UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

FUNDAMENTOS PARA EL ANALISIS Y DISEÑO DE SISTEMAS EN TIEMPO REAL

T E S I S

QUE PARA OBTENER LOS TITULOS DE:

Ingeniero Mecánico Electricista

P R E S E N T A

Carlos Enrique Vizcaino Sahagun

y de: Ingeniero en Computación

P R E S E N T A

Agustín David Pérez Meseguer

Director de Tesis: Ing. José Luis Rodríguez Arauz

MEXICO, D. F. 1985

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

I N D I C E

C A P I T U L O I

G E N E R A L I D A D E S

Desde hace mucho tiempo la automatización ha sido un campo que ha ocupado la atención de la investigación tecnológica y científica, lo cual se debe a muchos factores. Entre estos se encuentra principalmente el hecho de que el desmedido aumento de la población ha demandado un gran incremento en la productividad industrial y una mayor eficiencia en la administración. Cada uno de estos sistemas requiere una metodología distinta para conseguir sus objetivos; por ejemplo, los sistemas industriales aparte de aumentar la producción deben asegurar la calidad del producto, la duración (cuidado) de su maquinaria e instrumental y desde luego, la seguridad del personal que labore en la planta. Por otro lado, la inmensa cantidad de información que deben almacenar y procesar los sistemas administrativos, ha exigido métodos para aumentar la velocidad de procesamiento y evitar el error humano. En cualquier sistema se trata de eliminar (o minimizar) el error que pueden inducir el descuido y el tedio que produce al trabajador las rutinas monótonas de trabajo.

La ingeniería ha dedicado grandes esfuerzos para conseguir los objetivos mencionados y ahora se ve apoyada con el gran desarrollo tecnológico que ha permitido disponer de microcomputadoras cada vez más completas y componentes electrónicos con propósitos específicos, generales y a precios relativamente bajos; lo cual ha aligerado la tarea de implementar los esquemas (métodos) que permiten alcanzar las metas. Sin embargo, aun no esta completamente resuelto este problema (resolver en el sentido de conseguir objetivos) dado que la solución depende del conocimiento del sistema, de los factores (entes materiales y energéticos, medio ambiente, etc.) que intervienen en él, y de los objetivos propuestos. Aun más, una vez que la solución se ha

encontrado se debe investigar la posibilidad de implementarla (realización), lo cual es otra limitante al campo de la automatización. De aquí está claro entonces que no se puede generalizar la metodología de diseño de esquemas automáticos dado que cada sistema exigirá el suyo propio.

Este trabajo presenta las estructuras y consideraciones básicas para iniciar el proceso de diseño de un esquema de control automático basado en computadora digital, a los cuales se les llama "sistemas de control digital"; y se dice en "tiempo real" porque debe responder con la rapidez necesaria para conseguir los objetivos. Se inicia presentando las características fundamentales de los dos grupos de sistemas que se consideran de tiempo real y las estructuras generales de los esquemas de control. los cuales se van desglozando para analizar las partes que los componen.

Como podrá notarse, es tan vasta la información y diversidad de los dispositivos que componen un esquema de control, que sería necesario un trabajo con una extensión muchísimo mayor para intentar comprender un pequeño porcentaje de ellos, cuestión que sale del objetivo de este trabajo. No obstante se ha intentado presentar los dispositivos básicos y necesarios (para automatizar cualquier sistema), dando algunos ejemplos de aplicaciones y algunas consideraciones técnicas acerca de ellos.

La organización de esta tésis ha quedado como sigue:

En el capítulo II se menciona brevemente las características, estructuras y algunas aplicaciones de los sistemas en tiempo real. Presenta además los dos grupos de estos sistemas, los de control y los de software: en el primero se analizan los esquemas de control de malla abierta y de malla cerrada mencionando sus características básicas; el segundo analiza a los sistemas de software llamados "en línea" que son los clasificados en tiempo real. Contiene además, las propiedades generales de estos sistemas.

En el capítulo III se mencionan las principales tecnologías empleadas en la fabrcación de microprocesadores, dando las características de cada una de ellas para una mejor selección del microprocesador de acuerdo a la tarea especificada. Se presenta además una tabla comparativa entre las características de las distintas tecnologías y algunos microprocesadores que se encuentran actualmente en el mercado. Las tecnologías mencionadas son: la bipolar convencional y Schottky; las PMOS NMOS HMOS, VMOS, CMOS; SOS e IIL.

En el capítulo IV (Selección y comparación del microprocesador), se hace una evaluación de los requerimientos del sistema microcomputadora de acuerdo al tipo de aplicación o tarea a realizar por éste. En una segunda sección se presentan las características más importantes de los microprocesadores para una correcta selección de estos, se analiza desde el propósito del microprocesador, tamaño de palabra, velocidades de procesamiento y reloj, disipación de energía y precios hasta capacidades de manejo de interrupciones. Finalmente se presentan los microprocesadores de mayor uso comercial, destacando en cada uno sus características más importantes como son la orientación de aplicación y algunas otras que no son presentadas en manuales técnicos, y que son de gran utilidad para distintas aplicaciones. Termina con una tabla comparativa de los microprocesadores presentados.

Si bien la elección del microprocesador y las necesidades determinan el hardware adecuado, es necesario entender que probablemente la elección se haga a nivel de sistema; es decir, de una microcomputadora que se acerque lo más posible a las características de evaluación hechas por el diseñador. Debido a que actualmente se dispone de microcomputadoras tan completas que cuentan con los requisitos tanto de software como de hardware para el diseño de control en tiempo real.

En el capítulo V se presentan los fundamentos matemáticos del control digital, las consideraciones para modelar y entender el modelo de una planta, la discretización del modelo matemático con el objeto de programarlo fácilmente; se analizan un poco más los esquemas de control en malla abierta y cerrada, mostrando las ventajas y desventajas de cada uno de estos. Se presentan además tres controladores de uso general en sus formas continua y discreta indicando las aplicaciones, ventajas y desventajas de estos. Finalmente, en la última sección se muestra la estructura general de un esquema de control digital y cómo se puede organizar el algoritmo de control; es decir, cómo podría ser dicho algoritmo; indica por otro lado la problemática del diseño e implementación de un esquema de control digital en tiempo real.

Algunos tipos de transductores se muestran en el capítulo VI, indicando como efectúan la conversión de una señal cualquiera en una señal eléctrica. Presenta la aplicación de los transductores así como su problemática. Pueden verse tambien transductores de temperatura, presión, posición, luz, etc. y al final, una tabla que resume los transductores de temperatura y fuerza más usados.

En el capítulo VII se presentan las características escenciales de los controladores y algunos esquemas de control sencillos, con el fin de identificar cada una de las componentes del esquema. Se muestran además alguno convertidores D/A disponibles en el mercado junto con sus especificaciones y algunas aplicaciones. Se debe hacer notar que la cantidad de dispositivos que pueden identificarse dentro del medio de actuación es tan grande, que se prefirió hablar de los dispositivos básicos además de presentar cuatro esquemas de control para dar una idea de la problemática que representa diseñar sistemas de control digital.

En el capítulo VIII (Diseño de software en tiempo real), se analizan los requisitos principales que el sistema operativo debe cumplir para una aplicación de software en tiempo real, en la

siguiente sección se hace un análisis semejante desde el punto de vista sistema, es decir, requisitos y características de la máquina virtual. En las siguientes dos secciones, presenta la primera el diseño de un sistema de control en tiempo real de comunicación de dos computadoras en lenguaje de alto nivel y la segunda, el diseño de software en tiempo real en lenguaje máquina, analizando algunos algoritmos importantes. Finalmente, se hace un estudio comparativo para el diseño de software en tiempo real, entre lenguajes de alto nivel y lenguajes ensambladores.

Por último, los apéndices contemplan el conjunto de instrucciones de los microprocesadores vistos en el capítulo IV, algunos transductores y convertidores comerciales y en el último apéndice se muestran los programas de comunicación entre dos computadoras en lenguaje "C".

Esta tésis fue elaborada en su totalidad por computadora, primeramente en el procesador de palabras de la computadora Southwest Technical Products Corporation, posteriormente en una Hewlett Packard Mod. 125 y finalmente en una Columbia Printaform; la unidad de impresión fue una máquina de escribir Olympia con interfase serie RS-232C. La transferencia de información entre la SwTPc y HP-125 se hizo a través de los programas de comunicación desarrollados por los autores; y la transferencia de HP-125 a Columbia se hizo a través de los programas de comunicación de la microcomputadora Columbia. Ambas transferencias utilizaron los puertos serie RS-232C de las computadoras.

C A P I T U L O I I

SISTEMAS DE TIEMPO REAL

II.1 INTRODUCCION

Los sistemas de tiempo real han adquirido gran importancia en los últimos años. El dramático descenso en los costos de los circuitos integrados los han hecho accesibles para aplicaciones industriales, comerciales, educativas, domésticas e incluso para diversión. Como ejemplos se pueden mencionar: el sistema de control computarizado de una fábrica de pan, el sistema de control de un motor de pasos, sistemas de seguridad por medio de microcomputadoras, el sistema de ignición en un coche, el juego de los "invasores" en su televisión, las básculas con microprocesadores, los equipos de audio y hornos de microondas programables, los grandes sistemas de control en los nuevos gaseoductos, las grandes y modernas centrales telefónicas en nuestro México actual, etc.

Cada uno de los sistemas mencionados requieren de un respaldo de programas de acuerdo al tipo de aplicación, de tal manera que es necesario utilizar técnicas de diseño de software de tiempo real que sean útiles en un amplio rango de aplicaciones. Además, también se hace necesario tener conocimientos sobre teoría de control, electrónica, computación, circuitos integrados, interfases de comunicación, etc.

Sin embargo, antes de presentar estas técnicas de diseño, es necesario mencionar algunas características de los esquemas (sistemas) de control.

II.2 ESQUEMAS TEORICOS DE CONTROL

Existen dos esquemas de control usados para resolver los problemas de contro. El esquema de malla abierta, en el cual no existe realimentación de la señal de salida a la entrada, se muestra en la Fig. II.2.1 y el esquema de malla cerrada, que realimenta la señal de salida, es mostrado en la Fig. II.2.2.

Como muestran ambas figuras, un sistema de control "digital" es aquel en el cual la señal en uno o más puntos del mismo está expresada en un código numérico para ser procesada por una computadora digital.

La utilización de un esquema depende del sistema que se desea controlar (sistema controlado) y del medio ambiente en que esté ubicado este sistema. Un esquema de control de malla abierta es usado básicamente cuando es muy difícil medir la señal de salida; por ejemplo, en una lavadora ¿cómo medir la limpieza de la ropa?. Con esto podrá notarse que este control trata de ajustar un tiempo en el que se supone, la ropa quedará limpia, y en medios donde el ruido no es significativo. Es decir, debido a que la computadora no tiene información de la salida del sistema controlado no podrá conocer las condiciones en las que se encuentra ésta, por lo cual, la utilización de este esquema requerirá conocer la evolución del sistema cuando es sometido a distintas señales de prueba y esta información deberá estar almacenada en la memoria de la computadora. La señal de entrada, en código digital, es una señal de referencia que sirve a la computadora para evaluar la señal de control que deberá enviar al sistema controlado, con el fin de obtener una señal de salida dada. La Fig. II.2.3 muestra en bloques las partes de un esquema de control.

Para utilizar un esquema de control de malla abierta, Fig. II.2.1 , es necesario tener en cuenta que cualquier alteración,

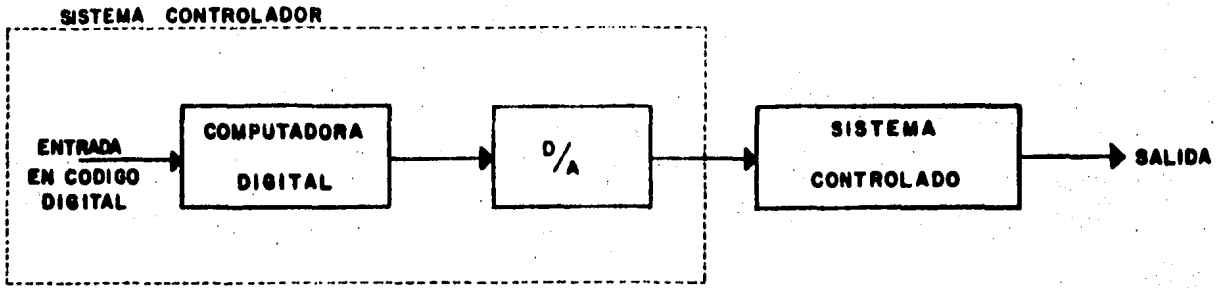


Figura II.2.1. ESQUEMA BASICO DE CONTROL DE MALLA ABIERTA

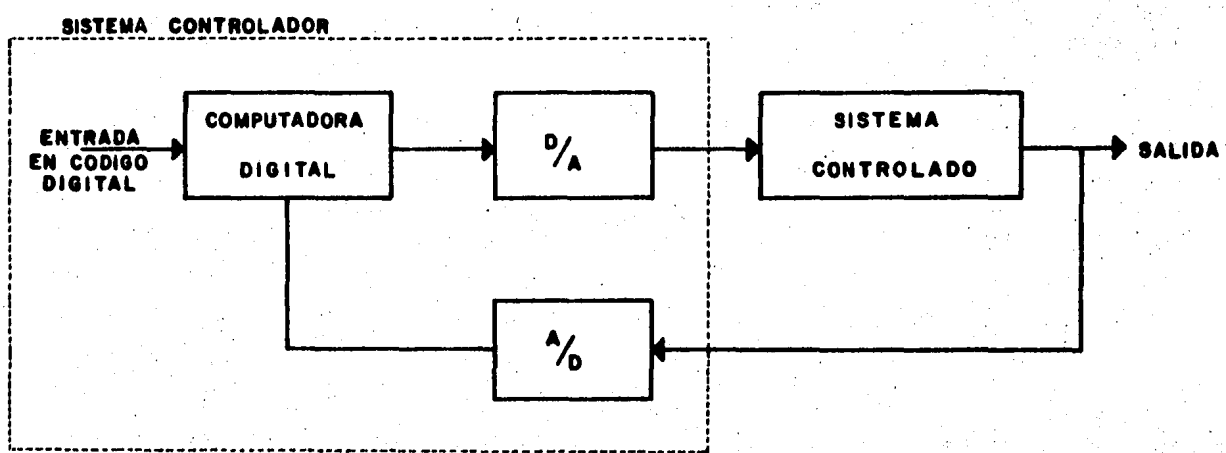


Figura II.2.2. ESQUEMA BASICO DE CONTROL DE MALLA CERRADA

producida por ruido, sobre la señal de entrada al sistema (señal de control), afectará la salida del mismo, pudiéndolo llevar a la inestabilidad y a causa de esto, provocar graves daños al sistema. Una ventaja de estos esquemas de control es la relativa sencillez del sistema controlador, haciendolos económicos.

El esquema de control de malla cerrada, Fig. II.2.2 , se usa únicamente cuando se tiene acceso a la señal de salida, ya que es necesario realimentarla a la computadora. La realimentación de la señal de salida es la manera de solucionar y eliminar en gran medida las inestabilidades provocadas por los efectos del ruido. A causa de esto, el sistema controlador será más complejo y por lo tanto, más caro. Se entiende fácilmente que la complejidad aumenta tanto en el hardware como en el software. Pero un sistema controlado muy sensible a pequeñas variaciones de la señal de entrada y en un medio ambiente ruidoso necesitará ser controlado mediante el esquema de malla cerrada.

De las Figuras II.2.1 y II.2.2 podemos observar que entre la computadora digital y el sistema controlado existe un bloque D/A, convertidor Digital-Analógico. Este se debe a que los sistemas controlados son en su mayoría contínuos en el tiempo, es decir, funcionan o actúan con señales analógicas (una entrada digital no transmite energía a un sistema contínuo) y se debe convertir a analógica la señal que envía la computadora digital.

La salida de un sistema contínuo será una señal contínua en el tiempo y para utilizar un esquema de control de malla cerrada, Fig. II.2.2 , en el que la señal de salida debe ser realimentada a la computadora digital, es necesario convertirla a digital mediante el bloque A/D, convertidor Analógico/Digital. Este convertidor A/D por sí mismo no podrá convertir la señal analógica a digital y por lo tanto, será necesario que otro bloque (dispositivo) ayude a este

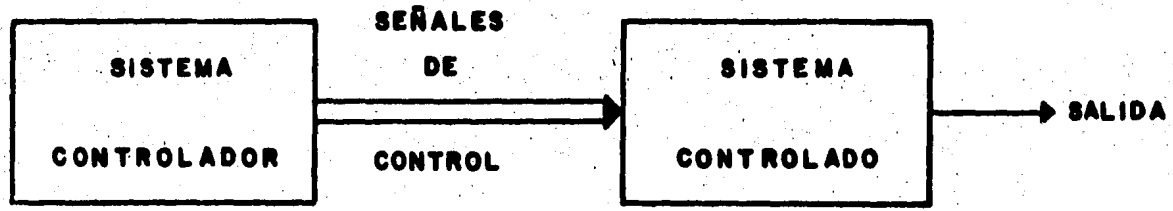


Figura. II.2.3. DIAGRAMA DE BLOQUES DE UN ESQUEMA DE CONTROL SIN REALIMENTACION.

convertidor. La Fig. II.2.4 muestra al dispositivo Sample/Hold que logra este objetivo y se nota como el bloque S/H.

El sample/hold es un dispositivo que muestrea una señal contínua y retiene esta muestra un determinado tiempo. El valor de la muestra en un instante de tiempo será el valor de la señal de salida en dicho instante. Una vez obtenida la muestra por el muestreador (sampler) ésta es mantenida por el retenedor (hold) durante un tiempo determinado por una indicación que se da al dispositivo.

La necesidad de tener un bloque S/H antes del A/D se debe a que el convertidor A/D necesita que la señal analógica que va a convertir, no varíe hasta que la conversión esté terminada. De otra manera no se tiene un equivalente digital de la señal analógica que debe llegar a la computadora digital.

De todo lo anterior puede observarse que la elección de un esquema de control depende de las exigencias y limitaciones que imponga el sistema controlado. Por otro lado, si la salida del sistema varía mucho (no queriendolo así) como resultado a pequeños cambios en la entrada y a los efectos del medio ambiente; y si es posible medirla, se recomienda un esquema de control de malla cerrada; no obstante sea más complejo y más caro.

De cualquier manera, la elección de un esquema (estrategia) de control constituye un compromiso entre la seguridad del personal que labore en (o con) el sistema controlado y de la maquinaria que lo constituya. la obtención de los mejores resultados posibles y la disposición económica.

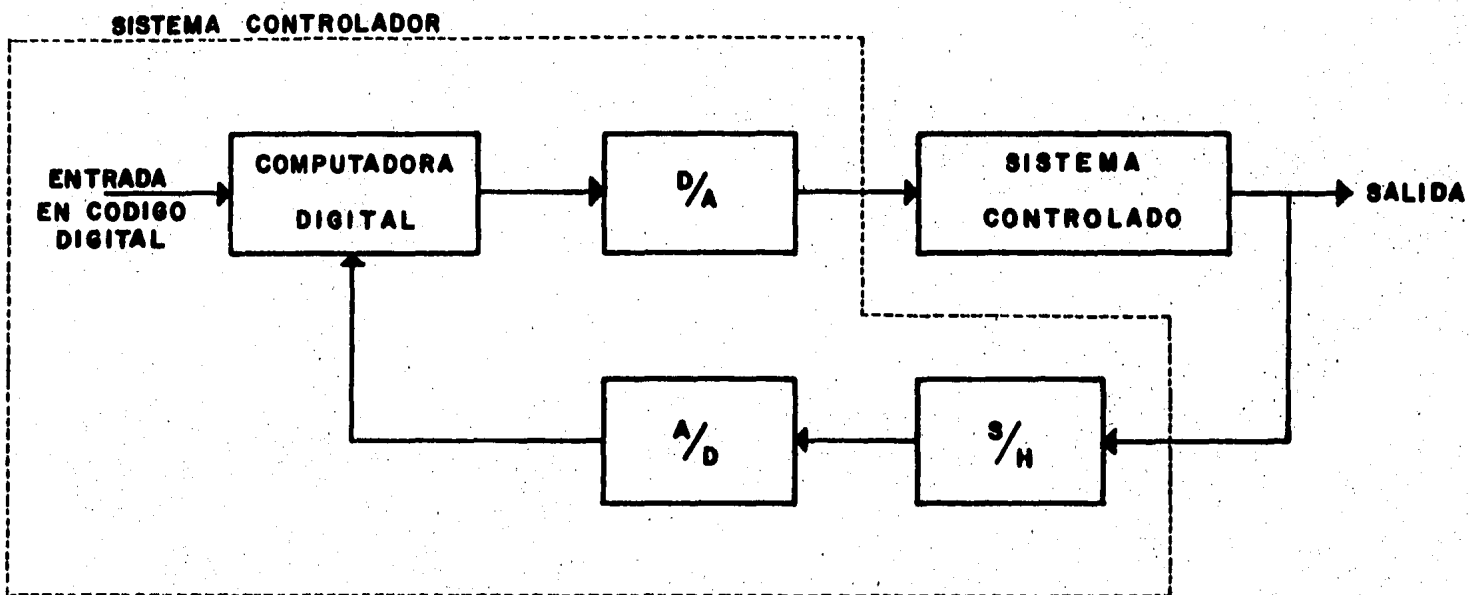


Figura. II.2.4 ESQUEMA DE CONTROL DE MALLA CERRADA.

II.2.1 Esquema Real de Control

Una vez vistos los esquemas teóricos de control, en la Fig. II.2.1.1 se presenta un diagrama de un lazo de control donde se diferencían las partes que lo componen y a continuación se definen.

La planta o proceso es una colección de equipos (motores, máquinas, tanques, tuberías, conexiones, etc.) interconectados con el propósito de lograr un objetivo: la obtención de un producto o un grupo de productos, de la mejor calidad y a costo aceptable.

En todo proceso existen condiciones o variables fundamentales (temperaturas, flujos, presiones, etc.) que rigen y determinan en gran medida la operación global del proceso y del sistema de producción. Los sistemas de control se implementan con el fin de manejar dichas variables para mantenerlas el mayor tiempo y tan cerca como sea posible de sus valores especificados, logrando con esto alcanzar los objetivos antes mencionados.

Los esquemas de control, por complejos que sean, pueden reducirse a la forma mostrada en la Fig. II.2.1.1. Como se observa en dicha figura, el lazo de control principia sensando (continuamente) la variable clave mediante el elemento primario de medición (transductor), éste detecta la variable y a partir de ella induce un efecto de tipo mecánico o eléctrico principalmente, que es tomado por el transmisor/indicador para producir la indicación y la señal transmisible, correspondientes a la magnitud de la variable medida. En el controlador, esta variable se compara con el valor especificado o de referencia y ante una diferencia, el controlador reconoce la magnitud y signo de la desviación con tal de definir las acciones destinadas a corregir el valor de la variable de proceso para ajustarlo al valor de referencia. El actuador y el elemento final de control se encargan de efectuar las acciones dictadas por el controlador; el actuador activa al elemnto final de control para que

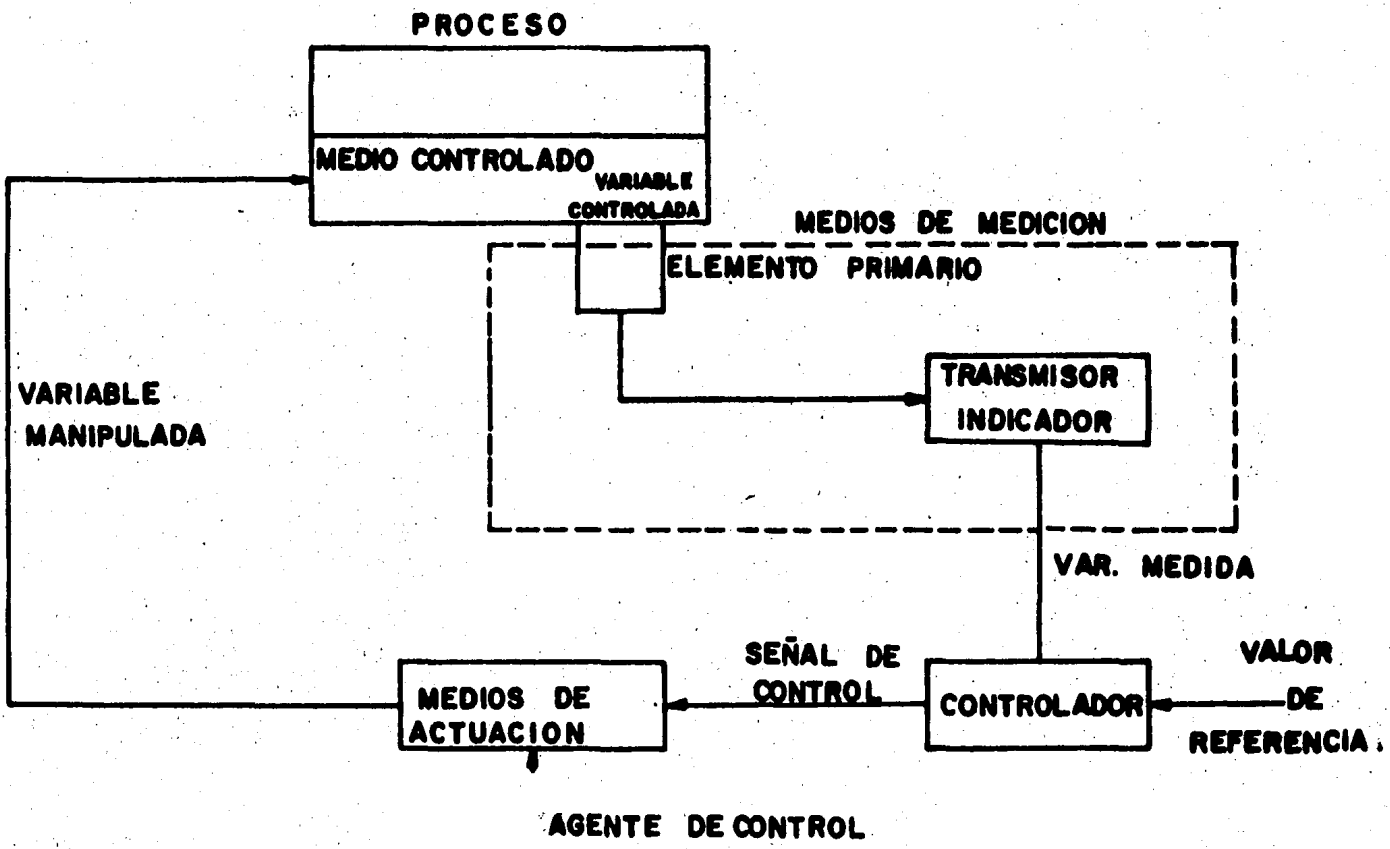


FIG. II.2.1.1. ESQUEMA DESCRIPTIVO DE UN LAZO DE CONTROL.

maneje adecuadamente aquella energía o material del proceso que interactuando con el medio, modifica y/o mantiene la magnitud de la variable fundamental.

Con el objeto de complementar la descripción del lazo de control, a continuación se definen diversos conceptos y elementos relativos a él:

- Medios del proceso. Son los distintos entes energéticos o materiales que desempeñan una función específica. Por ejemplo en el proceso de generación termoeléctrico un medio material es la bomba de agua de alimentación y un medio energético es el campo excitador del generador.

- Medio controlado. Es aquel medio en donde existe una condición o variable clave que es controlada.

- Variable controlada. Es aquella variable o condición que por tener una influencia notable en la operación de la planta, es medida y controlada. Por ejemplo en un calentador a fuego directo se toma como variable controlada el gasto de combustible, entonces el medio controlado es el sistema de suministro de combustible.

- Agente de control. Es el medio del proceso que al interactuar con el medio controlado, determina el comportamiento y la magnitud de la variable controlada.

- Variable manipulada. Es la condición del agente de control que es manejada por el lazo de control para influir sobre la variable controlada.

- Medios de medición. Consisten de aquellos elementos del esquema de control que están involucrados en la determinación y comunicación del valor de la variable controlada.

- Elemento primario. Es el elemento de medición que entra en contacto con el medio controlado para detectar la variable e inducir un efecto medible que la represente.

- Indicador/transmisor. Es el dispositivo que toma el efecto producido por el elemento primario para generar una indicación escalada y/o una señal transmisible que informen de la magnitud de la variable medida.

- Controlador. Es el dispositivo que opera en base a la variable medida para mantener, corregir o limitar el valor de ésta con respecto a un valor especificado.

- Actuador. Es el dispositivo que recibe la señal del controlador para regular la potencia suministrada al elemento final de control.

- Elemento final de control. Es el dispositivo que actúa directamente sobre la variable manipulada para mantener el valor especificado de la variable controlada.

La terminología empleada en los esquemas de control es la siguiente:

- Punto de control. Es el valor de la variable controlada el cual, bajo cualquier condición de ajuste fijado, el control automático opera para mantenerlo.

- Punto de ajuste. Es el ajuste que se hace en el controlador para proporcionar una señal de referencia.

- Desviación (señal de error). Es la diferencia entre el valor instantáneo de la variable controlada y el valor determinado por el punto de ajuste.

- Señal de control. Es la señal generada por el controlador, por medio de la cual indica al actuador las acciones que deben realizarse para asegurar el control.

II.3 SISTEMAS DE TIEMPO REAL

Una definición sencilla de un sistema en tiempo real sería aquel sistema que opera en tiempo real, esto es, que responde a la necesidad de actuar en un período de tiempo proporcional a la urgencia de la necesidad. En aplicaciones de control, el sistema se basa en proporcionar la información necesaria para actuar sobre el mundo real.

El tiempo real describe el procesamiento de información de una manera suficientemente "rápida", en la que el résultado del procesamiento está disponible inmediatamente para controlar el proceso que se está monitoreando.

Básicamente cualquier computadora puede configurarse para ejecutar algunas operaciones de tiempo real, el criterio primordial es que la computadora sea capaz de ejecutar una acción específica en un período de tiempo particular. La extensión de operar en tiempo real depende de la velocidad de ejecución.

A continuación se enunciaran sus características más importantes.

II.3.1 Sistemas de control

Un sistema de control en tiempo real reacciona, así como afecta al medio ambiente en el cual opera. Es una colección de dispositivos controlados por un programa almacenado que actúa como el elemento regulador en un sistema (esquema) de malla cerrada. Ver Fig. II.2.2.

Por conveniencia se divide el sistema de tiempo real en dos partes: El Sistema Controlado y El Sistema Controlador.

El Sistema Controlado consiste de los dispositivos físicos que el Sistema Controlador regula, mientras que el Sistema Controlador consiste del software junto con los dispositivos físicos de procesamiento, ver Fig. II.2.2.

En un sistema de procesamiento de datos, los dispositivos controlados pueden ser terminales, manejadores de discos, impresoras y lectoras de tarjetas; en un sistema de telecomunicaciones pueden ser multiplexores, manejadores de líneas y teletipos; en un sistema de control de procesos industriales válvulas, relevadores, interruptores, etc.. Normalmente estos dispositivos no funcionan independientemente de otros. Por ejemplo, en un sistema de seguridad la alarma sonará si algun sensor detecta un intruso. En un sistema de calefacción/aire acondicionado el termostato informará al sistema cuando activar el ventilador para mantener una temperatura establecida.

II.3.2 Sistemas de Software

Con la finalidad de obtener una definición del software controlador de un sistema de tiempo real, se analiza un pequeño número de sistemas de software.

II.3.2.1 Sistemas Contables en Lotes (BATCH)

En este tipo de sistemas, la computadora no controla directamente el sistema contable, sino que procesa los datos que son usados para el control administrativo. Un sistema computarizado comercial es una colección de programas de aplicaciones para: verificación de datos, nóminas, bases de datos y otros cálculos financieros. Estos programas manejan los datos y asisten indirectamente en el control del sistema administrativo. En un sistema convencional de BATCH, estos programas son ejecutados en lotes (como el pago de la tenencia del automóvil) debido a que la computadora no ejerce un control inmediato. Además, los programas no necesitan ejecutarse con apremio de tiempo. Los sistemas BATCH no son sistemas de tiempo real.

II.3.2.2 Sistemas en Línea

Los sistemas administrativos como sistemas en línea, deben mantener un continuo control sobre sí mismos. Esto lo hacen, asegurándose que para cada transacción que ocurra, los archivos relevantes y reportes sean actualizados antes de que la siguiente transacción afecte estos archivos. Esto implica que los programas de aplicación no siempre son ejecutados a tiempos dictados por el uso óptimo de los elementos de la computadora. Por el contrario, son ejecutados a tiempos impuestos por los requerimientos de las terminales en acceso y otros dispositivos que fueron diseñados para entradas individuales de transacciónes o preguntas. El sistema Boletrónico es un ejemplo de sistema en línea.

II.3.2.3 Sistemas Operativos

La colección de programas que controlan el sistema de cómputo por sí solo constituyen el sistema operativo de éste. Esta colección ha

sido diseñada para: (a) controlar los dispositivos físicos que constituyen el sistema; (b) para asegurarse que los recursos del hardware sean empleados optimamente y (c) proveer programas de utilería tales como editores y facilidades para organización de archivos. Los programas de control de dispositivos, rutinas de manejos de interrupciones y otros programas en el núcleo de un sistema operativo de multiprocesamiento, pueden considerarse como un sistema de tiempo real.

El sistema controlado en este caso consiste de: discos, impresoras, terminales, modems y dispositivos similares que constituyen el sistema de cómputo.

II.3.3 Sistemas de Control de Procesos y de Comunicaciones

Dentro de los sistemas de control de procesos se incluyen: control de plantas químicas, misiles teledirigidos, maquinaria de fabricación, algunas plantas automotrices, etc. Dentro de los sistemas de comunicación se incluyen: las redes de computadoras y central telefónica controlada por computadora. En estos sistemas de tiempo crítico, es obvio que los sistemas de tiempo real estan trabajando como un elemento de control y se puede agregar otro aspecto importante: la confiabilidad, pues esta clase de aplicaciones requiere de la operación correcta del sistema, del cual dependen equipos valiosos y hasta vidas humanas.

II.3.4 Propiedades Generales

A continuación se mencionan algunas de las propiedades importantes con que cuentan estos sistemas de tiempo real.

I.3.4.1 Tiempo de respuesta adecuado

Un sistema que no reaccione suficientemente "rápido" en el medio en que actúa, no puede considerarse un sistema de tiempo real. Sin embargo se debe considerar que "rápido" depende del sistema que se trate. Por ejemplo, para controlar el curso de un proyectil el sistema de tiempo real debe reaccionar en unos cuantos milisegundos; una respuesta dos segundos despues, sería crítica. Sin embargo, en un sistema bancario en línea, la respuesta de saldo de cuenta del Sr. Rodríguez puede tardarse unos cuantos segundos y seguiría siendo "rápido".

II.3.4.2 El sistema debe ser correcto y completo

Es decir, debe prever todas las situaciones y darles solución correcta. Se hace extremadamente difícil especificar los requerimientos de un sistema diseñado para controlar una porción del mundo real, debido al amplio rango de situaciones y condiciones que ocurren en él.

II.3.4.3 Confiabilidad

El sistema debe proveer un servicio para el cual fue diseñado, y garantizar un tiempo entre la falla y la reparación. . Un ejemplo de un sistema estrictamente confiable es la central telefónica controlada por computadora.

II.3.4.4 Economía

Se considera obvia esta razón, no obstante, vale la pena mencionar un ejemplo: un sistema en línea que garantice un máximo tiempo de respuesta de medio segundo, no puede competir con un sistema

que solo ofrece un tiempo de respuesta de dos segundos !por la mitad del precio de aquél.

II.3.4.5 Disponibilidad de los elementos del sistema

El diseñador debe considerar la disponibilidad de los componentes en México, para poder dar soporte al sistema a partir de componentes que se encuentren en el mercado nacional.

C A P I T U L O I I I

TECNOLOGIA DE LOS MICROPROCESADORES

III.1 INTRODUCCION

El objetivo de este capítulo es describir brevemente algunas tecnologías empleadas en la fabricación de microprocesadores y circuitos integrados, debido a que la comprensión de los procesos de manufactura son importantes para el diseñador o usuario de microcomputadoras, ya que el tipo de tecnología determina los siguientes puntos:

- La velocidad de operación del dispositivo.
- El grado de integración.
- El consumo de energía.
- El costo del circuito integrado.
- Las interfases requeridas por el circuito integrado.
- La confiabilidad del dispositivo.

III.1.1 Tipos de procesos y estructuras utilizadas

Existen en particular dos tipos de tecnologías de fabricación para circuitos integrados digitales y microprocesadores: (1) tecnología bipolar y (2) tecnología MOS (Metal-Oxide-Semiconductor). Estos dos tipos tienen a su vez procesos derivativos, que se usan en la fabricación de todas las microcomputadoras actuales y que serán usados para la fabricación de la mayoría de ellas en el futuro. La siguiente tabla hará más explícito lo anterior.

TIPOS DE PROCESOS DE FABRICACION DE MICROCOMPUTADORAS

BIPOLAR	MOS
. Bipolar (convencional)	. P-channel MOS-PMOS
. Schottky Bipolar	. N-channel MOS-NMOS
. Integrated Injection Logic IIL	. Complementary Symmetry MOS-CMOS
	. Silicon on Saphire SOS
	. VMOS
	. HMOS

Todos los procesos anteriores usan la estructura básica de juntura P-N para construir los transistores y de estos, los circuitos lógicos digitales que conforman una microcomputadora.

III.2 JUNTURAS P-N

La juntura P-N es la componente básica de todos los dispositivos semiconductores. Los semiconductores usados son: silicio, germanio y arseénico de galio. Cabe hacer notar que el silicio es el más usado de los tres.

Un semiconductor se dice del tipo-n o del tipo-p, dependiendo del exceso o falta de electrones en su estructura de cristal (no atómica), respectivamente. Es decir, debido a que los átomos de silicio son tetravalentes, al contaminar un cristal de silicio puro (semiconductor intrínseco) con átomos pentavalentes, tales como el fósforo, el resultado será un cristal con exceso de electrones (semiconductor extrínseco), formando el semiconductor tipo-n. De la misma manera, pero usando ahora átomos trivalentes como el boro, si se contamina

un cristal de semiconductor intrínseco, resultará un extrínseco falto de electrones de valencia o con huecos. Es decir, un hueco o ausencia de un electrón actúa como una carga positiva, produciendo una estructura de cristal con exceso de cargas positivas, o sea un semiconductor tipo-p.

Se entiende por un cristal semiconductor extrínseco, aquel cuya estructura es de un átomo contaminador (pentavalente o trivalente), con cuatro átomos de silicio tetravalente a su alrededor, imagínese ahora la estructura, con una contaminación de aproximadamente 10 exp +15 a 10 exp +21 átomos contaminadores por centímetro cúbico de silicio. Finalmente, aquellos elementos tales como el fósforo o arsénico, usados para producir silicio del tipo-n son conocidos como Donadores, debido por supuesto, a que ellos proporcionan electrones a la estructura de cristal de silicio. El boro, por otro lado es conocido como Receptor, debido a que acepta electrones de la estructura cristalina de silicio y produce un exceso de cargas positivas o huecos en el silicio.

Una juntura P-N de silicio, Fig. III.2.1, tiene la característica de conducir cuando un potencial positivo es aplicado al silicio tipo-p con respecto al silicio tipo-n, Fig. III.2.2. Un voltaje aplicado de esta manera se dice que es un voltaje de directa. Para el silicio, la conducción toma lugar cuando se alcanza el voltaje de ruptura de 0.7 volts.

Por otro lado, si el potencial aplicado al silicio tipo-p es negativo, Fig. III.2.3, entonces se dice que este es un voltaje de inversa y no permite por lo tanto la conducción. Así que, una juntura P-N actúa como un rectificador o diodo, Fig. III.2.4, conduciendo cuando el voltaje aplicado sobre él es en una dirección y actuando como un circuito abierto cuando el voltaje está en sentido inverso.

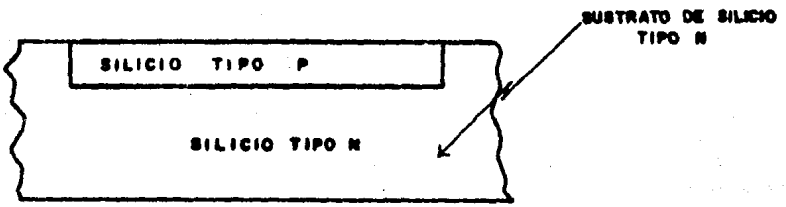


FIG. III.1.1 JUNTURA P-N

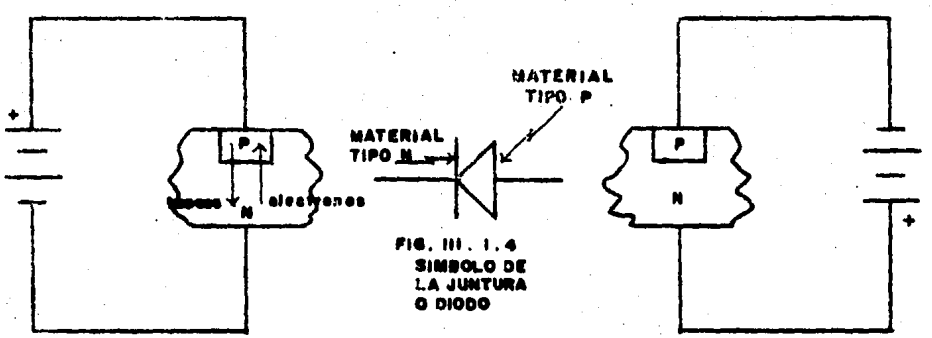


FIG. III.1.2
POLARIZACION EN
DIRECTA

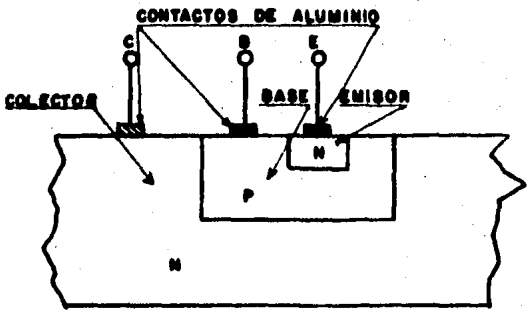
FIG. III.1.4
SIMBOLO DE
LA JUNTURA
O DIODO

FIG. III.1.3
POLARIZACION EN
INVERSA

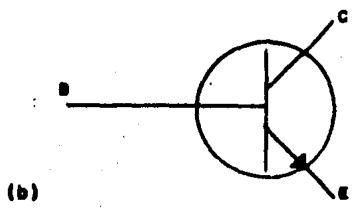
III.2.1 Tecnología Bipolar (convencional)

Si se unen dos junturas P-N o N-P y se crea una tercera N-P-N, Fig. III.2.1.1, o P-N-P, Fig. III.2.1.2, en una estructura cristalina de silicio, entonces se ha formado un transistor bipolar. El nombre de bipolar se debe a que se da lugar tanto flujo de electrones como de huecos. Este transistor bipolar tiene la característica de que la corriente proporcionada al material central controlará una corriente mucho mayor que fluye de una orilla a otra. Una de estas orillas actúa como fuente o emisor de cargas y la otra como receptor o colector de esas cargas. El centro o base controla la corriente que fluye de el emisor al colector o viceversa. Se dice que el transistor está Activo, cuando a la base se le inyecta suficiente corriente y está Cortado, cuando a la base no se le alimenta corriente o ésta es tan pequeña que no alcanza a polarizar en directa la juntura base-emisor.

Para aplicaciones de amplificación la juntura base-emisor deberá polarizarse en directa, mientras que la juntura base-colector estará en inversa. Para aplicaciones de conmutación o lógica digital, la juntura base-emisor se polariza en directa, y cuando el transistor enciende, la juntura base-colector se pone en directa, dando origen al estado de Saturación. Cabe decir que cuando la juntura base-colector está en directa, la corriente de huecos fluye del material tipo-p al material tipo-n y viceversa para la corriente de electrones. Este flujo de corrientes produce Cargas Almacenadas o exceso de portadores minoritarios (electrones en el material P, y huecos en el material N) en la vecindad de las junturas. Si despues de la polarización directa en la juntura base-colector, el voltaje se invierte (es decir, cuando el transistor se apaga), los portadores minoritarios fluyen en sentido contrario a través de la juntura, generando un flujo de corriente por

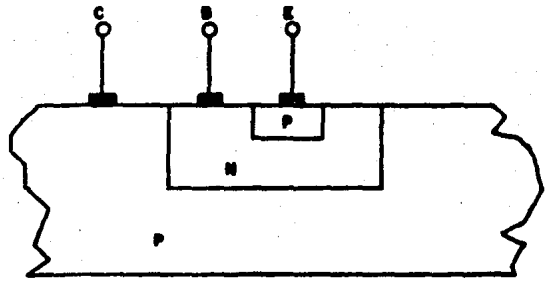


(a) TRANSISTOR BIPOLAR N.P.N.

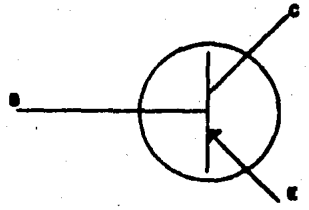


(b) SIMBOLO

FIG. III. 1.1.1 (a), (b)



(a) TRANSISTOR BIPOLAR P.N.P.



(b) SIMBOLO

FIG. III. 1.1.2 (a), (b)

permitiéndoles a los portadores minoritarios recombinarse, cesando entonces esta corriente. Este tiempo es bien conocido como Tiempo de Almacenaje y limita la frecuencia de conmutación de un estado de encendido a un estado de apagado en un transistor. Este retardo es superado por dos métodos muy comunes: (1) proceso de contaminación de oro y (2) proceso bipolar Schottky. Las ventajas del primer proceso son: (a) permitir la fácil recombinación de los portadores minoritarios y (b) reducir el Tiempo de Almacenaje a aproximadamente 6 nanosegundos; una de sus desventajas es la reduccion de la ganancia del transistor. Debido a que el Tiempo de Retardo de Propagacion= Retardo del Dispositivo + Tiempo de Almacenaje, se puede concluir que una reducción en el Tiempo de Almacenaje aparte de reducir el Retardo de Propagación, incrementa la frecuencia de pulsos que el dispositivo puede manejar. El proceso Schottky se tratará más ampliamente.

III.2.2 Tecnología Bipolar Schottky

Este proceso se basa en colocar un diodo Schottky entre la base y el colector, Fig. III.2.2.1, evitando así las desventajas del proceso de contaminación de oro y al mismo tiempo reducir efectivamente la carga almacenada en la juntura base-colector. El diodo Schottky es una juntura de rectificación que se forma colocando un metal en contacto con un semiconductor, Fig. III.2.2.2, es decir, se utiliza (generalmente) aluminio en contacto con silicio tipo-n, generándose en esta juntura un flujo de corriente de electrones del silicio hacia el aluminio, provocando un rápido equilibrio de estos electrones con los del metal sin resultados de cargas almacenadas. Debido a que el diodo Schottky tiene un voltaje de ruptura menor al de un diodo convencional (aproximadamente 0.4 V), la corriente de exceso de la base que normalmente manejaría el transistor hacia la saturación, ahora es controlada por el diodo Schottky y ya no fluye hacia la base.

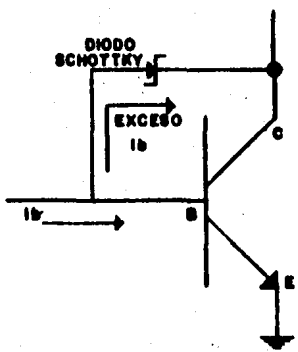


FIG. III. 1.2.1 TRANSISTOR BIPOLAR SCHOTTKY

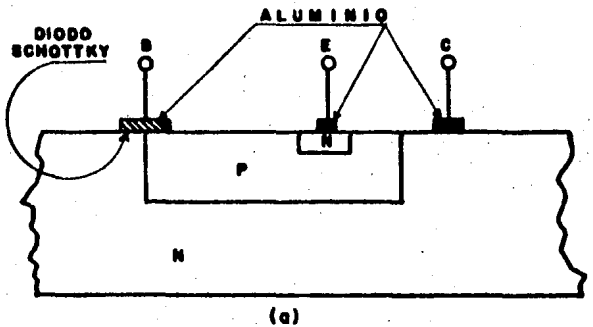
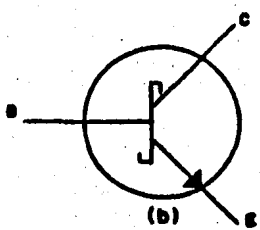


FIG. III. 1.2.2 (a) ESTRUCTURA, (b)



SIMBOLO

El transistor bipolar Schottky tiene las siguientes ventajas sobre el transistor contaminado en oro:

- El Tiempo de Almacenaje no se incrementa con la temperatura.

- Es menos sensible a cambios en la fuente de poder.

- Maneja trenes de pulsos a más altas frecuencias, debido a su bajo Retardo de Propagación (7 ns .vs. 14 ns).

III.3 TECNOLOGIAS MOS (METAL-OXIDE-SEMICONDUCTOR)

Otro proceso muy utilizado en la fabricacion de transistores es la tecnología MOS (Metal-Oxide-Semiconductor), en el cual, sobre un sustrato de silicio (n o p), se difunden dos depósitos fuertemente contaminados (contrarios al sustrato). La parte exterior localizada entre estos depósitos, se cubre por una lámina de bióxido de silicio sobre la cual se deposita una capa de metal. Esta capa de metal se llama la compuerta (gate), los depósitos se llaman fuente (source) y dranaje (drain), al sustrato que queda entre los depósitos se le llama canal (channel).

Si los depósitos fuertemente contaminados son del tipo-p, entonces el transistor es un PMOS o transistor MOS canal p, de la misma manera si son del tipo-n, se llamara NMOS o transistor MOS canal n. A continuación se describirán estas tecnologías brevemente.

III.3.1 PMOS (MOS CANAL-P)

Si se aplica un potencial a la fuente y al drenaje de un PMOS, el flujo de corriente entre estos puede controlarse por el potencial de la compuerta con respecto al sustrato. Si se aplica un potencial negativo menor que el voltaje de ruptura de la compuerta Vt (aprox.= -2 V) entonces, cargas positivas (huecos) son atraídas a la superficie del silicio entre los depositos, formando entre estos un canal de conducción. Cuando el PMOS conduce, si en la compuerta hay un voltaje mayor a Vt y por el otro lado se corta cuando el voltaje mencionado es cero, entonces se dice que este es un transistor PMOS de modo mejorado (enhancement mode). Por el contrario, si el PMOS conduce de la fuente al drenaje cuando el voltaje en la compuerta es cero o menor que Vt y no conduce cuando el voltaje en la compuerta es mayor a Vt, entonces se trata de un transistor PMOS de modo agotamiento (depletion mode). Es importante hacer notar, que debido a que el PMOS modo mejorado ofrece mayor inmunidad al ruido y menor consumo de energía, es más ampliamente usado que el modo agotamiento para la fabricación de microprocesadores y memorias. Una desventaja del PMOS, es que presenta una carga capacitiva a cualquier dispositivo, debido a que está aislado de los depósitos a través de una capa de bióxido de silicio (SiO_2). Otra de ellas es el proceso tan complicado para poder formar la compuerta del PMOS; es decir, en ocasiones el aluminio no era depositado completamente en la superficie del canal (situado entre la fuente y el drenaje), o en ocasiones se mezclaba este aluminio con los depósitos, haciendo que el canal de conducción fuera muy pobre o no se formara. En el primer caso, incrementará la capacitancia entre la compuerta y la fuente o el drenaje; en el segundo, en ambos casos la velocidad de conmutación del transistor se reducía y si esto se traslada a la fabricación de microprocesadores, se incrementaba tambien el tiempo de ejecución de instrucción (ver Fig. III.2.1).

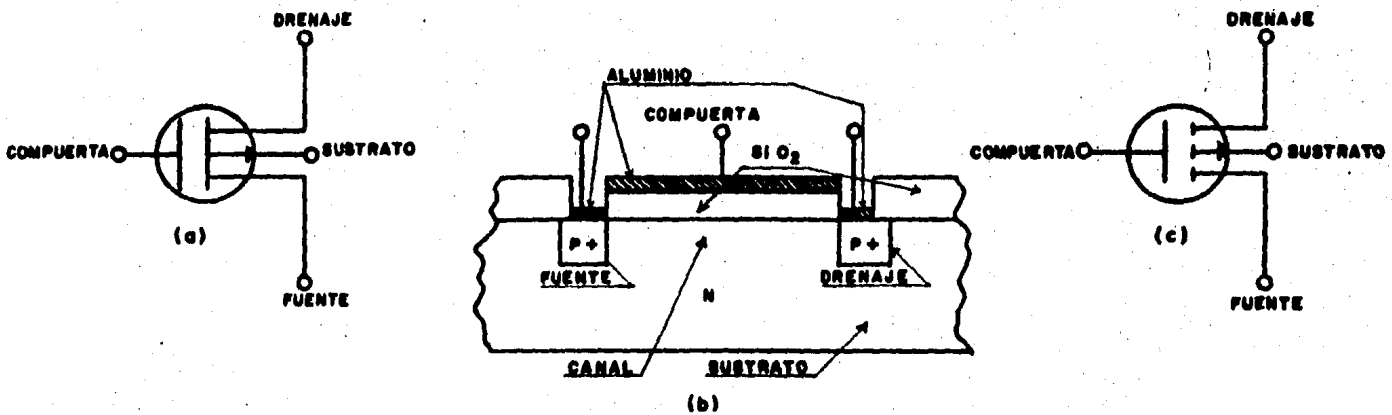


FIG. III.2.1 SIMBOLOS Y ESTRUCTURA DEL TRANSISTOR PMOS. (a) MODO AGOTAMIENTO, (b) ESTRUCTURA, (c) MODO MEJORADO

Las soluciones que se propusieron para eliminar o reducir algunos de estos problemas asociados con el aluminio de la compuerta, fué otro proceso que utilizaba en su lugar silicio policristalino, el cual era depositado antes de la difusión de los depósitos, evitando así mezclarse con estos. Sus ventajas se notaron de inmediato:

- Reducción de capacitancias.

- Incremento de velocidad de conmutació...

- Reducción del voltaje de ruptura de la compuerta.

- Dimensiones pequeñas de la compuerta.

Adicionalmente se menciona que los microprocesadores PMOS proceso silicio en la compuerta, operan con un reloj de frecuencia entre los 500 KHz a 1 MHz y un tiempo de ejecución de aproximadamente 10 microsegundos.

III.3.2 NMOS (MOS CANAL-N)

El transistor NMOS opera de la misma manera que el PMOS, excepto que en el canal de conducción circulan electrones y no huecos cuando el transistor está conduciendo.

Debido al incremento de movilidad en tres veces de los electrones en el NMOS, se obtiene tres veces más velocidad que en un PMOS proceso silicio en la compuerta y hasta cinco veces más que en un PMOS proceso aluminio. Un microprocesador NMOS ejecuta instrucciones en un promedio de 0.5 a 2 microsegundos y opera con reloj de frecuencias entre 1 MHz y 4 MHz.

III.3.3 HMOS (MOS de alto desempeño)

El HMOS se realiza reduciendo las dimensiones del proceso NMOS, para conseguir un incremento en su desempeño. Es decir reduciendo el tamaño del canal de conducción del NMOS, se obtiene, por un lado, una mayor densidad y velocidad en circuitos LSI y VLSI, con sus correspondientes decrementos en el consumo de energía. Algunos parámetros típicos del proceso HMOS son tamaños de canal de 3.5 micrómetros, tiempos de acceso de memoria de aproximadamente 50 nseg y consumo de energía de 1 picojoule a 5 volts. Los HMOS pretenden ser los dominantes de la tecnología MOS para sistemas de alta densidad LSI en circuitos integrados tales como memorias y microcomputadoras de 16 bits.

III.3.4. VMOS (MOS ACANALADO EN V)

Otro desarrollo para conseguir un tamaño de canal corto y con esto incrementar el desempeño y densidad, es el acanalado en V tecnología MOS o VMOS. En el proceso VMOS, el transistor NMOS es formado verticalmente a través de una serie de capas de semiconductor como se muestra en la Fig III.3.4.1.

La región n+ de la fuente es la capa más profunda y sirve como una tierra común a todos los transistores VMOS. La siguiente capa es el canal (P), la cual envuelve toda la periferia del acanalado V, proporcionando una gran área de canal con un dopado pequeño. La siguiente capa pequeña epitaxial es la "pi", la cual separa la capa P del canal de la región n+ drenaje. El efecto de esta capa es incrementar el voltaje de ruptura entre el drenaje y el sustrato y además, incrementar la velocidad de conmutación por medio de reducir la capacitancia de la juntura. Conclusión: VMOS consigue alta integración, incremento de velocidad de conmutación y reducción de capacitancia.

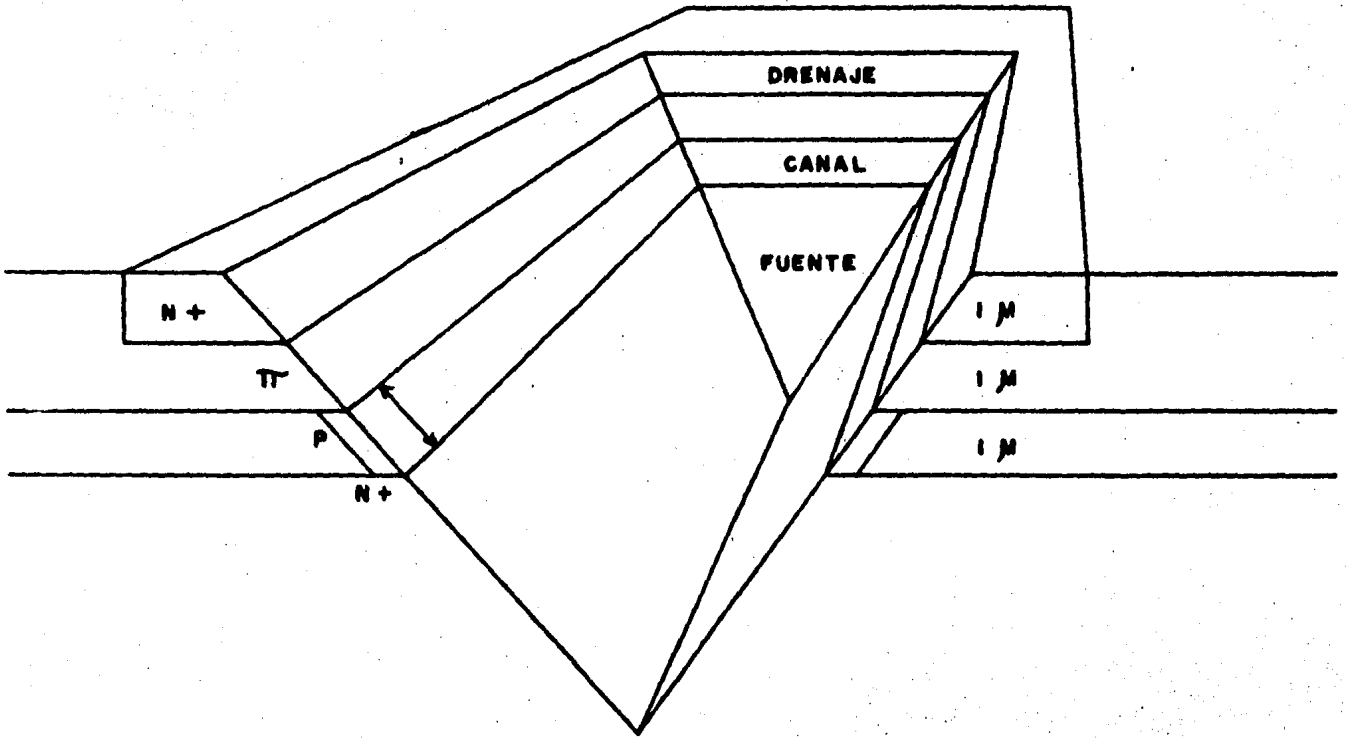


FIG. III.2.4.1 ESTRUCTURA DE UN TRANSISTOR VMOS.

III.3.5 CMOS (MOS COMPLEMENTARIO)

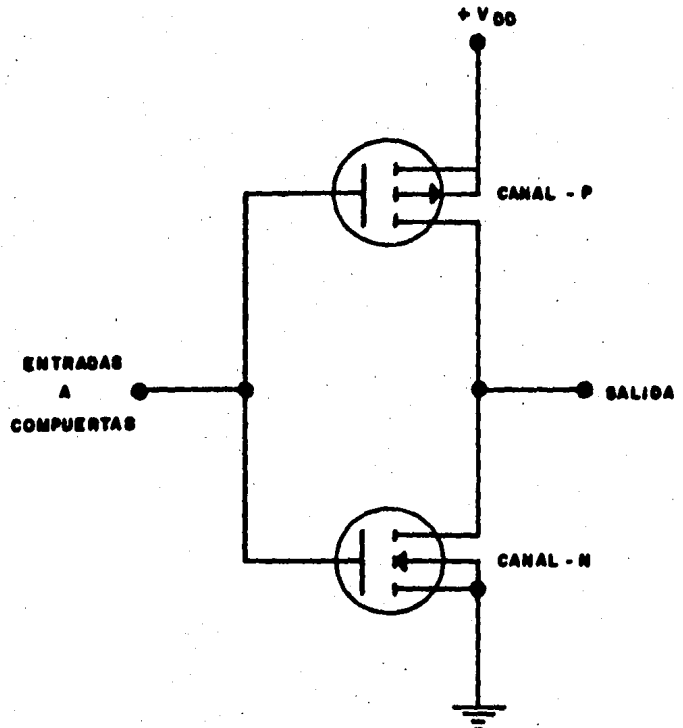
La tecnología CMOS o MOS complementario, combina los transistores canal-p y canal-n en el mismo sustrato en una estructura complementaria, como se muestra en la Fig. III.3.5.1.

En la compuerta CMOS, las compuertas de los transistores canal-p y canal-n, están conectadas de tal manera que para intervalos de conmutación, la salida tendrá estados cambiarios, es decir, solamente un transistor estará activo a un tiempo.

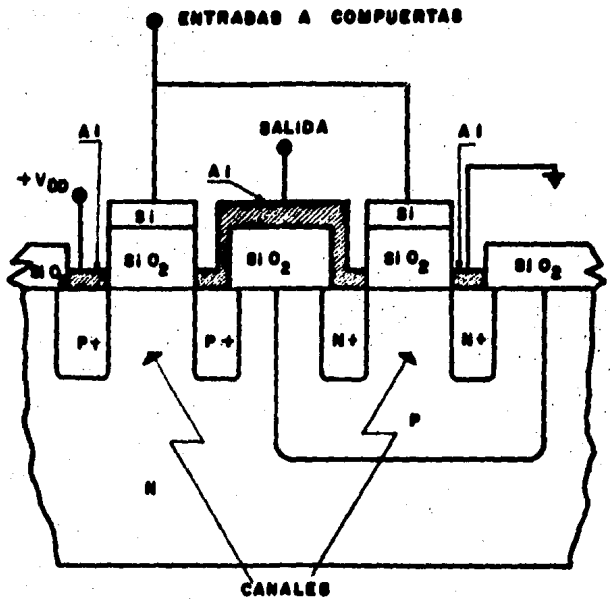
Si se aplica un "0" lógico o tierra a la entrada, el transistor canal-p se encenderá conectando el voltaje de polarización Vdd a la salida, produciendo un "1" lógico. El transistor canal-n se apagará cuando el voltaje de compuerta no es suficientemente positivo con respecto al sustrato. Contrariamente, si un voltaje positivo o "1" lógico es aplicado a la entrada, el transistor canal-n encenderá y el transistor canal-p se apagará, reflejándose a la salida un "0" lógico.

Este arreglo complementario significa que durante una operación estática existe siempre una alta impedancia entre Vdd y tierra y un flujo muy pequeño de corriente (microamperes).

La característica del proceso CMOS, es alta inmunidad al ruido (aproximadamente 40 % del voltaje de polarización), insensibilidad a variaciones de temperatura, baja disipación de energía en modo estático o baja frecuencia, operación bajo un amplio rango de voltaje de polarización (+5 a +18 volts), alto costo de producción y una baja densidad con respecto a PMOS y NMOS. Estas características del proceso CMOS lo hacen un buen candidato para usarse en sistemas automotrices de bajo poder operado por baterías.



SIMBOLO COMPUERTA CMOS



ESTRUCTURA COMPUERTA CMOS.

Fig: III 2.5.1 SIMBOLO Y ESTRUCTURA DE LA COMPUERTA CMOS.

III.4 OTROS PROCESOS

Estos son importantes debido a que algunos microprocesadores y circuitos integrados

III.4.1 SOS. Silicio en Safiro

En este proceso las principales ventajas son la efectiva eliminación de la capacitancia drenaje-sustrato que está presente en otros procesos MOS. Esta eliminación se obtiene por la difusión de las regiones drenaje en el sustrato de safiro. Debido a que hay menos capacitancia que cargar y descargar en una compuerta SOS, los requerimientos de potencia son menores que en una compuerta convencional CMOS y la velocidad de operación es mayor. Los dispositivos LSI SOS pueden operar a velocidades de aproximadamente 90 MHz con retardos de propagación de 1 a 2 nanosegundos.

III.4.2 IIL. Lógica de Inyección Integrada

La tecnología que combina la densidad de MOS con la velocidad de bipolares es IIL o algunas veces llamada MTL (lógica de transistores fusionados), la cual elimina el requerimiento de resistencias de carga reemplazándolas con transistores inyectores de corriente.

La compuerta IIL consiste de un transistor NPN con múltiples colectores abiertos (open collectors), que son integrados verticalmente en un sustrato de silicio. La diferencia entre este tipo de transistor y el convencional es que el transistor IIL NPN se encuentra vertical, es decir, la región del emisor es la parte más profunda en el sustrato y el colector está en la superficie.

RESUMEN DE TECNOLOGIA DE MICROPROCESADORES CON EJEMPLOS REPRESENTATIVOS

BIPOLAR		MOS		
no saturado Schottky	saturado IIL	CMOS	Canal-N	Canal-P
Intel 3001	T I SBP 0400 SBP 9900	Intersil 6100	Intel 8080, 8085 8748, 8086	Intel 4004 4040
Advanced Micro Devices Am 2901		RCA COSMAC	Motorola 6800, 6802, 6809, 68000	8008
			Fairchild F8	National IMP
			Tecn. MOS MCS 6502	PACE
			Zilog Z80, Z-8, Z8000	

TABLA III.3 CARACTERISTICAS DE LAS TECNOLOGIAS DE FABRICACION

CARACTERISTICAS	BIPOLAR SCHOTTKY	PMOS	NMOS COMP. SIL	HMOS	VMOS	CMOS	SOS	IIL
INTERVALO LOGICO DE VOLTAJE (MAX-MIN)	5	10	5	2 - 5	5	VOLTAJE SUMINIS.	VOLTAJE SUMINIS.	0.6-0.7
PRODUCTO VEL-POT (pJ)	10-100	50-100	5 - 50	0.5-1.0	0.5-1.0	2 - 40	0.5-30	0.2-2.0
RETARDO PROPAGA. DE COMP. (ns)	7	75	4-25	2	2	10-35	4-20	7-50
DISIPA. DE POT. POR COMP.	1-5 mW	1.7 mW	1.0 mW	0.4-1 mW	1.0 mW	1000-1 mW	0.05 mW	1.0 nW-100 uW
CORRIENTE POR COMP.	0.2 a 2 mA	50 uA a 1.0 mA	50 uA a 1.0 mA	0.2 a 1.0 mA	0.5 a 1.0 mA	nA	nA	1 nA a 1.0 mA
VOLT. DE POLARIZA.	5 VOLTS	-5 a -15 VOLTS	5 a 15 VOLTS	2 a 5 VOLTS	5 VOLTS	3 a 18 VOLTS	3 a 18 VOLTS	1 a 15 VOLTS
AREA POR COMPUERTA	20	10	5	2 a 3	2 a 3	10 a 30	15	5
COMPUERTA POR mm2	20 a 40	70 a 120	100 a 180	250	250	40 a 90	100 a 200	75 a 150
COMENTARIOS	Alta vel. Baja densidad Tec. del Proc. Bipolar Slice	Buen rendimiento, baja vel. buena densidad.	Alto rendimiento mayor densidad y vel. que PMOS. Tecnología dominante en microprocesadores.	NMOS reducido, con mayor funcionamiento, Alta densidad, bajo costo. Una de la tecnología dominante de los 80	Proceso más complejo que NMOS. Alta velocidad y densidad.	Operación estática de baja potencia, estruct. de canal N y P complementario. Un amplio rango de operación sobre variaciones de temp. y volt. Menor densidad.	Mismas caract. que CMOS. Mayor vel. y dens.	Elimina las resistencias de carga discretas. Tiene las capacidades de alta vel. de los bipolares y la alta densidad de MOS.

COMENTARIOS A LA TABLA III.3

La tecnología usada en la fabricación de microprocesadores, determina directa o indirectamente su precio, funcionamiento, capacidades, tamaño y aplicación.

Un resumen de las características importantes de las mejores tecnologías de fabricación discutidas en este capítulo, estan dadas en la tabla III.3. Los valores en esta tabla son valores promedio y cambiarán de acuerdo a las variaciones de las dimensiones debidas a improvisaciones en los procesos.

La característica Producto Velocidad-Potencia, es el producto del tiempo de retardo de propagación típica y la disipación de potencia promedio de la compuerta. Las unidades del producto velocidad-potencia son picojoules (pJ) cuando el retardo de propagación está en nanosegundos y la disipación de potencia en miliwatts. A menor valor del producto velocidad-potencia, mayor será la velocidad y menor la potencia de operación del dispositivo.

C A P I T U L O I V

SELECCION Y COMPARACION DEL MICROPROCESADOR

IV.1 INTRODUCCION

Debido a que el microprocesador es el elemento central en un sistema de microprocomputadora (ver Fig. IV.1) sus características tienen un efecto en el diseño de los circuitos e interfases. Es decir, dependiendo de qué tan compleja sea su filosofía de diseño, se determina cuantos dispositivos de soporte (familia) requerirá este microprocesador. Por ejemplo, una microcomputadora construída en un solo circuito integrado con RAM, ROM, reloj e interfases serie, requerirá de un menor número de dispositivos de soporte que un simple microprocesador.

Todos los microprocesadores tienen una orientación a ciertas tareas específicas, por ejemplo, en una aplicación donde se necesita una extensa transferencia de datos, se recomendaría un procesador orientado a E/S (tal es el caso del F8), el cual reduciría notablemente la complejidad de las interfases; por otro lado si lo que se necesita, es una gran capacidad de procesamiento de datos, se recomendaría un procesador orientado a esta finalidad (Z8000 o 68000), donde algunos de ellos ya procesan en una sola instrucción multiplicaciones o aritmética de punto flotante.

