII), y son ampliamente usados para comunicaciones de datos. Dado que un tamaño de palabra de procesamiento de 8 bits ofrece mayor resolución, existen muchas aplicaciones de control para las cuales estos μP's son apropiados. Para controladores de alta precisión, los μP de 16 bits alivian el problema del alcance de datos, dado que dos bytes hacen una palabra de 16 bits (word); este formato conduce por sí mismo a una manipulación de byte que es más eficiente en términos de espacio de memoria, tamaño de registros y velocidad de procesamiento.

Toda la información usada por el μP está almacenada en forma de palabras, que representan instrucciones de un programa o datos temporales. Las instrucciones del programa y los datos pueden estar mezclados en la memoria, y dado que ésta puede ser accesada aleatoriamente, la "localización" de estos entes no tiene significado. De hecho, el procesador no puede distinguir entre datos e instrucciones, excepto por el contexto, ya que ambos pueden tener diferente longitud. Una instrucción está dividida en código de operación y operando; el código de operación le dice al μP "qué hacer", y el operando u operandos son los datos o direcciones con los cuales trabajará la instrucción.

El nivel más bajo de microinstrucción dentro del control Decodificador-Secuenciador, consta de sólo cuatro acciones que pueden hacerse sobre los datos :

Sumar
Mover
Almacenar
Comparar

En unión con el microprograma de control del

Decodificador-Secuenciador, estas operaciones elementales pueden ser secuenciadas en macroinstrucciones más útiles y entendibles, las cuales conforman el conjunto de instrucciones del μP.

Generalmente estas instrucciones se dividen en las siguientes clases:

- Carga / Almacenamiento
- Aritméticas
- Lógicas
- Salto
- Corrimiento / Rotación
- Transferencia del control
- Entrada / Salida

ocasionalmente estas clases pueden estar combinadas, p. ej., la estructura del canal de Datos no distingue entre dispositivos externos y la memoria, asi "Carga/Almacenamiento" y "Entrada/Salida" comparten las mismas instrucciones.

Las instrucciones de "Carga" transfieren el contenido de localidades de memoria hacia registros seleccionados, mientras que las de "Almacenamiento" transfieren de registro hacia memoria.

Las operaciones "Aritméticas" se realizan comúnmente sobre dos números : uno de ellos puede estar en un registro y el otro en memoria, o ambos pueden estar en registros.

Las operaciones "Lógicas" incluyen AND, OR, XOR, NOT (Complemento a Uno y Complemento a Dos) sobre los datos (operandos).

Para realizar una instrucción de "Salto" se requiere efectuar primero una operación aritmética o lógica, colocando después una instrucción de bifurcación que verifica el resultado de la operación anterior contra alguna condición (tal como "cero" o "negativo"). Si la condición es satisfecha, el programa salta a

la dirección que se indica, de lo contrario el salto es ignorado y se continúa la ejecución con la siguiente instrucción en turno. Un Salto también puede ser Incondicional y siempre ser ejecutado.

Las operaciones de "Corrimiento" involucran la transferencia de todos los bits de una palabra, uno o más lugares hacia la derecha o hacia la izquierda. Si existe un punto binario implícito en cualquier parte de la palabra, el corrimiento es equivalente a multiplicar (izq.) o a dividir (der.) por 2. Existen diversas posibilidades de instrucciones de corrimiento. Un corrimiento lógico pierde bits en el extremo hacia el cual la palabra es recorrida mientras se adicionan ceros al extremo vacío. La rotación implica que el bit perdido en un extremo es añadido en el otro extremo. La operación de corrimiento puede o no incluir el bit de acarreo (carry). Un acarreo aritmético involucra reglas especiales para el bit de signo (MSbit).

Actualmente no existe una estandarización de instrucciones para μP's, pues cada fabricante crea su propio conjunto ("set") de instrucciones. Para propósitos de control, el μP debe considerarse como uno de los muchos medios de implementar la lógica requerida. Los factores que deben tomarse en cuenta incluyen : los requerimientos del trabajo (velocidad, precisión), encapsulado (medio ambiente, disipación de potencia), volumen que será producido, tiempo de desarrollo y costo, flexibilidad necesaria, complejidad y conocimientos teóricos sobre el problema, habilidad y conocimiento del personal de desarrollo.

Conforme los diseñadores encuentran más aplicaciones para los μP's, se ha presionado a la industria electrónica a desarrollar dispositivos con nuevas y óptimas arquitecturas y características para realizar ciertos tipos de tareas. En respuesta a estas necesidades la evolución-μP ha tomado 3 direcciones : Controlador dedicado, Procesador Bit-Slice y Procesador de propósito general.

1.2
EL MICROPROCESADOR 8088 : ESPECIFICACIONES TECNICAS

El microprocesador 8088 de Intel Corp., es un circuito integrado fabricado con tecnologia de alto funcionamiento o rendimiento (performance) Metal-Oxido-Semiconductor (HMOS) { N-channel, depletion load, silicon gate technology } con aproximadamente 29000 transistores. Está encapsulado en un paquete de 40 terminales, algunas de las cuales manejan dos señales dependiendo del modo de operación (Mínimo o Máximo) en que se habilite al µP.

El 8088 tiene atributos tanto de un µP de 8 bits como de uno de 16 bits. El canal de datos (Data Bus) tiene una amplitud de 8 bits y se encuentra multiplexado con el canal de direcciones (Address Bus), razón por la cual las terminales se identifican por AD7 (MSbit) a AD0 (LSbit). Por el término "multiplexado" debe entenderse, el uso del mismo grupo de líneas para enviar conjuntos de señales diferentes, pero en periodos de tiempo distintos. Ya demultiplexado, el canal de direcciones consta de 20 líneas (bits), que le proveen de la capacidad de direccionar 1 MByte de memoria; además puede direccionar 64 kBytes de puertos I/O (Entrada/Salida) con 16 líneas: A15-A0, ya que durante operaciones I/O las terminales A19-A16 permanecen siempre en "0-lógico".

Estas son las principales caracteristicas del µP 8088 :

* Microprocesador (CPU) : 8088 (INTEL) HMOS { iAPX 88/10 }

  + Frecuencia de Reloj : 4.77 [MHz] (8088)
    ú 8 [MHz] (8088-2)
  + Ancho de banda del Bus : 2 [Mbits/s]

+ Arquitectura interna : 16 bits
+ Bus (canal) de Datos : 8 bits
(interface externa)
+ Capacidad de direccionamiento
directo de memoria : 1 MByte

2^{20} = 1048576 bytes

20 líneas de dirección : A0-A19 .
{A0-A7} están multiplexadas con Bus de Datos {AD0-AD7}
No tiene manejo de memoria integrado ni memoria virtual

+ Capacidad de direccionamiento
directo de puertos I/O : 64 kBytes
+ Conjunto de 14 Registros internos de 16 bits c/u (con operaciones simétricas) :

Registros Aritméticos : 8 de 8-bits (4 de 16-bits)
{ AH, AL, BH, BL, CH, CL, DH, DL }
Apuntadores e Indices : 4 (16-bits)
{ SP, BP, SI, DI }
de Segmento : 4 (16-bits)
{ CS, DS, ES, SS }
de propósito general 8
{ idem que Aritméticos }
Apuntador de Instrucción : {IP} (16-bits)
Banderas (o Status) : {Flags}

+ Capacidad de realizar operaciones con Bytes, Palabras (Words) y Bloques de bytes o palabras (Blocks)
+ Capacidad de realizar operaciones aritméticas de 8 y 16 bits (signadas y no signadas), en notación binaria o decimal (BCD), incluyendo multiplicación y división
+ Capacidad de realizar operaciones lógicas de 8 y 16 bits
+ Capacidad de realizar operaciones con Strings
+ 24 Modos de Direccionamiento de Operando
+ Capacidad de realizar operaciones de "Acceso Directo a memoria" (DMA = Direct Memory Access)

+ Capacidad de manejo y atención a Interrupciones Mascarables (INT) {por Hardware y Software} y No Mascarable (NMI)
  Tiempo de respuesta a interrupciones : 8.6 [μs]
+ Soporta direccionamiento directo de puertos I/O (*Direct I/O*) y direccionamiento I/O mapeado desde memoria (*Memory-mapped I/O*).
+ Soporta manejo de Coprocesador numérico en Bus local (8087)
+ Conjunto de instrucciones : 117 instrucciones
+ Compatibilidad directa en software con μP 8086 y en general con 80188, 80186, 80286, 80386 (en modo NO-PROTEGIDO o extendido, es decir, en "modo de dirección real" {real address mode})
+ Compatibilidad hardware-periféricos con μP 8080/8085, 8086
+ Voltaje de operación : +5 V (±10%) con respecto a GND
+ Disipación de Potencia : 2.5 W
  Existe versión CHMOS disponible, con menor consumo de potencia : { Icc de operación = 10 mA/MHz , por tanto, 400 mW @ 8MHz (80C88-2) y 250 mW @ 5 MHz (80C88)}, { Icc standby = 500 μA }
+ Disponible en encapsulado DIP, Plástico y Cerámico con 40 terminales y para operación en rango de temperatura estándar (0 a 70°C) o extendido (militar)

Circuitos periféricos principales :

+ Generador de reloj : 8284A (82C84A)
+ Controlador de sistema (Bus) : 8288 (82C88)
+ Controlador de Interrupciones : 8259A (82C59A)
+ Controlador de DMA : 8237A (82C37A) , 82258
+ Contador/Temporizador programable : 8253 , 8254
+ Controlador para DRAM : 8207 , 8208
+ Controlador para disco (floppy) : 8272A , 82064 , 82072
+ Controlador CRT : 8275 (Intel) , MC6845 (Motorola)
+ Coprocesador matemático : 8087
+ Coprocesador para gráficas : 82786

+ Puertos paralelos (3 de 8 bits c/u) : 8255A (82C55A[],[-2])
+ Puerto serie USART : 8251A
+ Controlador de Comunicación Serie de múltiple protocolo (Multiple Protocol Serial Controller) : 8274
  Comunicación síncrona : Bisync,
  HDLC (High level Data Link Control)
  SDLC (Synchronous Data Link Control
  y asincrona
+ Circuitos para demultiplexar Datos de Direcciones :
  + Address Latch (ulilliza la señal ALE del µP)
    8282 / 8283 (Intel) ó 74LS373 (S,LS,HC) (TI)
  + Data flow (Tx/Rx) (utiliza la señal DT/$\overline{R}$ del µP)
    8286 / 8287 (Intel) ó 74LS245 (S,LS,HC) (TI)
    Data Bus Transceiver

+ etc.

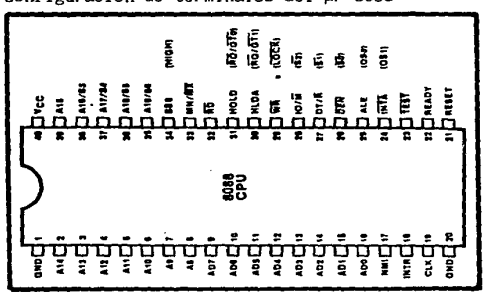
. . . . . .

En la (*figura* 1-1) se muestra la Configuración de terminales del Microprocesador 8088

. . . . . .

(*figura* 1-1)

Configuración de terminales del µP 8088



1.3 MODOS DE OPERACION

El μP 8088 puede ser configurado para trabajar en cualquiera de sus dos modos de operación posibles :

+ Modo de Sistema Mínimo y
+ Modo de Sistema Máximo

El modo MINIMO es seleccionado cuando se aplica un "1-lógico" (high) a la terminal de entrada { MN/$\overline{MX}$ } (pin 33). Este modo es utilizado para trabajar en sistemas pequeños, donde el procesador proporciona todas las líneas de control de los canales para el manejo de memoria y dispositivos periféricos. En este modo los pines de interface del μP 8088 son compatibles con los de un μP 8085; entonces, el 8088 puede conectarse directamente con cualquiera de los dispositivos de soporte de la familia de μP 8085. En el modo Mínimo no se necesita de un circuito (chip) "controlador de Canal (Bus)", puesto que los comandos-para-control-de-canal son decodificados y están disponibles directamente del 8088. Tampoco es soportada la interface "solicitud/concesión" de canal (*Request/Grant*) del 8088, y por tanto, no es posible conectar procesadores en paralelo ni coprocesadores, como el coprocesador matemático 8087.

El modo MAXIMO es seleccionado cuando se aplica un "0-lógico" (low) a la terminal de entrada { MN/$\overline{MX}$ } (pin 33). Este modo se utiliza para trabajar en sistemas grandes con más de un procesador, en donde es necesario tener un IC "Controlador o Arbitrador de Canal (Bus)" como el 8288 de Intel. Las microcomputadoras PC y PS (IBM o compatibles) trabajan en este modo de operación.

Dependiendo del modo de operación, algunas terminales del procesador cambian el significado de la señal que manejan.

A) Señales comunes para ambos modos de operación :

Nombre	F u n c i ó n	T i p o
AD7-AD0	Canal Direcciones/Datos	Bidireccional 3-Estados
A15-A8	Canal Direcciones	Salida 3-Estados
A19/S6 - A16/S3	Canal Direcciones/Estado	Salida 3-Estados
$\overline{RD}$	Control de Lectura	Salida 3-Estados
READY	Control de Estado de Espera	Entrada
$\overline{TEST}$	Espera bajo el control de prueba ("Wait for test" instruction)	Entrada
RESET	Reinicio (de ejecución)	Entrada
NMI	Interrupción NO Mascarable	Entrada Disparada por flanco (L-H) (edge triggered input)
INTR	Solicitud de Interrupción (Mascarable)	Entrada Disparada por nivel (High) (level triggered input)
MN/$\overline{MX}$	Control de Modo (Mínimo/$\overline{Máximo}$)	Entrada
CLK	Reloj del sistema Duty cycle = 33 %	Entrada
Vcc	Fuente de alimentación + 5 V (±10%)	Entrada
GND	Nivel de Referencia (Tierra digital)	Entrada

B) Señales exclusivas de modo Mínimo :

Nombre	F u n c i ó n	T i p o
IO/$\overline{M}$	Control de Acceso Puerto_IO / $\overline{Memoria}$	Salida 3-Estados
$\overline{WR}$	Control de Escritura	Salida 3-Estados
$\overline{INTA}$	Reconocimiento de Interrupción	Salida
ALE	Habilitación de Captura de direcciones	Salida
DT/$\overline{R}$	Control de Transmisión/$\overline{Recepción}$ de Datos	Salida 3-Estados
$\overline{DEN}$	Habilitación de Datos	Salida 3-Estados
HOLD	Solicitud de "Hold" (otro dispositivo solicita posesión del bus local)	Entrada
HLDA	Reconocimiento de "Hold"	Salida
$\overline{SSO}$	Estado (Current bus cycle status line)	Salida

C) Señales exclusivas de modo Máximo :

Nombre	F u n c i ó n	T i p o
$\overline{S2}$, $\overline{S1}$, $\overline{S0}$	Estado del Ciclo de Canal	Salida 3-Estados
$\overline{RQ/GT0}$, $\overline{RQ/GT1}$	Control de prioridad del Canal local (Request/Grant)	Bidireccional
$\overline{LOCK}$	Control de la prioridad del Canal (Bus)	Salida 3-Estados
QS1 , QS0	Estado de la Cola de Instrucciones	Salida

1 . 4

ARQUITECTURA INTERNA DEL MICROPROCESADOR 8088

El µP 8088 está dividido en dos secciones funcionales independientes :

Unidad de Interface de Canal

{ Bus Interface Unit } ..... { BIU }

Unidad de Ejecución

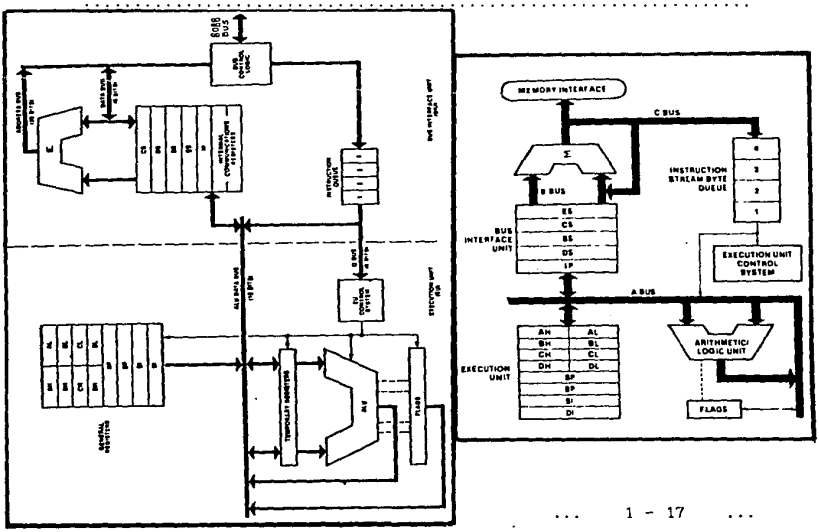
{ Execution Unit } ..... { EU }

La división del trabajo entre estas dos unidades incrementa la velocidad de procesamiento.

.....

En la { *figura* 1-2 } se muestra el Diagrama de bloques funcional del Microprocesador 8088 (Arquitectura Interna)

.....



Ambas unidades operan asíncronamente para darle al µP 8088 un mecanismo de captura y de ejecución de instrucciones al mismo tiempo.

1) UNIDAD DE INTERFACE DE CANAL
{ Bus Interface Unit } = { BIU }

Maneja todos los procesos de transferencia de Datos y Direcciones en los canales (buses) para la Unidad de Ejecución (EU).

La BIU realiza las siguientes funciones :

+ Captura de instrucciones mediante el método llamado de estructura tubular o por "Cola" ("Pipeline") : "Fetch" de instrucciones de la memoria.
+ Captura y almacenamiento de operandos :
  Lectura de datos de Memoria y Puertos I/O
  Escritura de datos en Memoria y Puertos I/O
+ Generación de direcciones para "sacarlas por" el Canal (Bus) de Direcciones.
+ Control del Canal (Bus) del sistema.

Para implementar estas funciones, la BIU consta de los siguientes elementos funcionales :
Cola de Instrucciones, Registros de Segmento, Apuntador de Instrucción, Registros de comunicación interna, Sumador de Dirección y Lógica de control de canal.

COLA DE INSTRUCCIONES : (Instruction Queue)

Para aumentar la velocidad de ejecución de un programa, la BIU realiza la precaptura (preFETCH) de 4 bytes de código-de-instrucción tomándolos de memoria (por adelantado) y los almacena en un grupo de registros FIFO (*First-In--First-Out*) para uso de la EU. Este grupo de 4 registros se denomina "Cola de Instrucciones", (cabe señalar que en el µP 8086 la Cola de

Instrucciones cuenta con 6 registros (bytes) en vez de 4 como sucede con el μP 8088). La BIU puede estar capturando bytes de instrucción mientras la EU está decodificando una instrucción o ejecutando una instrucción que no requiera uso de los buses del sistema. Cuando la EU está lista para su siguiente instrucción, simplemente la lee de la Cola en la BIU. Esto es más rápido que el proceso de mandar una dirección a la memoria del sistema y esperar a que ésta responda proporcionando el o los siguientes bytes de código-de-instrucción.

Excepto en los casos de instrucciones JUMP y CALL, en que la Cola debe ser vaciada de golpe (borrada) y recargada con bytes tomados a partir de una nueva dirección, este esquema de precaptura-y-cola (prefetch-and-queue) incrementa sustancialmente la velocidad de procesamiento. El proceso de "Captura-de-siguiente-instrucción mientras se ejecuta la instrucción presente" se denomina "Traslape" o "*Pipelining*" (μP con arquitectura "Pipeline").

Siempre que exista uno o más bytes de la Cola vacíos y si al mismo tiempo, la EU no está solicitando una lectura o escritura de operandos desde/en memoria o puerto I/O, la BIU está libre para la precaptura de la próxima instrucción secuencial. Si la Cola está llena y la EU no está solicitando acceso a operandos a memoria, la BIU realiza ciclos de canal (bus). Si la BIU está ocupada en el proceso de captura de una instrucción cuando la EU solicita una lectura o escritura de operandos, la BIU primero completa el ciclo de canal de captura de instrucción y después inicia el ciclo de lectura/escritura del dato.

REGISTROS DE SEGMENTO : { CS, DS, ES, SS }

* Se describirán posteriormente.

REGISTRO APUNTADOR DE INSTRUCCION : { IP }

* Se describirá posteriormente.

SUMADOR DE DIRECCION :

Su labor consiste en generar la "Dirección Física" que es enviada por el Bus de Direcciones en un momento dado.

LOGICA DE CONTROL DE CANAL :

La BIU también es responsable de generar las señales de control de Bus, tales como aquellas para la lectura o escritura de/en memoria o dispositivos de Entrada/Salida (puertos I/O).

2) UNIDAD DE EJECUCION
{ Execution Unit } = { EU }

La EU realiza las siguientes funciones :

+ Indica a la BIU de dónde capturar instrucciones o datos
+ Decodifica instrucciones
+ Ejecuta instrucciones

La EU extrae instrucciones de la parte alta de la Cola de la BIU, las decodifica, si es necesario genera direcciones para los datos y solicita a la BIU que realice los ciclos de canal de lectura o escritura; finalmente, realiza la operación especificada por la instrucción sobre los operandos. Durante la ejecución de una instrucción, la EU verifica las banderas de estado y control y las actualiza en base a los resultados de la ejecución de dicha instrucción. Si la Cola está vacia, la EU espera a que el próximo byte de instrucción sea capturado y recorrido a la parte alta de la Cola. Cuando la EU ejecuta una instrucción de salto (JUMP o CALL), la BIU automáticamente borra la Cola de instrucciones y comienza a capturar instrucciones (bytes de código ejecutable) desde esta nueva localidad.

Para implementar estas funciones, la EU consta de los siguientes elementos funcionales :

Unidad Aritmética y Lógica (ALU), Decodificador de Instrucción, Lógica de control de la Cola de instrucciones y registros temporales, Registro de Banderas o Estado, Registros de propósito general, Registros Apuntadores y Registros de Indice.

CIRCUITERIA DE CONTROL, DECODIFICADOR DE INSTRUCCION y ALU :

La EU contiene :

+ Circuitería de Control que dirige las operaciones internas.
+ Decodificador que traduce instrucciones (capturadas de memoria), en una serie de acciones que realiza la EU.
+ Unidad Aritmética y Lógica (ALU) de 16 bits que puede Sumar, Restar, AND, OR, XOR, incrementar, decrementar, complementar y efectuar corrimientos/rotaciones sobre números binarios.

REGISTRO DE BANDERAS O ESTADO (FLAGS) :

- Se describirá posteriormente.

REGISTROS DE PROPOSITO GENERAL , REGISTROS APUNTADORES y REGISTROS DE INDICE : { AX,BX,CX,DX } , { SP,BP } y { SI,DI }

- Se describirán posteriormente.

1 . 5
DESCRIPCION DE SEÑALES DEL MICROPROCESADOR 8088

Para ambos modos de operación, las señales del μP 8088 se pueden agrupar de la siguiente manera :
Alimentación, Reloj, Control y Estado, canal de Datos y canal de Direcciones.

Para la Alimentación se requiere un voltaje de entrada de +5 Volts (Vcc) en la terminal 40 y el nivel de referencia (GND) en las terminales 1 y 20.

La señal de Reloj @ 8 ó 4.77 [MHz] y con ciclo de trabajo de 30%, proviene de un IC 8284A y se dirige al pin 19 del 8088 para proveer la información básica de temporización al μP.

La terminal MN/$\overline{MX}$ habilita al procesador en modo Mínimo o en modo Máximo, si se conecta a +5V ó a GND, respectivamente.

En el 8088 existen 20 lineas de direcciones, divididas en tres grupos como sigue :

- AD7 - AD0 :
  Estas 8 señales (pines 9 a 16) se usan para transmitir información a memoria y puertos I/O en cada ciclo de Bus. Se encuentran multiplexadas en el tiempo : presentan los bits de dirección (Bus de Dirección) (A7 (MSbit) a A0 (LSbit)) al inicio del ciclo de bus (T1), y después, dentro del mismo ciclo, estas lineas son usadas como Bus de Datos del μP (T2,T3,Tw y T4).
- A15 - A8 :
  Estas 8 lineas de señal (pines 2 a 8 y 39) entregan los bits de dirección A8 a A15 para memoria y puertos I/O.
  Estas lineas no están multiplexadas y permanecen estables durante todo el ciclo de bus (T1-T4). No necesitan ser "latcheadas" por ALE para ser válidas.
- A19/S6 - A16/S3 :
  Al inicio de cada ciclo de bus (ciclo de máquina) (T1), estas lineas de señal (pines 35 a 38), proveen los 4 bits más significativos del Bus de Direcciones (A19-A16) para operaciones de Memoria. Durante operaciones I/O, estas lineas permanecen "Low". Durante el resto del ciclo, (T2,T3,Tw y T4), estas mismas lineas presentan información de estado

(status) interno del 8088. S6 permanece siempre "Low". El estado de la bandera de habilitación de interrupción (IF), es el bit S5, y es actualizado al inicio de cada ciclo de reloj. Los bits {S4,S3} se decodifican para identificar cuál de los registros de segmento fue usado para generar la dirección física que se envió por el canal de direcciones durante el ciclo de bus actual :

S4	S3	Registro de Segmento usado
0	0	EXTRA (Alternate Data Seg.)
0	1	STACK Segment
1	0	CODE Segment
1	1	DATA Segment

Los bits de estado $\overline{S2}, \overline{S1}$ y $\overline{S0}$ sólo son accesibles en modo máximo y son usados por el controlador de Bus; sirven para informar el tipo de operación que está efectuando el µP de acuerdo a la siguiente tabla :

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	F u n c i ó n
0	0	0	Reconocimiento de Interrupción
0	0	1	Lectura de puerto I/O
0	1	0	Escritura en puerto I/O
0	1	1	Paro (Halt)
1	0	0	Búsqueda de instrucción (Fetch)
1	0	1	Lectura de memoria
1	1	0	Escritura en memoria
1	1	1	No hay ciclo de bus (pasivo)

0 = Low ; 1 = High

La línea de salida $\overline{RD}$ indica que el procesador está realizando una operación (ciclo) de lectura a memoria o a puerto I/O, dependiendo del estado de la terminal IO/$\overline{M}$. Esta señal se usa para leer dispositivos que residen en el Bus local del µP.

La señal de READY es una linea de entrada proviniente de los dispositivos externos (memoria o dispositivos I/O) y funciona como un reconocimiento (proporcionado por estos dispositivos) para indicar que la transferencia de datos ha sido completada. La señal RDY de memoria o puerto I/O, es sincronizada por el generador de reloj 8284A para formar READY.

La señal de RESET coloca en un estado inicial la Cola de instrucciones y ciertos registros (de indice, apuntador de instrucción, de Segmento). Esta señal de entrada al µP, debe activarse "High" (al menos durante 4 ciclos de reloj) y está sincronizada internamente con el reloj del procesador. El µP permanece inactivo con todas sus salidas 3-Estados flotadas (OFF) durante todo el tiempo que RESET se encuentre en estado "High". La operación normal del µP se reinicia cuando RESET vuelve al estado "Low". Pin (21).
Cuando el µP entra a un estado de RESET, se realizan los siguientes eventos :

1) El registro de banderas (Flags) es borrado (clear=0000h)

2) El contenido del registro de Segmento de Código (CS) es puesto en FFFFh .

3) Los registros de Segmento de Datos (DS), Extra (ES), Stack (SS) y el apuntador de instrucción (IP) son puestos a "cero" (clear=0000h).

4) Una vez que la condición de RESET es removida (Pin 21 vuelve a "Low"), la primera instrucción ejecutada por el µP, es aquella que se encuentra en la dirección física de memoria "FFFF0h" : (CS:IP) = (FFFF:0000).
Nota :
Normalmente esta instrucción reside en EPROM del sistema y es un Salto JMP incondicional a otra área de memoria, ya que en esta zona sólo se cuenta con 16 bytes para escribir código ejecutable (FFFF0 a FFFFFh).

La señal de RESET para el sistema es generada a partir de una RED R-C con constante de tiempo adecuada y es pasada a través de un IC 8284A para ser sincronizada con el reloj del sistema; esta señal de salida es para μP y todos los dispositivos que requieran RESET.

Las terminales (entradas) NMI (Interrupción No Mascarable) (pin 17) y INTR (petición de Interrupción (mascarable)) (pin 18), son parte del sistema de interrupciones del 8088. Una señal apropiada aplicada a una de estas entradas, ocasionará que el 8088 interrumpa el programa que está ejecutando y vaya a ejecutar una rutina especificada. Una vez terminada esta rutina, el μP regresará a correr el programa interrumpido a partir de la siguiente instrucción a la última ejecutada antes de la solicitud de interrupción. Las interrupciones generadas por INTR pueden desactivarse por programación.

En el modo Mínimo se generan las siguientes señales de control de estado :

- $\overline{SSO}$ (estado SO)
- HLDA (reconocimiento de HOLD)
- $\overline{WR}$ (control de Escritura) : indica que el μP está realizando un ciclo de Escritura a memoria o puerto I/O
- IO/$\overline{M}$ (selección de memoria/puerto IO) : distingue un acceso a memoria de un acceso a puerto IO
- DT/$\overline{R}$ (selección de transmisión/recepción de datos) : para controlar la dirección de flujo de datos a través de un Transceiver (8286/8287 ó 74LS245) (Demultiplexaje del Bus de Datos para el sistema)
- $\overline{DEN}$ (datos disponibles) : salida de habilitación para el Transceiver que se activa en cada ciclo de acceso a memoria o puerto IO y ciclo $\overline{INTA}$.
- ALE (habilitación para captura de direcciones {"latch"}) : se utiliza en el demultiplexaje del Bus de Direcciones para el sistema (a través de un 8282/8283 ó un 74LS373)

$\overline{INTA}$ ("strobe" para el reconocimiento de interrupción (mascarable))

Combinando el estado lógico de las señales IO/$\overline{M}$, DT/$\overline{R}$ y $\overline{SSO}$, los "ciclos de Bus" del μP 8088 pueden ser decodificados de la siguiente manera :

IO/$\overline{M}$	DT/$\overline{R}$	$\overline{SSO}$	C i c l o s d e B u s
0	0	0	Acceso a Código (CS)
0	0	1	Lectura de Memoria
0	1	0	Escritura en Memoria
0	1	1	No operación (estado pasivo)
1	0	0	Reconocimiento de interrupción
1	0	1	Lectura de puerto I/O
1	1	0	Escritura en puerto I/O
1	1	1	Paro (estado "Halt")

ESTADO DE "HOLD" :

Cuando el 8088 es llevado a este estado, todas las señales de salida 3-Estados son flotadas (OFF : High Z). El estado de HOLD del 8088 se usa para habilitar la lógica de acceso directo a memoria (DMA) y para deshabilitar los μP's 8088 cuando más de un CPU accesa el mismo canal (bus) del sistema en una configuración Multi-CPU (procesadores en paralelo) (ej. un 8088 y un 8087 trabajando en paralelo).

En el modo Mínimo, el 8088 tiene una entrada de solicitud HOLD y una salida de reconocimiento HLDA. Después de recibir una entrada "High" en HOLD, el μP completa la ejecución de la instrucción en su ciclo de bus actual antes de entrar en estado de HOLD y manda una salida "High" por la terminal HLDA, flotando simultáneamente sus líneas de bus local y de control, para permitir que otro procesador o dispositivo tome posesión del bus del sistema. El 8088 muestrea la entrada HOLD en la transición L-H de la señal CLK (por tanto, HOLD debe ser estable cuando CLK esté haciendo esta transición). El estado HOLD permanece hasta

que el nivel lógico aplicado a esta entrada va "Low" de nuevo; la salida HLDA es forzada a "Low" también.

ESTADO DE "HALT" :

El μP 8088 entra en un estado de HALT después de la ejecución de la instrucción HALT. En este estado, ninguna señal del bus o de control es flotada (se envía información indefinida por el canal de datos y direcciones). No se ejecuta ningún ciclo de bus mientras el 8088 permanezca en HALT. Esta condición es terminada sólo por una interrupción o por un RESET (hardware).

ESTADO DE "$\overline{\text{TEST}}$" :

Esta terminal de entrada (pin 23), es muestreada (examinada) por la instrucción "Wait for Test". Si la entrada $\overline{\text{TEST}}$ es "Low", la ejecución continua (operación normal del 8088); si es "High", el 8088 espera en un estado de espera (ocio) "idle" hasta que vuelve al estado "Low".

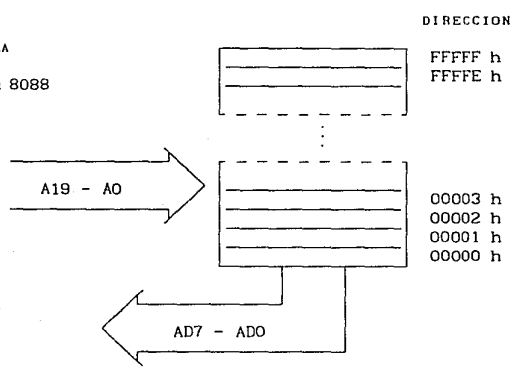
1.6 ORGANIZACION DE LA MEMORIA CON EL MICROPROCESADOR 8088

El μP 8088 cuenta con un Canal (bus) de Direcciones de 20 bits, igual que el 8086, pero la memoria no está dividida en 2 bancos (Localidades de Memoria Pares y Nones), como en el 8086. El 8088 tiene un Bus de Datos de sólo 8 bits (AD7-AD0) (a diferencia del 8086 que tiene 16 bits (AD15-AD0)), por lo que todos los dispositivos de memoria y puertos I/O del sistema-8088 deben conectarse a esas 8 líneas. Entonces la memoria para el μP

8088 funciona como un único banco de 1048576 bytes, es decir, está organizada como un arreglo lineal de 1 MByte, direccionado desde { 00000h } hasta { FFFFFh }.

{ *figura* 1-3 }

ESTRUCTURA DE LA MEMORIA EN UN MICROPROCESADOR 8088



Esta estructura de un único banco, significa que el 8088 no puede leer una palabra (word) de ó escribir una palabra en memoria en un solo ciclo de máquina como el 8086. El 8088 sólo puede leer o escribir bytes, por lo que requerirá de 2 ciclos de máquina para leer o escribir una palabra (1 Word = 2 Bytes). Las líneas de dirección A19-A0, a través de algunos decodificadores, seleccionan el byte deseado de memoria. El 8088 no produce la señal $\overline{BHE}$ del 8086, pues no la necesita.

La mayoría de los dispositivos de Memoria y Puertos I/O disponibles, están diseñados para µP's de 8 bits que tienen Buses de Datos de 8 bits; así el 8088 fue diseñado con ese mismo Bus de Datos a fin de poder conectarse más fácilmente a esos dispositivos. Por ejemplo, en un sistema-8088 un simple decodificador 3X8 74LS138 puede usarse para seleccionar puertos

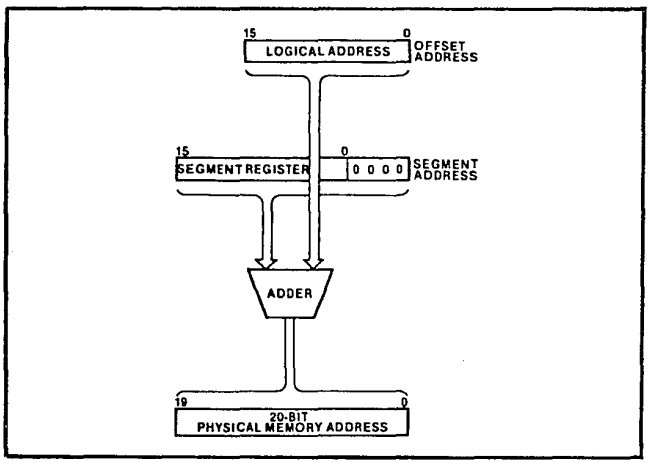
I/O; en cambio, en un sistema-8086, se requiere un decodificador de puertos más complejo (p.ej. una PROM 3625 con un algoritmo de selección pregrabado por el diseñador), ya que deben tenerse en cuenta los estados lógicos de las líneas A0 y $\overline{BHE}$.

La memoria está dividida lógicamente en 4 segmentos (Code, Data, Extra-data, Stack), de hasta 64 kBytes cada uno y conformando (cada uno de ellos) una región cuyos límites pueden ser cualesquiera factores exactos de 16 bytes . Es decir, en cualquier momento, el área de almacenamiento-operación del programador está limitada a una región de 65536 bytes, pero éste tiene la capacidad de mover esta región a cualquier frontera factor-de-16-bytes dentro del espacio de 1048576 bytes. Esto lo logra manipulando el contenido de los "registros de Segmento". El valor cargado en un registro de Segmento es usado para localizar, dentro del espacio de 1 MByte, la región de 64 kBytes en que las instrucciones o datos del 8088 operan en un momento dado. Como estos registros tienen 16 bits, se pueden especificar 65536 regiones diferentes de 64 kBytes, dentro del espacio de 1 MByte. La Dirección Física o real de memoria se forma recorriendo (Shift) el contenido del registro de segmento, 4 bits hacia la izquierda (agregando 4 ceros en el extremo derecho) y sumando ese número, ahora de 20 bits, a una dirección de 16 bits (Offset) generada por instrucción (o proviniente de un registro de índice, base o apuntador). Esto es, el contenido de algún registro de Segmento se multiplica por 10h para formar la dirección de Segmento (Segment Base address) y a esta cantidad se le suma otra de 16 bits llamada dirección Lógica o desplazamiento (Offset address) para formar la dirección Física de 20 bits, que será sacada por el Bus de direcciones {A19-A0} para seleccionar localidades dentro de la memoria instalada en el sistema-μP.

En la { *figura* 1-4 } se muestra el mecanismo de generación de Dirección de Memoria con el Microprocesador 8088

{ *figura* 1-4 }

Mecanismo de generación de Dirección de Memoria del μP 8088



Existen pues 4 registros de Segmento : uno se usa para direccionar código (CS), otro para accesar datos a los que hace referencia el código (DS), otro para direccionar referencias de datos en el código a través del Stack (SS) y otro más es un segmento alternativo para datos o Segmento Extra (ES), que se usa tipicamente en operaciones de "movimiento de datos en bloque" donde la operación va a verificarse entre dos regiones diferentes de 64 kBytes y se requiere tanto de un registro de segmento Fuente como de un registro de segmento Destino.

Estos 4 registros de segmento pueden todos apuntar a diferentes regiones de 64 kB dentro del espacio de 1 MB. Así, cuando los registros de Segmento han sido inicializados, las

instrucciones ejecutables del programa ven un espacio-de-código de 64 kB, un espacio-de-datos de 64 kB, un espacio-extra-de-datos de 64 kB y un espacio-de-Stack de 64 kB. Si en un momento dado el programa necesita hacer referencia a una zona fuera de estos espacios, debe primero manipular el registro de Segmento apropiado. Cabe señalar que los segmentos pueden traslaparse entre sí o aún apuntar todos al mismo espacio de 64 kB.

Al valor del Segmento se le denomina "Dirección Base" y a la dirección o desplazamiento "dentro" de un Segmento se le llama "Dirección Offset". Así, cualquier dirección dentro del espacio de direccionamiento de 1 MB, puede identificarse especificando una dirección Base y un Offset. Nótese que pueden generarse muchas combinaciones "Base-Offset" para especificar una misma dirección de memoria.

Así pues, todas las referencias a memoria son hechas con respecto a "direcciones base" contenidas en registros de Segmento de alta velocidad.

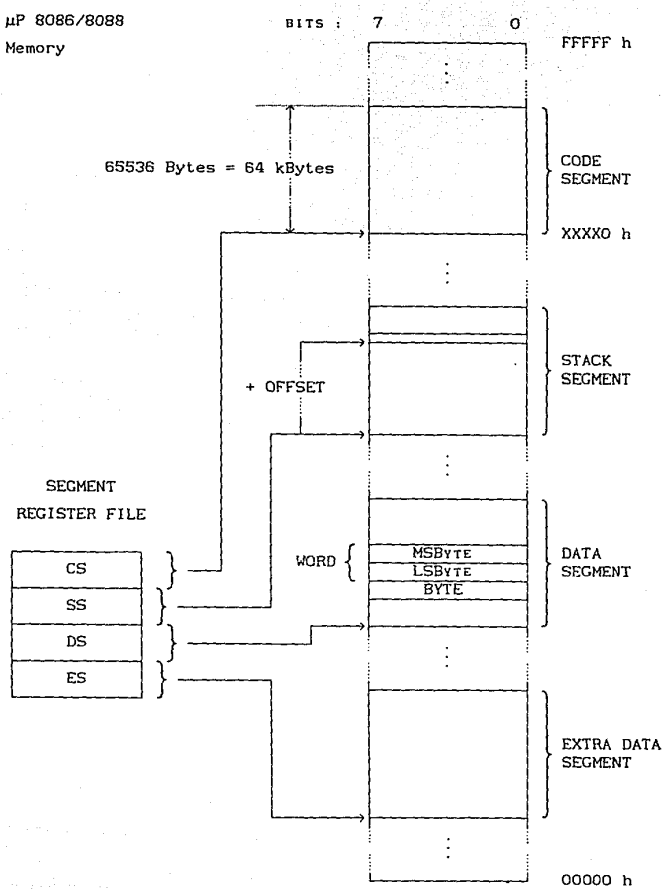
Los tipos de Segmento son escogidos según las necesidades de direccionamiento de los programas.

Toda la información en un tipo de Segmento comparte los mismos atributos lógicos (ej. código o datos).

Estructurando la memoria en áreas relocalizables de características similares y seleccionando automáticamente los registros de segmento apropiados, los programas resultan más cortos, más rápidos y más modulares (estructurados).

.....

En la { *figura* 1-5 } se muestra la Organización de la memoria para el Microprocesador 8088


µP 8086/8088
Memory
BITS : 7 0
FFFFF h
65536 Bytes = 64 kBytes
CODE SEGMENT
XXXX0 h
+ OFFSET
STACK SEGMENT
SEGMENT REGISTER FILE
CS
SS
DS
ES
WORD
MSByte
LSByte
BYTE
DATA SEGMENT
EXTRA DATA SEGMENT
00000 h

El registro de Segmento es automáticamente seleccionado de acuerdo a las reglas de la siguiente tabla :

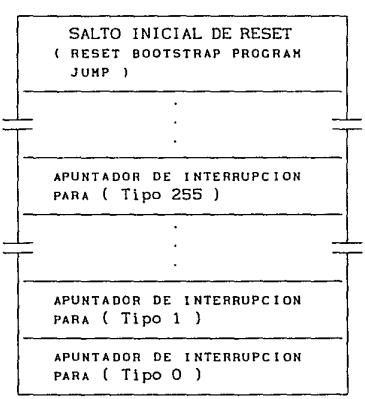
Necesidad de Referencia a Memoria	Registro de Segmento usado	Regla de Selección de Segmento
Instrucciones	CODE (CS)	Selección automática para todas las precapturas de Instrucción (prefetch).
Stack	STACK (SS)	Todas las operaciones PUSH y POP con el Stack. Referencias a memoria relativas al registro base BP, excepto referencias a datos.
Datos Locales	DATA (DS)	Referencias a datos cuando: son relativas al Stack, representan la dirección "fuente" (origen) de una operación de "string" (bloque de bytes/words), o son seleccionadas explícitamente usando una sobreasignación de segmento (segment override).
Datos Externos (Globales)	EXTRA (ES)	Referencias a datos cuando: representan la dirección "destino" de una operación de "string" (bloque de bytes/words), o son seleccionadas explícitamente usando una sobreasignación de segmento (segment override).

En la memoria, operandos tipo palabra (word) (16-bits) pueden posicionarse en direcciones límite pares o impares. Para operandos de dirección y datos, el byte menos significativo de la palabra (LSByte) es almacenado en la localidad cuya dirección tiene el valor más bajo, y el byte más significativo de la palabra (MSByte), en la siguiente más alta localidad adyacente de memoria. La BIU ejecutará automáticamente dos ciclos de lectura (fetch) o de escritura para operandos de 16 bits.

Ciertas localidades en la memoria están reservadas para operaciones específicas del μP.

{ *figura* 1-6 }

LOCALIDADES DE MEMORIA RESERVADAS EN UN SISTEMA μP 8088



Las localidades de las direcciones FFFF0 h a FFFFF h están reservadas para operaciones entre las que el diseñador del sistema debe incluir un salto (FAR JMP) incondicional a la rutina para "Inicialización del sistema". Después de un RESET (hardware) o al energizar inicialmente al sistema, el μP siempre empezará

ejecutando el código programado a partir de la localidad FFFF0 h, donde debe encontrarse el salto mencionado.

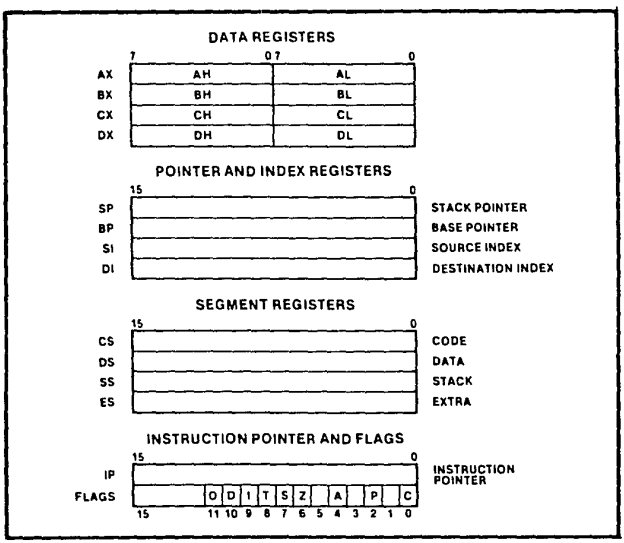
Las localidades de las direcciones 00000 h a 003FF h están reservadas para operaciones de interrupción. Apuntadores o vectores de 4 bytes que constan de una dirección de Segmento de 16 bits (base) y de una dirección de Offset de 16 bits (desplazamiento), dirigen el flujo del programa a una de las 256 posibles "rutinas de servicio a interrupción". Se asume que dichos elementos apuntadores o vectores han sido almacenados en sus respectivos lugares dentro de la memoria reservada (formando una tabla), antes de la ocurrencia de las correspondientes interrupciones.

1.7
REGISTROS INTERNOS DEL MICROPROCESADOR 8088

El μP 8088 cuenta con 14 registros internos de 16 bits que pueden clasificarse como sigue :

* 4 Registros de Datos (de propósito general) {*Data Registers*}
* 4 Registros Apuntadores e Indices {*Pointer and Index Regs.*}
* 4 Registros de Segmento {*Segment Registers*}
* 1 Registro Apuntador de Instrucción {*Instruction Pointer*}
* 1 Registro de Estado o Banderas {*Flags*}

{ *figura* 1-7 } : Conjunto de Registros del Microprocesador 8088



En el conjunto de instrucciones del μP 8088, no todos los registros pueden ser especificados como parte de cada instrucción. En muchos casos, una instrucción puede usar solamente un registro o conjunto de registros determinado, para realizar su función. Para ciertas instrucciones-8088 que ejecutan operaciones específicas, los registros tienen un uso implícito. La siguiente tabla lista aquellas operaciones de instrucción que implican un uso específico de los registros :

REGISTRO	O P E R A C I O N E S
AX	Multiplicación-Word, División-Word, Word-I/O
AL	Multiplicación-Byte, División-Byte, Byte-I/O, Transferencias (Translados de datos), Aritmética Decimal
AH	Multiplicación-Byte, División-Byte
BX	Transferencias (Translados de datos)
CX	Iteraciones (Loops) (contador), Operaciones con "Strings" (Bloques de Bytes/Words)
CL	Corrimientos (shifts) y Rotaciones variables
DX	Multiplicación-Word, División-Word, I/O-Indirecta
SP	Operaciones con el STACK
SI	Operaciones con "Strings" (Bloques) (dir.fuente)
DI	Operaciones con "Strings" (Bloques) (dir.destino)

REGISTROS DE DATOS :

Estos son 4 registros de PROPOSITO GENERAL de 16 bits cada uno, que normalmente son usados por las instrucciones para realizar operaciones aritméticas y lógicas, es decir, para almacenamiento temporal de resultados intermedios. La ventaja de usar estos registros internos para almacenamiento temporal de

datos es que, puesto que dichos datos están ya dentro de la Unidad de Ejecución (EU) del μP, pueden ser accesados mucho más rápidamente que si lo fueran de memoria externa.

Estos registros de 16 bits son llamados : AX, BX, CX y DX .

{ AX = Accumulator register } :

El registro AX sirve como ACUMULADOR principal para operaciones de 16 bits ; mantiene (acumula) los valores requeridos o entregados en la ejecución de varias instrucciones (tipo "Word" (palabra)).

{ BX = Base register } :

El registro BX funciona como un Acumulador de propósito general o como un registro base cuando se calculan direcciones de memoria (Modos de direccionamiento del μP).

{ CX = Count register } :

El registro CX sirve como Acumulador de propósito general y como un Contador para las instrucciones de Iteración múltiple; estas instrucciones terminan su ejecución cuando el contenido del registro CX es incrementado o decrementado hasta cero (0000h).

{ DX = Data register } :

El registro DX funciona como Acumulador de propósito general y como apuntador de puertos en operaciones de Entrada/Salida (I/O) con formato "variable"; la dirección del puerto que se desea accesar es cargada en el registro DX, previamente a la ejecución de las instrucciones IN o OUT ; de este modo, dado que DX es un registro de 16 bits, se pueden direccionar hasta 65536 puertos I/O diferentes (0000h a FFFFh).

Cada uno de los registros anteriores puede dividirse en dos registros independientes de 8 bits cada uno; en este caso, se identifican como parte alta (byte alto) y parte baja (byte bajo) del registro original. Los pares formados son : AH-AL de AX, BH-BL de BX, CH-CL de CX y DH-DL de DX ... {H=High, L=Low}. El acceso a estos registros de 8 bits, es para realizar las operaciones con Bytes dentro del conjunto de instrucciones del μP. De nuevo, el registro "AL" es llamado ACUMULADOR principal, pues

tiene algunas funciones y prerrogativas que los otros registros de propósito general no tienen ; mantiene (acumula) los valores requeridos o entregados en la ejecución de varias instrucciones (tipo "Byte").

REGISTROS APUNTADORES Y DE INDICE :

Registros Apuntadores : { SP , BP }

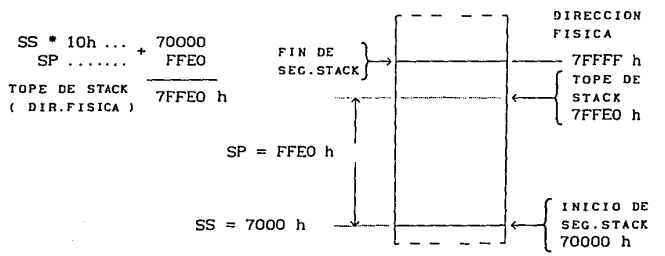
Registros de Indice : { SI , DI }

Este conjunto de registros de 16 bits se usa normalmente para generar direcciones de memoria efectivas, es decir, la porción correspondiente al OFFSET dentro de una dirección Física.

Los Registros APUNTADORES se emplean para almacenar desplazamientos (Offsets) dentro de la memoria del sistema con respecto al valor base fijado por los registros de Segmento (en este caso, el registro de STACK {SS}).

El Registro Apuntador de Stack (SP = Stack Pointer), contiene el valor del Offset de 16 bits medido desde el inicio del segmento de Stack (dirección cuyos 16 bits superiores son mantenidos en (fijados por) el registro de Segmento de STACK {SS}), hasta la localidad de memoria donde se almacenó la palabra (word) ingresada más recientemente en el STACK. Esta localidad de memoria que contiene la última "word" almacenada en el STACK, se llama "Tope de Stack" (Top of Stack).

Por ejemplo : Si SS = 7000 h y SP = FFE0 h , la Dirección Física del "Tope de Stack" es :



El Stack es una sección de memoria reservada para guardar direcciones y datos mientras se está ejecutando un subprograma.

El Registro Apuntador de Base (BP = Base Pointer), es un registro de 16 bits usado como Apuntador alterno o auxiliar en las operaciones con el Stack (sobretodo); se emplea para direccionar datos en el Stack. En este esquema de direccionamiento "default", BP actúa como un Offset dentro del Segmento de Stack (tal como [BX] se usa para direccionar memoria en DS). Para leer valores (datos) almacenados en el Stack, se carga BP con el valor de SP (MOV BP,SP), y entonces puede usarse BP como un apuntador secundario (offset) dentro del Stack; BP puede incrementarse, decrementarse, o pueden sumársele/restársele constantes a fin de ajustarlo para apuntar al dato que se quiere accesar (dejando intacto a SP). Aunque BP opera por "default" dentro de SS, también puede usarse como apuntador dentro de CS, DS y ES, usando las instrucciones con "sobreasignaciones de segmento" (Segment overrides).

Los Registros de INDICE tienen como función principal, la retención de desplazamientos de dirección (offsets de 16 bits) para instrucciones que accesan datos almacenados dentro de algún Segmento (generalmente DS o ES) en memoria. Estos registros se emplean también en la ejecución de instrucciones de transferencia de bloques (de bytes o words); así, el registro SI {Indice Fuente (*Source Index*)} guarda el offset de un byte-dato o word-dato dentro del Segmento de Datos DS, apuntando a los diferentes elementos del "Bloque Fuente (Source)", mientras que el registro DI {Indice Destino (*Destination Index*)} guarda el offset de un dato dentro del Segmento Extra de Datos ES, apuntando a los diferentes elementos del "Bloque Destino (Destination)". Además de generar direcciones efectivas de memoria (offsets), estos registros pueden ser usados para almacenamiento temporal de datos (tamaño word) o para realizar operaciones aritméticas y lógicas, tal como con los registros de propósito general descritos antes.

REGISTROS DE SEGMENTO :

La BIU - 8088 tiene 4 Registros de Segmento (de 16 bits c/u) :

CS : Segmento de Código { Code Segment }

DS : Segmento de Datos { Data Segment }

SS : Segmento de Stack { Stack Segment }

ES : Segmento Extra de datos { Extra Segment }

Los Registros de Segmento son usados para retener los 16 bits más significativos de las direcciones iniciales de los 4 segmentos de memoria con que el μP 8088 está trabajando en un instante determinado.

La BIU del 8088 manda direcciones de 20 bits, por lo que puede accesarse cualquiera de 1048576 (2^{20}) bytes en memoria. Sin embargo, en un momento dado, el 8088 sólo trabaja con 4 secciones de 65536 bytes (64 kBytes) o Segmentos dentro de este espacio de 1 MByte. Sólo 4 Segmentos pueden estar activos a la vez y éstos pueden estar separados entre sí, traslapados o aún coincidentes (encimados en su totalidad).

Resumiendo, los Registros de Segmento se emplean para posicionar los Segmentos de 64 kBytes dentro del espacio total de direccionamiento de 1 MByte y para este fin, son cargados con los 16 bits superiores de una dirección que identifica el comienzo de cada segmento activo en memoria. La BIU del μP, siempre inserta ceros para los 4 bits menos significativos de la dirección inicial de 20 bits de un segmento; por ejemplo, si CS = 457Ah, entonces el Segmento de Código empezará en la dirección Física 457A0h . En otras palabras, un segmento de 64 kBytes puede posicionarse en cualquier lugar dentro del espacio de direccionamiento de 1 MByte, pero el segmento siempre empezará en una dirección con ceros en sus 4 bits más bajos (cantidad factor de 10h => 16 bytes). De esta manera sólo será necesario manipular cantidades de 16 bits al trabajar con las direcciones iniciales de los segmentos.

Siempre que el μP 8088 realiza un acceso a memoria, la BIU produce la dirección física requerida de 20 bits, mediante el corrimiento del contenido de uno de los registros de segmento, 4

posiciones de bit hacia la izquierda (y agregando ceros) y sumándole a esa cantidad un desplazamiento u Offset.

Ante un tipo de referencia especifica a memoria, el Registro de Segmento usado por "default", es seleccionado por el hardware interno del μP 8088. Más aún, en algunos casos, es posible para el programador, "sobreencimar" las asignaciones de segmento "default" y, usando una instrucción de "prefijo de segmento", especificar un registro de Segmento diferente para la ejecución de alguna instrucción ... (cambio del segmento default) {Segment Override}.

El registro de Segmento que es usado con una referencia especifica a memoria, se define en la siguiente tabla :

USO DE LOS REGISTROS DE SEGMENTO

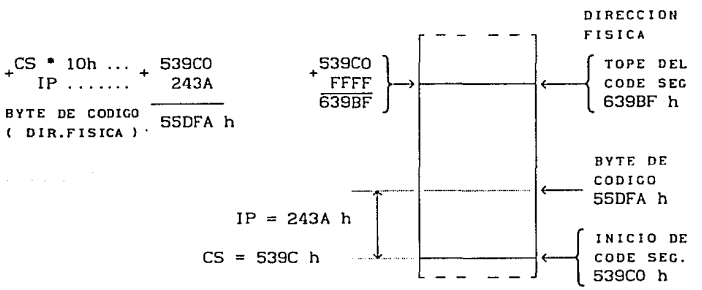
Tipo de Referencia a Memoria	Segmento Base Default	Segmento Base Alterno	Offset
"Fetch" de Instrucción	CS	Ninguno	IP
Operación de Stack	SS	Ninguno	SP
Variable (excepto las que siguen a continuación ↓)	DS	CS,ES,SS	Dirección Efectiva
"String"(bloque) Fuente	DS	CS,ES,SS	SI
"String"(bloque) Destino	ES	Ninguno	DI
BP usado como Registro Base	SS	CS,DS,ES	Dirección Efectiva

REGISTRO APUNTADOR DE INSTRUCCION :

El IP {*Instruction Pointer*} es un registro que identifica la localidad de memoria en la que se encuentra la próxima instrucción que será capturada. Es similar a un registro "Contador de programa", sin embargo, el IP contiene un desplazamiento u Offset

(de 16 bits) que indica de dónde será extraído (Fetched) el siguiente byte de código de instrucción dentro del rango de 64 kBytes establecido por el Segmento de Código (CS) (para ser cargado en los registros FIFO de la Cola de Instrucciones). Así pues, la dirección Física real (20 bits) del siguiente byte de instrucción, que es mandada por el Bus para direccionar memoria, es producida al efectuar un corrimiento del contenido del registro de Segmento de Código (CS) 4 posiciones de bit hacia la izquierda (agregando ceros en los 4 bits menos significativos) y sumando a esta nueva cantidad de 20 bits, el Offset contenido en el registro IP. Cada vez que una instrucción es capturada de la memoria, la BIU del μP actualiza el valor del IP para que apunte a la siguiente instrucción (byte de código ejecutable) en memoria.

Por ejemplo : Si CS = 539C h y IP = 243A h , la Dirección Física del "byte de código" será :



El tamaño del Segmento es de 65536 = (FFFF h) bytes (64 kBytes)

REGISTRO DE BANDERAS (FLAGS) :

Una Bandera (Flag) es un flip-flop que indica alguna condición producida por la ejecución de una instrucción, o controla ciertas operaciones en la EU (Unidad de Ejecución) del μP. En la EU del μP 8088, el registro de Banderas (Flags Register) tiene 16 bits

pero de ellos sólo 9 banderas activas. De estas banderas 6 son de Estado y 3 son de Control.

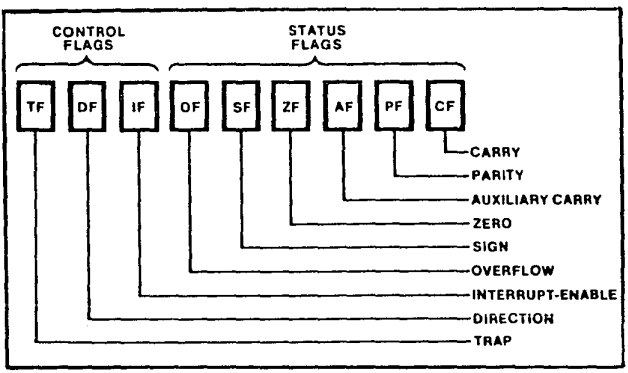
. . . . . .

{ *figura* 1-8 } : Banderas (Flags) del Microprocesador 8088

. . . . . .

{ *figura* 1-8 }

Banderas (Flags) del μP 8088



BANDERAS DE ESTADO (STATUS FLAGS) :

Indican condiciones producidas como resultado de la ejecución de operaciones aritméticas y lógicas. Estas 6 banderas son puestas en "1" (set) ó en "0" (reset) por la EU del μP, en base a esos resultados y el programador únicamente puede leerlas. También se llaman "Banderas condicionales" ya que ciertas instrucciones del 8088 verifican el estado de estas banderas para determinar cúal de entre dos acciones alternativas debe seguirse en la ejecución de la instrucción (ej. "conditional Jumps").

Las Banderas de Estado (Status) son :

CF : Carry Flag (Bandera de Acarreo) :
Este bit es puesto en "1"-lógico (set) {CF=1}, si se produce una condición de "carry out" o "borrow into" de la posición del bit más significativo del resultado de una operación aritmética (8 ó 16 bits). En caso contrario es puesto en "0" (reset) {CF=0}. Esta bandera es usada por instrucciones que suman y restan números multibyte; las instrucciones de Rotación pueden también aislar un bit en memoria o un registro, colocándolo en la bandera de Acarreo.

PF : Parity Flag (Bandera de Paridad) :
Este bit es puesto en "1" (set) {PF=1}, cuando el resultado de una operación tiene Paridad Par (contiene un número par de bits 1's). Esta bandera se usa para detectar errores en la transmisión de datos.

AF : Auxiliary Carry Flag (Bandera Auxiliar de Acarreo) :
Este bit es puesto en "1" (set) {AF=1}, cuando se produce una condición de "carry out" , "borrow from" o "borrow to" sobre un "nibble" (4 bits) en instrucciones aritméticas decimales (BCD). (Carry out of the low nibble into the high nibble or Borrow from the high nibble into the low nibble of an 8-bit quantity).

ZF : Zero Flag (Bandera Cero) :
Este bit es puesto en "1" (set) {ZF=1}, si el resultado de la última operación aritmética o lógica es "cero". Si el resultado es diferente de "cero", entonces {ZF=0}.

SF : Sign Flag (Bandera de Signo) :
Este bit indica que el bit más significativo del resultado es puesto en "1"-lógico y, entonces, se trata de un número negativo. Si por el contrario, el resultado es positivo, entonces {SF=0}. Puesto que en el µP 8088/86 los números negativos binarios se representan en notación estándar de

"complemento-a-2", SF indica el signo del resultado.

OF : Overflow Flag (Bandera de Sobreflujo) :
Este bit es puesto en "1" (set) {OF=1}, si ocurre una condición aritmética de sobreflujo, e indica que el resultado es demasiado grande para quedar contenido en el campo destino (registro, loc.mem.), es decir, se pierde un dígito significativo debido a que el tamaño del resultado excede (sobrepasa) la capacidad de la localidad destino. El μP dispone de una instrucción "Interrupt On Overflow" que genera una interrupción en esta situación.

BANDERAS DE CONTROL (CONTROL FLAGS) :
Los 3 bits (flags) restantes en el registro de Banderas se usan para controlar ciertas operaciones del procesador, y son deliberadamente puestos en "1"-lógico (set) ó en "0"-lógico (reset) mediante instrucciones específicas que el usuario pone en el programa.

Las Banderas de Control (Control) son :

TF : Trap Flag (Bandera de Trampa) :
Este es un bit de control que, cuando es fijado en "1"-lógico (set) {TF=1}, pone al μP 8088 en un modo de "Paso simple" (Single-step mode). Después de la ejecución de cada instrucción, se genera automáticamente una interrupción (Tipo-1) (permitiendo que un programa pueda ser inspeccionado conforme se va ejecutando instrucción por instrucción) . Por tanto, la Bandera TF y la INT_Tipo-1, facilitan la tarea de implementar un modo de supervisión - depuración de programas (Debugging) en que se ejecute una instrucción a la vez, y mediante una rutina de servicio a interrupción adecuada, se salve en stack el contenido de todos los registros del μP para su posterior análisis o tal

vez, despliegue en una pantalla (CRT). No existen instrucciones de μP que directamente pongan en "1" ó en "0" el bit TF ; estas operaciones son hechas mediante un PUSH del reg. de banderas en stack, cambio del bit TF al estado lógico deseado, y un POP para recuperar el reg. de banderas modificado del stack. La bandera TF es reseteada (0) cuando el μP 8088 realiza una interrupción tipo-1, por lo que el modo de "Paso simple" será deshabilitado durante la ejecución de la rutina de servicio a interrupción (a fin de que ésta pueda ejecutarse normalmente).

IF : Interrupt-enable Flag (Bandera habilitación-Interrupción) : Este es un bit de control que, cuando es puesto en "1"-lógico (set) {IF=1}, habilita al μP para el reconocimiento de solicitudes de Interrupción mascarable externa; cuando es "0" {IF=0} inhibe dicha respuesta a interrupción. La bandera IF no tiene efecto sobre interrupciones-no-mascarables externas ni sobre interrupciones generadas internamente (por software).
Para poner en "1"-lógico el bit IF, debe ejecutarse la instrucción de μP "STI" {*Set Interrupt Flag*} que habilita INTR a partir de la siguiente instrucción; para poner en "0" el bit IF, hay que ejecutar la instrucción "CLI" {*Clear Interrupt Flag*}.

DF : Direction Flag (Bandera de Dirección) :
Este es un bit de control que, cuando es puesto en "1"-lógico (set) {DF=1}, ocasiona que las operaciones sobre "Strings" (bloques de bytes/words) en el μP sean con "auto-decremento" de dirección, o sea, procesadas de la dirección más alta a la dirección (offset) más baja (por ejemplo, la transferencia de una fila o bloque de datos de una zona de memoria a otra). Si este bit es puesto en "0" {DF=0}, el "String" o bloque es procesado en un modo de "auto-incremento" de dirección. El estado de esta bandera afecta a instrucciones como "MOVS","CMPS" o "SCAS".

Para poner en "1"-lógico el bit DF, debe ejecutarse la instrucción de μP "STD" {*Set Direction Flag*} a fin de que los registros SI y/o DI sean automáticamente decrementados para apuntar al siguiente elemento del "String" (bloque) {procesamiento "High-to-Low address" o "de derecha a izquierda"}; para poner en "0" el bit DF, hay que ejecutar la instrucción "CLD" {*Clear Direction Flag*} fijando así el modo autoincremento {procesamiento "Low-to-High address" o "de izquierda a derecha"}.
El incremento/decremento de SI y/o DI será en 1 para "strings" o bloques de Bytes, y en 2 para "strings" o bloques de Words (palabras).

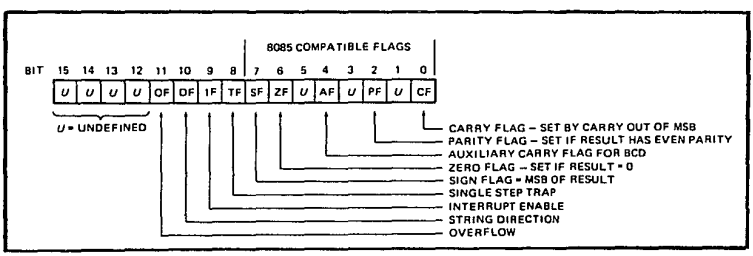
. . . . . .

{ *figura* 1-9 } : Formato del Registro de Banderas (Flags) del Microprocesador 8088

. . . . . .

{ *figura* 1-9 }

Formato del Registro de Banderas (Flags) del μP 8088



1.8
MODOS DE DIRECCIONAMIENTO DEL MICROPROCESADOR 8088

Cuando el μP 8088 ejecuta una instrucción, realiza una operación especifica sobre los datos. Estos datos pueden ser parte de la instrucción, residir en uno de los registros internos del 8088, estar almacenados en una localidad de memoria, o estar en un puerto de Entrada/Salida (I/O). Para accesar estos diferentes tipos de operandos, el μP 8088 está provisto de varios "Modos de Direccionamiento".

La dirección Física de 20 bits generada por la BIU (*Bus Interface Unit*) del μP, está compuesta de dos partes : un valor de Segmento o dirección Base y un valor de Offset o dirección efectiva (*efective-address*). El Offset que la EU (*Execution Unit*) calcula para un operando de memoria, se denomina "dirección efectiva del operando" o (EA), y es un número no signado de 16 bits que expresa la distancia del operando en bytes, medida desde el inicio del segmento en el que reside. La EU puede calcular la EA de un operando en memoria de varias maneras; la información codificada en el segundo byte de una instrucción, indica a la EU cómo calcularla. Un compilador o ensamblador extrae esta información de la sentencia o instrucción escrita por el programador. Así, los programadores de Lenguaje Ensamblador tienen acceso a todos los Modos de Direccionamiento.

La EU del μP calcula la Dirección-Efectiva (EA) (codificada en una instrucción) sumando un Desplazamiento, el contenido de un registro Base y el contenido de un registro Indice. Una instrucción de 8088 puede especificar CUALQUIER COMBINACION de estos tres componentes, para crear una Dirección Efectiva (EA).

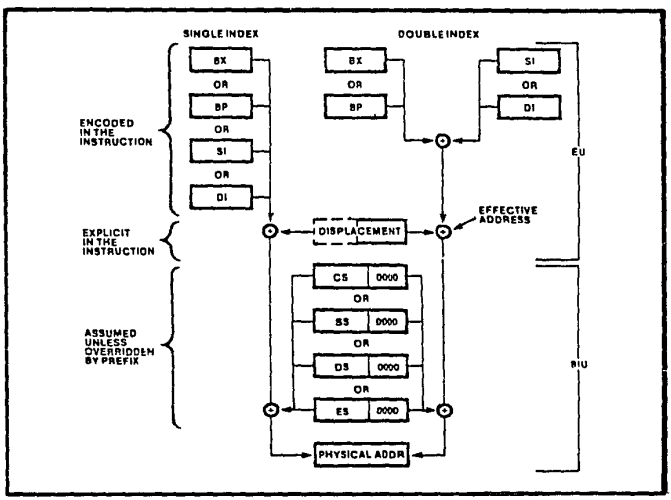
Esto permite una variedad de Modos de Direccionamiento de memoria.

. . . . . .

{ *figura* 1-10 } : Cómputo de Dirección Física de Memoria en el Microprocesador 8088

. . . . . .

{ *figura* 1-10 }



El desplazamiento es un número de 8 ó de 16 bits que está contenido en la instrucción. El desplazamiento se deriva generalmente de la posición del nombre del operando (una variable o etiqueta (*label*)) dentro del programa. El programador puede también modificar este valor o especificar explícitamente el desplazamiento (en forma numérica).

El programador puede seleccionar tanto a BX como a BP para servir como registro Base cuyo contenido sea usado en el cómputo de EA. Similarmente puede especificarse a SI o a DI para funcionar como registro Indice. El valor de desplazamiento es una constante. El contenido de los registros Base e Indice puede cambiar durante la ejecución; esto permite que con una sola instrucción se puedan accesar distintas localidades de memoria según los valores contenidos en los registros Base y/o Indice. Los cálculos de dirección efectiva con BP emplean el registro SS, por default, aunque también pueden especificarse DS o ES para la dirección de segmento base.

En el 8088 se tienen los siguientes Modos de Direccionamiento :

1) Direccionamiento de Registro
2) Direccionamiento Inmediato
3) Direccionamiento Directo
4) Direccionamiento de Registro Indirecto
5) Direccionamiento de Base
6) Direccionamiento de Indice (indexado)
7) Direccionamiento de Base-Indice (base indexado)
8) Direccionamiento de "Strings" (de bloques bytes/words)
9) Direccionamiento de Puertos I/O

Resumiendo :
Para obtener la dirección Física (real) (20 bits) del operando deben sumarse cuatro cantidades (la suma se realiza interna y automáticamente dentro del μP) :

1) Una dirección base de Segmento (el contenido del registro de segmento es multiplicado por 10h antes de utilizarse para generar la dirección Física).
2) El contenido de un registro Base. \ Dirección
3) El contenido de un registro Indice. - Efectiva
4) Un desplazamiento. / { EA }

En la siguiente tabla se resumen los modos de direccionamiento y los registros disponibles en cada modo (Modos de EA), así como los ciclos de reloj consumidos en cada uno.
(Un ciclo de reloj dura un Periodo { T } de reloj calculado a la frecuencia de operación del μP { f=1/T }).

Componentes de Dirección Efectiva (EA)		CICLOS DE RELOJ * (CLOCKS)
Desplazamiento únicamente		6
Base o Indice únicamente	{ BX,BP,SI,DI }	5
Desplazamiento		
+		9
Base o Indice	{ BX,BP,SI,DI }	
Base	{ BX+SI } , { BP+DI }	7
+		
Indice	{ BX+DI } , { BP+SI }	8
Desplazamiento	{ BX+SI+Desp }	
+	{ BP+DI+Desp }	11
Base		
+	{ BX+DI+Desp }	
Indice	{ BP+SI+Desp }	12

* : Agregar 2 ciclos de reloj (clocks) cuando se usen reasignaciones de Segmento default (Segment overrides)

MODOS DE DIRECCIONAMIENTO Y POSIBLES ARREGLOS DE REGISTROS

MODO DE DIRECCIONAMIENTO	POSIBLES ARREGLOS
Directo (DIRECT)	(etiqueta) (label) desplazamiento {n} (número)
Registro Indirecto (REGISTER INDIRECT)	[BX] [BP] [SI] [DI]
Base (BASED)	[BX+n] [BP+n]
Indice o Indexado (INDEXED)	[SI+n] [DI+n]
Base Indexado (BASED INDEXED)	[BX+SI] [BX+DI] [BP+SI] [BP+DI]
Base Indexado con desplazamiento (BASED INDEXED WITH DISPLACEMENT)	[BX+SI+n] [BX+DI+n] [BP+SI+n] [BP+DI+n]

n : representa un desplazamiento : { de 8 bits signado }
ó { de 16 bits }

1.9 INTERRUPCIONES CON EL MICROPROCESADOR 8088

Con el μP 8088 las interrupciones pueden ser iniciadas por Hardware o iniciadas por Software.

Las interrupciones por Software se originan :

a) Directamente, por programación mediante la ejecución de la instrucción "INT n" { n = vector (Int."type" #) }.

b) Indirectamente, por la lógica del programa mediante alguna condición producida en el μP por la ejecución de alguna instrucción (aritmética o lógica) (ej.condición de división entre cero) . Se produce una interrupción en la ejecución del programa, de manera automática. (Interrupciones Condicionales).

Las interrupciones por Hardware se originan :

a) Por una señal externa aplicada a la entrada NMI del μP (pin 17) ... Interrupción No Mascarable.

b) Por una señal externa aplicada a la entrada INTR del μP (pin 18) ... Interrupción Mascarable.

Las interrupciones por Software pueden ocurrir :

1) Siguiendo un intento de dividir entre cero. Se generará una solicitud especial de interrupción cada vez que se ejecute una división (DIV ó IDIV) cuyo divisor sea "cero" (el cociente excede el máximo valor). { Divide Error : Int_type 0 } . (No mascarable por IF).

2) Después de la ejecución de la instrucción "INT 3" Es una forma especial de "Software Int" que requiere de un solo byte de código (CCh). Suele emplearse como

Interrupción de "punto de ruptura" (breakpoint) para depuración de software (Debuggers). { One-byte Int. : Int_type 3 } . (No mascarable por IF).

3) Después de la ejecución de la instrucción de 2 bytes "INT n" .
(n = # de vector de interrupción = Int_type #).
(n = 0 a 255 ; el Int_type_# deseado se especifica como parte de la instrucción).
(No mascarable por IF).

4) Después de la ejecución de la instrucción "INTO", si en el registro de Banderas se tiene que OF=1 (Overflow flag). Permite procesar (atrapar) errores de Sobreflujo mediante una rutina de interrupción.
{ Interrupt on Overflow : Int_type 4 } .
(No mascarable por IF).

5) Una instrucción después de que la bandera TF (Trap flag) es puesta en estado "1"-lógico. Se usa para que el μP entre al modo de ejecución paso-a-paso en una secuencia de código.
{ Single Step : Int_type 1 } .
(No mascarable por IF).

Una solicitud de Interrupción No Mascarable (NMI) es iniciada cuando la lógica externa produce una transición "low-high" (L-H) en la terminal NMI del μP 8088 (edge triggered input).
{ NMI : Int_type 2 } .

Una solicitud de Interrupción Mascarable (INTR) es generada cuando la lógica externa produce un nivel "high" (H) en la terminal INTR del μP 8088 (level triggered input) y IF=1 en el registro de Banderas.

Los "tipos" de interrupción 0 y 2 ocurren sin que se presente una acción especifica por parte del programador, mientras que los

"tipos" 1, 3 y 4 requieren de un acto consciente del programador. Todos los tipos 0 a 4, excepto el 2 (NMI), son invocados por actividad de software y están directamente asociados con una instrucción especifica.

En el caso de que dos o más de estos tipos de interrupción ocurriesen simultáneamente, la prioridad de atención asignada es la siguiente :

Más alta	Error-División-por-Cero, INT n , INTO
↓	NMI
	INTR
Más baja	Int.Paso-simple (Single-step).

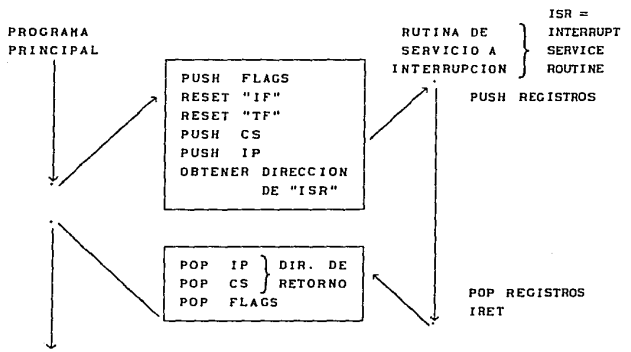
El Tiempo de Procesamiento de los distintos tipos de Interrupción es :

	Clase de Interrupción	Tiempo de procesamiento (Ciclos de Reloj) (Clocks)
*	Interrupción Mascarable externa (INTR)	61
*	Interrupción No Mascarable (NMI)	50
*	+ INT (con vector)	51
	+ INT (type 3)	52
	+ INTO	53
*	Interrupción de Paso-simple (Single-step)	50

Todas las interrupciones, ya sean originadas por Software o por Hardware, resultan en la transferencia del control a una nueva localidad con código ejecutable (programa) (Rutina de Servicio a Interrupción [ISR = *Interrupt Service Routine*]).

Al final de cada ciclo de instrucción, el μP verifica si se ha producido (solicitado) alguna interrupción. Si así es, el 8088 responde a esta interrupción desencadenando una serie de acciones que se muestran en la siguiente figura :

{ *figura* 1-11 } : Respuesta a Interrupción { μP 8088 }



En un sistema basado en μP 8088, el primer kByte de memoria (normalmente es RAM) que comprende de la dirección física (absoluta) 00000h a 003FFh , debe apartarse para la creación de la Tabla de Vectores de Interrupción, que almacena la dirección inicial de cada una de las 256 posibles rutinas de Servicio a Interrupción (Int_type_0 a Int_type_255). Cada elemento (vector o apuntador) de esta tabla consta de 2 valores de dirección de 16 bits c/u (4 bytes) y almacena el valor de CS y de IP para cada rutina de Servicio a Interrupción, es decir, cada apuntador de interrupción de 4-bytes es identificado por un número (Int_type) entre 0 y 255.

Los 5 primeros vectores de Interrupción corresponden a interrupciones-iniciadas-por-software y a la interrupción de hardware NMI. Los siguientes 27 "tipos" de interrupción (5 a 31), están reservados por Intel, y no deben ser usados si se desea tener compatibilidad con futuros µP's de Intel. Los restantes 224 "Int_types" (32 a 255), están disponibles para apuntar a rutinas de interrupción (hardware o software) del usuario.

Cuando el 8088 responde a una interrupción, automáticamente se dirige a la localidad especificada por el "tipo" de interrupción dentro de la tabla de apuntadores (vectores) para obtener la dirección inicial de la correspondiente Rutina de Servicio a Interrupción (ISR).

El usuario debe escribir las Rutinas de Servicio a Interrupción e inicializar la tabla de vectores (apuntadores) de interrupción con las direcciones iniciales apropiadas de esas rutinas. El 1^{ER} byte de cualquier Vector de la tabla debe contener el byte bajo del IP de la ISR {IPLOW}; el 2^{O} byte , el byte alto del IP de la ISR {IPHIGH}; el 3^{ER} byte , el byte bajo del CS de la ISR {CSLOW}; y el 4^{O} byte , el byte alto del CS de la ISR {CSHIGH}.

También es responsabilidad del usuario el invocar a estas interrupciones ya sea por hardware o por software.

Las interrupciones por Software definidas por el usuario son generadas con la instrucción de 2 bytes "INT n", que no puede enmascararse con el bit IF del registro de Banderas. La instrucción "INT n" permite la selección de cualquiera de los 256 vectores de la Tabla; la opción automática selecciona el vector 3. Esta instrucción puede usarse para transferir el control a rutinas que son dinámicamente relocalizables y cuya localización en memoria no es conocida por el programa base (calling program). Esta técnica también salva en el Stack el registro de Banderas

antes de saltar a la ISR y requiere de la instrucción IRET para recuperarlas del Stack y restablecer completamente el estado del programa principal.

Cuando una interrupción por programación (software) es reconocida se realizan las siguientes acciones :

1) Se decrementa el SP (Stack Pointer) en 2 y se guarda el registro de Banderas en el Stack (PUSH Flags).

2) Se deshabilita la entrada de interrupción mascarable (INTR) del μP al resetear la bandera de interrupción { IF=0 } en el registro de Banderas.

3) Se deshabilita el modo de "paso simple" al resetear la bandera de trampa o rastreo (*trace*) { TF=0 } en el registro de Banderas.

4) Se decrementa el SP en 2 y se guarda en el Stack el contenido del registro de Segmento de Código presente (CS del programa principal).
(PUSH CS ... de la dirección de retorno).

5) Se decrementa el SP en 2 y se guarda en el Stack el contenido del registro Apuntador de Instrucción presente (IP del programa principal).
(PUSH IP ... de la dirección de retorno).

6) Los nuevos valores de IP y CS correspondientes al inicio de la Rutina de Servicio a Interrupción (ISR), son obtenidos por el μP de la localidad especificada por el "tipo" de Int. (codificado como parte de la instrucción (00h a FFh)) dentro de la Tabla de Vectores de Interrupción preinicializada.
Cuando estos nuevos valores son cargados en los registros CS e IP, el μP empezará la ejecución de la ISR.
{ Se realiza un FAR CALL al inicio de la rutina escrita por el usuario para responder a la interrupción }.

La Interrupción No Mascarable (NMI) (Int_type_2) es la interrupción de hardware de más alta prioridad. La entrada NMI del μP es disparada por flanco (L-H), pero está sincronizada con el reloj del CPU (debe estar activa 2 ciclos de reloj para garantizar su reconocimiento) y puede ser removida antes de entrar a la ISR. Generalmente esta interrupción se reserva para fallas catastróficas, como falla en el suministro de energía o condición de tiempo muerto ("timeout") en un sistema temporizador de supervisión (Watchdog).

Cuando una interrupción No Mascarable es reconocida se realizan los siguientes eventos :

1) Se decrementa el SP (Stack Pointer) en 2 y se guarda el registro de Banderas en el Stack (PUSH Flags).

2) Se deshabilita la entrada de interrupción mascarable (INTR) del μP al resetear la bandera de interrupción { IF=0 } en el registro de Banderas.

3) Se deshabilita el modo de "paso simple" al resetear la bandera de trampa o rastreo (*trace*) { TF=0 } en el registro de Banderas.

4) Se guarda en el Stack la dirección de retorno al programa principal (CS e IP) y los nuevos valores a cargar en esos registros son tomados del Vector de Interrupción # 2, en la tabla de memoria.

Se transfiere el control a la Rutina de Servicio a Interrupción correspondiente, misma que el usuario deberá terminar con una instrucción IRET.

Las interrupciones Mascarables son iniciadas por el hardware del sistema y son activadas a través de un protocolo entre las terminales INTR e $\overline{\text{INTA}}$ del 8088 y enmascaradas por el bit IF del registro de Banderas.

Cuando una interrupción Mascarable es reconocida, {(Interrupt Flag IF=1) y (entrada INTR del μP recibe un nivel "high")} ocurren los siguientes eventos :

1) Se habilita el Bus de Datos en modo de entrada (input mode) : DT/$\overline{R}$=0 y $\overline{DEN}$.

2) Se mandan dos pulsos de Reconocimiento de interrupción (Interrupt Acknowledge) por la terminal $\overline{INTA}$ del μP. Los pulsos $\overline{INTA}$ indican a un dispositivo (hardware) externo (como el 8259A), que debe enviar un byte por el Bus de Datos (AD7-AD0) en respuesta al segundo pulso $\overline{INTA}$. Este byte identifica la fuente de la interrupción (número de vector o "tipo" de vector de INT : "*Interrupt Type*").

3) Cuando el μP lee este byte (8 bits) {"Tipo" de interrupción} procedente del dispositivo externo, lo multiplica por 4 y el valor resultante es usado como un apuntador (*pointer*) dentro de la tabla de Vectores de Interrupción.

4) Se decrementa el SP (Stack Pointer) en 2 y se guarda el registro de Banderas en el Stack (PUSH Flags).

5) Se deshabilita la entrada de interrupción INTR del μP al resetear la bandera de interrupción { IF=0 } en el registro de Banderas.

6) Se deshabilita el modo de "paso simple" al resetear la bandera de trampa o rastreo (*trace*) { TF=0 } en el registro de Banderas.

7) Se decrementa el SP en 2 y se guarda en el Stack el contenido del registro de Segmento de Código presente (CS del programa principal).
(PUSH CS ... de la dirección de retorno).

8) Se decrementa el SP en 2 y se guarda en el Stack el contenido del registro Apuntador de Instrucción presente (IP del programa principal).
(PUSH IP ... de la dirección de retorno).

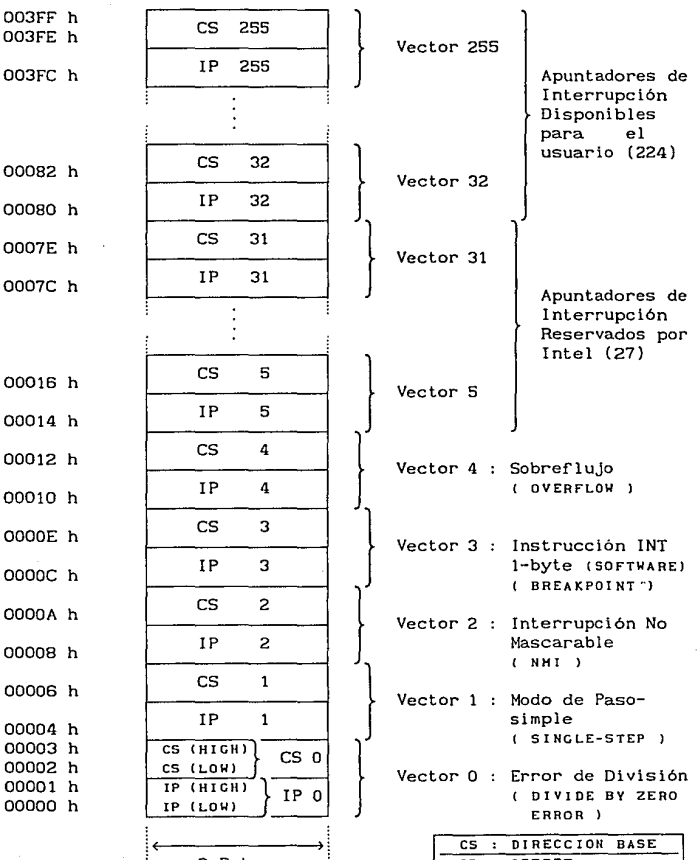
9) De la dirección apuntada por el valor generado en el punto (3) y de las 3 siguientes localidades-byte contiguas dentro de la tabla de Vectores de Interrupción, el μP obtiene los valores de IP y CS correspondientes al inicio de la Rutina de Servicio a Interrupción (ISR). Cuando estos nuevos valores son cargados en los registros CS e IP, el μP empezará la ejecución de la ISR.
{ Se realiza un FAR CALL al inicio de la rutina escrita por el usuario para responder a la interrupción }.

Al final de la Rutina de Servicio a Interrupción, el usuario debe colocar una instrucción IRET para regresar a ejecutar el programa principal interrumpido. Esta instrucción recupera del Stack los valores de IP, de CS y del Registro de Banderas (Flags) para restaurar en el μP las condiciones de operación (Status) previas a la atención de la interrupción. Esto rehabilitará a la entrada INTR del μP (puesto que se restaura la condición IF=1 en Flags), por lo que si un nivel lógico "high" continúa presente en la entrada INTR, provocará que el μP sea interrumpido de nuevo por la misma señal; si no se desea que esto ocurra, se tendrá que usar hardware externo para asegurarse de que dicha señal es conmutada al estado "low" de nuevo antes de habilitar INTR con la instrucción STI, o (por software, p.ej. con un 8259A) antes del final de la Rutina de servicio a interrupción-INTR.

.....

{ *figura* 1-12 } : Estructura de la Tabla de Vectores o Apuntadores de Interrupción en el Microprocesador 8088
(Se inicializa en la zona más baja de la memoria del sistema (generalmente RAM))

.....


Dirección de Memoria
Tabla de Vectores de Interrupción
Definición del Vector (Apuntador de Interrupción)
003FF h
003FE h
003FC h
CS 255
IP 255
Vector 255
Apuntadores de Interrupción Disponibles para el usuario (224)
00082 h
00080 h
CS 32
IP 32
Vector 32
0007E h
0007C h
CS 31
IP 31
Vector 31
Apuntadores de Interrupción Reservados por Intel (27)
00016 h
00014 h
CS 5
IP 5
Vector 5
00012 h
00010 h
CS 4
IP 4
Vector 4 : Sobreflujo (OVERFLOW)
0000E h
0000C h
CS 3
IP 3
Vector 3 : Instrucción INT 1-byte (SOFTWARE) (BREAKPOINT)
0000A h
00008 h
CS 2
IP 2
Vector 2 : Interrupción No Mascarable (NMI)
00006 h
00004 h
CS 1
IP 1
Vector 1 : Modo de Paso-simple (SINGLE-STEP)
00003 h
00002 h
00001 h
00000 h
CS (HIGH)
CS (LOW)
CS 0
IP (HIGH)
IP (LOW)
IP 0
Vector 0 : Error de División (DIVIDE BY ZERO ERROR)
2 Bytes
CS : DIRECCION BASE
IP : OFFSET

1 . 10
CONJUNTO DE INSTRUCCIONES DEL MICROPROCESADOR 8088

Para una descripción detallada e información de temporización sobre el conjunto de instrucciones del µP 8088, consultar :

* "The iAPX BOOK"
  publicado por Intel Corp.

* "iAPX 86/88 , 186/188 User's Manual"
  publicado por Intel Corp.

* "MACRO ASSEMBLER" (Programmer's Guide)
  publicado por IBM o por MICROSOFT Corp.

En el apéndice "A" de este documento se incluyen tablas de instrucciones del µP 8086/8088 (obtenidas de la segunda referencia mencionada arriba).

+

CAPITULO II

EL SISTEMA DIGITAL DE DESARROLLO SDM88-PC (HARDWARE)

SISTEMA DE DESARROLLO
BASADO EN EL MICROPROCESADOR 8088
Y EN UNA MICROCOMPUTADORA
TIPO PC (PERSONAL COMPUTER)
O PS (PERSONAL SYSTEM)
(IBM COMPATIBLE)

INTRODUCCION

En este capitulo se describe la configuración en *Hardware* del Sistema Digital de Desarrollo SDM88-PC (Sistema de Desarrollo basado en el Microprocesador 8088 y en una microcomputadora tipo PC (ó PS)) .

Este sistema consta en realidad de dos microcomputadoras trabajando conjuntamente : la microcomputadora PC (ó PS) y una "microcomputadora-interface" (*Hardware*) basada en un microprocesador 8088 externo que administra su propia memoria y controla sus propios periféricos I/O ; esta última microcomputadora es controlada por la primera (PC ó PS). Esto es así, porque el usuario puede operar sobre los componentes de la Interface (circuitos integrados programables de soporte controlados por un microprocesador 8088), a través de la microcomputadora PC (PS), haciendo uso de los recursos disponibles de este *Hardware* externo para el control de procesos o para la creación de instrumentos-prototipo concebidos para dar solución a una necesidad especifica.

El microprocesador 8088 de la interface (externo a la PC), se encarga de establecer un canal de comunicación con la microcomputadora PC (PS) a través de los puertos serie presentes en ambos subsistemas : 8250 en PC (PS) y 8251A en interface.
Este microprocesador externo también se ocupará de manejar sus bancos de memoria (EPROM y RAM) y de inicializar y controlar tanto los dispositivos I/O presentes en la interface, como los que eventualmente puedan anexarse o existir en sistemas (*Hardware*) de aplicación especifica diseñados por el usuario y que funcionen inicialmente como "satélites" del SDM88-PC , mientras se determina su factibilidad y sus requerimientos minimos de Hardware digital de control, con vistas a su operación como unidades autónomas.

Estos sistemas de aplicación desarrollados a partir del **SDM88-PC** (empleando al **SDM88-PC** como herramienta de diseño), son controlados lógicamente, por programas (*Software*) creados por el usuario en la PC (PS) . Posteriormente (dentro del ambiente del **SDM88-PC**), los programas en código hex del μP 8088, son transferidos (cargados) a RAM de la microcomputadora-interface, para actuar sobre el *Hardware* de la aplicación a través del *Hardware* de la interface (con o sin la supervisión de la PC, según se requiera).

La idea de diseñar un sistema que aprovechase los recursos de una microcomputadora estándar comercial, (despliegue en pantalla, interface de teclado, gran capacidad de almacenamiento (RAM y unidades de disco flexible y/o duro), extenso soporte de *software*, *hardware* y periféricos en constante desarrollo, etc.) y que tuviese el apoyo del nivel de programación que brinda un microprocesador de 16 bits y a la vez la ventaja de utilizar lógica de 8 bits en su electrónica (suficiente en la mayoría de los casos para aplicaciones de instrumentación y control), me condujo a seleccionar a la microcomputadora PC (PS) IBM compatible y al microprocesador 8088 de *intel*, obteniendo así un sistema muy poderoso a costo relativamente bajo, considerando las ventajas que ofrece en cuanto a versatilidad, vigencia y facilidad de manejo como herramienta auxiliar en el desarrollo de instrumentos prototipo controlados digitalmente, o bien, en su aplicación directa para actividades de control y/o adquisición y procesamiento de datos.

Otra gran ventaja del **SDM88-PC** es que las dos microcomputadoras que lo conforman, son totalmente compatibles en código objeto y lenguaje de máquina. Es decir, si la microcomputadora PC (PS) empleada (conectada a la interface por su puerto serie), tiene un microprocesador 8088, 8086, 80188, 80186, 80286 ú 80386, seguirá asegurándose una compatibilidad en código

básico de microprocesador (en modo NO-PROTEGIDO { set de instrucciones no extendido para multiprocesamiento }) con el μP 8088 de la interface. Esta característica, propia de todos los miembros de la familia 8086 de *intel*, da acceso inmediato a la base de *Software* más amplia del mundo y es constantemente aprovechada en el **SDM88-PC** para la creación de programas en lenguaje de máquina a ejecutarse en la interface, empleando ensambladores comerciales para la PC : (mediante algún editor de texto en formato ASCII, se escriben programas usando mnemónicos y posteriormente se compilan para generar automáticamente el código hexadecimal correspondiente para el μP, liberando al usuario de esta tediosa tarea). Sin embargo, el **SDM88-PC** también cuenta con los comandos necesarios para introducir código hexadecimal directamente en memoria RAM . Gracias a esta compatibilidad, también pueden emplearse *Debuggers* comerciales para revisar, desde el ambiente de la PC, la ejecución lógica de los programas desarrollados que se destinarán al control de la microcomputadora-interface *y/o* de sus aplicaciones-satélite.

En el Capítulo 5 se situará al **SDM88-PC** dentro del contexto de sistemas auxiliares para el diseño en instrumentación y para el control de procesos y se abundará sobre sus ventajas (a mi juicio) sobre equipos equivalentes disponibles en el mercado; también se hablará del funcionamiento de la Microcomputadora PC (PS) y de la Microcomputadora-interface (Hardware externo), como una **unidad integral**.

