73
24.

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES "CAMPUS ARAGÓN"

"DISEÑO DE UN SISTEMA DE CONTROL PARA LA INTERFAZ DNT QUE NOS MUESTRA UNA APLICACIÓN REAL DE LOS COMPONENTES QUE FORMAN A CUALQUIER SISTEMA ELECTRÓNICO DIGITAL CON CAPACIDAD DE PROCESAMIENTO DE DATOS"

T E S I S

QUE PARA OBTENER EL TITULO DE

INGENIERO EN COMPUTACIÓN

P R E S E N T A :

HUGO EDUARDO VAZQUEZ LUGO

México 1997.

CONTENIDO

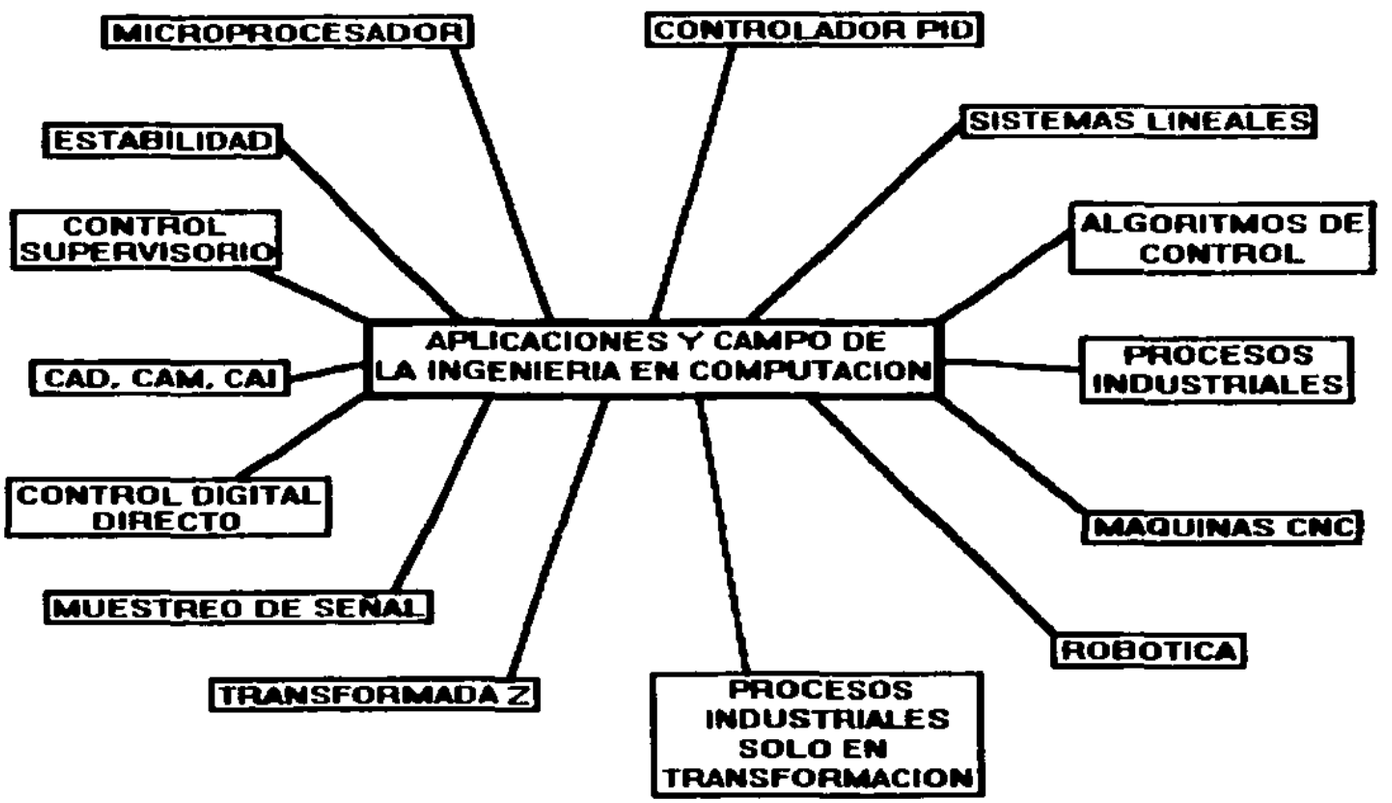
- Introducción.
- Capítulos.
- Conclusiones.
- Bibliografía.

INTRODUCCION.

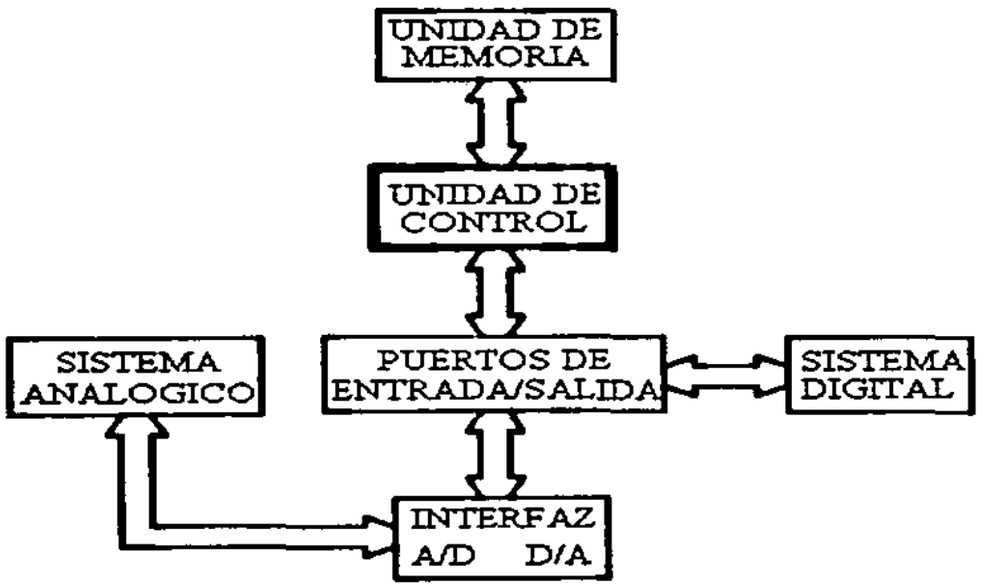
En este trabajo de tesis se sustenta la idea fundamental de lo que, en la actualidad, representa la Ingeniería en Computación para la Universidad Nacional Autónoma de México, sustrayendo los conocimientos de Comunicaciones y Electrónica, para unificarlos junto con los de la Informática.

Es de tal idea, que surge la tentación de realizar un ejemplo que implique el manejo de los términos de las materias mencionadas anteriormente y que, a su vez, den reconocimiento de la formación del Ingeniero en Computación en un sentido de diseño y creación de nuevas tecnologías, pero que a su vez, lo reconozca como un profesional capaz de manejar la tecnología actual para usarla o mejorarla con organización y responsabilidad, con el fin de obtener beneficios sociales, culturales y económicos, según le sea demandado.

El sistema desarrollado se enmarca en lo que conocemos como Ingeniería en Computación; parte de la ciencia que trata del estudio y desarrollo de sistemas físicos y lógicos que, vistos desde el punto de vista de esta rama, son electrónicos y léxicos, respectivamente. En la figura siguiente se muestra la fuente de trabajo y recursos del Ingeniero en Computación.



Estos sistemas han sustentado los medios que permiten la conectividad y desarrollo de unidades de control con unidades periféricas. La figura siguiente indica cada una de las unidades.



TITULO DE LA TESIS:
DISEÑO DE UN SISTEMA DE CONTROL PARA LA INTERFAZ DNT QUE NOS MUESTRA UNA APLICACION REAL DE LOS COMPONENTES QUE FORMAN A CUALQUIER SISTEMA ELECTRONICO DIGITAL CON CAPACIDAD DE PROCESAMIENTO DE DATOS

CAPITULO I:
CONCEPTOS BASICOS DE MICROPROCESADORES.

CAPITULO II:
ENTORNO DE LOS MICROPROCESADORES.

CAPITULO III:
EL MICROPROCESADOR Z80A.

CAPITULO IV:
CIRCUITOS DE MEMORIA.

CAPITULO V:
CIRCUITOS DE ENTRADA/SALIDA.

CAPITULO VI:
LA INTERFAZ DNT (DECODIFICADOR DE NUMEROS TELEFONICOS).

CAPITULO VII:
DISEÑO DEL CIRCUITO DIGITAL (HARDWARE).

CAPITULO VIII:
DISEÑO DE LOS PROGRAMAS DE CONTROL (SOFTWARE).

CAPITULO 1

CONCEPTOS BASICOS DE MICROPROCESADORES.

Es difícil dar una definición exacta de lo que es un µprocesador, sobre todo porque en los últimos años, el desarrollo de la electrónica en esta área ha dado lugar a la aparición de un sinnúmero de estos dispositivos, diseñados con enfoques hacia diversos campos de aplicación. Sin embargo, todos los µprocesadores tienen ciertos atributos comunes que los distinguen de otros elementos electrónicos.

Las características principales de un µprocesador son su "universalidad" y su "programabilidad", ya que hacen de él un dispositivo tan versátil y poderoso, que puede utilizarse como el elemento inteligente o "cerebro" en aplicaciones que van, desde una computadora personal, hasta un rastreador de satélites o un aparato detector de epilepsia.

Esencialmente, un µprocesador es un circuito de alta escala de integración (LSI), compuesto de muchos circuitos más simples como son: flip-flops, contadores, registros, decodificadores, comparadores, etcétera; todos ellos dentro de la misma pastilla de silicio, de modo que el µprocesador puede ser considerado un dispositivo lógico de propósito general o universal.

La PROGRAMABILIDAD se refiere a la capacidad que tiene un µprocesador para que su función sea definida a través de un programa. El PROGRAMA consta de una serie de órdenes o INSTRUCCIONES relacionadas, ejecutadas secuencialmente (una a la vez) por el µprocesador, y que pueden implicar operaciones lógicas o aritméticas. Las instrucciones se especifican por medio de un código especial, el cual constituye el lenguaje del µprocesador.

De los párrafos anteriores, se puede deducir la existencia de dos géneros diferentes de elementos que se complementan y que, juntos, delimitan el concepto del µprocesador como un dispositivo útil. El primero de ellos lo forman el circuito mismo, sus componentes electrónicos, sus alambres de interconexión y, en general, todo aquello determinado por la configuración física del circuito integrado. A este género se le conoce como el HARDWARE o soporte físico del µprocesador, ya que, por su misma naturaleza, no admite modificaciones. Por otro lado, se encuentran el código de instrucciones, los algoritmos y los programas que determinan el comportamiento del µprocesador, los cuales pueden ser alterados cuantas veces sea necesario, con el fin de adaptar y optimizar el desempeño del dispositivo a algún campo de aplicación particular. Este segundo género recibe el nombre de SOFTWARE o soporte lógico.

1.1 Unidades funcionales de un µprocesador.

En la terminología de µprocesadores, a cada grupo de circuitos que desempeña tareas similares se le llama UNIDAD FUNCIONAL, y el conjunto de unidades funcionales y la forma como están interconectadas se denominan ARQUITECTURA del µprocesador.

Así, los circuitos quedan agrupados en las siguientes unidades funcionales: las compuertas lógicas y el selector en la UNIDAD ARITMETICO/LOGICA; el oscilador y el divisor de frecuencia constituyen la UNIDAD DE CONTROL; el contador binario, en su papel de contador del programa, forma parte de los REGISTROS INTERNOS; y la memoria ROM, grabada con las instrucciones, es la MEMORIA DEL PROGRAMA.

Hay otras unidades funcionales que también pueden existir en un µprocesador; éstas son la MEMORIA DE DATOS y los PUERTOS DE ENTRADA/SALIDA.

Para que a un circuito se le pueda dar el nombre de µprocesador, debe contener en una sola pastilla de silicio, al menos, las siguientes unidades: unidad de control, unidad aritmético/lógica y algunos registros.

Cuando un µprocesador posee únicamente las tres unidades funcionales básicas, se le conoce como UNIDAD DE PROCESAMIENTO CENTRAL (CPU) o simplemente MICROPROCESADOR (µP). Si un mismo circuito integrado tiene, además de las unidades básicas, otras tales como memoria (de programa y/o de datos) y puertos, este circuito ya no se considera estrictamente un µprocesador, porque las unidades adicionales que contiene le dan la capacidad de una computadora. Por esta razón, a estos µprocesadores se les llama también MICROCONTROLADORES.

Más adelante, se muestra un diagrama de bloques de un µprocesador y de un µcontrolador (fig. 1-1).

A continuación se describirán las unidades funcionales que se encuentran dentro de la CPU.

1.1.1 La unidad de control.

La circuitería de control es la unidad funcional primaria dentro del µprocesador. Utilizando señales de reloj, la unidad de control mantiene la secuencia de eventos apropiada para llevar a cabo cualquier tarea de procesamiento; es decir, el µprocesador es un dispositivo síncrono.

La actividad fundamental de un µprocesador, regulada por la unidad de control, es cíclica; consiste en la búsqueda y obtención de datos e instrucciones, y en la ejecución secuencial de estas últimas. Después de que una instrucción ha sido obtenida y decodificada, la

circuitería de control envía las señales apropiadas a los dispositivos, tanto internos como externos, a la CPU, para iniciar la acción de procesamiento indicada por la instrucción.

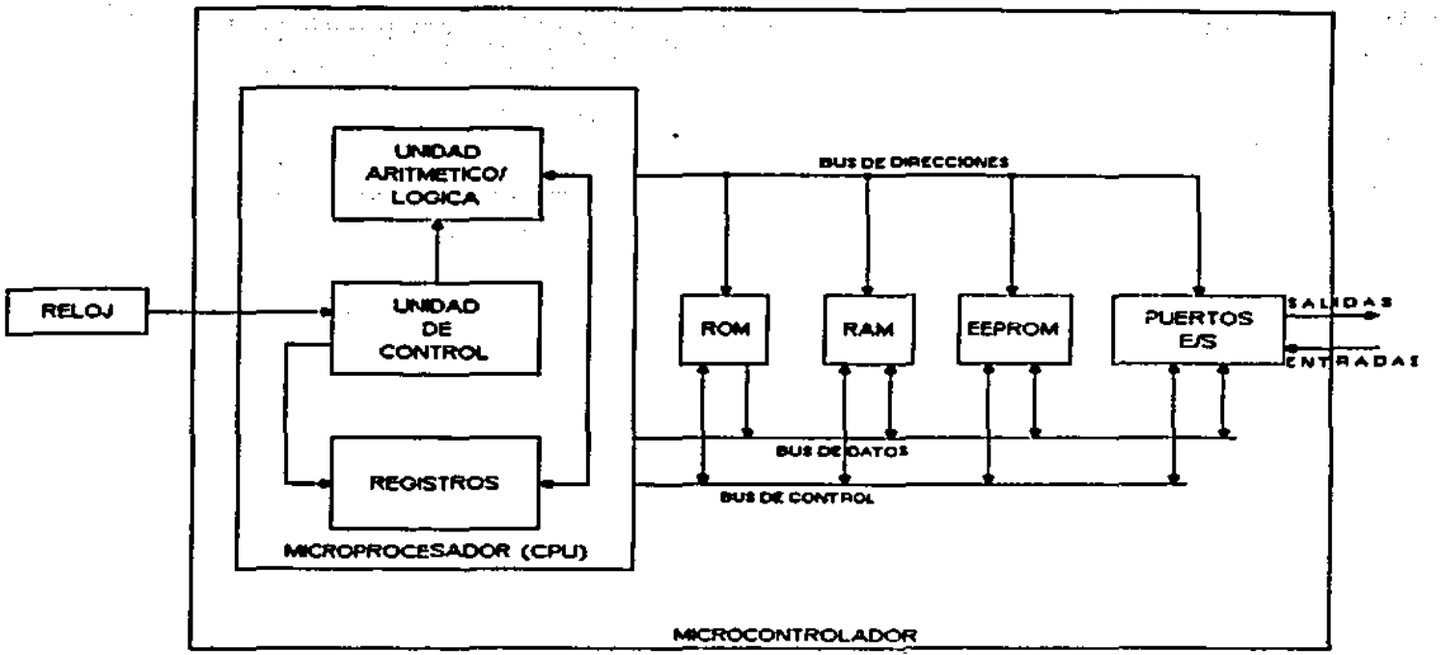


Figura 1-1

Frecuentemente, la unidad de control es capaz de responder a señales externas que alteran el estado del µprocesador, ya sea interrumpiendo temporalmente su funcionamiento, o provocando la ejecución de instrucciones especiales.

El corazón de la unidad de control lo constituye el GENERADOR DE CICLO DE MAQUINA (GCM), que se encarga de producir las señales de control, derivándolas de un reloj u oscilador maestro como referencia. A continuación se presenta el diagrama de bloques de la unidad de control, y su relación con otros elementos de la CPU (fig. 1-2).

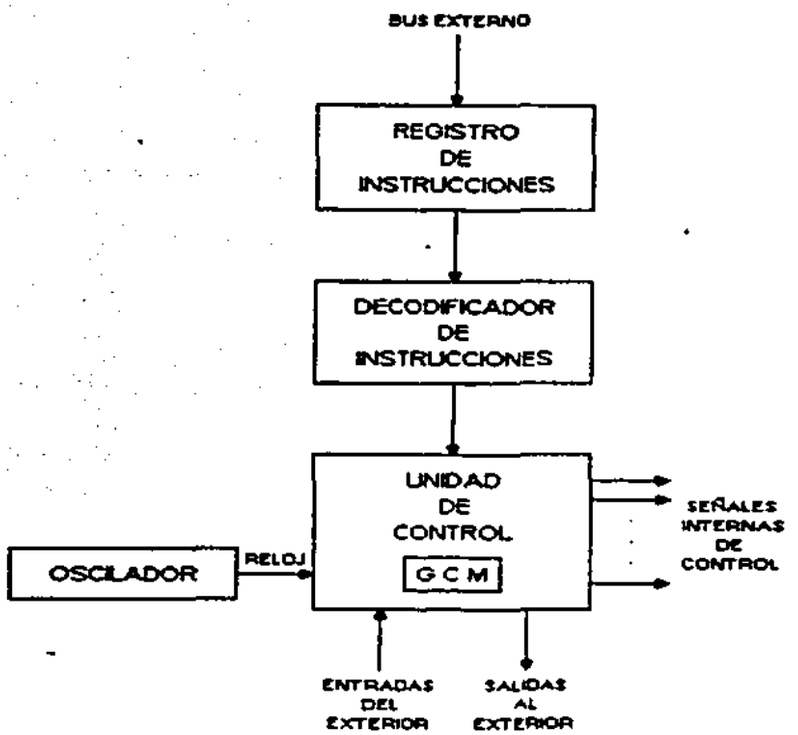


Figura 1-2

1.1.2 La unidad aritmético/lógica.

Todos los μprocesadores contienen una unidad aritmético/lógica que, con frecuencia, se conoce simplemente como ALU (Arithmetic/Logic Unit). La ALU, como su nombre lo indica, es la parte del μprocesador que lleva a cabo las operaciones aritméticas y lógicas en los datos binarios (fig.1-3). Algunas de ellas se aplican sobre dos operandos, mientras que otras requieren solamente uno.

Generalmente, la ALU es capaz de ejecutar las siguientes operaciones:

1. Suma aritmética.
2. Funciones lógicas AND, OR, XOR.
3. Complemento.
4. Rotación hacia la derecha o izquierda.

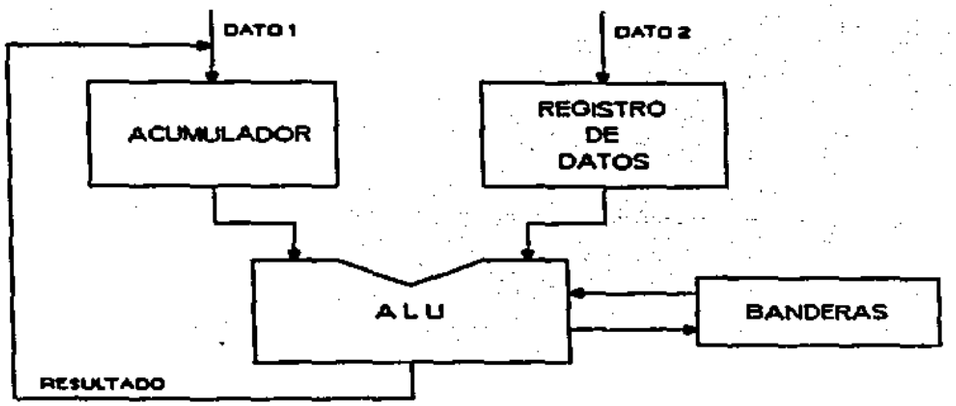


Figura 1-3

Además, la ALU contiene un conjunto de flip-flops llamados BANDERAS (flags), las cuales guardan información relacionada con el resultado de una operación aritmética o lógica. Por ejemplo, una de las banderas sirve para indicar si el resultado de la última operación fue cero.

1.1.3 Los registros internos.

Los registros internos son unidades de almacenamiento temporal dentro de la CPU. Algunos de ellos tienen usos específicos, mientras que otros son de propósito general. En esta sección, se estudiarán únicamente los registros de uso específico más importantes, dejando la descripción de los registros de propósito general para una sección posterior de este capítulo.

Para llevar una cuenta de cuál instrucción es la que debe ejecutar en seguida, la unidad de control mantiene un registro con la dirección de la siguiente instrucción en el programa. A este registro se le llama CONTADOR DEL PROGRAMA (Program Counter) o PC. El µprocesador actualiza el contenido del PC, incrementándolo cada vez que obtiene una instrucción de la memoria.

Una de las entradas de control al µprocesador, es la entrada de RESET (restablecer). Cuando se restablece el µprocesador, la unidad de control carga el contador del programa con ceros. Este valor inicial establece la dirección de memoria, de donde se va a obtener la primera instrucción.

Después que se ha obtenido una instrucción de la memoria, la CPU la almacena en un registro conocido como REGISTRO DE INSTRUCCIONES (Instruction Register) o IR (fig. 1-2). La instrucción almacenada en el IR es decodificada y usada para activar una de varias líneas. Cada línea representa un conjunto de actividades asociadas con la ejecución

de una instrucción particular. El dispositivo que traduce la instrucción en acciones concretas es el DECODIFICADOR DE INSTRUCCIONES (Instruction Decoder) (fig.1-2).

La primera palabra de una instrucción, es el código de operación para esa instrucción. El CODIGO DE OPERACION indica, a la unidad de control, las operaciones requeridas en la ejecución de la instrucción. Las instrucciones de un µprocesador, frecuentemente, requieren más información de la que puede contener una sola palabra de memoria. Por lo tanto, es común que las instrucciones consten de dos o tres palabras. La primera palabra es, siempre, el código de operación (OP code). Las otras palabras son datos que representan, ya sea una dirección, o una constante. Después de que el código de operación se ha leído de la memoria, y puesto en el registro de instrucciones, su decodificación indica si la instrucción requiere información adicional. Las partes de una instrucción de varias palabras, están contenidas en localidades sucesivas de la memoria.

Existe un registro íntimamente relacionado con la ALU, denominado ACUMULADOR. Generalmente, el acumulador contiene uno de los operandos que serán manipulados por la ALU y, también muy frecuentemente, el resultado de la operación se deposita en ese registro, reemplazando a uno de los operandos originales (fig. 1-3).

1.1.4 La memoria del programa.

Aunque anteriormente se dijo que la memoria no era parte integrante del µprocesador, considerándolo como CPU, es conveniente hablar ahora de la memoria del programa, puesto que sin ella, el µprocesador se convierte en un dispositivo inservible.

La MEMORIA DEL PROGRAMA es una memoria de lectura solamente. Como por ejemplo, una ROM o una EPROM que contiene el programa; es decir, la secuencia de instrucciones que va a determinar las tareas que realice el µprocesador. En algunos casos, la memoria del programa también almacena parámetros o tablas de datos que no sufren modificaciones. Usualmente, las EPROM se utilizan durante la etapa de desarrollo del prototipo del sistema, para así poder depurar y alterar el programa. Las ROM se prefieren cuando el sistema ya se encuentra en producción, y se ha llegado a la versión final del programa.

1.2 Diagrama a bloques de un µprocesador.

En los párrafos anteriores, se definieron las unidades funcionales básicas de un µprocesador, y se explicaron sus funciones. Sin embargo, por la misma estructura del µprocesador, que determina que todas ellas estén implantadas en una sola pastilla de silicio y encapsuladas en el circuito integrado, se dificulta estudiar por separado el comportamiento de cada una de ellas, cuando sólo se cuenta con un µprocesador comercial.

Por ello, se ha considerado muy conveniente que, antes de comenzar el estudio de un µprocesador, se analice uno en el que cada unidad funcional esté constituida por uno o más circuitos integrados, adquiriéndose así un conocimiento más profundo y detallado de las partes que componen al µprocesador.

A continuación se encuentra el diagrama de bloques del µprocesador, en el que se describen cada una de las partes que lo forman (fig. 1-4).

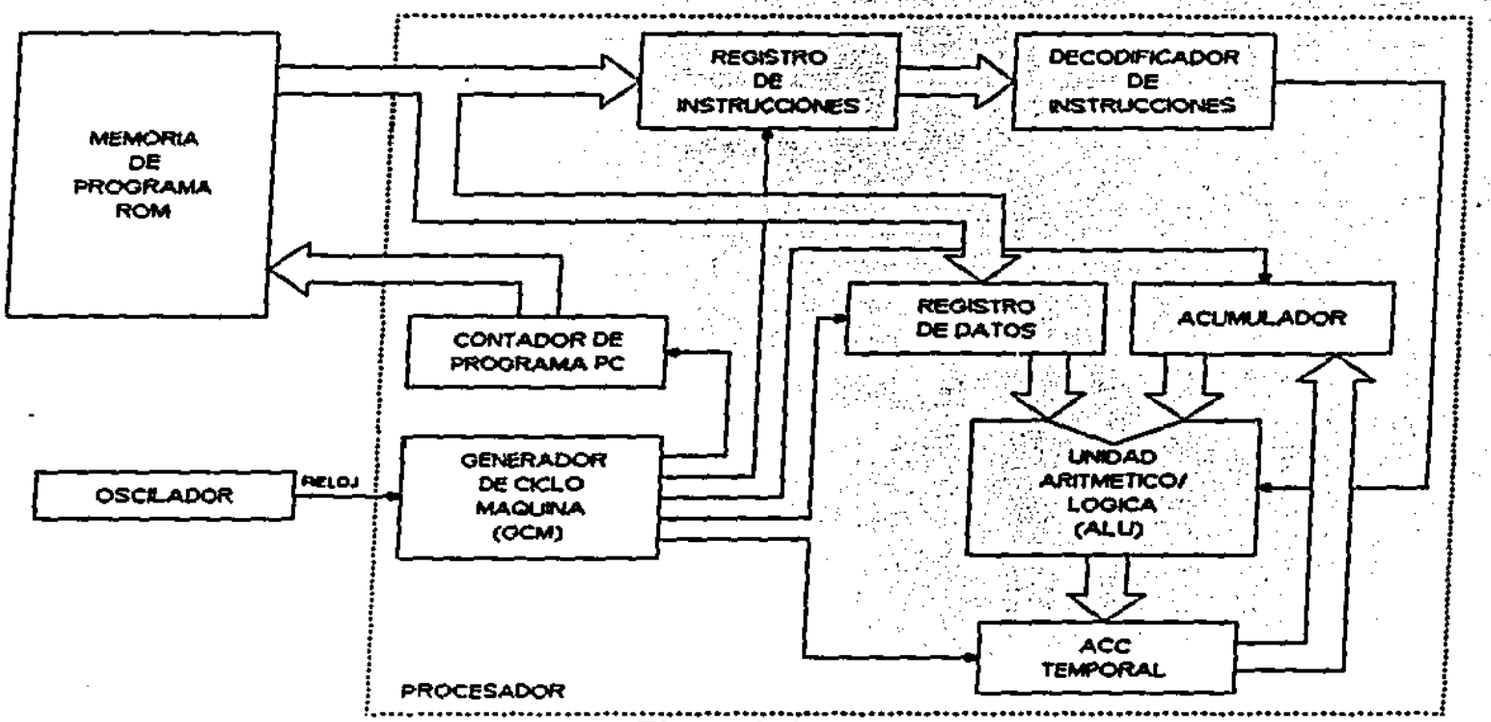


Figura 1-4

GENERADOR DE CICLO DE MAQUINA. Este circuito es el que marca el paso del procesador; su función es sincronizar al sistema por medio de señales de control que van a todos los registros. El GCM es alimentado por una señal de reloj, proveniente de un oscilador de onda cuadrada.

CONTADOR DEL PROGRAMA. Es un contador binario como los ya estudiados, que puede ser programable. Se incrementa por la acción del GCM.

UNIDAD ARITMETICO/LOGICA. Está formada por compuertas (AND, OR, XOR e inversores) que realizan funciones lógicas, y por circuitos que ejecutan operaciones aritméticas, como la suma. En esta unidad se procesan los datos, de acuerdo a las instrucciones del programa.

REGISTRO DE INSTRUCCIONES. Es un conjunto de flip-flops tipo latch (cerrojo) que forman un registro, en el cual se almacena toda instrucción, cada vez que se ha obtenido de la memoria.

DECODIFICADOR DE INSTRUCCIONES. Es el elemento que se encarga de convertir, cada instrucción, en una señal que seleccione la operación apropiada de la ALU. Un decodificador de binario a decimal puede servir para este propósito.

REGISTRO DE DATOS. Es un conjunto de latches que forma el registro, donde se guardan los datos provenientes de la memoria.

ACUMULADOR. Es un conjunto de latches que recoge el resultado de la última operación efectuada por la ALU. Sirve también para guardar uno de los datos de entrada a la ALU, cuando ésta ejecuta una operación.

ACUMULADOR TEMPORAL. Es un conjunto de latches que almacena temporalmente cualquier resultado proveniente de la ALU, mientras se transfiere al acumulador.

MEMORIA DEL PROGRAMA. Es una memoria de lectura solamente (ROM), que contiene las instrucciones y los datos necesarios para que el procesador ejecute la tarea que se le ha encomendado. Aunque en rigor no forma parte del µprocesador (CPU), es absolutamente indispensable para su funcionamiento.

1.3 Conceptos adicionales de un µprocesador.

1.3.1 Los registros internos de propósito general.

En la sección 1.1.3. se habló de los registros internos como una de las unidades funcionales básicas en un µprocesador. Entonces, solo se describieron los registros de uso específico, indispensables para el funcionamiento del µprocesador.

Sin embargo, con frecuencia, un µprocesador contiene un número de registros adicionales que no tienen asignada ninguna función en particular, sino más bien, se usan en tareas generales como lugares de almacenamiento temporal para guardar operandos o resultados intermedios.

La disponibilidad de registros de propósito general, elimina la necesidad de mover los datos, de un lado a otro, entre la memoria y el acumulador, mejorando la velocidad de procesamiento, así como la eficiencia.

Por las restricciones del número de bits que se puede incluir dentro del código de una instrucción, el número de registros de propósito general, normalmente, se limita a menos de ocho. Para identificarlos, se usan letras (A, B, C, D ...) o números (R0, R1, R2 ...).

1.3.2 La memoria de datos.

La MEMORIA DE DATOS es una memoria de lectura/escritura (RAM), cuyo objetivo es permitir el almacenamiento temporal de datos o programas de aplicación. Por sus características de lectura/escritura, la información que reside en ella se puede alterar fácilmente; sin embargo, ésta se pierde cuando se apaga la fuente de alimentación.

La memoria de datos se puede considerar como una extensión de los registros internos de propósito general, ya que cada una de sus localidades es, precisamente, un registro. La diferencia está en que los registros de la memoria de datos son externos al µprocesador y, por lo tanto, la velocidad de la transferencia de información es más lenta.

1.3.3 Los puertos de entrada/salida.

Los PUERTOS DE ENTRADA/SALIDA son los circuitos que se encargan de establecer el contacto del µprocesador (CPU) con el mundo exterior.

Los puertos se conectan a dispositivos periféricos que generan información para ser procesada por la CPU (dispositivos de entrada), o que aceptan datos provenientes del µprocesador, y los transforman en información significativa para el mundo exterior (dispositivos de salida).

Estos periféricos incluyen una gran variedad de elementos electrónicos y electromecánicos, cuya complejidad va, desde unos simples interruptores y lámparas indicadoras, hasta complicados sistemas de adquisición de datos, pantallas de video, etc.

1.3.4 Los buses de interconexión.

Un µprocesador se comunica con los otros circuitos del sistema, a través de grupos de líneas o buses de interconexión. Con base en el tipo de información que conducen, las líneas de interconexión se pueden agrupar en tres buses de: datos, direcciones y control.

El BUS DE DATOS es un conjunto de líneas bidireccionales, que transportan información del µprocesador hacia la memoria o puertos, y de éstos, hacia el µprocesador.

El BUS DE DIRECCIONES es unidireccional; es decir, por él solamente circula información proveniente del µprocesador. Comprende a las líneas que transmiten una dirección generada por la CPU, la cual selecciona a un puerto o a una localidad de memoria. Esta dirección especifica el origen o destino de la información que transitará por el bus de datos.

La sincronización, el sentido de la transferencia de información en el bus de datos (hacia adentro o hacia afuera del µprocesador), y el tipo de transferencia, se indican por medio de

señales de control originadas en la CPU; todas ellas conforman el llamado BUS DE CONTROL. Cada una de las señales en el bus de control es unidireccional, y algunas de ellas son salidas del µprocesador, mientras otras son entradas a él.

Además de los buses de interconexión externos, existen otros, dentro del µprocesador, que sirven para comunicar entre sí a la ALU, los registros internos, y la unidad de control. A estos se les denomina BUSES INTERNOS.

1.4 Clasificación de los µprocesadores.

Existen dos criterios principales para la clasificación de los µprocesadores: uno se basa en la longitud de palabra, y el otro, en la tecnología de fabricación.

La longitud de palabra se refiere al número de bits que puede procesar simultáneamente un µprocesador, y está determinada por su arquitectura; es decir, por el tamaño de los registros, de la ALU, y de los buses internos. Esta longitud ha ido creciendo a través de los años, desde los 4 bits del primer µprocesador, hasta los 32 bits de los µprocesadores más recientes.

Hoy en día, los µprocesadores de 4 bits se consideran obsoletos, mientras que los de 32 bits se reservan para aplicaciones muy complejas. En la generalidad de los casos, se utilizan µprocesadores de 8 bits y de 16 bits; los primeros son más usuales por haber sido más temprana su aparición, y porque tienen disponible un amplio soporte de programación y circuitería.

En lo que toca a las tecnologías de fabricación, los primeros µprocesadores se implantaron con tecnología PMOS; sin embargo, actualmente, la tecnología de fabricación de µprocesadores más difundida es la NMOS.

1.4.1 Los primeros µprocesadores.

En 1971, la compañía Intel anunció la aparición del primer µprocesador, dénominado el 4004. Este era de 4 bits, estaba implantando con tecnología PMOS, tenía 45 instrucciones, y ejecutaba 60,000 operaciones por segundo.

Al siguiente año, la misma compañía introdujo el 8008, que fue el primer µprocesador de 8 bits; también estaba implantando con tecnología PMOS. El 8008, además de tener una longitud de palabra mayor, contaba con 48 instrucciones, podía ejecutar 300,000 operaciones por segundo, y direccionaba 16 Kbytes de memoria. Sin embargo, para poder funcionar, requería casi 20 circuitos de soporte.

Hasta ese momento, el principal objetivo de los µprocesadores era reemplazar compuertas SSI (Small Scale Integration) y MSI (Medium Scale Integration).

1.4.2 El μprocesador 8080.

Avances posteriores en la tecnología de circuitos integrados permitieron que, a principios de 1974, Intel anunciara el 8080, un μprocesador de 8 bits mucho más poderoso. El 8080 tenía 78 instrucciones, en las cuales se incluían todas las del 8008. Además, su velocidad de operación era diez veces mayor que la del 8008, y podía direccionar hasta 64 kbytes de memoria. La tecnología de fabricación usada fue la NMOS, y gran parte de la lógica de soporte se incluyó en el mismo circuito del μprocesador; gracias a esto, era posible construir un sistema con sólo seis circuitos integrados.

Pero sobre todo, la principal diferencia del 8080 con respecto a los μprocesadores anteriores, era que no había sido diseñado simplemente para sustituir compuertas lógicas, sino que se le había dotado con la capacidad de una computadora. Esto originó una revolución tecnológica en el campo de la electrónica digital, a la que todavía no se le ve fin. Hasta la fecha, el 8080 ha sido el μprocesador más popular, convirtiéndose en un estándar de la industria.

1.4.3 La familia Motorola.

En respuesta al éxito del 8080, la compañía Motorola introdujo, también en 1974, un μprocesador de 8 bits con 72 instrucciones, el 6800. Al mismo tiempo, apareció una familia de circuitos periféricos, diseñados especialmente para conectarse al μprocesador.

En 1975, la compañía Mos Technology anunció dos μprocesadores: el 6501, que era compatible pata a pata con el 6800; y el 6502, cuyo circuito integrado incluía, además de un 6501, toda la circuitería para generar la señal de reloj. Hasta entonces, la señal de reloj se había generado en circuitos externos al μprocesador.

1.4.4 Los μprocesadores Z-80 y 8085.

En 1976, la compañía Zilog introdujo el Z-80: un μprocesador NMOS de 8 bits basado en el 8080, pero apreciablemente mejorado, tanto en software, como en hardware.

El Z-80 resultó ser un μprocesador mucho más rápido y fácil de usar, ya que sólo requería una fuente de alimentación de 5 volts, y toda la circuitería de soporte estaba incluída en el circuito integrado. Su código de instrucciones contenía las 78 del 8080, lo que le permitía ejecutar todos los programas escritos para este último, así como 80 instrucciones más; en total, 158.

Junto con el μprocesador Z-80 (Z80 CPU), Zilog introdujo varios circuitos periféricos, tales como el controlador de puertos en paralelo (Z80 PIO), el controlador de puertos en serie

(Z80 SIO), y el circuito de reloj contador (Z80 CTC). Todos ellos quedaron agrupados dentro de la familia Z-80.

En 1977, Intel anunció el μprocesador 8085, el cual combinaba el 8080, el circuito de reloj, y el controlador del sistema, en un solo circuito integrado. Al igual que el Z-80, el 8085 estaba fabricado con tecnología NMOS, y requería un voltaje único de 5 volts.