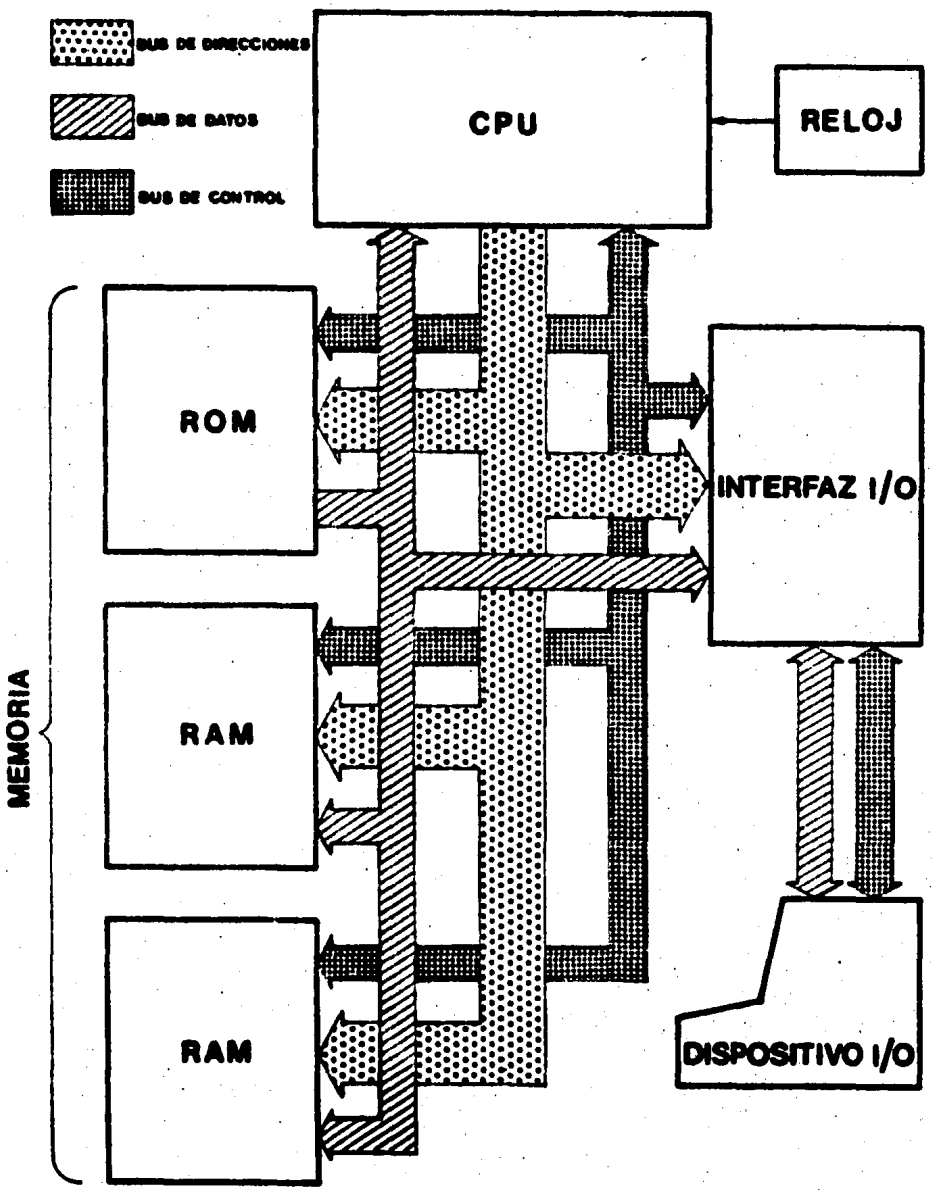


FIG. IV.1 SISTEMA MICROCOMPUTADORA

La selección del microprocesador también afecta el consumo de energía. Un microprocesador y su familia con tecnología bipolar, tendrán una gran velocidad de procesamiento pero también un gran consumo de energía. Sin embargo, un sistema basado en tecnología CMOS reduciran considerablemente este consumo de energía, así como la velocidad de procesamiento.

Por otro lado, la selección del microprocesador afecta grandemente la complejidad y tamaño de las tarjetas y conectores. Un procesador de 4 bits require la mitad de líneas que un procesador de 8 bits, su encapsulado es menor y también el número de conexiones.

El costo del sistema se ve afectado por la elección del microprocesador, sin embargo, si este es caro puede reemplazar un buen número de componentes en las interfases, circuitos impresos y conexiones E/S que reducen el costo total del sistema.

De ahora en adelante se hará referencia al sistema microcomputadora como sistema o como microcomputadora alternativamente. Debemos entender que las tres ascepciones significan lo mismo: el bloque "computadora digital" del modelo de control de malla cerrada de las Figs. II.2.2 y II.2.4.

IV.2 EVALUACION DE LOS REQUERIMIENTOS DEL SISTEMA

El mejor inicio para determinar el microprocesador adecuado para un sistema es evaluar los requerimientos del mismo. Estos requerimientos deben considerarse y pesarse las veces que sea necesario, hasta llegar a la decisión correcta.

IV.2.1 Consideración de la aplicación o tarea del sistema.

Se debe considerar primeramente la tarea que el sistema va a efectuar, debido a que existen microprocesadores que fueron diseñados para procesamiento de datos y otros enfocados como controladores. El uso de un microprocesador orientado al procesamiento de datos en una aplicación de control, puede incrementar la complejidad de la interfaz de control y desaprovechar grandemente su poder computacional. Este sistema desbalenceado si trabajaría, pero sería un sistema más complejo, tomaría más espacio físico y consumiría más energía y además, sería un sistema menos confiable que otro con un microprocesador orientado al control.

IV.2.2 Confiabilidad

Los microprocesadores y microcomputadoras son dispositivos que contienen una gran cantidad de flip-flops y celdas de memoria, lo cual quiere decir que son dispositivos con estados. Si ocurriera una alteración de la energía de suministro (picos o ausencia momentánea) dichos dispositivos podrían cambiar sus estados o salirse del programa control, causando daños (quizá graves) a la tarea que estuvieran efectuando.

Para evitar lo anterior, suelen agregarse al sistema, reguladores de voltaje y/o "no-breaks". Los primeros mantienen la energía de suministro en un nivel de voltaje constante y los segundos, son capaces de suministrar energía por sí mismos cuando el suministro se ha ausentado.

IV.2.3 Disponibilidad del sistema a la expansión

La microcomputadora debe ser capaz de ejecutar una tarea en un tiempo dado. Es decir, cuando se escogió el microprocesador, debió preverse un margen adecuado para expansión. Un sistema basado en un microprocesador es algunas veces menos versátil en expansión de procesamiento que uno implementado con componentes discretos. Si un microprocesador es puesto en sus límites computacionales y existe la necesidad de aumentar este poder, existen dos alternativas a seguir: una, aumentar la velocidad del software, si este no está optimizado, y otra, cambiar a un microprocesador más potente. En cambio, será más sencillo expander un sistema construido con componentes discretos.

IV.2.4 Componentes básicas del sistema

1) El microprocesador mismo y cualesquiera componentes adicionales de soporte, tales como el reloj, el cristal, etc.

2) La memoria. Que debe incluir tanto ROM (para programas) como RAM (para los datos).

3) Los cicuitos integrados de interfaz E/S. Que puede ser un UART si se necesita una conversión de serie a paralelo, o un PIO si se necesita una interfaz paralelo.

Además de estas componentes básicas, usualmente se necesitan circuitos de soporte. Estos son escencialmente "retenes" (latches) y "manejadores" (drivers). Los retenes se usan siempre que sea necesario conservar o retener información para transferirse, son usados típicamente para interconectar dispositivos externos al microprocesador.

Los manejadores se usan cuando a la salida del microprocesador se conectan muchos dispositivos, más de los que pueda manejar como carga. Un ejemplo de estos son los "buffers".

IV.2.5 Otras consideraciones

Evaluar la carga que tendrá el sistema, es decir, si el sistema será capaz de soportar las necesidades actuales y futuras o en todo caso, si el sistema no quedará sobrado aún con cargas futuras.

Todas las características físicas deben considerarse en el proceso de evaluación del sistema. El consumo de energía, la disipación térmica y si han sido especificados otros fabricantes (second source), deben considerarse. Una vez que el microprocesador candidato se ha elegido, se pueden construir y desarrollar diseños de hardware, programas fuentes, programas objeto y un prototipo del sistema.

La evaluación adecuada de los requerimientos del sistema harán un poco difícil el proceso en cuestión, pero el resultado será un sistema bien balanceado y de costo óptimo.

IV.3 CARACTERISTICAS IMPORTANTES DE LOS MICROPROCESADORES

Con la selección del microprocesador, se verán afectadas las características generales, y su entendimiento dará un mejor criterio de evaluación.

IV.3.1 Propósito del microprocesador

Generalmente en catálogos no se encuentra esta característica, siendo esta probablemente, la más importante en la selección del microprocesador para un sistema. La mayoría de los microprocesadores son dispositivos de propósito general. Sin embargo, tienen características propias de su diseño, que los hacen adaptarse mejor a ciertas aplicaciones.

Los dos mayores propósitos son Procesamiento Electrónico de Datos (PED) y Control. Estos propósitos pueden evaluarse mejor si se consideran algunas características tales como el tamaño de palabra, el conjunto de instrucciones y el soporte tanto en Hardware como en Software.

Un tamaño de palabra de 4 bits para el manejo de caracteres aritméticos y caracteres ASCII, es difícil de implementar, más aún, para representar un pequeño número como el 23754, es necesaria una cuádruple precisión aritmética. Sin embargo, haría un manejo adecuado para aplicaciones de control. Las microcomputadoras con un tamaño de palabra de 8 bits o más, indican una orientación PED. El LSI-11, TMS9900 y el 68000, son tres microprocesadores de 16 bits, los cuales tienen un gran uso en aplicaciones PED.

El conjunto de instrucciones da una idea del propósito del microprocesador. Por ejemplo, un conjunto de instrucciones que no permita corrimientos (shifts) y "complemento a dos" aritméticos, no es una buena selección para tareas PED, tal es el caso del 8080 de Intel, el cual no maneja tampoco números con signo y tiene deficiencias en los saltos condicionales (branchs). Esto revela su orientación al control.

El soporte tanto en hardware como en software, también da una idea del propósito del microprocesador. Por ejemplo, el 8086 de Intel un microprocesador orientado a PED, con un gran soporte de hardware y software ambos del tipo PED, muestra así su propósito.

IV.3.2 Tamaño de palabra del microprocesador

El tamaño de palabra de un microprocesador, definido como el número de líneas del bus de datos, tiene un gran efecto en la complejidad y capacidad del sistema.

Las instrucciones y los datos se almacenan en memoria con el mismo tamaño de la palabra del microprocesador. La ventaja de un microprocesador con un gran tamaño de palabra, es que éste puede manejar un gran rango de valores, proporcionando una mayor precisión aritmética. Otra ventaja, es que su conjunto de instrucciones es mucho mayor comparado con un microprocesador de tamaño de palabra menor, dando como resultado un mayor número de líneas en la memoria, en el bus de datos, en los conectores y generalmente en las interfases conectadas al bus. Es decir, no es lo mismo 4K de memoria para un microprocesador de 16 bits, que para uno de 8 bits, el cual usa la mitad de memoria que aquél.

Como una consecuencia de lo anterior se puede señalar, que un microprocesador de 4 bits fue diseñado generalmente, para aplicaciones de control. Los de 8 bits pueden ser utilizados para PED o control, pero en la mayoría de los casos son diseñados para propósitos generales.

La aritmética de doble precisión es bastante eficiente en estos dispositivos, y una precisión de 16 bits es más adecuada para trabajos PED. Los microprocesadores de 16 bits son usados exclusivamente para procesamiento de datos en los cuales se requiere más que simples funciones de control.

IV.3.3 Bit-Slice

Los procesadores Bit-Slice, no son microprocesadores en sí, más bien son dispositivos LSI puestos en cascada con el objeto de armar una ALU con sus registros. La Fig. IV.3.3 muestra esto.

La popularidad de estos dispositivos se debe a su alta velocidad (es la tecnología bipolar LSI más rápida), además, los procesadores Bit-Slice más usados son los de 4 bits; esto por razones térmicas. Como se sabe, los circuitos bipolares LSI disipan gran cantidad de energía y un procesador de 8 o 16 bits generaría demasiado calor para un solo encapsulado.

Los microprocesadores Bit-Slice se usan por lo general, para aplicaciones de grandes procesamientos y ejecuciones. Debido a su versatilidad, los sistemas construidos con ellos pueden emular efectivamente computadoras comunes. Siendo con esto, ampliamente utilizados para la construcción de minicomputadoras.

IV.3.4 Velocidad de procesamiento

La velocidad de procesamiento es la relación con la cual el microprocesador ejecuta el programa de aplicación, y esto depende de tres especificaciones básicas: la velocidad del reloj en el microprocesador, el número de ciclos requeridos para ejecutar una instrucción dada y el repertorio de instrucciones mismo.

Para ver la importancia de estos factores y la manera en que ellos se interrelacionan, se debe tener un entendimiento común de estos términos y de sus contribuciones funcionales.

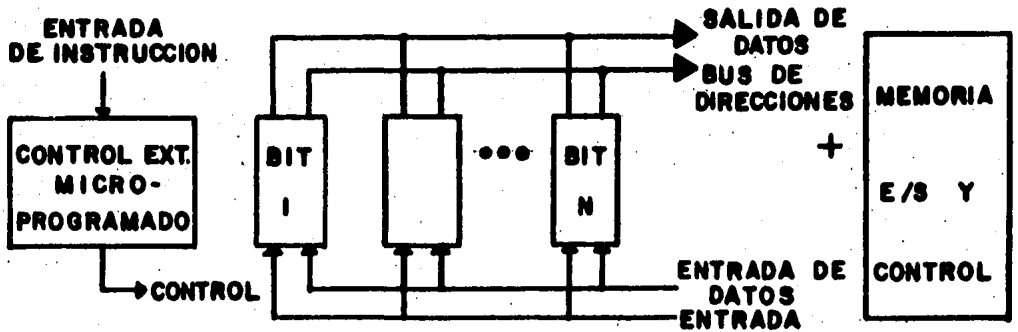


Fig: IV.3.3. PROCESADOR BIT - SLICE.

IV.3.4.1 Velocidad del reloj del procesador

La velocidad del reloj está definida como la frecuencia del reloj de entrada a el microprocesador (el número de pulsos de reloj producidos por segundo). Puesto que el reloj es el gobernador de todos los tiempos de operación dentro de un sistema, un reloj de alta velocidad permite que más operaciones se ejecuten dentro de un periodo dado; pero si este reloj es acoplado con periféricos de baja velocidad, hace que las interfases sean demasiado complejas.

IV.3.4.2 La relación entre adquisición y ejecución

En un microprocesador esta relación puede expresarse en microciclos (el número de ciclos o pasos de operación requeridos para ejecutar una instrucción dada). Un microciclo consiste de uno o más ciclos de reloj. Muchos microprocesadores MOS requieren de muchos microciclos para ejecutar una operación dada. Típicamente un microciclo puede usarse para tomar (fetch) una instrucción, uno o dos más para accesar un dato y varios más para ejecutar la operación de la instrucción adquirida.

El número de microciclos requeridos por una instrucción, se ve afectado por el modo de direccionamiento y por la complejidad de la instrucción. Un simple ADD, por ejemplo, puede tomar 14 microciclos, mientras que un MULTIPLY tomaría 52 en una microcomputadora TMS9900 de 16 bits.

IV.3.4.3 Repertorio de instrucciones

El tipo de instrucciones que un microprocesador puede ejecutar determina que tan adecuadamente realiza una tarea. Las instrucciones deben evaluarse desde la base de lo que pueden hacer, no de cuantas son.

El número de instrucciones que un microprocesador puede ejecutar, puede ser (muchas veces) un número que confunda, debido a que cada fabricante tiene su propia manera de contar instrucciones. Para INTEL, por ejemplo, las instrucciones "mueve inmediatamente a registro" y "mueve inmediatamente a memoria" las consideran como dos instrucciones completamente diferentes, mientras que MOTOROLA considera los direccionamientos "carga acumulador inmediatamente" y "carga acumulador extendido" para la misma instrucción, como dos direccionamientos diferentes. Aunque el 8080 tiene más instrucciones que el 6800, el 6800 tiene más instrucciones reales que el 8080, si se consideran todos los modos de direccionamiento.

El conjunto de instrucciones de un microprocesador debe ser orientado hacia el tipo de procesamiento que esté ejecutando. En una aplicación como "controlador" se le debe dar una atención especial a las instrucciones de E/S. En una aplicación de "procesamiento de datos", las instrucciones de manipulación de datos (corrimientos aritméticos, complemento a dos, saltos aritméticos), deben influir grandemente en la selección.

IV.3.4.4 Determinación de la velocidad del sistema

Una verdadera medición de qué tan rápido se ejecutará un programa por una microcomputadora, es el número de ciclos de reloj necesarios para ejecutarlo multiplicado por la velocidad del reloj.

Algunos microprocesadores utilizan un reloj de alta velocidad y usan muchas instrucciones muy sencillas (como ejemplo están el TMS9900, 8080 y Z80). Otros emplean un reloj de baja velocidad, pero usan pocas instrucciones poderosas (p.e. M6800 y 6500).

Una instrucción en el 8080 que hace exactamente lo mismo en el 6800 (p.e."carga el registro índice extendido"), le toma al primero 16 ciclos de reloj y al segundo 5 ciclos de reloj. El 6800 con un reloj de 1 MHz ejecuta esta instrucción en las dos terceras partes del tiempo que le lleva al 8080 con un reloj de 2 MHz.

Es obvio que de este ejemplo, la velocidad del reloj no es indicativa de una mayor o menor velocidad de procesamiento, a menos que se haga la comparación con el mismo microprocesador, es decir, el 6502 de 1 MHz tiene exactamente la mitad de velocidad de ejecución que el 6502 de 2 MHz.

IV.3.5 Disipación de energía

En sistemas donde la energía es crítica, la disipación de un microprocesador sera lo más importante. Esta se deriva de la tecnología y complejidad del dispositivo y en muchos casos de la velocidad de reloj.

Los microprocesadores con un gran tamaño de palabra, requieren de mayor complejidad para manejar los buses de datos, así que disiparán más energía que aquellos dispositivos de un pequeño tamaño de palabra, aún siendo de la misma tecnología.

Los microprocesadores bipolares de alta velocidad disipan más energía que cualesquiera de los microprocesadores en otra tecnología. Los de mediano consumo de energía son aquellos fabricados en las tecnologías NMOS y PMOS, mientras que los que menos consumo tienen son los CMOS.

La velocidad del reloj del microprocesador es otro causante importante del mayor o menor consumo de energía, es decir, que a menor velocidad de reloj menor será el consumo de energía. Tal es el caso, p.e., del RCA1802, el cual emplea tecnología CMOS; cuando su reloj

está a 1 MHz su consumo va hasta los 60 mW, mientras que cuando su reloj disminuye hasta los 10 KHz, su consumo es solo de 5 mW.

IV.3.6 Manejo de interrupciones

Es deseable el manejo de interrupciones por parte del microprocesador seleccionado (por ejemplo el M6800, 6502 o el Z80, el cual maneja hasta 3 niveles de interrupción), ya que a falta de éstas, las técnicas de programación se complican demasiado (ver secciones VIII.3.2.3 y VIII.5.1).

Un método complicado sería, p.e., tener un programa controlador maestro que estuviera interrogando a cada uno de los dispositivos conectados, bajo una estricta regla de tiempos, es decir, a cada dispositivo se le asignaría previamente un tiempo de consulta dependiendo de su aplicación (ver sección VIII.5.1).

IV.3.7 Capacidades de acceso directo a memoria

El acceso directo a memoria DMA, es el nombre dado a la operación en la cual un dispositivo (no el procesador) puede accesar (leer o escribir) directamente a la memoria, tomando temporalmente el control de las líneas de direcciones y datos.

Esta característica es extraordinaria si se hacen transferencias de grandes bloques de datos, evitandole al microprocesador ejecutar un programa de transferencia de datos. Es decir, el DMA es mucho más rápido que el programa ejecutado por el microprocesador.

Si el sistema que se pretende diseñar requiere muchos accesos a grandes velocidades por parte de dispositivos externos, se hace indispensable seleccionar un procesador con capacidades de DMA.

IV.3.8 Algunas otras características

IV.3.8.1 Ajuste decimal aritmético. El procesador debe contar con una instrucción DAA (Decimal Arithmetic Adjust), o similar.

IV.3.8.2 Costo. El costo a ser considerado, deberá incluir también el costo del sistema.

IV.3.8.3 Soporte en software. Es deseable que el procesador escogido tenga un amplio soporte en software. La disponibilidad de editores, ensambladores y lenguajes de alto nivel, ayudarán a un rápido desarrollo del sistema escogido. El nivel de soporte en software es menos crítico para pequeños microprocesadores tipo control, ya que ellos ejecutan programas de control pequeños.

IV.3.8.4 Filosofía de arquitectura. Básicamente dos tipos de microprocesadores se han desarrollado: microprocesadores orientados al manejo de registros (8086, Z8000, RCA1800) y microprocesadores orientados al manejo de memoria (6800, 6502, 9900). Estos son muy útiles cuando se trabaja con grandes bases de datos en memoria.

IV.4 ALGUNOS EJEMPLOS DE MICROPROCESADORES

Hasta este punto, las características de los microprocesadores han sido descritas en términos generales. En esta sección se mencionarán las características propias de algunos microprocesadores en especial, además de describirlos y evaluarlos.

IV.4.1 El microprocesador 8080

El INTEL 8080 fué el primer microprocesador en ganar una gran aceptación en el campo de las microcomputadoras, ayudando de hecho, a crear este campo. Fué el microprocesador más usado, por haber sido el primero en llegar al mercado. Sin embargo esta situación cambió rápidamente por productos superiores como son el 8085, el Z80, el 6502 y el 6800 y últimamente por microprocesadores de 16 bits como el 68000, el Z8000, el 8086, etc.

El conjunto de instrucciones del 8080 refleja un propósito enfocado al control. Está provisto con instrucciones de transferencia de datos e instrucciones de control de entrada/salida (por ejemplo IN y OUT). Cuenta también con instrucciones para simplificar el manejo de datos como saltos condicionales, llamadas a subrutinas y regreso de subrutinas (con instrucciones de salto con paridad par o impar). En la Fig. IV.4.1 se muestra el microprocesador 8080 con algunos dispositivos de soporte.

IV.4.1.1 Características

El 8080 tiene una arquitectura orientada al manejo de registros, contiene seis registros de 8 bits que pueden usarse individualmente o en pares para operaciones de 8 y 16 bits. Cuenta con un acumulador que actúa como un registro primario de trabajo.

El 8080 es también capaz de hacer operaciones de pila (stack). Un apuntador de pila (stack pointer) controla las inserciones a ésta, debido a que las direcciones de regreso de subrutinas son automáticamente guardadas en la pila cuando existe una llamada a subrutina, la anidación de subrutinas está limitada únicamente por la memoria RAM disponible.

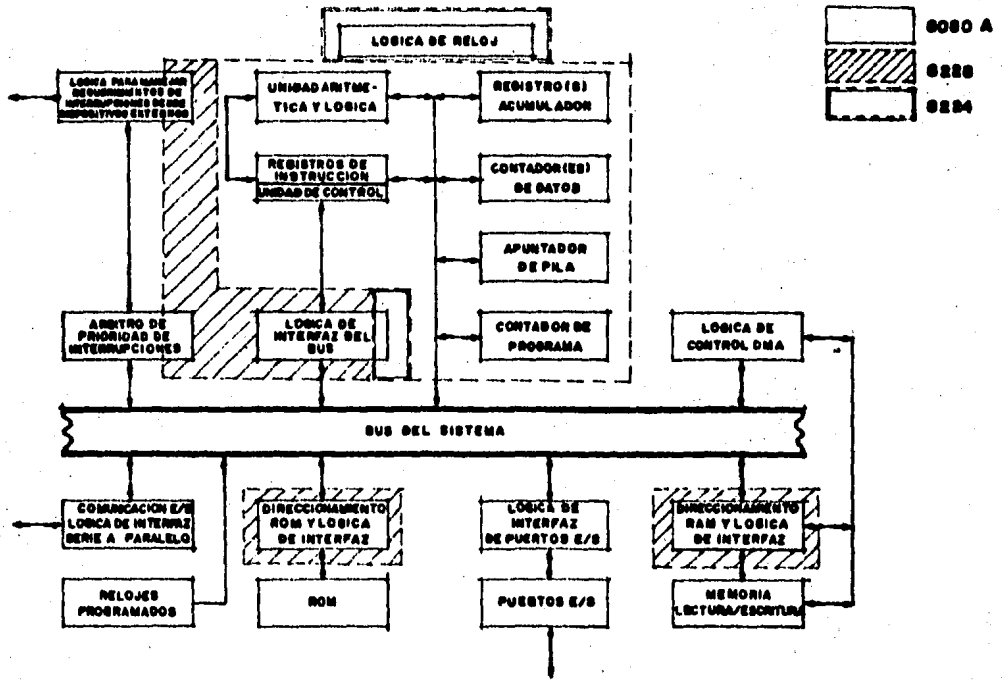


FIG.IV.3.1 . . MICROPROCESADOR 8080 Y ALGUNOS DISPOSITIVOS DE SOPORTE

IV.4.1.2 Dispositivos de soporte

El 8080 es el microprocesador que tiene la más amplia variedad de dispositivos de soporte y se le conoce como el microprocesador de 3 circuitos integrados (Fig. IV.4.1) por que se usa rutinariamente con el generador de reloj 8224 y con el manejador de Bus bidireccional 8228 que además, provee señales de control tales como reconocimiento de interrupciones, lectura y escritura en memoria y otras. Además de estos dispositivos mencionados, el 8080 dispone de la unidad de control de prioridad de interrupciones 8259 que maneja 8 peticiones de interrupciones externas.

Otro de los dispositivos de soporte que puede usar el 8080, es el dispositivo de control de acceso directo a memoria (DMA) 8257, que permite que los datos sean transferidos entre la memoria y circuitos externos sin pasar por el microprocesador. Existen muchos más dispositivos de soporte para el 8080 que no se mencionan aquí. Para mayor información sobre estos, se recomienda consultar el manual "An Introduction to Microcomputers, Vol. III" de Adam Osborne y Gerry Kane.

IV.4.1.3 Instrucciones

El 8080 cuenta aproximadamente con 100 instrucciones, estas varían de 1 a 3 bytes en tamaño, dependiendo del modo de direccionamiento de la instrucción. El conjunto de instrucciones se divide en siete grupos primarios:

- Instrucciones de transferencia de datos
- Instrucciones de control
- Instrucciones aritméticas

- Instrucciones de pila

- Instrucciones lógicas

- Instrucciones de decremento/incremento.

Cuenta además con cuatro modos de direccionamiento:

- Modo directo. El cual permite cargar directamente o almacenar el acumulador o el registro par H-L de la dirección especificada en dos bytes.

- Modo inmediato. Permite cargar cualquier registro sencillo o par con un dato.

- Modo implícito. Permite transferencias de registro a registro.

- Modo indexado. Permite que los contenidos de los registros pares DE, HL o BC, sean usados como apuntadores de 16 bits para los datos que estan siendo cargados o almacenados en memoria.

IV.4.2 El microprocesador 6800

El M6800 es otro de los microprocesadores ampliamente usado debido a que tiene características de diseño que lo hacen adecuado en aplicaciones tanto de control, como de PED.

El conjunto de instrucciones refleja que el M6800 fué diseñado como un microprocesador de propósito general, ya que provee poderosas instrucciones aritméticas y de comparación. En la Fig. IV.4.2 se muestra el microprocesador 6800 con algunos dispositivos de soporte.

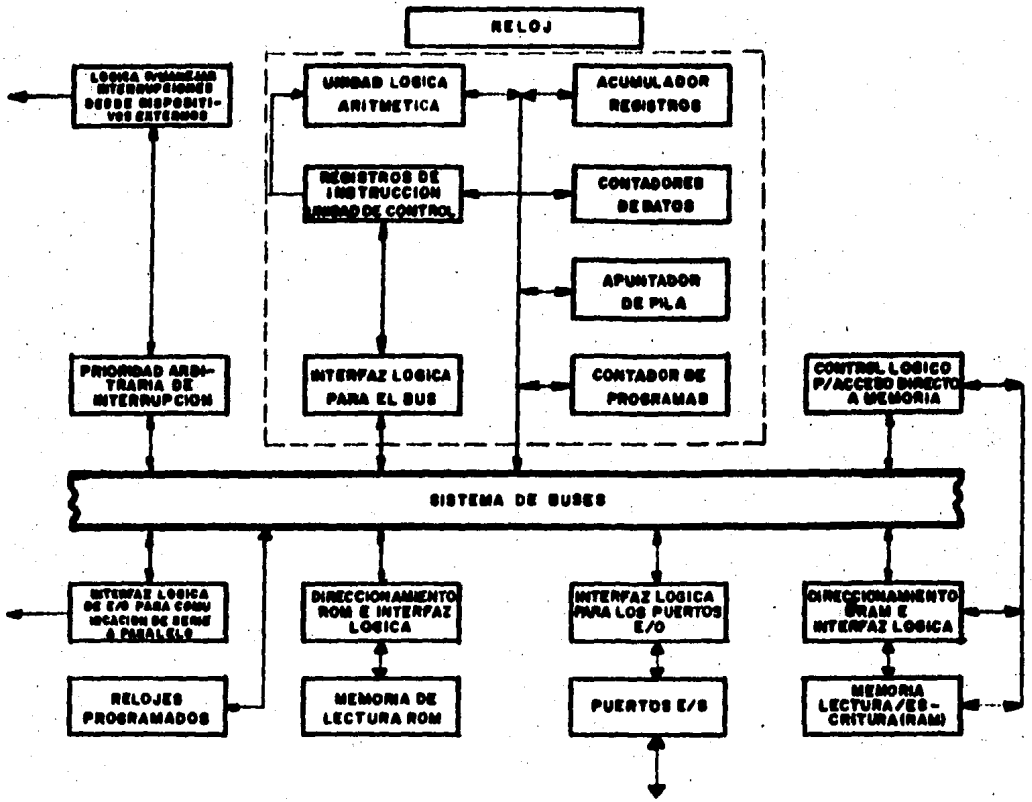


FIG. IV. 3. 2 .. MICROPROCESADOR 6800 Y ALGUNOS DISPOSITIVOS DE SOPORTE

IV.4.2.1 Características

El 6800 tiene una arquitectura orientada al manejo de memoria y su filosofía es usar un reloj de baja velocidad con muchas acciones por ciclo. La mayoría de las instrucciones del 6800 se ejecutan en 2, 3 o 4 ciclos de reloj (contra 8 o 9 para el 8080). Los datos pueden manipularse con dos acumuladores de 8 bits y disponiendo de un registro índice de 16 bits para manejo de direcciones, también posee un apuntador de pila de 16 bits que retiene la última entrada/primera salida de los datos que han sido almacenados en memoria RAM (apunta a la entrada más reciente a la pila) simplificando las llamadas a subrutinas.

El 6800 no provee instrucciones de E/S o canales independientes de E/S, por lo que los registros de los dispositivos son vistos por el microprocesador como localidades de memoria (mapeado en memoria), con lo cual, el procesador requiere pocas líneas de control simplificando considerablemente la programación de E/S. También dispone de la capacidad DMA.

El hecho de que muchas de las instrucciones sean poderosas y se ejecuten en tan pocos ciclos de reloj, hacen al 6800 adecuado para aplicaciones en tiempo real en las cuales la velocidad de ejecución es muy importante.

IV.4.2.2 Dispositivos de soporte

Los dispositivos de soporte básicos que usa el 6800 son: el generador de reloj 6871 o 6875; el PIA 6820 o 6821, el cual es un dispositivo de E/S programable de propósito general; el ACIA 6850, que provee una comunicación en serie y asíncrona de E/S.

Existen otros dispositivos de soporte que no se mencionan aquí y que pueden consultarse en el manual "An Introduction to Microcomputers Vol III" de Adam Osborne y Gerry Kane.

IV.4.2.3 Instrucciones

El 6800 cuenta con 72 instrucciones básicas que pueden usarse en diferentes modos de direccionamiento para ahorrar tiempo de ejecución y espacio de memoria.

Los modos de direccionamiento son seis y se mencionan a continuación:

- Direccionamiento extendido. Permite al microprocesador referirse a cualquier localidad de memoria comprendida entre los 64 K Bytes que puede accesar directamente. Las instrucciones con este direccionamiento constan de 3 bytes.

- Direccionamiento directo. Permite accesar cualquier localidad de memoria comprendida en los primeros 256 bytes (página cero). Las instrucciones con este direccionamiento constan de 2 bytes.

- Direccionamiento indexado. Permite accesar cualquier localidad de memoria comprendida entre el valor del registro índice y 255 localidades próximas. Las instrucciones con este direccionamiento constan de 2 bytes.

- Direccionamiento relativo. Se usa solamente para las instrucciones de saltos condicionados e incondicionados y salto a subrutina. Las instrucciones con este direccionamiento constan de 2 bytes.

- Direcccionamiento inmediato. El operando de la instrucción podrá tener 1 byte para instrucciones referidas a los acumuladores o dos bytes cuando se refieran al registro índice o al apuntador de pila. Las instrucciones con este direccionamiento constan de 2 o 3 bytes.

- Direcccionamiento inherente. Para las instrucciones que no necesitan incluir un operando para definir la operación a realizar. Las instrucciones con este direccionamiento constan de 1 byte.

IV.4.3 El microprocesador Z80

El microprocesador Z80 fué diseñado primero como un microprocesador 8080 mejorado y segundo para hacerlo atractivo al gran número de diseñadores que usaban ya el INTEL 8080.

Contiene más del doble de registros internos de aquél, aparte de manejar dos registros índice independientes para con esto agrandar las capacidades de direccionamiento.

Sus características de diseño lo catalogan como un verdadero microprocesador orientado a PED, sin embargo se debe mencionar que por ser un microprocesador 8080 mejorado, conserva todas las características de control de éste, haciendolo también un excelente controlador.

El Z80 ha ganado una gran aceptación tanto en aplicaciones computacionales o PED y control, como resultado de su naturaleza dual.

IV.4.3.1 Características

El Z80 tiene un conjunto de instrucciones más versátil que el 8080, debido a que las instrucciones de éste son un subconjunto de aquél. Otra característica importante, es que incluye algunas de las propiedades del Motorola 6800, como son instrucciones de direccionamiento indexado y saltos relativos de 2 bytes, capacidades de interrupción mascarables y no mascarables y requiere de una sola fuente de +5 volts.

Finalmente, incluye también propiedades poderosas que no se encuentran ni en el 8080, ni en el M6800, como son instrucciones de transferencias de bloques y busquedas de bloques en memoria, instrucciones de manejo de bits, rotación de digitos, una estructura de vectores de interrupciones con prioridad, entrada de reloj de una sola fase y circuitos simplificados de inicialización (reset).

El Z80 es un procesador con una arquitectura orientada al manejo de registros, contiene 18 registros de 8 bits y 4 de 16 bits (ver Fig. IV.4.3). También contiene 2 acumuladores y un registro de banderas.

IV.4.3.2 Dispositivos de soporte

Debido al gran parecido de las señales de interfase con el 8080 (en donde puede verse que la combinación de señales del Z80 generan los equivalentes en 8080), casi todos los dispositivos de soporte de éste pueden ser usados con el Z80, con algunas excepciones como son el 8259 (unidad de control de prioridades de interrupciones) y el TMS5501 (dispositivo de funciones múltiples).

Como dispositivos de soporte básicos, se tiene el Z80 PIO (interfase de E/S en paralelo), el cual cuenta con la capacidad de manejar interrupciones con características tales como definir las condiciones con las cuales iniciar una interrupción, un manejador de

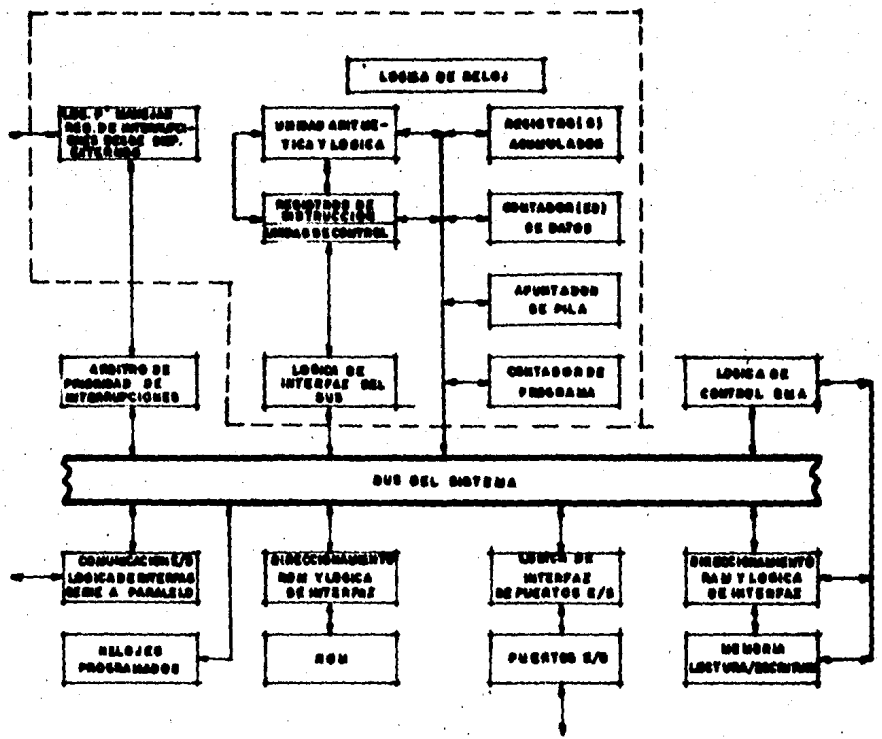


FIG.IV.4 . 3 MICROPROCESADOR Z 80 Y ALGUNOS DISPOSITIVOS DE SOPORTE.

prioridades de interrupción y respuesta por vectores de interrupción.

Otro dispositivo es el Z80 CTC (circuito programable de tiempos), el cual es un dispositivo programable que contiene 4 conjuntos de lógica de tiempos, donde cada conjunto puede programarse independientemente como un generador de intervalos de tiempo o como un contador de eventos externos.

Existen además otros dispositivos como el Z80 DMA, los cuales pueden consultarse a mayor detalle en el manual "An Introduction to Microcomputers" Vol III de Adam Osborne y Gerry Kane.

IV.4.3.3 Instrucciones

El Z80 cuenta con 158 instrucciones, las cuales varían de 1 a 5 bytes de acuerdo al modo de direccionamiento empleado. Su conjunto de instrucciones puede englobarse en 5 categorías, a saber:

- Instrucciones de transferencia de datos
- Instrucciones de procesamiento de datos
- Instrucciones de preguntas y saltos
- Instrucciones de entrada/salida
- Instrucciones de control

y en 8 modos de direccionamiento:

- Modo implícito (o implicado o de registro). Permite operaciones exclusivamente entre registros. Estas instrucciones constan de 1 y 2 bytes.

- Modo inmediato. Permite la carga de registros de 8 bits con constantes de 8 bits. Estas instrucciones constan de 2 bytes.

- Modo extendido. Permite cargar o almacenar en direcciones de memoria. Estas instrucciones constan de 3 o 4 bytes.

- Modo directo. Permite cargar o almacenar en direcciones de memoria de la página cero. Estas instrucciones constan de 3 o 4 bytes.

- Modo relativo. Permite instrucciones de salto o brincos que requieren 8 bits de código de operación, más 16 bits de dirección al cual dará el salto.

- Modo indexado. Permite el acceso de elementos de un bloque ó de una tabla sucesivamente. El principio de éste modo es que la instrucción especifica el registro índice y un desplazamiento, éste se suma al contenido del registro dando como resultado la dirección final. Estas instrucciones constan de 3 bytes.

- Modo indirecto. En este modo los registros pares de 16 bits BC, DE y HL pueden ser usados como direcciones de memoria, en donde se encuentra la parte baja de la dirección final, la siguiente localidad representa la parte alta de la dirección final. Estas instrucciones constan de 1 o 2 bytes.

- Modo página-cero. Este modo es exclusivo de la instrucción CALL, la cual requiere de 1 solo byte.

IV.4.4 El microprocesador 6502

La familia 6500 es descendiente directa de la tecnología Motorola 6800, sin embargo a pesar de ser diseñado posteriormante a éste tienen arquitectura e instrucciones similares, ver Fig. IV.4.4. Por otro lado, aún cuando conservan muchos códigos de operación idénticos, un programa en el M6800 no puede correr en el 6502, al contrario de lo que sucede con el Z-80 y el 8080.

IV.4.4.1 Características

Al igual que el 6800, el 6502 es un microprocesador orientado al PED con características como: aritmética de complemento a dos, así como también capacidades de control. Una de las principales razones de su éxito en el mercado fué haber reducido el costo de los microprocesadores.

El 6502 usa un reloj de baja frecuencia con muchas operaciones por ciclo y la mayoría de sus operaciones se ejecutan en 2 o 3 ciclos de reloj. Es un microprocesador orientado a memoria con un acumulador solamente (a diferencia del 6800 que tiene 2 acumuladores), 2 registros índice de 8 bits cuyos contenidos se usan para formar los desplazamientos indexados. Cuenta con capacidades de manejo de pila, sin embargo, su tamaño está limitado a 256 palabras debido a que el apuntador de pila es de 8 bits.

Una ventaja del 6502 sobre el 6800 y el 8080 es que cuenta con un modo de direccionamiento que ninguno de estos últimos tiene; el direccionamiento indirecto.

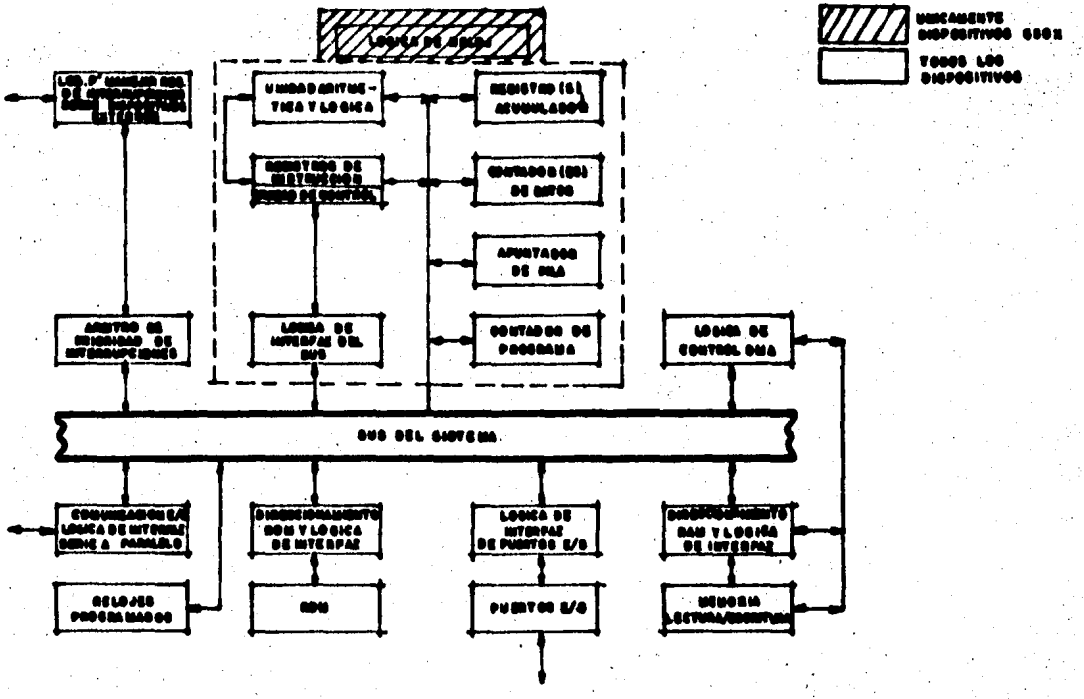


FIG.IV.4.4. MICROPROCESADOR 6502 Y ALGUNOS DISPOSITIVOS DE SOPORTE

IV.4.4.2 Dispositivos de soporte

El 6502 está bien soportado tanto en hardware como en software. Cuenta con relativamente pocos dispositivos de soporte propios, tales como el PIA 6520, el PIA 6522 y dos reguladores de tiempos (Timers), debido a que puede utilizar la mayoria de los dispositivos de soporte de la serie 6800 (utilizando poco o nada de soporte de circuitos externos).

IV.4.4.3 Instrucciones

El 6502 cuenta con un conjunto de 56 instrucciones básicas combinadas con 13 modos de direccionamiento, que son:

- Direccionamiento directo. Permite accesar cualquier localidad de memoria comprendida entre los primeros 256 bytes (página cero). Estas instrucciones constan de 2 bytes.

- Direccionamiento indexado. Permite que el contenido de cualquiera de los 2 registros índice (X o Y) se sume al segundo byte del código objeto para calcular la dirección de memoria. Estas instrucciones constan de 2 bytes.

- Direccionamiento indexado absoluto. A diferencia del anterior, el registro índice (X o Y) se suma al segundo y tercer byte (16 bits) del código objeto de la instrucción. Estas instrucciones constan de 3 bytes.

- Direccionamiento indirecto. Permite obtener indirectamente la dirección efectiva en cualquier parte de la memoria, a través del segundo y tercer byte del código objeto. Es decir, apunta a una dirección de memoria en donde está contenida la dirección efectiva. Estas instrucciones constan de 3 bytes.

- Direccionamiento indirecto pre-indexado. Permite accesar una localidad de memoria en la página cero. El contenido del registro índice X se suma al segundo byte del código objeto, esta suma apunta a una dirección en donde, finalmente se encuentra la dirección efectiva. Estas instrucciones constan de 2 bytes.

- Direccionamiento indirecto post-indexado. Al contenido de la dirección apuntada (en la página cero) por el segundo byte del código objeto, se le suma el contenido del registro índice Y para dar la dirección efectiva. Estas instrucciones constan de 2 bytes.

- Direccionamiento relativo. Permite instrucciones de salto o brincos que requieren 8 bits de código de operación, más 8 bits de dirección al cual dará el salto. Estas instrucciones constan de 2 bytEs.

- Direccionamiento implícito (o implicado o de registro). Permite operaciones exclusivamente entre registros. Estas instrucciones constan de 1 byte.

- Direcccionamiento inmediato. El operando de la instrucción podrá tener 1 byte para instrucciones referidas al acumulador. Las instrucciones con este direccionamiento constan de 2 bytes.

- Direccionamiento absoluto. En este direccionamiento el segundo y tercer bytes del código de operación, forman la dirección de la localidad de memoria donde está localizado el dato. Estas instrucciones constan de 3 bytes.

- Direccionamiento acumulador. Permite operaciones exclusivamente sobre el acumulador, como corrimientos o rotaciones. Estas instrucciones constan de 1 byte.

IV.4.5 El microprocesador M6809

Uno de los microprocesadores de mayor aceptación en aplicaciones de control y en diseños de sistemas con lenguajes de alto nivel estructurados es el Motorola 6809.

Su diseño supera a su antecesor el M6800 y a cualquier otro microprocesador de 8 bits. Cuenta con un mayor número de modos de direccionamientos, así como un mayor número de instrucciones, en donde resaltan capacidades aritméticas de 16 bits y multiplicación de 8 por 8 bits. La Fig. IV.4.5 muestra al microprocesador M6809 con algunos dispositivos de soporte.

IV.4.5.1 Características

El M6809 fué diseñado para operar en 8 y 16 bits, en donde sus nuevos registros (con respecto al M6800) le permiten grandes capacidades de control de procesos y capacidades de direccionamiento.

Sus instrucciones de salto condicional cubren todo el espacio de memoria, permite aplicar modernas técnicas de programación tales como posición independiente de programas, reentrancia y programación modular.

El M6809 cuenta con el más completo conjunto de modos de direccionamientos disponible en microprocesadores de 8 bits. Sus características tanto en software como en hardware, lo hacen un microprocesador ideal para la implementación de lenguajes de alto nivel o aplicaciones de control.

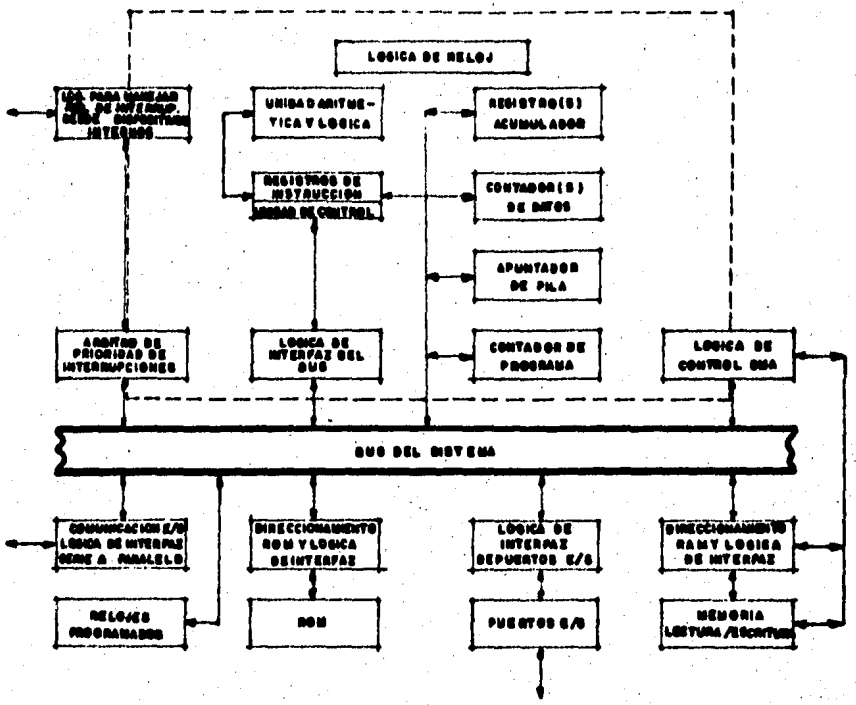


FIG. IV.4.5. MICROPROCESADOR 6809 Y ALGUNOS DISPOSITIVOS DE SOPORTE.

Dentro de las nuevas instrucciones del M6809 cabe resaltar dos de ellas, la primera es su direccionamiento de autoincremento a través de sus registros índice, ya que no se requieren actualizaciones a estos en un direccionamiento secuencial y la segunda, es la instrucción SYNC, la cual permite detener al microprocesador hasta que una interrupción externa sea generada, es decir, permite una sincronización de eventos del mundo real (ver sección VIII.2.2.3, Sincronización), y no solo eso, en un sistema de multiprocesamiento se puede utilizar para sincronizar muchos procesadores.

IV.4.5.2 Dispositivos de soporte

El M6809 usa los mismos métodos de manejo de E/S mapeados a memoria que el M6800, además puede utilizar todos los dispositivos de soporte de éste.

IV.4.5.3 Instrucciones

El M6809 cuenta con 59 instrucciones básicas, las cuales utilizan 10 modos de direccionamiento fundamentales, dando un total de 1464 instrucciones distintas. Los modos de direccionamiento se mencionan a continuación:

- Direccionamiento inherente o implícito. Permite instrucciones de solo 1 byte, tales como incrementos, decrementos, limpiar, corrimientos (a la derecha o izquierda), complementos, etc., sobre los acumuladores.

- Direccionamiento inmediato. Se refiere a aquellas instrucciones donde el operando sigue inmediatamente después del código de operación, es decir, este tipo de direccionamiento incluye el dato del operando dentro de la instrucción, tales como cargar, sumar, restar, AND, OR,

comparar, etc., sobre los acumuladores y registros índice. Tales instrucciones varían desde 2 bytes hasta 4 bytes, según los registros involucrados (8 o 16 bits).

- Direccionamiento extendido. Este modo se usa para accesos a memoria. Aquí, la dirección de memoria (operando) continúa del código de operación. Este modo de direccionamiento cubre los accesos a todo el espacio de memoria y consta de 3 o 4 bytes.

- Direccionamiento directo. Permite, al igual que el anterior, hacer accesos a memoria, con la diferencia de que la parte alta de memoria se graba previamente en el registro de página directa, es decir, la instrucción está formada por un código de operación y en seguida la parte baja de la dirección de memoria a accesar, logrando con esto cubrir las 256 páginas de memoria. Consta de 2 bytes.

- Direccionamiento de saltos relativo. Permite realizar saltos relativos a la dirección apuntada por el contador de programa, ya sea en un sentido o en otro, con desplazamientos máximos de 1 byte signados. Consta de 2 bytes.

- Direccionamiento relativo al contador de programa. Este modo a diferencia del anterior, permite los saltos relativos a través de todo el espacio de memoria, ayudando con esto a los programadores a escribir programas con una posición independiente completa. Consta de 3 a 5 bytes.

- Direccionamiento indexado desplazamiento cero. Este tipo de direccionamiento indexado permite al registro apuntador, apuntar directamente a la dirección efectiva del operando. En otras palabras, el registro apuntador contiene la

dirección del operando a ser usada por la instrucción, sin cálculo del desplazamiento. Consta de 2 o 3 bytes.

- Direccionamiento indexado desplazamiento constante. Este modo se presenta cuando el desplazamiento es constante en 5, 8 o 16 bits, es decir, en rangos de -16 a +15 bytes, de -128 a +127 bytes o de -32,768 a +32,767 bytes respectivamente. Consta de 2 a 4 bytes.

- Direccionamiento indexado desplazamiento en acumulador. Este modo es semejante al anterior, solo que el desplazamiento se encuentra en los acumuladores (8 o 16 bits). Consta de 2 a 3 bytes.

- Direccionamiento indexado de autoincremento/autodecremento. Este modo es muy útil, ya que elimina la necesidad de incrementar o decrementar el registro índice con instrucciones separadas. Consta de 2 a 3 bytes.

Tabla IV.1.a Comparación tabular entre algunos MICROS de 8 bits.

MICROS	CARACTERISTICAS			
	TECNOLOGIA	VEL. DE EJEC TIP.	NO. DE INSTRUCC	MODOS DE DIRECCIONAMIENTO
8080	NMOS	8 a 9	78	4
6800	NMOS	2 a 4	72	6
Z-80	NMOS	2 a 6	158	8
6502	NMOS	2 a 3	56	13
6809	NMOS	2 a 5	59	10

Tabla IV.1.b Continuación de la tabla (a)

	CARACTERISTICAS			
MICROS	FUENTES	REGISTROS	FRECUENCIA DE RELOJ	COSTO DOLARES
8080	+ 5 V - 5 V +12 V	10	0.5 a 4 MHz	3.95
6800	+ 5 V	6	CD a 2 MHz	2.95
Z-80	+ 5 V	22	5 KHz a 4 MHz	3.95
6502	+ 5 V	6	20 KHz a 2 MHz	4.95
6809	+ 5 V	9	CD a 2 MHz	11.95

Tabla IV.1.c Comparación tabular entre algunos MICROS de 16 bits.

MICROS	CARACTERISTICAS			
	TECNOLOGIA	VEL. DE EJEC TIP.	NO. DE INSTRUCC	MODOS DE DIRECCIONAMIENTO
8086	HMOS CANAL-N	2 a 30	70	30
68000	HMOS CANAL-N	4 a 158	56	14
Z8000	NMOS	3 a 20	105	10

Tabla IV.1.d Continuación de la tabla (c)

	CARACTERISTICAS			
MICROS	FUENTES	REGISTROS	FRECUENCIA DE RELOJ	COSTO DOLARES
8086	+ 5 V	19	4 MHz 5 MHz 8 MHz	24.95
68000	+ 5 V	33	8 MHz	49.95
Z8000	+ 5 V	47	8 MHz	44.95

C A P I T U L O V

FUNDAMENTOS DE CONTROL DIGITAL

V.1 INTRODUCCION

Este capítulo trata brevemente sobre el uso de computadoras digitales para control en tiempo real de sistemas dinámicos (plantas). Pretende además, establecer los principios que se deben considerar para utilizar un sistema de control digital en tiempo real.

El control de sistemas físicos con computadora digital, está siendo cada vez más popular debido, por un lado, a que existen microprocesadores orientados al control que facilitan esta tarea y por otro lado, a la gran flexibilidad de los programas de control y a la toma de decisiones (o capacidad lógica) de estos sistemas digitales, que pueden compartirse con la función de control para satisfacer otros requerimientos de las plantas.

Algunas de las principales aplicaciones de estos sistemas de control se encuentran en servomecanismos, procesos químicos y vehículos que se mueven sobre agua, tierra, aire o espacio. Sin embargo, como se dijo en el párrafo anterior, cada día se encuentran nuevas aplicaciones.

Hablar del tiempo real es un poco confuso, no obstante, como se indicó en el capítulo II, un sistema de tiempo real es aquel que responde "rápidamente" y la "rapidez" depende del sistema que se trate. Actualmente se ha relacionado íntimamente el concepto de tiempo real con el "estado del arte de la tecnología", esto es, qué tanto

hardware de propósito específico se tiene. Sin embargo, algunos sistemas de software se consideran de tiempo real porque fueron diseñados para responder "rápidamente". Así, un sistema de control digital en tiempo real, es la conjugación tanto del hardware como del software de tiempo real, con el objeto de resolver problemas de control de sistemas dinámicos en los que el tiempo es crítico.

V.2 MODELOS DE SISTEMAS DINAMICOS

En términos generales, frente a un sistema o fenómeno físico, el problema al que se enfrentan científicos e ingenieros por igual (cada gremio con su óptica y objetivos diferentes), es el de modelar la dinámica de dicho sistema o fenómeno físico; esto es, contar con herramientas (un modelo) para estudiar la evolución en el tiempo de ciertos aspectos del fenómeno, en presencia de diferentes conjuntos o secuencias de eventos o estímulos que influyen sobre el mismo.

Los modelos que se consideran son representaciones matemáticas de un sistema dado y para obtenerlos (proceso de modelado), se estudian los mecanismos internos que gobiernan el comportamiento del sistema y se infiere un modelo a partir de las leyes y relaciones físicas básicas.

Con el objeto de facilitar el proceso de modelado, se hacen una serie de consideraciones teóricas acerca de la planta; es decir, que la planta sea lineal, invariante en el tiempo, etc.; lo cual consigue que el modelo se aleje un poco del comportamiento real, sin embargo, el criterio de validación (de bondad) de un modelo, depende de qué tan bien explica el fenómeno y de la aplicación específica que se le va a dar. Porejemplo, el modelo de un diodo ideal para un rectificador de CA es bueno, sin embargo, para un demodulador de amplitud de señales de bajo voltaje es pésimo.

El modelo a que se hace referencia es la representación de entrada-salida de un sistema dinámico conocida como " función de transferencia ". Se debe aclarar que existen otras maneras de representar sistemas dinámicos que no se consideran en este capítulo, debido a que se alejan de su objetivo.

V.2.1 Modelos de sistemas contínuos

Un sistema dinámico contínuo, lineal, invariante en el tiempo, de parámetros concentrados y de orden "n", es un sistema gobernado por una ecuación diferencial con coeficientes constantes, de la siguiente forma:

$$y^{(n)}(t) + A_1 y^{(n-1)}(t) + \ldots + A_n y(t) = B_0 u^{(m)}(t) + B_1 u^{(m-1)}(t) + \ldots + B_m u(t) \qquad (V.2.1.1)$$

Esta representación es importante porque existe una gran cantidad de sistemas físicos que pueden modelarse por una ecuación como la anterior.

La función de transferencia de un sistema dinámico contínuo en el tiempo, se define como la relación de la transformada de Laplace de la señal de salida del sistema, a la transformada de Laplace de la señal de entrada. La Fig. V.2.1.1 muestra un sistema dinámico lineal cuya entrada es u(t) y la salida, como respuesta a dicha entrada, es y(t).

Un sistema como el de la Fig. V.2.1.1 podrá modelarse por la función de transferencia, obteniendo la transformada de Laplace tanto de la señal de salida como de la señal de entrada y relacionándolas como indica la ecuación (V.2.1.2) y el modelo será entonces como el que se muestra en la Fig. V.2.1.2.

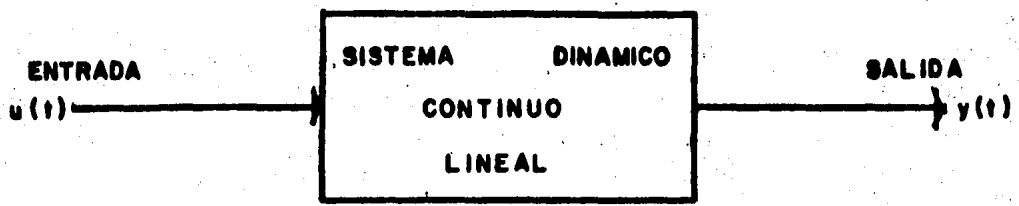


Fig: V.2.1.1 SISTEMA DINAMICO CONTINUO LINEAL

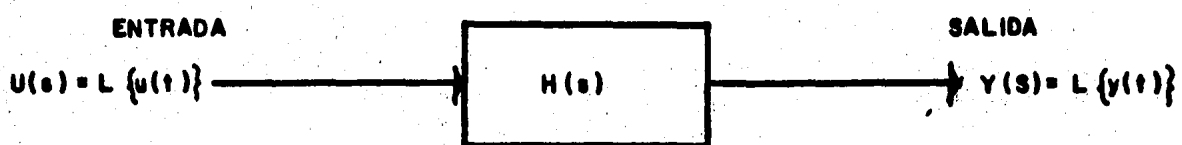


Fig: V.2.1.2 MODELO DE ENTRADA – SALIDA DE UN SISTEMA DINAMICO CONTINUO.

$$H(s) = \frac{L\,y(t)}{L\,u(t)} = \frac{Y(s)}{U(s)} \qquad \text{(V.2.1.2)}$$

cuando $y(t)=u(t)=0$ para $t \leq 0$ y donde L indica la transformada de Laplace.

Tomando la transformada de Laplace en ambos miembros de la ecuación (V.2.1.1), bajo la suposición de que todas las condiciones iniciales son cero, se obtiene:

$$H(S) = \frac{B_0 S^m + B_1 S^{m-1} + \ldots + B_m}{S^n + A_1 S^{n-1} + \ldots + A_n} \qquad \text{(V.2.1.3)}$$

siendo esta última ecuación la función de transferencia (modelo matemático) de un sistema dinámico gobernado por la ecuación (V.2.1.1).

La función de transferencia es una expresión que relaciona la salida y la entrada de un sistema lineal, en términos de los parámetros del sistema y es una propiedad del sistema en sí, independiente de la función de entrada o excitadora. La función de transferencia incluye las unidades necesarias para relacionar la entrada con la salida, sin embargo, no provee ninguna información respecto a la estructura física del sistema;

Usando este concepto se puede representar la dinámica de un sistema por ecuaciones algebráicas en "s". La potencia más alta de "s" en el denominador de la función de transferencia es igual al orden del término de la derivada más alta de la salida. Si la potencia más alta de "s" del denominador de la función de transferencia es "n", se dice que se trata de un sistema de "enésimo" orden.

V.2.2 Modelos de sistemas discretos

Un sistema discreto lineal, invariante en el tiempo y de orden "n", es gobernado por una ecuación en diferencias con coeficientes constantes, de la siguiente forma:

$$y(k) = Bou(k) + B1u(k-1) + \ldots + Bmu(k-m) \\ -A1y(k-1) - A2y(k-2) - \ldots - Any(k-n) \qquad (V.2.2.1)$$

donde y(k) y u(k) son la salida y entrada del sistema en el tiempo "k" (késima iteración) respectivamente. La Fig. V.2.2.1 ilustra, en bloques, un sistema discreto.

Es importante hacer notar que los modelos de sistemas discretos son útiles porque una gran variedad de sistemas físicos (muestreados), pueden modelarse por una ecuación como (V.2.2.1); además, son la clase de modelos que pueden procesarse por computadora digital, es decir, están expresados en la forma adecuada para ser introducidos (programados) en ella.

La función de transferencia de un sistema discreto se define como la relación de la transformada Zeta de la secuencia de salida del sistema, a la transformada Zeta de la secuencia de entrada. La ecuación (V.2.2.2) indica esto:

FIG. V. 2.2.1

REPRESENTACION DE UN SISTEMA DISCRETO

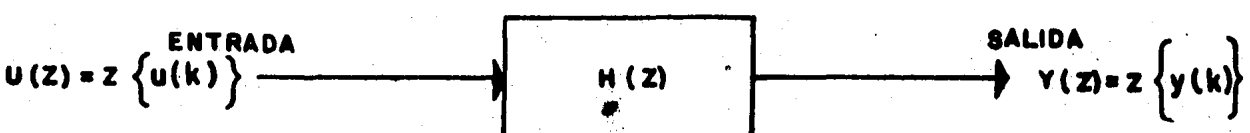


FIG. V. 2.2.2

MODELO DE ENTRADA - SALIDA DE UN SISTEMA DISCRETO

$$H(z) = \frac{Z\,y(k)}{Z\,u(k)} = \frac{Y(z)}{U(z)} \qquad (V.2.2.2)$$

donde Z representa la transformada Zeta.

Un sistema discreto como el de la Fig. V.2.2.1, podrá modelarse por la función de transferencia obteniendo la transormada Zeta tanto de la secuencia de salida como de la secuencia de entrada y relacionándolas como indica la ecuación (V.2.2.2). El modelo será entonces como el que se muestra en la Fig. V.2.2.2.

Tomando la transformada Zeta en ambos miembros de la ecuación (V.2.2.1), bajo la suposición de que todas las condiciones iniciales son cero, se obtiene:

$$Z\ y(k)+A1y(k-1)+...+Any(k-n) = Z\ Bou(k)+B1u(k-1)+...+Bmu(k-m)$$

debido a la linealidad de la transformada Zeta, esto queda

$$Z\ y(k) + A1Z\ y(k-1) + ... + AnZ\ y(k-n) =$$

$$BoZ\ u(k) + B1Z\ u(k-1) + ... + BmZ\ u(k-m)$$

Aplicando ahora la propiedad de corrimiento a la derecha de la transformada Zeta " $Z\ f(k-m) = Z^{-m}\ F(z)$ ", para condiciones iniciales nulas, se llega a:

$$Z\,y(k) + A_1 z^{-1} Z\,y(k) + \ldots + A_n z^{-n} Z\,y(k) =$$

$$B_0 Z\,u(k) + B_1 z^{-1} Z\,u(k) + \ldots + B_m z^{-m} Z\,u(k)$$

debido a que $Y(z) = Z\,y(k)$ y $U(z) = Z\,u(k)$ se obtiene:

$$(1 + A_1 z^{-1} + \ldots + A_n z^{-n})\,Y(z) = (B_0 + B_1 z^{-1} + \ldots + B_m z^{-m})\,U(z)$$

dividiendo ambos lados por el número complejo

$$(1 + A_1 z^{-1} + \ldots + A_n z^{-n})$$

y con ayuda de la ecuación (V.2.2.2), se llega a la deseada función de transferencia dada por:

$$H(z) = \frac{B_0 + B_1 z^{-1} + \ldots + B_m z^{-m}}{1 + A_1 z^{-1} + \ldots + A_n z^{-n}} \qquad \text{(V.2.2.3)}$$

La ventaja de trabajar con un modelo como la función de transferencia (para sistemas contínuos como discretos) es que una vez obtenido el modelo y dada la señal de entrada, es fácil conocer la señal de salida, esto es:

$$Y(z) = H(z)\ U(z) \qquad (V.2.2.4)$$

entonces,

$$Y(k) = Z^{-1}\ H(z)\ U(z)$$

donde Z^{-1} representa la transformada Zeta inversa.

A continuación se procede a dar una interpretación física de la variable "z". Supongase que todos los coeficientes de (V.2.2.3) son cero excepto B1 que se toma como 1. Entonces $H(z) = z^{-1}$. Pero H(z) representa la transformada de (V.2.2.1) y con estos coeficientes la ecuación en diferencias se reduce a:

$$y(k) = u(k-1)$$

El valor presente de la salida, y(k), es igual a la entrada "retardada un período". Así, se ve que la función de transferencia, z^{-1}, es un "retardo" de una unidad de tiempo. Esta situación se presenta en la Fig. V.2.2.3, donde se muestran las relaciones en el tiempo y en la transformada.

Es interesante notar que debido a que las relaciones (V.2.2.1) y (V.2.2.3) están todas compuestas de retardos, pueden expresarse en términos de z^{-1} (retardo unitario). para ilustrar esto, a continuación se presenta un ejemplo:

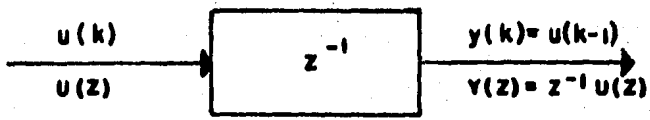


FIG. V. 2.2.3

RETARDO UNITARIO

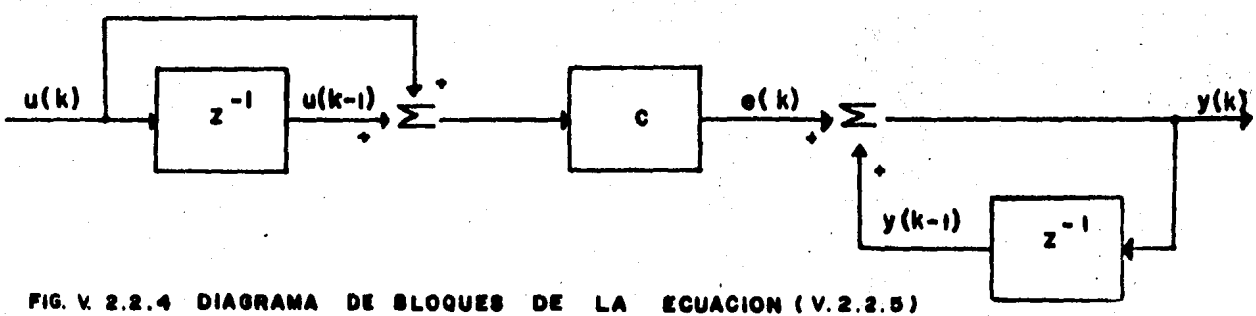


FIG. V. 2.2.4 DIAGRAMA DE BLOQUES DE LA ECUACION (V.2.2.5)

Se considera la siguiente ecuación en diferencias:

$$y(k) = y(k-1) + c\,u(k) + u(k-1) \qquad (V.2.2.5)$$

la cual se ilustrada en la Fig. V.2.2.4, usando la función de transferencia z^{-1} como símbolo para el retardo unitario.

De esta figura, el valor presente de u(k) pasa al primer sumador, donde se suma a su valor previo u(k-1) y la suma se multiplica por c, obteniendo la señal marcada como e(k). Después de esto, existe otro sumador en donde la salida previa, y(k-1), se suma al nuevo valor, e(k), para formar el siguiente valor de y(k).