Existe amplia y explícita literatura del dominio público que trata sobre el *Hardware* propio de las microcomputadoras tipo PC (PS), por lo que, en este documento, sólo se describirán las características de la parte del sistema **SDM88-PC** referente a la microcomputadora-interface (*Hardware* externo a la PC).

2.1
ESPECIFICACIONES TECNICAS

Microcomputadora-Interface
(Hardware externo a la Microcomputadora PC (PS))
en el Sistema de Desarrollo **SDM88-PC** :

* Microprocesador (CPU) : 8088 (INTEL) HMOS
  + Frecuencia de Reloj : 4.77 [MHz]
  + Arquitectura interna : 16 bits
  + Bus (canal) de Datos : 8 bits
  + Capacidad de direccionamiento directo de memoria : 1 MByte
    2^{20} = 1048576
    20 lineas de dirección A0-A19 . {A0-A7} están multiplexadas con Bus de Datos {AD0-AD7}
  + Conjunto de instrucciones : 117 instrucciones
  + Modos de Direccionamiento de Operando : 24
  + Compatibilidad directa en software con μP 8086 y en general con 80188, 80186, 80286, 80386 (en modo NO-PROTEGIDO o extendido)

* Circuito Temporizador para el microprocesador y sus periféricos (generador-sincronizador de CLOCK, RESET, READY) : 8284 A (8MHz) ó 8284 A-5 (5MHz)
  + Cristal Resonante para {F(μP)=4.77MHz}: 14.318 MHz =3*F(μP)

- Red R-C para generación de pulso de RESET para μP y periféricos (entra a terminal $\overline{RES}$ de 8284A para sincronización) : Switch (button) N.Open; Capacitor 1μF (@ ≥10 V); R = 560kΩ y 100Ω; Diodo 1N4001

- Lógica circuital para introducción de Estados de Espera (WAIT) en el sistema : 74LS164, 74LS10 y 74S04 a entrada RDY1 de 8284A

- Sistema de Buffers:
  - + para demultiplexar Bus de Direcciones : 74LS373
  - + para demultiplexar Bus de Datos : 74LS245
  - + para intensificar señal Bus de Control : 74LS244

- Circuitos decodificadores para Memoria y Puertos I/O (Entrada/Salida) : 74HC138 ó 74LS138 y lógica necesaria : (7427,74S04,74LS30)

- Memoria EPROM
  - + Tipo : 27C64 ó 2764 (UVEPROM) (8192 bytes) Tacc ≤ 300 [ns]

  Capacidad instalada : 8 kBytes Rango (FE000h-FFFFFh)

  Contenido :
  - Sistema Operativo para la interface
  - Rutinas definitivas para control e inicialización de periféricos
  - Rutinas de Comunicación Serie entre μC-interface y μC-PC (PS)
  - Rutinas de servicio a interrupción
  - Datos y Tablas de apuntadores iniciales del sistema
  - Otro código definitivo de control presente o futuro

* Memoria RAM

  + Tipo : 2016 {NMOS estática}
    (2048 bytes)
    Tacc ≤ 150 [ns]

  Capacidad instalada : 6 kBytes
    Rango { 00000h-017FFh }
    (1kB) para uso del sistema y
    (5kB) para empleo del usuario
    (6kB RAM instalados pero expandible hasta 16 kB)

  Capacidad decodificada : 16 kBytes
    Rango { 00000h-03FFFh }

  Contenido :
  - Tabla de Vectores de interrupción
  - Stack
  - Area de servicio para el sistema
  - Area para programas del usuario

* PERIFERICOS : { PUERTOS I/O } :

* Circuito Temporizador-Contador
  (para generación de baudajes,
  reloj ADC, u otros usos) : 8254 (@ 8 MHz)
  { *Programmable Interval Timer* }

* Puerto Serie (Entrada/Salida) En niveles RS-232
  Velocidad de Transmisión
  programable :
  300 , 1200 , 2400 ,
  4800 , 9600 [Bauds]

  Circuito (USART) Controlador de
  comunicación serie 8251 A
  { *Universal Synchronous & Asynchronous*
  Receiver & Transmitter }
  Funcionamiento en modo "Poleo" (Lectura de Registro STATUS) o
  por Interrupción (Sensado de RxRDY y TxRDY por IR0, IR1 _ 8259)

* Circuitos convertidores de nivel :

TTL ⟶ RS-232	{Tx}	MC 1488 (Cap.=560pF) (Fuentes: +12 & -12 V)
RS-232 ⟶ TTL	{Rx}	MC 1489 (Cap.=330pF) (Fuente: +5 V)

* Interrupciones — Hasta 8 interrupciones del tipo "enmascarable" con niveles de prioridad asignados

Circuito (PIC) Controlador de Interrupciones { *Priority Interrupt Controller* } — 8259 A

* Puertos Paralelos (Entrada/Salida) (PPI)
(6 puertos programables como Entrada/Salida, Handshaking o Bidireccionales (sólo PA)

{ *Programmable Peripheral Interface* }

{ 2 IC's } : 82C55 A [-2] (CHMOS) (8MHz): Zero Wait State Speed: Tww = 20 [ns] 3 puertos paralelos programables de 8-bit cada uno , por IC .

* PPI - #1 : Para manejo de DAC y ADC
  + Para manejo de DAC
    (PA = puerto de salida : 8 bits : Datos)
  + Para manejo de ADC
    (PB = puerto de salida : 5 bits : Control)
    (PC = puerto de entrada : 8 bits : Datos)
* PPI - #2 : Para empleo del usuario
  (3 puertos paralelos programables)
  Puertos PA y PB : { *Buffered* : 74LS245 }
  Puerto PC : { *Not Buffered* }

- Convertidor Digital/Analógico

  (DAC) — MC 1408 [P8]

  { 8-*bit multiplying D/A converter* } — Resolución : 8 bits; Max. error : ±0.19%; Settling time : 300ns

  Circuito de Referencia (V_{REF} = +2V)

  Diodo Zener 1N751A (@ 5.1V , 1/2W , max.P=400mW),
  Amp.Op. (Seguidor de Voltaje) LM307 ,
  Trimmpot 1kΩ (10 vueltas mín. para ajuste fino),
  Resistores (divisor de voltaje) {15kΩ y 10kΩ} @ 1/4W ,
  Capacitor 0.1μF (estabilidad)

  Amplificador Convertidor de Corriente a Voltaje
  (Salida I_{OUT} del DAC)

  Amp.Op. LF356
  R_o = (4.7kΩ)+(Trimmpot 1kΩ ...(1 vuelta))

- Convertidor Analógico/Digital

  (ADC) — ADC 0809 (National SC)

  { 8-*bit A/D converter* } — 8 entradas analógicas seleccionables por Multiplexor interno : IN0 (000) a IN7 (111); Resolución : 8 bits; Max. error : ± 1 LSBit

  Tiempo de Conversión :
  100 μs (typ) a F_c = 640 kHz
  En **SDM88-PC** ... alrededor de 50 μs pues frecuencia de operación (CLOCK) es : F_c = 1.19 MHz

  Circuito de Referencia y Alimentación (V_{REF} = +5.12V)

  Diodo Zener LM336 Z-5.0 (ajuste @ 5.12V ,
  R_{cal}=Trimmpot=10kΩ (10 vueltas))
  Amp.Op. (Seguidor de Voltaje integrado y Buffer) LM310

Resistores {5.6kΩ, 2.2kΩ, 22kΩ, 220kΩ} @ 1/4W ,
Capacitores 10μF (Tantalum) (2)
$V_{REF}^{+} = V_{cc} = 5.12$ [V]

Con este voltaje de Referencia el ADC tendrá 256 (2^8) pasos (steps) de 20 [mV] cada uno.

Operación en modo "Poleo" o "Interrupción" seleccionable por "jumper" : {Sensado de terminal EOC = *End-Of-Conversion*}

Operación en modo "Poleo" (Polling)
NOR (74LS02) de ($\overline{RD}$) y CS para ADC_EOC ($\overline{Y5}$)
Buffer Tri-state (74LS126) para leer estado de EOC por Bus de Datos del μP (D0) {Puerto IN de 1-bit}

Operación en modo "Interrupción" (Interrupt)
señal EOC dirigida a IR2 (int input) del PIC 8259A

- Sistema (circuito) indicador de "Status" en la operación del **SDM88-PC**
  (Bus de Datos μP) ⟶ (74LS373 como puerto de salida) ⟶
  ...⟶ (Decodificador BCD-7Seg: 7448) ⟶
  ...⟶ (Display numérico Cátodo común : MAN6780)

- Capacitores de 0.1 [μF] para eliminación (filtrado o "bypass") de ruido de alta frecuencia en las líneas de alimentación (voltaje de polarización) : [Colocados entre V_{cc} y tierra (GND), sobretodo en circuitos de alta o media integración (MSI, LSI, VLSI) y en circutos analógicos muy sensibles a variaciones del suministro de voltaje (desacoplo de fuentes)] El ruido, producido por la conmutación de estados lógicos de toda la circuiteria digital en las líneas de alimentación de las fuentes, puede ocasionar mal funcionamiento o errores intermitentes en la operación del sistema, si no es filtrado (o atenuado) apropiadamente.

* Fuentes de Alimentación :

+ 5 [V] DC	@ 2 [A] (mínimo)	[Regulada]
+ 12 [V] DC	@ 1 [A] (mínimo)	[Regulada]
- 12 [V] DC	@ 1 [A] (mínimo)	[Regulada]

Fuentes Reguladas y con capacitores de filtrado de valor elevado para eliminar ruido de baja frecuencia en las líneas de alimentación.

Trayectorias separadas para {GND-digital} y {GND-analógica} que se unen solamente en el punto de GND común de las 3 fuentes anteriores. (Con el fin de no introducir ruido debido a conmutación digital en circuitos analógicos).

En caso de tener fuentes de alimentación separadas +5V y ±12V, la SECUENCIA DE CONEXION para el sistema SDM88-PC debe ser :

A) Encendido

1) Unir las terminales GND de las 3 fuentes entre sí, y una vez hecho esto, conectar a la terminal GND del SDM88-PC . Conviene checar con multimetro en este momento que se tengan los voltajes correctos referidos a este punto común.

2) Conectar a terminal correspondiente fuente de +12 V

3) Conectar a terminal correspondiente fuente de -12 V

4) Conectar a terminal correspondiente fuente de +5 V

5) Conectar puerto Serie (RS-232) de Microcomputadora PC (PS) a puerto serie de SDM88-PC

B) Apagado

1) Desconectar enlace externo de puertos Serie

2) Desconectar terminal (viva) de fuente de +5 V

3) Desconectar terminal (viva) de fuente de -12 V

4) Desconectar terminal (viva) de fuente de +12 V

5) Desconectar terminal común de GND

Conviene que los procesos de conexión y desconexión duren el menor tiempo posible.

* Temperatura de Operación : 0 a +70 [°C]

La Interface (Microcomputadora-interface: Hardware externo) del sistema es manejada por una microcomputadora tipo PC o PS (IBM compatible), a través del puerto serie de la misma, bajo las especificaciones de la norma RS-232C :

Niveles de Voltaje :

SPACE (Espacio)	:	"0" lógico +3 a +15 [Volt]
MARK (Marca)	:	"1" lógico -3 a -15 [Volt]
Región de Transición	:	-3 a +3 [Volt]

La microcomputadora-interface entrega a la microcomputadora PC (PS) niveles de voltaje entre +12 y -12 [V] (por su línea Tx) a través del circuito integrado Convertidor de nivel TTL→RS-232 (1488), mientras que la μC PC (PS) entrega a la interface niveles entre +9 y -9 [V] (por su línea Tx) al circuito Convertidor de nivel RS-232→TTL (1489) en la línea Rx.

Las tierras (GND) de señal de la μC-Interface y de la μC-PC(PS) están conectadas entre sí, a fin de servir como referencia física para la identificación de niveles de voltaje como estados lógicos en los procesos de Transmisión/Recepción de información por Puerto Serie.

En el Sistema de Desarrollo **SDM88-PC** , el enlace con la Microcomputadora PC (PS) para hacer uso de los recursos que ésta ofrece, es parte fundamental en el funcionamiento y operación del sistema. La Microcomputadora PC es, a través de su teclado, el dispositivo por el cual el usuario, desencadena y sincroniza acciones en la Microcomputadora-interface (Hardware externo), organiza y transfiere información (bidireccionalmente) y visualiza resultados (status).

La Microcomputadora PC (PS) :

+ funciona como una terminal para el despliegue de información
+ interviene activamente como administradora del *Hardware* externo (microcomputadora-interface y/o circuitos de aplicación)
+ canaliza comandos para efectuar acciones especificas
+ maneja y organiza la información recibida (código, datos, programas {bytes}) para el microprocesador externo, almacenándola / recuperándola en/de su memoria o dispositivos de almacenamiento masivo (unidades de diskette o de disco duro)
+ arbitra la transferencia selectiva de bloques de información (bytes) desde o hacia la interface (memoria y/o puertos I/O externos)

Estas y otras muchas actividades son supervisadas y seleccionadas por el usuario a través de pantallas ordenadas en "Menúes". El Sistema cuenta con un Programa Monitor que permite al usuario cargar y ejecutar sus programas y operar sobre el *Hardware* de la microcomputadora-interface. Dicho Programa Monitor está dividido en dos partes : una residente en la EPROM de la microcomputadora-interface y otra residente en memoria RAM de la microcomputadora PC (PS) , después de cargarla de disco (a manera de Sistema Operativo para entrar en el ambiente del SDM88-PC).

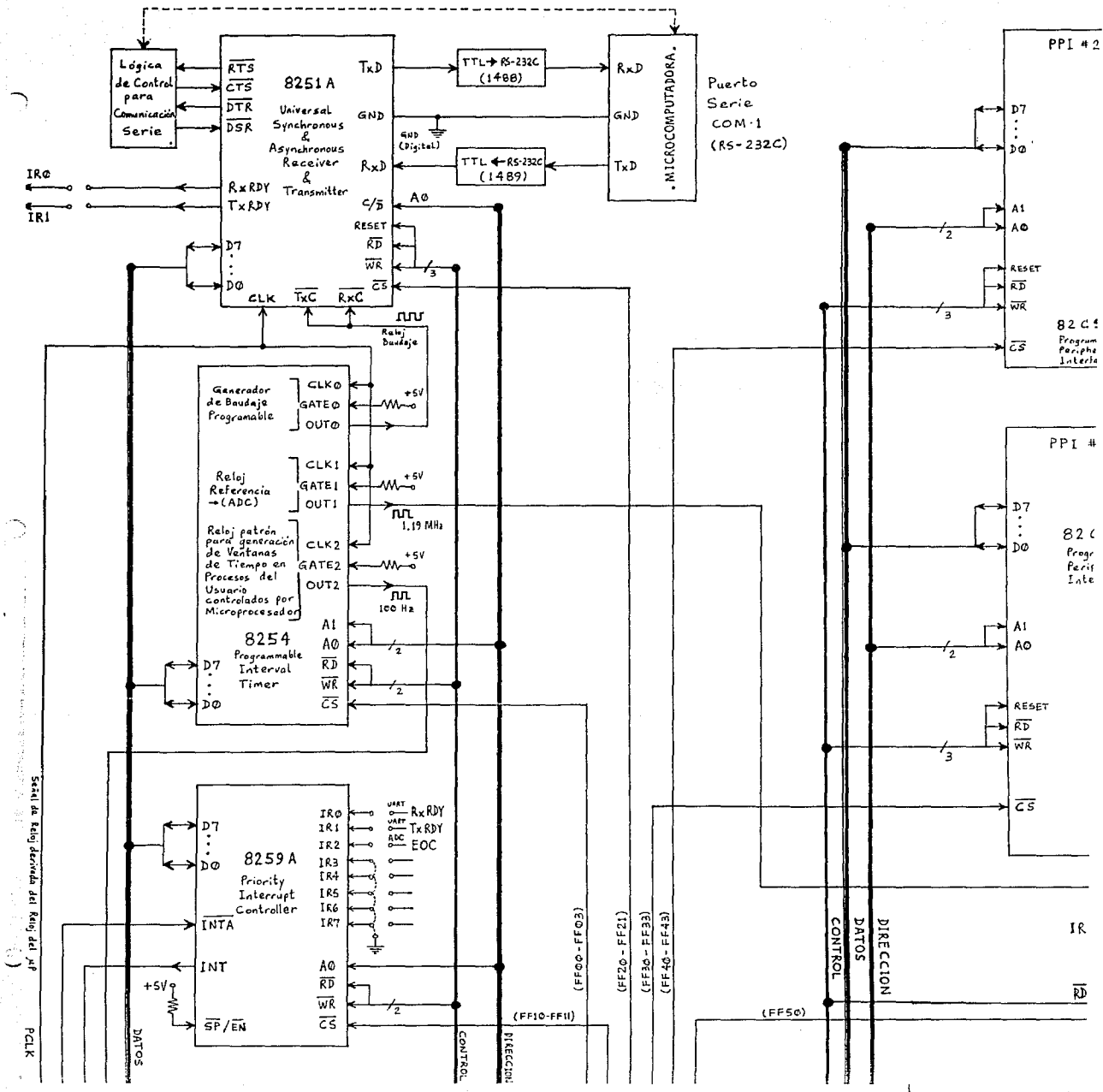
El Sistema de Desarrollo **SDM88-PC** , provee al diseñador de una arquitectura fundamental (CPU {Microprocesador}, Memoria RAM y EPROM, y Puertos I/O (Entrada/Salida)), ya probada y funcionando, para que a partir de su conocimiento y empleo, pueda fácilmente añadir, mediante la lógica de decodificación adecuada, los periféricos requeridos por su diseño : { timers, controladores jerárquicos de interrupción, controladores de comunicación avanzados, coprocesadores numéricos, sintetizadores digitales de voz o imagen, convertidores A/D y D/A, memoria dinámica o más memoria estática y/u otra circuitería de interface } , además de

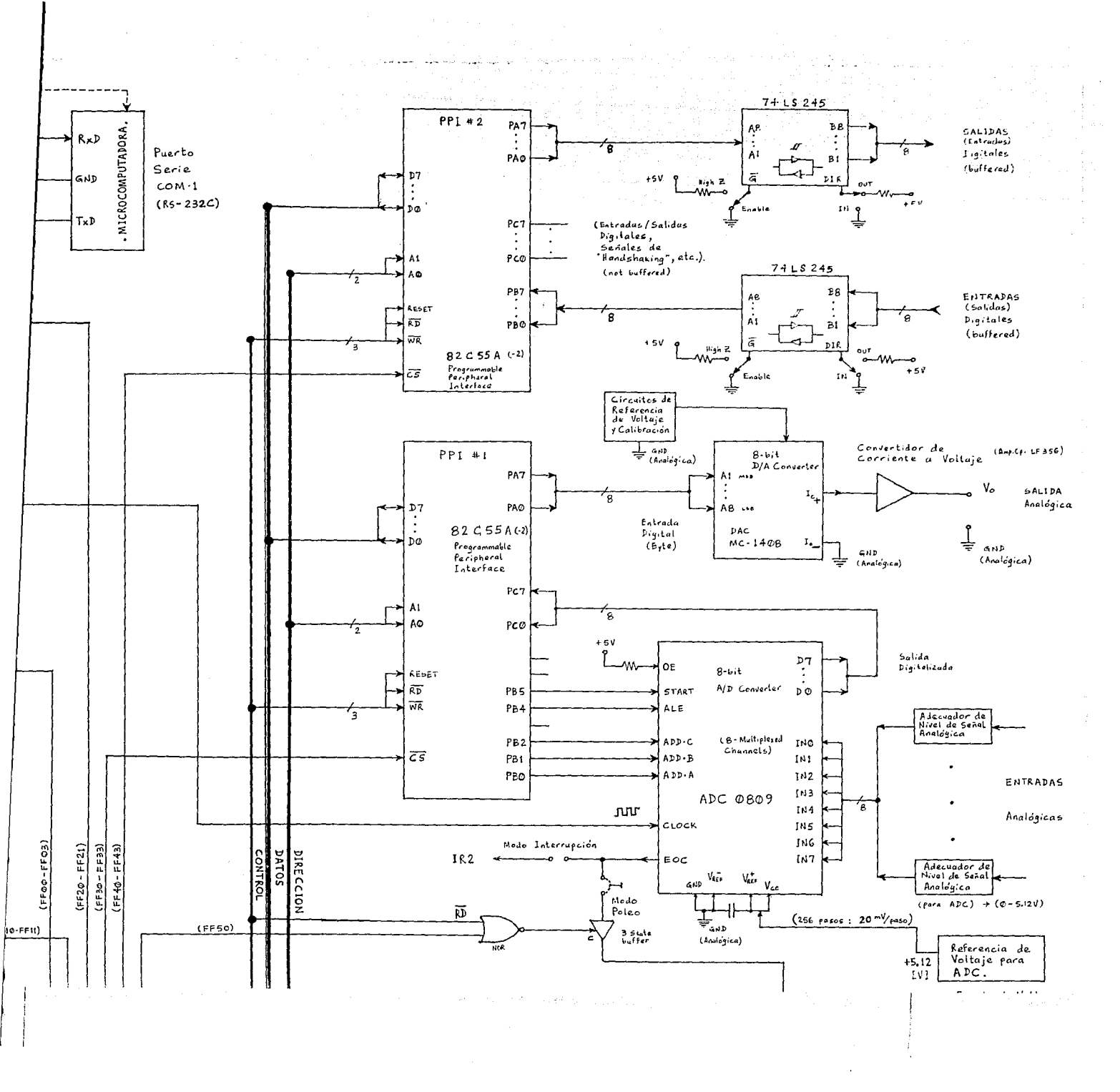
disponer de las herramientas necesarias para probar y depurar las secciones de *hardware* que vaya agregando al aprovechar las facilidades que el sistema ofrece para escribir y ejecutar módulos de *software* que manejen o actúen sobre dichos módulos de *hardware*. De esta manera, el diseñador podrá obtener rápidamente un prototipo armado y funcionando para ver si el instrumento de aplicación específica que busca desarrollar es "factible" de realizarse.

.....

En la { *figura* 2-1 } se muestra el Diagrama funcional completo de la arquitectura de la "Microcomputadora-Interface" del sistema de Desarrollo SDM88-PC .

.....


Lógica de Control para Comunicación Serie
RTS
CTS
DTR
DSR
8251 A
Universal Synchronous & Asynchronous Receiver & Transmitter
TxD
GND
RxD
C/D
RESET
RD
WR
CS
D7
D0
CLK
TxC
RxC
IR0
IR1
RxRDY
TxRDY
A0
TTL → RS-232C (1488)
TTL ← RS-232C (1489)
GND (Digital)
RxD
GND
TxD
• MICROCOMPUTADORA.
Puerto Serie COM·1 (RS-232C)
Reloj Baudaje
Generador de Baudaje Programable
CLK0
GATE0
OUT0
+5V
Reloj Referencia → (ADC)
CLK1
GATE1
OUT1
1.19 MHz
Reloj patrón para generación de Ventanas de Tiempo en Procesos del Usuario controlados por Microprocesador
CLK2
GATE2
OUT2
100 Hz
8254
Programmable Interval Timer
A1
A0
RD
WR
CS
8259 A
Priority Interrupt Controller
IR0
IR1
IR2
IR3
IR4
IR5
IR6
IR7
UART
USART
ADC
RxRDY
TxRDY
EOC
INTA
INT
SP/EN
PCLK
Señal de Reloj derivada del Reloj del µP
DATOS
CONTROL
DIRECCION
(FF00 - FF03)
(FF10-FF11)
(FF20 - FF21)
(FF30 - FF33)
(FF40 - FF43)
(FF50)
PPI # 2
PPI #
IR
RD


RxD
GND
TxD
• MICROCOMPUTADORA.
Puerto Serie COM-1 (RS-232C)
PPI #2
PA7
PA0
PC7
PC0
PB7
PB0
D7
D0
A1
A0
RESET
RD
WR
CS
82 C 55 A (-2)
Programmable Peripheral Interface
74 LS 245
AB
A1
BB
B1
G
DIR
Enable
IN
OUT
+5V
High Z
8
2
3
SALIDAS (Entradas) Digitales (buffered)
(Entradas / Salidas Digitales, Señales de "Handshaking", etc.). (not buffered)
ENTRADAS (Salidas) Digitales (buffered)
Circuitos de Referencia de Voltaje y Calibración
GND (Analógica)
PPI #1
8-bit D/A Converter
A1 msb
AB lsb
DAC MC-1408
Convertidor de Corriente a Voltaje
(Amp. Op. LF356)
Vo
SALIDA Analógica
Entrada Digital (Byte)
Salida Digitalizada
ADC 0809
OE
START
ALE
ADD-C
ADD-B
ADD-A
CLOCK
EOC
8-bit A/D Converter
(8-Multiplexed Channels)
IN0
IN1
IN2
IN3
IN4
IN5
IN6
IN7
Adecuador de Nivel de Señal Analógica
ENTRADAS Analógicas
(para ADC) → (0 - 5.12V)
(256 pasos : 20 mV/paso)
Referencia de Voltaje para ADC.
+5.12 [V]
Modo Interrupción
IR2
Modo Polco
3 State buffer
NOR
(FF00 - FF03)
(FF20 - FF21)
(FF30 - FF33)
(FF40 - FF43)
(FF50)
CONTROL
DATOS
DIRECCION

2 . 2
SEÑALES DE CONTROL Y SEÑALES DE DIRECCIONES / DATOS

En la Microcomputadora-Interface, el microprocesador 8088 trabaja en su modo de operación "Mínimo".

Terminal	Estado lógico
MN/$\overline{MX}$	H ("1")
$\overline{TEST}$	L ("0") . (para ejecución continua en μP)
	($\overline{TEST}$=1 ... idle state wait)
HOLD	L ("0")

El Bus (Canal) de señales de Control para el sistema consta de 12 señales :

- $\overline{RD}$
- $\overline{WR}$
- IO/$\overline{M}$ e $\overline{IO}$/M
- RESET OUT
- PCLK (Señal derivada (submúltiplo{1/2}) de CLK μP para sincronización de periféricos)
- CLK (μP)
- READY
- $\overline{DEN}$ y DT/$\overline{R}$
- ALE
- $\overline{INTA}$

Debido a su uso extensivo en el sistema (conexión a varios periféricos I/O y memoria), sólo las señales : $\overline{RD}$, $\overline{WR}$, IO/$\overline{M}$, $\overline{IO}$/M, PCLK y RESET OUT , se hacen pasar a través de intensificadores de señal (buffers) con el fin de satisfacer los requerimientos de voltaje TTL y de corriente exigidos para la correcta operación del sistema. El circuito utilizado para esto es un 74LS244 (*Octal*

Buffers/Line Drivers) (Buffers TTL unidireccionales). Estas 6 señales conforman un Bus de Control Principal (reducido). Las tres primeras son generadas por el μP 8088 y las dos últimas por el 8284A .

Las terminales que manejan tanto los Datos como las Direcciones deben ser demultiplexadas para poder manipular los Datos y las Direcciones en forma independiente.

Para los Datos se hace uso de un "*Transceiver*" (circuito que contiene tanto buffers transmisores como receptores juntos) para poder enviar (Write) o recibir (Read) datos hacia o desde cualquier parte del sistema (buffers bidireccionales). Tanto el sentido en que se transmiten los datos como el momento en el cual se realiza dicha transmisión son controlados por el microprocesador a través de las señales en las terminales $\overline{DEN}$ y DT/$\overline{R}$. Para realizar esta función se utilizó el circuito integrado 74LS245 (*Octal Bus Transceivers with Noninverted 3-State Outputs*). La señal $\overline{DEN}$ (Data Enable) generada por el microprocesador 8088, habilitará los buffers del Bus de Datos cuando sea verificada baja (Low). La señal DT/$\overline{R}$ (Data Transmit/Receive) también generada por el μP, se usa para especificar la dirección en que los buffers serán habilitados. Cuando DT/$\overline{R}$ es verificada alta (High) y si los buffers son habilitados por $\overline{DEN}$, transmitirán datos desde el μP hacia la localidad de Memoria (RAM) o Puerto I/O direccionado (Ciclo de Escritura (Write)). Cuando DT/$\overline{R}$ es verificada baja (Low) y si los buffers son habilitados por $\overline{DEN}$, permitirán el flujo de datos desde Memoria (ROM o RAM) o Puertos I/O hacia el μP (Ciclo de Lectura (Read)).

Las Direcciones deben estar presentes durante todo el tiempo en que se realiza una lectura/escritura de una memoria o de un puerto I/O; por tanto, se debe almacenar la dirección en la cual

se está trabajando. Esto se logra mediante "*latches*" con salida 3-estados. El momento en el cual la dirección es guardada está determinado por el borde de bajada de la señal en la terminal ALE del microprocesador. La señal ALE se emplea para habilitar los latches. La información mantenida a la salida de los latches después de la transición H-L de ALE, será A0-A19 . Los circuitos integrados (3) utilizados para el almacenamiento de las direcciones (20 bits), son 74LS373 (*Octal D-Type Transparent Latches with 3-State Outputs, common Output Control and common Enable*).

Resumiendo, la información de las lineas AD0 a AD7 del μP 8088 es demultiplexada de modo que el circuito 74LS245 captura la información correspondiente a los Datos (D0 a D7) mientras que los circuitos 74LS373 mantienen la información correspondiente a las Direcciones (A0 a A7 y A8 a A19).

2 . 3
GENERADOR DE RELOJ , RESET Y ESTADOS DE ESPERA

La señal de Reloj necesaria para operar a la Microcomputadora - Interface, (basada en el microprocesador 8088), se obtiene del circuito integrado 8284A (*Clock Generator and Driver for* iAPX 86,88 *Processors*). El 8284A está diseñado para generar y manejar el reloj de un sistema que use un procesador 8088/8086.

El circuito consta de un oscilador controlado por un cristal externo, un contador divisor-entre-tres y la lógica para sincronización de la señal READY (Multibus) y RESET.

El circuito oscilador está diseñado para usarse principalmente con un cristal resonante del cual se deriva la frecuencia básica de operación para el sistema. La frecuencia nominal del cristal debe ser tres veces la frecuencia de reloj requerida por el CPU; en el caso del **SDM88-PC** (si se usa un µP 8088 @ 5MHz) el cristal conectado al 8284A tiene una frecuencia de oscilación de 14.31818 [MHz], por lo que la frecuencia de reloj real enviada al 8088 es de 4.77273 [MHz]. El generador de reloj consiste en un contador síncrono divisor entre tres con una entrada especial (CSYNC) para sincronización de reloj y que debe mantenerse conectada a GND cuando se usa el oscilador interno (XTAL) para tener habilitado siempre el reloj del sistema. La entrada F/$\overline{C}$ permite seleccionar entre el cristal oscilador (Low) o la entrada EFI (High) como fuente de generación de reloj para el procesador. La salida de reloj en la terminal CLK tiene niveles MOS con un ciclo de trabajo de 33% y se conecta directamente a la entrada de reloj del µP. La salida en la terminal PCLK es una señal de reloj con niveles TTL cuya frecuencia es 1/2 de aquella en la salida CLK, con un ciclo de trabajo de 50% . PCLK es usada como una señal de reloj de propósito general dentro del sistema (sincronizada {como submúltiplo exacto} con la frecuencia maestra de reloj del procesador).

En el **SDM88-PC**, la señal PCLK sirve como reloj para el USART 8251A y como fuente de origen para otras frecuencias submúltiplo en el sistema generadas a través del contador-timer programable 8254 (reloj ADC, generador de Baudaje, Reloj de tiempo real, etc.).
En el **SDM88-PC**, PCLK tiene una frecuencia de 2.38636 [MHZ].

La señal de RESET de Hardware ($\overline{RST}$) y la señal RDY (para generación de estados de espera : "WAIT"), pasan también a través del 8284A para ser sincronizadas con la señal de reloj antes de ser enviadas al µP 8088.

La lógica de RESET proporciona una entrada $\overline{RES}$ para generar la señal RESET_OUT del sistema que podrá ser usada por los dispositivos que la requieran (8088, 8255, 8251). Esta señal es sincronizada con el borde de bajada de la señal CLK. La entrada $\overline{RES}$ que cuenta internamente con lógica "Schmitt trigger" es activada baja y genera la salida RESET de una duración adecuada dentro del 8284A . A fin de utilizar esta función del 8284A, se emplea una red RC para generar el pulso de RESET al aplicar potencia al sistema (*Power-Up* o RESET automático de encendido) o para efectuar un RESET manual a través de un interruptor (*Switch Push-button*...(Normalmente abierto)). La constante de tiempo de dicha red se calcula con base a las especificaciones que da el fabricante. (En **SDM88-PC** funcionó R=560kΩ y C=1μF).

Las entradas RDY1 y RDY2 del 8284A están disponibles para insertar estados de espera (WAIT) en un ciclo de máquina a fin de sincronizar al μP con dispositivos más lentos (ej. memoria EEPROM, puertos I/O lentos que responden a más baja velocidad que el μP, etc.); cada entrada tiene un calificador de habilitación respectivo (AEN1 y AEN2).

En el **SDM88-PC** la entrada RDY2 no es usada, mientras que la entrada RDY1 es controlada por una circuitería que puede introducir estados de espera (entre T3 y T4 de un ciclo de máquina) bajo distintas condiciones seleccionadas por *jumpers* :

a) introducción de estado(s) de espera sólo en operaciones de lectura/escritura de/en puertos I/O
b) introducción de estado(s) de espera en cualquier operación de lectura/escritura sea de/en puertos I/O o de/en memoria

La circuitería generadora de estados de espera (que en el **SDM88-PC** se basa en un registro de corrimiento entrada serie salida paralela de 8 bits (74LS164) animado por el reloj CLK del μP), permite seleccionar, mediante *jumpers*, la introducción del

número deseado de estados "WAIT" en un ciclo de máquina (desde 0 hasta 7). Si la entrada READY del 8088 es verificada baja, entonces el μP insertará el número deseado (establecido) de estados de espera (WAIT) en el ciclo de máquina que se está ejecutando en ese momento. Durante un estado WAIT, la información en los buses del sistema es mantenida constante (congelada): los estados lógicos en los buses al inicio del estado WAIT se preservan durante toda la duración de éste. La finalidad principal de insertar uno o más estados WAIT en un ciclo de máquina (instrucción) es la de dar a un dispositivo de memoria o puerto I/O direccionado, más tiempo para aceptar o enviar datos. Los estados de espera son introducidos al verificarse baja la entrada RDY1 del 8284A. Entonces el 8284A sincroniza internamente la señal de entrada RDY1 con la señal del reloj maestro del sistema y envía la señal resultante a la entrada READY del μP8088.

2.4
MEMORIA DEL SISTEMA (MAPA)

En la Microcomputadora-Interface la capacidad total para direccionamiento de memoria es de 1 MByte (1048576 Bytes) como corresponde al μP 8088/8086, aunque inicialmente sólo se tienen :

Decodificados :

16 kBytes de memoria RAM estática

8 kBytes de memoria UVEPROM

pero

Instalados :

6 kBytes de memoria RAM estática

8 kBytes de memoria UVEPROM

En el sistema se distinguen dos áreas de memoria de función específica : la parte baja del Mapa de Memoria está constituida por memoria RAM y la parte más alta por memoria UVEPROM .

MAPA DE MEMORIA (en la Microcomputadora-Interface del SDM88-PC)

	A19 ↓	DECO SELECT CB A	A0 ↓		
RAM :					
RAM (1) :	0000 00	00 0	000 0000 0000	00000	{ 2 kB }
			111 1111 1111	007FF	
RAM (2) :	0000 00	00 1	000 0000 0000	00800	{ 2 kB }
			111 1111 1111	00FFF	
RAM (3) :	0000 00	01 0	000 0000 0000	01000	{ 2 kB }
			111 1111 1111	017FF	
RAM (4) :	0000 00	01 1	000 0000 0000	01800	{ 2 kB }
			111 1111 1111	01FFF	
RAM (5) :	0000 00	10 0	000 0000 0000	02000	{ 2 kB }
			111 1111 1111	027FF	
RAM (6) :	0000 00	10 1	000 0000 0000	02800	{ 2 kB }
			111 1111 1111	02FFF	
RAM (7) :	0000 00	11 0	000 0000 0000	03000	{ 2 kB }
			111 1111 1111	037FF	
RAM (8) :	0000 00	11 1	000 0000 0000	03800	{ 2 kB }
			111 1111 1111	03FFF	

Memoria NO DECODIFICADA : 04000

⟶ FDFFF

	A19 ↓	A0 ↓		
UVEPROM :				
	1111 111	0 0000 0000 0000	FE000	{ 8 kB }
	1111 111	1 1111 1111 1111	FFFFF	

{2kB=2048}=>(11 líneas: A0-A10) ; {8kB=8192}=>(13 líneas: A0-A12)

Para las operaciones de acceso a memoria el pin IO/$\overline{M}$ del μP 8088 debe estar en estado lógico "0".

Los requerimientos de temporización del 8088 (@ 5MHz) establecen un tiempo de acceso (Tacc) máximo permisible de alrededor de 472 [ns] (a F(μP)=4.9MHz) para operar sobre las memorias bajo un esquema de "Cero tiempos de espera". Si a este valor se le resta el retardo de propagación inherente a los buffers *latches* 74LS373 del Bus de Direcciones (18 [ns]), se obtiene un tiempo neto máximo de 454 [ns] para Tacc (esto a 4.9 MHz, pero a 4.77 MHz existe un ligero margen adicional). Esta característica debe tenerse en cuenta al seleccionar los circuitos de memoria para el sistema.

MEMORIA RAM :

Para la memoria RAM se empleó el circuito integrado 2016. Esta es una memoria de Lectura/Escritura de Acceso Aleatorio estructurada en 2048 localidades de 8 bits (Bytes). Es completamente estática y tiene un tiempo de acceso máximo de 100, 120, 150 ó 200 [ns] según el sufijo (-10,-12,-15,-20) en la presentación "NMOS" o de 45, 55 ó 70 [ns] en la presentación "HMOS" (MCM2016H). La presentación NMOS satisface sobradamente los requerimientos de temporización del 8088 (@ 5MHz). Opera con fuente de +5 V . Todas sus entradas y salidas son compatibles con niveles TTL y tiene la capacidad de poner sus salidas en Alta Impedancia (Tri-state) cuando no es direccionada a través de su "Habilitador de Chip" ("Chip-Enable"). También puede utilizarse como equivalente la memoria estática CMOS 6116 (Tacc = 200 [ns]).

En el sistema, la decodificación de memoria RAM abarca las direcciones Físicas (reales o absolutas) :

00000 h a 03FFF h { 16 kBytes }

(espacio para 8 ICs 2016 seleccionados por el μP a través de un decodificador 3X8 74HC138 o 74LS138 : Entradas C,B,A=A13,A12,A11). Si se desea instalar mayor capacidad de memoria a la aquí señalada, será necesario decodificarla.

Actualmente sólo se encuentran instalados 3 ICs 2016 para un total de 6 kBytes de RAM (3 * {2k X 8} = 6144 Bytes) :

Direcciones Físicas en el sistema :

RAM (1) : 00000 → 007FF (hex)

RAM (2) : 00800 → 00FFF

RAM (3) : 01000 → 017FF

En el sistema SDM88-PC, la memoria RAM se usa como memoria de trabajo, en ella se guardan : los vectores de interrupción (direcciones absolutas CS:IP de las rutinas de Servicio a Interrupción), el Stack, los resultados intermedios de las operaciones que se realizan durante la ejecución del programa de control, los programas "código-comando" transferidos desde la μC-PC a la μC-Interface a través del puerto serie, así como los programas de aplicación desarrollados por el usuario en espera de ser ejecutados (después de ser cargados).

El MAPA DE MEMORIA FUNCIONAL de la RAM es :

Dirección Física	C o n t e n i d o
00000 h	Tabla de Vectores de Interrupción
.	(Int_Type_0) a (Int_Type_63)
.	{ 100h Bytes = 80h Words ...
00FFF h	transferidas desde UVEPROM a RAM }
00100 h	Area de Stack
.	Efectiva desde 00100h a 001FDh
.	Stack_Top = 001FE h
001FF h	{ 254 Bytes = 127 Words }
00200 h	Area de Memoria disponible para
.	expansión futura u otros usos del
002FF h	sistema

```
    00300 h                 Area de Servicio para el sistema
      .                     { Cargado de programas-comando
    003FF h                 transferidos desde µC PC }

    00400 h                 Area de Código y Datos de Programa
      .                     para el Usuario
    017FF h  ..........     { 5120  Bytes instalados }
ó   03FFF h  ..........     { 15360 Bytes decodificados }
```

Valores default para los Registros de Segmento en el µP 8088 durante accesos a RAM (operación en Area de RAM) :

```
CS = DS
CS = DS = 0030 h  .......{ por dirección física : 00300h }
SS      = 0010 h  .......{ por dirección física : 00100h }
     SP = 00FE h  .......{ 254 bytes = 127 words para
                           operaciones de stack }
                    [ 00100h + 00FEh = 001FEh = Stack_Top ]
```

Para autoejecución de programas-comando y ejecución de programas del usuario :

```
00300 h = Dirección Física Base para CS y DS en Area de RAM
        => CS = DS = 0030 h  { para condiciones de operación
                               Programas de Usuario }
```

Si el programa del Usuario es un archivo binario (*.COM) (código hex), debe ser cargado sin modificaciones (y sin el *header* extra de 100h bytes) directamente de disco o arreglo de memoria o buffer a la dirección física 00400h . Este programa no debería alterar CS=0030h (=DS) como base para trabajo en RAM pero nada impide hacerlo. Es conveniente que todos los "offsets" de "jumps" y "calls" absolutos del programa(s)-Usuario estén referidos a esta dirección base (00300h), por lo que es más aconsejable usar "jumps" y "calls" relativos o relocalizables en lo posible, como normalmente sucede con los "jumps" y "calls" intrasegmento.

Asi se tiene que, para la memoria RAM instalada (6kB), las direcciones fisicas en el sistema son :

RAM (1) : 00000 ⟶ 007FF h

RAM (2) : 00800 ⟶ 00FFF h

RAM (3) : 01000 ⟶ 017FF h

de donde :

- Area Exclusiva para el sistema : (direcciones fisicas)
  - Vectores de Interrupción : 00000 ⟶ 000FF h
  - Stack : 00100 ⟶ 001FF h
  - Expansión futura u otros usos : 00200 ⟶ 002FF h
- Area Accesible para el Usuario : { CS=0030h }
  - Area para uso del sistema :
    (cargado de programas-comando autoejecutables) :
    - Dirección Fisica :
      - 00300 ⟶ 003FF h
    - Offset dentro de CS :
      - 0000 ⟶ 00FF h
  - Area para programas (código & datos) del usuario :
    - Dirección Fisica :
      - 00400 ⟶ 017FF h
    - Offset dentro de CS :
      - 0100 ⟶ 14FF h

Resumiendo :

Para los 3 ICs RAM 2016 instalados, y considerando como Base de Segmento (CS=DS) = 0030h (por dirección fisica = 00300h), se tiene que los "offsets" válidos del Area de RAM accesible para el usuario van de {0000h} a {14FFh}, esto es, por chip :

RAM (1) = 0000 ⟶ 04FF h

RAM (2) = 0500 ⟶ 0CFF h

RAM (3) = 0D00 ⟶ 14FF h

pero el usuario debe cargar sus programas de aplicación a partir del offset 0100h , ya que del offset 0000 al 00FFh se considera área ocupada por el sistema para cargar Programas-comando.

MEMORIA UVEPROM :

Para la memoria EPROM se empleó el circuito integrado 2764. Esta es una memoria sólo de Lectura, reprogramable eléctricamente y borrable con luz ultravioleta, estructurada en 8192 localidades de 8 bits (Bytes). Es completamente estática y tiene un tiempo de acceso máximo de 200, 250 ó 300 [ns] según el sufijo (-20,-25,-30) en la presentación "NMOS" o de 150, 200, 250 ó 300 [ns] en la presentación "HVCMOS" (TMS27C64). Ambas presentaciones satisfacen sobradamente los requerimientos de temporización del 8088 (@ 5MHz). Opera con fuente de +5 V (en modo de Lectura). Todas sus entradas y salidas son compatibles con niveles TTL y tiene la capacidad de poner sus salidas en Alta Impedancia (Tri-state) cuando no es direccionada a través de su "Habilitador de Chip" ("Chip-Enable") $\overline{CE}$ ó de su "Habilitador de Salidas" ("Output Enable") $\overline{OE}$. Para el modo de programación se requiere una fuente de +25V para la 2764 ó de +12.5V para la 2764A ó la 27C64 (voltaje Vpp), pero todas las señales de programación son nivel TTL. Para programación fuera del sistema de µP, pueden usarse programadores de EPROMs comerciales como el desarrollado por *Modular Circuit Technologies* : "EPROM Writer V-1.0", Model: MCT-EPROM (C)1987 (EPROM Writer Card for IBM-PC/XT,AT). Las localidades pueden ser programadas de una en una, por bloques o de forma aleatoria. Para el conocimiento de cada uno de los 7 modos de operación de esta memoria así como de otros parámetros importantes, referirse a los respectivos manuales de MEMORIAS de Intel (2764 & 2764A), Fujitsu (2764) o Texas Instruments (TMS27C64).

Actualmente el sistema tiene una capacidad instalada de memoria UVEPROM de 8 kBytes abarcando las direcciones Físicas (reales o absolutas) :

FE000 h a FFFFF h { 8 kBytes }

(espacio para 1 IC 2764 seleccionado por el µP a través de lógica de decodificación TTL).

En el sistema **SDM88-PC**, la memoria UVEPROM se usa para almacenar :

- el programa monitor o sistema operativo para la µC-Interface (Hardware externo a la µC-PC (PS))
- las rutinas de inicialización del sistema (RESET)
- las rutinas de control para periféricos
- las rutinas de comunicación {µC-Interface} ⟷ {µC-PC(PS)} a través de puertos serie
- las rutinas de servicio a interrupción
- otro código definitivo de control (presente o futuro (expansión)), agregado por el usuario

Valores default para los Registros de Segmento en el µP 8088 durante accesos a EPROM (operación en Area de EPROM) :

CS = DS ó CS ≠ DS

CS = FE00 h .....{ por dirección física : FE000h } (EPROM)

DS = FE00 h (en EPROM) ó DS = 0030 h (en RAM)

...{FE000h} ...{00300h}

SS = 0010 h .....{ por dirección física : 00100h }

(permanece en RAM)

SP = 00FE h

[00100h + 00FEh = 001FEh = Stack_Top]

El MAPA DE MEMORIA FUNCIONAL de la EPROM es :

Localidad de Memoria en EPROM (offset hex) (=IP=offset si CS=FE00h)	C o n t e n i d o
0000	Area Reservada (no usada) :
.	para Expansión futura u otros usos
.	o por compatibilidad con el formato
.	*.COM de IBM-PC (asignación de PSP
.	at run time) : { 100h Bytes }.

```
    .                   Esta convención da la facilidad de
    .                   desarrollar rutinas de EPROM en una
    .                   PC y cargar (grabar) directamente
    00FF                el *.COM generado a la EPROM.
                                { *.COM = archivo binario }
    0100                Area de Salto (Jump) o derivación
    .                   (Case) inicial.
    .                   Salto al área principal de código
    .                   de programa (offset=0500h) o
    .                   rutinas iniciales para derivación
    .                   de control (estructura tipo CASE).
    .                   Salto inicial a la zona de código
    .                   ejecutable o bien rutina para
    .                   derivación de control p.ej. a
    .                   distintas aplicaciones o a
    .                   distintas opciones de
    .                   inicialización de periféricos del
    .                   sistema, grabadas en EPROM.
    01FF                { 100h Bytes }
```

Apuntadores (Pointers) de interrupción (Tabla de Vectores) (Direcciones absolutas {CS:IP} de rutinas de servicio a interrupción).

Apuntadores de interrupción dedicados (5) :

```
    0200                INT_type0 pointer :
     ...                "Divide by Zero error"
    0203
    0204                INT_type1 pointer :
     ...                "Single-step" (trace)
    0207
    0208                offset:   IPLOW     INT_type2 ptr
    0209                          IPHIGH    Non-Maskable
    020A                Seg.Base: CSLOW
    020B                          CSHIGH
```

```
   020C                    INT_type3 pointer :
    ...                    "1-byte Interrupt instruction"
   020F                    (interrupción por software)
   0210                    INT_type4 pointer :
    ...                    "Overflow"
   0213
```

Apuntadores de interrupción reservados (para compatibilidad con futuros productos de Intel) (27) :

```
   0214
    .
    ...                    INT_type5 ⟶ INT_type31 pointers
    .
   027F
```

Apuntadores de interrupción disponibles (32) :

(type32 a type63 ...no se implementó hasta el type255 posible)

Vectores de interrupción para el PIC 8259A instalado (8) :

```
   0280                    INT_type32 pointer  {HIGH PRIORITY}
    ...                    8259A (IR0) ... USART RxRDY
   0283                    "Receive_Byte (8251A ⟵ PC)"
   0284                    INT_type33 pointer
    ...                    8259A (IR1) ... USART TxRDY
   0287                    "Transmit_Byte (8251A ⟶ PC)"
   0288                    INT_type34 pointer
    ...                    8259A (IR2) ... ADC  EOC (L-H)
   028B                    (Read A/D data digitized)
   028C                    INT_type35 pointer
    ...                    8259A (IR3)
   028F
   0290                    INT_type36 pointer
    ...                    8259A (IR4)
   0293
   0294                    INT_type37 pointer
    ...                    8259A (IR5)
   0297
```

```
        0298                    INT_type38 pointer
         ...                    8259A (IR6)
        029B
        029C                    INT_type39 pointer  {LOW  PRIORITY}
         ...                    8259A (IR7)
        029F
      Vectores de interrupción No usados pero Disponibles (24) :
        02A0                    INT_type40 pointer
         ...
        02A3
        02A4                    INT_type41 pointer
         ...                       |
          .                        ↓
        02FB                    INT_type62 pointer
        02FC                    INT_type63 pointer
         ...
        02FF

"System Data Area" :
        0300                    Area de Datos para el Sistema
          .                     (Inicialización de variables que el
         ...                    sistema puede usar ... locs.mem.)
          .                     { 200h = 512 bytes }
        04FF

"Main Program Code Area" :
        0500                    Inicialización  de  periféricos
          .                     (puertos I/O) del sistema y áreas
          .                     de servicio en memoria
          .                     {SS, SP, DS, ES, (CS via FAR jump)}
          .                     {Tabla de Vectores de Interrupción}
          .                     {Rutinas  de  Control  Principal  :
          .                     transferencia  de  información_bytes
          .                     [μC-PC] ⟷ [μC-Interface] },
```

```
    .                   {Rutina NMI},
    .                   {otro código definitivo (firmware)
    .                   para rutinas de aplicación
    .                   especifica}, etc.
  0FFF                  { B00h = 2816 Bytes }
  1000                  {Subrutinas de propósito general
    .                   para el control de dispositivos},
    .                   {Rutinas _ Servicio a Interrupción:
    .                   USART Rx & Tx Int. , ADC_EOC Int. ,
    .                   etc.}
  1FEF                  { FF0h = 4080 Bytes }
```

Area de codificación de Salto Inicial después de RESET de Hardware o encendido del sistema : ("Initial {RESET} Startup-jump area") :

```
(15 bytes):          contenido hex .. mnemónicos
  1FF0                  90      nop
  1FF1                  90      nop
  1FF2                  EA      jmp     ; FAR jump code
                                        ; (inter-segment)
  1FF3                  00      IPLOW
  1FF4                  01      IPHIGH  ; ... IP = 0100h
  1FF5                  00      CSLOW
  1FF6                  FE      CSHIGH  ; ... CS = FE00h
                                ; Startup RESET jump
                                ; Dirección Física :
                                ;       FE000
                                ;     + 0100
                                ;       -----
                                ;       FE100   {UVEPROM}
  1FF7
    .                   área no usada
  1FFF
```

MEMORIA EEPROM :

Debido a la compatibilidad pin a pin existente entre las memorias 2716 (EPROM) y 2816 (EEPROM) con la memoria 2016 (RAM), pueden instalarse dichas memorias , si así se requiere para alguna aplicación, en el espacio decodificado para memoria RAM existente en la µC-Interface del **SDM88-PC**. La EEPROM XL2816A (EXEL) es una memoria de sólo Lectura, reprogramable y borrable eléctricamente, organizada en 2048 bytes, con ciclos de temporización muy similares a los de una RAM estática (tiempo de acceso en lectura desde 250 ns). Todas las entradas de control (incluyendo el pulso de borrado/escritura en $\overline{WE}$) son TTL compatibles, igual que todas las entradas y salidas de datos. La modificación de byte en el modo de programación-5V se inicia con un pulso TTL bajo de escritura de 150 ns; la información de los buses de Dirección y de Datos es capturada (*latcheada*) internamente, liberando al sistema para otras tareas durante el periodo de escritura ("Automatic Write Time-out": no requiere µP-WAIT). La XL2816A borra automáticamente el byte seleccionado antes de escribir el nuevo dato, completando un ciclo de borrado/escritura en un máximo de 10 ms (en este ciclo las terminales I/O están en Alta Z). Cuenta con mecanismos para evitar falsas escrituras ("power-up & down, power-noise" (ruido)). Degradación : soporta hasta 10000 ciclos de escritura por byte. Esta memoria EEPROM puede conformar un área de trabajo utilizada para almacenar los parámetros de operación del sistema en alguna aplicación y puede ser accesada por el usuario en cualquier momento; asimismo, puede emplearse para guardar resultados cuando se desea llevar el seguimiento de las variaciones en los resultados que registra un experimento en un periodo de tiempo determinado o para cualquier otro uso de almacenamiento permanente de datos que desee darle el usuario (ej. equipo auto-calibrante, almacenamiento de claves para cifrado de información con fines de seguridad (Data Encryption), generadores programables de caracteres, mapeo de terreno en aviación, etc.).

2.5
PUERTOS I/O (ENTRADA/SALIDA) DEL SISTEMA

El usar un decodificador que traduzca direcciones de memoria a señales de habilitación ("Chip Select") para puertos de entrada/salida, se conoce como "Técnica de Entrada/Salida por mapeo en memoria" o "Entrada/Salida mapeada desde memoria" (*Memory-mapped I/O*). En esta técnica de direccionamiento, la información será "escrita a" o "leida de" un puerto en la misma forma que se haria con cualquier localidad de memoria. Por ejemplo, en un sistema basado en el µP 8088, se usaría una instrucción como : { MOV AL,DS:BYTE PTR [0C000h] } para leer un byte_dato del puerto "C000h" al registro AL, en vez de usar las instrucciones : { MOV DX,0C000h } , { IN AL,DX }.

La ventaja de usar esta técnica consiste en que puede usarse cualquier instrucción que se refiera a manejo de memoria para escribir o leer datos en puertos I/O; asi por ejemplo, la simple instrucción: { ADD AL,DS:BYTE PTR [0C000h] } , puede ser usada para LEER un byte_dato de la dirección de puerto "C000h" y SUMAR dicho byte al contenido del registro AL.

La desventaja de la técnica radica en que parte del espacio para direccionamiento de memoria en el sistema se está ocupando para accesar puertos y por tanto ya no está disponible para memoria (reducción del espacio para direccionamiento de memoria).

Se puede usar la técnica de Entrada/Salida por mapeo en memoria con cualquier microprocesador, pero algunos de ellos, como los de la familia 8088/8086 de Intel, permiten establecer un espacio de direccionamiento separado para puertos de entrada y de salida. Se accesan puertos en estos espacios de direccionamiento separados, directamente con las instrucciones "IN" y "OUT". Esta técnica se conoce como "Entrada/Salida directa" (*Direct I/O*).

La ventaja de este método consiste en que no se consume nada del espacio de memoria del sistema para accesar puertos.

La desventaja radica en que sólo se pueden usar las instrucciones especializadas IN y OUT para leer o escribir datos.

En la μC-Interface **SDM88-PC** se utiliza la técnica "Entrada/Salida directa" para direccionar dispositivos I/O, empleándose para su habilitación selectiva, un decodificador 3X8 : 74HC138 ó 74LS138 que es comandado por lineas del Bus de Direcciones provinientes del μP y por la señal de control IO/$\overline{M}$.

Cabe señalar que con el μP 8088 se pueden direccionar hasta 65536 puertos diferentes { 0000 a FFFF h } , aunque dificilmente se puede llegar a usar alguna vez tal cantidad de puertos en un sistema digital.

2.5.1
MAPA DE PUERTOS I/O EN LA MICROCOMPUTADORA-INTERFACE DENTRO DEL SISTEMA SDM88-PC

Puertos I/O = {Input/Output} = {Entrada/Salida}

Nota (1) :
Para las operaciones de acceso a Puertos I/O el pin IO/$\overline{M}$ del μP 8088 debe estar en estado lógico "1".

Nota (2) :
Los bits {A14-A10}, {A8,A7} y {A3,A2}, son considerados como estados irrelevantes ("don't care" = x), por tanto, no es necesario conectarlos físicamente agragando lógica de compuertas adicional al decodificador de Puertos {DEC3x8,74HC138} que podría introducir retardos inaceptables para accesar los dispositivos I/O.

MAPA DE PUERTOS I/O (Microcomputadora-Interface del SDM88-PC)

μP 8088

BUS DE DIRECCIONES

	IO/$\overline{M}$ ↓		A15 ↓		DECO SELECT CBA	A0 ↓	
8254 :	1	$\overline{Y0}$	1xxx xx1x x		000	xx00	= FF00
						xx01	= FF01
						xx10	= FF02
						xx11	= FF03
8259A :	1	$\overline{Y1}$	1xxx xx1x x		001	xxx0	= FF10
						xxx1	= FF11
8251A :	1	$\overline{Y2}$	1xxx xx1x x		010	xxx0	= FF20
						xxx1	= FF21
82C55A(-2) :	1	$\overline{Y3}$	1xxx xx1x x		011	xx00	= FF30
						xx01	= FF31
						xx10	= FF32
						xx11	= FF33
82C55A(-2) :	1	$\overline{Y4}$	1xxx xx1x x		100	xx00	= FF40
						xx01	= FF41
						xx10	= FF42
						xx11	= FF43
74LS126 :	1	$\overline{Y5}$	1xxx xx1x x		101	xxxx	= FF50
disponible :	1	$\overline{Y6}$	1xxx xx1x x		110	----	= FF60
74LS373 :	1	$\overline{Y7}$	1xxx xx1x x		111	xxxx	= FF70

MAPA FUNCIONAL DE PUERTOS I/O :

8254 : *Programmable Interval Timer*

Dirección hex	Nombre del Registro	Función en el IC y función asignada dentro del sistema
FF00 :	Counter0	Counter 0 (Write)/R (Mode3) Generador de Baudaje para USART (8251A)
FF01 :	Counter1	Counter 1 (Write)/R (Mode3) Reloj para el Convertidor A/D (ADC-0809)
FF02 :	Counter2	Counter 2 (Write)/R Contador/Timer disponible para aplicaciones-usuario (usado actualmente para producir onda cuadrada de 100 Hz para disparar NMI y actualizar cuenta en rutina para generar retardos (delays) programables con resolución de 1/100 seg.)
FF03 :	PITimerControlW	Control Word Register (Write) : Prog._Config.

8259A : *Priority Interrupt Controller*

Dirección hex	Nombre del Registro	Función en el IC y función asignada dentro del sistema
FF10 :	PIntC0	{ICW1,OCW2,OCW3} (Write)
FF11 :	PIntC1	{ICW2,ICW4,ICW3,OCW1} (Write)

ICW = Initialization Command Word
OCW = Operation Command Word

8251A : *Universal Synchronous/Asynchronous Receiver & Transmitter*

Dirección hex	Nombre del Registro	Función en el IC y función asignada dentro del sistema
FF20 :	UARTdata	Read/Write USART Data Comunicación Serie con µC IBM PC (PS) (data to/from µP)
FF21 :	UARTControlStat	Write USART Control Word / Read USART Status Word Register (Read/Write)

82C55A (-2) : *Programmable Peripheral Interface*
(Puertos Paralelos : IC # 1) : PPI # 1

Dirección hex	Nombre del Registro	Función en el IC y función asignada dentro del sistema
FF30 :	PPort1A	Port A R/W (Write) OUTPUT : [PA0-PA7] D/A Converter data : 8 bits
FF31 :	PPort1B	Port B R/W (Write) OUTPUT : [PB0-PB7] A/D C. Control : (Channel Address, ALE, START)
FF32 :	PPort1C	Port C R/W (Read) INPUT : [PC0-PC7] A/D C. digitized data : 8 bits (ADC-0809 : 8-bit , µP- -compatible A/D converter with 8 channel multiplexer)
FF33 :	PPI1ControlW	PPI Control Word Register (Write) : Programming Configuration Word

82C55A (-2) : *Programmable Peripheral Interface*

(Puertos Paralelos : IC # 2) : PPI # 2

Dirección hex	Nombre del Registro	Función en el IC y función asignada dentro del sistema
FF40 :	PPort2A	Port A R/W (Write) OUTPUT : [PA0-PA7] Puerto SALIDA de propósito general : 8 bits
FF41 :	PPort2B	Port B R/W (Read) INPUT : [PB0-PB7] Puerto ENTRADA de propósito general : 8 bits
FF42 :	PPort2C	Port C R/W (Read) INPUT (default) ó (Write) OUTPUT [PC0-PC7] : 8 bits Disponible p' aplicaciones del usuario . Programable como : INPUT, OUTPUT : (Mode 0) ó Handshaking : (Modes 1 & 2)
FF43 :	PPI2ControlW	PPI Control Word Register (Write) : Programming Configuration Word

En PPI # 2 :

PPort2A y PPort2B están "*buffereados*" (74LS245-Bidirectional Buffer) ; PPort2C no está "*buffereado*".

Cada uno de los 3 puertos puede ser reprogramado por el usuario (escribiendo una nueva palabra de control (dir: FF43h)) según sus necesidades.

El sentido IN/OUT de los buffers 74LS245 debe ser modificado correspondientemente:

pin (1) = {DIR} : IN=(GND) ó OUT=(+5V)

74LS126 : *Bus Buffer Gate with Tri-State output*
(Puerto de 1 bit : INPUT)

Dirección hex	Nombre del Registro	Función asignada dentro del sistema
FF50 :	ADC_EOC_port	1-bit INPUT Port (Read) 74LS126 Output Enable = "C" (La Salida está en alta impedancia (deshabilitada) cuando la entrada de control "C" del 74LS126 es: "C"= LOW ("0"). "C" = ($\overline{Y5}$) NOR ($\overline{RD}$) El (1/4) 74LS126 conecta la salida "EOC" del ADC con el bit "D0" del Bus de Datos del µP.

Dirección (I/O) disponible

Dirección hex	Nombre del Registro	Función asignada dentro del sistema
FF60 :	----------	Dirección Base disponible para puertos I/O

74LS373 : *Octal D-Type Transparent Latches*
(usado como puerto Paralelo de Salida)

Dirección hex	Nombre del Registro	Función asignada dentro del sistema
FF70 :	DisplayPort (LED)	Display para Reporte del Status_operación d' sistema (Write) OUTPUT : [Q0-Q7] NOT ($\overline{Y7}$) = Enable_Latch para el 74LS373_OUTPUT_Port

2.5.2

8254 : CONTADOR - TEMPORIZADOR PROGRAMABLE : PROGRAMMABLE INTERVAL TIMER

El 8254 es un dispositivo contador/temporizador diseñado para resolver problemas comunes de control de temporización en sistemas de microcomputadoras. Consta de tres contadores de 16 bits independientes, cada uno de los cuales es capaz de manejar entradas de reloj desde DC hasta 5 MHz (8254-5), 8 MHz (8254) y 10 MHz (8254-2). Los contadores pueden funcionar en modo Binario o BCD. El 8254 cuenta con un comando de lectura latcheada de Status y de cuenta (Read-back command). El 8254 es compatible con todos los μP Intel y otros μP y es una mejora del 8253 ; está construido con tecnología HMOS. Requiere de fuente sencilla de +5 V para operar.

El 8254 es un dispositivo de multi-temporización de propósito general que puede ser tratado como un arreglo de puertos I/O a través del *software* del sistema.

El 8254 resuelve uno de los problemas más comunes en cualquier sistema basado en microprocesador : la generación de retardos (delays) de tiempo precisos bajo control de software. En vez de implementar ciclos (loops) de temporización en software, el programador configura al 8254 para que se ajuste a sus requerimientos y programa a uno de los contadores para el retardo deseado. Una vez concluido dicho retardo, el 8254 puede interrumpir al CPU. El "overhead" de software es mínimo y pueden implementarse fácilmente retardos de longitud variable. Esta técnica evita que el CPU quede dedicado a servir a rutinas de conteo para generar retardos por poleo, consumiendo tiempo útil de procesamiento.

Algunas de las funciones de conteo/temporización de uso común en las microcomputadoras y que pueden implementarse con el 8254 son : Reloj de Tiempo Real, Contador de Eventos, ONE-SHOT digital, Generador de Rateo o Baudaje (*Rate*) programable, Generador de Onda Cuadrada, Multiplicador Binario de Frecuencia, Generador de Formas de Onda complejas, Controlador de motores complejos, Frecuencimetro, etc.

El 8254 cuenta con 6 Modos-Contador de operación programables por software :

Modo 0 : Interrupt on Terminal Count
Modo 1 : Hardware Retriggerable One-Shot
Modo 2 : Rate Generator
Modo 3 : Square Wave Mode
Modo 4 : Software Triggered Strobe
Modo 5 : Hardware Triggered Strobe (Retriggerable)

Cada uno de los 3 contadores puede ser programado independientemente. Los contadores son descendentes. Las operaciones de "Carga de palabra de cuenta inicial" y de "decremento de cuenta" se verifican con el flanco de bajada (transición H-L) de la entrada CLK de un contador.

La mayor cuenta inicial posible es "0"; esto es equivalente a 2^{16} para cuenta binaria y 10^4 para cuenta BCD.

2.5.3

8259A : CONTROLADOR DE INTERRUPCIONES PROGRAMABLE : PROGRAMMABLE INTERRUPT CONTROLLER (PIC)

El Sistema SDM88-PC puede manejar hasta 8 interrupciones del tipo "enmascarable" con niveles de interrupción jerarquizados. La implementación del mecanismo de atención a dichas interrupciones

se logra a través del circuito Controlador Jerárquico de Interrupciones PIC 8259A, que proporciona gran flexibilidad en cuanto al manejo y a la configuración de los niveles de prioridad asignados a cada interrupción.

El PIC 8259A está diseñado para manejar interrupciones en sistemas de microcomputadora en tiempo real. Es un controlador de interrupciones programable con capacidad de manejar hasta 8 interrupciones con niveles de prioridad asignados. En el modo de "Prioridad fija", la entrada IR0 del IC tiene la más alta prioridad (más importante, primera en ser atendida), la entrada IR1 tiene la siguiente más alta prioridad, y así sucesivamente hasta llegar a la entrada IR7 que tiene la más baja prioridad. Esto significa que si dos señales de interrupción ocurren al mismo tiempo, el 8259A dará servicio primero a la que tenga más alta prioridad, asumiendo que ambas entradas no están enmascaradas (están habilitadas) en el registro de Enmascaramiento (Interrupt Mask Register: IMR) del 8259A :

Entradas de Interrupción individuales:

deshabilitar = enmascarar (disable = mask)
habilitar = desenmascarar (enable = unmask)

El 8259A es programado por el software del sistema como un periférico de Entrada/Salida y ofrece al programador una variedad de modos de prioridad a fin de que pueda configurar la manera en que las solicitudes de interrupción son procesadas por el 8259A para ajustarse a los requerimientos del sistema, (la operación del circuito en sus distintos modos puede ser programada bajo control del programa monitor). Los modos de prioridad pueden ser cambiados o reconfigurados dinámicamente en cualquier momento durante la ejecución del programa principal. Esto significa que la estructura de atención a interrupciones puede ser definida como se requiera, basándose en el "ambiente" global de operación establecido en el sistema en un momento dado.

El PIC cuenta también con la circuitería interna para operar de un modo "en cascada", con otros ICs 8259A, para manejar hasta 64 niveles de interrupciones.

Si la bandera de Interrupción del μP 8088/8086 (Interrupt Flag) es habilitada ("1") con la instrucción STI (Set Interrupt Flag) y la entrada INTR del μP recibe una señal "High" (activación de INTR por nivel y no por flanco L-H como sucede con NMI), el μP 8088/8086 realizará las siguientes acciones:

1) Cargará automáticamente el registro de banderas en el stack (Push Flags).
2) Reseteará IF (Interrupt Flag) y TF (Trace Flag).
3) Introducirá en el stack la dirección de la siguiente instrucción a ejecutarse antes de producirse la interrupción (Return address CS:IP).
4) Pondrá el bus de datos en modo de entrada (input mode).
5) Mandará dos pulsos de reconocimiento de interrupción (Interruput Acknowledge) a través de su pin $\overline{\text{INTA}}$. Los pulsos $\overline{\text{INTA}}$ indican a algún dispositivo (hardware) externo como el 8259A, que debe enviar el "Tipo de Interrupción" deseado al μP 8088/8086.
6) Cuando el μP recibe el "Tipo de Interrupción" del dispositivo externo, lo multiplicará por 4 para producir una dirección en la tabla de apuntadores de interrupción (RAM).
7) A partir de esa dirección y de las tres siguientes direcciones consecutivas (bytes), el μP obtendrá los valores de IP y CS para el inicio de la rutina de servicio a la interrupción (ISR). Tan pronto como estos valores son cargados en CS e IP, el μP ejecutará entonces la ISR.
8) En la ISR puede empezarse por guardar en el stack los registros que serán alterados en ella y que se requiere conservar sin cambio para el programa principal (PUSH regs.). Al finalizar la ISR, deberá recobrarse el valor

original de dichos registros afectados por la rutina (POP regs.) y mandarse una Palabra-comando que resetee el bit "x" del registro "In-Service" (EOI = End Of Interrupt command), ("x" es el # de entrada IRx INT que fue servida, p.ej. x=2 si IR2 , x=4 si IR4 , etc.), a fin de que interrupciones de prioridad más baja puedan ser servidas posteriormente, y finalmente, debe ejecutarse la instrucción "IRET" para que automáticamente se efecúe un {POP de IP}, un {POP de CS} y un {POP de Flags} a fin de regresar a la ejecución del programa principal que fue suspendido para dar atención a la interrupción.

En resumen, el 8259A "canaliza" jerárquicamente (a manera de embudo) las señales de interrupción procedentes de hasta 8 fuentes diferentes, en la única entrada para interrupción mascarable del μP 8088/8086 {INTR} y envia al μP un "tipo (vector) de interrupción" especificado para cada una de las 8 entradas de interrupción (IRx) ... x = 0,1,2,...,7 .

Actualmente en el sistema **SDM88-PC** se pueden habilitar a través de su 8259A las siguientes interrupciones (tipos) :

```
Int_type 32   (entrada IR0 del PIC) ... más alta prioridad
Int_type 33            IR1
   .                    .
   .                    .
   .                    .
Int_type 39            IR7          ... más baja prioridad
                                        de atención
```

Los vectores de interrupción (direcciones absolutas del inicio de las rutinas de servicio a interrupción correspondientes a cada tipo de interrupción que se pueda presentar en el sistema) deben cargarse en la zona baja de memoria del sistema (RAM) para poder responder adecuadamente cuando ocurran las interrupciones.

Cada Vector de interrupción en RAM abarca 4 bytes para contener la dirección de la rutina de servicio (ISR = Interrupt Service Routine), en el siguiente formato :

byte 1 : IPLOW
byte 2 : IPHIGH
byte 3 : CSLOW
byte 4 : CSHIGH

IP = Instruction Pointer ; CS = Code Segment) de la ISR

Para los tipos de interrupción implementados en el SDM88-PC para el 8259A los vectores de interrupción se localizan en las siguientes localidades de RAM :

```
Int_type 32 :  (IR0)  :    CS=0000 ; IP=0080
                           ( 32=20h  ;  20h * 4 = 80h )
                             Dir.Fis.: 00080h a 00083
Int_type 33 :  (IR1)  :    CS=0000 ; IP=0084
                             Dir.Fis.: 00084h a 00087
Int_type 34 :  (IR2)  :    CS=0000 ; IP=0088
                             Dir.Fis.: 00088h a 0008B
Int_type 35 :  (IR3)  :    CS=0000 ; IP=008C
                             Dir.Fis.: 0008Ch a 0008F
Int_type 36 :  (IR4)  :    CS=0000 ; IP=0090
                             Dir.Fis.: 00090h a 00093
Int_type 37 :  (IR5)  :    CS=0000 ; IP=0094
                             Dir.Fis.: 00094h a 00097
Int_type 38 :  (IR6)  :    CS=0000 ; IP=0098
                             Dir.Fis.: 00098h a 0009B
Int_type 39 :  (IR7)  :    CS=0000 ; IP=009C
                             Dir.Fis.: 0009Ch a 0009F
```

Actualmente está contemplada la habilitación selectiva (si el usuario así lo requiere), de mecanismos de interrupción para el manejo de la Transmisión/Recepción de Datos Serie por el 8251A y para el manejo de la adquisición de datos por el convertidor Analógico/Digital.

Tipo de Interrupción	Condición para que se presente
32 = 20h	Receive Byte (8251A ⟵ μC-PC)
33 = 21h	Transmit Byte (8251A ⟶ μC-PC)
34 = 22h	A/D c _ Read digitized data (leer info. digitalizada cuando la salida EOC del ADC pase de L a H : el ADC ha terminado de realizar una conversión)

La Interrupción tipo 32 se presenta cuando la salida "RxRDY" (pin 14) del USART 8251A tiene una transición L-H indicando que el 8251A ha recibido un caracter en su entrada serial y está listo para transferirlo al CPU. Aunque el receptor funcione continuamente, RxRDY sólo será verificada si el bit RxE (Receive Enable) en el registro de comando ha sido habilitado ("1" lógico) previamente. La salida RxRDY es conectada a la estructura de interrupciones del sistema (entrada IR0 del 8259A) cuando se desea ejecutar la rutina de "recepción de caracter" sólo cuando llegue un caracter al 8251A procedente de algún dispositivo de comunicación externo (μC-PC); de otro modo el CPU debe checar la condición del bit RxRDY del registro interno dc status usando una operación de Lectura de Status (Método Poleo). RxRDY será reseteado ("0") cuando el caracter recibido sea leído por el CPU.

La Interrupción tipo 33 se presenta cuando la salida "TxRDY" (pin 15) del USART 8251A tiene una transición L-H indicando al CPU que el USART 8251A está listo para aceptar un caracter dato o comando por su entrada paralela conectada al μP. Esta salida del 8251A puede ser usada como fuente de interrupción al sistema (a través de IR1 del 8259A) si no se desea mantener dedicado al μP mediante un esquema de Poleo (donde el CPU deba checar la condición del bit TxRDY del reg. de status mediante una operación de lectura de Status). Cabe señalar, sin embargo, que mientras el

bit TxRDY del registro de status es verificado siempre que el buffer de Transmisión (Data/Command) está vacio, la salida física TxRDY (pin 15) sólo se verificará si el buffer está vacío y el USART está habilitado para transmitir ($\overline{CTS}$ = Low y TxEN (Transmit Enable) = High). TxRDY será reseteado cuando el USART reciba un caracter desde el programa.

La Interrupción tipo 34 se presenta en el momento en que se termina la conversión del valor presente en alguno de los canales del convertidor Analógico/Digital. La transición L-H de la señal de salida EOC del ADC, anuncia al controlador de interrupciones que el convertidor A/D tiene un valor que enviar al microprocesador. En el SDM88-PC se puede conectar la salida EOC del ADC a la entrada IR2 del 8259A si se desea operar con una rutina de lectura (adquisición) de datos bajo un esquema de interrupciones (no de poleo).

2.5.4

8251A : PUERTO SERIE (Microcomputadora-Interface) : UNIVERSAL SYNCHRONOUS / ASYNCHRONOUS RECEIVER & TRANSMITTER (USART)

En un ambiente de comunicación un dispositivo interface debe convertir datos del sistema en formato paralelo, a un formato serie para su transmisión y convertir los datos entrantes al sistema en formato serie, a datos en paralelo, durante el proceso de recepción. El dispositivo interface debe también borrar o insertar bits o caracteres que sean funcionalmente exclusivos de la técnica de comunicación empleada. En esencia, la interface de comunicación debe aparecer "transparente" al CPU, como un simple medio de entrada y salida de datos en un sistema de comunicación orientado a byte (Byte-oriented system).

En el sistema SDM88-PC se pensó en utilizar un USART 8251A para establecer un canal de comunicación serie entre la Microcomputadora-Interface (Hardware externo basado en el μP 8088) y la Microcomputadora-PC (PS). El USART 8251A es un circuito diseñado para comunicación de datos con las familias de microprocesadores de Intel (MCS-48, 80, 85 y iAPX-86,88) : μP's : 8048, 8080, 8085, 8088, 8086. El chip está fabricado usando la tecnología de compuertas de silicio canal-N. Como otros dispositivos I/O en un sistema de microcomputadora, su configuración funcional es programada por el software del sistema para dotarlo de una mayor flexibilidad.

El 8251A se usa como un dispositivo periférico y es programado por el CPU para operar empleando virtualmente cualquier técnica de transmisión serie de datos actualmente en uso, incluyendo el método síncrono IBM-"bi-sync". Para el sistema SDM88-PC, la comunicación serie establecida entre el 8250 del puerto serie de la μC-PC y el 8251A de la μC-Interface, es Asíncrona, (8 bits de datos-información y sin paridad).

El USART acepta caracteres de datos del CPU en formato paralelo y entonces los convierte para su transmisión, en una corriente continua de datos serie. Simultáneamente, el 8251A puede recibir hileras o corrientes de datos en formato serie y convertirlas en caracteres de datos en paralelo para ser leídos por el CPU. El USART dará señalización al CPU para indicarle cuándo puede aceptar un nuevo caracter para transmitir o cuándo ha recibido un caracter para el CPU. El CPU puede también leer en cualquier momento el status completo del USART, que está contenido en un registro interno. El USART cuenta con la capacidad de detección de errores en los procesos de transmisión (Parity, Overrun & Framing errors) y generación de señales de control como SYNDET, TxEMPTY, TxRDY, RxRDY y de control de Modem : $\overline{DSR}$, $\overline{DTR}$, $\overline{CTS}$, $\overline{RTS}$.

FORMATOS DE COMUNICACION

La comunicación serie, ya sea en un enlace de datos o en un dispositivo periférico local, ocurre en una de dos formas básicas: Asíncrona o Síncrona. Estas dos formas son similares en el hecho de requerir de información de formato, marco o estructuración (framing information) adicionada a los datos para permitir la adecuada detección de caracteres en la terminal receptora. La mayor diferencia entre estas dos formas de comunicación es que el tipo Asíncrono requiere de la adición de información de marco (formato) a cada caracter enviado, mientras que el tipo Síncrono agrega información de marco a bloques de datos o mensajes. Dado que la forma Síncrona es más eficiente que la Asíncrona pero requiere de una decodificación más compleja, es encontrada típicamente en enlaces de datos de alta velocidad, mientras que la forma Asíncrona es usada en líneas de velocidad baja o moderada : 110, 300, 1200, 2400, 4800, 9600, 19200 Bauds.

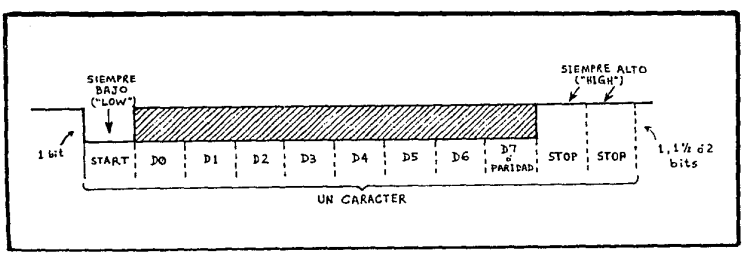
. . . . . .

En la { *figura* 2-2 } se muestra el Formato binario usado en la transmisión serie asíncrona de datos.

. . . . . .

{ *figura* 2-2 }

Formato binario en la transmisión serie asíncrona de datos.



Cuando no se están transmitiendo datos, la línea de señal permanece en un estado "1-lógico = High" constante o "estado de Marca" (Marking state). El inicio de un caracter de datos se manifiesta en la línea con un estado "0-lógico = Low" durante el periodo de un bit. Este bit se denomina "Bit de Inicio" (START bit). A continuación son enviados a la línea de transmisión los bits de datos (información) uno después de otro. El bit menos significativo (LSB) es enviado primero. Dependiendo del sistema, la palabra de datos puede constar de 5, 6, 7 u 8 bits (longitud del caracter de datos). En seguida de los datos puede mandarse o no un "bit de Paridad" (Parity bit), usado para detectar errores en los datos recibidos. Algunos sistemas no insertan o buscan un bit de paridad (tal es el caso del **SDM88-PC** que manda 1 bit START, caracter de datos de 8 bits, y 1 bit STOP). Después de los bits de datos y paridad la línea regresa a un estado "1-lógico = High" durante al menos el periodo de 1 bit para identificar el final del caracter enviado. Este bit siempre "High" se denomina "STOP bit".

En modo Asíncrono el 8251A tiene la capacidad de trabajar con caracteres de datos de 5, 6, 7 y 8 bits, generación de 1, $1^1/_2$ y 2 bits STOP, detección de START bit falso, generación de caracter Break, Detección y manejo automáticos de Break, Razón de Baudaje Asíncrona desde DC hasta 19200 Bauds y Síncrona desde DC hasta 64 kBauds.

Resumiendo, en el formato Asíncrono se transmiten los bits básicos de información agregando un bit de START al inicio de todos ellos y uno o más bits STOP al final según son transmitidos por la línea de comunicación. El bit START es un 0-lógico o SPACE, y es definido como el nivel de voltaje positivo por la norma RS-232C. El bit STOP es un 1-lógico o MARK, y se define como el nivel de voltaje negativo por el RS-232C. En la norma RS-232C , el nivel lógico HIGH o MARK es un voltaje entre -3 V y -15 V bajo carga (-25 V sin carga) y el nivel lógico LOW o SPACE

es un voltaje entre +3 V y +15 V bajo carga (+25 V sin carga). Comúnmente se emplean voltajes como ± 12 V . En las aplicaciones de Lazo de corriente (Current Loop), el estado MARK se manifiesta por flujo de corriente y el estado SPACE por ausencia de corriente eléctrica (20 ó 60 mA).

El bit START indica al receptor que debe empezar a ensamblar un caracter y le permite además (al receptor) sincronizarse con el transmisor. Puesto que esta sincronización sólo tiene que durar por el periodo de un caracter (ya que el siguiente caracter enviado contendrá un nuevo bit START), este método funciona bastante bien asumiendo la existencia de un receptor diseñado adecuadamente. Al final de un caracter se agregan uno o más bits STOP (basta 1 bit STOP para velocidades ≥ 300 Bauds), para garantizar que el bit START del siguiente caracter producirá una transición (H-L) en la linea de comunicación y para dar tiempo al receptor para "amarrarse" o sincronizarse con el transmisor si su reloj básico está corriendo ligeramente más lento que el del transmisor. Si, por el contrario, el reloj del receptor está corriendo ligeramente más rápido que el del transmisor, el receptor percibirá huecos entre caracteres pero seguirá todavia decodificando correctamente los datos. Debido a esta tolerancia a desviaciones de frecuencia pequeñas, no es necesario que los relojes del transmisor y del receptor estén "amarrados" a frecuencias de operación idénticas para lograr una comunicación Asincrona exitosa. Con respecto a este punto cabe señalar que el USART 8251A permite que la frecuencia de la señal de reloj aplicada a sus terminales ($\overline{TxC}$ = Transmitter Clock y $\overline{RxC}$ = Receiver Clock), sea 1 (modo Isosincrono), ó 16 ó 64 (modo Asincrono) veces el Baudaje de Transmisión y Recepción, dependiendo de la palabra de Modo con que el USART sea inicializado (Baud Rate Factor : 1X,16X,64X). Es necesario señalar que la operación con el factor 1X sólo es válida si los relojes de transmisor y receptor están sincronizados (Modo

Isosincrono). Al usar una frecuencia de reloj mayor que la razón de Baudaje, se permite al registro-de-corrimiento-Receptor sincronizarse (ser habilitado para realizar sus lecturas) en el centro de un "periodo de bit" en lugar de en una transición de niveles. Esto reduce la probabilidad de errores de lectura causados por ruido en las transiciones.

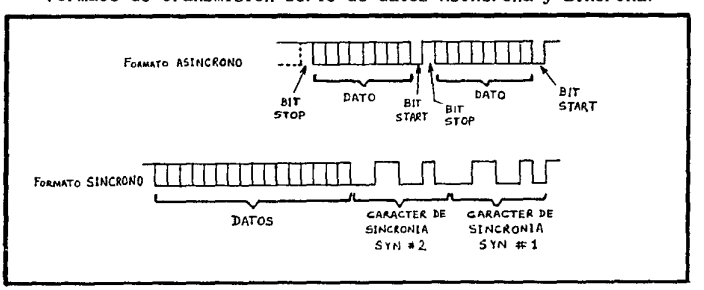
. . . . . .

En la { *figura* 2-3 } se muestra un ejemplo de los formatos de transmisión serie de datos Asincrona y Sincrona.

. . . . . .

{ *figura* 2-3 }

Formato de transmisión serie de datos Asincrona y Sincrona.



El formato Sincrono es tipico en su requerimiento de dos caracteres de sincronia (SYN) al inicio del mensaje. El formato Asincrono, también típico, requiere de un bit START precediendo cada caracter y un bit STOP sucediéndolo. En ambos casos se muestra la transmisión de dos caracteres de 8 bits.

En el modo Asincrono se requieren { 10*N } bits para transmitir N caracteres y en el modo Sincrono se usan { 8*N + 16 } bits. Para el ejemplo mostrado el modo Asincrono resulta más

eficiente, pues emplea 20 bits contra 32 del Sincrono. Sin embargo, al transmitir 1000 caracteres, el modo Asincrono requiere de 10000 bits contra 8016 para el modo Síncrono. Por tanto, para mensajes largos el modo Síncrono llega a ser más eficiente que el Asíncrono. La eficiencia del modo Asincrono es en general baja porque requiere de 10 (1 STOP bit) a 11 (2 STOP bits) periodos de bit para transmitir (un caracter de 7 bits con bit de paridad), como un caracter ASCII o (un caracter de 8 bits sin paridad), como un byte de información tipico en sistemas digitales de instrumentación y control (ej. SDM88-PC). El punto de cruce para el ejemplo mostrado en la figura 2-3 es 8 caracteres, pues ambos formatos requieren 80 bits para transmitirlos.

El USART 8251A ha sido diseñado para ajustarse a un amplio espectro de requerimientos en los modos Síncrono, Asíncrono e Isosíncrono (relojes del Transmisor y Receptor perfectamente sincronizados).

En el modo Síncrono el 8251A opera con caracteres de 5, 6, 7 u 8 bits. Se puede agregar y checar opcionalmente Paridad Par o Impar. La sincronización requerida puede alcanzarse ya sea externamente via hardware adicional o internamente via detección de caracteres SYN. La detección SYN puede basarse en 1 ó 2 caracteres que pueden o no ser iguales. El (los) caracter(es) SYN es(son) insertado(s) automáticamente en la corriente de datos si el software de comunicación falla en el suministro de datos a tiempo. La generación automática de caracteres SYN se requiere para prevenir la pérdida de sincronia.

En el modo Asincrono el 8251A opera con las mismas estructuras de datos y paridad que en el modo Síncrono. Además de agregar un bit START a estos datos, el USART puede añadir 1, $1^1/_2$ ó 2 bits STOP. El sistema Receptor checa que el formato o marco sea correcto habilitando una bandera de status si se detecta un error:

- de Paridad (*Parity Error*) ... PE flag
  (se produce cuando falla el chequeo de Paridad Par o Impar de caracter, según se defina)
- de SobreLectura (*Overrun Error*) ... OE flag
  (se produce cuando el CPU no lee un caracter antes de que el siguiente esté disponible; el caracter anterior es sobreescrito y se pierde)

y/o

- de Formato (*Framing Error*) ... FE flag
  (se produce cuando no se detecta un STOP bit válido al final de todos los caracteres)

El USART 8251A puede transmitir en los tres modos (Sincrono, Asíncrono e Isosincrono) en modo *Half* o *Full Duplex* y cuenta con un "Doble-buffer" interno (Double-buffered), lo cual significa que un caracter puede ser cargado en un buffer de almacenamiento intermedio mientras otro caracter está siendo serializado hacia el canal de comunicación por el registro-de-corrimiento-transmisor real; asi el software de comunicación tiene un periodo de caracter completo para responder a una solicitud de servicio (atención).

Aunque el 8251A da soporte al conjunto básico de señales de control : { $\overline{\text{CTS}}$ (*Clear To Send*), $\overline{\text{RTS}}$ (*Request To Send*), $\overline{\text{DSR}}$ (*Data Set Ready*), $\overline{\text{DTR}}$ (*Data Terminal Ready*) }, no soporta totalmente la señalización descrita en el estándar EIA-RS-232C (p.ej. las señales *Carrier Detect* (CF), *Ring Indicator* (CE) y las señales del canal secundario). En los casos en que sea necesario implementar estas señales (Control de Modem), se requerirá de un puerto adicional (y un software de control adecuado). El RS-232C define en total 25 señales de "pin" y especifica que el conector del "Equipo Terminal de Datos" DTE (*Data Terminal Equipment*) sea Macho (*Male*) y el conector del "Equipo Comunicador de Datos" DCE (*Data Communication Equipment*) sea Hembra (*Female*). Los conectores más comúnmente usados son el DB-25P (macho) y el DB-25S (hembra).

Para el sistema **SDM88-PC** se requirió de la implementación de un canal de comunicación serie de datos entre el puerto serie de la μC-PC(PS) (IBM PC Asynchronous Communication Adapter) con su UART INS8250, y la μC-Interface (hardware externo basado en el μP 8088) con su USART 8251A, para la transferencia de código hex de μP para el control de la μC-Interface desde la μC-PC (comandos) y para el bajado (*download*) de programas de aplicación (del usuario) en código de μP originalmente almacenados en disco en la μC-PC pero destinados para operar sobre el hardware de la μC-Interface.

La tarjeta de control de Comunicación Asincrona en la μC-IBM PC (PS) o compatible, está configurada como DTE, por lo que cuenta con un conector DB-25P (macho) para las conexiones RS-232C. Como se trata de un DTE, entonces la localización de las señales en el conector es como sigue :

GND (signal)	pin (7)
TxD	pin (2)
RxD	pin (3)
$\overline{\text{RTS}}$	pin (4)
$\overline{\text{CTS}}$	pin (5)
$\overline{\text{DTR}}$	pin (20)
$\overline{\text{DSR}}$	pin (6)
$\overline{\text{CD}}$ (Carrier Detect)	pin (8)

A fin de que la μC-IBM PC (PS) pueda transmitir o recibir, sus entradas $\overline{\text{CTS}}$, $\overline{\text{DSR}}$ y $\overline{\text{CD}}$ deben ser verificadas (Low). El software del BIOS (*Basic Input/Output System*) verifica las salidas $\overline{\text{RTS}}$ y $\overline{\text{DTR}}$. Otra manera de controlar el puerto serie de la μC-PC(PS) es a través de un lenguaje de alto nivel (después puede compilarse) como el BASIC de *Microsoft* (que ofrece la facilidad de habilitar y deshabilitar por software las señales de control de comunicación y se ocupa del control directo del hardware de comunicación de la PC y de los procesos de "handshaking", permitiendo además que el usuario configure un protocolo de comunicación serie de acuerdo a

sus necesidades; con BASIC el puerto serie de la PC es controlado como si fuese un Archivo de Disco: {OPEN,CLOSE,PRINT#,INPUT#}).

Además del software de control de comunicación residente en memoria de μC-PC(PS) debe estarse ejecutando paralelamente un software de control en la memoria de la μC-Interface a fin de establecer un protocolo de comunicación bidireccional (Handshaking Tx & Rx) entre los dos subsistemas del **SDM88-PC** . El programa de control en la interface está residente en su EPROM (2764).

El 8251A no proporciona los niveles de voltaje definidos por el EIA-RS-232C , por tanto se requiere de la adición de buffers (drivers y receptores) para realizar esta interface entre niveles TTL y RS-232. Para ello se emplean los circuitos integrados:

* MC1488 : *Quad TTL_to_RS-232 drivers*
  Requiere fuentes V^+ y V^-
  Se coloca un capacitor a GND (de 330pF a 680pF) en las salidas de estos drivers a fin de reducir el efecto de cruce (*cross talk*) entre alambres adyacentes y ajustar los tiempos de levantamiento y caida de las señales RS-232C dentro de un rango cuyo límite máximo sea 30 V/μs (slew rate máximo fijado por la norma).

y * MC1489 : *Quad RS-232_to_TTL receivers*
  Requiere fuente única de +5V

En el sistema **SDM88-PC**, la Microcomputadora-Interface entrega a la Microcomputadora PC (PS) niveles de voltaje entre +12 y -12 [V] (por su línea Tx) a través del circuito integrado Convertidor de nivel TTL $\longrightarrow$ RS-232 (1488), mientras que la μC PC (PS) entrega a la Interface niveles entre +9 y -9 [V] (por su línea Tx) al circuito Convertidor de nivel RS-232 $\longrightarrow$ TTL (1489) en la línea Rx.

Las tierras (GND) de señal de la μC-Interface y de la μC-PC(PS) están conectadas entre si, a fin de servir como referencia física para la identificación de niveles de voltaje como estados lógicos en los procesos de Transmisión/Recepción de información por Puerto Serie.

TRANSMISION Y RECEPCION DE CARACTERES CON UN USART 8251A

Los caracteres de datos pueden ser transmitidos y recibidos con un 8251A bajo un esquema de INTERRUPCION o un esquema de POLEO.

A) Transmisión de caracteres bajo un esquema de Interrupción :

El pin TxRDY del 8251A se conecta a una entrada de interrupción del μP o de un PIC 8259A (Priority Interrupt Controller). El transmisor y la salida TxRDY son habilitados al poner un "1" en el bit D0 de la palabra de control enviada al 8251A durante la rutina de inicialización : (Command Instruction Word : D0 = TxEN (Transmit Enable)). Cuando la entrada $\overline{CTS}$ del 8251A es verificada "Low", y el buffer del USART está listo para un caracter, el pin TxRDY irá al estado "High". Si la trayectoria de interrupción entre el μP y el 8259A está habilitada, el μP ejecutará una rutina de servicio a interrupción que escribirá un caracter en la dirección del registro de Datos del 8251A. Al escribir el caracter_dato el USART reseteará su salida TxRDY hasta que el buffer esté de nuevo listo para recibir un caracter.

B) Recepción de caracteres bajo un esquema de Interrupción :

El pin salida RxRDY del 8251A se conecta a una entrada de interrupción del μP o de un PIC 8259A (Priority Interrupt Controller). El receptor y la salida RxRDY son habilitados al poner un "1" en el bit D2 de la palabra de control enviada al 8251A durante la rutina de inicialización : (Command Instruction

Word : D2 = RxEN {Receive Enable}). Cuando un caracter ha entrado desde el canal de comunicación hacia el 8251A y dicho caracter está en el buffer del receptor listo para ser leido, el pin RxRDY irá al estado "High". Si la cadena de interrupción a través del 8259A y del μP está habilitada, el μP ejecutará una rutina de servicio a interrupción que leerá el caracter_dato. Cuando el μP lee un caracter_dato del 8251A, éste resetea su salida RxRDY desapareciendo así la condición de interrupción. Esta señal permanecerá baja (Low) hasta que otro caracter esté listo para ser leido por el μP.

C) Transmisión de caracteres bajo un esquema de Poleo :

El Registro de Status del 8251A es leido y checado por el μP una vez tras otra hasta que el bit TxRDY (D0) sea sensado como "High" (1-lógico) indicando así que el USART está listo para aceptar un caracter_dato o comando enviado desde el CPU. En algunos sistemas también será necesario checar el bit D7 del registro de status para garantizar que la entrada $\overline{DSR}$ del 8251A ha sido verificada por una señal proviniente de un Modem p.ej.. Cuando el (los) bit(s) requerido(s) del registro de status es(son todos) "High", un caracter_dato es escrito en el 8251A por el μP, para su consecuente serialización y salida hacia el canal de comunicación. El Registro de Status tiene la misma dirección interna que el Registro de Control. Al escribirse un caracter_dato en el registro de Datos del 8251A, se resetea automáticamente el bit TxRDY del registro de Status y permanece "Low" hasta que el buffer transmisor intermedio (Transmitter holding buffer) está de nuevo listo para recibir otro caracter enviado por el CPU; esto es, el bit TxRDY irá al estado "High" de nuevo cuando el "Tx holding buffer" (no necesariamente el "Tx Shift Reg."), esté vacio y otro caracter pueda ser enviado desde el μP. Por otra parte, el pin de salida TxEMPTY del 8251A irá al estado "High" cuando tanto el "Tx holding buffer" como el "Tx Shift Register" estén vacios.