El 8085 se optimizó para que pudiera formar un sistema completo, utilizando dos circuitos periféricos especiales: uno de ellos con memoria RAM, puertos de entrada/salida, y un circuito de reloj (8155 u 8156); y el otro con memoria ROM o EPROM, y puertos (8355 u 8755). Sin embargo, en programación, su aportación fue prácticamente nula, ya que sólo aumentó en dos las instrucciones del 8080.

1.4.5 Los μprocesadores de 16, 32 y 64 bits.

La década de los 80's representó para los μprocesadores el inicio de una revolucionaria época. La carrera se inició con la aparición de los μprocesadores de 32 bits, abanderados por el Intel 80386, y que representó un notable avance frente a los 8088, 8086 y 80286 de 16 bits. Su capacidad de direccionamiento de memoria de 4 Gigabytes representó 256 veces la capacidad del 80286, y 4096 veces la del 8088, aunado a la capacidad de soportar sistemas que ejecutan varias tareas a la vez, superando la capacidad del 80286.

Esto fue sólo el inicio, ya que al 386 le precedieron el 486 y el Pentium; de este último, Intel presentó a principios de 1995 la versión 6. Cabe hacer mención que la primera versión tuvo algunos problemas cuando se le exigían numerosos procesos de cálculo. Intel hizo una campaña para que el que haya adquirido tal versión, pueda sustituirla por la actual, sin cargo alguno.

La aplicación de los μprocesadores en el ámbito industrial y doméstico, ha sido el impulsor de su desarrollo. De tal forma, cabe mencionar que, paralelamente al desarrollo del μprocesador como pieza única, se fueron diseñando modelos que incorporaran en su encapsulado memorias y puertos de entrada/salida (analógicos y digitales), conocidos como μcontroladores. Actualmente, podemos ver a los μprocesadores trabajando conjuntamente con los μcontroladores, o cada uno por su cuenta, en sistemas digitales electrónicos, tales como:

- Controles lógicos programables (PLC'S).
- Controladores de procesos y temperatura.
- Controladores de sensores.
- Sistemas de variación de velocidad de motores.
- Servocontroles.
- Aparatos electrodomésticos.
- Sistemas de comunicación satelital y radiofrecuencia.
- y muchos otros.

CAPITULO 2

ENTORNO DE LOS MICROPROCESADORES.

El µprocesador necesita estar conectado con circuitos de apoyo para realizar una función, y ésta es físicamente aprovechable por medio de sistemas ejecutantes o sensibles al ser humano. El funcionamiento del µprocesador no se aprecia físicamente, pero sí podemos ver sus resultados a través de los sistemas actuantes, gracias a los dispositivos de entrada/salida, los cuales son el medio de comunicación entre el µprocesador y los sistemas actuantes; estos últimos, a su vez, son el medio de comunicación entre el µprocesador y el ser humano, o entre el µprocesador y otros sistemas inteligentes o dependientes. De hecho, lo anterior es válido para cualquier controlador electrónico, eléctrico, pneumático ó mecánico. La fig. 2-1 muestra un diagrama de los conceptos mencionados.

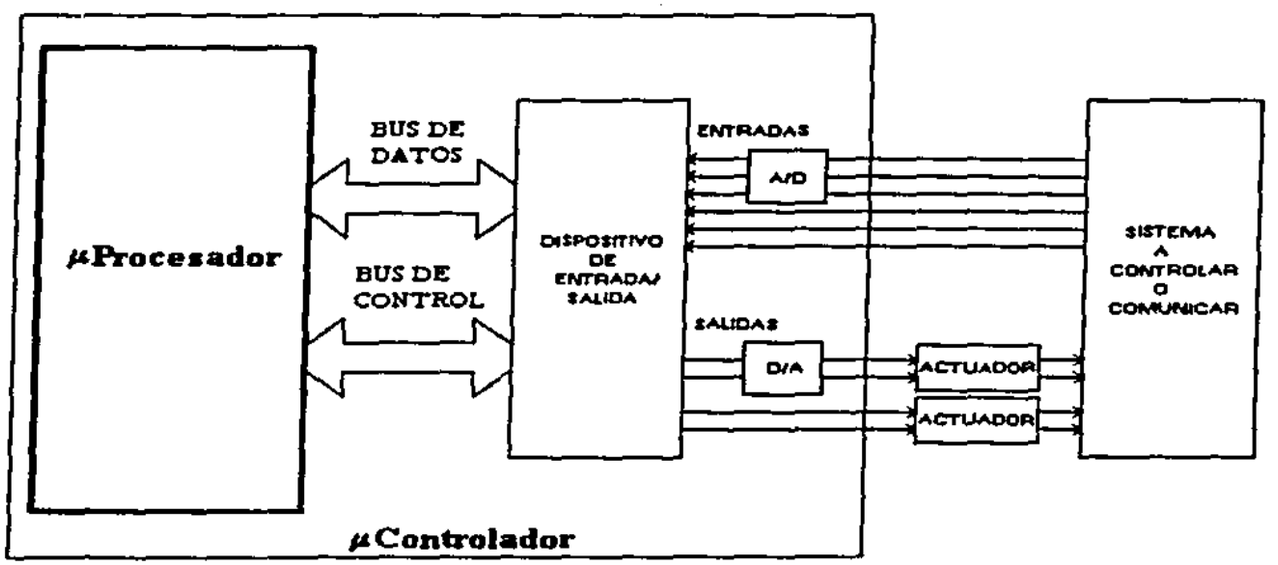


Figura 2-1

2.1 Periféricos.

Los periféricos son mecanismos externos que permiten al hombre interactuar con el µprocesador, y se dividen en dos grandes grupos: de entrada y de salida.

2.1.1 Periféricos de entrada.

Estos son los mecanismos que proveen información proveniente del exterior al µprocesador, y podemos clasificarlos en dos tipos: sensores y medios de información.

Los sensores son mecanismos que proporcionan información digital (on/off) o analógica (corriente o voltaje); entre estos encontramos:

- Botones.
- Interruptores mecánicos.
- Interruptores ópticos (fotoceldas).
- Interruptores de proximidad (inductivos y capacitivos).
- Interruptores de ultrasonido o láser.
- Convertidores analógico-digitales.

Con éstos, podríamos simular un teclado, un mouse, un lápiz óptico, un scanner, una alarma, etc.

Los medios de información proporcionan paquetes de datos, mediante una comunicación comercial específica (vía satélite, por radio frecuencia, telefónica, magnética, óptica, etc.), y pueden ser modems, sistema multimedia, o unidades de cinta y disco magnético.

2.1.2 Periféricos de salida.

Estos, como los de entrada, se clasifican en dos tipos: actuantes y medios de información.

Los periféricos actuantes son mecanismos que realizan una actividad física; estos pueden ser relevadores mecánicos o de estado sólido, focos, motores, válvulas, convertidores digitales-analógicos, etc.

Los medios de información permiten enviar paquetes de datos, utilizando también una comunicación comercial específica (vía satélite, por radio frecuencia, telefónica, magnética, óptica, etc.), y pueden ser modems, o unidades de cinta y disco magnético.

2.2 Memorias y dispositivos de Entrada/Salida.

La memoria permite al µprocesador almacenar información, ya sea permanente o temporalmente. Los dispositivos de entrada/salida permiten al µprocesador controlar y sensar los dispositivos periféricos. De hecho, para el µprocesador es lo mismo la memoria que los dispositivos de entrada/salida, pues igualmente lee o escribe en sus registros (direcciones físicas asignadas a unos u otros).

2.2.1 Memorias.

La memoria es un conjunto de registros para almacenamiento de información binaria. Una unidad de memoria es una colección de registros de almacenamiento, conjuntamente con los circuitos asociados necesarios para transferir la información hacia adentro y afuera de los registros.

El componente que forma las celdas binarias de los registros en una unidad de memoria debe tener ciertas propiedades básicas, de las cuales, las más importantes son las siguientes:

- Debe tener una propiedad dependiente de dos estados para la representación binaria.

- Debe ser pequeño en tamaño.

- El costo por bit de almacenamiento debe ser lo más bajo posible.

- El tiempo de acceso al registro de memoria debe ser razonablemente rápido.

Ejemplos de componentes de unidad de memoria son: los núcleos magnéticos, los C.I. semiconductores, las superficies magnéticas (cintas, tambores, discos) y las pistas de los discos compactos. En el presente trabajo enfocaremos nuestra atención a las unidades de memoria de C.I. semiconductores.

Una unidad de memoria almacena la información binaria en grupos llamados palabras, y cada palabra se almacena en un registro de memoria. Una palabra en la memoria es una entidad de *n* bits, los cuales se mueven hacia adentro y afuera del almacenamiento como una unidad. Una palabra de memoria puede representar un operando, una instrucción, o un grupo de caracteres alfanuméricos o cualquier otra información codificada en binario. La comunicación entre una unidad de memoria y lo que la rodea se logra por medio de dos señales de control y dos registros externos. Las señales de control especifican la dirección de la transferencia requerida: esto es, cuando una palabra debe ser acumulada en un registro de memoria, o cuando una palabra almacenada previamente debe ser transferida hacia afuera del registro de memoria. Un registro externo especifica el registro de memoria particular escogido entre los miles disponibles; el otro especifica la configuración de bits particular de la palabra en cuestión. Las señales de control y los registros se muestran en la fig. 2-2.

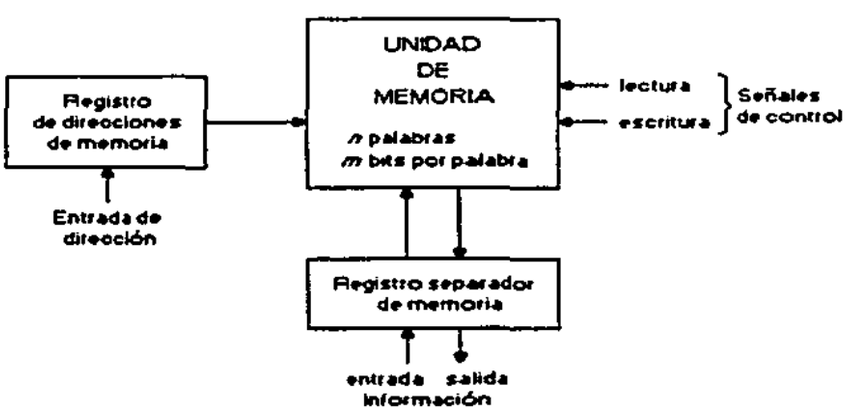


Figura 2-2

El registro de direcciones de memoria especifica la palabra de memoria seleccionada. A cada palabra en la memoria se le asigna un número disponible. Para comunicarse con una palabra de memoria específica, su número de localización o dirección se transfiere al registro de direcciones. Los circuitos internos de la unidad de memoria aceptan esta dirección del registro, y abren los caminos necesarios para buscar la palabra buscada. Un registro de dirección con n bits puede especificar hasta 2^n palabras de memoria. Las unidades de memoria del computador pueden contener un rango entre 1.024 palabras que necesitan un registro de direcciones de 10 bits, hasta 1'048.576 = 2^{20} palabras que necesitan un registro de direcciones de 20 bits.

Las dos señales de control aplicadas a la unidad de memoria se llaman de lectura y escritura. Una señal de escritura especifica una función transferencia entrante, mientras que una señal de lectura especifica una función de transferencia saliente; cada una es referenciada por la unidad de memoria. Después de aceptar una de las señales, los circuitos de control interno dentro de la unidad de memoria suministran la función deseada. En la actualidad, todas las unidades de memorias existentes en el mercado permiten leer la información de cualquier dirección sin destruir la información contenida, cosa que sucedía con algunos dispositivos antiguos.

A continuación se mencionan los tipos de memoria de C.I. semiconductores.

2.2.1.1 Memoria ROM.

Este tipo de memoria permite una sola grabación de información. Dicha grabación es especial y, en condiciones normales, el sistema de control al cual pertenezca no tendrá acceso a escritura, únicamente a lectura. Una vez grabada, no existe ningún medio para regrabarla o borrarla, ni aun cortando el suministro de energía. Es usada para que resida en ella el programa de control del µprocesador, una vez que éste haya sido depurado y pasado por exhaustivas pruebas de funcionamiento. Sirve también para almacenar tablas de información invariantes con el tiempo, tales como información de generación de caracteres para video, valores de configuración para comunicaciones, etc. Su aplicación principal es en la producción en serie de un sistema de control, por su bajo costo.

2.2.1.2 Memoria EPROM.

Esta permite grabar una y otra vez, pero se debe borrar antes de cada grabación mediante la exposición a luz ultravioleta. Esto es mediante una ventanilla que permite el paso de dicha luz. Igual que en la anterior, el sistema de control al cual pertenezca tendrá únicamente acceso a la lectura y no pierde la información al cortar la energía. Se usa principalmente en la etapa de depuración del programa del µprocesador.

2.2.1.3 Memoria EEPROM.

Este tipo de memoria permite grabar una y otra vez sobre sus registros, no pierde la información con el corte de energía, y el µprocesador tiene acceso a lectura y escritura de la memoria. Puede almacenar programas, tablas y datos temporales o fijos. Se utiliza en equipos que requieren conservar información aun cuando se pierda la energía. Es un C.I. muy caro.

2.2.1.4 Memoria RAM.

Funciona igual que la anterior, con la gran diferencia de que, al cortar la energía, pierde la información. Sus ventajas radican en que es mucho más rapida en los tiempos de acceso a escritura, y es mucho más barata.

2.2.2 Dispositivos de entrada/salida.

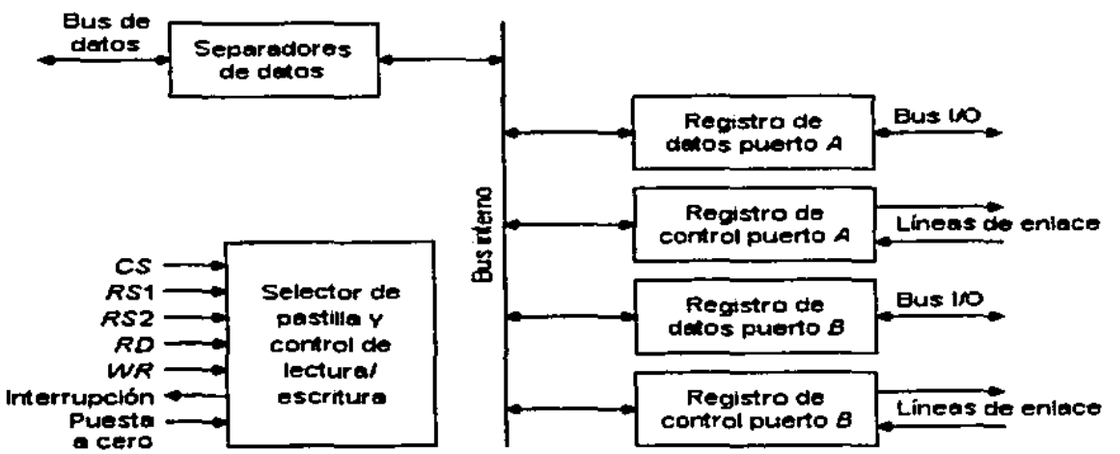
Un C.I. semiconductor de entrada/salida es un componente LSI que provee el enlace de interconexión entre el µprocesador y un periférico. Cuando está en el modo de salida de datos, la interconexión recibe información binaria del bus de datos al ritmo y modo de transferencia del µprocesador, y la transmite a un periférico al ritmo y modo de transferencia del periférico. El dispositivo se comporta de manera similar en el modo de entrada de datos, excepto que la dirección de transferencia está en el sentido opuesto. Un dispositivo I/O consiste de un número de registros, lógica de selección y circuitos de control que configuran las transferencias requeridas. La mayoría de los dispositivos de I/O pueden ser programados para acomodar una variedad de combinaciones de modos de operación. El µprocesador, por medio de instrucciones de programa, transfiere un byte a un registro de control dentro del dispositivo de I/O. Esta información de control coloca al dispositivo en uno de los modos disponibles para un periférico en particular, al cual está unido. Cambiando el byte de control, es posible cambiar las características del dispositivo I/O. Por esto, a menudo las unidades I/O se llaman "dispositivos programables de entrada/salida". Las instrucciones que transfieren la información de control al dispositivo I/O son incluidas en un programa, y pueden iniciar la interconexión para un modo particular de operación.

Los dispositivos I/O se diseñan usualmente para operar con un bus del sistema del µprocesador particular, sin ninguna lógica adicional diferente de la decodificación de direcciones. Hay una variedad de dispositivos I/O de uso comercial, y cada uno puede ser clasificado en una de las cuatro categorías.

2.2.2.1 Interconexión de comunicación en paralelo.

Transfiere datos entre el µprocesador y el periférico. A continuación se presenta una explicación exhaustiva de este tipo de interconexión, debido a que es la que se ocupa en el presente trabajo. Las posteriores interconexiones solo se explicarán brevemente.

Una interconexión periférica en paralelo, es un componente LSI que presenta un camino para transferir información binaria en paralelo entre el µprocesador y el dispositivo periférico. Una pastilla de interconexión contiene normalmente dos o más puertos I/O que se comunican con uno o más dispositivos externos, y una interconexión sencilla para comunicarse con el sistema del bus del µprocesador. El diagrama de bloque de una interconexión periférica típica en paralelo se muestra en la figura 2-3. Este consiste de dos puertos; cada uno de ellos tiene dos registros, un bus I/O de 8 bits y un par de líneas denominadas de *enlace*. La operación almacenada en el registro de control especifica el modo de operación del puerto. El puerto del registro de datos se usa para transferir datos al bus de datos y al bus I/O, y viceversa.



CS	*RS1*	*RS2*	Registro seleccionado
0	*X*	*X*	Ninguno — El bus de datos en alta impedancia
1	0	0	Registro de datos puerto *A*
1	0	1	Registro de control puerto *A*
1	1	0	Registro de datos puerto *B*
1	1	1	Registro de control puerto *B*

Figura 2-3

La interconexión se comunica con el µprocesador a través del bus de datos, el selector de pastilla y el control de lectura/escritura. Se debe agregar un circuito externo (usualmente

una compuerta AND) para detectar la dirección asignada a la interconexión. Este circuito habilita la entrada de selección de la pastilla cuando se selecciona la interconexión por medio del bus de direcciones. Las dos entradas de selección del registro *RS1* y *RS2* se conectan usualmente a las líneas de menor orden del bus de direcciones. Estas dos entradas seleccionan uno de los cuatro registros en la interconexión, como se explica en la tabla que acompaña el diagrama. El contenido del registro selector se traslada al µprocesador por medio del bus de datos cuando se habilita la entrada *RD*. El µprocesador carga un byte al registro seleccionado por medio del bus de datos cuando se habilita la entrada *WR*. La salida de interrupción se usa para interrumpir al µprocesador, y la entrada de reposición es para poner a cero la interconexión, una vez que se suministre potencia.

El µprocesador inicia cada puerto, transfiriendo un byte a su registro de control. Al cargar los bits adecuados a un registro de control en la iniciación del sistema, el programa puede definir el modo de operación del puerto. Las características del puerto dependen de las unidades comerciales usadas. En la mayoría de los casos, cada puerto puede ser llevado a un modo de entrada o salida. Esto se hace al transferir los bits en el registro de control que especifican la dirección de transferencia en los separadores del bus que accionan al bus I/O bidireccional. En adición, el puerto puede hacerse funcionar en una variedad de modos de operación. Los tres modos de operación encontrados en la mayoría de las pastillas de interconexión son:

- Transferencia directa sin línea de enlace.

- Transferencia con enlace.

- Transferencia con enlace usando interrupción.

Una interconexión se lleva al modo de transferencia directa cuando el dispositivo conectado al bus I/O está siempre listo para transferir información. Las líneas de enlace no se usan en este modo, y algunas pastillas de interconexión tienen un modo de programación para convertir a estas líneas, en líneas de transferencia de datos. La transferencia directa puede operar en un modo de entrada o salida. En el modo de entrada, una operación de lectura transfiere el contenido del bus I/O al bus de datos del µprocesador. En el modo de salida, una operación de escritura transfiere el contenido del bus de datos al registro de datos del puerto seleccionado. El byte recibido se aplica entonces al bus I/O. Las transferencias de entrada o salida directas son útiles solamente si los datos valederos pueden residir en el bus I/O por un tiempo largo, comparado con el tiempo de ejecución de la instrucción en el µprocesador. Si los datos I/O pueden ser valederos por un corto tiempo, la interconexión debe operar en el modo de enlace.

Las líneas de enlace son usadas para controlar la transferencia entre dos dispositivos que operan asíncronamente entre sí; es decir, cuando no comparten un reloj común. El enlace es un proceso usado comúnmente, y no está restringido para hacer interconexión con pastillas

solamente. Dos líneas de enlace, conectadas entre un dispositivo fuente y uno de destino, controlan las transferencias informándose entre sí de la condición de la transferencia por medio del bus común. El dispositivo fuente informa el destino por medio de una de las líneas de enlace cuando se tiene información valedera en el bus. El dispositivo de destino responde inhabilitando la segunda línea del enlace cuando ha sido aceptada la información del bus. La figura 2-3 muestra dos líneas de enlace en cada puerto; una es una línea de salida, y la otra es de entrada. Es costumbre referirse a estas líneas con símbolos, pero los símbolos adoptados son siempre distintos en las diferentes unidades comerciales. Debido a la variedad de símbolos usados para designar esas líneas, se prefiere no adoptar un símbolo sobre otro, sino referirse a las dos líneas como la línea de enlace de salida o entrada. La línea de enlace de entrada pondría a uno un bit en el registro de control dentro de la interconexión. Este bit será llamado indicador, teniendo en cuenta que el registro que retiene el bit indicador (el registro de control en este caso) puede ser leído por el µprocesador para comprobar la condición de la transferencia. El bit indicador se borra automáticamente en la interconexión después de una operación de lectura o escritura asociada con el correspondiente registro de datos.

La secuencia de enlace detallada para una pastilla comercial de interconexión se especifica con el diagrama de tiempo que acompaña a las especificaciones del producto. Debido a la variedad de procedimientos que se encuentran en la práctica, sería mejor explicar el método de enlace en términos generales, sin preferencia por un método específico. La transferencia con enlace depende de que si el puerto está en el modo de entrada o salida de información.

En el modo de enlace de salida, el µprocesador escribe un byte en el registro de datos del puerto de interconexión. La interconexión habilita la línea de enlace de salida para informar al dispositivo externo que un byte valedero está disponible en el bus I/O. Cuando el dispositivo externo acepta al byte del bus I/O, éste habilita la línea de enlace de entrada. Ello pone a uno al bit indicador en el registro de control. El µprocesador lee el registro que contiene el bit indicador para determinar si la transferencia fue completa. Si es así, el µprocesador puede escribir un nuevo byte al registro de datos del puerto de interconexión. Al escribir datos en un puerto dado, se borra automáticamente el bit indicador asociado con la transferencia de salida. El proceso puede repetirse para dar salida al siguiente byte.

En el modo de enlace de entrada, el dispositivo externo coloca un byte en el bus I/O y habilita la línea de enlace de la entrada de interconexión. La interconexión transfiere el byte a su registro de datos, y pone a uno al bit indicador en el registro de control. El µprocesador lee el registro que contiene el bit indicador para determinar si se requiere una transferencia de entrada. Si se pone a uno al bit indicador, el µprocesador lee el byte del registro de datos del puerto, y borra el bit indicador. La interconexión informa entonces al dispositivo conectado al bus I/O, a través de la línea de enlace de salida, que el nuevo byte puede ser aceptado. Una vez que el dispositivo de salida ha sido informado de que la interconexión está lista, puede iniciar la transferencia del siguiente byte, habilitando de nuevo el enlace de entrada.

En el método de enlace anteriormente descrito, el µprocesador debe leer periódicamente el registro de control para comprobar la condición del bit indicador. Si hay un número de puertos conectados al µprocesador, sería necesario hacerles un muestreo en sucesión para determinar aquellos que requieren una transferencia. Esta es una operación que consume tiempo, y que puede ser evitada si se inicia la interconexión para que opere en el modo de interrupción. La salida de interrupción mostrada en la figura 2-3 se usa entonces para solicitar una interrupción del µprocesador. La mayoría de las unidades comerciales presentan una línea de interrupción separada para cada puerto en la interconexión. Cada vez que se pone a uno un indicador en el puerto, la petición de interconexión que pertenece al puerto se habilita automáticamente para informar al µprogramador que se va a iniciar la transferencia. El µprocesador responde a la señal de interrupción del puerto que solicitó la acción, transfiere el byte de datos al registro de datos del puerto de interconexión, y viceversa.

2.2.2.2 Interconexión de comunicación serial.

Convierte los datos en paralelo del µprocesador a datos en serie para la transmisión, y convierte los datos en serie entrantes a datos paralelo para ser recibidos por el µprocesador.

2.2.2.3 Interconexión dedicada especial.

Es construida para comunicarse con un periférico particular de entrada y salida, o puede ser programada para operar con un periférico particular.

2.2.2.4 Interconexión de acceso directo a memoria (DMA).

Se usa para transferir datos, directamente entre un dispositivo externo y la memoria.

2.3 Decodificadores.

Cantidades discretas de información se presentan en sistemas digitales con códigos binarios. Un código binario de n bits es capaz de representar hasta 2^n elementos diferentes de información codificada. Un decodificador es un circuito combinacional que convierte la información binaria de n líneas de entrada a un máximo de 2^n líneas únicas de salida. Si la información decodificada de n bits tiene combinaciones no usadas o de no importa, la salida del decodificador tendrá menos de 2^n salidas.

Los decodificadores, como el del siguiente ejemplo, se llaman decodificadores en línea n a m, en donde $m <= 2^n$. Su propósito es generar 2^n (o menos) términos mínimos de n variables de entrada. El nombre decodificador se usa conjuntamente con cierto tipo de convertidores de código, tales como el decodificador BCD a decimal, o el BCD a siete segmentos.

Considérese el decodificador en línea de 3 a 8 de la fig. 2-4, y cuya tabla de verdad se muestra a continuación.

Entradas			Salidas							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Las tres entradas se decodifican en ocho salidas, y cada salida representa uno de los términos mínimos de las variables de 3 entradas. Los tres inversores generan el complemento de las entradas, y cada una de las ocho compuertas AND generan uno de los términos mínimos. Una aplicación particular de este decodificador sería una conversión binaria a octal. Las variables de entrada podrían representar un número binario, y las ocho salidas representarían los ocho dígitos en el sistema de numeración octal. Sin embargo, un decodificador en línea de 3 a 8 puede ser usado para decodificar cualquier código de 3 bits para generar ocho salidas, una para cada elemento del código.

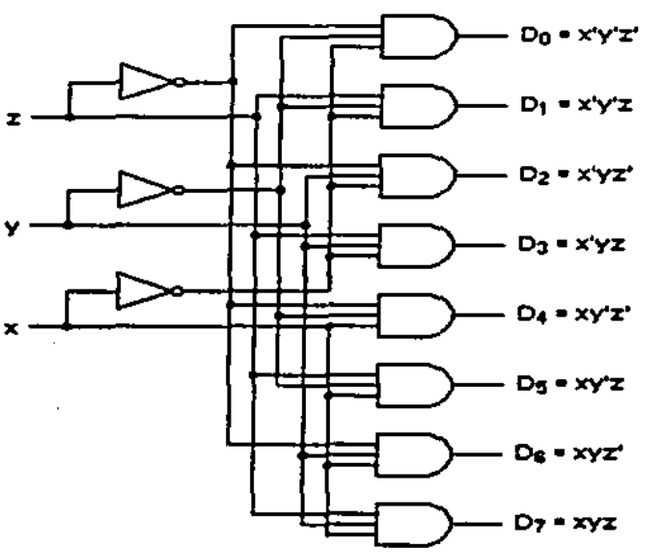


Figura 2-4

CAPITULO 3

EL MICROPROCESADOR Z-80A.

Para programar a un µprocesador, suponiendo que se cuenta con una computadora y con las herramientas auxiliares de programación que permitan comunicarse con él, únicamente es necesario conocer a un nivel aceptable el código de instrucciones y la estructura interna del µprocesador. Sin embargo, cuando se trata de utilizar al µprocesador como elemento central en la implantación de un sistema digital, por ejemplo una µcomputadora, se vuelve indispensable el conocimiento a fondo, no sólo de los aspectos anteriores, sino también de las características de las señales eléctricas y de la función de cada una de las terminales del circuito integrado, a fin de poder interconectar al µprocesador con los otros componentes del sistema.

En este capítulo se presentará una descripción del circuito integrado, que constituye el Z-80A, sus señales principales y la función que desempeña cada una de ellas en las tareas que realiza el µprocesador.

3.1 Descripción del circuito integrado.

El circuito integrado del µprocesador Z-80A está colocado en un encapsulado DIP de 40 terminales. En la figura 3-1a se encuentra su configuración.

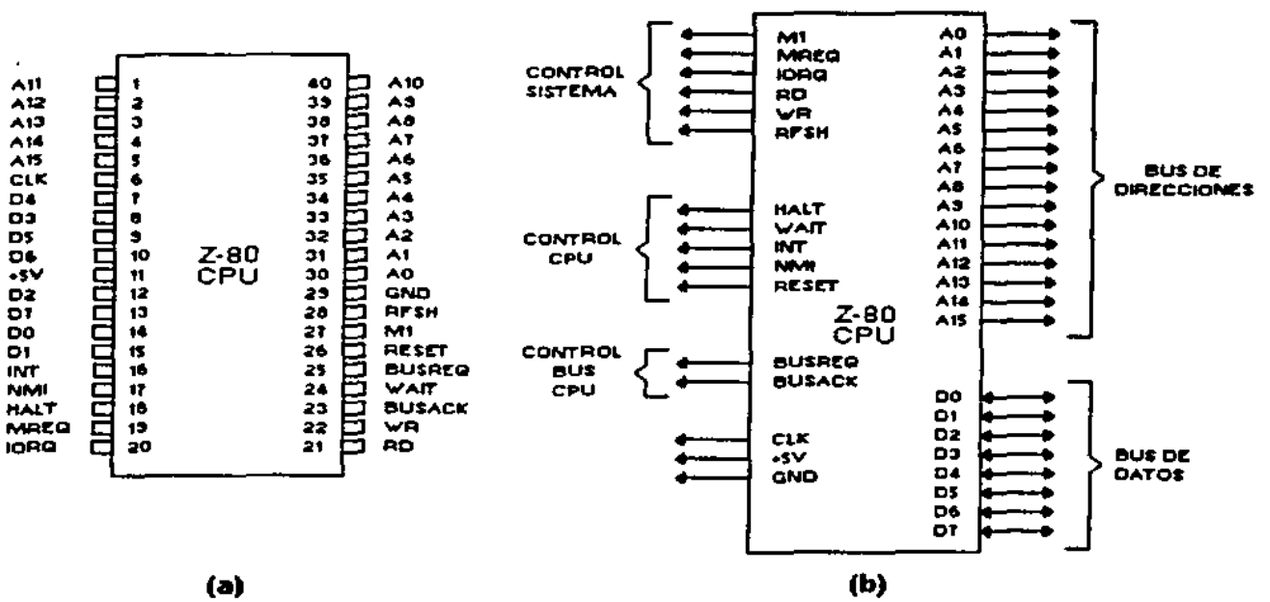


Figura 3-1

De las 40 terminales del circuito, 5 son únicamente de entrada, 24 son únicamente de salida y 8 son bidireccionales. Además, hay dos que sirven para conectar la fuente de alimentación, y otra que es la entrada de reloj.

Todas ellas tienen características eléctricas compatibles con la familia TTL: el intervalo del voltaje de entrada para 0 lógico (VIL) es de -0.3 V a 0.8 V; el intervalo del voltaje de entrada para 1 lógico (VIH) es de 2.0 V a VCC; el voltaje máximo de salida para 0 lógico (VOL) es de 0.4 V, y el voltaje mínimo de salida para 1 lógico (VOH) es de 2.4 V. La corriente de salida para 0 lógico (IOL) es de 1.8 mA, y la corriente de salida para 1 lógico (IOH) es de -250μA (se supone positiva la corriente en una terminal del CI si ésta entra).

3.2 La arquitectura del microprocesador Z-80A.

La figura 3-2 es un diagrama de bloques de la arquitectura interna del Z-80A. El Z-80A posee más registros internos y formas de direccionamiento que la mayoría de los otros μprocesadores de 8 bits; estas características le permiten procesar una mayor cantidad de información en un tiempo dado.

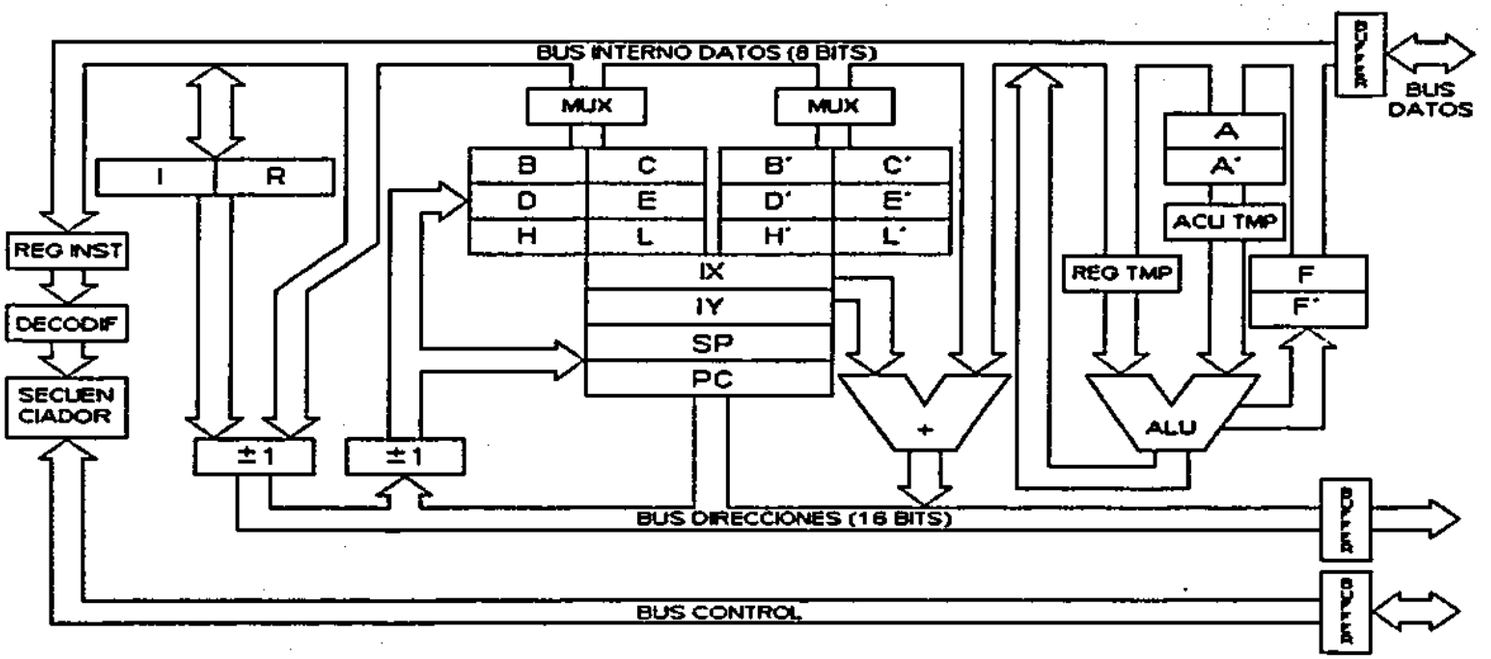


Figura 3-2

3.2.1 Los registros internos.

La CPU Z-80A contiene 280 bits de memoria de lectura/escritura, dividida en 18 registros de 8 bits y 4 registros de 16 bits, como se muestra en la figura 3-3.

Los registros incluyen dos conjuntos o "bancos" de seis registros de propósito general, que pueden usarse individualmente como registros de 8 bits, o en pares, como registros de 16 bits. El banco principal consiste en los registros BC, DE y HL, en tanto que el banco alterno comprende los registros BC', DE' y HL', (fig. 3-3). También hay dos acumuladores (A y A') y dos registros de banderas (F y F'). Sin embargo, sólo es posible trabajar a un tiempo con uno de los dos bancos de registros.

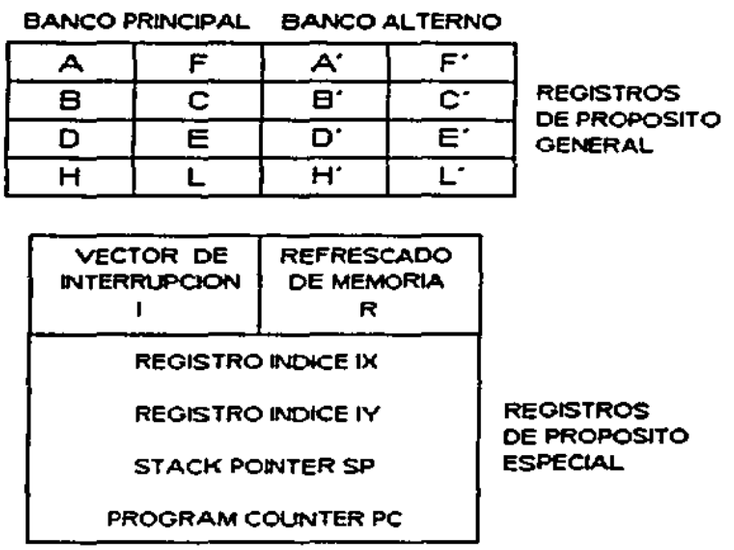


Figura 3-3

En los registros de uso específico, se encuentran: el contador del programa (Program Counter), el apuntador de pila (Stack Pointer), los registros índices (Index Registers), el registro de refrescado (Refresh Register), y el registro del vector de interrupción (Interrupt Register). Cada uno de ellos tiene funciones especiales que se describirán en el desarrollo del tema, conforme vaya siendo necesaria su utilización.

El contador del programa (PC), es un registro que guarda los 16 bits de la dirección de la instrucción que en ese momento se está obteniendo de la memoria. Siendo el PC un registro de 16 bits, es capaz de direccionar hasta $2^{16} = 65,536$ localidades de memoria. El PC se incrementa automáticamente, después que su contenido ha sido transferido a las líneas de dirección. Cuando ocurre un salto en el programa, el nuevo valor se coloca automáticamente en el PC, sustituyendo el valor resultante del incremento normal.