V.2.2.1 Equivalente discreto de sistemas contínuos con entradas escalonadas

La mayoría de los sistemas dinámicos son contínuos en el tiempo, esto es, tanto la señal de entrada, u, como la de salida, y, son contínuas (analógicas). Introducir una computadora digital en el esquema de control requiere convertir dichas señales a forma digital y para ello es necesario muestrearlas. Además, se presenta la necesidad de informar a la computadora la estructura matemática del sistema dinámico que se pretende controlar. Para hacer esto, se debe convertir la información del modelo a forma digital, lo cual requiere discretizarlo.

Es por lo anterior que a continuación se presenta un método para encontrar el equivalente discreto de un sistema contínuo con función

de transferencia H(s), cuando sus entradas son escalonadas. Se dice que entradas escalonadas porque se parte de un esquema de control como el mostrado en la Fig. V.2.2.1.1, en el cual puede verse claramente que la entrada a la planta, $\hat{u}$, es escalonada. Se requiere entonces, encontrar el equivalente discreto del sistema enmarcado en línea punteada, para obtener un modelo como el mostrado en la Fig. V.2.2.1.2.

Se debe hacer notar que la equivalencia de los modelos ocurrirá solamente en los instantes de muestreo (kT , k= 0,1,2, ...). Entre instantes de muestreo no se puede asegurar la equivalencia. Es decir, la salida del sistema, $\hat{y}(t)$, como respuesta a la entrada $\hat{u}(t)$, es muestreada para obtener la transformada Zeta de dichas muestras. La Fig. V.2.2.1.3 ilustra esto.

Suponga que la señal de entrada, $\hat{u}(t)$, a un sistema contínuo, es escalonada. Es decir, que permanece constante durante un período de muestreo (kT, kT+T , como se ve en la Fig. V.2.2.1.4 . Tambien puede entenderse como una señal contínua a la cual se le están tomando muestras que se retienen durante un período.

Si se tuviera solamente $\hat{u}(0)$, entonces $\hat{u}(t)$ será el escalón de amplitud u(0) seguido T segundos después por un escalón negativo de la misma amplitud. Las muestras de $\hat{y}(t)$, $\hat{y}(k)$, en respuesta al escalón de amplitud u(0), son las muestras de la señal que tiene como transformada de Laplace H(s)/s. Se puede simbolizar la transformada Zeta de estas muestras como:

$$Z\ L^{-1}\ (H(s)/s)$$

donde L^{-1} indica la transformada de Laplace inversa

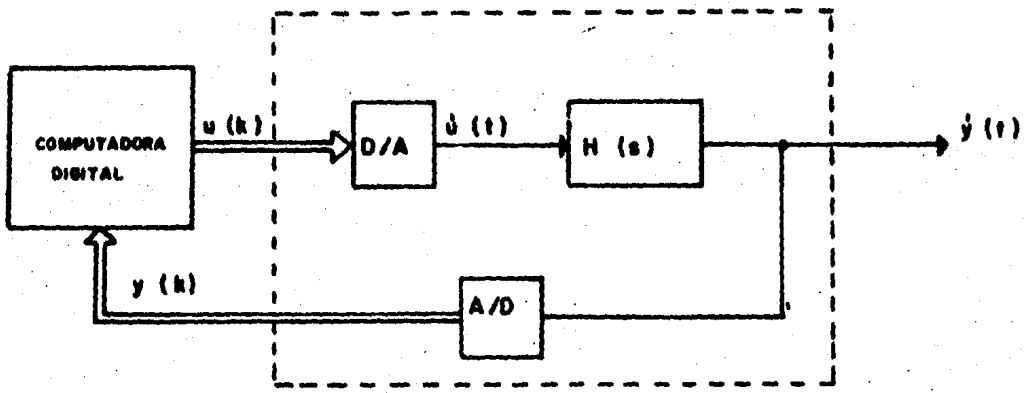


FIG. V.2.2.1.1. ESQUEMA DE CONTROL DIGITAL REALIMENTADO

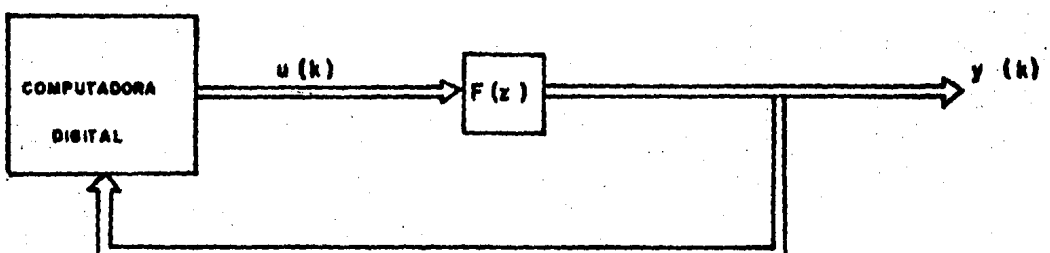


FIG. V.2.2.1.2 EQUIVALENTE DISCRETO DE H (s)

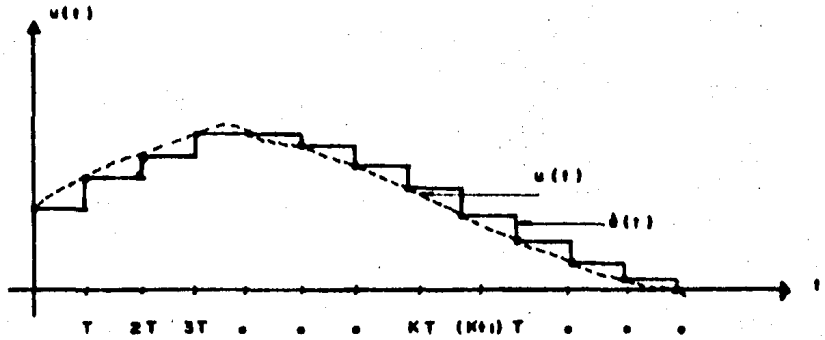


FIG. V.2.2.1.4 SEÑAL ESCALONADA

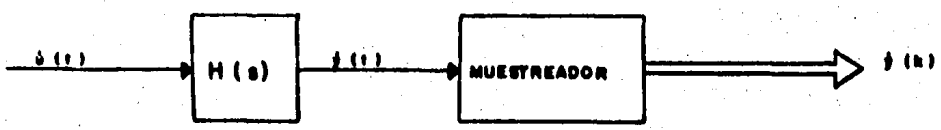


FIG. V.2.2.1.3 SISTEMA MUESTREADO

Las muestras debidas al escalón retardado un período tienen como transformada:

$$z^{-1} Z^{-1} L (H(s)/s)$$

entonces, la contribución total de $\acute{u}(0)$ a la transformada de las muestras $\acute{y}(k)$ es:

$$\acute{u}(0)(1 - z^{-1}) Z^{-1} L (H(s)/s)$$

Si $\acute{u}(1)$ fuera aplicada solamente, los efectos dinámicos deberán ser los mismos que los de $\acute{u}(0)$, pero retrasados un período para dár la contribución:

$$\acute{u}(1)z^{-1} (1 - z^{-1}) Z^{-1} L (H(s)/s)$$

Continuando con este razonamiento y sumando todas las componentes se llega a:

$$\acute{Y}(z) = \acute{u}(k)z^{-k} (1 - z^{-1}) Z^{-1} L (H(s)/s) \qquad (V.2.2.1.1)$$

Se sabe que la transformada Zeta está definida como:

$$D(z) = \sum_{k} D(k) z^{-k} \qquad \text{(V.2.2.1.2)}$$

Sustituyendo (V.2.2.1.2) en (V.2.2.1.1) se obtiene:

$$Y(z) = D(z)(1 - z^{-1}) Z^{-1} L^{-1} (H(s)/s)$$

Así, el equivalente discreto a H(s) está dado por $Y(z)/D(z)$. Esto queda expresado, finalmente, como:

$$F(z) = (1 - z^{-1}) Z^{-1} L^{-1} (H(s)/s) \qquad \text{(V.2.2.1.3)}$$

donde $F(z) = Y(z)/D(z)$ es el equivalente discreto de un sistema contínuo con función de transferencia H(s), cuando su entrada es escalonada.

Es importante consultar el Teorema de Muestreo para una mejor comprensión de lo anterior. Se recomienda el libro "Digital Control Systems" de Kuo, B. de la editorial Prentice Hall.

V.3 ESQUEMAS DE CONTROL

En el capítulo II se habló brevemente de los esquemas de control, mencionando algunas de sus ventajas, desventajas y aplicaciones. Aquí se pretende profundizar un poco más, analizándolos matemáticamente, introduciendo un nuevo elemento en el esquema de control: el controlador G(s), que es el dispositivo encargado de mantener, corregir o limitar el valor de la señal de salida, con respecto a un valor especificado.

V.3.1 Esquema de control de malla abierta

En estos esquemas, la salida no tiene efecto sobre la acción de control, es decir, la salida ni se mide ni se realimenta para ser comparada con la señal de entrada. La Fig. V.3.1 muestra la relación entrada-salida de tal esquema. Para cada entrada de referencia corresponde una condición de operación fijada, así, la exactitud del esquema depende de la calibración.

En la práctica el control de malla abierta solo se puede usar si la relación entre la entrada y la salida es conocida y si además, no hay perturbaciones. Cualquier esquema de control que funciona sobre una base de tiempos es de malla abierta. Por ejemplo, el control del tráfico por señales actuadas en función de tiempos (semáforos), el radio-despertador, etc.

La función de transferencia total está dada por:

$$\frac{Y(s)}{R(s)} = G(s)\,H(s) \qquad (V.3.1)$$

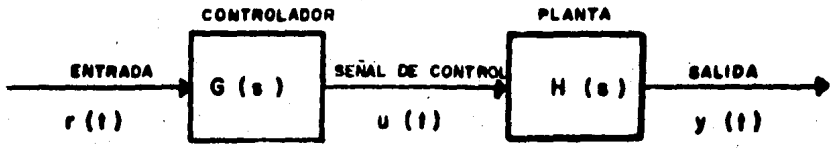


FIG. V. 3.1 ESQUEMA DE CONTROL DE MALLA ABIERTA.

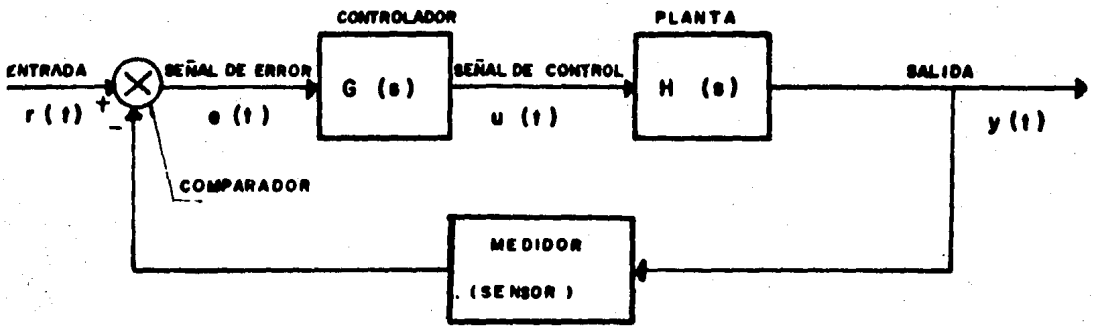


FIG. V. 3. 2. ESQUEMA DE CONTROL DE MALLA CERRADA.

La precisión de un esquema de control de malla abierta depende de lo siguiente :

1. La precisión con la que se disponga o calibre la relación entre la entrada y la salida.

2. La estabilidad de la calibración durante intervalos largos de tiempo.

3. Los efectos de las condiciones ambientales cambiantes.

Un esquema de control de malla abierta se caracteriza por:

1. Precisión moderada.

2. Sensibilidad a las condiciones ambientales (temperatura, vibraciones, golpes, variaciones de voltaje de línea, ruido, envejecimiento y carga).

3. Respuesta lenta a los cambios de la entrada.

Las ventajas del control en malla abierta son:

1. Simplicidad.

2. Bajo costo (para aplicaciones apropiadas de poca precisión).

La ecuación (V.3.1) da la información necesaria para calcular el controlador G(s). Se hace notar que en un esquema de control digital, Fig. II.2.1, el controlador es un programa almacenado en memoria,

mientras que aquí se ha considerado como un dispositivo físico con función de transferencia G(s). En la sección V.4 se verán algunos controladores digitales.

V.3.2 Esquema de control de malla cerrada

En estos esquemas, la señal de salida tiene efecto directo sobre la acción de control, es decir, son esquemas de control realimentado. La Fig V.3.2 muestra la relación entrada-salida de un esquema de control de malla cerrada. La señal de error actuante, que es la diferencia entre la señal de entrada y la realimentación (que puede ser la señal de salida o una función de ésta), entra al controlador G(s) con el objeto de reducir el error y llevar la salida del sistema al valor deseado.

El uso de la realimentación hace al sistema, en su respuesta, relativamente insensible a perturbaciones. Sin embargo, la estabilidad constituye un problema de importancia, por la tendencia a sobrecorregir errores, que pueden producir oscilaciones de amplitud constante o variable.

De la Fig. V.3.2, la función de transferencia total está dada por:

$$\frac{Y(s)}{R(s)} = \frac{G(s)\,H(s)}{1 + G(s)H(s)} \qquad \text{(V.3.2.1)}$$

además,

$$E(s) = R(s) - Y(s) \qquad \text{es el ERROR} \qquad \text{(V.3.2.2)}$$

Para llegar a las ecuaciones anteriores se ha supuesto que el sensor de la Fig. V.3.2 tiene una función de transferencia unitaria.

El controlador debe ajustar automáticamente la respuesta de salida para minimizar la señal de error, ecuación (V.3.2.2). Es decir, debe proporcionar a la planta la entrada de control u(t) adecuada hasta lograr que la salida y(t) permanezca en el valor fijado. La ecuación (V.3.2.1) da la información necesaria para calcular el controlador G(s).

La precisión de un esquema de malla cerrada depende de:

1. La precisión del o de los dispositivos que controlan la salida.

2. Los dispositivos que comparan la salida con la entrada.

3. La sensibilidad y la rapidez de los elementos de control que llevan a cabo la corrección automática del error.

Un esquema de malla cerrada se caracteriza por:

1. Alta precisión.

2. Rapidez de respuesta.

3. Independencia relativa de las condiciones de operación.

Las ventajas del control de malla cerrada son:

1. Alta precisión.

2. Respuesta rápida.

3. Precisión no muy dependiente de las condiciones de operación.

4. Flexibilidad.

Se debe notar que un esquema de control de malla cerrada es más complejo que el de malla abierta y por lo tanto, más caro. Además, se necesitan sensores para efectuar mediciones (situación no trivial).

V.4 CONTROLADORES DIGITALES

El problema de diseñar un sistema de control de seguimiento es lograr que la variable controlada (salida) "y" siga a la señal de entrada "r" tan cercanamente como sea posible. Para lograr este objetivo, el controlador es el elemento en el esquema de control que interviene para aproximar la respuesta y el comportamiento de una planta a un patrón específico. Es decir, el controlador envía la señal de control "u" de tal forma que la respuesta de la planta "y" a dicha señal, sea lo más próxima a la entrada "r".

Con el fin de ilustrar con mayor claridad la función del controlador, se presenta como ejemplo un intercambiador de calor en la Fig. V.4. En éste, la energía de entrada (el vapor en el serpentín) se transfiere al fluido del proceso para mantener alguna o algunas de sus variables en ciertos valores. Generalmente el término proceso se entiende como las funciones y operaciones utilizadas en el tratamiento de materiales para un propósito específico.

Si la variable medida fuese temperatura y estuviese por debajo del valor necesario, entonces la válvula de alimentación deberá abrirse más para permitir que aumente el flujo de vapor de entrada y

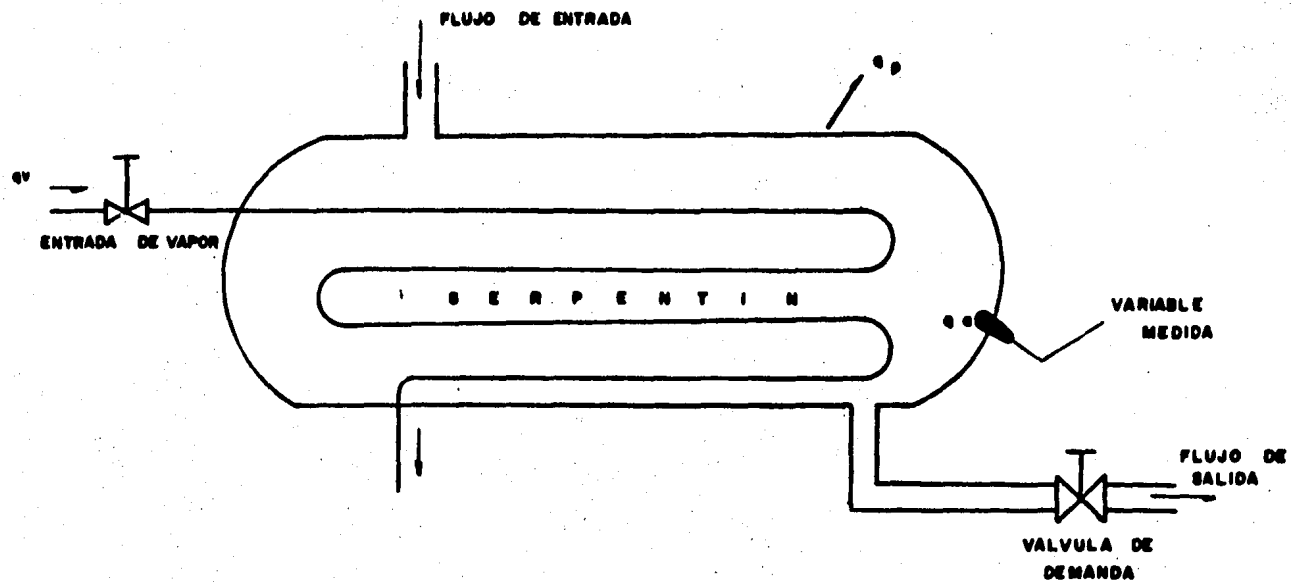


FIG. V.4. INTERCAMBIADOR DE CALOR

aumentar así la temperatura de la variable medida. Si por el contrario, la variable medida estuviese por encima del valor necesario, entonces se cerrará un poco la válvula de alimentación para lograr que disminuya el flujo de vapor de entrada.

A partir del valor de la variable medida y del valor necesario de ésta, el controlador es quien decide qué tanto deberá abrirse o cerrarse la válvula de alimentación de vapor.

Cuando la cantidad de energía de entrada (Qv) iguala a la cantidad de energía de salida (Qa + Qp), se dice que el proceso está en condición de estado estable (en balance) y entonces, las variables tendrán un valor estable. Cualquier variación de una u otra energía modificará la candición de balance, causando un cambio en las variables. De esta manera, se encuentra que la función primordial del controlador consiste en manipular la relación energética (o material) de entrada-salida, con el objeto de mantener las variables dentro de ciertos límites.

En diversos procesos existe una característica inherente e importante que ayuda a limitar las desviaciones de la variable controlada (haciendo más sencilla la labor de contro). Esta característica se llama "autorregulación" y consiste en la habilidad del proceso para balancear sus energías y tender hacia la restauración del equilibrio. El intercambio de calor es un ejemplo de estos procesos dado que si la cantidad de vapor aumenta, entonces la temperatura del fluido aumenta hasta alcanzar un nuevo punto de equilibrio, es decir, la temperatura no se eleva indefinidamente. Contrariamente a esto, la "no autorregulación" indica que el proceso no puede balancearse por sí mismo y por lo tanto, no encuentra un equilibrio natural o estado estable. Los procesos no autorregulados no pueden permanecer largos períodos de tiempo sin la aplicación de un controlador automático. Un proceso de este tipo es un tanque de almacenamiento con flujos de entrada y salida constantes dado que si

los flujos no son exactamente iguales, entonces el tanque tenderá a vaciarse o derramarse, debido a que la diferencia entre ellos determinará que el nivel disminuya o crezca indefinidamente.

Existen una gran variedad de estructuras para controladores que dependen de las características de la planta, el medio ambiente en donde se encuentre y de las señales de entrada y salida. Debido a la gran diversidad de controladores existentes, solamente se mencionan de una manera breve, el controlador de dos posiciones (Encendido/Apagado), el controlador PID (Proporcional Integral y Derivativo) y el controlador de Asignación de polos y ceros.

V.4.1 Controlador de dos posiciones (Encendido/Apagado)

Este es el más simple y económico de todos los controladores existentes, lo cual hace que sea muy utilizado tanto a nivel industrial como doméstico. La salida de éste se rige por la siguiente ecuación:

$$u(t) = K\, e(t) \qquad (V.4.1.1)$$

donde u(t) es la salida del controlador, K es la ganancia del mismo y e(t) es la señal de error o desviación.

La transformada de Laplace de la ecuación (V.4.1.1) es la siguiente:

$$U(s) = K\, E(s) \qquad (V.4.1.2)$$

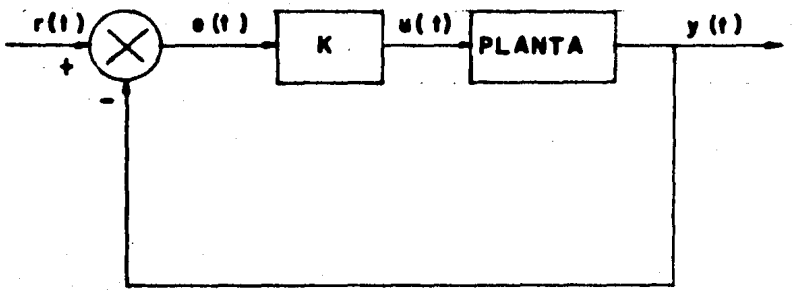


FIG. V.4.I.I ESQUEMA DE CONTROL REALIMENTADO CON UN ICONTROLADOR DE DOS POSICIONES

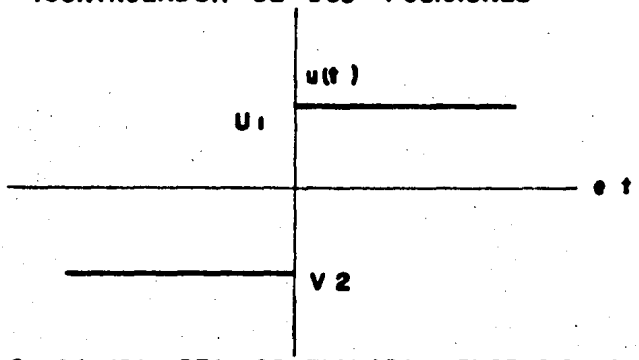


FIG. V.4.I.2 SALIDA DEL CONTROLADOR ENCENDIDO/APAGADO

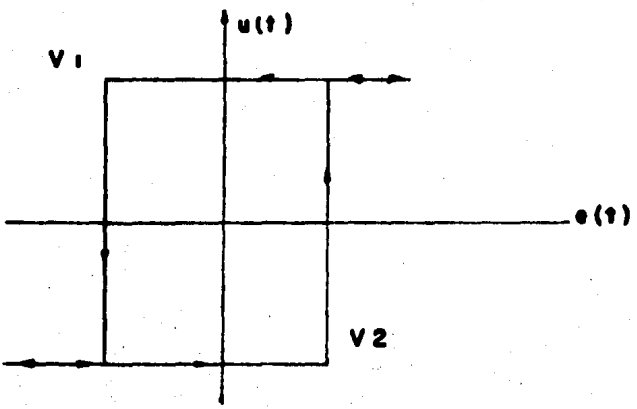


FIG. V.4-I.3 SALIDA DEL CONTROLADOR ENCENDIDO /APAGADO CON APERTURA DIFERENCIAL.

La Fig. V.4.1.1 muestra un esquema de control de malla cerrada donde se incluye este controlador en el lazo directo. De esta Fig., la señal de error e(t) está dada por:

$$e(t) = r(t) - y(t) \qquad (V.4.1.3)$$

El controlador Encendido/Apagado es un amplificador (en malla abierta) con ganancia K muy grande (tiende a infinito), por lo que la salida será máxima o mínima (Abierto o Cerrado) dependiendo del signo aritmético del error, sin importar su magnitud. En la Fig. V.4.1.2 se observa esto.

Se debe aclarar que este controlador provoca un gran número de conmutaciones en el actuador, desgastándolo rápidamente. Para evitar (en gran medida) lo anterior, se utiliza el controlador de dos posiciones con apertura diferencial, en el cual la acción de conmutación ocurre después de que el error de entrada alcanza cierto valor (apertura diferencial), en la Fig. V.4.1.3 se muestra esto.

V.4.2 Controlador PID (Proporcional Integral y Derivativo)

Este controlador presenta los efectos de tres acciones de control: Proporcional, Integral y Derivativa. A continuación se presentan individualmente cada una de estas acciones y, finalmente, en conjunto.

V.4.2.1 Acción Proporcional

El controlador de acción proporcional es aquel cuya salida es proporcional al error y se rige por la ecuación (V.4.1.1), solamente

que éste es un amplificador con ganancia K ajustable (a diferencia del anterior cuya ganancia tiende a infinito). A la ganancia del amplificador se le suele llamar Kp para indicar que es proporcional. La Fig. V.4.2.1 muestra como respondería este controlador a dos señales de error distintas.

Debe notarse que después de que la acción proporcional compensa un cambio en el proceso, las variables se mantienen fuera del punto de ajuste, por lo que se hace necesaria alguna otra acción de control que regrese al proceso a dicho punto.

V.4.2.2 Acción Integral

El controlador integral es aquel que se rige por la siguiente ecuación:

$$u(t) = \frac{K}{Ti} \int e(t)dt \qquad (V.4.2.2)$$

donde Ti es el tiempo de integración.

En la Fig. V.4.2.2.1 se muestra el diagrama de bloques de esta acción, cuya salida varía proporcionalmente a la intagral del error actuante.

La función de la acción integral es repetir la acción proporcional tantas veces por minuto como se le haya programado, hasta regresar a la variable controlada a su punto de ajuste. En realidad hace que la señal de salida del controlador sea la superficie bajo la curva de la señal de error hasta ese momento, por lo que dicha salida

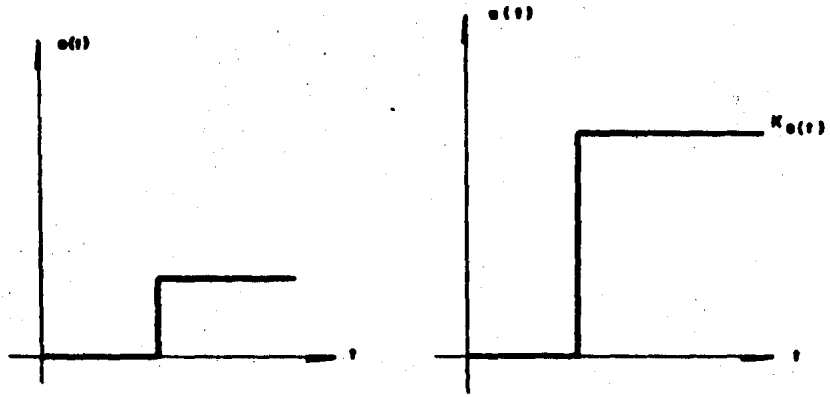
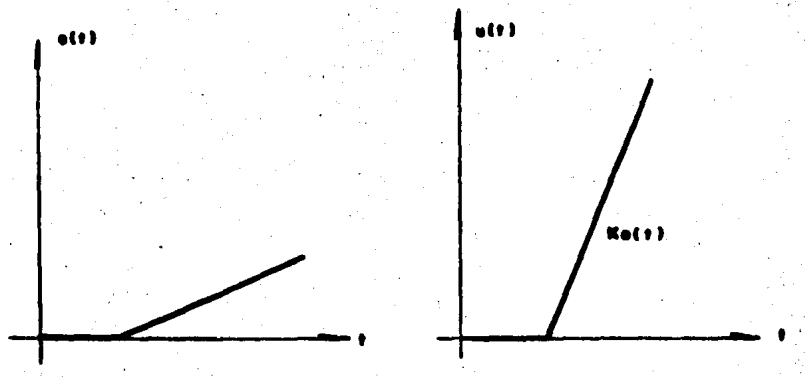


FIG. V.4.2.1 RESPUESTA DEL CONTROLADOR PROPORCIONAL.



puede tener un valor diferente de cero cuando el error es cero, lo cual es imposible en el caso del control proporcional puro.

Es importante hacer notar que la acción integral no reconoce la acción en la cual el proceso se está moviendo, sino que solo detecta si está arriba o abajo de su punto de ajuste. Esto debe tomarse en cuenta debido a que si se usa un alto valor de acción integral, el proceso puede volverse inestable.

V.4.2.3 Acción Derivativa

El controlador de acción derivativa es aquel cuya salida es proporcional a la derivada del error y se rige por la siguiente ecuación:

$$u(t) = KTd \frac{de(t)}{dt} \qquad (V.4.2.3)$$

donde Td es el tiempo derivativo.

Esta acción reconoce la velocidad y dirección en la cual se mueve el proceso y su salida es proporcional a la velocidad de variación del error. De esto puede apreciarse que el controlador derivativo tiene carácter anticipativo, sin embargo no puede anticiparse a una acción que aún no ha ocurrido.

Es importante hacer notar que la realización de un derivador puro es imposible dado que es un sistema no causal, lo cual significa que la salida del sistema en un tiempo dado, depende de la entrada un instante después. Es por esto que el controlador derivativo no es un

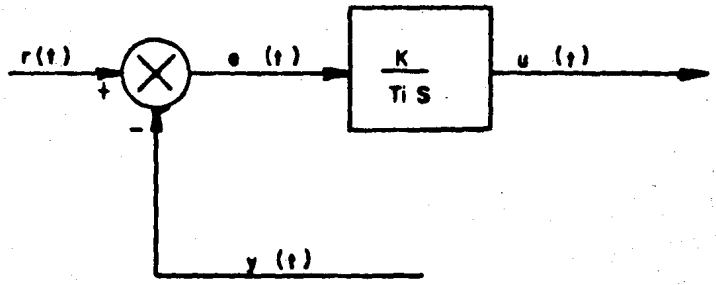


FIG. V. 4.2.2.1 ACCION INTEGRAL

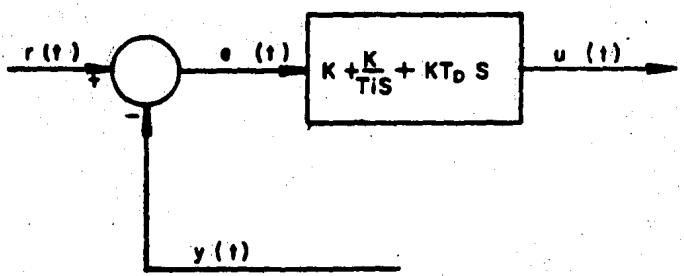


FIG. V. 4.2.4. ACCION PROPORCIONAL INTEGRAL Y DERIVATIVA.

derivador puro sino que se hace cercano a ello, es decir, la función de este controlador se aproxima a la de un derivador.

La desventaja de esta acción es que amplifica las señales de ruido y puede saturar al actuador. Además, no puede existir aisladamente debido a que actúa durante períodos transitorios. Sin embargo, ante una perturbación (de carga) el controlador mide la tendencia de cambio de la variable controlada, para generar una acción correctiva lo suficientemente grande que contrarreste, ya no la desviación que existe en ese momento, sino la desviación que existiría un tiempo Td posterior, si continuara la misma tendencia de variación.

La acción derivativa cesa cuando la variable controlada se mantiene en algún valor, no importando si está fuera del punto de control.

V.4.2.4 Acción Proporcional-Integral-Derivativa

Esta acción representa los efectos de cada una de las tres acciones individuales, presentando las características de todas ellas y se rige por la siguiente ecuación:

$$u(t) = Ke(t) + \frac{K}{Ti}\int e(t)dt + KTd\frac{de(t)}{dt} \qquad (V.4.2.4.1)$$

donde K es la ganancia, Ti el tiempo de integración y Td el tiempo derivativo.

La transformada de Laplace de la ecuación anterior es:

$$U(s) = \left(K + \frac{K}{Tis} + KTds\right) E(s) \qquad (V.4.2.4.2)$$

de esta ecuación se obtiene el diagrama de bloques de la acción PID y se muestra en la Fig. V.4.2.4. Los parámetros ajustables, K, Td y Ti se seleccionan de tal manera que mejoran el funcionamiento de todo el sistema. Es decir, es un controlador de tipo general y se pueden elegir los parámetros para dar una respuesta adecuada.

V.4.3 Discretización del controlador PID analógico

Debido a la experiencia que se tiene sobre este controlador, el controlador PID digital (discreto) se obtiene transformando las ecuaciones de aquél por discretización. Aparte de la experiencia otro factor que motivo el pronto éxito del controlador discreto fue que las reglas de sintonización o ajuste del PID, bien conocidas ya, pueden aplicarse a la versión digital.

La ecuación diferencial del controlador PID es:

$$u(t) = K\, e(t) + 1/Ti \int e(t)dt + Td\, de(t)/dt \qquad (V.4.3.1)$$

con los parámetros:

K ganancia
Ti tiempo de integración
Td tiempo derivativo

Para un período de muestreo To pequeño, la ecuación anterior puede convertirse a una ecuación en diferencias por discretización. El término derivativo se sustituye por una diferencia de primer orden, esto es:

$$\frac{de(t)}{dt} = \frac{Ae(k)}{To} = \frac{e(k) - e(k-1)}{To}$$

con lo cual:

$$Td\ de(t)/dt = Td/To\ \ e(k) - e(k-1) \qquad (V.4.3.2)$$

El término integral de la ecuación (V.4.3.1) puede aproximarse por integración rectangular o trapezoidal. Aplicando el método de integración rectangular se obtiene:

$$1/Ti \int_0^t e(t)dt = To/Ti \sum_{i=0}^{k} e(i-1) \qquad (V.4.3.3)$$

Con las expresiones (V.4.3.2) y (V.4.3.3), la ecuación (V.4.3.1) queda expresada en forma discreta como:

$$u(k) = K\ e(k) + To/Ti \sum_{i=o}^{k} e(i-1) + Td/To\ [e(k) - e(k-1)] \qquad (V.4.3.4)$$

Siendo esta última ecuación un algoritmo de control no recursivo. Para la información de la sumatoria todos los errores e(k) anteriores tienen que ser almacenados. Debido a que este algoritmo produce un cambio total de la variable manipulada u(k), se le llama algoritmo de posición.

Sin embargo, un algoritmo recursivo es más adecuado que el anterior para programarse. Estos se caracterizan por el cálculo de la variable manipulada u(k) presente o actual, basado en el valor previo de la variable manipulada u(k-1) y términos de corrección. Para obtener un algoritmo recursivo se tiene que restar a la ecuación (V.4.3.4) la siguiente ecuación:

$$u(k-1) = K\ e(k-1) + To/Ti \sum_{i=o}^{k-1} e(i-1) + Td/To[e(k-1) - e(k-2)] \qquad (V.4.3.5)$$

para obtener

$$u(k) - u(k-1) = q_0\, e(k) + q_1\, e(k-1) + q_2\, e(k-2) \qquad (V.4.3.6)$$

con parámetros

$$q_0 = K\ (1 + Td/To) \qquad (V.4.3.7)$$

$$q_1 = -K\,(1 + 2Td/To - To/Ti)$$

$$q_2 = K\,(Td/To) \qquad (V.4.3.7)$$

Ahora solamente es necesario calcular el cambio presente (o actual) de la variable manipulada, esto es

$$Au(k) = u(k) - u(k-1)$$

y es por esto que este algoritmo se le llama algoritmo de velocidad.

Aplicando ahora el método de integración trapezoidal a la ecuación (V.4.3.1) se obtiene

$$u(k) = K\,e(k) + To/Ti\,((e(o) - e(k))/2 + \sum_{i=1}^{k-1} e(i)) + Td/To\,(e(k) - e(k-1)) \qquad (V.4.3.8)$$

Despues de restar la correspondiente ecuación para u(k-1), se obtiene otra relación recursiva de la forma

$$u(k) = u(k-1) + q_0\,e(k) + q_1\,e(k-1) + q_2\,e(k-2) \qquad (V.4.3.9)$$

con parámetros:

$$q_0 = K\,(1 + To/2Ti + Td/To)$$

$$q_1 = -K\,(1 + 2Td/To - To/2Ti) \qquad \text{(V.4.3.10)}$$

$$q_2 = K\,(Td/To)$$

Para un período de muestreo pequeño los parámetros q0, q1 y q2 pueden calcularse usando los parámetros K, Ti y Td del controlador PID analógico; aplicando la ecuación (V.4.3.7) o la ecuación (V.4.3.10). No obstante, existen gran cantidad de trabajos publicados acerca de las reglas de sintonización o ajuste de los parámetros del controlador PID; por ejemplo se puede consultar el libro "Digital Control Systems" de Rolf Isermann, de la editorial Springer-Verlag.

V.4.4 Controlador de Asignación de polos y ceros

Cuando se requiere que un sistema de control de malla cerrada se comporte con una dinámica conocida; esto es, que dicha dinámica esté representada (modelada) por una función F, y cuando el modelo de la planta H (estable) se conoce exactamente (tan exactamente como sea posible), entonces la utilización de un controlador de Asignación de polos y ceros resuelve este problema.

Del esquema de control de malla cerrada (Fig. V.3.2), el controlador G(s) debe ser tal, que haga que

$$Y(s)/R(s) = F(s) \qquad \text{(V.4.4.1)}$$

donde F(s) representa el comportamiento deseado.

Lo anterior puede conseguirse si H(s) representa exactamente la dinámica de la planta, y si los ordenes de los polinomios del numerador y del denominador son iguales; además de que sea un sistema de fase mínima. Esto se aclarará a continuación:

La función de transferencia del esquema de control en malla cerrada (Fig. V.3.2) es:

$$\frac{Y(s)}{R(s)} = \frac{G(s)\,H(s)}{1 + G(s)\,H(s)} \qquad (V.4.4.2)$$

igualando esta expresión con (V.4.4.1) y desarrollandola se llaga a:

$$G(s) = \frac{F(s)}{(1 - F(s))\,H(s)} \qquad (V.4.4.3)$$

Si se sustituye (V.4.4.3) en (V.4.4.2) se llega a (V.4.4.1) que era lo deseado; esto es, que el sistema realimentado se comporte como F(s).

Una vez obtenida la función de transferencia del controlador, se puede discretizar conforme a los métodos vistos en secciones anteriores, para obtener de aquí la ecuación en diferencias que represente la dinámica del controlador. Con esta ecuación se procede a realizar el algoritmo de control, programandola en la computadora digital. Se debe recordar que en un sistema de control digital, el controlador es un programa grabado en memoria y que las señales de control (variable manipulada) se obtinen de los puertos de salida de la computadora.

Como se dijo anteriormente, el controlador de Asignación de polos y ceros intentará que el comportamiento del sistema de control realimentado se aproxime a la dinámica F(s); para lo cual la elección

de F(s) es muy importante, es decir, F(s) debe ser tal que la salida del sistema Y(s) sea la deseada. Es común que F(s) sea de segundo orden si H(s) es de primero o segundo orden, dado que en un sistema de segundo orden, los parámetros de diseño están definidos perfectamente (consultar el libro "Ingeniería de Control Moderna" de Katsuhiko Ogata, de la editorial Prentice Hall).

Dado que el problema de cancelación de polos y ceros no está resuelto aun, se debe estudiar a fondo, entonces con estos controladores se trata de cancelar los polos dominantes de H(s). Además se deben estudiar las señales de control y de salida de cada problema en particular para poder entonces elegir una F(s) adecuada al sistema de control realimentado. Por otro lado debe considerarse el orden de H(s) para determinar F(s).

No obstante lo anterior, es necesario mencionar que el uso de este controlador digital presenta algunos inconvenientes. En primer lugar si H(s) no representa exactamente la dinámica de la planta, los polos y ceros de G(s) que pretendan cancelar los de H(s) pueden no hacerlo; por otro lado, los ceros de H(s) son polos de G(s) y si algún cero de la planta estuviese ubicado en el semiplano derecho (sistema de fase no mínima), ahora se convertiría en un polo que haría inestable a G(s), haciendo inestable tambien al sistema realimentado. Así, el diseño de controladores de Asignación de polos y ceros de acuerdo a la ecuación (V.4.4.3), tiene que limitarse a plantas suficientemente amortiguadas y asintóticamente estables, con comportamiento de fase mínima.

V.5 CONTROL DIGITAL EN TIEMPO REAL

Una vez obtenido el controlador adecuado para la planta H, se debe proceder a implementar el esquema de control. Tomando en cuenta que se hará mediante una computadora, se deben elegir los dispositivos

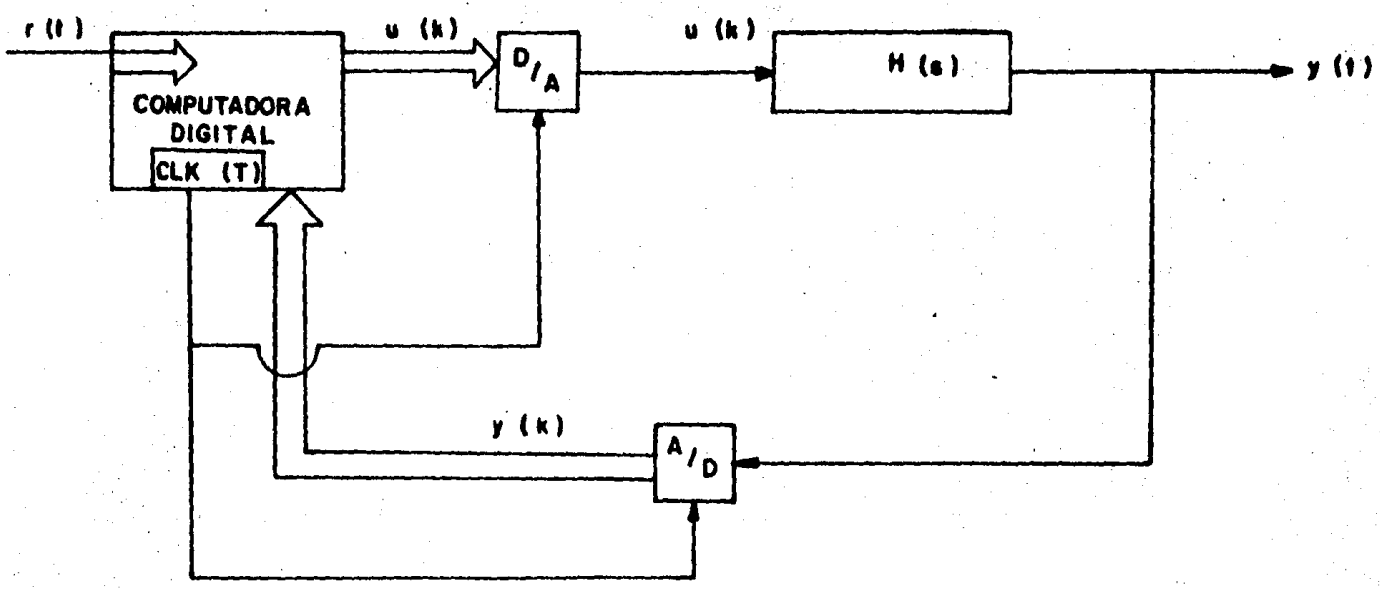


FIG. V.5.1 ESQUEMA REALIMENTADO DE CONTROL DIGITAL.

que ayuden en esta tarea y la elección de estos depende de las señales que se van a manipular; en general serán necesarios los sensores, los convertidores y los actuadores que ejecuten las órdenes del controlador para modificar alguna variable de la planta y conseguir así, los objetivos del control. Se debe señalar que algunos dispositivos requieren de otros para que funcionen como sea necesario; así un convertidor A/D necesitará un dispositivo S/H que le ayude a efectuar la conversión, y un sensor requerirá de otros dispositivos para que traten (acondicionen) la señal entregada por el si así se requiere. Como podrá notarse, el modelo H de la planta y las señales u(t) e y(t) (que dependen de H), son la información básica y necesaria para iniciar el diseño de un esquema de control digital.

De lo visto hasta ahora puede inferirse que cada planta impone el esquema de control, y éste los dispositivos que lo constituyen. No puede hablarse de un esquema ideal a una planta cualquiera, sin haber estudiado suficientemente dicha planta y sus señales. Para diseñar controladores se deben estudiar profundamente, aparte de la planta y sus señales, su ubicación (medio ambiente) para saber si existen señales que interfieran y destruyan las órdenes de control. Sin embargo, se ha dicho que un esquema de control digital en tiempo real, es un esquema realimentado como el que se muestra en la Fig. V.5.1, dado que es necesario sensar frecuentemente la señal de salida de la planta. Este esquema un poco más desglozado aparece en la Fig. V.5.2; debe recordarse que estos son esquemas generales cuya finalidad es ilustrar la estructra de un sistema de control.

En la Fig. V.5.2 puede observarse que dentro del bloque computadora, aparece parte del esquema de control. Esto es ilustrado así, no obstante debe entenderse que es un programa almacenado en memoria, que es quien va a dirigir las acciones de control. Dicho programa consta de las etapas necesarias para cumplir con los objetivos del control y de una manera sencilla se muestra, en la Fig. V.5.3, un procedimiento sistemático para efectuar un control digital

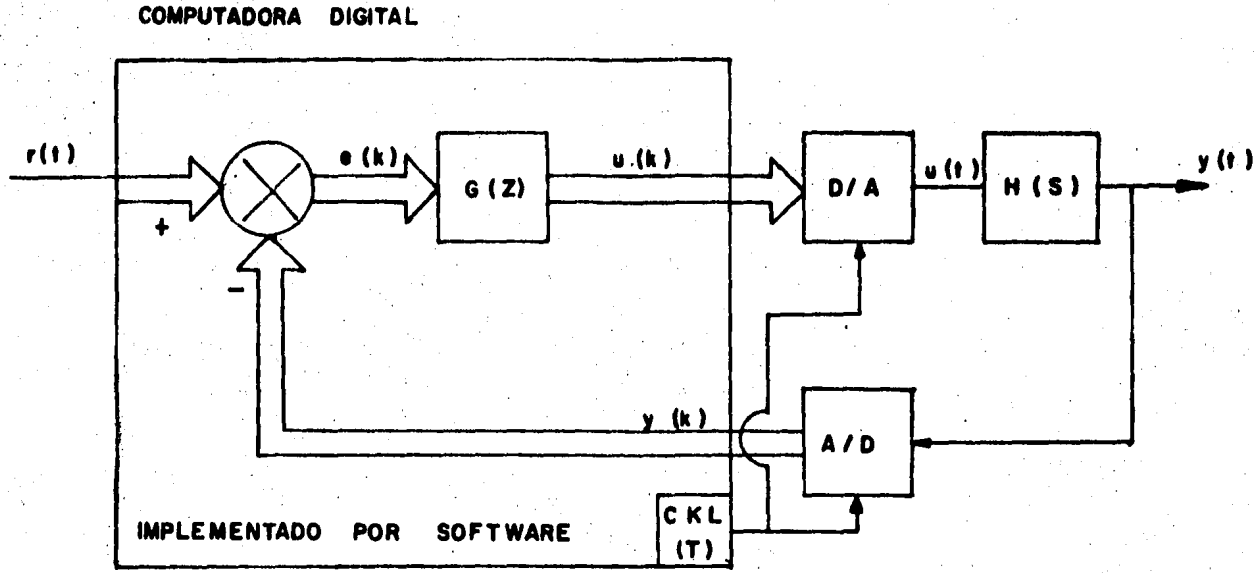


FIG. V.5.2 REPRESENTACION GRAFICA DEL ALGORITMO DE CONTROL.

en tiempo real. Este procedimiento o algoritmo de control contempla los siguientes pasos:

a) Inicializar todas las variables necesarias al algoritmo.

b) Leer la señal de referencia r(t) ya sea de memoria o de algun puerto de entrada.

c) Medir la señal de salida y(t) de algun puerto de entrada. Lo cual implica considerar el tiempo de sensado y conversión, y acondicionamiento si se requiere.

d) Calcular la señal de error e(t) con algun criterio adecuado.

e) Calcular la señal de control u(k) mediante el algoritmo del controlador.

f) Convertir a analógica la señal de control.

g) Esperar una señal (período de muestreo) para tomar otra muestras de las señales r(t) o y(t), para procesarlas nuevamente.

h) Volver a (b).

La manera en que se lleve a cabo la secuencia anterior dependerá de la planta, de sus señales, de los objetivos del control y de los dispositivos que componen el esquema, así como del medio ambiente.

Acerca de la problemática a la que se enfrenta el control digital en tiempo real y en funciñ del sistema, se puede mencionar lo siguiente:

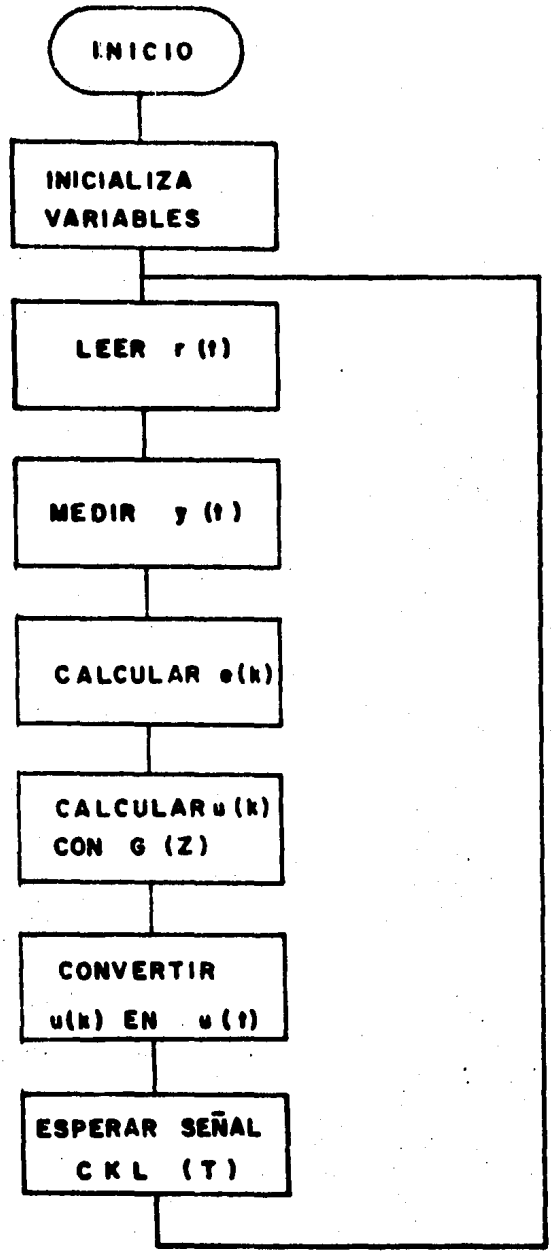


FIG. V.5.3 DIAGRAMA DE FLUJO DE UN ALGORITMO DE CONTROL.

La elección de los sensores. Se debe considerar el tipo de variable a medir, la precisión y linealidad del sensor. Esto se presenta en el capítulo VI.

La elección de los convertidores. Se deben tomar en cuenta factores como velocidad de conversión, precisión o número de dígitos y la linealidad. Sobre esto se habla en el capítulo VII.

La elección de la computadora. Ya que de ésta dependen factores tan importantes como la velocidad de procesamiento del algoritmo de control. Aunado a esto están la capacidad de memoria para almacenar; por un lado el programa en sí y las subrutinas de servicio; y por otro, las variables involucradas en el algoritmo y su evolución, si es necesario. Tambien se deben considerar los lenguajes que acepte la computadora y, finalmente, el costo del sistema. Todo esto está cubierto en los capítulos IV y VIII. Este último capítulo es de especial importancia para elaborar el algoritmo de control.

La elección del período de muestreo. El cual depende de la planta y de los objetivos del control; y afecta en el cálculo de la señal de control.

Hasta aquí se han expuesto las consideraciones más importantes y un sencillo procedimiento para iniciar el estudio de un sistema de control digital. Es importante recalcar que cada planta exige un estudio detallado y profundo para poder llegar al controlador adecuado.

C A P I T U L O V I

TRANSDUCTORES

VI.1 INTRODUCCION

Actualmente los sistemas de medición y control se usan en la industria con el propósito de mejorar eficiencias y reducir costos. En ello, el transductor es el eslabón escencial para cada proceso.

Se define al transductor como "un dispositivo alimentado por la energía de un sistema y que dona energía, usualmente en otra forma, a un segundo sistema". En este capítulo se tratarán algunos de los transductores que transformen parámetros físicos en magnitudes eléctricas, es decir, aquellos que donan señales eléctricas a un sistema de medición y/o control.

VI.2 GENERALIDADES

Del modelo de control de malla cerrada, Fig. II.2.4, se observa que se está sensando la señal de salida para convertirse a digital y después ser alimentada a la computadora. Si esta señal es eléctrica, esto es, voltaje o corriente, no habrá problema para ser retroalimentada ya que como se sabe, el convertidor A/D convierte señales eléctricas analógicas a digitales. Desde luego, antes del convertidor existe el dispositivo S/H que muestrea y retiene una señal eléctrica el tiempo necesario para efectuar la conversión. Sin embargo, si la señal de salida (que interesa controlar) no fuera eléctrica sino de otros tipos, tales como temperatura, presión,

posición, velocidad, intensidad de luz, etc.; ¿cómo hacer para retroalimentarla a la computadora? si bién se sabe que no puede convertirse a una señal eléctrica digital directamente.

Es en este problema, donde los transductores cobran importancia. Ellos son los que van a convertir la señal a la forma adecuada para ser manipulada, esto es, convertirán a eléctrica cualesquiera otros tipos de señales o magnitudes físicas. A este hecho se le denomina transducción (transduction) y así se le usa en este capítulo.

Los transductores deben tener la capacidad de variar algún parámetro eléctrico con algún estímulo físico. Para efectuar la transducción se toman los cambios de resistencia eléctrica, capacitancia, inductancia o alguna combinación de estos, para producir una corriente eléctrica, un voltaje (CA o CD), una frecuencia o una palabra digital que será única para cada valor permisible del estímulo aplicado.

La forma más común de transducción utiliza el cambio de la resistencia óhmica para indicar cambios en el estímulo aplicado. Existen varios materiales que transducen a través de termoresistencia, fotoresistencia, piezoresistencia (deformación) o simplemente por la posición del cursor de un potenciómetro.

Los dipositivos termoresistivos se conocen como termistores. Un termistor cambia su resistencia eléctrica con cambios en la temperatura aplicada.

La resistencia eléctrica de casi cualquier material conductor puede expresarse de la siguiente manera:

$$R = \rho (L / A)$$

donde R es la resistencia eléctrica en ohms, L es la longitud, A es el área de la sección transversal del conductor y ρ es la resistividad propia del material.

La resistividad, siendo una propiedad natural de los materiales, es constante en la mayoría de los conductores, de tal forma que se depende de la cantidad L/A para la transducción. Este cociente puede modificarse deformando el material y en la mayoría de los casos se modifica L, ya sea por compresión o por tensión.

Cuando un material se encuentra bajo tensión, la longitud se incrementará ligeramente y la sección transversal decrecerá, ya que el volúmen total es constante (LxA). Esta acción hace que se incremente el valor de L/A, por tanto la resistencia.

De manera similar, cuando se comprime el material, L disminuye, A aumenta y el factor L/A disminuye, disminuyendo la resistencia. Los transductores que operan en esa forma se denominan Galgas Extensométricas.

Los transductores se clasifican en pasivos (una entrada y una salida) y activos (una entrada, una salida y una excitación), en estos, la excitación pude usarse para proveer un incremento. En el nivel de salida. En los transductores pasivos, toda la energía eléctrica a la salida se deriva de la entrada física, debido a esto tienden a donar baja energía, requiriendo amplificación. Un ejemplo de estos son los termopares o termoacopladores.

El diseñador de interfases debe tomar en cuenta las características de ambos tipos de transductores. En los pasivos, la fórma de respuesta está limitada por la energía disponible en el fenómeno y por la eficiencia de los dispositivos de conversión. En los activos existe un grado de libertad adicional que, en algunas ocasiones, permitirá mejorar las condiciones de salida.

Una vez que el transductor se ha elegido para un determinado trabajo, se deben hacer provisiones para una excitación apropiada y para el acondicionamiento de la señal de salida. La naturaleza del acondicionamiento depende de las características eléctricas del transductor y del destino de la señal.

VI.3 TIPOS DE TRANSDUCTORES

A continuación se mencionarán algunos de los transductores más usados para distintas aplicaciones.

VI.3.1 Transductores Capacitivos

Un capacitor de placas paralelas se hace colocando frente a frente dos superficies conductoras paralelas. Si se hace que las placas se muevan una con respecto a la otra bajo la influencia de un estímulo aplicado, entonces la capacitancia o la reactancia capacitiva puede utilizarse como una propiedad de transducción.

El transductor capacitivo puede utilizarse para controlar la frecuencia de un oscilador. La transducción se hace por discriminación de frecuencias, ya sea por detección de fase o por conteo.

Existen varias formas de transductores capacitivos y estas varían dependiendo de la naturaleza del trabajo a ejecutar. Algunos hacen variar la distancia entre las placas, como los micrófonos capacitivos, algunos otros hacen rotar una placa con respecto a la otra, quedando así mayor o menor área cubierta por ambas placas.

VI.3.2 Transductores Inductivos

También es posible utilizar inductancias para lograr la transducción. Los inductores pueden utilizarse como elementos reactivos en un puente Wheatstone o para variar la frecuencia de un oscilador LC. Algunos transductores emplean el método del puente y solo pocos usan la técnica del oscilador.

La mayoría de los inductores utilizados en la transducción usan núcleos variables para cambiar la inductancia y son llamados propiamente dispositivos de permeabilidad variable. El estímulo aplicado varía la posición del núcleo cambiando la inductancia de la bobina.

Uno de los transductores inductivos más exitosos es el transformador de voltaje diferencial lineal (LDVT). Este transformador consiste en un devanado primario conectado a una fuente de excitación de CA y dos devanados secundarios (Fig VI.3.1). Los devanados secundarios se conectan cruzados, así las corrientes generadas por el primario se cancelan.

Cuando el núcleo está exactamente dentro de ambos devanados, las dos corrientes del secundario se cancelarán una a otra y el voltaje a la salida será cero. Si el núcleo se desplaza ligeramente, un devanado tendrá mayor inductancia que el otro, evitando que las corrientes se cancelen totalmente; así, el voltaje a la salida será distinto de cero. Además, la fase del voltaje (de CA) a la salida, depende de la dirección del desplazamiento del núcleo. Esta situación indica que se tiene información tanto de la amplitud del desplazamiento como de su dirección.

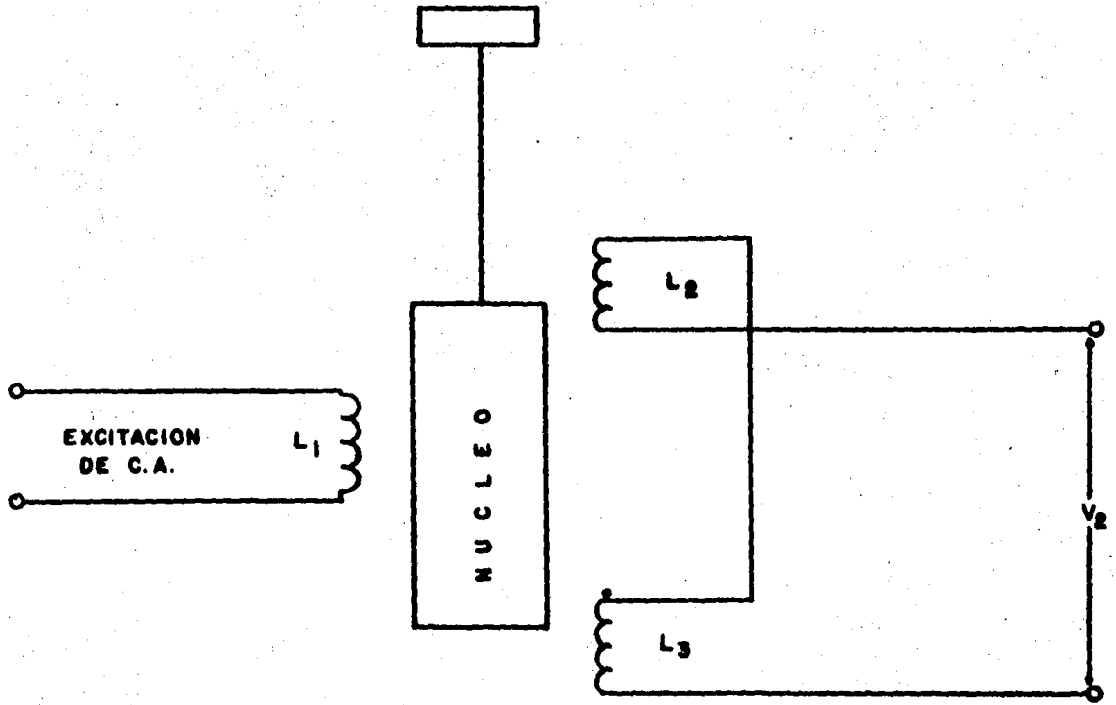


Fig. 3.2. TRANSFORMADOR DE VOLTAJE DIFERENCIAL LINEAL (T.V.D.L.)

VI.3.3 Transductores de Posición

El potenciómetro es el más común de los transductores de posición. Por ejemplo, el potenciómetro de la Fig. VI.3.3.A indica la posición en el eje x (o sea, a lo largo de una línea) del primer cuadrante del plano Cartesiano. Cuando el cursor se encuentra en el orígen está eléctricamente aterrizado y el potencial de salida será cero volts. A medida que el cursor se desplaza por el eje x en dirección positiva, el voltaje de salida aumenta hasta +Vcc, que es el voltaje máximo permisible.

Si se quiere registrar también un desplazamiento en dirección negativa se puede utilizar el dispositivo de la Fig. VI.3.3.B. En este circuito, un extremo del potenciómetro se conecta a Vee(-) en lugar de a tierra y el otro permanece a Vcc(+). Cuando el cursor del potenciómetro está en el orígen, se encuentra en el centro del rango de resistencias donde Vee y Vcc se cancelan mutuamente, a medida que el cursor se desplaza en dirección positiva, Vcc predominará sobre Vee, dando como resultado un voltaje positivo. De igual forma cuando se desplaza hacia la izquierda, predominará entonces Vee dando un voltaje (Vo) de salida negativo.

La forma que tomen los potenciómetros dependerá del tipo de movimiento que se intente medir. Si el movimiento es estrictamente rectilíneo, entonces se utilizará un potenciómetro de desplazamiento (slide). Por otro lado, si el movimiento va a ser circular o angular será necesario un potenciómetro rotatorio. Se puede designar a 0 grados como 0 volts de salida y a Vcc como 359 grados.

El LDVT se puede utilizar también como transductor de desplazamiento. De cualquier forma que sea el transductor de posición, necesitará algún mecanismo para conectar el cursor al dispositivo que será medido.

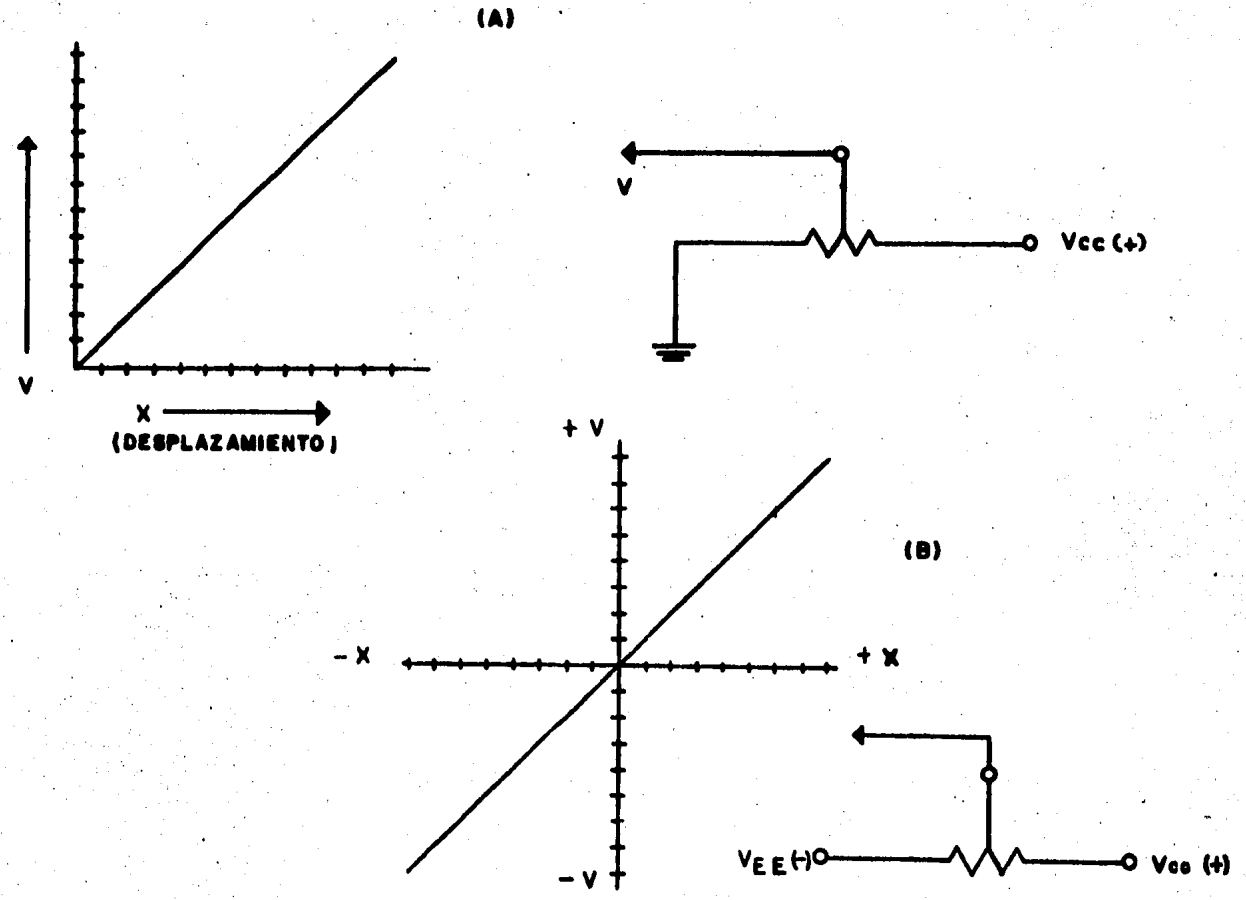


Fig: XI.3.3. TRANSDUCTORES DE DESPLAZAMIENTO (A) UN CUADRANTE B) DOS CUADRANTES.

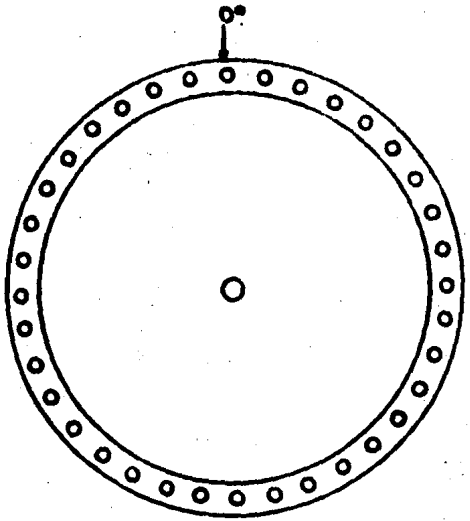
VI.3.4 Sensores de Posición en código digital

Los circuitos optoelectrónicos se pueden utilizar para crear un sensor de posición y son particularmente comunes en la medición de movimiento angular. Las Figs. VI.3.4.1 y VI.3.4.2 son ejemplos de dichos dispositivos.

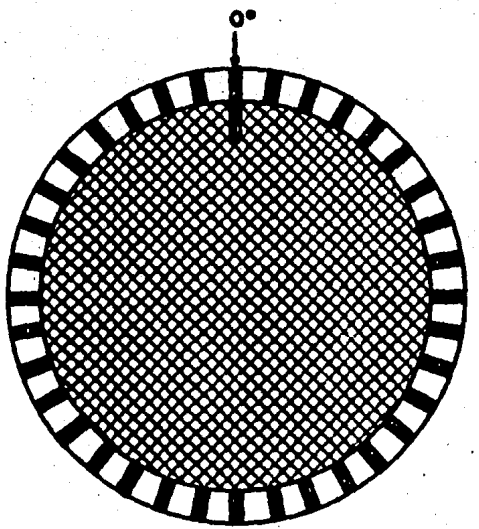
En la Fig. VI.3.4.1 se muestran dos discos codificados distintos, los cuales se montan concéntricamente al eje del dispositivo o por medio de engranes si así se requiere. La información de la posición la da el anillo externo del disco, mientras que la señal de sincronización la da el anillo interno para indicar 0/360 grados (número de vueltas) de giro.

En el disco A la información es codificada en agujeros, mientras que en el B es por medio de zonas claras u obscuras, técnica que permite codificar el disco por medios fotográficos. Ambos discos crean un pulso eléctrico cuando la luz pasa por sus aberturas. La Fig. VI.3.4.2.A es un ejemplo de una cabeza de lectura para discos codificados. La posición codificada de los discos pasa a través de un patrón de luz emitido por un LED y es recibida por un fototransistor (PT). Cuando la luz no llega al fototransistor, la salida será baja (cero lógico). Cuando la señal de luz no es bloqueada, ésta llega al PT y la salida es alta. La alternación de luz y obscuridad da como resultado un tren de pulsos, Fig. VI.3.4.2.B, a través del cual es posible inferir la posición.

Para entender mejor el funcionamiento de estos sensores, se hará un breve análisis del hipotético caso en que se use un disco codificado como el de la Fig. VI.3.4.1.B y un decodificador como el de la Fig. VI.3.4.2.C.