D) Recepción de caracteres bajo un esquema de Poleo :

Es un proceso similar al anterior, excepto que ahora el bit RxRDY (D1) del Registro de Status es poleado para determinar cuándo el USART ha recibido un caracter en su entrada serial y está listo para transferirlo al CPU (indicar cuándo un caracter está listo para ser leído por el μP). Asi, cuando el bit D1 es sensado "High", un caracter es leído por el CPU desde la dirección del Registro de Datos del 8251A. El bit RxRDY en el registro de Status es reseteado automáticamente cuando un caracter es leido del USART, y permanece en estado "Low" hasta que el 8251A tiene un nuevo caracter en su buffer receptor, listo para ser leído por el μP.

2.5.5

82C55A : PUERTOS PARALELOS (Microcomputadora-Interface) : PROGRAMMABLE PERIPHERAL INTERFACE (PPI)

El 82C55A es un dispositivo interface I/O programable de propósito general diseñado para usarse dentro de sistemas de microcomputadora basados en μP's de Intel. El PPI CHMOS es una versión de alto rendimiento del estándar industrial 8255A (NMOS) con el que es pin compatible. Se dice que es de alto rendimiento porque opera con "Cero estados de Espera" a frecuencia de 8 MHz con un 8086/88 o un 80186/88 (Speed: Tww = 20 [ns]). La tecnologia avanzada CHMOS III de Intel hace que este dispositivo consuma muy poca potencia de la fuente de alimentación y sus salidas son TTL compatibles. Cualquier terminal I/O en el puerto A, B o C puede sumir (sink) o proporcionar (source) 2.5 mA DC. Esta caracteristica permite al 82C55A habilitar directamente drivers tipo Darlington y Despliegues numéricos de alto voltaje que requieren sumir o suministrar dicha corriente. El PPI cuenta con circuiteria "Bus-Hold" en todos los puertos I/O permitiendo eliminar asi el uso de resistores de "Pull-up".

La función del PPI 82C55A es la de un dispositivo I/O de propósito general para conectar (servir de interface a) equipo periférico al bus del sistema de una microcomputadora. La configuración funcional del PPI es programada por el software del sistema a fin de que normalmente no se requiera lógica externa para servir como interface de estructuras o dispositivos periféricos.

El PPI tiene 24 líneas I/O divididas en 3 puertos de 8 bits (A, B y C). Estos puertos pueden ser configurados en una amplia variedad de caracteristicas funcionales por el software del sistema pero cada uno tiene sus propios rasgos distintivos o "personalidad" para incrementar el poder y flexibilidad del 82C55A :

- Puerto A :
  Puede usarse como un puerto de salida de datos de 8 bits (latch/buffer) o como un puerto de entrada de 8 bits (latch/buffer).
  Las terminales de este puerto cuentan con dispositivos mantenedores de nivel (Bus Hold) tanto "Pull-Up" como "Pull-Down".
- Puerto B :
  Puede usarse como un puerto de salida de datos de 8 bits (latch/buffer) o como un puerto de entrada de 8 bits (latch/buffer).
  Las terminales de este puerto cuentan con dispositivos mantenedores de nivel (Bus Hold) sólo de tipo "Pull-Up" .
- Puerto C :
  Puede usarse como un puerto de salida de datos de 8 bits (latch/buffer), o como un puerto de entrada de 8 bits (buffer/no latch for input); puede también dividirse en dos puertos de 4 bits bajo control de modo (cada puerto de 4 bits contiene un latch de 4 bits); finalmente puede producir señales de "handshake" para los puertos A y B

(control signal outputs & status signal inputs).
Las terminales de este puerto cuentan con dispositivos mantenedores de nivel (Bus Hold) sólo de tipo "Pull-Up" .

La entrada RESET del 82C55A es conectada a la línea de RESET del sistema, a fin de que cuando el sistema sea reseteado, todas las líneas de los 3 puertos sean configuradas como entradas. Esto se hace con el propósito de prevenir la destrucción de circuiteria conectada a las líneas de puerto (en condiciones iniciales de encendido del sistema o de Reset), que podría suscitarse si las líneas de algún puerto fueran inicializadas como salidas al entrar en conflicto los argumentos o estados lógicos iniciales indeterminados de dos o más de dichas líneas de salida (activando o desactivando indebidamente los dispositivos conectados a ellas). En resumen, cuando la terminal (entrada) RESET del 82C55A es puesta en estado "High", todos los puertos son configurados como entradas con las 24 líneas de puerto mantenidas en nivel "1-lógico" por los dispositivos internos "Bus hold". Una vez que la condición de RESET es removida, el 82C55A permanece en modo INPUT sin requerir inicialización adicional.

El PPI tiene 3 modos básicos de operación que pueden seleccionarse por el software del sistema :

MODO "0" : Basic Input/Output

El PPI se inicializa en este modo cuando se desea usar un puerto para Entrada o Salida simple sin "Handshaking". Si tanto el puerto A como el B son inicializados en Modo 0 entonces las dos mitades del puerto C pueden usarse juntas como un puerto adicional de 8 bits o como dos puertos individuales de 4 bits. Las dos mitades del puerto C son independientes, por tanto, una mitad puede inicializarse como entrada y la otra como salida. Cuando se usan como salidas, las líneas del puerto C pueden ser seteadas ("high") o reseteadas ("low") individualmente al mandar una

palabra de control especial a la dirección del registro de control del PPI (*Single Bit Set/Reset*) usando una simple instrucción OUT ; esta caracteristica reduce los requerimientos de software en aplicaciones de Control e Instrumentación.

** La configuración funcional MODO 0 provee operaciones simples de entrada/salida para cada uno de los tres puertos. No se requiere "Handshaking", y los datos son simplemente escritos a (OUT {Write}) o leidos de (IN {Read}) el puerto especificado.

Modo 0 ... Definiciones funcionales básicas :

- + Dos puertos de 8 bits y 2 puertos de 4 bits
- + Cualquier puerto puede funcionar como entrada o como salida
- + Las salidas están "latcheadas"
- + Las entradas no están "latcheadas"
- + En este modo son posibles 16 configuraciones I/O diferentes

MODO "1" : Strobed Input/Output

El PPI se inicializa en este modo cuando se desea usar el puerto A y/o el puerto B para operaciones de Entrada o Salida con "Handshake" (strobed I/O). En este modo algunas de las terminales del puerto C funcionan como líneas de "Handshake".

> Si el puerto B es inicializado en MODO 1 (Entrada o Salida), los pines PC0, PC1 y PC2 funcionan como líneas de Handshake para B.

> Si el puerto A es inicializado como Entrada en MODO 1, los pines PC3, PC4 y PC5 funcionan como señales de Handshake. Los pines PC6 y PC7 están disponibles para usarse como líneas de entrada o salida simple.

> Si el puerto A es inicializado como Salida en MODO 1, los pines PC3, PC6 y PC7 funcionan como señales de Handshake. Los pines PC4 y PC5 están disponibles para usarse como líneas de entrada o salida simple.

Cuando el puerto C está siendo usado como Status/Control para los

puertos A o B, sus bits pueden ser "seteados" o "reseteados" usando operaciones "Bit Set/Reset" tal como si fuesen puertos de salida de datos.

** La configuración funcional MODO 1 provee de medios para transferir datos I/O hacia o desde un puerto especificado en conjunción con señales "strobe" o de "handshaking". En MODO 1, el Puerto A y el Puerto B usan las líneas del Puerto C para generar o aceptar estas señales de "handshaking".

Modo 1 ... Definiciones funcionales básicas :

- + Dos Grupos (Grupo A y Grupo B)
- + Cada Grupo contiene un puerto de datos de 8 bits y un puerto de control/datos de 4 bits
- + El puerto de datos de 8 bits puede ser programado como Entrada o como Salida; tanto las entradas como las salidas están "latcheadas"
- + El puerto de 4 bits es usado para control y status del puerto de datos de 8 bits

MODO "2" : Strobed Bidirectional Bus I/O

Sólo el Puerto A puede ser inicializado en MODO 2 , para ser usado en transferencia bidireccional de datos supervisada por "Handshake". Asi, los datos pueden ser escritos (OUTput) o leídos (INput) en las mismas 8 líneas. El 82C55A puede usarse en este modo para extender el bus del sistema a un microprocesador esclavo o para transferir bytes de datos hacia o desde una tarjeta controladora de disco flexible (floppy), p.ej..

> Si el puerto A es inicializado en MODO 2, entonces los pines PC3 a PC7 son usados como líneas de Handshake para el puerto A. Los pines PC0, PC1 y PC2 pueden usarse para Entrada o Salida si el puerto B está en MODO 0. Estos tres pines se usarán como líneas de Handshake para el puerto B, si éste es inicializado en MODO 1 .

** La configuración funcional MODO 2 provee de medios para comunicarse con una estructura o dispositivo periférico a

través de un único bus de 8 bits, para tanto transmitir como recibir datos (Bus I/O Bidireccional). Se cuenta con señales de Handshaking para mantener una disciplina adecuada en el flujo de datos por el Bus, de manera similar que en el MODO 1. Se dispone también de generación de Interrupciones y de funciones de habilitación/deshabilitación.

Modo 2 ... Definiciones funcionales básicas :

+ Usado solamente en el Grupo A
+ Un puerto de datos (bus) bidireccional de 8 bits (Puerto A) y un puerto de control de 5 bits (Puerto C)
+ Tanto las entradas como las salidas están "latcheadas"
+ El puerto de control de 5 bits (Puerto C) es usado para funciones de control y status para el puerto bus bidireccional de 8 bits (Puerto A)

En la Microcomputadora-Interface (Hardware externo a la Microcomputadora PC (PS)) dentro del Sistema de Desarrollo **SDM88-PC** , se cuenta con 6 Puertos Paralelos programables (Entrada/Salida, Handshaking I/O o PA bidireccional), ya que se tienen conectados al bus de datos del sistema, dos PPI 82C55A [-2] { (CHMOS) , (8MHz): Zero Wait State , Speed: Tww = 20 [ns] }, con 3 puertos paralelos programables de 8 bits cada uno.

El PPI - #1 está dedicado para uso del sistema y se emplea en el manejo del Convertidor Digital/Analógico (DAC) y del Convertidor Analógico/Digital (ADC) :

+ Para manejo del DAC :

  (PA = puerto de salida : 8 bits : Datos)

  El byte (info.digital) cuyo valor se desea convertir a un nivel de voltaje analógico se escribe (a través de una instrucción OUT) en este puerto de salida. Dado que el DAC tiene una resolución de 8 bits se tendrán 256 valores-nivel analógicos posibles dentro de un rango de 0 a +10 Volts.

+ Para manejo del ADC :

(PB = puerto de salida : 5 bits : Control)

(PC = puerto de entrada : 8 bits : Datos)

En el SDM88-PC el Convertidor A/D es controlado por Software. La rutina de control/lectura del ADC está residente en la EPROM de la µC-Interface en versiones NEAR CALL y FAR CALL para que el usuario la invoque cuando lo requiera desde su programa de aplicación. El número del canal del ADC (entrada analógica {IN0-IN7}) que se desea digitalizar, es pasado a esta rutina de conversión en el registro "BL" del µP 8088; la rutina entonces manda este # al ADC (MUX select inputs) y genera, por software, las formas de onda de control para el ADC (según el diagrama de tiempos y parámetros indicados en el Manual "*Linear Databook* (2)" de National S.C. para el Convertidor A/D "ADC0809"). El µP polea el estado de la salida EOC (End-Of-Conversion) del ADC para saber cuándo ha terminado el proceso de conversión A/D y el valor binario correspondiente está disponible en las salidas D0-D7 del ADC . El valor binario correspondiente a la señal analógica digitalizada, es regresado en el registro "AL" del µP al ser leido por éste a través del puerto paralelo C del 82C55A. La salida OE (Output Enable) del ADC es mantenida en "1" lógico ya que no es necesario poner en alta impedancia la salida digital del ADC, pues no está conectada al bus de datos del sistema sino a las terminales de un puerto de entrada (Puerto C) dedicado exclusivamente para recibir esa información. La salida EOC del ADC es poleada por el µP en el bit D0 del bus de datos mediante un puerto (IN) de 1 bit implementado con un buffer Tri-State (74LS126) habilitado con señal $\overline{RD}$ y una señal selectora de puerto ($\overline{YS}$) procedente del decodificador de puertos I/O ("ADC_EOCport").

.....

La rutina de conversión A/D empleada tiene las siguientes caracteristicas :

Entrada : Dirección del Canal Analógico que se va a digitalizar ... registro "BL"

Salida : Información digitalizada (Byte_Dato) ... registro "AL"

Registros µP usados : AL , BL , DX , Flags

Registros µP alterados : AL , BL

Puertos I/O usados :

PPort1B (PPI 82C55A) : OUTPUT

PB5 = START Conversion

PB4 = ALE (Address Latch Enable)

PB2 = ADD_C \

PB1 = ADD_B - Channel Address (ADC-MUX)

PB0 = ADD_A /

PPort1C (PPI 82C55A) : INPUT

{ PC7 a PC0 } = ← Información digitalizada por ADC { D7 a D0 }

ADC_EOCport (Tri-State Gate 74LS126) : INPUT

Proporciona en D0 (Data Bus) el estado de la salida EOC del ADC (operación poleo)

.....

En la rutina de conversión A/D, la secuencia en que las señales de control deben enviarse al ADC (basándose en el diagrama de tiempos del ADC0809) es la siguiente :

1) Dirección (3 bits) del canal analógico de entrada deseado (ADC-MUX select inputs)
2) Después de al menos t_s = 50[ns], habilitar "high" la entrada ALE (la dirección es latcheada en el decodificador)
3) Después de otros t_D = 2.5[µs], habilitar "high" la entrada START conversion (se resetea el "*Successive Approximation Register*" (SAR))

4) Mandar a "low" la entrada START conversion y ALE (empieza la conversión)

5) Detectar (por poleo) cuándo la salida EOC pasa de "high" a "low"

6) Detectar (por poleo) cuándo la salida EOC pasa de "low" a "high" (la conversión termina : la información binaria {byte_dato} está disponible en la salida digital del ADC {D7-D0})

7) El µP lee los 8 bits {byte_dato} proporcionados por el ADC (si la salida digital del ADC se conecta al bus de datos del sistema, el proceso de lectura debe habilitar "high" la entrada OE del ADC ; si la salida digital del ADC está conectada a un puerto paralelo de entrada, el µP lee el dato seleccionando dicho puerto)

.....

Por ej., si el usuario desea digitalizar la información analógica que se aplica a la entrada analógica IN5 y guardar el valor digital resultante en una localidad de memoria RAM, deberá incluir dentro de su programa de aplicación las siguientes instrucciones :

```
MOV     BL,05h          ; cargar dirección de IN 5
CALL    f_AD_Read       ; efectuar conversión A/D
MOV     memloc,AL       ; almacenar en memoria el
                        ; valor digital resultante
```

.....

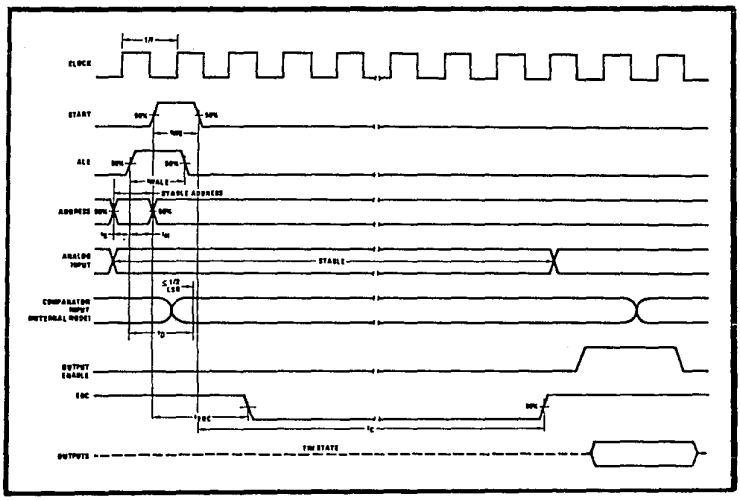
La rutina de conversión A/D descrita aqui funciona por Poleo, pero puede implementarse una rutina equivalente bajo un esquema de interrupción al µP, en que el estado (transición L-H) de la salida EOC del ADC inicie la ejecución de una rutina de lectura de la información digitalizada, por el hecho de conectar dicha salida (EOC) a la entrada de interrupción IR2 del PIC 8259A.

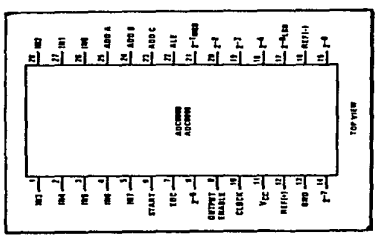
En la { *figura* 2-4 } se muestra el Diagrama de Tiempos (formas de onda) para adquisición de datos con el Convertidor Analógico/Digital ADC0808 / ADC0809

{ *figura* 2-4 }

Diagrama de Tiempos para el Convertidor A/D ADC0808/ADC0809

Linear Databook (National) ; ADC0808/ADC0809





El PPI - #2 es para empleo del usuario que lo puede programar según lo requiera en su programa de aplicación particular.

Al energizar al sistema y después de un RESET de hardware la rutina de Inicialización de periféricos residente en EPROM de la μC-Interface, configura a los 3 puertos paralelos programables del PPI-#2 (82C55A) en Modo 0 :

	Puerto A : OUTPUT
	Puerto B : INPUT
	Puerto C : INPUT
Puertos PA y PB :	{ *Buffered* : 74LS245 }
Puerto PC :	{ *Not Buffered* }

sin embargo, en su programa ejecutor de aplicación, el usuario puede cambiar está configuración inicial (incluso cambiar de Modo 0 a Modo 1 ó 2).

2.5.6

CONVERTIDOR DIGITAL/ANALOGICO Y CONVERTIDOR ANALOGICO/DIGITAL

- Convertidor Digital/Analógico
  (DAC) MC 1408 [P8]
  { 8-*bit multiplying D/A converter* } Resolución : 8 bits
  Max. error : ±0.19%
  Settling time : 300ns

  Circuito de Referencia (V_{REF} = +2.0 V)
  Amplificador Convertidor de Corriente a Voltaje
  (Salida I_{OUT} del DAC)
  Amp.Op. LF356 (Low noise, 5MHz BW, High Zin {$10^{12}\Omega$})
  R_o = (4.7kΩ)+(Trimmpot 1kΩ ...(1 vuelta))

Para mayor información sobre especificaciones, conexiones a un sistema y circuitos de aplicación, consultar :

Linear / Interface Devices Databook (Motorola) : MC1408, MC1508

- Convertidor Analógico/Digital
  { ADC }
  { *8-bit A/D converter* }

  ADC 0809 (National SC)
  8 entradas analógicas seleccionables por Multiplexor interno : IN0 (000) a IN7 (111)
  Resolución : 8 bits
  Max. error : ± 1 LSBit

  Tiempo de Conversión : (T_c)
  100 µs (typ) a F_c = 640 kHz
  En SDM88-PC ... T_c: alrededor de 50 µs, pues frecuencia de operación (CLOCK) es : F_c = 1.19 MHz

  Circuito de Referencia y Alimentación (V_{REF} = +5.12V)
  El ADC0809 requiere menos de 1 mA de corriente de alimentación, por lo que la implementación de su fuente se puede derivar a partir del circuito de Referencia.
  Diodo Zener LM336 Z-5.0 (ajuste @ 5.12 V)
  Amp.Op. (Seguidor de Voltaje integrado y Buffer) LM310
  $V_{REF}^{+} = V_{CC}$ = +5.12 [V]

  Con este voltaje de Referencia el ADC tendrá 256 (2^8) pasos (steps) de 20 [mV] cada uno.

  Operación en modo "Poleo" o "Interrupción" seleccionable por "jumper" : {Sensado de terminal EOC = *End-Of-Conversion*}

  Operación en modo "Poleo" (Polling)
  NOR (74LS02) de ($\overline{RD}$) y CS para ADC_EOC ($\overline{Y5}$)
  Buffer Tri-state (74LS126) para leer estado de EOC por Bus de Datos del µP (D0) {Puerto IN de 1-bit}

  Operación en modo "Interrupción" (Interrupt)
  señal EOC dirigida a IR2 (int input) del PIC 8259A

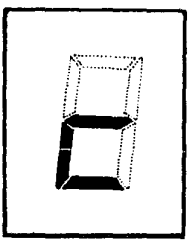
Para mayor información sobre especificaciones, conexiones a un sistema y circuitos de aplicación, consultar :
Linear Databook (Vol.2) (National S.C., 1988) : ADC0808, ADC0809

2.5.7

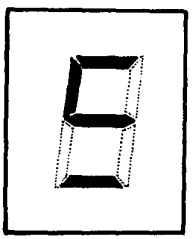
DESPLIEGUE INDICADOR DE ESTADO (STATUS) EN LA OPERACION DEL SISTEMA SDM88-PC

* Sistema (circuito) indicador de "Status"
en la operación del SDM88-PC
(Bus de Datos μP) ⟶ (74LS373 como puerto de salida) ⟶
...⟶ (Decodificador BCD-7Seg: 7448) ⟶
...⟶ (Display numérico Cátodo común : MAN6780)

Los despliegues que indican al usuario el estado (status) de operación en que se encuentra en un momento dado la μC-Interface del SDM88-PC (System Status prompts) son los siguientes :



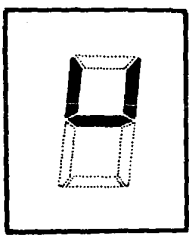
Rutina de Inicialización ejecutándose { INIT prompt }
(Areas de memoria y puertos I/O)
Se presenta cuando el sistema se energiza por primera vez o se aplica un RESET de hardware.
(Duración aproximada: 1 [s])
[7448_DCBA = 1010 = 0Ah]



Sistema Listo o Disponible
{ READY prompt }
El sistema está listo para recibir una nueva orden o comando.
Este despliegue es característico cuando termina de ejecutarse exitósamente un programa-comando o un programa de aplicación que no traba al sistema en un "loop" infinito.

La comunicación entre la µC-PC(PS) y la µC-Interface del **SDM88-PC** está permitida.

[7448_DCBA = 1101 = 0Dh]



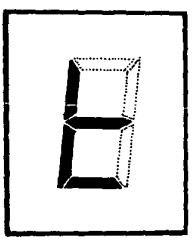
Sistema Ocupado o No_Disponible

{ BUSY prompt }

El µP 8088 de la µC-Interface está realizando alguna tarea o programa y, por tanto, no puede atender alguna solicitud del usuario.

La comunicación entre la µC-PC(PS) y la µC-Interface del **SDM88-PC** , NO está permitida (no es posible).

[7448_DCBA = 1100 = 0Ch]



Comunicación (Transferencia) de datos (bytes) en proceso

{ DATA TRANSFER prompt }

Se está efectuando un proceso de transmisión de bytes {Datos o Código hex de programa (µP)} desde la µC-PC(PS) hacia RAM de la µC-Interface dentro del ambiente (contexto) del sistema **SDM88-PC** .

[7448_DCBA = 1110 = 0Eh]

CAPITULO III

SISTEMA OPERATIVO (SOFTWARE DE CONTROL) EN EL SISTEMA DIGITAL DE DESARROLLO SDM88-PC

SISTEMA DE DESARROLLO
BASADO EN EL MICROPROCESADOR 8088
Y EN UNA MICROCOMPUTADORA
TIPO PC (PERSONAL COMPUTER)
O PS (PERSONAL SYSTEM)
(IBM COMPATIBLE)

INTRODUCCION

En este capítulo se describe el *Software* de Control ó SISTEMA OPERATIVO del Sistema Digital de Desarrollo SDM88-PC (Sistema de Desarrollo basado en el Microprocesador 8088 y en una microcomputadora tipo PC (ó PS)) .

Como ya se ha expuesto, el sistema **SDM88-PC** consta de dos microcomputadoras trabajando conjuntamente : la Microcomputadora PC (PS) y la Microcomputadora-Interface (*Hardware*) basada en un microprocesador 8088 externo que administra su propia memoria y controla sus propios periféricos I/O ; la µC-Interface es controlada por la µC-(PC ó PS). Este hecho supone la existencia de dos programas de control ejecutándose en paralelo : uno residente en la memoria de la µC-PC(PS) y otro en la memoria de la µC-Interface.

El *Hardware* conjunto del sistema es administrado por el *Software* de Control que, en unos casos actúa exclusivamente sobre los recursos de la µC-PC , y en otros casos establece un protocolo de comunicación entre la µC-PC (maestra) y la µC-Interface (esclava) para transferencia de comandos, código y datos a través de un canal de enlace serie (entre puertos serie).

El Software de control del **SDM88-PC** está configurado de tal manera que la parte residente en RAM de la µC-PC(PS) (después de ser cargada de disco), funciona como una interface Hombre-Máquina, ya que mediante despliegues autoexplicativos bajo una filosofía de Menúes, proporciona al usuario distintas alternativas para visualizar, introducir, manejar, organizar, almacenar y accesar información, y para operar directamente sobre los componentes de la µC-Interface (circuitos integrados programables de soporte controlados por el microprocesador 8088 externo). Para ello se

aprovechan los recursos que ofrece la Microcomputadora PC (PS) : (despliegue en pantalla, interface de teclado, gran capacidad de almacenamiento (RAM y unidades de disco flexible y/o duro), extenso soporte de *software*, *hardware* y periféricos en constante desarrollo, etc.).

Por otro lado, la parte (del Software) residente en EPROM de la μC-Interface cuenta con las rutinas necesarias para aceptar, almacenar en RAM y ejecutar los comandos y programas (código-μP8088) enviados por la μC-PC(PS), (también pueden aceptarse datos aislados: cualquier info. hex (bytes)). El contenido de la EPROM también contempla rutinas de Inicialización para los periféricos base y zonas de memoria de la μC-Interface, así como diversas rutinas de control elementales necesarias para el funcionamiento del sistema (Transmisión y Recepción de bytes por puerto serie, establecimiento de velocidad de transmisión) o para el manejo de sus dispositivos (Rutinas de Servicio a Interrupción, Rutina para generación de retardos de duración programable, Rutina para el manejo del ADC por poleo, etc. y más que se pueden implementar conforme se añadan o controlen otros dispositivos).

En este capítulo se describirá pues, la estructura del Software contenido en la EPROM de la μC-Interface, los requerimientos y justificaciones en el diseño y desarrollo del programa de control en la μC-PC(PS) (empleo de BASIC y Lenguaje Ensamblador), y se explicarán los distintos comandos (opciones) que el usuario tiene a su disposición desde la μC-PC(PS).

No debe perderse de vista que el sistema de desarrollo **SDM88-PC** está concebido como una herramienta auxiliar para el control de procesos o para el diseño (creación) de instrumentos-prototipo que den solución a una necesidad especifica.

Estos sistemas de aplicación desarrollados a partir del **SDM88-PC** son controlados lógicamente, por programas (*Software*) creados por el usuario en la PC (PS) . Posteriormente (dentro del ambiente del SDM88-PC), los programas en código hex del μP 8088, son transferidos (cargados) a RAM de la Microcomputadora-Interface, para actuar sobre el *Hardware* de la aplicación a través del *Hardware* de la Interface (con o sin la supervisión de la PC, según se requiera).

3 . 1

PROGRAMA MONITOR (SISTEMA OPERATIVO) RESIDENTE EN EPROM

(MICROCOMPUTADORA-INTERFACE)

(Hardware externo a la Microcomputadora PC (PS) en el Sistema de Desarrollo **SDM88-PC**)

- Memoria EPROM
  27C64 ó 2764 (UVEPROM)
  (8192 bytes)
  Rango { FE000h-FFFFFh } ... direcciones fisicas en el sistema

En la µC-Interface del sistema **SDM88-PC**, la memoria UVEPROM se usa para almacenar :

- el Programa Monitor o Sistema Operativo para la µC-Interface (Hardware externo a la µC-PC (PS))

- las rutinas para Inicialización del sistema (POWER-UP y RESET) { Areas de Memoria y Puertos I/O }

- los Datos y Tablas de apuntadores iniciales del sistema

- las rutinas de control para periféricos (subrutinas auxiliares administradoras de hardware para usar en programas de usuario)

- las rutinas de comunicación {µC-Interface} ⟷ {µC-PC(PS)} a través de puertos serie

- las rutinas de servicio a interrupción

- otro código definitivo de control (presente o futuro (expansión)), agregado por el usuario

La Localidad de Memoria en EPROM (offset hex) es igual al offset (IP = Instruction Pointer) en el seguimiento de los programas ejecutados por el µP 8088, si se hace previamente CS = FE00h .

El Mapa de Memoria descriptivo del Software de control residente en la EPROM es el siguiente :

Dirección Física { hex }	Localidad en UVEPROM 2764 { hex }	C O N T E N I D O
FE000	0000	Area Reservada (no usada) :
.		para Expansión futura u otros usos
.		o por compatibilidad con el formato
.		*.COM de IBM-PC (asignación de PSP
.		at run time) : { 100h Bytes }.
.		Esta convención da la facilidad de
.		desarrollar rutinas de EPROM en una
.		PC con algún Ensamblador comercial
.		y cargar (grabar) directamente el
...	00FF	el *.COM generado a la EPROM.
		{ *.COM = archivo binario }
FE100	0100	Area de Salto (Jump) o derivación
.		(Case) inicial.
.		Salto al área principal de código
.		de programa (offset=0500h) o
.		rutinas iniciales para derivación
.		de control (estructura tipo CASE).
.		Salto inicial a la zona de código
.		ejecutable o bien rutina para
.		derivación de control p.ej.

```
                        a distintas aplicaciones   o
.                       a distintas opciones de
.                       inicialización de periféricos del
.                       sistema, que se lleguen a grabar en
.                       EPROM.
...     01FF            { 100h Bytes }

FE200   0200            Apuntadores (Pointers) de
.                       Interrupción (Tabla de Vectores).
.                       Direcciones absolutas {CS:IP} de
.                       rutinas de servicio a interrupción.
        0200            Apuntadores de interrupción
                        dedicados (5) :
        0213            INT_type0 --> INT_type4 pointers
        0214            Apuntadores de interrupción
                        reservados (para compatibilidad con
                        futuros productos de Intel) (27) :
        027F            INT_type5 --> INT_type31 pointers
Apuntadores de interrupción disponibles (32) :
(type32 a type63 , no se implementó hasta type255)
                        Apuntadores de interrupción
                        disponibles para el PIC 8259A
                        instalado (8) :
        0280            INT_type32 pointer {HIGH PRIORITY}
                        8259A (IR0) ... USART RxRDY
                        "Receive_Byte (8251A <-- PC)"
        0284            INT_type33 pointer
                        8259A (IR1) ... USART TxRDY
                        "Transmit_Byte (8251A --> PC)"
        0288            INT_type34 pointer
                        8259A (IR2) ... ADC  EOC (L-H)
                        (Read A/D data digitized)
        028C            INT_type35 pointer
                        8259A (IR3)
```

```
                0290            INT_type36 pointer
                                8259A (IR4)
                0294            INT_type37 pointer
                                8259A (IR5)
                0298            INT_type38 pointer
                                8259A (IR6)
                029C            INT_type39 pointer  {LOW  PRIORITY}
                                8259A (IR7)
                                Apuntadores  de  interrupción
                                No usados pero Disponibles  (24) :
                02A0            INT_type40 pointer
                                    |
                                    ↓
                02FC            INT_type63 pointer
                ...
          ...   02FF

FE300           0300            "System Data Area" :
            .                   Area de Datos para el Sistema
            .                   (Inicialización de variables que el
            .                   sistema puede usar ... locs.mem.)
            .                   { 200h = 512 bytes }
            .                   Tabla de Apuntadores de dirección
            .                   de Subrutinas FAR ... {03D0-03FB}
            .                   Loc.Mem. para llevar conteo de
            .                   reloj de tiempo real implementado
            .                   por rutina NMI  :  variable  "TIME"
            .                   ... {03FC-03FD}
          ...   04FF

                "Main Program Code Area" :
FE500           0500            Inicialización del sistema :
            .                   * Inicialización de Areas de
            .                     servicio en memoria :
            .                   + STACK {SS,SP} (a zona de RAM).
```

+ Transferencia de Tabla de Vectores de Interrupción a RAM.
+ Inicialización de datos del Area de Servicio del sistema : Transferencia de apuntadores y datos de control (constantes y variables) desde EPROM a RAM para trabajar dentro de un único Data Segment residente en RAM.
+ Inicialización de DS (Data Segment) en Area de Servicio del sistema (RAM) :
DS=0030h { dir.fis.= 00300h }
{SS, SP, DS, ES, (CS via FAR jump)}
* Inicialización de Puertos I/O (Periféricos) :
+ "Status Report Display"
+ 82C55A { PPI #1 } para control y datos del DAC y del ADC
+ 82C55A { PPI #2 } puertos paralelos de propósito general
PA=OUT ; PB=IN ; PC=IN
(pueden reprogramarse después)
+ 8254 :
Counter 0 : Generador de Baudaje
Frec.base inicial = 19200 Hz
Counter 1 : Reloj del Convertidor A/D ... 1.19 MHz
Counter 2 : Entrada para NMI (interrupción cada 1/100 s para actualizar reloj de tiempo real)
Frec. = 100 Hz
+ 8259A (PIC)
Int_type32 para IRO

		+ 8251A {USART} {COMMAND word : Internal RESET} {Vel.Transm.inicial= 19200Hz/64 = = 300 Bauds ... (bit/s) } {MODE word = 4Fh = 1 STOP bit , NO parity , Char.Length= 8 bits , Baud Rate Factor = 64X } {COMMAND word : NO RESET = 37h = NO Hunt mode , NO IR , RTS=low , Reset Error Flags , Normal Oper., Rx Enable , DTR=low , Tx Enable } • Habilitación de Interrupciones mascarables [STI] • Espacio disponible para inicialización de periféricos futuros: EPROM virgen {0600-07FF}
...	07FF	
FE800	0800	Rutina principal de control (administración) de la Interface. Transferencia de información_bytes { [µC-PC] ⟶ [µC-Interface] } : rutina que recibe de la µC-PC(PS) una cadena de bytes (que integran un programa en lenguaje de máquina (µP 8088)), posicionándolos en una zona especifica de la mem. RAM y autoejecutando dicho programa (código-µP) una vez que se ha terminado de bajar a RAM. Esta rutina se ejecuta automáticamente al restablecer el sistema o bien al energizarlo inicialmente.

```
                                Al bajar un programa (código-µP) a
                                la µC-Interface, la secuencia que
                                envia la µC-PC(PS) es la siguiente:
                                1) Enviar un caracter de
                                   identificación que indica a la
                                   µC-Interface que los caracteres
                                   que siguen representan a un
                                   programa para el µP 8088.
                                2) Transmitir 4 bytes con
                                   información referente a la
                                   dirección inicial y a la
                                   dirección final del programa a
                                   bajar.
                                3) Transmitir la cadena de bytes
                                   correspondiente al programa que
                                   se esté bajando.
                                *) Una vez bajada la cadena de
                                   bytes, la µC-Interface
                                   efectúa un salto (jump) a la
                                   "  dirección  inicial  " ,
                                   autoejecutándose asi el programa
                                   que esa cadena representa.
                                Programa posicionado en RAM :
                                DS = 0030h      Dir.Fis.base=00300h
          ...   087F            { Prg_LoadAutoexec }

FE880             0880          Rutina de propósito general para
                                transferencia de una cadena de
                                bytes (código o datos) desde µC-PC
                                a RAM de µC-Interface.  Dirección
                                inicial y final (offset) asignadas.
                                Posicionamiento en RAM :
                                DS = 0030h      Dir.Fis.base=00300h
          ...   08FF            { DownLoad_bytes }
```

FE900	0900	Area de memoria disponible para
.		otro código definitivo (firmware)
.		correspondiente a futuras rutinas
.		de control, rutinas de aplicación
.		especifica, etc.
.	.	(EPROM virgen) : {0900-0FFF}
...	0FFF	{ AvailableMEM }
FF000	1000	Subrutinas de propósito general
.		para el control de dispositivos,
.		{ Common Prg. NEAR & FAR
.		Subroutines } :
.		+ Generación de retardo programable
.		N [milisegundos]
.		{nDelay,fDelay} (NEAR & FAR)
.		+ Recepción de un caracter por
.		8251A (por poleo)
.		{nReceive_Char,fReceive_Char}
.		+ Transmisión de un caracter por
.		8251A (por poleo)
.		{nTransmit_Char,fTransmit_Char}
.		+ Inicialización de 8251A para
.		cualquier razón de Baudaje
.		(velocidad de transmisión) y
.		cualquier formato Tx/Rx.
.		Deberán introducirse las palabras
.		de MODE y COMMAND para 8251A y el
.		factor de escala para fijar
.		frecuencia en 8254.
.	.	{ nIniUART_X_ModCmd,
.		fIniUART_X_ModCmd }
.		+ Control y Lectura de dato
.		digitalizado con conv. A/D
.		{n_AD_Read,f_AD_Read} (Por poleo)

```
                .                       Espacio disponible para otras
                .                       subrutinas residentes en EPROM
                .                       presentes o futuras :
                .                       (EPROM virgen) : {1280-1AFF}
            ...     1AFF

FFB00           1B00                    Rutinas de Servicio a Interrupción.
                .                       {Rutina NMI} "Real-Time-Clock" :
                .                       actualización de loc.mem. TIME cada
                .                       1/100 s .
                .                       Rutinas _ Servicio a Interrupción
                .                       INTERRUPCION mascarable procesada a
                .                       través del PIC 8259A .
                .                       USART Rx & Tx Int. , ADC_EOC Int. ,
                .                       etc.}
                .                       Area disponible para futuras
                .                       rutinas de INT (EPROM virgen).
            ...     1FEF

FFFF0           1FF0                    Area de codificación de Salto
                                        Inicial después de RESET-Hardware
                                        o encendido del sistema: (15 bytes)
                                        "Initial {RESET} Startup-jump area"
                                        contenido hex .. mnemónicos
                1FF0            90      nop
                1FF1            90      nop     ;Inter-Segment jump
                1FF2            EA      jmp     ; FAR jump code
                1FF3            00      IPLOW
                1FF4            01      IPHIGH  ; ... IP = 0100h
                1FF5            00      CSLOW
                1FF6            FE      CSHIGH  ; ... CS = FE00h
                        Startup RESET jump Dirección Física :
                        {CS}*10h+{IP} = {FE00h}*10h+{0100} =
                                      = FE000+0100 = FE100 (EPROM)
```

```
                1FF7
                 .                  área no usada
FFFFF           1FFF
```

.....

Valores default para los Registros de Segmento en el µP 8088 durante accesos a EPROM (operación en Area de EPROM) :

CS = DS ó CS ≠ DS

CS = FE00 h .....{ por dirección física : FE000h } (EPROM)

DS = FE00 h (en EPROM) ó DS = 0030 h (en RAM)

...{FE000h} ...{00300h}

SS = 0010 h .....{ por dirección física : 00100h }
(permanece en RAM)

SP = 00FE h

[00100h + 00FEh = 001FEh = Stack_Top]

.....

Después de Inicializado el sistema, la memoria RAM queda de la siguiente forma :

* Area Exclusiva para el sistema : (direcciones físicas)
  - Vectores de Interrupción : 00000 ⟶ 000FF h
  - Stack : 00100 ⟶ 001FF h
  - Expansión futura u otros usos : 00200 ⟶ 002FF h
* Area Accesible para el Usuario : { CS=0030h }
  - Area para uso del sistema :
  - (cargado de programas-comando autoejecutables) :
    - Dirección Física :
    - 00300 ⟶ 003FF h
    - Offset dentro de CS :
    - 0000 ⟶ 00FF h

Area para programas (código & datos) del usuario :

Dirección Física :

00400 ⟶ 017FF h

ó 00400 ⟶ 03FFF h

017FF h {5120 Bytes instalados}

ó 03FFF h {15360 Bytes decodificados}

Offset dentro de CS :

0100 ⟶ 14FF h (instalados)

0100 ⟶ 3CFF h (decodificados)

Valores default para los Registros de Segmento en el μP 8088 durante accesos a RAM (operación en Area de RAM) :

CS = DS = 0030 h .....{ por dirección física : 00300h }

SS = 0010 h .....{ por dirección física : 00100h }

SP = 00FE h .....{ 254 bytes = 127 words para operaciones de stack }

[00100h + 00FEh = 001FEh = Stack_Top]

Para autoejecución de programas-comando y ejecución de programas del usuario :

00300 h = Dirección Física Base para CS y DS en Area de RAM

=> CS = DS = 0030 h { para condiciones de operación Programas de Usuario }

Si el programa del Usuario es un archivo binario (*.COM) (código hex), debe ser cargado sin modificaciones (y sin el *header* extra de 100h bytes) directamente de disco o arreglo de memoria o buffer a la dirección física 00400h . Este programa no debería alterar CS=0030h (=DS) como base para trabajo en RAM pero nada impide hacerlo. Considerando que CS=0030 (dir.fís.=00300h) es la dirección base para cargado de código en RAM de μC-Interface, el usuario debe cargar sus programas de aplicación a partir del offset 0100h , ya que del offset {0000 al 00FFh} se considera área ocupada por el sistema para cargar Programas-comando.

3.2 EMPLEO DE UN LENGUAJE DE ALTO NIVEL (BASIC) Y LENGUAJE ENSAMBLADOR PARA EL DESARROLLO DEL PROGRAMA MONITOR O SISTEMA OPERATIVO CARGADO DE DISCO A RAM DE LA MICROCOMPUTADORA-PC (PS)

Los programas que requieren de mucho manejo de procesos binarios y manipulación de Hardware son normalmente escritos en lenguaje ensamblador porque a ese nivel se tiene un control directo sobre el Hardware. Sin embargo, programas administrativos, cientificos, etc., que involucran principalmente la manipulación de grandes cantidades de datos son escritos generalmente en lenguajes de alto nivel como BASIC, Pascal o C. Para programas tales como los de "Comunicaciones", que involucran ambos tipos de operaciones, el programa principal es generalmente escrito en un lenguaje de Alto o Medio nivel, y desde él se invoca a subrutinas en lenguaje Ensamblador para realizar los procesos de manejo binario y de hardware según se requiera.

El lenguaje BASIC (para las µC-IBM_PC(PS) y compatibles) cuenta con varios comandos y funciones especificamente diseñadas para ayudar en el desarrollo y operación de interfaces de Hardware con el Bus del sistema. Muchas aplicaciones de interfaces pueden implementarse usando BASIC, que es un medio de programación más sencillo que el lenguaje Ensamblador (8088/86). Además el BASIC ofrece al programador diversos comandos para el despliegue de información en pantalla, para entrada de datos por teclado, para comparación de condiciones lógicas que determinan el flujo de un programa, para el procesamiento de cantidades aritméticas y lógicas, para el almacenamiento, organización y procesamiento de datos (arreglos vectoriales y matriciales), para graficación y

despliegue de datos, etc., y en general, para el manejo de los recursos disponibles en la Microcomputadora.

El BASIC cuenta con un poderoso conjunto de instrucciones para el manejo de archivos de datos en Disco y, por extensión, el Puerto Serie de la Microcomputadora, puede ser manejado como si se tratara de uno de dichos archivos de disco {OPEN, CLOSE, PRINT#, INPUT#, etc.} y se permite al usuario establecer (habilitar o deshabilitar) los distintos elementos de temporización y control involucrados en la configuración de un protocolo de comunicación serie asincrona, que una vez inicializado por programación, el BASIC se encarga de operar automáticamente.

El BASIC permite también, implementar rutinas de detección y manejo de errores, así como de introducción exclusiva de respuestas acotadas (permisibles), conceptos que son elementos fundamentales en la elaboración de Programas de control robustos y bajo una filosofía de "interface Hombre-Máquina amigable".

Por estas razones y además por la compatibilidad y portabilidad buscada para el diseño a futuro, se escogió al BASIC como herramienta base de desarrollo para el programa de control en la Microcomputadora PC (PS) (después de analizar PASCAL y C).

Uno de los mayores intereses o preocupaciones cuando se usa BASIC es la rapidez de ejecución y respuesta ("performance") que tendrá el programa. El BASIC intérprete, es relativamente lento cuando se compara una misma función implementada en lenguaje Ensamblador. Sin embargo, si existen requerimientos de rendimiento o desempeño (performance) que el BASIC no pueda satisfacer adecuadamente, puede aún usarse BASIC para la mayor parte del programa de aplicación y emplear subrutinas en lenguaje Ensamblador para las funciones en que el tiempo sea un factor

crítico. Una alternativa que aún permite el uso de BASIC, y con un incremento en el desempeño general (performance), consiste en la COMPILACION del programa fuente de BASIC usando un Programa Compilador de BASIC comercial (preferiblemente de *Microsoft* ó de *IBM* para asegurar compatibilidad con el Hardware más común existente en el mercado). Un programa compilado típicamente correrá de 5 a 25 veces más rápido que la versión intérprete. Finalmente, pueden crearse programas híbridos con BASIC compilado y lenguaje Ensamblador para optimizar el manejo de código y memoria en la Microcomputadora e incrementar la velocidad de ejecución. Este último caso fue el empleado en el desarrollo del Sistema Operativo residente en la µC-PC(PS) del sistema SDM88-PC , obteniéndose como producto final, un único programa Ejecutable (SDM88PCC.EXE) que, con el sólo tecleo de su nombre (y CR), es cargado de disco a la memoria RAM de la Microcomputadora PC (PS) y se autoejecuta introduciendo inmediatamente al usuario, al ambiente de operación del sistema SDM88-PC (µC-PC(PS) y µC-Interface).

La operación de un Programa Intérprete consiste en leer una instrucción en lenguaje de alto nivel del programa fuente, convertir esa instrucción a código de máquina y, si no necesita información de otra instrucción, ejecutar el código para esa instrucción inmediatamente. El intérprete lee entonces la siguiente instrucción fuente de alto nivel, la traduce, y la ejecuta. Esto pasa con los programas en BASIC intérprete. La ventaja de usar un intérprete es que si se encuentra un error durante la elaboración del programa, basta tan sólo corregir el programa fuente e inmediatamente reejecutar el programa. La mayor desventaja es que el programa intérprete corre de 5 a 25 veces más lento que después de ser compilado. La razón es que con un intérprete cada instrucción debe ser convertida a código de máquina cada vez que el programa sea corrido : "el tiempo de conversión es parte del tiempo de ejecución".

En contraste, un Programa Compilador lee TODO el programa fuente elaborado en lenguaje de alto nivel, y después de realizar sobre él dos o más pasadas, lo convierte o traduce todo a una versión de código de máquina relocalizable. Sin embargo, antes de que el programa pueda ser ejecutado, esta versión de código objeto relocalizable debe ser ligada (linked) con algunos otros módulos objeto requeridos procedentes de las librerias del sistema, alguna libreria del usuario, o rutinas en lenguaje Ensamblador. El archivo de salida resultante del Programa Ligador (Linker), es entonces "localizado" (se le asigna una dirección absoluta a fin de que pueda ser cargado en memoria). Finalmente, el programa localizable completo (*.EXE) es cargado en memoria y puede entonces ser corrido sin ninguna traducción adicional. Por tanto, correrá más rápidamente que si fuese ejecutado por un intérprete. La mayor desventaja del método del compilador consiste en que si se encuentra un error, éste debe ser corregido en el programa fuente y la secuencia completa de Compilación-Ligado-Carga debe ser repetida.

La habilidad de invocar (CALL) subrutinas elaboradas en lenguaje Ensamblador desde BASIC, proporciona un método muy poderoso para el desarrollo de interfaces para aplicaciones, bajo requerimientos de alto rendimiento o desempeño (High-performance), manteniendo aún la facilidad de programación del lenguaje BASIC.

El procedimiento para invocar subrutinas de lenguaje Ensamblador desde un lenguaje intérprete de alto nivel es bastante confuso, especialmente en el caso de usar rutinas múltiples o anidadas, debido a la manera en que el intérprete administra la memoria del sistema { ver Apéndice "D" de "Microsoft GW-BASIC Interpreter : User's Guide " version 3.22 (1986-1987) ó Apéndice "C" de "IBM BASIC Reference Manual" }. Es más recomendable la opción de BASIC compilado para la mayoria de los programas híbridos, debido a la ventaja obvia en la velocidad de ejecución y

a la facilidad con que pueden ser llamados los módulos en lenguaje Ensamblador.

Invocar subrutinas de lenguaje Ensamblador desde programas compilados es más simple, porque los módulos objeto producidos por el Programa Ensamblador pueden ser simplemente ligados con módulos objeto generados por el Programa Compilador y módulos objeto de las Librerías. La principal tarea cuando se usan módulos creados con Ensamblador con módulos creados con Compilador, es asegurarse de que dichos módulos puedan "encontrarse" y "comunicarse entre sí". Puesto que los compiladores comunes de Pascal, C y BASIC emplean convenciones similares para lograr este fin, se decidió usar los productos de "Microsoft" para el desarrollo del Programa "Sistema Operativo" del **SDM88-PC** (en la μC-PC(PS)), (además con vistas a la compatibilidad y portabilidad del diseño).
Concretamente se emplearon :

1) *Microsoft QuickBASIC Compiler*
   version 4.00 (1982-1987)
2) *Microsoft Macro Assembler*
   version 5.10 (1981-1988)
3) *Microsoft Overlay Linker*
   version 3.64 (1983-1988)

La instrucción "CALL" de BASIC tiene el siguiente formato :

```
CALL numvar(var1,var2,var3,...,varN)
```

Para programas en BASIC Compilado, "numvar" es el nombre de la subrutina en lenguaje Ensamblador que se desea invocar. Cuando el programa en BASIC y el módulo en lenguaje Ensamblador son ligados, el "Linker" establece la conexión requerida entre la llamada (CALL) y la subrutina nombrada. "var1", "var2", "var3", etc. representan los nombres de las variables que serán pasadas "hacia" o "desde" la rutina en lenguaje Ensamblador. Pueden transferirse (pasarse) variables numéricas o variables "string".

Una de las ventajas del desarrollo de programas híbridos (Lenguaje de Alto Nivel + Subrutinas en Ensamblador), es que se pueden emplear "interrupciones por software" y a través de ellas, invocar las rutinas deseadas de muchos programas diferentes en un sistema y dentro de un ambiente de programación como el establecido por la IBM-PC(PS) y compatibles. Un buen ejemplo de esto lo constituye el BIOS (*Basic Input/Output System*) de las μC-IBM PC ó PS.

Las Microcomputadoras PC ó PS tipo IBM, tienen en sus ROMs una colección de rutinas, cada una de las cuales realiza algunas funciones especificas, tales como {lectura de un caracter desde el teclado}, {escritura de algunos caracteres a pantalla CRT}, {lectura de información contenida en un disco posicionándola en memoria RAM}, etc. Esta colección de rutinas se conoce como BIOS. Las rutinas del BIOS son llamadas con instrucciones INT (Software interrupts) que pueden colocarse dentro de un programa en lenguaje Ensamblador desarrollado por el usuario para la PC, según sus requerimientos (consultar IBM PC Technical Reference Manual).

La principal ventaja de invocar rutinas de este modo, es que el programador no necesita preocuparse en proporcionar la dirección absoluta donde realmente reside la rutina (la zona de vectores de interrupción en RAM baja ya ha sido inicializada automáticamente al encender la μC), ni en tratar de ligar la rutina con su programa. Todo lo que el programador debe saber es el Tipo de interrupción (#) que ejecuta la rutina deseada y el formato de los parámetros que necesita pasarle a la rutina (normalmente inicializando con la información adecuada a determinados registros del microprocesador).

Este recurso también fue usado en el desarrollo del programa de control (Sistema Operativo) en la μC-PC(PS) del **SDM88-PC.**

DESCRIPCION DE LOS COMANDOS Y FUNCIONES DEL BASIC QUE PUEDEN SER UTILES EN EL DESARROLLO DE APLICACIONES DE INTERFACE CON HARDWARE.

A) "COMANDOS" PARA INTERFACE CON HARDWARE

1) Comando BLOAD

Permite cargar en memoria un archivo binario desde un archivo abierto (OPEN file), contenido en una cinta, un diskette o disco duro. Puede ser, por ejemplo, un programa en lenguaje Ensamblador o datos para un programa. Puede ser cargado en cualquier sitio dentro de la memoria disponible del sistema. (Cuidado en no encimar el BASIC o su espacio de trabajo).

2) Comando BSAVE

Es el complemento de BLOAD. Permite almacenar (save) información binaria (datos) en cualquier dispositivo que tenga un archivo abierto (OPEN file). Pueden salvarse datos de cualquier parte de la memoria del sistema. Este comando puede usarse p.ej., para salvar datos aislados adquiridos de una interface, en un archivo para su posterior análisis.

3) Comando CLEAR

Puede usarse para reservar espacio de memoria en el tope de la RAM del sistema para datos o programas en lenguaje Ensamblador.

4) Comando CALL

Permite invocar o llamar un programa o subrutina en lenguaje Ensamblador desde BASIC. También permite pasar datos, argumentos, o parámetros definidos como variables de BASIC, desde BASIC hacia el programa o rutina en Ensamblador y viceversa (reservar el espacio adecuado según el tipo de variable (Entera, String o Real) para que la rutina en Ensamblador transfiera (coloque) sus resultados al programa en BASIC).

5) Comando DEF SEG

Permite definir el valor del Segmento activo (current segment) para operación dentro de la memoria del sistema. Tipicamente se usa para definir el valor de la dirección inicial de Segmento en memoria para otros comandos y funciones de BASIC que hacen referencia a memoria directamente. Por ejemplo, debe ejecutarse antes de usar comandos o funciones del tipo de : BLOAD, BSAVE, CALL, VARPTR, USR, POKE, o PEEK .

6) Comando DEF USR

Permite definir el valor de la dirección inicial de Offset en memoria (dentro del Segmento fijado previamente por DEF SEG) , de una rutina en lenguaje Ensamblador que será invocada posteriormente por la función USR.

7) Comando POKE

Permite escribir datos (bytes) a cualquier localidad especifica en la memoria del sistema, desde un programa en BASIC. El dato escrito puede ser una constante o una variable. A fin de escribir el dato dentro de la dirección de Segmento correcta, ésta debe fijarse previamente con el comando DEF SEG . En el comando (POKE) se especifica la porción de la dirección correspondiente al "offset" (dentro del Segmento activo fijado por DEF SEG). Este comando puede usarse para posicionar en la memoria del sistema una subrutina en Ensamblador (que posteriormente será llamada por CALL o USR) o para posicionar datos cualesquiera (p.ej. recolectados por una interface) en un área de memoria asignada por el programador, o bien, para escribir datos, desde BASIC, en puertos I/O mapeados por memoria (memory-mapped I/O ports).

8) Comando WAIT

Puede usarse para muestrear una dirección de puerto I/O y comparar su valor con otro valor "máscara". Si la máscara y el valor del puerto no concuerdan, el programa de BASIC se suspende

hasta que lo hagan. Este comando es útil para sincronizar el programa de BASIC con un conjunto de condiciones externas. El tipo de comparación usada no es aritmética sino lógica (XOR). Este comando debe usarse con cuidado, puesto que si no se verifica nunca una comparación positiva, el programa caerá en un "loop" infinito.

9) Comando OUT

Permite escribir datos (bytes) a una dirección de puerto I/O desde un programa en BASIC. El dato escrito puede ser una constante o el contenido de una variable de BASIC. Un único byte_dato puede ser escrito a cualquiera de las 65536 direcciones de puerto I/O soportadas por la arquitectura del μP 8088/86 y en general por cualquier procesador nuevo de Intel procedente de esta familia. Con este comando, muchos de los puertos que controlan los modos de operación de los ICs (chips) de la μC-PC (PS), pueden ser modificados (reprogramados) a través de BASIC.

B) "FUNCIONES" PARA INTERFACE CON HARDWARE

1) Función INP

Es el complemento del comando OUT. Permite leer una dirección de puerto I/O y asignar el valor leído a una variable de BASIC. Se puede accesar cualquiera de las 65536 direcciones de puerto I/O soportadas por la arquitectura de los μP Intel de la familia 8088/86.

2) Función PEEK

Es el complemento del comando POKE. Permite leer cualquier localidad de memoria dentro del sistema y asignar el valor leído a una variable de BASIC. Esta función requiere ser precedida por el comando DEF SEG para fijar el valor adecuado de SEGMENTO. La porción de la dirección correspondiente al OFFSET se especifica como parte de la función PEEK.

3) Función USR

Permite invocar desde BASIC una subrutina en lenguaje Ensamblador especificando un único dígito. Los resultados son asignados a una variable de BASIC. Permite también especificar un único argumento para la rutina en Ensamblador. Esta función debe ser precedida por los comandos DEF SEG y DEF USR que fijan los valores de dirección SEGMENTO:OFFSET que identifican la localización en memoria de la rutina invocada.

4) Función VARPTR

Permite encontrar la localización en memoria de una variable de BASIC. El valor es regresado en una variable de BASIC y corresponde al OFFSET a partir del valor activo del registro de SEGMENTO fijado con el comando DEF SEG . Esta función se usa comúnmente para localizar la dirección de una variable de BASIC a fin de que pueda ser pasada (transferida) a un programa o subrutina en Ensamblador. Así, las rutinas en Ensamblador pueden obtener y almacenar datos contenidos en una variable de BASIC. Esto permite una fácil comunicación e intercambio de datos y parámetros entre programas en lenguaje Ensamblador y programas en BASIC.

3 . 3

PROGRAMA MONITOR O SISTEMA OPERATIVO CARGADO DE DISCO A RAM DE LA MICROCOMPUTADORA-PC (PS)

(Comandos disponibles para el usuario en el Sistema de Desarrollo **SDM88-PC**)

En el Sistema de Desarrollo **SDM88-PC** , el enlace de la Microcomputadora-Interface con la Microcomputadora PC (PS) para hacer uso de los recursos que esta última ofrece, es parte fundamental en el funcionamiento y operación del sistema en su conjunto. La Microcomputadora PC (PS) es, a través de su teclado, el dispositivo por el cual el usuario, desencadena y sincroniza acciones en la Microcomputadora-Interface (Hardware externo), organiza y transfiere información (bidireccionalmente) y visualiza resultados (status).

La Microcomputadora PC (PS) :

+ funciona como una terminal para el despliegue de información
+ interviene activamente como administradora del *Hardware* externo (Microcomputadora-Interface y/o circuitos de aplicación)
+ canaliza comandos para efectuar acciones especificas
+ maneja y organiza la información recibida (código, datos, programas {bytes}) para el microprocesador externo, almacenándola / recuperándola en/de su memoria o dispositivos de almacenamiento masivo (unidades de diskette o de disco duro)
+ arbitra la transferencia selectiva de bloques de información (bytes) desde o hacia la interface (memoria y/o puertos I/O externos)
+ permite la ejecución de programas que, originalmente almacenados en disco, son cargados a RAM de la interface

Estas y otras muchas actividades son supervisadas y seleccionadas por el usuario a través de pantallas ordenadas en "Menúes" dentro del ambiente creado por el Sistema Operativo del **SDM88-PC** en la PC (PS) ... programa "SDM88PCC.EXE".

De esta manera, el usuario podrá disponer de las herramientas necesarias para probar y depurar las secciones de *hardware* que vaya agregando sobre la infraestructura base de la μC-Interface, al aprovechar las facilidades que el sistema ofrece para escribir y ejecutar módulos de *software* que manejen o actúen directamente sobre dichos módulos de *hardware*. Así, el diseñador podrá obtener rápidamente un prototipo armado y funcionando para ver si el instrumento de aplicación especifica que busca desarrollar es "factible" de realizarse. Dicho instrumento podrá ser "Stand-alone" o dependiente de la μC-Interface del SDM88-PC para su operación y control.

Como ya se ha dicho, el Sistema **SDM88-PC** cuenta con un Programa Monitor que permite al usuario cargar y ejecutar sus programas y operar sobre el *Hardware* de la Microcomputadora-Interface y, a través de él, sobre otro *Hardware* agregado que constituye la aplicación del usuario; dicho Programa Monitor está dividido en dos partes :

a) la residente en la EPROM de la Microcomputadora-Interface (ya descrita) , y
b) la residente en memoria RAM de la Microcomputadora PC (PS) , después de ser cargada de disco (SDM88PCC.EXE) (Sistema Operativo para entrar en el ambiente de operación del **SDM88-PC**).

A continuación se describen los comandos que, organizados en menúes, están a disposición del usuario desde la μC-PC(PS) y constituyen el "Software de Control en la Microcomputadora PC (PS)".

Estando frente a una Microcomputadora PC o PS IBM compatible, y dentro del sistema operativo MS-DOS (versión 2.0 ó superior), el usuario deberá teclear :

SDM88PCC <CR>

para cargar de disco en la memoria RAM de la μC, el programa que lo introduce en el ambiente de operación del sistema de Desarrollo SDM88-PC.

Primero aparece una portada de presentación y después se pasa a un proceso para "Configuración del Area de Trabajo" en la μC-PC(PS), en donde se solicita al usuario que :

A) Indique los nombres de las unidades de disco que desea accesar (dar de alta como válidas) durante la sesión de trabajo { A,B,C o D o combinaciones de ellas }, (esto para su protección, impidiéndole entrar accidentalmente a unidades de disco donde, por tener información valiosa, no es conveniente trabajar).

B) Asigne (teclee) los nombres de los directorios activos (de trabajo) en las unidades de disco seleccionadas como válidas.

C) Indique el tipo de monitor con que cuenta la microcomputadora donde está trabajando (Monocromático o Color).

D) El puerto serie de la microcomputadora queda configurado automáticamente para operar inicialmente a una velocidad de transmisión (Tx/Rx) de 300 Bauds (después podrá cambiarse).

Una vez que el usuario acepta la "configuración del área de trabajo" introducida, el sistema efectúa una rutina de diagnóstico para indicar al usuario si la μC-Interface está conectada correctamente a la μC-PC(PS) y por tanto se puede establecer la comunicación entre ellas, o si, por el contrario, no se ha logrado establecer un canal de comunicación por una de las siguientes causas : (1): la Interface no está conectada a la PC o no está conectada correctamente; (2): el Baudaje en la PC es distinto al Baudaje en la Interface; (3): la PC y/o la Interface no están funcionando adecuadamente por lo que es necesario aplicarle(s) un

RESET de Hardware. Si la causa del error en la comunicación se debe a (1) o a (2), el usuario puede seguir haciendo uso de los comandos (disponibles en el sistema operativo) en los que no se requiera establecer comunicación directa entre la PC y la Interface, ya que de lo contrario reaparecerá el aviso de "Error en la comunicación PC ⟷ Interface" y no se procesará dicho comando.

En seguida aparece el Menú Principal :

* MENU DE COMANDOS PARA CONTROL DE INTERFACE { SDM88-PC ⟷ PC } *

1 > INTRODUCIR UN PROGRAMA EN CODIGO DE MICROPROCESADOR 8088, EN RAM DE PC
2 > CARGAR DE DISCO UN PROGRAMA PARA 8088
3 > GUARDAR EN DISCO UN PROGRAMA PARA 8088
4 > BAJAR PROGRAMA A MEMORIA {RAM} DE INTERFACE SDM88-PC Y AUTOEJECUCION
5 > PASAR A MENU DE COMANDOS DE EDICION
6 > PASAR A MENU DE COMANDOS PARA MANEJO DE {MEMORIA} Y {PUERTOS I/O} DE INTERFACE SDM88-PC
7 > DEFINIR BAUDAJE :
{PC_Puerto Serie RS-232C} ⟷ {INTERFACE SDM88-PC}
8 > PASAR A MENU DE COMANDOS PARA MANEJO DE DISCO EN PC
Q > TERMINAR LA SESION ... REGRESAR A * DOS *

A continuación se describe brevemente lo que acontece al seleccionar cada una de las opciones del :
MENU DE COMANDOS PARA CONTROL DE INTERFACE { SDM88-PC ⟷ PC } .

1 > INTRODUCIR UN PROGRAMA EN CODIGO DE MICROPROCESADOR 8088, EN RAM DE PC

Se solicita al usuario que teclee un nombre para el programa que se va a introducir, asi como las direcciones inicial y final

(OFFSETs dentro del SEGMENTO de RAM (CS=0030)) en formato hexadecimal de 4 cifras (0000 a FFFF). En seguida se despliegan en la pantalla de la Microcomputadora las direcciones (Offsets) en forma sucesiva, debiendo el usuario introducir cada vez el valor del byte correspondiente en notación hexadecimal de 2 cifras (00 a FF). El programa asigna a cada dirección una variable de tipo cadena (string) que representa lo que ha de almacenarse en las localidades de memoria correspondientes de la Interface SDM88-PC; de esta manera, la cadena de bytes que representa al programa, queda contenida en un arreglo cuyo tamaño es igual al número de bytes que integran el programa introducido por el usuario. En la ejecución de esta opción, al igual que en todas aquellas que involucren "Toma de Bytes en pantalla", el presionar <R> hace que se retrocedan direcciones sucesivamente a fin de que el usuario pueda Repetir la introducción de byte(s) anterior(es), y el presionar <Q> (Quit) termina el proceso de introducción de bytes en cualquier momento (la dirección (Offset) final se ajusta automáticamente al último byte introducido). Cabe señalar también que, en todas las opciones o comandos en que se solicite al usuario la introducción de alguna cantidad hexadecimal (byte_dato o dirección (SEG u OFFSET)), el programa realizará un análisis para verificar que todos los caracteres introducidos sean dígitos hexadecimales, tengan la longitud apropiada en cada caso (2 ó 4 dígitos) y estén dentro de rango permisible; de lo contrario el cursor en pantalla regresará a la posición del dato erróneo y/o se pedirá al usuario que reintroduzca la información requerida, tantas veces como sea necesario, hasta que sea reconocida como válida por el programa de control.

2 > CARGAR DE DISCO UN PROGRAMA PARA 8088

La Microcomputadora requiere del usuario el nombre del programa a tomar de disco y la unidad (Disk drive) correspondiente. Con esta opción sólo se pueden cargar de disco archivos previamente creados o con la opción (1) anterior o dentro

del ambiente del SDM88-PC (con otros comandos p.ej. Revisión de memoria); los archivos creados dentro del ambiente SDM88-PC son cargados a disco con la extensión "*.SDM" y de aquí en adelante se designarán como archivos *.SDM . Una vez que se ha ejecutado esta opción, la cadena de bytes correspondiente al programa que se ha tomado de disco queda en un arreglo similar al resultante de la opción (1) (en RAM de PC). Se preveen rutinas de detección de error en caso de que el archivo solicitado no exista dentro del directorio activo correspondiente de la unidad de disco introducida (A,B,C o D) y de ausencia de disco (floppy) o de compuerta de disco abierta en las unidades A y B . En caso de detectarse un error, el programa *.SDM o datos cargados anteriormente en el buffer de memoria RAM-PC, sigue accesible para el usuario en memoria; si no se detecta error, el buffer es sobreescrito con la información correspondiente al programa (datos) cargado.

3 > GUARDAR EN DISCO UN PROGRAMA PARA 8088

Mediante esta opción el arreglo de bytes correspondiente a un programa (o datos) determinado se guarda en un archivo de disco junto con sus direcciones inicial y final. Se indica al usuario el nombre con el que se almacenará en disco (siempre con extensión *.SDM) y se da oportunidad para cambiarlo; también se solicita la introducción de la unidad de disco en cuyo directorio activo se desea almacenar. Se preveen rutinas de detección de error en caso de protección contra escritura, compuerta de unidad de disco floppy abierta, disco lleno, medio de grabación (disco) dañado. En caso de detectarse algún error en el acceso a disco, la información correspondiente al programa (datos) sigue accesible al usuario en el buffer RAM-PC.

4 > BAJAR PROGRAMA A MEMORIA (RAM) DE INTERFACE SDM88-PC Y AUTOEJECUCION

Esta opción permite "bajar" un programa (*.SDM) previamente

cargado en el buffer RAM-PC, transfiriendo los bytes desde RAM-PC a memoria RAM de Interface SDM88-PC (a través del enlace de comunicación serie). Se transmiten primero las direcciones (Offsets) inicial y final del programa para µP 8088 y en seguida la cadena de bytes que lo constituye. Una vez que el programa es "bajado" y se encuentra residente en RAM de Interface, ésta se encarga de iniciar su ejecución (por cuenta del programa monitor residente en EPROM de la µC-Interface) = "Autoejecución de un programa en RAM de Interface". Para el proceso de Transferencia de bytes a memoria RAM de Interface SDM88-PC, se realiza una Asignación automática de Segmento (CS=0030h) (Dirección física base en el sistema = 00300h) para posicionar el programa a transferir en el área de RAM de la µC-Interface; los Offsets (IP) (inicial y final) dentro de dicho SEGmento son asignados indirectamente por el usuario, pues al crear el programa (*.SDM), se introduce su dirección (OFFSET) inicial y final. El usuario debe evitar para sus programas, el uso de Offsets entre (0000 y 00FFh) pues invadiria el Area de Servicio del Sistema en RAM (y se podria sobreescribir el código de su programa transferido a esta zona). El área adecuada para Programas de Usuario en RAM de µC-Interface SDM88-PC es pues a partir del Offset (0100h) y su extensión hacia arriba depende de la cantidad de memoria RAM instalada en la µC-Interface (p.ej. Offset hasta "14FFh" con 6 kBytes de RAM instalados o hasta "3CFFh" con 16 kBytes RAM instalados). Si no existe programa (*.SDM) en el buffer RAM-PC al momento de accionar este comando, se informa al usuario de la inexistencia de un programa que "bajar y autoejecutar" y el accionamiento del comando queda sin efecto.

5 > PASAR A MENU DE COMANDOS DE EDICION

Al seleccionar esta opción, aparece en pantalla un nuevo menú que contiene comandos que permiten visualizar y modificar el contenido del buffer RAM-PC donde se almacena código o datos tipo (*.SDM). Para entrar a este menú, debe existir algún programa (o

datos) en el buffer RAM-PC, pues de lo contrario se avisa al usuario de su inexistencia y se regresa al menú principal. Las opciones-comando con que cuenta este menú se describirán más adelante.

6 > PASAR A MENU DE COMANDOS PARA MANEJO DE {MEMORIA} Y {PUERTOS I/O} DE INTERFACE SDM88-PC

Al seleccionar esta opción, aparece en pantalla un nuevo menú que contiene comandos que permiten al usuario interactuar directamente con el Hardware de la μC-Interface desde la μC-PC(PS). Para la ejecución de las opciones-comando de este menú se requiere que la μC-Interface esté conectada a la μC-PC(PS) y el canal de comunicación serie entre ambas, esté correctamente establecido (misma velocidad de Transmisión fijada en PC y en Interface, Interface en modo de espera para recepción de comando, etc.). Las opciones-comando con que cuenta este menú se describirán más adelante.

7 > DEFINIR BAUDAJE :

{PC_Puerto Serie RS-232C} ⟷ {INTERFACE SDM88-PC_USART 8251A}

Mediante esta opción se puede cambiar el Baudaje (Velocidad de Transmisión/Recepción) actual entre μC-PC(PS) y μC-Interface. Los puertos Serie de ambas μCs, operan en modo Asíncrono y bajo las especificaciones de la norma RS-232C. En la pantalla de μC-PC se muestra la razón de Baudaje actual y se despliega un menú de opciones de Baudaje posibles :

1 > 300 [Bauds]
2 > 1200 [Bauds]
3 > 2400 [Bauds]
4 > 4800 [Bauds]
5 > 9600 [Bauds]
Q > REGRESAR Menú : COMANDOS ⟶ CONTROL DE INTERFACE

Al seleccionar cualquiera de estas opciones, tanto la μC-PC(PS) como la μC-Interface quedan (re)programadas para Tx/Rx a

esa velocidad. Para una correcta comunicación entre la µC-PC(PS) y la µC-Interface, ambas deben operar al mismo Baudaje. Inicialmente y después de un RESET de Hardware a Interface, ésta queda funcionando a 300 Bauds; debe entonces compatibilizarse a µC-PC(PS), pues de lo contrario, cuando se intente usar un comando que involucre comunicación serie PC-Interface, aparecerá el aviso de advertencia que informa al usuario sobre un error en la comunicación entre ambas microcomputadoras.

8 > PASAR A MENU DE COMANDOS PARA MANEJO DE DISCO EN PC

Al seleccionar esta opción, aparece en pantalla un nuevo menú que contiene comandos que permiten al usuario analizar y cambiar el ambiente de trabajo en lo que respecta a unidades de disco en la µC-PC(PS). Las opciones-comando con que cuenta este menú se describirán más adelante.

Q > TERMINAR LA SESION ... REGRESAR A * DOS *

Al accionar esta alternativa, se advierte al usuario que se pierde el contenido del buffer en memoria RAM-PC y se le da la oportunidad de regresar al ambiente SDM88-PC sin pérdidas ni cambios en las condiciones previas a la selección de esta opción. Si aún después de esta advertencia el usuario decide terminar la sesión de trabajo dentro del SDM88-PC, entonces se pasa a una rutina de "Término de Sesión" en la cual se "Cierra" el puerto serie de la µC-PC(PS)", se deshabilita el estado de Detección de Errores (Error Trapping), se reprograma la terminal de video a modo texto monocromático y se regresa el control al sistema operativo MS-DOS, reapareciendo el "prompt" correspondiente a la unidad de disco activa desde la cual se invocó anteriormente al programa "SDM88PCC.EXE" para entrar en el ambiente de operación del SDM88-PC.

A continuación se describe brevemente lo que acontece al seleccionar cada una de las opciones del :
MENU DE COMANDOS DE EDICION : para código (o datos) con formato {*.SDM} contenido en buffer RAM-PC . {Opción (5) Menú Principal}. En cada oportunidad se muestra el nombre del programa en código μP 8088 con el que se trabaja en RAM-PC (programa que se está depurando).

* MENU DE COMANDOS DE EDICION *

1 > LISTAR PROGRAMA

Permite desplegar sucesivamente (dirección {offset} por dirección) el contenido de un programa en lenguaje de máquina para μP 8088/86 .. (bytes {hex}) . Se tienen las siguientes opciones :
<T> : LISTAR TODO EL PROGRAMA
<P> : LISTAR PARTE DEL PROGRAMA
(Se pide Dirección Inicial y Final a listar)
<Q> : CANCELAR LA OPERACION DE LISTADO
El usuario deberá presionar <Cualquier Tecla> para Listar sucesivamente (byte por byte) o bien <Q> para terminar el proceso de listado en cualquier instante.

2 > AÑADIR BYTES

Permite agregar bytes (instrucciones de μP en formato hex) al final de un programa residente en el buffer RAM-PC ... (APPEND). Se despliegan las Direcciones Inicial y Final actuales del programa y se solicita la "Nueva Dirección Final" (si no se conoce exactamente, puede ponerse una estimada, incluso más olgada). Este comando involucra "Toma de Bytes" de pantalla por lo que, la presión de tecla :

<R> : Repite la introducción de Byte(s) anterior(es)
<Q> : Termina el proceso de introducción de Bytes en cualquier momento, ajustando automáticamente la nueva dirección final al último byte introducido

3 > INSERTAR BYTES

Permite insertar bytes (instrucciones de μP en formato hex) en un programa residente en el buffer RAM-PC . Se solicita del usuario la Dirección Inicial de Inserción y la Dirección Final de Inserción, mismas que el programa de control verifica que estén comprendidas dentro del área delimitada por las direcciones inicial y final absolutas del programa (Offsets). Este comando involucra "Toma de Bytes" de pantalla por lo que la presión de las teclas <R> y <Q> durante el proceso de Inserción, tiene el mismo efecto que en el caso de la opción (2) de este menú.

4 > CAMBIAR BYTES

Permite cambiar uno o varios bytes (instrucciones de μP en formato hex) de un programa residente en el buffer RAM-PC . Se solicita del usuario la Dirección Inicial a Cambiar en contenido y la Dirección Final a Cambiar en contenido, mismas que el programa de control verifica que estén comprendidas dentro del área delimitada por las direcciones inicial y final absolutas del programa (Offsets). Este comando involucra "Toma de Bytes" de pantalla por lo que la presión de las teclas <R> y <Q> durante el proceso de Cambio de contenido, tiene el mismo efecto que en el caso de la opción (2) de este menú.

5 > BORRAR BYTES

Permite borrar uno o varios bytes (instrucciones de μP en formato hex) de un programa residente en el buffer RAM-PC . Se solicita del usuario la Dirección Inicial y Final del bloque cuyo contenido se va a borrar, mismas que el programa de control verifica que estén comprendidas dentro del área delimitada por las direcciones inicial y final absolutas del programa (Offsets). Se efectúa un corrimiento automático de bytes y direcciones si el área borrada está entre las direcciones (Offsets) inicial y final absolutas del programa. También puede borrarse la totalidad del programa del buffer de RAM-PC, si el usuario introduce sus

direcciones inicial y final absolutas.

6 > FIN DE EDICION

Al seleccionar esta alternativa el sistema pasa a un submenú que comprende las siguientes acciones :

<1> : PASAR a Menú : COMANDOS —→ EDICION (Regresar)

<2> : PASAR a Menú : COMANDOS —→ CONTROL DE INTERFACE

verificándose la operación elegida por el usuario.

.....

A continuación se describe brevemente lo que acontece al seleccionar cada una de las opciones del :
MENU DE COMANDOS PARA MANEJO DE {MEMORIA} Y {PUERTOS I/O} DE INTERFACE SDM88-PC . {Opción (6) Menú Principal}.

* MENU DE COMANDOS PARA MANEJO DE {MEMORIA} Y {PUERTOS I/O} DE INTERFACE SDM88-PC *

1 > EXAMINAR MEMORIA { EPROM y/o RAM } DE INTERFACE SDM88-PC

Permite desplegar sucesivamente en pantalla de la μC-PC(PS) el contenido de una o varias localidades de memoria ya sea EPROM o RAM de la μC-Interface SDM88-PC .
Primero se solicita al usuario que introduzca la Dirección Base fijada por el SEGmento (DS o ES) correspondiente al área de memoria que desea analizar (SEGmento = { #hex / 10h }) :

{ FE00 } —→ [Dirección física = FE000h] —→ EPROM

{ 0030 } —→ [Dirección física = 00300h] —→ RAM

Luego se debe introducir las Direcciones Inicial y Final (OFFSETs {hex} dentro del SEGmento fijado antes).
La presión de <Cualquier Tecla> permite Desplegar localidades de memoria sucesivamente y de <Q> termina el proceso de despliegue en cualquier momento.
Una vez que se ha terminado el proceso de análisis de memoria, el

programa pasa a una rutina que permite opcionalmente al usuario Almacenar en disco el bloque de memoria examinado o Regresar al Menú de Comandos para Manejo de Memoria y Puertos I/O. Si decide almacenar en disco el bloque analizado, puede hacerlo con las mismas direcciones (Offsets) inicial y final o asignando unas nuevas de acuerdo a sus necesidades (el programa checa que estén dentro del rango permisible 0000 a FFFFh). Si se presenta algún error en el proceso de acceso a disco (p.ej. protección contra escritura), se da oportunidad al usuario de reintentar (después de tomar la acción correctiva pertinente) o de cancelar la operación de acceso.

2 > CARGAR DATOS (Bytes) EN LA MEMORIA { RAM } DE INTERFACE SDM88-PC

Permite al usuario introducir Bytes en la memoria RAM de la Interface SDM88-PC. Para ello el programa asigna automáticamente la dirección base de SEGmento en (0030) (Dirección física base = 00300h) que corresponde al área de RAM en el sistema. Luego se solicita del usuario las direcciones (OFFSETs) inicial y final (dentro de ese SEGmento fijado) y se introducen uno a uno los bytes conforme aparecen en pantalla los Offsets correspondientes. Como este comando involucra "Toma de Bytes" de pantalla, la presión de la tecla :

<R> : Repite la introducción de Byte(s) anterior(es)

<Q> : Termina el proceso de introducción de Bytes en cualquier momento, ajustando automáticamente la nueva dirección (Offset) final al último byte introducido

Una vez terminado el proceso de introducción de bytes por parte del usuario, éste puede :

<C>: Iniciar la transmisión de bytes desde RAM de μC-PC(PS) a RAM de la μC-Interface SDM88-PC

ó <Q>: Cancelar la operación

Si se desea efectuar la operación, el bloque de bytes queda posicionado en las localidades de memoria indicadas dentro de RAM

de la Interface y el usuario puede <Revisar> y/o <Guardar en disco> el bloque de bytes transferido, pasando a la opción (1) de este menú (Examinar Memoria de Interface SDM88-PC).

3 > EJECUTAR UN PROGRAMA A PARTIR DE UNA DIRECCION DE MEMORIA DADA { RAM o EPROM } DE INTERFACE SDM88-PC

Permite al usuario "correr" un programa residente en EPROM o previamente cargado en RAM. Se requiere que el usuario introduzca la dirección base (SEGmento) y el OFFSET inicial del programa a ejecutar. La presión de la tecla <C> inicia la ejecución del programa y la presión de <Q> cancela la operación.

4 > TRANSFERIR BLOQUES DE BYTES EN MEMORIA { EPROM → RAM } o { RAM → RAM } DENTRO DE INTERFACE SDM88-PC

Permite al usuario "mover" bloques de bytes de una zona a otra dentro de la memoria de la µC-Interface. Se solicita del usuario la dirección base (SEGmento) así como las direcciones OFFSET inicial y final del "bloque Fuente (Source)"; en seguida se asigna automáticamente la dirección base (SEGmento) del "bloque Destino (Destination)" como {0030h}, que corresponde a la dirección física 00300h perteneciente a RAM de la Interface, y se pregunta por la dirección (OFFSET) inicial de dicho bloque Destino; el OFFSET final se calcula automáticamente y se acepta como válido si resulta menor o igual a {FFFFh}, pidiéndose al usuario, en caso contrario, que Reintente con nuevas direcciones (OFFSET) o Cancele la operación. Si todas las pruebas de validez son satisfechas, y el usuario decide efectuar la transferencia de bloques (ya que también puede cancelarla), el bloque fuente es "copiado" en la dirección destino, y el resultado de dicha operación puede <Revisarse> y/o <Guardarse en disco> pasando a la opción (1) de este menú (Examinar Memoria de Interface SDM88-PC).
Nota: se puede dar al bloque destino la misma dirección del bloque fuente y la información contenida en memoria no se altera ni se pierde.

5 > ACOMODAR BLOQUE(S) DE BYTES ALMACENADO(S) EN DISCO, → EN { RAM } DE INTERFACE SDM88-PC

Permite al usuario cargar en memoria RAM de µC-Interface, uno o varios bloques (programas (código) o datos) almacenados en disco con formato {*.SDM} en una de dos maneras:

1) Acomodando los bloques en sus direcciones originales (con las que fueron grabados en disco)
2) Acomodando los bloques a partir de una dirección (OFFSET) en forma SUCESIVA (asignación automática de direcciones).

Los bloques de bytes a cargar de disco (programas) deben existir dentro de los directorios activos en las distintas Unidades de disco (Disk drives) que se dieron de alta como "válidas" en la etapa de Configuración del área de trabajo. El programa requiere del usuario el número de bloques a cargar, y en base a él, empieza a solicitar de uno en uno los respectivos nombres con que están almacenados en disco y la unidad de disco en que se encuentran. Después se propone al usuario iniciar el proceso de transferencia a RAM de Interface <C> o Cancelar <Q>. En caso de continuar con la operación, se asigna automáticamente la dirección de SEGmento destino como {0030h} (dirección física base = 00300h) que corresponde a RAM de trabajo y, en caso de haber elegido la opción de acomodo sucesivo (adyacente) de bloques, se pregunta por la dirección OFFSET inicial de colocación. En seguida se van cargando los bloques de disco a RAM-PC y de ahí se transfieren a RAM-Interface (a través del Puerto Serie). Este proceso de carga-transferencia va realizándose sobre un bloque a la vez, es decir, se interrumpe entre un bloque y otro, para que el usuario decida si quiere Continuar <C> o Abortar el proceso <Q>. Conforme se van cargando y transfiriendo los bloques, se despliegan en pantalla sus nombres y las direcciones {OFFSET} que van ocupando en RAM de µC-Interface. Una vez concluida la carga-transferencia de todos los bloques deseados, el usuario puede <Revisar> y/o <Guardar en disco> el nuevo "Macrobloque" pasando a la opción (1) de este menú (Examinar Memoria de Interface SDM88-PC).

6 > PASAR A MENU DE COMANDOS PARA MANEJO DE ARCHIVOS BINARIOS CONTENIDOS EN DISCO

Al seleccionar esta opción aparece en pantalla un submenú que contiene comandos que posibilitan al usuario para CARGAR y LISTAR (visualizar el contenido de) cualquier tipo de archivos binarios generados desde el sistema operativo MS-DOS de la μC-PC (PS) mediante algún programa de aplicación (p.ej. Macro Ensamblador, Grabador de EPROMs, etc.). Los comandos de este menú hacen que el sistema SDM88-PC no sea una aplicación cerrada que sólo acepta programas generados bajo su propio formato {*.SDM}, sino que, a través de dichos comandos, pueda tenerse acceso a la creciente librería de programas auxiliares para desarrollo de software (en lenguaje de μP 80xxx) de que se dispone en el mercado (principalmente Ensambladores que, a partir de programas en lenguaje de máquina introducidos en forma de mnemónicos con un editor de texto ASCII, generan archivos objeto (*.OBJ), que después de Ligarse (LINK) y producir un archivo ejecutable (*.EXE), permiten obtener archivos binarios puros (*.COM) que contienen la información "neta" del código de μP que el usuario deseaba programar (de otra manera el usuario tendría que generar el código hexadecimal a través de tablas de instrucciones de μP, tarea laboriosa y susceptible de error)) .

... Los comandos de este menú son los siguientes : .....

1 > CARGAR ARCHIVO BINARIO { *.COM, *.BIN, *.EXE, *.ROM, etc. } A MEMORIA (BUFFER) RAM-PC

Permite cargar un archivo binario cualquiera contenido en disco a un Buffer especial en RAM de la μC-PC (PS) (con 640 kBytes RAM mínimo). El usuario debe introducir el nombre del archivo, incluyendo su extensión, según las convenciones típicas del formato del sistema operativo MS-DOS :

[d:][dir_path] filename.ext

opcional: [d:]=drive ; [dir_path]=trayectoria_dir

obligatorio: filename.ext = <nombre>.<extensión>

Ejemplos:

Cargar archivo binario "PROGX.COM" a Buffer RAM-PC

PROGX.COM

A:PROGX.COM

C:\SDMPRG\PRGBINX\PROGX.COM

En el proceso de búsqueda del archivo en disco se provee detección de errores (Archivo no encontrado, Dir_path no encontrado o Archivo no existe, Acceso a disco denegado, etc.).

En caso ser exitosa la lectura del archivo de disco se reporta en la pantalla de la PC, el Tamaño del archivo y el # de bytes leídos a RAM-PC (cantidades que deben ser iguales a menos que se exceda el tamaño actual del buffer RAM-PC reservado para este fin {4608=1200h bytes}(más que suficiente para el tamaño de los archivos binarios de código µP neto en la mayoría de las aplicaciones)). También se actualiza en pantalla, el despliegue del nombre del archivo binario residente en buffer RAM-PC (con todo y Drive, DIR_path y extensión, tal como lo tecleó el usuario). Al cargarse un archivo binario al Buffer RAM-PC, se le asigna por default un offset inicial de 0000h (mismo que después puede modificarse con la opción (3) de este menú).

2 > LISTAR ARCHIVO BINARIO CARGADO EN MEMORIA RAM-PC

Permite desplegar el contenido de un archivo binario cualquiera (en particular del cargado en el Buffer RAM-PC). Se muestra en pantalla la dirección (Offset) inicial y final del archivo cargado y se solicita del usuario la dirección (Offset) inicial y final a desplegar. La presentación del despliegue es en grupos de 16=10h bytes por renglón, indicando la dirección (Offset) base del primer byte en cada renglón mostrado.

3 > ASIGNAR (AJUSTAR) NUEVAS DIRECCIONES (OFFSETs) AL ARCHIVO BINARIO CARGADO EN MEMORIA RAM-PC

Permite la Relocalización de un Archivo binario (programa), es decir, la asignación de nuevas direcciones (Offsets) inicial y final. Es mediante este comando que el usuario asigna el Offset que desea tenga el programa, dentro de la memoria RAM de la µC-Interface (SDM88-PC), al momento de transferirlo (opción (4) de este menú).

El OFFSET asignado será medido a partir de la dirección física base 00300h (CS=0030h) que corresponde a RAM de la Interface, por lo que es necesario que dicho OFFSET sea mayor o igual a 0100h, para no cargar un programa de usuario en área reservada para servicio de sistema: 0000-00FF.

4 > CARGAR ARCHIVO BINARIO (CON DIRECCIONES ASIGNADAS) A MEMORIA RAM DE INTERFACE (SDM88-PC)

Permite la transferencia del bloque de bytes que componen el archivo binario contenido en el Buffer RAM-PC hacia la memoria RAM de la µC-Interface, a través del puerto serie de la µC-PC(PS). Para el proceso de transferencia, se hace una asignación automática de la dirección base de SEGmento = (0030h) (Dirección física=00300h) que corresponde al área de RAM de la Interface. Las direcciones de OFFSET inicial y final son las asignadas con la opción (3) de este menú. Se despliegan también las direcciones físicas que tendrá el programa en RAM de Interface, para que el usuario aprecie si el programa será cargado en memoria disponible físicamente (instalada). Entonces el usuario puede Iniciar la Transmisión de bytes a RAM de µC-Interface, <C>, o Cancelar la operación, <Q>, para asignar nuevas direcciones (Offsets), por

ejemplo. En caso de continuar con la operación, el archivo binario es posicionado en las direcciones indicadas (Offsets a partir de la dirección física base "00300h") en RAM de la μC-Interface (SDM88-PC).

Una vez en RAM de Interface, el usuario podrá :

A) <Revisar> y/o <Guardar en disco (con formato *.SDM) >, el bloque de bytes transferido, pasando a la opción para {Examinar Memoria de Interface SDM88-PC} ;

B) <Ejecutar> (correr) el archivo binario transferido, pasando a la opción para {Ejecutar Programa a partir de una dirección de memoria (SDM88-PC) dada} ;

C) <Transferir> el archivo binario a otra área de memoria (SDM88-PC), pasando a la opción para {Transferencia de bloques de bytes en memoria de Interface (SDM88-PC) } .

5 > REGRESAR Menú : COMANDOS —→ MANEJO DE MEMORIA Y PUERTOS I/O { INTERFACE SDM88-PC }

.....

7 > LECTURA / ESCRITURA —→ { PUERTOS I/O } DE INTERFACE SDM88-PC

Al seleccionar esta opción aparece en pantalla un submenú que posibilita al usuario para realizar una de las siguientes operaciones:

1 > LECTURA DE PUERTO (INPUT)

Este comando permite al usuario leer (accesar) la información digital presente en un puerto de Entrada del sistema, en un momento dado (manualmente).

El programa pide al usuario la dirección del puerto a accesar (0000h a FFFFh), para posteriormente, si se decide continuar con la operación, desplegar en

pantalla de la μC-PC(PS), la información (byte) leída de dicho puerto.

Este comando es útil, entre otras cosas, para verificar el contenido de los registros de configuración y status de operación de los distintos puertos (periféricos ICs) del sistema, así como para revisar la correcta conexión y funcionamiento de los diversos puertos (para Entrada de datos) que se "den de alta" en una aplicación "satélite" del sistema **SDM88-PC**.

2 > ESCRITURA EN PUERTO (OUTPUT)

Este comando permite al usuario escribir información digital (byte en paralelo por el Bus de Datos) en algún puerto de Salida del sistema, en un momento dado (manualmente).

El programa pide al usuario la dirección del puerto a accesar (0000h a FFFFh) y la información digital (byte) que se desea escribir en él, para posteriormente, si se decide continuar con la operación, escribir efectivamente la información digital (byte) en dicha dirección de puerto.

Este comando es útil, entre otras cosas, para alterar o reprogramar "manualmente" la operación de los distintos puertos (periféricos ICs) del sistema, a través de sus registros de configuración, así como para revisar la correcta conexión y funcionamiento de los diversos puertos (para Salida de datos) que se "den de alta" en una aplicación "satélite" del sistema SDM88-PC.

Q > REGRESAR Menú : COMANDOS —→ MANEJO DE MEMORIA Y PUERTOS I/O

8 > REGRESAR Menú : COMANDOS —→ CONTROL DE INTERFACE (SDM88-PC)

Al seleccionar esta opción, se regresa al Menú principal para operación del sistema **SDM88-PC**.

A continuación se describe brevemente lo que acontece al seleccionar cada una de las opciones del :
MENU DE COMANDOS PARA MANEJO DE DISCO EN PC
{Opción (8) Menú Principal}.

* MENU DE COMANDOS PARA MANEJO DE DISCO EN μC PC (PS) *

1 > EXAMINAR DIRECTORIO ACTIVO EN ALGUNA UNIDAD DE DISCO

Permite desplegar en pantalla de la μC-PC(PS) el conjunto de archivos contenidos dentro del directorio activo en la unidad de disco (válida) especificada por el usuario. Dentro de este comando, existen a su vez, 3 opciones :

1 > MOSTRAR TODO EL DIRECTORIO ACTIVO EN LA UNIDAD DE DISCO "X"
X={A,B,C o D} . Se despliegan todos los archivos contenidos en dicho directorio sin importar su nombre ni su extensión.

2 > MOSTRAR SOLO PROGRAMAS <*.SDM> DENTRO DEL DIRECTORIO ACTIVO
Se despliegan todos los archivos cuya extensión sea "SDM" (formato especial para archivos creados por el sistema SDM88-PC), sin importar su nombre.

3 > MOSTRAR PROGRAMAS {????????.SDM} DENTRO DEL DIRECTORIO ACTIVO
(P.ej.: Si se teclea < XY* > ó < XY?????? > como máscara de identificación, se mostrarán todos los programas empezados en : XY------.SDM)

2 > CAMBIAR DIRECTORIO ACTIVO (DIR_PATH) _ { DIRECTORIO DE TRABAJO } EN ALGUNA UNIDAD DE DISCO

Se despliega en pantalla de la μC-PC(PS) el Directorio activo actual y se inquiere al usuario sobre si desea cambiarlo. En caso afirmativo, se solicita la introducción de la nueva trayectoria-directorio (DIR_Path) que se desea activar en la unidad de disco especificada. En caso de que la nueva

trayectoria-directorio no exista en la unidad de disco, se avisa al usuario mediante el mensaje de error correspondiente.

3 > BORRAR PROGRAMA {*.SDM} DENTRO DEL DIRECTORIO ACTIVO EN ALGUNA UNIDAD DE DISCO

Permite eliminar un programa {*.SDM}. Basta con teclear su nombre, pues la extensión "SDM" se presupone. Si el usuario decide continuar con la operación, el programa es borrado del directorio activo de la unidad de disco que especificó. En caso de no existir el programa cuyo nombre se introdujo o estar protegido el disco contra escritura o no estar preparada la unidad de disco floppy (A o B), se despliega el mensaje de error correspondiente, y el accionamiento de este comando queda sin efecto.

4 > REGRESAR Menú : COMANDOS → CONTROL DE INTERFACE {SDM88-PC}

Al seleccionar esta opción, se regresa al Menú principal para operación del sistema SDM88-PC.

.....

Cabe señalar que, al entrar a todos los comandos y en los puntos críticos en la ejecución de algunos de ellos, el usuario podrá <C> Continuar o <Q> Cancelar (Abortar) la operación. Este mecanismo es útil para salir de algún comando accesado accidentalmente, o cuando se ha cometido algún error en la introducción de las entradas solicitadas por algún comando y que produciría efectos indeseados por resultado, o simplemente cuando se cambie de opinión sobre el accionamiento de un comando.

También existen mensajes de Advertencia que informan al usuario sobre las consecuencias de la acción que está a punto de realizar (p.ej. pérdida del contenido actual de buffer RAM-PC al usar ciertos comandos o teclas bajo determinadas circunstancias), recomendándose a la vez, alguna acción preventiva previa.