3.2.2 La unidad aritmético/lógica.

La unidad aritmético/lógica es de 8 bits, y ejecuta las instrucciones aritméticas y lógicas de la CPU. Internamente, la ALU se comunica con los registros y el bus externo de datos a través del bus interno de datos. El tipo de funciones ejecutadas por la ALU incluye:

Suma	Rotaciones y desplazamientos hacia la derecha o izquierda (aritméticos o lógicos).
Resta	Incrementar.
AND	Decrementar.
OR	Poner un bit en uno (set).
XOR	Poner un bit en cero (reset).
Comparación	Probar el estado de un bit (test).

3.2.3 La unidad de control.

Después de obtener una instrucción de la memoria, se coloca en el registro de instrucciones, y se decodifica. La unidad de control lleva a cabo esta función, y luego genera todas las señales de control para leer o escribir datos en los registros, controlar la ALU, y emitir las señales de control externas que se requieran.

El Z-80A opera con una fuente de alimentación de 5 V conectada a la terminal de +5 V. La tierra de la fuente se conecta a la terminal GND. Su consumo común de corriente es de 90 mA. Con base en la función que realizan, las demás terminales del circuito integrado se agrupan en tres buses: el bus de direcciones, el bus de datos y el bus de control (fig. 3-1b).

3.2.4 El bus de datos.

Las ocho terminales designadas como D0-D7 constituyen el bus de datos; todas son bidireccionales, con tercer estado y se activan en 1 lógico. El bus de datos se usa para la transferencia de datos, en grupos de 8 bits con la memoria o con dispositivos de entrada/salida; D0 transfiere el bit menos significativo y D7, el bit más significativo.

3.2.5 El bus de direcciones.

El bus de direcciones lo forman las terminales A0-A15; todas ellas son de salida, con tercer estado y activas en 1 lógico. Con las 16 líneas de direcciones, el Z-80A puede seleccionar una localidad dentro de 64 KB de memoria. Además, las 8 líneas menos significativas del

bus (A0-A7), permiten direccionamiento de hasta 256 puertos de entrada/salida. Finalmente, el bus de direcciones también se usa en la especificación de las direcciones para el refrescado de memorias dinámicas; durante el proceso de refrescado, las 7 líneas menos significativas (A0-A6) indican la dirección de refrescado.

3.2.6 El bus de control.

El bus de control incluye una amplia variedad de líneas, de las cuales, unas son entradas y otras son salidas. El bus de control se puede dividir en varios grupos de líneas relacionadas. El primer grupo se encarga del control de la CPU (CPU Control), y lo forman las terminales RESET, HALT, WAIT y NMI. El segundo grupo tiene como función el control del sistema (System Control); comprende las terminales M1, MREQ, IORQ, RD, WR y RFSH. El tercer grupo sirve para controlar el bus del μprocesador (CPU Bus Control); está constituido por las terminales BUSRQ y BUSAK.

3.2.7 La entrada ϕ y el generador de reloj.

El μprocesador Z-80A es un circuito que no puede funcionar solo. En primer lugar, necesita una señal de reloj conectada a la terminal de entrada del circuito integrado, para sincronizar sus acciones.

Las características eléctricas que debe tener la señal de reloj son: oscilación de una fase con niveles TTL (-0.3 a 0.45 V para 0 lógico, y Vcc-0.6 a Vcc+0.3 V para 1 lógico, fig. 3-4).

Figura 3-4

Una forma de satisfacer todos los requerimientos de voltajes, es por medio de una resistencia de activación (pull-up) de 330 ohms, conectada entre Vcc y la terminal de salida de un oscilador implantado con circuitos TTL, que genera la señal de reloj ϕ (fig. 3-5).

Algo que no es claro a primera vista es que el Z-80A, lo mismo que la mayoría de los μprocesadores, no tiene una frecuencia de operación única. En lugar de eso, tiene un intervalo de frecuencias y una frecuencia máxima recomendada por el fabricante. Arriba de esta frecuencia, la confiabilidad del sistema se deteriora. Mientras más alta sea la frecuencia de operación, el μprocesador tardará menos en ejecutar una instrucción y, por lo tanto, su capacidad de procesamiento será mayor.

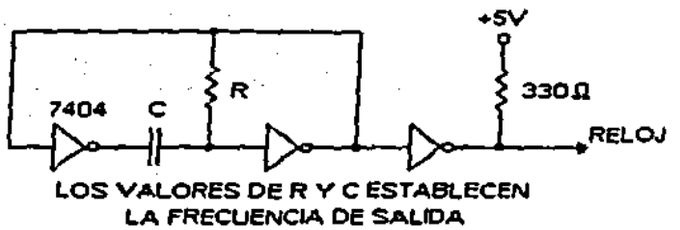


Figura 3-5

Existen varias versiones del μprocesador Z-80, cada una de las cuales tiene una frecuencia máxima de operación diferente. La versión original del Z-80 apareció con una frecuencia máxima de 2.5 MHz, y ya se considera obsoleta. El modelo más utilizado actualmente es el Z-80A, con velocidad de operación de 4 MHz.

Cualquier oscilador que cumpla con las especificaciones de la entrada ϕ puede servir para generar la señal de reloj del Z-80A. Si se va a operar al μprocesador debajo de la frecuencia máxima, y no se necesita que el valor de la frecuencia de oscilación sea muy preciso o estable, se puede utilizar un oscilador RC, como el de la figura 3-5. Este circuito es muy útil en las etapas de desarrollo y prueba de un sistema, porque siendo un oscilador de frecuencia variable, permite al diseñador disminuir el ritmo de operación del sistema (incrementando los valores de R y C). Sin embargo, no se recomienda cuando el μprocesador se use en aplicaciones en las que se requiera un conteo preciso de tiempo.

Para frecuencias cercanas al valor máximo, se requiere un oscilador controlado por un cristal de cuarzo, como el circuito de la figura 3-6. En este caso, la frecuencia de oscilación del cristal determina la frecuencia de la señal de reloj y, gracias a la estabilidad del cristal, se puede mantener un tiempo de ejecución constante. Cuando no se tiene un cristal cuya frecuencia sea la deseada, se puede utilizar uno con una frecuencia de oscilación más alta y después dividir la señal de salida del circuito oscilador por medio de flip-flops o contadores, hasta obtener el valor requerido.

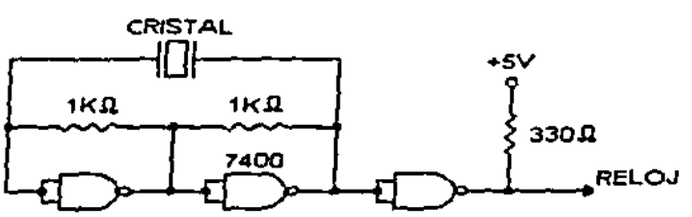


Figura 3-6

Nótese que los dos circuitos de reloj mostrados (figs. 3-5 y 3-6), tienen a la salida una resistencia de activación (pull-up) de 330 ohms para satisfacer los requerimientos, tanto de AC, como de DC. A fin de no afectar las características del oscilador, es mejor utilizar una compuerta separada que maneje la sección de activación.

3.2.8 La entrada de restablecimiento.

Aunque frecuentemente es ignorada, la función de restablecimiento (reset) es uno de los controles más necesarios de un µprocesador. Cuando se enciende por primera vez la fuente de alimentación de un µprocesador, los registros y flip-flops internos adoptan valores aleatorios, y la operación del circuito resulta impredecible. Por lo tanto, el µprocesador debe restablecerse para colocarlo en un estado inicial conocido.

Otra situación se presenta cuando se quiere detener la ejecución de un programa que no está corriendo correctamente. Al restablecer el circuito, el µprocesador regresa a la dirección más baja de memoria, lo que permite al diseñador reiniciar la ejecución del programa.

El Z-80A cuenta con la terminal de entrada llamada RESET, la cual, cuando recibe un pulso a 0 lógico, obliga al µprocesador adoptar un estado inicial conocido. En concreto, los efectos producidos por el pulso de restablecimiento son los siguientes:

- El contador del programa se carga con ceros: PC = 0000H.
- El flip-flop de interrupciones se pone en cero: IFF = 0.
- El registro del vector de interrupciones se carga con ceros: I = 00H.
- El registro de refrescado se carga con ceros: R = 00H.
- El modo de interrupción 0 se establece.

Así, en esencia, el restablecimiento del µprocesador equivale a un salto incondicional a la dirección 0000H producido por hardware. Además, mientras RESET está en 0 lógico, tanto el bus de datos como el bus de direcciones, se ponen en alta impedancia, y todas las líneas de salida del bus de control pasan a su estado inactivo.

La señal de RESET se puede generar manualmente, automáticamente, o por medio de un circuito que incluya las dos formas (power on reset), como el mostrado en la figura 3-7.

Cuando la fuente de alimentación del sistema se enciende, el capacitor se encuentra completamente descargado. El nivel 0 lógico en la entrada RESET debe mantenerse durante el tiempo suficiente, para que el nivel de voltaje de la fuente de alimentación se estabilice dentro del intervalo válido para Vcc. Este tiempo se obtiene de la constante de carga del capacitor (RC), determinada por los valores de R y C.

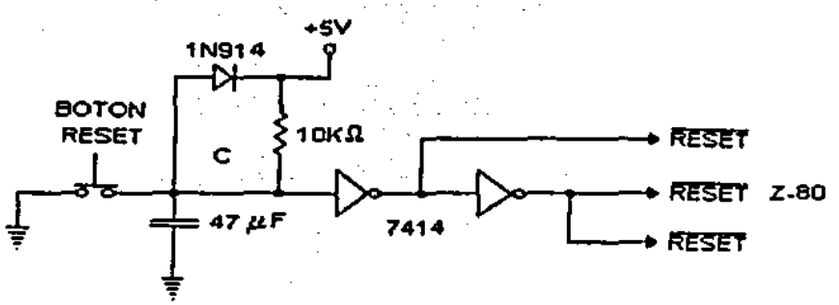


Figura 3-7

Una vez que el sistema está en operación, se puede detener la ejecución del programa y volverla a empezar a partir de la dirección 0, oprimiendo momentáneamente el interruptor del circuito de restablecimiento. Al presionar el interruptor, el capacitor queda en cortocircuito, descargándose rápidamente y provocando que la entrada RESET baje a 0 lógico. El μprocesador permanece en estado de "reset" mientras se mantiene oprimido el interruptor. Al soltarlo, el capacitor se carga a través de la resistencia, y la entrada RESET vuelve a 1 lógico, con lo cual el μprocesador reinicia la ejecución.

Las especificaciones del Z-80A exigen que la señal de reset esté activa en 0 lógico, por lo menos, durante 3 ciclos de reloj, lo que significa 1.2 μsegundos para una frecuencia de 2.5 MHz. Sin embargo, al calcular la constante de tiempo RC, es necesario tener en cuenta el tiempo de estabilización de las señales al encender el sistema, que es del orden de 20 msegundos; por lo tanto, RC debe ser mayor de 30 msegundos. Por ejemplo, con un capacitor de 10 μf y una resistencia de 15 KΩ, la duración en el 0 lógico del pulso aplicado a RESET es, aproximadamente, de 150 msegundos, lo cual es más que suficiente para asegurar que los voltajes de polarización han alcanzado sus niveles correctos en todos los circuitos.

Los inversores con gatillo Schmitt, del circuito de la fig. 3-7, aumentan la confiabilidad del diseño al introducir histéresis y acelerar la transición entre niveles lógicos; además, permiten la obtención de una señal de restablecimiento (reset) activa en 1 lógico, y otra activa en 0 lógico. El uso del diodo tiene por objeto descargar rápidamente el capacitor, para asegurar la generación del pulso de restablecimiento (reset), aun en los casos en que se apague y se vuelva a encender el sistema en un lapso muy corto.

3.3 Operación del microprocesador.

3.3.1 Definiciones.

Ya se ha visto que, durante la operación normal del μprocesador, todas las instrucciones son procesadas secuencialmente en tres fases: OBTENCION (Fetch), DECODIFICACION

(Decode) y EJECUCION (Execute). Cada una de estas fases requiere del transcurso de varios ciclos de reloj.

A un ciclo de reloj se le llama ESTADO (State). En el Z-80A, un estado se define como el periodo entre dos transiciones consecutivas de reloj (fig.3-8). Los estados se identifican con la letra "T": T1, T2, T3, etc.

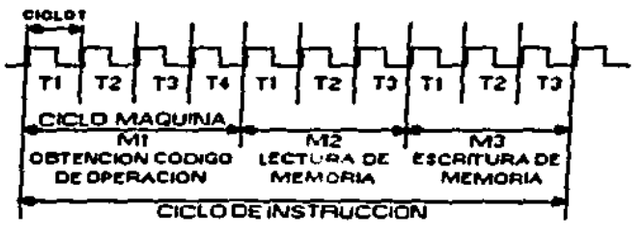


Figura 3-8

La obtención y ejecución de una instrucción constituye un CICLO INSTRUCCION (Instruction Cycle), el cual consiste en una o más operaciones elementales, en las que el μprocesador lleva a cabo alguna transferencia de la información, ya sea con la memoria o con un dispositivo de entrada/salida. Cada una de estas operaciones recibe el nombre de CICLO DE MAQUINA (Machine Cycle), y se desarrollan a través de una sucesión de estados (fig.3-8). Los ciclos de máquina se identifican con la letra "M": M1, M2, etc.

Un ciclo de máquina del Z-80A puede durar de tres a seis estados, y un ciclo de instrucción puede tener de uno a seis ciclos de máquina. La duración de cada ciclo de máquina y, por lo tanto, de cada ciclo de instrucción, es dependiente de la instrucción en particular. Invariablemente, el primer ciclo de máquina de una instrucción (M1) se emplea para la obtención y decodificación del código de operación, y puede durar cuatro, cinco o seis estados. Algunas instrucciones no requieren de más ciclos de máquina, fuera del utilizado en la obtención de la instrucción; otras necesitan ciclos adicionales para transferir datos entre la CPU y la memoria o los puertos de entrada/salida. Estos ciclos de máquina pueden tener de tres a cinco estados.

Con base en lo anterior, las instrucciones más rápidas del Z-80A duran cuatro estados; por ejemplo, INC r requiere un ciclo de máquina de cuatro estados, lo que equivale a 1.6 μsegundos para un reloj de 2.5 MHz. Las instrucciones más lentas son de 23 estados; por ejemplo, INC (IX+d) necesita 6 ciclos de máquina: el primero de seis estados, el segundo de cinco estados, y los cuatro restantes de tres estados cada uno, dando un total de 23 estados que representan 9.2 μsegundos para un reloj de 2.5 MHz.

3.3.2 Las señales M1, RD y WR.

Dado que por el bus de datos circulan tanto instrucciones como datos, y que además, la transferencia de datos es bidireccional, es decir, el µprocesador puede pedir o proporcionar datos, éste debe poseer alguna forma de indicarle a la circuitería externa (memoria, puertos de entrada/salida), tanto el tipo de información asociada a la transferencia (instrucciones o datos), como el sentido de la transferencia (lectura o escritura). Para ello, el Z-80A cuenta con tres salidas de control activas en 0 lógico: M1, RD y WR.

La señal M1 (Machine Cycle One) indica que en ese ciclo de máquina, el µprocesador va a obtener el código de operación de una instrucción. En las instrucciones que tienen un código de operación de dos bytes, la señal M1 se genera al obtener cada uno de los bytes del código de operación.

La señal RD (Read) se activa cuando el µprocesador requiere leer un dato de la memoria o de un puerto. Indica que el bus de datos se está usando como entrada.

La señal WR (Write) se genera cuando el µprocesador va a escribir un dato en la memoria o en un puerto. Indica que el bus de datos se está usando como salida, y que en sus líneas se encuentra el dato que va a ser almacenado.

Tanto la señal RD como la señal WR, provienen de terminales de salida con tercer estado.

3.3.3 Operaciones básicas y sus diagramas de tiempos.

Como se dijo en la sección 3.3.1, todas las instrucciones del Z-80A son meramente una serie de ciclos de máquina, en cada uno de los cuales, el µprocesador realiza una operación específica. Dentro de estas operaciones básicas, están las siguientes:

- Obtención del código de operación de una instrucción (OP code fetch).

- Lectura de un dato de la memoria (Memory data read).

- Escritura de un dato en la memoria (Memory data write).

Para entender mejor la relación que existe entre las señales del bus de datos, del bus de direcciones, y de las líneas de control descritas anteriormente, es conveniente examinar los diagramas de tiempos de las diferentes operaciones que efectúa el µprocesador. Estos diagramas muestran el comportamiento de las señales en el lapso que comprende la ejecución de un ciclo de máquina. A continuación, se analizarán los diagramas de tiempos de las tres operaciones básicas mencionadas arriba.

La obtención del código de operación de una instrucción se refiere al ciclo de máquina, en el cual, el µprocesador lee de la memoria un byte, que será interpretado como el código de operación de esa instrucción. La figura 3-9 muestra el diagrama de tiempos para esta operación, también llamada ciclo M1. Un ciclo de máquina comienza con la transición de 0 a 1 de la señal de reloj.

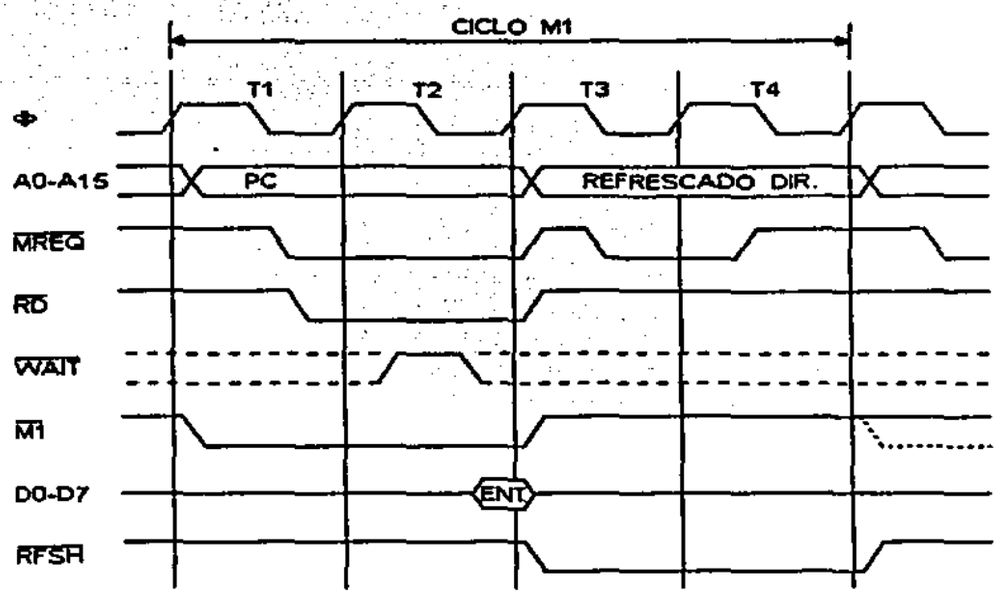


Figura 3-9

En la figura 3-9 se observa que, al inicio del primer ciclo de reloj (T1), la señal M1 se activa (baja a 0 lógico), indicando que se trata de una operación para la obtención de una instrucción. Al mismo tiempo, el contenido del contador del programa (PC) aparece en el bus de direcciones. Medio estado después, la señal de lectura RD también se activa, para indicarle a la memoria que coloque la información en el bus de datos. Durante el segundo estado (T2), se incrementa automáticamente el PC, y la memoria tiene tiempo de poner, en las líneas de datos, la instrucción almacenada en la localidad especificada por la dirección. El µprocesador utiliza la subida del reloj en el estado T3 para leer la instrucción y para desactivar las señales M1 y RD (regresan a 1 lógico). Durante los siguientes dos estados (T3 y T4), se decodifica el código de operación y, si es posible, el µprocesador ejecuta la instrucción. Es necesario aclarar que, en los diagramas aparecen señales que, por el momento, no se explican, pero que serán explicadas posteriormente.

En la operación de lectura de memoria, se transfiere un dato (información que no sea el código de operación de una instrucción) de una localidad de memoria al µprocesador, mientras que en la operación de escritura en memoria, ocurre una transferencia de

información del µprocesador a una localidad de memoria. En la figura 3-10 están los diagramas de tiempos correspondientes a cada una de estas operaciones.

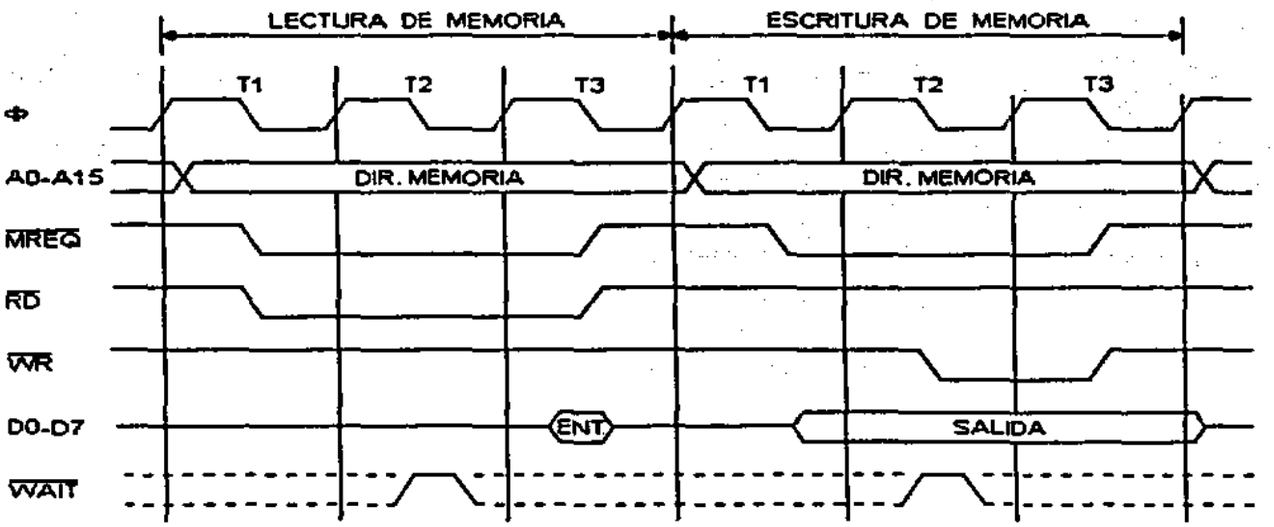


Figura 3-10

Al inicio del primer estado (T1), el µprocesador coloca en el bus de direcciones la localidad de memoria asociada a la transferencia. Esta dirección se obtiene del PC cuando el dato es parte de una instrucción; cuando no es así, entonces se obtiene de alguno de los registros apuntadores de 16 bits (BC, DE, HL, IX, IY o SP).

En el caso de una operación de lectura, medio estado después se activa la señal de lectura RD, como sucedió en la obtención de una instrucción. En la mitad del tercer estado (T3), el µprocesador lee el dato colocado por la memoria, y RD vuelve a 1 lógico, con lo que finaliza el ciclo de máquina.

Cuando se trata de una operación de escritura, el µprocesador coloca en el bus de datos, la información que va a escribir a partir de la segunda mitad del primer estado (T1). Entre la segunda mitad del segundo estado (T2) y la primera mitad del tercer estado (T3), se genera un pulso a 0 lógico en la señal de escritura WR, con el cual, el µprocesador ordena a la memoria que almacene el dato que se encuentra en ese momento en el bus de datos. Las líneas de datos mantienen la información válida hasta el término del tercer y último estado del ciclo de máquina.

Por ejemplo, la instrucción LD (pq),A es una instrucción de tres bytes que almacena el contenido del acumulador en la localidad de memoria, cuya dirección (pq) está indicada por el segundo y tercer bytes de la instrucción. Su ciclo de instrucción consta de cuatro ciclos

de máquina: el primero de cuatro estados y los otros, de tres; en total, 13 estados (5.2 µsegundos para un reloj de 2.5 MHz).

Se requieren tres ciclos de máquina para obtener la instrucción de la memoria. En el primero (M1), se lee el código de operación (OP code fetch) y se coloca en el registro de instrucciones, donde es decodificado. Luego se leen los dos bytes de la dirección, por medio de dos ciclos de lectura de memoria (memory read) y se almacenan temporalmente dentro del µprocesador. En este punto, la instrucción completa está en el µprocesador y puede ser ejecutada. La ejecución se lleva a cabo durante el cuarto ciclo de máquina (M4) y consiste en colocar, en el bus de direcciones, los dos bytes obtenidos durante M2 y M3; luego se pone el contenido del acumulador en el bus de datos, y en seguida se efectúa un ciclo de máquina de escritura en memoria (memory write), con lo cual, la localidad apuntada por la dirección, queda grabada con el contenido del acumulador. Con M4 termina el procesamiento de esta instrucción, y el µprocesador puede pasar a obtener el primer byte de la siguiente instrucción, en un nuevo ciclo de máquina M1.

En síntesis, el ciclo de instrucción de LD (pq),A se puede expresar como:

Mnemónico	**Byte de la instrucción**	**Ciclo de máquina**	**Estados**
LD (pq),A	Código de operación	Obtención de instrucción	4
	Dirección baja	Lectura de memoria	3
	Dirección alta	Lectura de memoria	3
		Escritura en memoria	3

La figura 3-11 presenta un diagrama de tiempos simplificado de la instrucción LD (pq),A.

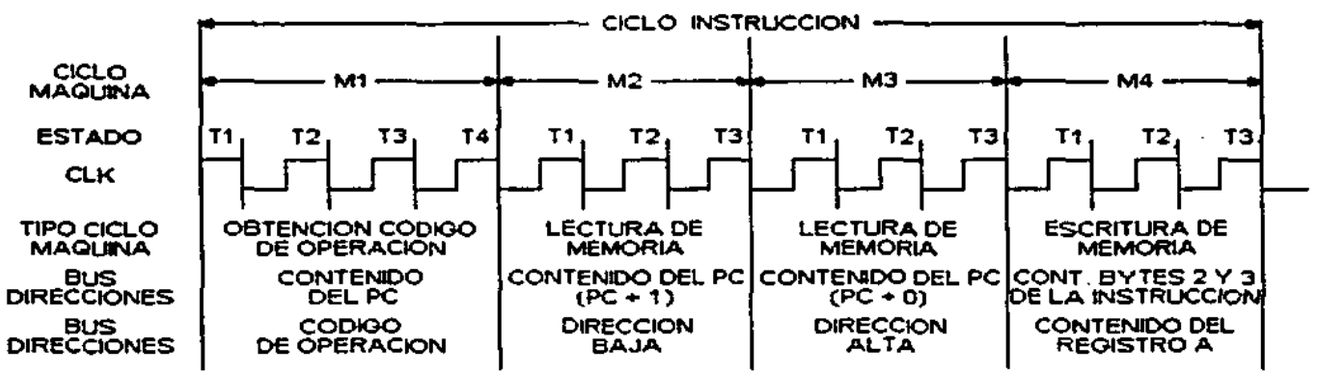


Figura 3-11

3.4 El estado de espera y la entrada WAIT.

Algunas veces, los circuitos de memoria o los dispositivos de entrada/salida que se interconectan con el µprocesador, responden más lentamente de lo previsto por los requerimientos de tiempo de los ciclos, los cuales hacen referencia a memoria o a los puertos de entrada/salida. Cuando esto ocurre, el dato que lee el µprocesador no es el correcto, o el dato que se escribe no se almacena en su destino.

El Z-80A cuenta con un mecanismo que le permite operar con una memoria de cualquier velocidad. Por medio de la entrada WAIT, la memoria puede colocar a la CPU en ESTADO DE ESPERA (wait state), hasta que esté lista para la transferencia de información.

El mecanismo funciona de la siguiente manera: durante el segundo estado (T2) de cada ciclo de máquina, al bajar la señal de reloj, el µprocesador revisa el estado de la entrada WAIT. Si es 0 lógico, entonces entra en un estado de espera (*tw*), que tiene una duración igual a un ciclo de reloj. Durante el estado de espera, el Z-80A vuelve a revisar la entrada WAIT y, si sigue activa, entra a un nuevo estado *tw* de tal forma, que se pueden seguir generando estados de espera, hasta que la línea de WAIT regrese a 1 lógico. Con esta técnica, es posible alargar un ciclo de máquina, hasta satisfacer el tiempo de acceso de cualquier memoria.

Mientras están en espera, todas las señales externas del µprocesador mantienen los mismos niveles lógicos que tenían al final de T2. La figura 3-12 muestra un ciclo de obtención de instrucción con dos estados de espera, y la figura 3-13 muestra la adición de dos estados de espera a un ciclo de lectura o escritura en memoria. En el caso de lectura de memoria, el efecto producido es el alargamiento en duración de las direcciones y de las señales MREQ, M1 y RD. En el caso de escritura en memoria, el alargamiento ocurre en las direcciones, en las señales MREQ y WR, y en las líneas de datos. Por supuesto, la duración del ciclo de instrucción aumenta en relación directa al número de estados de espera que se hayan agregado.

Haciendo referencia a las figuras 3-9 y 3-10 de la sección 3.3.3, nótese que los requerimientos de tiempo para el acceso a memoria son más severos durante el ciclo de obtención de instrucción (M1). Todos los demás accesos a memoria, tienen la mitad de un ciclo de reloj más para ser completados. Por esta razón, puede ser necesario agregar un estado de espera al ciclo M1 en algunas aplicaciones.

La figura 3-14 es un ejemplo de un circuito sencillo con el que se logra esto.

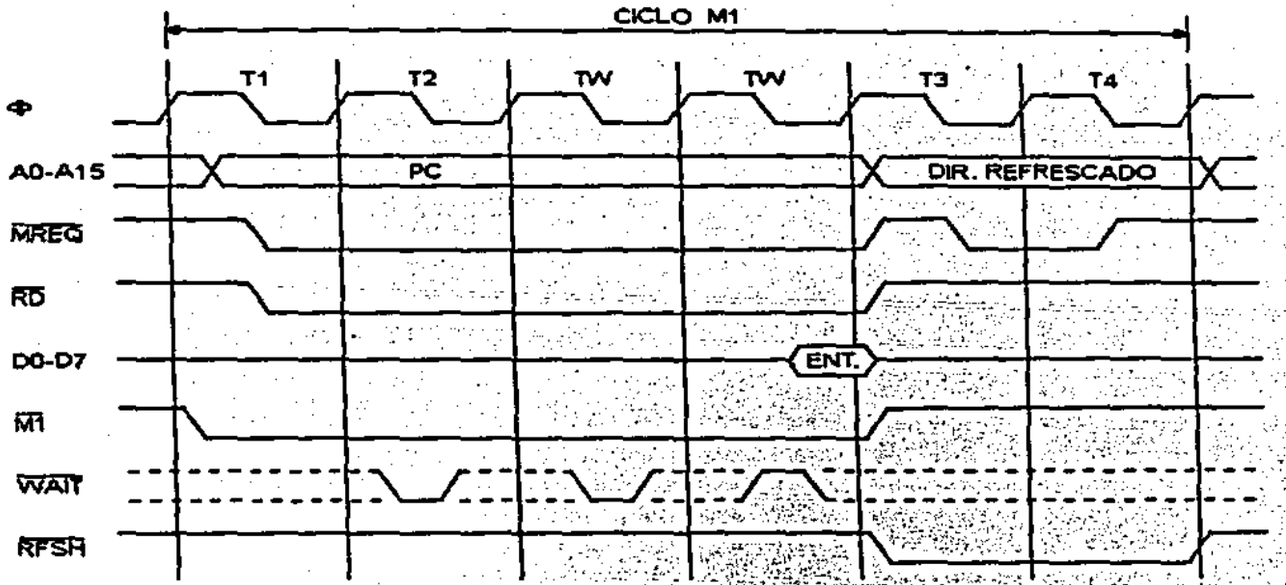


Figura 3-12

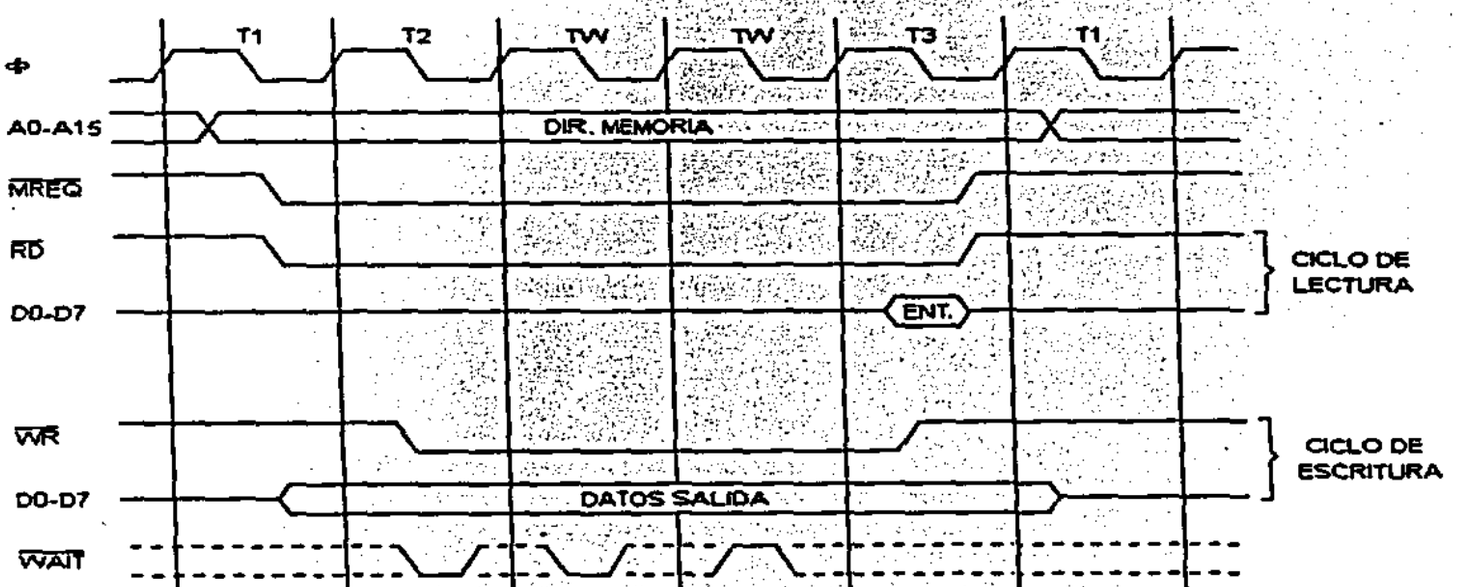


Figura 3-13

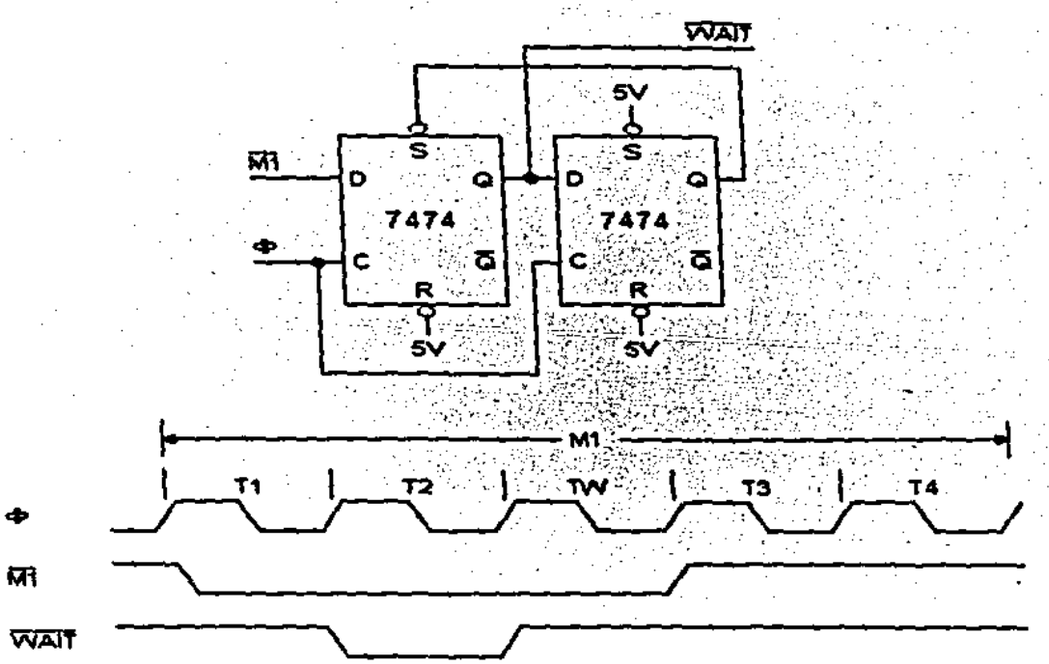


Figura 3-14

3.5 Programación elemental.

La función de cualquier sistema basado en un µprocesador se implanta por medio de transferencias de datos entre los registros del mismo, la memoria y los dispositivos de entrada/salida, así como de transformaciones de datos que ocurren principalmente en los registros internos del µprocesador.

Las clases de transferencias y de transformaciones que son posibles se especifican en el CODIGO DE INSTRUCCIONES. Cada µprocesador está diseñado para ejecutar un código particular de instrucciones.

En este capítulo se comenzará el estudio del código de instrucciones del µprocesador Z-80A, para lograr el aprovechamiento máximo de la capacidad de procesamiento de este dispositivo.

3.5.1 Formatos de las instrucciones.

Una instrucción especifica la operación que debe ser realizada por el µprocesador. Cada instrucción está dividida en secciones o CAMPOS, donde cada campo es un conjunto de bits que proporciona parte de la información requerida para saber el tipo de operación a ejecutar, y la localización de los operandos en los que ésta se realizará.

Desde un punto de vista simplificado, todas las instrucciones pueden ser representadas como un código de operación, seguido de campos adicionales que contienen los operandos de la instrucción, generalmente constantes numéricas o direcciones.

EL CODIGO DE OPERACION determina la actividad que se llevará a cabo. Además, indica cuál es el contenido de las siguientes localidades de memoria en el programa; es decir, si contienen un dato, una dirección, un desplazamiento u otro código de operación. Por razones de eficiencia, es conveniente que el código de operación resida totalmente en la primera palabra de la instrucción; éste es un factor limitante en el número de instrucciones disponibles en un µprocesador. Cuando su longitud de palabra es de 8 bits, se pueden tener hasta 256 códigos de operación diferentes en un byte.

En el caso del Z-80A, los códigos de operación son, en general, de un byte, a excepción de instrucciones especiales que requieren un código de operación de dos bytes. Tomando en cuenta la información que necesita cada instrucción, además del código de operación, en el Z-80A existen instrucciones de uno, dos, tres o cuatro bytes (fig. 3-15).