(A)



(B)

FIG. VI 3.4.1 DISCOS DE CODIGO SECUENCIAL

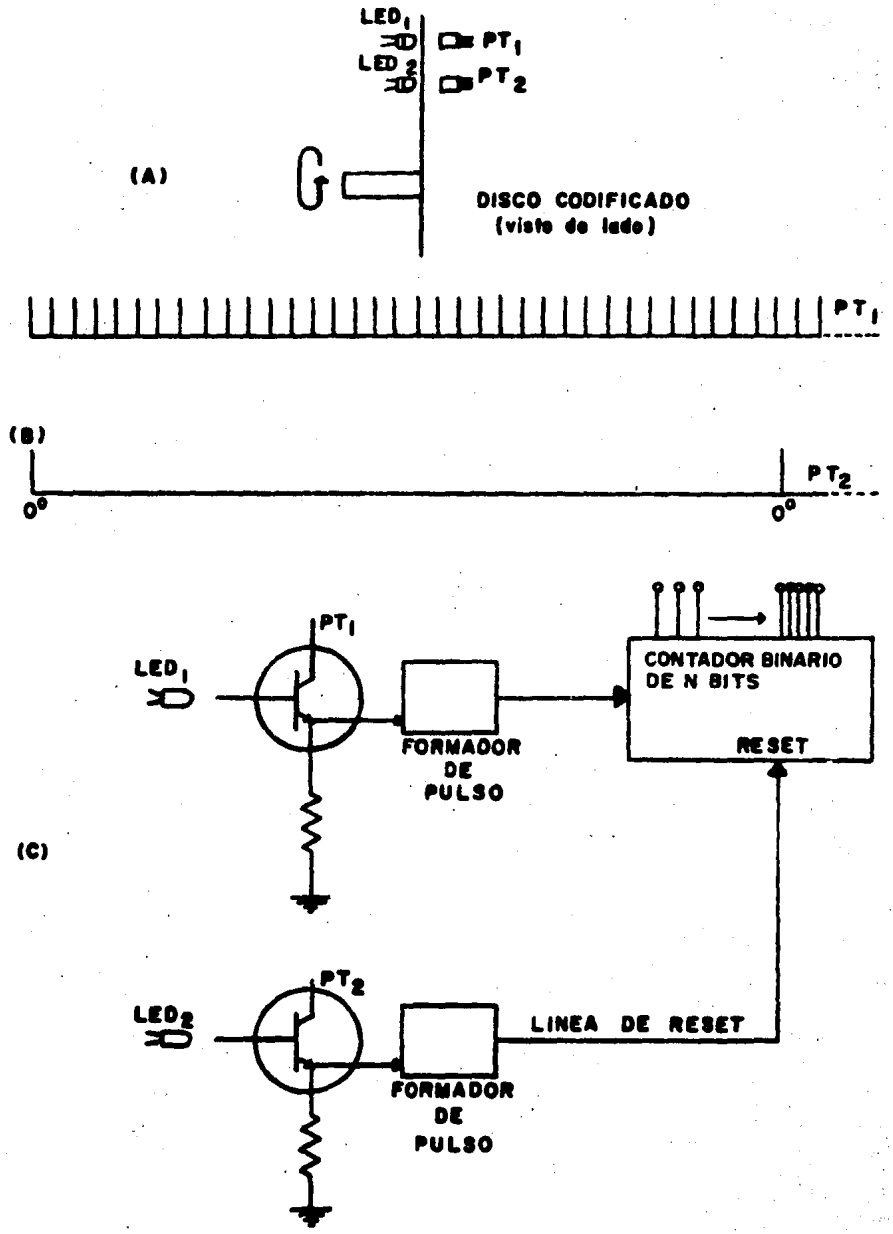


Fig: VI.3.4.2. CIRCUITOS DE DISCOS DE CODIGOS
(A) CONFIGURACION DEL TRANSDUCTOR (B) FORMAS DE ONDA
(C) DIAGRAMA DE BLOQUE CIRCUITO.

El principal elemento del decodificador es un contador binario de N bits, donde N es el número de regiones claras del anillo externo del disco. Si se desean conocer incrementos angulares de un grado, entonces se necesitarán 360 regiones claras en el anillo externo del disco y, por lo tanto, se necesitará un contador binario de 9 bits (2 exp 9 = 512 edos.) para registrar las 360 diferentes posiciones.

La región clara del anillo interno del disco se usa para indicar una posición de referencia (usualmente cero grados). Se creará un pulso en PT2 por cada 360 pulsos de PT1, con lo que se podrá reconocer la referencia o posición 0/360 grados. El pulso generado por PT2 sirve para restaurar a cero el contador binario.

En la Fig. VI.3.4.3 se ve un disco codificado en BCD (decimal codificado en binario) el cual dá una posición absoluta del disco en lugar de una relativa como en el caso de los discos A y B. Existen otros códigos más utilizados en este tipo de discos, como el GRAY que por sus características de ser código reflejado y con un solo cambio por número, permite una operación más confiable del disco.

Un transductor puede utilizarse para obtener información de velocidad y de aceleración. Recordando que la velocidad es la derivada de la posición y la aceleración es la derivada de la velocidad y la 2a derivada de la posición.

VI.3.5 Transductores de Presión

Existen tres formas comunes de transductores de presión:

- La galga extensométrica resistiva
- La inductiva
- Los dispositivos de estado sólido

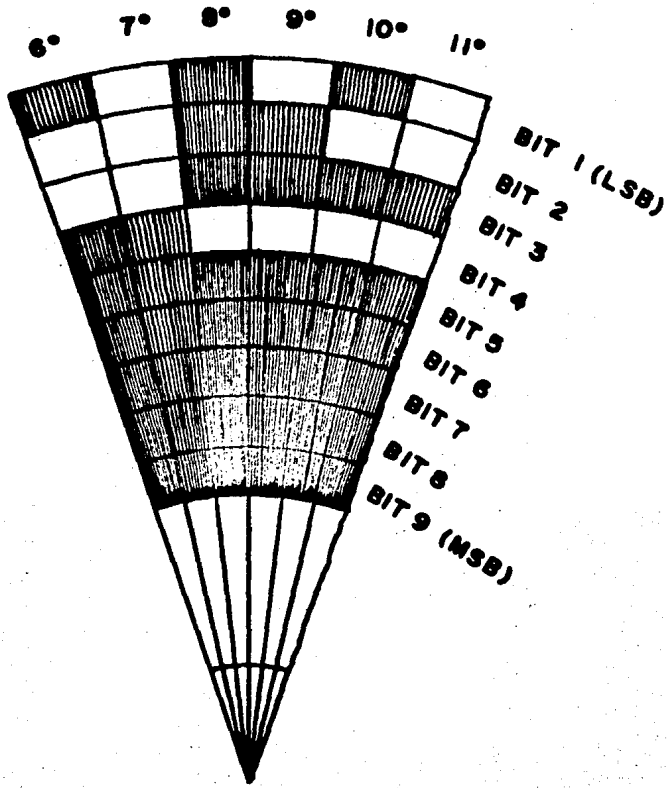


FIG. VI. 3. 4. 3 DISCO DE CODIGO BINARIO, GRAY O BCD

La galga extensométrica resistiva consiste de un puente Wheathstone de cuatro elementos montados en una cara de un diafragma metálico. El fluído es aplicado a la otra cara del diafragma, cambiando la resistencia de los elementos del puente.

La Fig. VI.3.5 muestra un transductor de presión de tipo inductivo fabricado por la firma Hewlett-Packard, el modelo 1280B/C. Este es también una forma de puente de Wheathstone, pero formado por elementos inductivos, lo cual hace que se requiera de una fuente de excitación de CA. Los elementos variables en el puente son las reactancias de los inductores L1 y L2, mientras que las resistencias R1 y R2 son los elementos fijos.

A presión cero (atmosférica) se condiciona que el núcleo se encuentre igualmente repartido dentro de L1 y L2. Mas cuando el diafragma se encuentra bajo presión, causa que el núcleo salga parcialmente de una bobina y entre en la otra; provocando un desbalanceo del sistema, parecido el del LDVT.

National Semiconductor´ s, fabrica una amplia línea de transductores de presión en circuitos integrados. Estos están constituidos por un puente resistivo, una fuente de excitación regulada, compensación de temperatura y los amplificadores necesarios para producir un voltaje de salida de alto nivel. Este tipo de transductores se denominan híbridos, debido a que el puente resistivo se fabrica mediante una técnica y la compensación y los amplificadores con otra; incluyéndose ambos en el mismo circuito integrado. Para información más detallada consulte los catálogos de National Semiconductor´s.

Dentro de esta línea existen transductores para diversas aplicaciones, desde transductores para presiones negativas (menores que la atmosférica) hasta transductores para muy pequeños rangos de presiones. En el esquema VI.1 se resumen estos transductores.

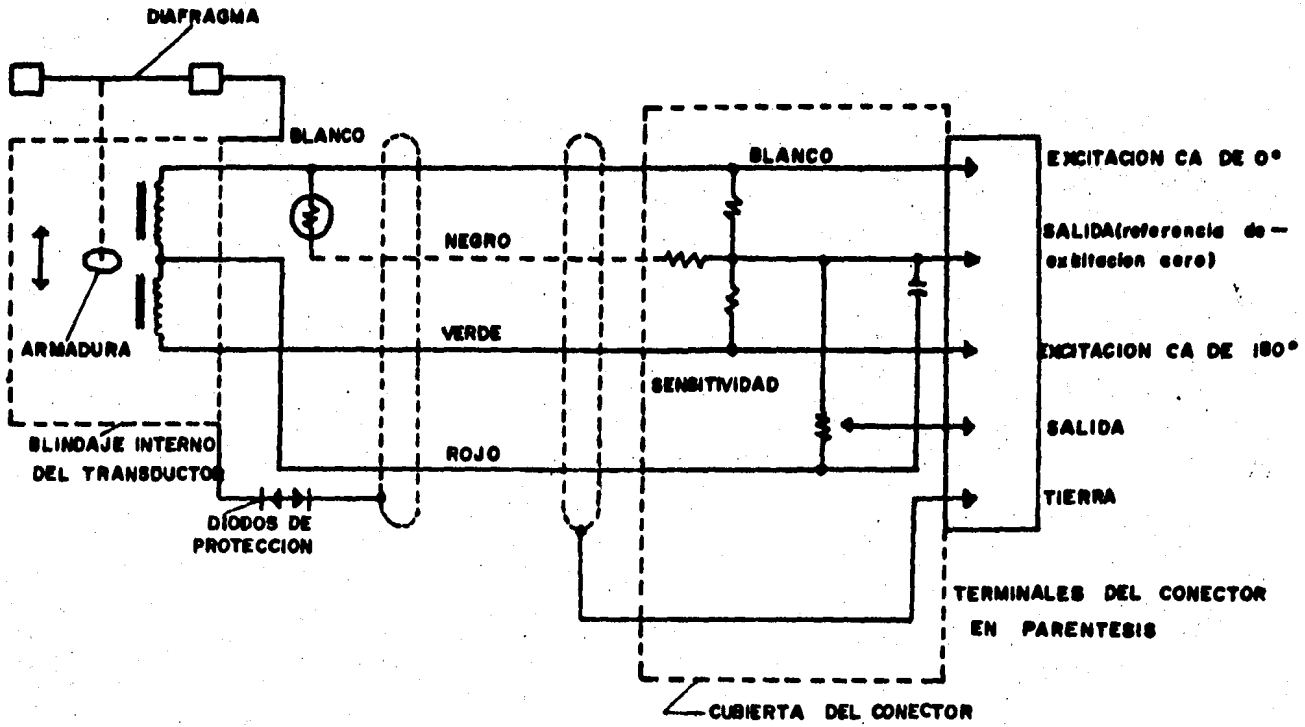


FIG.VI.3.5. TRANSDUCTOR INDUCTIVO DE PRESION DE FLUIDOS.

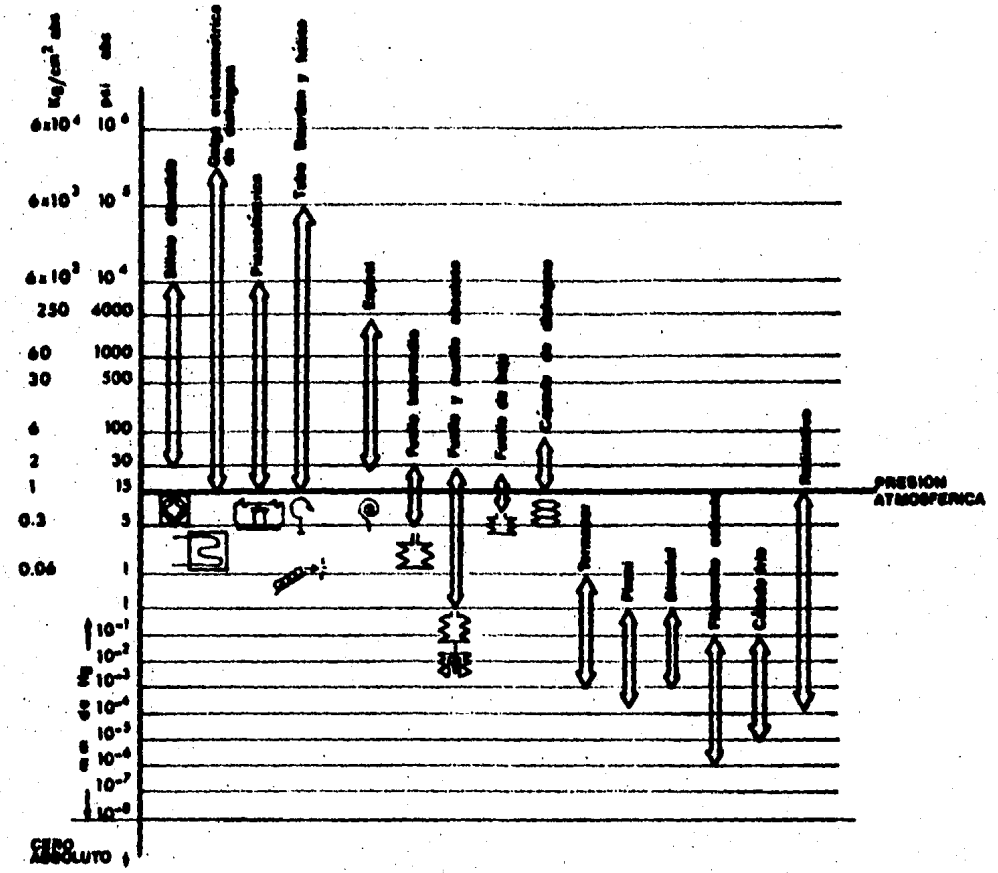


FIG.VI.1 TRANSDUCTORES DE PRESION Y CAMPO DE APLICACION

VI.3.6 Transductores de Temperatura

Existen tres formas básicas de transductores de temperatura:

- El Termistor

- El Termopar

- La Juntura de Semiconductores pn

Todo conductor eléctrico experimenta un cambio en su resistencia cuando se ve sometido a variaciones de temperatura. El termistor es un dispositivo en el cual esta propiedad se ha optimizado y es predecible.

La Fig. VI.3.6.1 muestra una curva de transferencia típica de un termistor. La curva es decididamente alineal, excepto para temperaturas entre T1 y T2, esta región se utiliza directamente para medir temperatura mediante el uso de un puente Wheathstone. Más alla de T2 es necesario linealizar la curva para evitar errores de transducción.

La curva presenta un coeficiente negativo, lo que quiere decir que a mayor temperatura presenta una menor resistencia.

Un termopar se forma mediante la unión de dos alambres de aleación diferente en V (Fig. VI.3.6.2). Cada material tiene asociada una propiedad llamada función de trabajo. Las diferencias en la función de trabajo del material A y del B dan un potencial dentro del rango de los milivolts que es función directa de la temperatura; es decir, cuando se calienta la unión se genera un voltaje en los extremos.

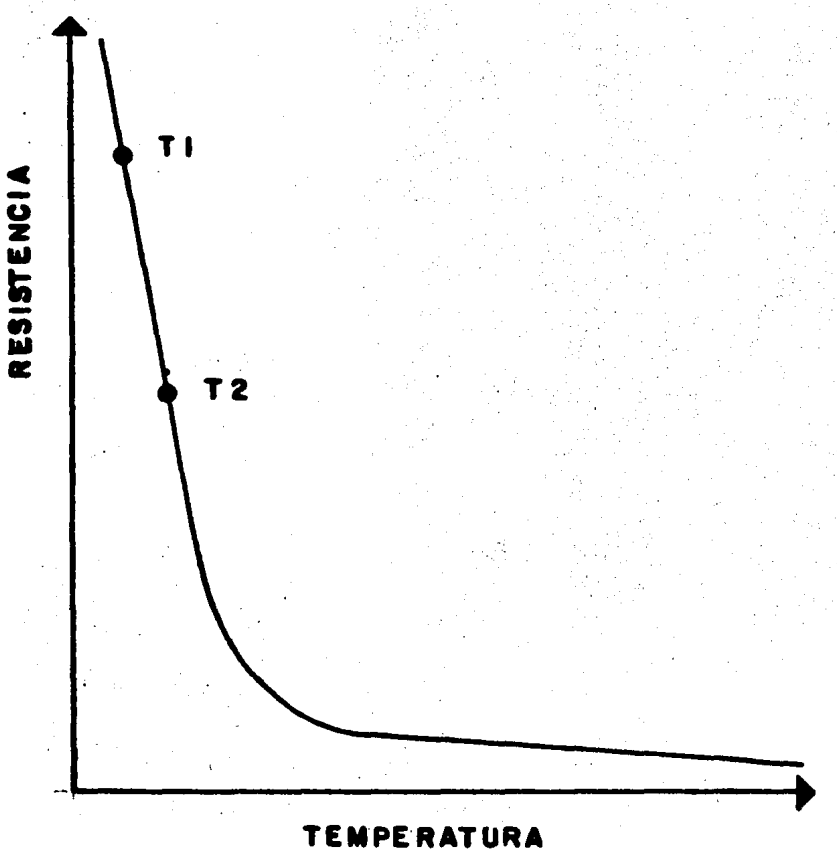
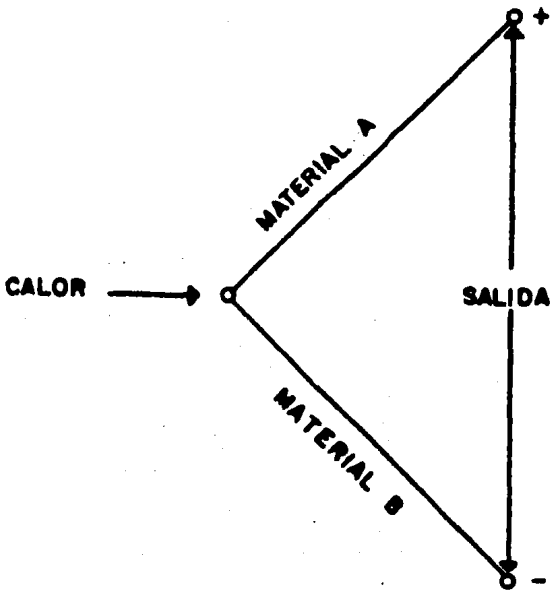


FIG. 3.6.1 FUNCION DE TRANSFERENCIA DE UN TERMISTOR TIPICO



T	COBRE VS. CONSTANTAN
E	CROMO VS. CONSTANTAN
J	HIERRO VS. CONSTANTAN
K	CROMO VS. ALUMINIO
G	TUNGSTENO VS. TUNGSTENO 26% RENIO
C	TUNGSTENO 5% RENIO VS. TUNGSTENO 26%
R	PLATINO VS. PLATINO 13% RODIO
S	PLATINO VS. PLATINO 10% RODIO
B	PLATINO 6% RODIO VS. PLATINO 30% RODIO

FIG. VI.3.6.2 TERMOPARES. CONFIGURACION Y COMPOSICION

Un problema asociado con los termopares es que se forma otro termopar entre el alambre que forma cada uno de los brazos y el alambre que lo conecta al dispositivo de medición. Esto es de poca consecuencia si los materiales del alambre son los mismos. Existen en la actualidad varios tipos de termopares que tienen diferentes propiedades y aplicaciones (Fig. VI.3.6.3).

Tipo J. Este termopar consiste en un alambre de acero y uno de aleación de constantan. Puede utilizarse en atmósferas químicamente reductoras hasta 1600 grados Farenheit.

Tipo T. Este se forma por alambres de cobre y constantan y se utilizan en atmósferas poco reactivas y hasta 750 grados Farenheit.

Tipo K. Este se forma por un alambre con aleación de cromo y otro con aleación de aluminio y se utiliza en atmósferas químicamente oxidantes a temperaturas no mayores de 2300 grados Farenheit.

Tipo E. Este se forma por alambres de cromo y constantan y se utiliza en atmósferas químicamente inertes y en vacío hasta los 1600 grados Farenheit.

La juntura pn tiene excelentes propiedades como transductor de temperatura, lo que la hace más aceptada aunque no llegue a los rangos de temperatura del termopar. Un par de transistores monolíticos conectados como diodo, ofrecen el mejor transductor de temperatura de semiconductor, donde el voltaje base-emisor (Vbe) es proporcional a la temperatura absoluta de la juntura.

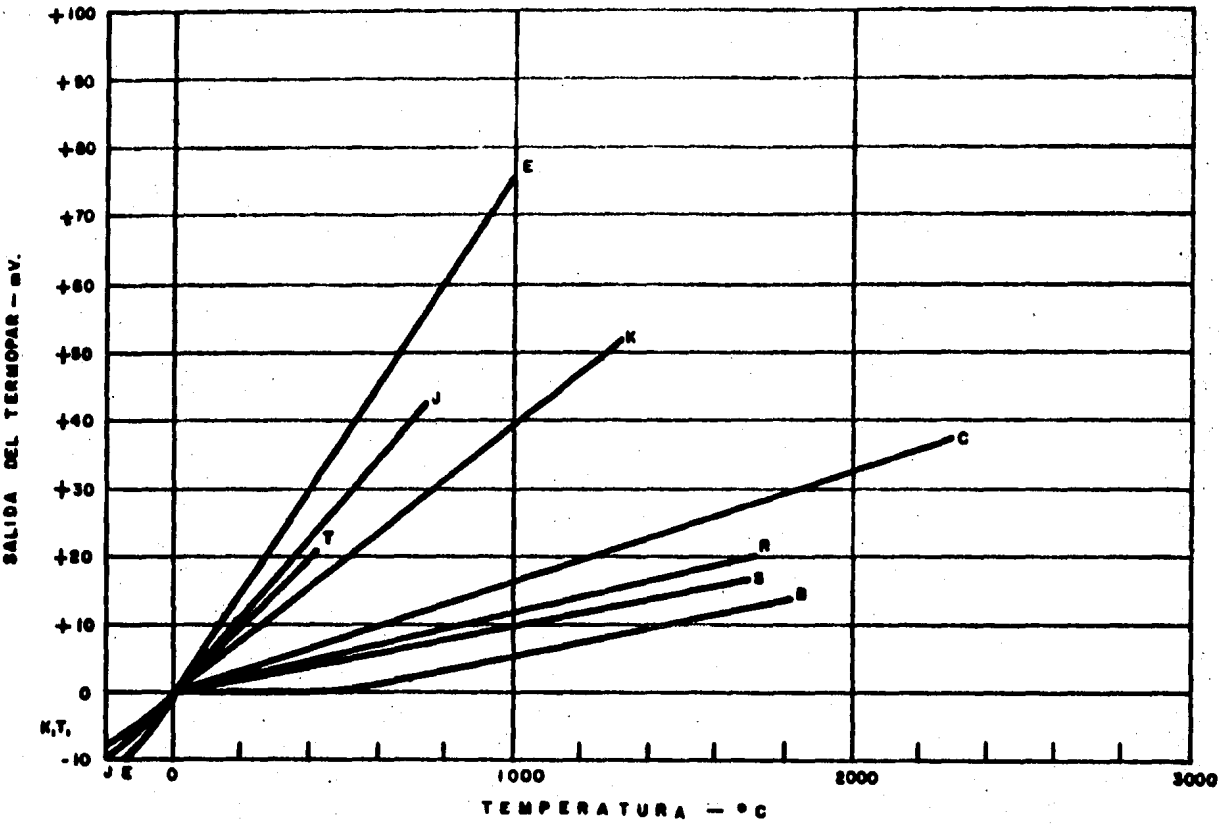


Fig. VI. 3.6.3 Caracteristicas de Salida de los Termopares.

National Semiconductor´s ofrece un par de transductores de temperatura en circuito integrado que incluyen un sensor de temperatura (juntura pn), una referencia de voltaje y un amplificador operacional. La salida del transductor es directamente proporcional a la temperatura (en grados Kelvin), en una relación de 10mV/grados K. Mediante el amplificador operacional puede obtenerse casi cualquier salida a partir del sensor. El rango de temperaturas de operación va desde -55 grados C hasta +125 grados C con una presición de +-4 grados C.

VI.3.7 Transductores de Luz (Fotométricos)

Los transductores de luz más utilizados son los fotoresistivos, los fotovoltáicos y los de estado sólido, como el fotodiodo y el fototransistor.

El transductor fotoresistivo emplea la característica de algunos elementos, de tener una resistencia eléctrica que varía proporcionalmente con la iluminación. Las fotoresistencias son elementos fotosensores pasivos que consisten de una mezcla de sulfuro de cadmio o seleniuro de cadmio o una mezcla de ambos materiales. Además, son elementos bipolares y por lo tanto se pueden utilizar en circuitos de CA y de CD.

Las celdas fotovoltáicas son dipolos activos cuya característica principal es la de producir un voltaje que se incrementa en forma logarítmica en función de la iluminación y se fabrican de Si o Se. La gráfica VI.3.7.1 muestra la sensitividad espectral de una celda fotovoltáica. Como se puede ver son más sensibles a la luz roja.

Los fotodiodos son junturas pn polarizadas en la región inversa. Cuando la luz incide en las cercanías de la juntura (Fig. VI.3.7.2) se incrementa el número de portadores de cargas lo que se traduce en un incremento en la corriente de inversa. Debido a esta

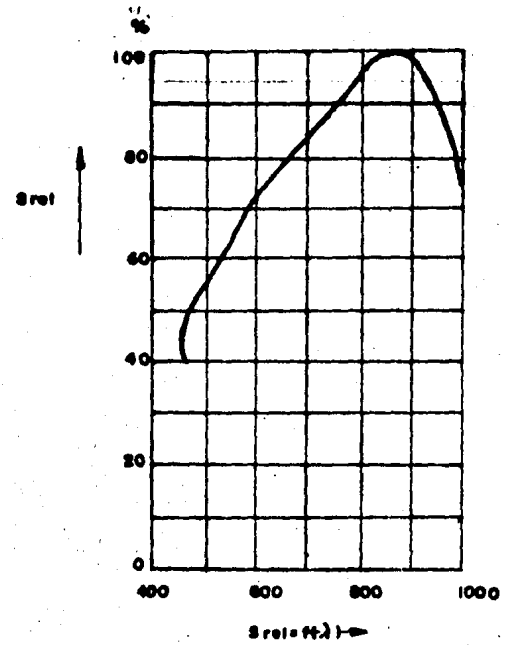


FIG. VI.3.7.1 SENSITIVIDAD ESPECTRAL RELATIVA DE UNA CELDA FOTO — VOLTAICA.

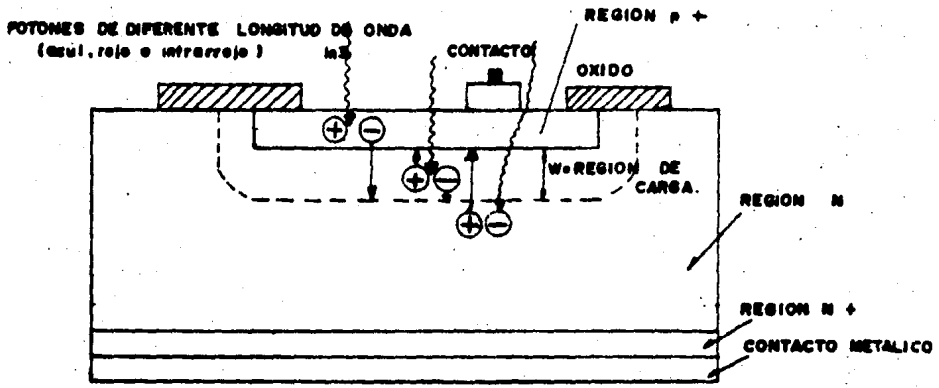


FIG. VI.3.7.2. FOTODIODO

característica los fodiodos se utilizan particularmente en la medición de intensidad de luz.

Cuando la juntura no se encuentra iluminada, fluye solamente una pequeña cantidad de corriente a través de ésta como resultado de los portadores generados térmicamente. Por otro lado, con luz se generan pares de portadores (electrón-hueco) adicionalmente en la región p y en la n. Los portadores generados en la región de carga son extraídos inmediatamente debido al campo eléctrico presente. Los portadores del campo remanente se deben difundir primero a la región de carga para separarse. Si estos huecos y electrones se recombinan antes, no contribuyen a la corriente del fotodiodo.

El transistor fotoeléctrico o fototransistor es excelente para utilizarse como receptor de luz de lámparas incandescentes, ya que su mayor sensitividad (Fig. VI.3.7.3) se encuentra en la región cercana al infrarrojo. El modo de operación del fototransistor es igual al de un fotodiodo pero con un amplificador. Generalmente tiene una fotosensitividad de 100 a 500 veces mayor que el fotodiodo. Cuando se encuentra iluminado, la juntura de la base actúa como un fotodiodo y ésta se amplifica por el factor B(beta) del transistor, dando como resultado una corriente B(beta) veces mayor entre el colector y el emisor (Fig. VI.3.7.3.B).

Contrariamente al fotodiodo, el fototransistor tiene una linealidad menor ya que el factor B(beta) depende de la corriente en la base.

Existen dispositivos más especializados como el LASCR (Ligth Activated Silicon Controled Rectifier) que consiste de un SCR disparado por la corriente generada por un fotodiodo. Este es un transductor de luz discreto ya que el SCR solamente puede tener dos estados, en conducción (on) o apagado (off).

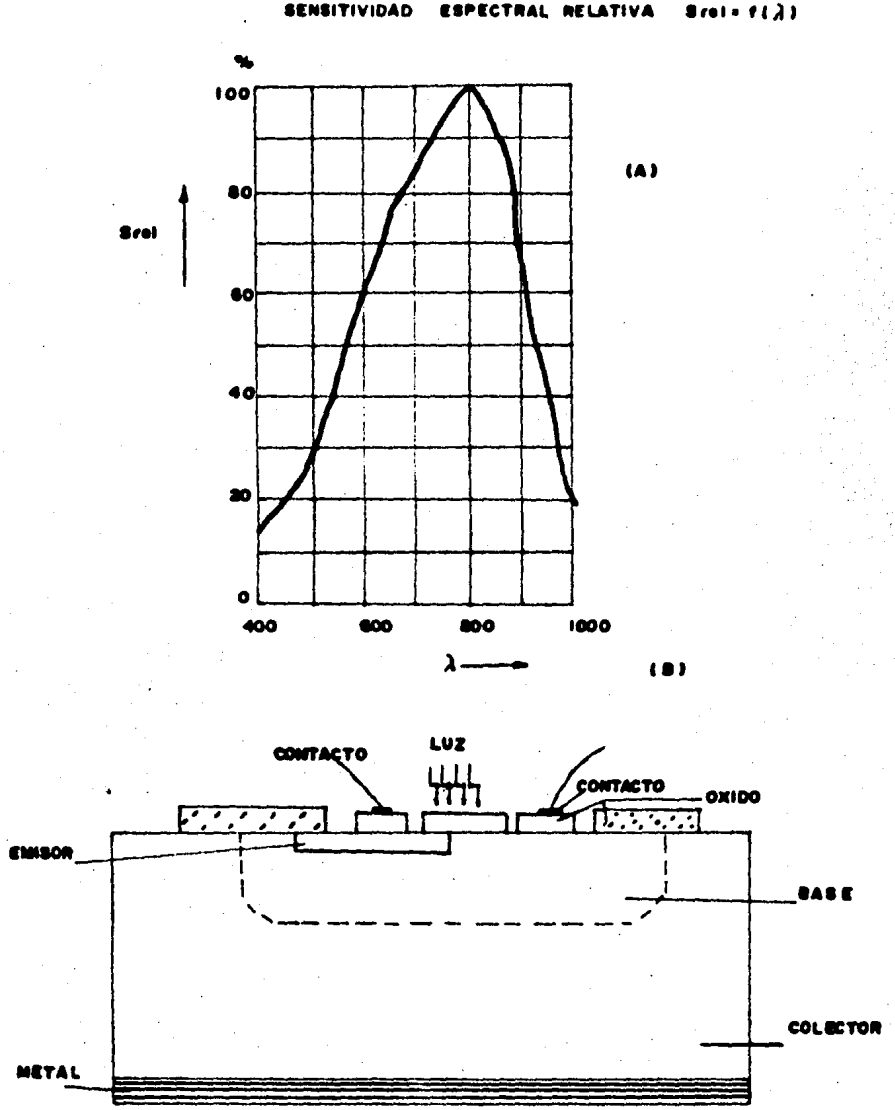


FIG. VI. 3.7.3. FOTOTRANSISTOR (A) SENSITIVIDAD (B) ESTRUCTURA

RESUMEN DE TRANSDUCTORES COMUNES

TEMPERATURA

TIPO	CARACTERISTICAS ELECTRICAS DE ENTRADA/SALIDA	COMENTARIOS
TERMOPARES	BAJA IMPEDANCIA, TIPICAMENTE DE 10 OHMS. A TEMPERATURA AMBIENTE LAS SALIDAS TIPICAS DEL ORDEN - DE MILIVOLTS. VARIACIONES DE -- DECIMAS DE MICROVOLT POR GRADO CENTIGRADO.	BAJO VOLTAJE DE SALIDA REQUIERE CONDICIONAMIENTO. TAMAÑO PEQUEÑO Y AMPLIO INTERVALO DE TEMPERATURA. REQUIERE REFERENCIA DE-UNA TEMPERATURA CONOCIDA. RESPUESTA NO LINEAL.
TERMISTORES	CAMBIA RESISTENCIA CON TEMPERATURA. COEFICIENTE DE TEMPERATURA NEGATIVO. A TEMPERATURA AMBIENTE, DISPONIBLE DE 50 A 1M OHM, SENSIBILIDAD CERCANA A 4% POR GRADO CENTIGRADO. EXISTEN CIRCUITOS PARA LINEALIZAR RESPUESTA.	ALTA SENSIBILIDAD. NO LINEAL, FUNCION EXPONENCIAL.
SENSORES SEMICONDUCTORES	MODIFICAN VOLTAJE, CORRIENTE O RESISTENCIA. TIPO VOLTAJE COMO EL DIODO, REQUIERE EXCITACION DE CORRIENTE. TIPO CORRIENTE COMO EL AD590 REQUIERE EXCITACION DE VOLTAJE. TIPO RESISTIVO PUEDE USAR EXCITACION DE CORRIENTE O VOLTAJE.	MUCHOS DISPOSITIVOS NO ESTAN CALIBRADOS Y REQUIEREN CONDICIONAMIENTO. EL AD590 ES LINEAL Y ESTA CALIBRADO, REQUIERE CONDICIONAMIENTO MINIMO.

FUERZA

TIPO	CARACTERISTICAS ELECTRICAS DE ENTRADA/SALIDA	COMENTARIOS
GALGA EXTENSOMETRICA	VARIA LA RESISTENCIA CON EL ESFUERZO APLICADO. CASI SIEMPRE SE USA EN CONFIGURACION PUENTE. LA IMPEDANCIA TIPICA ES DE 120 Y 350 OHMS. CAMBIO TIPICO DEL 0.1% SOBRE EL INTERVALO COMPLETO.	REQUIERE CONDICIONAMIENTO.
PIEZOELECTRICOS	CAMBIOS SIGNIFICATIVOS EN LA SALIDA. LOS CAMBIOS EN LA ENTRADA PRODUCEN CAMBIOS DE CARGA	

C A P I T U L O V I I

ACTUADORES

VII.1 INTRODUCCION

A la salida de la computadora digital, el valor de la variable de control, u(k), o sus cambios, están presentes brevemente como valores digitalizados. Los actuadores analógicos (p.e. neumáticos, hidráulicos o eléctricos) requieren un convertidor D/A (DAC) con almacenamiento intermedio y/o un elemento retenedor que mantenga el valor de la variable de control durante una muestra. La posición (u) o cambio deseado (u) para el actuador se transmite a éste como un voltaje directo entre 0 y 10 volts o como una corriente directa entre 0 y 20 mA (estos son valores estándar), donde es transformada y amplificada a la propia señal neumática, hidráulica o eléctrica.

Con lo anterior puede inferirse que el actuador es el elemento del esquema de control que recibe la señal del controlador para regular la potencia suministrada al elemento final de control (ver Sistemas de control en el cap. II). Es decir, es el elemento (o elementos) del esquema de control que se encarga de ejecutar las acciones mandadas por el controlador.

Es necesario aclarar que debido a que las señales que provienen del controlador (algoritmo de control) son de baja potencia, no pueden activar a los elementos finales de control (a la mayoría de ellos), es por esto que el actuador interviene como interfase entre estos y el controlador.

Debido a la gran variedad de actuadores disponibles, solamente una selección de algunos tipos se consideran aquí, incluidos en distintos sistemas de control; tales cgmo el control de intensidad de luz (Dimmer) o control de fase, control de C.A. encendido/apagado mediante interruptores electrónicos, control de pequeños motores de C.D., etc.. Sin embargo, es necesario analizar brevemente a los convertidores D/A (dada la necesidad mencionada al principio de este capítulo) y mencionar algunos convertidores disponibles en el mercado.

VII.2 CONVERTIDORES DIGITAL/ANALOGICO

El problema básico de conversión digital a analógica es cambiar niveles de voltaje digital a un voltaje analógico equivalente. Esta conversión puede realizarse fácilmente mediante un circuito resistivo que cambie el nivel digital a un voltaje binario ponderado equivalente (Equivalent Binary Weigth Voltage).

La estructura básica de todo convertidor D/A convencional contiene una red de resistencias de precisión, un conjunto de interruptores, alguna forma de escalamiento de voltaje para adaptar los manejadores de los interruptores a los niveles lógicos especificados y un voltaje de referencia. Cada interruptor cerrado agraga una cantidad de corriente (con peso binario) al bus de salida. El amplificador operacional a la salida de la red efectúa la conversión corriente a voltaje, dado que se pretende encontrar un voltaje analógico equivalente a un voltaje digital. Un convertidor D/A de resistencias ponderadas se presenta en la Fig. VII.2.1.

La corriente Ie obtenida en el bus de salida (Fig. VII.2.1) es la suma de las aportaciones debidas a cada bit. Si el bit es "1" lógico entonces el interruptor se cerrará conectándose a la fuente Vref, haciendo que fluya una corriente proporcional al peso del bit (este

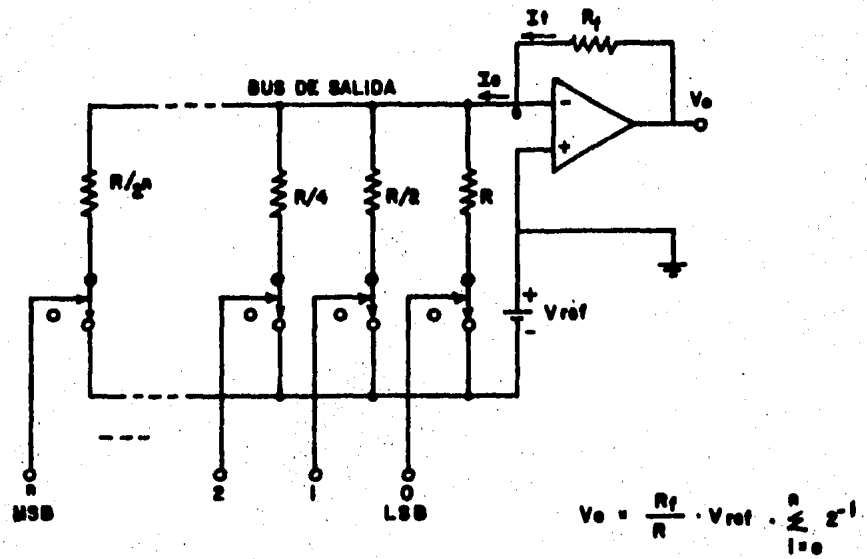


Fig: VII.2.1 Convertidor D/A de resistencias ponderadas

peso está dado por el valor de la resistencia). Si por el contrario, el bit es "0" lógico el interruptor se abrirá inhibiendo la aportación de corriente. Con esto se obtiene:

$$Ie = Vref \left(1/R + 1/(R/2) + 1/(R/4) + \ldots + 1/(R/2^{n}) \right)$$

$$= Vref \; 1/R \sum_{i=0}^{n} 2^{-i} \qquad \text{(VII.2.1)}$$

donde en la sumatoria aparecerán solamente aquellos términos para los cuales el bit i tenga un valor de "1" lógico.

En el nodo "e" (entrada negativa del amplificador operacional) se aplica la ley de corrientes de Kirchhoff, llegando a:

$$Ie = If \qquad (Ie - If = 0)$$

$$Vref \; 1/R \sum_{i=0}^{n} 2^{-i} = Vo/Rf$$

de donde

$$Vo = Rf/R \; Vref \sum_{i=0}^{n} 2^{-i} \qquad \text{(VII.2.2)}$$

siendo esta última expresión la que dará el voltaje analógico equivalente al voltaje digital.

Los convertidores D/A de resistencias ponderadas presentan el problema de la gran diversidad de los valores de las resistencias, ya que todas son diferentes. Un convertidor de 12 bits requiere un rango de valores de resistencias de 4,096 a 1. Por otro lado, la corriente que soporta la resistencia del MSB es mucho mayor que la del LSB lo que dificulta aún más la elección de dichas resistencias.

Para evitar lo anterior, el convertidor D/A con red resistiva R-2R, mostrado en la Fig. VII.2.2, requiere un limitado número de resitencias. Este hecho ha permitido la realización de convertidores D/A monolíticos, exactos y de bajo costo.

Utilizando el mismo procedimiento de análisis para el convertidor D/A de resistencias ponderadas, aplicado ahora al circuito de la Fig. VII.2.2, obtenemos:

$$Vo = Rf/2R\ Vref/3 \sum_{i=0}^{n} 2^{-i} \qquad \text{(VII.2.3)}$$

siendo esta última expresión la que dará el voltaje analógico equivalente al voltaje digital.

En este circuito, los interruptores aplican el voltaje de referencia o tierra a las aristas con resistencia 2R de la red resistiva, esta está terminada (por el LSB) por una resistencia 2R aterrizada, para mantener una operación apropiada. Este circuito se comporta como el anterior en cuanto a la operación binaria, debido a que genera una salida que es el producto de Vref y la fracción binaria. El voltaje de salida puede escalarse con la resistencia de realimentación, Rf.

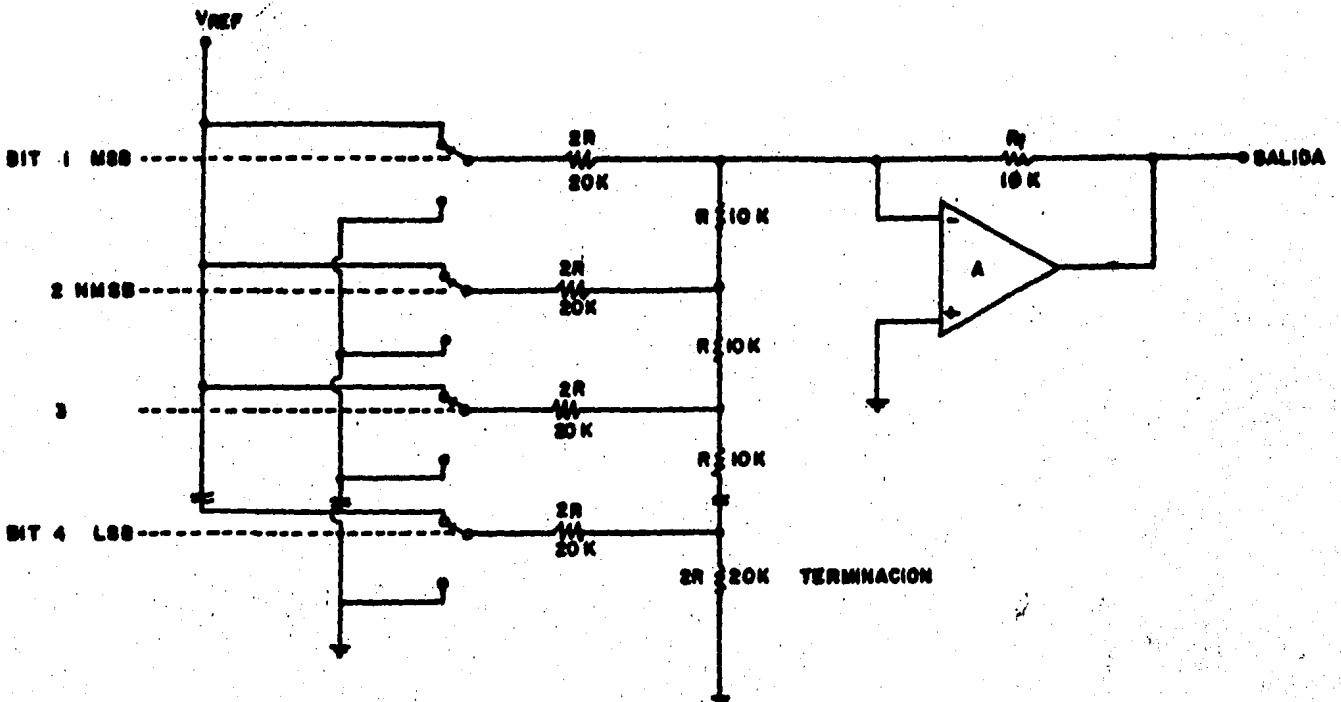


Fig: VII.2.2. Convertidor D/A R – 2R

Una ventaja importante de estos circuitos es que no está limitado en cuanto al número de bits, esto se debe a que las secciones R-2R pueden agregarse conforme sea necesario.

Hasta aquí se ha expuesto el funcionamiento básico de los convertidores D/A, la mayoría de los convertidores en el mercado funcionan bajo estos principios y a continuación se presentan algunos de estos.

VII.3 CONVERTIDORES D/A COMERCIALES

Como se ha mencionado, existen una gran variedad de convertidores D/A disponibles en el mercado. A continuación se presentan algunos de ellos, mencionando las características principales.

VII.3.1 El Convertidor DAC-08

Es uno de los convertidores D/A más populares en el mercado (Fig. VII.3.1.1), fabricado por Precision Monolithics, Inc.. Este dipositivo contiene la escalera R-2R estándar, interruptores electrónicos y un amplificador de referencia, pero requiere un voltaje de referencia externo.

El DAC-08 produce dos corrientes de salida que se dice son complementarias, mientras una aumenta la otra disminuye, pero su suma, corriente de escala completa, es constante. El valor de la suma es igual a:

$$Ifs = (Vref/Rref)\ (255/256)$$

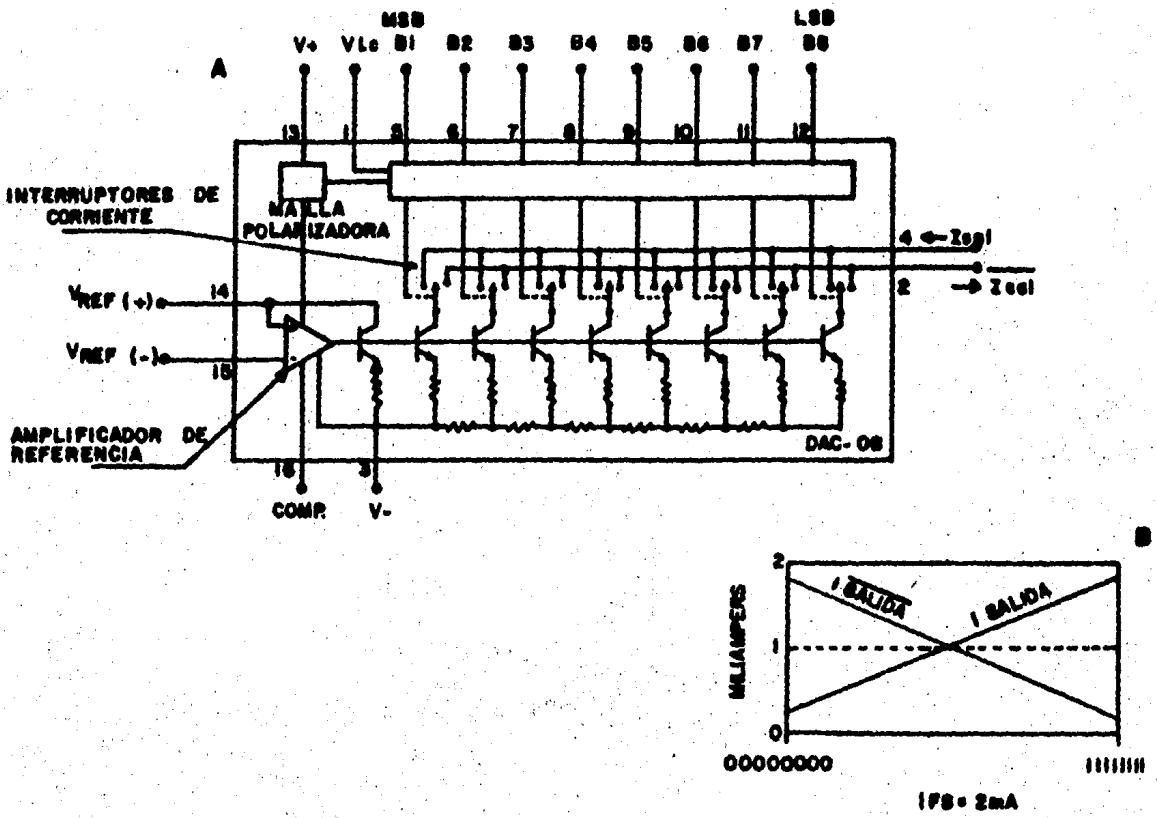


Fig: VII.3.1.1 Convertidor DAC—08

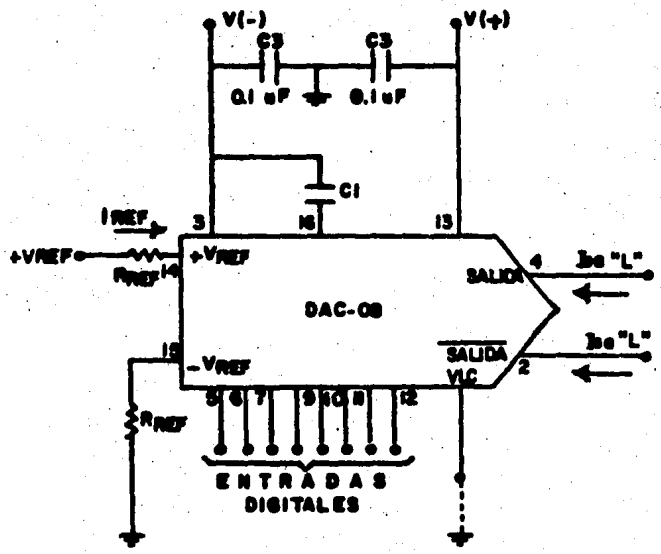


Fig: VII .3.1.2. Conexiones básicas del DAC-08

Donde Ifs es la corriente de escala completa y Vref y Rref son el voltaje y la resistencia de referencia, respectivamente.

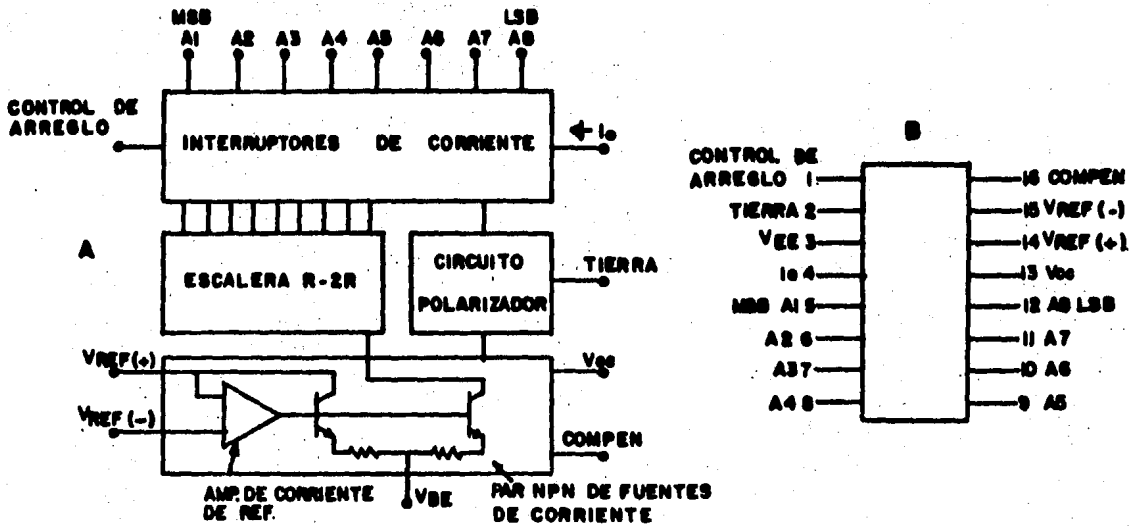
Este convertidor es capaz de realizar operaciones monotónicas de multiplicación sobre un rango de aproximadamente 32 dB de la corriente de referencia y el tiempo de asentamiento es de 85 ns. El DAC-08 está disponible en varios modelos que tienen diferentes niveles de calidad y rangos de temperatura. A continuación se presenta una tabla comparativa:

Modelo	Rango-Temp(°C)	No-linealidad
DAC-08AQ	-55/+125	+-0.1 %
DAC-08Q	-55/+125	+-0.19%
DAC-08NQ	0/+70	+-0.1 %
DAC-08EQ	0/+70	+-0.19%
DAC-08CQ	0/+70	+-0.39%

La Fig. VII.3.1.2 muestra las conexiones básicas para el DAC-08. El voltaje de umbral (VLC) controla los voltajes que serán reconocidos como niveles lógicos por las terminales de entrada, esta característica lo habilita para ajustarse a varias familias lógicas.

VII.3.2 El Convertidor D/A 1408

Es fabricado por Motorola Semiconductors Products. El 1408 (Fig. VII.3.2.1) es otro convertidor D/A de bajo costo y con salida de corriente, está provisto con las terminales de voltaje de referencia positivo y negativo y debe estar conectado de tal manera que la



Fig; VII.3.2.1. Convertidor D/A 1408

corriente de referencia siempre fluya por la terminal 14. El voltaje de referencia positivo requiere que la terminal 15 sea aterrizada a través de una resistencia y la terminal Vref (+) sea conectada al voltaje de referencia a través de una resistencia.

Al igual que el DAC-08, el 1408 cuenta con un amplificador de referencia, la escalera R-2R y 8 interruptores electrónicos. El tiempo de asentamiento es de 300 ns

VII.3.3 El Convertidor DAC-05

La mayoría de los convertidores D/A son de 8 bits, sin embargo cuando se requiere mejorar la resolución se presenta la necesidad de incrementar el número de bits. Mientras que los convertidores de 16 bits tienden a ser muy caros, varias compañías han desarrollado convertidores de 10 y 12 bits a bajo costo, un ejemplo de estos es el DAC-05 fabricado por Precision Monolithics, Inc., mostrado en la Fig. VII.3.3.1.

El DAC-05 puede usarse en modo multiplicativo o no multiplicativo debido a que la fuente de referencia interna debe ser conectada a la entrada del amplificador de referencia a través de una conexión externa. La fuente de referencia sale por la terminal 17, mientras que la entrada del amplificador de referencia está en la terminal 15.

Este convertidor es un dispositivo con salida de voltaje, debido a que cuenta con amplificadores operacionales para convertir la corriente de la salida de la escalera binaria a un nivel de voltaje. Puede encontrarse con un rango de operación (escala completa) de +-5 volts (sufijo X2) y de +-10 volts (sufijo X1). La siguiente tabla muestra los diferentes tipos del DAC-05:

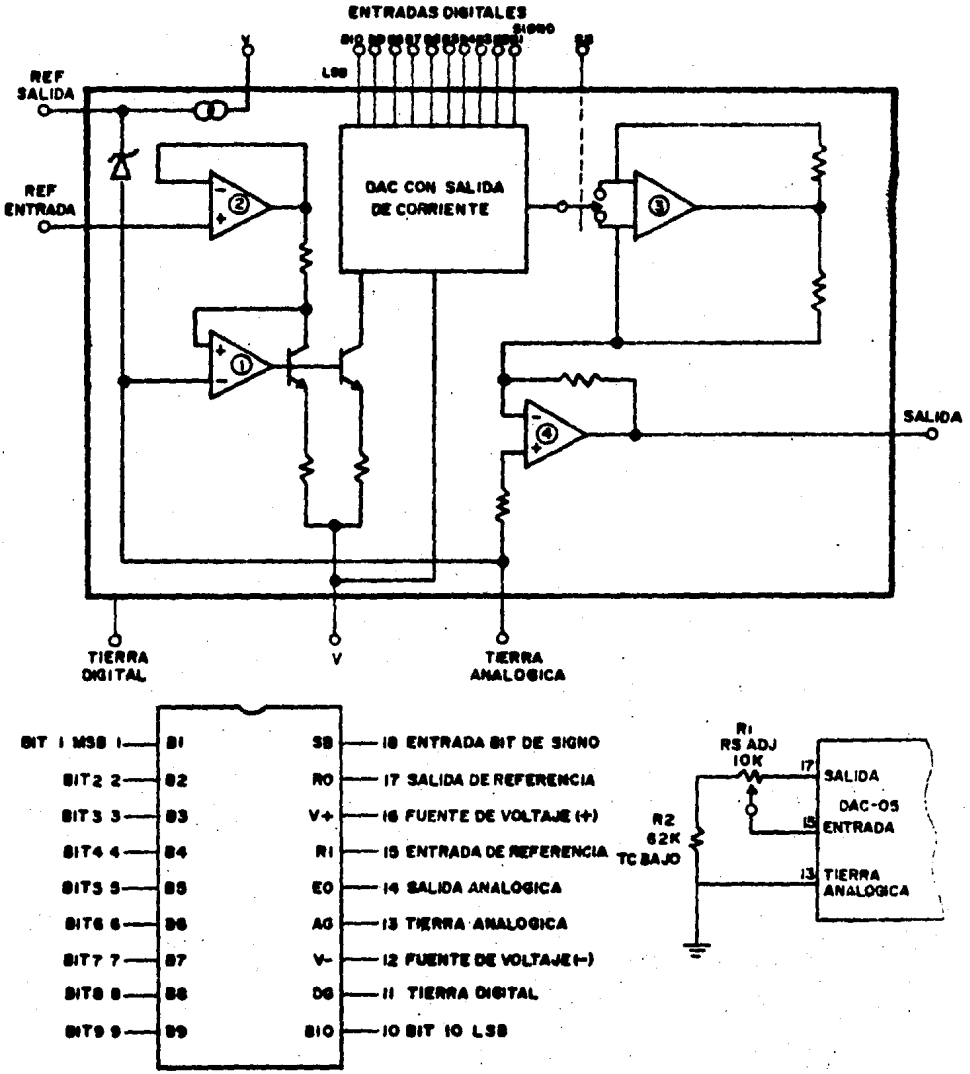


FIG. VII. 3.3.1 CONVERTIDOR DAC – 08

Número de tipo	Bits de entrada	Rango de Temperatura
DAC-05AX1 (o X2)	10	-55/+125
DAC-05BX1 (o X2)	9	-55/+125
DAC-05CX1 (o X2)	8	-55/+125
DAC-05EX1 (o X2)	10	0/+70
DAC-05FX1 (o X2)	9	0/+70
DAC-05GX1 (o X2)	8	0/+70

La temperatura está en grados centígrados.

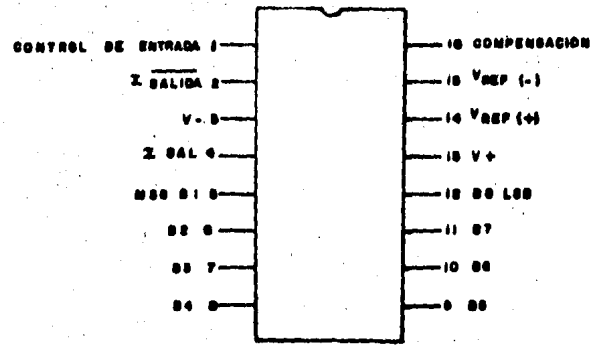
El DAC-05 es un dispositivo de operación bipolar, debido a que cuenta con un bit de signo que le permite distinguir a las entradas positivas y negativas. El bit de signo será "1" lógico para operación positiva, y "0" lógico para operación negativa.

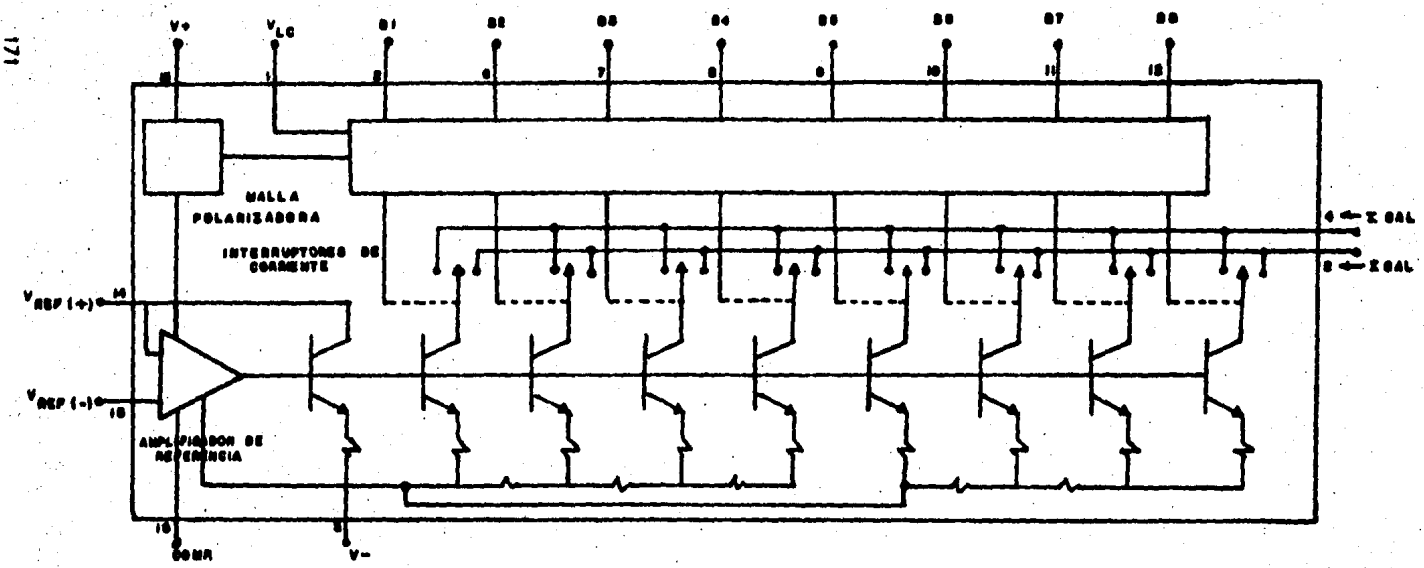
VII.3.4 El Convertidor DAC-20

Fabricado por Precision Monolithics, Inc., el DAC-20 (Fig. VII.3.4.1) es un convertidor de 2 digitos en BCD, de alta velocidad, operación multiplicativa y con salida de corriente. Este dispositivo usa la escalera R-2R estándar y es muy similar al DAC-08, debido a que tienen las mismas especificaciones excepto aquellas referentes al código y a la linealidad.

Existen 4 dispositivos disponibles en el mercado: el DAC-20C y el DAC-20E para operación entre 0 y 70 grados centigrados, con no linealidades máximas de +- 1/4 y +- 1/2 del LSB respectivamente y el DAC-20A y el DAC-20 para operación entre - 55 y + 125 grados centigrados, con linealidades similares respectivamente.

Fig: VII.3.4.1. CONVERTIDOR DAC – 20





La siguiente tabla muestra una comparación entre las características básicas del DAC-08 y el DAC-20:

Características	DAC-08	DAC-20
No linealidad	0.1 a 0.39 %	1/4 a 1/2 LSB
Tiempo de asentamiento	85 ns	85 ns
Código	binario	BCD
Multiplicación monotónica	si	si
Umbral lógico programable	si	si

VII.3.5 El Convertidor DAC-100

Es un convertidor de 8 o 10 bits, con salida de corriente y fuente de referencia interna. Usa la escalera R-2R estándar e interruptores electrónicos, el código de entrada es el binario complementario para operación unipolar y es compatible con la lógica de 5 volts TTL y CMOS. El DAC-100 se muestra en la Fig. VII.3.5.1 y a continuación se presentan algunas de sus características básicas:

No linealidad	0.05 a 0.3 %
Tiempo de asentamiento	375 ns
Código binario complementario	

Existen muchos más convertidores D/A que no se mencionan aquí, para mayor información sobre los mencionados y otros, se sugiere consultar la siguiente referencia: IC Converter Cookbook de Walter G. Jung.

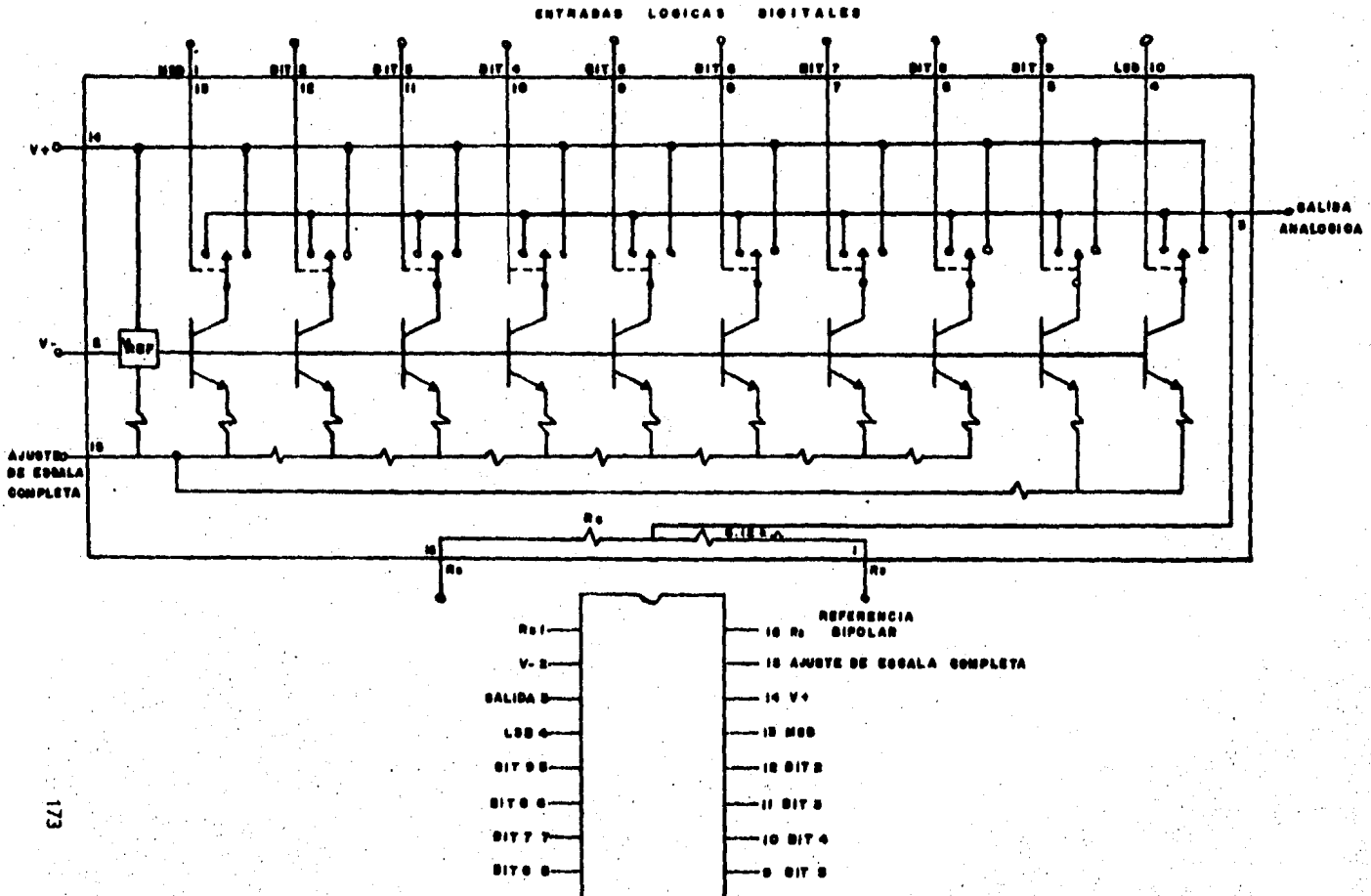


Fig: VII.3.5.1 CONVERTIDOR DAC 100

VII.4 CONTROL ENCENDIDO/APAGADO

Es sin duda el esquema de control más sencillo de realizar puesto que solamente controlan la energía de alimentación a la planta, para que ésta encienda o apague.

Para ejemplificar éstos esquemas, se presenta a continuación el control encendido/apagado de una lámpara de 120 volts y 60 Hz mediante una microcomputadora.

Una solución posible para éste control es un relevador magnético para controlar el alto potencial de C.A. con el que se alimenta a la lámpara y un transistor NPN de silicio para manejar al relevador. Casi cualquier transistor NPN que maneje los niveles necesarios de voltaje y corriente, podrá hacer esto.

La configuración del circuito que realiza el control encendido/apagado se presenta en la Fig. VII.4.1, en donde la resistencia se ha seleccionado para trabajar con niveles TTL, dado que se asume que el puerto de salida de la microcomputadora maneja estos niveles. Además, solamente se necesita un bit del puerto de salida.

Cuando el bit es "bajo" (0 volts), el transistor Q1 está en corte y no fluye corriente por la bobina del relevador K1. Debido a que K1 está desenergizado, la línea de 120 volts está abierta y la lámpara apagada. Cuando el bit va a "alto" (5 volts), el transistor se polariza en directa logrando que fluya corriente por el colector y se sature, lo cual hace que K1 encienda y se cierren los contactos del relevador, logrando así encender la lámpara.

El diodo D1 se usa para suprimir el pico de voltaje generado por la bobina de K1, evitando así daños al circuito. Dado que dicho pico de voltaje puede ser bastante alto, el diodo D1 debe tener un voltaje pico inversa alto para usarse en este circuito.

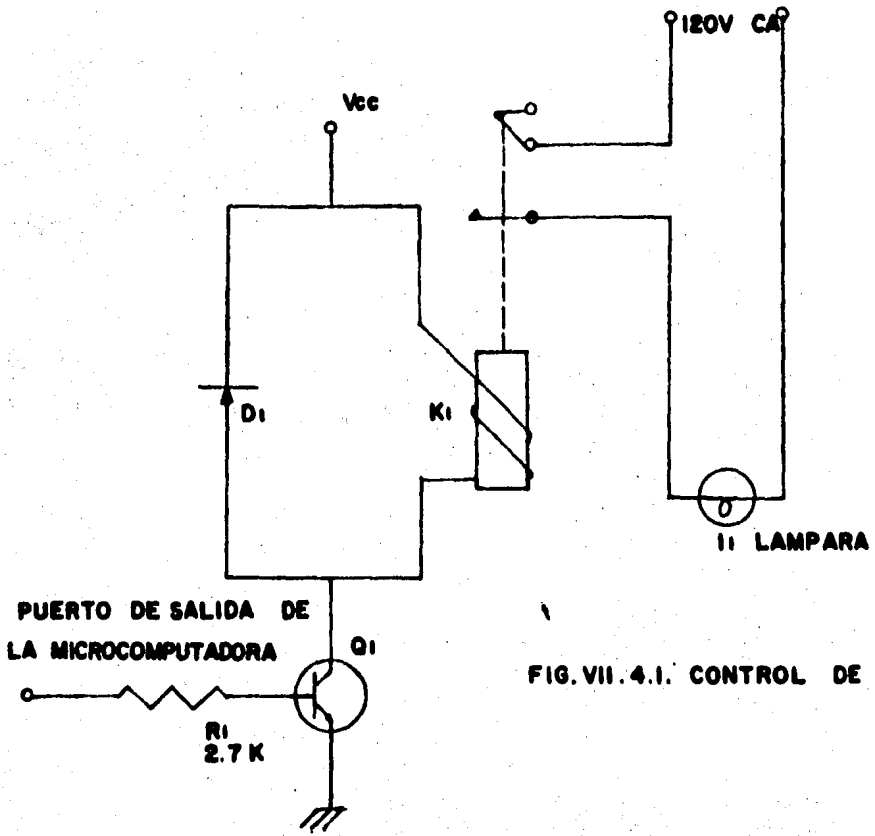


FIG. VII.4.1. CONTROL DE ENCENDIDO / APAGADO DE UNA LAMPARA.