Finalmente, se cuenta con un poderoso mecanismo para DETECCION DE ERRORES durante la operación del sistema SDM88-PC, tales como :

File not found : (Directorios)
" No existen programas <*.SDM> en el disco o en el directorio activo de la Unidad : {X} "

File not found : (Apertura (carga) o borrado de archivos)
" El programa < {Y}.SDM > no existe en el disco o en el directorio activo de la Unidad : {X} "

Disk not ready (diskette drive door open or diskette not in drive)
" La unidad de disco floppy {X} no está preparada "

Disk full :
" La capacidad del disco está agotada (disco lleno) "

Permission denied (diskette write protected) :
" El disco en la Unidad {X} está protegido contra escritura "

Disk media error (diskette controller detects hardware or media fault) :
" La superficie de grabación del disco en la Unidad {X} está dañada "

Bad file number &
Bad filename (too many characters) :
" Se asignó al programa un nombre ilegal (más de 8 caracteres) "

Path not found (MS-DOS unable to find path specified, operation not completed) :
" No se encontró el { Dir_Path } especificado ... (operación no completada) "

Path/File access error (MS-DOS unable to make correct path-to-filename connection, operation not completed) :
" No se pudo establecer la conexión correcta { Path-to-Filename } (operación no completada)

Subscript out of range (out of array's dimensions) :
" Se sobrepasó la dimensión reservada para Buffer de Bytes en PC {4608=1200h} , (operación no completada) ... Se almacenó hasta el byte # 4608 "

Error en la rutina de Recepción de Bytes en la μC-PC :
(Se recibió caracter NULL o se terminó el plazo de espera para recepción de caracter (actualmente fijado en 9 [s]))

Device I/O error &
Device unavailable or disabled :
" Acción correctiva :
RESET (Hardware) ⟶ Microcomputadora PC y/o Interface "

CAPITULO IV

ENLACE DEL SISTEMA SDM88-PC CON HERRAMIENTAS COMERCIALES PARA DESARROLLO DE PROGRAMAS EN LENGUAJE ENSAMBLADOR DE MICROPROCESADORES INTEL 80XXX (8088/86)

INTRODUCCION

Para la creación de programas en lenguaje de Máquina (lenguaje Ensamblador), a menos que sean muy simples y cortos, es conveniente emplear herramientas para desarrollo de Software (programas), a fin de hacer el trabajo más fácil (uso de mnemónicos en vez de código hexadecimal para las instrucciones de μP), más ordenado (documentado) y minimizando la cantidad de errores en la generación e introducción de código (hex) para el Microprocesador.

El sistema **SDM88-PC** cuenta con varios comandos que permiten al usuario operar sobre cualquier tipo de archivos binarios generados desde el sistema operativo MS-DOS de la μC-PC (PS) a través de alguno de estos programas auxiliares para desarrollo de Software (p.ej. Macro_Ensamblador, Debugger, Grabador de EPROMs, etc.). Con el **SDM88-PC**, pueden realizarse básicamente, operaciones de CARGA (de disco a RAM de μP-PC y de RAM de μP-PC a RAM de μC-Interface) y LISTADO (visualización de contenido) sobre los archivos binarios inicialmente residentes en disco (floppy o hard). Una vez que el contenido (bytes) de dichos archivos binarios está en RAM de μC-Interface SDM88-PC, podrán éstos ejecutarse, desplegarse en pantalla de PC, transferirse por bloques dentro de memoria, etc.

Es pues, gracias a estas características, que el sistema **SDM88-PC** "NO" es una aplicación cerrada que sólo acepta programas generados bajo su propio formato (*.SDM), sino que, a través de sus comandos especializados, puede tenerse acceso a la extensa librería de programas auxiliares para desarrollo de software (en lenguaje de μP 80xxx) disponible en el mercado, principalmente Ensambladores y archivos de soporte que, a partir de programas fuente en lenguaje de máquina introducidos en forma de mnemónicos

con un editor de texto ASCII, generan archivos objeto (*.OBJ), que después de Ligarse (LINK) y producir un archivo ejecutable (*.EXE), permiten obtener archivos binarios puros (*.COM), que contienen la información "neta" del código de μP que el usuario necesita programar, evitando así, la laboriosa tarea de generación del código hexadecimal de μP a través de tablas de instrucciones, y la introducción de dicho código a la memoria del sistema, proceso tardado y susceptible de error humano.

En este capítulo se describen brevemente varias herramientas comunes involucradas en el desarrollo de programas en lenguaje Ensamblador, que pueden usarse, en o para, un sistema basado en μP (Microcomputadora); la mayoría de estas herramientas son programas que son ejecutados (corridos) para realizar alguna función en el programa que el usuario está escribiendo. En este caso, los programas auxiliares de desarrollo están residentes y son usados dentro de la μC-PC (PS) para elaborar programas destinados a operar sobre el Hardware de la μC-Interface, en el sistema **SDM88-PC**. Bajo este esquema, el control lógico de algún proceso o de algún instrumento de aplicación a partir del **SDM88-PC**, se realiza finalmente gracias a programas (*Software*) creados por el usuario en la PC (PS) . Posteriormente (dentro del ambiente del **SDM88-PC**), los programas en código hex del μP 8088, son transferidos (cargados) a RAM de la Microcomputadora-Interface, para actuar sobre el *Hardware* de la aplicación a través del *Hardware* de la Interface (con o sin la intervención de la PC, según se requiera).

También se mostrará el Algoritmo o Ciclo de Desarrollo de un Programa, de manera general, y luego de manera particular para un sistema como el **SDM88-PC** basado en una μC-PC (PS) y en su característica μC-Interface.

Asimismo, se establecerán las diferencias entre un archivo binario en formato (*.EXE) y en formato (*.COM), en el ambiente de la μC-PC (PS) bajo el sistema operativo MS-DOS.

Finalmente, se incluyen tres ejemplos de programas-tipo para operar sobre el Hardware de la μC-Interface y se indica el proceso de generación del archivo binario adecuado, desde la μC-PC (PS), empleando productos del dominio público elaborados por "Microsoft".

4.1
HERRAMIENTAS PARA DESARROLLO DE PROGRAMAS EN LENGUAJE ENSAMBLADOR
(Disponibles en el ambiente de la μC-PC (PS))

EDITOR (Editor)

Un Editor es un programa que, cuando es ejecutado en un sistema, permite al usuario teclear (introducir) las instrucciones en lenguaje Ensamblador para su programa. Por ejemplo, ALTER que funciona en sistemas de Intel, EDLIN, WORDSTAR, ChiWriter, etc., que corren en PC's IBM y compatibles, etc.. La principal función de un Editor es asistir al usuario en la construcción de su programa en lenguaje Ensamblador en el formato adecuado, a fin de que el programa Ensamblador pueda traducirlo correctamente a lenguaje de máquina (código binario de μP). Esta forma del programa de usuario es llamada "Programa Fuente" (*Source Program*).

Conforme el usuario va tecleando su programa, el Editor almacena los códigos ASCII correspondientes a letras, números y símbolos en localidades sucesivas de RAM. Si el usuario comete un error tipográfico, el Editor le permitirá regresar y corregirlo, o si omite una instrucción del programa, le permitirá recorrer todo lo tecleado hacia abajo e insertar la línea.

Cuando se ha terminado de teclear todo el programa, deberá copiarse desde memoria a un archivo en disco magnético floppy o hard. Este archivo se denomina "Archivo Fuente" (*Source File*). Si posteriormente se encuentra que el programa tiene errores, puede usarse el Editor para cargar el Archivo Fuente de nuevo en RAM, hacer las correcciones necesarias en el Programa Fuente y salvar en disco dicho programa corregido.

ENSAMBLADOR (Assembler)

Un programa Ensamblador se usa para traducir mnemónicos en lenguaje Ensamblador al código binario correcto correspondiente a cada instrucción. El programa Ensamblador leerá el Archivo Fuente del usuario, del disco donde fue salvado después de editarlo. El p. Ensamblador generalmente lee el archivo fuente más de una vez. En la PRIMERA PASADA (first pass) a través del programa fuente, el p. Ensamblador localiza todo : determina el desplazamiento de variables o constantes de datos referidas por nombre y los "offsets" de Etiquetas (labels) para saltos (jumps), poniendo toda esta información en una Tabla de Símbolos (Symbol Table). En una SEGUNDA PASADA (second pass), el p. Ensamblador genera el código binario para cada instrucción y asigna direcciones a cada una.

El p. Ensamblador genera dos archivos en el disco floppy o hard :

* Archivo Objeto (Object file)
  Contiene los códigos binarios para las instrucciones y la información acerca de sus direcciones respectivas; esta información será eventualmente cargada en memoria y ejecutada.
* Archivo de Listado Ensamblador (Assembler List file)
  Contiene las instrucciones en lenguaje Ensamblador, los códigos binarios y el "offset" de cada instrucción. Generalmente este archivo se manda a impresora para tener una copia en papel, útil en el proceso de prueba y depuración del programa (ya que el listado indica cualquier error de sintaxis o formato). En el listado de un p. Ensamblador para 8088/86, la columna extrema izquierda da el offset de las variables o constantes de datos con respecto al inicio del Segmento de Datos, y los offsets de los bytes de código a partir del inicio del Segmento de Código. Notar que el p. Ensamblador NO genera direcciones físicas absolutas (un Ligador (Linker) o Localizador (Locator) hará esto después). La sección final del listado, da información adicional a cerca de los segmentos

y nombres (etiquetas) usados en el programa (Estadísticas).

LIGADOR (Linker)

El Ligador es un programa usado para reunir varios archivos objeto en un único "gran archivo objeto". Cuando se escriben programas extensos, es generalmente más eficiente dividir el programa total en módulos más pequeños. Cada módulo puede ser escrito individualmente, probado y depurado. Cuando todos los módulos trabajan satisfactoriamente, pueden ser ligados para formar un único programa funcional. Además, los módulos objeto de rutinas o programas útiles, pueden ser mantenidos en un "Archivo Librería" (Library file) y posteriormente ligarse con otros programas según se necesite. Cuando se crea un programa global compuesto por varios archivos objeto, las llamadas a subrutinas del archivo Librería, deben ser declaradas como "externas" en los archivos fuente constitutivos (Declaring external symbols).

El ligador produce un "Archivo ligado" (Link file) que contiene los códigos binarios de todos los módulos combinados; también produce un "Archivo de Mapa de Ligado" (Link Map file) que contiene la información de direcciones de los archivos ligados. Sin embargo, el ligador NO asigna direcciones absolutas al programa, sólo asigna direcciones relativas empezando desde cero. Este formato de programa se denomina "Relocalizable" (Relocatable), porque puede ser puesto en cualquier parte de la memoria para ser corrido. Si el programa va a ser ejecutado en un sistema como la μC IBM PC (PS), basta tan sólo con cargar el archivo ligado en memoria y correrlo (*.EXE). Si, en cambio, el programa va a ser corrido en un sistema como el "Intel Series IV" o como el SDM88-PC, entonces debe usarse previamente un "Programa Localizador" (Locator), para asignar direcciones absolutas al Archivo Ligado.

LOCALIZADOR (Locator)

Un Localizador es un programa usado para asignar las direcciones especificas en donde el código objeto será cargado en memoria. Un programa localizador que viene con el Sistema Operativo de disco (DOS) para IBM-PC, es llamado "EXE2BIN".

El procedimiento para producir un programa con direcciones absolutas que pueda ser cargado (bajado) a la µC-Interface **SDM88-PC** desde una µC-PC (PS) (IBM) es el siguiente :

1) Crear un archivo fuente (Source) {*.ASM} usando algún Editor (EDLIN, WORDSTAR, SEE, Turbo-C, etc.).

2) Ensamblar el archivo fuente con el MacroEnsamblador (MASM) de IBM o Microsoft para µC-PC (PS), a fin de producir un archivo Objeto {*.OBJ}.

3) Usar el programa "LINK" para producir un archivo relocalizable {*.EXE}

4) Usar el programa "EXE2BIN" (que da al programa una dirección absoluta de inicio como 0000:0100 h). En realidad, lo importante aquí es crear un archivo de disco que contenga el código "neto" de µP (bytes) (sin encabezados *headers* que sólo son significativos para el ambiente PC bajo MS-DOS).

5) Usar los recursos del programa de control del sistema **SDM88-PC** : { SDM88PCC.EXE } , para cargar de disco a RAM-PC el archivo binario producido por EXE2BIN, Listarlo, Asignarle Direcciones-Offset apropiadas para RAM de µC-Interface, y finalmente, cargarlo (bajarlo) (a través del puerto serie) desde RAM-PC a RAM de µC-Interface SDM88-PC , desde donde puede ser ejecutado (corrido). Concretamente, se usan los comandos del "Menú de Comandos para Manejo de Archivos Binarios contenidos en disco" y "Menú de Comandos para Manejo de Memoria y Puertos I/O de µC-Interface, SDM88-PC ". +

DEPURADOR (Debugger)

Si el programa del usuario no requiere hardware externo o requiere sólo hardware directamente accesible desde su sistema, entonces se puede usar un Depurador ("Debugger") para correr y depurar dicho programa a fin de identificar errores en la lógica del mismo.

Un Debugger es un programa que permite al usuario cargar su programa en código objeto en la memoria del sistema, ejecutar dicho programa, y depurarlo ("troubleshoot" o "debug").
El Debugger permite :

+ examinar el contenido de los registros de μP y localidades de memoria después de que el programa es ejecutado
+ cambiar el contenido de registros y localidades de memoria y reejecutar el programa
+ detener la ejecución después de cada instrucción a fin de poder revisar y/o alterar el contenido de registros o memoria
+ fijar puntos de ruptura "breakpoints" en cualquier punto del programa; cuando se corre el programa, el sistema ejecutará instrucciones hasta ese "breakpoint" y se detendrá; el usuario podrá entonces examinar el contenido de los registros y memoria para ver si los resultados son correctos en ese punto; si son correctos, podrá mover el "breakpoint" a una zona posterior dentro del programa; si NO son correctos, podrá revisar el programa hasta ese punto a fin de encontrar porqué no lo son.

La herramientas del "Debugger" pueden ayudar al usuario a "aislar" un problema en su programa. Cuando se encuentra el problema, se podrá corregir el algoritmo si es necesario. El usuario empleará el Editor para corregir su programa Fuente, reensamblar el programa corregido, religarlo y correrlo de nuevo.

Debuggers diseñados para la µC-PC (PS) IBM, por ejemplo CODEVIEW de Microsoft, permiten realizar sobre los programas todas las funciones descritas antes (examen, modificación, "single-step", etc.) además de una función de rastreo permanente "Trace" que muestra el contenido de todos los registros del µP y el status (estado) de banderas (flags), después de la ejecución de cada instrucción.

Un Debugger como el CODEVIEW de Microsoft, puede usarse como un auxiliar indirecto, en el desarrollo y depuración de programas destinados a ejecutarse en la µC-Interface del sistema SDM88-PC; estando en el ambiente MS-DOS de la µC-PC (PS), basta cargar con el CODEVIEW el programa fuente {*.ASM} junto con el programa ejecutable {*.EXE} preparado con información extra para el debugger, o el {*.COM} si se desea depurar en "modo ensamblador", y correr el programa paso a paso, utilizando todos los recursos (opciones y comandos) que ofrece esta herramienta.

Con respecto al CODEVIEW de Microsoft, hay que señalar que, a fin de que el debugger pueda desplegar información simbólica, el programa fuente debe ensamblarse con la opción {/ZI} y ligarse con la opción {/CO}; para otras situaciones pueden requerirse opciones adicionales. [Ver : *Microsoft CODEVIEW and Utilities Manual*]. Resumiendo : cuando se planea depurar un programa con MS-CODEVIEW, el archivo fuente {*.ASM} debe ser :

1) Ensamblado con la opción {/ZI} para escribir información sobre números de línea y datos simbólicos en el archivo objeto.

2) Ligado con la opción {/CO} para preparar un archivo ejecutable especial que contenga información sobre números de línea y datos simbólicos.

Toda la información necesaria para el debugger, está disponible en los archivos preparados en el formato {*.EXE}, en cambio, dicha información es eliminada, si el archivo es preparado en el formato

{*.COM}, pudiendo depurarse entonces sólo en "modo ensamblador", es decir, en lenguaje de máquina, sin despliegue de mnemónicos obtenidos del archivo fuente, al momento de estar corriendo el programa desde el debugger. El archivo {*.EXE} especial, puede correr fuera del ambiente del CODEVIEW, ya que la información extra en el archivo será ignorada. Sin embargo, para mantener el tamaño del archivo al mínimo, sólo debe usarse este formato *.EXE especial para depuración (debugging); cuando los errores lógicos sean identificados y corregidos, después del proceso de depuración, conviene crear una nueva versión del programa, ensamblándolo sin la opción {/ZI} y ligándolo sin la opción {/CO}, para obtener un archivo con "código puro" sin información simbólica.

El uso de un Debugger es la herramienta más eficaz para la identificación de errores lógicos en un programa en lenguaje Ensamblador. +

EMULADOR (Emulator)

Es otra opción que el usuario tiene para correr su programa. Un Emulador es una mezcla de Hardware y Software, que generalmente se usa para probar y depurar el Hardware y Software de un sistema externo, como por ejemplo, el prototipo de un instrumento basado en microprocesador. Como parte del hardware del Emulador está un cable (multi-alambre) para conectar el sistema anfitrión (host) con el sistema bajo prueba o desarrollo. Dicho cable cuenta con un conector terminal que se inserta en el instrumento bajo prueba en vez de su microprocesador. A través de esta conexión, el software del Emulador permite al usuario "bajar" su programa en código objeto (μP) a RAM del sistema bajo prueba y correrlo. Como en un Debugger, el Emulador permite cargar y correr programas, examinar y cambiar el contenido de registros del μP y de

localidades de memoria e insertar "breakpoints" en el programa. También realiza una función de rastreo (trace) permanente mostrando en pantalla de algún dispositivo auxiliar (ej. μC-IBM-PC), el contenido de los registros de μP, el estado de las banderas (flags) y la actividad en el bus de datos y direcciones conforme se ejecuta cada instrucción. El Emulador almacena esta información rastreada en una extensa área de RAM propia, a fin de que el usuario pueda, si así lo desea, generar un listado o impresión de dicha información para observar los resultados que produce su programa bajo un esquema de ejecución "paso a paso" ("single-step" o "step-by-step"). Otra poderosa característica de un Emulador, es la habilidad de poder usar tanto memoria del sistema (propia o p.ej. de una PC) como memoria contenida en el dispositivo bajo prueba o desarrollo, para almacenar el programa que se está depurando.

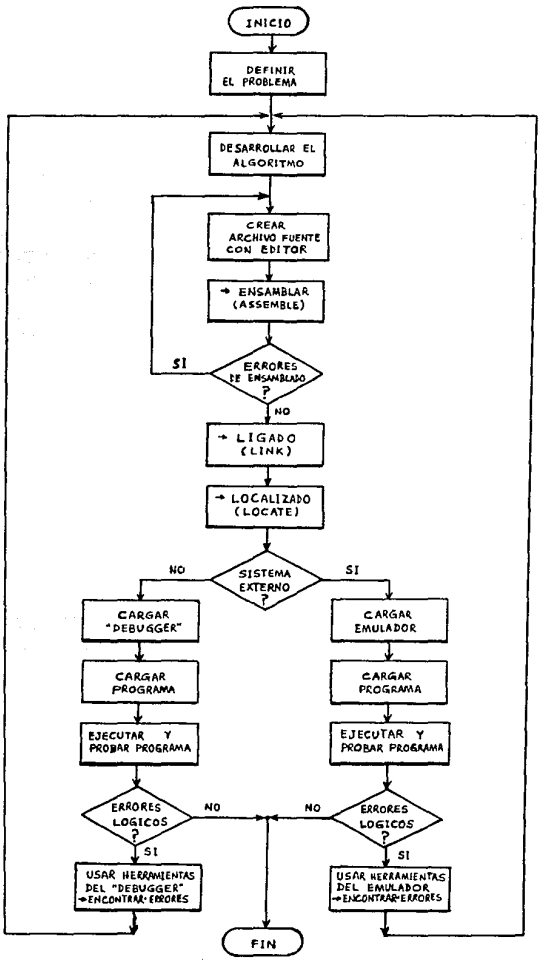
En la sección 5.2.2 del capítulo 5, se abundará sobre el empleo de un Emulador como alternativa para el desarrollo de un producto basado en microprocesador.

4.2
ALGORITMO O CICLO DE DESARROLLO PARA PROGRAMAS EN LENGUAJE ENSAMBLADOR
(Ambiente μC-PC (PS))

En la { *figura* 4-1 } se muestra en forma diagramática, el orden en que se emplean las distintas herramientas para el Desarrollo de Programas en Lenguaje Ensamblador, anteriormente descritas.

{ *figura* 4-1 }

Ciclo de Desarrollo para Programas en Lenguaje Ensamblador



1 > La primera y más importante etapa consiste en pensar (diseñar) muy cuidadosamente, "qué" se quiere que haga el programa y "cómo" se desea que lo haga : "Definición del problema".

2 > Usar un Editor para crear el archivo fuente del programa.

3 > Ensamblar el archivo fuente con un programa Ensamblador (Assembler). Si el archivo de listado Ensamblador (Assemble-list file) indica cualquier error en el programa, usar el Editor para corregir dichos errores. Permanecer en el ciclo de Edición-Ensamblado hasta que el programa Ensamblador no reporte errores en el listado.

4 > Si el programa consta de varios módulos, entonces usar un programa Ligador (Linker) para reunir sus respectivos módulos objeto en un único módulo objeto global. En algunos sistemas como la µC-IBM-PC (PS), debe usarse el "Linker" aún si el programa consta de un solo módulo.

5 > Si el programa lo requiere, usar un programa Localizador (Locator), para especificar en qué parte de la memoria se quiere poner (cargar) el programa.

El programa está ahora listo para ser cargado en memoria y ejecutado.

6_A > Si el programa NO interactúa con con ningún Hardware externo además del conectado o perteneciente directamente al sistema, entonces usar un programa Depurador (Debugger) para correr e identificar los errores lógicos que el programa bajo prueba pueda tener.

6_B > Si el programa está destinado a operar sobre un Hardware externo, como el prototipo de un instrumento basado en

microprocesador, entonces probablemente pueda usarse un sistema Emulador (del μP) para ejecutar y depurar el programa bajo análisis.

A continuación se describe el procedimiento para desarrollar un programa autónomo (stand-alone) en lenguaje Ensamblador, específicamente dentro del ambiente de la μC-IBM PC (PS), (por ejemplo, usando los productos de Microsoft), indicando también la situación (punto de enlace) del sistema de Desarollo SDM88-PC dentro del contexto de este :

CICLO DE DESARROLLO PARA PROGRAMAS EN LENGUAJE ENSAMBLADOR

1) Usar un Editor de texto para crear o modificar módulos fuente en lenguaje Ensamblador. Los programas en lenguaje Ensamblador son creados a partir de uno o más archivos fuente (Source files) : (archivos de texto que contienen sentencias que definen los datos e instrucciones del programa). El Editor usado debe ser capaz de producir archivos ASCII (American Standard Code for Information Interchange); las líneas (sentencias) deben estar separadas por una combinación de CR-LF (*Carriage_Return - Line_Feed*); si el Editor de texto usado tiene un modo de programación o No-documento (nondocument) para producir archivos ASCII, usar ese modo.
Por convención, a los módulos fuente se les da la extensión { .ASM } .
Los módulos fuente pueden estar organizados en una variedad de formas : poner todas las rutinas de un programa en un único gran módulo, o dividir las rutinas entre módulos.
(Nota: si el programa va a ser ligado con otros módulos en lenguaje de alto nivel, el código fuente para estos módulos debe también prepararse en este punto). (No es el caso).

2) Usar el Macro-Ensamblador MASM para ensamblar cada uno de los módulos del programa. MASM puede leer opcionalmente código de archivos anexos (include files) durante el ensamblado. Si se encuentran errores en un módulo, habrá que volver al paso (1) y corregirlos antes de continuar. Para cada archivo fuente {.ASM} , el MASM crea un archivo objeto {.OBJ} . También pueden crearse durante el ensamblado, de manera opcional, archivos de listado {.LST} y de referencia cruzada (cross-reference) {.CRF}. Los archivos {.CRF} producidos por el MASM tienen un formato binario y para ser entendidos por el usuario, deben convertirse usando el programa auxiliar CREF; la salida de este programa comando es un archivo ASCII {*.REF}.
(Nota: si el programa va a ser ligado con otros módulos en lenguaje de alto nivel, los módulos fuente deben compilarse a archivos objeto en este punto). (No es el caso).

3) Usar opcionalmente el programa auxiliar LIB para unir varios archivos objeto {.OBJ} en un único archivo librería que tenga la extensión default {.LIB}. Este procedimiento es usado generalmente para archivos objeto de uso común que pueden ligarse con varios programas diferentes. Puede también crearse con LIB un listado-librería opcional.

4) Usar el programa LINK para combinar todos los archivos objeto y módulos-librería que conforman un programa, bajo un único archivo ejecutable {.EXE}. También debe ligarse un archivo fuente que ha sido preparado en el formato {.COM}; sin embargo, en ese caso, el archivo {.EXE} resultante NO puede ser corrido. Con LINK puede también crearse opcionalmente un archivo {.MAP}.

5) Si así se requiere, usar el programa EXE2BIN para convertir archivos ejecutables {.EXE} a un formato binario {.COM}.

El archivo {.EXE} a convertir debe haber sido preparado desde los archivos fuente y objeto, en un formato {.COM} válido.
(Nota: EXE2BIN puede también usarse para preparar archivos binarios que serán leídos por un intérprete o compilador de lenguaje de alto nivel). (No es el caso).

6) Depurar (Debug) el programa para descubrir errores lógicos. El proceso de "debugging" involucra varias técnicas, incluyendo las siguientes :

+ Correr el programa y estudiar sus entradas y salidas
+ Estudiar los archivos fuente y de listado
+ Usar el programa auxiliar CREF para crear un listado de referencias cruzadas (archivo {.REF}).
+ Usar CODEVIEW (CV) {Microsoft} para depurar durante la ejecución (normalmente ésta es la herramienta más eficiente para depuración).

Si se descubren errores lógicos, debe regresarse al paso (1) para corregir el código fuente.

Todo o parte de este ciclo de desarrollo para programas puede automatizarse usando el programa MAKE (make description files) (más útil para el desarrollo de programas complejos que involucran varios módulos fuente), o usando archivos BATCH de DOS ordinarios (método más eficiente para el desarrollo de programas de un único módulo).

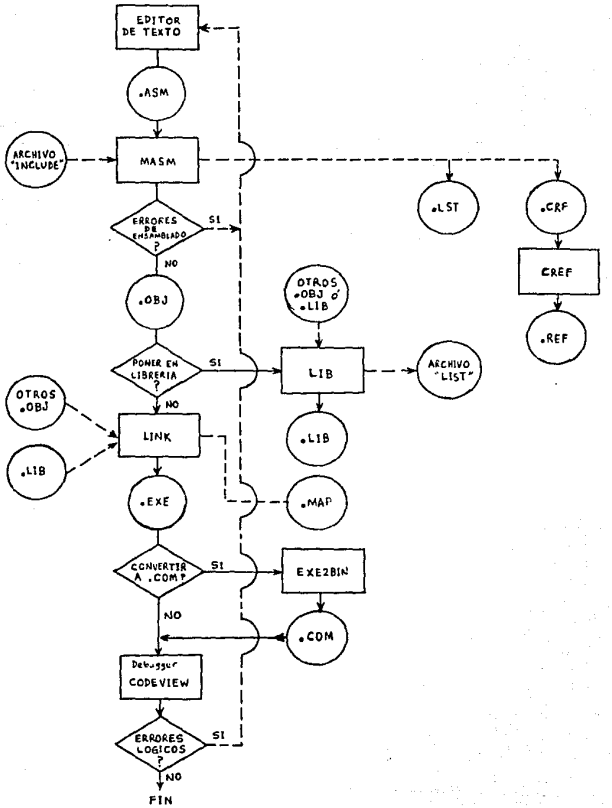
.....

En la { *figura* 4-2 } se muestra en forma diagramática el CICLO DE DESARROLLO PARA PROGRAMAS EN LENGUAJE ENSAMBLADOR desde el ambiente de la μC PC (PS) (y usando productos de Microsoft). También se indica el punto de enlace del sistema de Desarrollo **SDM88-PC** , dentro de este ciclo.

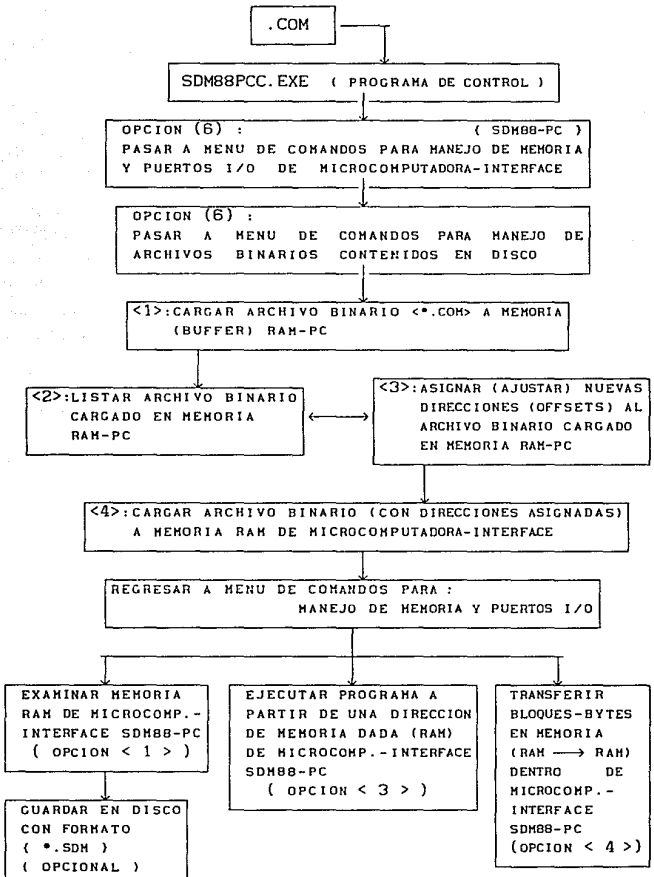
.....

{ *figura* 4-2 }.

Ciclo de Desarrollo para Programas en Lenguaje Ensamblador desde el ambiente de la µC PC (PS) IBM y punto de enlace con el sistema de Desarrollo SDM88-PC .



... punto de enlace con el Sistema de Desarrollo SDM88-PC :



4 . 3
DIFERENCIAS ENTRE EL FORMATO {*.EXE} Y {*.COM} PARA PROGRAMAS EN LENGUAJE ENSAMBLADOR

(Ambiente µC-PC (PS))

Un programa tipo-{*.COM}, reside en disco como una imagen absoluta de memoria, en un archivo con extensión ".COM". Este archivo no tiene un encabezado (header) ni alguna otra información de identificación interna. Por otro lado, un programa tipo-{*.EXE}, reside en disco como un tipo especial de archivo con un único encabezado, un mapa de relocalización, una verificación de suma (checksum) y otra información que es o puede ser usada por el sistema operativo "MS-DOS" : (Microsoft Disk Operating System).

{*.EXE} :

Este formato es el más común para programas que se ejecutan bajo DOS. En OS/2, un formato similar .EXE, es el único disponible para programas autónomos que aprovechen la característica de "multiprocesamiento" (multitasking). En el formato .EXE, los programas pueden tener varios Segmentos (multi-segment) y pueden ser de cualquier tamaño. Se pueden crear y ligar módulos usando tanto Ensamblador como compiladores de lenguajes de alto nivel (de una misma familia, ej. Microsoft o Borland). Módulos creados en diferentes lenguajes pueden combinarse para formar un único programa. Este es el formato recomendado (por Microsoft) para programas de tamaño considerable y amplio propósito.

{*.COM} :

Este formato es conveniente algunas veces para pequeños programas, limitados a un sólo Segmento (por tanto, no pueden ser mayores a 64 kBytes (a menos que usen "overlays")). Como no tiene

encabezado de archivo (header), un archivo *.COM es más reducido y requiere menos espacio de disco que un archivo *.EXE con el mismo programa. Esto hace que los programas en formato *.COM, sean una buena alternativa para pequeños programas autónomos en Ensamblador, de varios miles de bytes o menos.
Nota: El formato *.COM consiste en archivos ejecutables que no pueden contener información simbólica o de linea de archivo fuente para el debugger MS-CODEVIEW, por lo que sólo se puede depurar archivos *.COM en modo Ensamblador (assembly mode).

Así, tanto los archivos *.EXE como los *.COM, son programas en lenguaje de máquina cuya única diferencia es el formato-de-archivo -objeto con que se almacenan en disco.

En una μC-PC (PS) bajo DOS, el archivo *.COM está listo para ejecutarse. DOS puede cargarlo directamente desde disco a memoria. Una vez en memoria, DOS pasa el control al código ejecutable del programa situado a partir del offset 0100h, dentro del segmento apartado para dicho programa.

El archivo *.EXE no está listo para ejecutarse inmediatamente. Su archivo objeto en disco tiene un campo de encabezado que contiene información creada por el ligador (linker), siendo la más importante, la información de relocalización.

El archivo *.COM no es relocalizable, pues no tiene un encabezado equivalente al del archivo *.EXE . En cambio, el programa *.COM debe ser "relocalizable-por-Segmento-de-Código" (Code Segment relocatable), lo cual implica que el programa siempre será cargado en el mismo offset, mientras que el Segmento de Código CS puede ser modificado. Todos los offsets en el programa permanecen sin cambio.

Mientras el archivo *.COM es "relocalizable-por-CS", el archivo *.EXE puede ser relocalizado en varios segmentos diferentes. Esto limita el tamaño de los archivos *.COM a un máximo de 64 kBytes (a menos que el programa cargue los otros segmentos como parte de su ejecución (overlays)). Un archivo *.EXE puede contener múltiples segmentos que sean dinámicamente relocalizados en el espacio de programa reservado.

Para un archivo *.COM, DOS fija los registros : CS, DS, ES, y SS, al segmento en el cual carga el programa; esto inicializa al registro SP para apuntar a la última localidad de memoria disponible en dicho segmento. Así, el programa ocupa la primera parte del segmento, y el Stack la última.

Para un archivo *.EXE, los valores para los registros CS, IP, SS y SP, están especificados en el encabezado de archivo (header). DOS fija los registros DS y ES para apuntar al segmento en el cual carga el programa. El registro CS apunta al segmento identificado como el que contiene la localidad inicial de código del programa.

Mientras que un programa *.COM debe empezar en el offset 0100h del segmento-de-programa, un archivo *.EXE puede especificar una localidad inicial diferente. La directiva (pseudoOP) de Ensamblador "END", puede contener un valor de dirección :

ejemplo : END Start_Location

Esto indica al programa Ensamblador y al Ligador que debe transferir el control a la localidad de memoria etiquetada con "Start_Location", cuando el programa sea cargado. Esta localidad se conoce como "punto de entrada" (entry point) para el programa.

Los programas *.COM y *.EXE son designados genéricamente como "programas transitorios" (transient programs). Estos programas se apropian del bloque de memoria en que han sido colocados y casi toman control total de los recursos del sistema mientras se están

ejecutando. Cuando el programa termina (por haber sido abortado por DOS o por haber terminado su trabajo y regresado sistemáticamente a DOS), el bloque de memoria es entonces liberado (de ahí el término *transitorio*) y puede ser usado por el siguiente programa en linea a ser cargado.

El "Prefijo de Segmento de Programa" (Program Segment Prefix) ó PSP , es un área reservada de 256 (100h) bytes, que es preparada por el DOS en la base del bloque de memoria en que se coloca un "programa transitorio". El PSP contiene algunos eslabones con el DOS que pueden ser usados por el programa transitorio, una información que es salvada por DOS para sus propios propósitos y una información que es transferida entre DOS y el programa transitorio (que puede o no ser usada, según lo requiera el programa).

Tanto los archivos *.EXE como los *.COM usan el PSP. Para un archivo *.EXE, los registros DS y ES apuntan al área de datos, mientras que CS y SS, son fijados durante las etapas de ensamblado y ligado. Para un archivo *.COM, todos los registros apuntan al PSP. Esto da a ambos tipos de archivo, acceso inmediato a los datos en el PSP. La ventaja del archivo *.COM es que el registro CS identifica al PSP.

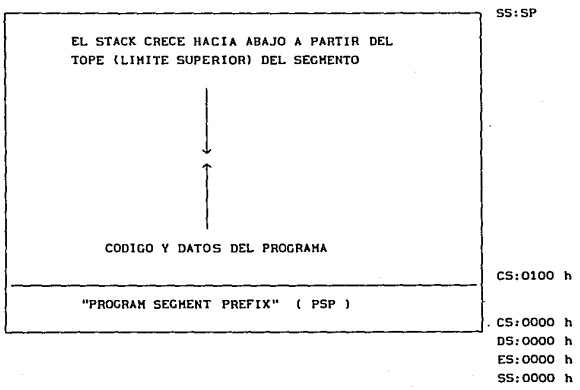
.....

A continuación se muestran las imágenes de memoria típicas de un archivo { *.COM } y de un archivo { *.EXE }, justo después de haberse cargado en memoria de μC-PC (PS) .

.....

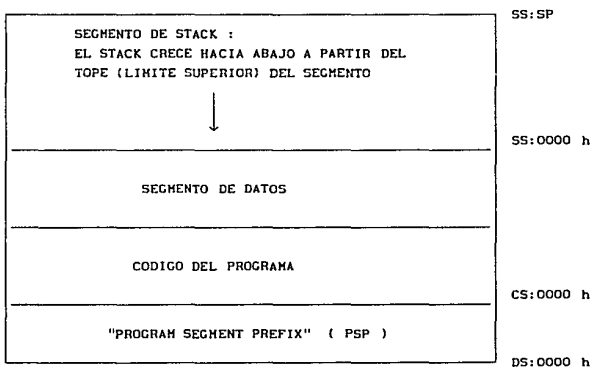
EN UN ARCHIVO *.COM , EL CONTENIDO DEL ARCHIVO ES CARGADO EN MEMORIA JUSTO SOBRE EL PSP. EL CODIGO Y LOS DATOS DEL PROGRAMA ESTAN MEZCLADOS (INCLUIDOS) DENTRO DEL UNICO SEGMENTO FISICO Y TODOS LOS REGISTROS DE SEGMENTO TIENEN EL MISMO VALOR : APUNTAN AL "PSP". EL REGISTRO STACK POINTER "SP" CONTIENE FFFEh SI LA MEMORIA INSTALADA LO PERMITE; DE OTRO MODO, ES INICIALIZADO CON UN VALOR TAN ALTO COMO SEA POSIBLE EN MEMORIA (MENOS 2 BYTES) (MS-DOS HACE UN "PUSH 0000" (WORD) EN STACK ANTES DE LA ENTRADA DEL ARCHIVO).

Imagen de memoria de un archivo { *.COM } típico después de cargarse :



EN UN ARCHIVO *.EXE , EL CONTENIDO ES RELOCALIZADO Y CARGADO EN MEMORIA SOBRE EL PSP. EL CODIGO, LOS DATOS Y EL STACK, RESIDEN EN SEGMENTOS SEPARADOS Y NO NECESITAN ESTAR EN EL ORDEN MOSTRADO. EL PUNTO DE ENTRADA ("ENTRY POINT") PUEDE ESTAR EN CUALQUIER LUGAR DENTRO DEL SEGMENTO DE CODIGO, Y ES ESPECIFICADO POR LA DIRECTIVA (PSEUDO-OP) "END" EN EL MODULO PRINCIPAL DEL PROGRAMA. CUANDO EL PROGRAMA RECIBE EL CONTROL, LOS REGISTROS "DS" Y "ES" APUNTAN AL PSP ; GENERALMENTE, EL PROGRAMA SALVA ESTE VALOR AUTOMATICAMENTE (EN STACK) (PARA PODER REGRESAR A "DOS") Y ENTONCES REASIGNA LOS REGISTROS "DS" Y "ES" PARA QUE APUNTEN A SU AREA DE DATOS.

Imagen de memoria de un archivo { *.EXE } tipico después de cargarse :



Resumen de las principales diferencias entre programas :

	{ *.COM }	{ *.EXE }
MAXIMA EXTENSION	(65536 BYTES) -(256 BYTES PSP) -(2 BYTES STACK)	SIN LIMITE
PUNTO DE ENTRADA ("ENTRY POINT")	PSP:0100 h	DEFINIDO POR PSEUDO-OP "END"
"CS" INICIAL	PSP	SEGMENTO QUE CONTIENE MODULO CON "PUNTO DE ENTRADA"
"IP" INICIAL	0100 h	OFFSET DEL "PUNTO DE ENTRADA" DENTRO DE SU SEGMENTO
"DS" INICIAL	PSP	PSP
"ES" INICIAL	PSP	PSP
"SS" INICIAL	PSP	SEGMENTO CON ATRIBUTO "STACK"
"SP" INICIAL	FFFE h O PALABRA TOPE SUPERIOR EN MEM. DISPONIBLE	EXTENSION DEL SEGM. DEFINIDA CON EL ATRIBUTO "STACK"
STACK INICIAL	"WORD" = 0000 h	INICIALIZADO O NO
EXTENSION DE STACK	(65536 BYTES) -(256 BYTES PSP) -(EXTENSION DEL CODIGO EJECUTABLE)	DEFINIDA EN EL SEGMENTO CON EL ATRIBUTO "STACK"
LLAMADAS DE SUBRUTINA	NEAR (DENTRO DE SEGMENTO)	NEAR O FAR (DENTRO O FUERA)
EXTENSION DEL ARCHIVO	TAMAÑO EXACTO DEL PROGRAMA (CODIGO EJECUTABLE + DATOS) (HEX)	TAMAÑO DEL PROGRAMA MAS "HEADER" (MULTIPLO DE 512 BYTES)
METODO DE SALIDA "DOS"	INT 21h; FUNC.4Ch	INT 21h; FUNC.4Ch

Los programas en formato { *.COM } son almacenados en archivos de disco que contienen una imagen absoluta de las instrucciones (en código de máquina {μP}) a ser ejecutadas. Puesto que estos archivos no tienen información de relocalización (header), son mucho más compactos, y se cargan más rápidamente para su ejecución en μC-PC (PS), que archivos *.EXE equivalentes.

Puesto que los programas *.COM son cargados inmediatamente sobre el (después del) PSP y no tienen un encabezado (header) que especifique algún punto de entrada (entry point), deben SIEMPRE tener un origen de 0100h {=IP} (que es el tamaño del PSP). La localidad 0100h debe contener alguna instrucción ejecutable. La máxima extensión de un programa *.COM es 65536 bytes, menos el tamaño del PSP (256 bytes) y una palabra obligatoria de stack (2 bytes).

Para la generación (conversión) de archivos {*.COM} a partir de archivos {*.EXE} dentro del ambiente de la μC-PC (PS), deben tenerse en cuenta las siguientes consideraciones :
(Trabajando con productos de Microsoft)

1 > El programa desarrollado por el usuario sólo puede tener un Segmento (máx. 64 kBytes = 65536 Bytes); es decir, puesto que el programa va a ser convertido en un archivo {*.COM}, es requisito que todas sus áreas de "código ejecutable" y "datos", estén contenidas dentro de un único segmento de código (realmente debe llamarse Segmento de código/datos). Por tanto, la directiva ".MODEL" del programa Ensamblador, no puede usarse para definir Segmentos "default" para la creación de archivos {*.COM} . Los tipos calificadores para el programa Ensamblador y Ligador : "Align", "Combine" y "Class", no necesitan especificarse puesto que no tienen razón de ser, cuando se trabaja con programas de un solo módulo (como los {*.COM}).

2 > Todos los registros de Segmento deben ser inicializados apuntando a un mismo segmento, mediante el uso de la directiva (pseudoOp) "ASSUME". Esto indica al programa Ensamblador (MASM), NO al microprocesador, qué segmento (de software) asociar con cada registro de segmento (CS,DS,ES,SS). Es decir, la directiva "ASSUME", simplemente "dice" al programa Ensamblador qué registros de Segmento va a emplear el usuario para apuntar a los diversos segmentos que pueda tener su programa, a fin de que el Ensamblador pueda proveer las sobre-asignaciones locales de segmento (Segment overrides), cuando sean necesarias. Es importante notar que la directiva "ASSUME", NO se ocupa de cargar los registros de Segmento con los valores apropiados; sólo notifica al programa Ensamblador sobre las intenciones del usuario para hacer eso dentro de su programa (con las instrucciones de software apropiadas, ej.: ({MOV AX,0030h},{MOV DS,AX}))

3 > En su programa, el usuario debe emplear la directiva "ORG" (ORG cambia el apuntador de instrucción {IP} por software) para empezar a ensamblar a partir del byte 256 (0100h), es decir, el programa DEBE TENER su origen en el offset 0100h. Esto deja espacio para el PSP del DOS (que, si el programa fuera a ejecutarse en el ambiente μC-PC (PS), se cargaría en memoria automáticamente "at run time"). También debe emplearse la directiva "END", que indica al programa Ensamblador que se ha alcanzado el final del archivo fuente y además especifica el "punto de entrada" (entry point) para el programa. Si el "punto de entrada" no es una etiqueta localizada en el offset 0100h, el archivo {*.EXE} especial resultante de los procesos de ensamblado y ligado de este programa fuente, no podrá ser convertido en un archivo {*.COM}. (Se dice que el archivo {*.EXE} previo al {*.COM}, es especial, porque no puede ejecutarse normalmente como un archivo *.EXE ordinario, en virtud de que ha sido relocalizado de manera característica, en preparación para la generación de un archivo *.COM).

Para crear un archivo {*.COM}, no debe definirse especificamente un segmento de STACK, por lo que, durante el proceso de Ligado (LINK) de un programa {*.COM}, el Ligador (Linker) desplegará el mensaje de advertencia :

Warning : no stack segment

que puede ser ignorado.
La entrada para el Linker es un archivo {*.OBJ} procedente del MASM y la salida es un archivo {*.EXE}, que debe ser convertido a un archivo {*.COM} con la utileria EXE2BIN del MS-DOS, antes de ejecutarse. Entonces el archivo {*.EXE} puede ser borrado de disco.

4 > La sentencia PROC usada dentro del lenguaje Ensamblador para designar a las subrutinas, tiene un atributo NEAR (si la subrutina puede ser llamada solamente por otro código dentro del mismo segmento) o un atributo FAR (si la subrutina puede ser llamada por código localizado en cualquier parte dentro del espacio de direccionamiento de memoria del µP 8088/8086 {mismo o distinto segmento}). Formalmente, las subrutinas dentro de un programa {*.COM} deben tener el atributo NEAR, puesto que todo el código ejecutable reside en un sólo segmento. Sin embargo, en lo referente al código para µC-Interface SDM88-PC, si alguna vez se requiere invocar una subrutina como PROC FAR, puede "hacerse un truco" : escribir dentro del código del programa los Bytes que conforman el código de operación de la instrucción FAR CALL (9Ah) y la dirección absoluta (IP-word),(CS-word) con el criterio "LSByte first", como si se tratara de bytes aislados de datos. Así, dentro de la µC-Interface, el efecto obtenido será igual al correspondiente al uso del mnemónico CALL FAR PTR Subr_name , siendo "Subr_name" el nombre de una subrutina declarada como "PROC FAR". Pero, para la correcta generación de un archivo {*.COM} en µC-PC (PS) bajo DOS, no deben usarse ni etiquetas, ni subrutinas (PROC) declaradas como FAR .

5 > Aunque puede incluirse cualquier información de "Datos" dentro del programa uni-segmento del usuario, ésta no debe ser ejecutada (no debe confundirse con "código ejecutable"). Para ello, puede usarse la instrucción "JMP" para saltar sobre el area de datos, o pueden ponerse los datos al final, después de que el programa regrese el control al sistema operativo (esto es aplicable tanto dentro del ambiente de la μC-Interface como de la μC-PC (PS) ... en este último caso, los datos podrían ponerse después de las instrucciones :

```
MOV  AX, 4C00h
INT  21h
```

que emplean las funciones del BIOS, para regresar el control al sistema operativo DOS .

****

En lo que respecta al **SDM88-PC**, lo relativo al PSP y al protocolo de carga y localización en memoria de μC-PC (PS), no es aplicable, ya que el programa no está destinado para ejecutarse en un ambiente PC (PS), sino en la μC-Interface. Sin embargo, sí es necesario cumplir con los requisitos de programación para que pueda generarse el archivo *.COM en la μC-PC (PS), ya que sólo el formato binario de los archivos {*.COM} es el adecuado para preparar automáticamente, código ejecutable que puede ser "bajado" (cargado) tanto en chips EPROM (p.ej. para la implementación del sistema operativo de la μC-Interface) como en chips RAM (programas del usuario actuantes sobre el hardware de la μC-Interface). Esto es así porque, como se mencionó antes, los archivos { *.COM } almacenados en disco, contienen una imagen binaria (hex) exacta de las instrucciones (en código de máquina {μP}) a ser ejecutadas (ya que son generados sin información de relocalización (header)).

Cabe mencionar también, que {*.COM} es la forma más compacta de guardar programas destinados a la μC-Interface, pues estos archivos ocupan un espacio en disco, EXACTAMENTE igual al número

neto de bytes que corresponden al código ejecutable y/o datos del programa introducido por el usuario; incluso los archivos {*.SDM} ocupan un espacio aproximadamente tres veces mayor que los archivos {*.COM} correspondientes a los mismos programas (ya que en los {*.SDM}, por cada byte de información almacenado, se guardan también dos bytes extra {CR}-{LF}).

****

A continuación se incluyen tres listados (*.LST) como ejemplos de programas-tipo para operar sobre el Hardware de la µC-Interface { **SDM88-PC** } :

DACTRIAN.ASM ⟶ DACTRIAN.COM
ADCCH012.ASM ⟶ ADCCH012.COM
BAUDRATE.ASM ⟶ BAUDRATE.COM

También se indica el procedimiento general típico de generación del archivo binario adecuado, desde el sistema operativo MS-DOS de la microcomputadora PC (PS), empleando programas auxiliares de desarrollo elaborados por "Microsoft":
MASM , CREF , LINK , EXE2BIN .

En este ejemplo del proceso de generación del {*.COM}, se muestra lo que el usuario debe teclear desde MS-DOS, y se intercalan despliegues de DIRectorio para observar los archivos que son creados después del uso de MASM , CREF , LINK y EXE2BIN . Finalmente los archivos que el usuario debe conservar en disco son *.ASM (programa fuente) y *.COM (archivo binario listo para ser bajado a RAM de µC-Interface SDM88-PC); si acaso también *.LST como referencia de código/datos y para obtención de copia en impresora. Aquí se muestra, a manera de ejemplo, la obtención de DACTRIAN.COM a partir de DACTRIAN.ASM , pero el procedimiento sería igual para cualquier archivo fuente *.ASM . +

```
D:\SDM88-PC>DIR

 Volume in drive D is USERF_&_TST
 Directory of  D:\SDM88-PC

.            <DIR>      10-25-89   3:06p
..           <DIR>      10-25-89   3:06p
MASM     EXE   110703   2-01-88   1:00p
LINK     EXE    65475   2-01-88   1:00p
CREF     EXE    28427   2-01-88   1:00p
EXE2BIN  EXE     3050   1-01-88  12:00a
DACTRIAN ASM     8835  10-25-89  10:00a
        7 File(s)    1449984 bytes free

D:\SDM88-PC>
```

```
D:\SDM88-PC>MASM /A/B63/MX/ML/V/W2/Z/C/L/X DACTRIAN.ASM, DACTRIAN.OBJ, DACTRIAN.
LST, DACTRIAN.CRF
Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Microsoft Corp 1981, 1988.  All rights reserved.


    204 Source  Lines
    204 Total   Lines
     15 Symbols

  47106 + 279880 Bytes symbol space free

      0 Warning Errors
      0 Severe  Errors

D:\SDM88-PC>
```

```
D:\SDM88-PC>DIR DACTRIAN.*

 Volume in drive D is USERF_&_TST
 Directory of  D:\SDM88-PC

DACTRIAN ASM     8835  10-25-89  10:00a
DACTRIAN LST    13632  10-25-89  10:12a
DACTRIAN OBJ      279  10-25-89  10:12a
DACTRIAN CRF     1813  10-25-89  10:12a
        4 File(s)    1425408 bytes free

D:\SDM88-PC>
```

```
D:\SDM88-PC>CREF DACTRIAN.CRF, DACTRIAN.REF
Microsoft (R) Cross-Reference Utility  Version 5.10
Copyright (C) Microsoft Corp 1981-1985, 1987.  All rights reserved.


12 Symbols

D:\SDM88-PC>
D:\SDM88-PC>ERASE DACTRIAN.CRF

D:\SDM88-PC>


D:\SDM88-PC>DIR DACTRIAN.*

 Volume in drive D is USERF_&_TST
 Directory of  D:\SDM88-PC

DACTRIAN ASM      8835  10-25-89  10:00a
DACTRIAN LST     13632  10-25-89  10:12a
DACTRIAN OBJ       279  10-25-89  10:12a
DACTRIAN REF       891  10-25-89  10:15a
        4 File(s)    1425408 bytes free

D:\SDM88-PC>



D:\SDM88-PC>LINK /INFO/MAP/LINENUM/NOI DACTRIAN.OBJ, DACTRIAN.EXE, DACTRIAN.MAP,


Microsoft (R) Overlay Linker  Version 3.64
Copyright (C) Microsoft Corp 1983-1988.  All rights reserved.

Libraries [.LIB]:
**** PASS ONE ****
DACTRIAN.OBJ(DACTRIAN.ASM)
**** LIBRARY SEARCH ****
**** ASSIGN ADDRESSES ****
LINK : warning L4021: no stack segment
  1 segment "CodeSeg" class "" length 133H bytes
**** PRINT MAP ****
**** PASS TWO ****
DACTRIAN.OBJ(DACTRIAN.ASM)
**** WRITING EXECUTABLE ****

D:\SDM88-PC>
```

```
D:\SDM88-PC>DIR DACTRIAN.*

 Volume in drive D is USERF_&_TST
 Directory of  D:\SDM88-PC

DACTRIAN ASM      8835  10-25-89   10:00a
DACTRIAN LST     13632  10-25-89   10:12a
DACTRIAN OBJ       279  10-25-89   10:12a
DACTRIAN MAP       864  10-25-89   10:19a
DACTRIAN REF       891  10-25-89   10:15a
DACTRIAN EXE       819  10-25-89   10:19a
        6 File(s)    1417216 bytes free

D:\SDM88-PC>



D:\SDM88-PC>EXE2BIN DACTRIAN.EXE DACTRIAN.COM

D:\SDM88-PC>
D:\SDM88-PC>ERASE DACTRIAN.EXE

D:\SDM88-PC>



D:\SDM88-PC>DIR DACTRIAN.*

 Volume in drive D is USERF_&_TST
 Directory of  D:\SDM88-PC

DACTRIAN ASM      8835  10-25-89   10:00a
DACTRIAN LST     13632  10-25-89   10:12a
DACTRIAN OBJ       279  10-25-89   10:12a
DACTRIAN MAP       864  10-25-89   10:19a
DACTRIAN REF       891  10-25-89   10:15a
DACTRIAN COM        51  10-25-89   10:21a
        6 File(s)    1417216 bytes free

D:\SDM88-PC>
```

Archivo: DACTRIAN.COM

```
0000:BB 30 00 BE DB BB 00 0A BB 00 00 B9 00 01 BB 07      .0..............
0010:43 FE C0 E2 F9 BB FF 00 B9 00 01 BB 07 43 FE C8      C............C..
0020:E2 F9 BA 30 FF BB 00 0A B9 00 02 BA 07 EE 43 E2      ...0..........C.
0030:FA EB F2      ...
```

```
                        ; Print Format : PAGE length,width
                                         PAGE   50  , 132
                        ;*******************************************************************************
                        ; +
                        ; Programa :
                        ;
                        ; DACTRIAN.ASM
                        ;
                        ; Reciclado de informacion leida de RAM, para alimentar entradas digitales
                        ; de un D/A c.  para la generacion de onda triangular (rampa ascendente y
                        ; descendente).
                        ;
                        TITLE   DACTRIAN ( D/A c. Test ) _ SDM88-PC
                        ;
                        ; ( DACTRIAN.ASM )
                        ;   Este archivo tiene un formato  ( *.COM )  (Segmento unico), en vez de uno
                        ;   ( *.EXE ) (PoliSegmentado).
                        ;   El archivo esta hecho para que su  ( *.COM )  generado, ( DACTRIAN.COM ) ,
                        ;   pueda cargarse directamente (sin modificaciones ni ajustes),
                        ;   en Memoria RAM del Sistema SDM88-PC , dentro de la zona destinada para
                        ;   programas de usuario.
                        ;   Este programa tiene codigo relocalizable (relative jumps).
                        ;
                        ;   ( CS:0100h )  es la localidad fijada por el formato  ( *.COM ) .
                        ;   Formato  ( *.COM ) :
                        ;   Requiere de la inclusion tanto del Codigo del programa como de los datos,
                        ;   dentro de un mismo y unico segmento y dar un offset inicial de  0100 h  ,
                        ;   pues los bytes 0 a FFh son ocupados por el    "Program Segment Prefix"
                        ;   asignado por la PC (at run time).
                        ;   Por eso, en este programa se define al inicio :  ORG 0100h
                        ;
                        ; Descripcion:
                        ; Este programa efectua un loop que incrementa ciclicamente el contenido del
                        ; registro AL desde ( 00h ) hasta ( FFh ), para la generacion de una rampa
                        ; ascendente, guardando estos valores en 256 localidades adyacentes de
                        ; memoria RAM.  Posteriormente, el reg. AL es decrementado sucesivamente para
                        ; la generacion de los valores correspondientes a una rampa descendente y estos
                        ; datos almacenados en las sigs. 256 locs. adyacentes de RAM.
                        ; Una vez hecho esto, se efectua un nuevo loop de duracion indefinida, donde se
                        ; leen estos valores de RAM y se mandan al puerto paralelo de salida :
                        ; ( PPortIA = FF30h = PPortDAC ]  que suministra la informacion digital a un
                        ; convertidor Digital/Analogico de 8-bits : MC1408-P8 o DAC-08.
                        ; El efecto obtenido sera que en la salida del D/A c. se estara generando una
                        ; ONDA TRIANGULAR periodica visible a traves de un osciloscopio.
                        ; Es pues una prueba o diagnostico para la interface D/A.
                        ; Se usa al reg. "BX" como apuntador, [BX], al buffer de memoria de datos: (200h).
```

```
                        ; El puerto paralelo de salida utilizado puede ser :
                        ; * Uno de los 3 puertos paralelos de 8 bits del 82C55A PPI, programado como
                        ;   salida, o
                        ; * Un "Octal D-type Transparent Latch" 74LS373
                        ;   acondicionado para tal fin al conectar sus entradas 8D-1D al bus de Datos
                        ;   del sistema; sus salidas 8Q-1Q a las entradas digitales del DAC :
                        ;   A1(MSB)-A8(LSB); "Output Control" conectado a GND y "Enable Latch" usado como
                        ;   Chip Select (CS), conectado a la salida correspondiente de un decodificador
                        ;   para puertos I/O (74LS138) (Yx), a traves de un inversor.
                        ; Ambas opciones se probaron y funcionan, pero con la primera se obtuvo una
                        ; señal mas limpia de ruido digital.
                        ; El programa termina en un loop infinito por lo que el control NO se regresa a
                        ; la rutina principal de control del  SDM88-PC : "Manager Routine", por lo que,
                        ; para salir de dicho ciclo y dejar al sistema preparado para recibir alguna
                        ; nueva orden (a traves del puerto serie de la PC), habra que dar un RESET al
                        ; sistema SDM88-PC (ajustando tambien de nuevo la velocidad de Tx/Rx serie en
                        ; PC a 300 Bauds en caso de haberse alterado previamente a la ejecucion de este
                        ; programa).
                        ;
                        ; Programa para operar sobre el hardware del sistema digital :
                        ;                                "   ( SDM88-PC )   "
                        ; basado en el Microprocesador 8088  ( @ 4.77 MHz )  y en una Microcomputadora
                        ; Personal IBM-compatible.
                        ; Periodo de operacion experimental = Ts = 210 [ns].
                        ; SDM88-PC :
                        ; " (S)istema de (D)esarrollo basado en el (M)icroprocesador (8088)
                        ;   y en una microcomputadora personal tipo IBM (PC) (PS)"
                        ;
                        ; Notas:
                        ; Los codigos de operacion de las instrucciones de este programa que a su vez
                        ; conformaran el contenido a ser cargado en RAM de SDM88-PC, se observaran
                        ; directamente a traves del Debugger "CODEVIEW" , mientras que el archivo
                        ;                                DACTRIAN.LST
                        ; se puede usar tan solo como auxiliar en la corroboracion de dichos codigos y
                        ; como referencia completa del codigo fuente.
                        ;
                        ; Este programa NO funciona en un ambiente microcomputadora IBM PC (PS), pues
                        ; se refiere a direcciones especificas de un hardware externo a la PC.
                        ;
                        ; Desarrollado con : Microsoft Macro Assembler ..... Version : 5.10
                        ;                    for the MS-DOS Operating System.
                        ;
                        ; Microprocesador  : intel 8088
                        ;
                        ; Registros usados   : AX,BX,CX,DX,DS,CS,IP,Flags
                        ; Puertos I/O usados : PPortIA  o  PPortDAC
```

```
                        ; Subrutinas usadas  : ninguna
                        ; Rutinas de Servicio a Interrupcion : ninguna
                        ;
                        ;
                        ; Autor   :  Agustin Eduardo Alvarez Vaca
                        ; Fecha   :  Abril/13/1989
                        ; Version :  1.0
                        ;
                        ;
                        ; Para convertir de formato de archivo (*.EXE)  a  (*.COM)
                        ; * Solo puede usarse un unico segmento (64kBytes max.)
                        ;*****************************************************************************


                        ; ***  DECLARACIONES GLOBALES  ***

                        ; Procedures :
                        ;             ( ninguno )
                        ; Variables :  ( named mem.locs. )
                        ;             ( ninguna )
                        ; Labels :
                        ; Etiquetas de flujo de Proceso :
                        PUBLIC      Code_block
                        PUBLIC      Load_Data_RUp,Load_Data_RDown
                        PUBLIC      Out_DAC_Reset,Out_DAC
                        ; Constants :
                        PUBLIC      DSRAMVar
                        PUBLIC      DACbuffer,SizeBuffer
                        PUBLIC      PPort1A  ; PPortDAC

                        ;

                        ; ***  CONSTANTES  ***

                        ; PARAMETROS : Direcciones del sistema dentro del mapa de Memoria y del mapa de
                        ;              Puertos I/O, usados por este programa controlador del Hardware
                        ;              del "SDM88-PC"

                        ; Direcciones de PUERTOS_I/O
                        ; 82C55A PPI # 1
= FF30                  PPort1A     EQU     0FF30h          ; OUT - Info. Digital dirigida a DAC
                        ; PPortDAC

                        ; Direcciones de memoria para variables del programa  ( Prog_Data )
= 0030                  DSRAMVar    EQU     0030h           ; Data Segment RAM  (phys.addr.=00300h)
= 0A00                  DACbuffer   EQU     0A00h           ; offset de buffer de datos DAC dentro DS
```

```
                                ; Tamaño del buffer de memoria para el DAC
= 0200                          SizeBuffer      EQU     0200h           ; (puede ser cualquier numero dentro
                                                                        ; de los limites de RAM instalada)

                                ; ***  SEGMENTO  ***

                                ; Programa para el Sistema Digital "SDM88-PC"

0000                            CodeSeg         SEGMENT                 ; definir Code Segment

                                                ASSUME  cs:CodeSeg

                                                ; fijar contador de localidad a 0100h ( formato archivo ( *.COM ) )

0100                                            ORG     0100h

0100                            Code_block:

                                ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                                ; DATOS ->
                                ;       ( no existen )
                                ; <- DATOS
                                ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

0100  BB 0030                                   mov     bx,DSRAMVar
0103  8E DB                                    mov     ds,bx           ; inicializar DS
0105  BB 0A00                                  mov     bx,DACbuffer    ; offset de buffer_datos_DAC dentro de DS

                                        ; Rampa Ascendente :
0108  B8 0000                                  mov     ax,0000h        ; info. digital inicial para D/A c
010B  B9 0100                                  mov     cx,SizeBuffer/2 ; inicializar contador p'datos de rampa ascendente
010E                            Load_Data_RUp:
010E  88 07                                    mov     BYTE PTR ds:[bx],al
0110  43                                       inc     bx                      ; generacion de rampa ascendente
0111  FE C0                                    inc     al                      ; Tabla en RAM
0113  E2 F9                                    loop    Load_Data_RUp

                                        ; Rampa Descendente :
0115  B8 00FF                                  mov     ax,00FFh        ; info. digital inicial para D/A c
0118  B9 0100                                  mov     cx,SizeBuffer/2 ; inicializar contador p'datos de rampa descendente
011B                            Load_Data_RDown:
011B  88 07                                    mov     BYTE PTR ds:[bx],al
011D  43                                       inc     bx                      ; generacion de rampa descendente
011E  FE C8                                    dec     al                      ; Tabla en RAM
0120  E2 F9                                    loop    Load_Data_RDown
```

```
                                        ;mov    dx,PPortDAC
0122  BA FF30                           mov     dx,PPort1A      ; (=FF30h)
0125                            Out_DAC_Reset:
0125  BB 0A00                           mov     bx,DACbuffer    ; (=0A00h)
0128  B9 0200                           mov     cx,SizeBuffer   ; (Tamaño buffer= # definido por usuario)
012B                            Out_DAC:
012B  8A 07                             mov     al,BYTE PTR ds:[bx] ; leer datos de Tabla en RAM
012D  EE                                out     dx,al           ; mandar byte a D/A c
012E  43                                inc     bx              ; inc BX ... RAM buffer mem.apuntador
012F  E2 FA                             loop    Out_DAC         ; para generar patron de onda de 1 ciclo
                                                                ; (rampa en este caso)
0131  EB F2                             jmp     Out_DAC_Reset   ; reciclar buffer de memoria para generar
                                                                ; patron de onda periodico a la salida de DAC
0133                            CodeSeg ENDS
                                ;
                                        END     Code_block
```

Segments and Groups:

N a m e	Length	Align	Combine Class
CodeSeg	0133	PARA	NONE

Symbols:

N a m e	Type	Value	Attr
Code_block	L NEAR	0100	CodeSeg Global
DACbuffer	NUMBER	0A00	Global
DSRAMVar	NUMBER	0030	Global
Load_Data_RDown	L NEAR	011B	CodeSeg Global
Load_Data_RUp	L NEAR	010E	CodeSeg Global
Out_DAC	L NEAR	012B	CodeSeg Global
Out_DAC_Reset	L NEAR	0125	CodeSeg Global
PPort1A	NUMBER	FF30	Global
SizeBuffer	NUMBER	0200	Global
@Cpu	TEXT	0101h	
@FileName	TEXT	DACTRIAN	
@Version	TEXT	510	

204 Source Lines
204 Total Lines
15 Symbols

47106 + 279880 Bytes symbol space free

0 Warning Errors
0 Severe Errors

DACTRIAN.REF

```
Microsoft Cross-Reference  Version 5.10       Wed Oct 25 10:15:54 1989
DACTRIAN ( D/A c. Test ) _ SDM88-PC

  Symbol Cross-Reference            (# definition, + modification)      Cref-1

@Cpu . . . . . .  1#
@Version . . . . . .  1#

CodeSeg. . . . . . 148#  150   200
Code_block . . . . . 115   156#  204

DACbuffer. . . . . 120   138#  166   189
DSRAMVar . . . . . 119   137#  164

Load_Data_RDown. . . . . 116   180#  184
Load_Data_RUp. . . . . 116   171#  175

Out_DAC. . . . . 117   191#  195
Out_DAC_Reset. . . . . 117   188#  197

PPort1A. . . . . 121   133#  187

SizeBuffer . . . . . 120   141#  170   179   190


  12 Symbols
```

DACTRIAN.MAP

```
LINK : warning L4021: no stack segment

  Start  Stop   Length Name                   Class
  00000H 00132H 00133H CodeSeg

   Address         Publics by Name

  0000:0100       Code_block
  0000:0A00  Abs  DACbuffer
  0000:0030  Abs  DSRAMVar
  0000:011B       Load_Data_RDown
  0000:010E       Load_Data_RUp
  0000:012B       Out_DAC
  0000:0125       Out_DAC_Reset
  0000:FF30  Abs  PPort1A
  0000:0200  Abs  SizeBuffer

   Address         Publics by Value

  0000:0030  Abs  DSRAMVar
  0000:0100       Code_block
  0000:010E       Load_Data_RUp
  0000:011B       Load_Data_RDown
  0000:0125       Out_DAC_Reset
  0000:012B       Out_DAC
  0000:0200  Abs  SizeBuffer
  0000:0A00  Abs  DACbuffer
  0000:FF30  Abs  PPort1A

Program entry point at 0000:0100

Segments               1
Groups                 1
Bytes in symbol table  32784
```

```
                        ; Print Format : PAGE length,width
                                 PAGE   50  , 132
                        ;*******************************************************************************
                        ; +
                        ; Programa :
                        ;
                        ; ADCCH012.ASM
                        ;
                        ; Adquisicion de informacion (señal) analogica y digitalizacion de la misma
                        ; mediante un convertidor Analogico/digital: ADC0809.
                        ; Lectura de tres señales analogicas por Channel-0 , Channel-1 y Channel-2 .
                        ; Informacion digital almacenada en buffer en RAM, en localidades contiguas:
                        ; byte_CH0,byte_CH1,byte_CH2,byte_CH0,byte_CH1,byte_CH2,byte_CH0,... etc.
                        ; Reciclado de informacion leida de RAM, para alimentar entradas digitales
                        ; de un D/A c.  (Buffer de cualquier numero de elementos).
                        ; La rutina de control para el A/D c. es residente en EPROM y se invoca como
                        ; FAR call.
                        ; Las señales analogicas a leer por el ADC (CH-0 , CH-1 y CH-2) deben tener una
                        ; frecuencia menor a 200 [Hz], para poder reconstruirse apropiadamente a la
                        ; salida del DAC.
                        ;
                        TITLE   ADCCH012 ( A/D c. Test ) _ SDM88-PC
                        ;
                        ; ( ADCCH012.ASM )
                        ;   Este archivo tiene un formato  ( *.COM )  (Segmento unico), en vez de uno
                        ;   ( *.EXE ) (PoliSegmentado).
                        ;   El archivo esta hecho para que su  ( *.COM )  generado, ( ADCCH012.COM ) ,
                        ;   pueda cargarse directamente (sin modificaciones ni ajustes),
                        ;   en Memoria RAM del Sistema SDM88-PC , dentro de la zona destinada para
                        ;   programas de usuario.
                        ;   Este programa tiene codigo relocalizable ya que la rutina f_AD_Read es
                        ;   invocada desde EPROM, a traves de una direccion absoluta.
                        ;
                        ;   ( CS:0100h )  es la localidad fijada por el formato  ( *.COM ) .
                        ;   Formato  ( *.COM ) :
                        ;   Requiere de la inclusion tanto del Codigo del programa como de los datos,
                        ;   dentro de un mismo y unico segmento y dar un offset inicial de  0100 h  ,
                        ;   pues los bytes 0 a FFh son ocupados por el    "Program Segment Prefix"
                        ;   asignado por la PC (at run time).
                        ;   Por eso, en este programa se define al inicio :  ORG 0100h
                        ;
                        ; Descripcion:
                        ; Este programa efectua un loop que posiciona sucesivamente en localidades
                        ; adyacentes dentro de un buffer en memoria RAM, la informacion digitalizada de
                        ; tres señales analogicas (una leida por Channel-0 , otra por Channel-1 y otra
                        ; mas por Channel-2), suministrada por un convertidor Analogico/Digital ADC0809.
```

```
                        ; Una vez hecho esto, se efectua un nuevo loop, leyendo estos valores de RAM
                        ; y mandandolos al puerto paralelo de salida [ PPortIA=FF30h ] que suministra
                        ; la informacion digital a un convertidor Digital/Analogico de 8-bits :
                        ; MC1408-P8 o DAC-08.
                        ; Primero se leen las localidades donde se almaceno la informacion digitalizada
                        ; correspondiente al Canal-0, y se realizan 1000h=4096 ciclos de despliegue en
                        ; el DAC (contador=reg.SI); luego las localidades que contienen la info. del
                        ; Canal-1, y se efectuan tambien 4096 ciclos de despliege; por ultimo, las locs.
                        ; que conservan la info. del Canal-2, llevandose a cabo 4095 ciclos de despliegue.
                        ; (aprox. 22[s]), luego de nuevo las localidades CH-0 y se repite indefinidamente
                        ; el proceso.  Las localidades de RAM donde se almacena la informacion
                        ; correspondiente a uno cualquiera de los tres canales estan separadas dos bytes
                        ; entre si; p.ej. para el canal-0 : (0,3,6,9,C,...), para el canal-1 : (1,4,7,...)
                        ; para el canal-2 : (2,5,8,...).
                        ; El efecto obtenido sera que en la salida del D/A c. se estara generando una
                        ; replica periodica de la porcion de señal leida previamente por el Canal-0 del
                        ; A/D c. durante unos 22[s] (a freq.de microp.= 4.77[MHz] y de ADC = 1.2 [MHz]),
                        ; luego durante el mismo lapso, de la señal leida por el Canal-1, finalmente,
                        ; de la señal leida por el Canal-2, y asi ciclica y alternadamente.
                        ; La señal del DAC sera visible a traves de un osciloscopio.
                        ; Es pues una prueba o diagnostico para la interface A/D y D/A.
                        ; El tamaño del buffer de RAM puede tener cualquier numero de elementos
                        ; (definido en la constante "SizeBuffer"). El tamaño neto del buffer es en
                        ; realidad  3*SizeBuffer  pues se almacenan alternadamente los bytes
                        ; correspondientes a CH0 , a CH1 y a CH2.
                        ; Se usa al reg. "DI" como apuntador, [DI], al buffer de memoria de datos.
                        ; El ADC0809 es controlado digitalmente por el microprocesador a traves de un
                        ; puerto paralelo de salida [ PPortIB=FF31h ] y uno de entrada [ ADC_EOC_port=FF50h ]
                        ; para, mediante "polling", sensar EOC (End-Of-Conversion) y saber cuando la
                        ; info. digital puede ser leida por el puerto paralelo de entrada
                        ; [ ADCdata=PPortIC=FF32h ] y de ahi pasar al bus de Datos del sistema.
                        ; Los 3 puertos paralelos usados son de un 82C55A PPI.
                        ; El protocolo de control de A/D c. esta contenido en la rutina "f_AD_Read",
                        ; que, residente en EPROM de SDM88-PC (SDM88S03) (FE00:1240), se invoca como
                        ; FAR procedure.
                        ; Debe proporcionarse tan solo la direccion del canal que se desee leer
                        ; (000 a 111) en los 3 bits menos significativos del registro "BL".
                        ; La info. digitalizada correspondiente a la señal analogica leida con el
                        ; A/D c., sera regresada por esta rutina en el reg. "AL".
                        ; La señal de Reloj (ck) para el ADC0809, es suministrada por el Canal 1 del
                        ; Timer Programable 8254, por lo que inicialmente se programa a una frecuencia
                        ; menor a 1.2 MHz, apropiada para que el A/D c. funcione correctamente.
                        ; El programa termina en un loop infinito por lo que el control NO se regresa a
                        ; la rutina principal de control del  SDM88-PC : "Manager Routine". Por tanto,
                        ; para salir de dicho ciclo y dejar al sistema preparado para recibir alguna
                        ; nueva orden (a traves del puerto serie de la PC), habra que dar un RESET al
```

```
                        ; sistema SDM88-PC (ajustando tambien de nuevo la velocidad de Tx/Rx serie en
                        ; PC a 300 Bauds en caso de haberse alterado previamente a la ejecucion de este
                        ; programa).
                        ;
                        ; Programa para operar sobre el hardware del sistema digital :
                        ;                          *   ( SDM88-PC )   *
                        ; basado en el Microprocesador 8088  ( @ 4.77 MHz )  y en una Microcomputadora
                        ; Personal IBM-compatible.
                        ; Periodo de operacion experimental = Ts = 210 [ns].
                        ; SDM88-PC :
                        ; * (S)istema de (D)esarrollo basado en el (M)icroprocesador (8088)
                        ;   y en una microcomputadora personal tipo IBM (PC) *
                        ;
                        ; Notas:
                        ; Los codigos de operacion de las instrucciones de este programa que a su vez
                        ; conformaran el contenido a ser cargado en RAM de SDM88-PC, se observaran
                        ; directamente a traves del Debugger "CODEVIEW" , mientras que el archivo
                        ;                                ADCCH012.LST
                        ; se puede usar tan solo como auxiliar en la corroboracion de dichos codigos y
                        ; como referencia completa del codigo fuente.
                        ;
                        ; Este programa NO funciona en un ambiente microcomputadora IBM PC, pues se
                        ; refiere a direcciones especificas de un hardware externo a la PC.
                        ;
                        ; Desarrollado con : Microsoft Macro Assembler  .....  Version : 5.10
                        ;                    for the MS-DOS Operating System.
                        ;
                        ; Microprocesador  : intel 8088
                        ;
                        ; Registros usados   : AX,BX,CX,DX,DI,SI,DS,CS,IP,Flags
                        ; Puertos I/O usados : PPortIA (PPortDAC) , PPortIB , PPortIC
                        ;                      ADC_EOC_port,ADCdata
                        ;                      PITimerControlW , Counter1
                        ; Subrutinas usadas  : f_AD_Read
                        ; Rutinas de Servicio a Interrupcion : ninguna
                        ;
                        ;
                        ; Autor    :   Agustin Eduardo Alvarez Vaca
                        ; Fecha    :   Jun/28/1989
                        ; Version  :   1.0
                        ;
                        ;
                        ; Para convertir de formato (*.EXE) a (*.COM)
                        ; * solo un segmento puede ser usado  (max.64kBytes)
                        ;
                        ;*******************************************************************************
```

```
                                ; ***  DECLARACIONES GLOBALES  ***

                                ; Procedures :
                                ;PUBLIC         f_AD_Read
                                ; Variables :  ( locs.Mem referidas por nombre )
                                ;               ( ninguna )
                                ; Labels :
                                ; Etiquetas de flujo de Proceso :
                                PUBLIC          Code_block
                                PUBLIC          AnalogRD
                                PUBLIC          Out_DAC_Reset,Out_DAC
                                PUBLIC          ChangeCH_display,CH_flag_set
                                ; Constants :
                                PUBLIC          DSRAMVar
                                PUBLIC          ADCbuffer,SizeADCbuffer
                                PUBLIC          Chn10,Chn11,Chn12,NChannelCycles
                                PUBLIC          PPortIA
                                PUBLIC          PITimerControlW,Counter1
                                PUBLIC          FreqScaleF

                                ;

                                ; ***  CONSTANTES  ***

                                ; PARAMETROS : Direcciones del sistema dentro del mapa de MEMORIA y de PUERTOS I/O
                                ;               usadas por este programa para manejo del Hardware de "SDM88-PC"

                                ; Direcciones de Puertos I/O

                                ; 8254 P.Timer
171 = FF03                      PITimerControlW  EQU    0FF03h          ; 8254 Control Word
172 = FF01                      Counter1        EQU     0FF01h          ; A/D c. Clock

                                ; 82C55A (#1) Programmable Peripheral Interface (3_8-bit_Parallel_Ports)
175 = FF30                      PPortIA         EQU     0FF30h          ; OUT (PA0-PA7) (D/A conv. data)
                                ;PPortIB        EQU     0FF31h          ; OUT [PB0-PB7] (A/D conv.cntrl.(CH.addr.,ALE,START
                                )
                                ;PPortIC        EQU     0FF32h          ; IN  [PC0-PC7] (A/D conv.data input)

                                ; Direcciones de Memoria para las variables del programa ( Prog_Data )
180 = 0030                      DSRAMVar        EQU     0030h           ; Data Segment RAM  (phys.addr.=00300h)
181 = 0A00                      ADCbuffer       EQU     0A00h           ; offset del buffer de datos A/D c. dentro de DS

                                ; Tamaño del buffer A/D c. en memoria
```

```
                        SizeADCbuffer  EQU     0200h           ; (puede ser cualquier # dentro de los
                                                               ; limites de RAM instalada)
                        ; Numero de ciclos por canal para despliegue de informacion
                        NChannelCycles EQU     1000h           ; 4096 Ciclos de despliegue_DAC por Canal

                        ; Factor de escala para fijar frecuencia en 8254
                        FreqScaleF     EQU     0002h           ; factor de escala para dividir frecuencia
                                                               ; de entrada de 8254  ( 2.38095 [MHz] )
                                                               ; para generar señal CLK de ADC

                        ; *** SEGMENTO ***

                        ; Programa para el sistema Digital "SDM88-PC"

0000                    CodeSeg        SEGMENT                 ; definir Code Segment

                                       ASSUME  cs:CodeSeg

                                       ; fijar contador de localidad (IP) en 0100h (formato archivo (*.COM))

0100                                   ORG     0100h

                        ; NO se genera archivo *.COM en PC, si en el programa se declaran etiquetas
                        ; o subrutinas como "FAR" ( FAR LABELs o FAR PROCs ), usar en cambio
                        ; NEAR LABELs,  NEAR CALLs/JMPs  y cuando se requieran FAR CALLs/JMPs , usar
                        ; directivas "DB" y "DW" para definir Bytes y Words que representen
                        ; los OpCodes, Offsets (IP) y Segment-bases (CS) de estas instrucciones.

0100                    Code_block:

                        ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                        ; DATOS ->
                        ;          ( no existen )
                        ; <- DATOS
                        ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

                                       ;8254 : Channel 1 : A/D c. clock  (max.= 1.2 MHz)
                                       ;       ( 1.1905 MHz ... Factor de Escala = 0002h )
0100  BA FF03                          mov     dx,PITimerControlW ; (=FF03h)
0103  B0 76                            mov     al,01110110b    ; sqr.wave gen. , Binary counter
0105  EE                              out     dx,al
0106  BA FF01                          mov     dx,Counter1     ; (=FF01h)
0109  B8 0002                         mov     ax,FreqScaleF   ; factor de escala para fijar frecuencia
010C  EE                              out     dx,al           ; factor de escala (LSByte)
010D  8A C4                           mov     al,ah
010F  EE                              out     dx,al           ; factor de escala (MSByte)
```

```
                                
0110  B8 0030                  mov     bx,DSRAMVar     ; (=0030h)
0113  8E DB                    mov     ds,bx           ; inicializar DS
                                
0115  33 DB                    xor     bx,bx           ; clear BX
0117  B9 0200                  mov     cx,SizeADCbuffer ; inicializar contador = Tamaño de buffer en MEM
011A  BF 0A00                  mov     di,ADCbuffer    ; offset de buffer de datos de A/D c. dentro de DS
                                
                        ; # de canal de A/D para leer informacion analogica :
= 0000                  Chn10   EQU     00h             ; channel 0 (direccion)
= 0001                  Chn11   EQU     01h             ; channel 1
= 0002                  Chn12   EQU     02h             ; channel 2
                                
011D                    AnalogRD:
                                ;Lectura Channel-0 (IN0) :
011D  B3 00                     mov     bl,Chn10        ; apuntar a channel 0
                                ;call   f_AD_Read       ; en EPROM  ( FE00:1240 )
011F  9A                        DB      9Ah             ; FAR call OpCode
0120  1240                      DW      1240h           ; IP
0122  FE00                      DW      0FE00h          ; CS
0124  88 05                     mov     BYTE PTR ds:[di],al ; escribir info. digital al buffer de MEM
                                ;Lectura Channel-1 (IN1) :
0126  47                        inc     di              ; apuntar a la siguiente loc. de memoria
0127  FE C3                     inc     bl              ; apuntar a channel 1
                                ;call   f_AD_Read       ; en EPROM  ( FE00:1240 )
0129  9A                        DB      9Ah             ; FAR call OpCode
012A  1240                      DW      1240h           ; IP
012C  FE00                      DW      0FE00h          ; CS
012E  88 05                     mov     BYTE PTR ds:[di],al ; escribir info. digital al buffer de MEM
                                ;Lectura Channel-2 (IN2) :
0130  47                        inc     di              ; apuntar a la siguiente loc. de memoria
0131  FE C3                     inc     bl              ; apuntar a channel 2
                                ;call   f_AD_Read       ; en EPROM  ( FE00:1240 )
0133  9A                        DB      9Ah             ; FAR call OpCode
0134  1240                      DW      1240h           ; IP
0136  FE00                      DW      0FE00h          ; CS
0138  88 05                     mov     BYTE PTR ds:[di],al ; escribir info. digital al buffer de MEM
013A  47                        inc     di              ; apuntar a la siguiente loc. de memoria
013B  E2 E0                     loop    AnalogRD
                                
013D  B7 00                     mov     bh,Chn10        ; bandera "channel-display"
013F  BE 1000                   mov     si,NChannelCycles ; fijar # de ciclos para despliegue de info. por ca
                        nal
0142  BA FF30                   mov     dx,PPort1A      ; (=FF30h) = PPortDAC
0145                    Out_DAC_Reset:
0145  B9 0200                   mov     cx,SizeADCbuffer ; (Tamaño buffer ADC= # definido por usuario)
```

```
0148  BF 0A00                          mov     di,ADCbuffer    ; (=0A00h)
014B  80 FF 00                        cmp     bh,00h          ; prueba para despliegue de info. CH0
014E  74 07                           jz      Out_DAC         ; iniciar ciclo de despliegue apuntando a "CH0 data"
0150  47                              inc     di              ; apuntar a "CH1 data"
0151  80 FF 01                        cmp     bh,01h          ; prueba para despliegue de info. CH1
0154  74 01                           jz      Out_DAC         ; iniciar ciclo de despliegue
0156  47                              inc     di              ; apuntar a "CH2 data"
0157                          Out_DAC:
0157  8A 05                           mov     al,BYTE PTR ds:[di] ; leer dato de RAM
0159  EE                              out     dx,al           ; mandar byte a D/A c
015A  83 C7 03                        add     di,0003h        ; inc DI ... apuntador en buffer RAM
                                                              ; apuntar al siguiente dato de CHx (channelX-data
                              )
015D  E2 F8                           loop    Out_DAC         ; para generar patron de onda

015F  4E                              dec     si              ; decrementar contador de 0-ciclo
0160  74 02                           jz      ChangeCH_display
0162  EB E1                           jmp     Out_DAC_Reset   ; reciclar buffer en memoria para generar
                                                              ; patron de onda periodica a la salida del DAC
0164                          ChangeCH_display:
0164  FE C7                           inc     bh              ; modificar bandera "channel-display"
0166  80 FF 03                        cmp     bh,03h
0169  75 02                           jnz     CH_flag_set     ; dentro de 0-channel valido
016B  B7 00                           mov     bh,Chnl0        ; reset bandera "channel-display"
016D                          CH_flag_set:
016D  BE 1000                         mov     si,NChannelCycles ; reinicializar contador de # de ciclos de despli
egue por canal
0170  EB D3                           jmp     Out_DAC_Reset   ; reciclar buffer en memoria para generar
                                                              ; patron de onda periodica a la salida del DAC

0172                          CodeSeg ENDS

                              ;

                              END     Code_block
```

Segments and Groups:

N a m e	Length	Align	Combine Class
CodeSeg	0172	PARA	NONE

Symbols:

N a m e	Type	Value	Attr
ADCbuffer	NUMBER	0A00	Global
AnalogRD	L NEAR	011D	CodeSeg Global
CH_flag_set	L NEAR	016D	CodeSeg Global
ChangeCH_display	L NEAR	0164	CodeSeg Global
Chnl0	NUMBER	0000	Global
Chnl1	NUMBER	0001	Global
Chnl2	NUMBER	0002	Global
Code_block	L NEAR	0100	CodeSeg Global
Counter1	NUMBER	FF01	Global
DSRAMVar	NUMBER	0030	Global
FreqScaleF	NUMBER	0002	Global
NChannelCycles	NUMBER	1000	Global
Out_DAC	L NEAR	0157	CodeSeg Global
Out_DAC_Reset	L NEAR	0145	CodeSeg Global
PITimerControlW	NUMBER	FF03	Global
PPortlA	NUMBER	FF30	Global
SizeADCbuffer	NUMBER	0200	Global
@Cpu	TEXT	0101h	
@FileName	TEXT	ADCCH012	
@Version	TEXT	510	