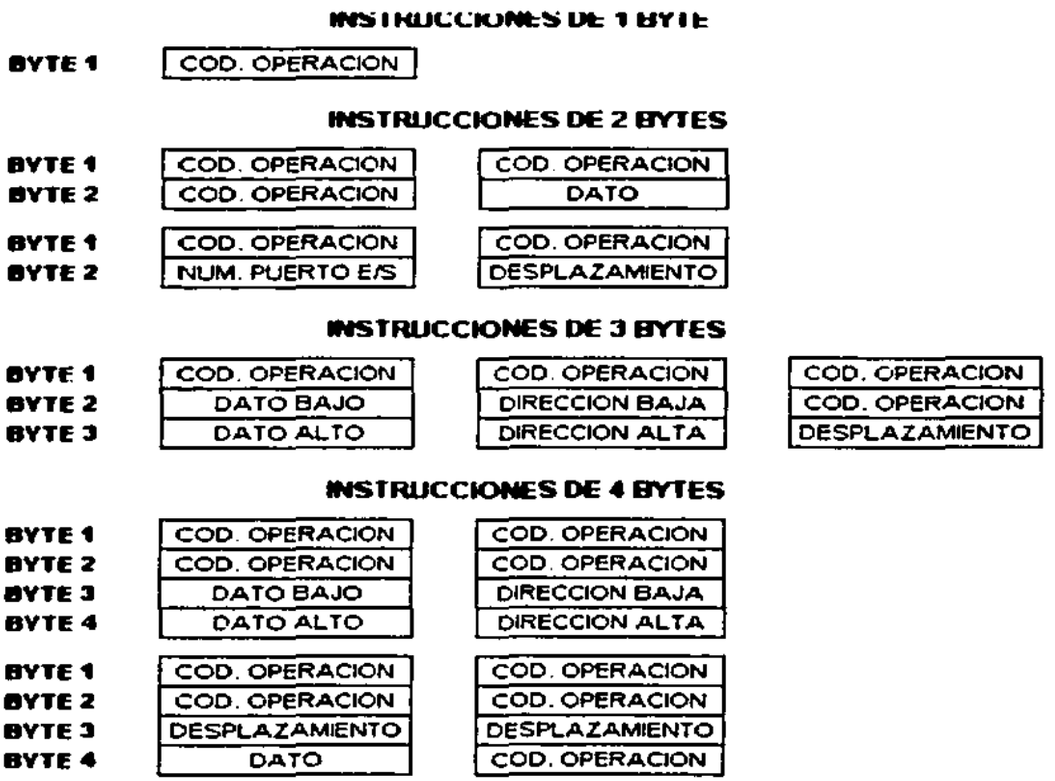


Figura 3-15

Por cada byte de la instrucción, la unidad de control tiene que realizar un ciclo de máquina, durante el cual se lee este byte de la memoria. Por lo tanto, mientras más corta sea una instrucción, más rápido será ejecutada.

3.5.1.1 Instrucciones de un byte.

Las instrucciones de un byte son, en principio, las más rápidas. Un ejemplo común de estas instrucciones en el Z-80A es "LD *r,r'*", que significa "cargar el contenido del registro *r'* en el registro *r*". Esta es una operación de registro a registro, que permite la transferencia de información entre dos de los registros internos del μprocesador.

Cualquier instrucción debe ser representada internamente en sistema binario. La expresión "LD *r,r'*" es meramente simbólica, y se denomina el MNEMONICO de la instrucción; sin embargo, ésta es la forma más conveniente de mostrarla desde el punto de vista del programador. El código binario que constituye la instrucción dentro de la memoria es: 0 1 D D D S S S.

Esta representación todavía es parcialmente simbólica. Cada una de las letras S y D ocupa el lugar de un bit. Las letras "D D D" se refieren a los tres bits que identifican al registro que actúa como destino de la información. Estos tres bits permiten seleccionar uno de los ocho registros disponibles. Por ejemplo, la clave para el registro C es 001.

De modo similar, "S S S" representa a los tres bits que apuntan al registro donde se origina la transferencia de la información. La convención que se sigue es que el registro *r'* es el origen, y el registro *r* es el destino. Aun cuando, aparentemente, el mnemónico de la instrucción está al revés (primero el destino y después el origen), se ha escogido esta convención para mantener la compatibilidad con la notación binaria, la cual ha sido diseñada para facilitarle la tarea a la unidad de control.

3.5.1.2 Instrucciones de dos bytes.

Algunas instrucciones requieren que un dato esté inmediatamente después del código de operación; entonces son necesarios dos bytes.

Por ejemplo, la instrucción ADD A,*n* es una instrucción de dos bytes, que suma el contenido del segundo byte de la instrucción con el acumulador. Este segundo byte es un dato o constante numérica, que es tratado como un grupo de ocho bits sin ningún significado particular. El código binario de esta instrucción es 1 1 0 0 0 1 1 0, seguido de un dato de ocho bits.

3.5.1.3 Instrucciones de tres bytes.

En ocasiones, la instrucción puede necesitar la especificación de alguna dirección. Una dirección requiere de 16 bits, o sea de dos bytes. De lo anterior se desprende que, incluyendo el código de operación, la instrucción es de tres bytes.

La instrucción JP *pq* tiene como función producir un salto en el programa a la dirección señalada por "*pq*". Como las direcciones tienen una longitud de 16 bits, la instrucción ocupa tres bytes.

En binario, esta instrucción está representada por el código 1 1 0 0 0 0 1 1, seguido de 8 bits para el byte bajo de la dirección, y 8 bits más para el byte alto de la misma.

3.5.2 Clases de instrucciones.

Con base en la función que desempeñan las instrucciones que es capaz de ejecutar el μprocesador Z-80A se pueden agrupar en varias clases, cada una de las cuales se analizarán a continuación.

3.5.2.1 Instrucciones de transferencia de datos.

Las instrucciones de transferencia de datos mueven información entre los registros internos de la CPU, o entre los registros y la memoria externa. Todas estas instrucciones deben especificar el lugar de donde la información será obtenida (origen), y el lugar donde será trasladada (destino).

Las instrucciones de transferencia de datos en el Z-80A se pueden agrupar en cinco categorías: transferencia de 8 bits, transferencias de 16 bits, intercambios, operaciones de pila y transferencias de bloques.

3.5.2.2 Instrucciones de procesamiento de datos.

Las instrucciones de procesamiento de datos operan sobre la información almacenada en el acumulador y otros registros de propósito general, o en localidades de memoria externa. Los resultados de las operaciones se colocan en el acumulador, y el estado de las banderas se determina de acuerdo con aquéllos.

Estas instrucciones incluyen: operaciones aritméticas (suma y resta), operaciones lógicas (AND, OR, XOR, complemento), operaciones de incremento y decremento, operaciones de rotación y desplazamiento, y operaciones de manipulación de bits (set, reset, test).

3.5.2.3 Instrucciones de transferencia de control.

Las instrucciones de transferencia de control son aquellas que trasladan el control del programa a una dirección determinada. Estas instrucciones cambian el flujo normal de ejecución a otro en el que una sección diferente del programa es procesada.

Existen dos tipos importantes de instrucciones de transferencia de control: las que trasladan el contenido dentro del programa principal, denominadas "saltos", y las que temporalmente lo transfieren a una sección o subrutina fuera del programa principal. La transferencia de control puede ser condicional (dependiendo del estado de ciertas banderas del µprocesador), o incondicional.

3.5.2.4 Instrucciones de entrada/salida.

El grupo de instrucciones de entrada/salida en el Z-80A permite una amplia gama de transferencias de datos entre las localidades de memoria externa, los registros internos de propósito general, y los dispositivos externos de entrada/salida.

3.5.2.5 Instrucciones de control.

Las instrucciones de control proporcionan señales de sincronización, y pueden detener o interrumpir la ejecución de un programa. Estas instrucciones afectan el modo de operación de la CPU o manipulan la información de su estado (status) interno.

3.5.3 Algoritmos y diagramas de flujo.

Un ALGORITMO es la especificación, paso a paso, de la solución de un problema dado. Este algoritmo puede ser expresado en cualquier lenguaje o simbolismo, y debe terminar en un número finito de pasos. En programación, un algoritmo es un método lógico o computacional para producir el resultado deseado. Casi siempre es necesario un paso intermedio entre el diseño del algoritmo y la codificación del programa; en este paso se construye el diagrama de flujo.

Un DIAGRAMA DE FLUJO es la representación gráfica de un algoritmo, indicando el orden en que las operaciones son llevadas a cabo, y las decisiones que determinan esta secuencia.

El diagrama de flujo es el equivalente en software de lo que es el diagrama de bloques en hardware. Sus finalidades son: simplificar la codificación del algoritmo en el lenguaje particular de la computadora o µprocesador; y facilitar la comprensión del algoritmo por otras personas.

3.5.3.1 Reglas de los diagramas de flujo.

Los diagramas de flujo consisten en una secuencia de símbolos geométricos que contienen los pasos del algoritmo. Los símbolos básicos se muestran en la figura 3-16.

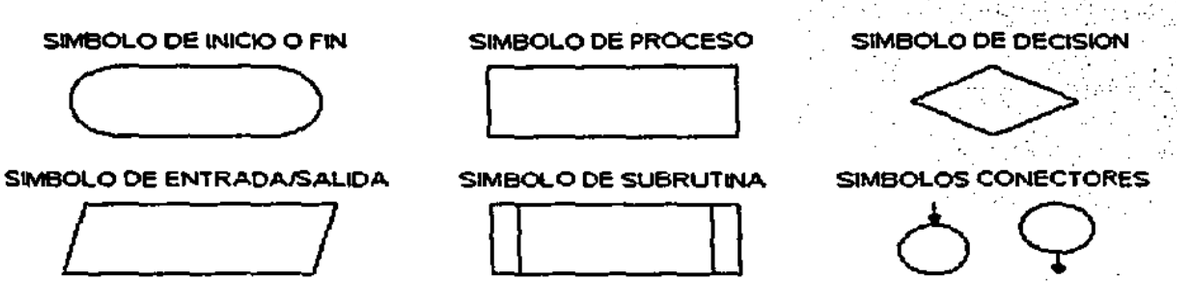


Figura 3-16

Un rectángulo representa una operación o proceso. Un rombo o diamante representa una decisión que da lugar a una transferencia condicional de control. Un óvalo o elipse representa el inicio o el fin de la secuencia de pasos. Los símbolos están conectados por líneas sólidas con puntas de flecha que indican el flujo del proceso. Mientras sea posible, el flujo debe ser de arriba hacia abajo y de izquierda a derecha.

Como es muy importante mantener la claridad y el orden en los diagramas de flujo, frecuentemente se utilizan conectores que indican los puntos de rompimiento y reanudación de la trayectoria de flujo, con el fin de evitar el cruce de líneas o permitir la continuación del diagrama en una hoja distinta. Los conectores se representan con círculos con un número o caracter, que sirve para identificar los conectores que enlazan una misma trayectoria de flujo.

Los enunciados que indican las operaciones o decisiones asociadas con cada símbolo aparecen dentro de él. Estas frases no precisan ninguna formalidad; sin embargo, se recomienda que sean cortas, expresadas con conceptos generales aplicables a cualquier lenguaje de programación, y completamente claras con respecto a lo que se quiere que haga la computadora o µprocesador. A fin de lograr lo anterior es preciso que, previamente al diseño del diagrama de flujo, se tenga la comprensión total del problema a resolver.

Normalmente existirá más de un camino o algoritmo para la solución del problema, cada uno con sus pros y sus contras. Es mucho más sencillo analizar todas las alternativas, con las ventajas y desventajas que presentan, en términos de diagramas de flujo, que con programas ya codificados.

En el proceso de creación de un programa, es más conveniente pasar de lo general a lo particular. Primero, se desarrolla un diagrama de flujo que represente al algoritmo de una

manera global, plasmando a grandes rasgos las etapas o tareas que se tienen que realizar para llegar a la solución final del problema. Este diagrama de flujo se divide progresivamente en pasos cada vez más detallados, hasta que cada símbolo pueda ser convertido en una instrucción o en un grupo de ellas.

Un concepto muy importante en el diseño de diagramas de flujo es el de PROGRAMACION ESTRUCTURADA. Por programación estructurada se entiende la partición del algoritmo o programa principal en módulos o procedimientos. Un PROCEDIMIENTO consta del conjunto de pasos necesarios para llevar a cabo una tarea del algoritmo. Cada procedimiento posee puntos de entrada y salida únicos, y puede considerarse independiente de los demás, a no ser por los parámetros comunes que se utilizan en la transmisión de información entre ellos.

Las ventajas de la programación estructurada son: simplificar el desarrollo de cada parte del algoritmo por separado, permitiendo concentrar la atención en pequeños detalles; y que produce programas que son más confiables, fáciles de entender, documentar y modificar.

En problemas pequeños, quizá no sean evidentes las grandes ventajas de esta técnica; sin embargo, cuando se trata de problemas complicados, éstas se convierten en factores decisivos.

3.5.4 Primeras instrucciones.

Habiendo conocido ya las características generales del código de instrucciones del Z-80A, ha llegado el momento de comenzar su estudio pormenorizado y el diseño de programas sencillos en los que aquéllas se apliquen.

3.5.4.1 Instrucciones de transferencia de datos.

En cualquier µprocesador, una gran cantidad de instrucciones está dedicada a la transferencia de datos entre los elementos del sistema. Estos pueden ser:

1. Registros internos del µprocesador.
2. Memoria.
3. Dispositivos de entrada/salida.

Todas estas instrucciones tienen un origen y un destino. El origen es el registro o localidad de donde se toma el dato, y el destino es el registro o localidad adonde se copia el dato. En una operación de transferencia de información, el contenido de la localidad que actúa como origen no sufre alteración.

Por el momento, se hablará de las instrucciones que tiene el Z-80A para la transferencia de datos entre sus registros internos. Para que se pueda trasladar la información entre dos

registros, previamente debió haberse cargado en alguno de ellos (el que va a ser origen) el dato a transferir. Con este fin, existe una instrucción que carga un valor de ocho bits "*n*" en el registro interno "*r*". Su mnemónico es:

LD *r,n*

Su representación binaria consta de dos bytes. El primero de ellos es el código de operación, en el cual, por medio de tres bits, se especifica el registro interno, que puede ser cualquiera de A, B, C, D, E, H, y L (tabla 3-1). El segundo byte corresponde al dato que será colocado en el registro. Por ejemplo, después de la ejecución de la instrucción LD C,3BH el contenido del registro C será 3BH.

CODIGO	REGISTRO
000	B
001	C
010	D
011	E
100	H
101	L
111	A

Tabla 3-1

La instrucción que se encarga de pasar un dato de un registro a otro es:

LD *r,r'*

En ella, una copia del contenido del registro "*r'*" se coloca en el registro "*r*". Su código binario es de un byte; de los ocho bits, tres son para especificar el registro fuente, y tres son para especificar el registro destino, de acuerdo a los valores de la tabla 3-1. Por ejemplo, si el registro A contiene el número 8CH y el registro H contiene el número 60H, la instrucción LD H,A trae por resultado que ambos contengan 8CH.

3.5.4.2 Instrucciones lógicas.

El Z-80A proporciona instrucciones que realizan las siguientes funciones lógicas: AND, OR (inclusiva), XOR (exclusiva) y NOT (complemento). Todas ellas operan únicamente con datos de ocho bits, uno de los cuales debe estar en el acumulador. En las funciones que requieren dos operandos, el otro puede estar contenido en el segundo byte de la instrucción (como dato inmediato), o en un registro interno de propósito general. El resultado siempre queda almacenado en el acumulador.

La instrucción equivalente a la operación NOT es la que obtiene el complemento a uno del acumulador:

CPL

Los bits en 0 del acumulador se vuelven 1, y los bits en 1 se convierten en 0. Por ejemplo, si en el acumulador hay 3DH, después de ejecutar la instrucción CPL, su contenido será C2H.

La instrucción AND realiza la función lógica AND, bit a bit, entre el contenido del acumulador y otro dato de un byte localizado en alguno de los lugares mencionados antes. Las formas de instrucción AND son:

AND con dato inmediato	AND con registro
AND *n* (2 bytes)	AND *r* (1 byte)

Uno de los usos importantes de dicha instrucción es poner en ceros (masking) uno o más bits del acumulador. Por ejemplo, para poner en ceros los cuatro bits más significativos de un byte en el acumulador, se puede utilizar el valor (mascarilla) 00001111. La instrucción con la que se obtiene este resultado es AND 0FH.

La instrucción OR realiza la función lógica OR entre 2 bytes, uno de los cuales está en el acumulador. Dos de las versiones de la instrucción OR son:

OR con dato inmediato	OR con registro
OR *n* (2 bytes)	OR *r* (1 byte)

La función OR se caracteriza por el hecho de que, para que el resultado sea 1, basta con que alguno de los operandos sea 1. Por lo tanto, el uso obvio de la instrucción OR es poner en 1 cualquier bit del contenido del acumulador. Si el acumulador contiene el dato AAH (10101010), y se ejecuta la instrucción OR 05H, el valor final del acumulador será AFH (10101111).

La instrucción XOR pone en 1 los bits del acumulador que difieren de los bits correspondientes del otro operando; aquellos que son iguales, los pone en 0. En otras palabras, se complementan los bits del acumulador para los cuales los bits del otro operando están en 1 lógico. Los otros bits del acumulador no se alteran. Dos formas de la instrucción XOR son:

XOR con dato inmediato	XOR con registro
XOR *n* (2 bytes)	XOR *r* (1 byte)

La instrucción XOR se usa en comparaciones. Si cualquier bit es diferente, la OR exclusiva de dos datos dará un resultado distinto de cero. Por otro lado, cuando a un valor se le aplica la función OR exclusiva con sí mismo, el resultado es siempre cero; de ahí que la instrucción XOR A, que realiza la OR exclusiva del acumulador con sí mismo, sea una alternativa a la instrucción LD A,0 para poner en cero el acumulador.

3.5.4.3 Instrucciones aritméticas.

El Z-80A posee instrucciones que ejecutan dos de las cuatro operaciones aritméticas básicas: suma y resta. En ambas operaciones, el μprocesador toma el dato almacenado en el acumulador como uno de los operandos. Al igual que en las instrucciones lógicas, el otro operando se obtiene del segundo byte de la instrucción o de un registro.

He aquí dos formas de la instrucción para sumar dos cantidades de ocho bits:

suma con dato inmediato	**suma con registro**
ADD A,*n* (2 bytes)	ADD A,*r* (1 byte)

En ambos casos, el contenido del acumulador se suma con el dato del otro operando, y el resultado se coloca en el acumulador, sustituyendo al valor que tenía antes de la operación. Por ejemplo, si en el acumulador hay 48H y en el registro B hay 02H, al ejecutar la instrucción ADD A,B el acumulador quedará con 4AH y el registro B con 02H.

La instrucción para efectuar una resta con datos de ocho bits, sigue un formato semejante al de la suma:

resta con dato inmediato	**resta con registro**
SUB *n* (2 bytes)	SUB *r* (1 byte)

En la instrucción de resta, el contenido del acumulador es el minuendo de la operación, del cual es restado el valor del otro operando o sustraendo. Como en la suma, el resultado también es guardado en el acumulador. Suponiendo que el acumulador tiene el dato 90H, la ejecución de la instrucción SUB 01H deja en el acumulador el valor 8FH.

Además, en el conjunto de instrucciones del Z-80A, hay un par de ellas que sirven para incrementar o decrementar el contenido de un registro o del acumulador.

La instrucción de incremento se representa por:

INC *r*

Su código binario es de un byte, y su efecto es aumentar en 1 el valor del dato almacenado en el registro "*r*". Por ejemplo, si en el registro D hay 0FH, después de la ejecución de INC D habrá 10H.

La instrucción de decremento tiene el formato:

DEC *r*

También es una instrucción de un byte, la cual disminuye en 1 el valor del dato almacenado en el registro "*r*". Por ejemplo, si el contenido del registro L es 27H, después de que se efectúa la instrucción DEC L, su contenido será 26H.

3.5.4.4 Funcionamiento del contador del programa.

Como ya se mencionó, el registro PC en el Z-80A proporciona la dirección de la localidad de memoria de donde se va a obtener la instrucción que corresponda en ese momento. Cuando la instrucción es de un byte, únicamente se requiere leer una localidad de memoria para obtenerla. Sin embargo, cuando consta de dos, tres o cuatro bytes, es necesaria la referencia a varias localidades consecutivas de memoria, a fin de recabarla en su totalidad.

Para la operación eficiente de un µprocesador, es deseable que el contador del programa esté siempre preparado a obtener de la memoria el siguiente byte de una instrucción, o la instrucción subsiguiente. Esto es precisamente lo que ocurre en el Z-80A. Cada vez que se obtiene un byte de la memoria de programa, el PC se incrementa automáticamente, en forma que queda apuntando a la siguiente localidad; si el contenido de ésta es parte de la misma instrucción, se repite la operación de obtención e incremento hasta que ha sido leída toda, después de lo cual, se pasa a su decodificación y ejecución. Debido a lo anterior, al comienzo de la ejecución de cualquier instrucción, el contador del programa ya se encuentra apuntando a la localidad de memoria que contiene el primer byte de la instrucción que sigue.

3.5.4.5 Instrucciones de transferencia de control incondicional.

Las instrucciones de transferencia de control más simples son los saltos incondicionales. La ejecución de una instrucción de salto incondicional altera el flujo secuencial normal del programa, sustituyendo el contenido del PC por un nuevo valor, determinado a partir de la información proporcionada en la misma instrucción. Este valor indica la localización del primer byte de la siguiente instrucción que será ejecutada y, por lo tanto, el control es transferido a la misma.

En el Z-80A existen dos clases de saltos incondicionales: absolutos y relativos.

Los saltos absolutos son aquellos que pueden transferir el control a cualquier instrucción del programa, sin importar la separación que exista entre la instrucción de salto y la instrucción a donde se salta. La manera como se logra esto, es incluyendo en la instrucción de salto la dirección completa de la instrucción a donde se desea brincar, con lo cual, al cargar el PC con ese valor, se produce automáticamente la transferencia de control. Dado que el µprocesador Z-80A tiene capacidad de direccionar hasta 64 K de memoria, para poder especificar una dirección en ese rango, son necesarios 16 bits. La instrucción que implanta un salto absoluto incondicional a la dirección "*pq*" es:

JP *pq*

Esta es de tres bytes, de los cuales el primero es el código de operación, el segundo son los ocho bits menos significativos de la dirección y el tercero son los ocho bits más significativos de la misma. Por ejemplo, para saltar a la dirección E876H se usa la instrucción JP E876H, codificada como C3H-76H-E8H. Al final de su ejecución, el contenido del PC es E876H.

Por otra parte, en los saltos relativos, la transferencia de control está determinada por la posición que guarda la instrucción de salto con respecto a la instrucción a donde se brinca. La instrucción de salto relativo, al contrario de la de salto absoluto, no contiene la dirección de la instrucción adonde se desea saltar, sino únicamente la distancia que separa a ambas. En este caso, la transferencia de control se efectúa agregando el valor de esta distancia al contenido del contador del programa. Con ello, el PC es desplazado hasta quedar apuntando al primer byte de la instrucción adonde se quería transferir el control, completando con esto la operación.

El mnemónico de la instrucción que ejecuta un salto relativo incondicional es:

JR *e*

donde "*e*" representa el valor del desplazamiento que se añade al contador del programa. La instrucción es de dos bytes; el primero de ellos corresponde al código de operación (18H), y el segundo al valor del desplazamiento. Para poder implantar saltos hacia adelante y hacia atrás, el desplazamiento se representa como un número de ocho bits en complemento a dos, lo cual permite saltos a instrucciones cuya dirección se encuentre dentro del rango de 127 bytes hacia adelante o 128 bytes hacia atrás de la dirección del primer byte de la instrucción de salto. En los saltos hacia adelante se usan desplazamientos positivos, y en los saltos hacia atrás, desplazamientos negativos.

Además, al calcular el valor del desplazamiento, hay que tomar en cuenta que en el momento de la ejecución del salto, el PC ya ha sido incrementado para apuntar a la siguiente instrucción y, por lo tanto, su contenido es dos unidades mayor que la dirección del primer byte de la instrucción JR *e*. Como a ese nuevo valor del PC es al que se le va a

sumar o restar el valor del desplazamiento, se vuelve necesario ajustar el desplazamiento, disminuyendo en dos su valor calculado para compensar el incremento del PC. Como resultado del ajuste anterior, la codificación del segundo byte de la instrucción "JR *e*" no es realmente el desplazamiento "*e*", sino "*e* - 2"; de tal suerte que el rango efectivo queda entre -126 bytes y 129 bytes, con respecto a la dirección de instrucción de salto.

Afortunadamente, se ha facilitado el uso de las instrucciones de saltos relativos, y permite que, en lugar del desplazamiento "*e*", se especifique la dirección real del destino del salto, dejando la tarea del cálculo "*e*" al programa ensamblador.

Por ejemplo, si se quiere transferir el control a la instrucción de la dirección 158H por medio de una instrucción de salto relativo localizada en la dirección 123H; como la distancia que separa a las dos instrucciones es 35H bytes, y como el salto será hacia adelante, entonces el valor del desplazamiento es positivo (*e* = 35H). La codificación en hexadecimal será 18H - 33H, donde 33H es el desplazamiento ajustado (*e* - 2 = 33H) y el mnemónico de la instrucción será JR ETIQUETA (ETIQUETA debe apuntar a la dirección 158H).

Si, por el contrario, se desea transferir el control a una instrucción que se encuentra antes del salto, por ejemplo en la dirección 120H, la distancia que separa a las instrucciones es 03H; pero como el salto será hacia atrás, el desplazamiento es negativo (*e* = FDH). La codificación en hexadecimal será ahora 18H - FBH, donde FBH es nuevamente el desplazamiento ajustado (*e* - 2 = FBH), y su mnemónico será JR ETIQUETA (ETIQUETA debe apuntar a la dirección 120H).

La ventaja de la instrucción de salto relativo es que ocupa menos espacio en memoria que la de salto absoluto, ya que el desplazamiento requiere solo de ocho bits (1 byte) en lugar de los 16 bits (2 bytes) que son necesarios para expresar una dirección. Sin embargo, tiene la limitación de que debe existir cierta proximidad entre la instrucción de salto y la instrucción a donde se busca transferir el control.

3.5.5 Las banderas del Z-80A.

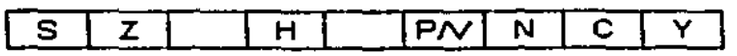
Todos los µprocesadores poseen un conjunto de flip-flops indicadores, llamados banderas, que reflejan ciertas condiciones resultantes de alguna operación efectuada por la ALU.

Dado que las banderas son flip-flops, sólo pueden adoptar uno de dos estados: 1 lógico (set) ó 0 lógico (reset).

Dentro del código de instrucciones, hay algunas que afectan el estado de una o varias banderas poniéndolas en 0 ó en 1; otras no alteran el estado de las banderas, y finalmente existe un tercer grupo, cuya ejecución depende del valor que tenga alguna bandera.

Así, las banderas juegan un papel muy importante en cualquier programa, ya que le confieren al µprocesador la capacidad de tomar decisiones, basado en las condiciones que ellas reflejan.

En el Z-80A, como en muchos otros µprocesadores, las banderas están agrupadas dentro de un registro de 8 bits llamado REGISTRO DE BANDERAS (Flag Register) F, o REGISTRO DE ESTADO (Status Register). De los ocho bits, solo seis están ocupados por banderas distribuidas en la forma siguiente:



Las banderas contenidas en el registro F son:

- S: Bandera de signo (sign).
- Z: Bandera de cero (zero).
- H: Bandera de acarreo intermedio (half carry).
- P/V: Bandera de paridad/desbordamiento (parity/overflow).
- N: Bandera de suma/resta (add/substract).
- CY: Bandera de acarreo (carry).

3.5.5.1 Las banderas acarreo y cero.

El bit de la posición menos significativa en el registro de banderas es la bandera acarreo o CY, probablemente la que se utiliza con más frecuencia.

Esta bandera indica si hay un acarreo hacia afuera del bit más significativo del resultado en una operación de suma. Esto es, si los bits del acumulador no son suficientes para almacenar el resultado, entonces la bandera CY se pone en uno. En caso contrario (si no hay acarreo), la bandera CY se pone en 0. Por ejemplo, si el acumulador tiene FEH y se ejecuta la instrucción ADD A,03, el resultado es 101H; como éste no puede representarse en 8 bits, entonces el acumulador queda con 01H, y la bandera acarreo se pone en 1. Si se vuelve a realizar la instrucción ADD A,03, el resultado es ahora 04H, quedando almacenado en el acumulador; como no hay acarreo, la bandera CY se pone en 0.

Cuando la operación es una resta, la bandera CY se pone en 1 si en el bit más significativo del minuendo se generó un préstamo (borrow). Cuando esto no ocurre, la bandera CY queda en 0. En otras palabras, la bandera CY se activa siempre que el sustraendo sea mayor

que el minuendo. Por ejemplo, si el acumulador tiene ceros y se ejecuta la instrucción SUB 01, la bandera CY se pone en 1 y el resultado almacenado en el acumulador es FFH.

Las instrucciones lógicas AND, OR y XOR siempre ponen la bandera CY en 0 lógico. Existen además otras instrucciones que la afectan en diversas formas, entre ellas están las instrucciones de rotación y desplazamiento, que serán tratadas más adelante.

La bandera acarreo puede manipularse directamente por medio de dos instrucciones de un byte. Una de ellas sirve para poner la bandera en 1 lógico, independientemente de su estado anterior. Su mnemónico es:

SCF

La otra tiene por función complementar su estado lógico, de manera que si está en 0 pasa a 1, y si está en 1 pasa a 0. Su mnemónico es:

CCF

A continuación se presentan algunas instrucciones aritméticas en las que la bandera acarreo actúa como operando.

Primeramente está la instrucción de suma con acarreo, que tiene los siguientes mnemónicos:

Suma con dato inmediato y acarreo	**Suma con registro y acarreo**
ADC A,*n*	ADC A,*r*

En ambos casos, el contenido del segundo operando y el valor de la bandera acarreo se suman al contenido del acumulador. El resultado se coloca en el acumulador, sustituyendo el contenido previo a la operación, y la bandera CY se pone en 1 lógico si el resultado no cabe en ocho bits. Por ejemplo, si en el acumulador hay 19H y la bandera acarreo está en 1, después de ejecutar la instrucción ADC A,30H, el contenido del acumulador será 4AH (30H + 19H + 1), y la bandera CY se pondrá en 0 lógico.

Una segunda instrucción es la de resta con préstamo, cuyos mnemónicos son:

Resta con dato inmediato y préstamo	**Resta con registro y préstamo**
SBC A,*n*	SBC A,*r*

En ella, el contenido del segundo operando y el valor de la bandera acarreo se restan al contenido del acumulador. Como es una resta, al acarreo se le denomina préstamo. El

resultado de la operación se coloca en el acumulador, reemplazando el contenido anterior, y la bandera CY se pone en 1 lógico cuando el sustraendo (segundo operando más acarreo) es mayor que el minuendo (acumulador). Por ejemplo, si el contenido del acumulador es 78H, el contenido del registro L es 7FH, la bandera acarreo está en 1 lógico, y se ejecuta la instrucción SBC A,L; al finalizar, el acumulador queda con 8FH, y la bandera CY se pone en 1 lógico.

Las instrucciones de suma con acarreo y de resta con préstamo se utilizan para la implantación de operaciones aritméticas con números mayores de un byte.

El bit de la posición 6 del registro F se usa para saber si el resultado de una operación es 0, y se le llama la bandera cero o Z.

Esta bandera se pone en 1 lógico, siempre que el resultado de una operación aritmética o lógica es 0, y se pone en 0 lógico cuando el resultado es diferente de 0. Por ejemplo, si el contenido del acumulador es 6AH, el del registro C es 69H, la bandera acarreo está en 1 lógico y se efectúa la instrucción SBC A,C; al terminar la ejecución, en el acumulador habrá 00H, la bandera Z estará en 1 lógico y la bandera CY en 0 lógico.

Con la bandera Z se puede probar una variedad de condiciones, algunas de las más comunes son:

- La igualdad de 2 operandos después de una comparación.

- El que un contador llegue a 0 después de incrementarlo o decrementarlo.

- Preguntar por el valor de un bit.

3.5.5.2 Las banderas restantes del registro F.

Además de las banderas acarreo y cero, el registro F contiene otras 4 que, aunque se utilizan con menor frecuencia que las dos primeras, son también importantes como indicadores de la ocurrencia de ciertas condiciones.

Estas cuatro banderas adquieren su mayor significación en el contexto de las operaciones aritméticas.

El Z-80A cuenta con una bandera denominada signo (S) en la posición 7 del registro F, que sirve para indicar el signo del resultado en las operaciones aritméticas. Este µprocesador maneja los números con signo, empleando la notación de complemento a 2. En ella, el bit más significativo (bit 7) se usa para representar el signo; si es 0, el número es positivo; si es 1, el número es negativo (el número cero se considera positivo).

La bandera S refleja el valor del bit más significativo de un resultado; es decir, guarda el estado del bit 7 del acumulador. Se pone en 1 lógico cuando el resultado es negativo, y en 0 lógico cuando es positivo.

La bandera de paridad y sobreflujo (P/V) es una bandera con doble propósito que ocupa el bit 2 del registro F. Indica la paridad del resultado cuando se ejecutan operaciones lógicas, y representa el desbordamiento cuando se realizan operaciones aritméticas de números con signo que utilizan el complemento a 2.

En el caso de que se presente desbordamiento al efectuar una operación aritmética, la bandera P/V se pone en 1 lógico si el resultado excedió al número positivo máximo (127 o 7FH), o al número negativo mínimo (-128 u 80H) que se puede representar con la notación de complemento a 2.

La paridad es una medida del número total de unos en el resultado de una operación lógica. Ocurre paridad par cuando el número de bits en 1 del resultado es par; en este caso la bandera P/V se pone en 1 lógico. Si el número de bits en 1 del resultado es impar, entonces éste tiene paridad impar, y la bandera P/V se pone en 0 lógico. Cuando el resultado consta únicamente de ceros, se considera que tiene paridad par, y por lo tanto la bandera P/V será 1.

El Z-80A tiene dos banderas cuyo estado no se puede probar, pero que son utilizadas internamente por el µprocesador para operaciones en aritmética BCD. Estas dos banderas son las de suma/resta (N) y la de acarreo intermedio (H), que corresponden a las posiciones 1 y 4 del registro F, respectivamente.

La bandera H registra el acarreo que se produce del bit 3 al bit 4 en el resultado de las instrucciones aritméticas, y se utiliza para ajustar el resultado de las operaciones a un número en BCD. Durante una operación de suma, la bandera H se pone en 1 lógico cuando hay un acarreo del bit 3 al bit 4; durante una operación de resta, se pone en 1 lógico cuando hay un préstamo del bit 4 al bit 3.

Los algoritmos para corrección (ajuste) de los resultados con números BCD para la suma son diferentes de los utilizados en la resta. La bandera N sirve para indicarle al Z-80A el tipo de la última operación que ejecutó, poniéndose en 0 lógico después de una instrucción de suma o de incremento, y en 1 lógico después de una instrucción de resta o de decremento.

3.6 Programación intermedia.

En este capítulo se estudiarán los conceptos de pila (stack) y subrutina, la relación que existe entre ellos y se continuará con el análisis de las instrucciones del Z-80A, haciendo especial énfasis en aquellas que sirven para implantar la pila y las subrutinas. También se

hablará de la transferencia de parámetros hacia o desde las subrutinas y de la generación de subrutinas de espera.

3.6.1 La pila y sus operaciones.

La PILA es una estructura de almacenamiento de datos primordial en cualquier computadora o µprocesador.

Se puede establecer una analogía entre la pila y un conjunto de objetos, por ejemplo platos, que se colocan en un recipiente que tiene una plataforma sostenida por un resorte. Los platos se apilan uno encima de otro como se observa en la figura 3-17. Si en el recipiente vacío se introduce un primer plato, quedará sobre la plataforma y por el empuje del resorte estará a la vista en la parte superior del recipiente. Sin embargo, conforme se van añadiendo platos a la pila, el resorte va cediendo por el peso de éstos, y solo permanece a la vista el último plato colocado, mientras que el primero de ellos se localiza ahora en el fondo de la pila y dentro del recipiente.

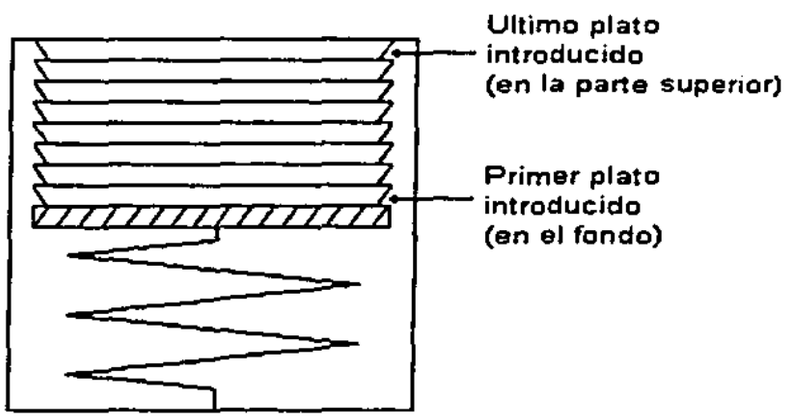


Figura 3-17

De lo anterior se deduce que el único plato al que se tiene acceso en un momento dado es aquel que se encuentra a la vista en la parte superior del recipiente y que, coincidentemente, es el último que se agrego a la pila. No se puede hacer uso de ningún otro plato de la pila si antes no se retiran todos los que están encima del que se desea, ya que por encontrarse dentro del recipiente, no hay forma de sacarlo de en medio.

Esta forma de almacenamiento y de acceso es precisamente la que se lleva a cabo en una pila. La pila en un sistema de µprocesador es una colección de registros o de localidades de memoria, organizados de manera tal que el último dato almacenado (escrito) en la pila es el primero que está disponible para ser sacado (leído) de él; o dicho de otra forma, el primer

elemento introducido en la pila está siempre en la parte más baja de este, mientras que el elemento depositado más recientemente queda ubicado en la parte superior.

Debido a su comportamiento, la pila se define como una estructura LIFO (last in - first out) y a la única localidad susceptible de ser leída o grabada con algún dato se le nombra parte superior (top) de la pila.