Otra manera de realizar el control encendido/apagado de la lámpara, es mediante el uso de un optoaislador, un SCR y circuitos lógicos. En la Fig. VII.4.2 se muestran dos configuraciones distintas para este tipo de control.

En el circuito de la Fig. VII.4.2.A puede observarse que si la entrada (al inversor 1) es "baja", entonces la salida del inversor 2 es también "baja" y el LED enciende, logrando que el fototransistor Q1 conduzca. El colector de Q1, así, permanece en un potencial cero o cercano, lo cual mantiene al SCR apagado y por lo tanto, a la lámpara también.

Aplicando un nivel "alto" a la entrada, el LED se apaga y corta al transistor Q1, encendiendo al SCR. Cuando el voltaje en el colector de Q1 va a "alto", una corriente circula hacia la compuerta del SCR y lo enciende, encendiendo también a la lámpara. En el circuito de la Fig. VII.4.2.A la conmutación sel SCR se hace manualmente, abriendo el interruptor S1. Este arreglo permite que un solo pulso mantenga encendida a la lámpara.

El circuito de la Fig. VII.4.2.B, por otro lado, debe pulsarse contínuamente para poder mantener a la lámpara encendida. Mientras más largo sea el tren de pulsos aplicado al circuito, mayor será el tiempo en que permanezca encendido el SCR y la lámpara.

El diodo D1 (Fig. VII.4.2.B) rectifica el voltaje de línea, de tal manera que una vez en cada medio ciclo de la corriente que fluye por el SCR cae a cero. Si la compuerta del SCR no está excitada cuando empiece el siguiente medio ciclo, entonces el SCR permanece apagado. En algunos casos, la compuerta del SCR puede pulsarse directamente por el puerto de salida de la microcomputadora, pero debido a que en este circuito se involucra un voltaje alterno de 120 volts, el aislamiento provisto por el optoaislador es muy necesario.

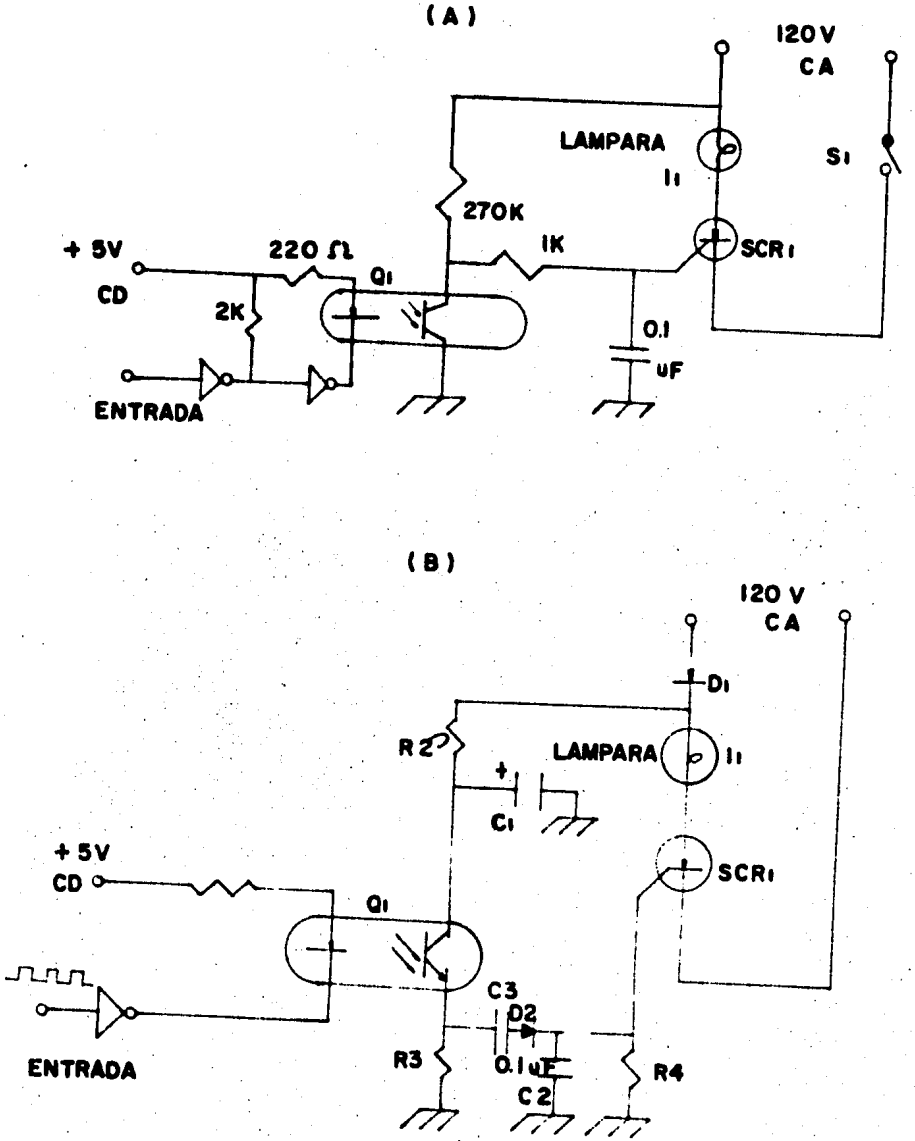


FIG. VII.4.2. DOS CONFIGURACIONES PARA EL CONTROL DE ENCENDIDO / APAGADO DE UNA LAMPARA.

La compuerta del SCR, en el circuito de la Fig. VII.4.2.8, es controlada por la malla RC, R4C2C3 y el diodo D2. Si el LED es pulsado a través del inversor, entonces el fototransistor es pulsado también y esos pulsos aparecen en la resistencia del emisor R3. Los pulsos en R3 son acoplados a través de C3 y D2, los cuales cargan al capacitor C2. Mientras los pulsos estén presentes, C2 permanecerá cargado y mantendrá a la compuerta del SCR excitada. Pero si cesa el tren de pulsos, C2 se descarga y el SCR permanecerá apagado después del primer ciclo de media onda (dado que se ha rectificado el voltaje de línea) en el cual, el voltaje en la compuerta esté por debajo del umbral de encendido.

VII.5 CONTROL DE INTENSIDAD DE LUZ

Los diseños tradicionales de control de intensidad de luz, usan indistintamente un reóstato para regular la corriente o un autotransformador para ajustar el voltaje a través de la lámpara. Cualesquiera de estos dispositivos analógicos requieren que una perilla sea girada para cambiar la brillantez de la lámpara y ninguno de estos es adecuado a controlarse mediante una microcomputadora.

Los controles modernos de intensidad de luz usan la técnica del control proporcional de fase de C.A., en donde un interruptor semiconductor en serie con la lámpara es abierto y cerrado 120 veces por segundo. La operación del interruptor se sincroniza para permitir que la corriente fluya por la lámpara, únicamente durante una fracción controlada de cada medio ciclo del voltaje alterno de 60 Hz.

El interruptor más usado comunmente para este tipo de control es el Triac. Un triac es un dispositivo de control que una vez encendido no puede apagarse hasta que la corriente que circula por él, caiga a cero.

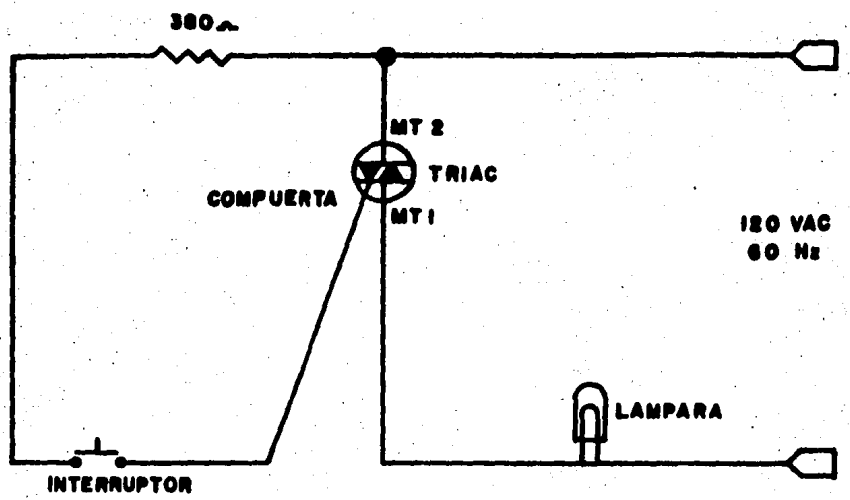


Fig: VII.5.1. Circuito básico para el control de fase de C.A.

La Fig. VII.5.1 muestra el arreglo básico para el control de fase de C.A., en donde un triac y una lámpara se conectan en serie con la línea de 120 volts y 60 Hz y, un interruptor en serie con la compuerta del triac es abierto y cerrado 120 veces por segundo, o una vez cada medio ciclo del voltaje de la línea. Al principio de cada medio ciclo el triac no conduce, de tal manera que no fluye corriente por la lámpara, si se cierra momentáneamente el interruptor, entonces un pulso de corriente llegará a la compuerta del triac, lo cual hará que se encienda y se establezca la conducción entre sus terminales. La resistencia de 380 ohms limita el valor del pulso de corriente para evitar que se dañe el triac.

La Fig. VII.5.2 ilustra cuando fluye o no corriente por la lámpara durante cada medio ciclo del voltaje alterno. En el instante en que el triac es encendido (cierre momentáneo del interruptor) la corriente circula por él y por la lámpara hasta el final del medio ciclo positivo, cuando la corriente cae a cero, el triac es entonces apagado y permanece así hasta el siguiente cierre momentáneo del interruptor que ocurre durante el medio ciclo negativo, con lo cual, el triac enciende de nuevo y permanece encendido hasta que termine el medio ciclo negativo.

De esta manera, la corriente fluye por la lámpara únicamente durante una fracción de cada medio ciclo del voltaje alterno. El tamaño de esta fracción depende de qué tan tarde, en cada medio ciclo, el interruptor sea cerrado momentáneamente y el triac sea encendido. Mientras mayor sea el retardo para encender el triac durante cada medio ciclo, menor será la potencia suministrada a la lámpara y, por lo tanto, disminuirá la brillantez.

Es necesario aclarar que el circuito mostrado en la Fig VII.5.1 sirve solamente para ilustrar el procedimiento del control de intensidad de luz, debido a que ningún interruptor mecánico puede abrirse y cerrarse con la velocidad y sincronización necesaria para

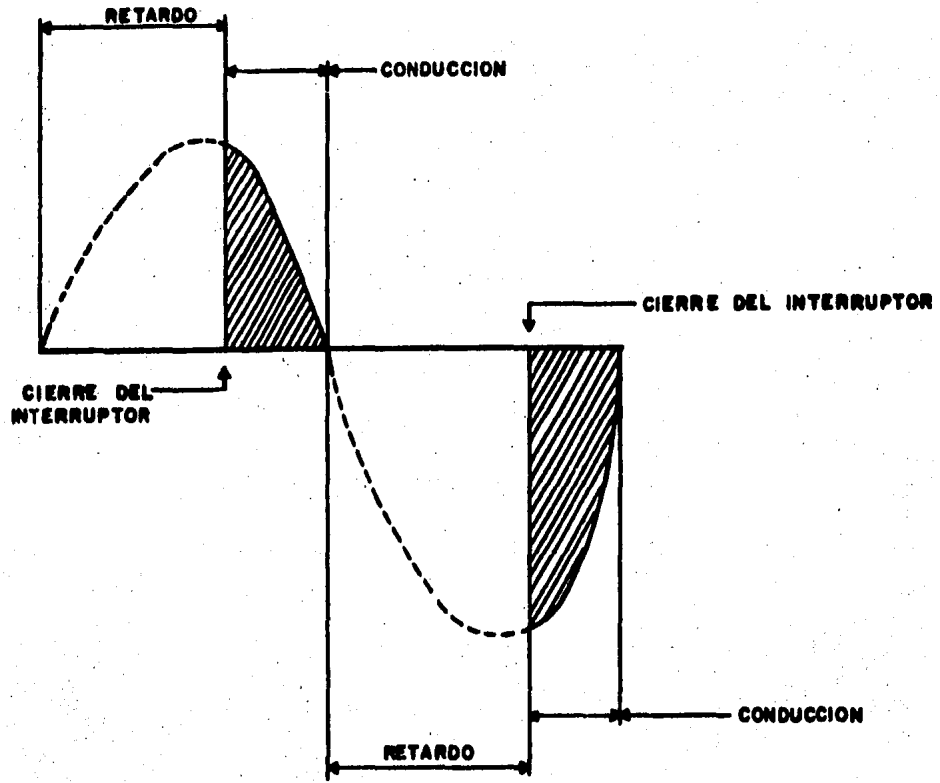


Fig. VII.5.2. Gráfica del Flujo de corriente a través del TRIAC y la lámpara

realizar el control proporcional de fase de C.A.. A continuación se presenta la solución a este problema.

VII.5.1 Control de intensidad de luz mediante una microcomputadora

Un circuito sencillo para acoplar la salida de la microcomputadora a la compuerta del triac se muestra en la Fig. VII.5.1.1. El dispositivo de acoplamiento es el Motorola MOC 3011, que es un manejador de triac aislado ópticamente y consiste de un led y un interruptor bilateral de silicio acoplado ópticamente.

En operación, cada pulso generado por la microcomputadora hace que el led produzca un destello que dispara al interruptor de silicio, haciendo que conduzca momentáneamente. Este envía un pulso a la compuerta del triac, encendiendolo por el resto del medio ciclo del voltaje alterno. El cicuito opera como el descrito anteriormente, excepto que el interruptor ha sido sustituido por el interruptor bilateral de silicio y el triac es encendido por cada pulso generado por la microcomputadora.

Una característica importante de este circuito es el aislamiento eléctrico, debido a que el MOC 3011 permite a la microcomputadora controlar el circuito de la lámpara permaneciendo aislado eléctricamente de él.

El control de fase requiere 120 pulsos por segundo, en donde cada pulso debe ocurrir con un retardo controlado despues del inicio de cada medio ciclo del voltaje alterno. Parte de la generación de los pulsos puede efectuarse por software, sin embargo, existe un requerimiento importante de hardware, dado que la microcomputadora puede producir el pulso apropiadamente solamente si conoce en qué momento inicia cada medio ciclo. Esto es, la microcomputadora debe

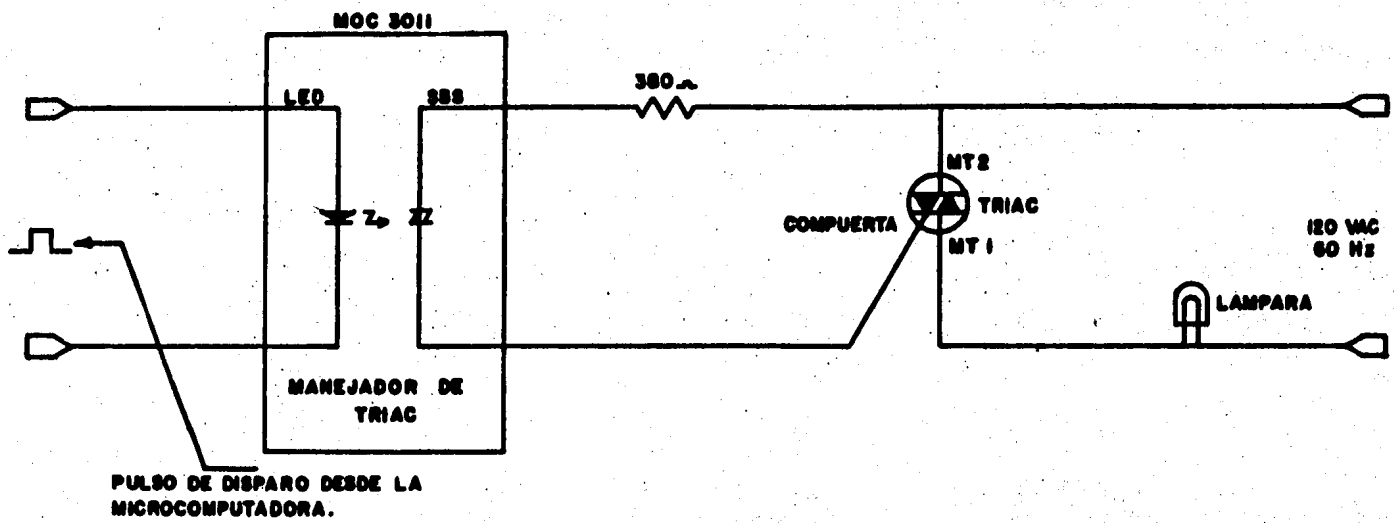


Fig: VII.5.1.1. Acoplamiento del TRIAC y la Microcomputadora.

estar sincronizada a la línea de voltaje alterno.

Para resolver el problema de diseño, es necesario utilizar un circuito que genere un cambio de nivel lógico al principio de cada medio ciclo; así, la microcomputadora podrá usar este cambio para sincronizarse a la línea. La Fig. VII.5.1.2 muestra el circuito que realiza este objetivo.

El sincronizador (Fig. VII.5.1.2) está compuesto por dos cicuitos integrados (CI): el CI1 es un detector de cruce por cero el cual produce un pulso de corriente (por la terminal 4) al principio de cada medio ciclo del voltaje alterno y, el CI2 que es un optoaislador, el cual toma el pulso de corriente producido por CI1 para "jalar" un pulso de corriente a través de R2 y generar momentáneamente un "0" lógico a la salida del sincronizador. Esta salida es el cambio de nivel lógico requerido.

La manera en que se completa el diseño de control de fase depende del plan a seguir. Uno de ellos es conectar la salida del sincronizador a una de las entradas de interrupción de la microcomputadora y cada vez que se reciba una interrupción, mediante una iteración de programa calcule el retardo deseado. Al final del retardo, el programa debe generar un pulso de salida para disparar el triac y esperar la siguiente interrupción de sincronización.

Como podrá notarse, el control de fase por este método está muy limitado, debido al amplio intervalo de potencia que se le debe suministrar a la lámpara para ajustarla desde la obscuridad hasta la máxima brillantez, lo cual requiere que el programa corra completamente en una fracción de tiempo (que depende del retardo) de cada medio ciclo del voltaje alterno.

Otra manera de llevar a cabo el diseño de control de fase, es hacer que la microcomputadora calcule un número que represente el

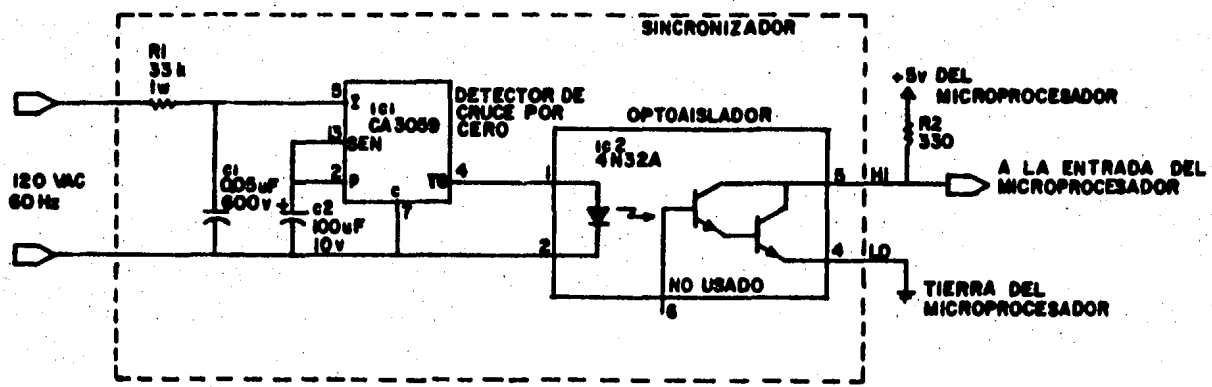


Fig: VII.5.1.2. SINCRONIZADOR

retardo deseado y lo deposite en un dispositivo periférico que usará este número para generar el pulso necesario y disparar el triac. Dicho dispositivo es el Timer programable, el cual puede configurarse (por software) de tal manera que, manejado por una señal externa (como la señal de sincronización), genera un pulso de salida despues de cada pulso de entrada. El intervalo de tiempo entre los pulsos de entrada y salida es, justamente, el retardo programado.

Por este método, el control de fase es más versátil debido a que el Timer programable releva a la microcomputadora de todo el procesamiento requerido para calcular el retardo y generar el pulso de salida. Con un Timer programable la microcomputadora se libera de correr programas complejos, ya que solamente necesita cargar el Timer con el nuevo valor del retardo cada vez que se requiera un cambio.

Además de las conexiones con el bus de la microcomputadora, el Timer tiene también compuertas de entrada (G) y salidas (O) y dentro de él, 3 registros direccionables que son:

- Un registro de control de 8 bits de escritura solamente, el cual se usa para establecer el modo de operación.

- Un registro "latch" de 16 bits de escritura solamente, cuyo contenido se divide en dos bytes, llamados M y L para el MSB y el LSB respectivamente. Estos dos bytes son colocados en el "latch" por el programa que ejecuta la microcomputadora y que pueden ser cambiados, por el programa, en cualquier momento.

- Un registro contador de 16 bits de lectura solamente, en donde se cargan los bytes M y L del "latch" cuando en la compuerta de entrada se presenta momentáneamente un "0" lógico. Si ocurre esto, entonces el registro contador decrementa su contenido en cada ciclo de reloj del microprocesador. Cuando el conteo llega a cero, un pulso de voltaje es entregado a la

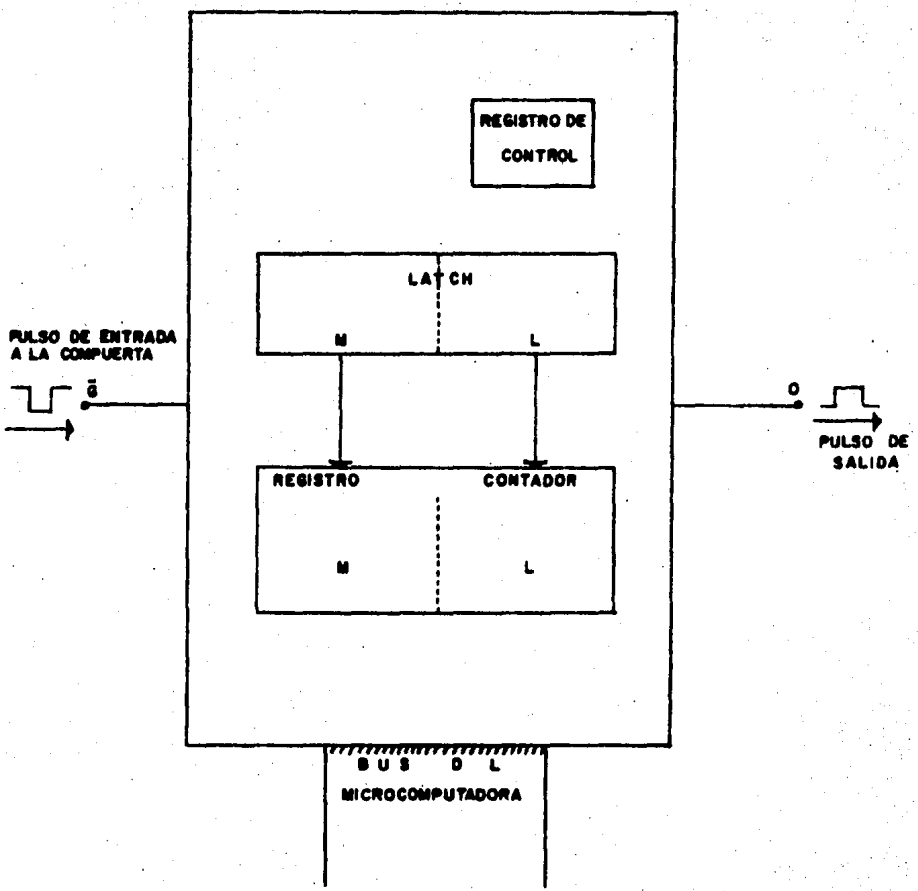


Fig: VII.5.1.3. Diagrama Simplificado de un Timer Programable.

salida del Timer. En la Fig. VII.5.1.3 se puede observar un diagrama simplificado de un Timer.

Para la aplicación de control de fase, la señal de sincronizacio.. debe conectarse a la compuerta del Timer, mientras que la salida de éste se usará para disparar el triac.

Cuando el Timer (Regulador de Tiempos) recibe un "0" lógico en la compuerta, el registro contador carga el número almacenado en el "latch" (sin alterar el contenido de éste) e inicia la cuenta regresiva.

VII.6 CONTROL DE MOTORES PEQUEÑOS DE CD

Encender y apagar una planta es, como se dijo antes, el problema más sencillo y un ejemplo de un esquema de control de malla abierta.

En la Fig. VII.6.1 se muestra el esquema de control encendido/apagado, en malla abierta, de un motor de CD mediante una microcomputadora y transistores de potencia.

El transistor Q1 se utiliza para manejar al transistor Q2. Si Q1 es un 2N3053 y Q2 es un 2N3055, entonces este circuito puede manejar motores que consuman 6 u 8 amperes. Debido a que la "beta" del transistor 2N3055 es pequeña (de 20 a 70), no puede ser manejado directamente por el puerto de salida de la microcomputadora y es por esto que se requiere el transistor Q1.

Para un puerto de salida compatible con niveles TTL el valor de la resistencia R1, mostrada en la Fig. VII.6.1, es suficiente; mientras que el valor de la resistencia R2 dependerá del nivel de Vcc. La operación de este circuito es como sigue:

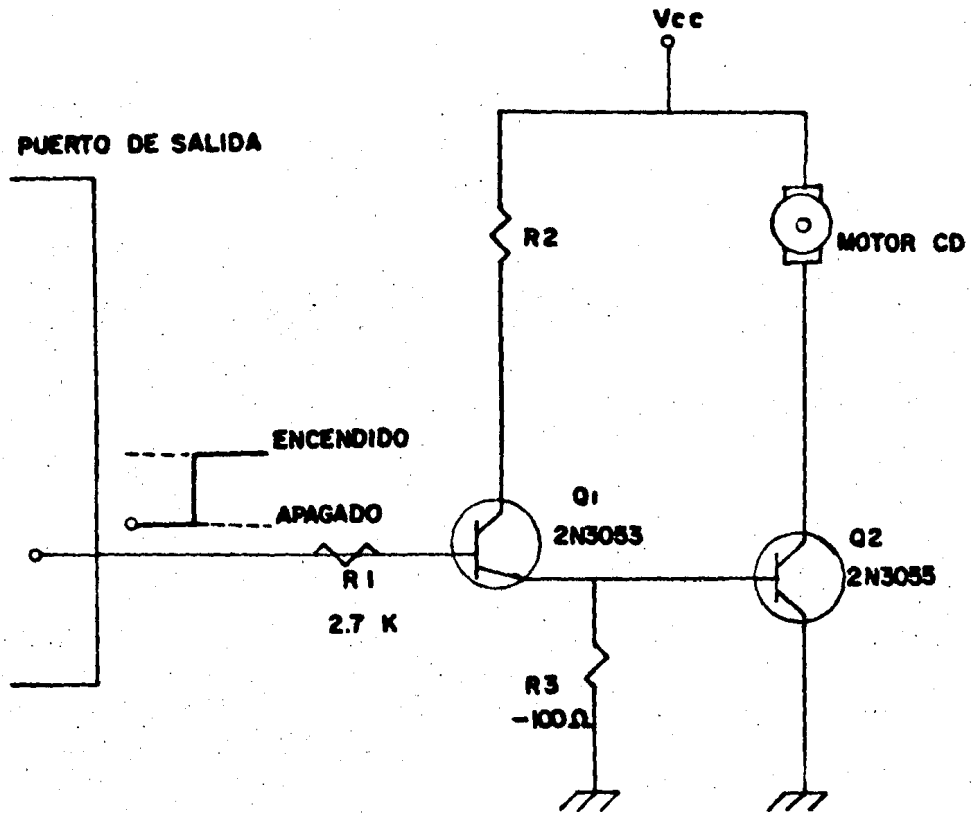


FIG. VII.6.1. CONTROL E/A EN MALLA ABIERTA DE UN MOTOR DE CD.

1. Un nivel "bajo" en el bit del puerto de salida de la microcomputadora apaga a los transistores Q1 y Q2, así que el motor se mantiene apagado.

2. Un nivel "alto" polariza en directa al transistor Q1 hasta que se satura, esto hace que el voltaje en su emisor sea "alto" y así, enciende el transistor Q2.

3. Ahora el transistor Q2 se polariza en directa hasta que se satura, con lo cual el motor se enciende y permanece así hasta que en el puerto de salida de la microcomputadora aparezca un nivel "bajo".

Se debe aclarar que el valor de la resistencia R3 deberá ser tal que limite el valor de la corriente que fluye en la base del transistor Q2, a un valor seguro.

VII.7 CONTROL DE MALLA CERRADA DE UN MOTOR DE CD

En un esquema de malla abierta el motor será encendido y apagado por una orden de la microcomputadora. En un esquema de malla cerrada, por otro lado, las órdenes de control del motor pueden modificarse por los datos recibidos desde éste. Aquí se usa un motor de CD para levantar una carga suspendida por una cuerda desde una altura Yo hasta Y1. En otras palabras, se pretende controlar la posición de la carga y para esto, es necesario utilizar un transductor de posición (ver capítulo VI); éste puede ser un potenciómetro de precisión que se acopla al eje de motor a través de un tren de engranes. En la Fig. VII.7.1 se presenta el sistema de control utilizado.

A la salida del potenciómetro, el voltaje E representa la altura de la carga (Y) que es una fracción de Eref, proporcional a la altura, esto es:

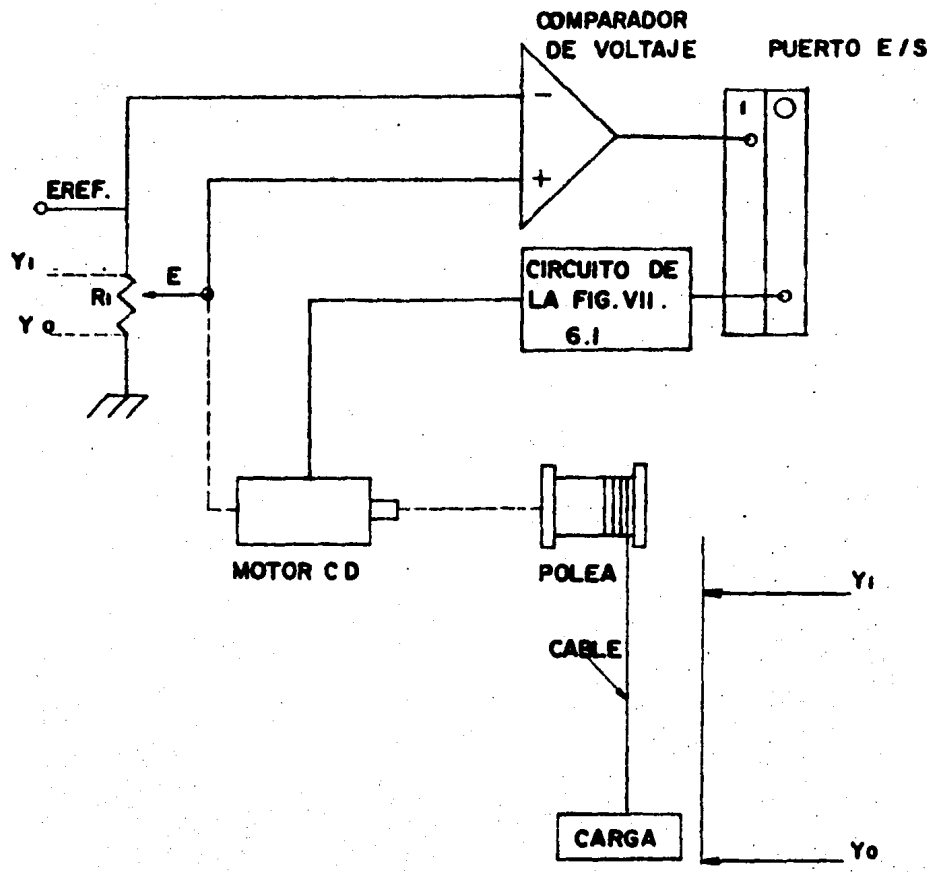


FIG. VII.6.1 CONTROL E/A EN MALLA ABIERTA DE UN MOTOR DE CD.

E = 0 a Yo

E = Eref a Y1

Ambos voltajes, E y Eref, se conectan a un comparador de voltaje que produzca niveles de salida de acuerdo a la regla siguiente:

CONDICION	SALIDA DEL COMPARADOR
E = 0	bajo
E = Eref	alto

La salida del comparador se conecta a un solo bit del puerto de entrada de la microcomputadora, mientras que el motor se conecta a un bit del puerto de salida, a través de un circuito como el de la Fig. VII.6.1.

Para controlar la posición de la carga se debe escribir un programa (algoritmo de control) que encienda el motor, colocando un nivel "alto" en el bit del puerto de salida y que verifique periódicamente el bit del puerto de entrada, el cual indica la posición Y. Mientras el bit del puerto de entrada permanezca en "bajo" el programa mantendrá al motor encendido, pero cuando se detecta un "alto", indicando que la carga está en la posición Y1, el programa debe cambiar a "bajo" el bit del puerto de salida, con lo cual se apaga el motor. Este circuito tiene varias desventajas, una de ellas es el hecho de que la carga no pueda bajarse una vez que ha alcanzado la posición Y1, además de que no puede reconocer más de dos estados, i.e., Yo e Y1.

En la Fig. VII.7.2 se muestra la modificación al circuito original que permitirá a la microcomputadora dictar la altura (Y) a la cual el motor se detendrá. El voltaje Eref aún estará aplicado al potenciómetro (transductor de posición) como antes, pero solamente una fracción de Eref está aplicada al comparador.

En este circuito (Fig. VII.7.2) la microcomputadora ordenará la posición en la que el motor deberá detenerse, esta posición estará representada por un valor digital en el puerto de salida, el cual está conectado a un convertidor D/A cuya salida se conecta a la entrada del comparador. En la otra entrada del comparador se conecta el cursor del potenciómetro en donde, como se dijo antes está presente un voltaje equivalente a la posición. La salida del comparador se conecta a un bit del puerto de entrada de la microcomputadora.

El voltaje de salida del convertidor D/A es :

$$Esal = \frac{A}{2^n} Eref \qquad (VII.7.1)$$

donde Esal es el voltaje de salida del convertidor, Eref es el voltaje de referencia, n es el número de bits del convertidor y A es el valor de la palabra digital que se aplica a la entrada del convertidor y representa la posición de apagado del motor.

Debido a que Y es proporcional a E y una fracción de Eref y, Esal también es una fracción de Eref, se puede establecer la altura a la cual la salida del comparador vaya a nivel "alto" (indicando a la microcomputadora detener el motor) sustituyendo A en la ecuación (VII.7.1).

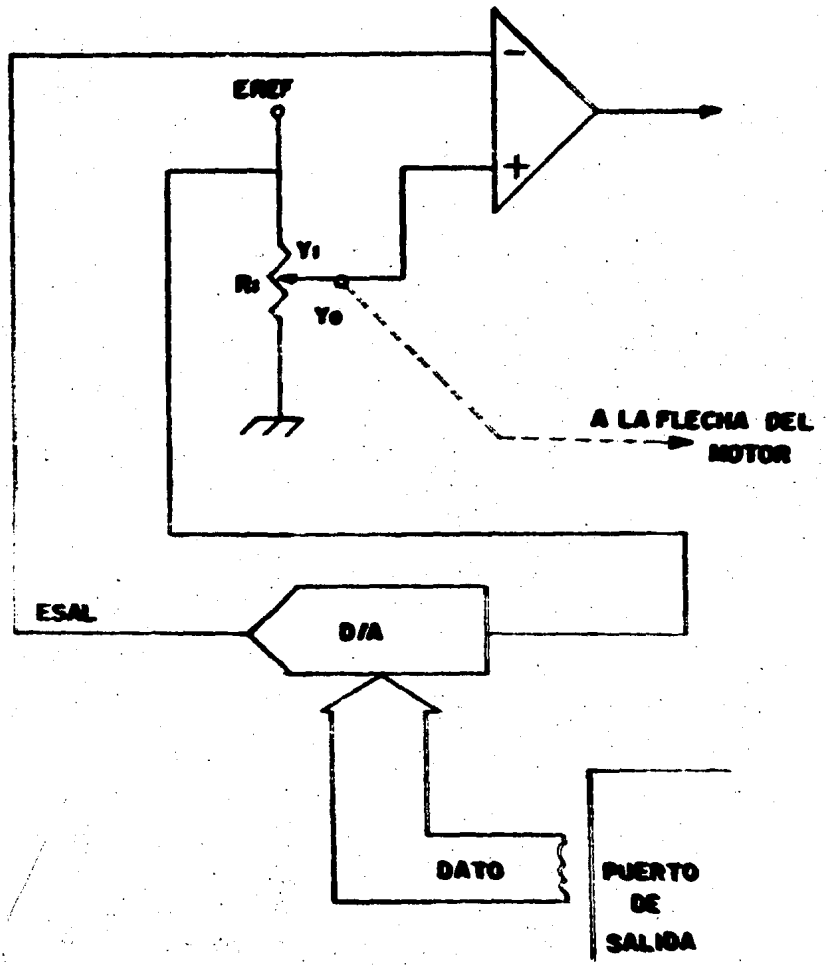


FIG. VII 7.2. CONTROL PROPORCIONAL DE UN MOTOR DE CD.

Para resolver el problema de bajar la carga una vez que ha alcanzado cierta altura, se presenta el circuito de la Fig. VII.7.3 que maneja al motor para dar marcha hacia adelante o hacia atrás. Se debe notar que para que este circuito funcione apropiadamente necesitan dos fuentes, Vcc(+) y Vcc(-).

Si se aplica un potencial positivo a la entrada del circuito (punto A en la Fig. VII.7.3) el transistor Q2 se polarizará en inversa y el Q1 en directa, haciendo que el potencial aplicado al motor (punto B) sea positivo y gire hacia adelante. Para invertir la dirección de rotación del motor es necesario invertir la polaridad en punto B. Esto se hace aplicando un potencial negativo al punto A, con lo cual el transistor Q1 es polarizado en inversa y Q2 en directa.

En el circuito de la Fig. VII.7.3 se han usado los interruptores CMOS 4016 y 4066 para suministrar las entradas positiva y negativa. La entrada positiva se conecta cuando la línea de control de S1 es "alto", mientras que el potencial negativo se aplica si la línea de control de S2 es "alto".

Se puede controlar la velocidad del motor si al punto A del circuito de la Fig. VII.7.3 se conecta la salida de un convertidor D/A en lugar de las fuentes V(+) o V(-). Si el voltaje de salida del convertidor es bipolar, entonces la velocidad y dirección de rotación del motor pueden controlarse por las órdenes dadas desde el puerto de salida de la microcomputadora.

Un servomecanismo controlado continuamente mediante un motor de CD, en sus versiones analógica y digital, se muestra en la Fig. VII.7.4. Estos circuitos emplean la técnica de control de retroalimentación negativa (malla cerrada) y ambos pueden describirse por la ecuación :

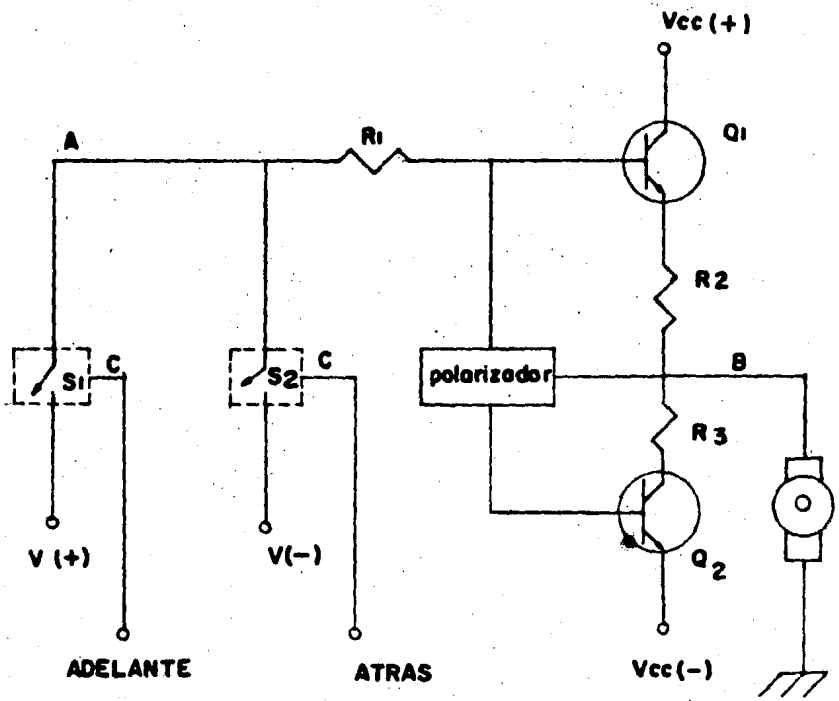


FIG. VII. 7.3. CIRCUITO MODIFICADOR DE POLARIDAD.

$$\frac{Esal}{Een} = \frac{H}{1 + HB} \qquad \text{(VII.7.2)}$$

donde Esal es el voltaje de salida, Een el voltaje de entrada, H es la ganacia de malla abierta y B es la ganancia de la retroalimentación.

En los circuitos de la Fig. VII.7.4 el valor de H es la ganancia del servoamplificador mientras que B es la unidad, es decir, no hay ganancia ni atenuación en la retroalimentación.

La versión analógica del circuito está construída de amplificadores operacionales y transistores de potencia. El sumador es un amplificador diferencial debido a que solamente se necesitan dos señales de entrada para producir la señal de error o diferencia.

El motor es manejado por la salida del servoamplificador (Em), este voltaje es el producto de la ganancia del servoamplificador y el voltaje de error Ee, esto es:

$$Em = AvEe \qquad \text{(VII.7.3)}$$

donde Ee es la diferencia entre la señal de posición Ep (dada por el transductor) y la señal de control Ec que indica cual debe ser la posición. Ec es suministrada por la microcomputadora a través de un convertidor D/A.

$$Ee = Ec - Ep \qquad \text{(VII.7.4)}$$

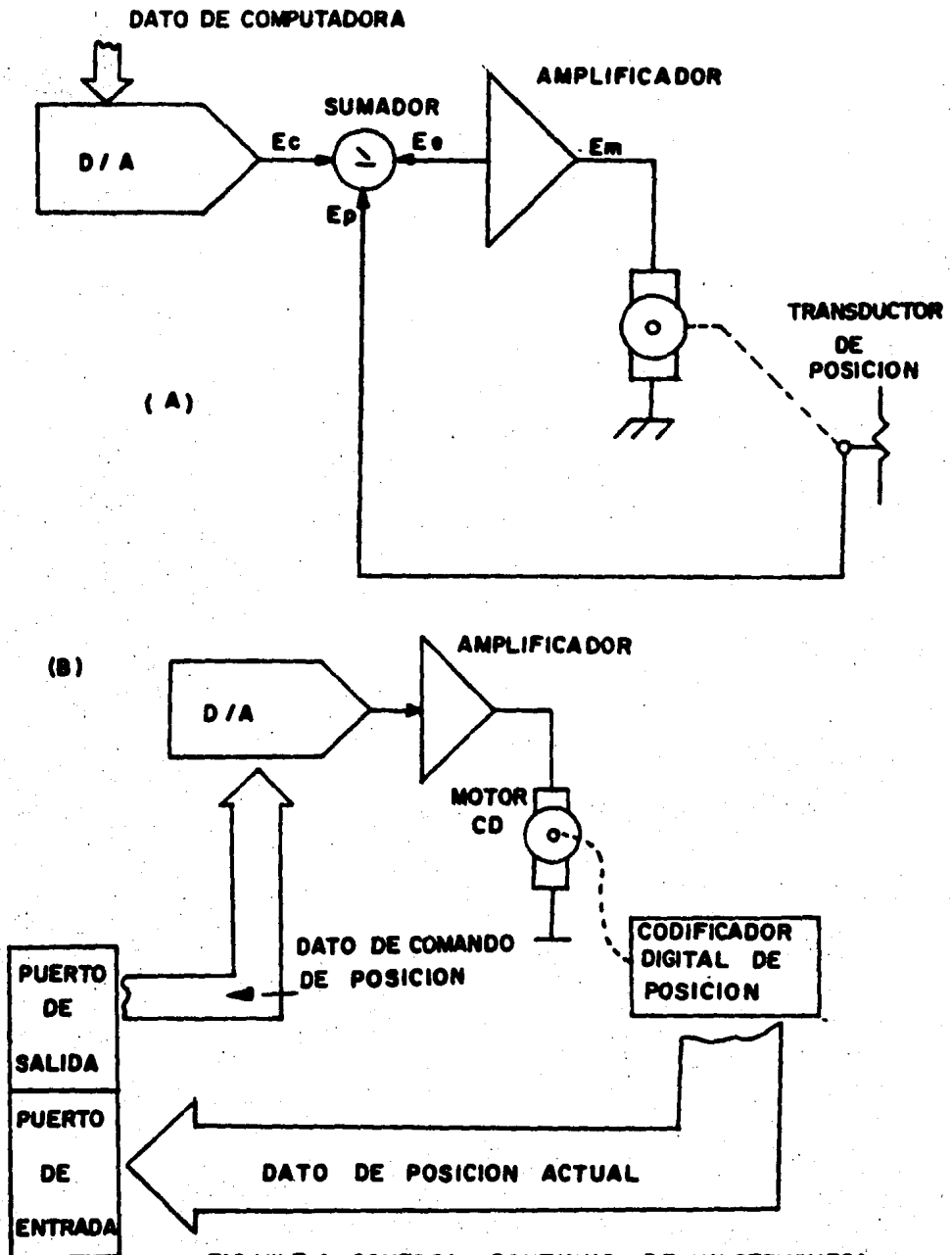


FIG. VII.7.4. CONTROL CONTINUO DE UN SERVOMECA - — NISMO (A) METODO ANALOGICO (B) DIGITAL.

de tal manera que:

$$Em = Av\ (Ec - Ep) \qquad (VII.7.5)$$

Si la posición es correcta, entonces Ec = Ep y por la ecuación (VII.7.5), Em es 0 y el motor se apaga. Pero si Ec es distinto a Ep, el motor se enciende. Debido a que el motor es de CD su velocidad de operación depende de Em y claramente puede notarse que mientras mayor sea la diferencia entre las posiciones correcta y actual, mayor será el valor de Em y por lo tanto, la velocidad del motor.

La velocidad de respuesta se establece mediante la amplificación o ganancia del sistema y por la respuesta en frecuancia. Si la señal tiende a sobreamortiguarse (muy lenta) incrementa la respuesta en frecuencia del sistema; si es sobreamortiguada aparece el sobrepaso pero reduce la respuesta en frecuencia del sistema.

La versión digital se muestra en la Fig. VII.7.4.8. En este caso el transductor de posición se indica como un bloque debido a que puede ser un potenciómetro con un convertidor A/D, o uno de los tranductores (discos) codificados digitalmente del capítulo VI. En esta versión, la comparación entre las posiciones correcta y actual se hace por software. En un sistema de 8 bits hay 256 estados diferentes para representar puntos de posición. La microcomputadora compara la palabra que indica la posición actual con la palabra de la posición correcta, ejecutando una resta binaria análoga a la ecuación (VII.7.4). Ambos sistemas son esencialmente autoregulados.

C A P I T U L O V I I I

DISEÑO DE SOFTWARE EN TIEMPO REAL (STR)

VIII.1 INTRODUCCION

Vistos los capítulos anteriores, la culminación de un sistema en tiempo real se lleva a cabo en la implementación tanto del hardware como del software.

Hasta aquí, se considera que los requerimientos del hardware se han visto casi en su totalidad, en un nivel general y aisladamente.

En este capítulo se mencionan los elementos básicos del software en tiempo real. Primeramente mencionando las partes importantes y características del sistema operativo ideal, después introduciendo el concepto de máquina virtual, en la cual a partir del hardware básico, los diferentes niveles de software (monitor, ensamblador, superlenguajes, etc.) van conformando esta máquina cada vez más poderosa.

El entender el concepto de máquina virtual y más aún, contar con ella, da la pauta a un mejor diseño y manejo de un sistema de tiempo real. Sus principales aplicaciones son aquellas donde el control y las rutinas de servicio en la computadora se vuelven sumamente complicadas, y por ende, el sistema se convierte en un sistema sofisticado. No obstante, su poderío puede sobrado para sistemas sencillos, es válido decir que los programas de aplicación se reducen a niveles francamente pequeños.

Posteriormente, conjugando los conceptos anteriores, se hace un diseño de comunicacioón entre dos computadoras.

Por último las dos siguientes secciones analizan primero, el diseño de software en tiempo real en lenguaje ensamblador y, segundo una comparación entre los superlenguajes y los lenguajes a nivel ensamblador.

VIII.2 EL SISTEMA OPERATIVO

Algunas tareas de tiempo real pueden ser ejecutadas únicamente por programas que controlan directamente la computadora sin intervención del sistema operativo. Sin embargo, mientras las tareas puedan soportar ligeros retardos en su ejecución, es recomendable hacer uso de los sistemas operativos estándares disponibles actualmente.

El sistema operativo "ideal" tendrá capacidades de tiempo real, multiusuario y multitareas. Incidentalmente, la capacidad de multiusuario es ventajosa sobre los sistemas de un solo usuario, debido a que permite la ejecución de varias tareas simultáneas.

Unix es un ejemplo de un excelente sistema operativo multitareas, aunque para muchas tareas de tiempo real no es el mejor, existen Unix modificados y sistemas compatibles con Unix, que sí proveen amplias facilidades para tiempo real. Un ejemplo es el HP/Unix, el cual tiene un núcleo en tiempo real escrito por Hewlett Packard que se comunica con Unix a un nivel de llamadas del sistema (system-call). Otro ejemplo es el sistema operativo Unos, y en microcomputadoras existe el Uniflex de Technical Systems Consultans, el Cromix de Cromemco, el

Para aplicaciones de tareas en tiempo real, el sistema operativo debe permitir al programador tomar control de dispositivos de E/S, memoria física, prioridades de tareas, procesamiento de excepciones e integridad de datos.

VIII.2.1 Control sobre dispositivos de E/S

Un programador debe controlar los dispositivos de E/S para monitorear y controlar los periféricos de la computadora. Para esto el sistema debe proveer facilidades de manejo de interrupciones y tener la habilidad de regresar el control a la tarea especificada.

VIII.2.2 Control sobre memoria física

El control sobre la memoria física es necesaria para evitar SWAPPING (intercambios automáticos de información entre la memoria y dispositivos de almacenamiento masivos) y controlar el flujo de datos. El swapping provoca retardos significativos cuando la alarma de fuego suena y la tarea de servicio no puede ser "bajada" rápidamente a memoria. El sistema operativo ideal debe permitir al programador inhibir el swapping e identificar y controlar las áreas de memoria física que sirven para almacenar datos (buffer) de los dispositivos de E/S. El compartir estas áreas entre tareas (ver seccion VIII.3.2.2 Comunicación) puede ser escencial cuando una tarea ejecuta la entrada de datos, una segunda agrupa los datos, la tercera graba los datos normalizados a disco, la cuarta analiza los datos para evaluacion estadística y una quinta usa las estadísticas para controlar un dispositivo externo. El uso de memoria compartida y operaciones asíncronas son necesarias para un trabajo eficiente.

VIII.2.3 Control sobre prioridades

Por otro lado, el programador debe tener control de las prioridades de las tareas, para asegurar que las tareas de tiempo real no serán detenidas por causa de otras tareas de menor importancia. Los mecanismos de control de prioridades permiten al programador decidir y controlar cuál tarea "tomará" el procesador. En un sistema operativo de tiempo real, a cada tarea se le garantiza que en el peor de los retardos, tendrá el tiempo suficiente para completar su trabajo.

VIII.2.4 Programación de excepciones

Debido a que la naturaleza de los sistemas en tiempo real no están a salvo de condiciones de error, es necesario contar con un nivel adicional de control asíncrono por el programador (ésto se conoce como procesamiento de excepciones).

En tareas de control interdependientes, la pérdida de control puede resultar fatal y además que la aplicación no concluya con éxito. Esto significa que el programador debe verificar todas las posibles condiciones de error relacionadas con la operación del programa, tales como errores de E/S y errores de "overflow/underflow" aritméticos.

Además, es necesario "atrapar" otro tipo de errores no concernientes a las operaciones del programa, como son errores en la línea de comunicaciones, errores de disco y/o memoria y fallas de energía.

VIII.2.5 Integridad de datos

La integridad de datos es otro factor sobre el cual el programador debe mantener un riguroso control. Una falla de energía es el único ejemplo de un evento externo que provoca un impacto en la

integridad de datos. Por lo tanto, para procesos en tiempo real, es necesario un mecanismo para determinar que datos han sido perdidos y cuales permanecen aún válidos. Este mecanismo debe formar parte del sistema operativo.

VIII.2.6 Despachador y colas

Dos aspectos críticos en un medio de tiempo real son uno, la naturaleza aleatoria de eventos físicos y dos, la ocurrencia simultánea de procesos físicos. Debido a esto el manejo de interrupciones y propiedades de multitareas son los principales atributos de un sistema operativo en tiempo real. Es decir, el corazón del sistema operativo debe ser el mecanismo de manejo de multitareas: el Despachador.

El resto del sistema operativo, se establece alrededor de este núcleo y sirve a las demandas específicas del medio de aplicación.

En particular, las listas o colas y sus manejadores que están alrededor del despachador, son construídos para comunicarse con los diferentes recursos físicos soportados por el sistema operativo. Es decir, una cola puede contener aquellas tareas que están listas para ser ejecutadas en el procesador (p.e. procesos o programas), otra cola puede tener tareas que esperan accesos al hardware de E/S, y otra cola puede contener tareas que esperen por algun evento específico a ocurrir.

En cualquier sistema operativo multitareas, el despachador usa las colas como entrada. Su salida, por otro lado, es una simple tarea que ha sido activada y permitida para ejecutarse en la unidad central de proceso. El algoritmo de despachador define en gran parte al sistema operativo.

En un sistema, el despachador simplemente selecciona la primera tarea a ser ejecutada, permitiendole que corra hasta que termine o hasta que consuma un periodo específico de tiempo. Este tipo primitivo de algoritmo fue muy común en grandes computadoras corriendo en BATCH.

En sistemas un poco más sofisticados, que pueden ser usados interactivamente a través de terminales, el despachador puede seleccionar tareas en el esquema ROUND-ROBIN (ver Fig. VIII.2.6.1), y permite a cada una un tiempo específico de proceso. Una vez que la tarea excede el tiempo permitido, es colocada al final de la cola y forzada a esperar la ejecución de todas las tareas que vayan adelante de ella.

El esquema Round-Robin con periodos de ejecución iguales, es adecuado si cada tarea no es más importante que cualquiera de las otras. Sin embargo, si se requiere que alguna tarea tenga mayor prioridad, es necesario un algoritmo de despachador más sofisticado, es decir, un algoritmo que reconozca que unas tareas son más importantes pero, que tampoco excluya de la unidad de proceso las tareas de baja prioridad.

Una solución es el uso de varias colas, donde el lapso de tiempo de ejecución este relacionado a la prioridad de los elementos de la cola.

En este caso, el despachador deberá permitir a todas las tareas de cada cola de prioridad diferente, el acceso a ejecución, siempre que a las de menor prioridad les asigne un tiempo de ejecución menor.

Un refinamiento adicional permitirá que las tareas de mayor prioridad suspendan a la tarea en ejecución. Esta técnica llamada de pre-vaciado, es una importante característica para sistemas de tiempo real (ver Fig. VIII.2.6.2).

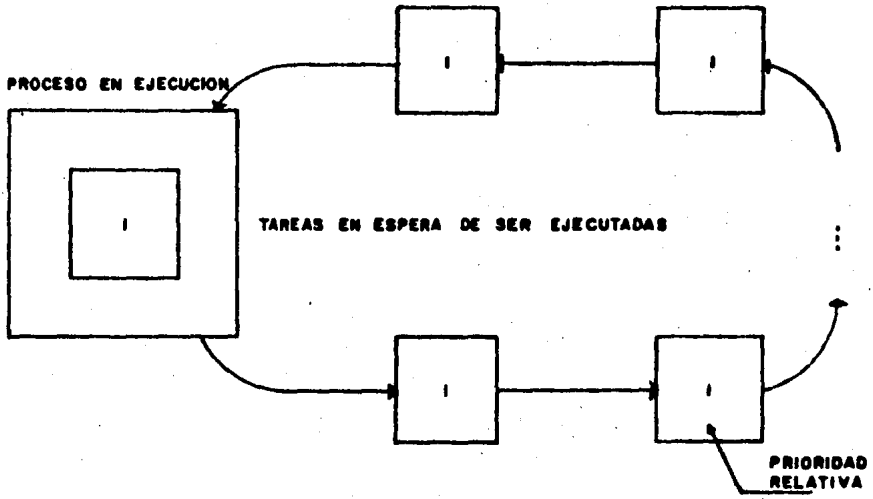


Fig: VIII.2.6.1 Esquema ROUND - ROBIN

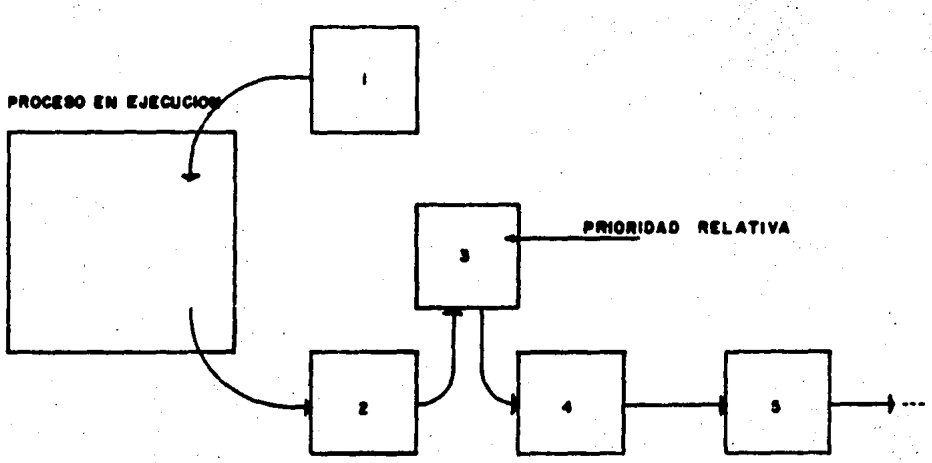


Fig VIII.2.6.2 Esquema de PRE-VACIADO, basado en prioridades

VIII.3 LA MAQUINA VIRTUAL

Como primera instancia, el diseño de software en tiempo real (STR), surge como una necesidad de desarrollar una tarea extremadamente difícil, ya que por una parte se cuenta con un problema bién definido y complejo (p.e. industria) y por el otro una computadora que solo es capaz de procesar instrucciones.

Otro problema que pasa el diseñador de STR es la máquina donde va a implementar dicho software. Es decir, no existe una máquina que cuente en su nivel básico de software, con instrucciones que interpreten la simbología del STR, la solución a esto es construir una máquina virtual adecuada.

Esta máquina virtual deberá contar con niveles de software entre el usuario y el hardware, estas capas de software van creando una máquina virtual sucesivamente dando paso a un nivel de abstracción cada vez más sofisticado, que permitirán facilitar y acercarse más a los requerimientos del diseñador. Cada capa de software que conforma un nivel de una máquina virtual, automáticamente traslada los datos y las acciones en otra capa que ejecuta las mismas acciones que la anterior pero en forma más complicada y más compleja para el diseñador.

Este traslado automático de abstracciones de alto nivel a operaciones detalladas de bajo nivel repercute en una reducción significativa de la complejidad del sistema de STR.

Si esta máquina virtual no existe, entonces el diseñador deberá ser cuidadoso en las operaciones detalladas del hardware y por ende desviará su objetivo de resolver su problema original en resolver problemas de aprendizaje del hardware a un nivel muy detallado (ver sección VIII.5).

Desde el punto de vista del problema o sistema a ser diseñado, la especificación formalizada y técnicas de diseño proveen al diseñador con una descripción más estructurada y mejor definida del problema.

Desde el punto de vista del hardware, las capas sucesivas de software proveen al diseñador con una máquina virtual más y más conveniente.

Esto le deja al diseñador, crear un mapeo entre la especificación de requerimientos y los datos y acciones requeridos por la máquina virtual. Este trabajo no es sencillo, sin embargo es menos complejo que tratar de mapear el problema directamente en el hardware.

Es decir, imagínese un científico que desea crear un programa simple para calcular algunas operaciones. Su problema es diseñar el software que resuelva una serie de ecuaciones y fórmulas. Sin un soporte de software (provisto por una máquina virtual), este científico se tendrá que concentrar en trasladar las ecuaciones en un gran número de instrucciones en lenguaje de máquina. Un problema que puede ser expresado convenientemente en pocas líneas de fórmulas matemáticas, viene a ser un problema complejo en código máquina. Es lógico suponer lo que pasaría si este científico contara con un lenguaje de alto nivel, por ejemplo FORTRAN.

Si se traslada este problema al diseñador de STR, es claro que deseará una máquina virtual enfocada a su medio.

Se enfocan los siguientes párrafos a describir las propiedades de una máquina virtual.

VIII.3.1 Propiedades de la máquina virtual en tiempo real

En primera instancia un sistema de tiempo real realiza un conjunto de actividades o tareas. Por ejemplo, controlar un dispositivo puede ser una tarea y, mantener un registro para su uso posterior en el sistema puede ser otra.

Todas las tareas en el sistema estan interrelacionadas e interactúan una con otra. Algunas tareas pueden ser más vitales que otras (dependiendo de las condiciones del medio), afectando el sistema controlado a cualquier tiempo.

Las variaciones contínuas del medio hacen difícil predecir cual tarea dentro de un conjunto de tareas disponibles, será requerida para ser ejecutada por el sistema de tiempo real. Es aquí donde se presenta un conjunto de tareas interdependientes, todas llevadas a ejecutarse dentro de estrictos límites de tiempo. A menos que una forma de abstracción se pueda aplicar, el diseño de tareas será imposible de realizar.

Un método de lograr esta abstracción es diseñando, creando y usando una máquina virtual que adopte esta filosofía. Si esta máquina puede ser desarrollada para un medio de tiempo real, entonces el problema de diseño puede ser reducido a proporciones manejables.

Así, se planteará la máquina virtual que puede proveer las bases necesarias para el desarrollo de sistemas en tiempo real. Se ha visto hasta ahora que la existencia de una máquina virtual permite al diseñador obtener un grado de abstracción a partir del hardware (computador).

Sin embargo, esta máquina virtual debe hacer más que simplemente reducir la programacion compleja. Debe poder aceptar las necesidades

de su medio y área de aplicación, debe ser diseñada de tal manera que facilite y fomente la creación de diseños claros, lógicos y de fácil implementación, además del diseño de sistemas tan simples como sea posible.

Es con este criterio en mente con el que se discutirá el diseño de la máquina virtual de tiempo real.

VIII.3.2 Facilidades que requiere la máquina virtual

Es importante plantear (en primera instancia) a un nivel básico los requerimientos que debe cumplir la máquina virtual. Se tratará esto desde el punto de visto software y en diagramas de bloques.

VIII.3.2.1 Procesos

Un sistema de tiempo real puede ser visto como aquél que realiza un conjunto de tareas ó actividades. Un sistema de tiempo real en su versión más básica realizará una y solo una actividad, donde esta actividad se define como la combinación de un programa residente simple y el hardware de procesamiento. Esta dctividad será ejecutada por un PROCESO.

Visualizando gráficamente a este proceso se vería como la Fig. VIII.3.2.1.1. Suponiendo que este proceso se trata del control de una válvula de una tubería de gas, en la cual se sensa el flujo del mismo.

Visualizando ahora el proceso por completo se ve que el control de esta válvula de gas regula la cantidad de combustible en una caldera la cual a su vez reporta una presión "x" para un uso "y". Ver Fig. VIII.3.2.1.2.

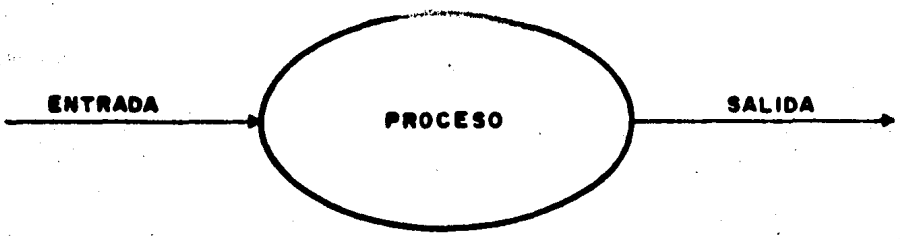


FIG. XII. 3.2.1.1 SIMBOLO DEL PROCESO

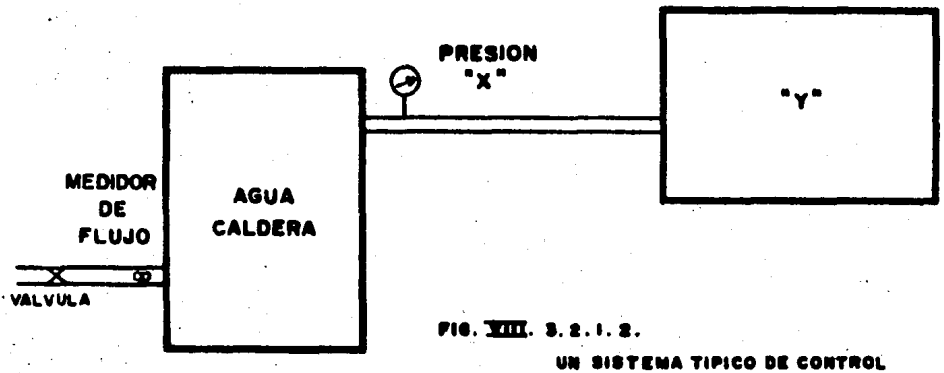


FIG. XIII. 3.2.1.2.
UN SISTEMA TIPICO DE CONTROL
CON PROCESOS INTERDEPENDIENTES

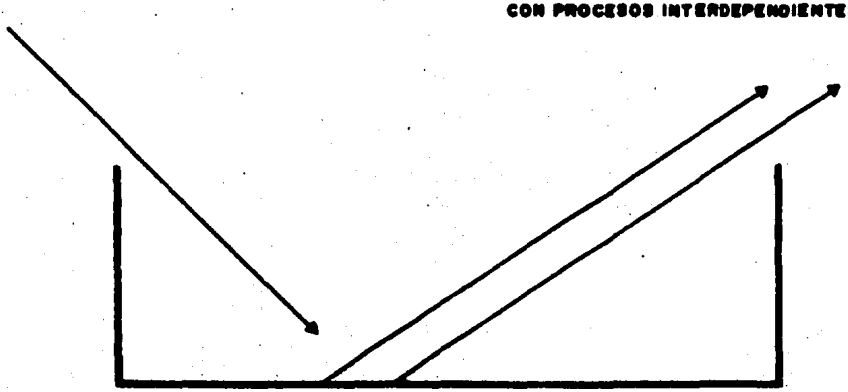


FIG. XIII. 3.2.2 LA CHAROLA

Viendo esto gráficamente como procesos independientes en la Fig. VIII.3.2.1.3. Se observa, que esta representación indica tres procesos independientes, es decir, las tareas que los controlan no tendrán efectos una sobre otra, más aún, no existe comunicación entre ellos.

Partiendo de esto, la máquina virtual debe considerar una interacción entre ellos, pero esta interacción no debe ser desordenada, es decir, debe existir una perfecta sincronización de un evento a otro: p.e. cuando el proceso de control de presión indique una señal crítica, debe interactuar inmediatamente con el proceso de control de válvula, es claro pues que se debe definir la comunicación entre procesos.

VIII.3.2.2 Comunicación

La comunicación entre procesos puede ser clasificada en dos etapas:

- Transferencia directa de información de un proceso a otro.

- Cada proceso puede accesar o actualizar información compartida entre ellos. Donde ésta información puede ser disponible a alguno o todos los procesos.

Para poder proveer información directa entre procesos en la máquina virtual, se introduce el concepto de canal (channel o pipe), donde éste canal proporciona el medio por el cual un proceso puede enviarle información a otro proceso con la filosofía de que más de un "artículo" (donde el artículo es la parte más básica de informacion) puede ser transferido entre procesos y además de que el último artículo en llegar al canal es el último artículo en salir de él (FIFO).