381 Source Lines
381 Total Lines
22 Symbols

47006 + 283564 Bytes symbol space free

0 Warning Errors
0 Severe Errors

```
Microsoft Cross-Reference  Version 5.10       Wed Oct 25 16:04:16 1989
BAUDRATEs en SDM88-PC _ Hardware Interface

  Symbol Cross-Reference            (# definition, + modification)      Cref-1

@Cpu . . . . . .  1#
@Version . . . . . .  1#

APtr_fIUART_X_MC . . . . . .  142   235#  365

BaudParams . . . . . .  128   211#  260   318

CodeSeg. . . . . .  170#  172   172   377
Code_block . . . . . .  132   185#  381
Counter0 . . . . . .  139   154#

DSEPROMVar . . . . . .  141   161#  297
DSRAMVar . . . . . .  141   163#  257   315
DisplayPort. . . . . .  140   158#  253

ESRAMVar . . . . . .  141   162#  300

PITimerControlW. . . . . .  139   155#
Prog_start . . . . . .  133   187   252#

TIME . . . . . .  138   241#  276+  286   333+  342
Tx_ChrF. . . . . .  134   279#  288
Tx_ChrN. . . . . .  135   336#  344

endBaudParams. . . . . .  131   223#  262   320

newBaudRateF . . . . . .  134   265#  294
newBaudRateN . . . . . .  135   323#  350


19 Symbols
```

```
                        ; Print Format : PAGE length,width
                          length=50 lines (default) (carta)
                                PAGE   50  , 132
                        ;*******************************************************************************
                        ; *
                        ; Programa :
                        ;
                        ; BAUDRATE.ASM
                        ;
                        TITLE   BAUDRATEs en SDM88-PC _ Hardware Interface
                        ;
                        ; ( BAUDRATE.ASM )
                        ;   Este archivo tiene un formato  ( *.COM )  (Segmento unico), en vez de uno
                        ;   ( *.EXE ) (PoliSegmentado).
                        ;   El archivo esta hecho para que su  ( *.COM )  generado, ( BAUDRATE.COM ) ,
                        ;   pueda cargarse directamente (sin modificaciones ni ajustes),
                        ;   en Memoria RAM del Sistema SDM88-PC , dentro de la zona destinada para
                        ;   programas de usuario.
                        ;
                        ;   ( CS:0100h )  es la localidad fijada por el formato  ( *.COM ) .
                        ;   Formato  ( *.COM ) :
                        ;   Requiere de la inclusion tanto del Codigo del programa como de los datos,
                        ;   dentro de un mismo y unico segmento y dar un offset inicial de  0100 h  ,
                        ;   pues los bytes 0 a FFh son ocupados por el    "Program Segment Prefix"
                        ;   asignado por la PC (at run time).
                        ;   Por eso, en este programa se define al inicio :  ORG 0100h
                        ;
                        ; Descripcion:
                        ; Este programa efectua un loop que transmite un caracter unico por el 8251A
                        ; a distintos baudajes: 300,1200,2400,4800,9600,19200.
                        ; La transmision se realiza cambiando sucesivamente el baudaje en intervalos
                        ; de 20 segundos.  Para esta temporizacion, se emplea la rutina   "NMIhandler"
                        ; residente en EPROM, que se actualiza cada 1/100[s] mediante la localidad de
                        ; memoria etiquetada con TIME.
                        ; Para la inicializacion del conjunto 8254-8251A y la transmision del byte,
                        ; se emplean primero las Subrutinas: "fIniUART_X_ModCad" y "fTransmit_Char"
                        ; que, residentes en EPROM, son invocandas como "Direct FAR calls",
                        ; (terminan con FAR ret (CBh)); ( transmision del byte : 7Fh ) .
                        ; Posteriormente, se transfieren de  EPROM  a  RAM , las Subrutinas:
                        ; "nIniUART_X_ModCad" y "nTransmit_Char" (version NEAR de las anteriores pues
                        ; terminan con NEAR ret (C3h)), y son invocadas desde RAM como "NEAR calls",
                        ; repitiendo el ciclo anterior con baudaje diferente cada 20 [s]; ( transmision
                        ; del byte : 4Fh ) .
                        ; Terminado el ciclo, se re-inicializa la razon de baudaje (Tx/Rx) del SDM88-PC
                        ; a 300 Bauds, ajustando el factor de escalamiento en el 8254 y programando al
                        ; 8251A, invocando de nuevo la subrutina "fIniUART_X_ModCad" residente en EPROM,
```

```
                        ; como "Indirect FAR call".
                        ; Finalmente, el control se regresa a la rutina principal de control del
                        ; SDM88-PC : "Manager Routine" , dejando al sistema preparado para recibir
                        ; alguna nueva orden a traves del puerto serie de la PC.
                        ;
                        ; Programa para operar sobre el hardware del sistema digital :
                        ;                    "   ( SDM88-PC )   "
                        ; basado en el Microprocesador 8088  ( @ 4.77 MHz )  y en una Microcomputadora
                        ; Personal IBM-compatible.
                        ; Periodo de operacion experimental = Ts = 210 [ns].
                        ; SDM88-PC :
                        ; " (S)istema de (D)esarrollo basado en el (M)icroprocesador (8088)
                        ;   y en una microcomputadora personal tipo IBM (PC) "
                        ;
                        ; Notas:
                        ; Los codigos de operacion de las instrucciones de este programa que a su vez
                        ; conformaran el contenido a ser cargado en RAM de SDM88-PC, se observaran
                        ; directamente a traves del Debugger "CODEVIEW" , mientras que el archivo
                        ;                                BAUDRATE.LST
                        ; se puede usar tan solo como auxiliar en la corroboracion de dichos codigos y
                        ; como referencia completa del codigo fuente.
                        ;
                        ; Este programa NO funciona en un ambiente microcomputadora IBM PC, pues se
                        ; refiere a direcciones especificas de un hardware externo a la PC.
                        ;
                        ; Desarrollado con : Microsoft Macro Assembler ..... Version : 5.10
                        ;                    for the MS-DOS Operating System.
                        ;
                        ; Microprocesador  : intel 8088
                        ;
                        ; Registros usados   : AX,BX,CX,DX,SI,DI,SP,CS,DS,ES,SS,IP,Flags (BP no usado)
                        ; Puertos I/O usados : Counter0,Counter2,UARTdata,UARTControlStat,DisplayPort
                        ; Subrutinas usadas  :
                        ;               (NEAR) :
                        ;                 nTransmit_Char
                        ;                 nIniUART_X_ModCad
                        ;               (FAR) :
                        ;                 fTransmit_Char
                        ;                 fIniUART_X_ModCad
                        ; Rutinas de Servicio a Interrupcion :
                        ;                 NMIhandler
                        ;
                        ;
                        ; Autor   :   Agustin Eduardo Alvarez Vaca
                        ; Fecha   :   Mayo/28/1989
                        ; Version :   2.0
```

```
                        ;
                        ;
                        ; To convert from  *.EXE  to  *.COM  file format
                        ; Therefore:
                        ; *    Only one segment can be used  (max.64kBytes)
                        ; *    The align, combine and class types need not be given, since they make
                        ;      no difference for single-module  *.COM  programs
                        ; *    All segment register are initialized to the same segment by using the
                        ;      ASSUME directive.  This tells the Assembler (not the processor) which
                        ;      segment to associate with each segment register
                        ; *    The  ORG  directive must be used to start assembly at byte 256 (0100h)
                        ;      This leaves room for the DOS Program Segment Prefix, which is automa-
                        ;      tically loaded into PC's memory at run time
                        ; *    Although any program data must be included in the single segment, it
                        ;      must not be executed.  You can use the JMP instruction to skip over
                        ;      data (as shown here) or you can put the data at the end after the pro-
                        ;      gram returns to DOS:  using DOS function:      mov  ax,4C00h
                        ;                                                     int  21h
                        ;      in case of a PC program, or after the program control returns to main
                        ;      control Routine ( Manager Routine ) in SDM88-PC digital system.
                        ;
                        ; See: Chapter 1  ..... Microsoft Macro Assebler 5.0     Programmer's Guide
                        ;                                                        (*.COM)  format
                        ;*****************************************************************************


                        ; ***  DECLARACIONES GLOBALES  ***

                        ; Procedures :
                        ;NEAR
                        ;PUBLIC         nTransmit_Char
                        ;PUBLIC         nIniUART_X_ModCad
                        ;FAR
                        ;PUBLIC         fTransmit_Char
                        ;PUBLIC         fIniUART_X_ModCad
                        ; Variables :  ( locs.MEM referidas por nombre )
                        PUBLIC          BaudParams
                        ; Labels :
                        ; Etiquetas para flujo de Proceso
                        PUBLIC          endBaudParams
                        PUBLIC          Code_block
                        PUBLIC          Prog_start
                        PUBLIC          newBaudRateF,Tx_ChrF
                        PUBLIC          newBaudRateN,Tx_ChrN
                        ;PUBLIC         Prg_LoadAutoexec
                        ; Constants :
```

```
                        PUBLIC      TIME
                        PUBLIC      Counter0,PITimerControlW
                        PUBLIC      DisplayPort
                        PUBLIC      DSEPROMVar,ESRAMVar,DSRAMVar
                        PUBLIC      APtr_fIUART_X_MC

                        ;

                        ; *** CONSTANTES ***

                        ; PARAMETROS : Direcciones del sistema dentro del mapa de MEMORIA y de PUERTOS I/O
                        ;              usadas por este programa para manejo del Hardware de "SDM88-PC"

                        ; Direcciones de Puertos I/O

                        ; 8254       Programmable Interval Timer
FF00 =                  Counter0    EQU     0FF00h          ; Generador de Baudaje para (USART 8251A)
FF03 =                  PITimerControlW EQU 0FF03h          ; 8254  Control Word

                        ; 74LS373
FF70 =                  DisplayPort EQU     0FF70h          ; OUT -Indicador : Display status

                        ; Direcciones de Memoria para las variables del programa ( Prog_Data )
FE00 =                  DSEPROMVar  EQU     0FE00h          ; Data segment UVEPROM  =  OpSysSeg
0030 =                  ESRAMVar    EQU     0030h           ; Extra segment (transfer:UVEPROM->RAM)_data
0030 =                  DSRAMVar    EQU     0030h           ; Data segment RAM  (00300->00FFF) 3328 Bytes instal
                        ados

                        ; ***  SEGMENTO  ***

                        ; Programa para el sistema Digital "SDM88-PC"

0000                    CodeSeg     SEGMENT                 ; definir Code Segment

                                    ASSUME  cs:CodeSeg , ds:CodeSeg

                                    ; fijar contador de localidad (IP) en 0100h (formato archivo (*.COM))

0100                                ORG     0100h

                        ; NO se genera archivo *.COM en PC, si en el programa se declaran etiquetas
                        ; o subrutinas como "FAR" ( FAR LABELs o FAR PROCs ), usar en cambio
                        ; NEAR LABELs,  NEAR CALLs/JMPs  y cuando se requieran FAR CALLs/JMPs , usar
                        ; directivas "DB" y "DW" para definir Bytes y Words que representen
                        ; los OpCodes, Offsets (IP) y Segment-bases (CS) de estas instrucciones,
```

```
                        ; (como se muestra en este programa).
                        
                        Code_block:
                                    nop
0100 90                             jmp     SHORT Prog_start ; saltar sobre area de datos (definida dentro de seg
0101 EB 3D              mento unico)
                                                             ; a area de codigo ejecutable (programa)
                        
                        ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                        ; DATA ->
                        
                        ; Tabla USART & Prog.Timer :  para subr.: (nIniUART_X_ModCmd) & (fIniUART_X_ModCmd)
                        ; Baud Rate * freq.BR Gen.* Scale Factor * BR factor *          8251A              *
                        ; BR=(bps)  |  8254 (CH0) | PCLK/freq.BR |   8251A   | MODE word | COMMAND word |
                        ; (bit/s)   |     (Hz)    |  -> ( AX )   |           | -> ( BH ) | -> ( BL )   |
                        ;           |             |              |           |           |             |
                        ;    300          19200          007C h        64 X        4F h          37 h
                        ;   1200          19200          007C h        16 X        4E h          37 h
                        ;   2400          38400          003E h        16 X        4E h          37 h
                        ;   4800          76800          001F h        16 X        4E h          37 h
                        ;   9600         595238.1        0004 h        64 X        4F h          37 h
                        ;                (9300.6)  ... (3.12% error) ... OK
                        ;  19200         297619.05       0006 h        16 X        4E h          37 h
                        ;                (18601.2) ... (3.12% error) ... OK
                        ;
                        ; Base freq.= (8088 CLK)/2 = PCLK = 2.3809524 (MHz) ... (8284A PCLK output)
                        ;
0110                                OFFSET  0110h
                        ; Parametros de Baudaje para inicializar USART 8251A & Prog.Timer 8254
0110 007C               BaudParams  DW      007Ch           ; 8254 Timer Scale Factor (AX) ( 300 Bauds )
0112 4F37                           DW      4F37h           ; 8251A : Mode w. (BH) & Command w. (BL)
0114 007C                           DW      007Ch           ; ... ( 1200 Bauds )
0116 4E37                           DW      4E37h
0118 003E                           DW      003Eh           ; ... ( 2400 Bauds )
011A 4E37                           DW      4E37h
011C 001F                           DW      001Fh           ; ... ( 4800 Bauds )
011E 4E37                           DW      4E37h
0120 0004                           DW      0004h           ; ... ( 9600 Bauds )
0122 4F37                           DW      4F37h
0124 0008                           DW      0008h           ; ... ( 19200 Bauds )
0126 4E37                           DW      4E37h
0128                    endBaudParams LABEL BYTE            ; Siguiente pos.de byte despues de tabla BaudParams
                                                            ; dar nombre a esta loc.mem.(byte)
                                                            ; etiqueta usada para referenciar la posicion de un
                        dato
```

```
                        ;Apuntadores de Direccion (vectores)
                        ;Tabla de Subrutinas FAR (a ser invocadas como FAR Indirect CALLs).
                        ;AddrPtr_FARsub :
                        ;(Las siguientes constantes (EQU) representan Offsets de localidades de memoria
                        ; dentro de "DS=0030h" donde los "IP" y "CS" de estas subrutinas, son almacenados
                        ; durante la rutina de INICIALIZACION de SDM88-PC).
                        ;APtr_fTx_Char    EQU   00D8h            ; fTransmit_Char     apuntador de direccion
                        ;                 DW    10A0h,0FE00h     ; IP,CS
= 00DC                  APtr_flUART_X_MC EQU   00DCh            ; flniUART_X_ModCmd  apuntador de direccion
                        ;                 DW    1190h,0FE00h     ; IP,CS
                        ; ( Ver FAR indirect CALL a "flniUART_X_ModCmd" en este programa (al final) )
                        ; ***

                        ;CONSTANTES :
= 00FC                  TIME            EQU     00FCh           ; Offset en DS (RAM) de TIME, loc.Mem. que
                                                                ; almacena la variable de control del sistema
                                                                ; para mantener la cuenta "Real-Time-Clock (RTC)"
                                                                ; usada por la Rutina de Servicio a
                                                                ; Interrupcion NMI en EPROM
                        ; <- DATA
                        ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


0140                                    ORG     0140h

0140                    Prog_start:
0140  BA FF70                          mov     dx,DisplayPort  ; display port output (status)
0143  B0 FE                           mov     al,0FEh         ; desplegar "t" mientras prog. este activo
0145  EE                              out     dx,al

0146  BB 0030                         mov     bx,DSRAMVar     ; (=0030h) por Phys.address 00300h
0149  8E DB                           mov     ds,bx           ; SDM88-PC  area RAM

014B  8D 36 0110 R                    lea     si,BaudParams   ; offset de una loc.MEM. (apuntador "SI")
                                      ; or :  mov     si,OFFSET BaudParams ; cargar direccion de "BaudParams"
014F  BF 0128 R                       mov     di,OFFSET endBaudParams ; offset de una loc.MEM. (apuntador "DI")
                                      ; or :  lea     di,endBaudParams     ; cargar direccion de "endBaudParams"

0152                    newBaudRateF:
0152  57                              push    di                      ; salvar Baud-Rate-Param. (apuntador: end-of-ta
                        ble)
0153  56                             push    si                      ; salvar Baud-Rate-Param. (apuntador dentro de
                        tabla)
0154  8B 04                          mov     ax,WORD PTR ds:[si]   ; (Freq.Scale factor) 8254 -> AX
0156  8B 5C 02                       mov     bx,WORD PTR ds:[si+2] ; (Mode & Command words) 8251A -> BH,BL = BX
                                    ; call    FAR PTR flniUART_X_ModCmd  ; 8254 & 8251A init. (subrutina en EPROM)
```

```
0159 9A                                 DB      9Ah             ; far CALL OpCode
015A 1190                               DW      1190h           ; IP (offset)  .... fIniUART_X_ModCmd
015C FE00                               DW      0FE00h          ; CS (Seg-base)

015E BB FB30                            mov     bx,0FB30h               ; factor para duracion de 20 [s]
0161 B9 1E 00FC                         mov     WORD PTR ds:[TIME],bx ; usando NMIhandler (Int.Serv.Routine)
                                                                        ; (duracion para la prueba a un baudaje X)

0165                            Tx_ChrF:
0165 B0 7F                              mov     al,7Fh          ; Caracter_dato a transmitir
                                ; call  FAR PTR fTransmit_Char  ; 8251A subrutina Txmit (en EPROM)
0167 9A                                 DB      9Ah             ; far CALL OpCode
0168 10A0                               DW      10A0h           ; IP (offset)  .... fTransmit_Char
016A FE00                               DW      0FE00h          ; CS (Seg-base)

016C A0 00FD                            mov     al,BYTE PTR ds:[TIME+1] ; Real Time Clock es actualizado
016F 0A C0                              or      al,al                   ; cada 1/100 [s] por NMIhandler
0171 78 F2                              js      Tx_ChrF

0173 5E                                 pop     si
0174 83 C6 04                           add     si,4            ; apuntar a sig. loc.Mem. "Baud-Rate-parameters"
0177 5F                                 pop     di
0178 3B F7                              cmp     si,di
017A 72 D6                              jb      newBaudRateF    ; repetir procedimiento Tx a un nuevo Baudaje

                                ; Transferir rutina "nTransmit_Char" desde EPROM a RAM
017C BB FE00                            mov     bx,DSEPROMVar   ; (=FE00h)   (Source)      (EPROM)
017F 8E DB                              mov     ds,bx           ;                ) Phys.addr.= FF080h -) nTransmit_Char
0181 BE 1080                            mov     si,1080h        ; (=offset)                             NEAR PROC version
0184 BB 0030                            mov     bx,ESRAMVar     ; (=0030h)   (Destination) [RAM]
0187 8E C3                              mov     es,bx           ;          ) Phys.addr.= 00B00h
0189 BF 0B00                            mov     di,0B00h        ; (=offset)
018C B9 0010                            mov     cx,0010h        ; 10h bytes a transferir
018F FC                                 cld                     ; clear "direction flag" para autoinc. SI & DI
0190 F3/ A4                             rep     movsb           ; transferir bytes desde (DS:SI) a (ES:DI)
                                ; Transferir rutina "nIniUART_X_ModCmd" desde EPROM a RAM
                                                                ; (=FE00:1120)   (Source)      [EPROM]
0192 BE 1120                            mov     si,1120h        ; Phys.addr.= FF120h -) nIniUART_X_ModCmd
                                                                ; (=0030:0B20)   (Destination) [RAM]
0195 BF 0B20                            mov     di,0B20h        ; Phys.addr.= 00B20h
0198 B9 0050                            mov     cx,0050h        ; 50h bytes a transferir
019B FC                                 cld                     ; clear "direction flag" para autoinc. SI & DI
019C F3/ A4                             rep     movsb           ; transferir bytes desde (DS:SI) a (ES:DI)

019E BB 0030                            mov     bx,DSRAMVar     ; (=0030h) por Phys.address 00300h
01A1 8E DB                              mov     ds,bx           ; SDM88-PC  area RAM
```

```

 01A3  8D 36 0110 R                 lea     si,BaudParams    ; offset de una loc.MEM. (apuntador "SI")
                                    ; or :  mov     si,OFFSET BaudParams ; cargar direccion de "BaudParams"
 01A7  BF 0128 R                    mov     di,OFFSET endBaudParams ; offset de una loc.MEM. (apuntador "DI")
                                    ; or :  lea     di,endBaudParams     ; cargar direccion de "endBaudParams"

 01AA                      newBaudRateN:
 01AA  57                           push    di                      ; salvar Baud-Rate-Param. (apuntador "end-of-ta
                           ble")
 01AB  56                           push    si                      ; salvar Baud-Rate-Param. (apuntador dentro de
                           tabla)
 01AC  8B 04                        mov     ax,WORD PTR ds:[si]   ; (Freq.Scale factor) 8254 -> AX
 01AE  8B 5C 02                     mov     bx,WORD PTR ds:[si+2] ; (Mode & Command words) 8251A -> BH_BL = BX
                                  ; call    nIniUART_X_McdCmd  ; 8254 & 8251A init. (subrutina en RAM)
 01B1  E8                           DB      0E8h            ; near CALL OpCode
 01B2  066C                         DW      066Ch           ; desplazamiento 16-bit ... nIniUART_X_ModCmd
                                                            ; 0820-01B4 = 066Ch
 01B4  BB F830                      mov     bx,0F830h          ; factor para duracion de 20 [s]
 01B7  89 1E 00FC                   mov     WORD PTR ds:[TIME],bx ; usando NMIhandler (Int.Serv.Routine)
                                                            ; (duracion para la prueba a un baudaje X)

 01BB                      Tx_ChrN:
 01BB  B0 4F                        mov     al,4Fh          ; Caracter_dato a transmitir
                                  ; call    nTransmit_Char  ; 8251A Txmit (subrutina en RAM)
 01BD  E8                           DB      0E8h            ; near CALL OpCode
 01BE  0640                         DW      0640h           ; desplazamiento 16-bit ... nTransmit_Char
                                                            ; 0800-01C0 = 0640h
 01C0  A0 00FD                      mov     al,BYTE PTR ds:[TIME+1] ; Real Time Clock es actualizado
 01C3  0A C0                        or      al,al                 ; cada 1/100 [s] por NMIhandler
 01C5  78 F4                        js      Tx_ChrN

 01C7  5E                           pop     si
 01C8  83 C6 04                     add     si,4            ; apuntar a sig. loc.Mem. "Baud-Rate-parameters"
 01CB  5F                           pop     di
 01CC  3B F7                        cmp     si,di
 01CE  72 DA                        jb      newBaudRateN    ; repetir procedimiento Tx a un nuevo Baudaje

                                  ; Reinicializacion de 8254 & 8251A para 300 Bauds Tx/Rx

 01D0  B8 007C                      mov     ax,007Ch        ; (Freq.Scale factor) 8254 -> AX
 01D3  BB 4F37                      mov     bx,4F37h        ; (Mode & Command words) 8251A -> BH_BL = BX
                                                            ; 8254 Timer Scale Factor (AX) ( 300 Bauds )
                                                            ; 8251A : Mode w. (BH) & Command w. (BL)
                                  ; Far Direct CALL :
                                  ; call    FAR PTR fIniUART_X_ModCmd  ; 8254 & 8251A init. (subrutina en EPROM)
                                  ; DB      9Ah             ; far CALL OpCode
```

```
                                ; DW      1190h           ; IP (offset)  .... flniUART_X_ModCad
                                ; DW      0FE00h          ; CS (Seg-base)

                                ; Far Indirect CALL :
                                  mov     si,APtr_flUART_X_MC
01D6 BE 00BC                      call    DWORD PTR ds:[si]
01D9 FF 1C

                                ; jmp     FAR PTR Prg_LoadAutoexec ; Regresar a "SDM88-PC Manager Routine"
                                                                   ; dejando al sistema preparado para
                                                                   ; recibir una nueva orden (en EPROM)
01DB EA                           DB      0EAh            ; far JUMP OpCode
01DC 0800                         DW      0800h           ; IP (offset)  .... Prg_LoadAutoexec
01DE FE00                         DW      0FE00h          ; CS (Seg-base)

                                ; ¡ el area de DATOS pudo haberse colocado aqui !

01E0                    CodeSeg   ENDS

                                ;

                                  END     Code_block
```

Segments and Groups:

Name	Length	Align	Combine	Class
CodeSeg	01E0	PARA	NONE	

Symbols:

Name	Type	Value	Attr
APtr_fIUART_X_MC	NUMBER	00DC	Global
BaudParams	L WORD	0110	CodeSeg Global
Code_block	L NEAR	0100	CodeSeg Global
Counter0	NUMBER	FF00	Global
DSEPROMVar	NUMBER	FE00	Global
DSRAMVar	NUMBER	0030	Global
DisplayPort	NUMBER	FF70	Global
ESRAMVar	NUMBER	0030	Global
endBaudParams	L BYTE	0128	CodeSeg Global
newBaudRateF	L NEAR	0152	CodeSeg Global
newBaudRateN	L NEAR	01AA	CodeSeg Global
PITimerControlW	NUMBER	FF03	Global
Prog_start	L NEAR	0140	CodeSeg Global
TIME	NUMBER	00FC	Global
Tx_ChrF	L NEAR	0165	CodeSeg Global
Tx_ChrN	L NEAR	01B8	CodeSeg Global
@Cpu	TEXT	0101h	
@FileName	TEXT	BAUDRATE	
@Version	TEXT	510	

307 Source Lines
307 Total Lines
23 Symbols

47100 + 283471 Bytes symbol space free

0 Warning Errors
0 Severe Errors

```
Microsoft Cross-Reference  Version 5.10       Wed Oct 25 14:09:44 1989
ADCCH012 ( A/D c. Test ) _ SDM88-PC

  Symbol Cross-Reference            (# definition, + modification)      Cref-1

@Cpu . . . . . .  1#
@Version . . . . . .  1#

ADCbuffer. . . . . .  155   181#  236   275
AnalogRD . . . . . .  150   243#  268

CH_flag_set. . . . . .  152   296   298#
ChangeCH_display . . . . . .  152   290   293#
Chn10. . . . . .  156   239#  245   270   297
Chn11. . . . . .  156   240#
Chn12. . . . . .  156   241#
CodeSeg. . . . . .  198#  200   303
Code_block . . . . . .  149   212#  307
Counter1 . . . . . .  158   172#  225

DSRAMVar . . . . . .  154   180#  231

FreqScaleF . . . . . .  159   190#  226

NChannelCycles . . . . . .  156   187#  271   299

Out_DAC. . . . . .  151   277   280   282#  287
Out_DAC_Reset. . . . . .  151   273#  291   300

PITimerControlW. . . . . .  158   171#  222
PPort1A. . . . . .  157   175#  272

SizeADCbuffer. . . . . .  155   184#  235   274


 20 Symbols
```

CAPITULO V

EMPLEO DE LAS MICROCOMPUTADORAS Y SISTEMAS DIGITALES DE DESARROLLO BASADOS EN MICROPROCESADORES EN LA CREACION DE APLICACIONES DE CONTROL E INSTRUMENTACION DE PROPOSITO ESPECIFICO

APLICACION DEL SISTEMA DE DESARROLLO SDM88-PC EN LA CREACION DE UN INSTRUMENTO PROTOTIPO ANALIZADOR DIGITAL DE RESPUESTA EN FRECUENCIA

INTRODUCCION

En este capítulo se hará una breve descripción del uso de las microcomputadoras y sistemas de desarrollo basados en microprocesador, para el control de procesos o para la creación de instrumentos-prototipo concebidos para dar solución a una necesidad específica.

Se hablará de las consideraciones básicas en el diseño de un sistema cualquiera, del balance *Hardware-Software* , y de algunos métodos o aproximaciones que pueden ayudar al desarrollo de un Prototipo, haciendo énfasis en la descripción de las características y ventajas que para ello ofrece el Sistema de Desarrollo que diseñamos, **SDM88-PC** .

Finalmente se mostrará, como uno de múltiples ejemplos de aplicación que pueden concebirse tomando como *Hardware* base o herramienta inicial de desarrollo al sistema **SDM88-PC** , la elaboración de un instrumento prototipo :

" Analizador Digital de Respuesta en Frecuencia "

que permite conocer el comportamiento en Magnitud y Angulo de Fase de la Función de Transferencia de un sistema (circuito analógico) cualquiera.

Se hará una breve descripción de los métodos de "Respuesta en Frecuencia" y de las representaciones comúnmente utilizadas de funciones de transferencia senoidales; se analizará la base teórica-matemática del algoritmo de funcionamiento del instrumento diseñado, así como el *Hardware* y *Software* de control requeridos.

5 . 1
CONTROL DE PROCESOS

5.1.1
APLICACION DE LAS MICROCOMPUTADORAS EN EL CONTROL DE PROCESOS

Las Ciencias de la Computación, en términos tanto de *Hardware* como de *Software*, han avanzado a una fase en donde ahora es factible usar esta tecnología en la operación de plantas piloto de pequeña escala así como en plantas de producción de gran escala.

Al emplear computadoras es posible extender el control del proceso más allá de las simples mallas cerradas de control de parámetros tales como temperatura, pH, presión, flujo, carga, voltaje, etc., a estrategias más sofisticadas de control interactivo. Eventualmente, será posible realizar procesos de optimización, en base a modelos dinámicos, al utilizar una estrategia de control interactiva. La combinación de la tecnología de la computadora y el modelado dinámico en las manos del ingeniero, está conduciendo a una gran eficiencia y alta productividad en los procesos. A fin de realizar la optimización del proceso, es necesario desarrollar un sistema que incluya tanto el *Hardware* como el *Software* requeridos para generar los datos iniciales, analizar la información resultante, determinar soluciones óptimas e implantar decisiones de control. Un sistema apoyado en una computadora es capaz de realizar estas tareas.

Existen diversas formas o aproximaciones en que las computaroras digitales son usadas en aplicaciones de control, destacando las siguientes:

Aproximaciones para Control auxiliado por Computadora Digital

1) : **Control Digital Directo** (DDC) :

La computadora digital reemplaza a los controladores analógicos del sistema de control convencional y ejecuta un programa que contiene a la ecuación controladora para dicho sistema. Asi, al menos en teoria, muchos controladores analógicos pueden ser reemplazados por un programa de computadora apropiado.

Un ejemplo de esto se tiene con los Filtros Digitales, que pueden realizar las funciones de los Filtros Analógicos correspondientes (Paso-Bajas , Paso-Altas , Paso-Banda o Supresores de Banda de frecuencia) , pero efectuando el filtrado de una señal, mediante la toma de muestras de dicha señal con un convertidor A/D, realizando operaciones matemáticas sobre las muestras adquiridas, y entregando el resultado como salida a través de un convertidor D/A. Esta aproximación puede producir fácilmente una curva de respuesta que es dificil, si no imposible de producir, con circuiteria analógica. Esta aproximación digital tiene además la ventaja de que la respuesta del filtro puede ser alterada o cambiada bajo control de programa (*Software*). (Lo mismo vale cuando se habla de la función de transferencia de cualquier otro sistema de control).

Los filtros digitales son usados en sintetizadores de voz, sistemas de mejoramiento y refuerzo de señal para imagen satélite, procesamiento de imágenes y muchas otras aplicaciones.

2) : **Control Supervisor o de "Setpoint"** (DSC) :

En esta acción, un programa de computador maestro constantemente calcula y actualiza los *Setpoints* de los controladores analógicos en base a una estrategia de operación predeterminada. El hardware de la computadora digital ajusta los *Setpoints* de los controladores

analógicos para mantener la operación de una planta en algún nivel óptimo.

Por ejemplo, monitoreo y supervisión de las distintas variables físicas que intervienen en los procesos de regulación y control de una planta Termoeléctrica, de una planta Termonuclear, de un Oleoducto, de una Refinería Petrolera, etc., con el fin de mantener un nivel óptimo de funcionamiento y eficiencia, alertar a los operadores de posibles fallas y ayudar a la toma de decisiones correctivas de manera automática o manual emergente.

5.1.2

LAS VARIABLES DE UN PROCESO

Al tratar el control de procesos, es útil distinguir las variables físicas de acuerdo a la siguiente clasificación:

1) : **Salidas :**

Son las variables clave del proceso que serán mantenidas a un valor deseado o conveniente, esto es, controladas.

2) : **Entradas :**

Son las variables que, cuando cambian, causan que una o más salidas cambien.

A su vez, las Entradas pueden clasificarse como :

a) : **Entradas de Control :**

Estas variables son cambiadas por el controlador para llevar y mantener las salidas en su valor *Setpoint* . También son conocidas como "Variables Manipuladas".

b) : **Entradas de Perturbacion :**

Son todas las otras variables del proceso que afectan las salidas en cualquier forma.

Generalmente causan cambios indeseados en las salidas del proceso. También son conocidas como "Cargas del proceso".

5.1.3
MODELADO DE PROCESOS

En la actualidad se emplean modelos matemáticos dinámicos para describir procesos. Algunos procesos son *lineales*, de modo que pueden ser descritos por ecuaciones algebraicas o diferenciales lineales, y por tanto, es aplicable el principio de *Superposicion*: ("La respuesta de un sistema lineal producida por varias excitaciones actuando simultáneamente es igual a la suma de las respuestas que produce en el sistema cada excitación actuando por separado", {las condiciones iniciales deben considerarse también como excitaciones}). Muchos otros procesos son *NO lineales* y son descritos por ecuaciones diferenciales no lineales complejas. Sin embargo, el comportamiento de éstos últimos alrededor de algún nivel de operación normal, puede ser aproximado por ecuaciones diferenciales lineales. Esta aproximación es usualmente aplicable, pues en la práctica, en los procesos que están bajo control, el interés se centra en su comportamiento alrededor de algún nivel de operación de estado estable.

El conocimiento de la *FUNCION DE TRANSFERENCIA* de un proceso, es también necesario para la implantación de estrategias de control avanzadas. La Función de Transferencia de un proceso puede ser desarrollada a partir de un análisis teórico o de pruebas experimentales en una planta.

Es importante considerar, que el estado de un proceso puede ser descrito por algunas de sus variables y que ellas frecuentemente interactúan unas con otras, es decir, las variables

que deben ser controladas no son independientes. Este tipo de control es llamado *Multivariable* , y aunque es mucho más complejo que el caso usual de variables independientes, es particularmente apropiado para Procesamiento Digital. El número de variables independientes tales como : temperatura, presión, concentración, etc., que pueden ser controladas independientemente en un proceso dado, no está limitado pero es una función del número total de variables posibles y de las ecuaciones que las relacionan. La diferencia entre el número de variables y las ecuaciones que las relacionan, es llamada *Grados de Libertad del Sistema* .

5.1.4

NIVELES DE CONTROL

Existen tres niveles de control de proceso que deben ser incorporados en un sistema. Cada nivel más alto involucra un grado más alto de sofisticación en la programación y requiere más conocimiento de la dinámica del proceso.

El **Primer Nivel de Control** involucra operaciones secuenciales, tales como manipulación de válvulas o arranque y paro de bombas asociadas con el sistema de encendido, procesos de alarma, secuencias automáticas o semiautomáticas, recalibración de instrumentos y procedimientos de "falla-salvamento" con la información obtenida y/o procesada. Los datos iniciales se analizan para realizar las conversiones necesarias que pueden incluir factores de escala, comparación con curvas de calibración, incorporación de factrores de corrección o conversión de unidades. Se debe verificar el valor de los datos contra condiciones de alarma; si se presenta una situación de alarma, deben enviarse algunos mensajes al operador junto con instrucciones para tratar con el problema. El sistema también debe incluir procedimientos automáticos para el manejo de fallas de potencia. Una falla de

potencia de corto periodo puede deshabilitar el sistema de cómputo, mientras que una falla de largo periodo puede afectar el proceso. Con el fin de proteger la integridad del sistema de cómputo, una función de restablecimiento por falla de potencia permite al sistema suavizar la salida para fluctuaciones de potencia de corto periodo, sin ninguna pérdida aparente de servicio o de datos. En el caso de una falla de potencia de largo periodo, la computadora o el sistema de control digital (basado en microprocesador), debe iniciar un procedimiento para llevar al proceso a un estado de "falla-salvamento". Dicho procedimiento puede implementarse a partir del recurso de la Interrupción No Mascarable (NMI) , (disponible en todos los microprocesadores), y que puede ser activada (disparada) mediante circuitos que monitoreen la continuidad, niveles, frecuencia, etc., del suministro de energía eléctrica (o de otro tipo), cuando se detecte una condición anormal de falla. El Microprocesador central derivará entonces el curso del programa a una rutina de Servicio a la Interrupción No Mascarable, en donde deberán indicarse las acciones a seguir para protección de la integridad del *Software* del sistema y los *Datos* del proceso (guardar en disco, en memoria no volátil EEPROM o en memoria tipo CMOS respaldada por batería la información importante para el proceso), así como del *Hardware* del sistema (aterrizar en zona segura las cabezas R/W de discos duros, aislar etapas de circuitería particularmente sensibles a daño durante transitorios, etc.). Cuando la potencia es restablecida la computadora lleva al proceso a su estado de trabajo y proporciona instrucciones especiales para que el operador verifique el funcionamiento apropiado del sistema completo. Para la mayoría de estas operaciones, la base de tiempo está en el orden de minutos u horas, así que manipulaciones a alta velocidad no son críticas para el proceso, aunque existen excepciones, como en el caso de los sistemas de control que integran un Transbordador espacial, por citar algún ejemplo. La mayor ventaja que un sistema controlado por computadora digital

ofrece sobre un sistema tradicional de *timers* y relevadores, es la facilidad con la cual, la secuencia lógica, puede ser cambiada tanto en su formato como en su base de tiempo, para acomodar mejorias en sus procedimientos de operación.

El **Segundo Nivel de Control** involucra manipulación de los parámetros del proceso en mallas cerradas individuales. Existen dos conceptos básicos para el control de parámetros de procesos: *Control Digital Directo* (DDC) y *Control Digital de Setpoint* (DSC). Determinar una justificación económica de un método sobre el otro es un problema complicado pues no existe una respuesta general ya que la solución depende de circunstancias particulares. La comparación económica por costo del *Hardware* es simplemente el costo de N controladores-grabadores con sus correspondientes *Interfaces* , contra el costo de N unidades de mantenimiento de señal y la porción de la computadora destinada para DDC. Así el costo depende del número de mallas de control. En la mayoria de los casos, el costo de un sistema DDC respaldado por controladores, será mayor que el de un sistema DSC. Sin embargo, en muchas aplicaciones, el costo no es la única consideración. En general, DDC puede proporcionar mejor control de un proceso dado que los algoritmos de control son funciones matemáticamente almacenadas en vez de una representación eléctrica. Por lo tanto, para un proceso continuo de gran escala, lo que aparenta ser una pequeña mejoria en el control del proceso, resulta en un mayor ahorro debido a la mejoria en la operación de la planta y a la versatilidad y flexibilidad en la modificación de parámetros (por *Software*) según requerimientos dinámicos convenientes.

El **Tercer Nivel de Control** se refiere al incremento en la productividad y optimización del proceso. Es en este nivel, en el que el sistema controlado por computadora es más valioso, ya que ésta es usada para monitorear muchos parámetros de entrada y para implementar decisiones de control en base a la dinámica del

proceso. El primer paso es la Adquisición de datos, que incluye proceso de sensado, registro y almacenamiento de datos. El segundo paso involucra la interpretación de datos al utilizar la información generada para elaborar la dinámica del sistema. El tercer paso es la realización de tareas para análisis de datos con respecto a los procesos especificados que pueden involucrar la resolución de fórmulas, ecuaciones algebraicas relativamente simples o ecuaciones diferenciales más complejas. Un sistema de control apoyado en una computadora digital, reduce el tiempo requerido para determinar la dinámica de un proceso, probar estrategias de control y efectuar el modelado del proceso. Finalmente, desde que los sistemas con computadora son capaces de realizar simultáneamente tareas de adquisición de datos y ejecución de programas de simulación de procesos en base a modelos dinámicos, el ingeniero de desarrollo de procesos puede rápidamente refinar un modelo matemático e identificar los parámetros del modelo, de modo que puedan desarrollarse métodos para predecir las relaciones entre las variables del proceso.

5 . 2

DESARROLLO DEL PROTOTIPO DE UN INSTRUMENTO DE PROPOSITO ESPECIFICO CONTROLADO DIGITALMENTE POR MICROCOMPUTADORA (MICROPROCESADOR)

El primer paso en el desarrollo de un nuevo instrumento es definir muy cuidadosa y exactamente la función que se desea realice el instrumento. El siguiente paso es decidir qué partes del instrumento se desea hacer en *Hardware* y qué partes se desea hacer en *Software*. Es entonces cuando el diseñador podrá decidir cómo quiere o cómo le conviene realizar cada una de estas partes.

Para el *Software*, conviene dividir el total de las Tareas de Programación en módulos que puedan ser probados y depurados (*debug*) individualmente.

Para el *Hardware*, existen diferentes aproximaciones entre las que el diseñador puede elegir :

a) : Diseño Asistido por Computadora (CAD = *Computer-Aided Design*)

b) : Diseño usando un *Emulador* para el Microprocesador deseado

c) : Diseño usando un *Sistema Digital de Desarrollo* basado en algún microprocesador (*Microcomputer-Prototyping Board*) ... como el Sistema **SDM88-PC** .

5.2.1

DISEÑO ASISTIDO POR COMPUTADORA

Una aproximación para crear el *Hardware* necesario para el prototipo de un instrumento es con un Sistema CAD o de Diseño Asistido por Computadora. Este sistema puede ser desde una poderosa Estación de Trabajo (*Workstation*) para Ingeniería con microcomputadoras equipadas con el microprocesador intel 80386 conectadas en Red de Area Local (LAN) con minicomputadora VAX 11/780 en una configuración que soporta varios usuarios, como las construidas por Compañías como *Mentor Graphics Corp.*, *Valid Logic Systems Inc.*, *Daisy Systems Corp.*, *Metalogic Inc.*, *Lattice Logic Ltd.*, *Silicon Compilers Inc.*, *California Institute of Technology*, *Fujitsu Ltd.*, *AT&T Bell Laboratories*, etc., o bien una simple microcomputadora tipo IBM-PC o IBM-PS con programas tales como el *PCAD system* desarrollado por *Personal CAD Systems Inc.*, *Electronic Design Automation Division*. Los programas en estos sistemas permiten, primero que todo, diseñar y dibujar fácilmente un diagrama esquemático para el *Hardware*. Se puede seleccionar por número u otros medios, el simbolo esquemático de un circuito que se desee incluir en una determinada parte del diseño global, "trayéndolo" a la pantalla de la terminal o microcomputadora, de una extensa libreria de dispositivos standard o comunes contenida en uno o varios archivos de disco. Se usa un Manejador Gráfico o *Mouse* para mover el simbolo a la posición deseada y para trazar líneas de señal que lo conecten con otros simbolos. Se puede mover según se requiera el dispositivo dibujado, y las lineas de conexión lo seguirán. Existen distintos módulos en una librería de dispositivos entre los que destacan : Módulo SSI (*inversores, buffers, registros, conectores, Flip-Flops, Transistores Pull Up&Down, Compuertas de 2 ó mas entradas, buffers 3-state, Data bus, PAL, std.PLA, etc.*) ; Módulo MSI (*Contadores Sincronos, Contadores en Anillo (ripple), Sumador, Registros de Corrimiento, MUXs, DECs, FLAIR (Folded PLA) para implementacion de cualquier*

funcion logica, etc.) ; Módulo LSI (*Memorias diversas ROM, RAM, registros FIFO, etc.)* ; Módulo VLSI (*Microprocesadores, Perifericos, etc.*).

Cuando se ha concluido el proceso de Edición Gráfica y el dibujo del esquemático, se genera un archivo llamado "Listado de Red" (*Net-List*), que contiene la información a cerca de qué terminales de los distintos componentes están unidas con qué otras. Es entonces cuando se puede usar otro programa en el sistema de CAD para "simular" la operación del circuito. Por simular se quiere decir "ejecutar o correr" el circuito en *Software*. Esta etapa del diseño ayuda a determinar si las señales están conectadas correctamente y si los parámetros de temporización y retardos son aceptables. El usuario del sistema puede definir y colocar puntos de prueba (*test points*) en cualquier parte del circuito para análisis de las formas de onda (como en un analizador lógico). En un simulador lógico interactivo como el TAD (*Themis Architectural Design*), cada corrida puede detenerse y reiniciarse, se puede asignar un número de retardos a cada salida de compuerta además de la definición arbitraria de puntos de prueba. En los Simuladores, la velocidad es un factor critico por lo que se dispone de Aceleradores de Hardware o "Procesadores Aritméticos de Enteros" que pueden conectarse en cascada o bien de Computadoras de Propósito especifico como *IBM's Logic Simulation Machine* o *HAL* (*Hardware Logic*) de NEC Corp.. Otro parámetro que debe considerarse en el Hardware-Software para Simulación es la gran cantidad de memoria requerida. Si el circuito es muy extenso y la memoria del sistema es poca, se efectúan *disk-swappings* en detrimento de la velocidad de simulación. Cuando por su reciente creación o gran complejidad no se dispone de modelos de determinados Circuitos Integrados, por ejemplo algunos microprocesadores o circuitos de aplicación muy especifica, se han implementado sistemas con adaptadores para la inserción de dichos IC's, en los que el simulador alimenta con sus

señales a los dispositivos reales y transfiere las salidas de estos chips a la parte del circuito que está siendo simulada; ejemplos de estos sistemas son el *Realchip* (Valid Logic Systems Inc.) y el *PMX* (*Physical Modeling Extension*) (Daisy Systems Corp.). Para circuitos analógicos o híbridos existen simuladores analógicos, como *Spice* (University of California - Berkeley) o *Simon Simulator* (de Simon Software), con los que se puede simular Hardware de comunicaciones, capacitores de conmutación (switcheo), Transistores, etc., y cuentan con una librería de chips analógicos. En estos simuladores los resultados pueden desplegarse en el dominio del tiempo o de la frecuencia y en Amplitud, Fase y Retardo de Grupo.

Si el circuito pasa satisfactoriamente las pruebas de Simulación, se puede generar una impresión del esquemático en una impresora (*printer*) o graficador (*plotter*).

El siguiente paso consiste en el diseño de la "tarjeta de Circuito Impreso" para el circuito en el que se ha trabajado. Otro programa en el sistema CAD, con algo de ayuda por parte del usuario, producirá el "trabajo de arte" (*Artwork*) o "ruteado de pistas de conexión", para la tarjeta de circuito impreso. Incluso, existen sistemas que generan la cinta (*tape*) de control para la máquina que automáticamente hace las perforaciones requeridas sobre la tarjeta de circuito impreso; esto es, los resultados previos del diseño, se convierten en las "entradas" directas de un Lenguaje de Programación de Control Numérico (que después genera las Instrucciones de Máquina para controlar el perforado de la tarjeta y en general la Manufactura del circuito impreso y la tarjeta electrónica terminada).

Próximamente se espera que la Estación de Trabajo en Ingeniería pueda conectarse directamente a la máquina generadora de la tarjeta de circuito impreso, a la máquina que obtiene las

partes del almacén, a la máquina que integra, posiciona y suelda los circuitos integrados en la tarjeta , y a la máquina que realiza las pruebas iniciales de funcionamiento en la tarjeta. Este concepto es llamado "Manufactura Integrada por Computadora" (*Computer Integrated Manufacturing*) o CIM y parece ser la tendencia actual de la industria.

Después de que se ha armado la tarjeta con las partes requeridas, puede ésta energizarse y realizarse un chequeo para detectar componentes anormalmente calientes o defectuosas. Si no existen problemas aparentes, se puede proceder a probar la tarjeta con ayuda de Osciloscopios, Analizadores Lógicos, Frecuencímetros, Multímetros, Generadores de Funciones, etc., pero probablemente la mejor herramienta para probar la tarjeta sea usando un Emulador.

Resumiendo:

El módulo de operador en una Estación de Trabajo (*Workstation*) tiene cuatro submódulos separados:

1) : **EDITOR GRAFICO** (base de datos para dibujar símbolos y construir módulos circuitales) que conducen al **ESQUEMATICO** (base de datos del diseño).

2) : **SIMULADOR LOGICO** (verificación del diseño lógico y diagrama de tiempos).

3) : Base de Impresión (*footprint*) (base de conexiones y ruteado para tarjeta de circuito impreso).

4) : Geometría Real del *Layout* (base de datos del circuito impreso en dimensiones físicas reales)

Estos cuatro submódulos son almacenados en una base de datos (depósito electrónico de datos o información), desde donde pueden ser accesados por otras utilerías de la Estación de Trabajo.

El proceso de diseño completo consta de los siguientes pasos:

1) : Usuario - Editor Gráfico - construcción de Esquemático ... < Pantalla de terminal CRT > + < Mouse >

2) : *WIRE Command* (interconexión de módulos y dispositivos traídos de librerías

3) : Formación de Base de Datos del Diseño (*Net-List*):

- Validación del dibujo
- Generación del programa de prueba
- Prueba de las primeras partes

4) : Simulación Lógica

Para comprobación del funcionamiento, el programa provee los modelos para los módulos interconectados. Los dispositivos traídos de Librerías vienen con sus propios modelos (algoritmos) de software.

5) : Dispositivos complejos no incluidos en Librerías (ejem. algunos Microprocesadores), son simulados a través de una extensión basada en Hardware donde se conectan físicamente esos chips (*Realchip Hardware Extension*). *Workstation* puede combinar modelos basados en *Hardware* y *Software* en la misma simulación.

6) : Verificación interactiva del diseño : el usuario suministra el tipo de estímulo y el simulador lógico responde reportando la reacción del circuito.

Establecimiento de Tolerancias en diagramas de tiempo por simulación (pueden agregarse o eliminarse niveles de retardo).

7) : Si el circuito no responde como se espera, el usuario regresa al Editor Gráfico, Modifica, Recompila y Reestimula al circuito hasta que funcione como se desea.

8) : Posicionamiento de Módulos y otros circuitos (automáticamente por compilador).

9) : Ruteado de interconexiones entre módulos.

10) : Evaluación de efectos de capacitancia (parásita) y otros parámetros de las interconexiones (pistas).

11) : Creación automática del patrón de perforaciones en cinta (Control Numérico), para la futura tarjeta de circuito impreso.

12) : Producción de la Máscara de Circuito Impreso.

13) : Pruebas de *Hardware* y *Software*.

Finalmente conviene aclarar, que aunque aquí se ha hecho referencia al término CAD para designar a todo el proceso automatizado de Diseño con computadora, lo correcto es identificar a las distintas etapas de este proceso bajo distintos conceptos; así, se tiene que :

CAE (*Computer-Aided Engineering*)

- Captura de Esquemático (Modelos físicos y lógicos)
- Librerías de circuitos standard (archivos que soportan el símbolo y el funcionamiento de cada componente)
- Generación de *Net-List* (Archivo que registra las interconexiones entre los distintos elementos) {Base de Datos fundamental}.
- Simulación (generación de estímulos para obtener salidas susceptibles de ser analizadas).
  { Información tomada del *Net-List* }.

CAD (*Computer-Aided Design*)

- Sólo soporta dibujo o graficación (no diseño lógico ni eléctrico) (ej.: circuitos impresos, piezas mecánicas, etc.).
- Es el "puente de unión" entre CAE y CAM.

CAM (*Computer-Aided Manufacturing*)

- Sistemas y Técnicas automatizadas de Manufactura : Control Numérico, Control de Procesos, Robótica, Planificación de Necesidades Materiales (MRP).
- Fabricación, Procesos de Manufactura, posicionamiento y soldado de componentes dentro de tarjeta de circuito

impreso.

- Recepción de la información del *Net-List* para generar y realizar perforado para componenetes (*drill*) fisicamente sobre el Cobre de la tarjeta de circuito impreso.

5.2.2

DISEÑO USANDO UN EMULADOR PARA EL MICROPROCESADOR DESEADO (Emulator)

Una segunda aproximación para crear y probar el *Hardware* necesario para el prototipo de un instrumento es empleando un Emulador. Existen varias Compañias que hacen Emuladores para lòs distintos Microprocesadores que hay en el mercado; por ejemplo el Emulador de 16-bit *Applied Microsystems* ES-1800 de *Applied Microsystems Corp.* que cuenta con conectores para insertar al microprocesador 8086-8088 y al microprocesador 80186-80188, y que trabaja con la microcomputadora IBM-PC o PS y otras compatibles.

El *Hardware* de un Emulador consta de: Circuiteria de Control, Memoria para almacenar el estado (*status*) del sistema y los datos rastreados (*trace*), después de la ejecución de cada instrucción, y un cable "umbilical" (de interface) con un conector al final del mismo.

Para usar el Emulador, el usuario debe remover el microprocesador del instrumento o unidad prototipo bajo prueba, e insertar en su lugar, el conector del cable umbilical (interface). El Emulador contiene un microprocesador que realmente ejecutará (correrá) los programas de prueba creados por el usuario siempre bajo control del Emulador. Entonces, el Emulador da al usuario una "Ventana" para analizar el Estado y Operación de la circuiteria del instrumento-prototipo bajo control de un sistema de Desarrollo o Microcomputadora (PC,PS).

El *Software* del Emulador es similar a un poderoso programa Monitor o Depurador (*Debugger*). El usuario puede emplear los comandos del Emulador y la memoria del sistema para analizar y probar cada una de las partes del prototipo. Por ejemplo, se puede escribir un pequeño programa para probar la RAM en el prototipo, cargar este programa dentro de la RAM del sistema y ejecutar (correr) el programa bajo supervisión (control) del Emulador. Para ayudar al diseñador en el proceso de depuración (detección de errores *{debugging}*), los Emuladores permiten la inserción de "puntos de ruptura" (*Breakpoints*) en el flujo del programa, examinar y cambiar el contenido de los Registros del Microprocesador y de las Localidades de Memoria, y realizar un proceso de "Rastreo" (*trace*) que muestra el contenido de los Registros después de que se ejecuta cada instrucción de un programa residente en memoria. Algunos Emuladores tienen una característica adicional como aquella presente en los Analizadores Lógicos, a fin de que el usuario pueda realizar un rastreo de la secuencia de señales de *Hardware* en un grupo de líneas para revisar temporización, retrasos, sincronización, etc. .

Recientemente *intel* ha creado sistemas Emuladores bastante poderosos como:

a) : El sistema I^2ICE (*Sistema Integrado de Instrumentacion y Emulacion en circuito*) provee ayuda para el desarrollo de sistemas que utilizan los microprocesadores 8088/8086, 80188/80186 y 80286. Soporta programas escritos en "C", PL/M, FORTRAN, Pascal y lenguaje Ensamblador. Hasta 4 sistemas I^2ICE pueden ser soportados por una sola IBM PC o PS. Entre sus características se cuentan: Emulación en tiempo real; Depuración simbólica de programas (acceso a localidades de memoria o variables usando nombres definidos por programa, manteniendo una tabla de símbolos virtuales); Capacidad de Rastreo múltiple de la ejecución; Permite controlar hasta 4

microprocesadores con un solo sistema base; Provee desensamblado de código y ensamble de una sola linea para corrección de programas de linea; Capacidad de añadir al chasis del sistema tarjetas de memoria de alta velocidad; Software para el sistema base I^2ICE, Software para el Emulador y Software para Pruebas. El sistema puede ser basado en una microcomputadora personal IBM PC o PS con 512 kB de memoria y dos unidades de diskette de doble densidad o un disco duro.

b) : El sistema ICE-386 , Emulador en circuito provee una herramienta sofisticada para el desarrollo de *Software* y *Hardware* para sistemas basados en el microprocesador 80386 . Entre otras muchas de sus caracteristicas destacan: Emulación en tiempo real hasta 16 MHz del 80386; Modificación de los Registros y memoria por el usuario; Ensamble y desensamble del código en mnemónicos-80386; etc.. Este sistema puede basarse en una IBM PC-AT o PS con 2 MB de RAM, disco duro 20 MB, una unidad de diskette e interface serial.

5.2.3

DISEÑO USANDO UN SISTEMA DIGITAL DE DESARROLLO BASADO EN MICROPROCESADOR (como el Sistema SDM88-PC)

(Microcomputer-Prototyping Board)

Una tercera aproximación para crear y probar el *Hardware* necesario para el prototipo de un instrumento es empleando un "Sistema de Desarrollo basado en algún microprocesador" o "Microcomputadora para la creación de instrumentos-prototipo". Estos *Kits* para Desarrollo y Evaluación de sistemas son Microcomputadoras completas en una sola tarjeta. Incluyen CPU, Memoria RAM y ROM, Puertos de I/O (Entrada/Salida), Teclado, Exhibidores numéricos, áreas de expansión para el usuario,

manuales de operación y programación.

Una ventaja de esta aproximación es que estos sistemas dan al diseñador la arquitectura básica de CPU (Microprocesador), Memoria RAM y ROM, y Puertos I/O (Entrada/Salida), ya probada y funcionando. Entonces, a partir del conocimiento y empleo de esa arquitectura fundamental, el diseñador puede fácilmente añadir los periféricos requeridos por su diseño (timers, controladores jerárquicos de interrupción, controladores de comunicación avanzados, coprocesadores numéricos, sintetizadores digitales de voz o imagen, convertidores A/D y D/A, memoria dinámica o más memoria estática y/u otra circuiteria de interface).

Algunos de los Sistemas de Desarrollo disponibles cuentan con Programas Monitor que permiten al usuario cargar y ejecutar sus programas, examinar contenido de Registros, establecer puntos de ruptura dentro de los programas, etc..

La mayor ventaja de esta aproximación, es que permite al diseñador obtener rápidamente un prototipo armado y funcionando para ver si el instrumento es "factible" de realizarse. Si el instrumento es factible, entonces puede diseñar una tarjeta (autónoma) con el *Hardware* especifico (*custom*) que se ajuste exactamente a sus necesidades.

En el mercado, existen disponibles varios Sistemas de Desarrollo basados en algún microprocesador comercial: Z-80 (*Zilog*), serie 6800 o serie 68000 (*Motorola*) o serie 80XXX (*intel*). De *intel* destacan por ejemplo, los sistemas SDK-51 (microcontroladores 8031 y 8032), SDK-85 (microprocesador 8085A) y SDK-86 (microprocesador 8086), y recientemente otros basados en los microprocesadores 80286, 80386 y ¡ 80486 ! a nivel experimental.

5.2.4

DISEÑO USANDO EL SISTEMA DIGITAL DE DESARROLLO SDM88-PC

(Sistema de Desarrollo basado en el Microprocesador 8088 y en una Microcomputadora PC (o PS) IBM compatible)

(Microcomputer-Prototyping Board)

(Ventajas con respecto a Sistemas disponibles en el mercado).

Como Sistema de Desarrollo, el **SDM88-PC** , provee al diseñador de una arquitectura base (CPU (Microprocesador), Memoria RAM y EPROM, y Puertos I/O (Entrada/Salida)), ya probada y funcionando, para que a partir de su conocimiento y empleo, pueda fácilmente añadir los periféricos requeridos por su diseño , y disponga de las herramientas necesarias para probar y depurar las secciones de *hardware* que vaya agregando al aprovechar las facilidades que el sistema ofrece para escribir y ejecutar módulos de *software* que manejen o actúen sobre los módulos de *hardware*.

Los sistemas de Desarrollo disponibles en el mercado, cuentan normalmente con un puerto de comunicación como el "RS-232C" o el "20 mA Current loop", sin embargo, dentro de su programa monitor o sistema operativo, no proveen al usuario de un protocolo de comunicación para que pueda conectarse, de una forma natural, a sistemas de cómputo mayores, (por ejemplo, una microcomputadora tipo PC o PS con unidades de disco para almacenamiento de programas, monitor que permita visualizar bloques de datos, teclado para transmitir comandos o iniciar una acción, más memoria disponible, un respaldo muy amplio de *Software*, etc., etc.) para un aprovechamiento directo de sus recursos, durante el proceso de construcción, pruebas, depuración y evaluación del circuito diseñado (armado inicialmente en el área de expansión del *kit* de desarrollo).

Por esta razón, los Sistemas de Desarrollo (no Emuladores) existentes (al menos los disponibles en el mercado), pueden

considerarse prácticamente autónomos y hasta cierto punto difíciles de enlazar con sistemas mayores (microcomputadoras) para la mayor parte de los usuarios, considerando que el fabricante no proporciona una información detallada (listado) del *Software* de control o Programa Monitor del sistema, de modo que se puedan implementar rutinas residentes en el ROM básico de control para enlazar de forma automática y natural al *Hardware* del Sistema de Desarrollo con una Microcomputadora.

En el Sistema de Desarrollo **SDM88-PC** , el enlace con la Microcomputadora para hacer uso de los recursos que ofrece, es parte fundamental en el funcionamiento y operación del sistema, y no una aplicación adicional (poco documentada) que establece un nexo "artificial" sólo factible, tal vez, a través de *Software* Emulador de Terminal para un tipo determinado de Microcomputadora (de dudoso desempeño para todos los casos posibles). Así, en el sistema **SDM88-PC** , la Microcomputadora no sólo funciona pasivamente como una terminal para despliegue de información, sino que interviene activamente como administradora del *Hardware* externo (interface), canalizando comandos para efectuar acciones específicas, manejando y organizando la información recibida (código, datos, programas (bytes) para el microprocesador externo), almacenándola / recuperándola en/de su memoria o dispositivos de almacenamiento masivo (unidades de diskette o de disco duro), o bien arbitrando la transferencia selectiva de bloques de información (bytes) desde o hacia la interface (memoria y/o puertos I/O externos), entre otras muchas actividades supervisadas y seleccionadas por el usuario a través de pantallas ordenadas en "Menúes". La Microcomputadora es pues, a través de su teclado, el dispositivo por el cual el usuario, desencadena y sincroniza acciones en la interface (Hardware externo), organiza y transfiere información (bidireccionalmente) y visualiza resultados (status).

En el sistema de Desarrollo **SDM88-PC** , la Microcomputadora y la Interface (Hardware Externo), funcionan como una **unidad integral.**

Es en este aspecto donde el sistema de Desarrollo que diseñamos: SDM88-PC manifiesta una clara ventaja en *VERSATILIDAD* (almacenamiento, visualización, organización de comandos, etc.), *COMODIDAD* y *FACILIDAD* (de manejo y control) y *ALCANCES* tales como la posibilidad del empleo de *Software* relacionado, (constantemente desarrollado y actualizado para microcomputadora tipo IBM-PC,PS), para la creación de programas que a su vez puedan correrse directamente en el sistema **SDM88-PC** {Ensambladores}, o el empleo de programas auxiliares depuradores del *Software* del usuario (fuera de linea, a nivel del sistema operativo de la PC) {Debuggers} , o bien para la elaboración de programas que tomen control absoluto del *Hardware* externo del sistema durante la ejecución de una aplicación especifica creada por el usuario {Lenguajes de Alto y Medio Nivel, o combinaciones de Lenguaje de Máquina con Lenguaje de Alto Nivel vía códigos objeto y con un Linker, etc.}, como se mostrará posteriormente con el "Analizador Digital de Respuesta en Frecuencia", donde la interface (Hardware externo) del SDM88-PC pasa a ser un satélite o una extensión del hardware de la microcomputadora para el control de procesos digitales y/o analógicos que mediante un enfoque modular, conforman un sistema de instrumentación dedicado a la solución de una necesidad particular.

El Sistema SDM88-PC puede ser basado en una Microcomputadora personal PC/XT, PC/AT o PS , IBM compatible (HP,Compaq,Wise,Printaform ,Corona,Olivetti,etc.), con 640 kBytes de memoria RAM, sistema operativo DOS versión 2.0 ó superior, y por lo menos una unidad de diskette (doble densidad {360kB} 5¼"; alta densidad {720 kB} 5¼" o 3½" ó {1.44 MB} 3½") o bien una unidad de disco duro {*Hard disk*}. El microprocesador

central de la Microcomputadora puede ser : 8088, 8086, 80188, 80186, 80286 u 80386, ya que existe compatibilidad en código objeto entre todos los miembros de la familia de microprocesadores 8086/88 de *intel* , lo que da acceso a la base de software más amplia del mundo (por la simbiosis comercial *intel-IBM*).

La familia de microprocesadores *intel* 8086/88, inicia con los microprocesadores de 8/16 bits 8088 y el 80188, de alta integración. La línea de 16 bits incluye al 8086, (que junto con el 8088) es estándar en la industria, el altamente integrado 80186 y el poderoso microprocesador de alto rendimiento 80286, con capacidades de multiproceso y manejo de memoria. La más reciente adición disponible en el mercado, el 80386, es un microprocesador de 32 bits CHMOS de alta eficiencia, que ofrece soporte a multitareas, unidad de manejo de memoria integrada y paginación. Finalmente, está el 80486, con bus externo e interno de 64 bits, que a la fecha de este escrito, ya existe pero está en etapa experimental.

Para la Interface (Hardware externo) en el sistema SDM88-PC , se eligió a la familia *intel* por la amplia línea de periféricos, controladores y circuitos integrados avanzados, disponibles para sus microprocesadores y por ser una Compañía con bastante respaldo en constante desarrollo y evolución. A pesar de que la familia *Motorola* tiene también dispositivos muy avanzados y eficientes, *intel* es la familia seleccionada por IBM (tal vez por razones históricas eventuales), para sus arquitecturas en computadoras personales. *IBM* ha creado un estándar mundial para este tipo de equipos a nivel comercial por lo que, con vistas a la estandarización de un nuevo equipo desarrollado (SDM88-PC), es conveniente seguir la línea *intel-IBM*, al menos por el momento.

Entonces, debido a la selección de un microprocesador de *intel* para el diseño del sistema SDM88-PC , se garantiza la

compatibilidad en código con la familia de microcomputadoras IBM compatibles, fácilmente asequibles en el mercado.

Dentro de la familia de microprocesadores *intel*, se eligió al 8088 que es un microprocesador con una arquitectura externa (bus externo) de 8 bit pero una arquitectura interna (bus interno) de 16 bit. La razón de su existencia en el mercado con este canal de datos (8 bit), es establecer una continuidad entre el 8086 de 16 bit en su canal de datos (arquitectura externa), y los antiguos procesadores de 8 bit fabricados por *intel*, el 8080 y el 8085. Esta continuidad es especialmente importante para aquellos que han desarrollado sistemas basados en este tipo de productos. El 8088, teniendo el mismo tamaño de canal de datos, puede reemplazar a uno de esos primeros procesadores de 8 bit (incluso al Z-80 muy parecido al 8085), en un sistema ya existente, sin necesidad de hacerle grandes cambios en *Hardware*, aunque un poco más (no demasiados) en *Software*. De esta manera, se aumenta la capacidad del sistema en cuanto al número de operaciones que realiza y en cuanto al almacenamiento de datos, con un costo muy bajo.

Cabe mencionar también que, para la mayoría de las aplicaciones dedicadas de control industrial, es suficiente un microprocesador de 8 bit (cuando no un microcontrolador como el 8031/8032 o el 8051/8052), dada la alta especificidad de las tareas a supervisar.

Además, ya que la subfamilia 8086/8088 comprende bajo un mismo concepto a los microprocesadores 8086 (16 bit) y 8088 (8 bit), (y su conjunto completo de dispositivos de soporte (periféricos)), y debido a que el 8086 y el 8088 tienen la misma estructura interna, son totalmente compatibles en *Software* y pueden utilizar los mismos componentes de soporte. Esto permite un intercambio completo de programas entre sistemas de 8 y 16 bit. Esta subfamilia ha logrado aceptación generalizada y es compatible con la línea completa de coprocesadores de *intel* (XXX87).

La Interface (Hardware externo a la Microcomputadora en el Sistema de Desarrollo **SDM88-PC** , consta de:

- Microprocesador 8088 (CPU)
- Circuito Temporizador para el microprocesador y sus periféricos (generador-sincronizador de RESET, WAIT y CLOCK) : 8284
- Circuito para introducción de Estados de Espera en el sistema
- Sistema de Buffers para Bus de Datos, Direcciones y Control
- Circuitos decodificadores para memoria y puertos de Entrada/Salida
- Memoria EPROM que contiene sistema operativo para la interface, subrutinas definitivas para control de periféricos, rutinas de servicio a interrupción y datos (tablas de apuntadores) iniciales del sistema ; (8kB) : 2764
- Memoria RAM (1kB) para uso del sistema y (5kB) para empleo del usuario (6kB RAM instalados pero expandible hasta 16 kB) : 2016
- Circuito Temporizador-Contador (para generación de baudajes, reloj ADC, u otros usos) : 8254
- USART 8251A para la comunicación serie entre interface y microcomputadora
- Controlador de Interrupciones : 8259
- Puertos Paralelos de Entrada/Salida 8255A-2 : uno para manejo de DAC y ADC y otro para empleo del usuario (3 puertos paralelos programables por chip)
- Convertidor Digital/Analógico (MC1408) de 8 bits
- Convertidor Analógico/Digital (ADC0809) de 8 bits (8 canales multiplexados)

La Interface (Hardware externo) del sistema es manejada por una microcomputadora a través del puerto serie de la misma, bajo las especificaciones de la norma RS-232C .

Al ser controlado por una microcomputadora, el sistema **SDM88-PC** está diseñado para aprovechar los recursos que ésta ofrece, entre los que destacan :

- almacenamiento de programas creados por el usuario en disco para su posterior ejecución en el sistema de desarrollo

- facilidades de edición y modificación de código (programas) a través de la interacción con un teclado y una pantalla que permite una visión global del trabajo realizado con dicho código, así como de contenido de memoria RAM (y EPROM) del sistema, lectura de puertos de Entrada/Salida, revisión del contenido de un disco (programas almacenados: directorio)

- revisión del "status" presente del sistema y capacidad de cambiar la velocidad de Transmisión/Recepción entre interface y microcomputadora : (300, 1200, 2400, 4800 y 9600 Bauds)

- ejecución de comandos (para modificar el modo de operación de la interface, revisar contenidos de memoria y puertos I/O y ejecución de programas de control desarrollados por el usuario usando el hardware de la interface), desde el teclado de la microcomputadora y bajo una filosofía de "Menúes", haciendo más amigable y autoexplicativa la interacción usuario-sistema

- control absoluto del usuario sobre los recursos de la interface a través del teclado y con la retroalimentación visual que ofrece la pantalla, así como la existencia de mecanismos detectores de errores (por software) que impiden su introducción al sistema y mecanismos de "inicio de ejecución" y "cancelación" de comandos que dan al usuario

control sobre las decisiones y los tiempos de accionamiento para efectos de una eventual sincronización requerida

- enlace con Ensambladores de Microprocesador 8088-8086 comerciales con la inherente facilidad de trabajar los programas en lenguaje de máquina a nivel de mnemónicos en vez de a nivel de códigos de operación hexadecimales, y, dado que dichos mnemónicos se introducen en un archivo creado por un programa editor, se pueden crear programas fuente documentados, con todos los titulos y comentarios que el usuario considere pertinentes para una fácil identificación posterior; todo esto permite también la creación de programas (o subrutinas) bajo una filosofía modular, pudiéndo asi posteriormente unir módulos para la creación de programas de control para una aplicación especifica, sin tener que reescribir código desarrollado anteriormente

Para el proceso de diseño conviene dividir el *hardware* en módulos, armar y dar de alta dichos módulos de uno en uno a la vez, creando paralelamente, un *software* de prueba para el manejo y/o análisis de las secciones de *hardware* que se vayan añadiendo.

5.3
DEFINICION DE UN SISTEMA O INSTRUMENTO CONTROLADO DIGITALMENTE POR MICROCOMPUTADORA (MICROPROCESADOR)

5.3.1
DEFINICION DEL SISTEMA

La "interface" y configuración de los sistemas e instrumentos de control auxiliados por computadora, son muy dependientes del uso final del sistema. Por ejemplo, los requerimientos para una planta piloto son muy diferentes de los de una planta de producción o de los de un instrumento para Ciencia o Ingeniería.

En una planta piloto, el sistema es utilizado principalmente como una herramienta de investigación. En una planta de producción, la computadora es usada para monitorear el proceso, controlarlo y ayudar en la optimización al mantener el control en la calidad y la uniformidad del producto. Mientras que en un instrumento autónomo, el objetivo se centra en el análisis y/o medición de procesos o eventos puntuales, como auxiliar en la investigación y desarrollo tecnológico o bien en la docencia.

Es un error tratar de incorporar estas tres funciones en un mismo sistema de aplicación. Estas funciones deben ser claramente definidas desde el comienzo, con el fin de diseñar un sistema que funcione como una **unidad integral** .

5.3.2

CONSIDERACIONES EN EL DISEÑO DE UN SISTEMA

La utilización de las señales procedentes del mundo real (del proceso) a través de sensores, transductores, ciruitos digitales y analógicos, etc., y su integración en el sistema controlado por computadora, involucra desarrollo tanto de *Hardware* como de *Software* .

La configuración del sistema en *Hardware* básicamente involucra la selección de módulos electrónicos compatibles. La selección y desarrollo del *Software* para análisis de datos y control de procesos, son mucho más complicados.

Durante la configuración de un sistema completo, *Hardware* y *Software* deben ser estructurados para operar como una **unidad integral** . Más aún, el diseño debe ser lo suficientemente flexible para incorporar crecimiento y expansión tanto en *Hardware* como en *Software* .

El *Hardware* del sistema consta de tres secciones principales :

1) Sensores e instrumentación :
(circuitería digital y/o analógica de adquisición, transducción y acondicionamiento)

2) "Interface" de la computadora :
Interconexión entre elementos de *Hardware*, de *Software* y Seres Humanos.

a) Interface de *Hardware* : es la circuitería y trayectorias físicas que deben conectar e intercambiar señales electrónicas en un orden preestablecido

b) Interface de *Software* : es el conjunto de programas

de control y administración del *hardware* y los protocolos y mensajes específicos establecidos entre los distintos módulos de código (subrutinas) o programas

c) Interface *Hombre-Maquina* (*People/Machine Interface*) :
es el método de interacción entre una persona y la computadora. Existen dos formas de estas interfaces : (1) la Forma Visual , que aparece en una pantalla de video o en un informe impreso (reportes de texto y cantidades numéricas, simbolos, imágenes, reportes gráficos, ventanas, colores o sombreados, flechas indicadoras, etc.), y (2) la Forma Interactiva , que es la conversación entre el usuario y la computadora desde una terminal ("Sistemas Amigables con el Usuario": empleo de "Menúes" para muestra de alternativas o "Pantallas de Ayuda" que puedan exhibirse en cualquier momento, etc.)
Existen varios dispositivos de *Hardware* que pueden hacer más cómoda la interface *Hombre-Maquina* : (Pantallas fotosensitivas al tacto, Teclado con definición de teclas funcionales, *Mouse* (Ratón), *Light Pen* (Pluma de luz), mesa digitalizadora, *Joy stick* (mando de palanca), etc., como elementos de dirección y de entrada.

3) Sistema de Cómputo :
Sistema (Máquina) Programable para el Procesamiento de información .
El concepto engloba a :
la Computadora (CPU), todos los dispositivos periféricos unidos a ella (Terminales, Discos, Cintas, Impresoras, Plotters, etc.) y el Sistema

Operativo (programa de control principal que maneja y administra el medio ambiente de la computadora). *Microcomputadoras* , *Minicomputadoras* , *Mainframes* , elegidas de acuerdo a la carga total de trabajo y rendimiento requeridos.

La "Interface" de la computadora involucra una elección entre flexibilidad en *Hardware* o incrementar los requerimientos de *Software* .

Ningún sistema de cómputo puede operar sin *Software*, esto es, sin el conjunto de instrucciones almacenadas en su memoria o *programas*, necesarios para ejecutar tareas dentro del, y con el sistema.

El *Software* puede ser clasificado en tres categorias :

- Programa Administrador (Sistema Operativo de la Computadora) : es el programa de control principal que supervisa y determina la operación del sistema de cómputo en todo momento, encargándose de la ejecución de comandos, control de tareas, métodos de acceso (*I/O*), canalización de información y administración de los recursos de *Hardware* del sistema.

- Programas de Aplicación : son programas para realizar tareas de propósito especifico (buscando satisfacer una necesidad con el manejo y procesamiento de la información), que hacen uso de los recursos y funciones principales del sistema.

- Programas de Soporte del Sistema : son programas que auxilian al usuario en el desarrollo de programas de aplicación, éstos incluyen :

a) Lenguaje del Procesador (Ensambladores) y *Linkers* (convierten programas en lenguaje de Alto Nivel a programas en Lenguaje de Máquina).

b) Editores (facilitan la creación o modificación de programas escritos por el usuario).

c) Depuradores (*Debuggers*) (ayudan a la localización de errores en la lógica de los programas del usuario con características como : ejecución de programas paso a paso, rastreo (*tracing*) de variables y datos, despliegue del estado (*status*) del procesador, establecimiento de puntos de ruptura (*breakpoints*) en el flujo del programa, modificación del valor de variables en línea, etc.).

Es conveniente recordar algunos puntos importantes cuando se consideran alternativas para el diseño de la "interface" de computadora y la configuración del sistema :

- Es importante **definir claramente los objetivos del sistema** ya que los requerimientos para una planta piloto son muy diferentes de los de una planta de producción o de los de un instrumento autónomo de análisis y/o medición.

- A continuación, es necesario **seleccionar las funciones que son requeridas y especificar la computadora e "interface" que realizará estas tareas.** Esto involucra comparar las alternativas "*Hardware-Software*" tanto desde el punto de vista económico como operacional.

- Finalmente, la más importante consideración, es que *Hardware* y *Software* **deben funcionar como una UNIDAD INTEGRAL.** Así, el diseño tanto de *Hardware* como el de *Software*, deben ser evaluados juntos, con el fin de desarrollar un apropiado diseño global del sistema.

5 . 4
APLICACION DEL SISTEMA DE DESARROLLO SDM88-PC EN LA CREACION DE UN INSTRUMENTO PROTOTIPO : ANALIZADOR DIGITAL DE RESPUESTA EN FRECUENCIA

5.4.1
DEFINICION DEL INSTRUMENTO DE APLICACION : OBJETIVOS Y ESTRATEGIA DE DISEÑO HARDWARE-SOFTWARE

Para ilustrar la metodologia en la elaboración de sistemas de control de procesos o instrumentos de aplicación especifica, usando como herramienta de desarrollo al Sistema : **SDM88-PC** , (filosofía: *Microcomputer-Prototyping Board*), se pensó en el diseño de un :

"Analizador Digital de Respuesta en Frecuencia" ,

que permita el análisis gráfico y numérico (cualitativo y cuantitativo), de la Magnitud y el Angulo de Fase de la Función de Transferencia (a excitación senoidal) de un sistema (circuito analógico) cualquiera (filtros pasivos, filtros activos, arreglos de amplificadores operacionales, amplificadores de audio, etc.), para determinar sus caracteristicas (p.ej. ancho de banda, frecuencia central, margen de ganancia y de fase, etc.).

Así, gracias a la facilidad para cambiar los parámetros de análisis, al despliegue de resultados de una manera gráfica y numérica y al manejo interactivo del aparato por parte del usuario sobre las curvas de respuesta (función de transferencia) del sistema analizado, se podrán generar conclusiones con respecto a la estabilidad de dicho sistema bajo prueba, a su linealidad y calidad de desempeño en algún rango (frecuencias) de interés, etc.. Además, a partir del análisis y resultados numéricos (en todos los puntos muestreados) de la Magnitud y Fase, entregados por este instrumento, se podrán aplicar otros criterios de control

como "Márgenes de Ganancia y de Fase", "Criterio de estabilidad de Nyquist", etc., de una manera experimental, y se podrán tabular los valores de Magnitud y Fase, proporcionados por el programa, en forma conveniente para obtener diagramas logarítmicos o lineales según las representaciones comúnmente utilizadas para funciones de transferencia senoidales : Diagrama logaritmico de Bode, Diagrama polar o Traza de Nyquist y Diagrama del logaritmo de la Amplitud en función de la Fase.

El conocimiento de la Función de Transferencia de un proceso o de un sistema, es requerido para la implementación de estrategias de control avanzadas. El Analizador Digital de Respuesta en Frecuencia, permite obtenerla de manera experimental (empleando parámetros reales y aplicándolos físicamente sobre el circuito diseñado). Esto permite comprobar (o aún obviar), el desarrollo de la función de Transferencia generado mediante un análisis teórico-matemático (a veces muy laborioso, incosteable o imposible de hacer por el número de variables que intervienen en el funcionamiento del sistema y que conforman su desempeño).

La finalidad más obvia de este instrumento, es que el diseñador disponga de un medio rápido y confiable para caracterizar, ajustar y comprobar el funcionamiento de sus sistemas (circuitos analógicos de filtrado y control), de acuerdo a una respuesta esperada y prevista según sus cálculos matemáticos y métodos de diseño. Si el sistema bajo análisis no entrega la respuesta esperada, entonces podrá ir haciendo los ajustes y añadiendo los factores de compensación necesarios (en el papel y sobre su hardware) para obtener la respuesta correcta. Todo este proceso de calibración sucesiva y evaluación del sistema bajo prueba, podrá hacerse con la retroalimentación física y real que ofrece el Analizador Digital de Respuesta en Frecuencia.

También, gracias a que se cuenta con varios rangos y subrangos

para el barrido de frecuencia, se podrá caracterizar el funcionamiento del circuito en una amplia gama de valores de frecuencia de interés práctico (desde 20 Hz hasta unos 600 kHz ... valor límite estable determinado por el generador de la señal de prueba (XR-2206) y la frecuencia de conmutación de Switches Analógicos para el muestreo (CD-4016) ; empleando Switches Analógicos LF11331 o LF13331 (*Quad SPST JFET Analog Switches, normally open & with disable*), fabricados por *National Semiconductor Corp.*, se podrian lograr frecuencias estables mayores para el barrido de análisis (valores cercanos a 1 MHz debido a la limitación impuesta por XR-2206)).

Para realizar la prueba de Respuesta en frecuencia, se debe disponer de generadores de señal senoidal adecuados. Esta señal debe estar razonablemente libre de armónicas o distorsión. Ahora, para crear un instrumento autónomo, supervisado y sincronizado en sus funciones por microprocesador, el Generador de Funciones debe estar accesible en forma de Circuito Integrado a fin de tener un mayor control sobre él, cosa que seria más complicada con un generador de señal externo (aparato). Por esta razón se eligió un Generador de Funciones Monolítico Integrado XR-2206 que puede producir onda Senoidal (y Cuadrada) de buena calidad (Baja distorsión armónica senoidal, tipicamente 0.5%) y susceptible de ajuste para lograr buena simetria y forma de onda y capaz de operar en un rango que va desde 0.01 Hz hasta 1 MHz .

Hasta ahora se ha mencionado que el sistema bajo prueba debe ser electrónico, pero nada impide adecuar la señal senoidal electrónica de prueba para que adquiera una forma mecánica o neumática, y asi, poder analizar otro tipo de sistemas. Esto se lograria intercalando transductores de señal adecuados tanto a la entrada (conversión de señal eléctrica a mecánica o neumática), como a la salida (conversión de señal mecánica o neumática a eléctrica) del sistema a caracterizar.

El Sistema de Desarrollo **SDM88-PC** es tomado como soporte de *Software* y como *Hardware* base a partir del cual se irán añadiendo, probando y evaluando de una en una, las etapas que conforman el instrumento de aplicación, hasta obtener la versión final.

Tanto el *Hardware* como el *Software* se irán desarrollando en módulos, y de uno en uno a la vez, se irán incorporando (dando de alta) al circuito global; esto con el fin de facilitar la identificación de errores y su depuración así como la calibración de las distintas secciones del instrumento.

En el *Software*, tanto de la interface como de la microcomputadora, se dividirá el total de las Tareas de Programación en módulos, a fin de poder ser probados y depurados (*debug*) individualmente.

El *Hardware* constará de etapas que se encargarán del barrido de frecuencia sobre el circuito bajo prueba, muestreo, filtrado, amplificación y adecuación de señal, así como de la medición de frecuencia; la información digital será adquirida, almacenada secuencialmente y transmitida hacia la microcomputadora en forma serial, por la interface de *Hardware* del sistema **SDM88-PC**.

El *Software* se encargará de sincronizar y desencadenar acciones sobre el *Hardware* (circuitos programables controlados por el microprocesador) para efectuar los procesos de barrido de frecuencia, adquisición/almacenamiento-organizado-de-datos, y transferencia de datos desde interface a microcomputadora, y, ya residentes en la memoria de ésta última, realizará el procesamiento matemático de la información digital muestreada y su escalamiento para graficación, así como la lectura de parámetros suministrados por el usuario, el despliegue de "menúes" para selección de alternativas y el desencadenamiento condicional de acciones de procesamiento para cada opción elegida, el despliegue

de resultados y ajuste/lectura de frecuencia de manera interactiva (mediante el desplazamiento de un cursor gráfico por parte del usuario). Además, ejecutará cuando se requiera, distintas rutinas, tales como : toma de opción de usuario, inicialización de puertos de comunicación y periféricos digitales I/O del instrumento de aplicación en general, manejo de errores, ajuste de baudaje, etc. .

Como se ha observado, el primer paso en el desarrollo de un nuevo instrumento consiste en la definición cuidadosa y exacta de la función que se desea realice. El siguiente paso es decidir las partes del instrumento que se harán en *Hardware* y las partes que se harán en *Software*. Es entonces cuando se podrá analizar cómo conviene realizar cada una de estas partes.

5.4.2
METODOS DE RESPUESTA EN FRECUENCIA : CARACTERISTICAS, VENTAJAS, OBSERVACIONES Y CONCEPTOS AFINES

5.4.2.1
RESPUESTA EN FRECUENCIA : DEFINICION

Por el término "Respuesta en Frecuencia" se entiende la respuesta en estado de régimen permanente (respuesta estacionaria o de estado estable), de un sistema ante una entrada senoidal.

Los métodos de Respuesta en Frecuencia son los más convencionales y disponibles para que los ingenieros puedan efectuar el análisis y diseño de sistemas de control. En ellos, se varía la frecuencia de la señal de entrada en un cierto rango y se estudia la respuesta de frecuencia resultante.

El criterio de estabilidad de Nyquist, permite estudiar la estabilidad absoluta y la relativa de sistemas lineales de malla cerrada con el conocimiento de las características de Respuesta en Frecuencia de malla abierta, sin necesidad de determinar las raíces de la ecuación característica generadas de un análisis matemático. Esta es una ventaja del procedimiento de Respuesta en Frecuencia.

Otra ventaja de este método, es que las mediciones de Respuesta en Frecuencia, en general son simples y pueden efectuarse con exactitud usando generadores de señal senoidal fácilmente asequibles y equipos de medición precisos. Frecuentemente se puede determinar experimentalmente, la función de Transferencia de componentes complicados en pruebas de Respuesta en Frecuencia. Además, el método tiene las ventajas de que permite diseñar un sistema de manera que los efectos del ruido indeseable sean despreciables, y de que este análisis y diseño pueda extenderse a ciertos sistemas de control no lineales.

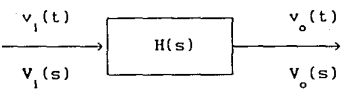
Aunque la Respuesta en Frecuencia de un sistema da una imagen cualitativa de la respuesta transitoria, la correlación entre frecuencia y respuestas transitorias, es indirecta (excepto en el caso de sistemas de segundo orden). Al proyectar un sistema en malla cerrada, se puede ajustar la característica de Respuesta en Frecuencia, usando varios criterios de diseño, para obtener características aceptables de respuesta transitoria. Cuando se comprende esta correlación indirecta entre mediciones de la respuesta transitoria y la Respuesta en Frecuencia, se puede utilizar ventajosamente este método para el diseño de un sistema, al interpretar las características dinámicas deseadas, en términos de las características de Respuesta en Frecuencia. Este análisis de un sistema, indica gráficamente qué modificaciones hay que hacer en la función de Transferencia de malla abierta, para obtener las características deseadas de respuesta transitoria.

5.4.2.2

FUNDAMENTO TEORICO DE LA OBTENCION DE RESPUESTAS ESTACIONARIAS ANTE ENTRADAS SENOIDALES

Se pueden obtener las características de Respuesta en Frecuencia de un sistema, directamente de la Función de Transferencia Senoidal; es decir, la función de Transferencia en la cual " s " es reemplazada por " jω " siendo " ω " la frecuencia angular, { ω = 2πf [rad/s] },{ f [Hz] }.

Sea el siguiente, un sistema lineal e invariante en el tiempo:



La Entrada se designa como $v_i(t)$ y la Salida como $v_o(t)$, mientras que $H(s)$ es la Función de Transferencia del Sistema.

La Entrada es Senoidal y está dada por :

$$v_i(t) = A_m \operatorname{sen}(\omega t)$$

Se supone que la Función de Transferencia puede ser escrita como una relación de dos polinomios en " s " , es decir

$$H(s) = \frac{p(s)}{q(s)} = \frac{p(s)}{(s+s_1)\ (s+s_2)\ \ldots\ (s+s_n)}$$

entonces, la Transformada de Laplace de la Salida $V_o(s)$ es

$$V_o(s) = H(s)\ V_i(s) = \frac{p(s)}{q(s)}\ V_i(s) \qquad (1)$$

donde $V_i(s)$ es la Transformada de Laplace de la Entrada $v_i(t)$.

$$V_i(s) = L\left\{v_i(t)\right\} = L\left\{A_m \operatorname{sen}(\omega t)\right\} = \frac{A_m \omega}{s^2 + \omega^2}$$

El estudio se limitará exclusivamente a sistemas estables. Para esos sistemas, las partes reales de las $-s_i$ son negativas. La respuesta estacionaria (permanente) de un sistema estable, lineal e invariante en el tiempo ante una entrada senoidal, no depende de las condiciones iniciales. (Por tanto, se pueden suponer condiciones iniciales cero).

Si $V_o(s)$ tiene únicamente polos distintos, el desarrollo en fracciones parciales de la Ec.(1) da

$$V_o(s) = \frac{p(s)}{q(s)} \quad \frac{A_m \omega}{s^2 + \omega^2} \qquad (2)$$

$$= \frac{a}{s + j\omega} + \frac{\bar{a}}{s - j\omega} + \frac{b_1}{s + s_1} + \frac{b_2}{s + s_2} + \ldots + \frac{b_n}{s + s_n}$$

donde a y las b_i ($i = 1,2,\ldots,n$) son constantes y $\bar{a}$ es el complejo conjugado de a .

La Transformada inversa de Laplace de la Ec.(2) da

$$v_o(t) = a\, e^{-j\omega t} + \bar{a}\, e^{+j\omega t} + b_1 e^{-s_1 t} + b_2 e^{-s_2 t} + \ldots + b_n e^{-s_n t}$$

$$(t \geq 0) \qquad (3)$$

Para un sistema estable, $-s_1$, $-s_2$, ... , $-s_n$ tienen partes reales negativas. Por tanto, a medida que t tiende a infinito, los términos $e^{-s_1 t}$, $e^{-s_2 t}$, ... , $e^{-s_n t}$ tienden a

cero. Asi, todos los miembros de la Ec.(3) , con excepción de los dos primeros, desaparecen en el estado estacionario.

Si $V_o(s)$ involucra polos múltiples s_j de multiplicidad m_j , entonces $v_o(t)$ contendrá términos como

$$t^{h_j} e^{-s_j t} \qquad (h_j = 0, 1, 2, \ldots, m_j - 1) .$$

Como las partes reales de los $-s_j$ son negativas para un sistema estable, los términos $t^{h_j} e^{-s_j t}$ tienden a cero cuando t tiende a infinito.

Entonces, independientemente de que el sistema sea o no del tipo de polos distintos, la respuesta permanente es

$$v_o(t) = a\, e^{-j\omega t} + \bar{a}\, e^{+j\omega t} \qquad (4)$$

donde se puede calcular la constante a de la Ec.(2) del siguiente modo : [notar que : $s^2 + \omega^2 = (s+j\omega)(s-j\omega)$]

$$a = H(s) \left. \frac{A_m\, \omega}{s^2 + \omega^2} \, (s+j\omega) \right|_{s=-j\omega} = - \frac{A_m\, H(-j\omega)}{j\, 2}$$

y su complejo conjugado

$$\bar{a} = H(s) \left. \frac{A_m\, \omega}{s^2 + \omega^2} \, (s-j\omega) \right|_{s=+j\omega} = + \frac{A_m\, H(+j\omega)}{j\, 2}$$

como $H(j\omega)$ es una cantidad compleja, se la puede escribir en forma Fasorial del siguiente modo :

$$H(j\omega) = |H(j\omega)|\, e^{j\phi}$$

donde $|H(j\omega)|$ representa la magnitud, amplitud, módulo o valor absoluto, y ϕ representa el ángulo de $H(j\omega)$ (ángulo de fase), es decir,

$$\phi = \angle H(j\omega) = \tan^{-1}\left\{ \frac{\text{parte Imaginaria de } H(j\omega)}{\text{parte Real de } H(j\omega)} \right\}$$

El ángulo ϕ puede ser negativo, positivo o cero. En forma similar se obtiene la siguiente expresión para $H(-j\omega)$ (forma Fasorial) :

$$H(-j\omega) = |H(-j\omega)|\, e^{-j\phi} = |H(j\omega)|\, e^{-j\phi}$$

Entonces, la Ec.(4) se puede escribir como

$$v_o(t) = A_m\, |H(j\omega)|\, \frac{e^{j(\omega t+\phi)} - e^{-j(\omega t+\phi)}}{j\,2}$$

por la identidad de Euler se obtiene que : $\text{sen}\ \alpha = \dfrac{e^{j\alpha} - e^{-j\alpha}}{j\,2}$

$$v_o(t) = A_m\, |H(j\omega)|\ \text{sen}(\omega t+\phi)$$

$$= |V_o(j\omega)|\ \text{sen}(\omega t+\phi) \qquad (5)$$

donde $|V_o(j\omega)| = A_m\, |H(j\omega)|$.

Se ve que un Sistema estable, lineal e invariante en el tiempo, sometido a una entrada Senoidal, y llegado al estado de régimen permanente, presenta una salida senoidal de la misma frecuencia que la entrada . Pero en general, la Amplitud y la Fase de la salida, son diferentes a las de la entrada. De hecho, la Amplitud de la salida está dada por el producto de la Amplitud de la Entrada $\{ A_m \}$ y la magnitud de la Función de Transferencia $\{ |H(j\omega)| \}$, mientras que el Angulo de Fase difiere del de la Entrada en el valor $\{ \phi = \angle H(j\omega) \}$.

En la { *figura* 5-1 } se muestra un ejemplo de señales senoidales de Entrada y Salida.

Sobre esta base, se obtiene el siguiente importante resultado: para Entradas SENOIDALES :

$$|H(j\omega)| = \left| \frac{V_o(j\omega)}{V_i(j\omega)} \right| = \left\{ \begin{array}{l} \text{RELACION DE AMPLITUD DE LA} \\ \text{SENOIDE DE SALIDA A LA} \\ \text{SENOIDE DE ENTRADA} \end{array} \right\}$$

$$\angle H(j\omega) = \angle \frac{V_o(j\omega)}{V_i(j\omega)} = \left\{ \begin{array}{l} \text{ANGULO DE FASE DE LA} \\ \text{SENOIDE DE SALIDA} \\ \text{CON RESPECTO A LA} \\ \text{SENOIDE DE ENTRADA} \end{array} \right\}$$

Por tanto, se pueden obtener las caracteristicas de Respuesta de un sistema ante una Entrada Senoidal, directamente de :

$$\frac{V_o(j\omega)}{V_i(j\omega)} = H(j\omega)$$

La Función de Transferencia Senoidal $H(j\omega)$, relación entre $V_o(j\omega)$ y $V_i(j\omega)$, es una cantidad compleja que puede ser representada por la Magnitud (Amplitud) y el Angulo de Fase con la Frecuencia como parámetro.

Angulo de Fase Negativo = Retardo de Fase
Angulo de Fase Positivo = Adelanto de Fase

La Función de Transferencia Senoidal de cualquier Sistema lineal, puede obtenerse reemplazando " jω " en lugar de " s " en la Función de Transferencia del Sistema.

{ *figura* 5-1 }

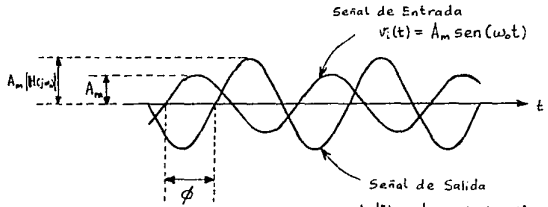
Señales Senoidales de Entrada $v_1(t)$ y de Salida $v_o(t)$ (Sistema Analógico), y posibilidades en cuanto al Angulo de Fase ϕ .

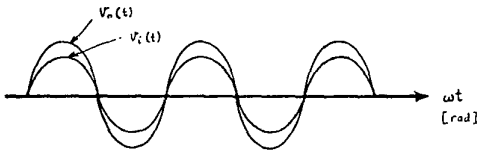
$V_i(t)$ y $V_o(t)$ a una misma frecuencia angular ω_o de operación

$\omega_o = 2\pi F_o \quad [\frac{rad}{s}]$

$F_o = \frac{1}{T_o} \quad [Hz]$

T_o [s] = Periodo

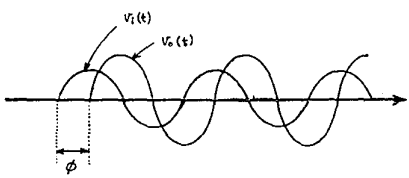




V_o en fase con V_i : $\angle H(j\omega_o) = \phi = 0°$

Angulo de Fase de la Senoidal de Salida con respecto a la Senoidal de Entrada.

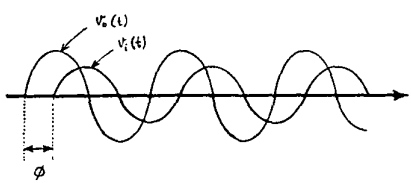
$$\angle H(j\omega_o) = \angle \frac{V_o(j\omega)}{V_i(j\omega_o)} = \angle V_o(j\omega_o) - \angle V_i(j\omega_o) = \phi_o - \phi_i = \phi \quad [rad]$$



$\angle H(j\omega_o) = \phi < 0°$

ϕ (negativo)

V_o Atrasado con respecto a V_i (Retardo de Fase)



$\angle H(j\omega_o) = \phi > 0°$

ϕ (positivo)

V_o Adelantado con respecto a V_i (Adelanto de Fase)

Para caracterizar totalmente un Sistema lineal en el dominio de la frecuencia, se deben especificar tanto la relación de Magnitud (Amplitud) como el Angulo de Fase, como funciones de la frecuencia ω .

5.4.2.3

REPRESENTACION DE LAS CARACTERISTICAS DE RESPUESTA EN FRECUENCIA DE SISTEMAS LINEALES

La Función de Transferencia Senoidal, es una función compleja de la frecuencia ω , y es caracterizada por su Módulo y su Angulo de Fase, (con la frecuencia como parámetro).

Existen tres representaciones comúnmente utilizadas de Funciones de Transferencia Senoidales :

1) Diagrama Logaritmico o Diagrama de Bode
2) Diagrama Polar o Traza de Nyquist
3) Diagrama del logaritmo de la Amplitud (Magnitud) en función de la Fase (Diagrama de Nichols)

5.4.2.3_A

DIAGRAMA LOGARITMICO O DIAGRAMA DE BODE

Se puede representar una Función de Transferencia Senoidal, por dos diagramas distintos : uno que da la Amplitud (Magnitud) en función de la frecuencia, y el otro el ángulo de Fase en función de la frecuencia.

Un Diagrama Logaritmico o Diagrama de Bode de una Función de Transferencia Senoidal, consta de dos trazados:

a) diagrama del logaritmo del Módulo (Magnitud) [dB]
b) diagrama del Angulo de Fase [grados sexagesimales]

Ambos son representados en función de la frecuencia en escala logarítmica. (En el Analizador Digital de Respuesta en Frecuencia, la frecuencia aparece en escala lineal (en eje horizontal), por las razones que se darán después).

La representación normal de la Magnitud Logaritmica de $H(j\omega)$ es :

$|H(j\omega)|_{dB} = 20 \log|H(j\omega)|$ [deciBeles = dB] { log base 10 }

(Unidad de amplitud usada en esta representación = deciBel [dB])
(Base de logaritmos = 10 ..... $\log_{10}$) .

En la representación logaritmica se dibujan las curvas en papel semilogaritmico, utilizando la escala logarítmica para frecuencias (eje horizontal) y la escala lineal ya sea para Magnitud (pero en dB) o para Angulo de Fase (en grados sexagesimales) (eje vertical). El campo de frecuencias de interés determina la cantidad de ciclos logaritmicos necesarios en la abscisa.

La ventaja principal de usar un diagrama logaritmico, es que se puede convertir la multiplicación de amplitudes en adición. Además, se dispone de un método simple para trazar una curva aproximada del logaritmo de la Amplitud.

El método está basado en la aproximación asintótica (por lineas rectas asintotas), misma que es suficiente si sólo se necesita una información global sobre las caracteristicas de Respuesta en Frecuencia. En caso de requerirse curvas exactas, se pueden efectuar fácilmente correcciones a esas determinaciones asintóticas básicas. Se pueden dibujar rápidamente las curvas de Angulo de Fase, si se tiene una plantilla para la curva de ángulo de fase de $1+j\omega$.

Cabe señalar también, que se puede realizar muy simplemente la determinación experimental de una Función de Transferencia, si se representan los datos de Respuesta en Frecuencia, en forma de un diagrama logarítmico.

La representación logarítmica es útil, porque presenta las características de alta y baja frecuencia de la Función de Transferencia en un solo diagrama. Es muy ventajoso el poder expandir el rango de bajas frecuencias utilizando una escala logarítmica de frecuencias, ya que a dichas frecuencias son muy importantes las características en los sistemas utilizados (sobre todo en aplicaciones de control no electrónico). Notar que debido a la escala de frecuencia logarítmica, es imposible trazar curvas hasta la frecuencia cero (DC); sin embargo, esto no crea un problema importante.

5.4.2.3_B

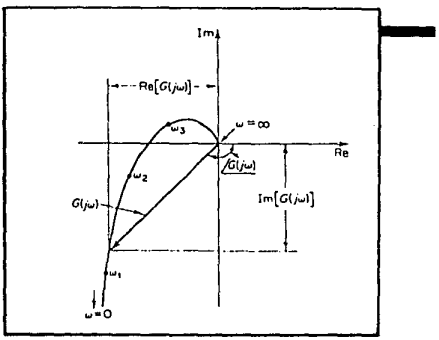
DIAGRAMA POLAR O TRAZA DE NYQUIST

El Diagrama Polar o de Nyquist de una Función de Transferencia Senoidal $H(j\omega)$, es un diagrama de la Magnitud de $H(j\omega)$ en función del ángulo de Fase de $H(j\omega)$ en coordenadas polares al variar ω desde cero hasta "infinito". Entonces, el Diagrama Polar es el lugar geométrico de los vectores $|H(j\omega)| \angle H(j\omega)$ al variar ω desde cero a infinito. Debe tenerse en cuenta que en los Diagramas polares, se mide un ángulo de Fase positivo (negativo) en sentido antihorario (en sentido horario) a partir del eje Real positivo.

En la { *figura* 5-2 } , se muestra un ejemplo de este tipo de diagramas.

{ *figura* 5-2 }

Ejemplo de un Diagrama Polar o de Nyquist



Cada punto del Diagrama Polar de $H(j\omega)$, representa el punto terminal de un vector para un valor determinado de ω .

En el Diagrama Polar es importante mostrar la graduación de frecuencia sobre el diagrama. Las proyecciones de $H(j\omega)$ en los ejes Real e Imaginario , son las componentes Real e Imaginaria de la Función de Transferencia del sistema analizado.

Para poder construir el Diagrama polar de $H(j\omega)$, tanto la Magnitud $|H(j\omega)|$ como el Angulo de Fase $\angle H(j\omega)$ deben ser calculados directamente para cada frecuencia ω . Sin embargo, como el Diagrama Logarítmico (Bode) es fácil de construir, la información necesaria para trazar el diagrama polar puede obtenerse directamente de aquél, si es dibujado previamente y se convierten deciBeles [dB] en una Magnitud Lineal (ordinaria) utilizando la siguiente expresión que resulta de despejar $|H(j\omega)|$ de : $[dB] = 20 \log_{10}(|H(j\omega)|)$:

valor Magnitud lineal = $|H(j\omega)| = \text{antilog}_{10}\left[\frac{[dB]}{20}\right] = 10^{\left[\frac{dB}{20}\right]}$

Una ventaja de utilizar un Diagrama Polar o de Nyquist es que presenta las caracteristicas de Respuesta en Frecuencia de un sistema en todo el rango de frecuencias en un único diagrama. Una desventaja es que el diagrama no indica claramente las contribuciones de cada uno de los factores individuales de la Función de Transferencia de malla abierta.

5.4.2.3_C

DIAGRAMA DEL LOGARITMO DE LA MAGNITUD EN FUNCION DEL ANGULO DE FASE (DIAGRAMA DE NICHOLS)

Este otro método de representar gráficamente las caracteristicas de Respuesta en Frecuencia, consiste en realizar el diagrama de una Magnitud logarítmica en [deciBeles] en función del Angulo de Fase o "Margen de Fase" para un rango de frecuencias de interés. { El Margen de Fase es la diferencia entre el ángulo de Fase efectivo ϕ y -180°; es decir, $\phi-(-180^{\circ}) = 180^{\circ} + \phi$ }. La curva está graduada en términos de la frecuencia ω .