Normalmente son dos las operaciones que involucran a la pila. La primera de ellas ocurre cuando se deposita un dato en la parte superior de la pila; ésta operación se conoce como "push". La segunda operación, llamada "pop" es aquella en la que se lee la información contenida en el registro que ocupa la parte superior de la pila, después de lo cual este registro queda vacío y la parte superior de la pila se traslada al siguiente registro ocupado.

En μprocesadores, básicamente existen dos formas de implantar una pila. Algunos μprocesadores proporcionan un número fijo de registros internos para ser usados como pila. Este modo de implantación tiene la ventaja de una alta velocidad de acceso, pero también la desventaja de una "profundidad" pequeña, generalmente entre 4 y 16 palabras, que está limitada por el hardware del μprocesador (número de registros disponible) (fig. 3-18).

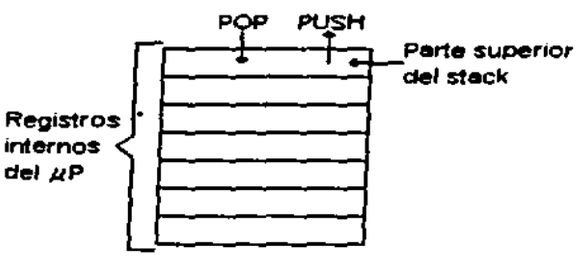


Figura 3-18

Una implantación alternativa más flexible se logra destinando para la pila una porción de memoria RAM externa conectada al μprocesador. Esta solución se ha escogido en la mayoría de los μprocesadores de propósito general, entre ellos el Z-80A.

Su principal ventaja es que la información contenida en la pila permanece en la misma posición mientras que la pila se "mueve", expandiéndose o contrayéndose, a medida que se realizan operaciones de introducción de datos a la pila (push) y extracción de datos a la pila (pop). De esta manera, la profundidad o longitud de la pila puede crecer lo que sea necesario, siempre y cuando no sobrepase los límites definidos en la memoria (fig. 3-19).

Esta última implantación requiere de un registro adicional dentro del μprocesador, cuya función es la de apuntar en todo momento a la localidad de memoria que sea entonces la parte superior de la pila. El Z-80A posee un registro de uso dedicado, denominado APUNTADOR DE PILA (stack pointer) SP, que cumple con la función anterior. El registro

SP tiene una longitud de 16 bits y, por lo tanto, puede apuntar a cualquier localidad de los 64 KB de memoria direccionables por el μprocesador.

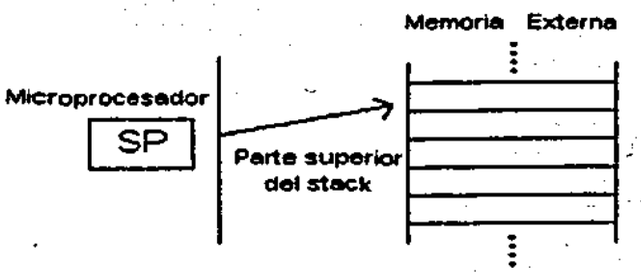


Figura 3-19

El comienzo del área de memoria dedicada a la pila se especifica cargando el apuntador de pila (SP) con un valor inicial. Esta definición es necesaria porque sistemas diferentes utilizan diferentes configuraciones de memoria. El registro SP se puede inicializar por medio de la instrucción

LD SP,*nn*

la cual carga el valor inmediato de 16 bits "*nn*" en SP. Esta instrucción ocupa 3 bytes en la memoria; el primero es el código de operación, el segundo los 8 bits menos significativos del valor y el tercero los 8 bits más significativos del mismo.

El apuntador de pila debe inicializarse antes de que cualquiera de las instrucciones que usa la pila sea ejecutada. Por lo tanto, la inicialización de la pila tiene que ser una de las primeras instrucciones en un programa.

En el Z-80A, al igual que en otros μprocesadores como el 8080 y el 8085 la pila crece hacia abajo, esto significa que conforme se van depositando datos en la pila, se decrementa el registro SP y la pila se expande hacia localidades de memoria con direcciones más bajas. Por lo anterior, comúnmente el apuntador de pila se carga inicialmente con la dirección más alta disponible en la memoria. Por ejemplo, LD SP,2000H.

Para inicializar el registro SP con un valor que se haya determinado por medio del programa, se puede utilizar la instrucción

LD SP,HL

la cual permite trasladar al registro SP el contenido del registro par HL. Esta instrucción es de un byte.

La pila se requiere para implantar tres conceptos de programación: subrutinas, interrupciones y almacenamiento temporal de información a alta velocidad. El rol que juega la pila en las subrutinas se describirá posteriormente.

Las operaciones push y pop hacen posible la utilización de la pila como lugar de almacenamiento temporal. En el Z-80A existe la instrucción

PUSH *qq*

que deposita en la pila el contenido del registro par (BC, DE o HL) indicado por "*qq*"; su código de operación es de un byte.

Por ejemplo, la instrucción PUSH DE guarda en la parte superior de la pila el contenido del registro par DE. El registro SP no puede especificarse porque crea una situación de indeterminación.

La instrucción PUSH transfiere el contenido del registro alto del par (B, D o H) a la localidad de memoria que está una dirección más abajo que la apuntada por el valor inicial del SP, y transfiere el contenido del registro bajo (C, E o L) a la localidad de memoria cuya dirección es dos más abajo que el valor inicial del SP. Luego el apuntador depila se decrementa en dos para apuntar a la nueva "parte superior" de la pila. Como en le Z-80A todas las instrucciones relacionadas con la pila implican una transferencia de dos bytes, entonces la parte superior de la pila en realidad consiste en dos localidades de memoria cuyas direcciones están definidas por SP y SP+1.

La operación pop se implanta en el Z-80A con la instrucción

POP *qq*

la cual transfiere el dato almacenado en la parte superior de la pila al registro par especificado en la instrucción; su código de operación es de un byte. POP es la instrucción inversa de PUSH. En ella el contenido de la localidad de memoria apuntada por SP se carga en el registro bajo y el contenido de la localidad de memoria apuntada por SP+1 se carga en el registro alto. La operación pop realmente no saca información de la pila, simplemente la copia en un registro par. Sin embargo, las localidades de memoria de donde los datos fueron copiados se consideran vacías, así que cuando ocurre una operación push, estas localidades se vuelven a usar. Es importante hacer notar que el registro SP en el Z-80A siempre apunta a la última localidad ocupada de la pila.

Hay instrucciones especiales push y pop que depositan o sacan de la pila el contenido del acumulador (registro A) y el estado de las banderas (registro F). Estas instrucciones tratan a los registros A y F como si fueran un registro par, siendo A el registro alto y F el registro bajo. Al registro par AF también se le conoce como la PALABRA DE ESTADO DEL

PROCESADOR (processor status word), PSW. Los mnemónicos de estas instrucciones son:

PUSH AF POP AF

Las instrucciones PUSH y POP automáticamente decrementan (PUSH) o incrementan (POP) el registro SP. Por lo tanto, una vez que se ha inicializado la pila no hay necesidad de llevar cuenta de las direcciones de la pila en que se ha guardado información.

Como ejemplo de las instrucciones PUSH y POP y de sus efectos en los registros internos y en la pila, se presenta una forma de intercambiar el contenido de los registros pares BC y HL, utilizando la pila para almacenamiento temporal de información. Si se supone que el apuntador de pila ya ha sido cargado con un valor inicial (LD SP,1000H), sólo se requieren cuatro instrucciones de un byte para lograr el intercambio:

```
PUSH HL
PUSH BC
POP  HL
POP  BC
```

Nótese que la secuencia que siguen las operaciones PUSH y POP no corresponde a la que normalmente se utilizaría en una estructura LIFO como es la pila; sin embargo, esta alteración del orden es precisamente lo que ocasiona que el contenido de los registros sea intercambiado. En la figura 3-20 se muestra el contenido de los registros después de que la instrucción indicada ha sido ejecutada.

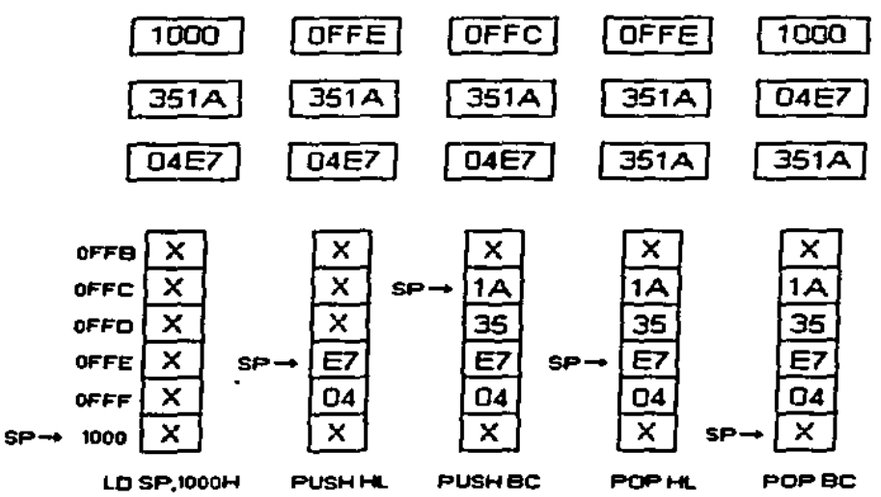


Figura 3-20

Hay otra instrucción que intercambia el contenido de la parte superior de la pila con el contenido del registro par HL. Su mnemónico es

EX (SP),HL

Es una instrucción de un byte y tiene la característica especial de que no afecta el valor del apuntador de la pila y, por lo tanto, permite el acceso al contenido de la pila (como si fuera memoria de datos con SP como apuntador), sin cambiar la posición de SP y sin perder el contenido del registro par HL.

El apuntador de pila puede utilizarse también como registro de 16 bits en tareas de índole más general. Para esto el Z-80A cuenta con varias instrucciones. En primer término existen dos que permiten la transferencia simultánea de dos bytes entre el registro SP y la memoria. Sus mnemónicos son

LD SP,(*pq*) LD (*pq*),SP

Ambas instrucciones son de cuatro bytes. La primera transfiere el contenido de la localidad apuntada por la dirección "*pq*" a los 8 bits menos significativos del registro SP y el contenido de la localidad con dirección "*pq*+1" a los 8 bits más significativos del mismo registro. La segunda realiza la transferencia en sentido inverso, es decir, los 8 bits menos significativos del registro SP pasan a la localidad de memoria direccionada por "*pq*" y los 8 bits más significativos de este registro se guardan en la localidad siguiente a la dirección "*pq*".

Asimismo, hay cinco instrucciones de tipo aritmético en las que el registro SP juega el papel de operando. Sus mnemónicos se dan a continuación:

ADD HL,SP	SBC HL,SP	INC SP
ADC HL,SP		DEC SP

La instrucción ADD HL,SP es interesante porque permite determinar la dirección contenida en el apuntador de la pila. En la mayoría de los programas no es usual el desear conocer la dirección guardada en el registro SP. Sin embargo, los sistemas operativos, monitores y programas de depuración frecuentemente deben informar al usuario la dirección de la localidad de memoria a la cual se encuentra apuntando el registro SP o verificar si la pila no está invadiendo el área de programa o de datos. El contenido del registro SP se puede cargar en el registro par HL con la siguiente secuencia de instrucciones:

```
LD   HL,0000
ADD HL,SP
```

Primero se carga con 0000H el registro HL y en seguida se suman los contenidos de los registros HL y SP. De esta forma, en el registro HL se obtiene una copia del contenido SP.

3.6.2 Las subrutinas.

Frecuentemente se encuentra la necesidad de que en diferentes puntos de un programa se lleve a cabo un proceso que requiere la ejecución del mismo grupo de instrucciones. En otras ocasiones es común que la misma tarea forme parte de distintos programas.

Una alternativa de solución consiste en reproducir dentro del programa el conjunto de instrucciones que realizan el proceso, tantas veces como éste sea requerido (fig. 3-21). Sin embargo aunque ésta parezca la manera más directa de atacar el problema, es también, la mayoría de las veces, la más ineficiente, al menos en lo que al uso de memoria se refiere. En cuanto al flujo del programa, éste es meramente secuencial. En la fig. 3-21 el flujo de ejecución se indica con flecha. Las flechas 2 y 4 representan la ejecución del proceso X en dos ocasiones.

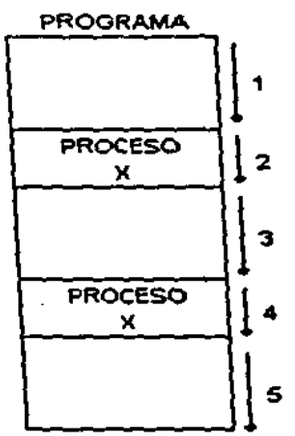


Figura 3-21

La solución mas efectiva, en términos del ahorro de memoria, se obtiene si el grupo de instrucciones que se repite aparece una sola vez en el programa, pero con capacidad para ser ejecutado desde todos los puntos en que aquel lo pide. La estructura de programación que implementa esta solución es la subrutina.

Una SUBRUTINA es simplemente un con nto de instrucciones al que se tiene acceso desde cualquier punto del programa princip. Como una subrutina conceptualmente queda fuera del flujo secuencial de ejecución d programa principal, son necesarios ciertos mecanismos para poder llegar a ella, y una v que se han ejecutado las instrucciones que la

componen, debe ser posible regresar al punto donde se quedo la ejecución del programa principal.

La acción de pasar del programa principal a la subrutina se denomina "llamada" a la subrutina y la acción de volver al programa principal después de llevar a cabo las tareas determinadas por la subrutina se conoce como "retorno" de la subrutina. El Z-80A posee dos instrucciones que proporcionan los medios para llamar y retornar de una subrutina. Sus mnemónicos son:

CALL *pq* RET

La instrucción CALL se utiliza para llamar a la subrutina que comienza en la dirección "*pq*". Es una instrucción de 3 bytes, el primero lo ocupa el código de operación, el segundo los 8 bits menos significativos de la dirección y el tercero los 8 bits mas significativos de la misma. La instrucción RET (return) permite el regreso desde la subrutina al programa principal; sólo ocupa un byte.

La figura 3-22 muestra el mismo programa de la figura 3-21, con la diferencia que el proceso X se ha codificado como una subrutina. Este cambio ha afectado a la secuencia de ejecución del programa de la siguiente manera: las lineas del programa principal son ejecutadas sucesivamente hasta que se encuentra la primera instrucción CALL. Su ejecución resulta en la transferencia de la secuencia de ejecución al proceso X (flecha 2a), después de lo cual éste se ejecuta como cualquier otra sección del programa (flecha 2). La última instrucción de la subrutina es un RET que causa el regreso de la secuencia de ejecución al programa principal (flecha 2b). La instrucción ejecutada después del return es la que sigue al CALL en el programa principal. La ejecución del programa continua normalmente hasta que aparece una segunda llamada al proceso X. De nuevo se transfiere el control a la subrutina (flecha 4a) y el proceso X es ejecutado (flecha 4). Al llegar a la instrucción RET ocurre un retorno al programa principal, esta vez a la instrucción siguiente al segundo CALL (flecha 4b).

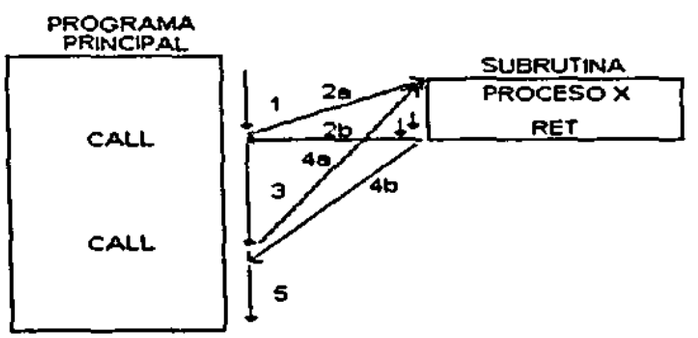


Figura 3-22

No obstante las ventajas ya citadas, las subrutinas presentan una desventaja que se puede detectar comparando los flujos de ejecución de las figuras 3-21 y 3-22. Se observa que el

uso de subrutinas resulta en una ejecución mas lenta debido a las instrucciones extras CALL y RET.

3.6.3 Los mecanismos de las subrutinas.

Ahora se examinará la forma en que las instrucciones CALL y RET y la estructura de la pila implantan en el μprocesador los mecanismos de las subrutinas.

La localización de una subrutina se identifica por la dirección de su primera instrucción. El efecto de la instrucción CALL es provocar que la ejecución se transfiera a la subrutina, de esto se desprende que la mencionada instrucción ademas del código de operación, contenga la dirección de la primera localidad de memoria ocupada por la subrutina.

Por otro lado, la instrucción RET, que provoca el retorno al programa principal, debe "recordar" la localización de la instrucción que sigue al CALL. Esto es posible sólo si la dirección de esa instrucción ha sido preservada en algún lado.

Antes de transferir el control a una subrutina, la instrucción CALL se encarga de almacenar en la pila la dirección de la instrucción que le sigue. Se puede decir entonces que una llamada a subrutina es esencialmente una operación push del contenido del PC, seguida de un salto.

Hay que recordar que la dirección de la siguiente instrucción al CALL, que es la que se quiere preservar, es precisamente el contenido del contador del programa en el momento de la ejecución del CALL. Esto es porque el PC se incrementa automáticamente cada vez que se usa para obtener una instrucción de la memoria, de tal suerte que al momento en que el μprocesador ejecuta una instrucción, el PC ya se encuentra apuntando a la dirección de la siguiente instrucción. Por lo tanto, cuando la instrucción CALL deposita en la pila el contenido del PC, lo que efectivamente esta preservando es la dirección del punto de retorno. La transferencia de la secuencia de ejecución a la subrutina se logra cargando el contador del programa con la dirección obtenida en la instrucción CALL. De esta forma, la siguiente instrucción que se obtiene de la memoria proviene de la subrutina.

La ejecución de la instrucción RET es equivalente a una operación pop, en la que la dirección de la instrucción siguiente al CALL es transferida de la pila al contador del programa. Así, la próxima instrucción que se obtiene de la memoria es precisamente la siguiente al CALL, regresando el control al programa principal. En la figura 3-23 se muestra la transferencia de controla dos subrutinas: SUB A que empieza en la dirección 412H y SUB B que empieza en la dirección 500H. Se puede observar que la ejecución de la primera subrutina termina antes de que se le llame a la segunda.

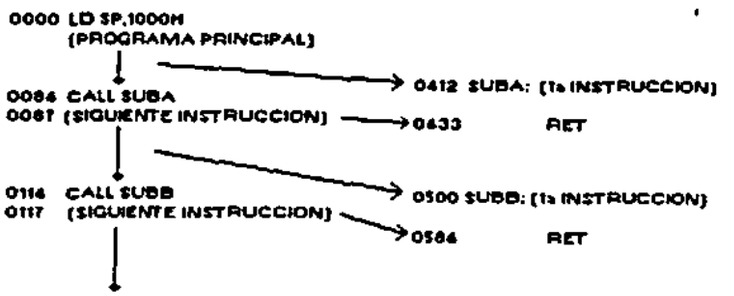


Figura 3-23

La figura 3-24 muestra el contenido de los registros PC y SP, así como de la pila antes y después de la ejecución de cada CALL y RET.

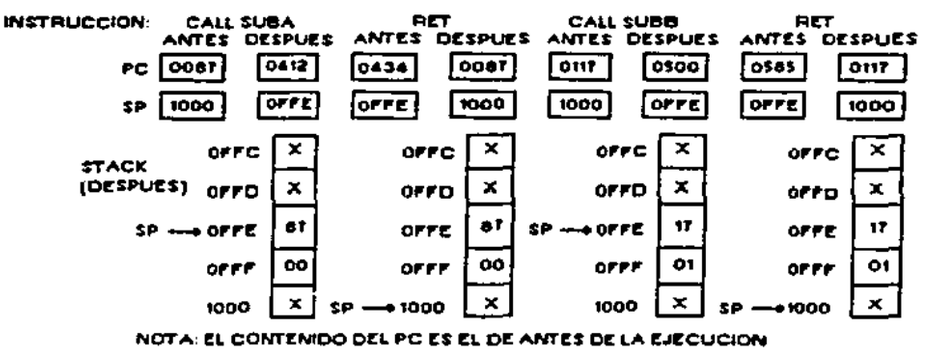


Figura 3-24

Cuando una subrutina llama a otra subrutina se produce la situación conocida como "anidamiento". La segunda subrutina puede a su vez llamar a una tercera, etc. Cada CALL sucesivo, sin que intervenga un RET crea un "nivel de anidamiento" adicional. El efecto de cada instrucción CALL es guardar en la pila la dirección apropiada para el retorno. Las instrucciones RET y sus efectos en la pila aseguran que el control del programa eventualmente regrese a la instrucción siguiente al primer CALL.

El número de subrutinas anidadas esta limitado por el tamaño del área de memoria destinada para pila. Como el registro SP normalmente se inicializa con una dirección alta de memoria y la pila crece hacia abajo, es posible que si se ejecutan muchas instrucciones PUSH o CALL sin alternarse con instrucciones POP o RET, esto cause que la pila se salga de sus límites establecido y destruya la información que se encontraba en esas direcciones. Para evitarlo, hay que tener cuidado de inicializar el SP dejando un espacio suficiente para que aún en su crecimiento máximo, la pila no invada otras áreas de memoria.

Cuando se llama a una subrutina, es de esperarse que esta use algunos de los registros internos para llevar acabo su tarea. Si la información que está en los registros antes de

llamar a la subrutina se va a utilizar después de regresar de ella, entonces es necesario preservar esta información.

El contenido de los registros puede ser almacenado en la pila ya sea por el programa principal o por la misma subrutina. Cuando la subrutina es llamada desde varios puntos en el programa principal se requiere menos memoria si la subrutina se encarga de guardar los registros. Para preservar los registros internos de propósito general al inicio de la subrutina, y restituirlos al final de ella, se puede hacer uso de la siguiente secuencia de instrucciones:

```
PUSH BC      ;primera instrucción de la subrutina
PUSH DE      ;guardar los registros internos
PUSH HL
PUSH AF
.
.            (otras instrucciones de la subrutina)
.
POP AF       ;restituir los valores originales de los registros
POP HL
POP DE
POP BC
RET          ;return (restituir PC)
```

Es importante hacer notar que si las instrucciones PUSH son parte de la subrutina, la información que se depositó por medio de ellas en la pila debe extraerse de la pila utilizando instrucciones POP antes de que la instrucción RET sea ejecutada, de lo contrario el control del programa será transferido a una localidad de memoria distinta a la que contiene la instrucción siguiente al CALL, causando un error lógico. El programa debe asegurarse que cuando la instrucción RET se ejecuta, la dirección de retorno apropiada se encuentre en la parte superior de la pila. Esto se alcanza fácilmente cuidando que las instrucciones PUSH y POP ocurran en pares.

3.6.4 Llamadas y retornos condicionales.

Junto con las instrucciones básicas para llamar a una subrutina y regresar de ella (CALL y RET), existen otras instrucciones de llamada y de retorno, cuya acción depende de que el estado de alguna bandera sea cero o uno.

En el Z-80A, el estado de las banderas Z, C, P/V y S da lugar a ocho condiciones diferentes que son:

NZ:	no cero (Z = 0)	PO:	paridad non (P = 0)
Z:	cero (Z = 1)	PE:	paridad par (P = 1)
NC:	no carry (C = 0)	P:	positivo (S = 0)
C:	carry (C = 1)	M:	negativo (S = 1)

La instrucción de llamada condicional a subrutina es:

CALL *cc.pq*

la cual causa una transferencia a la subrutina que empieza en la dirección "*pq*" solamente si la condición "*cc*" es verdadera; "*cc*" es un conjunto de tres bits (bits 3, 4 y 5 del código de operación), que especifican alguna de las ocho condiciones anteriores. Si la condición asociada con el CALL condicional no se cumple, entonces se continúa la ejecución con la instrucción siguiente al CALL. Si la condición se cumple, el PC es almacenado en la pila y la dirección contenida en la instrucción CALL se convierte en el nuevo contenido del PC (igual que una llamada a subrutina normal). La duración de la instrucción CALL condicional depende de que la condición haya sido o no satisfecha.

Similarmente, la instrucción:

RET *cc*

sirve para regresar condicionalmente de una subrutina. La condición "*cc*" puede ser cualquiera de las que se aplican en la llamada condicional. En el caso del retorno (return) condicional, el regreso de la subrutina al programa principal se efectúa únicamente cuando la condición es verdadera. Cuando se utilizan retornos condicionales en una subrutina, la última instrucción de ésta no tiene que ser necesariamente una instrucción RET. Esto no representa ningún problema siempre y cuando la ejecución de la subrutina termine en el momento en que se cumpla la condición "*cc*". La duración de los retornos condicionales también es variable.

3.6.5 Paso de parámetros en subrutinas.

Al utilizar una subrutina es de esperarse que ésta trabajará con ciertos datos. Estos datos de entrada, también denominados PARAMETROS o ARGUMENTOS, deben ser transferidos o "pasados" a la subrutina desde el programa principal. Asimismo, los resultados generados por la subrutina deben ser entregados al programa que la llamó. Por ejemplo, en el caso de una subrutina que realice la multiplicación de dos números es necesario que el programa principal le proporcione los números con los que va a efectuar la operación y que la subrutina, una vez ejecutada la multiplicación, le dé a conocer el resultado de ésta al programa principal.

Hay varias maneras de pasar información entre el programa principal y las subrutinas.
Se pueden transferir datos por medio de:

1. Registros internos.
2. Localidades de memoria reservadas.
3. Apuntadores a listas de parámetros en memoria.
4. La pila.

El método a seguir frecuentemente lo determina la cantidad de datos en cuestión. Cuando el número de datos por transferir es menor que el número de registros internos de propósito general, es conveniente pasar los parámetros a través de los registros. En este caso, las instrucciones que cargan los datos en los registros internos preceden a la instrucción CALL y a su conjunto se le conoce como "secuencia de llamada" (calling sequence). Una vez dentro de la subrutina, ésta obtiene sus parámetros de los registros preestablecidos, y los resultados que genera los coloca en los registros escogidos para esta propósito antes de que la instrucción RET sea ejecutada. El usar registros tiene la ventaja de que la subrutina permanece independiente de la memoria.

Parámetros y resultados también pueden transferirse entre el programa principal y las subrutinas o entre subrutinas, por medio de localidades de memoria reservadas. Una localidad de memoria reservada es aquella que se define en RAM para contener el valor de una variable o parámetro. Hay que tener cuidado de que las instrucciones en la secuencia de llamada y en la subrutina se refieran a la misma localidad. El uso de la memoria tiene la ventaja de una mayor flexibilidad (se puede pasar más información) pero disminuye la eficiencia y asocia a la subrutina con un área de memoria determinada. Cuando una subrutina requiere de un gran número de argumentos, éstos pueden colocarse en la memoria, pero en lugar de pasar toda la información, lo que el programa principal transmite es simplemente un apuntador conteniendo la dirección inicial del bloque de argumentos. El apuntador puede ser un registro interno o una localidad de memoria reservada.

La pila también puede usarse para pasar parámetros. La información requerida por la subrutina es colocada en la pila por el programa principal, utilizando instrucciones PUSH. Cuando la subrutina es llamada, lo que hay en la parte superior de la pila es la dirección de retorno y debajo de ella se encuentran los parámetros de la subrutina. Esta debe sacar los parámetros de la pila dejando la dirección de retorno en la parte superior, lo que complica la manipulación de la pila. Una solución a este problema es que la subrutina saque la dirección de retorno de la pila y la guarde temporalmente en alguna localidad de memoria o en un registro interno. Después de esto, la subrutina puede sacar los parámetros de la pila conforme los vaya necesitando. Cuando todos los parámetros han sido procesados, la subrutina coloca de nuevo la dirección de retorno en la parte superior de la pila, dejándolo listo para cuando ocurra el retorno al programa principal.

La instrucción EX (SP),HL proporciona un medio de obtener parámetros de la pila de uno por uno, dejando la dirección de retorno en la parte superior de la pila. Después de que el control ha sido transferido a la subrutina, la secuencia de instrucciones - POP HL EX (SP),HL - saca la dirección de retorno de la pila y la coloca en el registro par HL; luego intercambia la dirección de retorno en HL con el parámetro que se encuentra en la parte superior de la pila. Como resultado, el valor del parámetro queda en HL y la dirección de retorno vuelve a la parte superior de la pila. Esta secuencia puede repetirse hasta que todos los parámetros se hayan sacado de la pila.

Los resultados de la subrutina se pueden pasar al programa principal con la misma técnica. Antes de que ocurra el retorno al programa principal, la subrutina coloca los resultados en la pila, dejando a la dirección de retorno en la parte superior. La ejecución de la instrucción RET saca la dirección de retorno de la pila. En seguida el programa principal puede obtener los resultados extrayéndolos de la pila. La utilización de la pila para depositar parámetros o resultados tiene la misma ventaja que el uso de registros: es independiente de localidades específicas de memoria. Naturalmente también tiene sus desventajas: llena la pila con datos y, por lo tanto, reduce el número de niveles de anidamiento posibles para las llamadas a subrutinas. También complica el uso de la pila y hasta puede ser necesaria la creación de pilas múltiples.

3.6.6 Tiempos de espera.

En muchas aplicaciones de los μprocesadores resulta necesario generar tiempos de espera con los cuales se define la duración de eventos, se sincroniza la ocurrencia de varias acciones o simplemente se obliga al μprocesador a que no haga nada durante un tiempo. Una forma de generar estos tiempos de espera es por medio de software. Cada instrucción del μprocesador posee una duración o tiempo de ejecución determinado, el cual se mide por el número de estados (ciclos de reloj) que transcurren en su obtención y ejecución. Por ejemplo, la instrucción LD *r,n* dura 7 estados, la instrucción DEC *r* requiere de 4 estados y la instrucción JP NZ,*pq* dura 10 estados.

Los tiempos de espera se pueden implantar con una secuencia de instrucciones, por ejemplo, un lazo o rutina de conteo cuya duración total sea igual al intervalo especificado. Como el tiempo de ejecución de cada instrucción es conocido, lo único que hay que hacer es calcular el valor inicial que debe tener el registro que actúa como contador del número de iteraciones en el lazo, para obtener el tiempo de espera deseado. En la figura 3-25 el registro B se carga inicialmente con un valor inicial N. Después, el registro se decrementa sucesivamente hasta llegar a cero, momento en el cual se termina la ejecución del lazo.

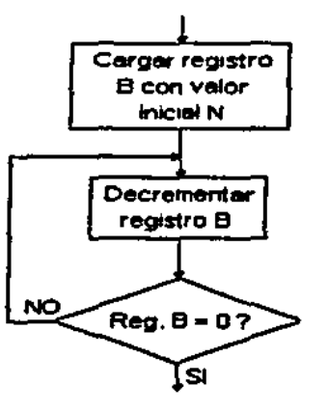


Figura 3-25

Su codificación consta de las siguientes instrucciones (la duración de cada una se halla entre paréntesis):

```
                LD   B,N          (7)
LAZO:           DEC  B            (4)
                JP   NZ,LAZO      (10)
```

La duración T del lazo es función de N y está dada por la fórmula T = N x 14 *ts*, donde 14 es el número de estados que transcurren en el decremento y el salto condicional para cada iteración y *ts* es la duración de un estado.

Existen varios tipos del μprocesador Z-80. El GCM es uno de ellos, trabaja a 2 MHz (Z-80) y el de otro trabaja a 4 MHz (Z-80A). En el primer caso, *ts* = 500 *n*s y en el segundo, *ts* = 250 *n*s. Considerando *ts* = 500 *n*s entonces T = N x 14 x 500 *n*s = N x 7 μs y, por lo tanto, si el registro B se carga con un valor inicial N = 60, la duración del lazo será T = 420 μs. Nótese que la mínima duración del lazo se obtiene cargando el registro B con N = 1, dando por resultado T = 7 μs. Mientras que el máximo tiempo de espera se logra con N = 0. La razón es que al decrementar el registro B por primera vez, su contenido pasa a ser FFH (255) antes del salto condicional, requiriendo un total de 256 iteraciones para que llegue de nuevo al valor de cero; de modo que entonces T = 256 x 7 μs = 1.792 ms.

El lazo se puede integrar como una subrutina para ser llamada desde el programa principal. Por ejemplo, si en un programa se desea contar los milisegundos que dura cierto evento, la porción del programa que se encarga del conteo podría implantarse con una subrutina que genere un tiempo de 1 ms, como se muestra en seguida:

Programa principal

```
.
.
.
CALL ESPERA     (17)
INC  C          (4)
.
.
```

Subrutina de espera

```
ESPERA:   LD   B,N          (7)
LAZO:     DEC  B            (4)
          JP   NZ,LAZO      (10)
          RET               (10)
```

En el programa principal, el registro C es el que se encarga de llevar la cuenta de los milisegundos transcurridos, mientras que en la subrutina de espera, el registro B actúa como contador del número de iteraciones del lazo. Queda por determinar el valor inicial N con el que debe cargarse este registro para que el tiempo que transcurra entre cada incremento del registro C sea de 1 ms. En este caso, al hacer el cálculo, además del lazo básico hay que tomar en cuenta la duración de las instrucciones de llamada y retorno de la subrutina, el incremento del registro C y la inicialización del registro B con N. Así pues, la duración total está dada por T = [17 + 7 + (N x 14) + 10 + 4] *ts*. De donde, para T = 1 ms se puede obtener el valor inicial del registro B a partir de N = (1 ms - 38 *ts*)/14 *ts*. Si T = 500 *n*s, entonces con el valor inicial N = 140 se logra un intervalo de 0.999 ms.

Aún cuando en el ejemplo anterior se haya alcanzado la duración deseada con una precisión muy aceptable, utilizando solo las instrucciones indispensables para implantar el lazo y la subrutina, esto no es lo normal en la mayoría de las ocasiones. Por el contrario, es muy frecuente que se tengan que añadir instrucciones con el único fin de "ajustar" la duración del lazo o rutina de espera al valor deseado. La instrucción NOP es particularmente útil en estos casos, ya que hace gastar tiempo al μprocesador sin alterar el estado de los registros o las banderas. La instrucción NOP ocupa un byte de memoria y tarda 4 estados en realizarse. Los lazos de espera vistos han implicado el uso de un registro de 8 bits que lleva la cuenta del número de iteraciones. Para implantar intervalos más largos se puede utilizar un registro par de 16 bits como contador. La instrucción DEC ss decrementa el contenido del registro par especificado; sin embargo, no afecta ninguna de las banderas. La siguiente rutina muestra la utilización del registro par BC como contador y la forma de detectar la condición cero:

```
        LD   BC,N              ;inicializar al registro par BC con
                               ;un valor de 16 bits
LAZO:DEC BC            (6)     ;decrementar al registro par BC
        LD   A,B       (4)     ;comprobar si B = C = 0H por medio
                               ;de la operación B OR C
        OR   C         (4)
        JP   NZ,LAZO   (10)
```

La duración del lazo es T = N x (6 + 4 + 4 + 10) *ts* = N x 24 *ts*, donde N puede variar desde 0 hasta 65535 (FFFFH). Para *ts* = 500 *n*s el rango de duración del lazo va de 12 μs a 786.432 ms. Al igual que en el lazo con un registro de 8 bits, la duración máxima se obtiene cargando el registro par BC con ceros.

Otra alternativa para la generación de intervalos de mayor duración es la creación de lazos anidados, esto es, poniendo un lazo de espera dentro de otro, como lo demuestra el diagrama de flujo de la figura 3-26. El lazo del registro B tiene N iteraciones y además, por estar dentro del lazo del registro C, se repite M veces, por lo tanto tiene una duración M x N. En seguida se muestra la codificación de los dos lazos anidados.

```
        LD   C,M       (7)
LAZOC:  LD   B,N       (7)
LAZOB:  DEC B          (4)
        JP NZ,LAZOB    (10)
        DEC C          (4)
        JP NZ,LAZOC    (10)
```

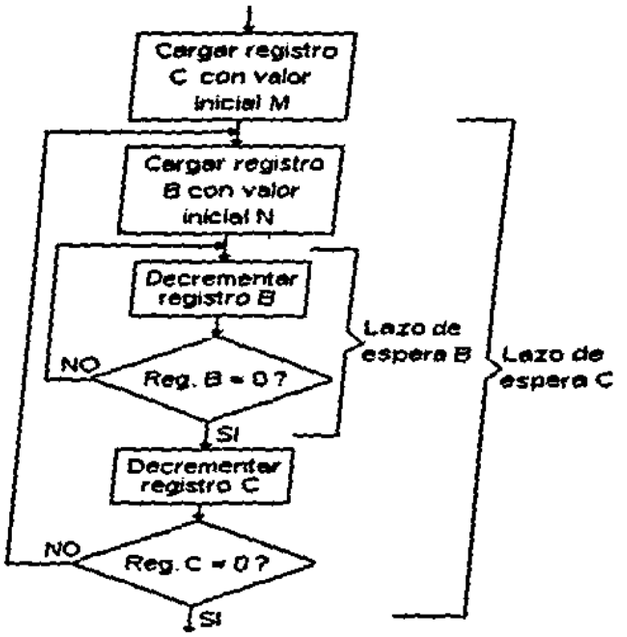


Figura 3-26

El tiempo total de espera es $T = [7 + (14 \times N) + 4 + 10] \times M \times ts$, donde $14 \times N \times ts$ es precisamente la duración del lazo de conteo que se analizó previamente (fig. 3-25).

Tomando $ts = 500$ *ns* y los valores de M y N para duración máxima, se obtiene $T = [21 + (14 \times 256)] \times 256 \times 500$ *ns* = 461.440 ms para los dos lazos anidados.

3.6.7 Modos de direccionamiento.

La mayoría de las instrucciones ejecutadas por un µprocesador requieren datos. La palabra "direccionamiento" se refiere al mecanismo por medio del cual un µprocesador tiene acceso a los lugares donde se encuentran los datos que va a procesar y a los lugares donde se almacenarán los resultados de las operaciones. A las diferentes maneras de implantar el mecanismo de acceso se les llama MODOS DE DIRECCIONAMIENTO. Cada modo proporciona, dentro de una instrucción, la información necesaria para que la CPU pueda localizar los datos sobre los que operará. En la actualidad hay muchos diseñadores que piensan que en un µprocesador la variedad y capacidad de los modos de direccionamiento son más importantes que el número de instrucciones, ya que cambiando el modo de direccionamiento se puede hacer que una misma instrucción trabaje de diversas formas.