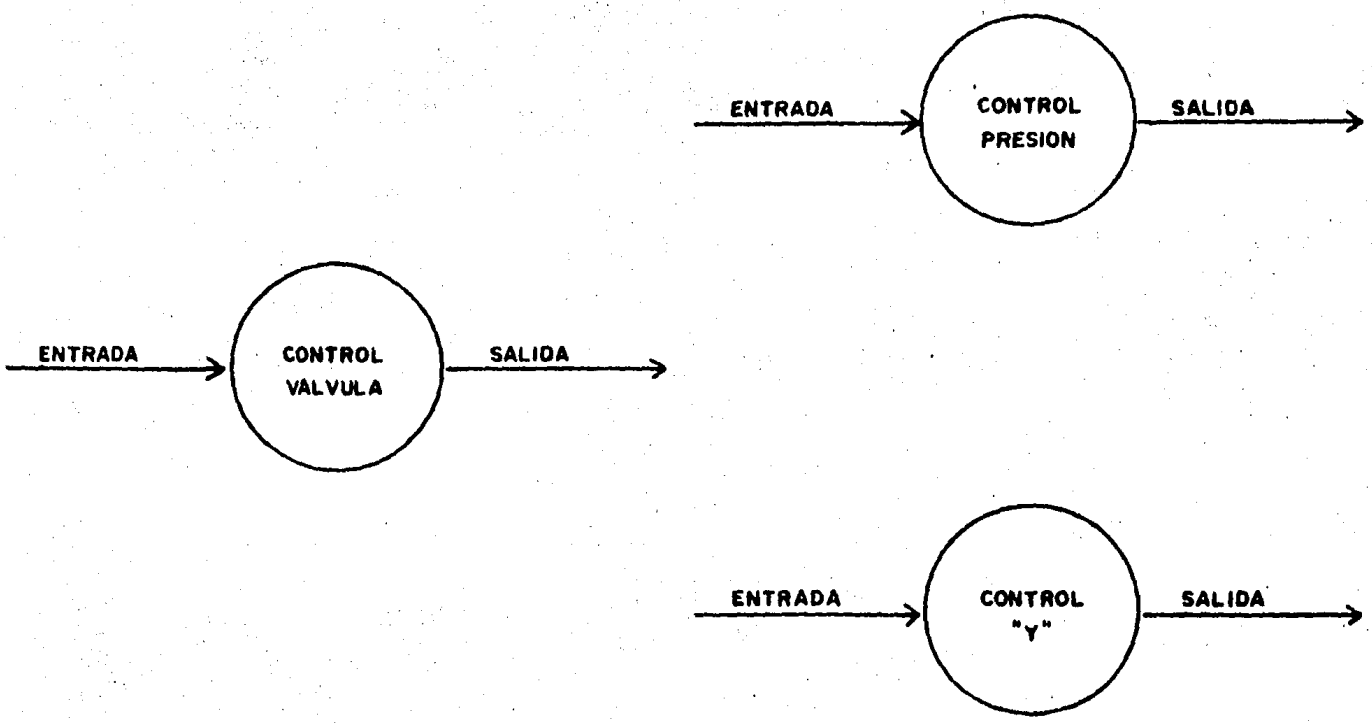


FIG. VIII. 3.2.1.3. SISTEMA CON PROCESOS AISLADOS

Por otro lado para proveer información que pueda ser accesada por uno o varios procesos, se introduce el concepto de "charola" o "receptáculo" (ver Fig. VIII.3.2.2), donde cada artículo de información en la charola puede ser leído o escrito por los procesos.

Las charolas forman parte importante de la máquina virtual, ya que de la misma manera que los procesos son modelos de las actividades que un sistema en tiempo real debe ejecutar, las charolas también forman los modelos de los artículos en el sistema.

Cabe aclarar que estos artículos pueden ser dispositivos físicos y mecanismos en el sistema controlado, o bien artículos conceptuales como pueden ser archivos lógicos.

VIII.3.2.3 Sincronización

Otro concepto importante de la máquina virtual es la sincronización entre procesos. Esta sincronización engloba la habilidad de que un proceso inhiba o estimule a otro proceso o inclusive a él mismo. En otras palabras, poder contar con capacidades de detener, parar momentáneamente o hacer continuar el proceso involucrado.

Estas habilidades se pueden simular en dos formas:

- Eventos significativos

- Interrupciones

Para el primer caso, la sincronización entre procesos gira alrededor de la ocurrencia de un evento significativo en el sistema. Es decir, suponiendo que un proceso necesita suspenderse por un tiempo

hasta que otro proceso active una válvula, se ve claramente que el evento significativo es la acción de activar la válvula, así que el primer proceso debe "esperar" a que el evento ocurra, mientras que el segundo debe "avisar" que el evento ha ocurrido.

Para esto Dijkstra (1968) introdujo el concepto de "semáforo", el cual es un simple dato que puede tomar solo valores enteros no negativos y ser manipulado por tres procedimientos: "inicializa" (semáforo, valor), "espera" (semáforo) y "avisa" (semáforo); "inicializa" establece el valor inicial al "semáforo", "avisa" simplemente incrementa el valor del "semáforo" en 1, mientras que "espera" decrementa el valor de éste, únicamente si el resultado no es negativo.

Los procesos que desean sincronizar sus actividades ejecutan operaciones de "espera" y "avisa" en "semáforos" compartidos, es decir, si un proceso ejecuta una operación "espera" y el valor del "semáforo" es "1" o mayor, entonces el proceso puede decrementar el semáforo y continuar. Si, por otro lado, el "semáforo" tiene un valor de "cero" en el momento en que se ejecuta la operación "espera", entonces el decremento del "semáforo" resultaría negativo. Esto implica (dado que no puede tomar valores negativos), que el proceso debe esperar hasta el momento en que otro proceso ejecute una operación "avisa" en el "semáforo", permitiéndole el decremento al "semáforo" y continuar.

A cualquier momento, el valor de un "semáforo" es igual a su valor inicial, más el número de operaciones "avisa" que han sido aplicadas, menos el número total de operaciones "espera" que han sido completadas (esto es, aceptadas). Debido a que el valor del "semáforo" no puede ser negativo, esto implica que el número de operaciones "espera" completadas en un "semáforo" siempre será menor o igual al valor inicial del "semáforo" más el número de operaciones "avisa" que han ocurrido.

Ambas operaciones (avisa y espera) son operaciones indivisibles, es decir, una vez iniciadas deben ser concluidas. El procesador no puede ser interrumpido mientras ellas se ejecutan. Esta regla asegura que el incremento y decremento del "semáforo" ocurrirá sin interrupción, y por lo tanto las rutinas involucradas siempre estarán sincronizadas.

La acción de activar la válvula (evento) puede ser transferida por medio de un canal o una charola, es decir, el evento de "esperar" puede ser pensado como el principio de la acción de leer información de un canal o una charola; por otro lado el evento de "avisar" puede ser pensado como la acción de escribir en el canal o charola, que la válvula ha sido activada.

Para el segundo caso, el evento de "avisar" puede no ser ejecutado por un proceso en el sistema, aquí el diseñador puede preferir que la válvula por sí sola avise de que ha sido activada. Esta característica es en general una interrupción de haruware.

Hasta aquí se pueden resumir las propiedades que debe cumplir la máquina virtual, se verá ahora un ejemplo de como conformar un sistema de información en el que un usuario puede accesar o actualizar información de un banco de datos.

VIII.3.2.4 Ejemplo

Imagínese a un pequeño sistema de información en el que por medio de una terminal una persona puede interrogar y posiblemente actualizar información en varios archivos residentes en una computadora.

En primera instancia, esta persona debe conocer el sistema y teclear su clave de acceso, una vez hecho esto puede mandar comandos

línea por línea, leer información y si es aceptado por el sistema, escribir datos a los archivos.

Para poder modelar este sistema primero se identificarán los artículos del sistema así como los procesos que los componen.

VIII.3.2.4.1 Artículos

a) Los Usuarios. Cada usuario del sistema debe tener una clave de acceso y algún código de seguridad que indique a que archivos puede tener acceso para leer o escribir.

b) Las Terminales. En este simple sistema de información cada terminal debe ser alojada en una localidad única de la memoria de la computadora, en la cual pueda colocar los caracteres que por ella son tecleados, de la misma manera existe una localidad de memoria donde ella puede accesar la respuesta del computador.

c) Los Comandos. Cada comando al sistema debe tomar la forma de una línea de texto y representar una acción. Tales acciones pueden ser "lee un registro de archivo" o "muévete al siguiente registro".

d) Las Respuestas. Estas serán una cadena de caracteres, las cuales se desplegarán en las terminales como una respuesta a los comandos.

e) Los Archivos. Estos son los archivos de información que los usuarios pueden manipular.

VIII.3.2.4.2 Actividades

a) Manejo de entradas por terminal. El sistema debe manejar cada caracter tan rápido como sea posible para no encimar los caracteres tecleados en ella, y poder formar los comandos que esten llegando. Cada comando consiste de una cadena de caracteres

terminados por una indicación de "fin de línea". Esta actividad mandará caracteres hasta encontrarse con un caracter de fin de línea, en este punto "avisará" que el comando ha sido leído.

b) Manejo de salidas a terminales. El sistema podrá construir respuestas al usuario en la forma de cadena de caracteres. Es claro que se debe proveer una actividad que coloque estas respuestas caracter por caracter en la localidad de salida de la terminal.

c). Manejador de archivos. Es necesario incluir una actividad que controle la organización y acceso a la información de los archivos del sistema.

d) Control. El sistema debe incluir tambien un intérprete de comandos, además pueda enviar las respuestas de validez de estos y poder controlar tambien la transferencia de información entre el manejador de archivos y el manejador de salidas a terminal.

VIII.3.2.5 Diseño del ejemplo

Como se mencionó anteriormente cada artículo del sistema es modelado por una charola y cada actividad por un proceso. Una posible solución se vería como la Fig. VIII.3.2.5.

VIII.3.2.5.1 Explicación del diseño

El proceso Manejador de Entradas construye los comandos caracter por caracter y los deposita en la charola de comandos. Cuando este recibe el indicador de fin de línea manda el aviso por el canal C1, indicando que el comando ha sido recibido.

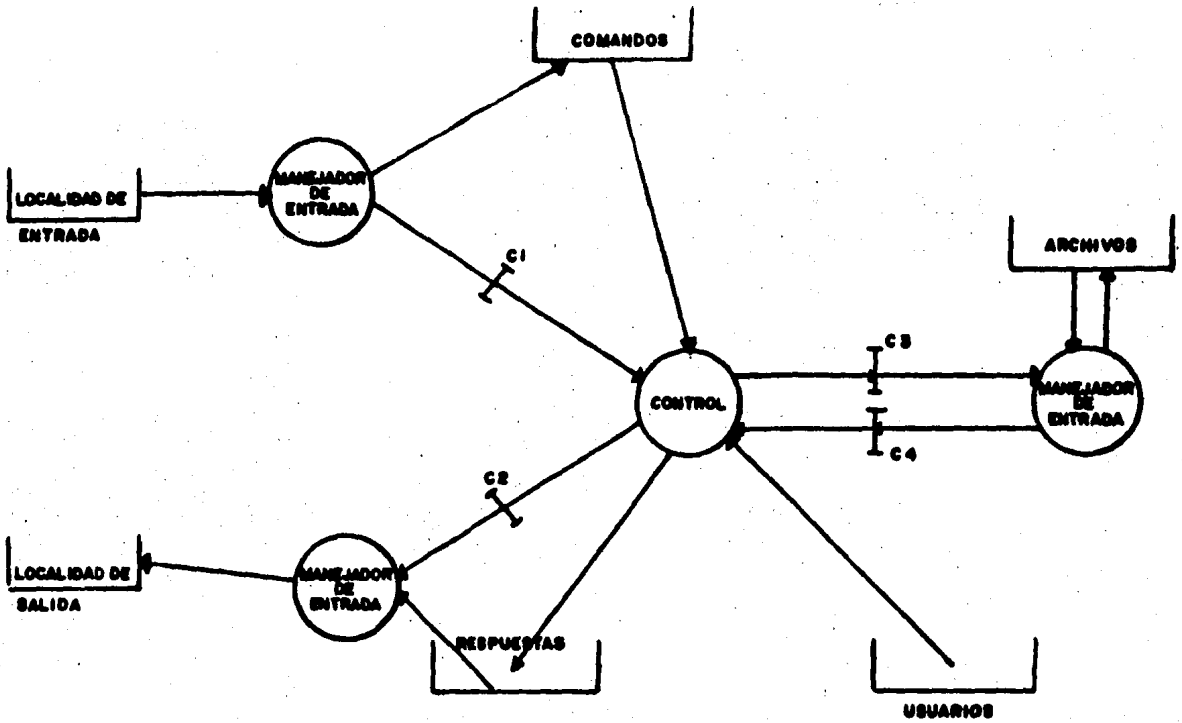


FIG. VIII. 3. 2. 5 MODELO DE UN SISTEMA DE CONDUCTA DE INFORMACION CON LOS CONCEPTOS DE MAQUINA VIRTUAL

Cuando el proceso Manejador de Salidas recibe un mensaje por el canal C2, transfiere las respuestas específicas del mensaje a la localidad de salida de terminal.

Cuando el proceso de Control recibe el aviso del proceso de Entrada, éste checa la validez del comando, consultando la charola de usuarios validando la entrada de estos y manda el aviso requerido al proceso Manejador de Archivos a través del canal C3, recibiendo por el canal C4 el reconocimiento del Manejador de Archivos. Después de esto coloca la respuesta del comando en la charola de respuestas y manda el aviso por C2.

De esta manera se ha formado un diseño de software de amplio alcance, modelando los artículos como charolas, actividades como procesos y sincronización de comunicaciones como canales.

VIII.4 DISEÑO DE UN SISTEMA DE COMUNICACION

Se considera de vital importancia llevar a la realidad los conceptos antes mencionados, para un mejor entendimiento de ellos.

Se plantea la necesidad de interconectar dos computadoras, por un lado una minicomputadora HP3000 serie 40, y por el otro, una microcomputadora SwTPc con un microprocesador M6809 y sistema operativo UniFlex, el cual soporta obviamente un compilador "C".

El principal problema de interconexión estriba primero, en que la HP3000 utiliza un protocolo software a través de sus puertos RS232 para enlazar sus terminales, mientras que la SwTPc utiliza un protocolo hardware a través de estos mismos puertos. Segundo, que cada puerto serie de ambas computadoras, generan señales propias que reconocen el encendido de sus terminales y por lo tanto el enlace no es trivial. Tercero, el diseño de los programas en tiempo real

residirán en SwTPc y tratarán de convertir las terminales de ésta en terminales de la HP3000, más aún, se podrá enviar y recibir archivos y por último poder utilizar el poderío de la impresora HP3000 para enviar listados de archivos desde la SwTPc directamente a ella.

En la Fig. VIII.4 se ve un esquema de esta conexión.

VIII.4.1 Modelado del sistema

Se modelarán primeramente los artículos y los procesos del sistema de comunicación.

VIII.4.1.1 Artículos

a) Los usuarios. Nuevamente, cada usuario debe tener una clave de acceso y algún código de seguridad en el sistema.

b) El teclado SwTPc. El sistema utiliza una localidad de memoria para recibir los caracteres tecleados.

c) La pantalla SwTPc. Existe en el sistema otra localidad de memoria donde básicamente se reciben los caracteres "eco" que vienen de la HP3000, además desde luego, de recibir las respuestas del sistema.

d) Las respuestas. Estas son generadas como una respuesta a los comandos, para ser enviadas a La pantalla SwTPc.

e) Respuestas de protocolo. Estas, por otro lado, son las respuestas generadas para el enlace HP3000-SwTPc y que serán enviadas al puerto de enlace.

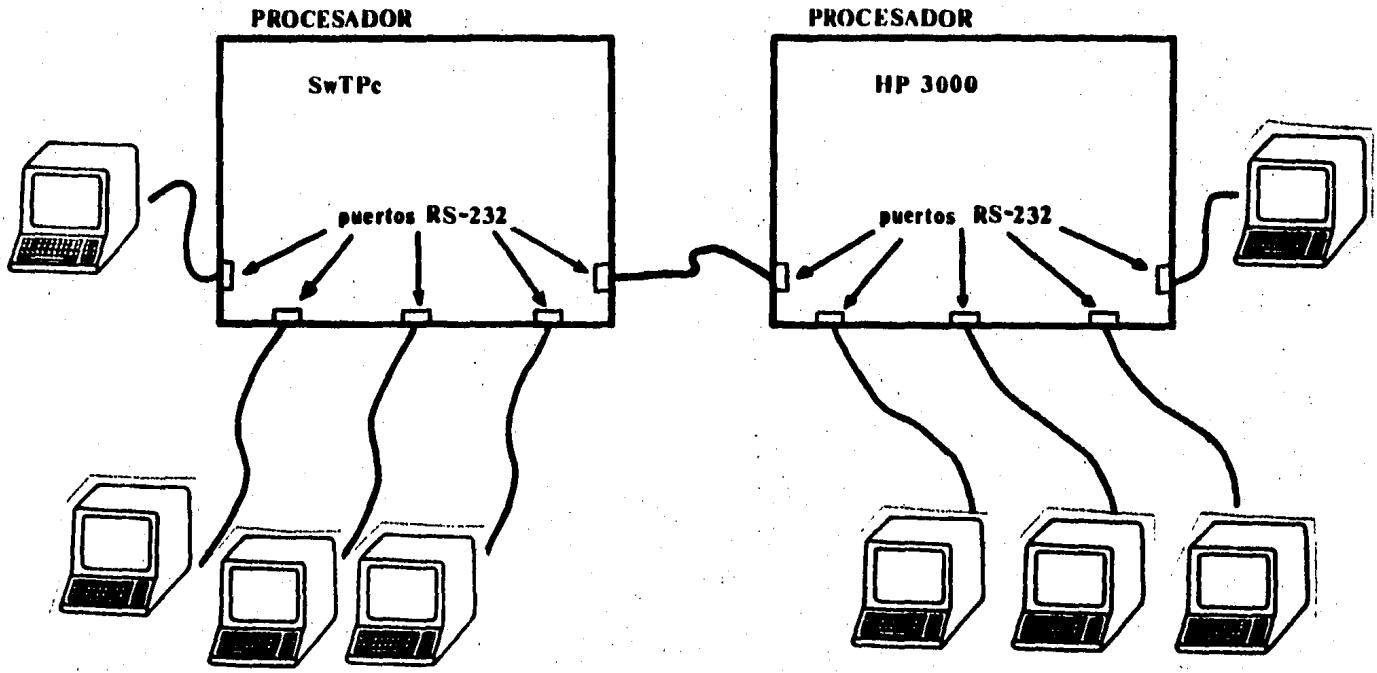


FIG. VIII.4 ESQUEMA DE CONEXION HP 3000 - SwTPc

f) Salida para protocolo HP3000. Aquí se depositan las respuestas de protocolo para ser reconocidas posteriormente por la HP3000 en el puerto de enlace.

h) Entrada de comunicación HP3000. Esta es la localidad de memoria que recibe la información de la HP3000, a través del puerto de enlace.

i) Salida de comunicación HP3000. Esta es la localidad que recibe los caracteres generados en la terminal SwTPc, para ser enviados a la HP3000 a través del puerto de enlace.

j) Los comandos. Los comandos de este sistema invocan los programas a ejecutarse y es aquí donde se depositan; estos comandos pueden ir acompañados de parámetros.

k) Los archivos. Estos son los archivos de información que los usuarios pueden manipular y que pueden ser invocados como parámetros de los comandos.

VIII.4.1.2 Actividades

a) Manejador de entradas SwTPc/salidas HP3000. Este proceso maneja cada caracter que entra por la terminal y decide si van a "Salida de comunicación HP3000" o a "Los comandos", en cuyo caso "avisa" que se trata de estos últimos.

b) Manejador de salidas SwTPc/comunicación HP3000. Este proceso maneja los caracteres de "Entrada de comunicación HP3000" y decide si va a "La pantalla SwTPc" o en caso contrario, que se trata de caracteres que conforman el protocolo, "avisando" de esto y manejando por último las "Respuestas de protocolo" para enviarlas a la "Salida para protocolo HP3000".

c) Manejador de archivos. Este proceso entra en acción cuando se pretende enviar o recibir archivos de una computadora a otra. En ambos casos se encarga del manejo de archivos de ambas computadoras.

d) Control. Esta actividad se encarga de la validez de los comandos que invocan los programas, también de la validez de apertura de archivos y canales de comunicación, más aún, genera las respuestas adecuadas a todo lo anterior y lo transfiere a los puertos involucrados.

VIII.4.2 Diseño del sistema

Como se vió en la sección VIII.3.2.5 el diseño quedaría como en la Fig. VIII.4.2.

Los programas de comunicación aparecen en el Apéndice E y están escritos en lenguaje "C". Existen en el programa "term3000" una instrucción llamada "fork()", la cuál significa que se genere una tarea idéntica, en cuyo caso es necesario identificar cual tarea es la madre y cual es la hija; esto también genera un canal de comunicación entre las ahora dos tareas que corren en paralelo.

Estos programas corren actualmente en Integra Ingeniería, S. A. sin problema alguno. El tiempo de desarrollo fué aproximadamente de tres días en su primera versión, posteriormente se les hizo algunos cambios sin modificar su estructura inicial.

VIII.4.2.1 Explicación del diseño

El proceso "Manejador de entradas SwTPc" primeramente recibe caracter por caracter y decide si son comandos, en cuyo caso los deposita en la charola correspondiente y cuando recibe un caracter de

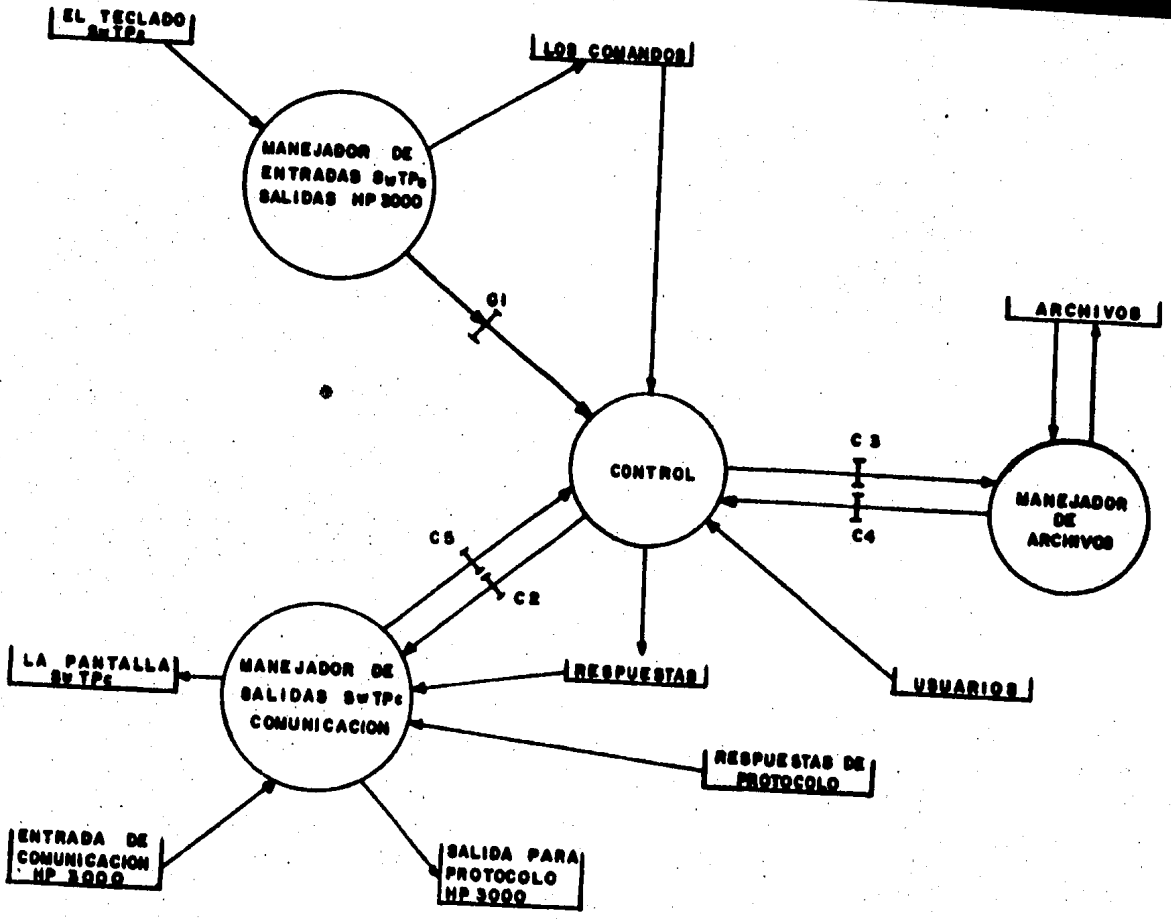


FIG. VM . 4 . 2 DISEÑO DE UN SISTEMA DE — COMUNICACION HP 3000-SwTPs

línea nueva "avisa" a "Control" a través del canal C1, que puede tomarlos, aquellos caracteres que no forman parte de un comando son enviados hacia la "Salida de comunicación HP3000" para su proceso.

El proceso "Manejador de salidas SwTPc/comunicación HP3000" es "avisado" a través del canal C2 que puede recoger "Las respuestas" y poder enviarlas a "La pantalla SwTPc" o bien recoger de las "Respuestas de protocolo" los caracteres que sirven de enlace con HP3000 y depositarlos en la "Salida para protocolo HP3000". Además recibe de la charola "Entrada de comunicación HP3000" y decide si son caracteres "eco" y enviados a "La pantalla SwTPc", o si son caracteres que forman parte del protocolo con HP3000, en cuyo caso, "avisa" a "Control" a través del canal C6.

La actividad "Control" por un lado recibe la indicación para tomar aquellos comandos que necesiten una respuesta sobre la terminal que los haya generado, esto se debe a dos razones, una en la que los parámetros de los comandos sean omitidos, y otra en la que los comandos sean correctos pero los parámetros incurran en un error, ya sea porque "El usuario" no esté registrado en el sistema, o el archivo parámetro sea un archivo no existente. Es decir, "Control" valida comandos, usuarios y existencia de archivos.

VIII.4.3 Postulados de diseño de software en tiempo real

Es claro, que no existe una metodología de diseño universal y que la decisión del metodo usado depende del criterio del diseñador, sin embargo, aquí se proponen algunos postulados para el diseño de software en tiempo real.

1) Diseñar y construir una máquina virtual que soporte procesos asíncronos.

2) Identificar todos los artículos en el sistema y expresarlos como estructuras de datos.

3) Colectar la estructuras de datos en charolas y entonces especificar los procesos que actuen en esas charolas.

4) Especificar cualquier prerrequisito de sincronización ó canales de información necesarios para la interacción correcta de procesos.

5) Usar un metodo de diseño de estructura de datos para crear las estructuras de procesos.

6) Expresar las estructuras de datos y funciones de procesos, como programas en lenguajes de alto nivel.

Si los pasos anteriores son incorporados en una estructura de diseño y ésta es aplicada a la producción de software, el resultado corresponderá cercanamente al sistema bajo control.

VIII.5 DISEÑO DE SOFTWARE EN TIEMPO REAL EN LENGUAJE DE MAQUINA

Como se ha observado hasta esto momento, no se ha mencionado el diseño de software a un nivel ensamblador o lenguaje de máquina.

En esta sección se dan las bases primordiales para el desarrollo de éste, primeramente, porque es el software más utilizado por aficionados y por otro lado es el software clásico empleado en universidades para objetivos didácticos, además, es importante mencionar que la máquina virtual no se encuentra al alcance de aficionados o universitarios y sin embargo se cuenta con equipos como Apple, Radio-Shack, Started-Kit, Cromemco (puede contar con Cromix, un sistema tipo Unix), etc., en los que las capacidades de multitareas

son nulas.

Probablemente el mayor problema con el que se enfrentan los principiantes en el diseño software en tiempo real, es como armonizar una gran variedad de dispositivos de E/S asíncronos.

Si se tienen uno o dos dispositivos conectados a la computadora, el problema es relativamente simple, sin embargo, cuando se tienen muchos dispositivos a conectar, el problema se agrava. Al mismo tiempo, si la naturaleza de los dispositivos externos es tal que no es importante cuando atenderlos, se puede crear un programa sencillo, a pesar de ser algo poco común que suceda en la práctica.

Cuando se necesite aceptar datos de un dispositivo, ya sea un teclado o un convertidor A/D, el software requerido sera diseñado de tal forma que examine el puerto de entrada con la suficiente frecuencia, para evitar la pérdida de datos.

Sea una computadora como la de la Fig. VIII.5.1, en la que, es claro que cada puerto de E/S debe ser atendido con una determinada frecuencia lo suficientemente rápida para poder atender a todos los dispositivos.

El primer análisis importante por hacer, es determinar cuales tareas deben completarse a tiempo, cuanto llevarán (en hardware y/o software), y las prioridades relativas de cada una.

Es decir, no hay que perder de vista que en diseños en máquinas que no cuentan con multitareas, cada tarea se irá ejecutando secuencialmente y es responsabilidad del software desarrollado (ó del programador), asignarle a cada una un tiempo específico de procesador, el cual garantize el éxito de cada tarea.

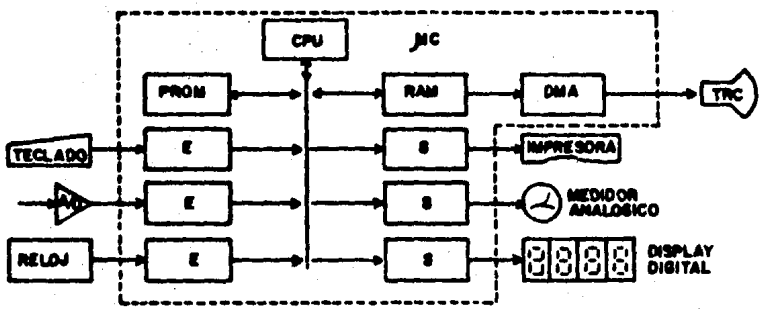


Fig. VIII.5.1 Microcomputadora con varios dispositivos de E/S

FUNCION	PERIODO (m SEG)	TIEMPO DE EJECUCION (m SEG)	PRIORIDAD
BUSQUEDA EN TECLADO	40	2	2
LECTURA DEL A/D	100	0.5	3
LECTURA DEL RELOJ	15	0.1	1
ACTUALIZACION DEL TRC	1000	55	6
IMPRESION DE CARACTERES	100	0.1	7
SALIDA AL MEDIDOR	50	0.1	4
ACTUALIZACION DEL DISPLAY DIGITAL	40	0.2	5

Tabla VIII.1 Tiempos reales y prioridades, del Sistema de la Fig. VIII.5.1

Para esto es importante determinar por un lado, la frecuencia con que se debe activar cada rutina de servicio y por otro, el tiempo que debe tomar cada rutina en ejecutarse (ver tabla VIII.1). Los datos en esta tabla se han tomado de la industria y dan una idea real de estos tiempos, por ejemplo, se establece la necesidad de revisar el teclado una vez cada 40 ms con una rutina de lectura que tome un tiempo aproximado de procesador de 2 ms. Esto significa que el teclado será examinado 25 veces por segundo, es decir, consumirá 50 ms del procesador por cada segundo.

Es claro aquí, que se debe diseñar un software que active la subrutina que maneja el teclado en al menos cada 40 ms. Esto es, en lugar de estar disponible a manejar el teclado en un tiempo arbitrario, se debe conformar a "tiempo real". Casi todos los microprocesadores son usados en aplicaciones que requieren tener conocimiento del tiempo real. Esto contrasta con equipos grandes de cómputo, en los cuales mucho (o casi todo) el software, se ejecuta sin importar el tiempo transcurrido en el reloj maestro.

De la tabla VIII.1 se observa que revisar el teclado y actualizar el display de leds, ambos toman períodos de 40 ms y probablemente estas dos funciones podrían ser activadas como una sola función. Por otro lado, es obvio que existe una disparidad entre una función que debe ser activada en períodos de 15 ms y otra que tiene un período de 1 s.

Una aparente proposición útil, se vería como en la Fig. VIII.5.2, en la que el programa Supervisor tiene la responsabilidad de decidir cual función se debe activar. Aquí, cada función es implementada como una o más subrutinas; el Supervisor procesa una tabla que especifica cual subrutina debe activar, sin embargo, es importante notar que cada subrutina no debe tardar un tiempo mayor que el período más corto en el sistema, es decir, en este ejemplo, ninguna subrutina debe tomar un tiempo mayor de 15 ms.

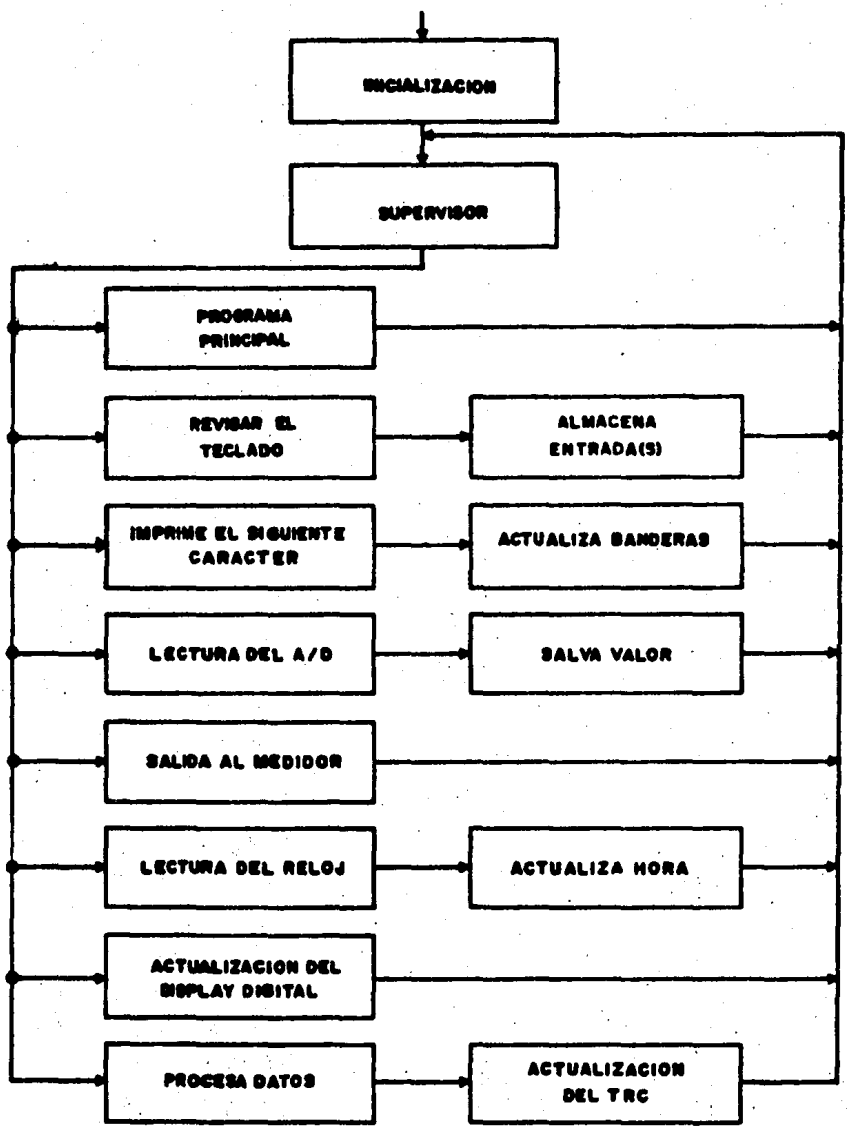


FIG. VIII. 3. 2 ESQUEMA DE UN SISTEMA EN TIEMPO REAL BASADO EN UN SUPERVISOR

Como se puede observar, esto último es una limitación muy severa en algunas funciones, es mas, en general puede resultar imposible de implementar.

Tambien existen otros problemas, por ejemplo, todas las subrutinas son independientes, aún en sus interfases de E/S, es por esto que deben ser capaces de comunicarse entre ellas, es decir, cada subrutina puede ser un productor de datos de un tipo y un consumidor de datos de otro tipo. Entre el productor y el consumidor debe existir un acuerdo necesario del formato de datos, valores disponibles y localidades. Todas estas convenciones de liga deben ser definidas para cada subrutina de tal manera que cada una de ellas pueda ser escrita para conformar los acuerdos comunes.

VIII.5.1 Interrupciones y encuestas (polling)

Existen básicamente dos maneras de permitir la E/S de eventos múltiples asíncronos, los cuales ocurren con aparente simultaneidad en el microprocesador.

Si la computadora cuenta con un esquema de interrupción, cualquier programa puede ser diseñado para ser suspendido mientras otro de mayor urgencia tome poder del procesador. Si este esquema no existe en la computadora o quizás, si no se desea usarlo, entonces es necesario una revisión de estados externo o encuesta de estados que permita obtener el mismo resultado que en el esquema de interrupciones.

El esquema propuesto en la Fig. VIII.5.2 puede ser modificado para aislar las subrutinas dependientes del tiempo de aquellas independientes del tiempo. Aquí, el Supervisor puede encuestar entre las rutinas independientes hasta realizar las tareas del microprocesador. Cada tarea es diseñada para ejecutarse en un período

de tiempo máximo el cual es menor que el tiempo requerido por la frecuencia más alta de una subrutina de E/S. Al final de cada ejecución de subrutina, el control es regresado al Supervisor, en este punto el Supervisor decide en que momento activar una de las subrutinas dependientes del tiempo o permite que se active otra de las subrutinas independientes.

Se puede observar, que este control primitivo puede causar grandes problemas de programación. Cada subrutina debe escribirse dentro de las especificaciones convenidas (por ejemplo la Tabla VIII.1), es decir, dentro de los tiempos impuestos por la estructura del software.

En realidad en aplicaciones simples en las que los microprocesadores no están saturados en trabajo, este esquema trabaja muy bien. Sin embargo, en caso contrario, las únicas alternativas posibles son las interrupciones.

Una interrupción es la manera de producir por hardware una llamada a subrutina. Cuando ocurre una condición externa (es decir, una interrupción), la terminal INT del CPU es aterrizada causando que se ejecute una subrutina, la cual está en una dirección específica y fija.

Si al suceder una interrupción, el programa que está en ejecución no puede continuar, se dice que éste ha sido "interrumpido", mientras que la subrutina a ser ejecutada se llama "rutina de servicio de interrupción".

Las interrupciones son usadas frecuentemente para permitir que muchos eventos asíncronos sean coordinados por la misma computadora.

Cuando un evento externo requiere ser comunicado con el procesador éste es sensado y sostenido en un "latch de interrupción"

(ver Fig. VIII.5.1.1), esto se debe a que el CPU puede encontrarse ocupado y no puede responder inmediatamente; una vez que el CPU reconoce la interrupción, el "latch" es inicializado.

Cuando se diseñe el programa de interrupción, es importante que la última instrucción de éste regrese el control al programa interrumpido. Para esto, el microprocesador debe proveer facilidades primero, de salvar los registros y las banderas de estado y segundo, de restaurarlos evitando la destrucción del programa interrumpido.

En algunos microprocesadores esto se realiza automáticamente, mientras que en otros la rutina de servicio debe mantener la continuidad del programa interrumpido (ver Fig. VIII.5.1.2), haciendo uso de la pila.

Cuando las interrupciones son usadas para transferir datos ya sea de entrada o de salida, deben manejarse las siguientes tareas por el programa de interrupción:

- Salvar el contenido de todos los registros que se usarán durante la interrupción.

- Si existe más de una interrupción, identificar cuál necesita el servicio e invocar el software apropiado.

- Reconocer la interrupción de tal manera que el dispositivo de interrupción libere la terminal de interrupción del CPU.

- Efectuar las transferencias de datos.

- Verificar si ésta es la última serie de interrupciones. Si este es el caso, informar al programa principal por ejemplo, que todo el registro ha sido leído.

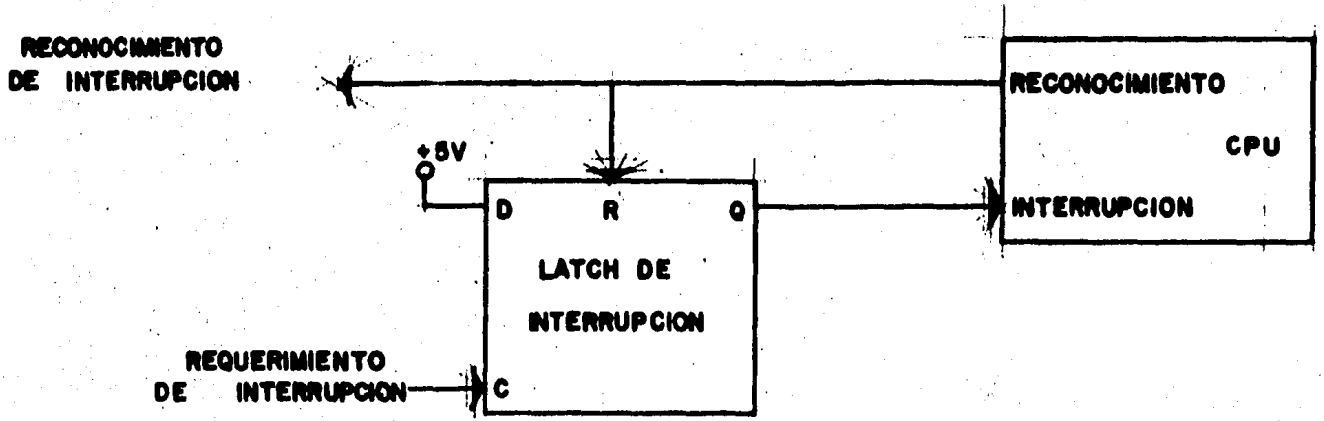


Fig: VIII.5.1.1 Interfase para sensar interrupciones.

PROGRAMA DE INTERRUPCION

```
      PUSH    PSW   ; SALVA (A) Y BANDERAS
      PUSH    B     ; SALVA (B,C)
      PUSH    D     ; SALVA (D,E)
      PUSH    H     ; SALVA (H,L)
       .
       .
       .
       .
(PROCESAMIENTO DE INTERRUPCION)
       .
       .
       .
       .
      POP     H     ; RESTAURA TODOS
      POP     D     ;(NOTE EL ORDEN)
      POP     B
      POP     PSW
      EI            ; REHABILITA INTERRUPCIONES
      RET           ; SALIDA
```

FIG. VIII.5.1.2 UNA PEQUEÑA RUTINA DE SERVICIO.

- Restaurar los registros y las banderas a su estado original (antes de la interrupción).

- Habilitar interrupciones. Rehabilitar las interrupciones que fueron detectadas y detenidas para prevenir interferencias y luego regresar al programa interrumpido.

Ocasionalmente se necesitará más de una interrupción en el sistema, por ejemplo, si se requiere manejar un período de reloj de 60 Hz que produzca una interrupción cada 16.67 ms, y por otro lado un canal de comunicación que genere interrupciones esporádicamente, para causar alguna transferencia de datos.

La mayoría de los microprocesadores tienen capacidad de manejar múltiples fuentes de interrupción, teniendo algo en común: de alguna manera deberán identificar cual interrupción ha ocurrido pudiendolo detectar de dos maneras: primero por vectores de interrupción y segundo, por sensado de interrupciones.

En los vectores de interrupción, varias subrutinas de servicio están perfectamente direccionadas en el diseño del CPU, de tal forma que al producirse una interrupción de alguna manera se establece cuál es la dirección específica de la rutina de servicio. En este tipo de diseño, por ejemplo en el 8080 o en el 9900, es necesario utilizar algunas terminales extras o señales de control para permitir establecer la dirección de la rutina de servicio requerida. Sin embargo, aquí también se presentan serios problemas, por ejemplo, si dos interrupciones llegan de diferentes puntos, se requiere algún circuito o alguna lógica que permita serializar estas interrupciones. Por ejemplo, se pueden utilizar CI como es el caso de la Fig. VIII.5.1.3, el cual al recibir dos o más interrupciones al mismo tiempo, únicamente una de ellas (dependiendo de como se alambre el circuito) alcanzará la salida y podrá enviar la interrupción.

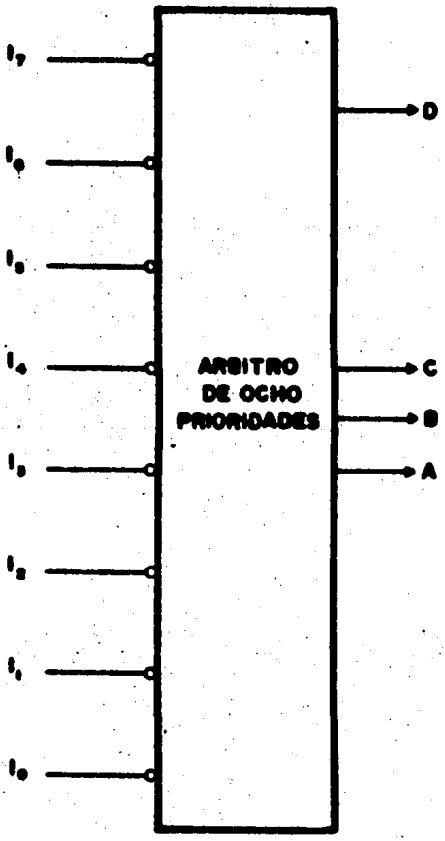


TABLA DE VERDAD

ENTRADAS								SALIDAS			
I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	A	B	C	D
H	H	H	H	H	H	H	H	X	X	X	L
L	H	H	H	H	H	H	H	L	L	L	H
X	L	H	H	H	H	H	H	L	L	H	H
X	X	L	H	H	H	H	H	L	H	L	H
X	X	X	L	H	H	H	H	L	H	H	H
X	X	X	X	L	H	H	H	H	L	L	H
X	X	X	X	X	L	H	H	H	L	H	H
X	X	X	X	X	X	L	H	H	H	L	H
X	X	X	X	X	X	X	L	H	H	H	H

FIG. VIII . 5.1.3 ARBITRO DE PRIORIDAD DE INTERRUPCIONES

Por otro lado para el caso de sensado de interrupciones, cuando ocurre una de ellas, es necesario interrogar a todos los interruptores en potencia para poder determinar el originador. Esto se puede hacer con un circuito como el de la Fig. VIII.5.1.4. Esta filosofía de sensado es muy común en microprocesadores del tipo 6800, en donde cada fuente de interrupción tiene su propio "latch de requerimiento de interrupción". De esta manera las interrupciones son distinguidas una de otra por medio de la línea "SENSAR", donde cada línea de "SENSAR" está conectada individualmente a un puerto de entrada de tal manera que el CPU puede leerlas resolviendo así la prioridad. El CPU reconoce las interrupciones individualmente a través de otro puerto de salida donde cada bit de éste está conectado a los diferentes "latches de requerimiento de interrupción"; los "latches" a su vez, son "limpiados" por un programa de control. Las líneas de "HABILITAR" son a su vez conectadas también a un puerto de salida (el cual tiene también un "latch") y de esta manera las interrupciones pueden ser ignoradas individualmente por hardware pero bajo un programa de control. Aunque es claro que éste sistema requiere mayor programación, provee mayor flexibilidad en algunos casos

La comunicación entre un programa de interrupción y los programas principales que dependen de él, puede realizarse con un "semáforo". Cuando el programa de interrupción ha completado sus tareas después de varias interrupciones, éste lo indica colocando un código en algún lugar de memoria específico. El programa principal verifica ocasionalmente esta localidad para ver si puede arrancar otra tarea. La teoría de los "semáforos" puede verse en este mismo capítulo, en la sección VIII.3.2.3.

VIII.5.2 Regulación de tiempos

En sistemas de tiempo real es implícita la necesidad de contabilizar el tiempo para la regulación de tareas con la presencia

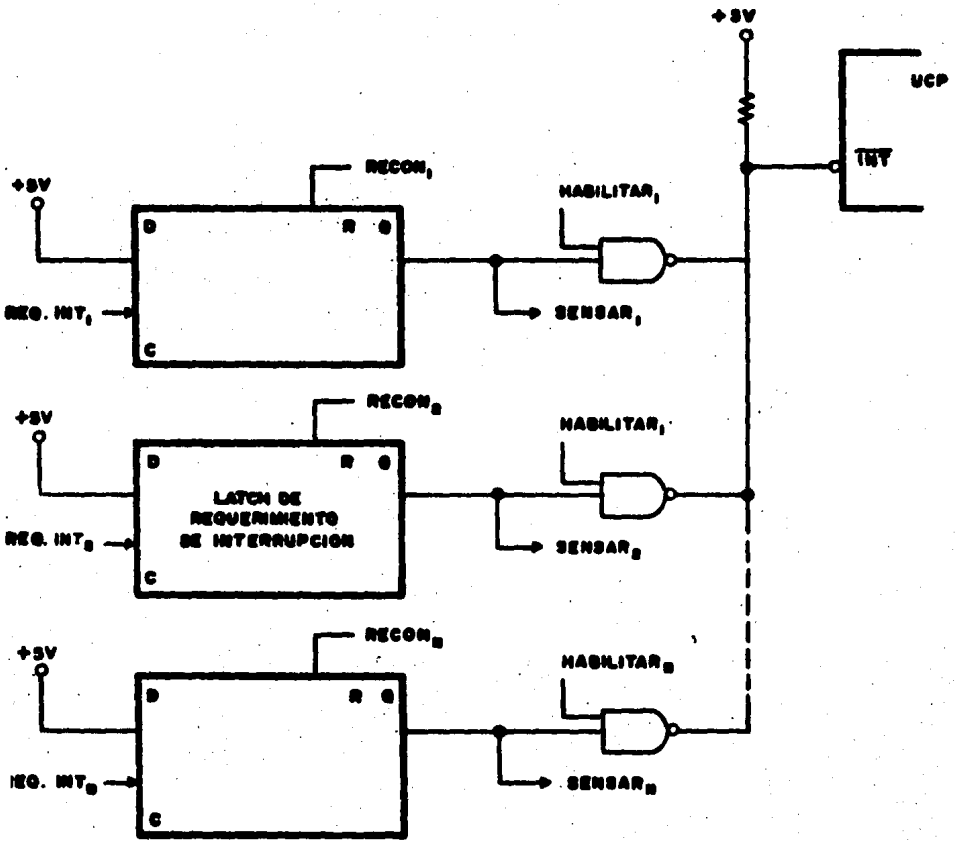


FIG. XXIX 5.1.4 SENSADO DE MULTIPLES INTERRUPCIONES

de eventos importantes, es decir, en los microprocesadores para tiempo real será escencial controlar la regulación de tiempos.

Por ejemplo, algunos eventos de salida requieren la generación de un simple pulso, esta operación es muy sencilla si se utiliza un bit de un puerto o memoria el cual se enciende y posteriormente es apagado; en algunos microprocesadores esto se realiza mediante una sola instrucción.

Algunas tareas, sin embargo, requieren de pulsos de cierta duración específica, en donde esta duración puede representar varios ciclos de instrucción, de tal forma que una secuencia de instrucciones que hagan nada, es decir, produzcan retardos, serán útiles para regular ésta operación. Un ejemplo de este tipo de instrucciones pueden ser NOP (no operación), saltos condicionales que nunca ocurren (es decir, un "salto si es cero" y el contenido del registro no es cero) o "salto a la siguiente localidad" y llamadas a subrutinas que solo contienen la instrucción de retorno (subrutinas vacías). Estas instrucciones son útiles sobre un rango de 10 a 100 ms.

Para intervalos mayores se requerirá el uso de iteraciones, si la regulación de tiempos está controlado por software.

Es claro que conociendo la duración de cada instrucción se pueden conformar retardos que bien pueden servir para mantener la salida de pulsos el tiempo que se desee (ver Fig. VIII.5.2.1), bien en espera de alguna acción o simple regulación de tiempo (ver Fig. VIII.5.2.2).

Para evaluar el tiempo de ejecución de una subrutina de retardo (de la Fig. VIII.5.2.2), es necesario calcular los tiempos de:

- El tiempo del cuerpo de conteo multiplicado por el valor inicial del contador.

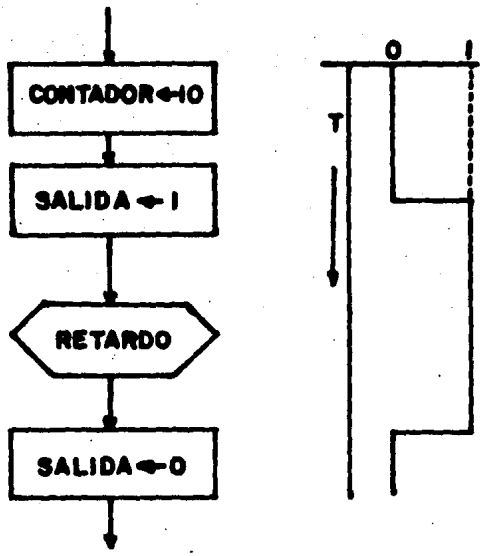


Fig: VIII.5.2.1 Generación de ancho de pulsos conocidos por Software

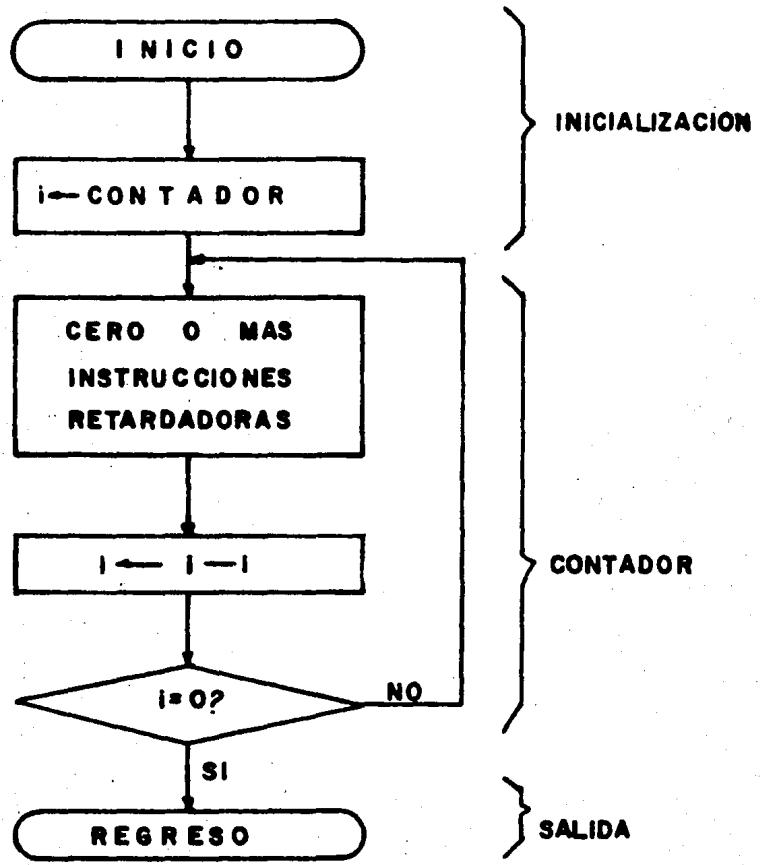


Fig: VIII.5.2.2. Regulación de tiempo por Software.

- Llamada e inicialización de variables.

- El tiempo de retorno y reestablecimiento de variables o registros.

El contador puede ser implementado fácilmente en la rutina de retardo por uno o más registros del microprocesador.

Para generar una salida por ejemplo de 1 ms, la rutina de retardo es llamada repetidamente (Fig. VIII.5.2.1). Desgraciadamente siempre existirán algunos errores de tiempo, es decir, en este caso si se desea ajustar a que la subrutina consuma precisamente 100 ms se deberá considerar también el tiempo debido a las llamadas y retornos de subrutina.

Para obtener una mayor precisión de tiempo será necesario acortar el tiempo de la subrutina e incrementar el contador. En algunos casos la iteración de retardo requerirá de unos cuantos microsegundos; en este caso el contador inicial podrá ser ajustado para compensar el tiempo consumido en la llamada y el retorno, de tal forma que el tiempo total de retardo sea preciso.

Este tipo de esquemas pueden sustituir fácilmente a los osciladores comunmente usados en funciones digitales, por ejemplo, cuando se envían datos a un teletipo se requiere poner en serie la información y enviarla en intervalos de 9 ms. En este caso el oscilador podría ser emulado por un programa.

Por otro lado si se desea registrar el tiempo en un instrumento basado en microcomputadora será necesario (por ejemplo) interrumpirlo periódicamente con una señal cuya frecuencia sea lo suficientemente exacta para el propósito del instrumento.

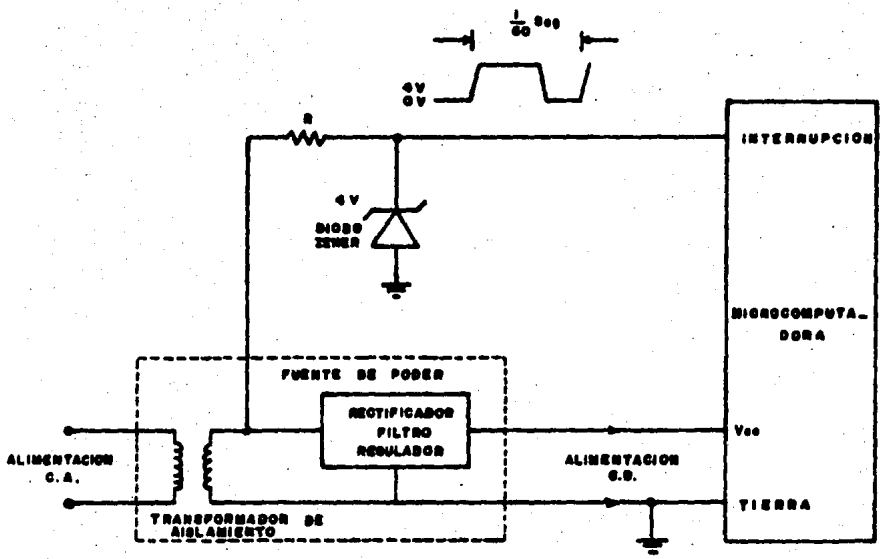


Fig: VIII.5.2.3. Derivación de tiempo de la línea de C.A.

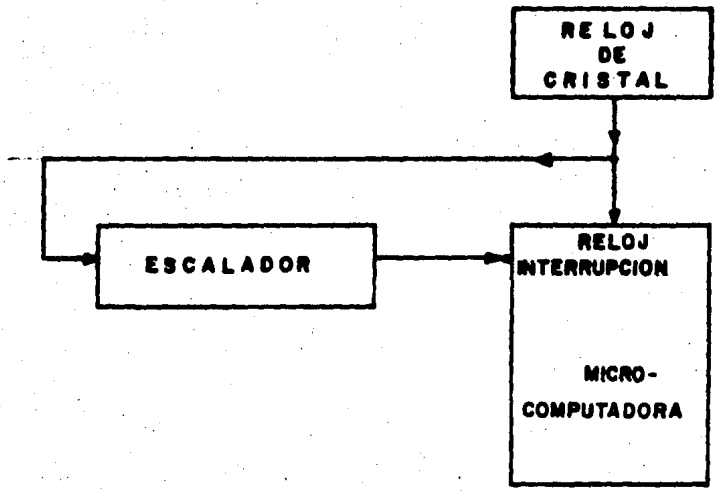


Fig:VS 5.2.4 DERIVACION DE TIEMPO DEL RELOJ MAESTRO

Sea el caso de una pantalla digital (display), en la que se requiera desplegar las horas, minutos y segundos, para esto una señal de interrupción podrá ser derivada de la línea de CA (ver Fig. VIII.5.2.3). La rutina de servicio para esta interrupción contabiliza en las localidades RAM las horas, minutos, segundos y sesentavos de segundo.

Para generar intervalos de tiempo (por ejemplo, pulsos de 50 ms de duración en una línea de un puerto de salida), se puede seguir tambien la misma filosofía de dividir el tiempo en intervalos de sesentavos de segundo. Si se desea incrementar este número de intervalos, se puede utilizar un cristal (probablemente el del computador) el cual puede ser escalado (ver Fig. VIII.5.2.4) para obtener las interrupciones a la frecuencia necesitada.

Un regulador de tiempos programable (discutido en la sección VII.5.1) es otro ejemplo al probleme anterior. Este puede ser usado produciendo interrupciones a intervalos de tiempo fijos, los cuales son contabilizados para determinar cuando actualizar los eventos.

Una tercer propuesta para proporcionar control de tiempo real de eventos consiste en forzar el software de tal manera que a pesar de los saltos e iteraciones, el intervalo de tiempo entre dos puntos específicos en un programa, permanezca constante.

Sea el caso de la Fig. VIII.5.2.5, en donde al presentarse una interrupción, el período T tendrá variaciones, sin embargo, si forzamos a éste a ser constante permitiendo interrupciones, se le debe compensar en la misma proporción que el tiempo tomado por cada interrupción. Esto se puede realizar incrementando un contador cada vez durante cada interrupción y entonces, usar una iteración de espera en el programa principal compensando de esta manera el tiempo utilizado por la rutina de servicio (ver Fig. VIII.5.2.6).

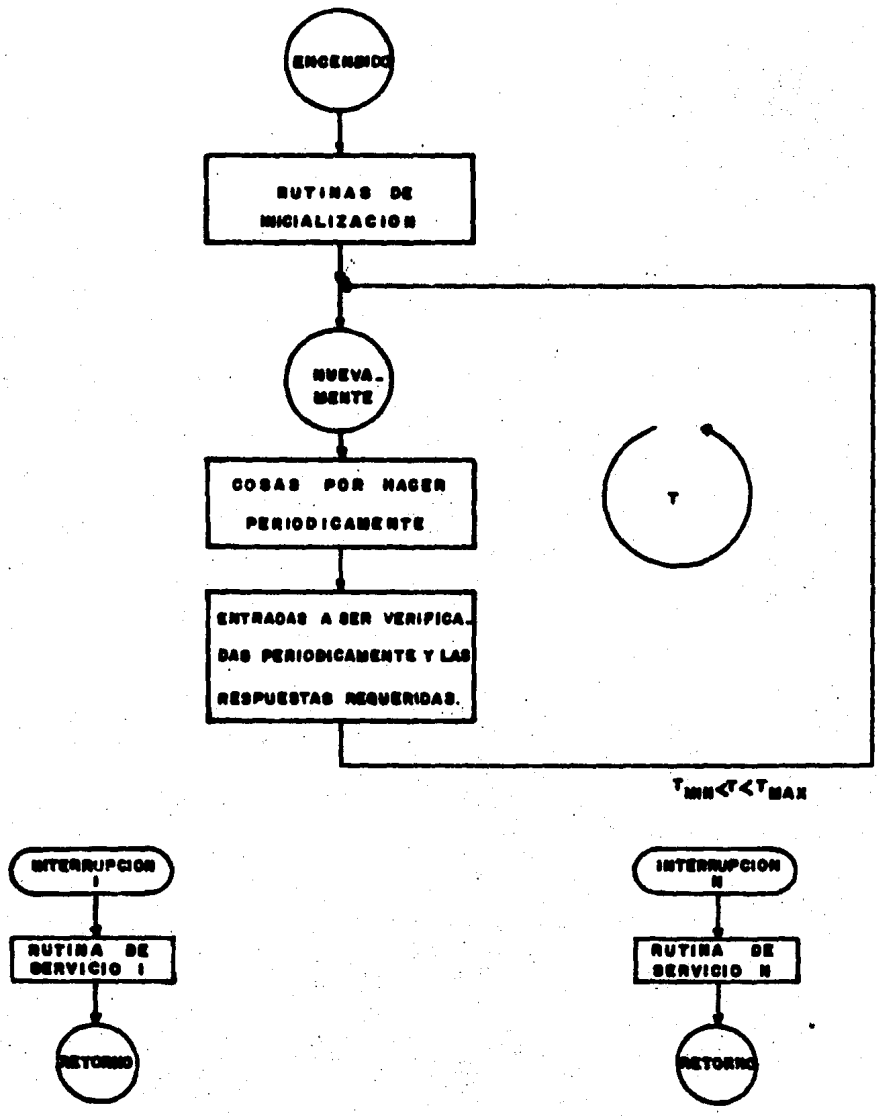


Fig: VIII.5.2.5. Diagrama de flujo de un esquema por interrupciones.

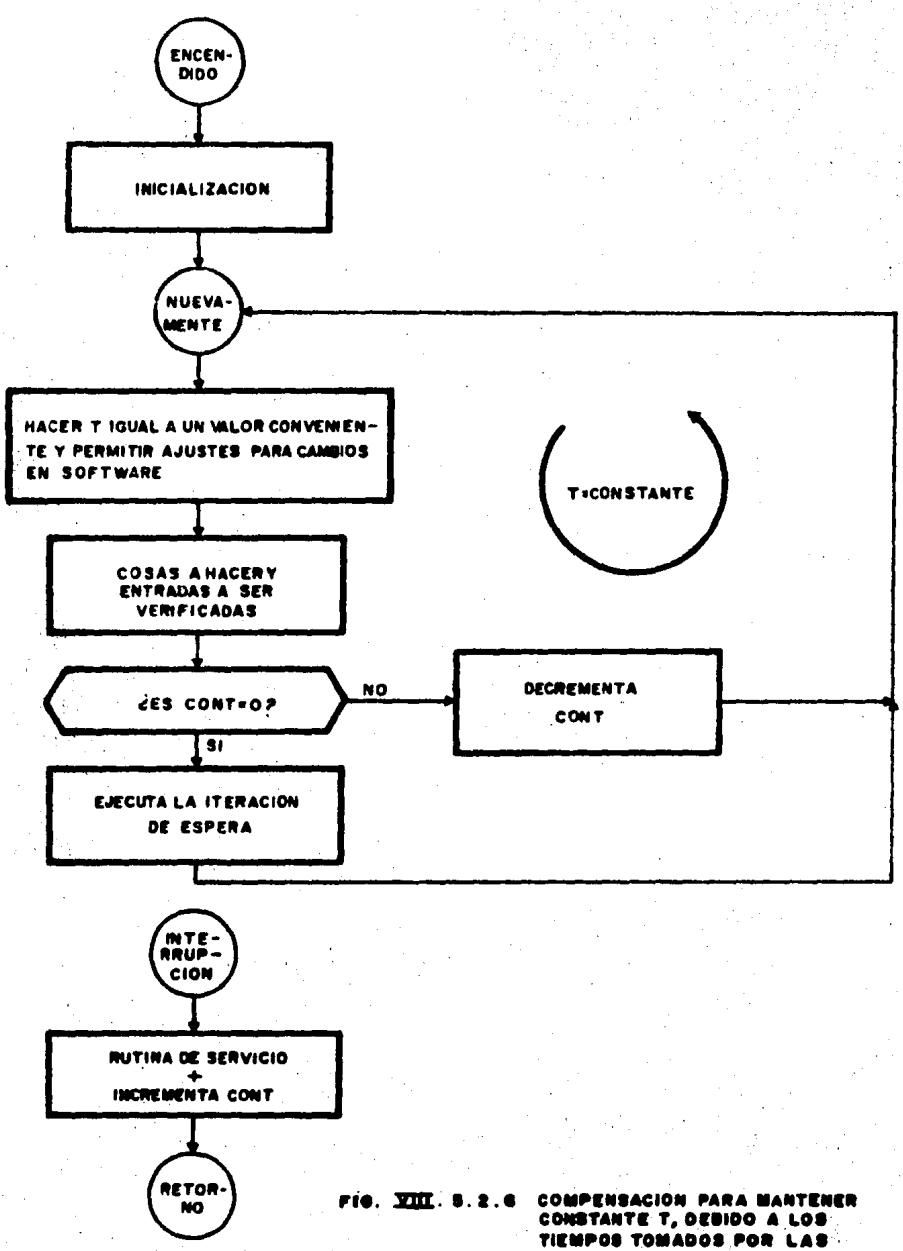


FIG. VIII. 5.2.6 COMPENSACION PARA MANTENER CONSTANTE T, DEBIDO A LOS TIEMPOS TOMADOS POR LAS INTERRUPCIONES

El DMA, por otro lado, puede ser manejado similarmente, si acaso es satisfactorio mantener el valor promedio T precisamente igual a T. El número de ciclos de reloj requeridos para una transferencia completa de DMA es una cantidad conocida, por lo que se puede compensar sin ningún problema.

Usando esta propuesta de compensación, cada salto requeriré usualmente una iteración de espera para mantener el tiempo constante sin importar cual salto es ejecutado (ver Fig. VIII.5.2.7). Obviamente no se pueden considerar en este esquema saltos cuyos tiempos de ejecución dependan de los datos a ser manipulados.

En la Fig. VIII.5.2.8 se puede observar otro ejemplo con varios saltos y compensaciones, aquí es más notorio lo tedioso del cálculo de la duración de cada salto en un programa. Sin embargo, esta técnica es muy valiosa para el manejo de eventos cuyos intervalos de ejecución son muy cortos (por ejemplo menos de 100 microsegundos) y dentro de éstos, arrancar otras tareas.

Para intervalos de tiempo pequeños, el tiempo requerido por una interrupción para salvar y restaurar los registros del CPU, resulta significativo. Si se desea, en alguna aplicación ejecutar cuatro tareas repetitivamente, seguidas de un evento el cual es requerido cada 100 microsegundos, entonces, un esquema como el de la Fig. VIII.5.2.9 tiene sentido. Si alguna tarea, involucra saltos, estos pueden ser fácilmente compensados.

VIII.6 LOS LENGUAJES DE ALTO NIVEL .VS. LOS LENGUAJES ENSAMBLADORES

Los sistemas de software de tiempo real tradicionalmente han sido escritos en códigos a nivel ensamblador. El argumento principal a esto ha sido la naturaleza crítica en tiempo de la mayoría de las aplicaciones en tiempo real. Los programas escritos por programadores

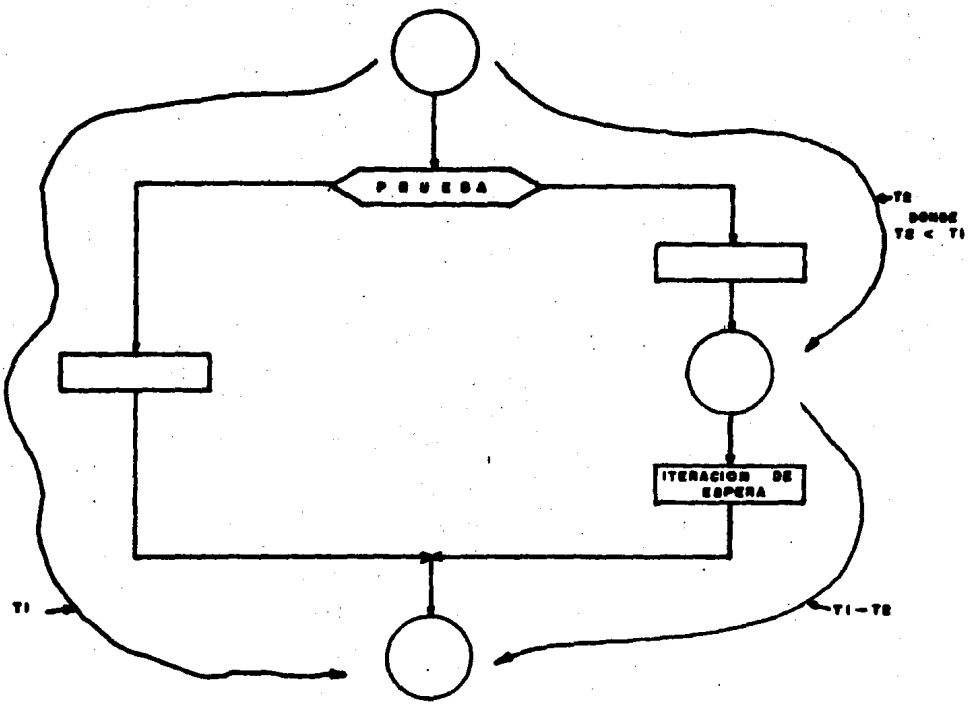


Fig: VIII.5.2.7 Compensación para saltos alternos.