En el Diagrama de Bode, las caracteristicas de Respuesta en Frecuencia de $H(j\omega)$ se trazan en papel semilogaritmico en dos curvas separadas: la curva del logaritmo de la Magnitud [dB] y la curva del Angulo de Fase [$\circ$] ; mientras que en el Diagrama de Nichols, ambas curvas del diagrama de Bode están combinadas en una sola. Se puede construir fácilmente el diagrama de Nichols, leyendo valores del logaritmo de la Magnitud y del Angulo de Fase del diagrama de Bode. Cabe señalar que en este tipo de diagrama (Logaritmo de la Magnitud en función del Angulo de Fase), un cambio en la constante de ganancia de $H(j\omega)$, simplemente desplaza la curva hacia arriba (para ganancia creciente) o hacia

abajo (para ganancia decreciente), pero la forma de la curva se mantiene constante. La distancia vertical entre los puntos $\omega = 0$ y $\omega = \omega_r$ (frecuencia de resonancia), es el valor pico de $H(j\omega)$ en deciBeles (M_r).

Las ventajas del Diagrama de Nichols son que se puede determinar rápida y fácilmente la estabilidad relativa del sistema de malla cerrada, así como la compensación requerida.

.....

En la { *figura* 5-3 } (página siguiente), se comparan las curvas de Respuesta en Frecuencia de la Función de Transferencia :

$$G(j\omega) = H(j\omega) = \frac{1}{1 + 2\zeta\left(j\frac{\omega}{\omega_n}\right) + \left(j\frac{\omega}{\omega_n}\right)^2} \qquad (\zeta > 0)$$

en las tres distintas representaciones gráficas analizadas anteriormente :

- Diagrama Logarítmico (Bode)
- Diagrama Polar (Nyquist)
- Diagrama del Logaritmo de la Magnitud en función del Angulo de Fase (Nichols)

ω = Frecuencia Angular [rad/s] (variable independiente)
(parámetro para todas las representaciones)

ω_n = Frecuencia Natural No Amortiguada [rad/s]

ω_r = Frecuencia de Resonancia [rad/s]

$M_p = M_r$ = Sobrepaso, Sobreimpulso Máximo o Magnitud Pico

ζ = Factor de Amortiguamiento del sistema (*damping factor*)
{ { ζ=0 Sistema Oscilatorio},{ 0<ζ<1 Sistema Subamortiguado},
{ ζ=1 S.Criticamente Amortiguado},{ ζ>1 S.Sobreamortiguado})

{ *figura* 5-3 }

Representaciones Gráficas de la Respuesta en Frecuencia de para la Función de Transferencia $G(j\omega)=H(j\omega)$:

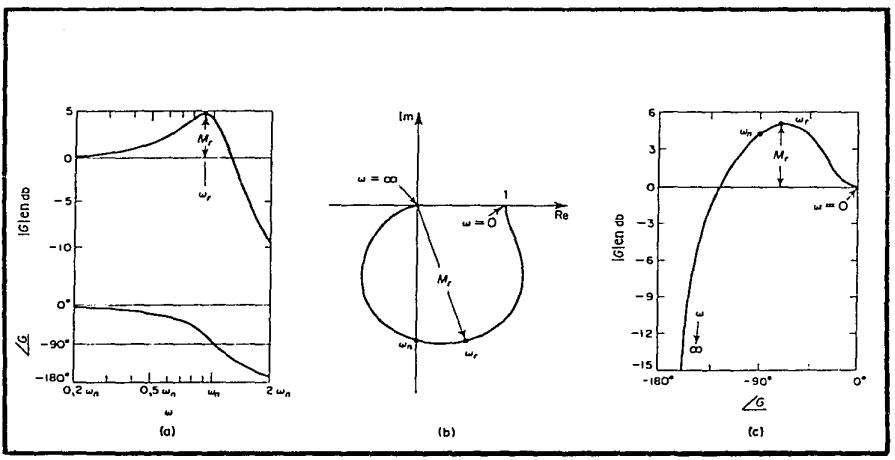
$$G(j\omega) = H(j\omega) = \frac{1}{1 + 2\zeta\left(j\frac{\omega}{\omega_n}\right) + \left(j\frac{\omega}{\omega_n}\right)^2}$$

$(\zeta > 0)$

a) Diagrama Logaritmico (Bode)

b) Diagrama Polar (Nyquist)

c) Diagrama del Logaritmo de la Magnitud en función del Angulo de Fase (Nichols)



5.4.2.4

MARGENES DE FASE Y DE GANANCIA

Supóngase que para un sistema se han representado en el plano complejo { Re vs. Im } (Diagrama polar o Traza de Nyquist) las curvas de Respuesta en Frecuencia para diferentes valores de ganancia de malla abierta K . Generalmente, para un valor muy grande de ganancia K, el sistema se vuelve inestable. Al disminuir la ganancia hacia determinado valor, el lugar geométrico de $G(j\omega)$ (ó $H(j\omega)$) pasa por el punto { -1+j0 } . Esto significa que con este valor de ganancia el sistema está al borde de la inestabilidad, y que presentará oscilaciones mantenidas. Para un valor pequeño de ganancia K, el sistema es estable. En general, cuanto más cerca pasa el lugar geométrico de $G(j\omega)$ del punto { -1+j0 }, más oscilatorio se vuelve el sistema. Se puede utilizar la proximidad de $G(j\omega)$ a este punto como una medida del margen de estabilidad. Normalmente esta cercanía se representa en términos de margen de fase y margen de ganancia.

a) Margen de Fase :

Es la cantidad de retardo de fase adicional necesaria a la frecuencia de cruce o de transición de ganancia para que el sistema quede al borde de la inestabilidad. La frecuencia de cruce de ganancia es aquella para la cual el valor absoluto $|G(j\omega)|$ (ó $|H(j\omega)|$) de la Función de Transferencia de malla abierta, es la unidad. El Margen de Fase (γ) es 180° más el Angulo de Fase ϕ de la Función de Transferencia de malla abierta a la frecuencia de cruce de ganancia, es decir :

$$\gamma = 180^\circ + \phi$$

En el diagrama de Nyquist se puede trazar una línea desde el origen al punto en el cual la circunferencia de radio unitario cruza al lugar geométrico de $G(j\omega)$. El ángulo desde el eje Real negativo a esta línea es el Margen de

Fase. El Margen de Fase es positivo para { $\gamma > 0$ } y negativo para { $\gamma < 0$ } . Para que un sistema de fase mínima sea estable, el Margen de Fase debe ser positivo.

b) **Margen de Ganancia :**

Esta cantidad es el reciproco de la magnitud de la Función de Transferencia $|G(j\omega)|$ a la frecuencia a la cual el Angulo de fase es -180° .

Definiendo la frecuencia de cruce o de transición de fase ω_1 como la frecuencia a la cual el Angulo de Fase de la Función de Transferencia de malla abierta es igual a -180°, se tiene que el Margen de Ganancia K_g es :

$$K_g = \frac{1}{|G(j\omega_1)|}$$

y expresado en deciBeles :

$$K_g \text{ [dB]} = 20 \log_{10}(K_g) = - 20 \log_{10}(|G(j\omega_1)|)$$

El Margen de Ganancia expresado en [dB] es positivo si K_g es mayor que la unidad, negativo si es menor que la unidad y cero si es igual a 1 .

Un Margen de Ganancia Positivo (en [dB]), significa que el sistema es estable, y un Margen de Ganancia Negativo (en [dB]), implica que el sistema es inestable.

Para un sistema de fase minima estable, el Margen de Ganancia indica cuánto se puede incrementar la ganancia antes de que el sistema se haga inestable. Para un sistema inestable, el Margen de Ganancia indica en cuánto hay que reducir la ganancia para que el sistema se haga estable.

El Margen de Ganancia de un sistema de primer orden o de segundo orden es infinito, pues los diagramas polares de esos sistemas no cruzan el eje Real negativo. Por tanto, teóricamente esos sistemas no pueden ser inestables.

Para un sistema de fase no minima, no se satisface la condición de estabilidad a menos que el diagrama de $G(j\omega)$ rodee al punto { -1+j0 } . Por tanto, un sistema estable de fase no minima ha de tener Márgenes de Fase y Ganancia negativos.

.....

En la { *figura* 5-4 } , se muestran los Márgenes de Ganancia y de Fase tanto para un sistema Estable como para un Inestable en :

a) Diagramas Polares (Nyquist)
b) Diagramas Logaritmicos (Bode)
c) Diagramas del logartimo de la Magnitud en función del Angulo de Fase (Nichols)

Función de Transferencia del sistema : $G(j\omega)$

Notar que en los diagramas logaritmicos, el punto critico en el plano complejo corresponde a las lineas { 0 dB } y { -180° }

.....

Comentarios sobre los MARGENES de GANANCIA y de FASE .

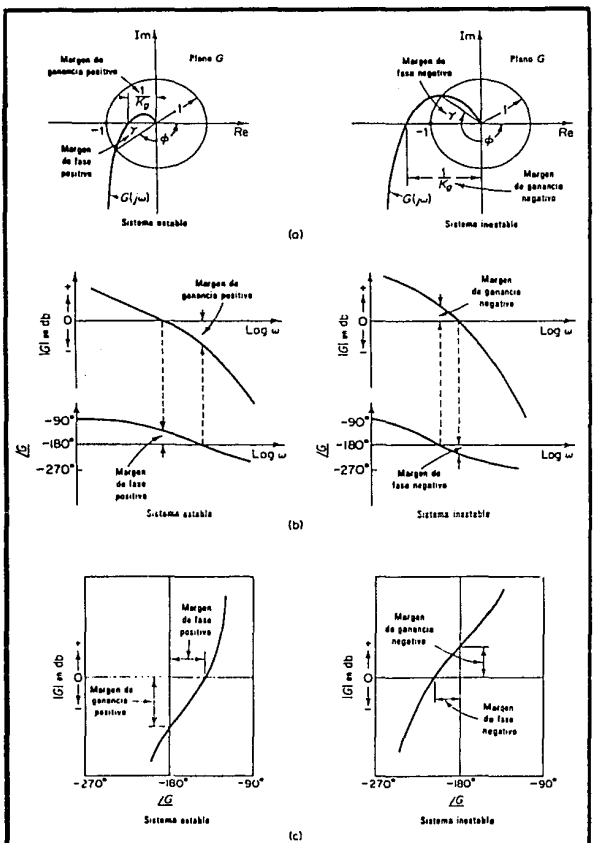
Los Márgenes de Ganancia y de Fase de un sistema de control son una medida de la cercania del diagrama polar al punto {-1+j0}. Por tanto, se pueden usar como criterio de proyecto.

Notar que ni el Margen de Ganancia solo, ni el Margen de Fase solo, dan información suficiente para la determinación de la estabilidad relativa, por lo que deben proporcionarse ambos.

Para un sistema de fase minima, tanto el Margen de Ganancia como el de Fase han de ser positivos para que el sistema sea estable. Los márgenes negativos indican inestabilidad.

{ *figura* 5-4 }

Márgenes de Ganancia y de Fase en sistema Estables e Inestables



Márgenes adecuados de Ganancia y de Fase dan seguridad contra variaciones en los componentes del sistema y se los especifica para valores definidos de frecuencia. Ambos valores acotan el comportamiento del sistema de malla cerrada en la vecindad de la frecuencia de resonancia. Para lograr un funcionamiento satisfactorio, el Margen de Fase debe estar entre 30° y 60° y el Margen de Ganancia debe ser superior a los 6 dB . Con estos valores, un sistema de fase minima tiene garantizada la estabilidad, aún si la ganancia de malla abierta y las constantes de tiempo de los componentes varian dentro de ciertos límites. Aunque los Márgenes de Ganancia y de Fase dan sólo estimaciones burdas de la relación de amortiguamiento efectiva (ζ) del sistema de malla cerrada, brindan un medio conveniente para proyectar sistemas de control o ajustar las ganancias de constantes de los sistemas.

Para sistemas de fase mínima, las características de Magnitud y de Fase de la Función de Transferencia de malla abierta están definidamente relacionadas. El requisito de que el Márgen de Fase esté entre 30° y 60° significa que en un diagrama logarítmico, la pendiente de la curva del logaritmo de la magnitud a la frecuencia de cruce de ganancia es más gradual que -40 dB/década. En la mayor parte de los casos prácticos, es deseable una pendiente de -20 dB/década (1 polo) a la frecuencia de cruce de Ganancia para tener estabilidad. Si es de -40 dB/década (2 polos), el sistema puede ser tanto estable como inestable. (Sin embargo, aún si el sistema es estable, el Margen de Fase es pequeño). Si la pendiente a la frecuencia de cruce de ganancia es de -60 dB/década o mayor (3 ó más polos), el sistema es inestable.

5.4.2.5

CONCEPTO DE FASOR : Representación de un ONDA SENOIDAL en el dominio del tiempo por un FASOR en el dominio de la frecuencia

Una onda senoidal de frecuencia angular ω es cualquier función del tiempo t definida en $(-\infty, \infty)$ y de la forma :

$$A_m \cos(\omega t+\phi) \quad \text{o} \quad A_m \operatorname{sen}(\omega t+\phi)$$

donde : A_m : Amplitud [V , A , etc.]

ω : Frecuencia Angular [rad/s] ($\omega = 2\pi f$) f[Hz]

ϕ : Angulo de Fase [rad ó °]

A_m , ω , ϕ : son constantes Reales

Además, se cumple el siguiente Teorema Fundamental :

"La suma algebraica de cualquier número de ondas senoidales de la misma frecuencia angular ω , y de cualquier número de sus derivadas de cualquier orden, es también una senoidal de la misma frecuencia angular ω ".

La implicación de este Teorema fundamental sugiere que se puede tratar a las sinusoides por métodos algebraicos.

Una Sinusoide con frecuencia angular ω queda completamente especificada por su **Amplitud** A_m y su **Fase** ϕ . Este hecho conduce a la idea de representar a una Sinusoide por el número complejo :

$$\mathbf{A} \equiv A_m e^{j\phi}$$

donde : $A_m = |\mathbf{A}|$ = Magnitud del número complejo $\mathbf{A}$

$\phi = \angle \mathbf{A}$ = Fase del número complejo $\mathbf{A}$

Es decir, la sinusoide $x(t) \equiv A_m \cos(\omega t+\phi)$, está representada por el número complejo $\mathbf{A} \equiv A_m e^{j\phi}$ y,

recíprocamente, dado el número complejo $A = A_m e^{j\phi}$ y la frecuencia angular ω , se puede recobrar la sinusoide como sigue :

$$x(t) = \mathbb{R}e\left[A\, e^{j\omega t}\right] = \mathbb{R}e\left[A_m e^{j(\omega t+\phi)}\right]$$

$$= \mathbb{R}e\left[A_m \cos(\omega t+\phi) + jA_m \operatorname{sen}(\omega t+\phi)\right]$$

$$= A_m \cos(\omega t+\phi) = x(t)$$

$$\{ A_m , \phi , \omega , t \} \in \mathbb{R} \quad ; \quad j = \sqrt{-1}$$

El número complejo **A** , que representa a la sinusoide $A_m \cos(\omega t+\phi)$, se denomina por conveniencia, el **FASOR** que representa a la sinusoide.

Por definición, el Fasor **A** está dado por $A \equiv A_m e^{j\phi}$

Por ejemplo :

Si $v(t) = 110 \cos(2\pi 60t+\frac{\pi}{3})$ [Volt]

entonces el Fasor que representa a la sinusoide es :

$$A = 110\, e^{j(\frac{\pi}{3})}$$

esto es :

$$v(t) = \mathbb{R}e\left[A\, e^{j2\pi 60t}\right]$$

Observaciones :

1) El conocimiento del **Fasor** que representa una sinusoide determina su Amplitud y su Fase, pero no su frecuencia.
Por tanto, es importante, que cuando se hacen cálculos con Fasores, se tenga en consideración la frecuencia de dichos Fasores.

2) Se puede especificar una sinusoide por una función SENO en vez de por una función COSENO, teniéndose entonces :

$$y(t) = A_m \, \mathrm{sen}(\omega t + \phi)$$

En este caso, la representación fasorial $\mathbf{A} \equiv A_m \, e^{j\phi}$ también es válida. Sin embargo, la recuperación de la sinusoide estaria dada por :

$$y(t) = \mathbb{I}m\left[\mathbf{A}\, e^{j\omega t}\right] = \mathbb{I}m\left[A_m e^{j(\omega t+\phi)}\right]$$

que es la representación-fasorial-por-Parte-Imaginaria ($\mathbb{I}m$) en vez de :

$$x(t) = \mathbb{R}e\left[\mathbf{A}\, e^{j\omega t}\right] = \mathbb{R}e\left[A_m e^{j(\omega t+\phi)}\right]$$

que es la representación-fasorial-por-Parte-Real ($\mathbb{R}e$) cuando la sinusoide se define como :

$$x(t) = A_m \cos(\omega t + \phi)$$

3) Si se grafica en el plano complejo la función (número complejo) $\mathbf{A}\, e^{j\omega t}$, sus coordenadas estarian dadas por :

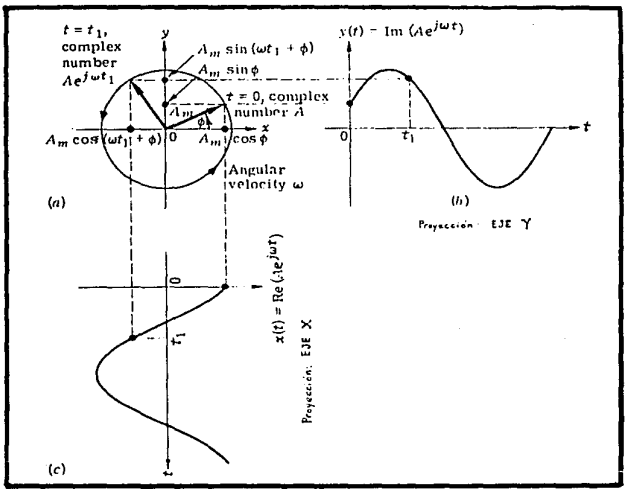
$$x(t) = \mathbb{R}e\left[\mathbf{A}\, e^{j\omega t}\right] \qquad y(t) = \mathbb{I}m\left[\mathbf{A}\, e^{j\omega t}\right]$$

Se puede pensar en $x(t)$ como la proyección sobre el eje X del punto $\mathbf{A}\, e^{j\omega t}$, que rota sobre una circunferencia de radio A_m a la velocidad angular de ω [rad/s] en sentido contrario a las manecillas del reloj.
De manera análoga, la proyección sobre el eje Y del punto $\mathbf{A}\, e^{j\omega t}$ da $y(t)$.

En la { *figura* 5-5 } se muestra la representación del Fasor rotante $\mathbf{A}\, e^{j\omega t}$.

{ *figura* 5-5 }

Representación del Fasor rotante $A\,e^{j\omega t}$



La respuesta completa de un circuito lineal ante una excitación senoidal consta de dos partes :

1) Respuesta Transitoria :
Se desvanece a medida que el tiempo aumenta y depende de las características naturales de los elementos y de las condiciones iniciales del circuito, por lo que se conoce también como "Respuesta Natural" de un circuito.

2) Respuesta Permanentente :
Esta respuesta no desaparece a medida que $t \to \infty$ y es forzada u originada por la fuente de excitación, conservando las características de ella.
Se conoce también como "Respuesta (Senoidal) de Estado Estable (*Steady-State response*).

Una de las formas para encontrar la Respuesta Total de un circuito es resolviendo la ecuación diferencial que lo representa. La Respuesta Transitoria se obtiene mediante la solución homogénea de la ecuación diferencial (igualación a cero), mientras que la Respuesta Permanente se obtiene encontrando la solución particular (que por ser la excitación senoidal, será del tipo : $A \operatorname{sen}\theta + B \cos\theta$). Sin embargo, este método es poco práctico para calcular la respuesta más importante de un circuito : la permanente.

El empleo de Fasores (cantidades complejas) facilita el cálculo de Respuestas Permanentes (no Transitorias ni Totales), a funciones excitatrices senoidales.

Una corriente o voltaje senoidal está caracterizado por tres parámetros :

V_m o I_m	=	Amplitud	[V o A]
ϕ	=	Angulo de Fase	[rad, $^\circ$ o grados eléctricos]
ω	=	Frecuencia Angular	[rad/s]

Si se considera un voltaje senoidal, puede representarse como:

$$v(t) = V_m \cos(\omega t+\phi) \qquad (R)$$

ó

$$v(t) = V_m \operatorname{sen}(\omega t+\phi) \qquad (I)$$

Ahora bien, como se vió antes en forma genérica, usando la identidad de Euler : { $e^{j\theta} = \cos\theta + j\operatorname{sen}\theta$ } , se puede descomponer al número complejo $V_m e^{j(\omega t+\phi)}$ en dos partes, una real y otra imaginaria :

$$V_m e^{j(\omega t+\phi)} = V_m \cos(\omega t+\phi) + j\, V_m \operatorname{sen}(\omega t+\phi)$$

Si de $V_m e^{j(\omega t+\phi)} = V_m e^{j\omega t} e^{j\phi}$, se llama Fasor **V** a la cantidad compleja $V_m e^{j\phi}$, o sea : $\mathbf{V} \equiv V_m e^{j\phi}$, entonces se tiene que la expresión del voltaje senoidal en el dominio del tiempo, puede recuperarse de :

$v(t) = \mathbb{R}e\left[\mathbf{V}\, e^{j\omega t}\right]$ si se emplea la forma COSENO { Ec. (R) }

como convención para representar sinusoides

{ representación-fasorial-por-Parte-Real ($\mathbb{R}e$) }

o de :

$v(t) = \mathbb{I}m\left[\mathbf{V}\, e^{j\omega t}\right]$ si se emplea la forma SENO { Ec. (I) }

como convención para representar sinusoides

{ representación-fasorial-por-Parte-Imaginaria ($\mathbb{I}m$) }

Se puede observar que el número complejo **V** contiene toda la información referida a la Amplitud (V_m) y al Angulo de Fase (ϕ) de la senoide (a una frecuencia ω dada).

Ahora bien, como la respuesta senoidal se debe a la fuente de excitación, la frecuencia (o velocidad) angular ω de ambas es la misma. Asi, se puede afirmar que ω no variará y todos los términos que contengan esta variable resultarán redundantes en los cálculos de las soluciones de las ecuaciones que representan al circuito; se puede decir que conociendo la Amplitud y la Fase de la señal, ésta podrá ser representada fielmente; en otras palabras, conociendo el FASOR de la señal, se conocerán las caracteristicas de la misma.

Dado que un Fasor $V \equiv V_m e^{j\phi}$ es la representación en números complejos de una señal senoidal que originalmente se encuentra en el dominio del tiempo, puede escribirse en forma polar como : $V = V_m \angle \phi$.
Por ser el Fasor un número complejo se denota con letras mayúsculas remarcadas o negritas.

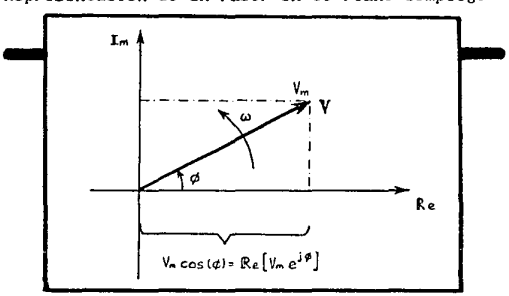
En la práctica se prefiere considerar a la Magnitud del Fasor igual al valor "rms" (root-mean-square = raiz cuadrada de la media de los cuadrados) de la función original en el dominio del tiempo. Los valores "rms" se usan en la transformación en vez de los valores pico de las funciones de tiempo, porque el valor "rms" es un número más significativo para describir el efecto de un voltaje o de una corriente periódica (efecto de efectividad relativa de un voltaje o corriente alterna en la transferencia de potencia y energia). El Angulo del Fasor es igual al argumento de la función coseno (o seno) cuando t=0 (Angulo de Fase ϕ).

El METODO FASORIAL se utiliza para resolver problemas de Redes o Circuitos Eléctricos cuando las excitaciones de Voltaje y de Corriente que se aplican a las redes son todas sinusoides de la MISMA FRECUENCIA (Frecuencia Angular $\omega = 2\pi f$ [rad/s] ; f[Hz])

El término FASOR se aplica sólo a VOLTAJES o CORRIENTES sinusoidales de la misma frecuencia. La Impedancia (Z) y la Admitancia (Y), no son Fasores, sino operadores fasoriales, ya que son números complejos que actúan para cambiar la Magnitud y el Angulo de los Fasores asociados de Voltaje y Corriente.
Ecuaciones Fasoriales Voltaje-Corriente : $V = Z\,I$; $I = Y\,V$

La representación gráfica de cualquier Fasor en el Plano Complejo, consiste en una Recta dirigida (flecha), dibujada a escala, que se mueve con una velocidad (frecuencia) angular ω en sentido antihorario, y que parte del origen de dicho plano.

{ *figura* 5-6 }
Representación de un Fasor en el Plano Complejo



Ejemplos:

1. TRANSFORMACION DE ONDA SENOIDAL EN EL DOMINIO DEL TIEMPO A FASOR

$v(t) = 50\ \mathrm{sen}(300t-60^\circ)$ [V]

Como $\mathrm{sen}(\omega t+\phi) = \cos(\omega t+\phi-90^\circ)$, $\Rightarrow$ $v(t) = 50\ \cos(300t-150^\circ)$

$\mathbf{V} = V_m e^{j\phi} = 50\ e^{-j150^\circ} = 50\ \angle -150^\circ$ [V]

2. TRANSFORMACION DE FASOR A ONDA SENOIDAL EN EL DOMINIO DEL TIEMPO

$I = 5\ \angle -15^\circ$ [A]

$i(t) = I_m \cos(\omega t+\phi) = 5\ \cos(\omega t-15^\circ)$

Como $\cos(\omega t+\phi) = \mathrm{sen}(\omega t+\phi+90^\circ)$, $\Rightarrow$ $i(t) = 5\ \mathrm{sen}(\omega t+75^\circ)$ [A]

5.4.3

BASE TEORICO-MATEMATICA DEL PRINCIPIO DE FUNCIONAMIENTO DEL INSTRUMENTO

Como se ha establecido antes, en el análisis de los sistemas estables, lineales e invariantes con el tiempo, es de gran importancia la "Respuesta Senoidal Permanente". Este análisis se puede realizar por distintos métodos : Diagrama de Bode, de Nyquist, de Nichols y el análisis Fasorial.

Según se demostró en la sección { 5.4.2.2 } , si a un sistema estable, lineal e invariante con el tiempo se le aplica una excitación senoidal de frecuencia angular ω_o :

$$v_i(t) = A_m \operatorname{sen}(\omega_o t) \qquad ; \; A_m = \text{Amplitud pico [Volt]} \in \mathbb{R}$$

$$\omega_o = 2\pi f_o \; [\text{rad/s}] \quad ; \quad f_o \; [\text{Hz}]$$

entonces, su Respuesta Permanente será :

$$v_o(t) = B_m \operatorname{sen}\left(\omega_o t + \phi\right) \tag{1a}$$

donde : $B_m = A_m \, |H(j\omega_o)|$; $\phi = \angle H(j\omega_o)$

MAGNITUD DE LA RESPUESTA — ANGULO DE FASE

es decir :

$$v_o(t) = A_m \, |H(j\omega_o)| \operatorname{sen}\left(\omega_o t + \angle H(j\omega_o)\right) \tag{1b}$$

que también puede escribirse como :

$$v_o(t) = \mathrm{Im}\left[A_m \, H(j\omega_o) \, e^{j\omega_o t} \right] \tag{2}$$

donde :

$$H(j\omega_o) = |H(j\omega_o)|\, e^{j\phi} = |H(j\omega_o)| \angle\phi = |H(j\omega_o)| \angle H(j\omega_o) \qquad (3)$$

es el FASOR asociado a la FUNCION DE TRANSFERENCIA (senoidal) del sistema, que tiene a la frecuencia angular ω como parámetro.

$|H(j\omega_o)|$ = Magnitud (Módulo) de la Función de Transferencia

$\angle H(j\omega_o) = \phi$ = Angulo de Fase

Nótese que ϕ es función de $\omega_o \Rightarrow \phi = f(\omega_o) = \phi(\omega_o)$

Si se desea representar en el plano complejo dicho Fasor, es necesario conocer sus componentes Real e Imaginaria :

$$|H(j\omega_o)| \cos(\phi) \qquad y \qquad |H(j\omega_o)| \operatorname{sen}(\phi)$$

Esto puede hacerse a partir del Fasor asociado a la Respuesta (Salida) obtenida del sistema :

$$V_o(j\omega_o) = |V_o(j\omega_o)| \angle V_o(j\omega_o)$$

$$= B_m e^{j\phi} = A_m |H(j\omega_o)| e^{j\phi}$$

$$= A_m |H(j\omega_o)| \angle\phi$$

$$= A_m H(j\omega_o)$$

$$V_o(j\omega_o) = A_m |H(j\omega_o)| \cos(\phi) + j A_m |H(j\omega_o)| \operatorname{sen}(\phi)$$

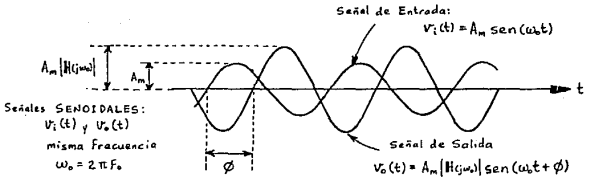
COMPONENTE REAL COMPONENTE IMAGINARIA

Efectuando la división del Fasor de la señal de Salida entre el Fasor de la señal de Entrada: $V_i(j\omega_o) = A_m \angle 0^\circ$, se obtiene el Fasor de la Función de Transferencia : $H(j\omega_o) = V_o(j\omega_o) / V_i(j\omega_o)$ El proceso y representación-fasor se muestra en la { *figura* 5-7 }.

{ figura 5-7 } Representación Fasorial de la Función de Transferencia Senoidal

Representación Fasorial:

$\frac{\text{SALIDA}}{\text{ENTRADA}}$ = TRANSFERENCIA



Análisis a una determinada Frecuencia angular ω_o :

RESPUESTA (SALIDA) DEL SISTEMA

Magnitud : $M_o = A_m|H(j\omega_o)|$

Ang. Fase : $\phi_o = \phi$; $\phi = F(\omega_o)$

Fasor: $V_o(j\omega_o) = A_m|H(j\omega_o)| \cdot e^{j\phi}$

$= M_o\, e^{j\phi_o} = M_o \angle \phi_o$

$s = j\omega$

$\omega = 2\pi F$

$F = \frac{1}{T}$

$\omega = \omega_o \left[\frac{rad}{s}\right]$

@ F_o [Hz]

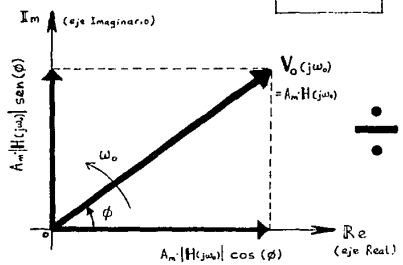
EXCITACION (ENTRADA) AL SISTEMA

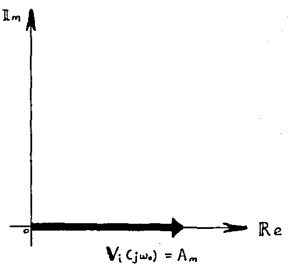
Magnitud : $M_i = A_m$

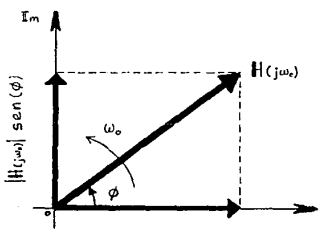
Ang. Fase : $\phi_i = 0°$ (Cero) → Referencia

Fasor: $V_i(j\omega_o) = A_m \cdot e^{j0°}$

$= M_i\, e^{j\phi_i} = M_i \angle \phi_i$







FUNCION DE TRANSFERENCIA SENOIDAL :

Magnitud: $|H(j\omega_o)| = \frac{|V_o|}{|V_i|} = \frac{M_o}{M_i}$

Angulo de Fase: $\angle H(j\omega_o) = \angle \frac{V_o}{V_i} =$

$= \phi_o - \phi_i = \phi$

Fasor: $H(j\omega_o) = |H(j\omega_o)| \cdot e^{j\phi}$

$H(j\omega_o) = \frac{V_o(j\omega_o)}{V_i(j\omega_o)}$	$\lvert H(j\omega_o)\rvert = \left\lvert \frac{V_o(j\omega_o)}{V_i(j\omega_o)} \right\rvert$ Relación de Amplitud de la Sinusoide de Salida a la Sinusoide de Entrada.	$\angle H(j\omega_o) = \angle \frac{V_o(j\omega_o)}{V_i(j\omega_o)}$ Relación de la Fase de la Sinusoide de Salida con respecto a la Sinusoide de Entrada.

Para obtener físicamente las componentes de $H(j\omega_o)$, es necesario multiplicar la Ec.(1) por $\{ \operatorname{sen}(\omega_o t) \}$ y $\{ \cos(\omega_o t) \}$ y posteriormente filtrar los productos resultantes.

Asi se tiene :

a) Empleando la identidad trigonométrica :

$$\operatorname{sen}(\alpha)\,\operatorname{sen}(\beta) = \frac{1}{2}\cos(\alpha-\beta) - \frac{1}{2}\cos(\alpha+\beta)$$

$$\operatorname{sen}(\omega_o t)\left[A_m\,|H(j\omega_o)|\,\operatorname{sen}(\omega_o t+\phi) \right] \qquad ; \quad \phi = \angle H(j\omega_o)$$

$$= \frac{1}{2} A_m\,|H(j\omega_o)| \left[\cos(\omega_o t+\phi-\omega_o t) - \cos(\omega_o t+\phi+\omega_o t) \right]$$

$$= \frac{1}{2} A_m\,|H(j\omega_o)| \left[\cos(\phi) - \cos(2\omega_o t+\phi) \right] = \{S_1(t)\} \qquad (4)$$

b) Empleando la identidad trigonométrica :

$$\operatorname{sen}(\alpha)\cos(\beta) = \frac{1}{2}\operatorname{sen}(\alpha+\beta) + \frac{1}{2}\operatorname{sen}(\alpha-\beta)$$

$$\cos(\omega_o t)\left[A_m\,|H(j\omega_o)|\,\operatorname{sen}(\omega_o t+\phi) \right] \qquad ; \quad \phi = \angle H(j\omega_o)$$

$$= \frac{1}{2} A_m\,|H(j\omega_o)| \left[\operatorname{sen}(\omega_o t+\phi-\omega_o t) + \operatorname{sen}(\omega_o t+\phi+\omega_o t) \right]$$

$$= \frac{1}{2} A_m\,|H(j\omega_o)| \left[\operatorname{sen}(\phi) + \operatorname{sen}(2\omega_o t+\phi) \right] = \{S_2(t)\} \qquad (5)$$

Las ecuaciones (4) y (5) constan de dos componentes : una de valor promedio y otra de alta frecuencia. Las componentes que interesan son las que representan el valor promedio de cada producto, por lo que es necesario utilizar un filtro Pasa-Bajas para eliminar las componentes no deseadas.

De esta manera, a la salida de los filtros Pasa-Bajas resultan términos proporcionales a las componentes Real e Imaginaria de la Función de Transferencia del sistema $H(j\omega_o)$:

Del producto $S_1(t)$ [Ec.(4)] queda :

$$\frac{1}{2} A_m \left| H(j\omega_o) \right| \cos(\phi) \quad \ldots\ldots \text{ Componente Real}$$

Del producto $S_2(t)$ [Ec.(5)] queda :

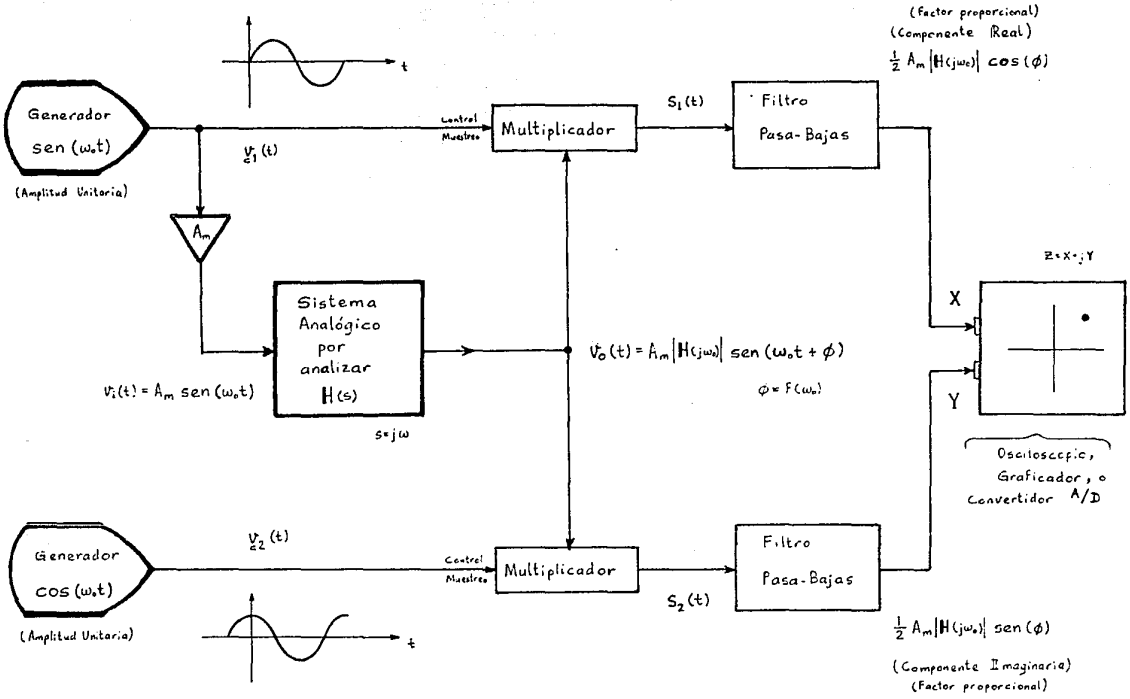
$$\frac{1}{2} A_m \left| H(j\omega_o) \right| \operatorname{sen}(\phi) \quad \ldots\ldots \text{ Componente Imaginaria}$$

Si se conectan las salidas de los filtros Pasa-Bajas a los canales X (Re) y Y (Im) de un graficador u osciloscopio, en la pantalla aparecerá el extremo del Fasor de interés (una cantidad proporcional).

.....

En la { *figura* 5-8 } se muestra el diagrama del procedimiento para realizar lo anterior .
Diagrama de bloques conceptual del instrumento.

.....



{ *figura* 5-8 }
Diagrama de bloques conceptual del dispositivo

5.4.4

BASE TEORICO-MATEMATICA PARA LA IMPLEMENTACION FISICA DEL INSTRUMENTO Y SELECCION DE LA FUENTE DE SEÑAL

Para llevar a cabo la multiplicación de la Ecuación (1) por { $\text{sen}(\omega_o t)$ } y por { $\cos(\omega_o t)$ } es necesario generar dos señales senoidales de la misma amplitud defasadas $\frac{\pi}{2}$ [radianes] (90°) para toda frecuencia ω_o .

Nótese que si las señales que aparecen en la { *figura* 5-8 } no fuesen SENOIDALES sino CUADRADAS, defasadas $\frac{\pi}{2}$ [rad] , las señales que aparecerian a la salida de los Filtros Pasa-Bajas, no sufrirían ningún cambio importante, ya que tan solo diferirían en el factor de proporcionalidad incorporado, que, como en el caso de las señales senoidales, se puede determinar fácilmente a partir de un análisis matemático : (Multiplicación con señales Cuadradas aproximadas por su serie de Fourier). Más adelante se demostrará esto con más detalle.

En el desarrollo del prototipo se optó por trabajar con señales cuadradas para las funciones $v_{c_1}(t)$ y $v_{c_2}(t)$ (ver *figura* 5-8), debido a que los circuitos necesarios para generar un defasaje de $\frac{\pi}{2}$ [rad] en la gama de frecuencias en las que opera el dispositivo, resultaron ser más económicos, sencillos y de comportamiento más estable, que los necesarios para generar dos señales senoidales defasadas el mismo ángulo.

THE LINEAR CONTROL CIRCUITS DATA BOOK FOR ENGINEERS

TEXAS INSTRUMENTS, 2ND. ED., PP. 145 , 387-391

FUNCTION GENERATOR SYSTEMS DATA BOOK

EXAR INTEGRATED SYSTEMS (MAY 1980)

INTERSIL DATA BOOK (PP. S-232 , S-240)

OPERATIONAL AMPLIFIERS , DESIGN AND APPLICATIONS

TOBEY-GRAEME-HUELSMAN (MC.GRAW HILL) , 1981

Con el fin de generar las dos señales cuadradas con el defasamiento deseado, se utilizó el Generador de Funciones monolítico (IC) " XR-2206 ".

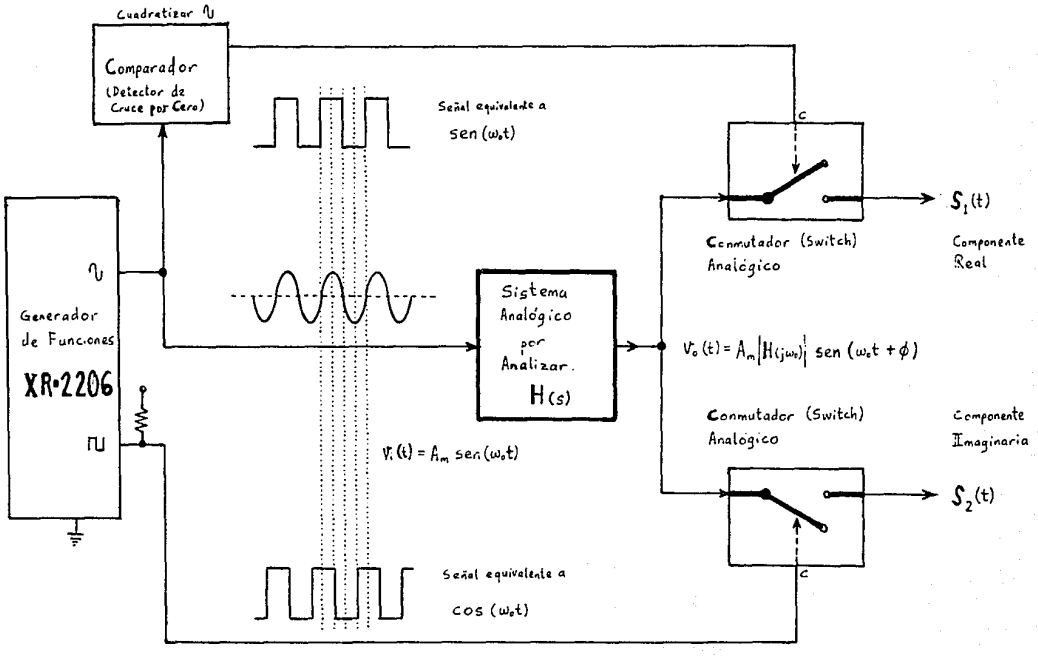
> MONOLITHIC FUNCTION GENERATOR XR-2206

- EXAR'S DATA BOOK
- EXAR'S OSCILLATOR PRODUCTS (HANDBOOK)

El circuito integrado XR-2206 , puede producir simultáneamente dos señales de la misma frecuencia pero de forma diferente : { Senoidal y Cuadrada } ó { Triangular y Cuadrada }. Para el instrumento se empleó la opción para generación de ondas { Senoidal y Cuadrada }, que como puede apreciarse en la { *figura* 5-9 } , son proporcionadas directamente (por el XR-2206) con el defasamiento requerido de $\frac{\pi}{2}$ [rad] = 90° .

Asi, del XR-2206 se obtiene directamente una de las señales cuadradas { la que representa a $\cos(\omega_o t)$ } . La otra señal cuadrada necesaria { la que representa a $\text{sen}(\omega_o t)$ } , se obtiene a la salida de un Comparador con umbral ó V_{REF} de cero Volt y cuya entrada es la señal senoidal proporcionada por el XR-2206 (Comparador en configuración : Detector de Cruce por Cero). La señal senoidal debe proveerse al comparador, sin componente de DC, es decir, como una señal pura de AC (0 Volt de offset).

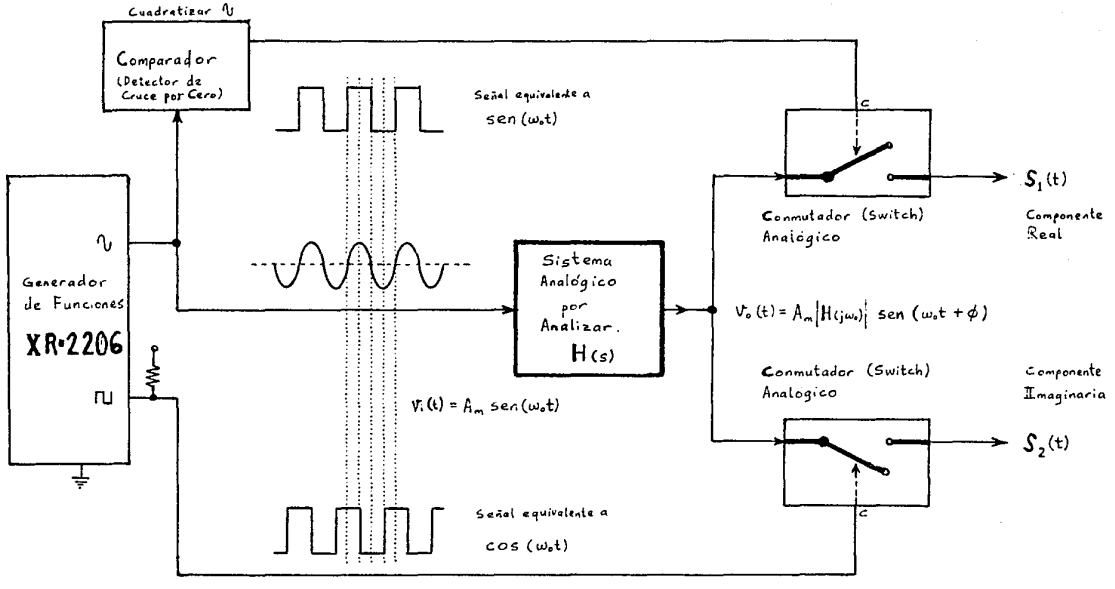
Para multiplicar las señales cuadradas por la salida del "Sistema Analógico bajo análisis" , se empleó la configuración mostrada en la { *figura* 5-9 } , que consiste en la utilización de conmutadores analógicos (switches) controlados por las mismas señales cuadradas generadas, (acondicionadas para entregar a las



{ *figura* 5-9 }

Diagrama de bloques del circuito para realizar la multiplicación de $v_o(t)$ por dos señales cuadradas defasadas : $\frac{\pi}{2}$ [rad] = 90° entre sí

$$\left\{ V_o(t) = A_m \left| H(j\omega_o) \right| \operatorname{sen}(\omega_o t + \phi) \right\}$$



{ *figura* 5-9 }

Diagrama de bloques del circuito para realizar la multiplicación de $v_o(t)$ por dos señales cuadradas defasadas : $\frac{\pi}{2}$ [rad] = 90° entre sí

$$\left\{ V_o(t) = A_m \left| H(j\omega_o) \right| \operatorname{sen}(\omega_o t + \phi) \right\}$$

entradas de control de los switches, niveles adecuados de voltaje : "1-lógico (+5V) : SW.closed" y "0-lógico (-5V) : SW.open") . { CD-4016 *Analog Switches* [National Semiconductor Corp.] }.

De esta manera, los *Switches* analógicos muestrean a la señal de salida del "Sistema bajo análisis" :

$$v_o(t) = A_m \, |H(j\omega_o)| \, \text{sen}\left(\omega_o t + \phi\right)$$

producida por la señal aplicada a la entrada del sistema :

$$v_i(t) = A_m \, \text{sen}(\omega_o t)$$

que proviene del generador de funciones XR-2206 (pero acondicionada para entregarla sin componente de DC), y que se encuentra en fase con una de las dos señales cuadradas.

El resultado de las multiplicaciones se muestra a continuación :

a) $S_1(t)$ = [Respuesta Senoidal del Sistema bajo análisis] * [Desarrollo en Serie Trigonométrica de Fourier de la ONDA CUADRADA que representa al SENO : " $\text{sen}(\omega_o t)$ "]

$$S_1(t) = \left[A_m \, |H(j\omega_o)| \, \text{sen}(\omega_o t + \phi) \right] * \left[\frac{M}{2} + \frac{2M}{\pi} \left\{ \text{sen}(\omega_o t) + \frac{1}{3}\,\text{sen}(3\omega_o t) + \frac{1}{5}\,\text{sen}(5\omega_o t) + \frac{1}{7}\,\text{sen}(7\omega_o t) + \ldots \right\} \right]$$

M = MAGNITUD DE LA ONDA CUADRADA

$\frac{M}{2}$ = COMPONENTE DE DC = VALOR PROMEDIO DE f(t) (O.CUADRADA) DURANTE UN PERIODO

Considerar que M = 1 para la onda Cuadrada ya que es usada como una señal para conmutación (control lógico) de circuitos

{Muestreo = Dejar o No-Dejar pasar la señal} = Multiplicar por "1" ó por "0" la señal senoidal muestreada.

Empleando la identidad trigonométrica :

$\operatorname{sen}(\alpha)\,\operatorname{sen}(\beta) = (1/2)\cos(\alpha-\beta) - (1/2)\cos(\alpha+\beta)$

y tomando $\alpha = (\omega_o t+\phi)$, se tiene :

$$S_1(t) = A_m\,|H(j\omega_o)|\left\{\frac{1}{2}\operatorname{sen}(\omega_o t+\phi) + \frac{1}{\pi}\left[\cos(\phi) - \cos(2\omega_o t+\phi)\right]\right.$$

$$+ \frac{1}{3\pi}\left[\cos(-2\omega_o t+\phi) - \cos(4\omega_o t+\phi)\right]$$

$$+ \frac{1}{5\pi}\left[\cos(-4\omega_o t+\phi) - \cos(6\omega_o t+\phi)\right]$$

$$\left.+ \frac{1}{7\pi}\left[\cos(-6\omega_o t+\phi) - \cos(8\omega_o t+\phi)\right] + \ldots\right\}$$

Considerando que : $\cos(-\theta) = +\cos(\theta)$

se tiene para los términos :

$$\cos(-a+b) = \cos(-(a-b)) = \cos(a-b)$$

entonces $S_1(t)$ queda finalmente :

$$S_1(t) = A_m\,|H(j\omega_o)|\left\{\frac{1}{2}\operatorname{sen}(\omega_o t+\phi) + \frac{1}{\pi}\left[\cos(\phi) - \cos(2\omega_o t+\phi)\right]\right.$$

$$+ \frac{1}{3\pi}\left[\cos(2\omega_o t-\phi) - \cos(4\omega_o t+\phi)\right]$$

$$+ \frac{1}{5\pi}\left[\cos(4\omega_o t-\phi) - \cos(6\omega_o t+\phi)\right]$$

$$\left.+ \frac{1}{7\pi}\left[\cos(6\omega_o t-\phi) - \cos(8\omega_o t+\phi)\right] + \ldots\right\}$$

De esta expresión el único término que no varia con el tiempo " t " (término constante de DC para una frecuencia angular ω_o) es :

$$\frac{1}{\pi} A_m \left|H(j\omega_o)\right| \cos(\phi)$$

Recordar que el Angulo de Fase ϕ es función de la frecuencia ω_o , es decir : $\phi = f(\omega_o) = \phi(\omega_o)$

b) $S_2(t)$ = [Respuesta Senoidal del Sistema bajo análisis] * [Desarrollo en Serie Trigonométrica de Fourier de la ONDA CUADRADA que representa al COSENO : " $\cos(\omega_o t)$ "]

$$S_2(t) = \left[A_m \left|H(j\omega_o)\right| \operatorname{sen}(\omega_o t+\phi) \right] * \left[\frac{M}{2} + \frac{2M}{\pi} \left(\cos(\omega_o t) - \frac{1}{3}\cos(3\omega_o t) + \frac{1}{5}\cos(5\omega_o t) - \frac{1}{7}\cos(7\omega_o t) + \ldots \right) \right]$$

M = MAGNITUD DE LA ONDA CUADRADA

$\frac{M}{2}$ = COMPONENTE DE DC = VALOR PROMEDIO DE f(t) (O.CUADRADA) DURANTE UN PERIODO

Considerar que M = 1 para la onda Cuadrada ya que es usada como una señal para conmutación (control lógico) de circuitos {Muestreo = Dejar o No-Dejar pasar la señal} = Multiplicar por "1" ó por "0" la señal senoidal muestreada.

Empleando la identidad trigonométrica :
$\operatorname{sen}(\alpha)\cos(\beta) = (1/2)\operatorname{sen}(\alpha+\beta) + (1/2)\operatorname{sen}(\alpha-\beta)$
y tomando $\alpha = (\omega_o t+\phi)$, se tiene :

$$S_2(t) = A_m \, |H(j\omega_o)| \left\{ \frac{1}{2} \operatorname{sen}(\omega_o t+\phi) + \frac{1}{\pi}\left[\operatorname{sen}(\phi) + \operatorname{sen}(2\omega_o t+\phi) \right] \right.$$

$$- \frac{1}{3\pi} \left[\operatorname{sen}(-2\omega_o t+\phi) + \operatorname{sen}(4\omega_o t+\phi) \right]$$

$$+ \frac{1}{5\pi} \left[\operatorname{sen}(-4\omega_o t+\phi) + \operatorname{sen}(6\omega_o t+\phi) \right]$$

$$\left. - \frac{1}{7\pi} \left[\operatorname{sen}(-6\omega_o t+\phi) + \operatorname{sen}(8\omega_o t+\phi) \right] + \ldots \right\}$$

Considerando que : $\operatorname{sen}(-\theta) = -\operatorname{sen}(\theta)$

y factorizando, se tiene para los términos :

$$[\ \operatorname{sen}(-a+b) + \operatorname{sen}(c+d)\ ] = [\ \operatorname{sen}(-(a-b)) + \operatorname{sen}(c+d)\ ]$$
$$= [\ -\operatorname{sen}(a-b) + \operatorname{sen}(c+d)\ ]$$
$$= -\ [\ \operatorname{sen}(a-b) - \operatorname{sen}(c+d)\ ]$$

entonces $S_2(t)$ queda finalmente :

$$S_2(t) = A_m \, |H(j\omega_o)| \left\{ \frac{1}{2} \operatorname{sen}(\omega_o t+\phi) + \frac{1}{\pi}\left[\operatorname{sen}(\phi) + \operatorname{sen}(2\omega_o t+\phi) \right] \right.$$

$$+ \frac{1}{3\pi} \left[\operatorname{sen}(2\omega_o t-\phi) - \operatorname{sen}(4\omega_o t+\phi) \right]$$

$$- \frac{1}{5\pi} \left[\operatorname{sen}(4\omega_o t-\phi) - \operatorname{sen}(6\omega_o t+\phi) \right]$$

$$\left. + \frac{1}{7\pi} \left[\operatorname{sen}(6\omega_o t-\phi) - \operatorname{sen}(8\omega_o t+\phi) \right] - \ldots \right\}$$

De esta expresión el único término que no varía con el tiempo " t " (término constante de DC para una frecuencia angular ω_o) es :

$$\frac{1}{\pi} A_m |H(j\omega_o)| \operatorname{sen}(\phi)$$

Recordar que el Angulo de Fase ϕ es función de la frecuencia ω_o , es decir : $\phi = f(\omega_o) = \phi(\omega_o)$

Resumiendo :

De las expresiones anteriores, los únicos términos que interesan son :

$$\frac{1}{\pi} A_m |H(j\omega_o)| \cos(\phi) \qquad y \qquad \frac{1}{\pi} A_m |H(j\omega_o)| \operatorname{sen}(\phi)$$

por lo que es necesario eliminar los términos restantes (de alta frecuencia), mediante filtros Pasa-Bajas.

Las salidas de los filtros Pasa-Bajas serán proporcionales a :

$$|H(j\omega_o)| \cos(\phi) \qquad y \qquad |H(j\omega_o)| \operatorname{sen}(\phi)$$

COMPONENTE REAL CANAL X — COMPONENTE IMAGINARIA CANAL Y

que al aplicarse respectivamente a los canales X e Y de un graficador u osciloscopio, originarán un punto cuyas coordenadas son proporcionales a las que tiene el extremo del Fasor correspondiente a :

$$|H(j\omega_o)| \angle H(j\omega_o)$$

{ Función de Transferencia del Sistema bajo análisis }

En el instrumento diseñado, los valores de las componentes, procedentes de los filtros Pasa-Bajas (voltajes DC), serán leídos

por dos entradas analógicas contiguas del Convertidor Analógico/Digital del sistema **SDM88-PC** (ADC-0809), para posteriormente procesar digitalmente esa información (en la microcomputadora), compensando factores de proporcionalidad y haciendo los escalamientos pertinentes para mostrar al usuario, en forma gráfica y numérica, las curvas de Respuesta en frecuencia (Magnitud y Angulo de Fase ⇒ Función de Transferencia), del sistema bajo análisis.

5.4.5

ANALIZADOR DIGITAL DE RESPUESTA EN FRECUENCIA : DESCRIPCION DEL " HARDWARE " DE CONTROL

La manera más rápida de obtener un prototipo depurado y operando es " una pequeña parte a la vez ". Debido a que los problemas tienden a interactuar, tratar de depurar una sección muy grande de una sola vez, puede ser frustrante y consumir demasiado tiempo.

Por tanto, a fin de proceder de una manera sistemática en el diseño e implementación del prototipo, se tomó como soporte de *Software* y como *Hardware* base, al Sistema de Desarrollo **SDM88-PC** , (que es una arquitectura, ya probada y depurada, controlada por microprocesador que cuenta con todos los periféricos (memoria y puertos I/O) necesarios para interactuar con, y en su caso controlar a, otros sistemas "satélite", según especificaciones dadas (programadas) por el usuario). A partir de este *Hardware* "base", se fueron añadiendo, probando y evaluando "*de una en una*", las etapas que conforman el instrumento de aplicación, hasta obtener la versión final completa; esto, con el fin de facilitar la identificación de errores y su corrección, así como la calibración de las distintas secciones del instrumento.

Conforme se iba obteniendo una sección de *Hardware* funcionando, se escribia y depuraba un módulo de *Software* que usase o manejase ese módulo de *Hardware*, si éste podía ser controlado digitalmente; si, en cambio, dicho módulo era puramente analógico, se probaba por separado aislándolo de las secciones ya depuradas, y, una vez caracterizada y calibraba su respuesta (con ayuda de osciloscopio, generador de funciones, voltímetro, etc.), se integraba al circuito base ya implementado y probado, para inmediatamente volver a examinar la operación de todo el circuito después de la inclusión de la nueva sección dada de alta.

.....

El *Hardware* consta de varias etapas que se encargarán fundamentalmente :

1) del suministro y acondicionamiento de la señal senoidal de prueba aplicada a la entrada del circuito o sistema bajo análisis
2) de la realización de un barrido de frecuencia con esa señal sobre el circuito (sistema) por analizar
3) del muestreo, filtrado, amplificación y adecuación de la señal de salida del sistema probado, a fin de poder digitalizarla con el convertidor A/D del SDM88-PC
4) de la medición de frecuencia de la señal de prueba aplicada, durante el proceso de análisis interactivo del usuario sobre las curvas de respuesta graficadas en la microcomputadora

La información digital será adquirida, almacenada secuencialmente y transmitida hacia la microcomputadora en forma serial, por la interface de *Hardware* del sistema SDM88-PC , para su posterior procesamiento, escalamiento y despliegue en la microcomputadora.

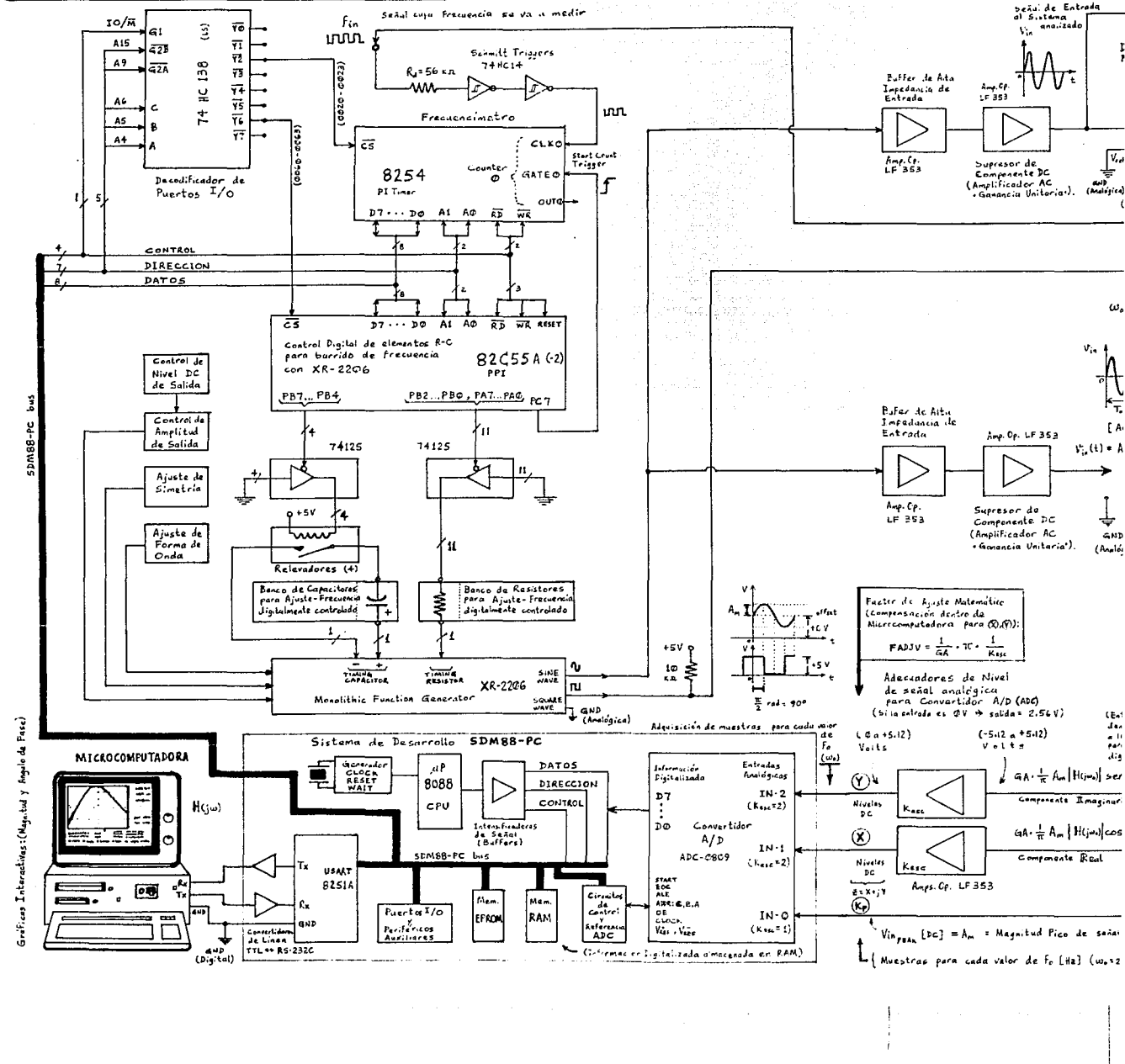
A continuación se muestra un diagrama modular funcional completo del instrumento : **"Analizador Digital de Respuesta en Frecuencia"** { *figura* 5-10 } (Aplicación para el sistema SDM88-PC)

Analizador Digital de Respuesta en Frecuencia

Magnitud y Angulo de Fase (Bode → Nyquist)

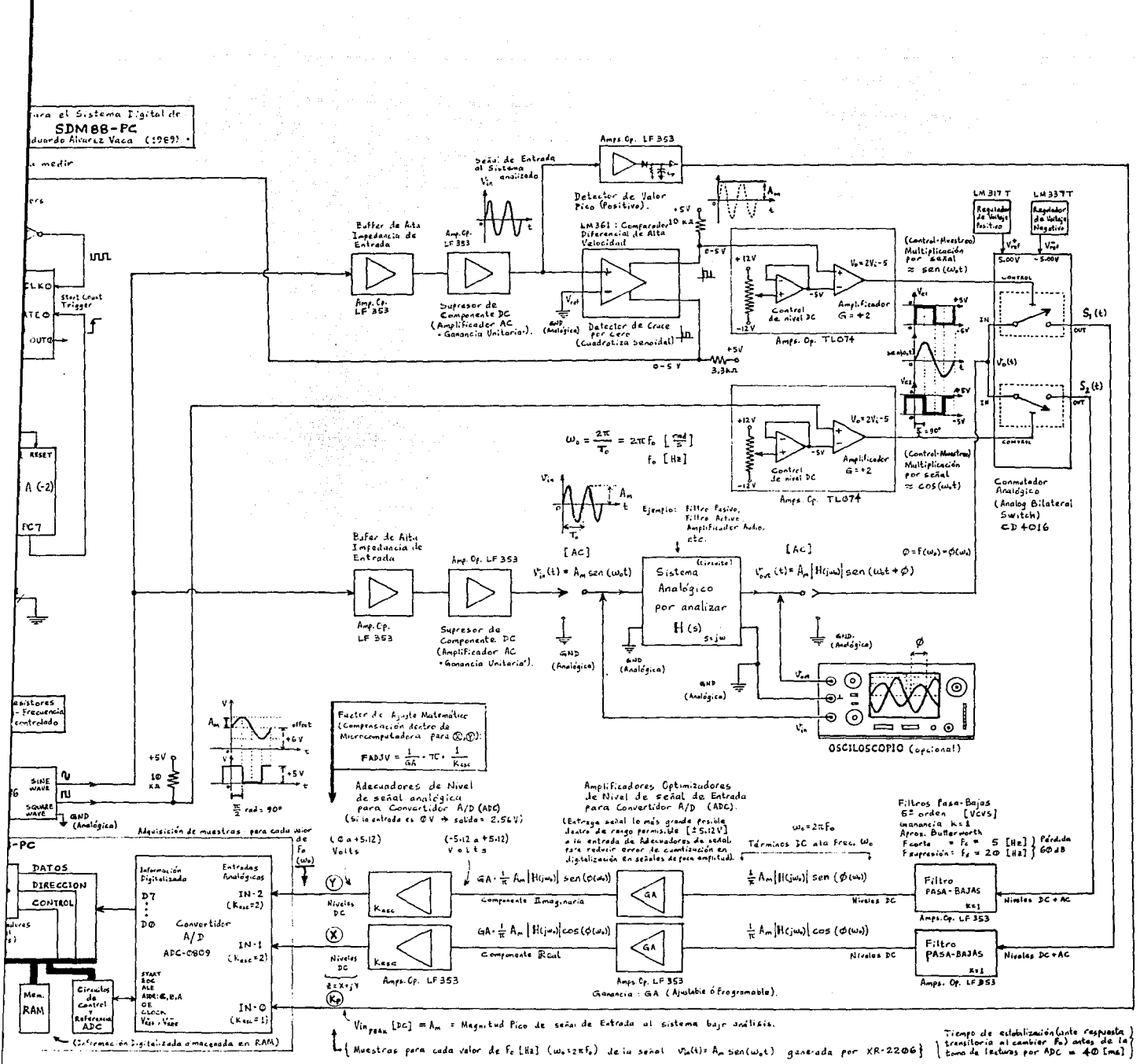
Aplicación para el Sistema Digital de Desarrollo: **SDM88-PC**

Agustín Eduardo Alvarez Vaca (1989).



ura el Sistema Digital de
SDM88-PC
duardo Alvarez Vaca (1989)

e medir



$\omega_o = \frac{2\pi}{T_o} = 2\pi F_o \ [\frac{rad}{s}]$ $\quad F_o\ [Hz]$

$V_{in}(t) = A_m \operatorname{sen}(\omega_o t)$ [AC]

$V_{out}(t) = A_m |H(j\omega)| \operatorname{sen}(\omega t + \phi)$ [AC]

$\phi = F(\omega_o) = \phi(\omega)$

$FADJV = \frac{1}{GA} \cdot TC \cdot \frac{1}{K_{esc}}$

$\omega_o = 2\pi F_o$ Términos DC ala Frec. ω_o

$\frac{1}{\pi} A_m |H(j\omega_o)| \operatorname{sen}(\phi(\omega_o))$

$GA \cdot \frac{1}{\pi} A_m |H(j\omega_o)| \operatorname{sen}(\phi(\omega_o))$

$\frac{1}{\pi} A_m |H(j\omega_o)| \cos(\phi(\omega_o))$

$GA \cdot \frac{1}{\pi} A_m |H(j\omega_o)| \cos(\phi(\omega_o))$

$Z = X + jY$

$Vin_{PEAK}\ [DC] = A_m$ = Magnitud Pico de señal de Entrada al sistema bajo análisis.

{ Muestras para cada valor de F_o [Hz] ($\omega_o = 2\pi F_o$) de la señal $V_{in}(t) = A_m \operatorname{sen}(\omega_o t)$ generada por XR-2206 }

{ Tiempo de estabilización (ante respuesta transitoria al cambiar F_o) antes de la toma de lecturas por ADC = 40 [ms] }

{ Información Digitalizada almacenada en RAM }

El **"Analizador Digital de Respuesta en Frecuencia "** , al que de ahora en adelante se hará referencia como "ADRF" , consta de una sección digital y de otra puramente analógica.

La sección digital recibe directamente del **SDM88-PC** , todo el Bus (canal) de Datos (D7-D0), parte del Bus de Direcciones (A0,A1,A4,A5,A6,A9,A15), y parte del Bus de Control ($\overline{RD}$,$\overline{WR}$,RESET,IO/$\overline{M}$), y consta de :

- un Decodificador 3X8 (74HC138), usado como decodificador de puertos I/O (periféricos para el ADRF controlados por microprocesador : 8254 y 8255)

- un 8254 (*Programmable Interval Timer*) usado como contador de pulsos para la sección de medición de frecuencia del ADRF (frecuencimetro)

- un 82C55A (-2) (*Programmable Peripheral Interface*) con 3 puertos paralelos programables como I/O de 8-bit cada uno, usado como parte de la lógica de control para fijar la frecuencia de operación del XR-2206 y para proporcionar (bajo control-sincronización del microprocesador), el pulso de disparo para iniciar el cómputo de frecuencia (conteo de pulsos) de la señal de prueba generada por el XR-2206

- Lógica de Control para habilitación/deshabilitación de componentes R-C que determinan la frecuencia de la señal entregada por el XR-2206

Las distintas etapas de la sección analógica se describirán funcionalmente a continuación, indicando asimismo su relación con la sección digital.

Para que un graficador u osciloscopio (canales X,Y) dibuje la traza polar o bien para que se puedan dibujar las trazas de Magnitud y Fase en la pantalla de la microcomputadora, se requiere llevar a cabo un "Barrido de Frecuencia", partiendo de un valor inicial "f1 . (ω_i)" a otro valor final "f2 . (ω_f)".

Para realizar la prueba de Respuesta en frecuencia, se debe disponer de generadores de señal senoidal adecuados. Esta señal debe estar razonablemente libre de armónicas o distorsión. Ahora, para crear un instrumento autónomo, supervisado y sincronizado en sus funciones por microprocesador, el Generador de Funciones debe estar accesible en forma de Circuito Integrado a fin de tener un mayor control sobre él, cosa que sería más complicada con un generador de señal externo (aparato).

Con estas consideraciones en mente, se escogió para el diseño al Generador de Funciones Monolítico " XR-2206 " , como fuente de señal (senoidal) para realizar el barrido de frecuencia sobre el sistema (circuito) bajo análisis, agregándole la circuitería analógica y digital necesaria para que el SDM88-PC pueda controlarlo digitalmente.

El Generador de Funciones Monolítico (en circuito integrado) XR-2206, puede producir onda Senoidal (y Cuadrada) de buena calidad (Baja distorsión armónica senoidal, tipicamente 0.5%) y susceptible de ajuste para lograr buena simetría y forma de onda, además de poder controlarse la amplitud de la señal (senoidal) de salida y hacerse un ajuste fino del nivel de offset (DC) alrededor del punto medio de los niveles de voltaje aplicados a las terminales de fuente de alimentación para el circuito ({0,+12V...,26Vmax} configuración fuente unipolar ó {+12,-12V} fuente bipolar). En el caso del ADRF , se usó la configuración de fuente unipolar con un voltaje de polarización de +12 V , por lo que la señal entregada por el XR-2206 tiene un offset DC de 6 V .

La amplitud de la señal senoidal de salida puede ajustarse en un rango de hasta 3 V pico (swing de 6 V pico a pico), (aprox. 60 mV pico por cada kΩ aumentado al resistor de control de amplitud {hasta 50 kΩ}). La amplitud de la onda cuadrada depende de la fuente de alimentación a que se conecte su terminal de salida a través de un resistor de *pull-up* ; el swing típico de salida es de 12 V pico a pico, pero en el caso del ADRF fue de 0 a +5 V pues la salida de onda cuadrada se conectó a la fuente de +5 V mediante un resistor de 10 kΩ .

La Frecuencia de operación para el XR-2206, puede ser seleccionada-ajustada externamente, en un rango de 0.01 Hz a 1 MHz : { Máx.frec. 1MHz con C=1nF y R1=1kΩ } , { Min.frec. 0.01Hz con C= 50μF y R1=2MΩ } ... (frecuencia de señal de salida = $f_o = \frac{1}{R_1 C}$ [Hz] con un factor de precisión de ± 2%) .

En el ADRF, la frecuencia de operación para el XR-2206 es seleccionada bajo programación por el **SDM88-PC** . Esto se logra a través de una circuitería digital intermedia de control (8255, 74125, relevadores disparados por niveles TTL), que se encarga de conectar y/o aislar Resistores y Capacitores con valores ponderados en forma de múltiplos, que, concentrados como "Bancos de elementos", determinan la frecuencia de oscilación (operación) del XR-2206. De esta manera, la frecuencia del generador, es controlada digitalmente (por *Software*).

Para realizar un barrido de frecuencia, basta mandar a través de los puertos paralelos de salida del 8255, la información de control (bytes para ajuste de frecuencia en XR-2206) para conmutar (switchear) Resistores y Capacitores de sendos bancos de dichos elementos conectados al IC XR-2206 para así, fijar su frecuencia de oscilación (operación). El proceso de conmutación o *switcheo* se efectúa en forma escalonada (por rangos); de este modo se

tienen 2047 posibilidades de conexión factibles para los Resistores : (00000000001 a 11111111111) (2047 > 11 bits) para cada uno de los 4 rangos fijados por los capacitores de temporización : (puede haber 2 a 11 resistores de temporización conectados en paralelo y a GND).

Se usa un PPI 82C55A como puerto paralelo de control para suministrar la "palabra (Word) de Frecuencia" al XR-2206, ya que es un circuito de operación muy estable que mantiene (latcheados) en sus puertos de salida los niveles de voltaje suministrados a través del Bus de Datos por escritura direccionada { WRite _ OUT } y aisla esos niveles de toda perturbación debida a la actividad (cambios de nivel) existente en dicho Bus (canal) ; así, la estabilidad en los niveles de voltaje y el aislamiento al ruido de conmutación digital, se refleja en la calidad (estabilidad en frecuencia y limpieza) de la señal analógica de salida entregada por el XR-2206 .

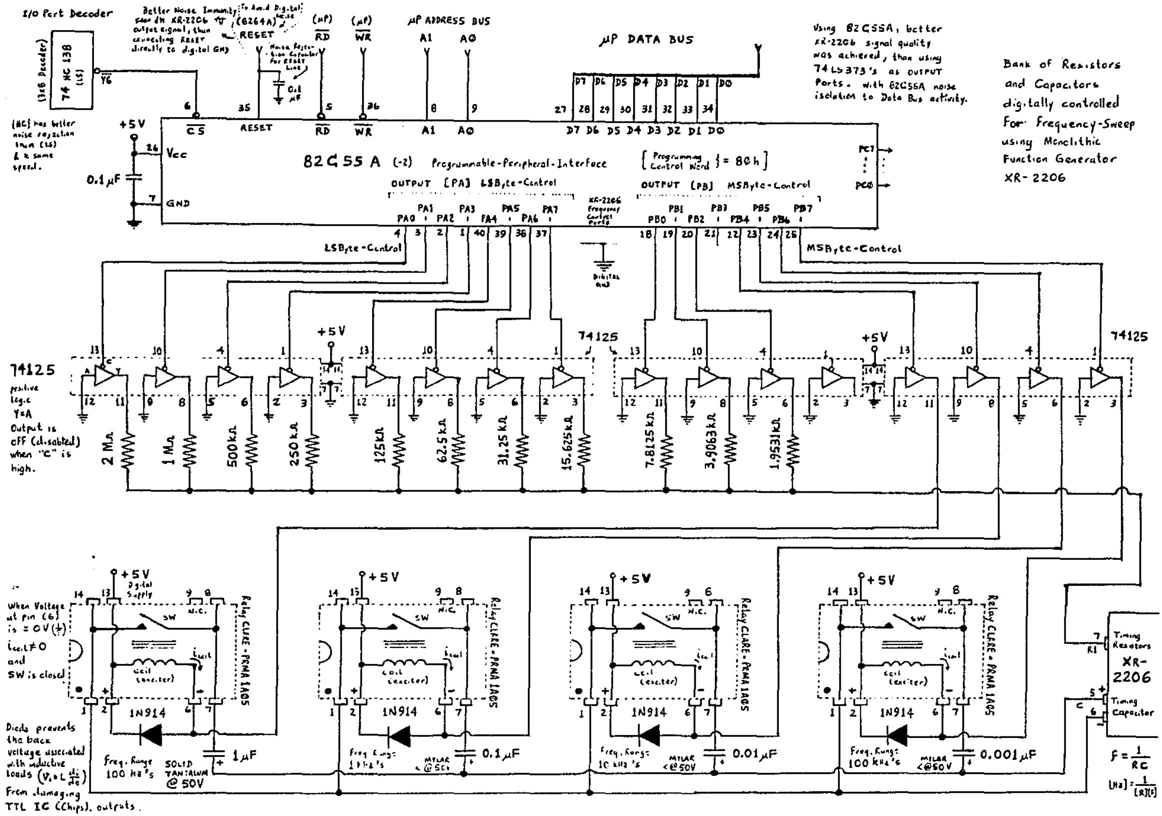
.....

En la { *figura* 5-11 } se muestra un diagrama detallado del circuito utilizado para el control digital de la frecuencia de la señal de salida (senoidal y cuadrada) en el XR-2206.

.....

En la figura se observa que existen 4 capacitores de temporización, cada uno de los cuales, (o varios en paralelo), puede(n) ser conectado(s) al XR-2206 (pin 5,6 = timing capacitor) para definir un nuevo rango de frecuencia de operación :
De manera aproximada y (+/- cuantitativa) :

1	[microFarad]	=>	0.5	—→ 500	[Hz]
0.1	[microFarad]	=>	5	—→ 5000	[Hz]
0.01	[microFarad]	=>	50	—→ 50000	[Hz]
0.001	[microFarad]	=>	500	—→ 500000	[Hz]



Bank of Resistors and Capacitors digitally controlled For Frequency-Sweep using Monolithic Function Generator XR-2206
I/O Port Decoder
74 HC 138 (LS)
(3x8 Decoder)
Y6
(HC) has better noise rejection than (LS) & a same speed.
Better Noise Immunity seen in XR-2206 output signal, than connecting RESET directly to digital GND
RESET
(μP) RD
(μP) WR
μP ADDRESS BUS
A1
A0
μP DATA BUS
D7 D6 D5 D4 D3 D2 D1 D0
Using 82C55A, better XR-2206 signal quality was achieved, than using 74LS373's as OUTPUT Ports. With 82C55A noise isolation to Data Bus activity.
+5V
0.1μF
Vcc
GND
CS
RD
WR
82C55A (-2) Programmable-Peripheral-Interface
[Programming Control Word] = 80h
OUTPUT [PA] LSByte-Control
OUTPUT [PB] MSByte-Control
PC7
PC0
PA0 PA1 PA2 PA3 PA4 PA5 PA6 PA7
PB0 PB1 PB2 PB3 PB4 PB5 PB6 PB7
XR-2206 Frequency Control Ports
LSByte-Control
MSByte-Control
DIGITAL GND
74125
74125 positive logic Y=A Output is OFF (disabled) when "C" is high.
2 MΩ
1 MΩ
500 kΩ
250 kΩ
125 kΩ
62.5 kΩ
31.25 kΩ
15.625 kΩ
7.8125 kΩ
3.9063 kΩ
1.9531 kΩ
When Voltage at pin (6) is ≈ 0V (L) $i_{coil} \neq 0$ and SW is closed
+5V Digital Supply
SW
N.C.
Coil (exciter)
i_{coil}
Relay CLARE-PRMA 1A05
1N914
Diode prevents the back voltage associated with inductive loads $(V_L = L\frac{di}{dt})$ From damaging TTL IC (Chips) outputs.
Freq. Range 100 Hz's
SOLID TANTALUM @ 50V
1μF
Freq. Range 1 kHz's
MYLAR <@50V
0.1μF
Freq. Range 10 kHz's
MYLAR <@50V
0.01μF
Freq. Range 100 kHz's
MYLAR <@50V
0.001μF
Timing Resistors
XR-2206
Timing Capacitor
$f = \frac{1}{RC}$
$[Hz] = \frac{1}{[R][F]}$

El rango base se establece con la habilitación de uno (o varios) de los capacitores y, posteriormente, si se cambian (habilitan/deshabilitan) sucesivamente los valores de resistencia de temporización (pin 7 = timing resistor), a través de la palabra (*word*) digital escrita en los puertos de salida para ese control, se puede realizar así un barrido de frecuencia (dentro del rango fijado por el capacitor de temporización activo). De esta manera, existen 2047 posibilidades factibles de conexión sucesiva de Resistores para cada uno de los cuatro rangos de frecuencia capacitiva. La Resistencia efectiva de temporización estará dada por la Resistencia equivalente de un arreglo paralelo (las más de las veces).

El banco de Capacitores consta de 4 elementos :
1.0 , 0.1 , 0.01 y 0.001 [microFarad]
El banco de Resistores consta de 11 elementos :
2 M , 1 M , 500 k , 250 k , 125 k ,
62.5 k , 31.25 k , 15.625 k , 7.812 k
3.906 k y 1.953 k [ohm]

Frecuencia de Operación Teórica : $f = \frac{1}{R_{eq} C}$

f [Hz] ; Req [ohm] ; C [Farad]
R_{eq} = Resistencia Equivalente (conectada al pin 7 del XR-2206)
C = Capacitancia (conectada entre pines 5 y 6 del XR-2206)

Nótese que la Frecuencia de operación es inversamente proporcional a la Resistencia R (directamente proporcional a la Conductancia G), e inversamente proporcional a la Capacitancia C. Esto significa que si el Resistor de Temporización es menor (en ohms), o si el Capacitor de Temporización es menor (en microFarad), la Frecuencia de la señal generada por el XR-2206 será mayor (Hz).

Por esta razón los resistores de mayor valor se asocian a los bits menos significativos de un byte (igual que los capacitores de mayor valor), porque son los que, con su variación, determinan menores frecuencias de operación.
De esta manera, un barrido de frecuencia podrá hacerse de bajas a altas frecuencias, con la variación progresiva (incremental) desde los bits menos significativos a los más significativos.

Los capacitores se conectan entre el pin 5 y el 6 del XR-2206 (mediante relevadores controlados digitalmente con buffer 74125); los resistores se conectan entre el pin 7 y GND (usando el buffer 74125 como trayectoria a GND: conexión establecida o flotada ↑Z).

La información digital para el control de los bancos de elementos de temporización, es proporcionada a través de puertos paralelos de salida :

I/O Address	Data Bus							
	D7	D6	D5	D4	D3	D2	D1	D0
0060 h → LSByte	R7	R6	R5	R4	R3	R2	R1	R0
0061 h → MSByte	C3	C2	C1	C0	--	R10	R9	R8

* Nivel en bits : {High}=activa elemento R ó C ; {Low}=desactiva

El puerto paralelo de salida utilizado fue :

* Un "Programmable Peripheral Interface" 82C55A(-2) programando sus 3 puertos paralelos de 8 bits, como OUTPUT. Estas salidas se conectan a las entradas de control de los 74125 (considerando que en éstos últimos: "Output is OFF (disabled) when <control> is high" (por eso en programa de control, debe efectuarse un { not AL } antes de { out DX,AL }).

Aquí se usaron 2 puertos paralelos (PB_MSByte) y (PA_LSByte), uno para cada byte de la Word de control que fija una frecuencia de oscilación para el XR-2206 .

PALABRA DE CONTROL QUE ESTABLECE FRECUENCIA DE OPERACION EN XR-2206 :

|Data Bus||Output Port 82C55A||Control 74125||Elem. temporización controlado|

MSByte:				
D7	PB7	pin	(1)	C3 = 0.001 µF
D6	PB6	pin	(4)	C2 = 0.01 µF
D5	PB5	pin	(10)	C1 = 0.1 µF
D4	PB4	pin	(13)	C0 = 1.0 µF
D3	PB3	pin	(1)	-- ---
D2	PB2	pin	(4)	R10 = 1.953 kΩ
D1	PB1	pin	(10)	R9 = 3.906 kΩ
D0	PB0	pin	(13)	R8 = 7.812 kΩ
LSByte:				
D7	PA7	pin	(1)	R7 = 15.625 kΩ
D6	PA6	pin	(4)	R6 = 31.25 kΩ
D5	PA5	pin	(10)	R5 = 62.5 kΩ
D4	PA4	pin	(13)	R4 = 125 kΩ
D3	PA3	pin	(1)	R3 = 250 kΩ
D2	PA2	pin	(4)	R2 = 500 kΩ
D1	PA1	pin	(10)	R1 = 1 MΩ
D0	PA0	pin	(13)	R0 = 2 MΩ

CAPACITORES: Tantalio y Mylar. RESISTORES: Pr. 1% Pelicula Metálica
Capacitores conectados de menor a mayor valor (ya que f=k(1/C)), y aumentando en múltiplos de 10 a partir de 0.001 µF .
Resistores conectados de menor a mayor => de mayor a menor Conductancia (G=1/R) (ya que f=k(1/R)=kG), y aumentando en múltiplos de 2 a partir de 1.9531 kΩ .

De esta manera, se podrá generar a la salida del XR-2206 una onda senoidal y otra cuadrada, de frecuencia variable: "Barrido de Frecuencia" de bajas a altas frecuencias.
pin (2) : onda senoidal (o triangular)
pin (11) : onda cuadrada (defasada 90° con respecto a la senoidal)

La señal senoidal entregada por el XR-2206 , (con un offset DC de 6 V), es entonces acondicionada haciéndola pasar por un *Buffer* de Alta impedancia de entrada (Amplificador Operacional en configuración "*seguidor de voltaje*") y un circuito supresor de componente de DC ("especie" de Filtro Pasa-Altas Activo (con Amp.Op.)), para obtener una senoidal AC pura.

Esta senoidal AC , es alimentada al circuito o sistema analógico bajo análisis, como señal de Entrada (señal de prueba).

De esta señal senoidal (AC) de Entrada se obtiene el valor máximo [Volt pico] mediante un circuito "Detector de Valor Pico Positivo" , que es un arreglo a base de Amplificadores Operacionales, diodos, Capacitor y Resistencia de Descarga (cte. de tiempo elevada), con el cual se mantiene, a su salida, el nivel máximo de voltaje pico de una señal alterna (no necesariamente periódica o regular) que se conecte a su entrada. El capacitor queda cargado al valor más positivo de la Entrada y gracias a una resistencia de descarga convenientemente seleccionada y conectada en paralelo al capacitor, el circuito Detector de Pico puede efectivamente "seguir a la señal de entrada", manteniendo al capacitor cargado al valor pico sólo el tiempo necesario para que, en el caso del ADRF, este valor de DC pueda ser leido por una de las entradas analógicas de un Convertidor A/D { ADC-0809...canal IN0 }. El valor de R_d y C_p , debe calcularse para obtener una constante de tiempo lo suficientemente larga [seg.] para satisfacer los requerimientos de captura del nivel DC máximo detectado en un momento determinado.

Por otra parte, la señal senoidal AC , es también cuadratizada, empleando un Comparador en configuración "Detector de Cruce por Cero". Dicho comparador debe ser de alta velocidad (respuesta rápida) a fin de evitar errores de defasamiento adicional con respecto a la señal cuadrada obtenida directamente

del XR-2206, (el defasamiento entre ambas señales debe ser lo más próximo posible a 90°). Por ello se usa un Comparador diferencial de alta velocidad "LM 361" con tiempos de retardo típicos de de 3 [ns] a 20 [ns] max.; cuenta además con salidas TTL complementarias de máximo sesgo (*slew rate*) y puede ser operado a partir de fuentes para Amplificadores Operacionales (± 12 V en el caso del ADRF). Aunque las salidas del comparador son niveles TTL , se conectan a la fuente digital (+5V) mediante resistores de *pull-up* , a fin de que su rango de variación esté entre 0 y 5 V .

Así, se cuenta con 2 señales cuadradas, defasadas 90° entre sí : una, proviniente del comparador (senoidal cuadratizada), que representa a $\{\text{sen}(\omega_o t)\}$, y otra, proviniente directamente del Generador de Funciones XR-2206, que representa a $\{\cos(\omega_o t)\}$. Estas señales varian entre 0 y +5 Volt, por lo que se hacen pasar a través de un arreglo de Amplificadores Operacionales de Bajo Ruido (TL-074), a fin de multiplicarlas por +2 y agregarles un offset de -5V , esto es, $V_o = 2V_1 - 5$, para que así el rango de variación esté entre -5V y +5V , que son los valores de voltaje que harán conmutar a los *Switches* Analógicos CD-4016, al aplicarlos a sus entradas de control : (-5V=*switch abierto* $\longrightarrow$ señal NO PASA) , (+5V=*switch cerrado* $\longrightarrow$ señal PASA).

Para el muestreo de la señal analógica de salida del sistema bajo análisis, se emplearon Conmutadores Analógicos (*Analog Bilateral Switch*) CD-4016 ; el circuito integrado cuenta con 4 *switches* que pueden ser usados combinados o por separado. En cada *switch*, cuando el voltaje de control iguala al voltaje del pin 7 $\{V_{SS}\}$, el *switch* permanece apagado (OFF) y se comporta como una impedancia muy alta ($10^{12}\Omega$); en cambio, cuando el voltaje de control iguala el voltaje del pin 14 $\{V_{DD}\}$, el *switch* se enciende (ON) y se comporta casi como un resistor lineal bilateral de 300Ω. La resistencia de encendido (R_{ON}) depende ligeramente de la

polaridad y magnitud de los voltajes que están siendo muestreados (*switcheados*). Las señales que pasen a través del *switch*, pueden ser analógicas o digitales, pero nunca deben exceder el voltaje V_{DD} aplicado al IC, ni valer menos del voltaje V_{SS} ; dicho de otro modo, el *switch* puede manejar señales digitales o analógicas de cualquier valor o polaridad, siempre y cuando sus voltajes permanezcan dentro de los límites fijados por las fuentes de alimentación aplicadas al IC. A ± 5 V , la frecuencia máxima de señal que puede aplicarse al *switch* es de 5 MHz. Los niveles de referencia o voltajes de alimentación del CD-4016 deben ser muy estables y simétricos (no menores a 5 V), por lo que se usaron reguladores variables de voltaje : LM317T (positivo) y LM337T (negativo), para tener mayor control sobre el ajuste de los voltajes de polarizacion del IC.

Aunque el CD-4066 es una versión mejorada del CD-4016, pues tiene una menor R_{ON} (90Ω), el CD-4016 sigue siendo la mejor opción cuando se tienen aplicaciones que requieran mayor impedancia de aislamiento en estado OFF (*ultralow-leakage*)(menor fuga de corriente), como en circuitos de *Sample-Hold* o en Muestreo de señales, como en el caso del ADRF.

Cabe señalar que, gracias a que se cuenta con varios rangos y subrangos para el barrido de frecuencia, se podrá caracterizar el funcionamiento del circuito bajo prueba en una amplia gama de valores de frecuencia de interés práctico (desde 20 Hz hasta unos 600 kHz ... valor límite estable determinado por el generador de la señal de prueba (XR-2206) y la frecuencia de conmutación de Switches Analógicos para el muestreo (CD-4016 operando a ± 5 V) ; sin embargo, empleando Switches Analógicos LF11331 o LF13331 (*Quad SPST JFET Analog Switches, normally open & with disable*), fabricados por *National Semiconductor Corp.*, se podrían lograr frecuencias estables mayores para el barrido de análisis (valores cercanos a 1 MHz debido a la limitación impuesta por XR-2206)).

Las señales muestreadas (a la salida de los switches) : $S_1(t)$ y $S_2(t)$, se hacen pasar por Filtros Pasa-Bajas, a fin de eliminar las componentes de alta frecuencia, y entregar sólo niveles (bipolares) de DC , proporcionales a las componentes Real e Imaginaria de la señal de salida del sistema bajo análisis.

Existen varias aproximaciones y técnicas para llevar a cabo el diseño de dichos filtros Pasa-Bajas :

HUELSMAN & ALLEN

INTRODUCTION TO THE THEORY AND DESIGN OF ACTIVE FILTERS

MC.GRAW HILL , 1980

TOBEY , GRAEME , HUELSMAN

OPERATIONAL AMPLIFIERS , DESIGN AND APPLICATIONS

MC.GRAW HILL , 1981

Tomando en cuenta que sólo deben pasar señales de corriente directa (DC) a través de los filtros y que la minima frecuencia de operación del dispositivo ADRF se establece en 20 Hz , se escogió una aproximación Butterworth ; método de implementación VCVS ; ganancia total del filtro K=1 ; una pérdida de 60 dB entre la frecuencia de corte (5 Hz) y la frecuencia de supresión (20 Hz) . Para lograr esta plantilla de diseño (Pérdida de 60 dB entre f_c y f_s), se determinó que el filtro debia de ser de 5^o orden.

A la salida de los Filtros Pasa-Bajas, se colocaron Amplificadores Operacionales con el fin de optimizar los niveles de señal de entrada para el convertidor A/D. Se busca entregar una señal lo más grande posible dentro del rango permisible [± 5.12 V], a la entrada de los Adecuadores de señal para ADC, a fin de reducir el error (ruido) de cuantización (distorsión) inherente a la digitalización de señales de poca amplitud. Este error de cuantización hace que las señales analógicas pequeñas (como las salidas de filtros pasivos), se vean distorsionadas

(escalonadas) al digitalizarse.

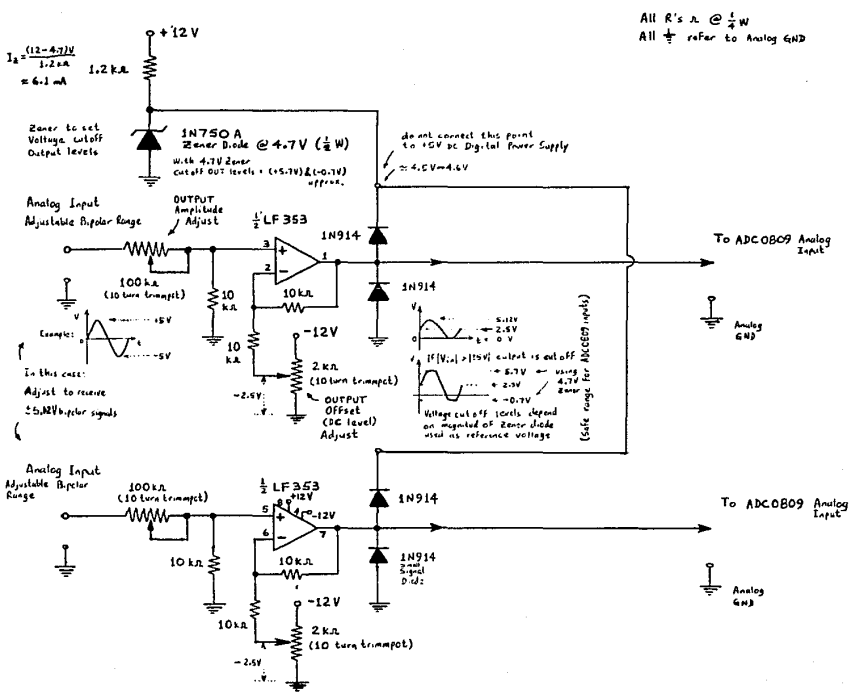
Por tanto, este Amplificador de señal presente en el *Hardware* del sistema analizador (entre el Filtro Pasa-Bajas y el Adecuador de señal), puede tener una ganancia fija conveniente, de acuerdo a las magnitudes de las señales de salida de los sistemas que comúnmente se pretenda analizar, o bien, ser un Amplificador (No Inversor) de Ganancia Programable controlada digitalmente, con el fin de entregar a la entrada del Amplificador-Adecuador-de-señal, una señal lo más grande posible dentro de los limites permisibles establecidos en el diseño ([-5.12 V a +5.12 V] en el caso del ADRF), seleccionando la mayor ganancia posible sin saturación o distorsión después de una serie de barridos de frecuencia sucesivos como sondeo, previos al despliegue definitivo (óptimo).