A través del estudio del conjunto de instrucciones del Z-80A en los capítulos anteriores, se ha observado que la mayoría de ellas actúan sobre datos almacenados en los registros

internos y en la memoria. Este µprocesador ofrece 10 tipos de direccionamiento, en los cuales están incluidas todas las modalidades de las instrucciones. Estas son:

1. Inmediato
2. Inmediato extendido
3. Implícito
4. Registro
5. Registro indirecto
6. Extendido
7. Relativo
8. Indexado
9. Página cero
10. Bit

A continuación se explicará detalladamente en que consiste cada uno de estos modos.

3.6.7.1 Direccionamiento inmediato.

En este modo de direccionamiento, el byte que sigue (inmediato) al código de operación contiene al operando de la instrucción. Es decir, el dato que se requiere como operando es parte integrante de los bytes que constituyen la instrucción (fig. 3-27). Este tipo de direccionamiento es útil cuando se necesita cargar o efectuar una operación aritmética o lógica con una constante. Por ejemplo, para sumar 21 al acumulador se puede usar la instrucción ADD A,*n*. En este caso el valor inmediato "*n*" es 21.

Figura 3-27

3.6.7.2 Direccionamiento inmediato extendido.

Como su nombre lo indica, este modo de direccionamiento, es una extensión del direccionamiento inmediato, solo que ahora el operando esta formado por los dos bytes que siguen al código de operación (fig. 3-28). Esto permite incluir datos o direcciones de 16 bits. El primero de los dos bytes del dato o dirección contiene los 8 bits menos significativos. Este modo de direccionamiento lo emplean las instrucciones que cargan los registros pares con un valor de 16 bits y las instrucciones de transferencia de control tales como saltos absolutos y llamadas a subrutinas. Por ejemplo, para saltar a la dirección 5678H se puede utilizar la instrucción JP *pq* en donde la dirección "*pq*" es el número de 16 bits 5678H.

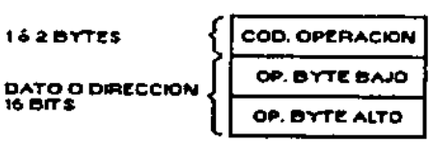


Figura 3-28

3.6.7.3 Direccionamiento implícito.

El direccionamiento implícito se refiere a instrucciones en donde el código de operación automáticamente implica que determinado registro o registros contienen los operandos. El código de operación es fijo, es decir, no tiene campos variables (fig. 3-29). Con este direccionamiento se puede nombrar el conjunto de instrucciones aritméticas y lógicas, donde el acumulador siempre actúa como uno de los operandos y como destino de los resultados. Como ejemplo esta la instrucción CPL que obtiene el complemento a 1 del acumulador.

Figura 3-29

3.6.7.4 Direccionamiento de registro.

En este modo de direccionamiento la instrucción utiliza a los registros internos para recibir o proporcionar un dato. El código de operación tiene un campo o grupo de bits variable por medio del cual se especifica el o los registros involucrados en la operación (fig. 3-30). Las instrucciones que usan este tipo de direccionamiento son las de transferencia de datos, las aritméticas, las lógicas, las de rotación, las de desplazamiento y las de manipulación de bits. Por ejemplo, la instrucción LD *r,r'* ordena que el contenido del registro "*r'*" se cargue en el registro "*r*", en donde "*r*" y "*r'*" pueden ser cualquiera de los registros internos de 8 bits. El código de operación de esta instrucción tiene tres bits para especificar el registro fuente y tres bits para el registro destino. Como los registros internos son muy pocos y están dentro de la CPU, las instrucciones con direccionamiento de registro o implícito son las más rápidas en el Z-80A.

Figura 3-30

3.6.7.5 Direccionamiento de registro indirecto.

Este tipo de direccionamiento emplea el contenido de uno de los registros pares BC, DE o HL como apuntador que contiene la dirección de una localidad de memoria; con esto se puede tener acceso a cualquier localidad (fig. 3-31). Antes de usar instrucciones que manejen este modo de direccionamiento, se debe cargar al registro par con la dirección requerida. La notación para el direccionamiento de registro indirecto se hace encerrando al registro par entre paréntesis. Por ejemplo, la instrucción AND (HL) efectúa la función lógica AND entre le acumulador y la localidad de memoria apuntada por HL, si el contenido de HL es 5FFFH, entonces la localidad de memoria referida en la operación es la que tiene dirección 5FFFH.

Figura 3-31

3.6.7.6 Direccionamiento extendido.

Las instrucciones que utilizan direccionamiento extendido proporcionan en la misma instrucción e inmediatamente después del código de operación la dirección de 16 bits de la localidad de memoria indicada en la operación. Con este tipo de direccionamiento se tiene acceso directo a cualquier localidad de memoria y no es necesario un registro apuntador. En el primer byte de la dirección se guardan los 8 bits menos significativos de ésta y en el segundo byte, los 8 bits más significativos (fig. 3-32).

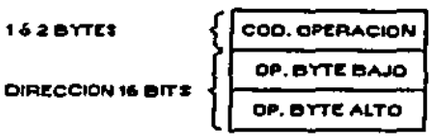


Figura 3-32

La notación "(*pq*)" se usa para indicar el contenido de la localidad de memoria con dirección "*pq*". Por ejemplo, la instrucción LD A,(1200H) carga en el acumulador el contenido de la localidad de memoria 1200H. El direccionamiento extendido es probablemente uno de los que se emplean con más frecuencia para la transferencia de datos entre la memoria y los registros internos vía el acumulador.

3.6.7.7 Direccionamiento relativo.

Las instrucciones de transferencia de control con direccionamiento relativo utilizan el byte que sigue al código de operación para especificar un desplazamiento a partir del valor que tenga el contador del programa en el momento de la ejecución de estas instrucciones; este desplazamiento se añade al contenido del PC para obtener la dirección efectiva a donde ocurrirá la transferencia de control. El valor del desplazamiento se interpreta como un número con signo en complemento a 2, lo cual permite la realización de desplazamientos hacia adelante (positivos) y hacia atrás (negativos). La estructura de una instrucción con direccionamiento relativo se muestra en la figura 3-33.

Figura 3-33

Este tipo de direccionamiento proporciona instrucciones más cortas ya que el desplazamiento ocupa sólo un byte en lugar de los 16 bits que se requieren para representar una dirección. Además, como el desplazamiento equivale a la distancia existente entre la instrucción de transferencia de control y la instrucción a donde se transfiere el control, y esta distancia se mantiene constante cuando se cambia la posición del programa en la memoria, los programas que usan direccionamiento relativo son "relocalizables". Por otro lado, la inserción de nuevas líneas en un programa con instrucciones de direccionamiento relativo se complica, ya que la separación entre las instrucciones clave se puede ver afectada. El Z-80A usa direccionamiento relativo únicamente en las instrucciones de salto, permitiendo saltos condicionales o incondicionales hasta 126 localidades hacia atrás o hasta 129 hacia adelante, si se toma como referencia el valor del PC correspondiente al primer byte de la instrucción. Como ejemplo está la instrucción JR 19H, suponiendo que su código de operación se encuentra en la localidad con dirección 1000H. Considerando que el punto de referencia es 1000H + 2 y que el valor del desplazamiento es 19H, la dirección absoluta de la instrucción a donde se efectúa el salto es 1001H + 19H = 101BH.

3.6.7.8 Direccionamiento indexado.

En este direccionamiento, el byte que sigue al código de operación de la instrucción especifica un desplazamiento. Este desplazamiento está representado como un número de 8 bits en complemento a 2, que se suma a uno de los dos registros índices (IX o IY) para formar la dirección efectiva de la localidad de memoria que se va a usar como operando en la instrucción. El contenido del registro índice no se altera (fig. 3-34).

Figura 3-34

Para representar el direccionamiento indexado se utiliza la notación (IX + d) o (IY + d), donde "d" es el desplazamiento, y los paréntesis indican que el valor calculado es un apuntador de memoria.

Este modo de direccionamiento resulta ser el más apropiado para la construcción de estructuras de datos tales como tablas y listas. Además, al igual que el direccionamiento relativo, el direccionamiento indexado hace que un programa sea relocalizable.

Un ejemplo de direccionamiento indexado es la instrucción LD A,(IX + 7H), que ordena cargar en el acumulador el contenido de la localidad de memoria apuntada por el registro índice más el desplazamiento. Si se considera que en IX hay 13F0H y dado que el desplazamiento especificado es 07H, la dirección de la localidad de memoria será 13F0H + 07H = 13F7H.

3.6.7.9 Direccionamiento en la página cero.

Este modo de direccionamiento es usado únicamente por la instrucción Restart (RST) que es una llamada a una subrutina de un solo byte. Esta instrucción causa que la ejecución del programa continúe en una de ocho localidades específicas en la "página cero" de la memoria (fig. 3-35). En un sistema desarrollado con el Z-80A se define como página el área de memoria que se pueda direccionar con ocho bits, es decir, 256 localidades. De esta manera, las primeras 256 localidades de memoria (direcciones 0000H a 00FFH) constituyen la página cero.

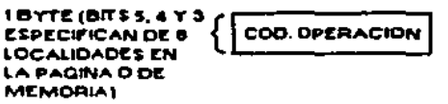


Figura 3-35

La forma en que la instrucción Restart implanta el direccionamiento en la página cero es la siguiente: a la parte alta de la dirección (byte más significativo) le asigna el valor 00H. Con esto se asegura que la dirección final quede confinada dentro de las primeras 256 localidades. Para especificar la parte baja de la dirección (byte menos significativo) utiliza tres bits con los cuales selecciona una de ocho direcciones en la página cero. Estas direcciones son: 0000H, 0008H, 0010H, 0018H, 0020H, 0028H, 0030H y 0038H. La ventaja de este direccionamiento es que permite que con una instrucción de un byte se especifique una dirección completa de 16 bits donde se localizan subrutinas que son llamadas con frecuencia, ahorrando así espacio en memoria. Por ejemplo, la instrucción RST 30H llama a la subrutina que comienza en la dirección 0030H.

3.6.7.10 Direccionamiento de bit.

Aunque no es considerado un modo de direccionamiento auténtico, el direccionamiento de bit se menciona aquí por el gran número de instrucciones que posee el Z-80A para manipulación de bits. Estas instrucciones permiten efectuar operaciones a nivel de bits, sobre los contenidos de localidades de memoria o de los registros internos seleccionados por uno de tres posibles tipos de direccionamiento: registro, registro indirecto e indexado. Tres bits del código de operación de las instrucciones con direccionamiento de bit especifica cuál de los ocho bits del operando (registro o memoria) es el involucrado en la operación (fig. 3-36).

Figura 3-36

Un ejemplo de este modo de direccionamiento es la instrucción SET *b,r* que pone en 1 lógico el bit "*b*" del registro "*r*". Si la instrucción es SET 3,B el bit 3 del registro B toma el

valor de 1, los otros siete bits no sufren alteración. En este caso se usó direccionamiento de registro.

3.7 Otras instrucciones.

En esta sección se continúa con la descripción del conjunto de instrucciones del Z-80A. En particular, aquí se tratan otros grupos de instrucciones básicas como son las de comparación, de rotación, de desplazamiento y de transferencia de control condicional.

3.7.1 Instrucciones de comparación.

La comparación de dos bytes para determinar cuál es mayor en términos de magnitud se puede hacer por medio de las instrucciones de comparación. Estas instrucciones permiten comparar dos datos de 8 bits, uno de los cuales debe estar en el acumulador y el otro en alguno de los registros de propósito general o como dato inmediato contenido en la instrucción. Dos formas de la instrucción de comparación son:

Comparación con dato inmediato	**Comparación con registro**
CP *n* (instrucción de 2 bytes)	CP *r* (instrucción de 1 byte)

donde "*n*" es un dato de 8 bits y "*r*" es uno de los registros de propósito general.

Al ejecutarse alguna de estas instrucciones se realiza una operación funcionalmente similar a la resta, en donde el dato inmediato o el contenido del registro se le resta al acumulador; sin embargo, a diferencia de aquélla la instrucción de comparación no afecta a los operandos, es decir, los registros permanecen con sus valores originales y únicamente se altera el estado de las banderas de acuerdo con el resultado de la resta. Si los datos sujetos a comparación son interpretados como números binarios sin signo (magnitudes), entonces la bandera Z se pone en 1 lógico cuando los datos son iguales; si el contenido del acumulador es mayor o igual que el otro dato, la bandera CY se pone en 0 lógico; cuando el contenido del acumulador es menor, la bandera CY se pone en 1 lógico. Por ejemplo, si el acumulador tiene el dato 13H, el registro B el dato 0FH y el registro C el dato 57H; después de ejecutar la instrucción CP B, Z=0 y CY=0 (AB); después de ejecutar la instrucción CP C, Z=0 y CY=1 (A); y después de ejecutar la instrucción CP 13H Z=1 y CY=0 (A = 13H). En los tres casos anteriores, el contenido de los registros A, B y C no se altera.

Después de ejecutar una instrucción de comparación se puede examinar el registro de banderas para determinar si los datos comparados son iguales o cuál de ellos es mayor. Normalmente esto se hace por medio de las instrucciones de salto condicional.

3.7.2 Instrucciones de rotación y desplazamiento.

El µprocesador Z-80A cuenta con un poderoso grupo de instrucciones que le permiten rotar o desplazar los bits contenidos en el acumulador o en cualquier registro de propósito general (B, C, D, E, H y L). En todas ellas la bandera acarreo (CY) juega un papel muy importante.

Antes de examinarlas, conviene definir los conceptos de rotación y de desplazamiento para aclarar la diferencia que existe entre ellos. Un desplazamiento o corrimiento (shift) consiste en mover el contenido del registro una posición a la derecha o a la izquierda. Es decir, se toma el valor del bit en cada posición y se transfiere a la posición de un bit más arriba o de un bit más abajo. Existen dos tipos de desplazamiento: desplazamiento lógico y desplazamiento aritmético. En ambos, el bit que sale por un lado del registro pasa a la bandera acarreo, en tanto que por el otro lado entra un 0, excepto en el caso de un desplazamiento a la derecha en donde el bit que entra al registro es igual al valor del bit más significativo. Esta diferencia surge porque el desplazamiento aritmético mantiene el signo del número, considerando su representación en complemento a dos, mientras que el desplazamiento lógico no lo toma en cuenta.

Una rotación (rotate) consiste en un desplazamiento en el cual el bit que sale del registro es el mismo que entra por el otro lado. Hay dos clases de rotación: rotación de 9 bits y rotación de 8 bits o circular. La rotación de 9 bits trata a la bandera acarreo como un bit más del registro y, por lo tanto, la rotación se efectúa en 9 bits. La rotación circular únicamente toma en cuenta los 8 bits del registro al realizar la operación; sin embargo, el bit que sale de un lado del registro no solo se reinserta en el otro extremo, sino que una copia de su valor se coloca en la bandera acarreo. En síntesis, una rotación difiere de un desplazamiento por el hecho de que el bit que entra en el registro es el que salió por el otro extremo o es el contenido de la bandera acarreo, en lugar de un valor externo 0 o 1.

El Z-80A tiene 4 instrucciones para la rotación del contenido de los registros internos de propósito general; sus mnemónicos son:

Rotación a la izquierda	**Rotación a la derecha**
RL *r*	RR *r*
RLC *r*	RRC *r*

donde "*r*" puede ser cualquiera de los registros A, B, C, D, E, H y L. La cuatro instrucciones tienen un código de operación de 2 bytes. Las instrucciones RL *r* y RR *r* implantan una rotación de 9 bits, girando el contenido del registro a través de la bandera acarreo. Las instrucciones RLC *r* y RRC *r* llevan a cabo una rotación circular de 8 bits.

Además de las cuatro instrucciones anteriores, existen otras cuatro que operan únicamente sobre el acumulador; sus mnemónicos son:

Rotación a la izquierda	**Rotación a la derecha**
RLA	RRA
RCA	RRCA

Estas instrucciones realizan con el acumulador rotaciones equivalentes a las que se describieron en el párrafo anterior para cualquier registro interno de propósito general.

La razón para esta aparente duplicación de funciones, dado que el acumulador ya está incluido en las instrucciones que operan sobre los registros, es mantener la compatibilidad con el conjunto de instrucciones del μprocesador 8080. Además, las instrucciones de rotación exclusiva del acumulador tienen código de operación de un byte y, por lo tanto, son más eficientes que las de rotación de un registro.

Las instrucciones de rotación se usan generalmente para aislar y después examinar un bit particular de un dato. Por ejemplo, si se quiere separar el bit más significativo del registro D que tiene el dato 19H, se puede hacer por medio de la instrucción RLC D. Después de la ejecución, en el registro D habrá 32H y la bandera acarreo tendrá un 0 que es el valor del bit más significativo. Si lo que se desea es aislar al bit menos significativo del acumulador cuyo contenido es F7H, se puede utilizar la instrucción RRC A. Posteriormente a su ejecución, el acumulador tendrá FBH y en la bandera acarreo habrá un 1 que es el valor del bit menos significativo.

En el Z-80A existen tres instrucciones que permiten realizar desplazamientos en los registros de propósito general; sus mnemónicos son:

Desplazamiento a la izquierda	**Desplazamiento a la derecha**
SLA *r*	SRA *r*
	SRL *r*

donde "*r*" puede ser cualquiera de los registros A, B, C, D, E, H y L. Todas ellas tienen un código de operación de 2 bytes.

La instrucción SLA *r* sirve para efectuar un desplazamiento aritmético o lógico a la izquierda del dato contenido en el registro "*r*"; el bit 7 pasa a la bandera acarreo y el bit 0 recibe un cero.

FALTA PAGINA

No. 80

en la que "*pq*" es la dirección a donde se efectúa el brinco en caso de que la condición sea verdadera, y "*cc*" es una de las siguientes ocho condiciones:

NZ:	no cero (Z=0)
Z:	cero (Z=1)
NC:	sin acarreo (CY=0)
C:	con acarreo (CY=1)
PO:	paridad non (P=0)
PE:	paridad par (P=1)
P:	signo positivo (S=0)
M:	signo negativo (S=1)

La instrucción JP *cc,pq* ocupa tres bytes, el primero con el código de operación, el segundo con la parte baja de la dirección y el tercero con la parte alta de la misma.

Es importante entender el significado de cada una de las condiciones anteriormente expuestas para no caer en errores de programación. Note que las condiciones se refieren a alguna característica del resultado de una operación reflejada en las bandera y no a las banderas mismas.

La condición "Z" es verdadera si el resultado de una instrucción es 0 y entonces la bandera Z es 1, mientras que la condición "NZ" es verdadera cuando el resultado de una instrucción es diferente de cero y en este caso la bandera Z es 0.

La condición "C" se cumple cuando el resultado de una instrucción produce un acarreo con valor 1 (CY=1), mientras que la condición "NC" se da cuando el resultado de una instrucción no produce acarreo (CY=0).

La condición "PO" es satisfecha cuando el resultado de una instrucción tiene un número non (impar) de bits con nivel 1 o si no hay sobreflujo (P/V=0), mientras que la condición "PE" es verdadera siempre que el resultado de una instrucción tenga un número par de bits con nivel 1 o si existe desbordamiento (P/V=1).

La condición "P" se produce cuando el resultado de una instrucción es positivo (S=0), mientras que la condición "M" aparece si el resultado de una instrucción es negativo (S=1). Un número se considera positivo si el bit 7 es 0 y negativo si es 1.

Un ejemplo de las instrucciones de salto condicional absoluto es el siguiente. Si por alguna operación previa la bandera acarreo está en 0 lógico y se ejecuta la instrucción JP NC, 0B3AH, al fin de su ejecución el control del programa se habrá transferido a la instrucción que se halle en la dirección 0B3AH.

Las instrucciones de salto condicional relativo, al igual que las de salto incondicional relativo, no contienen una dirección sino únicamente la distancia en bytes que separa a la instrucción de salto de la instrucción a donde se desea transferir el control. El mnemónico de la instrucción de salto condicional relativo es:

JR *cc,e*

en la que "*e*" representa el valor de la diferencia entre la dirección adonde se va a saltar y el valor que tiene el PC al iniciar la ejecución de la instrucción, es decir, la dirección del primer byte de la instrucción de salto más dos; "*cc*" es la condición.

Estas instrucciones solo pueden examinar el estado de las banderas cero y acarreo (CY). Por lo tanto, hay cuatro condiciones posibles: NZ, Z, NC y C. Su representación binaria es de dos bytes, el primero tiene el código de operación y el segundo el valor del desplazamiento "*e*" expresado como un número de ocho bits en complemento a dos. Por ejemplo, si se quiere saltar a una instrucción que se encuentra 16 bytes después, cuando la bandera cero sea 1 lógico, se puede utilizar la instrucción JR Z,0EH.

Las instrucciones de salto condicional frecuentemente se usan como elementos de decisión en la construcción de estructuras de programación básicas. La estructura de programación más sencilla consiste en uno o más elementos de proceso en secuencia con una sola entrada y una sola salida (figura 3.37a).

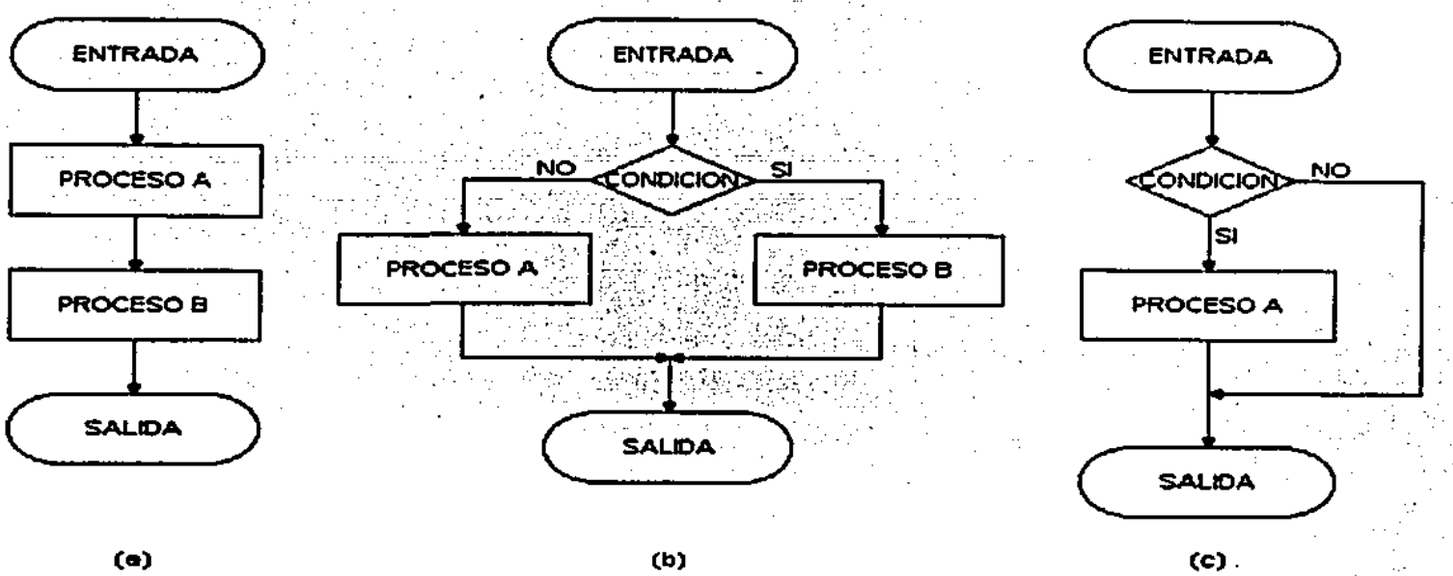


Figura 3-37

El elemento de decisión se puede combinar con uno o dos elementos de proceso secuencial para crear la estructura denominada IF-THEN/ELSE, como se muestra en la figura 3.37b. A partir de la condición examinada, se ejecuta uno de los dos procesos. Si uno de ellos es un proceso vacío, se llega a una forma más simple de la estructura IF-THEN/ELSE, llamada estructura IF-THEN (figura 3.37c).

3.7.4 Lazos.

En programación, un LAZO es una secuencia de instrucciones que se ejecuta repetitivamente. Una instrucción de transferencia de control al final del lazo reanuda la ejecución al comienzo de la secuencia. Las instrucciones de salto condicional proporcionan un medio para realizar un lazo el numero de veces deseado y luego salirse de él, continuando con el resto del programa.

Existen dos estructuras de lazo básicas: la estructura DO WHILE y la estructura DO UNTIL, ambas mostradas en la figura 3.38. En la estructura DO WHILE primero se prueba la condición que gobierna la terminación del lazo y, dependiendo del resultado obtenido, el proceso se ejecuta o se abandona el lazo. Con esta estructura es posible entrar y salir del lazo sin haber ejecutado el proceso. En la estructura DO UNTIL, cuando se entra al lazo primero se ejecuta el proceso, y después se prueba la condición de terminación. Por lo tanto, el proceso siempre se ejecuta cuando menos una vez.

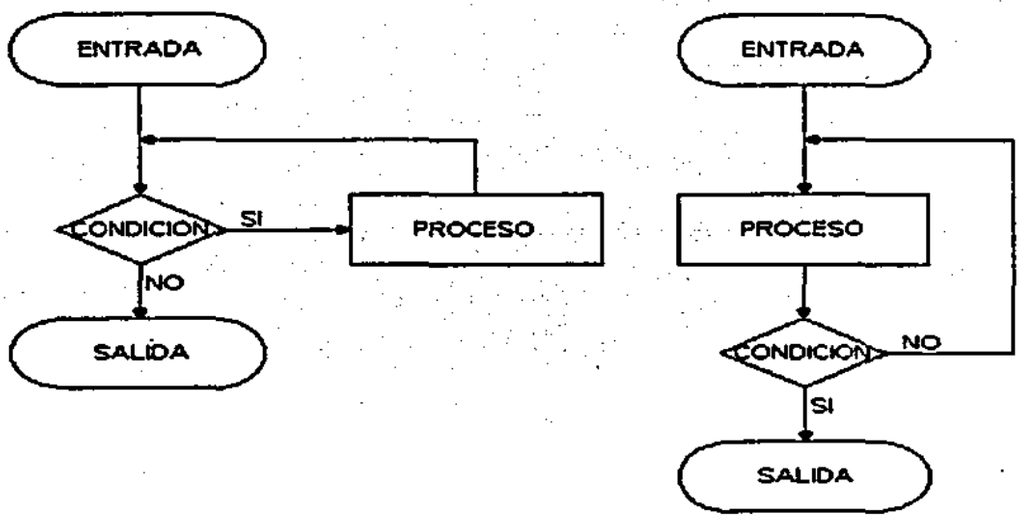


Figura 3-38

Hay dos métodos para controlar el número de veces que se ejecuta un lazo. El primero cuenta el número de iteraciones del lazo. El segundo espera la ocurrencia de un evento específico y, cuando éste se da, el lazo se abandona. Como al contar el número de repeticiones del lazo también se requiere esperar la ocurrencia de un evento -llegar al número de veces deseado-, los métodos pueden parecer equivalentes a primera vista. Sin embargo, en el primero se conoce el número de ejecuciones antes de entrar al lazo, mientras que en el segundo, no.

Al primer método se le denomina CONTEO y requiere que, antes de entrar al lazo, el registro que actúa como CONTADOR sea cargado con un valor inicial, el cual es incrementado o decrementado cada vez que el lazo se ejecuta. Por medio de una instrucción de salto condicional se detecta cuando el contador llega a su valor final y en ese momento se abandona el lazo.

Cualquiera de las dos estructuras mencionadas puede ser utilizada en un lazo de conteo. En este caso la estructura DO WHILE es mejor conocida como "cuenta - luego ejecuta" y la estructura DO UNTIL es llamada "ejecuta - luego cuenta". La selección de la estructura determina el valor inicial del registro contador.

Dentro de las instrucciones del Z-80A aplicables a la construcción de lazos están las instrucciones INC y DEC, con las que se puede incrementar o decrementar el contador del lazo. También se tienen las instrucciones de salto condicional JP *cc* y JR *cc* con las cuales se puede transferir el control a otra parte del programa o del lazo, dependiendo del estado

de alguna de las banderas del μprocesador. En general, el contador se carga inicialmente con el número de veces que se va a ejecutar el lazo y se va decrementando conforme se repite la secuencia de instrucciones hasta que llega a cero; esta condición se detecta fácilmente por medio de las instrucciones JP NZ o JR NZ que examinan el estado de la bandera Z, de modo que si el contador no es cero todavía, se efectúa el salto y el lazo se vuelve a ejecutar.

Además de las anteriores, el μprocesador Z-80A posee una instrucción especial que agrupa el decremento del contador y el salto condicional dependiente de la bandera Z en un mismo código de operación, simplificando con ello la implantación de lazos con conteo. El mnemónico de esta instrucción es:

DJNZ *e*

Funciona de la siguiente manera: se asocia como contador específico el registro B; cada vez que se ejecuta la instrucción, dicho registro se decrementa en uno (DEC D), y en seguida, para determinar si su contenido es cero, se prueba el estado de la bandera Z por medio de un salto relativo condicional (JR NZ,*e*), con la cual se repite la ejecución del lazo en tanto el registro B no alcance el valor de cero; cuando esto sucede se da por terminado el lazo, y la ejecución del programa prosigue secuencialmente. La instrucción ocupa dos bytes, el primero con el código de operación y el segundo con el desplazamiento "*e*", que se calcula de acuerdo a las reglas establecidas para las instrucciones de salto relativo.

El siguiente lazo con estructura DO UNTIL realiza la suma de los nueve primeros números enteros consecutivos:

```
0100   LD B,09H
0102   LD C,01H
0104   XOR A
0105   ADD A,C
0106   INC C
0107   DJNZ 0105H
```

El mismo proceso codificado con estructura DO WHILE requiere un valor inicial de 10 para sumar los nueve números:

```
0100   LD B,0AH
0102   LD C,01H
0104   XOR A
0105   DEC B
0106   JR Z,010BH
0108   ADD A,C
0109   INC C
010A   JR 0105H
```

En otras palabras, para ejecutar N veces un lazo usando una estructura DO UNTIL se necesita un valor inicial N, mientras que para repetir N veces un lazo con estructura DO WHILE se requiere un valor inicial N+1. Además, se observa que con esta última estructura no es posible utilizar la instrucción DJNZ *e*.

Si no se utiliza el método de conteo, el lazo debe terminar con alguna condición o símbolo especial que se haya escogido de antemano. En el siguiente ejemplo se toma el contenido del registro D, rotándolo a la izquierda las veces necesarias para que su valor sea mayor que 80H. Así pues, el número de repeticiones del lazo varía dependiendo del contenido inicial del registro D y la condición de salida del lazo se establece por medio de una comparación con 80H y de un salto condicional dependiente de la bandera CY.

```
0100   LD A,D
0101   RLC A
0102   CP 80H
0104   JR C,0101H
```

3.7.5 Los registros pares.

El µprocesador Z-80A tiene la capacidad de agrupar los seis registros internos de propósito general de 8 bits, para formar tres registros de 16 bits llamados, REGISTROS PARES. Las parejas se hacen de la siguiente manera: B con C, D con E y H con L. De este modo se constituyen los registros pares representados con las letras BC, DE y HL.

La característica principal de los registros pares es que pueden funcionar como una unidad de 16 bits en determinadas instrucciones, pero sin que los registros del par pierdan su identidad como registros de ocho bits. Los registros de la izquierda en cada pareja (B, D y H) alojan a los ocho bits más significativos (byte alto) del registro par, mientras que los registros de la derecha (C, E y L) guardan a los ocho bits menos significativos (byte bajo).

La combinación del acumulador (A) y del registro de banderas (F) también forma un registro de 16 bits, el registro AF; sin embargo no se considera un registro par, ya que realmente no opera como una unidad de 16 bits y su construcción sólo tiene sentido en unas pocas instrucciones especializadas.

Dentro de la arquitectura del Z-80A los registros pares tienen tres propósitos fundamentales:

1. Agilizar la transferencia de información.
2. Aumentar el poder de procesamiento aritmético.
3. Proporcionar medios de acceso a la memoria.

Las instrucciones que cumplen los dos primeros propósitos son tratadas en esta sección y aquellas que se refieren al tercero se dejan para la siguiente sección.
El Z-80A tiene una instrucción que permite cargar directamente un dato de 16 bits en cualquiera de los registros pares; su mnemónico es:

LD *ss*,*nn*

Donde "*ss*" es uno de los registros pares y "*nn*" es un valor de 16 bits. La instrucción consta de tres bytes.

El primero es el código de operación en el cual, por medio de un campo variable de 2 bits, se especifica el registro par (BC, DE o HL).

El segundo byte contiene la parte baja del número de 16 bits, es decir, el dato que se carga en el registro de más bajo orden del registro par especificado (C, E o L).

El tercero contiene la parte alta del número de 16 bits, o sea, el dato que se carga en el registro de más alto orden del registro par especificado (B, D o H).

Por ejemplo, después de ejecutar la instrucción LD BC,1234H el registro B se carga con 12H y el registro C con 34H. La ventaja de esta instrucción es que dos registros se cargan simultáneamente con un valor inicial.

Dado que la capacidad de procesamiento aritmético es una característica importante de cualquier μprocesador de propósito general como el Z-80A, este, aún cuando es de 8 bits, cuenta con instrucciones que le permiten realizar algunas operaciones con datos de 16 bits.

Se ha observado que en las instrucciones aritméticas el registro más importante es el que desempeña el papel de acumulador, ya que sirve tanto de operando como de lugar de almacenamiento para el resultado.

El acumulador normal de la ALU (registro A) no se puede usar en las operaciones de 16 bits debido a que es un registro de 8 bits, de ahí que sea necesario definir un registro de 16 bits que actúe como acumulador. Es así que en estos casos el registro par HL actúa como acumulador.

Existen tres instrucciones para sumar, sumar con acarreo y restar con préstamo al contenido del registro HL el dato de 16 bits de cualquiera de los registros BC, DE o HL. El resultado de la operación se almacena en el registro HL. Los mnemónicos de estas instrucciones son respectivamente:

ADD HL,*ss* ADC HL,*ss* SBC HL,*ss*

en donde "*ss*" es uno de los registros pares. Cualquiera de ellas pone la bandera CY en 1 lógico cuando el resultado no puede ser contenido en 16 bits, es decir, cuando es mayor que FFFFH. La bandera Z únicamente es afectada por las instrucciones ADC HL,*ss* y SBC HL,*ss*. Por ejemplo, si el registro DE tiene el dato 214CH el contenido del registro HL es 034AH y estos se suman con la instrucción ADD HL,DE; al finalizar la ejecución el contenido del registro DE será 214CH, el registro HL tendrá el resultado 2496H, la bandera CY estará en 0 lógico y la bandera Z no se verá afectada.

Como la única instrucción disponible para resta de 16 bits es SBC HL,*ss* que afecta a la bandera CY, es necesario asegurarse de que ésta tenga nivel 0 al ejecutarse la primera instrucción SBS; para esto se puede utilizar la instrucción OR A, la cual pone la bandera CY en 0 lógico sin afectar el contenido del acumulador.

Hay también dos instrucciones que incrementan y decrementan en uno el contenido el contenido de un registro par; sus mnemónicos son:

INC *ss* DEC *ss*

Al igual que en las anteriores "*ss*" puede ser cualquiera de los registros pares BC, DE o HL. Estas instrucciones permiten implantar fácilmente contadores de 16 bits para lazos; sin embargo, es importante notar que su ejecución no altera los valores de las banderas. Debido a esta característica, no se puede probar directamente (por medio de un salto condicional) si el contador alcanzó el valor cero, habiendo necesidad de recurrir a trucos de programación.

La forma más usual para determinar si es cero el contenido de un registro par es realizando una función OR entre sus dos registros; el resultado de la función es cero únicamente cuando el contenido de ambos registros es cero; además, el estado de las banderas se establece de acuerdo al resultado, permitiendo la toma de decisiones a través de saltos condicionales.

A continuación se muestra un ejemplo:

```
1800 LD BC, 4000H
.
.
.
180A INICIO DE LAZO
.
.
.
```

```
1830 DEC BC
1831 LD A,B
1832 OR C
1833 JP NZ, 180AH
```

Finalmente, existe una instrucción de un byte en la cual el contenido del registro par HL se utiliza para cambiar la secuencia de ejecución de un programa. Esta instrucción se denomina BRINCO INDIRECTO y su mnemónico es:

JP (HL)

donde "(HL)" significa "el contenido del registro par HL". En este caso, el contenido del registro HL es una dirección, y lo que hace la instrucción JP (HL) es cargarlo en el PC, transfiriéndose el control a la instrucción con esa dirección, y a partir de ella continúa la ejecución del programa. Por ejemplo, si HL tiene 2040H y ejecuta la instrucción JP (HL), el programa continuará corriendo a partir de la dirección 2040H.

La instrucción JP (HL) es bastante poderosa y muy útil en la creación de una ESTRUCTURA DE SELECCION, la cual es una estructura de programación en la que se selecciona uno de varios procesos alternativos, dependiendo del valor de una palabra de control.

3.7.6 Instrucciones con la memoria.

La memoria de un µprocesador se puede considerar como una colección de registros direccionables. Los registros que se encuentran dentro de la CPU se denominan registros internos, mientras que las localidades de memoria se llaman registros externos. En tanto que los registros internos se identifican con un nombre, A, B, C, D, E, H y L, los registros externos se identifican con una dirección.

Como se mencionó anteriormente, el Z-80A tiene capacidad de direccionar hasta 65,536 localidades de memoria (64 KB) de 8 bits cada una, ya que cuenta con un bus de direcciones de 16 líneas y con un bus de datos de 8 líneas.

El término "instrucciones con la memoria" se aplica al grupo de instrucciones que trasladan datos entre la memoria y los registros internos o que ejecutan operaciones que afectan a éstos y a aquélla.

3.7.6.1 Instrucciones de transferencia de datos.

Uno de los modos más simples de transferir un dato desde o hacia la memoria es por medio de una instrucción en la que la dirección de la localidad de memoria sea parte de la

instrucción. El Z-80A posee un grupo de instrucciones con esta característica, dentro del cual primeramente está la instrucción que carga el acumulador con el contenido de una localidad de memoria; su mnemónico es:

LD A,(*pq*)

donde "*pq*" es un número de 16 bits que representa la dirección de la localidad de memoria. La instrucción tiene tres bytes, el primero es el código de operación, el segundo es la parte baja de la dirección y el tercero es la parte alta de la misma.