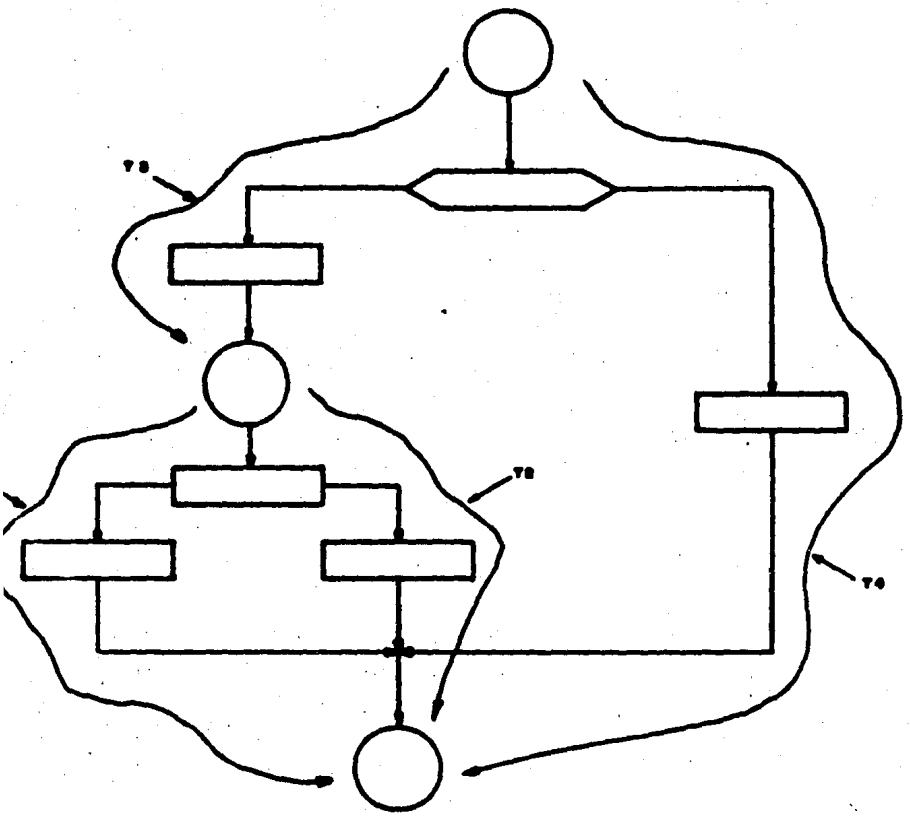


Fig: VIII.3.2.3. Compensación para saltos múltiples.

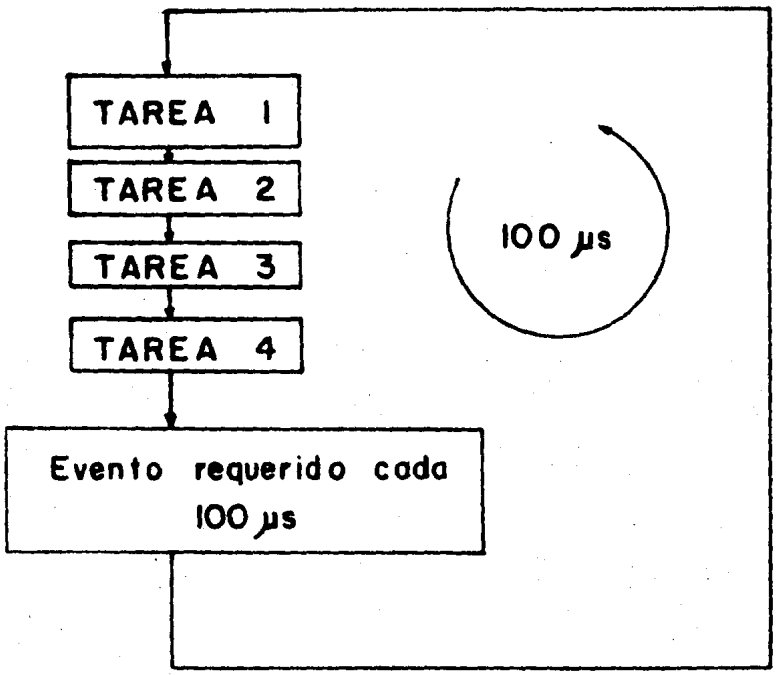


FIG. VIII. 5.2.9 ESQUEMA PARA INTERVALOS DE TIEMPO PEQUEÑOS, EN DONDE UNA RUTINA ES SOLICITADA CADA 100 µs.

experimentados en lenguaje ensamblador son generalmente más eficientes tanto en velocidad como en requerimientos de memoria, que aquellos programas producidos por un compilador.

VIII.6.1 Eficiencia

La crítica más fuerte en contra de los lenguajes de alto nivel, es que el código que producen los compiladores es relativamente ineficiente cuando se consideran pequeñas cantidades de código de programa (p.e. 100 a 200 líneas). Esto se debe a que un compilador no puede ser tan astuto como un programador y no encuentra optimizaciones locales en programas particulares.

Conforme el programa va aumentando en tamaño, se va haciendo más y más difícil para el programador mantener una comprensión a fondo de toda la estructura del programa.

Conforme la complejidad del programa aumenta, la habilidad a optimizar del programador disminuye; el compilador, por otro lado, es indiferente al tamaño del programa y su optimización opera activamente con igual eficiencia.

Para un cierto tamaño de programa, un programador puede producir el doble de código eficiente que un compilador. No obstante, para programas de tamaño moderado, un compilador moderno no puede esperar producir un código tan eficiente como el escrito por un programador en código ensamblador.

VIII.6.2 Costos

Cuando son comparados sobre la duración de un producto, los lenguajes de alto nivel tienen considerables ventajas sobre la implementación en lenguajes ensambladores.

Las aplicaciones pueden ser codificadas más rápidamente en un lenguaje de alto nivel, este código producido es fácil de leer y en consecuencia está mejor documentado, también es fácil de mantener, corregir y extender.

Más aún, produciendo un código bien estructurado donde las decisiones importantes y los puntos de control sean resaltados, es posible implementar un esquema razonable de examinación ó autoprueba.

Los compiladores de alto nivel cuentan con suficientes verificaciones para evitar muchas de las fallas de la programación en nivel ensamblador. Por ejemplo, es imposible transferir inadvertidamente el control fuera de los límites del programa, algo sumamente fácil de sufrir en nivel ensamblador.

Durante la fase de mantenimiento, un producto codificado en alto nivel es, sin lugar a dudas, más económico que uno codificado en lenguaje ensamblador.

C O N C L U S I O N E S

Todo estudio teórico se basa en suposiciones y se apoya en la física matemática para desarrollarse hasta conseguir sus objetivos que pueden ser: explicar un fenómeno, su representación, su predicción, etc., para posteriormente pasar al plano experimental y poder reconstruirlo y usarlo para algun fin.

De la misma manera, el diseñar involucra bases físico-matemáticas (conocimientos teóricos) profundas por una parte, y por otra las bases experimentales que hacen concluir el objetivo (modelar, analizar, construir, etc.).

Si hay alguien interesado en el diseño de sistemas en tiempo real (o en cualquier diseño) primero debe conseguir toda la información necesaria; segundo, clasificarla; tercero, analizarla lo cual comprende estudios teórico-experimentales y, finalmente llevarla a cabo; en esta última etapa la experiencia es fundamental. Es aquí donde este trabajo mediante la presentación de ideas, ejemplos prácticos, limitaciones, problemática, etc. ubica al investigador en el campo del diseño de sistemas automáticos, con el objetivo de facilitar el camino del diseño.

Los capítulos que se contemplaron en este trabajo cubrieron parcialmente cada una de las etapas involucradas en el diseño de sistemas automáticos. Sin embargo es necesario aclarar que no se mencionaron más técnicas de diseño, ni más dispositivos, porque sería imposible cubrir toda esta información. Esencialmente este trabajo eslabona los avances teóricos con los prácticos para ofrecer una guía al diseñador.

B I B L I O G R A F I A

ALLWORTH, S.; INTRODUCTION TO REAL-TIME SOFTWARE DESIGN; ED. SPRINGER VERLAG; HONG KONG,1981

AMANEYRO, E. y GARIBAY, R.; TECNICAS DE CONTROL EN PROCESOS INDUSTRIALES; TESIS PROFESIONAL; U.N.A.M., FAC. DE ING.,1983

ANGELO, E; BJTs, FETs AND MICROCIRCUITS, CAPITULOS 4 Y 5; ED. Mc GRAW HILL; TOKIO,1969

ARTWICK, B.; MICROCOMPUTER INTERFACING; ED. PRENTICE HALL; E.U.A.,1980

AUSLANDER, D. y SAGUES, P.; MICROPROCESSORS FOR MEASUREMENTS AND CONTROL; ED. OSBORNE/Mc GRAW HILL; E.U.A.,1981

CADZOW, J.; DISCRETE-TIME SYSTEMS; ED. PRENTICE HALL; E.U.A.,1973

CARR, J.; DIGITAL INTERFACING WITH AN ANALOG WORLD: ED. TAB BOOKS; E.U.A.,1978

DAVIS, S.; RETROALIMENTACION Y SISTEMAS DE CONTROL; ED. FONDO EDUCATIVO INTERAMERICANO; MEXICO,1977

FRANKLIN, G. y POWEL, J.; DIGITAL CONTROL OF DYNAMIC SYSTEMS; ED. ADDISON WESLEY; E.U.A.,1980

GARLAND, H.; INTRODUCTION TO MICROPROCESSOR SYSTEM DESIGN; ED. Mc GRAW HILL; E.U.A.,1979

GARRET, P. H.; ANALOG I/O/ DESIGN, ACQUISITION: CONVERSION: RECOVERY; ED. RESTON PUBLISHING COMPANY; E.U.A.,1981

GEREZ, V. y MURRAY-LASSO, M.; TEORIA DE SISTEMAS Y CIRCUITOS, CAPITULOS 4,5,6, y 9; ED. REPRESENTACIONES Y SERVICIOS DE INGENIERIA, S. A.; MEXICO,1972

ISERMANN, R.; DIGITAL CONTROL SYSTEMS; ED. SPRINGER VERLAG: E.U.A., 1981

JUNG, W.; IC CONVERTER COOKBOOK; ED. HOWARD W. SAMS; E.U.A.,1978

KERNIGHAN, B. y RITCHIE, D.; THE "C" PROGRAMMING LANGUAGE; ED. PRENTICE HALL; E.U.A.,1978

KRUTZ, R.; MICROPROCESSORS AND LOGIC DESIGN; ED. JOHN WILEY AND SONS; E.U.A.,1980

KUO, B. C.; DIGITAL CONTROL SYSTEMS; ED. HOLT, RINEHART AND WINSTON; E.U.A.,1980

MADNICK, S. y DONOVAN, J.; OPERATING SYSTEMS, CAPITULO 4; ED. Mc GRAW HILL; TOKIO,1974

MALVINO, A.; PRINCIPIOS DE ELECTRONICA, CAPITULOS 2 y 3; ED. Mc GRAW HILL; MEXICO,1982

MOMPIN POBLET, JOSE; TRANSDUCTORES Y MEDIDORES ELECTRONICOS; ED. MARCOMBO BOIXAREU; ESPAÑA,1979

OGATA, K.; INGENIERIA DE CONTROL MODERNA; ED. PRENTICE HALL; ESPAÑA, 1976

OGDIN, C.; MICROCOMPUTER DESIGN, CAPITULOS 8 y 9; ED. PRENTICE HALL; E.U.A.,1978

OSBORNE, A. y KANE, G.; AN INTRODUCTION TO MICROCOMPUTERS: VOL 2 y 3; ED. OSBORNE/Mc GRAW HILL; E.U.A.,1978

PEATMAN, J.; DIGITAL HARDWARE DESIGN, CAPITULO 6; ED. Mc GRAW HILL; TOKIO,1980

PEATMAN, J.; MICROCOMPUTER BASED DESIGN, CAPITULOS 6 y 7; ED. Mc GRAW HILL; TOKIO,1977

RAMIREZ, E. y WEISS, M.; MICROPROCESSING FUNDAMENTALS, CAPITULOS 3,4,6 y 10; ED. Mc GRAW HILL; TOKIO,1980

SCOTT, P. R. D.; MODEMS IN DATA COMMUNICATIONS; ED. NCC PUBLICATIONS; INGLATERRA,1980

SHEINGOLD, D.; TRANSDUCER INTERFACING HANDBOOK; ED. ANALOG DEVICES; E.U.A.,1981

TITUS, J., TITUS, C., RONY, P. y LARSEN, D.;MICROCOMPUTER ANALOG CONVERTER SOFTWARE AND HARDWARE INTERFACING: ED. HOWARD W. SAMS; E.U.A.,1978

ZAKS, R. y LESEA, A.; MICROPROCESSOR INTERFACING TECHNIQUES; ED. SYBEX; E.U.A.,1979

A P E N D I C E A

CONJUNTOS DE INSTRUCCIONES

Table 4-4. A Summary of 8080A/8080A Microcomputer Instruction Set

TYPE	MNEMONIC	OPERANDS	BYTES	STATUSES						OPERATION PERFORMED
				C	AC	Z	S	P	BUB*	
I/O	IN	DEV	2							(A)←(DEV) Input to A from device DEV (DEV = 0 to 255)
	OUT	DEV	2							(DEV)←(A) Output from A to device DEV (DEV = 0 to 255)
PRIMARY MEMORY REFERENCE	LDAX	RP	1							(A)←((RP)) Load A using address implied by BC (RP = 00 or BC (RP = 01
	STAX	RP	1							((RP))←(A) Store A using implied addressing as for LDAX
	MOV	RDES	1							(RDS)←((H,L)) Load any register using address implied by HL
	MOV	MRDS	1							((H,L))←(RDS) Store any register using address implied by HL
	LDA	ADDR	3							(A)←(ADDR) i.e., (A)←((B, C)) Load A, use direct addressing
	STA	ADDR	3							(ADDR)←(A) i.e., ((B, C))←(A) Store A, use direct addressing
	LHLD	ADDR	3							(L)←(ADDR), (H)←(ADDR + 1) i.e., (L)←((B, C)), (H)←((B, C) + 1) Load H and L registers, use direct addressing
	SHLD	ADDR	3							(ADDR)←(L), (ADDR + 1)←(H) i.e., ((B, C))←(L), ((B, C) + 1)←(H) Store H and L registers, use direct addressing
SECONDARY MEMORY REFERENCE ARITHMETIC OPERATIONS	ADD	M	1	X	X	X	X	X	0	(A)←(A) + ((H,L)) Add to A
	ADC	M	1	X	X	X	X	X	0	(A)←(A) + ((H,L)) + (C) Add with Carry to A
	SUB	M	1	X	X	X	X	X	1	(A)←(A) - ((H,L)) Subtract from A
	SBB	M	1	X	X	X	X	X	1	(A)←(A) - ((H,L)) - (C) Subtract from A with borrow
	ANA	M	1	0**	X1**	X	X	X		(A)←(A) ∧ ((H,L)) AND with A
	XRA	M	1	0**	0**	X	X	X		(A)←(A) ∨ ((H,L)) Exclusive-OR with A
	ORA	M	1	0**	0**	X	X	X		(A)←(A) ∨ ((H,L)) OR with A
	CMP	M	1	X	X	X	X	X	1	(A) - ((H,L)) Discard result but set flags. Compare with A
	INR	M	1		X**	X	X	X	0	((H,L))←((H,L)) + 1 Increment memory
	DCR	M	1		X**	X	X	X	1	((H,L))←((H,L))-1 Decrement memory

Table 4-4. A Summary of 8080A/9080A Microcomputer Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	AC	Z	S	P	SUB*	
IMMEDIATE	LXI	RP,DATA16	3							(RP)←DATA16 Load 16-bit immediate data into BC (RP = B), DE (RP = D), HL (RP = H) or SP (RP = SP)
	MVI	M,DATA	2							((H,L))←DATA Load 8-bit immediate data into memory location with address implied by HL
	MVI	REG,DATA	2							(REG)←DATA Load 8-bit immediate data into any register
JUMP	JMP	ADDR	3							(PC)←ADDR Jump to instruction with label ADDR
	PCHL		1							(PC)←(H,L) Jump to instruction at location implied by HL
SUBROUTINE CALL AND RETURN IMMEDIATE AND STACK	CALL	ADDR	3							((SP))←(PC), (PC)←ADDR, (SP)←(SP)-2 Jump to subroutine starting at ADDR
	CC	ADDR	3							((SP))←(PC), (PC)←ADDR, (SP)←(SP)-2 Jump to subroutine if C = 1
	CNC	ADDR	3							((SP))←(PC), (PC)←ADDR, (SP)←(SP)-2 Jump to subroutine if C = 0
	CZ	ADDR	3							((SP))←(PC), (PC)←ADDR, (SP)←(SP)-2 Jump to subroutine if Z = 1
	CNZ	ADDR	3							((SP))←(PC), (PC)←ADDR, (SP)←(SP)-2 Jump to subroutine if Z = 0
	CP	ADDR	3							((SP))←(PC), (PC)←ADDR, (SP)←(SP)-2 Jump to subroutine if S = 0
	CM	ADDR	3							((SP))←(PC), (PC)←ADDR, (SP)←(SP)-2 Jump to subroutine if S = 1
	CPE	ADDR	3							((SP))←(PC), (PC)←ADDR, (SP)←(SP)-2 Jump to subroutine if even parity
	CPO	ADDR	3							((SP))←(PC), (PC)←ADDR, (SP)←(SP)-2 Jump to subroutine if odd parity
	RET		1							(PC)←((SP))[(SP)←(SP)+2 Return from subroutine
	RC		1							(PC)←((SP)), (SP)←(SP)+2 Return from subroutine if C = 1
	RNC		1							(PC)←((SP)), (SP)←(SP)+2 Return from subroutine if C = 0
	RZ		1							(PC)←((SP)), (SP)←(SP)+2 Return from subroutine if Z = 1
	RNZ		1							(PC)←((SP)), (SP)←(SP)+2 Return from subroutine if Z = 0
	RM		1							(PC)←((SP)), (SP)←(SP)+2 Return from subroutine if S = 1

Table 4-4. A Summary of 8080A/9080A Microcomputer Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	AC	Z	S	P	SUB*	
SUBROUTINE CALL AND RETURN INSTRUCTIONS AND STACK CONTROL	RP		1							(PC)←((SP)), (SP)←(SP)+2 Return from subroutine if S = 0
	RPE		1							(PC)←((SP)), (SP)←(SP)+2 Return from subroutine if even parity
	RPO		1							(PC)←((SP)), (SP)←(SP)+2 Return from subroutine if odd parity
IMMEDIATE OPERATE	ADI	DATA	2	X	X	X	X	X	0	(A)←(A)+DATA Add immediate to A
	ACI	DATA	2	X	X	X	X	X	0	(A)←(A)+DATA+(C) Add with carry immediate to A
	SUI	DATA	2	X	X	X	X	X	1	(A)←(A)-DATA Subtract immediate from A
	SBI	DATA	2	X	X	X	X	X	1	(A)←(A)-DATA-(C) Subtract immediate with borrow from A
	ANI	DATA	2	0**	X†	X	X	X		(A)←(A) A DATA AND immediate with A
	XRI	DATA	2	0**	0**	X	X	X		(A)←(A)⊻DATA Exclusive-OR immediate with A
	ORI	DATA	2	0**	0**	X	X	X		(A)←(A)∨DATA OR immediate with A
	CPI	DATA	2	X	X	X	X	X		Compare immediate with A
JUMP ON CONDITION	JC	ADDR	3							(PC)← ADDR Jump if C = 1
	JNC	ADDR	3							(PC)← ADDR Jump if C = 0
	JZ	ADDR	3							(PC)← ADDR Jump if Z = 1
	JNZ	ADDR	3							(PC)← ADDR Jump if Z = 0
	JP	ADDR	3							(PC)← ADDR Jump if S = 0
	JM	ADDR	3							(PC)← ADDR Jump if S = 1
	JPE	ADDR	3							(PC)← ADDR Jump on even parity
	JPO	ADDR	3							(PC)← ADDR Jump on odd parity

Table 4-4. A Summary of 8080A/8080A Microcomputer Instruction Set (Continued)

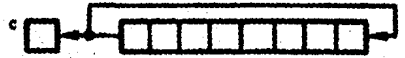
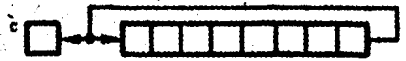
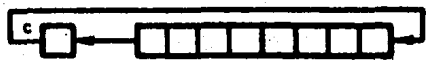
TYPE	MNEMONIC	OPERANDS	BYTES	STATUSES						OPERATION PERFORMED
				Z	AC	S	P	CY		
REGISTER-TO-REGISTER, MEMORY AND IMMEDIATE OPERATIONS	MOV	D,S	1							(REG)←(REG) Move any register (s) to any register (d)
	XCHG		1							(D)←→(H), (E)←→(L) Exchange DE with HL
	SPHL		1							(SP)←(HL) Transfer HL to SP
	ADD	REG	1	X	X	X	X	X	0	(A)←(A)+(REG) Add any register to A
	ADC	REG	1	X	X	X	X	X	0	(A)←(A)+(REG)+(C) Add with Carry any register to A
	SUB	REG	1	X	X	X	X	X	1	(A)←(A)-(REG) Subtract any register from A
	SBB	REG	1	X	X	X	X	X	1	(A)←(A)-(REG)-(C) Subtract any register with borrow from A
	ANA	REG	1	0**	X)	X	X	X		(A)←(A) ∧ (REG) AND any register with A
	XRA	REG	1	0**	0**	X	X	X		(A)←(A)∨(REG) Exclusive-OR any register with A
	ORA	REG	1	0**	0**	X	X	X		(A)←(A) ∨ (REG) OR any register with A
	CMP	REG	1	X	X	X	X	X	1	(A) - (REG) Discard result but set flags Compare any register with A
	DAD	RP	1	X					0	(H,L)←(H,L)+(RP) Add to HL
REGISTER OPERATE	INR	REG	1		X**	X	X	X	0	(REG)←(REG)+1 Increment any register
	DCR	REG	1		X**	X	X	X	1	(REG)←(REG)-1 Decrement any register
	CMA		1							(A)←($\overline{A}$) Complement A
	DAA		1	X	X**	X	X	X		Decimal adjust A
	RLC		1	X						 Rotate A left with branch carry
	RRC		1	X						 Rotate A right with branch carry

Table 4-4. A Summary of 8080A/8080A Microcomputer Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				C	AC	Z	S	P	BUB*	
REGISTER OPERATE (CONTINUED)	RAL		1	X						 Rotate A left with carry
	RAR		1	X						 Rotate A right with carry
	INX	RP	1							(RP)←(RP) + 1 Increment RP. RP = BC, DE, HL or SP
	DCX	RP	1							(RP)←(RP) - 1 Decrement RP
STACK	PUSH	RP	1							((SP))←(RP), (SP)←(SP) - 2 Push RP contents onto stack
	POP	RP	1							(RP)←((SP)), (SP)←(SP) + 2 Pop stack into RP. RP = BC, DE, HL or PSW
	XTHL		1							(H,L)←→((SP)) Exchange HL with top of stack
INTERRUPT	EI		1							Enable interrupts
	DI		1							Disable interrupts
	RST	N	1							Restart at address 8•N. N = 0 through 7.
STATUS	STC		1	1						(C)←1 Set Carry
	CMC		1	X						(C)←$\overline{(C)}$ Complement Carry
	NOP		1							No operation
	HLT		1							Halt

Statuses: C = Carry; A_C = Carry out of bit 3; Z = Zero; S = Sign; P = Parity; X = Status set or reset; 0 = Status reset; 1 = Status Set; Blank = Status unchanged

* - BUB status is present in NSC 8080A only

** NSC 8080A does not modify these status flags

† The AMD 8080A always resets A_C to 0 for all Boolean instructions. The Intel 8080 sets A_C to 1 for all AND instructions, and resets A_C to 0 for all other Boolean instructions.

Table 9-1. A Summary of the MC6800 Instruction Set

TYPE	MNEMONIC	OPERANDS	BYTES	STATUS						OPERATIONS PERFORMED
				C	Z	S	O	A_C	I	
PRIMARY MEMORY REFERENCE AND IMM	LDA	ACX,ADR8	2		X	X	0			(ACX)—(MEM) Load A or B using base page direct, extended direct, or indexed addressing.
		ACX,ADR16	3							
	STA	ACX,ADR8	2		X	X	0			(MEM)—(ACX) Store A or B using direct, extended, or indexed addressing.
		ACX,ADR16	3							
	LDX	ADR8	2		X	X	0			(XH:XL)—(MEM), (XL:01)—(MEM + 1) Load Index register using direct, extended, or indexed addressing. Sign status reflects Index register bit 15.
		ADR16	3							
	STX	ADR8	2		X	X	0			(MEM)—(XH:XL), (MEM + 1)—(XL:01) Store contents of Index register using direct, extended, or indexed addressing. Sign status reflects Index register bit 15.
		ADR16	3							
	LDS	ADR8	2		X	X	0			(SP:XH)—(MEM), (SP:XL)—(MEM + 1) Load Stack Pointer using direct, extended, or indexed addressing. Sign status reflects Stack Pointer bit 15.
		ADR16	3							
	STS	ADR8	2		X	X	0			(MEM)—(SP:XH), (MEM + 1)—(SP:XL) Store contents of Stack Pointer using direct, extended, or indexed addressing. Sign status reflects Stack Pointer bit 15.
		ADR16	3							
SECONDARY MEMORY REFERENCE (MEMORY OPERATE)	ADD	ACX,ADR8	2	X	X	X	X	X		(ACX)—(ACX) + (MEM) Add to Accumulator A or B using base page direct, extended direct, or indexed addressing.
		ACX,ADR16	3							
	ADC	ACX,ADR8	2	X	X	X	X	X		(ACX)—(ACX) + (MEM) + C Add with carry to Accumulator A or B using direct, extended, or indexed addressing.
		ACX,ADR16	3							
	AND	ACX,ADR8	2		X	X	0			(ACX)—(ACX) ∧ (MEM) AND with Accumulator A or B using direct, extended, or indexed addressing.
		ACX,ADR16	3							
	BIT	ACX,ADR8	2		X	X	0			(ACX) ∧ (MEM) AND with Accumulator A or B, but only Status register is affected.
		ACX,ADR16	3							
	CMP	ACX,ADR8	2	X	X	X	X			(ACX) - (MEM) Compare with Accumulator A or B only Status register is affected.
		ACX,ADR16	3							
	EOR	ACX,ADR8	2		X	X	0			(ACX)—(ACX) ∨ (MEM) Exclusive-OR with Accumulator A or B using direct, extended, or indexed addressing.
		ACX,ADR16	3							
	ORA	ACX,ADR8	2		X	X	0			(ACX)—(ACX) ∨ (MEM) OR with Accumulator A or B using direct, extended, or indexed addressing.
		ACX,ADR16	3							
	SUB	ACX,ADR8	2	X	X	X	X			(ACX)—(ACX) - (MEM) Subtract from Accumulator A or B using direct, extended, or indexed addressing.
		ACX,ADR16	3							
	SBC	ACX,ADR8	2	X	X	X	X			(ACX)—(ACX) - (MEM) - C Subtract with carry from Accumulator A or B using direct, extended, or indexed addressing.
		ACX,ADR16	3							
	CPX	ADR8	2		X	X	X			(XH:XL) - (MEM), (XL:01) - (MEM + 1) Compare with contents of Index register only Status register is affected. Sign and Overflow indicate reflect result on most significant byte.
		ADR16	3							
	CLR	ADR8	2	0	1	0	0			(M)—00_{16} Clear memory location using extended or indexed addressing.
		ADR16	3							
	COM	ADR8	2	1	X	X	0			(M)—$\overline{(M)}$ Complement contents of memory location forms complement.
		ADR16	3							
	NEG	ADR8	2	X	X	X	X			(M)—00_{16} - (M) Negate contents of memory location forms complement. Carry status is set if result is 00_{16} and reset otherwise. Overflow status is set if result is 80_{16} and reset otherwise.
		ADR16	3							

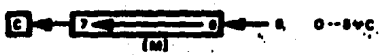
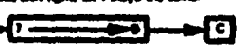
TYPE	MNEMONIC	OPERANDS	BYTES	STATUS C	STATUS Z	STATUS S	STATUS O	STATUS A_C	STATUS I	OPERATION PERFORMED
SECONDARY MEMORY REFERENCE MEMORY OPERATE (CONTINUED)	DEC	ADR8 ADR16	2 3		X	X	X			(M)←(M) - 1 Decrement contents of memory location, using extended or indexed addressing. Overflow status is set if operand was 80_{16} before execution, and cleared otherwise.
	INC	ADR8 ADR16	2 3		X	X	X			(M)←(M) + 1 Increment contents of memory location, using extended or indexed addressing. Overflow status is set if operand was $7F_{16}$ before execution, and cleared otherwise.
	ROL	ADR8 ADR16	2 3	X	X	X	X			 , O←S↑C Rotate contents of memory location left through carry.
	ROR	ADR8 ADR16	2 3	X	X	X	X			 , O←S↑C Rotate contents of memory location right through carry.
	ASL	ADR8 ADR16	2 3	X	X	X	X			 O←S↑C Arithmetic shift left. Bit 0 is set to 0.
	ASR	ADR8 ADR16	2 3	X	X	X	X			 O←S↑C Arithmetic shift right. Bit 7 stays the same.
	LSR	ADR8 ADR16	2 3	X	X	0	X			 O←S↑C Logical shift right. Bit 7 is set to 0.
	TST	ADR8 ADR16	2 3	0	X	X	0			(M) - 00_{16} Test contents of memory location for zero or negative value.
IMMEDIATE	LDA	ACX,DATA	2		X	X	0			(ACX)←DATA Load A or B immediate.
	LDX	DATA16	3		X	X	0			(XHI)←(S2), (XLO)←(S3) Load Index register immediate. Sign status reflects Index register bit 15.
	LDS	DATA16	3		X	X	0			(SP)←(S2), (XLO)←(S3) Load Stack Pointer immediate. Sign status reflects Stack Pointer bit 15.
IMMEDIATE OPERATE	ADD	ACX,DATA	2	X	X	X	X	X		(ACX)←(ACX) + DATA Add immediate to Accumulator A or B.
	ADC	ACX,DATA	2	X	X	X	X	X		(ACX)←(ACX) + DATA + C Add immediate with carry to Accumulator A or B.
	AND	ACX,DATA	2		X	X	0			(ACX)←(ACX) ∧ DATA AND immediate with Accumulator A or B.

Table 9-1. A Summary of the MC6800 Instruction Set (Continued)

TYPE	MNEMONIC	OPERANDS	BYTES	STATUS						OPERATION PERFORMED
				C	Z	S	O	A_C	I	
IMMEDIATE OPERATE CONTINUED	BIT	ACX,DATA	2		X	X	0			(ACX) A DATA AND immediate with Accumulator A or B, but only the Status register is affected.
	CMP	ACX,DATA	2	X	X	X	X			(ACX) - DATA Compare immediate with Accumulator A or B (only the Status register is affected).
	EOR	ACX,DATA	2		X	X	0			(ACX)—(ACX)⊕DATA Exclusive-OR immediate with Accumulator A or B.
	ORA	ACX,DATA	2		X	X	0			(ACX)—(ACX) V DATA OR immediate with Accumulator A or B.
	SUB	ACX,DATA	2	X	X	X	X			(ACX)—(ACX) - DATA Subtract immediate from Accumulator A or B.
	SBC	ACX,DATA	2	X	X	X	X			(ACX)—(ACX) - DATA - C Subtract immediate with carry from Accumulator A or B.
	CPX	DATA16	3		X	X	X			(XHI) - (M), (XLOW) - (M+1) Compare immediate with contents of Index register (only the Status register is affected). Sign and Overflow status reflect result on most significant byte.
JUMP	JMP	ADR8	2							(PC)—(X) + ADR8 or
		ADR16	3							(PCHI)—(M), (PCLOW)—(M+1) Jump to indexed or extended address.
	JSR	ADR8	2							((SP))—(PCLOW), ((SP)-1)—(PCHIGH), (SP)—(SP)-2
		ADR16	3							(PC)—(X) + ADR8 or
										(PCHIGH)—(M), (PCLOW)—(M+1) Jump to subroutine (indexed or extended addressing).
	BRA	DISP	2							(PC)—(PC) + DISP + 2 Unconditional branch relative to present Program Counter contents.
	BSR	DISP	2							((SP))—(PCLOW), ((SP)-1)—(PCHIGH), (SP)—(SP)-2, (PC)—(PC) + DISP + 2 Unconditional branch to subroutine located relative to present Program Counter contents.
BRANCH ON CONDITION	BCC	DISP	2							(PC)—(PC) + DISP + 2 if the given condition is true:
	BCS	DISP	2							C = 0 (Branch if carry clear)
	BEQ	DISP	2							C = 1 (Branch if carry set)
	BGE	DISP	2							Z = 1 (Branch if equal to zero)
	BGT	DISP	2							S⊻O = 0 (Branch if greater than or equal to zero)
	BHI	DISP	2							Z V (S⊻O) = 0 (Branch if greater than zero)
	BLE	DISP	2							C V Z = 0 (Branch if Accumulator contents higher than compared)
	BLS	DISP	2							Z V (S⊻O) = 1 (Branch if less than or equal to zero)
	BLT	DISP	2							C V Z = 1 (Branch if Accumulator contents less than or same as compared)
	BMI	DISP	2							S⊻O = 1 (Branch if less than zero)
	BNE	DISP	2							S = 1 (Branch if minus)
	BVC	DISP	2							Z = 0 (Branch if not equal to zero)
	BVS	DISP	2							O = 0 (Branch if overflow clear)
	BPL	DISP	2							O = 1 (Branch if overflow set)
										S = 0 (Branch if plus)

Table 9-1. A Summary of the MC6800 Instruction Set (Continued)

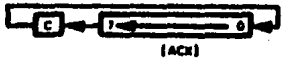
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS C	Z	S	O	A_C	I	OPERATION PERFORMED
REGISTER-EXAMINING OPS	TAB		1		X	X	0			(B)←(A) Move Accumulator A contents to Accumulator B.
	TBA		1		X	X	0			(A)←(B) Move Accumulator B contents to Accumulator A.
	TXS		1							(SP)←(X)-1 Move Index register contents to Stack Pointer and decrement.
	TSX		1							(X)←(SP)+1 Move Stack Pointer contents to Index register and increment.
ARITHMETIC OPERATIONS	ABA		1	X	X	X	X	X		(A)←(A)+(B) Add contents of Accumulators A and B.
	CBA		1	X	X	X	X			(A)-(B) Compare contents of Accumulators A and B. Only the Status register is affected.
	SBA		1	X	X	X	X			(A)←(A)-(B) Subtract contents of Accumulator B from those of Accumulator A.
READ/WRITE OPERATE	CLR	ACX	1	0	1	0	0			(ACX)←00_{16} Clear Accumulator A or B.
	COM	ACX	1	1	X	X	0			(ACX)←$\overline{(ACX)}$ Complement contents of Accumulator A or B (ones complement).
	NEG	ACX	1	X	X	X	X			(ACX)←00_{16}-(ACX) Negate contents of Accumulator A or B (twos complement). Carry status is set if result is 00_{16} and reset otherwise. Overflow status is set if result is 80_{16} and reset otherwise.
	DAA		1	X	X	X	X			Decimal adjust A. Convert contents of A (the binary sum of BCD operands) to BCD format. Carry status is set if value of upper four bits is greater than 9, but not cleared if previously set.
	DEC	ACX	1		X	X	X			(ACX)←(ACX)-1 Decrement contents of Accumulator A or B. Overflow status is set if operand was 80_{16} before execution, and cleared otherwise.
	DEX		1		X					(X)←(X)-1 Decrement contents of Index register.
	DES		1							(SP)←(SP)-1 Decrement contents of Stack Pointer.
	INC	ACX	1		X	X	X			(ACX)←(ACX)+1 Increment contents of Accumulator A or B. Overflow status is set if operand was $7F_{16}$ before execution, and cleared otherwise.
	INX		1		X					(X)←(X)+1 Increment contents of Index register.
	INS		1							(SP)←(SP)+1 Increment contents of Stack Pointer.
	ROL	ACX	1	X	X	X	X			 (ACX) O←S⊻C Rotate Accumulator A or B left through carry.

Table 9-1. A Summary of the MC6800 Instruction Set (Continued)

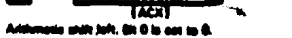
TYPE	MNEMONIC	OPERANDS	BYTES	STATUS						OPERATION PERFORMED
				C	Z	S	O	A_C	I	
REGISTER OPERATE CONTINUED	ROR	ACX	1	X	X	X	X			 0 → 8 → C Rotate Accumulator A or B right through carry.
	ASL	ACX	1	X	X	X	X			 0 ← 8 ← C Arithmetic shift left. Bit 0 is set to 0.
	ASR	ACX	1	X	X	X	X			 0 → 8 → C Arithmetic shift right. Bit 7 stays the same.
	LSR	ACX	1	X	X	0	X			 0 → 8 → C Logical shift right. Bit 7 is set to 0.
	TST	ACX	1	0	X	X	0			[ACX] - 00_{16} Test contents of Accumulator A or B for zero or negative value.
STACK	PSH	ACX	1							[[SP]]←[ACX] [SP]←[SP] - 1 Push contents of Accumulator A or B onto top of Stack and decrement Stack Pointer.
	PUL	ACX	1							[SP]←[SP] + 1 [ACX]←[[SP]] Increment Stack Pointer and pull Accumulator A or B from top of Stack.
	RTS		1							[PCHI]←[[SP] + 1], [PCLO]←[[SP] + 2], [SP]←[SP] + 2 Return from subroutine. Pull PC from top of Stack and increment Stack Pointer.
INTERRUPT	CLI		1						0	I ← 0 Clear interrupt mask to enable interrupts.
	SEI		1						1	I ← 1 Set interrupt mask to disable interrupts.
	RTI		1	X	X	X	X	X	X	[CR]←[[SP] + 1], [B]←[[SP] + 2], [A]←[[SP] + 3], [XHI]←[[SP] + 4], [XLO]←[[SP] + 5], [PCHI]←[[SP] + 6], [PCLO]←[[SP] + 7], [SP]←[SP] + 7 Return from interrupt. Pull registers from Stack and increment Stack Pointer.

Table 9-1. A Summary of the MC6800 Instruction Set (Continued)

TYPE	MNEMONIC	OPERANDS	BYTES	STATUS						OPERATIONS PERFORMED
				C	Z	S	O	H_C	I	
INTERRUPT CONTROL	SWI		1						1	((SP))—(PCL) ((SP)-1)—(PCH) ((SP)-2)—(IXL) ((SP)-3)—(IXH) ((SP)-4)—(A) ((SP)-5)—(B) ((SP)-6)—(CR) (SP)—(SP)-7, (PCH)—(FFFA$_{16}$) (PCL)—(FFFB$_{16}$) Software interrupt: push registers onto Stack, decrement Stack Pointer, and jump to interrupt subroutine.
	WAI		1						1	((SP))—(PCL) ((SP)-1)—(PCH) ((SP)-2)—(IXL) ((SP)-3)—(IXH) ((SP)-4)—(A) ((SP)-5)—(B) ((SP)-6)—(CR) (SP)—(SP)-7 Push registers onto Stack, decrement Stack Pointer, and wait for interrupt. If (I)=1 when WAI is executed, a non-maskable interrupt is required to exit the Wait state. Otherwise, (I)=1 when the interrupt occurs.
STATUS	CLC		1	0						C—0 Clear carry
	SEC		1	1						C—1 Set carry
	CLV		1				0			O—0 Clear overflow status bit
	SEV		1				1			O—1 Set overflow status bit
	TAP		1	↕	↕	↕	↕	↕	↕	(CR)—(A) Transfer contents of Accumulator A to Status register.
	TPA		1							(A)—(CR) Transfer contents of Status register to Accumulator A.
	NOP		1							No Operation

*Address Bus: A8-A7: (C)
A8-A15: (B)

Table 7-2. A Summary of the Z80 Instruction Set

TYPE	MNEMONIC	OPERANDS(s)	BYTES	STATUS						OPERATION PERFORMED
				C	Z	S	P/O	A_C	N	
I/O	IN	A,port	2							(A)←(port) Input to Accumulator from directly addressed I/O port. Address Bus: A0-A7: port A8-A15: (A)
	IN	reg,(C)	2		X	X	P	X	0	(reg)←((C)) Input to register from I/O port addressed by the contents of C.* If second byte is 70_{16} only the flags will be affected.
	INIR		2		1	?	?	?	1	Repeat until (B)=0: ((HL))←((C)) (B)←(B) - 1 (HL)←(HL) + 1 Transfer a block of data from I/O port addressed by contents of C to memory location addressed by contents of HL, going from low addresses to high. Contents of B serve as a count of bytes remaining to be transferred.*
	INDR		2		1	?	?	?	1	Repeat until (B)=0: ((HL))←((C)) (B)←(B) - 1 (HL)←(HL) - 1 Transfer a block of data from I/O port addressed by contents of C to memory location addressed by contents of HL, going from high addresses to low. Contents of B serve as a count of bytes remaining to be transferred.*
	INI		2		X	?	?	?	1	((HL))←((C)) (B)←(B) - 1 (HL)←(HL) + 1 Transfer a byte of data from I/O port addressed by contents of C to memory location addressed by contents of HL. Decrement byte count and increment destination address.*
	IND		2		X	?	?	?	1	((HL))←((C)) (B)←(B) - 1 (HL)←(HL) - 1 Transfer a byte of data from I/O port addressed by contents of C to memory location addressed by contents of HL. Decrement both byte count and destination address.*
	OUT	port,A	2							(port)←(A) Output from Accumulator to directly addressed I/O port. Address Bus: A0-A7: port A8-A15: (A)
	OUT	(C),reg	2							((C))←(reg) Output from register to I/O port addressed by the contents of C.*
	OTIR		2		1	?	?	?	1	Repeat until (B)=0: ((C))←((HL)) (B)←(B) - 1 (HL)←(HL) + 1 Transfer a block of data from memory location addressed by contents of HL to I/O port addressed by contents of C, going from low memory to high. Contents of B serve as a count of bytes remaining to be transferred.*

*Address Bus: A8-A15: (B)
A0-A15: (C)

Table 7-2. A Summary of the Z80 Instruction Set (Continued)

TYPE	MNEMONIC	OPERANDS(S)	BYTES	STATUS C	Z	S	P/O	A_C	N	OPERATION PERFORMED
I/O (Continued)	OTDR		2		1	?	?	?	1	Repeat until (B)=0: ((C))←((HL)) (B)←(B) - 1 (HL)←(HL) - 1 Transfer a block of data from memory location addressed by contents of HL to I/O port addressed by contents of C, going from high memory to low. Contents of B serve as a count of bytes remaining to be transferred.*
	OUTI		2		X	?	?	?	1	((C))←((HL)) (B)←(B) - 1 (HL)←(HL) + 1 Transfer a byte of data from memory location addressed by contents of HL to I/O port addressed by contents of C. Decrement byte count and increment source address.*
	OUTD		2		X	?	?	?	1	((C))←((HL)) (B)←(B) - 1 (HL)←(HL) - 1 Transfer a byte of data from memory location addressed by contents of HL to I/O port addressed by contents of C. Decrement both byte count and source address.*
PRIMARY MEMORY REFERENCE	LD	A,(addr)	3							(A)←(addr) Load Accumulator from directly addressed memory location.
	LD	HL,(addr)	3							(H)←(addr + 1), (L)←(addr) Load HL from directly addressed memory.
	LD	rp,(addr) ry,(addr)	4							(rpHI)←(addr + 1), (rpLOW)←(addr) or (ryHI)←(addr + 1), (ryLOW)←(addr) Load register pair or index register from directly addressed memory.
	LD	(addr),A	3							(addr)←(A) Store Accumulator contents in directly addressed memory location.
	LD	(addr),HL	3							(addr + 1)←(H), (addr)←(L) Store contents of HL to directly addressed memory location.
	LD	(addr),rp (addr),ry	4							(addr + 1)←(rpHI), (addr)←(rpLOW) or (addr + 1)←(ryHI), (addr)←(ryLOW) Store contents of register pair or index register to directly addressed memory.
	LD	A,(BC) A,(DE)	1							(A)←((BC)) or (A)←((DE)) Load Accumulator from memory location addressed by the contents of the specified register pair.
	LD	reg,(HL)	1							(reg)←((HL)) Load register from memory location addressed by contents of HL.
	LD	(BC),A (DE),A	1							((BC))←(A) or ((DE))←(A) Store Accumulator to memory location addressed by the contents of the specified register pair.
	LD	(HL),reg	1							((HL))←(reg) Store register contents to memory location addressed by the contents of HL.
	LD	reg,(ry + disp)	3							(reg)←((ry) + disp) Load register from memory location using base relative addressing.
	LD	(ry + disp),reg	3							((ry) + disp)←(reg) Store register to memory location addressed relative to contents of index register.

Table 7-2. A Summary of the Z80 Instruction Set (Continued)

TYPE	MNEMONIC	OPERANDS	BYTES	STATUS						OPERATION PERFORMED
				C	Z	S	P/O	A_C	N	
BLOCK TRANSFER AND SEARCH	LDIR		2				0	0	0	Repeat until (BC)=0: ((DE))←((HL)) (DE)←(DE) + 1 (HL)←(HL) + 1 (BC)←(BC) - 1 Transfer a block of data from the memory location addressed by the contents of HL to the memory location addressed by the contents of DE, going from low addresses to high. Contents of BC serve as a count of bytes to be transferred.
	LDDR		2				0	0	0	Repeat until (BC)=0: ((DE))←((HL)) (DE)←(DE) - 1 (HL)←(HL) - 1 (BC)←(BC) - 1 Transfer a block of data from the memory location addressed by the contents of HL to the memory location addressed by the contents of DE, going from high addresses to low. Contents of BC serve as a count of bytes to be transferred.
	LDI		2				X	0	0	((DE))←((HL)) (DE)←(DE) + 1 (HL)←(HL) + 1 (BC)←(BC) - 1 Transfer one byte of data from the memory location addressed by the contents of HL to the memory location addressed by the contents of DE. Increment source and destination addresses and decrement byte count.
	LDD		2				X	0	0	((DE))←((HL)) (DE)←(DE) - 1 (HL)←(HL) - 1 (BC)←(BC) - 1 Transfer one byte of data from the memory location addressed by the contents of HL to the memory location addressed by the contents of DE. Decrement source and destination addresses and byte count.
	CPIR		2		X	X	X	X	1	Repeat until (A)=((HL)) or (BC)=0: (A) - ((HL)) (only flags are affected) (HL)←(HL) + 1 (BC)←(BC) - 1 Compare contents of Accumulator with those of memory block addressed by contents of HL, going from low addresses to high. Stop when a match is found or when the byte count becomes zero.
	CPDR		2		X	X	X	X	1	Repeat until (A)=((HL)) or (BC)=0: (A) - ((HL)) (only flags are affected) (HL)←(HL) - 1 (BC)←(BC) - 1 Compare contents of Accumulator with those of memory block addressed by contents of HL, going from high addresses to low. Stop when a match is found or when the byte count becomes zero.

Table 7-2. A Summary of the Z80 Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS C	Z	S	P/O	A_C	N	OPERATION PERFORMED
BLOCK TRANSFER AND SEARCH (Continued)	CPI		2		X	X	X	X	1	(A) - ((HL)) (only flags are affected) (HL)←(HL) + 1 (BC)←(BC) - 1 Compare contents of Accumulator with those of memory location addressed by contents of HL. Increment address and decrement byte count.
	CPD		2		X	X	X	X	1	(A) - ((HL)) (only flags are affected) (HL)←(HL) - 1 (BC)←(BC) - 1 Compare contents of Accumulator with those of memory location addressed by contents of HL. Decrement address and byte count.
SECONDARY ARITHMETIC REFERENCE	ADD	(HL) (Ixy + disp)	1 3	X	X	X	O	X	0	(A)←(A) + ((HL)) or (A)←(A) + ((Ixy) + disp) Add to Accumulator using implied addressing or base relative addressing.
	ADC	(HL) (Ixy + disp)	1 3	X	X	X	O	X	0	(A)←(A) + ((HL)) + C or (A)←(A) + ((Ixy) + disp) + C Add with Carry using implied addressing or base relative addressing.
	SUB	(HL) (Ixy + disp)	1 3	X	X	X	O	X	1	(A)←(A) - ((HL)) or (A)←(A) - ((Ixy) + disp) Subtract from Accumulator using implied addressing or base relative addressing.
	SBC	(HL) (Ixy + disp)	1 3	X	X	X	O	X	1	(A)←(A) - ((HL)) - C or (A)←(A) - ((Ixy) + disp) - C Subtract with Carry using implied addressing or base relative addressing.
	AND	(HL) (Ixy + disp)	1 3	0	X	X	P	1	0	(A)←(A) ∧ ((HL)) or (A)←(A) ∧ ((Ixy) + disp) AND with Accumulator using implied addressing or base relative addressing.
	OR	(HL) (Ixy + disp)	1 3	0	X	X	P	1	0	(A)←(A) ∨ ((HL)) or (A)←(A) ∨ ((Ixy) + disp) OR with Accumulator using implied addressing or base relative addressing.
	XOR	(HL) (Ixy + disp)	1 3	0	X	X	P	1	0	(A)←(A) ⊻ ((HL)) or (A)←(A) ⊻ ((Ixy) + disp) Exclusive-OR with Accumulator using implied addressing or base relative addressing.
	CP	(HL) (Ixy + disp)	1 3	X	X	X	O	X	1	(A) - ((HL)) or (A) - ((Ixy) + disp) Compare with Accumulator using implied addressing or base relative addressing. Only the flags are affected.
	INC	(HL) (Ixy + disp)	1 3		X	X	O	X	0	((HL))←((HL)) + 1 or ((Ixy) + disp)←((Ixy) + disp) + 1 Increment using implied addressing or base relative addressing.
	DEC	(HL) (Ixy + disp)	1 3		X	X	O	X	1	((HL))←((HL)) - 1 or ((Ixy) + disp)←((Ixy) + disp) - 1 Decrement using implied addressing or base relative addressing.

Table 7-2. A Summary of the Z80 Instruction Set (Continued)

TYPE	MNEMONIC	OPERANDS	BYTES	STATUS C	Z	S	P/V	A_C	N	OPERATION PERFORMED
ROTATE SHIFT AND	RLC	(HL) (xy + disp)	2 4	X	X	X	P	0	0	 ((HL)) or ((xy) + disp) Rotate contents of memory location (implied or base relative addressing) left with branch Carry.
	RL	(HL) (xy + disp)	2 4	X	X	X	P	0	0	 ((HL)) or ((xy) + disp) Rotate contents of memory location left through Carry.
	RRC	(HL) (xy + disp)	2 4	X	X	X	P	0	0	 ((HL)) or ((xy) + disp) Rotate contents of memory location right with branch Carry.
	RR	(HL) (xy + disp)	2 4	X	X	X	P	0	0	 ((HL)) or ((xy) + disp) Rotate contents of memory location right through Carry.
	SLA	(HL) (xy + disp)	2 4	X	X	X	P	0	0	 ((HL)) or ((xy) + disp) Shift contents of memory location left and clear LSB (Arithmetic Shift).
	SRA	(HL) (xy + disp)	2 4	X	X	X	P	0	0	 ((HL)) or ((xy) + disp) Shift contents of memory location right and preserve MSB (Arithmetic Shift).
	SRL	(HL) (xy + disp)	2 4	X	X	X	P	0	0	 ((HL)) or ((xy) + disp) Shift contents of memory location right and clear MSB (Logical Shift).
IMMEDIATE	LD	reg,data	2							(reg)←data Load immediate into register.
	LD	rp,data16 xy,data16	3 4							(rp)←data16 or (xy)←data16 Load 16 bits of immediate data into register pair or index register.
	LD	(HL),data (xy + disp),data	2 4							((HL))←data or ((xy) + disp)←data Load immediate into memory location using implied or base relative addressing.

Table 7-2. A Summary of the 280 Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS						OPERATION PERFORMED
				C	Z	S	P/O	AC	N	
JUMP	JP	label	3							[PC]←label Jump to instruction at address represented by label.
	JR	disp	2							[PC]←[PC] + 2 + disp Jump relative to present contents of Program Counter.
	JP	(HL) (ix)/(iy)	1 2							[PC]←[HL] or [PC]←[ix] Jump to address contained in HL or index register.
SUBROUTINE CALL AND RETURN	CALL	label	3							[[SP]-1]←[PCHI] [[SP]-2]←[PCLO] [SP]←[SP]-2 [PC]←label Jump to subroutine starting at address represented by label.
	CALL cond	cond,label	3							Jump to subroutine if condition is satisfied; otherwise, continue in sequence.
	RET		1							[PCLO]←[[SP]] [PCHI]←[[SP] + 1] [SP]←[SP] + 2 Return from subroutine.
	RET cond	cond	1							Return from subroutine if condition is satisfied; otherwise, continue in sequence.
IMMEDIATE OPERATE	ADD	data	2	X	X	X	O	X	0	[A]←[A] + data Add immediate to Accumulator.
	ADC	data	2	X	X	X	O	X	0	[A]←[A] + data + C Add immediate with Carry.
	SUB	data	2	X	X	X	O	X	1	[A]←[A] - data Subtract immediate from Accumulator.
	SBC	data	2	X	X	X	O	X	1	[A]←[A] - data - C Subtract immediate with Carry.
	AND	data	2	0	X	X	P	1	0	[A]←[A] ∧ data AND immediate with Accumulator.
	OR	data	2	0	X	X	P	1	0	[A]←[A] ∨ data OR immediate with Accumulator.
	XOR	data	2	0	X	X	P	1	0	[A]←[A] ⊻ data Exclusive-OR immediate with Accumulator.
	CP	data	2	X	X	X	O	X	1	[A] - data Compare immediate data with Accumulator contents; only the flags are affected.

Table 7-2. A Summary of the Z80 Instruction Set (Continued)

TYPE	MNEMONIC	OPERANDS	BYTES	STATUS						OPERATION PERFORMED
				C	Z	S	P/O	A_C	N	
JUMP GROUP	JP	cond,label	3							If cond, then (PC)←label Jump to instruction at address represented by label if the condition is true.
	JR	C,disp	2							If C=1, then (PC)←(PC)+2+disp Jump relative to contents of Program Counter if Carry flag is set.
	JR	NC,disp	2							If C=0, then (PC)←(PC)+2+disp Jump relative to contents of Program Counter if Carry flag is reset.
	JR	Z,disp	2							If Z=1, then (PC)←(PC)+2+disp Jump relative to contents of Program Counter if Zero flag is set.
	JR	NZ,disp	2							If Z=0, then (PC)←(PC)+2+disp Jump relative to contents of Program Counter if Zero flag is reset.
	DJNZ	disp	2							(B)←(B)-1 If (B)≠0, then (PC)←(PC)+2+disp Decrement contents of B and Jump relative to contents of Program Counter if result is not 0.
8-BIT LOAD GROUP	LD	dst,src	1							(dst)←(src) Move contents of source register to destination register. Register designations src and dst may each be A, B, C, D, E, H or L.
	LD	A,I	2		X	X	I	0	0	(A)←(I) Move contents of Interrupt Vector register to Accumulator.
	LD	A,R	2		X	X	I	0	0	(A)←(R) Move contents of Refresh register to Accumulator.
	LD	I,A	2							(I)←(A) Load Interrupt Vector register from Accumulator.
	LD	R,A	2							(R)←(A) Load Refresh register from Accumulator.
16-BIT LOAD GROUP	LD	SP,HL	1							(SP)←(HL) Move contents of HL to Stack Pointer.
	LD	SP,xy	2							(SP)←(xy) Move contents of Index register to Stack Pointer.
	EX	DE,HL	1							(DE)←→(HL) Exchange contents of DE and HL.
	EX	AF,AF'	1							(AF)←→(AF') Exchange program status and alternate program status.
	EXX		1							$\begin{pmatrix}(BC)\\(DE)\\(HL)\end{pmatrix} \longleftrightarrow \begin{pmatrix}(BC')\\(DE')\\(HL')\end{pmatrix}$ Exchange register pairs and alternate register pairs.

Table 7-2. A Summary of the Z80 Instruction Set (Continued)

TYPE	MNEMONIC	OPERANDS	BYTES	STATUS						OPERATION PERFORMED
				C	Z	S	P/V	H_C	N	
16-BIT ARITHMETIC GROUP	ADD	ss	1	X	X	X	O	X	0	(A)←(A) + (ss) Add contents of register to Accumulator.
	ADC	ss	1	X	X	X	O	X	0	(A)←(A) + (ss) + C Add contents of register and Carry to Accumulator.
	SUB	ss	1	X	X	X	O	X	1	(A)←(A) - (ss) Subtract contents of register from Accumulator.
	SBC	ss	1	X	X	X	O	X	1	(A)←(A) - (ss) - C Subtract contents of register and Carry from Accumulator.
	AND	ss	1	0	X	X	P	1	0	(A)←(A) ∧ (ss) AND contents of register with contents of Accumulator.
	OR	ss	1	0	X	X	P	1	0	(A)←(A) ∨ (ss) OR contents of register with contents of Accumulator.
	XOR	ss	1	0	X	X	P	1	0	(A)←(A) ∨ (ss) Exclusive-OR contents of register with contents of Accumulator.
	CP	ss	1	X	X	X	O	X	1	(A) - (ss) Compare contents of register with contents of Accumulator. Only the flags are affected.
	ADD	HL,ss	1	X				?	0	(HL)←(HL) + (ss) 16-bit add register pair contents to contents of HL.
	ADC	HL,ss	2	X	X	X	O	?	0	(HL)←(HL) + (ss) + C 16-bit add with Carry register pair contents to contents of HL.
	SBC	HL,ss	2	X	X	X	O	?	1	(HL)←(HL) - (ss) - C 16-bit subtract with Carry register pair contents from contents of HL.
	ADD	IX,pp	2	X				?	0	(IX)←(IX) + (pp) 16-bit add register pair contents to contents of Index register IX (pp=BC, DE, IX, SP)
	ADD	IY,rr	2	X				?	0	(IY)←(IY) + (rr) 16-bit add register pair contents to contents of Index register IY (rr=BC, DE, IY, SP).
GENERAL PURPOSE ARITHMETIC AND CPU CONTROL GROUPS	DAA		1	X	X	X	P	X		Decimal adjust Accumulator, causing that Accumulator contents are the sum or difference of BCD operands.
	CPL		1					1	1	(A)←(Ā) Complement Accumulator (take complement).
	NEG		2	X	X	X	O	X	1	(A)←(Ā) + 1 Negate Accumulator (take complement).
	INC	ss	1		X	X	O	X	0	(ss)←(ss) + 1 Increment register contents.
	INC	IX IY	1							(pp)←(pp) + 1 or (rr)←(rr) + 1 Increment contents of register pair or Index register.
	DEC	ss	2		X	X	O	X	1	(ss)←(ss) - 1 Decrement register contents.
	DEC	IX IY	1 2							(pp)←(pp) - 1 or (rr)←(rr) - 1 Decrement contents of register pair or Index register.

Table 7-2. A Summary of the Z80 Instruction Set (Continued)

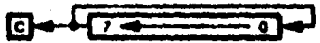
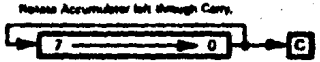
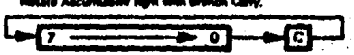
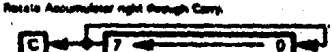
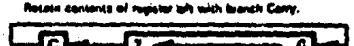
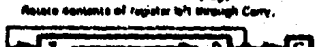
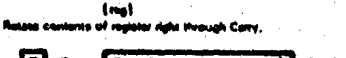
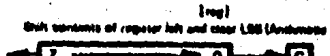
TYPE	MNEMONIC	OPERANDS	BYTES	STATUS C	Z	S	P/O	H_C	N	OPERATION PERFORMED
REGISTER SHIFT AND ROTATE	RLCA		1	X				0	0	 (A) Rotate Accumulator left with branch Carry.
	RLA		1	X				0	0	 (A) Rotate Accumulator left through Carry.
	RRCA		1	X				0	0	 (A) Rotate Accumulator right with branch Carry.
	RRA		1	X				0	0	 (A) Rotate Accumulator right through Carry.
	RLC	reg	2	X	X	X	P	0	0	 (reg) Rotate contents of register left with branch Carry.
	RL	reg	2	X	X	X	P	0	0	 (reg) Rotate contents of register left through Carry.
	RRC	reg	2	X	X	X	P	0	0	 (reg) Rotate contents of register right with branch Carry.
	RR	reg	2	X	X	X	P	0	0	 (reg) Rotate contents of register right through Carry.
	SLA	reg	2	X	X	X	P	0	0	 (reg) Shift contents of register left and clear LSB (Arithmetic Shift).
	SRA	reg	2	X	X	X	P	0	0	 (reg) Shift contents of register right and preserve MSB (Arithmetic Shift).

Table 7-2. A Summary of the Z80 Instruction Set (Continued)

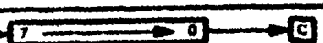
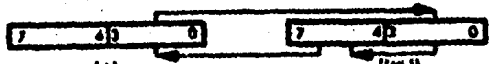
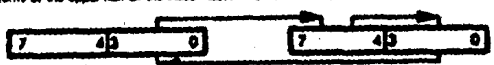
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS C	Z	S	P/O	A_C	N	OPERATION PERFORMED
ROTATE SHIFT AND ROTATE (Continued)	SRL		2	X	X	X	P	0	0	 {reg} Shift contents of register right and clear MSB (Logical Shift).
	RLD		2		X	X	P	0	0	 Rotate one BCD digit left between the Accumulator and memory location (implied addressing). Contents of the upper half of the Accumulator are not affected.
	RRD	reg	2		X	X	P	0	0	 Rotate one BCD digit right between the Accumulator and memory location (implied addressing). Contents of the upper half of the Accumulator are not affected.
BIT MANIPULATION	BIT	b,reg	2		X	?	?	1	0	Z←$\overline{reg(b)}$ Zero flag contains complement of the selected register bit.
	BIT	b,(HL) b,(ry + disp)	2 4		X	?	?	1	0	Z←$\overline{((HL)b)}$ or Z←$\overline{((ry)+disp)b}$ Zero flag contains complement of selected bit of the memory location (implied addressing or base relative addressing).
	SET	b,reg	2							reg(b)←1 Set indicated register bit.
	SET	b,(HL) b,(ry + disp)	2 4							((HL)b)←1 or ((ry) + disp)b)←1 Set indicated bit of memory location (implied addressing or base relative addressing).
	RES	b,reg	2							reg(b)←0 Reset indicated register bit.
	RES	b,(HL) b,(ry + disp)	2 4							((HL)b)←0 or ((ry) + disp)b)←0 Reset indicated bit in memory location (implied addressing or base relative addressing).
STACK	PUSH	qr ry	1 2							((SP)-1)←(pcHI) ((SP)-2)←(pcLO) (SP)←(SP)-2 Put contents of register pair or index register on top of Stack and decrement Stack Pointer.
	POP	qr ry	1 2							(pcLO)←((SP)) (pcHI)←((SP) + 1) (SP)←(SP) + 2 Put contents of top of Stack in register pair or index register and increment Stack Pointer.
	EX	(SP),HL (SP),ry	1 2							(H)←((SP) + 1) (L)←((SP)) Exchange contents of HL or index register and top of Stack.

Table 7-2. A Summary of the Z80 Instruction Set (Continued)

TYPE	MNEMONIC	OPERANDS	BYTES	STATUS						OPERATION PERFORMED
				C	Z	S	P/O	A_C	N	
INTERRUPT	DI		1							Disable interrupts.
	EI		1							Enable interrupts.
	RST	n	1							((SP)-1)←(PCHI) ((SP)-2)←(PCLO) (SP)←(SP)-2 (PC)←(n-n)$_{16}$ Restart to designated location.
	RETI		2							Return from interrupt.
	RETN		2							Return from nonmaskable interrupt.
	IM	0 1 2	2							Set interrupt mode 0, 1, or 2.
STATIC	SCF		1	1				0	0	C ← 1 Set Carry flag.
	CCF		1	X				?	0	C ← $\overline{C}$ Complement Carry flag.
	NOP		1							No operation — volatile memories are refreshed.
	HALT		1							CPU halts, executes NOPs to refresh volatile memories.

Table 10-2. A Summary of the MCS6500 Microcomputer Instruction Set

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				S	Z	C	O			
NO AND PRIMARY MEMORY REFERENCE	LDA	ADR ADR,X (ADR,X) (ADR),Y ADR16 ADR16,X ADR16,Y	2 2 2 2 3 3 3	X	X					(A)←(ADR) or (A)←((X)+ADR) or (A)←M((X)+ADR)) or (A)←M(M(ADR)+(Y)) or (A)←(ADR16) or (A)←(ADR16)+(X)) or (A)←(ADR16)+(Y)) Load Accumulator from memory using any of the following addressing modes: Base page direct Base page indexed (X register) Pre-indexed indirect Post-indexed indirect Extended direct Absolute indexed (Register X or Register Y)
	STA	adr adr16	2 3							M(adr)←(A) or M(adr16)←(A) Store Accumulator to memory using any of the addressing modes permitted with LDA.
	LDX	ADR or ADR,Y ADR16 or ADR16,Y	2 3	X	X					(X)←(ADR) or (X)←(ADR16) or (X)←((Y)+ADR) or (X)←(ADR16+(Y) Load Index Register X from memory using direct, extended, base page indexed or absolute indexed addressing, indexing through Register Y.
	STX	ADR or ADR,Y ADR16	2 3							(ADR)←(X) or (ADR16)←(X) or ((Y)+ADR)←(X) Store Index Register X to memory using direct, extended or base page indexed addressing, indexing through Register Y.
	LDY	ADR or ADR,X ADR16 or ADR16,X	2 3	X	X					(Y)←(ADR) or (Y)←(ADR16) or (Y)←((X)+ADR) or (Y)←(ADR16+(X)) Load Index Register Y from memory using direct, extended, base page indexed or absolute indexed addressing, indexing through Register X.
	STY	ADR or ADR,X ADR16	2							(ADR)←(Y) or (ADR16)←(Y) or ((X)+ADR)←(Y) Store Index Register Y to memory using direct, extended, or base page indexed addressing, indexing through Register X.
MEMORY ARITHMETIC	ADC	adr adr16	2 3	X	X	X	X			(A)←(A)+M(adr)+C or (A)←(A)+M(adr16)+C Add contents of memory location, with carry, to those of Accumulator, using any of the addressing modes permitted with LDA. Base flag is not valid in Decimal Mode.
	AND	adr adr16	2 3	X	X					(A)←(A) ∧ M(adr) or (A)←(A) ∧ M(adr16) AND contents of Accumulator with those of memory location addressed via any of the modes permitted with LDA.
	BIT	ADR8 ADR16	2 3	?	X		6			(A) ∧ (ADR8) or (A) ∧ (ADR16) AND contents of Accumulator with those of memory location. Only the status bits are affected. Direct or extended addressing modes may be used.

Table 10-2 A Summary of the MCS6500 Microcomputer Instruction Set (Continued)

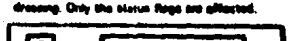
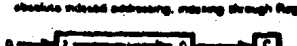
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				S	Z	C	O			
SECONDARY MEMORY REFERENCE (MEMORY OPERATE) (CONTINUED)	CMP	a8 a16	2 3	X	X	X				(A) - M(a8) or (A) - M(a16) Compare contents of Accumulator with those of memory location, affecting status bit only. Any of the addressing modes permitted with LDA may be used.
	EOR	a8 a16	2 3	X	X					(A)←(A)∨M(a8) or (A)←(A)∨M(a16) Exclusive-OR contents of Accumulator with those of memory location, using any of the addressing modes permitted with LDA.
	ORA	a8 a16	2 3	X	X					(A)←(A)∨M(a8) or (A)←(A)∨M(a16) OR contents of Accumulator with those of memory location, using any of the addressing modes permitted with LDA.
	SBC	a8 a16	2 3	X	X	X	X			(A)←(A) - M(a8) - $\overline{C}$ or (A)←(A) - M(a16) - $\overline{C}$ Subtract contents of memory location, with borrow, from contents of Accumulator. Any addressing mode permitted with LDA may be used. Note that Carry reflects the complement of the borrow.
	INC	ADR8 or ADR,X ADR16 or ADR16,X	2 3	X	X					(ADR)←(ADR) + 1 or (ADR16)←(ADR16) + 1 or ((X) + ADR)←((X) + ADR) + 1 or (ADR16 + (X))←(ADR16 + (X)) + 1 Increment contents of memory location using direct, extended, base page indexed or absolute indexed addressing, indexing through Register X.
	DEC	ADR or ADR,X ADR16 or ADR16,X	2 3	X	X					(ADR)←(ADR) - 1 or (ADR16)←(ADR16) - 1 or ((X) + ADR)←((X) + ADR) - 1 or (ADR16 + (X))←(ADR16 + (X)) - 1 Decrement contents of memory location using direct, extended, base page indexed or absolute indexed addressing, indexing through Register X.
	CPX	ADR ADR16	2 3	X	X	X				(X) - (ADR) or (X) - (ADR16) Compare contents of X register with those of memory location, using direct or extended addressing. Only the status flags are affected.
	CPY	ADR ADR16	2 3	X	X	X				(Y) - (ADR) or (Y) - (ADR16) Compare contents of Y register with those of memory location using direct or extended addressing. Only the status flags are affected.
	ROL	ADR or ADR,X ADR16 or ADR16,X	2 3	X	X	X				 (ADR) or (ADR16) or ((X) + ADR) or (ADR16 + (X)) Rotate contents of memory location left through Carry, using direct, extended, base page indexed or absolute indexed addressing, indexing through Register X.
	ASL	ADR or ADR,X ADR16 or ADR16,X	2 3	X	X	X				 (ADR) or (ADR16) or ((X) + ADR) or (ADR16 + (X)) Arithmetic shift left contents of memory location using direct, extended, base page indexed or absolute indexed addressing, indexing through Register X.
	LSR	ADR or ADR,X ADR16 or ADR16,X	2 3	0	X	X				 (ADR) or (ADR16) or ((X) + ADR) or (ADR16 + (X)) Logical shift right contents of memory location, using direct, extended, base page indexed or absolute indexed addressing, indexing through Register X.