Después de los Amplificadores optimizadores de nivel se conecta un circuito basado en Amplificador Operacional que se designa funcionalmente bajo el nombre de "Adecuador de nivel de señal analógica para convertidor Analógico/Digital" o simplemente "Circuito Adecuador de Señal".

Debido a que en el sistema **SDM88-PC** , el convertidor Analógico/Digital ADC-0809, está implementado con una fuente de Voltaje de Referencia de +5.12 V , (para que con sus 8 bits se tengan 2^8=256 niveles de conversión digital ó pasos de { 5.12/256 = 20 mV } cada uno), entonces, el rango de la señal analógica de entrada para el ADC va de 0 a 5.12 Volt ; por tanto, para poder registrar señales bipolares, debe hacerse un acondicionamiento de las mismas. Para el ADRF - **SDM88-PC** , esto se hizo mediante un circuito analógico que transforma un rango que va de {-5.12 V} a {+5.12 V}, en un rango que va de 0 a +5.12 V .

.....

En la { *figura* 5-12 } se muestra un diagrama detallado del circuito utilizado para Adecuación de Señal analógica antes de ADC


All R's Ω @ 1/4 W
All ⏚ refer to Analog GND
+12 V
I_Z = (12 − 4.7)V / 1.2 kΩ ≈ 6.1 mA
1.2 kΩ
Zener to set Voltage cutoff Output levels
1N750A Zener Diode @ 4.7V (1/4 W)
With 4.7V Zener cutoff OUT levels = (+5.7V) & (−0.7V) approx.
do not connect this point to +5V DC Digital Power Supply
≈ 4.2V − 4.6V
Analog Input Adjustable Bipolar Range
OUTPUT Amplitude Adjust
1/2 LF 353
1N914
To ADC0809 Analog Input
100 kΩ (10 turn trimmpot)
10 kΩ
10 kΩ
1N914
Analog GND
Example:
+5V
−5V
In this case: Adjust to receive ±5.12V bipolar signals
−12V
2 kΩ (10 turn trimmpot)
−2.5V
OUTPUT Offset (DC level) Adjust
5.12V
2.5V
0 V
If |V_out| > |5V| circuit is cut off
+5.7V
+2.3V
−0.7V
using 4.7V Zener
Voltage cutoffs levels depend on magnitude of Zener diode used as reference voltage
(Safe range for ADC0809 inputs)
Analog Input Adjustable Bipolar Range
100 kΩ (10 turn trimmpot)
1/2 LF 353
+12V
−12V
1N914
To ADC0809 Analog Input
10 kΩ
10 kΩ
1N914 Small Signal Diode
Analog GND
10 kΩ
−12V
2 kΩ (10 turn trimmpot)
−2.5V

Así, se tiene para el Adecuador de Señal :

Voltaje de Entrada	Voltaje de Salida
+5.12 V	+5.12 V
0.00 V	+2.56 V
-5.12 V	0.00 V

Si por alguna razón se excede este rango de niveles de voltaje a la entrada, el circuito Adecuador de Señal deberá truncar la señal (a su salida), para, de esa manera, evitar daño a las entradas analógicas del convertidor A/D .

En caso de que se requiera que el rango de las señales de entrada sea diferente, se pueden emplear circuitos de atenuación o amplificación antes del Adecuador de Señal correspondiente, o bien calibrar este circuito para que acepte un rango diferente de señales de entrada y las escale entre 0 y +5.12 V . Esto se logra mediante los potenciómetros de precisión (*trimmpots*) que pueden apreciarse en el diagrama. El procedimiento de calibración es el siguiente :

1) Aislar del sistema al circuito Adecuador de Señal
2) Polarizar al circuito Adecuador y aplicar a su entrada una señal bipolar que abarque la totalidad del rango que se desea dejar pasar (p.ej. voltaje pico a pico de una señal senoidal AC proporcionada por un generador de funciones externo)
3) Manipular el *trimmpot* (100 kΩ) para ajustar la Amplitud Pico a Pico de la señal a la salida del circuito Adecuador, al rango permisible deseado ([± 5.12 V] en este caso)
4) Centrar la señal bipolar (alterna) adicionándole un offset DC igual a la mitad del rango total permisible (*trimmpot* de 2 kΩ) ... { p.ej. si se conecta la entrada del circuito Adecuador a GND, se verá a su salida un voltaje de DC (línea en osciloscopio), que deberá desplazarse mediante ajuste con el *trimmpot*, hasta un valor de 5.12/2= +2.56 V }
5) Repetir pasos 3 y 4 hasta obtener el resultado deseado

6) Aumentar la amplitud de la señal de prueba a la entrada del circuito Adecuador para verificar que efectivamente todo exceso resulte truncado a la salida, a niveles seguros que no lleguen a dañar las entradas del circuito que se intenta proteger (ADC después del Adecuador, en este caso)

7) Variar la frecuencia de la señal de entrada y observar el desempeño del circuito Adecuador con la frecuencia (observar el efecto sobre la señal original a la salida del circuito para determinar si se satisfacen los requerimientos esperados) ; en el caso del ADRF este factor no es critico pues prácticamente se está trabajando con niveles DC a la salida de los filtros Pasa-Bajas

8) Una vez calibrado el circuito Adecuador de Señal, integrarlo al sistema electrónico dentro del cual operará y observar el funcionamiento global de dicho sistema después de la adición

Cabe mencionar que, para la gran mayoría de las etapas del ADRF que requieren de Amplificadores Operacionales, se empleó el circuito integrado " LF353 " (*Wide Bandwidth Dual JFET Input Operational Amplifier*) fabricado por *National Semiconductor Corp.* Estos Amp.Op. de tecnología BI-FET II ^{TM}, son de entradas JFET por lo que tienen alta impedancia de entrada ($10^{12}\Omega$) y no cargan a los circuitos que los preceden; son de respuesta rápida, bajo costo y vienen dos dentro de cada IC (8-pin) ; requieren baja corriente de polarización (3.6mA) y aún asi mantienen gran ancho de banda (4 MHz) y rápido sesgo (*Slew Rate*) (13 V/μs); además, las entradas JFET hacen que las corrientes de entrada de polarización (bias) y offset sean muy bajas (Low Input bias current = 50 pA), (Low Input noise voltage = 16 nV/$\sqrt{\text{Hz}}$, & current = 0.01 pA/$\sqrt{\text{Hz}}$) y se tengan bajas fugas o corrimientos indeseados (drift) en offset de voltaje. Este Amp.Op. es de "Bajo ruido" y tiene baja distorsión armónica (< 0.02% @ ganancia 10 y BW 20Hz-20kHz).

Finalmente, los valores muestreados, ya a niveles de voltaje adecuados, son leidos de manera dosificada (bajo control de *Software*-Microprocesador) por el Convertidor Analógico/Digital conectado al bus del sistema SDM88-PC , y almacenados en memoria RAM (en localidades adyacentes para cada grupo de 3 valores leidos {IN0, IN1, IN2} correspondientes a una frecuencia f_o durante el Barrido), para su posterior transferencia a memoria RAM de la Microcomputadora, donde serán procesados convenientemente para el despliegue de información.

Durante el Barrido de Frecuencia, entre cada variación discreta de frecuencia para la adquisición de los valores de un nuevo grupo de muestras con el ADC (se leen 3 valores IN0, IN1, IN2 bajo una misma frecuencia f_o de la señal de prueba), se generará un retardo (tiempo de espera) por *software* de 0.04 [s] = 40 [ms]. Esto con el fin de dar tiempo para que la respuesta del circuito bajo análisis se estabilice al desaparecer la respuesta transitoria inducida al cambiar la frecuencia de prueba "f_o" durante el proceso de Barrido. Tener en cuenta que si el circuito bajo análisis es un filtro Pasa-Bajas, su respuesta transitoria tarda más tiempo en desaparecer que en los filtros Pasa-Altas.
(En el caso del ADRF se usa una rutina (*software* calibrado residente en EPROM : "NMIhandler") para producir retardos (*delays*) programables de tiempo muy precisos).

Por lo que respecta al módulo MEDIDOR DE FRECUENCIA se dirá que, dado que el comparador LM361 cuenta con dos salidas TTL complementarias (defasadas 180°), que varian según una misma señal de entrada, una de esas dos salidas, (la negada), es tomada como señal de entrada (pulsos) para un contador (Counter 0) del 8254 (*Programmable Interval Timer*), que funge como Frecuencímetro. La señal de pulsos a contabilizar se dirige a la entrada CLK 0 del Contador ó Canal 0 del 8254, después de ser acondicionada y filtrada de ruido digital y transitorios que ocasionen falsos

disparos y cuentas erróneas, al pasar previamente a través de un resistor de amortiguamiento (*Damping Resistor*), útil sobre todo al medir bajas frecuencias, y dos intensificadores de sesgo (transición) de señal en cascada (74HC14 = Schmitt Trigger).

Una vez procesada y desplegada la información, es decir, cuando se está ya en la etapa de análisis de resultados gráficos, el usuario indica cuándo desea conocer la frecuencia de operación, y mediante la expedición del comando correspondiente, transmitido por el puerto serie de la microcomputadora al sistema **SDM88-PC** , el microprocesador desencadena la operación de conteo al mandar un pulso de disparo (para iniciar la cuenta) al GATE 0 del 8254 a través de un puerto de salida de 1 bit (PC7 del 8255). Todo esto es supervisado por *Software* de control para "lectura de Frecuencia" ejecutado desde RAM del **SDM88-PC**. Este *Software*, se encarga también de generar ventanas de tiempo de 1 [s] y 0.1 [s] y de efectuar lecturas (*latcheadas*) de los registros del 8254 después de transcurridos esos periodos, a fin adquirir las cuentas (#) de pulsos (frecuencia), que son almacenadas en RAM y posteriormente transmitidas a la microcomputadora para su procesamiento y despliegue (frecuencímetro digital con despliegue en pantalla de microcomputadora).

El programa de control para medición de la frecuencia de una señal, se encarga de accionar el *Hardware* apropiado con el fin de adquirir la cuenta de pulsos entregada por uno de los contadores de 16 bit, del 8254 (8MHz), en dos lecturas:

- Lectura (1) :
  con una ventana de 1 [s] (lectura ≤ FFFFh (65535))
  { resolución= 65535 [Hz] }
- Lectura (2) :
  con una ventana de 0.1[s]
  { determinar cuántos ciclos de 65535 pulsos dió el contador excitado por la señal de entrada cuya frecuencia se mide }

El Contador { Counter 0 } del 8254 es programado en modo 1 : "Hardware Retriggerable one-shot".
En Modo 1 : (Min.Count=1 ; Max.Count=0=2^16=65536)

* El control del pulso de disparo (Start-count trigger), lo realiza el microprocesador con 1 bit, a través de un puerto paralelo (82C55A-2 en este caso).
* Se genera una ventana de tiempo muy precisa con duración de 1.0 [s] , para sensar el # de pulsos que alimentan al reloj del contador en un segundo :

Frecuencia = (#pulsos)/(1 seg.) = [Hz]

* Para la temporización por ventanas, se emplea la rutina "NMIhandler" residente en EPROM, que actualiza cada 1/100[s] una localidad de memoria etiquetada como TIME y que conserva la cuenta periódicamente actualizada, para efectos de término-de-conteo por comparación mediante poleo ("polling"). Esta rutina se ejecuta cada centésima de segundo ya que el pin 17 del microprocesador 8088 { NMI = *Non-Maskable Interrupt* = Interrupción NO MASCARABLE } es disparado por una señal de CK con frecuencia = 100 [Hz], generada por el {Counter 2} del Timer 8254. La duración de la ventana de tiempo se mide en centésimas de segundo (1/100[s]), y en este caso es de 1[s]=100/100[s] , para la medición de frecuencia de la señal de entrada (cuenta ≤ 65535), y de 0.1[s]=10/100[s] , para la ventana que determina el número de ciclos completos de 65535 cuentas que dió el contador.

* Cálculo del factor de entrada para la rutina "NMIhandler":

Duración	# de {1/100[s]}	Cálculo	Factor (cte) _Duración
0.1 [s]	10 = 000Ah	0000h-000Ah =	FFF6h
1.0 [s]	100 = 0064h	0000h-0064h =	FF9Ch

Finalmente, el control se regresa a la rutina principal de control del SDM88-PC : "Manager Routine" , dejando al sistema preparado para recibir alguna nueva orden a través del puerto

serie de la microcomputadora.

Es entonces cuando el resultado puede leerse de memoria :

Segment: 0030h Offset:

Lect.Ventana 1s ... 0400h ... CounterOFreq (LSByte Counter0) {b0}
0401h ... CounterOFreq (MSByte Counter0) {b1}
Lect.Ventana 0.1s . 0402h ... CounterOFreq (LSByte Counter0) {b2}
0403h ... CounterOFreq (MSByte Counter0) {b3}

Para convertir la cuenta de frecuencia de hexadecimal a decimal : deben encontrarse los equivalentes decimales de los bytes leídos (para obtener el valor decimal de la lectura del contador) y restar dicho valor de FFFFh (65535), ya que los contadores del 8254 son descendentes.

Lect.V(1s) = 65535 - Lectura decimal del Contador
= 65535 - ({b1d}*256+{b0d})
b1d y b0d son equivalentes decimales de bytes leídos
Lect.V(0.1s) = 65535 - Lectura decimal del Contador
= 65535 - ({b3d}*256+{b2d})
b3d y b2d son equivalentes decimales de bytes leídos

Frecuencia =
int [({Lect.V(0.1s)}*10) / 65535] * 65535 + {Lect.V(1s)}

Ejemplo : Si la frecuencia a medir es 100000 [Hz] resultará :

Lectura contador : para Ventana de 1.0 [s] : 795Eh
para Ventana de 0.1 [s] : D8EFh
b0=5Eh ; b1=79h ; b2=EFh ; b3=D8h
b0d=94 ; b1d=121 ; b2d=239 ; b3d=216 {equiv. decimales}

Lect.V(1s) = 65535 - (121*256+94) = 65535 - (31070) = 34465 pulsos (nótese que la Lectura con la ventana de 1 [s] representa lo que se pase de un múltiplo entero de 65535 , en este ejemplo sería : 100000 - 65535 = 34465)

Lect.V(0.1s) = 65535 - (216*256+239) = 65535 - (55535) = 10000
pulsos

(nótese que esta lectura es la frecuencia real dividida entre 10 , por tanto, la máxima frecuencia posible a medir sin reciclamiento será : 65535*10 = 655350 Hz)

Entonces, el cómputo de la frecuencia real será :

Frecuencia = int [({10000}*10) / 65535] * 65535 + {34465}
= int [100000/65535] * 65535 + 34465
= int [1.5259] * 65535 + 34465 = 1 * 65535 + 34465
= 65535 + 34465
= 100000 [Hz]

Otra forma de obtener el resultado (frecuencia decimal) es :
restando en hex y convirtiendo el resultado a decimal :
Lect.V(1s) = FFFFh - {b1,b0} = XXXXh —> convertir a decimal
Lect.V(0.1s) = FFFFh - {b3,b2} = YYYYh —> convertir a decimal
y aplicar la fórmula anterior para el cómputo de frecuencia.

+

Finalmente es importante señalar que los sistemas físicos (sistemas bajo análisis con el ADRF), tienen algún tipo de alinealidades. Por tanto, es necesario considerar cuidadosamente la amplitud de la señal senoidal de entrada (señal de prueba). Si la amplitud de la señal de entrada es excesivamente grande, el sistema analizado tiende a saturarse y la prueba de Respuesta en Frecuencia da resultados inexactos. Por otro lado, una señal pequeña puede producir errores debidos a zona muerta. Por esta razón, hay que realizar una cuidadosa elección de la amplitud de la señal senoidal de entrada. Esto se logra con el Generador XR-2206, al manipular el potenciómetro de precisión (*trimmpot*) en el subcircuito de control de la Amplitud de salida, pudiéndose

ajustar ésta en un rango que va desde 60[mV] hasta 3[V] de voltaje pico (en la onda senoidal).

Es necesario muestrear la forma de onda de la salida del sistema bajo análisis para asegurarse de que su forma es sinusoidal y que el sistema está funcionando durante el periodo de prueba en la región lineal. La forma de onda a la salida del sistema no es sinusoidal (o aparece deformada) si éste está funcionando en una región no lineal.

+

El Mapa de Puertos I/O para el ADRF es :

SALIDA DECODIFICADOR	DIRECCION PUERTO I/O {hex}	REGISTRO ACCESADO DEL PUERTO I/O

8254 {*Programmable Interval Timer*} :

$\overline{Y2}$	0020	CounterOFreq	{Contador de pulsos} (Frecuencímetro)
	0021	Counter 1	{disponible}
	0022	Counter 2	{disponible}
	0023	PITFreqCtrlW	{Registro de Control}

8255 {*Programmable Peripheral Interface*} : (82C55A[-2])

$\overline{Y6}$	0060	PPortXRL	{LOW Timing Byte}
PA (out)			[R0-R7]
	0061	PPortXRH	{HIGH Timing Byte}
PB (out)			[R8-R10],[C0-C3]

PA y PB : CONTROL DIGITAL XR-2206 (BARRIDO DE FRECUENCIA)

	0062	PPortFreqCtrl	{ PC7 : Start-count
PC (out)			trigger (Frecuencímetro) }
	0063	PPI_Nyq_CtrlW	{Registro de Control}

5.4.6

ANALIZADOR DIGITAL DE RESPUESTA EN FRECUENCIA : DESCRIPCION DEL " SOFTWARE " DE CONTROL

El *Software* de control para la administración de los recursos de la microcomputadora, del sistema **SDM88-PC** y del *Hardware* específico del instrumento de aplicación "ADRF", consistirá en un programa maestro (principal) ejecutándose en la microcomputadora, que se encargará de transmitir selectiva y sincronizadamente, el código de microprocesador 8088 apropiado al **SDM88-PC** , para que éste, a su vez, efectúe en paralelo determinadas tareas de control sobre el *hardware* del ADRF.

De esta manera, en ciertos momentos, se estarán ejecutando en realidad, dos programas en paralelo e interactuando entre sí : uno residente en la memoria de la microcomputadora y el otro en la memoria del **SDM88-PC** . La comunicación entre ambos dispositivos se realizará como siempre, a través de sus respectivos puertos serie.

El *Software* se encargará de sincronizar y desencadenar acciones sobre el *Hardware* (circuitos programables controlados por el microprocesador) para efectuar los procesos de barrido de frecuencia, adquisición/almacenamiento-organizado-de-datos, y transferencia de datos desde la interface **SDM88-PC** a la microcomputadora, y, ya residentes en la memoria de ésta última, realizará el procesamiento matemático de la información digital muestreada y su escalamiento para graficación, así como la lectura de parámetros proporcionados por el usuario, el despliegue de "menúes" para selección de alternativas y el desencadenamiento condicional de acciones de procesamiento para cada opción elegida, el despliegue de resultados y ajuste/lectura de frecuencia de manera interactiva (mediante el desplazamiento de un cursor gráfico por parte del usuario). Además, mediante el *Software*, se

ejecutarán cuando se requiera, distintas rutinas, tales como : toma de opción de usuario, inicialización de puertos de comunicación en microcomputadora e interface **SDM88-PC** , transmisión de comandos (código de microprocesador 8088) desde microcomputadora a **SDM88-PC** , inicialización de periféricos digitales I/O del "ADRF", manejo de errores, ajuste de baudaje, etc. .

Las rutinas de control para la interface { **SDM88-PC** }, se componen de código de microprocesador 8088 que está "embebido" dentro del programa de control principal que se ejecuta en la microcomputadora (en variables reservadas para su almacenamiento). Estas rutinas, que normalmente tienen código relocalizable, son cargadas en áreas determinadas dentro de memoria RAM del **SDM88-PC**, e invocan como "FAR calls" (a través de dirección absoluta), a subrutinas residentes en EPROM para la realización de tareas específicas usando los periféricos del sistema (subrutinas como : "f_AD_Read" para el ADC, "fTransmit_Char" y "fReceive_Char" para el USART 8251A, "fDelay" para generación de retardos de duración programable, etc.).

Parte del código que se ejecuta desde RAM del **SDM88-PC** , efectúa un ciclo (*loop*) que posiciona sucesivamente en localidades adyacentes dentro de un buffer en memoria RAM, la información digitalizada de tres señales analógicas (una leída por el Canal-0 {IN0}, otra por el Canal-1 {IN1} y otra más por el Canal-2 {IN2}), suministrada por un convertidor Analógico/Digital ADC0809 :

{ IN 0 } : Magnitud pico (A_m) de la señal de Entrada :

$v_i(t) = A_m \operatorname{sen}(\omega_o t)$ ... (Excitación) al circuito bajo prueba

{ IN 1 } : (Respuesta) Componente Real (cantidad proporcional)

$X = A_m \, |H(j\omega_o)| \cos(\phi(\omega_o))$

{ IN 2 } : (Respuesta) Componente Imaginaria (cant. proporcional)

$Y = A_m \, |H(j\omega_o)| \operatorname{sen}(\phi(\omega_o))$

{ Componentes del Fasor Respuesta : { $V_o(j\omega_o) = X + jY$ }

IN-0 : (proviene de un Detector de valor Pico al que se le aplica la señal $v_1(t)$).

IN-1 , IN-2 : (provienen de : salida muestreador (CD4016) {multiplicación *cos ó *sen} _ FiltroPasaBajas _ Amplificador optimizador de nivel _ Adecuador de Señal para ADC).

Cada una de las muestras tomadas por canal, corresponde a una frecuencia de análisis distinta suministrada por el IC "Monolithic Function Generator XR-2206", controlado digitalmente en lo que respecta a la generación del barrido de frecuencia dentro de un determinado rango.

La Adquisición de información (señal) analógica y digitalización de la misma se realiza mediante un convertidor Analógico/digital: ADC0809.

Con la frecuencia de microprocesador fijada a 4.77 [MHz] y del ADC a 1.2 [MHz] , las señales analógicas a leer por el ADC (CH-0 , CH-1 y CH-2) deben tener una frecuencia menor a 200 [Hz] para poder reconstruirse apropiadamente. En esta aplicación se leerán prácticamente señales de DC pues son proporcionadas por las salidas de los filtros pasa bajas.

La señal de Reloj (ck) para el ADC0809, es suministrada por un Canal (Counter 1) del Timer Programable 8254 del **SDM88-PC**, por lo que inicialmente se programa a una frecuencia menor o igual a 1.2 MHz, apropiada para que el convertidor A/D funcione correctamente.

El ADC0809 es controlado digitalmente por el microprocesador (controlado por *software*) a través de un puerto paralelo de salida [PPort1B=FF31h] (perteneciente a un 82C55A [-2] PPI), y uno de entrada [ADC_EOC_port=FF50h] (de 1 bit, usando una compuerta

Tri-state 74125) para, mediante poleo ("polling"), sensar la terminal EOC (End-Of-Conversion) (L→H) y saber cuándo la información digital puede ser leída por el puerto paralelo de entrada [ADCdata=PPort1C=FF32h] (de un 82C55A), y de ahí pasar al bus de Datos del sistema, de donde a su vez, es leída por el microprocesador que la canaliza a memoria RAM para su almacenamiento. El protocolo de control de A/D c. está contenido en la rutina "f_AD_Read", que, residente en EPROM del SDM88-PC , {FE00:1240}, se invoca como FAR procedure (FAR call). Debe proporcionarse (desde programa) tan sólo la dirección del canal que se desee leer (000 a 111) en los 3 bits menos significativos del registro "BL" (el ADC0809 cuenta con un multiplexor interno para seleccionar uno de entre 8 canales analógicos de entrada). La información digitalizada correspondiente a la señal analógica leída con el A/D c., será regresada por esta rutina en el registro "AL" del microprocesador 8088 (dentro del sistema SDM88-PC).

La Información digital es almacenada en un buffer en RAM del SDM88-PC, en localidades contiguas:
byte_IN0, byte_IN1, byte_IN2, byte_IN0, byte_IN1, byte_IN2, byte_IN0,
... etc.
Por tanto, las localidades de RAM donde se almacena la información correspondiente a uno cualquiera de los tres canales, están separadas dos bytes entre sí : para el canal-0 : {0,3,6,9,C,...} , para el canal-1 : {1,4,7,...} , para el canal-2 : {2,5,8,...} .
El tamaño del buffer de RAM puede tener cualquier número de elementos (definido en la constante "SizeBuffer"). El tamaño neto del buffer en RAM es en realidad " 3*SizeBuffer ", pues se almacenan alternadamente los bytes correspondientes a CH0 , a CH1 y a CH2. Se usa al registro "DI" del 8088 como apuntador, [DI], al buffer de memoria de datos.

Es importante señalar que entre cada lectura (tomada a frecuencia diferente), se introduce un retraso (delay) de

aproximadamente 40 [ms] , para leer valores de estado estable después de que pase el transitorio en la señal de Salida por cada cambio de frecuencia en la señal de Entrada al circuito bajo análisis.

La rutina (*Software*) administradora-controladora del SDM88-PC durante los procesos de "Barrido de Frecuencia" y "Adquisición Analógica (de los valores de voltaje correspondientes a las componentes Real e Imaginaria de la Respuesta del sistema analizado y de la magnitud pico de la señal de Entrada)", es ejecutada bajo supervisión del microprocesador 8088 del SDM88-PC, mientras el programa ejecutado simultáneamente en la microcomputadora, la hace entrar en un ciclo de espera de duración variable, que depende del número de muestras (de la señal de salida entregada por el sistema bajo análisis), que se están adquiriendo (número fijado indirectamente por el usuario al seleccionar uno de 5 posibles subrangos dentro de uno de los 4 rangos posibles para el barrido de frecuencia). Así el número de muestras leídas con el convertidor A/D es :

512 muestras (subrangos "1")
256 muestras (subrangos "2")
170 muestras (subrangos "3")
128 muestras (subrangos "4")
102 muestras (subrangos "5")

los subrangos "1", corresponden a una mayor frecuencia final de barrido y los subrangos "5", a una menor frecuencia final, dentro del rango seleccionado para el análisis. Los valores de frecuencia son calculados matemáticamente de acuerdo a una ecuación obtenida a partir de pruebas experimentales para caracterización de la relación "Frecuencia generada VS. valor de elementos R-C del circuito de temporización para el XR-2206" ; los resultados de dichas pruebas experimentales fueron ajustados con el método de Mínimos Cuadrados. De esta manera, los valores reales de frecuencia final asignados para el barrido por

el *software* de control en la microcomputarora, nunca quedarán fuera de los límites físicos de frecuencia que puede generar el IC XR-2206.

Después del proceso de Barrido de Frecuencia y Adquisición de Datos, la microcomputadora sale del estado de espera (controlado por *software*) y empieza la ejecución de un protocolo de comunicación establecido entre el puerto serie del **SDM88-PC** (8251A) y el puerto serie de la microcomputadora, para la transferencia de la información digitalizada (por el ADC) desde la memoria RAM del **SDM88-PC** a la RAM de la microcomputadora. Dicho de otro modo, una vez llenado el Buffer en RAM de la interface, se efectúa un nuevo ciclo (*loop*), leyendo estos valores de RAM y transmitiéndolos por el puerto serie (8251A), a RAM de la microcomputadora , donde son cargados y clasificados en Arreglos de Variables, para su ajuste y procesamiento matemático y gráfico.

Nota:

Para dosificar la transmisión de bytes, (compatibilizar temporización entre interface y PC), antes de transmitir cada byte-dato de alguno de los 3 canales, la interface espera recibir de PC un caracter de identificación-canal para sincronización, verificando que sea distinto a un caracter (flag) de identificación para "fin de Transferencia".

Este programa administrador en RAM de interface, debe terminar regresando el control (mediante un FAR jump), a la rutina principal de control del **SDM88-PC** : "Manager Routine" residente en EPROM, quedando el sistema preparado para recibir alguna nueva orden (a través del puerto serie de la PC). La velocidad de Tx/Rx serie entre interface **SDM88-PC** y microcomputadora PC, sigue siendo compatible.

Como se mencionó anteriormente, en la microcomputadora, la información digitalizada queda almacenada, organizada y clasificada en arreglos de variables Reales (tipo vector) propios del lenguaje de alto o medio nivel empleado para programación, a partir de los cuales puede someterse a procesamientos matemáticos, (de cálculo, de ajuste y de escalamiento), a fin de poder desplegarse en forma gráfica y numérica en pantalla.

.....

En la { *figura* 5-13 } , se muestra en forma diagramática el proceso matemático a que son sometidos dentro de la microcomputadora, los valores digitalizados por el convertidor Analógico/Digital, correspondientes a los distintos valores (muestrales) de frecuencia (f_o) generados dentro del rango de frecuencia seleccionado para el barrido de análisis.

.....

El usuario podrá seleccionar el tipo de gráfica que desee analizar (Magnitud o Angulo de Fase), y desplazarse sobre dichas curvas (función de transferencia del sistema) observando los valores numéricos correspondientes a cada punto, además de tener la posibilidad de medir la frecuencia [Hz] en cualquier punto que considere de interés.

Dentro de Microcomputadora:

Procesamiento Matemático de valores leídos del Convertidor Analógico/Digital para los distintos valores (muestrales) de frecuencia (F_o) generados c

Convertidor A/D : 8-bit = 2^8 = 256 pasos (niveles de tipificación digital para señales analógicas).

ADC-0809 : Voltaje de Referencia para Conversión A/D = V^+_{REF} = + 5.12 [V]

Canales: Entradas Analógicas:

IN 1 : → $X = GA \cdot \frac{1}{\pi} \cdot A_m \cdot |H(j\omega_o)| \cdot \cos(\phi(\omega_o))$

X = Componente Real

IN 2 : → $Y = GA \cdot \frac{1}{\pi} \cdot A_m \cdot |H(j\omega_o)| \cdot \operatorname{sen}(\phi(\omega_o))$

Y = Componente Imaginaria

IN 0 : → K_p = Magnitud de la señal de Entrada al Sistema bajo análisis

$Z_c = X + jY$

Valores Leídos (ADC):

$0 \leq X \leq 255$ (00) (FF)

$0 \leq Y \leq 255$ (00) (FF)

Estandarización a Voltaje de Referencia de Conversión A/D : V^+_{REF} = + 5.12 [V]

$\frac{X}{256} \cdot V^+_{REF} = R \quad \{0 \leq R < 5.12$ [V]

$\frac{Y}{256} \cdot V^+_{REF} = S \quad \{0 \leq S < 5.12$ [V

Los valores "X", "Y" y "K_p" deben digitalizarse con el Convertidor Analógico/Digital y procesarse-ajustarse matemáticamente para cada valor de frecuencia F_o generado por el XR-2206, (para cada punto muestral) durante el barrido de análisis.

$\omega_o = 2\pi F_o$ [$\frac{rad}{s}$] ; F_o [Hz]

Valor Pico Positivo

(No requiere Adecuador de Nivel de señal para entrada al ADC)

Valor leído (ADC): $0 \leq K_p \leq 255$ (00) (FFh)

$\frac{K_p}{256} * V^+_{REF} = T$

$0 \leq T < 5.12$ [V]

Señal de Entrada [$V_i(t) = A_m \operatorname{sen}(\omega_o t)$]

• Constante de Escala de Conversión (Span relativo)

$\frac{5.12 \text{ Volt}}{256 \text{ pasos}} = 20 \frac{mV}{paso}$

para IN0: $K_{esc} = \frac{\text{Span } (K_p)}{V^+_{REF}} = \frac{5.12\text{ [V]}}{5.12\text{ [V]}} = 1$ (V^+_{REF} ⇩ Span ADC)

∴ No requiere Adecuador de Señal para entrar a ADC

• Parte Real : T (Valor leído)

• Parte Imaginaria : U = 0

• Magnitud: $M_i = \sqrt{T^2 + U^2} = T$

• Angulo de Fase: $\phi_i = \tan^{-1}\left(\frac{U}{T}\right) = 0°$

La señal de entrada se supone como señal de Referencia para la señal de Salida (∴ $\phi_i = 0°$)

• El Generador de Funciones XR-2206 entrega como máximo una señal senoidal de 3 [V_{PEAK}] (Volt Pico)

∴ $M_{i\,max} = 3$ [V]

• Fasor de la Señal de Entrada al sistema analizado:

$\{V_i(s)\}_{s=j\omega|_{\omega_o}} \quad V_i = M_i \angle\phi_i = M_i \angle 0° = A_m \angle 0°$

$M_i = |V_i| = A_m$ (Amplitud Pico Max. de Senoidal de Entrada)

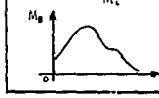
$\phi_i = \angle V_i = 0°$ (Referencia de Fase) (para la señal de Salida)

Función de Trasferencia del Sistema analizado:

$H(s) = H(j\omega_o) = \frac{V_o(j\omega_o)}{V_i(j\omega_o)} = \frac{A_c + jB_c}{T + jU_\phi} = \frac{M_o \angle\phi_o}{M_i \angle\phi_i} = |H(j\omega_o)| \angle H(j\omega_o) = M_H \angle\phi_H$

Magnitud Lineal (M_H)

$M_H = \frac{M_o}{M_i} = \frac{A_m |H(j\omega_o)|}{A_m} = |H(j\omega_o)|$



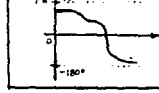
$M_H \geq 0$ [Adimens.]

Función siempre (+)

Angulo de Fase (ϕ_H)

Defasamiento de la señal de Salida con respecto a la señal de Entrada.

$\phi_H = \phi_o - \phi_i = \phi_o - 0° = \phi_o$ [°]



$-180° \leq \phi_H \leq +180°$

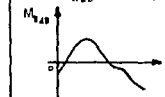
[° = grados]

Función (+), (-)

Magnitud Logarítmica ($M_{H\,dB}$)

(en decibeles)

$M_{H\,dB} = 20 \cdot \log_{10}(M_H)$ [dB]



$M_{H\,dB} > 0$ si $M_H > 1$

$M_{H\,dB} = 0$ si $M_H = 1$

$M_{H\,dB} < 0$ si $0 < M_H < 1$

Función (+), (-)

Ajuste Matemático (Compensación

(sólo variables leídas por IN1 (X),

IN1 : Componente Real

$\frac{10.24 \text{ Volt}}{256 \text{ pasos}} = 40 \frac{mV}{paso}$ ... Constante Conversi

Span Relativo (Real)

$K_{esc} = \frac{\text{Span (X)}}{\text{Span (ADC)}} = \frac{10.24}{5.12} = 2$

IN2 : Componente Imaginaria

$\frac{10.24 \text{ Volt}}{256 \text{ pasos}} = 40 \frac{mV}{paso}$ ... Constan Conversi

Span Relativo (Real)

$K_{esc} = \frac{\text{Span (Y)}}{\text{Span (ADC)}} = \frac{10.24}{5.12} = 2$

Corrección Matemática de Componente:

$A_c = A \cdot FADJV = A_m \cdot |H(j\omega_o)| \cdot c$

(Componente Re) Salida del Sistema

$B_c = B \cdot FADJV = A_m \cdot |H(j\omega_o)| \cdot s$

(Componente Im) Salida del Sistem

$\{-5.12 < A_c < +5.12$ [V] ; $\{-5.12 < B_c$

• Componente Real : A_c (Valor

• Componente Imaginaria : B_c (Valor

• Magnitud: $M_o = \sqrt{A_c^2 + B_c^2}$

• Angulo de Fase: $\phi_o = \tan^{-1}\left(\frac{B_c}{A_c}\right)$

$\phi_o[°] = \phi_o$ [rad] $\cdot \frac{18}{\pi}$

• Fasor de la Señal de Salida analizado:

$\{V_o(s)\}_{s=j\omega|_{\omega_o}} \quad V_o = M_o \angle\phi_o = A_m |H(j\omega$

$M_o = |V_o| = A_m |H(j\omega_o)|$

$\phi_o = \angle V_o = \tan^{-1}\left(\frac{B_c}{A_c}\right.$

↳ Ajustado para quedar

$-180° <$

l para los distintos valores (muestrales) de Frecuencia (f_o) generados dentro del rango de Frecuencia seleccionado para el barrido de análisis.

as).

* SPAN= Rango de la variable del proceso (Voltage mínimo ↔ máximo), esto es equivalente al rango de la entrada normal del sistema estandarizado. → en ADC {0 → V_{REF}^+}

leídos (ADC):

$0 \leq X \leq 255$ (FF)

$0 \leq Y \leq 255$ (FF)

Estandarización a Voltaje de Referencia de Conversión A/D: $V_{REF}^+ = +5.12$ [V]

$\frac{X}{256} \cdot V_{REF}^+ = R \quad \{0 \leq R < 5.12\ [V]\}$

$\frac{Y}{256} \cdot V_{REF}^+ = S \quad \{0 \leq S < 5.12\ [V]\}$ } R+jS

Desdoblamiento de valores a niveles de Voltaje previos a Amplificadores-Acondicionadores-de-señal-analógica para ADC. {0 → 5.12} → {-5.12 → +5.12}

$A = 2\cdot(R-2.56) \quad \{-5.12 < A < +5.12\}$ [V]

$B = 2\cdot(S-2.56) \quad \{-5.12 < B < +5.12\}$ [V] } A+jB

" y "Kp" deben digitalizarse con el Convertidor rocesarse-ajustarse matemáticamente para uencia F_o generado por el XR-2206, (para l) durante el barrido de análisis.

$\omega_o = 2\pi F_o$ $\left[\frac{rad}{s}\right]$; F_o [Hz]

Trasferancia del Sistema

$\frac{V_o(j\omega_o)}{V_i(j\omega_o)} = \frac{A_c + jB_c}{T + jU_\phi}$

$= \frac{M_o \angle \phi_o}{M_i \angle \phi_i} = \frac{|H(j\omega_o)|\ \angle H(j\omega_o)}{}$

$= M_H \angle \phi_H$

Lineal (M_H)

$\frac{M_o}{M_i} = \frac{A_m |H(j\omega_o)|}{A_m} = |H(j\omega_o)|$

$M_H \geq 0$ [Adimens.]

Función siempre (+)

F[Hz]

de Fase (ϕ_H)

imiento de la señal de Salida con sto a la señal de Entrada.

$= \phi_o - \phi_i = \phi_o - 0° = \phi_o$ [°]

$-180° \leq \phi_o \leq +180°$ [° = grados]

F[Hz]

Función (+),(-)

tud Logarítmica (M_{HdB})

n deciBeles)

$M_{dB} = 20\cdot\log_{10}(M_H)$ [dB]

$M_{HdB} > 0$ si $M_H > 1$

$M_{HdB} = 0$ si $M_H = 1$

$M_{HdB} < 0$ si $0 < M_H < 1$

F[Hz]

Función (+),(-)

Ajuste Matemático (Compensación) (sólo variables leídas por IN1 (X), IN2 (Y)):

IN1: Componente Real

$\frac{10.24\ Volt}{256\ pasos} = 40\ \frac{mV}{paso}$ ... Constante de Conversión

Span Relativo (Real)

$K_{ESC} = \frac{Span\ (X)}{Span\ (ADC)} = \frac{10.24}{5.12} = 2$

IN2: Componente Imaginaria

$\frac{10.24\ Volt}{256\ pasos} = 40\ \frac{mV}{paso}$ ... Constante de Conversión

Span Relativo: (Real)

$K_{ESC} = \frac{Span\ (Y)}{Span\ (ADC)} = \frac{10.24}{5.12} = 2$

Corrección Matemática de Componentes:

$A_c = A \cdot FADJV = A_m \cdot |H(j\omega_o)| \cdot \cos(\phi(\omega_o))$

(Componente Re) Salida del Sistema Analizado

$B_c = B \cdot FADJV = A_m \cdot |H(j\omega_o)| \cdot \text{sen}(\phi(\omega_o))$

(Componente Im) Salida del Sistema Analizado

$\{-5.12 < A_c < +5.12\}$ [V] ; $\{-5.12 < B_c < +5.12\}$ [V]

- Componente Real : A_c (Valor leído y ajustado)
- Componente Imaginaria : B_c (Valor leído y ajustado)
- Magnitud: $M_o = \sqrt{A_c^2 + B_c^2}$; $\{M_o \geq 0\}$ [V]
- Angulo de Fase: $\phi_o = \tan^{-1}\left(\frac{B_c}{A_c}\right)$ [rad]

$\phi_o[°] = \phi_o[rad] \cdot \frac{180°}{\pi\ rad}$; $\{-180° < \phi_o \leq 180°\}$

- Fasor de la Señal de Salida del sistema analizado:

$\{V_o(\omega)\}_{seno(\omega_o t)}$ $V_o = M_o \angle \phi_o = A_m |H(j\omega_o)| \angle \phi_o$

$M_o = |V_o| = A_m |H(j\omega_o)| = \sqrt{A_c^2 + B_c^2}$

$\phi_o = \angle V_o = \tan^{-1}\left(\frac{B_c}{A_c}\right)$

Le Ajustado para quedar en rango $-180° < \phi_o \leq +180°$

Factor de Ajuste o Compensación: FADJV

$FADJV = \left(\frac{1}{GA}\right)\cdot(\pi)\cdot\left(\frac{1}{K_{ESC}}\right)$

Ajuste por Software: $FADJV = \frac{1}{4A} \cdot \pi \cdot \frac{1}{K_{ESC}}$

El factor $\left(\frac{1}{GA}\right)$, anula al factor (GA) presente a la entrada ADC en componentes X, Y.

GA= Ganancia de Amplificador de Señal (Amp. Op. No inversor) para aprovechamiento máximo del Span de conversión Fijado por ($V_{REF}^+ = +5.12$ [V]) del Convertidor A/D.

Se busca que: $\left\{\frac{\text{Valor binario leído}}{2^{(\text{\# bits del ADC})}} \cdot V_{REF}^+{}_{\text{conversor A/D}}\right\}$ → Sea lo más grande posible

para reducir así, el Error (ruido) de Cuantización (distorsión) que hace que las señales analógicas pequeñas (como las salidas de filtros pasivos), se vean distorsionadas (escalonadas) al digitalizarse.

Este Amplificador de señal presente en el Hardware del sistema analizador (entre Filtro Pasa-Bajos y Adecuador de señal), puede tener una Ganancia Fija conveniente, de acuerdo a las magnitudes de las señales de salida de los sistemas que comunmente se pretenda analizar, o bien, ser un Amplificador (No inversor) de Ganancia Programable controlada digitalmente, con el Fin de entregar a la entrada del Amplif.-Adecuador-de-señal, una señal lo más grande posible dentro de los límites permisibles establecidos [-5.12, +5.12] [V], seleccionando la mayor ganancia posible sin saturación o distorsión después de una serie de barridos de frecuencia sucesivos previos al despliegue definitivo (óptimo).

El Factor (π) = 3.141592654, anula al factor $\left(\frac{1}{\pi}\right)$; inherente al procesamiento de control durante el Muestreo (de la señal de la Salida del Sistema analizado), con señales cuadradas como aproximaciones a las funciones "sen($\omega_o t$)" y "cos($\omega_o t$)". Este factor de atenuación existente en el proceso físico, se hace evidente en el análisis matemático al desarrollar en series trigon. de Fourier, las Funciones de onda Cuadrada representativas de "sen($\omega_o t$)" y "cos($\omega_o t$)" en el Muestreo (generadas a partir del XR-2206 y que accionan el control de switches analógicos CD4016).

Factor de Span Relativo: K_{ESC}

El Factor $\left(\frac{1}{K_{ESC}}\right)$, anula al Factor ($K_{ESC}$) que es un Factor de Escalamiento introducido en el Hardware por los "Amplificadores-Adecuadores-de-Nivel-de-Señal-Analógica-para-ADC", para procesar dentro del rango de voltaje de operación del ADC (0 → 5.12V), señales con un mayor rango de voltajes (mayor "span") → (-5.12 → +5.12) ⇒ span = 10.24V.

$K_{ESC} = \frac{\text{Span Real de Señal Analizada}}{\text{Span propio del Convertidor A/D}}$: [Es un Factor de escalamiento o ganancia LOGICO]

Para IN1, IN2 ... $K_{ESC} = \frac{Span\ (X)}{Span\ (ADC)} = \frac{Span\ (Y)}{Span\ (ADC)} = \frac{|V^+| + |V^-|}{V_{REF\ conv}^+} = \frac{|+5.12| + |-5.12|}{5.12} = \frac{10.24}{5.12} = 2$

Es importante señalar, que aunque en las distintas técnicas de Respuesta en Frecuencia tradicionales se acostumbra tomar logarítmica a la escala horizontal "ω", en el *software* de control para esta aplicación (ADRF), se considera una escala horizontal lineal en la frecuencia "f". Esto se hace así, porque muchas veces en la práctica resulta más conveniente un análisis en forma local (lineal), pues la zona de interés para la caracterización de la respuesta de un dispositivo, es reducida.

Una escala logarítmica en la frecuencia es útil para el análisis de sistemas de control, cuando se requiere un panorama completo de su comportamiento (a bajas y altas frecuencias ... por décadas u octavas); de esta manera, la información se comprime y esto permite tener en una sola gráfica, una visión global (panorámica) del funcionamiento del sistema. Sin embargo, si la zona en que se espera opere el dispositivo es restringida (en la frecuencia), es probable que se pierda algo de definición en la gráfica como resultado de esta compresión en las abscisas.

En ciertas condiciones frecuentes en la práctica, un despliegue con escala lineal en la frecuencia, es más conveniente pues, cuando la zona de interés sugiere un análisis en forma local; por ejemplo, en el análisis de filtros de aparatos de Telecomunicaciones (análisis de bandas en canales de comunicación, variación de señal moduladora alrededor de una onda portadora), o en la caracterización de Sistemas de Amplificadores Operacionales, Filtros Eléctricos, Amplificadores sintonizados, etc., cuando se tienen pequeñas variaciones de frecuencia alrededor de un cierto valor de interés (bandas de paso, frecuencia central, frecuencia de corte o supresión, etc.), y en general, cuando se quiera realizar un análisis en detalle, con buena resolución gráfica, dentro de una "ventana" o zona de interés de rango reducido (local) en la frecuencia.

Sin embargo, si para alguna aplicación se requieren gráficas con escala logarítmica en la frecuencia, éstas podrán obtenerse manualmente, haciendo una tabulación previa, a partir de la información numérica (Magnitud, Fase y frecuencia) proporcionada en pantalla para cada punto muestreado durante el barrido de frecuencia (dentro de las limitaciones impuestas por el *hardware* del ADRF).

A continuación se describirá en forma modular, la estructura del programa (*software*) de control en la microcomputadora (Rutinas para la realización-coordinación de tareas) para la operación del " **Analizador Digital de Respuesta en Frecuencia** " .

Este programa puede ser desarrollado en cualquier lenguaje de alto nivel (BASIC, Pascal, Fortran) o medio nivel (C) (de preferencia un lenguaje compilado). En este caso, fue creado en BASIC { editado con GWBASIC (ver.3.22) y posteriormente compilado con QuickBASIC (ver.4.0) , ambos de *Microsoft* }, debido a las facilidades que este lenguaje ofrece para manejo y establecimiento de comunicación a través del puerto serie.

" **Analizador Digital de Respuesta en Frecuencia** "
{ Software de Control en Microcomputadora } $\longrightarrow$ Módulos :

A) **RUTINAS AUXILIARES DE PROPOSITO GENERAL :**

1) Verificación de conexión correcta :
{ Microcomputadora $\longleftrightarrow$ **SDM88-PC** } .

2) Transmisión de caracter (byte) por microcomputadora.

3) Recepción de caracter (byte) por microcomputadora.

4) "Bajado" de Programa-Comando (código de microprocesador), desde microcomputadora a memoria de interface (SDM88-PC) y autoejecución (Transmisión por puerto serie RS-232).

5) Desglose en bytes componentes de dirección o dato decimal (word-sized) dado en una variable.

6) Toma de opción de pantalla por presión de tecla.

7) Rutina de programación de Velocidad de Transmisión {Tx/Rx}, tanto en microcomputadora como en interface: [300 Bd, 1200 Bd, 2400 Bd, 4800 Bd, 9600 Bd] { > **Comando mandado desde microcomputadora a interface** < } .
Para una correcta comunicación microcomputadora ⟷ interface, ambas deben operar al mismo baudaje.
Esta rutina es transparente al usuario y programa una velocidad de Tx/Rx de 300 Bauds para operaciones normales y breves y 4800 Bauds cuando se hace uso intensivo del puerto de comunicación serie; por ejemplo, durante transferencia de comando-código desde microcomputadora a interface **SDM88-PC** ó durante transferencia de información digitalizada desde RAM-interface hacia RAM-microcomputadora.

8) Rutina interactiva de Manejo de Errores (Hardware o Entradas del usuario).

9) Rutina de "Término de Sesión" (dejar a interface y microcomputadora operando a 300 Bd para que no exista conflicto en caso de accionar Reset-Hardware en interface; deshabilitar puerto Serie en microcomputadora; deshabilitar "Estado de chequeo para detección de errores"; limpiar pantalla).

10) Rutina de "Borrado de Renglones en pantalla" (edición de texto de información para usuario).

B) **RUTINAS PARA CONTROL DE TAREAS DE PROPOSITO ESPECIFICO EN ADRF :**

1) Inicialización de recursos de memoria en microcomputadora (Arreglos de variables enteras y reales).

2) Portada (pantalla) de presentación.

3) Inicialización gráfica y definición de parámetros según Resolución Gráfica disponible : (CGA = Color-Graphics-Adapter = {BW:640x200pixels} , EGA = Enhanced - Graphics - Adapter = {Color:640x350pixels}).

4) Inicialización de Puerto Serie en microcomputadora.

5) "Reset" a sistemas de comunicación serie de interface (SDM88-PC) y microcomputadora (PC ó PS) . (Reset por software a interface para reinicializar 8251A y fijar velocidad de Transmisión/Recepción Interface ⟷ Microcomputadora en 300 Bauds). { > **Comando mandado desde microcomputadora a interface** < }

6) Inicialización de periféricos (ICs) digitales programables que intervienen en el control del Hardware de aplicación "Analizador Digital de Respuesta en Frecuencia" : { > **Comando mandado desde microcomputadora a interface** < }

* 82C55A [-2] :
  Control para elementos R-C de Ajuste de Frecuencia en XR-2206 y pulso de disparo (trigger) para iniciar cuenta de pulsos en frecuencímetro.
* 8254 :
  Frecuencímetro (Contador (Counter 0) : 16 bit).

7) Introducción de Frecuencia Inicial tentativa por el usuario para el Barrido de Frecuencia sobre el circuito por analizar.

8) Generación de parámetros para procesamiento :

- Número de Muestras a tomar para cada uno de 5 posibles subrangos de Barrido de Frecuencia dentro de cada uno de 4 posibles rangos (fijados por capacitor de ajuste de frecuencia en XR-2206 : 0.001μF, 0.01μF, 0.1μF y 1μF)
- Coeficientes de la Ecuación Cuadrática :

  $MM\ x^2 + BB\ x + F1 = 0$

  obtenidos por el método de ajuste por Mínimos Cuadrados aplicado sobre la tabla de valores experimentales : elementos_R-C vs. frecuencia_generada_por_XR-2206

  Nota :

  La ecuación resultante del ajuste de curva de : elementos_R-C vs. frecuencia_generada_por_XR-2206 , por el método de Mínimos Cuadrados, fue obtenida para el Capacitor de 0.01μF (es decir, variando Resistencia y manteniendo C=cte=0.01μF)

  Variable independiente (x) = valor de Resistencia

  Variable dependiente (F) = Frecuencia generada por XR-2206 de acuerdo a valor R-C vigente

$$F(x) = \left[\left(44.16915 - 1.487781 * 10^{-3} x \right) x \right] K$$

$$K = \frac{0.01\ \mu F}{C_r} = \left\{ \text{FACTOR DE ESCALAMIENTO EN FRECUENCIA DEPENDIENDO DEL CAPACITOR } C_r \text{ ELEGIDO} \right.$$

C_r = 1μF , 0.1μF , 0.01 μF , 0.001μF

9) Cálculo de Frecuencias Tope posibles para cada Rango y cada Subrango posible, no en [Hz] sino en palabra binaria para controlar activación de elementos resistivos de ajuste de frecuencia para XR-2206 a través de puertos paralelos (82C55A).

XO(I) = Cuenta (valor #) para generar Frecuencia Inicial de barrido XR-2206

XF = Cuenta (valor #) para generar Frecuencia Final de barrido XR-2206 de acuerdo al número de muestras asignadas para cada subrango (XF: variable intermedia)

FS(I,K) = Frecuencia Final aproximada en [Hz] obtenida con ecuación de Ajuste por Mínimos Cuadrados

I = Rango (fijado por Capacitor) { para Barrido de Frecuencia }

K = Subrango

Nota :

Si, dado el valor de la Frecuencia Inicial introducida por el usuario, no es posible operar en un rango determinado (por resultar la frecuencia fuera de rango permisible para XR-2206), entonces se hace FS(I,K)=0 .

10) Selección de Rango y Subrango para Barrido de Frecuencias :

+ Se indica la Frecuencia Final aproximada (teórica) en [Hz] para cada uno de los posibles (5) Subrangos dentro de cada uno de los 4 Rangos.
+ Dicha Frecuencia Final depende de la Frecuencia Inicial introducida por el usuario.
+ Se indica al usuario cuando no es posible seleccionar una opción [Rango{I},Subrango{K}] , por no poderse generar la frecuencia final con XR-2206 ("fuera de Rango").
+ Para cada opción [Rango{I},Subrango{K}] , aparece la siguiente pantalla-Menú en microcomputadora :

    Frecuencia Final Aproximada = xxxxxx [Hz]

    < C > : Continuar con opciones siguientes

    < R > : Regresar a opciones previas

    < P > : Iniciar Procesamiento con opción actual

    < Q > : Abandonar programa

11) Captura y Generación de Parámetros para Procesamiento (según opción [Rango{I},Subrango{K}] elegida) :

+ Cálculo de Frecuencia Inicial Aproximada (teórica) en [Hz].

+ Cálculo de Frecuencia Final Aproximada (teórica) en [Hz].
+ Determinación de {Palabra Binaria} = {Tamaño del Buffer de Datos que serán digitalizados por ADC} . (Número de Muestras correspondientes al Subrango K elegido).
+ Determinación de Palabra Binaria con información de Resistores a activar para generación de Frecuencia Inicial de Barrido con XR-2206.
+ Determinación de Máscara Binaria con información de activación para un Capacitor (el correspondiente al Rango (I) elegido), para "ORear" con la palabra binaria que contiene la información de los Resistores que serán activados ; esta operación determina la configuración de la Palabra binaria de "Ajuste de Frecuencia" para XR-2206 (información para determinar la habilitación / deshabilitación de elementos R-C de temporización para fijar la frecuencia de oscilación en el XR-2206).

12) Rutina de Barrido de Frecuencia sobre circuito bajo análisis y Adquisición de Datos por ADC (Digitalización de información analógica) en interface {SDM88-PC} . { > **Comando mandado desde microcomputadora a interface** < }

13) Rutina de Transferencia de Datos de Tabla (información digitalizada) en memoria RAM de interface a memoria RAM de microcomputadora. { > **Comando mandado desde microcomputadora a interface** < }

14) Adquisición de Datos Digitalizados por ADC-0809 (Programa paralelo en microcomputadora).
Establecimiento de protocolo Tx/Rx serie entre interface y microcomputadora.
La información digital recibida consta de N*3 bytes, donde N es el número de muestras correspondientes al Subrango K de análisis elegido :

Subrango (1) ⇒ 512 muestras
Subrango (2) ⇒ 256 muestras
Subrango (3) ⇒ 170 muestras
Subrango (4) ⇒ 128 muestras
Subrango (5) ⇒ 102 muestras

Cada paquete de 3 bytes correspondiente a una muestra se compone de la siguiente forma :

1er byte : Caracter digitalizado ADC_IN0 :
(entrada analógica IN 0)
Magnitud Pico de la señal de Excitación al sistema (circuito) bajo prueba :
$v_1(t) = A_m \, sen(\omega_o t)$ ⇒ Magnitud Pico = A_m
Arreglo : VI(i)

2° byte : Caracter digitalizado ADC_IN1 :
(entrada analógica IN 1)
Parte Real de la señal de Respuesta del circuito bajo prueba : (factor proporcional)
$x = GA \frac{1}{\pi} A_m |H(j\omega_o)| \cos(\phi)$; Kesc = 2
Arreglo : AT(i)

3er byte : Caracter digitalizado ADC_IN2 :
(entrada analógica IN 2)
Parte Imaginaria de la señal de Respuesta del circuito bajo prueba : (factor proporcional)
$y = GA \frac{1}{\pi} A_m |H(j\omega_o)| \, sen(\phi)$; Kesc = 2
Arreglo : BT(i)

Respuesta del sistema bajo análisis : AT(i) + j BT(i) .(Fasor).
Se mandan caracteres de sincronia-identificación entre cada byte durante el proceso de transmisión de información desde interface (SDM88-PC) hacia microcomputadora.

15) Procesamiento y Escalamiento matemático de datos leídos a valores de voltaje apropiados (Acondicionador de señal) para

generación de arreglos :

VHMAG(i) = Magnitud Lineal de función de Transferencia
[ ] adimensional

VHDB(i) = Magnitud Logaritmica de función de Transferencia
[dB] decibeles

VHPHASE(i) = Angulo de Fase de función de Transferencia
[°] grados sexagesimales

a) Ajuste de datos digitalizados en un rango entre 0 y V_{REF}^{+} de ADC (+5.12V) ... ADC-0809. (Para IN0,IN1,IN2).

b) Ajuste de datos digitalizados en (IN1,IN2) en un rango entre +5.12V y -5.12V (Voltajes reales antes del circuito acondicionador de señal para ADC) y multiplicación por factor de ajuste mat.(compensación) : $FADJV = \frac{1}{GA} \pi \frac{1}{Kesc}$
(ver diagrama de procesamiento { *figura* 5-13 })

c) Cálculo de Magnitud Lineal, Magnitud Logaritmica y Angulo de Fase de la función de Transferencia para cada punto muestral.

AT(i) = Componente Real de $V_o(j\omega_o)$ (Respuesta)
BT(i) = Componente Imaginaria de $V_o(j\omega_o)$ (Respuesta)
VI(i) = Magnitud Valor Pico de $V_i(j\omega_o)$ (Excitación)

* Magnitud Lineal :

$$VHMAG(i) = \frac{\sqrt{AT(i)^2 + BT(i)^2}}{VI(i)} \quad ; \quad VHMAG(i) \geq 0 \quad \forall\, i$$

* Magnitud Logaritmica :

$$VHDB(i) = 20 \log_{10}\{VHMAG(i)\}$$

$VHDB(i) > 0$ si $VHMAG(i) > 1$
$= 0$ si $VHMAG(i) = 1$
< 0 si $0 < VHMAG(i) < 1$

- Angulo de Fase (en grados sexagesimales)

  rango : $-180^\circ < VHPHASE \leq +180^\circ$

Cálculo en Radianes :

Si { AT(i) > 0 }

$\Rightarrow$ $ANGRAD = angtan\left[\frac{BT(i)}{AT(i)}\right]$ [rad]

Si { AT(i) = 0 }

$\Rightarrow$ si { BT(i) = 0 }

$\Rightarrow$ ANGRAD = indeterminado

$\Rightarrow$ si { BT(i) > 0 }

$\Rightarrow$ $ANGRAD = +\frac{\pi}{2}$ [rad]

$\Rightarrow$ si { BT(i) < 0 }

$\Rightarrow$ $ANGRAD = -\frac{\pi}{2}$ [rad]

Si { AT(i) < 0 }

$\Rightarrow$ si { BT(i) $\geq$ 0 }

$\Rightarrow$ $ANGRAD = angtan\left[\frac{BT(i)}{AT(i)}\right] + \pi$ [rad]

$\Rightarrow$ si { BT(i) < 0 }

$\Rightarrow$ $ANGRAD = angtan\left[\frac{BT(i)}{AT(i)}\right] - \pi$ [rad]

Conversión de radianes a grados sexagesimales :

$VHPHASE(i) = ANGRAD * \frac{180^\circ}{\pi}$

16) Menú recurrente de selección de
variable a graficar vs. Frecuencia [Hz]

Después de la graficación y análisis, al seleccionar opciones <1>, <2> y <3> , siempre se vuelve a este menú, a menos que se presione la opción <Q> para salir del programa.

Las opciones del menú son :

1 > Magnitud Lineal de la función de Transferencia $VH=\frac{V_o}{V_i}$

2 > Magnitud Logaritmica de la función de Transferencia [dB]

3 > Angulo de Fase de la función de Transferencia [°=grados]

4 > Asignación de Nuevos Parámetros de Análisis : {f1},{f2}

Q > Terminar la Sesión ... Regresar a " DOS "

17) Rutina de Derivación de control de acuerdo a opción elegida.
Opción seleccionada :

4 > : Regresar al punto "5" : Reset...

Q > : Pasar a Rutina de "Término de Sesión" y abandonar programa de control.

1 > , 2 > , 3 > : Asignación de la variable asociada a la opción elegida a un Arreglo variable (vector) único de ordenadas (cotas) para graficación : "YORDG(i)" e inicialización de variables tipo "string" con mensajes distintivos para el despliegue de dicha opción en pantalla.

18) Rutina de Graficación de Datos tomados por la Interface aprovechando la resolución gráfica disponible EGA ó CGA .

a) Despliegue de información para el usuario, sobre parámetros que se van a graficar :

+.....+

INFORMACION SOBRE PARAMETROS GRAFICADOS

PARAMETROS DE GRAFICACION: Función de Transferencia $\{V_o/V_i\}$

Eje Horizontal : Frecuencia [Hz]

Eje Vertical : Magnitud Lineal [ ] ... Ganancia = V_o/V_i

ó Magnitud Logaritmica [dB]

ó Angulo de Fase [grados(°)]

Frecuencias de Barrido para la caracterización del circuito bajo análisis :

f1 = Frecuencia Inicial aproximada

f2 = Frecuencia Final aproximada

Para cada punto muestreado se desplegará el valor numérico correspondiente de :

[Mag.Lin.$\{V_o/V_i\}$] , [Mag.Log.$\{V_o/V_i\}$] y [Ang.Fase $\{V_o/V_i\}$]

Ganancia [dB] = 20 log (V_o/V_i)

Atenuación [dB] = 20 log (V_i/V_o)

Presionar < F > => cómputo de Frecuencia [Hz] en algún punto de interés

Se traza cuadrícula de referencia y divisiones principales escaladas para el Eje Vertical (líneas horizontales), así como Eje Horizontal cuando exista cruce por cero.

Los valores procesados provienen del muestreo (digitalización) mediante un Convertidor Analógico/Digital, de las señales de Entrada $\{V_i\}$ y Salida $\{V_o\}$.

+.....+

b) Cálculo de cotas extremas (Mínima y Máxima) en intervalo de graficación.

c) Cálculo de Parámetros de graficación : (Escala, coordenadas del origen para la gráfica escalada en términos de "pixels" en pantalla).

d) Inicialización gráfica y Sistema de ejes de Referencia y trazo de escalamiento en eje vertical.

e) Graficación de la Curva de Respuesta en Frecuencia de la variable seleccionada (Magnitud o Angulo de Fase).

19) Rutina de Manejo (posicionamiento) de Cursor Gráfico y variación automática de frecuencia de operación para el punto de análisis.

El Número de "Punto Muestreado" para posicionamiento de Cursor se monitorea con la variable entera "ICRS%" : {0 ≤ ICRS% ≤ (NMP%-1)}

ICRS% = 0 es el índice correspondiente al primer punto muestreado

NMP% = Número de puntos de muestreo correspondientes al Subrango K elegido para el Barrido de Frecuencia (análisis) (dentro de un Rango I)

Opciones de posicionamiento de cursor mediante presión de tecla :

POSICION CURSOR: {1}:f1|{2}:f2|{>}:+10|{<}:-10|{+}:+1|{-}:-1|{Q}:↓

{1} : f1 = Cursor en punto de muestreo INICIAL (f1)
{ ICRS% = 0 }

{2} : f2 = Cursor en punto de muestreo FINAL (f2)
{ ICRS% = NMP%-1 }

{>} : +10 = Avanzar cursor 10 puntos de muestreo respecto de la posición actual (avance de 10 en 10 puntos hacia la derecha) :
Frecuencia de análisis Aumenta rápidamente
{ ICRS% = ICRS%+10 }

{<} : -10 = Retroceder cursor 10 puntos de muestreo respecto de la posición actual (avance de 10 en 10 puntos hacia la izquierda) :
Frecuencia de análisis Disminuye rápidamente
{ ICRS% = ICRS%-10 }

{+} : +1 = Avanzar cursor 1 punto de muestreo respecto de la posición actual (avance de 1 en 1 hacia la derecha) :
Frecuencia de análisis Aumenta lentamente
{ ICRS% = ICRS%+1 }

{-} : -1 = Retroceder cursor 1 punto de muestreo respecto de la posición actual (avance de 1 en 1 hacia la izquierda) :
Frecuencia de análisis Disminuye lentamente
{ ICRS% = ICRS%-1 }

{Q} : ↓ = Terminar proceso de análisis gráfico

{F} : = Desplegar Frecuencia [Hz] de operación para punto de muestreo correspondiente a la posición actual del cursor gráfico

Al mover el cursor dentro del rango de graficación se van desplegando simultáneamente los valores numéricos de VHMAG, VHDB y VHPHASE para el punto de muestreo posicionado, además de que automáticamente se modifica y actualiza la frecuencia de la señal senoidal de Entrada al sistema bajo análisis, proporcionada por el XR-2206 . Esto último se logra generando la palabra binaria que produce la frecuencia adecuada correspondiente al punto muestral posicionado en la gráfica (Máscara de Habilitación para Resistores y Capacitores de ajuste de frecuencia para XR-2206), y transfiriendo dicha palabra de control desde la microcomputadora a la interface SDM88-PC como parte de un código-comando que se encarga de colocarla en los puertos paralelos de salida que actúan sobre los bancos de Capacitores y Resistores de Temporización para el generador de funciones (XR-2206) .

20) Rutina de Cómputo y Lectura de Frecuencia de Señal (XR-2206) inyectada al circuito bajo análisis, mediante un contador de 16-bit del IC 8254 que funciona como frecuencímetro.

a) Transferencia desde microcomputadora a interface y ejecución de programa-comando (código de μP 8088) para control de Frecuencímetro-8254 : Reset (a ceros) de contador; generación de pulso de disparo (trigger) para iniciar conteo de pulsos dentro de ventanas de tiempo de 1[s] y 0.1[s] ; lectura "*latcheada*" o congelada de la cuenta al término de las ventanas de tiempo; almacenamiento en RAM de dichos resultados (2 words = 4 bytes) .
b) Transferencia de información de frecuencia leída y almacenada en RAM de interface a microcomputadora (Protocolo Tx/Rx) $\longrightarrow$ (4 bytes) .
c) Ya en microcomputadora, cálculo de frecuencia decimal para despliegue en pantalla [Hz] .

Dado que la ejecución de esta rutina dura alrededor de 1.5 [s] (@ 4800 Bauds), sólo es invocada cada vez que el usuario presione la tecla { F } , que es una opción adicional a las de "movimiento

del cursor gráfico", mientras se tiene en pantalla la curva de respuesta de la variable seleccionada. Entonces, la opción { F } despliega la Frecuencia de operación en [Hz] para el punto de muestreo correspondiente a la posición actual del cursor gráfico (dentro de los límites de la gráfica de la variable analizada).
La razón por la que no se despliega la frecuencia de operación para cada punto de muestreo es que, debido al cómputo de frecuencia y las ventanas de tiempo, el desplazamiento del cursor en la gráfica sería demasiado lento; por esto, la medición de frecuencia sólo se realizará en los puntos que el usuario decida son de interés, al posicionarse con el cursor dentro de la gráfica analizada.

21) Despliegue de Parámetros Numéricos en el intervalo de análisis.
Cuando el usuario decide terminar con el análisis interactivo sobre la gráfica de Respuesta en Frecuencia de la variable seleccionada, (al presionar <Q>), se despliegan en pantalla los siguientes valores numéricos :

Frecuencia Inicial aproximada (teórica)
Frecuencia Final aproximada (teórica)
Ordenada Inferior (Mínima)
Ordenada Superior (Máxima)
Escala en el Eje Vertical (# de unidades a que equivale cada división principal trazada en el eje Vertical: escalamiento)

22) Rutina opcional (existente pero no habilitada actualmente dentro del programa) para almacenar en disco una copia de la imagen (gráfica) desplegada en pantalla.

5.4.7

ALGUNAS INDICACIONES SOBRE LA DETERMINACION EXPERIMENTAL DE FUNCIONES DE TRANSFERENCIA

1) Generalmente es más fácil efectuar medidas exactas de Amplitud que medidas exactas de desplazamiento de Fase. Las mediciones de desplazamiento de Fase pueden introducir errores causados por instrumentación o por interpretación de los registros experimentales.

2) La Respuesta en Frecuencia del equipo de medición utilizado para la determinación de la salida del sistema, debe tener curvas de Amplitud en función de la frecuencia prácticamente planas. Además, el Angulo de Fase debe ser casi proporcional (lineal) a la frecuencia.

3) Dado que los sistemas fisicos tienen algún tipo de alinealidades, es necesario considerar cuidadosamente la amplitud de las señales senoidales de entrada a fin de que la respuesta del sistema no se sature (si la entrada es demasiado grande) o sea errónea debido a operación en "zona muerta" (si la entrada es demasiado pequeña). Además, hay que asegurarse de que la salida del sistema conserve su forma de onda senoidal para concluir que durante el periodo de prueba, dicho sistema analizado está funcionando en una región lineal.

4) Si el sistema en estudio está funcionando continuamente por dias y semanas, no hace falta detener su funcionamiento normal para las pruebas de Respuesta en Frecuencia. Se puede superponer la señal de prueba senoidal a las entradas normales. Entonces, para sistemas lineales, se superpondrá a la salida normal, la respuesta correspondiente a la señal de prueba. Para la determinación de la función de Transferencia con el sistema en funcionamiento normal, también se usan

frecuentemente señales estocásticas (señales de ruido blanco). Utilizando funciones de correlación, se puede obtener la función de Transferencia del sistema sin interrumpir su funcionamiento normal.

5.4.8

CONCLUSIONES

El primer paso en el desarrollo y en el análisis de un circuito, instrumento o proyecto de control, es desarrollar un modelo matemático del sistema en estudio. Obtener un modelo analiticamente puede ser bastante difícil. Puede ser necesario obtenerlo por medio de un análisis experimental. Como se ha visto, la importancia de los métodos de Respuesta en Frecuencia es que la función de Transferencia de un sistema, o de cualquiera de sus componentes, puede ser determinada experimentalmente por simples mediciones de respuesta de frecuencia.

Si se han medido la relación de Amplitud (Magnitud) y el Desplazamiento (Angulo) de Fase en un número suficiente de frecuencias dentro del rango de frecuencia de interés, se los puede representar en el diagrama de Bode, y a partir de él, de otras maneras (Nyquist, Nichols, etc.). Luego se puede determinar la función de Transferencia por aproximaciones asintóticas. Se construyen curvas asintóticas del logaritmo de la Magnitud (módulo) constituidas por varios segmentos. Con algunos tanteos y correcciones en frecuencias de transición, generalmente es posible hallar una aproximación bastante cercana a la curva. (Nota : si se representa la frecuencia en [Hz] en lugar de en [rad/s], hay que convertir las frecuencias de transición a [rad/s], antes de calcular las constantes de tiempo).

Por tanto, muchas veces es necesario un análisis gráfico y numérico (cualitativo y cuantitativo), de la Magnitud y el Angulo de Fase de la Función de Transferencia (a excitación senoidal) de un sistema (circuito) cualquiera.

Es en estos casos, donde pueden apreciarse las ventajas y facilidades que proporciona el ADRF (**Analizador Digital de Respuesta en Frecuencia**), al automatizar el proceso de **ADQUISICION - ORGANIZACION - DESPLIEGUE_GRAFICO_ESCALADO de Muestras** , liberando al usuario de estas tareas rutinarias y susceptibles de error, mediante el empleo de un **Hardware** de control (**Interface**) y una **Microcomputadora** .

El diseñador podrá entonces concentrarse exclusivamente en el desarrollo de su sistema (circuito, instrumento, etc.), evaluándolo de manera interactiva al observar la respuesta (gráfica y numérica) que entrega después de cada modificación realizada sobre él. O bien, si se trata de un sistema (circuito) ya existente, se podrán generar conclusiones con respecto a su estabilidad, a su linealidad y calidad de desempeño en algún rango de frecuencias de interés, gracias a la facilidad para cambiar los parámetros de análisis, al despliegue de resultados de una manera gráfica y numérica y al manejo interactivo del ADRF por parte del usuario, sobre las curvas de respuesta (función de Transferencia) del sistema analizado.

El proceso de "Modificación del sistema (con vistas a su optimización) y/o la caracterización de su respuesta correspondiente", podrá repetirse tantas veces como se requiera, y, gracias a una herramienta auxiliar como el ADRF, el tiempo de "Diseño-Prueba-Evaluación" del prototipo o el tiempo de "Caracterización" de un sistema ya existente, disminuye notablemente.

+

C O N C L U S I O N E S

Como se ha apreciado en el presente trabajo, la situación del **SDM88-PC**, dentro del contexto de los Sistemas de Desarrollo disponibles comercialmente, es ventajosa, y sus posibles aplicaciones son muy diversas, por ejemplo :

1) Adquisición de datos y capacidad de procesarlos fuera de línea en una microcomputadora empleando algún lenguaje de alto nivel

2) Procesamiento en tiempo real de señales (en cierto rango de frecuencia)

3) Control lógico de secuencias

4) Instrumentación electrónica empleando microcomputadoras y microprocesadores

5) Auxiliar en el diseño, construcción y evaluación de prototipos de instrumentos controlados digitalmente

6) Enseñanza de la teoria y práctica de los microprocesadores

... entre otras muchas aplicaciones de campo, en la industria, en el laboratorio, etc.

Si bien este proyecto se desarrolló alrededor del microprocesador 8088 de Intel Corp., es claro que la misma idea básica puede realizarse empleando otros procesadores como el Z-80, 8085, 6809, 8086 (con la memoria (EPROM y RAM) organizada en dos

"bancos" de hasta 524288 bytes c/u : direcciones pares e impares, en vez de un solo "banco" como con el 8088) o incluso otros más evolucionados como el 80188/86, 80286, 80386 de Intel, los de la serie 68000 de Motorola, o los de la serie TMS de Texas Instruments, etc.); también pueden usarse Microcontroladores como el 8031/8032 {ROM externa}, el 8051/8052 {ROM interna} de 8 bits o los de la familia 80188/80186 de 16 bits de alta integración.

Se planea desarrollar a futuro, nuevas versiones de **SDMxxx-PC (PS)** basadas en otro Microprocesador y en algún Microcontrolador.

+

A P E N D I C E A

CONJUNTO DE INSTRUCCIONES DEL MICROPROCESADOR 8088

8088 Instruction Set Summary

Table D-1 summarizes the 8088 instruction set in alphabetical order. For each instruction, it shows the general assembler format and which flags are affected. In the *Flags* column, - means unchanged, * means may have changed, and ? means undefined.

Table D-1. 8088 Instruction set.

			Flags								
Mnemonic	*Assembler Format*		*OF*	*DF*	*IF*	*TF*	*SF*	*ZF*	*AF*	*PF*	*CF*
AAA	AAA		?	-	-	-	?	?	*	?	*
AAD	AAD		?	-	-	-	*	*	?	*	?
AAM	AAM		?	-	-	-	*	*	?	*	?
AAS	AAS		?	-	-	-	?	?	*	?	*
ADC	ADC	destination,source	*	-	-	-	*	*	*	*	*
ADD	ADD	destination,source	*	-	-	-	*	*	*	*	*
AND	AND	destination,source	0	-	-	-	*	*	?	*	0
CALL	CALL	target	-	-	-	-	-	-	-	-	-
CBW	CBW		-	-	-	-	-	-	-	-	-
CLC	CLC		-	-	-	-	-	-	-	-	0
CLD	CLD		-	0	-	-	-	-	-	-	-
CLI	CLI		-	-	0	-	-	-	-	-	-
CMC	CMC		-	-	-	-	-	-	-	-	*
CMP	CMP	destination,source	*	-	-	-	*	*	*	*	*
CMPS	CMPS	dest-string,source-string	*	-	-	-	*	*	*	*	*
CMPSB	CMPSB		*	-	-	-	*	*	*	*	*
CMPSW	CMPSW		*	-	-	-	*	*	*	*	*
CWD	CWD		-	-	-	-	-	-	-	-	-
DAA	DAA		?	-	-	-	*	*	*	*	*

Table D-1. 8088 Instruction set (continued).

			Flags								
Mnemonic	*Assembler Format*		*OF*	*DF*	*IF*	*TF*	*SF*	*ZF*	*AF*	*PF*	*CF*
DAS	DAS		?	-	-	-	*	*	*	*	*
DEC	DEC	destination	*	-	-	-	*	*	*	*	-
DIV	DIV	source	?	-	-	-	?	?	?	?	?
ESC	ESC	ext-opcode,source	-	-	-	-	-	-	-	-	-
HLT	HLT		-	-	-	-	-	-	-	-	-
IDIV	IDIV	source	?	-	-	-	?	?	?	?	?
IMUL	IMUL	source	*	-	-	-	?	?	?	?	*
IN	IN	accumulator,port	-	-	-	-	-	-	-	-	-
INC	INC	destination	*	-	-	-	*	*	*	*	-
INT	INT	interrupt-type	-	-	0	0	-	-	-	-	-
INTO	INTO		-	-	0	0	-	-	-	-	-
IRET	IRET		*	*	*	*	*	*	*	*	*
JA/JNBE	JA	short-label	-	-	-	-	-	-	-	-	-
JAE/JNB	JAE	short-label	-	-	-	-	-	-	-	-	-
JB/JNAE/JC	JB	short-label	-	-	-	-	-	-	-	-	-
JBE/JNA	JBE	short-label	-	-	-	-	-	-	-	-	-
JCXZ	JCXZ	short-label	-	-	-	-	-	-	-	-	-
JE/JZ	JE	short-label	-	-	-	-	-	-	-	-	-
JG/JNLE	JG	short-label	-	-	-	-	-	-	-	-	-
JGE/JNL	JGE	short-label	-	-	-	-	-	-	-	-	-
JL/JNGE	JL	short-label	-	-	-	-	-	-	-	-	-
JLE/JNG	JLE	short-label	-	-	-	-	-	-	-	-	-
JMP	JMP	target	-	-	-	-	-	-	-	-	-
JNC	JNC	short-label	-	-	-	-	-	-	-	-	-
JNE/JNZ	JNE	short-label	-	-	-	-	-	-	-	-	-
JNO	JNO	short-label	-	-	-	-	-	-	-	-	-
JNP/JPO	JNP	short-label	-	-	-	-	-	-	-	-	-
JNS	JNS	short-label	-	-	-	-	-	-	-	-	-
JO	JO	short-label	-	-	-	-	-	-	-	-	-
JP/JPE	JP	short-label	-	-	-	-	-	-	-	-	-
JS	JS	short-label	-	-	-	-	-	-	-	-	-
LAHF	LAHF		-	-	-	-	-	-	-	-	-
LDS	LDS	reg16,mem32	-	-	-	-	-	-	-	-	-
LEA	LEA	reg16,mem16	-	-	-	-	-	-	-	-	-
LES	LES	reg16,mem32	-	-	-	-	-	-	-	-	-
LOCK	LOCK		-	-	-	-	-	-	-	-	-
LODS	LODS	source-string	-	-	-	-	-	-	-	-	-
LODSB	LODSB		-	-	-	-	-	-	-	-	-
LODSW	LODSW		-	-	-	-	-	-	-	-	-
LOOP	LOOP	short-label	-	-	-	-	-	-	-	-	-
LOOPE/ LOOPZ	LOOPE	short-label	-	-	-	-	-	-	-	-	-

Table D-1. 8088 instruction set (continued).

			Flags								
Mnemonic	*Assembler Format*		*OF*	*DF*	*IF*	*TF*	*SF*	*ZF*	*AF*	*PF*	*CF*
LOOPNE/ LOOPNZ	LOOPNE	short-label	-	-	-	-	-	-	-	-	-
MOV	MOV	destination,source	-	-	-	-	-	-	-	-	-
MOVS	MOVS	dest-string,source-string	-	-	-	-	-	-	-	-	-
MOVSB	MOVSB		-	-	-	-	-	-	-	-	-
MOVSW	MOVSW		-	-	-	-	-	-	-	-	-
MUL	MUL	source	*	-	-	-	?	?	?	?	*
NEG	NEG	destination	*	-	-	-	*	*	*	*	*
NOP	NOP		-	-	-	-	-	-	-	-	-
NOT	NOT	destination	-	-	-	-	-	-	-	-	-
OR	OR	destination,source	0	-	-	-	*	*	?	*	0
OUT	OUT	port,accumulator	-	-	-	-	-	-	-	-	-
POP	POP	destination	-	-	-	-	-	-	-	-	-
POPF	POPF		*	*	*	*	*	*	*	*	*
PUSH	PUSH	source	-	-	-	-	-	-	-	-	-
PUSHF	PUSHF		-	-	-	-	-	-	-	-	-
RCL	RCL	destination,count	*	-	-	-	-	-	-	-	*
RCR	RCR	destination,count	*	-	-	-	-	-	-	-	*
REP	REP		-	-	-	-	-	-	-	-	-
REPE/REPZ	REPE		-	-	-	-	-	-	-	-	-
REPNE/ REPNZ	REPNE		-	-	-	-	-	-	-	-	-
RET	[pop-value]		-	-	-	-	-	-	-	-	-
ROL	ROL	destination,count	*	-	-	-	-	-	-	-	*
ROR	ROR	destination,count	*	-	-	-	-	-	-	-	*
SAHF	SAHF		-	-	-	-	*	*	*	*	*
SAL/SHL	SAL	destination,count	*	-	-	-	*	*	?	*	*
SAR	SAR	destination,count	*	-	-	-	*	*	?	*	*
SBB	SBB	destination,source	*	-	-	-	*	*	*	*	*
SCAS	SCAS	dest-string	*	-	-	-	*	*	*	*	*
SCASB	SCASB		*	-	-	-	*	*	*	*	*
SCASW	SCASW		*	-	-	-	*	*	*	*	*
SHR	SHR	destination,count	*	-	-	-	0	*	?	*	*
STC	STC		-	-	-	-	-	-	-	-	1
STD	STD		-	1	-	-	-	-	-	-	-
STI	STI		-	-	1	-	-	-	-	-	-
STOS	STOS	dest-string	-	-	-	-	-	-	-	-	-
STOSB	STOSB		-	-	-	-	-	-	-	-	-
STOSW	STOSW		-	-	-	-	-	-	-	-	-
SUB	SUB	destination,source	*	-	-	-	*	*	*	*	*

Table D-1. 8088 instruction set (continued).

			Flags								
Mnemonic	*Assembler Format*		*OF*	*DF*	*IF*	*TF*	*SF*	*ZF*	*AF*	*PF*	*CF*
TEST	TEST	destination,source	0	-	-	-	*	*	?	*	0
WAIT	WAIT		-	-	-	-	-	-	-	-	-
XCHG	XCHG	destination,source	-	-	-	-	-	-	-	-	-
XLAT	XLAT	source-table	-	-	-	-	-	-	-	-	-
XOR	XOR	destination,source	0	-	-	-	*	*	?	*	0

Table 1-15 Effective Address Calculation Time

EA COMPONENTS		CLOCKS*
Displacement Only		6
Base or Index Only	(BX,BP,SI,DI)	5
Displacement + Base or Index	(BX,BP,SI,DI)	9
Base + Index	BP + DI, BX + SI	7
	BP + SI, BX + DI	8
Displacement + Base + Index	BP + DI + DISP	11
	BX + SI + DISP	
	BP + SI + DISP	12
	BX + DI + DISP	

*Add 2 clocks for segment override

effected by the interaction of the EU and BIU when memory operands must be read or written. If the EU accesses to memory, it may have to wait for up to nine clock if the BIU has already started an instruction fetch bus cycle. (The EU can detect the need for a memory operand and post a bus request far enough in advance of its need for this operand to avoid waiting a full 4-clock bus cycle). If the queue is full the EU does not have to wait because the BIU is idle. (This assumes the BIU can obtain the bus on demand and no other processors are competing for the bus).

With typical instruction mixes, the time actually required to execute a sequence of instructions will be within 5 – 10% of the sum of the individual timings provided in Table 1-16. Cases can be constructed, however, in which execution time may be much higher than the sum of the figures provided in the table. The execution time for a given sequence of instructions is always repeatable, assuming comparable external conditions (interrupts, coprocessor activity, etc.). If the execution time for a given series of instructions must be determined exactly, the instructions should be run on an execution vehicle such as the iSBC 88/25 or 86/30 board.

MACHINE INSTRUCTION ENCODING AND DECODING

Machine instruction encoding and decoding is primarily the concern of the programmer. It is presented here for the hardware designer since such encoding and decoding

Table 1-16 Instruction Set Reference Data

AAA — AAA (no operands); ASCII adjust for addition — Flags O D I T S Z A P C: U U U X U X

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	8(8)	—	1	AAA

AAD — AAD (no operands); ASCII adjust for division — Flags O D I T S Z A P C: U X X U X U

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	60(15)	—	2	ADD

AAM — AAM (no operands); ASCII adjust for multiply — Flags O D I T S Z A P C: U X X U X U

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	83(19)	—	2	AAM

AAS — AAS (no operands); ASCII adjust for subtraction — Flags O D I T S Z A P C: U U U X U X

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	8(7)	—	1	AAS

*For the 8086 (80186) add four clocks for each 16-bit word transfer with an odd address. For the 8088 (80188) add four clocks for each 16-bit word transfer.

8086/8088 CPU

Table 1-16 Instruction Set Reference Data (continued)

ADC — ADC destination, source; Add with carry — Flags O D I T S Z A P C: X X X X X

Operands	Clocks	Transfers*	Bytes	Coding Example
register, register	3(3)	—	2	ADC AX, SI
register, memory	9(10) + EA	1	2-4	ADC CX, BETA [SI]
memory, register	16(10) + EA	2	2-4	ADC ALPHA [BX] [SI], DI
register, immediate	4(4)	—	3-4	ADC BX, 256
memory, immediate	17(16) + EA	2	3-6	ADC GAMMA, 30H
accumulator, immediate	4(3-4)	—	2-3	ADC AL, 5

ADD — ADD destination, source; Addition — Flags O D I T S Z A P C: X X X X X

Operands	Clocks	Transfers*	Bytes	Coding Example
register, register	3(3)	—	2	ADD CX, DX
register, memory	9(10) + EA	1	2-4	ADD DI, [BX].ALPHA
memory, register	16(10) + EA	2	2-4	ADD TEMP, CL
register, immediate	4(4)	—	3-4	ADD CL, 2
memory, immediate	17(16) + EA	2	3-6	ADD ALPHA, 2
accumulator, immediate	4(3-4)	—	2-3	ADD AX, 200

AND — AND destination, source; Logical and — Flags O D I T S Z A P C: O X X U X O

Operands	Clocks	Transfers*	Bytes	Coding Example
register, register	3(3)	—	2	AND AL, BL
register, memory	9(10) + EA	1	2-4	AND CX, FLAB WORD
memory, register	16(10) + EA	2	2-4	AND ASCII [DI], AL
register, immediate	4(4)	—	3-4	AND CX, 0F0H
memory, immediate	17(16) + EA	2	3-6	AND BETA, 01H
accumulator, immediate	4(3-4)	—	2-3	AND AX, 01010000B

BOUND — BOUND destination, source; Array bounds check — Flags O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
register, memory	(35)	2	2	BOUND AX, ALPHA

CALL — CALL target; Call a procedure — Flags O D I T S Z A P C: X X X X X

Operands	Clocks	Transfers*	Bytes	Coding Example
near-proc	19(14)	1	3	CALL NEAR_PROC
far-proc	28(23)	2	5	CALL FOR_PROC
memptr 16	21(19) + EA	2	2-4	CALL PROC_TABLE [SI]
regptr 16	16(13)	1	2	CALL AX
memptr 32	37(38) + EA	4	2-4	CALL [BX].TASK [SI]

CBW — CBW (no operands); Convert byte to word — Flags O D I T S Z A P C: U U U X U X

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2(2)	—	1	CBW

*For the 8086 (80[illegible] four clocks for each 16-bit word transfer with an [illegible]word address. For the 8088 (80188) add four clocks for each 16[illegible] word

Table 1-15 Instruction Set Reference Data (continued)

CLC	CLC (no operands) Clear carry flag			Flags O D I T S Z A P C 0
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	2(2)	—	1	CLC

CLD	CLD (no operands) Clear direction flag			Flags O D I T S Z A P C 0
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	2(2)	—	1	CLD

CLI	CLI (no operands) Clear interrupt flag			Flags O D I T S Z A P C 0
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	2(2)	—	1	CLI

CMC	CMC (no operands) Complement carry flag			Flags O D I T S Z A P C X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	2(2)	—	1	CMC

CMP	CMP destination, source Compare destination to source			Flags O D I T S Z A P C X X X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, register	3(3)	—	2	CMP BX, CX
register, memory	9(10) + EA	1	2-4	CMP DH, ALPHA
memory, register	9(10) + EA	1	2-4	CMP [BP + 2], SI
register, immediate	4(3) + EA	—	3-4	CMP BL, 02H
memory, immediate	10(10) + EA	1	3-6	CMP [BX].RADAR [DI], 3420H
accumulator, immediate	4(3-4)	—	2-3	CMP AL, 00010000B

CMPS	CMPS des-string, source-string Compare string			Flags O D I T S Z A P C X X X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
dest-string, source-string	22(22)	2	1	CMPS BUSS1, BUFF2
(repeat) dest-string, source-string	9 + 22/rep (5 + 22/rep)	2/rep	1	REPE CMPS ID, KEY

CWD	CWD (no operands) Convert word to doubleword			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	5(4)	—	1	CWD

DAA	DAA (no operands) Decimal adjust for addition			Flags O D I T S Z A P C X X X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	4(4)	—	1	DAA

*For the 8088 (80188) add four clocks for each 16-bit word transfer with an odd address. For the 8088 (80188) add four clocks for each 16-bit word transfer.

8086/8088 CPU

Table 1-16 Instruction Set Reference Data (continued)

DAS	DAS (no operands) Decimal adjust for subtraction			Flags O D I T S Z A P C U X X X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	4(4)	—	1	DAS

DEC	DEC destination Decrement by 1			Flags O D I T S Z A P C X X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg 16	3(3)	—	1	DEC AX
reg8	3(3)	—	2	DEC AL
memory	15(15) + EA	2	2-4	DEC ARRAY [SI]

DIV	DIV source Division, unsigned			Flags O D I T S Z A P C U U U U U
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg 8	80-90(29)	—	2	DIV CL
reg 16	144-162(38)	—	2	DIV BX
mem 8	86-96 + EA (35)	1	2-4	DIV ALPHA
mem 16	150-168 + EA(94)	1	2-4	DIV TABLE [SI]

ENTER	ENTER Procedure entry			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
locals, level	L = 0(15) L = 1(25) L > 1 (22 + 16(n − 1))	—	4	ENTER 28, 3

ESC	ESC external-opcode, source Escape			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
immediate, memory	8(6) + EA	1	2-4	ESC 6,ARRAY [SI]
immediate, register	2(2)	—	2	ESC 20, AL

HLT	HLT (no operands) Halt			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	2(2)	—	1	HLT

*For the 8088 (80188) add four clocks for each 16-bit word transfer with an odd address. For the 8088 (80188) add four clocks for each 16-bit word transfer

Table 1-16 Instruction Set Reference Data (continued)

IDIV	IDIV source Integer division			Flags O D I T S Z A P C U U U U U U
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg 8	101-112 (44-52)	—	2	IDIV BL
reg 16	165-184 (53-61)	—	2	IDIV CX
mem 8	107-118 + EA(50-56)	1	2-4	IDIV DIVISOR_BYTE [SI]
mem 16	171-190 + EA(58-67)	1	2-4	IDIV [BX].DIVISOR_WORD

IMUL	IMUL source Integer multiplication			Flags O D I T S Z A P C X U U U U X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
immed 8	(22-24)	—	3	IMUL 6
immed 16	(29-32)	—	4	IMUL 20
reg 8	80-98 (25-28)	—	2	IMUL CL
reg 16	128-154 (34-37)	—	2	IMUL BX
mem 8	86-104 + EA(31-34)	1	2-4	IMUL RATE_BYTE
mem 16	134-160 + EA(40-43)	1	2-4	IMUL RATE_WORD [BP] [DI]

IN	IN accumulator, port Input byte or word			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
accumulator, immed 8	10(10)	1	2	IN AL, 0FFEAH
accumulator, DX	8(8)	1	1	IN AX, DX

INC	INC destination Increment by 1			Flags O D I T S Z A P C X X X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg 16	3(3)	—	1	INC CX
reg 8	3(3)	—	2	INC BL
memory	15(15) + EA	2	2-4	INC ALPHA [DI] [BX]

INS	INS source-string, port Input string			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
dest-string, port	(14)	2	1	INS BUFF1, USART D
(repeat) dest-string, port	(9 + 8/rep)	2/rep	1	REP INS BUFF1, USART D

*For the 8086 (80186) add four clocks for each 16-bit word transfer with an odd address. For the 8088 (80188) add four clocks for each 16-bit word transfer.

8086/8088 CPU

Table 1-16 Instruction Set Reference Data (continued)

INT	INT interrupt-type Interrupt			Flags O D I T S Z A P C O O
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
immed 8 (type = 3)	52(45)	5	1	INT 3
immed 8 (type ≠ 3)	52(47)	5	2	INT 67

INTR†	INTR (external maskable interrupt) Interrupt if INTER and IF = 1			Flags O D I T S Z A P C O O
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	61	7	N/A	N/A

INTO	INTO (no operands) Interrupt if overflow			Flags O D I T S Z A P C O O
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	53 or 4(48 or 4)	5	1	INTO

IRET	IRET (no operands) Interrupt Return			Flags O D I T S Z A P C R R R R R R R R R
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	32(28)	3	1	IRET

JA/JNBE	JA/JNBE short-label Jump if above/Jump if not below nor equal			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
Short-label	16 or 4(13 or 4)	—	2	JA ABOVE

JAE/JNB	JAE/JNB short-label Jump if above or equal/Jump if not below			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4(13 or 4)	—	2	JAE ABOVE_EQUAL

JB/JNAE	JB/JNAE Jump if below/Jump if not above nor equal			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4(13 or 4)	—	2	JB BELOW

JBE/JNA	JBE/JNA short-label Jump if below or equal/Jump if not above			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4(13 or 4)	—	2	JNA NOT_ABOVE

*For the 8086 (80186) add four clocks for each 16-bit word transfer with an odd address. For the 8088 (80188) add four clocks for each 16-bit word transfer.

†INTR is not an instruction, it is included in table 1-16 only for timing information.

Table 1-16 Instruction Set Reference Data (continued)

JC	JC short-label Jump if carry			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4(13 or 4)	—	2	JC CARRY__SET

JCXZ	JCXZ short-label Jump if CX is zero			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4(16 or 5)	—	2	JCXZ COUNT__DONE

JE/JZ	JE/JZ short-label Jump if equal/Jump if zero			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4(13 or 4)	—	2	JZ ZERO

JG/JNLE	JG/JNLE short-label Jump if greater/Jump if not less nor equal			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4(13 or 4)	—	2	JG GREATER

JGE/JNL	JGE/JNL short-label Jump if greater or equal/Jump if not less			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4(13 or 4)	—	2	JGE GREATER__EQUAL

JL/JNGE	JL/JNGE short-label Jump if less/Jump if not greater nor equal			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4(13 or 4)	—	2	JL LESS

JLE/JNG	JLE/JNG short-label Jump if less or equal/Jump if not greater			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4(13 or 4)	—	2	JNG NOT__GREATER

*For the 8086 (80186) add four clocks for each 16-bit word transfer with an odd address. For the 8088 (80188) add four clocks for each 16-bit word transfer.