Para transferir un dato del acumulador a la memoria se tiene la instrucción:

LD (*pq*),A

que realiza la operación inversa a la instrucción anterior y también es de tres bytes.

Sin embargo, este tipo de transferencia de información entre el μprocesador y la memoria no está limitado únicamente al acumulador ni tampoco a un solo byte de datos. Existen cuatro instrucciones que trabajan con los registros pares, las cuales permiten la transferencia de dos bytes en una sola operación. Sus mnemónicos son:

LD *ss*,(*pq*)	LD (*pq*),*ss*
LD HL,(*pq*)	LD (*pq*),HL

Con la instrucción LD *ss*,(*pq*) dos bytes almacenados en localidades consecutivas de memoria se pueden trasladar al registro par especificado en "*ss*", donde "*ss*" puede ser sustituido por BC, DE o HL. La dirección "*pq*" contenida en la instrucción especifica la localización del dato que se transfiere al registro menos significativo del par. En el registro más significativo de l par se carga el contenido de la localidad de memoria, cuya dirección es la siguiente a la dada en la instrucción.

La instrucción LD (*pq*),*ss* transfiere el contenido de un registro par a dos localidades consecutivas de memoria. El dato del registro menos significativo pasa a la localidad apuntada por la dirección que se indica en la instrucción y el dato del registro más significativo se carga en la localidad siguiente. Ambas instrucciones son de cuatro bytes, los dos primeros están ocupados por el código de operación y los dos últimos por los 16 bits de la dirección.

Las instrucciones LD HL,(*pq*) y LD (*pq*),HL también sirven para transferir dos bytes entre la memoria y un registro par. Por un lado, son más cortas que las dos anteriores porque ocupan tres bytes en lugar de cuatro, pero más limitadas porque están restringidas al registro par HL.

Para mostrar este grupo de instrucciones está el siguiente ejemplo: considere que el acumulador tiene 27H, el registro BC tiene 4F68H, el registro HL tiene 3315H, el contenido de la localidad de memoria 2B00H es C9H y el de la 2B01H es 45H. Después de que se ejecuta la secuencia siguiente:

```
LD (2B01H),A

LD BC,(2B00H)

LD (2B00H),HL
```

El acumulador queda con 27H, el registro BC con 27C9H, el registro HL con 3315H, la dirección 2B00H con 15H y la dirección 2B01H con 33H.

Otra manera de implantar el direccionamiento de la memoria para la transferencia de información es que la dirección de la localidad esté contenida en un registro. Así, la instrucción únicamente tiene que especificar el registro donde se encuentra la dirección, en lugar de la dirección misma, lo cual representa un ahorro en el número de bytes de la instrucción.

Al registro en el que está la dirección se le llama APUNTADOR, ya que su contenido "apunta" a una localidad de memoria. En el caso del Z-80A, para tener acceso a cualquier localidad, se requiere un registro de 16 bits y es aquí donde los registros pares cumplen otra de sus funciones: como apuntadores de memoria.

De éstos, el registro HL es el que tiene mayor versatilidad como apuntador y un gran número de instrucciones lo utilizan con este propósito. Entre ellas están las que transfieren un dato entre una localidad de memoria y uno de los registros de propósito general, y la que carga una localidad de memoria con un valor de 8 bits. Sus respectivos mnemónicos son:

LD *r*,(HL) LD (HL),*r* LD (HL),*n*

donde "*r*" puede ser cualquiera de los registros A, B, C, D, E, H o L y "*n*" es un número de 8 bits. Las dos primeras son de un byte y la tercera es de dos bytes (el segundo byte es el número "*n*"). En los mnemónicos de estas instrucciones el nombre del registro HL se pone entre paréntesis "(HL)" para indicar "la localidad de memoria apuntada por la dirección contenida en el registro par HL". El registro H proporciona los 8 bits más significativos de la dirección (dirección alta) y el registro L proporciona los 8 bits menos significativos

(dirección baja). Por ejemplo, supóngase que el registro H tiene 25H, el registro L tiene 0AH y que se ejecutan las instrucciones:

```
LD (HL),99H        LD E,(HL)
```

Después de la ejecución, el contenido de la localidad de memoria con dirección 250AH, lo mismo que el del registro E, será 99H.

Si se comparan los códigos de las instrucciones LD *r*,(HL) y LD (HL),*r* con el de la instrucción LD *r*,*r´*, se observa que son semejantes; lo mismo ocurre al comparar el código de la instrucción LD (HL),*n* con el de la instrucción LD *r*,*n*. La razón es que las instrucciones con la memoria se pueden obtener de aquéllas con los registros si en el campo de tres bits que designa al registro (según sea el caso), se coloca el código 110 (6) que es precisamente el que se tiene asignado a una localidad de memoria cuando está apuntada por HL, tratándola como si fuera un registro externo. De esta manera, la tabla de códigos de registros es la siguiente:

CODIGO	REGISTRO
000	B
001	C
010	D
011	E
100	H
101	L
111	A

Los registros pares BC y DE también pueden usarse como apuntadores de memoria, pero la transferencia de datos se puede realizar únicamente entre el acumulador y una localidad de memoria. Las instrucciones con estos registros son:

LD A,(*ss*) LD (*ss*),A

Donde "*ss*" puede ser sustituido por cualquiera de los registros BC o DE. Ambas instrucciones son de un byte, en ellas los registros B y D contienen el byte de mayor orden de la dirección y los registros C y E contienen el byte de menor orden de la misma. Por ejemplo, si el contenido del registro A es 56H, el del registro B es 43H y el del registro C es 12H, la instrucción LD (BC),A carga el dato 56H en la localidad de memoria con dirección 4312H.

La instrucción LD *ss*,*nn* es de gran utilidad para cargar a los registros pares con las direcciones de interés antes de usar cualquiera de las instrucciones que los emplean como apuntadores de memoria.

CAPITULO 4

CIRCUITOS DE MEMORIA

Una consideración muy importante en cualquier sistema que tenga un µprocesador es la memoria. Tanto las instrucciones del programa como los datos, deben estar almacenados en algún tipo de memoria, de donde puedan ser obtenidos en el momento apropiado para que el µprocesador lleve a cabo su función. Aun cuando el Z-80 posee un conjunto de registros internos, éstos no pueden usarse para guardar instrucciones, sino únicamente para el almacenamiento temporal de datos; además, su número es muy reducido. Por lo tanto, surge la necesidad de interconectar al µprocesador con circuitos externos de memoria.

En este capítulo se examinará la operación de diferentes tipos de memoria, las técnicas para su interconexión y sincronización con un µprocesador, así como el diseño de sistemas de memoria.

4.1 Tipos de memoria.

La memoria externa se puede dividir en dos grandes categorías: memoria de lectura solamente o ROM (read only memory), y memoria de lectura y escritura o RWM (read/write memory), mejor conocida como RAM. La ROM se utiliza para guardar pasos de un programa o datos que no requieran modificación. El contenido de estas localidades de memoria se considera permanente, y sólo en algunos casos y con dispositivos especiales puede ser alterado. Por otra parte, la RAM se usa para almacenar información que cambia constantemente mientras el µprocesador está en operación, como en el caso de los resultados de cálculos efectuados por el programa. A su vez, la memoria de lectura y escritura se divide en dos clases: RAM estática, que guarda la información mientras no se desconecte la fuente de alimentación; y la RAM dinámica, que necesita de un refrescado periódico para mantener la información.

Independientemente de sus características particulares, la función de cualquier memoria es la misma: proporcionar, cuando el µprocesador lo requiera, ya sea una instrucción para ser ejecutada, o una localidad donde se pueda almacenar un dato.

4.2 Características de la interconexión con la memoria.

Los µprocesadores se comunican con la memoria a través de sus buses de direcciones, datos y control. El acceso a una localidad de memoria se hace por medio de una dirección. Para cada operación con memoria, el µprocesador especifica en el bus de direcciones la ubicación de la localidad de memoria con la que se va a efectuar la transferencia de información. Además, por medio de las señales del bus de control, el µprocesador indica el sentido del flujo de la información en el bus de datos; es decir, si se trata de una operación de lectura (transferencia de un dato de la memoria al µprocesador) o de una operación de

escritura (transferencia de un dato del µprocesador a la memoria). Finalmente, el µprocesador cuenta con medios para distinguir entre una transferencia con la memoria y una transferencia con los puertos de entrada/salida.

Por otra parte, para conectarse al µprocesador, un circuito de memoria tiene:

- Entradas de direcciones.
- (Entradas)/salidas de datos.
- Entradas de control para seleccionar el circuito (chip select).
- Entradas de control para inhabilitar (output disable) o habilitar (output enable) los buffers de tercer estado de las salidas.
- Además, en las memorias RAM existe una entrada de control para especificar la lectura o escritura (R/W) de la memoria.

La figura 4-1 muestra un circuito de memoria común, y la figura 4-2 muestra las líneas de interconexión más comunes entre el µprocesador y la memoria.

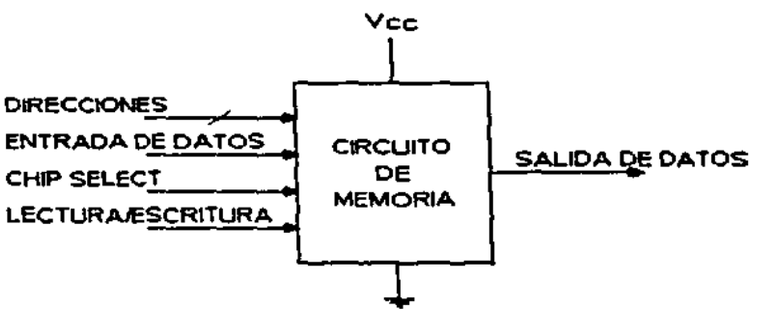


Figura 4-1

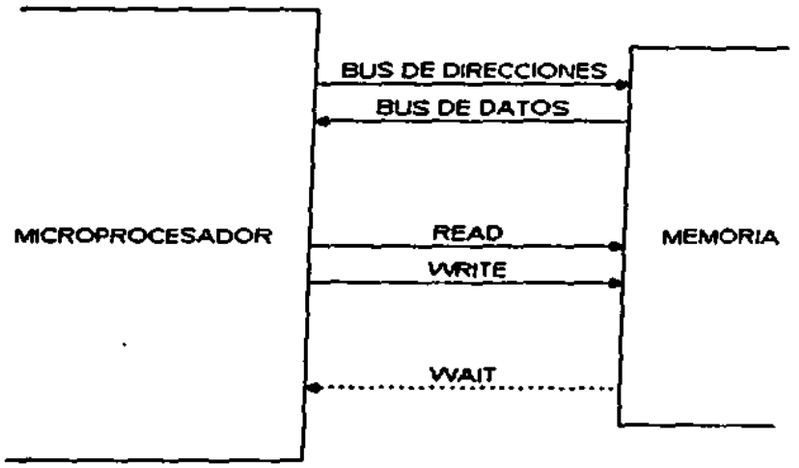


Figura 4-2

4.3 Operación de los circuitos de memoria.

Para asegurar la operación correcta de una memoria, existen restricciones de tiempo en la secuencia que deben seguir las direcciones, datos y señales de control. Estas restricciones están marcadas en las hojas de especificaciones del fabricante, como parte de las características de AC y en los diagramas de tiempos.

El tipo de memoria más simple, en términos de su operación, es la ROM. En la operación básica de una ROM:

1. Se aplica una dirección a las entradas de dirección de la ROM.
2. Se selecciona el circuito de la ROM, activando sus entradas de selección de circuito (chip select, CS).
3. El contenido de la localidad de memoria seleccionada aparece en las salidas de datos de la ROM, después de un periodo igual a su tiempo de acceso.

Suponiendo que un circuito de memoria es seleccionado con el nivel lógico apropiado en su entrada de selección (chip select), el tiempo transcurrido desde la subsecuente aplicación de una dirección en sus entradas de dirección hasta la aparición, a la salida de la memoria, de una copia estable del dato seleccionado, es el tiempo de acceso t_A.

Si existe un valor estable en las entradas de dirección de una ROM, y la entrada de selección (chip select) se activa para seleccionar la ROM; el retraso entre la activación de las señales de selección (chip select) apropiada y la estabilización del dato a la salida, es t_{CO} o tiempo de acceso de CS select. Estos dos parámetros, t_A y t_{CO}, se muestran en el

diagrama de tiempos de la figura 4-3. El diagrama muestra la aplicación de las señales de dirección y de selección (chip select) al tiempo correcto, para que sus efectos a la salida ocurran simultáneamente. El punto de referencia es la aparición de datos válidos a la salida. Como se ve en el diagrama, t_{CO} es generalmente menor que t_A. Esto se debe a que la lógica de selección (chip select) está conectada directamente a los buffers de salida.

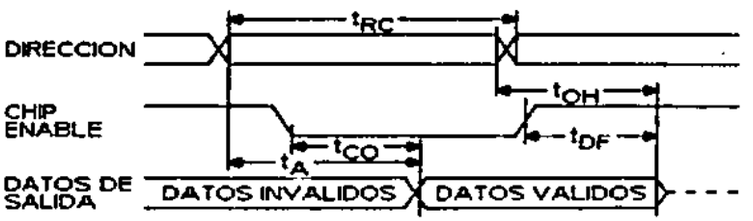


Figura 4-3

Hay otros tres parámetros que se muestran en el diagrama de tiempos: dos de ellos son el tiempo de sostenimiento de la salida (output hold time), t_{OH}, y el tiempo que transcurre desde la desactivación de la línea de selección (chip select) hasta que las salidas flotan (chip deselect to output float time), t_{DF}. Estos parámetros están referidos al instante en que el dato de salida válido pasa a ser un dato inválido, o las líneas de datos se ponen a flotar. El t_{OH} indica cuánto tiempo es todavía válido el dato de salida después de que la dirección ha cambiado. El t_{DF} indica el tiempo que permanece válido el dato de salida cuando se ha dejado de seleccionar el circuito de memoria. El tercer parámetro, tiempo del ciclo de lectura (read cycle time), t_{RC}, especifica la velocidad máxima a la cual, diferentes localidades de memoria pueden ser leídas sucesivamente.

Como se indicó anteriormente, existen dos tipos de memorias RAM: estáticas y dinámicas. Las memorias estáticas tienen los requisitos de operación más sencillos. Además de las líneas de dirección, habilitación de chip (chip enable) y datos de salidas necesarias en las memorias ROM, las memorias RAM requieren líneas de datos de entrada y una línea de control, que determina si la operación es de lectura o de escritura. La línea de control R/W se mantiene en 1 lógico para una operación de lectura, y en 0 lógico para una operación de escritura. Las entradas de dirección, habilitación de chip (chip enable), deshabilitación de salida (output disable) y R/W, deben de seguir un orden apropiado para la operación correcta de la memoria. La secuencia de operaciones, y los requerimientos de tiempo para leer una RAM estática son similares a los de una ROM.

La secuencia básica de operaciones para escribir en una memoria RAM estática es la siguiente:

1. Se aplica una dirección a las entradas de dirección de la RAM.

2. Se selecciona el circuito de la RAM, activando sus entradas de habilitación de chip (chip enable).

3. El dato que va a ser escrito (almacenado) en la memoria, se aplica en las entradas de datos.

4. Se manda un pulso negativo, de 1 a 0 lógico, por la línea $\overline{R/W}$.

5. Las señales de dirección y de selección de chip (chip enable), pueden cambiarse para lectura o escritura de otra localidad de memoria.

La figura 4-4 muestra el diagrama de tiempos para la operación de escritura de una RAM. El pulso de escritura (write pulse), generalmente un 0 lógico, debe ocurrir en la entrada R/W por un periodo mínimo t_{WP}; como indica la realización de la operación de escritura, es una referencia conveniente para especificar otros parámetros asociados con ella, y empieza cuando la línea R/W sufre una transición de 1 a 0 lógico. Sin embargo, en la mayoría de las memorias, los niveles lógicos en las líneas de datos no son importantes, sino hasta que la línea R/W cambia de nuevo de 0 a 1 lógico, porque casi todas ellas no aceptan el dato de entrada sino hasta en esta transición. Esta es una situación similar a la que existe en un flip-flop disparado en la transición positiva.

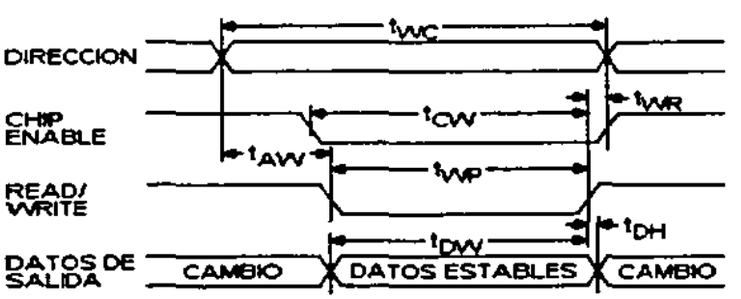


Figura 4-4

Para almacenar información que sea válida, las líneas de datos deben permanecer estables durante un intervalo previo a la transición de 0 a 1 de la línea R/W; a este intervalo se le llama tiempo de preparación del dato (data set up time), t_{DW}. También es necesario mantener el dato estable durante un periodo de tiempo después de la transición de 0 a 1 de la línea R/W; a éste se le conoce como tiempo de sostenimiento del dato (data hold time), t_{DH}.

Siempre que las entradas de selección cambian, transcurre cierto tiempo antes de que las salidas de los decodificadores de dirección se hayan estabilizado en su valor final. Durante estos transitorios, otras localidades de memoria son direccionadas involuntariamente. Si el pulso de escritura se aplica antes de que estos transitorios terminen, el dato que va a ser

escrito en la memoria puede almacenarse en varias localidades de memoria, además de aquella a la que está destinado. Para eliminar esto, las líneas de dirección deben ser estables durante un periodo de tiempo que anteceda y siga a la ocurrencia del pulso de escritura. El tiempo de retraso de escritura (write delay), t_{AW}, indica cuánto tiempo antes de la transición de 1 a 0 del pulso de escritura la dirección debe ser estable; y el tiempo de recuperación de escritura (write recovery time), t_{WR}, indica cuánto tiempo debe de mentenerse estable la dirección después de la transición de 0 a 1 de R/W.

Análogamente a las entradas de dirección, las de habilitación de chip (chip enable) deben ser estables por un periodo, conocido como el tiempo de habilitación de chip (chip enable) a escritura (chip enable to write time), t_{CW}, antes de la transición de 0 a 1 de la línea R/W. Finalmente, el tiempo del ciclo de escritura (write cycle time), t_{WC}, especifica el tiempo mínimo entre operaciones de escritura. En la mayoría de las memorias estáticas, los ciclos de lectura y escritura duran lo mismo.

4.4 Direccionamiento y control de la memoria.

El µprocesador Z-80 tiene capacidad de direccionar hasta 65,536 localidades de memoria de 8 bits (64 KB), por medio de las 16 líneas del bus de direcciones.

Para el control de la memoria, el Z-80 cuenta con tres señales. Las dos primeras son la señal de lectura (RD) y la señal de escritura (WR), explicadas en el capítulo 5; la tercera es una señal que se genera cuando el µprocesador está realizando una operación con la memoria, ya sea de lectura o de escritura. Esta señal recibe el nombre de MREQ (Memory Request), se activa en 0 lógico y tiene salida de tercer estado. MREQ indica que el bus de direcciones contiene una dirección válida para la transferencia de información con la memoria.

Haciendo referencia a los diagramas de tiempos de la sección 3.3.3, se observa que en el ciclo de obtención del código de operación de una instrucción (fig. 3-9), la señal MREQ se activa a la mitad del estado T1, lo mismo que la señal RD, y ambas se desactivan al inicio del estado T3. Durante este lapso, las líneas del bus de direcciones se mantienen estables y muestran el contenido del PC.

En los ciclos de lectura o escritura en memoria (fig. 3-10), la señal MREQ baja a 0 lógico a la mitad del estado T1 y así permanece hasta la mitad del estado T3, cuando sube a 1 lógico. En ese lapso, el bus de direcciones muestra la dirección de la localidad de memoria con la que se realiza la operación. En el caso de lectura, MREQ y RD ocurren simultáneamente. En el caso de escritura, el dato es colocado por el µprocesador cuando se activa MREQ a la mitad de T1, mientras que la señal WR no baja a 0 sino hasta el siguiente estado. La señal WR regresa a 1 lógico junto con MREQ a la mitad de T3.

Por medio de las tres señales de control anteriores se pueden generar dos señales que, junto con las líneas de dirección, determinen la selección, así como la lectura o escritura en una

localidad de memoria. Estas señales son: MEMR = MREQ * RD para lectura y MEMW = MREQ * WR para escritura, ambas son activadas en 0 lógico (fig. 4-5).

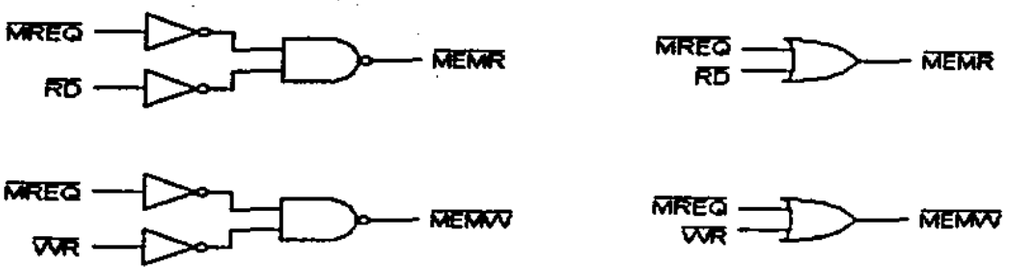


Figura 4-5

4.5 Interconexión de memorias al μP Z80A.

Memorias ROM

Un sistema basado en el Z-80 debe contar, cuando menos, con una ROM en donde se encuentre almacenado el programa a ejecutar.

Como las memorias ROM son memorias de lectura solamente, su estructura es la menos compleja y, por lo tanto, son las más fáciles de interconectar al μprocesador. Generalmente, un circuito de memoria ROM cuenta con dos entradas de control: una es la entrada de selección del circuito (chip select), y la otra es la entrada para activar sus buffers de salida o ponerlos en estado de alta impedancia (output enable); su objeto es evitar que el μprocesador y la memoria traten de controlar el bus de datos simultáneamente. En la figura 4-6 se muestra el diagrama característico de una ROM.

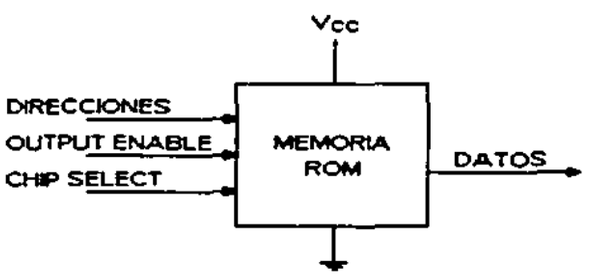


Figura 4-6

La interconexión de una ROM al μprocesador se hace uniendo las líneas de direcciones y de datos de la memoria a los buses correspondientes del μprocesador; en tanto que para la conexión de las entradas de control, hay que tomar en cuenta las siguientes consideraciones: el circuito debe ser seleccionado únicamente si se trata de una operación con memoria, es

decir, cuando la señal MREQ esté activa. Además, es muy conveniente definir con precisión el lugar ocupado por el circuito dentro del espacio total de memoria direccionable por el µprocesador; para ello se recurre a las líneas del bus de direcciones no utilizadas en el acceso a alguna de las localidades de memoria del circuito, las cuales se conectan, junto con MREQ, a la entrada de selección (chip select). Finalmente, para que la ROM controle el bus de datos sólo durante el tiempo en que el µprocesador lee el contenido de alguna de sus localidades, la señal MEMR se conecta a la entrada que habilita la salida (output enable), logrando con ello que el estado de los buffers de salida de la memoria sea de alta impedancia en cualquier otro momento.

De las memorias de lectura solamente, las más populares son las EPROM con longitud de palabra de 8 bits, que incluyen a las familias 25XX y 27XX. Dentro de ellas, uno de los circuitos más usados a nivel comercial es la 2716. La memoria 2716 es una EPROM borrable con luz ultravioleta, organizada en 2,048 localidades de 8 bits (2KB X 8), resultando una capacidad total de 16 Kbits. Está encapsulada en un circuito de 24 terminales, opera con una fuente de 5 V, tiene un amplio acceso de 450 ns, sus entradas y sus salidas son compatibles con TTL y su consumo de potencia es de 525 mW (fig. 4-7).

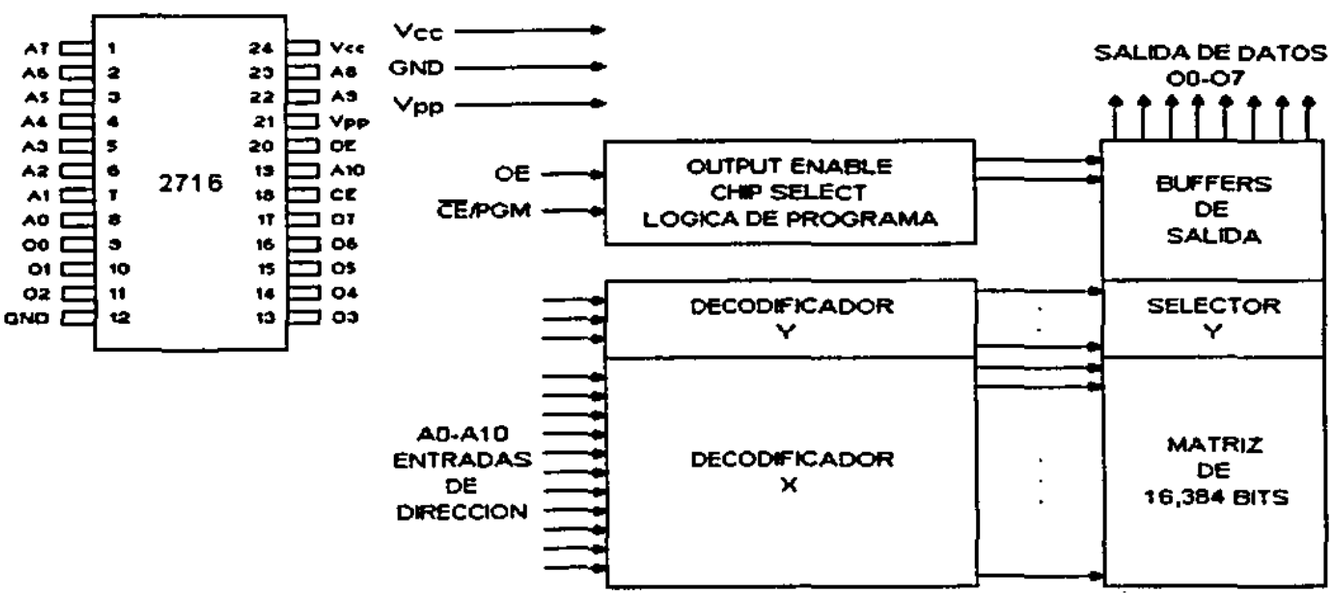


Figura 4-7

La figura 4-8 es un diagrama de la interconexión de una EPROM 2716 con el Z-80. Como la 2716 es de 2K X 8, las once líneas menos significativas del bus de direcciones del µprocesador (A0-A10) se conectan a las once entradas de direcciones correspondientes de la memoria. Las cinco líneas de direcciones restantes del µprocesador (A11-A15) se

utilizan para definir la posición de la memoria y, junto con MREQ, se conectan a la entrada CE de la 2716.

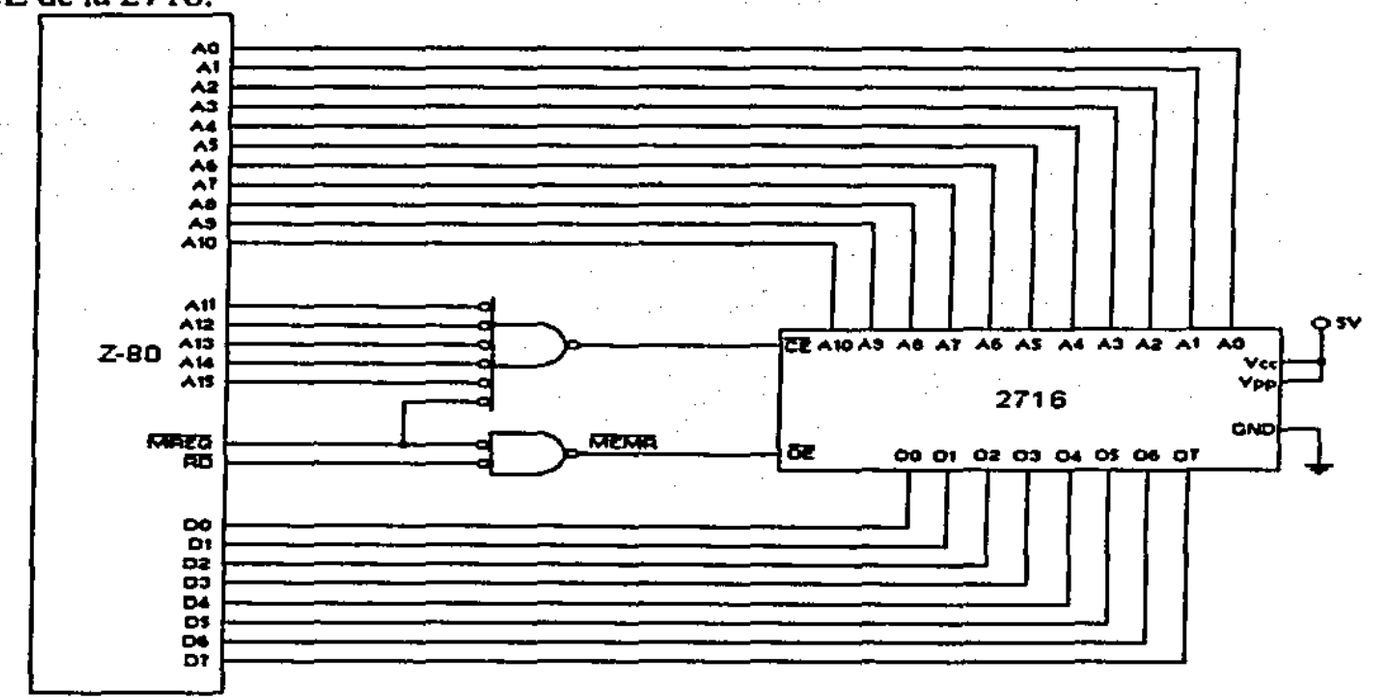


Figura 4-8

El estado de los buffers de las salidas de datos O0-O7 de la EPROM se controla con la línea MEMR que va conectada a la entrada OE.

4.6 Organización de sistemas de memoria.

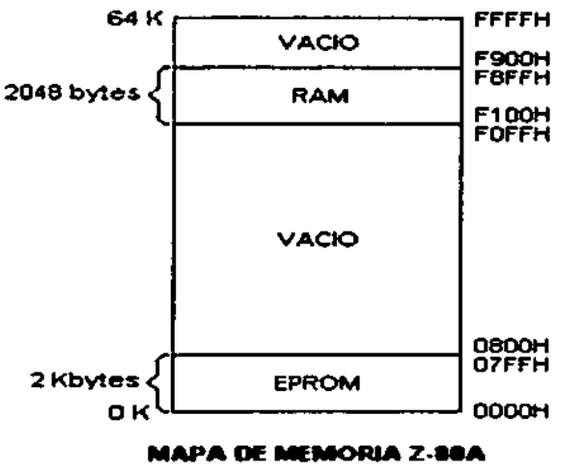
El mapa de memoria.

De acuerdo a lo que se mencionó al principio del capítulo y en la sección 3.5, existe la necesidad de definir la ubicación de un circuito de memoria dentro del espacio de direccionamiento al que tiene acceso el µprocesador.

Se le llama MAPA DE MEMORIA a la representación de los bloques en que se ha dividido el espacio de memoria direccionable por el µprocesador. Cada bloque o partición corresponde al rango de direcciones ocupado por un circuito de memoria, de acuerdo a la asignación que se haya hecho de las líneas del bus de direcciones que no van conectadas a las entradas de direcciones del circuito de memoria.

Por ejemplo, como en general una EPROM almacena el programa, éste normalmente ocupa las direcciones más bajas del espacio de memoria. A la 2716 de la figura 4-8 se le han asignado los primeros 2 KB de memoria (direcciones 0000H a 07FFH), conectando las líneas de dirección A11-A15 en forma tal, que el circuito se active únicamente cuando todas ellas están en 0 lógico.

Si se considera que la EPROM y la RAM están conectadas al mismo µprocesador, se obtiene el mapa de memoria mostrado en la figura 4-9.



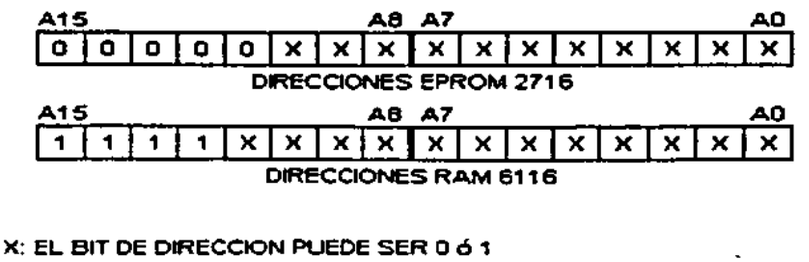


Figura 4-9

Requerimientos de tiempo.

Para que un circuito de memoria pueda ser conectado directamente al µprocesador, se necesita no sólo la compatibilidad en los niveles eléctricos de las señales que los unen, sino también la compatibilidad con respecto a los tiempos y secuencia con que estas señales son generadas, así como a su duración.

Así pues, los parámetros de tiempo característicos de una memoria deben ajustarse a los requerimientos de tiempo que presenta el procesador. Es claro que la relación entre unos y otros se verá afectada, tanto por la forma en que se conecten las líneas de dirección y control del µprocesador a las entradas de selección y habilitación de la memoria, como por el retraso en la propagación de las señales producido por circuitos adicionales que se encuentren entre la memoria y el µprocesador, tales como selectores y buffers.

En la lectura de una memoria, los parámetros más importantes son: el tiempo de acceso (direccionamiento a dato válido) (t_A), que se refiere a las entradas de direcciones; el tiempo de selección del circuito a dato válido (t_{CO}), que se refiere a la entrada de selección chip

select (CS); y el tiempo de habilitación de los buffers de salida a dato válido (t_{OE}), que se refiere a la entrada de habilitación de salida (output enable) (OE).

Su significado en función de los requerimientos de tiempo del µprocesador es el siguiente: t_A debe ser menor que el lapso que hay entre la aparición de la dirección y el momento en que el µprocesador lee el dato; t_{CO} debe ser menor que el tiempo que transcurre entre la aparición de las señales utilizadas para la selección del circuito y el momento en que el µprocesador lee el dato; t_{OE} debe ser menor que el intervalo entre la aparición de las señales usadas en la habilitación de los buffers de salida y el instante en que el µprocesador lee el dato.

Por ejemplo, en el diagrama de la figura 4-8, como la señal CE se obtiene de la unión de MREQ con las direcciones A11-A15, el tiempo de selección de la memoria se calcula a partir de la señal que se activa al último, en este caso MREQ, más el retraso debido a las compuertas lógicas externas que establecen las condiciones de selección. Asimismo, el tiempo de habilitación de los buffers de salida lo determinan la aparición de las señales MREQ y RD y el tiempo de propagación de la compuerta que las une.

Los parámetros más relevantes en la escritura de una memoria son: el ancho del pulso de escritura (t_{WP}), que se refiere a la entrada R/W o WE; el tiempo de acceso -direcciones a pulso de escritura- (t_{AW}), que se refiere a las entradas de direcciones; el tiempo de selección del circuito a escritura (t_{CW}), que se refiere a la entrada de selección (chip select, CS); y el tiempo de preparación del dato para escritura (t_{DW}), que se refiere a las entradas de datos.

Su relación con las señales del µprocesador es la siguiente: la duración de la señal de escritura del µprocesador debe ser mayor que t_{WP}; el intervalo de tiempo comprendido entre la aparición de la dirección y la aparición de la señal de escritura debe ser mayor que t_{AW}; el lapso que hay entre la selección del circuito y la acción de escritura debe ser mayor que t_{CW}; el tiempo que transcurre desde que el µprocesador coloca el dato en las líneas correspondientes, hasta el momento de escritura, debe ser mayor que t_{DW}.

CAPITULO 5

CIRCUITOS DE ENTRADA/SALIDA.

Los grandes computadores usan muy a menudo buses separados en el CPU, para comunicarse con la memoria y la interconexión I/O. Un bus I/O de los grandes computadores consiste en un bus de datos y uno de direcciones similar al bus que se comunica con la memoria. El bus de datos I/O transfiere los datos a los dispositivos externos y viceversa, y el bus de direcciones I/O se usa para seleccionar un dispositivo I/O particular a través de su interconexión. El número de líneas de direcciones en un bus I/O es menor que un bus de memoria, porque hay un menor número de unidades I/O para seleccionar que palabras en un sistema de memoria.

Un µprocesador tiene un límite para el número de terminales que pueden ser acomodados dentro de una pastilla de CI. No hay siuficientes patillas en una pastilla de µprocesadores para suministrar buses separados para comunicarse separadamente con la memoria y el I/O. Invariablemente, todos los µprocesadores usan un sistema de bus común para seleccionar palabras de memoria y unidades de interconexión. Si una pastilla de interconexión tiene un número de registros, cada uno se selecciona por medio de sus propias direcciones. de la misma manera que se selecciona una palabra de memoria. El bus del µprocesador no distingue entre un registro de interconexión y una palabra de memoria. Es responsabilidad del usuario, por medio de instrucciones del programa, especificar la dirección apropiada que seleccione uno u otro. Hay dos maneras de asignar las direcciones para seleccionar los registros de memoria e interconexión. Un método es el llamado I/O con *mapa de memoria* y el otro es el llamado I/O *aislado*.

5.1 Entrada/Salida Aislada.

Con la organización del I/O aislado, el µprocesador especifica en sí mismo cuando la dirección en el bus de direcciones es para una palabra de memoria, o para un registro de interconexión. Esto se hace por medio de una o dos líneas de control adicionales que se fabrican con el µprocesador. Por ejemplo, un µprocesador puede tener una línea de control de salida marcada *M/IO*. Cuando *M/IO* = 1, esto significa que la dirección del bus de direcciones es para una palabra de memoria. Cuando *M/IO* = 0, la dirección es para un registro de interconexión. Esta línea de control debe ser conectada a las entradas de selección de RAM, ROM y de las pastillas de interconexión, de la misma manera que la línea 11 del bus fue conectada èn el ejemplo previo para el caso del I/O con mapa de memoria.