Table 10-2 A Summary of the MCS6500 Microcomputer Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				S	Z	C	O			
IMMEDIATE	LDA	DATA	2	X	X					(A)—DATA Load Accumulator with immediate data.
	LDX	DATA	2	X	X					(X)—DATA Load Index Register X with immediate data.
	LDY	DATA	2	X	X					(Y)—DATA Load Index Register Y with immediate data.
IMMEDIATE OPERATE	ADC	DATA	2	X	X	X	X			(A)—(A) + DATA + C Add immediate, with Carry, to Accumulator. The Zero flag is not valid in Decimal Mode
	AND	DATA	2	X	X					(A)—(A) ∧ DATA AND immediate with Accumulator.
	CMP	DATA	2	X	X	X				(A) - DATA Compare immediate with Accumulator. Only the status flags are affected.
	EOR	DATA	2	X	X					(A)—(A) ∨ DATA Exclusive-OR immediate with Accumulator.
	ORA	DATA	2	X	X					(A)—(A) ∨ DATA OR immediate with Accumulator.
	SBC	DATA	2	X	X	X	X			(A)—(A) - DATA - $\overline{C}$ Subtract immediate, with borrow, from Accumulator. Note that Carry reflects the complement of the borrow.
	CPX	DATA	2	X	X	X				(X) - DATA Compare immediate with Index Register X. Only the status flags are affected.
	CPY	DATA	2	X	X	X				(Y) - DATA Compare immediate with Index Register Y. Only the status flags are affected.
JUMP	JMP	LABEL (LABEL)	3							(PC)—LABEL or (PC)—(LABEL) Jump to new location, using extended or indirect addressing.
	JSR	LABEL	3							((SP))—(PCH). ((SP)-1)—(PCL). (SP)—(SP)-2. (PC)—LABEL Jump to subroutine beginning at address given in bytes 2 and 3 of the instruction.
BRANCH ON CONDITION	BCC	DISP	2							If C = 0, then (PC)—(PC) + 1 + DISP Branch relative if Carry flag is clear.
	BCS	DISP	2							If C = 1, then (PC)—(PC) + 1 + DISP Branch relative if Carry flag is set.
	BEQ	DISP	2							If Z = 1, then (PC)—(PC) + 1 + DISP Branch relative if result is equal to zero
	BMI	DISP	2							If S = 1, then (PC)—(PC) + 1 + DISP Branch relative if result is negative.
	BNE	DISP	2							If Z = 0, then (PC)—(PC) + 1 + DISP Branch relative if result is not zero.
	BPL	DISP	2							If S = 0, then (PC)—(PC) + 1 + DISP Branch relative if result is positive.
	BVC	DISP	2							If O = 0, then (PC)—(PC) + 1 + DISP Branch relative if Overflow flag is clear.

Table 10-2. A Summary of the MC68000 Microcomputer Instruction Set (Continued)

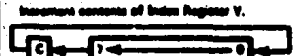
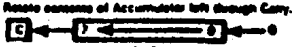
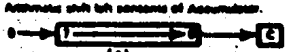
TYPE	MNEMONIC	OPERANDS	BYTES	STATUS						OPERATION PERFORMED
				S	Z	C	O			
BRANCH OR CONDITION (CONTINUED)	BVS	DISP	2							If O = 1, then (PC)←(PC) + 1 + DISP Branch relative if Overflow flag is set.
REGISTER-REGISTER MOVE	TAX		1	N	N					(A)→(X) Move Accumulator contents to Index Register X.
	TXA		1	N	N					(X)→(A) Move contents of Index Register X to Accumulator.
	TAY		1	N	N					(A)→(Y) Move Accumulator contents to Index Register Y.
	TYA		1	N	N					(Y)→(A) Move contents of Index Register Y to Accumulator.
	TSX		1	N	N					(SP)→(X) Move contents of Stack Pointer to Index Register X.
	TXS		1							(X)→(SP). Move contents of Index Register X to Stack Pointer.
REGISTER OPERATE	DEX		1	N	N					(X)→(X)-1 Decrement contents of Index Register X.
	DEY		1	N	N					(Y)→(Y)-1 Decrement contents of Index Register Y.
	INX		1	N	N					(X)→(X) + 1 Increment contents of Index Register X.
	INY		1	N	N					(Y)→(Y) + 1 Increment contents of Index Register Y.
	ROL	A	1	N	N	N				 (A) Rotate contents of Accumulator left through Carry.
	ASL	A	1	N	N	N				 (A) Arithmetic shift left contents of Accumulator.
	LSR	A	1	0	N	N				 (A) Logical shift right contents of Accumulator.
STACK	PHA		1							((SP))←(A), (SP)←(SP)-1 Push Accumulator contents onto Stack.
	PLA		1	N	N					(A)←((SP) + 1), (SP)←(SP) + 1 Load Accumulator from top of Stack (PULL).
	PHP		1							((SP))←(SR), (SP)←(SP)-1 Push Status register contents onto Stack.

Table 10-2. A Summary of the MCS6500 Microcomputer Instruction Set (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				S	Z	C	O			
STACK CONTROL	PLP		1	X	X	X	X			(SR)←[[SP] + 1], [SP]←[SP] + 1 Load Status register from top of Stack (PULL).
	RTS		1							[PCLOW]←[[SP] + 1] [PCHIGH]←[[SP] + 2], [SP]←[SP] + 2, [PC]←[PC] + 1 Return from subroutine.
INTERRUPT	CLI		1							I ← 0 Enable interrupts by clearing interrupt disable bit of Status register.
	SEI		1							I ← 1 Disable interrupts.
	RTI		1	X	X	X	X			[SR]←[[SP] + 1], [PCLOW]←[[SP] + 2], [PCHIGH]←[[SP] + 3], [SP]←[SP] + 3, [PC]←[PC] + 1 Return from interrupt; restore Status register and Program Counter from top of Stack.
	BRK		1							[[SP]]←[PCHIGH], [[SP]-1]←[PCLOW], [[SP]-2]←[SR], [SP]←[SP]-3, [PCLOW]←[FFFE], [PCHIGH]←[FFFF], I ← 1, B ← 1 Programmed interrupt. BRK cannot be disabled.
STATUS	CLC		1			0				C ← 0 Clear Carry flag.
	SEC		1			1				C ← 1 Set Carry flag.
	CLD		1							D ← 0 Clear Decimal Mode.
	SED		1							D ← 1 Set Decimal Mode.
	CLV		1			0				O ← 0 Clear Overflow flag.
	NOP		1							No Operation.

Table 9-22. A Summary of the New and Enhanced Instructions for the MC6809

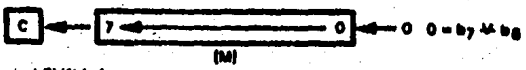
TYPE	MNEMONIC	OPERANDS	BYTES	STATUS								OPERATION PERFORMED
				E	F	C	Z	S	V	H	I	
PRIMARY MEMORY REFERENCE	LDD	ADR8	2				X	X	0			[ACA] — [MEM], [ACB] — [MEM + 1] Load double Accumulator using base page direct, extended direct, indirect or indexed addressing.
		ADR16	3									
		OFFSET,R	2+									
	STD	ADR8	2				X	X	0			[MEM] — [ACA], [MEM + 1] — [ACB] Store double Accumulator using direct, extended, indirect or indexed addressing.
		ADR16	3									
		OFFSET,R	2+									
	LDU	ADR8	2				X	X	0			[REGHI] — [MEM], [REGLO] — [MEM + 1] Load specified register (U or Y) using direct, extended, indirect or indexed addressing. Sign status reflects REG bit 15.
		ADR16	3									
		OFFSET,R	2+									
	LDY	ADR8	3				X	X	0			
		ADR16	4									
		OFFSET,R	3+									
	STU	ADR8	2				X	X	0			[MEM] — [REGHI], [MEM + 1] — [REGLO] Store contents of specified register (U or Y) using direct, extended, indirect or indexed addressing. Sign status reflects REG bit 15.
		ADR16	3									
		OFFSET,R	2+									
	STY	ADR8	3				X	X	0			
		ADR16	4									
		OFFSET,R	3+									
SECONDARY MEMORY REFERENCE (MEMORY OPERATE)	ADDD	ADR8	2			X	X	X	X	X		[ACD] — [ACD] + [MEM]: [MEM + 1] Add 16-bit value from locations MEM and MEM + 1 to D Accumulator using direct, extended, indirect or indexed addressing.
		ADR16	3									
		OFFSET, R	2+									
	CMPD	ADR8	3			X	X	X	X			[ACD] – [MEM]: [MEM + 1] Compares 16-bit number from locations M and M + 1 with contents of D Accumulator and sets status bits as appropriate. Only Status register is affected.
		ADR16	4									
		OFFSET,R	3+									
	CMPS	ADR8	3			X	X	X	X			[REG] – [MEM]: [MEM + 1] Compares 16-bit number from locations M and M + 1 with contents of register (S, U, Y or X) specified in the mnemonic and sets status bits as appropriate. Only Status register is affected.
	CMPU	ADR16	4									
	CMPY	OFFSET,R	3+									
	CMPX	ADR8	2			X	X	X	X			
		ADR16	3									
		OFFSET,R	2+									
	LSL	ADR8	2			X	X	X	X			 Logical Shift Left.
		ADR16	3									
		OFFSET,R	2+									
	SUBD	ADR8	2			X	X	X	X			[ACD] — [ACD] - [MEM]: [MEM + 1] Subtract 16-bit number contained in locations MEM and MEM + 1 from number contained in D Accumulator using direct, extended, indirect or indexed addressing.
		ADR16	3									
		OFFSET,R	2+									

Table 9-22. A Summary of the New and Enhanced Instructions for the MC6809 (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS								OPERATION PERFORMED
				E	F	C	Z	S	V	H	I	
IMMEDIATE	LDD	DATA16	3				X	X	0			(ACA) — (82), (ACB) — (83) Load Accumulator Immediate. Sign status reflects bit 15
	LDU	DATA16	3				X	X	0			(UHI) — (82), (ULO) — (83) Load User Pointer Immediate.
	LDY	DATA16	4				X	X	0			(YHI) — (83), (YLO) — (84)
IMMEDIATE OPERATE	ADDD	DATA16	3			X	X	X	X	X		(ACD) — (ACD) + (82): (83) Add 16-bit number following Op-code to contents of D Accumulator.
	SUBD	DATA16	3			X	X	X	X			(ACD) — (ACD) - (83): (83) Subtract 16-bit number following Op-Code from contents of D Accumulator.
	CMPD	DATA16	4			X	X	X	X			(ACD) - (83): (84) Compare immediate contents of D Accumulator and 16-bit number following three byte) Op-code. Only status bits are affected.
	CMPS CMPU CMPY	DATA16	4			X	X	X	X			(REG) - (83): (84) Compare immediate contents of designated Register (S, U, Y or X) specified in instruction with 16-bit number following three byte) Op-code. Only Status bits are affected.
	CMPX	DATA16	3			X	X	X	X			
JUMP	LBRA	DISP16	3									(PC) — (PC') + DISP16 Unconditional long branch relative to present Program Counter contents.
	LBSR	DISP16	3									(ISP) - 1) — (PCLO), ((SP) - 2) — (PCHI), (SP) — (SP) - 2 (PC) — (PC') + DISP16 Unconditional long branch to subroutine located relative to present Program Counter contents.
BRANCH ON CONDITION	BHS	DISP	2									(PC) — (PC') + DISP if condition true C = 0
	BLO	DISP	2									C = 1 (PC) — (PC') + DISP16 if condition true

Table B-22. A Summary of the New and Enhanced Instructions for the MC6809 (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS								OPERATION PERFORMED
				E	F	C	Z	S	V	H	I	
BRANCH ON CONDITION (Continued)	LBCC	DISP16	4									Conditions are the same as shown in the Branch On Condition Table for the MC6800.
	LBCS	DISP16	4									
	LBEQ	DISP16	4									
	LBGE	DISP16	4									
	LBGT	DISP16	4									
	LBHI	DISP16	4									
	LBHS	DISP16	4									
	LBLE	DISP16	4									
	LBLO	DISP16	4									
	LBLS	DISP16	4									
	LBLT	DISP16	4									
	LBMI	DISP16	4									
	LBNE	DISP16	4									
	LBPL	DISP16	4									
	LBVC	DISP16	4									
	LBVS	DISP16	4									
REGISTER TO REGISTER MOVE	EXG	R1, R2	2									(R1) — — (R2) Exchange contents of specified registers. Status register not affected unless R1 or R2 is Status register.
	TFR	R1, R2	2									(R2) — (R1) Transfer contents of R1 to R2. Status register is not affected unless R2 is Status register.
REGISTER-REGISTER OPERATE	ABX		1									(X) — (X) + (B) Add unsigned contents of B Accumulator to Index register.
	MUL		1				X					(D) — (A) x (B) Multiply unsigned numbers in Accumulators A and B and place result in D. Carry bit is set if Accumulator B bit 7 is set.
	SEX		1				X	X	0			(A) — FF_{16} if Accumulator B bit 7 = 1 (A) — 00_{16} if Accumulator B bit 7 = 0 Transform an 8-bit twos complement number in B to a 16-bit twos complement number in D.

Table 9-22. A Summary of the New and Enhanced Instructions for the MC6809 (Continued)

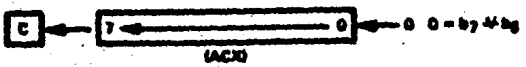
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS E	F	C	Z	S	V	H	I	OPERATION PERFORMED
	LEAS	OFFSET,R	2+									(S) — EA
	LEAU	OFFSET,R	2+									(U) — EA
	LEAX	OFFSET,R	2+				X					(X) — EA
	LEAY	OFFSET,R	2+				X					(Y) — EA EA is the Effective Address Form the Effective Address EA according to the addressing variation used. Load this address into designated register (for later use) rather than outputting it on Address Bus at this time.
	LSL	ACX	1			X	X	X	X			
STACK	PSHS	LIST	2									Test Post Byte and stack as follows. Condition: b7 = 1; (SP) — (SP) - 1, ((SP)) — (PCLO) (SP) — (SP) - 1, ((SP)) — (PCHI) b6 = 1; (SP) — (SP) - 1, ((SP)) — (ULO) (SP) — (SP) - 1, ((SP)) —(UHI) b5 = 1; (SP) — (SP) - 1, ((SP)) — (YLO) (SP) — (SP) - 1, ((SP)) — (YHI) b4 = 1; (SP) — (SP) - 1, ((SP)) — (XLO) (SP) — (SP) - 1, ((SP)) — (XHI) b3 = 1; (SP) — (SP) - 1, ((SP)) — (DP) b2 = 1; (SP) — (SP) - 1, ((SP)) — (B) b1 = 1; (SP) — (SP) - 1, ((SP)) — (A) b0 = 1; (SP) — (SP) - 1, ((SP)) — (CR) Push any, all, none or any subset of registers onto Hardware Stack (except the Hardware Stack Pointer itself).

Table 9-22. A Summary of the New and Enhanced Instructions for the MC6809 (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS E	F	C	Z	S	V	H	I	OPERATION PERFORMED
STACK (Continued)	PSHU	LIST	2									Test Post Byte and stack as follows. Condition: b7 = 1; (U) — (U) - 1, [(U)] — (PCLO) (U) — (U) - 1, [(U)] — (PCHI) b6 = 1; (U) — (U) - 1, [(U)] — (SFLO) (U) — (U) - 1, [(U)] — (SFHI) b5 = 1; (U) — (U) - 1, [(U)] — (YLO) (U) — (U) - 1, [(U)] — (YHI) b4 = 1; (U) — (U) - 1, [(U)] — (XLO) (U) — (U) - 1, [(U)] — (XHI) b3 = 1; (U) — (U) - 1, [(U)] — (DP) b2 = 1; (U) — (U) - 1, [(U)] — (B) b1 = 1; (U) — (U) - 1, [(U)] — (A) b0 = 1; (U) — (U) - 1, [(U)] — (SR) Push any, all, none or any subset of registers onto User Stack (except the User Stack Pointer itself).
	PULS	LIST	2									Test Post Byte and unstack as follows. Condition: b0 = 1; (SR) — [(SP)], (SP) — (SP) + 1 b1 = 1; (A) — [(SP)], (SP) — (SP) + 1 b2 = 1; (B) — [(SP)], (SP) — (SP) + 1 b3 = 1; (DP) — [(SP)], (SP) — (SP) + 1 b4 = 1; (XHI) — [(SP)], (SP) — (SP) + 1 (XLO) — [(SP)], (SP) — (SP) + 1 b5 = 1; (YHI) — [(SP)], (SP) — (SP) + 1 (YLO) — [(SP)], (SP) — (SP) + 1 b6 = 1; (UHI) — [(SP)], (SP) — (SP) + 1 (ULO) — [(SP)], (SP) — (SP) + 1 b7 = 1; (PCHI) — [(SP)], (SP) — (SP) + 1 (PCLO) — [(SP)], (SP) — (SP) + 1 Pull any, all, none or any subset of registers from Hardware Stack (except the Hardware Stack Pointer itself). The Status register bits are determined by byte pulled from Stack.

Table 9-22. A Summary of the New and Enhanced Instructions for the MC6809 (Continued)

TYPE	MNEMONIC	OPERANDS	BYTES	STATUS								OPERATION PERFORMED
				E	F	C	Z	S	V	H	I	
STACK (Continued)	PULU	LIST	2									Test Post Byte and unstack as follows: Condition: b0 = 1; (CC) ← ((U)), (U) ← (U) + 1 b1 = 1; (A) ← ((U)), (U) ← (U) + 1 b2 = 1; (B) ← ((U)), (U) ← (U) + 1 b3 = 1; (DP) ← ((U)), (U) ← (U) + 1 b4 = 1; (XHI) ← ((U)), (U) ← (U) + 1 (XLO) ← ((U)), (U) ← (U) + 1 b5 = 1; (YHI) ← ((U)), (U) ← (U) + 1 (YLO) ← ((U)), (U) ← (U) + 1 b6 = 1; (SPHI) ← ((U)), (U) ← (U) + 1 (SPLO) ← ((U)), (U) ← (U) + 1 b7 = 1; (PCHI) ← ((U)), (U) ← (U) + 1 (PCLO) ← ((U)), (U) ← (U) + 1 Pull any, all, none or any subset of registers from User Stack (except the User Stack Pointer itself). Status register bits are determined by byte pulled from Stack.
INTERRUPT	RTI		1									Pull registers from Hardware Stack in accordance with value of E of Status Register. If E = 0, pull the subset. (CC) ← ((SP)), (SP) ← (SP) + 1 (PCHI) ← ((SP)), (SP) ← (SP) + 1 (PCLO) ← ((SP)), (SP) ← (SP) + 1 If E = 1, pull the full complement. (CC) ← ((SP)), (SP) ← (SP) + 1 (A) ← ((SP)), (SP) ← (SP) + 1 (B) ← ((SP)), (SP) ← (SP) + 1 (DP) ← ((SP)), (SP) ← (SP) + 1 (XHI) ← ((SP)), (SP) ← (SP) + 1 (XLO) ← ((SP)), (SP) ← (SP) + 1 (YHI) ← ((SP)), (SP) ← (SP) + 1 (YLO) ← ((SP)), (SP) ← (SP) + 1 (UHI) ← ((SP)), (SP) ← (SP) + 1 (ULO) ← ((SP)), (SP) ← (SP) + 1 (PCHI) ← ((SP)), (SP) ← (SP) + 1 (PCLO) ← ((SP)), (SP) ← (SP) + 1 Status bits are as received from Stack.

Table 9-22. A Summary of the New and Enhanced Instructions for the MC6809 (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS								OPERATION PERFORMED
				E	F	C	Z	S	V	H	I	
INTERRUPT (Continued)	CWAI		2									(SR) — (SR) ∧ (82) This may clear SR bits. E — 1 (SP) — (SP) - 1, [(SP)] — (PC) (SP) — (SP) - 1, [(SP)] — (PC) (SP) — (SP) - 1, [(SP)] — (U) (SP) — (SP) - 1, [(SP)] — (U) (SP) — (SP) - 1, [(SP)] — (Y) (SP) — (SP) - 1, [(SP)] — (Y) (SP) — (SP) - 1, [(SP)] — (X) (SP) — (SP) - 1, [(SP)] — (X) (SP) — (SP) - 1, [(SP)] — (DP) (SP) — (SP) - 1, [(SP)] — (B) (SP) — (SP) - 1, [(SP)] — (A) (SP) — (SP) - 1, [(SP)] — (SR) Pushes registers onto Stack and waits for an interrupt. When non-masked interrupt occurs, vectors to corresponding interrupt service routine. FIRQ enters its service routine with all registers saved, but since E = 1, they will unstack correctly on RTI. (System busses are not floated by CWAI.)

Table 9-22. A Summary of the New and Enhanced Instructions for the MC6809 (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUS E	F	C	Z	S	V	H	I	OPERATION PERFORMED
INTERRUPT (Continued)	SWI		1									E — 1 (SP) — (SP) - 1, [(SP)] — (PCLO) (SP) — (SP) - 1, [(SP)] — (PCHI) (SP) — (SP) - 1, [(SP)] — (ULO) (SP) — (SP) - 1, [(SP)] — (UHI) (SP) — (SP) - 1, [(SP)] — (YLO) (SP) — (SP) - 1, [(SP)] — (YHI) (SP) — (SP) - 1, [(SP)] — (XLO) (SP) — (SP) - 1, [(SP)] — (XHI) (SP) — (SP) - 1, [(SP)] — (DP) (SP) — (SP) - 1, [(SP)] — (B) (SP) — (SP) - 1, [(SP)] — (A) (SP) — (SP) - 1, [(SP)] — (SR) I — 1, F — 1, [PC] — [FFFA]: [FFF8] Transfer control to interrupt subroutine.
	SWI2		2									E — 1 Push registers onto Hardware Stack (same as above). [PC] — [FFF4]: [FFF5] Transfer control to interrupt subroutine.
	SWI3		2									E — 1 Push registers onto Hardware Stack (same as above). [PC] — [FFF2]: [FFF3] Transfer control to interrupt subroutine.
	SYNC		1									Stop processing instructions: Float system busses: wait for an interrupt. When an interrupt occurs, resume processing as follows: i) If interrupt is enabled, transfer to the service routine. ii) If interrupt is disabled, continue execution at next instruction in sequence.
STATUS	ANDCC	DATA	2			↕						[SR] — [SR] ∧ DATA AND immediate. Used to clear SR bits.
	ORCC	DATA	2									[SR] — [SR] ∨ DATA OR immediate. Used to set SR bits.
	BRN	DISP	2									Branch Never. This is a No Operation
	LBN	DISP16	4									Long Branch Never. This is a No Operation.

APENDICE B

RESPUESTA DE LOS TERMOPARES

THERMOCOUPLE RESPONSES

TABLE 7-1. VOLTAGE AS A FUNCTION OF TEMPERATURE

	B		E		J		K	
Temperature °C	Output mV	Tempco µV/°C	Output mV	Tempco µV/°C	Output mV	Tempco µV/°C	Output mV	
- 300	–	–	- 9.834	38.1	- 7.890	21.9	- 6.081	
- 100	–	–	- 5.237	48.1	- 4.632	41.1	- 3.553	
0	+ 0.000	–	0.000	58.7	0.000	50.4	0.000	
+ 25	- 0.002	0.1	1.496	60.9	1.277	51.7	1.000	
+ 100	+ 0.033	0.9	6.317	67.5	5.269	54.4	4.096	
+ 200	+ 0.178	2.0	13.419	74.0	10.777	56.5	8.137	
+ 300	0.431	3.05	21.033	77.9	16.325	56.4	12.207	
+ 400	0.786	3.86	28.943	80.0	21.846	56.2	16.395	
+ 500	1.241	5.0	36.998	80.9	27.388	56.0	20.640	
+ 600	1.791	5.95	45.085	80.7	33.096	56.5	24.902	
+ 700	2.430	6.8	53.110	79.8	38.130	62.3	29.128	
+ 800	3.154	7.66	61.022	78.4	46.496	64.8	33.277	
+ 900	3.957	8.4	68.783	76.7	51.875	62.4	37.325	
+1000	4.833	9.1	76.358	75.0	57.942	60.2	41.269	
+1100	5.777	9.76	–		63.777	57.8	45.108	
+1200	6.783	10.35	–		69.536	57.2	48.828	
+1300	7.846	10.8	–		–		52.398	
+1400	8.962	11.2	–		–		–	
+1500	10.094	11.6	–		–		–	
+1600	11.257	11.7						
+1700	12.426	11.7						
+1800	13.586	11.6						

TABLE 7-2. TEMPERATURE AS A FUNCTION OF VOLTAGE READING

	B		E		J		K
mV	°C	°C/mV	°C	°C/mV	°C	°C/mV	°C
-10.000	–		–		–		–
- 9.000	–		- 94.4	21.70	- 100.1	26.16	- 153.7
- 2.000	–		- 36.3	18.40	- 40.8	21.10	- 63.1
- 1.000	–		- 17.3	17.80	- 20.1	20.40	- 26.9
0.000	+ 42.0	4°/µV	0.0	17.08	0.0	19.84	0.0
+ 1.000	449.9	220	16.8	16.64	19.6	19.26	+ 25.0
+ 2.000	634.2	160	33.2	16.21	38.9	19.09	+ 49.6
+ 5.000	1018.2	109	80.3	15.19	96.1	18.43	+ 122.0
+10.000	1491.8	87	163.0	14.02	186.8	18.03	246.3
+20.000	–		296.7	12.99	366.8	18.13	486.0
+30.000	–		413.2	12.47	546.3	17.57	720.8
+40.000	–		537.1	12.38	713.9	15.94	967.5
+50.000	–		661.1	12.47	870.2	16.78	1222.3
+60.000	–		787.0	12.71	1035.0	18.98	
+70.000	–		916.9	13.09	–		

	R		S		T		
Tempco µV/°C	Output mV	Tempco µV/°C	Output mV	Tempco µV/°C	Output mV	Tempco µV/°C	Temperature °C
10.2	–		–		- 5.603	16.6	- 200
30.5	–		–		- 3.378	28.4	- 100
38.5	0.000	5.29	0.000	5.4	0.000	38.8	0
40.5	0.141	8.0	0.142	8.8	0.992	40.7	+ 25
41.4	0.647	7.6	0.646	7.3	4.277	46.8	+ 100
39.9	1.468	8.86	1.440	8.46	8.288	53.2	+ 200
41.5	2.400	9.75	2.322	9.1	14.860	58.1	+ 300
41.8	3.407	10.35	3.260	9.5	20.869	61.9	+ 400
42.6	4.471	10.9	4.234	9.9	–		+ 500
42.5	5.582	11.3	5.237	10.15	–		+ 600
41.9	6.741	11.8	6.274	10.58	–		+ 700
41.0	7.940	12.3	7.345	10.9	–		+ 800
39.9	9.360	12.7	8.448	11.2	–		+ 900
38.9	10.993	13.2	9.586	11.5	–		+1000
37.6	11.849	13.6	10.754	11.9			+1100
36.5	13.224	13.9	11.947	12.0			+1200
34.9	14.624	14.0	13.156	12.2			+1300
	16.035	14.1	14.388	12.2			+1400
	17.449	14.1	15.576	12.1			+1500
	18.842	13.9	16.771	11.8			+1600
	20.215	13.5	17.942	11.5			+1700
	–		–				+1800

	R		S		T		
°C/mV	°C	°C/mV	°C	°C/mV	°C	°C/mV	mV
	–		–		–		-10.000
43.40	–		–		-166.5	49.8	- 5.000
28.13	–		–		- 66.1	30.0	- 2.000
26.49	–		–		- 26.8	27.6	- 1.000
26.35	0.0	180.6	0.0	188.2	0.0	25.8	0.000
24.66	+ 146.8	122.7	+ 148.4	126.8	+ 26.2	24.8	+ 1.000
24.27	268.2	108.4	264.3	111.7	49.2	23.4	+ 2.000
24.43	648.2	90.1	670.8	88.8	115.3	20.8	+ 5.000
24.60	961.7	78.8	1038.8	88.3	213.3	18.8	+10.000
23.90	1884.1	73.8	–		366.9	16.3	+20.000
23.96	–		–		–		+30.000
28.46	–		–		–		+40.000
27.78	–		–		–		+50.000
							+60.000
							+70.000

Elemento	Deriva	Span mínimo	Precisión	Repetibilidad	Temp. máx. C	Dist. máx. al receptor	Linealidad
Resistencia de níquel	0.3 C año	<11 C	0.5 C	0.05 C	300	<300 m	[illegible]
platino	<0.05 C año	<3 C	0.01 C	0.03 C	950	<1 500 m	[illegible]
cobre		-	0.1 C	-	120	-	Buena
Termistor	sin envejecer 1 C año envejecido <0.05 C año	<1 C	0.005	0.05–0.11 C	400	<1 500 m	Pobre
Termopar de cobre-constantan (T)	<0.5 C año a>11 C año [illegible]		2.0–2%, (0.4–0.8 C)	~0.11 C	370	<1 500 m [illegible]	Buena
hierro-constantan (J)	[illegible]		1.3–0.8%,(1.1–2.2 C)	↓	800	[illegible]	Buena
cromel-alumel (K)	[illegible]		2.4%,(±3 C)	↓	1 100	[illegible]	Muy buena
Pt-Pt Rd (R y S)			1–3 C	↓	1 600	[illegible]	Buena a alta temperatura
Radiación óptico	–	900 C	1–3 C	Muy buena	5 000	↓	Pobre varía con la cuarta potencia de la temperatura
total	–	220 C	±0.5%	↓	5 000	↓	
relación	–	400 C	±0.5%	↓	↓	↓	
fotoeléctrico	–	110 C	±0.5%	±0.25%	↓	↓	

Elemento	Ambiente de trabajo	Desventajas	Ventajas
Resistencia de níquel	[illegible]	Bajo rango de temperatura, frágil	Buena [illegible] Span estrecho
platino	↓	Más caro que el termopar o termistor, frágil	[illegible]
cobre	↓	[illegible]	[illegible]
Termistor	[illegible]	[illegible]	[illegible]
Termopar de cobre-constantan (T)	[illegible]	Span amplio [illegible]; Baja temperatura máxima	Pequeño tamaño, Respuesta rápida; [illegible]
hierro-constantan (J)	[illegible]	Más caro que T o J	[illegible]
cromel-alumel (K)	[illegible]	Más caro que K	[illegible]
Pt-Pt Rd (R y S)	[illegible]	[illegible]	[illegible]
Radiación óptico, total, relación, fotoeléctrico	[illegible]	Pobre linealidad; Más caro que el termopar; Difícil determinar temp. exacta [illegible]; Caro	Sin contacto con el material; Buena repetibilidad; Respuesta rápida [illegible]; Respuesta rápida

Tabla IX. Características de los transductores eléctricos de temperatura

A P E N D I C E C

ESPECIFICACIONES DE LOS CONVERTIDORES

DAC-08 Y MC1408

A

Manufacturers' Data Sheets

This appendix contains catalog specification sheets for a selected group of IC d/a and a/d devices from those discussed in this book. Because of the broad range of devices being covered, it is not feasible to provide specification data for all of them within this section. The devices which are presented here are chosen as being representative of each device type and category, as evidenced by strong industry popularity and/or multiple source availability. The data sheets reproduced here are those of the original manufacturers.

Due to space limitations, it is not possible to reproduce these data sheets in their entirety, therefore presentation here is confined to the essential "electrical characteristics" information, for devices performing in a standard commercial environment (0-70°C). It is suggested that the reader avail himself/herself of more complete and most recent data, by writing directly to the manufacturer in question.

PMI

DAC-08

8 BIT HIGH SPEED MULTIPLYING D/A CONVERTER

UNIVERSAL DIGITAL LOGIC INTERFACE

GENERAL DESCRIPTION

The DAC-08 series of 8 bit monolithic multiplying Digital to Analog Converters provide very high speed performance coupled with low cost and outstanding applications flexibility. [illegible]

[illegible]

[illegible]

[illegible]

FEATURES

- Fast Settling Output Current [illegible]
- Full Scale Current Prematched to ±1 LSB
- Direct Interface to TTL, CMOS, ECL, HTL, PMOS
- [illegible]
- High Output Impedance and Compliance . . . -10V to +18V
- Differential Current Outputs
- Wide Range Multiplying Capability . . . . 1 MHz Bandwidth
- Low FS Current Drift [illegible]
- Wide Power Supply Range . . . . . ±4.5V to ±18V
- Low Power Consumption [illegible]
- Low Cost

[illegible] CRT display drivers, high speed modems and other applications where low cost, high speed and complete [illegible] are required.

EQUIVALENT CIRCUIT

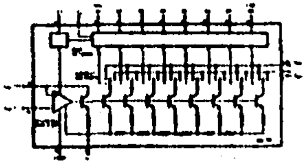
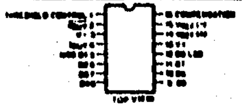


FIGURE 1

ORDERING INFORMATION AND PIN CONNECTION



TOP VIEW

16 PIN HERMETIC DUAL IN LINE [illegible]

MODEL	TEMP RANGE	NONLINEARITY
DAC-08A[illegible]	-55°/+125°C	±0.1%
DAC-08[illegible]	-55°/+125°C	±0.19%
DAC-08E[illegible]	0°/70°C	±0.1%
DAC-08[illegible]	0°/70°C	±0.19%
DAC-08C[illegible]	0°/70°C	±0.39%

[illegible]

The premium performance of this product is achieved through an advanced processing technology. All Precision Monolithics products are [illegible]

TECHNICAL SPECIFICATIONS DECEMBER 197[illegible]

[illegible] PATENTS APPLIED FOR

Courtesy Precision Monolithics, Inc.

ELECTRICAL CHARACTERISTICS

Parameter	Symbol	Conditions	DAC-08A Min	DAC-08A Typ	DAC-08A Max	DAC-08E Min	DAC-08E Typ	DAC-08E Max	DAC-08H Min	DAC-08H Typ	DAC-08H Max	Units
Resolution			8	8	8	8	8	8	8	8	8	Bits
Monotonicity			8	8	8	8	8	8	8	8	8	Bits
Conversion Time	t_c	T_A = 0°C to 70°C	-	-	85	-	-	85	-	-	85	ns
		90 to 10% of FS change, T_A = 25°C	-	85	135	-	85	135	-	85	135	ns
Propagation Delay												
Each bit	t_{PLH}	T_A = 25°C	-	35	60	-	35	60	-	35	60	ns
All bits switched	t_{PHL}		-	35	60	-	35	60	-	35	60	ns
Full Scale Tempco	TC$_{FS}$		-	±10	±50	-	±10	±50	-	±10	±50	ppm/°C
Output Voltage Compliance	V_{OC}	Full scale current change < 0.5 LSB Typical > 30 Megohm min.	-10	-	+10	-10	-	+10	-10	-	+10	Volts
Full Scale Current	I_{FS}	Vcomp = 10.000V R14, R15 = 5.000Ω T_A = 25°C	1.984	1.992	2.000	1.94	1.99	2.04	1.96	1.99	2.04	mA
Full Scale Symmetry	I_{FSS}	I_{FS} – I_{FSS}	-	±0.5	±4.0	-	±1.0	±8.0	-	±0.5	±4.0	µA
Zero Scale Current	I_{ZS}		-	0.1	0.5	-	0.3	2.0	-	0.3	2.0	µA
Output Current Range	I_{FS}	V_+ = 4.5V	0	2.0	2.1	0	2.0	2.1	0	2.0	2.1	mA
		V_+ = +7.5V to +18V	0	2.0	4.2	0	2.0	4.2	0	2.0	4.2	mA
Logic Input Levels												
Logic "0"	V_{IL}	V_{LC} = 0V	-	-	0.8	-	-	0.8	-	-	0.8	Volts
Logic "1"	V_{IH}		2.0	-	-	2.0	-	-	2.0	-	-	Volts
Logic Input Current		V_{LC} = 0V										
Logic "0"	I_{IL}	V_{IN} = -10V to +0.8V	-	-0.2	-4.0	-	-0.2	-4.0	-	-0.2	-4.0	µA
Logic "1"	I_{IH}	V_{IN} = 2.0V to 18V	-	0.002	10	-	0.002	10	-	0.002	10	µA
Logic Input Swing	V_{IN}	V_- = -15V	-10	-	+10	-10	-	+10	-10	-	+10	Volts
Logic Threshold Voltage	V_{THR}	V_- = -15V	-10	-	+13.5	-10	-	+13.5	-10	-	+13.5	Volts
Reference Bias Current	I_{15}		-	-1.0	-3.0	-	-1.0	-3.0	-	-1.0	-3.0	µA
Reference Input Slew Rate	dI/dt	See Fig. 5, 10	4.0	8.0	-	4.0	8.0	-	4.0	8.0	-	mA/µs
Power Supply Sensitivity	PSS$_{(+)}$	V_+ = 4.5V to 18V	-	±0.0003	±0.01	-	±0.0003	±0.01	-	±0.0003	±0.01	%/%
	PSS$_{(-)}$	V_- = -4.5V to -18V, Input = 1.0 mA	-	±0.0003	±0.01	-	±0.002	±0.01	-	±0.0003	±0.01	%/%
Power Supply Current	I_+	V_S = +15V, Input = 2.0 mA	-	2.3	3.8	-	2.3	3.8	-	2.3	3.8	mA
	I_-		-	-4.3	-6.8	-	-4.3	-6.8	-	-4.3	-6.8	mA
	I_+	V_S = +5V, -15V, Input = 2.0 mA	-	2.3	3.8	-	2.3	3.8	-	2.3	3.8	mA
	I_-		-	-4.3	-7.8	-	-4.3	-7.8	-	-4.3	-7.8	mA
	I_+	V_S = +15V, Input = 2.0 mA	-	2.3	3.8	-	2.3	3.8	-	2.3	3.8	mA
	I_-		-	-4.3	-7.8	-	-4.3	-7.8	-	-4.3	-7.8	mA
Power Dissipation	P_D	+5V, Input = 1.0 mA	-	33	60	-	33	60	-	33	60	mW
		+5V, -15V, Input = 2.0 mA	-	108	180	-	108	180	-	108	180	mW
		+15V, Input = 2.0 mA	-	135	210	-	135	210	-	135	210	mW

Courtesy Precision Monolithics, Inc.

ORDERING INFORMATION

Device	Temperature Range	Package
MC1408L8	0°C to +70°C	Ceramic DIP
MC1408L7	0°C to +70°C	Ceramic DIP
MC1408L6	0°C to +70°C	Ceramic DIP
MC1408P8	0°C to +70°C	Plastic DIP
MC1408P7	0°C to +70°C	Plastic DIP
MC1408P6	0°C to +70°C	Plastic DIP
MC1508L8	-55°C to +125°C	Ceramic DIP

MC1408
MC1508

Specifications and Applications Information

EIGHT-BIT MULTIPLYING DIGITAL-TO-ANALOG CONVERTER

SILICON MONOLITHIC INTEGRATED CIRCUIT

EIGHT-BIT MULTIPLYING DIGITAL-TO-ANALOG CONVERTER

designed for use where the output current is a linear product of an eight bit digital word and an analog input voltage

- Eight-Bit Accuracy Available in Both Temperature Ranges
  - Relative Accuracy: ±0.19% Error maximum
  - MC1408L8, MC1408P8, MC1508L8
- Seven and Six-Bit Accuracy Available with MC1408 Designated for 7 or 6 Bits plus Package Suffix
- Fast Settling Time – 300 ns typical
- Noninverting Digital Inputs are TTL and CMOS Compatible
- Output Voltage Swing – +0.4 V to -5.0 V
- High Speed Multiplying Input
  - Slew Rate 4.0 mA/µs
- Standard Supply Voltages: +5.0 V and -5.0 V to -15 V

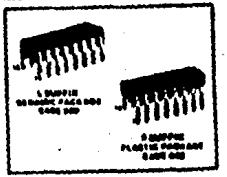


FIGURE 1 – 8-BIT D/A TRANSFER CHARACTERISTIC

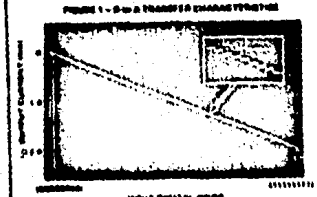
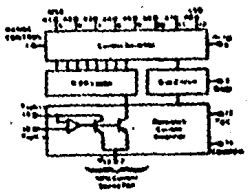


FIGURE 2 – 8-BIT D/A DIAGRAM



TYPICAL APPLICATIONS

- Programmable Gain Amplifiers
- Analog Multiplication
- Digitally Controlled Filters
- Audio Digitizing and Decoding
- Programmable Power Supplies
- Analog Signal Manipulation
- Digital Signal Multiplexers
- Analog Signal Division
- Digital Addition and Subtraction
- Speech Compression and Expansion
- Digitally Tuned Amps

Courtesy Motorola Semiconductor Products, Inc.

MAXIMUM RATINGS [illegible]

Rating	Symbol	Value	Unit
[illegible]	[illegible]	[illegible]	[illegible]

ELECTRICAL CHARACTERISTICS [illegible]

Characteristic	Figure	Symbol	Min	Typ	Max	Unit
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

MOTOROLA Semiconductor Products Inc.

Courtesy Motorola Semiconductor Products, Inc.

OPERATING TEMPERATURE RANGE:
C SUFFIX 0 to +70°C

A/D CONVERTERS

PRELIMINARY

LD110/LD111A HIGH PERFORMANCE 3½ DIGIT A/D CONVERTER SET

LD110 DIGITAL A/D PROCESSOR / LD111A ANALOG A/D PROCESSOR

Features

- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]
- Auto Zero
- Auto Polarity
- [illegible]
- Differential Input Capability
- LD111A Directly Replaces LD111
- [illegible]
- [illegible]
- TTL Compatible Outputs

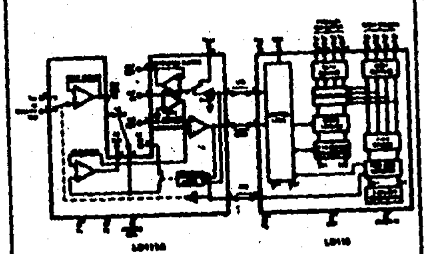
Applications

- [illegible]
- A/D Converters
- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]

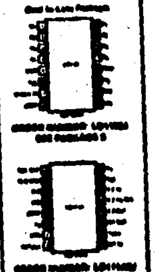
Description

[illegible]

FUNCTIONAL DIAGRAM



PIN CONFIGURATIONS



Courtesy Siliconix, Inc.

A P E N D I C E D

PROGRAMAS DE COMUNICACION HP-3000-SwTPc

CONTENIDO

INTRODUCCION

Estos programas fueron realizados con el objetivo de demostrar o facilmente en un lenguaje como es "C", el problema de unicacion entre dos computadoras, se reduce a simples programas uelos, a diferencia de programar en lenguaje ensamblador; por otra te, de la necesidad de compartir el gran potencial de acenamiento de los dos equipos, por un lado, y por el otro, de liar el numero de terminales de la computadora HP3000.

Estos programas no pretenden sustituir las terminales de HP3000, que no cuentan con el software necesario para implementar "VIEW" quete de diselo de formas en pantalla para captura de datos).

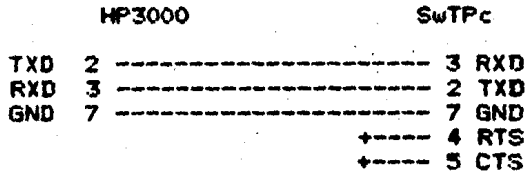
Estan orientados a aquellos usuarios que pretendan respaldar hivos en ambas maquinas, y que cuenten con los minimos ocimientos de los sistemas operativos MPE y UniFlex.

Este paquete fue implementado en el lenguaje "C", y por ende solo de ser ejecutado desde una terminal SwTPc.

TEORIA TECNICA

1.- Coneccion

La coneccion de los computadores se realiza a traves de sus puertos seriales, con un cable el cual tiene la siguiente configuracion:



de tal forma que toman poder del puerto serial en SwTPc haciendo totalmente transparente al sistema MPE la coneccion.

Es recomendable que el puerto serie utilizado en SwTPc, este localizado en cualquiera de los "slots" cero al cinco, debido a que UniFlex solo maneja doce terminales, y el "slot" siete lo ocupa para las impresoras seriales.

Con lo anterior, evitamos dejar inutilizadas terminales de SwTPc, aprovechando al maximo los dos equipos.

Estos programas no se limitan a usar solo una terminal SwTPC como terminal de HP3000, siempre y cuando tengamos mas puertos disponibles en ambas maquinas y por supuesto mas cables de coneccion entre ellas. Es decir, si tenemos solo un cable de coneccion, nadamas los podremos usar en una terminal SwTPc a un mismo tiempo.

Otra recomendacion para el usuario SwTPc, es deshabilitar el puerto serie que se use de enlace con HP3000 de la siguiente manera:

1.- Editar el archivo /etc/ttytlist, en uniusuario.
2.- Cambiar el signo "+" por un signo "-", en el(los) puerto(s) correspondiente(s).
3.- Entrar a multiusuario.

INSTALACION DEL PAQUETE DE COMUNICACION

Este paquete de comunicacion consta por lo pronto de cuatro programas, a saber:

- term3000
- list3000
- hp_a_sw
- sw_a_hp

los cuales deberan residir en /bin, por lo tanto:

1.- Montar el floppy disk en el drive 1

2.- Teclear:

```
++ /etc/mount /dev/fd1 usr2
++ copy /usr2/*3000 /bin
++ copy /usr2/*_a_* /bin
++ /etc/unmount /dev/fd1
```

MANEJO DEL PAQUETE DE COMUNICACION

Como vimos anteriormente, si contamos con un solo cable de coneccion entre las computadoras, podremos utilizar el paquete de comunicacion en una terminal de SwTPc. Si contamos con dos cables de coneccion, entonces los podremos usar en dos terminales a un mismo tiempo, etc.

El paquete de comunicacion puede ser usado por cualquier usuario y en cualquier terminal en SwTPc, no importando la cuenta de acceso a Uniflex.

term3000

Sintaxis: ++ term3000

Uso: term3000 permite usar una terminal SwTPc como terminal de HP3000.

Salida: para abandonar la sesion de term3000, previamente terminar la sesion de HP3000 de la siguiente manera:
:BYE
posteriormente presionar la tecla break, que esta localizada en la parte superior derecha del teclado. Otra manera de abandonar esta sesion es presionando al mismo tiempo las teclas de CTRL y C.

Notas: No es permitido utilizar el paquete "VIEW" de HP3000. Todos los procesos que se realizen residiran en HP3000.

list3000

Sintaxis: ++list3000 <arch. en swtpc> <cuenta en hp3000>

Uso: list3000 permite transferir un archivo de SwTPc directamente a la impresora del equipo HP3000.
El argumento <arch. en swtpc> es el nombre del archivo que reside en SwTPc, y que puede pertenecer al usuario en turno o a otra cuenta, para lo cual se debe dar la ruta del archivo, es decir, si queremos enviar el archivo "sale" que pertenece a la cuenta /usr/jorge teclear:
++list3000 /usr/jorge/sale <cuenta en hp3000>
otro ejemplo:
++list3000 miarchivo <cuenta en hp3000>
El argumento <cuenta en hp3000>, es una cuenta registrada en MPE de HP3000 y que es vital para poder transferir el archivo a la impresora.

Notas: Si el argumento <cuenta en hp3000> consta de caracteres como ",", ";", blancos u otros caracteres que sirvan como separadores, es necesario delimitar el argumento con comillas, ejemplo:
<cuenta en hp3000> = USUS,MGR.SYS
<arch. en swtpc> = /usr/jorge/sale

++list3000 /usr/jorge/sale "USUS,MGR.SYS"

Salida: Automatica. Despues teclear ++term3000 para dar fin de sesion.

hp_a_sw

Sintaxis: ++hp_a_sw <arch. en swtpc> <arch en hp3000> <cuenta en hp3000>

Uso: hp_a_sw permite transferir un archivo que reside en HP3000 de la cuenta <cuenta en hp3000>, hacia SwTPc.
El argumento <arch. en swtpc> representa la ruta y archivo en el que "caera" el archivo que viene de HP3000. Es importante antes de correr este programa crear el archivo en SwTPc de la siguiente manera:
++create <arch. en swtpc>
ejemplo:
si el archivo en SwTPc se llamara bckup y lo queremos en la ruta /usr/jorge, entonces teclear:
++create /usr/jorge/bckup
El argumento <arch. en hp3000> representa el archivo a transferir y que se encuentra en <cuenta en hp3000>
ejemplo:
Sea la cuenta MGR.MANAGER,SYS y el archivo a transferir CF001, entonces teclear:
++hp_a_sw /usr/jorge/bckup CF001 "MGR.MANAGER,SYS"

Notas: Si el argumento <cuenta en hp3000> consta de caracteres como ",", ";", blancos u otros caracteres que sirvan como separadores, es necesario delimitar el argumento con comillas.

Salida: Automatica. Despues teclear ++term3000 para dar fin de sesion o verificacion.

sw_a_hp

Sintaxis: ++sw_a_hp <arch. en swtpc> <arch. en hp3000> <cuenta en hp3000>

Uso: sw_a_hp permite transferir el <arch. en swtpc> hacia el <arch. en hp3000> de la cuenta <cuenta en hp3000>. El argumento <arch. en swtpc> representa la ruta y archivo en SwTPc que sera transferido hacia la cuenta <cuenta en hp3000> "cayendo" en <arch. en hp3000>. Es importante crear antes el <arch. en hp3000> de la siguiente manera:

```
++term3000
 :HELLO <cuenta en hp3000>
 :BUILD <arch. en hp3000>;REC=-80,1,F,ASCII;DISC=1000
 :BYE
 :"BREAK" (la tecla en SwTPc)
++etc.
```

Ejemplo:

<arch. en swtpc> = trans1
ruta en SwTPc = /usr/jorge
<arch. en hp3000> = BCKUP
<cuenta en hp3000> = MGR.MANAGER,CLIENTE
teclear:

```
++sw_a_hp /usr/jorge/trans1 BCKUP "MGR.MANAGER,CLIENTE"
```

Notas: Si el argumento <cuenta en hp3000> consta de caracteres como ",", ":", blancos u otros caracteres que sirvan como separadores, es necesario delimitar el argumento con comillas.

Salida: Automatica. Despues teclear ++term3000 para dar fin de sesion o verificacion.

```
          /*      PROGRAMA:    term3000       */
#include <modes.h>
#include <sgtty.h>
#include <stdio.h>
#include <signal.h>

  /* PROGRAMA PARA UTILIZAR LAS TERMINALES SwTPc COMO TERMINALES */
  /* DE LA HP3000. NOTA: NO SE PERMITE UTILIZAR "VIEW"           */

#define  RETERR  -1
#define  BIEN     0
#define  ENQ      5
#define  ACK      "6"
main (argn, argv)
     int  argn;
     char *argv[]; {
     int  ret, hijaid, termid, status;
     char buffer[512];

     int ent, sal, ret1, ret2;
     char let[10];

     printf("%c", 0x0C); /* SE "LIMPIA" LA PANTALLA EN SwTPc   */

     printf("\n\n       Enlace de SWTPc como terminal de HP 3000\n\n");

     hijaid = fork();          /*   CREA LA TAREA HIJA     */
     if (hijaid == BIEN)
        goto tarea_hija;

               /*    Este codigo lo ejecuta solo la tarea madre */

     if (hijaid == RETERR)
        {fprintf(stderr,"Error crear la tarea %d\n", errno);
         status = 1;
         goto sale;
        }

     ent=0;    /* SE ASIGNA EL "standar input"      */
     sal=open("/dev/tty03",1); /* SE ABRE EL PUERTO SERIE COMO SALIDA */

     ret1  = setraw(ent); /* SE ANULA EL "eco" DEL "standar input"  */
     ret2  = setraw(sal); /* Y DEL PUERTO SERIE                     */

     write(sal,"\n",1);

     while(let[0] != 3)
     {                        /* AQUI SE RECIBEN LOS CARACTERES DEL     */
      read(ent,let,1);        /* TECLADO (standar input) Y SON ENVIADOS */
      let[0] = let[0] & 0x7f; /* AL PUERTO SERIE (HP3000)               */
      if (let[0] != 3)
         write(sal,let,1);
     }

     setcooked(ent); /* SE RESTABLECE EL "eco" DEL "standar input"  */
     setcooked(sal); /* Y DEL PUERTO SERIE                          */

     close(ent); /* SE CIERRA EL "standar input"    */


     kill(hijaid,SIGTERM); /* SE ENVIA UNA INTERRUPCION A LA TAREA */
                           /* HIJA                                 */
espera:
```

```
      termid = wait(&status);  /* espera que termine          */
      if (termid != hijaid)    /* confirma que es la hija     */
         goto espera;          /* sigue esperando             */
      goto sale;               /* termina la tarea madre      */

  /* ESTA SECCION SOLO ES EJECUTADA POR LA TAREA HIJA:     */
tarea_hija:

      ent=open("/dev/tty03",2); /* ABRE EL PUERTO SERIE COMO E/S */
      sal=1;                    /* ASIGNA EL "standar output"    */

      ret1  = setraw(ent); /* SE ANULA EL "eco" DEL PUERTO SERIE */
      ret2  = setraw(sal); /* Y DEL "standar output"             */

      printf("\n\r");

      while(let[0] != 3)  /* AQUI SE REALIZA EL ENLACE COMO TERMINAL */
      {                   /* SE LEEN LOS CARACTERES "eco" GENERADOS  */
       read(ent,let,1);   /* POR LA HP3000 Y ENVIADOS A LA PANTALLA  */
                          /* (standar output) DE LA SWTPc            */
       if (let[0] == ENQ)
         {
          write(ent,ACK,1);
          continue;
         }
       write(sal,let,1);
      }                 /* TERMINA EL ENLACE COMO TERMINAL     */

      setcooked(ent);  /* SE RESTABLECE EL "eco" AL PUERTO SERIE */
      setcooked(sal);  /* Y DEL "standar output"                 */

      close(sal);      /* SE CIERRA EL "standar output"          */

      status = 0;
sale:
      exit(status);        /* termina la tarea madre o la hija */ }
setraw(f)
  int f; {
        struct sgttyb ttbuf;
        gtty(f,&ttbuf);
        ttbuf.sg_flags |= RAW;
        stty(f,&ttbuf); }
setcooked(f)
   int f; {
        struct sgttyb ttbuf;
        gtty(f,&ttbuf);
        ttbuf.sg_flags &= (~RAW);
        stty(f,&ttbuf); }
unbuf(f) {
        setbuf(f,0); }
```

```
          /*      PROGRAMA:    list3000       */

#include <modes.h>
#include <sgtty.h>
#include <stdio.h>
#include <signal.h>

#define  PMODE   0x08
#define  RETERR  -1
#define  BIEN    0
#define  ENQ     5
#define  ACK     "6"
#define  XON     17

/* PROGRAMA PARA ENVIAR UN LISTADO DE LA SWTPC A LA HP3000 */

#define  XOFF    "19"
#define  SOH     1
#define  SOI     15

#define  BOF     " "
#define  EOF     "0x19"
#define  CR      "13"
#define  LF      "10"
#define  BLANCO  "32"
#define  INI     "5613"
     char hello[40]   = "HELLO ";            /* GENERACION DEL HELLO */
     char filecmd[25] = "FILE LIST;DEV=LPn";/* CONFIGURACION DE LA */
     char yes[25]     = "YESn";              /*    IMPRESORA EN LA    */
     char bye[25]     = "BYEn";              /*        HP3000         */
     char mensaje[6]  = "*0x08";
main (argn, argv)
     int  argn;
     char *argv[]; {
     int  ret, hijaid, termid, status;
     char buffer[512];

     int hp3000, sal, ret1, ret2, ent1, i, ret3, contlf, len;
     char let[10], fcopy[81], archent[81];
     char file[10], mientras;

     if (argn < 3)     /* VALIDACION DE LOS ARGUMENTOS  */
        {
         printf("n Faltan parametros n");
         exit (0);
        }

     else              /* ABRE EL ARGUMENTO <arch en swtpc> */
        {strcpy(archent,argv[1]);
         ent1 = open(archent,0);    /* abre solo para leer */
         if (ent1 < 0)
            {printf("Error al abrir 2s lr 2snl 2snl",
                     archent,hello,fcopy);
             exit(0);
            }
        }

     strcat(hello,argv[2]);  /* CONCATENA EL HELLO CON <cuenta en hp> */
     strcat(hello,"n");
        /* COMANDO FCOPY CON SU ASIGNACION DEL ARCHIVO DE SALIDA */
     strcpy(fcopy,"FCOPY FROM;TO=*LISTn");

     hp3000=open("/dev/tty03",2); /* ABRE EL CANAL SERIE */
```

```
sal=1;                          /* ASIGNAL EL "standar output" */

ret1  = setraw(hp3000);    /* SE ANULAN LOS "ecos" DEL PUERTO SERIE */
ret2  = setraw(sal);       /* Y DEL "standar output"                */

/*   ESPERA ENLACE CON LA HP 3000 !!!!!!!!!!!!!!!!!!!!!!!!!!!! */

write(hp3000,"n",1);
prompt(hp3000,sal,let);      /*  RUTINA DE PROTOCOLO   */

/*  YA ESTAN ENLAZADAS          !!!!!!!!!!!!!!!!!!!!!!!!  */

len=strlen(hello);              /*    ENVIA EL       */
write(hp3000,hello,len);        /* HELLO DE HP 3000 */
prompt(hp3000,sal,let);

len=strlen(filecmd);
write(hp3000,filecmd,len);    /* ASIGNA LA IMPRESORA */
prompt(hp3000,sal,let);

len=strlen(fcopy);             /*   ENVIA EL       */
write(hp3000,fcopy,len);       /* COMANDO FCOPY    */
prompt(hp3000,sal,let);

len=strlen(yes);
write(hp3000,yes,len);         /* CONTESTA YES          */
prompt(hp3000,sal,let);

printf(" Enviando: ");
mientras = -1;

while (mientras != 0)
      {
       mientras = read(ent1,let,1);

       if (mientras == 0)
          {let[0] = 25;
           write(hp3000,let,1); /* SE INICIA LA TRANSFERENCIA DEL  */
           prompt(hp3000,sal,let);  /* ARCHIVO AL SPOOLER DE LA HP */
           break;
          }

       let[0] = let[0] & 0x7f;
       write(hp3000,let,1);

       if (let[0] == 13)
          {
           prompt(hp3000,sal,let);
          }
      }

len=strlen(bye);
write(hp3000,bye,len);        /*  COMANDO  BYE  */

printf("nll");
setcooked(hp3000);   /* SE RESTABLECE EL "eco" DEL PUERTO SERIE */
setcooked(sal);      /* Y DEL "standar output"                  */

close(ent1);         /* CIERRA EL ARCHIVO EN SwTPc   Y TAMBIEN  */
close(hp3000);       /* EL PUERTO SERIE                         */
```

```
sale:
      exit(status);         /* TERMINA EL PROGRAMA     */ }
prompt(hp3000,sal,let)
      int hp3000, sal;
      char *let; {
      int *inpp;
      int i;
      char inp[2], aster, backsp;

      inpp = inp;


      *let=0;
      while(*let != XON)
            {
             read(hp3000,let,1);
             *let = *let & 0x7f;

             inp[1] = *let;
             inp[0] = 0;

            if (*let == ENQ)
               {
                write(hp3000,ACK,1);
                continue;
               }

             if (*let == 25)
                 continue;
            }

        i = 0;
        while(i < 3 )
             {aster = '*';
              backsp = 8;
              write(sal,&backsp,1);
              write(sal,&aster,1);
              write(sal,&backsp,1);
              write(sal,&aster,1);
              ++i;
             } }
setraw(f)
  int f; {
        struct sgttyb ttbuf;
        gtty(f,&ttbuf);
        ttbuf.sg_flag |= RAW;
        stty(f,&ttbuf); }
setcooked(f)
   int f; {
        struct sgttyb ttbuf;
        gtty(f,&ttbuf);
        ttbuf.sg_flag &= (~RAW);
        stty(f,&ttbuf); }
unbuf(f) {
        setbuf(f,0); }
```

```
          /*          PROGRAMA      hp_a_sw       /*

#include <modes.h>
#include <sgtty.h>
#include <stdio.h>
#include <signal.h>

#define  PMODE  0x08
#define  RETERR  -1
#define  BIEN    0
#define  ENQ     5
#define  ACK     "6"
#define  XON     17
#define  XOFF    "19"
#define  SOH     1
#define  SOI     15
#define  SIG     "17"

 /*       PROGRAMA PARA ENVIAR ARCHIVOS DE LA HP3000 A LA SwTPc       */
 /*       RECUERDE QUE UNICAMENTE USTED  PUEDE  HACER  USO  DEL       */
 /*       PAQUETE DE COMUNICACION EXCEPTO EN  EL  CASO  MARCADO       */
 /*       POR EL MANUAL (APENDICE D)                                  */

#define  BOF     " "
#define  EOF     13
#define  CR      "13"
#define  LF      "10"
#define  BLANCO  "32"
#define  INI     "5613"
      char hello[40]="HELLO ";   /*  GENERACION DEL COMANDO HELLO   */
      char bye[25]="BYEn";       /*  GENERACION DEL COMANDO BYE     */

 main (argn, argv)
      int  argn;
      char *argv[];
 {
      int *ret, hijaid, termid, status;
      char buffer[512];

      int cont;
      int hp3000, i, ret1, ret2, sal1, i, ret3, len, contlf;
      char let[10], archsal[81], archent[81];
      char file[10], mientras;

      ret = write(2, "Inicia el proceso-----nnn", 26);
      if (ret == RETERR)
         {fprintf(stderr,"Error al enviar el mensaje 1 13dnn", errno);
          status = 2;
          goto 1e;
         }

      /*  VALIDACION DE LOS ARGUMENTOS   */

      if (argn < 3)
         { printf("r !  FALTO NOMBRE DE ARCHIVOS !  r");
           exit(1);
         };
      if (argn < 4)
         { printf("r !  FALTO NOMBRE DE LA CUENTA EN HP !  r");
           exit(1);
         }

      strcpy(archsal,argv[1]);        /* AQUI SE CONCATENA EL ARGUMENTO   */
```

```
strcpy(archent,"FCOPY FROM=");  /* <arch.  en SwTPc> CON EL COMANDO */
strcat(archent,argv[2]);        /* FCOPY; ADEMAS LA CONCATENACION   */
strcat(archent,";TO=");         /* DEL ARGUMENTO <arch en hp3000>   */
strcat(hello,argv[3]);          /* AL MISMO COMANDO                 */
strcat(hello,"\n");  /* TAMBIEN SE FORMA EL "HELLO <cuenta en hp300>

sal1 = open(archsal,1);    /* abre solo para escribir */

if (sal1 < 0)
   { printf("Error al abrir %s %s\n %s\n",
             archsal,hello,archent);
     exit(0);
   }

hp3000=open("/dev/ttyO3",2);/* SE ABRE EL PUERTO SERIE DE LA SwTPc */
sal=1;                      /* Y SE ASIGNA EL "standar output"     */

printf("La salida es: %x  \n\n",hp3000);

ret1  = setraw(hp3000);    /* SE ANULAN LOS "ecos"  DEL PUERTO SERIE*/
ret2  = setraw(sal);       /* Y DEL "standar output"                */


/*   ESPERA ENLACE CON LA HP 3000 !!!!!!!!!!!!!!!!!!!!!!!!!!!! */

write(hp3000,"\n",1);
prompt(hp3000,sal,let);          /* RUTINA DE PROTOCOLO   */

/*  YA ESTAN ENLAZADAS            !!!!!!!!!!!!!!!!!!!!!!!  */

len=strlen(hello);               /*    ENVIA EL       */
write(hp3000,hello,len);         /* HELLO DE HP 3000 */

prompt(hp3000,sal,let);

len=strlen(archent);             /*    ENVIA EL       */
write(hp3000,archent,len);       /* COMANDO FCOPY     */
cont = 0;

while(mientras != XON)
{
 read(hp3000,let,1);
 let[0] = let[0] & 0x7f;         /* FILTRO SECUENCIAS DE ESCAPE */
 mientras = let[0];
 cont = cont++;

 if (let[0] == ENQ)
   {
    write(hp3000,ACK,1);
    continue;
   }

 if (cont == 256 )
    {
   write(hp3000,XOFF,1);
   while(cont != 0)
        cont = cont - 1;
   write(hp3000,SIG,1);
   }
 /* SE INICIA LA TRANSFERENCIA ENTRE COMPUTADORAS  */
```

```
        write(sal,let,1);              /* SALE A PANTALLA */
        if (let[0] != 10)
           write(sal1,let,1);          /* SE GRABA EN DISCO */
      }
      len = strlen(bye);
      write(hp3000,bye,len);

      contlf = 0;
      while(contlf < 3)
           {
            read(hp3000,let,1);
            let[0] = let[0] & 0x7f;

             if (let[0] == ENQ)
               {write(hp3000,ACK,1);
                continue;
               }

            if (let[0] != 17)
           /*   write(sal,let,1);      */

            if (let[0] == 10)
                contlf++;

           }

      printf("nll");
      close(sal1);            /* CIERRA EL ARCHIVO EN SwTPc          */
      setcooked(hp3000);      /* RESTABLECE EL "eco" AL PUERTO SERIE */
      setcooked(sal);         /* Y AL "standar output                */

      close(hp3000);          /* CIERRA EL PUERTO SERIE              */


 sale:
      exit(status);         /* TERMINA EL PROGRAMA   */
 }
prompt(hp3000,sal,let)
      int hp3000, sal;
      char *let;
 {
      int *inpp;
     char inp[2];

      inpp = inp;


      *let=0;
     while(*let != XON)
           {
            read(hp3000,let,1);
            *let = *let & 0x7f;

            inp[1] = *let;
            inp[0] = 0;

           if (*let == ENQ)
              {
               write(hp3000,ACK,1);
               continue;
```

```
            }

            if (*let < 32)
                {
                 printf("control: 2x nl",*inpp);
                 continue;
                }

            write(sal,let,1);
           }
            printf("salio nl");
}

setraw(f)
  int f;
{
        struct sgttyb ttbuf;
        stty(f,&ttbuf);
        ttbuf.sg_flags |= RAW;
        stty(f,&ttbuf);
}

setcooked(f)
   int f;
{
        struct sgttyb ttbuf;
        stty(f,&ttbuf);
        ttbuf.sg_flags &= (~RAW);
        stty(f,&ttbuf);
}

unbuf(f)
{
        setbuf(f,0);
}
```

```
          /*      PROGRAMA:    sw_a_hp      */

#include <modes.h>
#include <sgtty.h>
#include <stdio.h>
#include <signal.h>

#define  PMODE  0x08
#define  RETERR  -1
#define  BIEN    0
#define  ENQ     5
#define  ACK     "6"
#define  XON     17
#define  XOFF    "19"
#define  SOH     1
#define  SOI     15

#define  BOF     " "
#define  EOF     "0x19"
#define  CR      "13"
#define  LF      "10"
#define  BLANCO  "32"
#define  INI     "5613"

      /* PROGRAMA PARA ENVIAR ARCHIVOS DE LA SwTPc A LA HP3000  */

     char hello[40] = "HELLO ";  /* GENERACION DEL COMANDO HELLO */
     char bye[25]   = "BYE\n";   /* GENERACION DEL COMANDO BYE   */
     char mensaje[6]  = "*0x08";
main (argn, argv)
     int  argn;
     char *argv[]; {
     int  ret, hijaid, termid, status;
     char buffer[512];

     int hp3000, sal, ret1, ret2, ent1, i, ret3, contlf, len;
     char let[10], fcopy[81], archent[81];
     char file[10], mientras;

           /* VALIDACION DE LOS ARGUMENTOS   */

     if (argn < 4)
        { printf("\r !  FALTO NOMBRE DE ARCHIVOS !  \r");
          exit(1);
        };

     strcpy(archent,argv[1]);     /* CONCATENACION DE LOS ARGUMENTOS CON */
     strcpy(fcopy,"FCOPY FROM:TO=");  /* CON SUS RESPECTIVOS COMANDOS    */
     strcat(fcopy,argv[2]);
     strcat(fcopy,"\n");
     strcat(hello,argv[3]);
     strcat(hello,"\n");
                                 /* SE ABRE EL ARGUMENTO <arch en swtpc>  */
     ent1 = open(archent,0);     /* abre solo para leer */

     if (ent1 < 0)
        { printf("Error al abrir %s \r %snl %snl",
                  archent,hello,fcopy);
          exit(0);
        }

     hp3000=open("/dev/tty03",2); /* SE ABRE EL PUERTO SERIE DE SwTPc   */
     sal=1;                        /* Y ASIGNA EL "standar output"      */
```

```
ret1   = setraw(hp3000); /* SE ANULAN LOS "ecos" DEL PUERTO SERIE  */
ret2   = setraw(sal);    /* Y DEL "standar output"                 */


/*   ESPERA ENLACE CON LA HP 3000 !!!!!!!!!!!!!!!!!!!!!!!!!!!! */

write(hp3000,"n",1);
prompt(hp3000,sal,let);

/*  YA ESTAN ENLAZADAS           !!!!!!!!!!!!!!!!!!!!!!!   */

len=strlen(hello);
write(hp3000,hello,len);        /* HELLO DE HP 3000 */

prompt(hp3000,sal,let);     /* RUTINA DE PROTOCOLO */

len=strlen(fcopy);             /*    ENVIA EL     */
write(hp3000,fcopy,len);       /* COMANDO FCOPY */

prompt(hp3000,sal,let);

mientras = -1;

while (mientras != 0)
      {
       mientras = read(ent1,let,1); /* LEE EL ARCHIVO EN SwTPc  */

       if (mientras == 0)
          {let[0] = 25;
           write(hp3000,let,1);
           prompt(hp3000,sal,let);
           break;
          }

    /* SE INICIA LA TRANSFERENCIA DEL ARCHIVO A LA HP3000 */

       let[0] = let[0] & 0x7f;
       write(hp3000,let,1);
       write(sal,let,1);

       if (let[0] == 13)
          {
           prompt(hp3000,sal,let);
          }
      }




*let=0;
while(*let != XON)
     {
      write(hp3000,CR,1);
      read(hp3000,let,1);
      *let = *let & 0x7f;
      if (*let == XON)
         break;
     }

len=strlen(bye);
```

```
     write(hp3000,bye,len);          /* COMANDO BYE  */
     printf("nll");

     setcooked(hp3000);    /* RESTABLECE EL "eco" AL PUERTO SERIE */
     setcooked(sal);       /* Y AL "standar output"               */

     close(ent1);          /* CIERRA EL ARCHIVO DE SwTPc          */
     close(hp3000);        /* CIERRA EL PUERTO SERIE

sale:
     exit(status);         /*   TERMINA EL PROGRAMA    */ }
prompt(hp3000,sal,let)
     int hp3000, sal;
     char *let; {
     int *inpp;
     int i;
     char inp[2], aster, backsp;

     inpp = inp;


     *let=0;
     while(*let != XON)
          {
           read(hp3000,let,1);
           *let = *let & 0x7f;

           inp[1] = *let;
           inp[0] = 0;
           if (*let == ENQ)
              {
               write(hp3000,ACK,1);
               continue;
              }

           if (*let == 25)
               continue;
          }

       i = 0;
       while(i < 3 )
            {aster = '*';
             write(sal,&aster,1);
             backsp = 8;
             write(sal,&backsp,1);
             write(sal,&aster,1);
             write(sal,&backsp,1);
             ++i;
            } }
setraw(f)
  int f; {
       struct sgttyb ttbuf;
       gtty(f,&ttbuf);
       ttbuf.sg_flag |= RAW;
       stty(f,&ttbuf); }
setcooked(f)
   int f; {
       struct sgttyb ttbuf;
       gtty(f,&ttbuf);
       ttbuf.sg_flag &= (~RAW);
       stty(f,&ttbuf); }
unbuf(f) {
       setbuf(f,0); }
```