Table 1-16 Instruction Set Reference Data (continued)

JMP	JMP target Jump			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	15(13)	—	2	JMP SHORT
near-label	15(13)	—	3	JMP WITHIN__SEGMENT
far-label	15(13)	—	5	JMP FAR__LABEL
memptr 16	18(17) + EA	1	2-4	JMP [BX].TARGET
regptr 16	11(11)	—	2	JMP CX
memptr 32	24(26) + EA	2	2-4	JMP OTHER.SEG [SI]

JNC	JNC short-label Jump if not carry			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4(13 or 4)	—	2	JNC NOT__CARRY

JNE/JNZ	JNE/JNZ short-label Jump if not equal/Jump if not zero			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4(13 or 4)	—	2	JNE NOT__EQUAL

JNO	JNO short-label Jump if not overflow			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4(13 or 4)	—	2	JNO NO__OVERLOW

JNP/JPO	JNP/JPO short-label Jump if not parity/Jump if parity odd			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4(13 or 4)	—	2	JPO ODD__PARITY

JNS	JNS short-label Jump if not sign			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4(13 or 4)	—	2	JNS POSITIVE

JO	JO short-label Jump if overflow			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4(13 or 4)	—	2	JO SIGNED__OVRFLW

*For the 8086 (80186) add four clocks for each 16-bit word transfer with an odd address. For the 8088 (80188) add four clocks for each 16-bit word transfer

Table 1-16 Instruction Set Reference Data (continued)

JP/JPE	JP/JPE short-label Jump if parity/Jump if parity even			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4(13 or 4)	—	2	JPE EVEN__PARITY

JS	JS short-label Jump if sign			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4(13 or 4)	—	2	JS NEGATIVE

LAHF	LAHF (no operands) Load AH from flags			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	4(2)	—	1	LAHF

LDS	LDS destination, source Load pointer using DS			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg 16, mem 16	16(18) + EA	2	2-4	LDS SI, DATA, SEG [DI]

LEA	LEA destination, source Load effective address			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg 16, mem 16	2(6) + EA	—	2-4	LEA BX, [BP] [DI]

LEAVE	LEAVE (no operand) Restore stack for procedure exit			Flags O D I T S Z A P C U U X U X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	(8)	1	1	LEAVE

LES	LES destination, source Load pointer using ES			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg 16, mem 32	16(18) + EA	2	2-4	LES DI, [BX], TXT__BUFF

LOCK	LOCK (no operands) Lock bus			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	2(2)	—	1	LOCK XCHG FLAG, AL

*For the 8088 (80188) add four clocks for each 16-bit word transfer with an odd address. For the 8088 (80188) add four clocks for each 16-bit word transfer.

8086/8088 CPU

Table 1-16 Instruction Set Reference Data (continued)

LODS	LODS source-string Load string			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
source-string (repeat) source-string	12(10) 9 + 13/rep (6 + 11/rep)	1 1/rep	1 1	LODS CUSTOMER__NAME REP LODS NAME

LOOP	LOOP short-label Loop			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short label	17/5(15/5)	—	2	LOOP AGAIN

LOOPE/LOOPZ	LOOPE/LOOPZ short label Loop if equal/Loop if zero			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short label	18 or 6(16 or 6)	—	2	LOOPE AGAIN

LOOPNE/LOOPNZ	LOOPNE/LOOPNZ short label Loop if not equal/Loop if not zero			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short label	19 or 5(16 or 5)	—	2	LOOPNE AGAIN

NMI†	NMI (external nonmaskable interrupt) Interrupt if NMI = 1			Flags O D I T S Z A P C O O
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	50	5	N/A	N/A

MOV	MOV destination, source Move			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
memory, accumulator	10(9)	1	3	MOV ARRAY [SI], AL
accumulator, memory	10(8)	1	3	MOV AX, TEMP__RESULT
register, register	2(2)	—	2	MOV AX, CX
register, memory	8(12) + EA	1	2-4	MOV BP, STACK__TOP
memory, register	9(9) + EA	1	2-4	MOV COUNT [DI], CX
register, immediate	4(3-4)	—	2-3	MOV CL, 2
memory, immediate	10(12-13) + EA	1	3-6	MOV MASK [BX] [SI], 2CH
seg-reg, reg 16	2(2)	—	2	MOV ES, CX
seg-reg, mem 16	8(9) + EA	1	2-4	MOV DS, SEGMENT__BASE
reg 16, seg-reg	2(2)	—	2	MOV BP, SS
memory, seg-reg	9(11) + EA	1	2-4	MOV [BX], SEG__SAVE, CS

*For the 8088 (80188) add four clocks for each 16-bit word transfer with an odd address. For the 8088 (80188) add four clocks for each 16-bit word transfer.

†NMI is not an instruction, it is included in table 1-16 only for timing information.

Table 1-16 Instruction Set Reference Data (continued)

MOVS	MOVS dest-string, source-string Move string			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
dest-string, source-string	18(9)	2	1	MOVS LINE EDIT_DATA
(repeat) dest-string, source-string	9 + 17/rep (8 + 8/rep)	2/rep	1	REP MOVS SCREEN, BUFFER

MOVSB/MOVSW	MOVSB/MOVSW (no operands) Move string (byte/word)			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	18(9)	2	1	MOVSB
(repeat) (no operands)	9 + 17/rep (8 + 8/rep)	2/rep	1	REP MOVSW

MUL	MUL source Multiplication, unsigned			Flags O D I T S Z A P C X U U U U X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg 8	70-77 (26-28)	—	2	MUL BL
reg 16	118-133 (35-37)	—	2	MUL CX
mem 8	76-83 + EA(32-34)	1	2-4	MUL MONTH [SI]
mem 16	124-139 + EA(41-43)	1	2-4	MUL BAUD_RATE

NEG	NEG destination Negate			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register	3(3)	—	2	NEG AL
memory	16(3) + EA	2	2-4	NEG MULTIPLIER

*0 if destination is 0

NOP	NOP (no operands) No Operation			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	3(3)	—	1	NOP

NOT	NOT destination Logical not			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register	3(3)	—	2	NOT AX
memory	16(3) + EA	2	2-4	NOT CHARACTER

*For the 8086 (80186) add four clocks for each 16-bit word transfer with an odd address. For the 8088 (80188) add four clocks for each 16-bit word transfer.

Table 1-16 Instruction Set Reference Data (continued)

OR	OR destination, source Logical inclusive or			Flags O D I T S Z A P C O X X U X O
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, register	3(3)	—	2	OR AL, BL
register, memory	9(10) + EA	1	2-4	OR DX, PORT_ID [DI]
memory, register	16(10) + EA	2	2-4	OR FLAG_BYTE, CL
accumulator, immediate	4(3-4)	—	2-3	OR AL, 01101100B
register, immediate	4(4)	—	3-4	OR CX, 01H
memory, immediate	17(16) + EA	2	3-6	OR [BX], CMD_WORD, 0CFH

OUT	OUT port, accumulator Output byte or word			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
Immed 8, accumulator	10(9)	1	2	OUT 44, AX
DX, accumulator	8(7)	1	1	OUT DX, AL

OUTS	OUTS port, source-string Output string			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
port, source-string	(14)	2	1	OUTS PORT2, BUFF2
(repeat) port, source-string	(8) + 8/rep)	2/rep	1	REP OUTS PORT2, BUFF2

POP	POP destination Pop word off stack			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register	8(10)	1	1	POP DX
seg-reg (CS illegal)	8(8)	1	1	POP DS
memory	17(20) + EA	2	2-4	POP PARAMETER

POPA	POPA (no operands) Pop all registers			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	(51)	8	1	POPA

POPF	POPF (no operands) Pop all registers			Flags O D I T S Z A P C R R R R R R R R
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	8(8)	1	1	POPF

PUSH	PUSH source Push word onto stack			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register	11(10)	1	1	PUSH SI
seg-reg (CS legal)	10(9)	1	1	PUSH ES
memory	16(16) + EA	2	2-4	PUSH DRETURN_CODE [SI]

*For the 8086 (80186) add four clocks for each 16-bit word transfer with an odd address. For the 8088 (80188) add four clocks for each 16-bit word transfer.

Table 1-16 Instruction Set Reference Data (continued)

PUSHA	PUSHA (no operands) Push all registers			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	(36)	8	4	PUSHA

PUSHF	PUSHF (no operands) Push flags onto stack			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	10(9)	1	1	PUSHF

RCL	RCL destination, count Rotate left through carry			Flags O D I T S Z A P C X C
Operands	Clocks	Transfers*	Bytes	Coding Example
register, 1	2(2)	—	2	RCL CX, 1
register, CL	8+4/bit(5+1/bit)	—	2	RCL AL, CL
memory, 1	15(15)+EA	2	2-4	RCL ALPHA, 1
memory CL	20+4/bit(17+1/bit)+EA	2	2-4	RCL [BP], PARM, CL
register, n	(5+1/bit)	—	3	RCL CX, 5
memory, n	(17+1/bit)	2	3-5	RCL ALPHA, 5

RCR	RCR destination, count Rotate right through carry			Flags O D I T S Z A P C X C
Operands	Clocks	Transfers*	Bytes	Coding Example
register, 1	2(2)	—	2	RCR BX, 1
register, CL	8+4/bit(5+1/bit)	—	2	RCR BL, CL
memory, 1	15(15)+EA	2	2-4	RCR [BX], STATUS, 1
memory CL	20+4/bit(17+1/bit)+EA	2	2-4	RCR ARRAY, [DI], CL
register, n	(5+1/bit)	—	3	RCR BX, 5
memory, n	(17+1/bit)	2	3-5	RCR ALPHA, 5

REP	REP (no operands) Repeat string operation			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2(2)	—	1	REP MOVS DEST, SRCE

REPE/REPZ	REPE/REPZ (no operands) Repeat string operation while equal/while zero			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2(2)	—	1	REPE CMPS DATA, KEY

*For the 8086 (80186) add four clocks for each 16-bit word transfer with an odd address. For the 8088 (80188) add four clocks for each 16-bit word transfer.

Table 1-16 Instruction Set Reference Data (continued)

REPNE/REPNZ	REPNE/REPNZ (no operands) Repeat string operation while not equal/not zero			Flags O D I T S Z A P C U U U X U X
Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2(2)	—	1	REPNE SCAS INPUT_LINE

RET	RET optional-pop-value Return from procedure			Flags O D I T S Z A P C
Operands	Clocks	Transfers*	Bytes	Coding Example
(intra-segment, no pop)	16(16)	1	1	RET
(intra-segment, pop)	20(18)	1	3	RET 4
(inter-segment, no pop)	26(22)	2	1	RET
(inter-segment, pop)	25(25)	2	3	RET 2

ROL	ROL destination, count Rotate left			Flags O D I T S Z A P C X X
Operands	Clocks	Transfers*	Bytes	Coding Example
register, 1	2(2)	—	2	ROL BX, 1
register, CL	8+4/bit(5+1/bit)	—	2	ROL DI, CL
memory, 1	15(15)+EA	2	2-4	ROL FLAG_BYTE [DI],1
memory CL	20+4/bit(17+1/bit)+EA	2	2-4	ROL ALPHA, CL
register, n	(5+1/bit)	—	3	ROL BX, 5
memory, n	(17+1/bit)	2	3-5	ROL BETA, 5

ROR	ROR destination, count Rotate right			Flags O D I T S Z A P C X X
Operands	Clocks	Transfers*	Bytes	Coding Example
register, 1	2(2)	—	2	ROR BX, 1
register, CL	8+4/bit(5+1/bit)	—	2	ROR BX, CL
memory, 1	15(15)+EA	2	2-4	ROR PORT_STATUS, 1
memory CL	20+4/bit(17+1/bit)+EA	2	2-4	ROR CMD_WORD, CL
register, n	(5+1/bit)	—	3	ROR BX, 5
memory, n	(17+1/bit)	2	3-5	ROR BETA, 5

SAHF	SAHF (no operands) Store AH into flags			Flags O D I T S Z A P C R R R R R
Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	4(3)	—	1	SAHF

*For the 8086 (80186) add four clocks for each 16-bit word transfer with an odd address. For the 8088 (80188) add four clocks for each 16-bit word transfer.

Table 1-16 Instruction Set Reference Data (continued)

SAL/SHL	SAL/SHL destination Shift arithmetic left/Shift logical left			Flags O D I T S Z A P C X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, 1	2(2)	—	2	SAL AL, 1
register, CL	8 + 4/bit(5 + 1/bit)	—	2	SAL DI, CL
memory, 1	15(15) + EA	2	2-4	SAL [BX], OVERDRAW, 1
memory, CL	20 + 4/bit(17 + 1/bit) + EA	2	2-4	SAL STORE_COUNT, CL
register, n	(5 + 1/bit)	—	3	SAL AH, 5
memory, n	(17 + 1/bit)	2	3-5	SAL ALPHA, 5

SAR	SAR destination, source Shift arithmetic right			Flags O D I T S Z A P C X X U X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, 1	2(2)	—	2	SAR DX, 1
register, CL	8 + 4/bit(5 + 1/bit)	—	2	SAR DI, CL
memory, 1	15(15) + EA	2	2-4	SAR N_BLOCKS, 1
memory CL	20 + 4/bit(17 + 1/bit) + EA	2	2-4	SAR N_BLOCKS, CL
register, n	(5 + 1/bit)	—	3	SAR DX, 5
memory, n	(17 + 1/bit)	2	3-5	SAR DGLTH, 5

SBB	SBB destination, source Subtract with borrow			Flags O D I T S Z A P C X X X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, register	3(3)	—	2	SBB BX, CX
register, memory	9(10) + EA	1	2-4	SBB DI, [BX], PAYMENT
memory, register	16(10) + EA	2	2-4	SBB BALANCE, AX
accumulator, immediate	4(3-4)	—	2-3	SBB AX, 2
register, immediate	4(4)	—	3-4	SBB CL, 1
memory, immediate	17(16) + EA	2	3-6	SBB COUNT, [SI], 10

SCAS	SCAS dest-string Scan string			Flags O D I T S Z A P C X X X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
dest-string	15(15)	1	1	SCAS INPUT_LINE
(repeat) dest-string	9 + 15/rep (5 + 15/rep)	1/rep	1	REPNE SCAS BUFFER

*For the 8086 (80186) add four clocks for each 16-bit word transfer with an odd address. For the 8088 (80188) add four clocks for each 16-bit word transfer.

8086/8088 CPU

Table 1-16 Instruction Set Reference Data (continued)

SEGMENT†	SEGMENT override prefix Override to specified segment			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	2(2)	—	1	MOV SS:PARAMETER AX

SHR	SHR destination, count Shift logical right			Flags O D I T S Z A P C X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, 1	2(2)	—	2	SHR SI, 1
register, CL	8 + 4/bit(5 + 1/bit)	—	2	SHR SI, CL
memory, 1	15(15) + EA	2	2-4	SHR ID_BYTE [SI] [BX], 1
memory CL	20 + 4/bit(17 + 1/bit) + EA	2	2-4	SHR INPUT_WORD, CL
register, n	(5 + 1/bit)	—	3	SHR SI, 5
memory, n	(17 + 1/bit)	2	3-5	SHR ALPHA, 5

SINGLE STEP†	SINGLE STEP (Trap flag interrupt) Interrupt if TF = 1			Flags O D I T S Z A P C O O
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	50	5	N/A	N/A

STC	STC (no operands) Set carry flag			Flags O D I T S Z A P C C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	2(2)	—	1	STC

STD	STD (no operands) Set direction flag			Flags O D I T S Z A P C 1
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	2(2)	—	1	STD

STI	STI (no operands) Set interrupt enable flag			Flags O D I T S Z A P C 1
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	2(2)	—	1	STI

*For the 8086 (80186) add four clocks for each 16-bit word transfer with an odd address. For the 8088 (80188) add four clocks for each 16-bit word transfer.

†ASM-86 incorporates the segment override prefix into the operand specification and not as a separate instruction. SEGMENT is included in table 1-16 only for timing information.

‡SINGLE STEP is not an instruction. It is included in table 1-16 only for timing information.

Table 1-16 Instruction Set Reference Data (continued)

STOS	STOS dest-string Store byte or word string			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
dest-string	11(10)	1	1	STOS PRINT_LINE
(repeat) dest-string	9 + 10/rep) (6 + 9/rep)	1/rep	1	REP STOS DISPLAY

SUB	SUB destination, source Subtraction			Flags O D I T S Z A P C X X X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, register	3(3)	—	2	SUB CX, BX
register, memory	9(10) + EA	1	2-4	SUB DX, MATH_TOTAL [SI]
memory, register	16(10) + EA	2	2-4	SUB [BP + 2], CL
accumulator, immediate	4(3-4)	—	2-3	SUB AL, 10
register, immediate	4(4)	—	3-4	SUB SI, 5280
memory, immediate	17(16) + EA	2	3-6	SUB [BP], BALANCE, 1000

TEST	TEST destination, source Test or non-destructive logical and			Flags O D I T S Z A P C O X X U X O
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, register	3(3)	—	2	TEST SI, DI
register, memory	9(10) + EA	1	2-4	TEST SI, END_COUNT
accumulator, immediate	4(3-4)	—	2-3	TEST AL, 00100000B
register, immediate	5(4)	—	3-4	TEST BX, 0CC4H
memory, immediate	11(10) + EA	—	3-6	TEST RETURN_COUNT, 01H

WAIT	WAIT (no operands) Wait while TEST pin not asserted			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	4 + 5(6)n	—	1	WAIT

XCHG	XCHG destination, source Exchange			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
accumulator, reg 16	3(3)	—	1	XCHG AX, BX
memory, register	17(17) + EA	2	2-4	XCHG SEMAPHORE, AX
register, register	4(4)	—	2	XCHG AL, BL

XLAT	XLAT source-table Translate			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
source-table	11(11)	1	1	XLAT ASCII_TAB

*For the 8086 (80186) add four clocks for each 16-bit word transfer with an odd address. For the 8088 (80188) add four clocks for each 16-bit word transfer.

Table 1-16 Instruction Set Reference Data (continued)

XOR	XOR destination, source Logical exclusive or			Flags O D I T S Z A P C O X X U X O
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, register	3(3)	—	2	XOR CX, BX
register, memory	9(10) + EA	1	2-4	XOR CL, MASK_BYTE
memory, register	16(10) + EA	2	2-4	XOR ALPHA [SI], DX
accumulator, immediate	4(3-4)	—	2-3	XOR AL, 01000010B
register, immediate	4(4)	—	3-4	XOR SI, 00C2H
memory, immediate	17(16) + EA	2	3-6	XOR RETURN_CODE, 0D2H

*For the 8086 (80186) add four clocks for each 16-bit word transfer with an odd address. For the 8088 (80188) add four clocks for each 16-bit word transfer

directly affects bus activity. As an example of the encoding and decoding process, consider writing a MOV instruction in ASM-86 in the form:

MOV destination,source

This will cause the assembler to generate 1 of 28 possible forms of the MOV machine instruction. A programmer rarely needs to know the details of machine instruction formats or encoding. An exception may occur during debugging when it may be necessary to monitor instructions fetched on the bus, read unformatted memory dumps, etc. This section provides the information necessary to translate or decode an 8086 or 8088 machine instruction.

To pack instructions into memory as densely as possible, the 8086 and 8088 CPUs utilize an efficient coding technique. Machine instructions vary from one to six bytes in length. One-byte instructions, which generally operate on single registers or flags, are simple to identify; the keys to decoding longer instructions are in the first two bytes. The format of these bytes can vary, but most instructions follow the format shown in Figure 1-28.

The first six bits of a multibyte instruction generally contain an opcode that identifies the basic instruction type: ADD, XOR, etc. The following bit, called the D field, generally specifies the "direction" of the operation: 1 = the REG field in the second byte identifies the destination operand, 0 = the REG field identifies the source operand. The W field distinguishes between byte and word operations: 0 = byte, 1 = word.

One of three additional single-bit fields, S, V or Z, appears in some instruction formats (refer to Table 1-17). S, in conjunction with W, indicates the sign extension of immediate fields in arithmetic instructions. V distinguishes between single-and variable-bit shifts and rotates. Z is a compare bit with the zero flag in conditional repeat and loop instructions.

The second byte of the instruction usually identifies the instruction's operands. The MOD (mode) field indicates whether one of the operands is in memory or whether both operands are registers (refer to Table 1-18). The REG (register) field identifies a register that is one of the instruction operands (refer to Table 1-19). In a number of instructions, particularly the immediate-to-memory variety, REG is used as an extension of the opcode to identify the type of operation. The encoding of the R/M (register/memory) field (refer to Table 1-20) depends on how the mode field is set. If MOD = 11

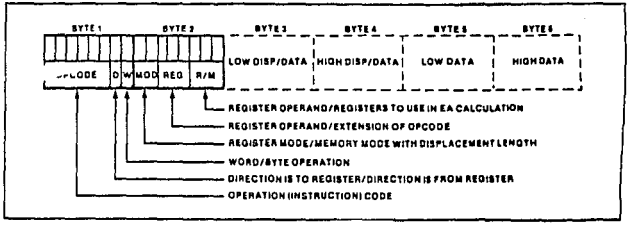


Figure 1-28 Typical 8086/88 Machine Instruction Format

Table 1-17 Single-Bit Field Encoding

Field	Value	Function
S	0	No sign extension
	1	Sign extend 8-bit immediate data to 16 bits if W=1
W	0	Instruction operates on byte data
	1	Instruction operates on word data
D	0	Instruction source is specified in REG field
	1	Instruction destination is specified in REG field
V	0	Shift/rotate count is one
	1	Shift/rotate count is specified in CL register
Z	0	Repeat/loop while zero flag is clear
	1	Repeat/loop while zero flag is set

(register-to-register mode), then R/M identifies the second register operand. If MOD selects memory mode, then R/M indicates how the effective address of the memory operand is to be calculated.

Table 1-18 Mode (MOD) Field Encoding

CODE	EXPLANATION
00	Memory Mode, no displacement follows*
01	Memory Mode, 8-bit displacement follows
10	Memory Mode, 16-bit displacement follows
11	Register Mode (no displacement)

*Except when R/M = 110, then 16-bit displacement follows

Table 1-19 REG (Register) Field Encoding

REG	W=0	W=1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Bytes 3 through 6 of an instruction are optional fields that usually contain the displacement value of a memory operand and/or the actual value of an immediate constant operand.

The displacement value may contain one or two bytes; the language translators generate one byte whenever possible. The MOD field indicates how many displacement bytes are present. Following Intel convention, if the displacement is two bytes, the most-significant byte is stored second in the instruction. If the displacement is only a single byte, the 8086 or 8088 automatically sign-extends this quantity to 16-bits before using the information in further address calculations. Immediate values always follow any displacement values that may be present. The second byte of a two-byte immediate value is the most significant.

Table 1-22 lists the instruction encodings for all 8086/8088 instructions. This table can be used to predict the machine encoding of any ASM-86 instruction. Table 1-23 lists the 8086/8088 machine instructions in order by the binary value of their first byte. This table can be used to decode any machine instruction from its binary representation. Table 1-21 is a key to the abbreviations used in Tables 1-22 and 1-23. Figure 1-29 is a more compact instruction decoding guide.

1.3 DEVICE PIN DEFINITIONS

The following paragraphs present functional descriptions of all input/output signals and electrical descriptions of all of the input/output pins on the 8086 and 8088 40-pin DIP's.

Table 1-20 Register/Memory Field Encoding

MOD = 11			EFFECTIVE ADDRESS CALCULATION			
R/M	W=0	W=1	R/M	MOD=00	MOD=01	MOD=10
000	AL	AX	000	(BX)+(SI)	(BX)+(SI)+D8	(BX)+(SI)+D16
001	CL	CX	001	(BX)+(DI)	(BX)+(DI)+D8	(BX)+(DI)+D16
010	DL	DX	010	(BP)+(SI)	(BP)+(SI)+D8	(BP)+(SI)+D16
011	BL	BX	011	(BP)+(DI)	(BP)+(DI)+D8	(BP)+(DI)+D16
100	AH	SP	100	(SI)	(SI)+D8	(SI)+D16
101	CH	BP	101	(DI)	(DI)+D8	(DI)+D16
110	DH	SI	110	DIRECT ADDRESS	(BP)+D8	(BP)+D16
111	BH	DI	111	(BX)	(BX)+D8	(BX)+D16

1.3.1 Functional Description of All Signals

Figure 1-30 shows the 8086/8088 DIP pin assignments and Table 1-24 provides a complete functional description of each device pin signal and correlates the description to the pin number and associated signal symbol.

1.3.2 Electrical Description of Pins

The absolute maximum ratings for the 8086/8088 device are as follows.

ABSOLUTE MAXIMUM RATINGS

Ambient Temperature Under Bias	0°C to 70°C
Storage Temperature	−65°C to +150°C
Voltage on Any Pin with Respect to GND	−1.0 to +7V
Power Dissipation	2.5 Watt

Stresses above those listed above may cause permanent damage to the device. These values present stress ratings only and functional operation of the device at these or any other conditions above those indicated in the operational sections of the device specifications is not implied. Exposure to absolute maximum conditions for extended periods of time may affect the device reliability.

Table 1-25 presents the D.C. voltage characteristics of the 8086/8088 CPU's. Table 1-26 lists the A.C. characteristics timing requirements and timing responses for minimum complexity systems, and Table 1-27 lists the A.C. characteristics timing requirements and timing responses for maximum complexity systems (using 8288 bus controller). Figure 1-31 and Figure 1-32 presents waveforms for the minimum mode and maximum mode operation related to the preceding A.C. characteristics tables.

1.3.3 OPERATING MODES

One of the unique features the 8086 and 8088 CPU's allow the user is the ability to select between two functional definitions of a subset of the 8086/8088 outputs. This enables the user to tailor the intended CPU system environment. This "system tailoring" is accomplished by strapping the CPU's MN/MX* (minimum/maximum) input pin. Table 1-28 defines the 8086 and 8088 pin assignments for both the minimum and maximum modes of operation.

In the minimum mode, the CPU's support small systems by strapping the MN/MX* pin to +5V. In this mode of operation, the 8086/8088 CPU generates all bus control signals (DT/R*, DEN*, ALE and either M/IO* or IO/M*) and the command output signals (RD*, WR* or INTA*). The CPU also provides a mechanism for requesting bus access (HOLD/HLDA) that is compatible with bus master type controllers (e.g., the Intel 8237A DMA Controller).

When a bus master requires bus access in the minimum mode, it activates the HOLD input to the CPU through its request logic. In response to the "hold" request, The CPU activates HLDA as an acknowledgement to the bus master, requesting the bus, and simultaneously floats the system bus and control lines. Since a bus request is asynchronous, the CPU samples the HOLD input on the positive transition of each CLK signal and activates HLDA at the end of either the current bus cycle (if a bus cycle is in progress) or idle clock period. The CPU maintains the hold state until the bus master inactivates the HOLD input. At that time the CPU regains control of the system bus. Note that during a "hold" state, the CPU continues to execute instructions until a bus cycle is required.

In the minimum mode, the I/O-memory control line for the 8088 CPU is the reverse of the corresponding control line for the 8086 CPU (M/IO* on the 8086 and IO/M* on the 8088). Since the 8088 CPU is an 8-bit device, this conditioning provides compatibility with existing MCS®-85 systems specific MCS-85 family devices (e.g., the Intel 8155/56).

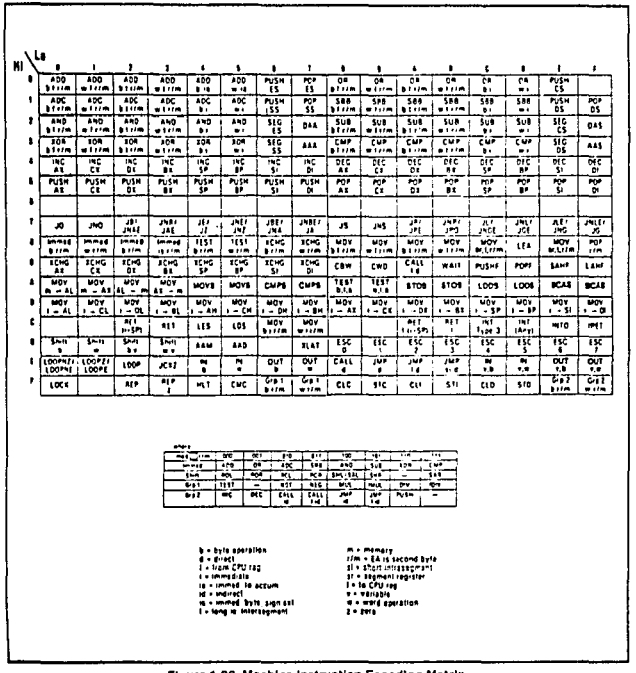


Figure 1-29 Machine Instruction Encoding Matrix

Table 1-21 Key to Machine Instruction Encoding and Decoding

IDENTIFIER	EXPLANATION
MOD	Mode field; described in this chapter.
REG	Register field; described in this chapter.
R/M	Register/Memory field; described in this chapter.
SR	Segment register code: 00=ES, 01=CS, 10=SS, 11=DS.
W, S, D, V, Z	Single-bit instruction fields; described in this chapter.
DATA-8	8-bit immediate constant.
DATA-SX	8-bit immediate value that is automatically sign-extended to 16-bits before use.
DATA-LO	Low-order byte of 16-bit immediate constant.
DATA-HI	High-order byte of 16-bit immediate constant.
(DISP-LO)	Low-order byte of optional 8- or 16-bit unsigned displacement; MOD indicates if present.
(DISP-HI)	High-order byte of optional 16-bit unsigned displacement; MOD indicates if present.
IP-LO	Low-order byte of new IP value
IP-HI	High-order byte of new IP value
CS-LO	Low-order byte of new CS value.
CS-HI	High-order byte of new CS value.
IP-INC8	8-bit signed increment to instruction pointer.
IP-INC-LO	Low-order byte of signed 16-bit instruction pointer increment.
IP-INC-HI	High-order byte of signed 16-bit instruction pointer increment.
ADDR-LO	Low-order byte of direct address (offset) of memory operand; EA not calculated.
ADDR-HI	High-order byte of direct address (offset) of memory operand; EA not calculated.
—	Bits may contain any value.
XXX	First 3 bits of ESC opcode.
YYY	Second 3 bits of ESC opcode.
REG8	8-bit general register operand.
REG16	16-bit general register operand.
MEM8	8-bit memory operand (any addressing mode).
MEM16	16-bit memory operand (any addressing mode).
IMMED8	8-bit immediate operand.
IMMED16	16-bit immediate operand.
SEGREG	Segment register operand.
DEST-STR8	Byte string addressed by DI.

In the maximum mode (MN/MX* pin strapped to ground), the Intel 8288 Bus controller is added to provide sophisticated bus control functions and compatibility with the MULTIBUS architecture. (Combining an Intel 8289 Arbiter with the 8288 permits the CPU to support multiple processors on the system bus.) The bus controller, instead of the CPU (see Figure 1-33), provides all bus control and command outputs. This allows the pins previously delegated to these functions to be redefined to support multiprocessing functions.

1.3.4 Minimum Mode System Overview/Description

The minimum mode 8086 (see Figure 1-34) is optimized

Table 1-21 Key to Machine Instruction Encoding and Decoding (continued)

IDENTIFIER	EXPLANATION
SRC-STR8	Byte string addressed by SI.
DEST-STR16	Word string addressed by DI.
SRC-STR16	Word string addressed by SI,
SHORT-LABEL	Label within ±127 bytes of instruction.
NEAR-PROC	Procedure in current code segment.
FAR-PROC	Procedure in another code segment.
NEAR-LABEL	Label in current code segment but farther than −128 to +127 bytes from instruction.
FAR-LABEL	Label in another code segment.
SOURCE-TABLE	XLAT translation table addressed by BX.
OPCODE	ESC opcode operand.
SOURCE	ESC register or memory operand.

for small to medium (one or two boards), single CPU systems. Minimum mode system architecture is directed at satisfying requirements of the lower to middle segment of high performance 16-bit applications. The CPU maintains the full megabyte memory space, 64K-byte I/O space and 16-bit data path. The CPU directly provides all bus control (DT/R*, DEN*, ALE, M/IO*), commands (RD*, WR*, INTA*) and a simple CPU preemption mechanism

Table 1-22 8086/88 Instruction Encoding

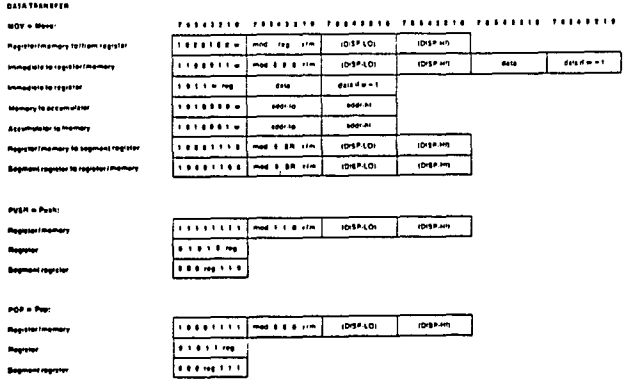


Table 1-22 8086/88 Instruction Encoding (continued)

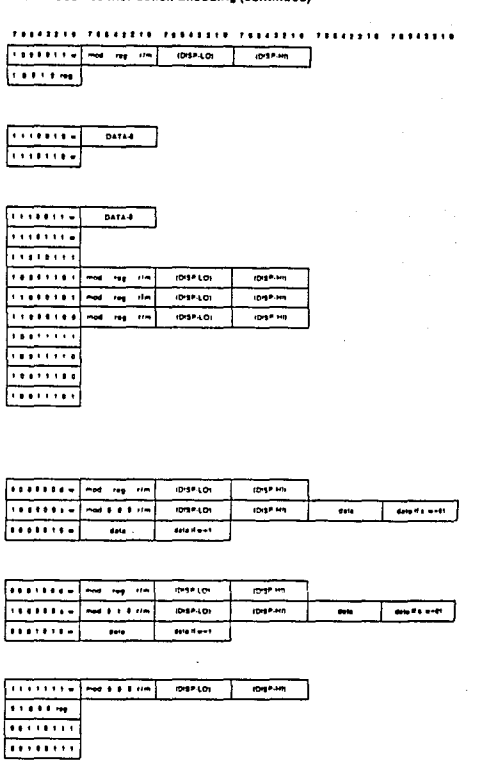


Table 1-22 8086/88 Instruction Encoding (continued)

ARITHMETIC (Cont'd.)

SUB = Subtract:

	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Reg/memory and register to either	0 0 1 0 1 0 d w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Immediate from register/memory	1 0 0 0 0 0 s w	mod 1 0 1 r/m	(DISP-LO)	(DISP-HI)	data	data if s: w=01
Immediate from accumulator	0 0 1 0 1 1 0 w	data	data if w=1			

SBB = Subtract with borrow:

Reg/memory and register to either	0 0 0 1 1 0 d w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Immediate from register/memory	1 0 0 0 0 0 s w	mod 0 1 1 r/m	(DISP-LO)	(DISP-HI)	data	data if s: w=01
Immediate from accumulator	0 0 0 1 1 1 0 w	data	data if w=1			

DEC Decrement:

Register/memory	1 1 1 1 1 1 1 w	mod 0 0 1 r/m	(DISP-LO)	(DISP-HI)
Register	0 1 0 0 1 reg			
AAS Change sign	1 1 1 1 0 1 1 w	mod 0 1 1 r/m	(DISP-LO)	(DISP-HI)

CMP = Compare

Register/memory and register	0 0 1 1 1 0 d w	mod reg r/m	(DISP-LO)	(DISP-HI)	
Immediate with register/memory	1 0 0 0 0 0 s w	mod 1 1 1 r/m	(DISP-LO)	(DISP-HI)	data	data if s: w=1
Immediate with accumulator	0 0 1 1 1 1 0 w	data			
AAS ASCII adjust for subtract	0 0 1 1 1 1 1 1				
DAS Decimal adjust for subtract	0 0 1 0 1 1 1 1				
MUL Multiply (unsigned)	1 1 1 1 0 1 1 w	mod 1 0 0 r/m	(DISP-LO)	(DISP-HI)	
IMUL Integer multiply (signed)	1 1 1 1 0 1 1 w	mod 1 0 1 r/m	(DISP-LO)	(DISP-HI)	
AAM ASCII adjust for multiply	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0	(DISP-LO)	(DISP-HI)	
DIV Divide (unsigned)	1 1 1 1 0 1 1 w	mod 1 1 0 r/m	(DISP-LO)	(DISP-HI)	
IDIV Integer divide (signed)	1 1 1 1 0 1 1 w	mod 1 1 1 r/m	(DISP-LO)	(DISP-HI)	
AAD ASCII adjust for divide	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0	(DISP-LO)	(DISP-HI)	
CBW Convert byte to word	1 0 0 1 1 0 0 0				
CWD Convert word to double word	1 0 0 1 1 0 0 1				

LOGIC

NOT Invert	1 1 1 1 0 1 1 w	mod 0 1 0 r/m	(DISP-LO)	(DISP-HI)
SAL/SAL Shift logical/arithmetic left	1 1 0 1 0 0 v w	mod 1 0 0 r/m	(DISP-LO)	(DISP-HI)
SAR Shift logical right	1 1 0 1 0 0 v w	mod 1 0 1 r/m	(DISP-LO)	(DISP-HI)
SAR Shift arithmetic right	1 1 0 1 0 0 v w	mod 1 1 1 r/m	(DISP-LO)	(DISP-HI)
ROL Rotate left	1 1 0 1 0 0 v w	mod 0 0 0 r/m	(DISP-LO)	(DISP-HI)

Table 1-22 8086/88 Instruction Encoding (continued)

LOGIC (Cont'd.)

	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
ROR Rotate right	1 1 0 1 0 0 v w	mod 0 0 1 r/m	(DISP-LO)	(DISP-HI)		
RCL Rotate through carry flag left	1 1 0 1 0 0 v w	mod 0 1 0 r/m	(DISP-LO)	(DISP-HI)		
RCR Rotate through carry right	1 1 0 1 0 0 v w	mod 0 1 1 r/m	(DISP-LO)	(DISP-HI)		

AND = And:

Reg/memory with register to either	0 0 1 0 0 0 d w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 0 0 r/m	(DISP-LO)	(DISP-HI)	data	data if w=1
Immediate to accumulator	0 0 1 0 0 1 0 w	data	data if w=1			

TEST = And function to flags no result:

Register/memory and register	1 0 0 0 0 1 0 w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Immediate data and register/memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m	(DISP-LO)	(DISP-HI)	data	data if w=1
Immediate data and accumulator	1 0 1 0 1 0 0 w	data				

OR = Or:

Reg/memory and register to either	0 0 0 0 1 0 d w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 0 0 1 r/m	(DISP-LO)	(DISP-HI)	data	data if w=1
Immediate to accumulator	0 0 0 0 1 1 0 w	data	data if w=1			

XOR = Exclusive or:

Reg/memory and register to either	0 0 1 1 0 0 d w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Immediate to register/memory	1 0 0 0 0 0 0 w	data	(DISP-LO)	(DISP-HI)	data	data if w=1
Immediate to accumulator	0 0 1 1 0 1 0 w	data	data if w=1			

STRING MANIPULATION

REP = Repeat	1 1 1 1 0 0 1 z
MOVS = Move byte/word	1 0 1 0 0 1 0 w
CMPS = Compare byte/word	1 0 1 0 0 1 1 w
SCAS = Scan byte/word	1 0 1 0 1 1 1 w
LODS = Load byte/wd to AL/AX	1 0 1 0 1 1 0 w
STOS = Stor byte/wd from AL/A	1 0 1 0 1 0 1 w

Table 1-22 8086/88 Instruction Encoding (continued)

CONTROL TRANSFER

CALL = Call:	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Direct within segment	1 1 1 0 1 0 0 0	IP-INC-LO	IP-INC-HI			
Indirect within segment	1 1 1 1 1 1 1 1	mod 0 1 0 r/m	(DISP-LO)	(DISP-HI)		
Direct intersegment	1 0 0 1 1 0 1 0	IP-lo	IP-hi			
		CS-lo	CS-hi			
Indirect intersegment	1 1 1 1 1 1 1 1	mod 0 1 1 r/m	(DISP-LO)	(DISP-HI)		

JMP = Unconditional Jump:

Direct within segment	1 1 1 0 1 0 0 1	IP-INC-LO	IP-INC-HI		
Direct within segment-short	1 1 1 0 1 0 1 1	IP-INC8			
Indirect within segment	1 1 1 1 1 1 1 1	mod 1 0 0 r/m	(DISP-LO)	(DISP-HI)	
Direct intersegment	1 1 1 0 1 0 1 0	IP-lo	IP-hi		
		CS-lo	CS-hi		
Indirect intersegment	1 1 1 1 1 1 1 1	mod 1 0 1 r/m	(DISP-LO)	(DISP-HI)	

RET = Return from CALL:

Within segment	1 1 0 0 0 0 1 1		
Within seg adding immed to SP	1 1 0 0 0 0 1 0	data-lo	data-hi
Intersegment	1 1 0 0 1 0 1 1		
Intersegment adding immed to SP	1 1 0 0 1 0 1 0	data-lo	data-hi

JE/JZ = Jump on equal/zero	0 1 1 1 0 1 0 0 IP-INC8
JL/JNGE = Jump on less/not greater or equal	0 1 1 1 1 1 0 0 IP-INC8
JLE/JNG = Jump on less or equal/not greater	0 1 1 1 1 1 1 0 IP-INC8
JB/JNAE = Jump on below/not above or equal	0 1 1 1 0 0 1 0 IP-INC8
JBE/JNA = Jump on below or equal/not above	0 1 1 1 0 1 1 0 IP-INC8
JP/JPE = Jump on parity/parity even	0 1 1 1 1 0 1 0 IP-INC8
JO = Jump on overflow	0 1 1 1 0 0 0 0 IP-INC8
JS = Jump on sign	0 1 1 1 1 0 0 0 IP-INC8
JNE/JNZ = Jump on not equal/not zero	0 1 1 1 0 1 0 1 IP-INC8
JNL/JGE = Jump on not less/greater or equal	0 1 1 1 1 1 0 1 IP-INC8
JNLE/JG = Jump on not less or equal/greater	0 1 1 1 1 1 1 1 IP-INC8
JNB/JAE = Jump on not below/above or equal	0 1 1 1 0 0 1 1 IP-INC8
JNBE/JA = Jump on not below or equal/above	0 1 1 1 0 1 1 1 IP-INC8
JNP/JPO = Jump on not par/par odd	0 1 1 1 1 0 1 1 IP-INC8
JNO = Jump on not overflow	0 1 1 1 0 0 0 1 IP-INC8

Table 1-22 8086/88 Instruction Encoding (continued)

CONTROL TRANSFER (Cont'd)

RET = Return from CALL:	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
JNS = Jump on not sign	0 1 1 1 1 0 0 1	IP-INC8				
LOOP = Loop CX times	1 1 1 0 0 0 1 0	IP-INC8				
LOOPZ/LOOPE = Loop while zero/equal	1 1 1 0 0 0 0 1	IP-INC8				
LOOPNZ/LOOPNE = Loop while not zero/equal	1 1 1 0 0 0 0 0	IP-INC8				
JCXZ = Jump on CX zero	1 1 1 0 0 0 1 1	IP-INC8				

INT = Interrupt:

Type specified	1 1 0 0 1 1 0 1	DATA-8
Type 3	1 1 0 0 1 1 0 0	
INTO = Interrupt on overflow	1 1 0 0 1 1 1 0	
IRET = Interrupt return	1 1 0 0 1 1 1 1	

PROCESSOR CONTROL

CLC = Clear carry	1 1 1 1 1 0 0 0			
CMC = Complement carry	1 1 1 1 0 1 0 1			
STC = Set carry	1 1 1 1 1 0 0 1			
CLD = Clear direction	1 1 1 1 1 1 0 0			
STD = Set direction	1 1 1 1 1 1 0 1			
CLI = Clear interrupt	1 1 1 1 1 0 1 0			
STI = Set interrupt	1 1 1 1 1 0 1 1			
HLT = Halt	1 1 1 1 0 1 0 0			
WAIT = Wait	1 0 0 1 1 0 1 1			
ESC = Escape (to external device)	1 1 0 1 1 x x x	mod y y y r/m	(DISP-LO)	(DISP-HI)
LOCK = Bus lock prefix	1 1 1 1 0 0 0 0			
SEGMENT = Override prefix	0 0 1 reg 1 1 0			

(HOLD, HLDA) compatible with existing DMA controllers (e.g., 8259A Interrupt Controller).

In the minimum mode the 8088 CPU provides an SS0 status output. This output is equivalent to S0 in the maximum mode and can be decoded with DT/R* and – IO/M*, which are equivalent to S1* and S2* respectively, to provide the same CPU cycle status information (see Table 1-29). This type of decoding could be used in a minimum mode 8088-based system to allow dynamic RAM refresh during passive CPU cycle.

1.3.6 Maximum Mode System Overview/Description

The maximum mode (see Figure 1-35) extends the system architecture to support multiprocessor configurations and local instruction set extension processors (coprocessors). By adding the 8288 bipolar bus controller, the 8086 outputs assigned to bus control and commands in the minimum mode are redefined to allow these extensions and enhance general system performance. Specifically, (1) two prioritized levels of processor preemption (RQ*/GT0*, RQ*/GT1*) allow multiple processors to reside on the 8086's local bus and share its interface to the system bus, (2) Queue status (QS0, QS1) is available to allow external devices like ICE™-86 or special instruction set extension co-processors (such as the 8087 Numeric Co-processor) to track the CPU instruction execution, (3) access control to shared resources in multiprocessor systems is supported by a hardware bus lock mechanism and (4) system command and configuration options are expanded via devices like the 8288 bus controller and 8289 bus arbiter.

QUEUE STATUS

The queue status indicates what information is being removed from the internal queue and when the queue is being reset due to a transfer of control (Table 1-30). By monitoring the S0*, S1*, S2* status lines for instructions entering the 8086 (1, 0, 0 indicates code access while A0 and BHE* indicate word or byte) and QS0, QS1 for instructions leaving the 8086's internal queue, it is possible to track the instruction execution. Since instructions are executed from the 8086's internal queue, the queue status is presented each CPU clock cycle and is not related to the bus cycle activity. This mechanism (1) allows a co-processor to detect execution of an ESCAPE instruction which directs the co-processor to perform a specific task and (2) allows ICE™-86 to trap execution of a specific memory location.

An example of a circuit used by ICE is given in Figure 1-36. The first up down counter tracks the depth of the queue while the second captures the queue depth on a match. The second counter decrements on further fetches from the queue until the queue is flushed or the count goes to zero indicating execution of the match address. The first counter decrements on fetch from the queue (QS0 = 1) and increments on code fetches into the queue. Note that a normal code fetch will transfer two bytes into the queue so two clock increments are given to the counter (T2OI and T3OI) unless a single byte is loaded over the upper half of the bus (A0-P is high). Since the execution unit (EU) is not synchronized to the bus interface unit (BIU), a fetch from the queue can occur simultaneously with a transfer into the queue. The Exclusive-OR gate driving the ENP input of the first counter allows these simultaneous operations to cancel each other and not modify the queue depth.

HARDWARE LOCK

To address the problem of controlling access to shared resources, the maximum mode 8086 provides a hardware LOCK* output. The LOCK* output is activated through the instruction stream by execution of the LOCK prefix instruction. The LOCK* output goes active in the first CPU clock cycle following execution of the prefix and remains active until the clock following the completion of the instruction following the LOCK prefix. To provide bus access control in multiprocessor systems, the LOCK* signal should be incorporated into the system bus arbitration logic resident to the CPU.

Table 1-23 Machine Instruction Decoding Guide

1ST BYTE HEX	1ST BYTE BINARY	2ND BYTE	BYTES 3, 4, 5, 6	ASM-86 INSTRUCTION FORMAT
00	0000 0000	MOD REG R/M	(DISP-LO),(DISP-HI)	ADD REG8/MEM8,REG8
01	0000 0001	MOD REG R/M	(DISP-LO),(DISP-HI)	ADD REG16/MEM16,REG16
02	0000 0010	MOD REG R/M	(DISP-LO),(DISP-HI)	ADD REG8,REG8/MEM8
03	0000 0011	MOD REG R/M	(DISP-LO),(DISP-HI)	ADD REG16,REG16/MEM16
04	0000 0100	DATA-8		ADD AL,IMMED8
05	0000 0101	DATA-LO	DATA-HI	ADD AX,IMMED16
06	0000 0110			PUSH ES
07	0000 0111			POP ES

Table 1-23 Machine Instruction Decoding Guide (continued)

1ST BYTE HEX	1ST BYTE BINARY	2ND BYTE	BYTES 3,4,5,6	ASM-86 INSTRUCTION FORMAT
08	0000 1000	MOD REG R/M	(DISP-LO),(DISP-HI)	OR REG8/MEM8,REG8
09	0000 1001	MOD REG R/M	(DISP-LO),(DISP-HI)	OR REG16/MEM16,REG16
0A	0000 1010	MOD REG R/M	(DISP-LO),(DISP-HI)	OR REG8,REG8/MEM8
0B	0000 1011	MOD REG R/M	(DISP-LO),(DISP-HI)	OR REG16,REG16/MEM16
0C	0000 1100	DATA-8		OR AL,IMMED8
0D	0000 1101	DATA-LO	DATA-HI	OR AX,IMMED16
0E	0000 1110			PUSH CS
0F	0000 1111			(not used)
10	0001 0000	MOD REG R/M	(DISP-LO),(DISP-HI)	ADC REG8/MEM8,REG8
11	0001 0001	MOD REG R/M	(DISP-LO),(DISP-HI)	ADC REG16/MEM16,REG16
12	0001 0010	MOD REG R/M	(DISP-LO),(DISP-HI)	ADC REG8,REG8/MEM8
13	0001 0011	MOD REG R/M	(DISP-LO),(DISP-HI)	ADC REG16,REG16/MEM16
14	0001 0100	DATA-8		ADC AL,IMMED8
15	0001 0101	DATA-LO	DATA-HI	ADC AX,IMMED16
16	0001 0110			PUSH SS
17	0001 0111			POP SS
18	0001 1000	MOD REG R/M	(DISP-LO),(DISP-HI)	SBB REG8/MEM8,REG8
19	0001 1001	MOD REG R/M	(DISP-LO),(DISP-HI)	SBB REG16/MEM16,REG16
1A	0001 1010	MOD REG R/M	(DISP-LO),(DISP-HI)	SBB REG8,REG8/MEM8
1B	0001 1011	MOD REG R/M	(DISP-LO),(DISP-HI)	SBB REG16,REG16/MEM16
1C	0001 1100	DATA-8		SBB AL,IMMED8
1D	0001 1101	DATA-LO	DATA-HI	SBB AX,IMMED16
1E	0001 1110			PUSH DS
1F	0001 1111			POP DS
20	0010 0000	MOD REG R/M	(DISP-LO),(DISP-HI)	AND REG8/MEM8,REG8
21	0010 0001	MOD REG R/M	(DISP-LO),(DISP-HI)	AND REG16/MEM16,REG16
22	0010 0010	MOD REG R/M	(DISP-LO),(DISP-HI)	AND REG8,REG8/MEM8
23	0010 0011	MOD REG R/M	(DISP-LO),(DISP-HI)	AND REG16,REG16/MEM16
24	0010 0100	DATA-8		AND AL,IMMED8
25	0010 0101	DATA-LO	DATA-HI	AND AX,IMMED16
26	0010 0110			ES: (segment override prefix)
27	0010 0111			DAA
28	0010 1000	MOD REG R/M	(DISP-LO),(DISP-HI)	SUB REG8/MEM8,REG8
29	0010 1001	MOD REG R/M	(DISP-LO),(DISP-HI)	SUB REG16/MEM16,REG16
2A	0010 1010	MOD REG R/M	(DISP-LO),(DISP-HI)	SUB REG8,REG8/MEM8
2B	0010 1011	MOD REG R/M	(DISP-LO,(DISP-HI)	SUB REG16,REG16/MEM16
2C	0010 1100	DATA-8		SUB AL,IMMED8
2D	0010 1101	DATA-LO	DATA-HI	SUB AX,IMMED16
2E	0010 1110			CS: (segment override prefix)
2F	0010 1111			DAS
30	0011 0000	MOD REG R/M	(DISP-LO),(DISP-HI)	XOR REG8/MEM8,REG8
31	0011 0001	MOD REG R/M	(DISP-LO),(DISP-HI)	XOR REG16/MEM16,REG16
32	0011 0010	MOD REG R/M	(DISP-LO),(DISP-HI)	XOR REG8,REG8/MEM8
33	0011 0011	MOD REG R/M	(DISP-LO),(DISP-HI)	XOR REG16,REG16/MEM16
34	0011 0100	DATA-8		XOR AL,IMMED8
35	0011 0101	DATA-LO	DATA-HI	XOR AX,IMMED16
36	0011 0110			SS: (segment override prefix)

Table 1-23 Machine Instruction Decoding Guide (continued)

1ST BYTE HEX	1ST BYTE BINARY	2ND BYTE	BYTES 3,4,5,6	ASM-86 INSTRUCTION FORMAT
37	0011 0110			AAA
38	0011 1000	MOD REG R/M	(DISP-LO),(DISP-HI)	CMP REG8/MEM8,REG8
39	0011 1001	MOD REG R/M	(DISP-LO),(DISP-HI)	CMP REG16/MEM16,REG16
3A	0011 1010	MOD REG R/M	(DISP-LO),(DISP-HI)	CMP REG8,REG8/MEM8
3B	0011 1011	MOD REG R/M	(DISP-LO),(DISP-HI)	CMP REG16,REG16/MEM16
3C	0011 1100	DATA-8		CMP AL,IMMED8
3D	0011 1101	DATA-LO	DATA-HI	CMP AX,IMMED16
3E	0011 1110			DS: (segment override prefix)
3F	0011 1111			AAS
40	0100 0000			INC AX
41	0100 0001			INC CX
42	0100 0010			INC DX
43	0100 0011			INC BX
44	0100 0100			INC SP
45	0100 0101			INC BP
46	0100 0110			INC SI
47	0100 0111			INC DI
48	0100 1000			DEC AX
49	0100 1001			DEC CX
4A	0100 1010			DEC DX
4B	0100 1011			DEC BX
4C	0100 1100			DEC SP
4D	0100 1101			DEC BP
4E	0100 1110			DEC SI
4F	0100 1111			DEC DI
50	0101 0000			PUSH AX
51	0101 0001			PUSH CX
52	0101 0010			PUSH DX
53	0101 0011			PUSH BX
54	0101 0100			PUSH SP
55	0101 0101			PUSH BP
56	0101 0110			PUSH SI
57	0101 0111			PUSH DI
58	0101 1000			POP AX
59	0101 1001			POP CX
5A	0101 1010			POP DX
5B	0101 1011			POP BX
5C	0101 1100			POP SP
5D	0101 1101			POP BP
5E	0101 1110			POP SI
5F	0101 1111			POP DI
60	0110 0000			(not used)
61	0110 0001			(not used)
62	0110 0010			(not used)
63	0110 0011			(not used)
64	0110 0100			(not used)
65	0110 0101			(not used)
66	0110 0110			(not used)
67	0110 0111			(not used)

8086/8088 CPU

Table 1-23 Machine Instruction Decoding Guide (continued)

1ST BYTE HEX	1ST BYTE BINARY	2ND BYTE	BYTES 3,4,5,6	ASM-86 INSTRUCTION FORMAT
68	0110 1000			(not used)
69	0110 1001			(not used)
6A	0110 1010			(not used)
6B	0110 1011			(not used)
6C	0110 1100			(not used)
6D	0110 1101			(not used)
6E	0110 1110			(not used)
6F	0110 1111			(not used)
70	0111 0000	IP-INC8		JO SHORT-LABEL
71	0111 0001	IP-INC8		JNO SHORT-LABEL
72	0111 0010	IP-INC8		JB/JNAE/JC SHORT-LABEL
73	0111 0011	IP-INC8		JNB/JAE/JNC SHORT-LABEL
74	0111 0100	IP-INC8		JE/JZ SHORT-LABEL
75	0111 0101	IP-INC8		JNE/JNZ SHORT-LABEL
76	0111 0110	IP-INC8		JBE/JNA SHORT-LABEL
77	0111 0111	IP-INC8		JNBE/JA SHORT-LABEL
78	0111 1000	IP-INC8		JS SHORT-LABEL
79	0111 1001	IP-INC8		JNS SHORT-LABEL
7A	0111 1010	IP-INC8		JP/JPE SHORT-LABEL
7B	0111 1011	IP-INC8		JNP/JPO SHORT-LABEL
7C	0111 1100	IP-INC8		JL/JNGE SHORT-LABEL
7D	0111 1101	IP-INC8		JNL/JGE SHORT-LABEL
7E	0111 1110	IP-INC8		JLE/JNG SHORT-LABEL
7F	0111 1111	IP-INC8		JNLE/JG SHORT-LABEL
80	1000 0000	MOD 000 R/M	(DISP-LO),(DISP-HI),DATA-8	ADD REG8/MEM8,IMMED8
80	1000 0000	MOD 001 R/M	(DISP-LO),(DISP-HI),DATA-8	OR REG8/MEM8,IMMED8
80	1000 0000	MOD 010 R/M	(DISP-LO),(DISP-HI),DATA-8	ADC REG8/MEM8,IMMED8
80	1000 0000	MOD 011 R/M	(DISP-LO),(DISP-HI),DATA-8	SBB REG8/MEM8,IMMED8
80	1000 0000	MOD 100 R/M	(DISP-LO),(DISP-HI),DATA-8	AND REG8/MEM8,IMMED8
80	1000 0000	MOD 101 R/M	(DISP-LO),(DISP-HI),DATA-8	SUB REG8/MEM8,IMMED8
80	1000 0000	MOD 110 R/M	(DISP-LO),(DISP-HI),DATA-8	XOR REG8/MEM8,IMMED8
80	1000 0000	MOD 111 R/M	(DISP-LO),(DISP-HI),DATA-8	CMP REG8/MEM8,IMMED8
81	1000 0001	MOD 000 R/M	(DISP-LO),(DISP-HI),DATA-LO,DATA-HI	ADD REG16/MEM16,IMMED16
81	1000 0001	MOD 001 R/M	(DISP-LO),(DISP-HI),DATA-LO,DATA-HI	OR REG16/MEM16,IMMED16
81	1000 0001	MOD 010 R/M	(DISP-LO),(DISP-HI),DATA-LO,DATA-HI	ADC REG16/MEM16,IMMED16
81	1000 0001	MOD 011 R/M	(DISP-LO),(DISP-HI),DATA-LO,DATA-HI	SBB REG16/MEM16,IMMED16

Table 1-23 Machine Instruction Decoding Guide (continued)

1ST BYTE HEX	1ST BYTE BINARY	2ND BYTE	BYTES 3,4,5,6	ASM-86 INSTRUCTION FORMAT
81	1000 0001	MOD 100 R/M	(DISP-LO),(DISP-HI), DATA-LO,DATA-HI	AND REG16/MEM16,IMMED16
81	1000 0001	MOD 101 R/M	(DISP-LO),(DISP-HI), DATA-LO,DATA-HI	SUB REG16/MEM16,IMMED16
81	1000 0001	MOD 110 R/M	(DISP-LO),(DISP-HI), DATA-LO,DATA-HI	XOR REG16/MEM16,IMMED16
81	1000 0001	MOD 111 R/M	(DISP-LO),(DISP-HI), DATA-LO,DATA-HI	CMP REG16/MEM16,IMMED16
82	1000 0010	MOD 000 R/M	(DISP-LO),(DISP-HI), DATA-8	ADD REG8/MEM8,IMMED8
82	1000 0010	MOD 001 R/M		(not used)
82	1000 0010	MOD 010 R/M	(DISP-LO),(DISP-HI), DATA-8	ADC REG8/MEM8,IMMED8
82	1000 0010	MOD 011 R/M	(DISP-LO),(DISP-HI), DATA-8	SBB REG8/MEM8,IMMED8
82	1000 0010	MOD 100 R/M		(not used)
82	1000 0010	MOD 101 R/M	(DISP-LO),(DISP-HI), DATA-8	SUB REG8/MEM8,IMMED8
82	1000 0010	MOD 110 R/M		(not used)
82	1000 0010	MOD 111 R/M	(DISP-LO),(DISP-HI), DATA-8	CMP REG8/MEM8,IMMED8
83	1000 0011	MOD 000 R/M	(DISP-LO),(DISP-HI), DATA-SX	ADD REG16/MEM16, IMMED8
83	1000 0011	MOD 001 R/M		(not used)
83	1000 0011	MOD 010 R/M	(DISP-LO), (DISP-HI), DATA-SX	ADC REG16/MEM16,IMMED8
83	1000 0011	MOD 011 R/M	(DISP-LO),(DISP-HI), DATA-SX	SBB REG16/MEM16,IMMED8
83	1000 0011	MOD 100 R/M		(not used)
83	1000 0011	MOD 101 R/M	(DISP-LO),(DISP-HI), DATA-SX	SUB REG16/MEM16,IMMED8
83	1000 0011	MOD 110 R/M		(not used)
83	1000 0011	MOD 111 R/M	(DISP-LO),(DISP-HI), DATA-SX	CMP REG16/MEM16,IMMED8
84	1000 0100	MOD REG R/M	(DISP-LO),(DISP-HI)	TEST REG8/MEM8,REG8
85	1000 0101	MOD REG R/M	(DISP-LO),(DISP-HI)	TEST REG16/MEM16,REG16
86	1000 0110	MOD REG R/M	(DISP-LO),(DISP-HI)	XCHG REG8,REG8/MEM8
87	1000 0111	MOD REG R/M	(DISP-LO),(DISP-HI)	XCHG REG16,REG16/MEM16
88	1000 1000	MOD REG R/M	(DISP-LO),(DISP-HI),	MOV REG8/MEM8,REG8
89	1000 1001	MOD REG R/M	(DISP-LO),(DISP-HI)	MOV REG16/MEM16/REG16
8A	1000 1010	MOD REG R/M	(DISP-LO),(DISP-HI)	MOV REG8,REG8/MEM8
8B	1000 1011	MOD REG R/M	(DISP-LO),(DISP-HI)	MOV REG16,REG16/MEM16
8C	1000 1100	MOD 0SR R/M	(DISP-LO),(DISP-HI)	MOV REG16/MEM16,SEGREG
8C	1000 1100	MOD 1-- R/M		(not used)
8D	1000 1101	MOD REG R/M	(DISP-LO),(DISP-HI)	LEA REG16,MEM16
8E	1000 1110	MOD 0SR R/M	(DISP-LO),(DISP-HI)	MOV SEGREG,REG16/MEM16
8E,	1000 1110	MOD 1-- R/M		(not used)
8F	1000 1111	MOD 000 R/M	(DISP-LO),(DISP-HI),	POP REG16/MEM16
8F	1000 1111	MOD 001 R/M		(not used)
8F	1000 1111	MOD 010 R/M		(not used)

Table 1-23 Machine Instruction Decoding Guide (continued)

1ST BYTE HEX	1ST BYTE BINARY	2ND BYTE	BYTES 3,4,5,6	ASM-86 INSTRUCTION FORMAT
8F	1000 1111	MOD 011 R/M		(not used)
8F	1000 1111	MOD 100 R/M		(not used)
8F	1000 1111	MOD 101 R/M		(not used)
8F	1000 1111	MOD 110 R/M		(not used)
8F	1000 1111	MOD 111 R/M		(not used)
90	1001 0000			NOP (exchange AX,AX)
91	1001 0001			XCHG AX,CX
92	1001 0010			XCHG AX,DX
93	1001 0011			XCHG AX,BX
94	1001 0100			XCHG AX,SP
95	1001 0101			XCHG AX,BP
96	1001 0110			XCHG AX,SI
97	1001 0111			XCHG AX,DI
98	1001 1000			CBW
99	1001 1001			CWD
9A	1001 1010	DISP-LO	DISP-HI,SEG-LO, SEG-HI	CALL FAR_PROC
9B	1001 1011			WAIT
9C	1001 1100			PUSHF
9D	1001 1101			POPF
9E	1001 1110			SAHF
9F	1001 1111			LAHF
A0	1010 0000	ADDR-LO	ADDR-HI	MOV AL,MEM8
A1	1010 0001	ADDR-LO	ADDR-HI	MOV AX,MEM16
A2	1010 0010	ADDR-LO	ADDR-HI	MOV MEM8,AL
A3	1010 0011	ADDR-LO	ADDR-HI	MOV MEM16,AL
A4	1010 0100			MOVS DEST-STR8,SRC-STR8
A5	1010 0101			MOVS DEST-STR16,SRC-STR16
A6	1010 0110			CMPS DEST-STR8,SRC-STR8
A7	1010 0111			CMPS DEST-STR16,SRC-STR16
A8	1010 1000	DATA-8		TEST AL,IMMED8
A9	1010 1001	DATA-LO	DATA-HI	TEST AX,IMMED16
AA	1010 1010			STOS DEST-STR8
AB	1010 1011			STOS DEST-STR16
AC	1010 1100			LODS SRC-STR8
AD	1010 1101			LODS SRC-STR16
AE	1010 1110			SCAS DEST-STR8
AF	1010 1111			SCAS DEST-STR16
B0	1011 0000	DATA-8		MOV AL,IMMED8
B1	1011 0001	DATA-8		MOV CL,IMMED8
B2	1011 0010	DATA-8		MOV DL,IMMED8
B3	1011 1011	DATA-8		MOV BL,IMMED8
B4	1011 0100	DATA-8		MOV AH,IMMED8
B5	1011 0101	DATA-8		MOV CH,IMMED8
B6	1011 0110	DATA-8		MOV DH,IMMED8
B7	1011 0111	DATA-8		MOV BH,IMMED8
B8	1011 1000	DATA-LO	DATA-HI	MOV AX,IMMED16
B9	1011 1001	DATA-LO	DATA-HI	MOV CX,IMMED16
BA	1011 1010	DATA-LO	DATA-HI	MOV DX,IMMED16
BB	1011 1011	DATA-LO	DATA-HI	MOV BX,IMMED16

Table 1-23 Machine Instruction Decoding Guide (continued)

1ST BYTE HEX	1ST BYTE BINARY	2ND BYTE	BYTES 3,4,5,6	ASM-86 INSTRUCTION FORMAT
BC	1011 1100	DATA-LO	DATA-HI	MOV SP,IMMED16
BD	1011 1101	DATA-LO	DATA-HI	MOV BP,IMMED16
BE	1011 1110	DATA-LO	DATA-HI	MOV SI,IMMED16
BF	1011 1111	DATA-LO	DATA-HI	MOV DI,IMMED16
C0	1100 0000			(not used)
C1	1100 0001			(not used)
C2	1100 0010	DATA-LO	DATA-HI	RET IMMED16 (intraseg)
C3	1100 0011			RET (intrasegment)
C4	1100 0100	MOD REG R/M	(DISP-LO),(DISP-HI)	LES REG16,MEM16
C5	1100 0101	MOD REG R/M	(DISP-LO),(DISP-HI)	LDS REG16,MEM16
C6	1100 0110	MOD 000 R/M	(DISP-LO),(DISP-HI), DATA-8	MOV MEM8,IMMED8
C6	1100 0110	MOD 001 R/M		(not used)
C6	1100 0110	MOD 010 R/M		(not used)
C6	1100 0110	MOD 011 R/M		(not used)
C6	1100 0110	MOD 100 R/M		(not used)
C6	1100 0110	MOD 101 R/M		(not used)
C6	1100 0110	MOD 110 R/M		(not used)
C6	1100 0110	MOD 111 R/M		(not used)
C7	1100 0111	MOD 000 R/M	(DISP-LO),(DISP-HI), DATA-LO,DATA-HI	MOV MEM16,IMMED16
C7	1100 0111	MOD 001 R/M		(not used)
C7	1100 0111	MOD 010 R/M		(not used)
C7	1100 0111	MOD 011 R/M		(not used)
C7	1100 0111	MOD 100 R/M		(not used)
C7	1100 0111	MOD 101 R/M		(not used)
C7	1100 0111	MOD 110 R/M		(not used)
C7	1100 0111	MOD 111 R/M		(not used)
C8	1100 1000			(not used)
C9	1100 1001			(not used)
CA	1100 1010	DATA-LO	DATA-HI	RET IMMED16 (intersegment)
CB	1100 1011			RET (intersegment)
CC	1100 1100			INT 3
CD	1100 1101	DATA-8		INT IMMED8
CE	1100 1110			INTO
CF	1100 1111			IRET
D0	1101 0000	MOD 000 R/M	(DISP-LO),(DISP-HI)	ROL REG8/MEM8,1
D0	1101 0000	MOD 001 R/M	(DISP-LO),(DISP-HI)	ROR REG8/MEM8,1
D0	1101 0000	MOD 010 R/M	(DISP-LO),(DISP-HI)	RCL REG8/MEM8,1
D0	1101 0000	MOD 011 R/M	(DISP-LO),(DISP-HI)	RCR REG8/MEM8,1
D0	1101 0000	MOD 100 R/M	(DISP-LO),(DISP-HI)	SAL/SHL REG8/MEM8,1
D0	1101 0000	MOD 101 R/M	(DISP-LO),(DISP-HI)	SHR REG8/MEM8,1
D0	1101 0000	MOD 110 R/M		(not used)
D0	1101 0000	MOD 111 R/M	(DISP-LO),(DISP-HI)	SAR REG8/MEM8,1
D1	1101 0001	MOD 000 R/M	(DISP-LO),(DISP-HI)	ROL REG16/MEM16,1
D1	1101 0001	MOD 001 R/M	(DISP-LO),(DISP-HI)	ROR REG16/MEM16,1
D1	1101 0001	MOD 010 R/M	(DISP-LO),(DISP-HI)	RCL REG16/MEM16,1
D1	1101 0001	MOD 011 R/M	(DISP-LO),(DISP-HI)	RCR REG16/MEM16,1
D1	1101 0001	MOD 100 R/M	(DISP-LO),(DISP-HI)	SAL/SHL REG16/MEM16,1

8086/8088 CPU

Table 1-23 Machine Instruction Decoding Guide (continued)

1ST BYTE HEX	1ST BYTE BINARY	2ND BYTE	BYTES 3,4,5,6	ASM-86 INSTRUCTION FORMAT
D1	1101 0001	MOD 101 R/M	(DISP-LO),(DISP-HI)	SHR REG16/MEM16,1
D1	1101 0001	MOD 110 R/M		(not used)
D1	1101 0001	MOD 111 R/M	(DISP-LO),(DISP-HI)	SAR REG16/MEM16,1
D2	1101 0010	MOD 000 R/M	(DISP-LO),(DISP-HI)	ROL REG8/MEM8,CL
D2	1101 0010	MOD 001 R/M	(DISP-LO),(DISP-HI)	ROR REG8/MEM8,CL
D2	1101 0010	MOD 010 R/M	(DISP-LO),(DISP-HI)	RCL REG8/MEM8,CL
D2	1101 0010	MOD 011 R/M	(DISP-LO),(DISP-HI)	RCR REG8/MEM8,CL
D2	1101 0010	MOD 100 R/M	(DISP-LO),(DISP-HI)	SAL/SHL REG8/MEM8,CL
D2	1101 0010	MOD 101 R/M	(DISP-LO),(DISP-HI)	SHR REG8/MEM8,CL
D2	1101 0010	MOD 110 R/M		(not used)
D2	1101 0010	MOD 111 R/M	(DISP-LO),(DISP-HI)	SAR REG8/MEM8,CL
D3	1101 0011	MOD 000 R/M	(DISP-LO),(DISP-HI)	ROL REG16/MEM16,CL
D3	1101 0011	MOD 001 R/M	(DISP-LO),(DISP-HI)	ROR REG16/MEM16,CL
D3	1101 0011	MOD 010 R/M	(DISP-LO),(DISP-HI)	RCL REG16/MEM16,CL
D3	1101 0011	MOD 011 R/M	(DISP-LO),(DISP-HI)	RCR REG16/MEM16,CL
D3	1101 0011	MOD 100 R/M	(DISP-LO),(DISP-HI)	SAL/SHL REG16/MEM16,CL
D3	1101 0011	MOD 101 R/M	(DISP-LO),(DISP-HI)	SHR REG16/MEM16,CL
D3	1101 0011	MOD 110 R/M		(not used)
D3	1101 0011	MOD 111 R/M	(DISP-LO),(DISP-HI)	SAR REG16/MEM16,CL
D4	1101 0100	00001010		AAM
D5	1101 0101	00001010		AAD
D6	1101 0110			(not used)
D7	1101 0111			XLAT SOURCE-TABLE
D8	1101 1000	MOD 000 R/M		
	1XXX	MOD YYY R/M	(DISP-LO), (DISP-HI)	ESC OPCODE,SOURCE
DF	1101 1111	MOD 111 R/M		
E0	1110 0000	IP-INC-8		LOOPNE/ LOOPNZ SHORT-LABEL
E1	1110 0001	IP-INC-8		LOOPE/ LOOPZ SHORT-LABEL
E2	1110 0010	IP-INC-8		LOOP SHORT-LABEL
E3	1110 0011	IP-INC-8		JCXZ SHORT-LABEL
E4	1110 0100	DATA-8		IN AL,IMMED8
E5	1110 0101	DATA-8		IN AX,IMMED8
E6	1110 0110	DATA-8		OUT AL,IMMED8
E7	1110 0111	DATA-8		OUT AX,IMMED8
E8	1110 1000	IP-INC-LO	IP-INC-HI	CALL NEAR-PROC
E9	1110 1001	IP-INC-LO	IP-INC-HI	JMP NEAR-LABEL
EA	1110 1010	IP-LO	IP-HI,CS-LO,CS-HI	JMP FAR-LABEL
EB	1110 1011	IP-INC8		JMP SHORT-LABEL
EC	1110 1100			IN AL,DX
ED	1110 1101			IN AX,DX
EE	1110 1110			OUT AL,DX
EF	1110 1111			OUT AX,DX
F0	1111 0000			LOCK (prefix)
F1	1111 0001			(not used)
F2	1111 0010			REPNE/REPNZ
F3	1111 0011			REP/REPE/REPZ
F4	1111 0100			HLT
F5	1111 0101			CMC

Table 1-23 Machine Instruction Decoding Guide (continued)

1ST BYTE HEX	1ST BYTE BINARY	2ND BYTE	BYTES 3,4,5,6	ASM-86 INSTRUCTION FORMAT
F6	1111 0110	MOD 000 R/M	(DISP-LO),(DISP-HI), DATA-8	TEST REG8/MEM8,IMMED8
F6	1111 0110	MOD 001 R/M		(not used)
F6	1111 0110	MOD 010 R/M	(DISP-LO),(DISP-HI)	NOT REG8/MEM8
F6	1111 0110	MOD 011 R/M	(DISP-LO),(DISP-HI)	NEG REG8/MEM8
F6	1111 0110	MOD 100 R/M	(DISP-LO),(DISP-HI)	MUL REG8/MEM8
F6	1111 0110	MOD 101 R/M	(DISP-LO),(DISP-HI)	IMUL REG8/MEM8
F6	1111 0110	MOD 110 R/M	(DISP-LO),(DISP-HI)	DIV REG8/MEM8
F6	1111 0110	MOD 111 R/M	(DISP-LO),(DISP-HI)	IDIV REG8/MEM8
F7	1111 0111	MOD 000 R/M	(DISP-LO),(DISP-HI), DATA-LO,DATA-HI	TEST REG16/MEM16,IMMED16
F7	1111 0111	MOD 001 R/M		(not used)
F7	1111 0111	MOD 010 R/M	(DISP-LO),(DISP-HI)	NOT REG16/MEM16
F7	1111 0111	MOD 011 R/M	(DISP-LO),(DISP-HI)	NEG REG16/MEM16
F7	1111 0111	MOD 100 R/M	(DISP-LO),(DISP-HI)	MUL REG16/MEM16
F7	1111 0111	MOD 101 R/M	(DISP-LO),(DISP-HI)	IMUL REG16/MEM16
F7	1111 0111	MOD 110 R/M	(DISP-LO),(DISP-HI)	DIV REG16/MEM16
F7	1111 0111	MOD 111 R/M	(DISP-LO),(DISP-HI)	IDIV REG16/MEM16
F8	1111 1000			CLC
F9	1111 1001			STC
FA	1111 1010			CLI
FB	1111 1011			STI
FC	1111 1100			CLD
FD	1111 1101			STD
FE	1111 1110	MOD 000 R/M	(DISP-LO),(DISP-HI)	INC REG8/MEM8
FE	1111 1110	MOD 001 R/M	(DISP-LO),(DISP-HI)	DEC REG8/MEM8
FE	1111 1110	MOD 010 R/M		(not used)
FE	1111 1110	MOD 011 R/M		(not used)
FE	1111 1110	MOD 100 R/M		(not used)
FE	1111 1110	MOD 101 R/M		(not used)
FE	1111 1110	MOD 110 R/M		(not used)
FE	1111 1110	MOD 111 R/M		(not used)
FF	1111 1111	MOD 000 R/M	(DISP-LO),(DISP-HI)	INC MEM16
FF	1111 1111	MOD 001 R/M	(DISP-LO),(DISP-HI)	DEC MEM16
FF	1111 1111	MOD 010 R/M	(DISP-LO),(DISP-HI)	CALL REG16/MEM16 (intra)
FF	1111 1111	MOD 011 R/M	(DISP-LO),(DISP-HI)	CALL MEM16 (intersegment)
FF	1111 1111	MOD 100 R/M	(DISP-LO),(DISP-HI)	JMP REG16/MEM16 (intra)
FF	1111 1111	MOD 101 R/M	(DISP-LO),(DISP-HI)	JMP MEM16 (intersegment)
FF	1111 1111	MOD 110 R/M	(DISP-LO),(DISP-HI)	PUSH MEM16
FF	1111 1111	MOD 111 R/M		(not used)

Table 1-24 8086/8088 Device Pin Descriptions

The following pin function descriptions are for iAPX 86 systems in either minimum or maximum mode. The "Local Bus" in these descriptions is the direct multiplexed bus interface connection to the 8086 (without regard to additional bus buffers).

Symbol	Pin No.	Type	Name and Function
AD_{15}-AD_0	2-16, 39	I/O	**Address Data Bus:** These lines constitute the time multiplexed memory/I/O address (T_1) and data (T_2, T_3, T_W, T_4) bus. A_0 is analogous to $\overline{BHE}$ for the lower byte of the data bus, pins D_7-D_0. It is LOW during T_1 when a byte is to be transferred on the lower portion of the bus in memory or I/O operations. Eight-bit oriented devices tied to the lower half would normally use A_0 to condition chip select functions. (See $\overline{BHE}$.) These lines are active HIGH and float to 3-state OFF during interrupt acknowledge and local bus "hold acknowledge."
A_{19}/S_6, A_{18}/S_5, A_{17}/S_4, A_{16}/S_3	35-38	O	**Address/Status:** During T_1 these are the four most significant address lines for memory operations. During I/O operations these lines are LOW. During memory and I/O operations, status information is available on these lines during T_2, T_3, T_W, and T_4. The status of the interrupt enable FLAG bit (S_5) is updated at the beginning of each CLK cycle. A_{17}/S_4 and A_{16}/S_3 are encoded as shown. This information indicates which relocation register is presently being used for data accessing. These lines float to 3-state OFF during local bus "hold acknowledge."
$\overline{BHE}/S_7$	34	O	**Bus High Enable/Status:** During T_1 the bus high enable signal ($\overline{BHE}$) should be used to enable data onto the most significant half of the data bus, pins D_{15}-D_8. Eight-bit oriented devices tied to the upper half of the bus would normally use $\overline{BHE}$ to condition chip select functions. $\overline{BHE}$ is LOW during T_1 for read, write, and interrupt acknowledge cycles when a byte is to be transferred on the high portion of the bus. The S_7 status information is available during T_2, T_3, and T_4. The signal is active LOW, and floats to 3-state OFF in "hold." It is LOW during T_1 for the first interrupt acknowledge cycle.
$\overline{RD}$	32	O	**Read:** Read strobe indicates that the processor is performing a memory of I/O read cycle, depending on the state of the S_2 pin. This signal is used to read devices which reside on the 8086 local bus. $\overline{RD}$ is active LOW during T_2, T_3 and T_W of any read cycle, and is guaranteed to remain HIGH in T_2 until the 8086 local bus has floated. This signal floats to 3-state OFF in "hold acknowledge."
READY	22	I	**READY:** is the acknowledgement from the addressed memory or I/O device that it will complete the data transfer. The READY signal from memory/IO is synchronized by the 8284A Clock Generator to form READY. This signal is active HIGH. The 8086 READY input is not synchronized. Correct operation is not guaranteed if the setup and hold times are not met.
INTR	18	I	**Interrupt Request:** is a level triggered input which is sampled during the last clock cycle of each instruction to determine if the processor should enter into an interrupt acknowledge operation. A subroutine is vectored to via an interrupt vector lookup table located in system memory. It can be internally masked by software resetting the interrupt enable bit. INTR is internally synchronized. This signal is active HIGH.
TEST	23	I	**TEST:** input is examined by the "Wait" instruction. If the TEST input is LOW execution continues, otherwise the processor waits in an "idle" state. This input is synchronized internally during each clock cycle on the leading edge of CLK.

A_{19}/S_6	A_{16}/S_3	Characteristics
0 (LOW)	0	Alternate Data
0	1	Stack
1 (HIGH)	0	Code or None
1	1	Data
A_{18} to 0 (LOW)		0

$\overline{BHE}$	A_0	Characteristics
0	0	Whole word
0	1	Upper byte from to odd address
1	0	Lower byte from to even address
1	1	None

A P E N D I C E B

EJEMPLO DE LA CARACTERIZACION DE UN SISTEMA CON EL ANALIZADOR DIGITAL DE RESPUESTA EN FRECUENCIA { ADRF }

(CIRCUITO ANALOGICO en este caso)

A manera de ejemplo para mostrar el funcionamiento y tipo de despliegues del **Analizador Digital de Respuesta en Frecuencia**, se puede conectar entre la terminal que proporciona la señal senoidal de prueba (→ a la entrada del sistema bajo análisis) y la terminal que en el ADRF recibe la señal analógica de salida del sistema bajo análisis, el circuito propuesto a continuación, que puede emplearse como una de las bandas elementales de un ecualizador gráfico de audio.

Mediante análisis convencional de redes eléctricas se demuestra que dicho circuito puede presentar una atenuación o bien una amplificación de entre 11 y 12 deciBeles a una frecuencia cercana a los 5000 Hz (debido a la tolerancia en exactitud inherente a los componentes con que es construido, con respecto a los valores nominales). En este circuito la máxima amplificación se logra cuando la constante K en el potenciómetro es igual a cero (K=0) (constante posicional del cursor: $0 \leq K \leq 1$).

Con el ADRF, y seleccionando el { Rango (2) - Subrango (1) }, se fija una frecuencia aproximada para el barrido de análisis de { f1=971 a f2=23204.66 [Hz] }, obteniéndose los siguientes resultados para la Función de Transferencia { V_o/V_i } :

MAGNITUD LINEAL :
Valor máximo : ... frecuencia = 4976 [Hz]
Magnitud Lineal {V_o/V_i} = + 3.579068 [ ]
Magnitud Logaritmica {V_o/V_i} = + 11.075399 [dB]
Angulo de Fase {V_o/V_i} = -173.829819 [°]

ANGULO DE FASE :
{ 180 ° } : ... frecuencia = 5367 [Hz]
Magnitud Lineal {V_o/V_i} = + 3.526278 [ ]
Magnitud Logaritmica {V_o/V_i} = + 10.946330 [dB]
Angulo de Fase {V_o/V_i} = +180.000000 [°]

BANDAS DE PASO :

Valores a 3dB del valor de magnitud máxima = $V_{MAX} / \sqrt{2}$

* Banda de paso Inferior ⟶ frecuencia = 2908 [Hz]
  Magnitud Lineal $\{V_o/V_i\}$ = + 2.534942 [ ]
  Magnitud Logaritmica $\{V_o/V_i\}$ = + 8.079361 [dB]
  Angulo de Fase $\{V_o/V_i\}$ = -147.917801 [°]
* Banda de paso Superior ⟶ frecuencia = 8871 [Hz]
  Magnitud Lineal $\{V_o/V_i\}$ = + 2.523565 [ ]
  Magnitud Logaritmica $\{V_o/V_i\}$ = + 8.040292 [dB]
  Angulo de Fase $\{V_o/V_i\}$ = +154.411484 [°]

(Frecuencias reales de barrido : f1=1050 y f2=24236 [Hz])
(Circuito ajustado para máxima amplificación [cursor del potenciómetro en {K=0}]).

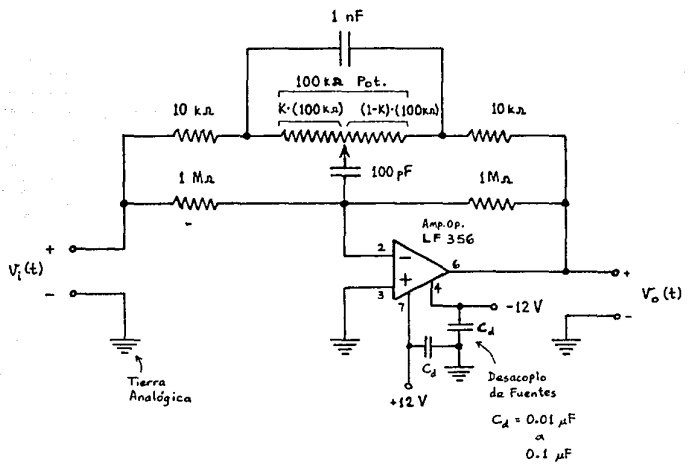
Los puntos cuyos resultados numéricos se muestran aquí, (al igual que cualquier otro punto que se hubiese deseado seleccionar dentro del rango de despliegue (barrido)), fueron seleccionados al posicionar adecuadamente el cursor gráfico en pantalla, mediante desplazamientos (adelante-atrás) controlados por el usuario desde el teclado de la microcomputadora PC (PS).

A continuación se muestra un diagrama del circuito electrónico bajo análisis con el ADRF y sus correspondientes curvas de respuesta en frecuencia como son desplegadas en la pantalla de la microcomputadora PC (PS en este caso). Estas curvas corresponden a la Función de Transferencia, es decir, a la razón del voltaje de salida con respecto al voltaje de entrada (V_o/V_i) del circuito bajo prueba.

Nota : Aquí se muestran las curvas (V_o/V_i) de Magnitud Lineal y de Angulo de Fase, pero también es posible desplegar la curva de Magnitud Logaritmica.

+

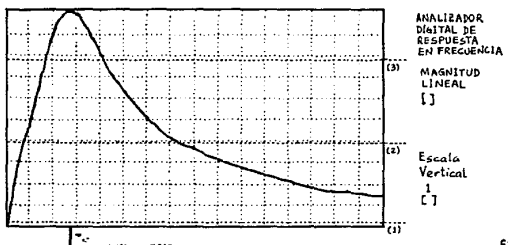
Circuito analizado con el ADRF.



Circuito que puede emplearse como una de las bandas elementales de un ecualizador gráfico de audio.

Despliegues, en pantalla de Microcomputadora PC (PS), de Respuesta en Frecuencia del circuito bajo análisis.

Despliegue: MAGNITUD · LINEAL [ ] razón : $\frac{Vo}{Vi}$



F1 = 971.00 [Hz] Mag.Lin. {Vo/Vi} = + 3.579068 [ ]
F2 = 23204.66 [Hz] Mag.Log. {Vo/Vi} = + 11.075399 [dB]
F = 4976 [Hz] Ang.Fase {Vo/Vi} = -173.829819 [°]
<F>

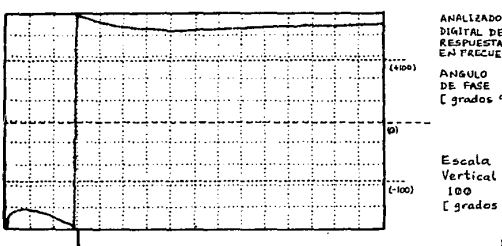
POSICION CURSOR: {1}:F1 . {2}:F2 . {>}:+10. {<}:-10. {+}:+1. {-}:-1. {Q}:↓

F1= Frec. Inicial aprox.
F2= Frec. Final aprox.
Ordenada Máx.: +3.579068
Ordenada Mín.: +0.961143

Una división principal en el eje vertical = 1 [ ]

Muestras ADC = 512

Despliegue: ANGULO · DE · FASE [°] (grados sexagesimales)



F1 = 971.00 [Hz] Mag.Lin. {Vo/Vi} = + 3.526278
F2 = 23204.66 [Hz] Mag.Log. {Vo/Vi} = + 10.946330
F = 5367 [Hz] Ang.Fase {Vo/Vi} = + 180.000000

F1= Frec. Inicial aprox.
F2= Frec. Final aprox.
Ordenada Máx.: +180.000000
Ordenada Mín.: -179.483841

Una división principal en el eje vertical = 100 [°]

Nota: (Se usó Microcomputadora IBM-PS/2 Modelo 50 Z → Resolución gráfica: EGA (640 x 350 pixels)).

BIBLIOGRAFIA

- Morgan Ch., Waite M.
  Introducción al Microprocesador 8086/8088
  McGraw Hill, 1984

- Triebel, Walter A.
  INTEL 8086 (μP)
  Prentice-Hall, 1985

- Osborne, Adam
  16 I.E. Sixteen-bit Microprocessor Handbook
  McGraw Hill, 1981

- Russell R., Alexy G.
  The 8086 Book (includes 8088)
  Osborne/McGraw Hill, Berkeley Ca., 1980

- Bradley, D.J.
  Assembly Language Programming for the IBM-PC
  Prentice-Hall, 1984

- Bik Chung Yeung
  8086/8088 Assembly Language Programming
  John Wiley & Sons, 1984

- Stone H.
  Microcomputer Interfacing
  Addison Wesley, 1982

- Hayes J.P.
  Digital System Design & Microprocessors
  McGraw Hill, 1985

- Guthikonda V.Rao
  Microprocessors and Microcomputer systems
  Van Nostrand Reinhold Company

* Hall D.V.
Microprocessors & Interfacing
McGraw Hill, 1986

* Eggebrecht L.C.
Interfacing to the IBM Personal Computer
Howard W.Sams & Co., 1983-87

* Duncan R.
Advanced MS DOS
Microsoft Press, 1986

* Kruglinsky D.
Guía a las comunicaciones del IBM-PC
McGraw Hill, 1984

* McNamara J.E.
Technical Aspects of Data Communication
2nd. Ed., DEC Press, 1982

* Serie Mundo electrónico
Interconexión de Periféricos a Microprocesadores
Marcombo, 1983

* Tocci R.J.
Sistemas Digitales (Principios y Aplicaciones)
Prentice-Hall, 1977-1986

* Goldsbrough P.F., Rony P.R.
Microcomputer Interfacing with the 8255 PPI Chip
Howard W. Sams & Co., 1980

* Lancaster D.
CMOS Cookbook
Howard W. Sams , 1986

* Lancaster D.
TTL Cookbook
Howard W. Sams , 1985

* Freedman A.
Glosario de Computación
McGraw Hill, 1983-84

* Ogata K.
Ingeniería de Control Moderna
Prentice-Hall, 1980

* Tobey, Graeme, Huelsman
Operational Amplifiers (Design & Applications)
McGraw Hill, 1981-85

* Huelsman, Allen
Introduction to the theory and design of active filters
McGraw Hill, 1980

* Desoer Ch., Kuh E.
Basic Circuit Theory
McGraw Hill, 1969-1984

* Fitzgerald, Higginbotham, Grabel
Fundamentos de Ingeniería Eléctrica
McGraw Hill, 1975-1982

* Gerez Greiser, Murray-Lasso
Teoría de Sistemas y Circuitos (Tomo 1)
Representaciones y Servicios de Ingeniería S.A.
México, 1972

* Arcila Rodríguez W., Vidal Macedo J.L.
Apuntes de Análisis de Circuitos Eléctricos (1a. parte)
DIME, Facultad de Ingeniería UNAM, 1985

* Hsu H.P.
Análisis de Fourier
Fondo Educativo Interamericano S.A., 1970

* Zarrella J.
Designing with the 8088 Microprocessor
Microcomputer Applications

- Carr J.J.
  Designing Microprocessor-based instrumentation
  Prentice-Hall, 1982

- Bibbero R.J.
  Microprocessors in Instruments and Control
  John Wiley & sons, 1977

- Lenk J.D.
  Handbook of Microcomputer based instrumentation and controls
  Prentice-Hall, 1984

- Gordon L.M.
  Process Automation/2 : Feedback Control Modes
  Chemical Engineering, Agosto,8,1983

- Deshpande P.B., Ash R.H.
  Elements of Computer Process Control with advanced Control Applications
  Instrument Society of America, 1981

- Creus A.
  Instrumentación Industrial
  Publicaciones Marcombo S.A., 1979

- Schindler Max
  Technology Reports :
  Electronic Design Magazine
  1) Computer-Aided Engineering (CAE) (November,15,1984) Part 1.
  2) CAE (December,13,1984) Part 2.

- Gunther S., Schoenberg V.E.
  CAE Workstation sets up direct connection to board design system
  Design Entry :
  Electronic Design Magazine (November,15,1984)

- Lee B., Ritzman D., Snapp W.
  Silicon Compiler teams with VLSI Workstation to customize CMOS ICs
  Design Entry :
  Electronic Design Magazine (November,15,1984)

M A N U A L E S :

* iAPX 86/88, 186/188 User's Manual
  Hardware Reference and Programmer's Reference
  INTEL Corp. , 1985 INTEL Corp. , 1983

* Microprocessor and Peripheral Handbook
  { Data sheets, application notes, article reprints and other design information }
  { Vol. 1 } : 8088, 8284A, 8259A
  { Vol. 2 } : 8254, 82C55A
  Order # : 230843
  INTEL Corp. , 1987
  Intel Literature sales
  P.O. Box 58130
  Santa Clara, Ca. 95052-8130, U.S.A

* Microcommunications Handbook
  (8251A)
  INTEL Corp. , 1986

* MOS Microprocessors and Peripherals
  Advanced Micro Devices Inc. , 1985

* The TTL Data Book for Design Engineers
  Texas Instruments Inc. , 1981

* Logic Databook
  { Vol. 1 } : CMOS, MM74HC, CD4xxx
  National Semiconductor Corp. , 1984

* Memory Databook (TMS 27C64)
  Texas Instruments Inc. , 1985

* Memory Databook
  Fujitsu Microelectronics , 1982

* EEPROM Memory Databook
  EXEL Microelectronics Inc. , 1987

* EXAR Databook (XR-2206)
(Linear, Control, Interface, Custom)
EXAR Corp. , 1985

* EXAR's Oscillators Products (Handbook)
EXAR Corp.

* Linear Databook
{ Vol. 1 } : Voltage Regulators, Operational Amplifiers, Buffers, Voltage Comparators, Instrum.Amp.
{ Vol. 2 } : Active filters, Analog Switches/Multiplexers, A/D converters, D/A converters, Sample&Hold, Voltage References
{ Vol. 3 } : - (Audio,Radio,Video,Motion,Special Func. IC's)
National Semiconductor Corp. , 1988

* Linear Applications Databook
National Semiconductor Corp. , 1986

* The Linear Control Circuits Databook for Engineers
Texas Instruments, 2nd. ed.

* Interface Databook (Digital Electronics & Communications)
National Semiconductor Corp.
(Application Notes)...(serial communication)

* Macro Assembler (Programmer's Guide)
Version 5.0/5.10
Microsoft Corp., 1987-88

* IBM Macro Assembler Reference & User's Guide
(Formats, Assembler program & support utilities)
(Microprocessor Instructions & assembler Directives)
Version 2.0
IBM Corp. , 1984

* Mixed-Language Programming Guide
Macro Assembler (5.0/5.10)
Microsoft Corp., 1987-88

- CodeView & Utilities
  (LINK, LIB, MAKE, EXEPACK, EXEMOD, SETENV, ERROUT)
  Microsoft Corp., 1987-88

- GWBASIC (User's Guide & User's Reference)
  Version 3.22
  Microsoft Corp., 1986-87

- IBM-PC BASIC COMPILER
  Version 1.0
  IBM Corp. , 1984
  { Aunque se usó Microsoft BASIC Compiler
  (Quick Basic Version 4.00), 1987 }

- Disk Operating System (DOS)
  Version 3.30
  IBM Corp., 1987