En la organización I/O aislado, el µprocesador debe entregar instrucciones de entrada y salida diferentes, y cada una de ellas debe asociarse con una dirección. Cuando el µprocesador busca y decodifica el código de operación de una instrucción de entrada y salida, éste lee la dirección asociada con la instrucción, y la coloca en el bus de direcciones.

Al mismo tiempo, hace la línea de control *M/IO* igual a 0 para informar a los componentes externos que esta dirección es para una interconexión, y no para la memoria. Así, durante un ciclo de búsqueda o un ciclo de ejecución de referencia de memoria, el µprocesador habilita el control de lectura o escritura, y lleva la línea *M/IO* a 1. Durante la ejecución de una instrucción de entrada o salida, el µprocesador habilita el control de lectura o escritura, y lleva la línea *M/IO* a 0.

El método I/O separado, aísla la memoria y las direcciones I/O, de manera que no se afecte el espacio de memoria por la asignación de la dirección de la interconexión. Debido a este aislamiento, todo el espacio de direcciones disponible por el bus de direcciones no es afectado por el direccionamiento de la interconexión, como en el método de I/O con mapa de memoria.

5.2 Entrada/Salida Mapeada a memoria.

En el método I/O con mapa de memoria, el µprocesador trata el registro de interconexión como parte del sistema de memoria. La dirección originada para los registros de interconexión no puede ser usada para palabras de memoria, reduciendo así el espacio de memoria disponible. En una organización I/O con mapa de memoria, no hay instrucciones de entrada y salida, porque el µprocesador puede manipular los datos I/O que residen en los registros de interconexión con las mismas instrucciones que se usan para manipular los lugares de memoria. Cada interconexión se organiza como un conjunto de registros que responden a los comandos de lectura y escritura en el espacio de dirección normal del µprocesador. Típicamente se reserva un segmento del espacio de direcciones total para los registros de interconexión, pero en general pueden estar localizados en cualquier dirección, siempre y cuando no haya una palabra de memoria que corresponda a esa dirección.

La organización de los I/O con mapa de memoria es conveniente para sistemas que no necesiten espacio disponible de memoria de las líneas del bus de direcciones. Un µprocesador con un bus de datos de 16 bits, que requiere una memoria menor que 32 KB, puede usar otras 32 KB direcciones disponibles del bus para accesar los registros de la interconexión. Una configuración específica para un I/O con mapa de mèmoria puede configurarse, modificando ligeramente las conexiones de dirección mostradas en la fig. 5-1.

La línea de dirección 11 del diagrama no se usa para accesar la memoria. Se dejará ahora que esta línea distinga entre la memoria y la interconexión, de manera que, cuando el bit de la línea sea 1, el bus de direcciones seleccione una memoria de palabra, y cuando el bit sea 0, seleccione un registro de interconexión. Para lograr esta nueva condición, se debe aplicar a una compuerta AND cada línea que va al *CS* en las RAM y ROM de la fig. 5-1 con el bit de la línea 11 de dirección. Las entradas de selección de pastilla de todas las unidades de interconexión, deben estar condicionadas al valor del complemento de la línea 11, además de la dirección asignada.

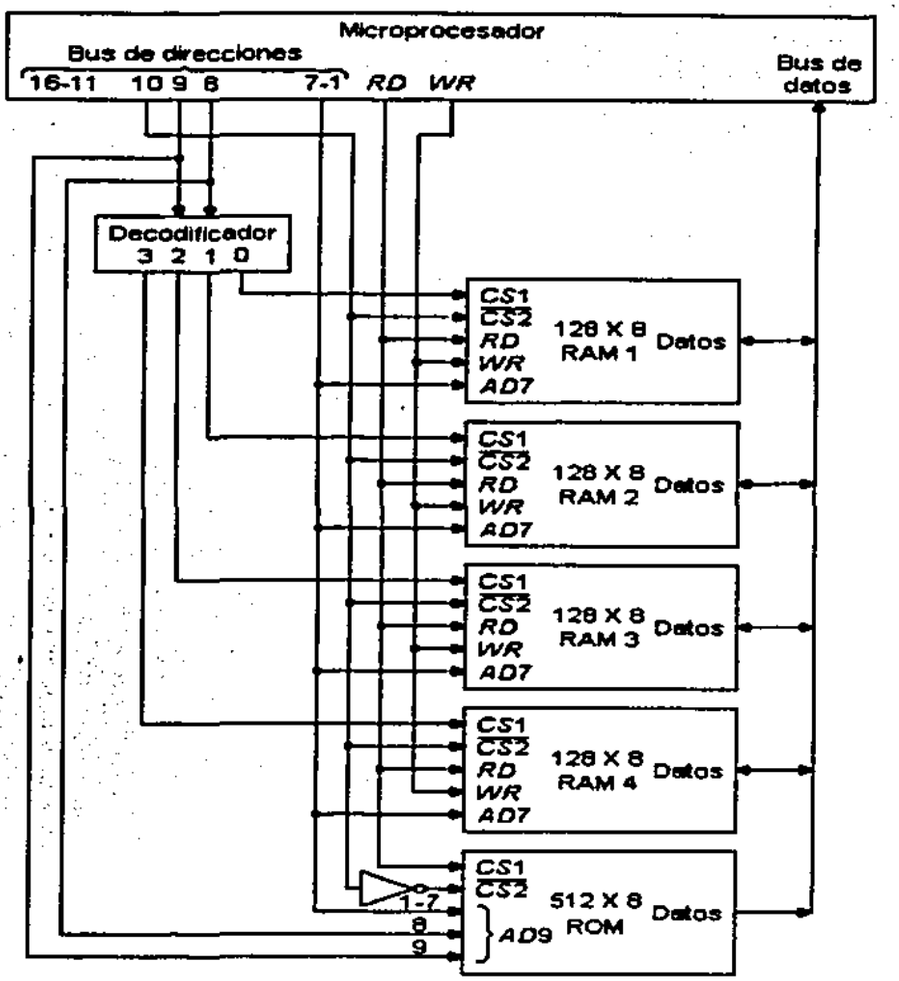
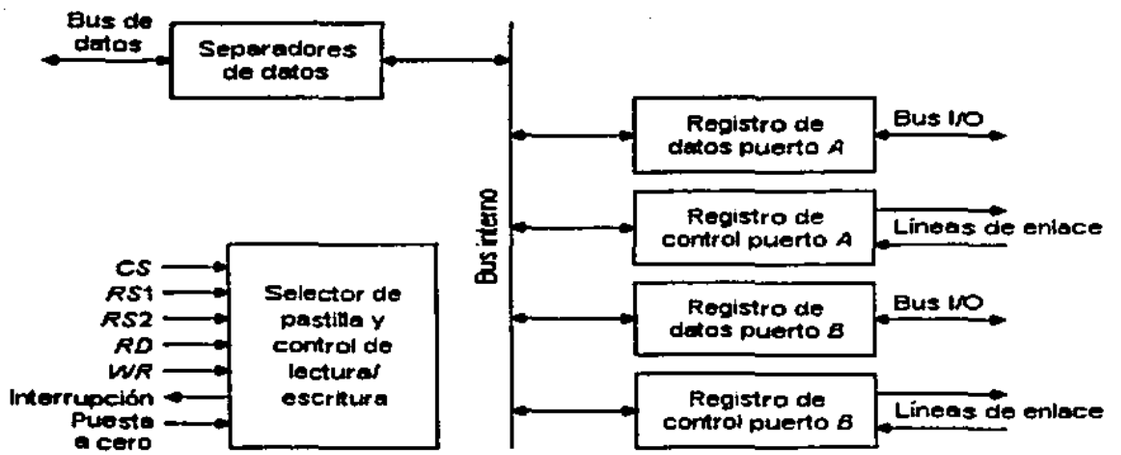


Figura 5-1

5.3 Interconexión periférica en paralelo.

Una interconexión periférica en paralelo es un componente LSI, que presenta un camino para transferir información binaria en paralelo entre el microprocesador y el dispositivo periférico. Una pastilla de interconexión contiene normalmente dos o más puertos I/O que se comunican con uno o más dispositivos externos y una interconexión sencilla para comunicarse con el sistema del bus del microprocesador. El diagrama de bloque de una interconexión periférica típica en paralelo se muestra en la figura 5-2. Este consiste de dos puertos. Cada puerto tiene dos registros, un bus I/O de 8 bits y un par de líneas denominadas de *enlace*. La operación almacenada en el registro de control especifica el

modo de operación del puerto. El puerto del registro de datos se usa para transferir datos al bus de datos y al bus I/O, y viceversa.



CS	*RS1*	*RS2*	Registro seleccionado
0	*X*	*X*	Ninguno — El bus de datos en alta impedancia
1	0	0	Registro de datos puerto *A*
1	0	1	Registro de control puerto *A*
1	1	0	Registro de datos puerto *B*
1	1	1	Registro de control puerto *B*

Figura 5-2

La interconexión se comunica con el microprocesador a través del bus de datos, el selector de pastilla y el control de lectura/escritura. Se debe agregar un circuito externo (usualmente una compuerta AND) para detectar la dirección asignada a la interconexión. Este circuito habilita la entrada de selección de la pastilla cuando se selecciona la interconexión por medio del bus de direcciones. Las dos entradas de selección del registro *RS1* y *RS2* se conectan usualmente a las líneas de menor orden del bus de direcciones. Estas dos entradas seleccionan uno de los cuatro registros en la interconexión, como se explica en la tabla que acompaña el diagrama. El contenido del registro selector se traslada al microprocesador por medio del bus de datos cuando se habilita la entrada *RD*. El microprocesador carga un byte al registro seleccionado por medio del bus de datos cuando se habilita la entrada *WR*. La salida de interrupción se usa para interrumpir al microprocesador, y la entrada de reposición es para poner a cero la interconexión una vez que se suministre potencia.

El microprocesador inicia cada puerto, transfiriendo un byte a su registro de control. Al cargar los bits adecuados a un registro de control en la iniciación del sistema, el programa puede definir el modo de operación del puerto. Las características del puerto dependen de

las unidades comerciales usadas. En la mayoría de los casos, cada puerto puede ser llevado a un modo de entrada o salida. Esto se hace al transferir los bits en el registro de control que especifican la dirección de transferencia en los separadores del bus que accionan el bus I/O bidireccional. En adición, el puerto puede hacerse funcionar en una variedad de modos de operación. Los tres modos de operación encontrados en la mayoría de las pastillas de interconexión son:

1. Transferencia directa sin línea de enlace.

2. Transferencia con enlace.

3. Transferencia con enlace usando interrupción.

Una interconexión se lleva al modo de transferencia directa cuando el dispositivo conectado al bus I/O está siempre listo para transferir información. Las líneas de enlace no se usan en este modo, y algunas pastillas de interconexión tienen un modo de programación para convertir estas líneas en líneas de transferencia de datos. La transferencia directa puede operar en un modo de entrada o salida. En el modo de entrada, una operación de lectura transfiere el contenido del bus I/O al bus de datos del microprocesador. En el modo de salida, una operación de escritura transfiere el contenido del bus de datos al registro de datos del puerto seleccionado. El byte recibido se aplica entonces al bus I/O. Las transferencias de entrada o salida directas son útiles solamente si los datos valederos pueden residir en el bus I/O por un tiempo largo, comparado con el tiempo de ejecución de la instrucción en el microprocesador. Si los datos I/O pueden ser valederos por un corto tiempo, la interconexión debe operar en el modo de enlace.

Las líneas de enlace son usadas para controlar la transferencia entre dos dispositivos que operan asíncronamente entre sí, es decir, cuando no comparten un reloj común. El enlace es un proceso usado comúnmente, y no está restringido para hacer interconexión con pastillas solamente. Dos líneas de enlace, conectadas entre un dispositivo fuente y uno de destino, controlan las transferencias informándose entre sí de la condición de la transferencia por medio del bus común. El dispositivo fuente informa el destino por medio de una de las líneas de enlace cuando se tiene información valedera en el bus. El dispositivo de destino responde inhabilitando la segunda línea del enlace cuando ha sido aceptada la información del bus. La figura 5-2 muestra dos líneas de enlace en cada puerto. Una es una línea de salida, y la otra de entrada. Es costumbre referirse a estas líneas con símbolos, pero los símbolos adoptados son siempre distintos en las diferentes unidades comerciales. Debido a la variedad de símbolos usados para designar esas líneas, se prefiere no adoptar un símbolo sobre otro, sino referirse a las dos líneas como la línea de enlace de salida o entrada. La línea de enlace de entrada pondría a uno un bit en el registro de control dentro de la interconexión. Este bit será llamado indicador, teniendo en cuenta que el registro que retiene el bit indicador (el registro de control en este caso) puede ser leído por el microprocesador para comprobar la condición de la transferencia. El bit indicador se borra

automáticamente en la interconexión después de una operación de lectura o escritura asociada con el correspondiente registro de datos.

La secuencia de enlace detallada para una pastilla comercial de interconexión se especifica con el diagrama de tiempo que acompaña las especificaciones del producto. Debido a la variedad de procedimientos que se encuentran en la práctica, sería mejor explicar el método de enlace en términos generales, sin preferencia por un método específico. La transferencia con enlace depende de si el puerto está en el modo de entrada o salida de información.

En el modo de enlace de salida, el microprocesador escribe un byte en el registro de datos del puerto de interconexión. La interconexión habilita la línea de enlace de salida para informar al dispositivo externo que un byte valedero está disponible en el bus I/O. Cuando el dispositivo externo acepta el byte del bus I/O, éste habilita la línea de enlace de entrada. Ello pone a uno el bit indicador en el registro de control. El microprocesador lee el registro que contiene el bit indicador para determinar si la transferencia fue completa. Si es así, el microprocesador puede escribir un nuevo byte al registro de datos del puerto de interconexión. Al escribir datos en un puerto dado, se borra automáticamente el bit indicador asociado con la transferencia de salida. El proceso puede repetirse para dar salida al siguiente byte.

En el modo de enlace de entrada, el dispositivo externo coloca un byte en el bus I/O y habilita la línea de enlace de la entrada de interconexión. La interconexión transfiere el byte a su registro de datos, y pone a uno un bit indicador en el registro de control. El microprocesador lee el registro que contiene el bit indicador para determinar si se requiere una transferencia de entrada. Si se pone a uno el bit indicador, el microprocesador lee el byte del registro de datos del puerto, y borra el bit indicador. La interconexión informa entonces al dispositivo conectado al bus I/O a través de la línea de enlace de salida, que el nuevo byte puede ser aceptado. Una vez que el dispositivo de salida ha sido informado de que la interconexión está lista, puede iniciar la transferencia del siguiente byte, habilitando de nuevo el enlace de entrada.

En el método de enlace anteriormente descrito, el microprocesador debe leer periódicamente el registro de control para comprobar la condición del bit indicador. Si hay un número de puertos conectados al microprocesador, sería necesario hacerles un muestreo en sucesión para determinar aquellos que requieren una transferencia. Esta es una operación que consume tiempo y que puede ser evitada si se inicia la intercionexión para que opere en el modo de interrupción. La salida de interrupción mostrada en la figura 5-2 se usa entonces para solicitar una interrupción del microprocesador. La mayoría de las unidades comerciales presentan una línea de interrupción separada para cada puerto en la interconexión. Cada vez que se pone a uno un indicador en el puerto, la petición de interconexión que pertenece al puerto se habilita automáticamente para informar al microprogramador que se va a iniciar la transferencia. El microprocesador responde a la

señal de interrupción del puerto que solicitó la acción, y transfiere el byte de datos al registro de datos del puerto de interconexión, y viceversa.

5.4 El puerto PPI8255A.

El Intel 8255A es un dispositivo de entrada/salida programable de propósito general, diseñado para empleo con µprocesadores. Posee 24 terminales de entrada/salida, las cuales pueden ser programadas individualmente en 2 grupos de 12, y utilizadas en 3 modos principales de operación. En el primer modo (MODO 0), cada grupo de 12 terminales de entrada/salida puede ser programado en juegos de 4 para ser entrada o salida. En MODO 1, el segundo modo, cada grupo puede ser programado para tener 8 líneas de entrada o salida. De las 4 terminales que restan, tres son utilizadas para verificar e interrumpir señales de control. El tercer modo de operación (MODO 2) es un modo de bus bidireccional, el cual utiliza 8 líneas para un bus bidireccional, y 5 líneas, tomando prestada una del otro grupo, para la verificación.

Descripción funcional del 8255A.

Generalidades.

El 8255A es un dispositivo de interfaz periférico programable (IPP), diseñado para uso en sistemas de µcomputadoras. Su función es interconectar un componente de entrada/salida de propósito general con el bus del sistema de la µcomputadora. La configuración funcional del 8255A es programada por el software del sistema, por lo que normalmente, ninguna lógica externa es necesaria para interconectar dispositivos periféricos o estructuras.

Buffer del bus de datos.

Este buffer de 8 bits bidireccional de 3 estados es utilizado para interconectar el 8255A con el bus de datos del sistema. Los datos son transmitidos o recibidos por el buffer, en la ejecución de instrucciones de entrada o salida hecha por la CPU. Las palabras de control y la información de estado, son también transferidas a través del buffer del bus de datos.

Lectura/escritura y lógica de control.

La función de este bloque es manejar todas las transferencias internas y externas, tanto de datos, como de control o de las palabras de estado. Acepta entradas de la dirección de los buses de dirección y control de la CPU, y a su vez, emite comandos a ambos grupos de control.

Chip Select (CS).

Selección de circuito. Un "bajo" en esta terminal de entrada, habilita la comunicación entre el 8255A y la CPU.

Read Enabled (RD).

Lectura. Un "bajo" en esta terminal de entrada, habilita al 8255A para mandar los datos o información de estados a la CPU en el bus de datos. En esencia, permite a la CPU "leer" del 8255A.

Write Enabled (WR).

Escritura. Un "bajo" en esta terminal de entrada, habilita a la CPU para escribir datos o palabras de control en el 8255A.

Direcciones (A_0 y A_1).

Selección de puerto 0 y selección de puerto 1. Estas señales de entrada, en conjunción con las entradas de lectura y escritura, controlan la selección de uno de los tres puertos o los registros de palabra de control. Ellos están normalmente conectados a los bits menos significativos del bus de direcciones (A_0 y A_1).

Restablecimiento (RESET).

Restablecimiento. Un "alto" en esta entrada, limpia al registro de control, y todos los puertos (A, B, C) son puestos al modo de entrada.

Controles del grupo A y grupo B.

La configuración funcional de cada puerto es programada por el software de los sistemas. En esencia, la CPU manda una palabra de control al 8255A. La palabra de control contiene información tal como "modo", "puesta de bit", "restablecimiento de bit", etcétera, que inicializa la configuración funcional del 8255A.

Cada bloque de control (grupo A y grupo B) acepta "comandos" de la lógica de control de lectura/escritura, recibe "palabras de control" del bus de datos interno, y emite los comandos apropiados a sus puertos asociados.

Grupo de control A -- Puerto A y parte alta del puerto C (C7 - C4).

Grupo de control B -- Puerto B y parte baja del puerto C (C3 - C0).

En el registro de palabra de control, únicamente se puede escribir. Ninguna operación de lectura del registro de palabra de control está permitida.

Puertos A, B y C.

El 8255A contiene tres puertos de 8 bits (A, B y C). Todos pueden ser configurados en una gran variedad de características funcionales por el software del sistema, pero cada uno posee sus propias características especiales para acrecentar más la potencia y flexibilidad del 8255A.

Puerto A. Un latch/buffer de salida de datos de 8 bits y un latch de entrada de datos de 8 bits.

Puerto B. Un latch/buffer de entrada/salida de datos de 8 bits y un buffer de entrada de datos de 8 bits.

Puerto C. Un latch/buffer de salida de datos de 8 bits y un buffer de entrada de datos e 8 bits (no hay latch para la entrada). Este puerto puede ser dividido en dos puertos de 4 bits mediante el control de modo. Cada puerto de 4 bits contiene un latch de 4 bits, y puede ser utilizado para las salidas de señal de control y entradas de señal de estado, en conjunción con los puertos A y B.

Descripción operacional del 8255A.

Selección de modo.

Existen tres modos básicos de operación que pueden ser seleccionados por el software del sistema:

Modo 0 -- Entrada/salida básica.

Modo 1 -- Entrada/salida de strobe.

Modo 2 -- Bus bidireccional.

Cuando la entrada de restablecimiento se va a "alto", todos los puertos serán puestos en el modo de entrada (es decir, todas las 24 líneas estarán en estado de alta impedancia). Después de que el restablecimiento se ha quitado, el 8255A puede permanecer en el modo de entrada sin inicialización adicional requerida. Durante la ejecución del programa del sistema, cualquiera de los otros modos pueden ser seleccionados, utilizando una sola instrucción de salida. Esto permite a un único 8255A, suministrar lo necesario a una variedad de dispositivos periféricos con una simple rutina de mantenimiento de software.

Los modos para el puerto A y el puerto B pueden ser definidos separadamente, mientras que el puerto C es dividido en dos partes, como se necesite por las definiciones del puerto A y puerto B. Todos los registros de salida, incluyendo los flip-flops de estado, serán restablecidos en cualquier tiempo que el modo sea cambiado. Los modos pueden estar combinados, así que su definición funcional puede estar "hecho a la medida" para casi cualquier estructura de entrada/salida. Por ejemplo, el grupo B puede ser programado en modo 0 para monitorear simples cierres de interruptores o para desplegar resultados computacionales, y el grupo A puede ser programado en modo 1 para monitorear un teclado o un lector de cinta.

CAPITULO 6

LA INTERFAZ DNT.

La interfaz DNT surge de la necesidad que tiene la Procuraduría General de la República de controlar sus líneas telefónicas en cuanto a sus llamadas de salida, ya que es normal que el personal las utilice con fines extralaborales.

La interfaz por sí sola representaba información que sólo entendía gente capacitada en la electrónica digital, sin embargo se requería de una presentación que facilitara el uso y la interpretación de la información.

Es a partir de esto, que decidimos diseñar un circuito digital que convirtiera esa información (BINARIA), a una información organizada en una secuencia de números decimales, lo cual lleva a una fácil comprensión del usuario.

Además. el desarrollar este circuito digital nos permite procesar la información para almacenarla y hacer una revisión posterior de ella.

6.1 Uso de la interfaz DNT.

Durante el desarrollo de este circuito se amplió el campo de uso (o empleo) de la interfaz DNT, en otras áreas dentro de la misma PGR y fuera de ella en lugares como teléfonos de México y Bolsas de Valores.

En la PGR, se le encontraba utilidad en la investigación y rastreo de información relacionada a personas involucradas en casos o asuntos de índole policiaco.

En Teléfonos de México podría colocarse a los usuarios para realizar un control más riguroso en cuanto a llamadas de larga distancia, debido a que en muchas ocasiones los usuarios asisten a las sucursales de esta empresa reclamando llamadas sin justificación aparente. Así, viendo su aparato DNT, se saldría de toda duda acerca de estas llamadas.

En las bolsas de valores resultaría de suma importancia debido a que en éstas, gran parte de compra y venta de acciones se realiza por teléfono y es muy importante saber si se realizó la llamada realmente, ya que dependiendo si se realizó o no, pueden existir pérdidas o ganancias considerables.

6.2. Características de la interfaz DNT.

La interfaz DNT es básicamente un convertidor analógico/digital. En la fig. 6-1 se muestran las entradas y salidas de la interfaz.

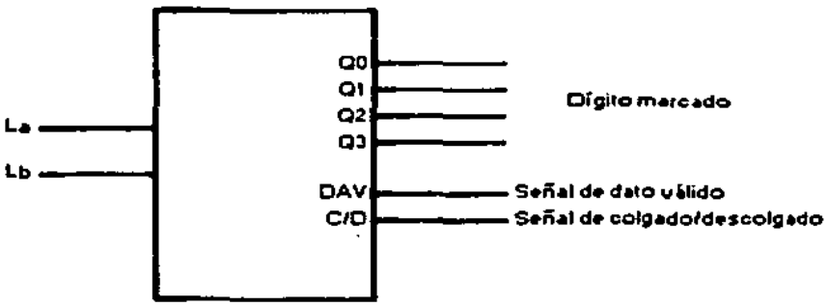


Figura 6-1

Entrada (Analógica).

La entrada es una línea telefónica (2 hilos) proveniente de una central analógica ya sea con -48 volts ó -25 volts. En este caso, en la entrada no importa la polaridad.

Salida (Digital).

Los 4 bits de Q0-Q3 nos muestran en BCD el número marcado por el usuario (0-9), donde del (1-9) hay correspondencia uno a uno, pero el cero aparece como 10 decimal, (1010) binario.

El bit DAV nos indica cuando el número que aparece en Q0-Q3 debe ser considerado como el número marcado.

Si DAV = 1, el número es válido
Si DAV = 0, el número no es válido.

El bit C/D, nos indica cuando el aparato se encuentra colgado o descolgado, cuando

C/D = 0, se encuentra descolgado, y cuando
C/D = 1, se encuentra colgado.

Obviamente cuando C/D = 1 el aparato telefónico no está siendo utilizado.

6.3. Análisis de control de la interfaz DNT.

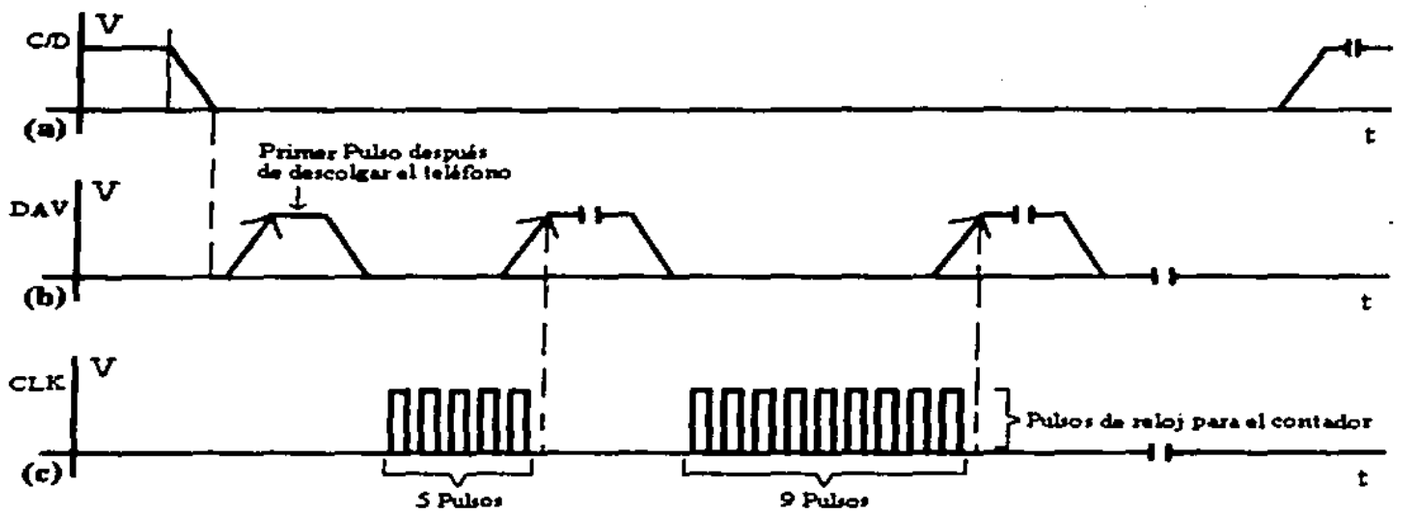
Para aprovechar los dígitos que nos entrega la DNT es necesario realizar un análisis de tiempos, ya que internamente la interfaz DNT trabaja con un contador que nos muestra mediante Q0-Q3 la secuencia desde cero hasta el número que se marcó, es decir, si nosotros marcamos el cinco, aparecerá en Q0-Q3 la secuencia de cero a cinco en binario y en donde

el único número que nos interesa es el último, el cuál es señalado por el bit de dato válido, es decir, al llegar Q0-Q3 a cinco, se mantendrá por un momento y DAV se va a uno haciendo válido ese dígito.

Por otro lado, es importante ver que cuando C/D esta en alto (colgado), los bits Q0-Q3 y DAV no se toman en cuenta ya que no se está realizando ninguna llamada.

El hecho de que se realice una llamada, C/D va a bajo y nos da la pauta para analizar DAV.

En la fig. 6-2 se presentan los diagramas de tiempo.



(a) C/D, Al descolgar, se genera un primer pulso en DAV, el cual debe ser ignorado.

(b) DAV, En estas transiciones (positivas) la IDNT nos indica que el número en Q0-Q3 es el válido.

(c) DATO, Esta señal es serial, pero al pasar por el contador se refleja en Q0-Q3 como numero cuando el dato es válido.

Figura 6-2

Un número telefónico empieza cuando C/D va a una transición negativa y termina cuando va a una transición positiva, durante el tiempo en que C/D está en bajo se marcan los dígitos de la llamada correspondiente de acuerdo al diagrama de DAV y dígito serial.

Al marcar un dígito en este tiempo, tal como se ve en el diagrama de tiempos, se genera un conteo que se refleja en Q0-Q3 y que nos indica por medio de DAV (en su transición positiva), que el dígito en Q0-Q3 es el que se marcó.

Analizando el diagrama de dígito-señal vemos que el número de pulsos es igual al número del dígito marcado y que en el último pulso, el dígito válido va de cero a uno.

Resultó que en el caso de números marcados dentro de la población en que se encuentra instalada la IDNT no hay problema en cuanto al conteo de estos pulsos; pero al realizar una llamada de larga distancia o hacia un teléfono celular, se presenta un efecto singular. Sucede que al marcar una larga distancia, después del código se genera un pulso automático que indica a la central la característica de la llamada, de tal forma que si se marcó como último dígito de lada un uno, en la interfaz aparecen dos pulsos de los cuales el último no debe tomarse en cuenta.

Lo mismo pasa al marcar hacia un teléfono celular después del 90 y antes del 5, cuando se marca el cero aparecen 10 pulsos, donde el último no debe tomarse en cuenta.

CAPITULO 7

DISEÑO DEL CIRCUITO DIGITAL (HARDWARE).

7.1 Requerimientos de control para la interfaz DNT.

Se vio en el capítulo 6 que la IDNT nos da información relacionada con los números marcados. Esta información debe ser procesada para almacenarla y hacerla visible al sentido humano. De tal forma se estructuró un sistema de control capaz de realizarlo. La base: un control digital, dotado de un teclado con el cual el usuario tuviera acceso al sistema, y un display de cristal líquido por el cual viera la información. Una memoria conteniendo las ordenes del controlador y las llamadas almacenadas. La fig. 7-1 presenta la conjunción de estos elementos.

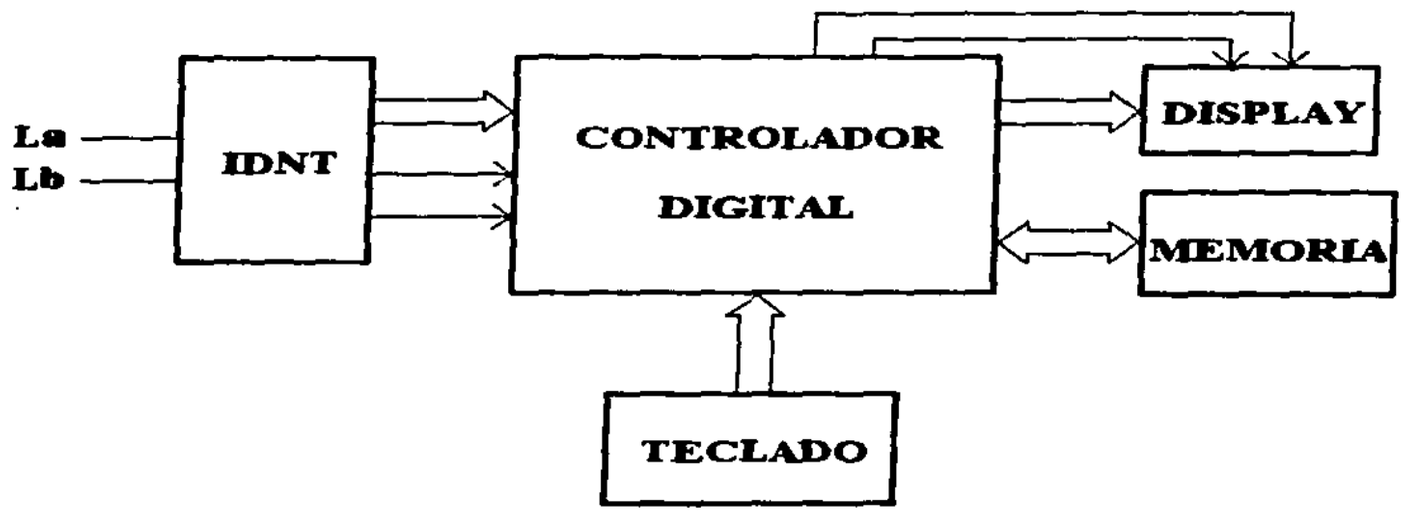


Figura 7-1

7.2 Diseño del mapa de memoria y Entrada/Salida.

El mapa de memoria sirve tanto para la interconexión del µprocesador con la memoria y periféricos, como para el direccionamiento en los programas de control, ya que queda determinada la posición de cada elemento física y lógicamente. Los elementos a considerar en el diseño son:

- La aplicación y flexibilidad para futuras expansiones, para así definir el tamaño de la memoria programa (sólo lectura).

- Las tablas y punteros, ayudan en la definición del tamaño de la memoria

de programa (sólo lectura, en caso de tablas) y de la memoria de datos (lectura y escritura, en tablas y punteros).

- Los periféricos.

7.2.1 Memoria de programa (sólo lectura).

La memoria principal está compuesta de 8 KB EPROM, y se designó en los 8 primeros KB del mapa debido a que al inicializarse el µprocesador, el contador del programa es cargado con la dirección de memoria 0000, de la cual tomará el primer código de operación.

7.2.2 Memoria de datos (lectura y escritura).

La memoria de escritura y lectura utiliza los siguientes 8 KB, y es memoria EEPROM, en esta se direcciona el puntero de la pila y el puntero de almacenamiento de llamadas. El puntero de la pila se carga con 3FFF, que es el final de la EEPROM; el puntero de almacenamiento (Dirección 2000) se carga con 2010, que indica el principio de la primera llamada. La memoria comprendida de la dirección 2010 a la 267F se designa para guardar la base de almacenamiento para los números telefónicos de las llamadas detectadas.

7.2.3 Periféricos (lectura y escritura).

Para los periféricos se seleccionó el método de entradas/salidas mapeadas, por lo cual el puerto de I/O ocupa los siguientes 4Bytes (Direcciones 4000 a 4003).

7.2.4 Mapa de memoria seleccionado.

La fig. 7-2 muestra el mapa de memoria. Nótese que la última dirección utilizada es la 4003H, correspondiente al puerto de configuración PR del dispositivo de entrada/salida.

7.3 Diseño del sistema de escritura para EEPROM.

Como pudo observarse en la sección anterior, la memoria de lectura y escritura está formada por una memoria EEPROM y no una memoria RAM. La razón es que la primera lográ conservar los datos sin necesidad de energía. Más sin embargo presenta una desventaja respecto a la segunda: la velocidad de escritura. La RAM en este aspecto es más rapida y en condiciones normales de operación puede fácilmente soportar tiempos de escritura generados por el µprocesador con una base de reloj de 4MHZ. La EEPROM no cuenta con esa capacidad, por lo cual se diseñó un circuito de control de escritura para ajustar al µprocesador al mismo ritmo.

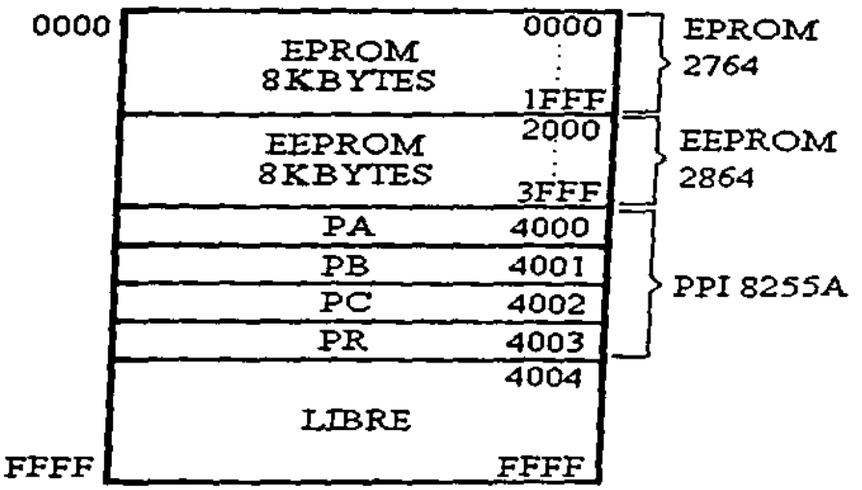


Figura 7-2

Como se vio en el capítulo 4, la entrada WAIT nos sirve para realizar un tiempo de escritura más largo. Se implemento mediante el apoyo de un C.I. TTL, modelo 74121, que es un multivibrador monoestable. A continuación se presentan sus características y forma de uso.

C.I. TTL 74121.

Es un circuito multivibrador monoestable con entrada de disparo schmitt con las siguientes características generales:

- Programación del ancho del pulso de salida.
  Con la resistencia interna Rint, el ancho es de 35 *ns*.
  Con resistencia y capacitor externos Rext/Cext el ancho se puede programar en el rango de 40 *ns* hasta 28 segundos.

- Tiene 2 entradas schmitt sensibles a transiciones negativas, y una entrada schmitt sensible a transiciones positivas.

- Compensación interna, que lo hace independiente de la temperatura.

- Identificación de sus pins.

  1 Q, pulso de salida complementario.
  2 NC, no conexión.
  3 A1, primer entrada schmitt sensible a transición negativa.
  4 A2, segunda entrada schmitt sensible a transición negativa.

5 B, entrada schmitt sensible a transición positiva.
6 Q, pulso de salida.
7 GND, conexión a tierra.
8 NC, no conexión.
9 Rint, conexión a resistencia interna.
10 Cext, conexión para capacitor externo.
11 Rext/Cext, conexión para resistencia y capacitor externos.
12 NC, no conexión.
13 NC, no conexión.
14 Vcc, conexión al voltaje de alimentación.

La figura 7-3 contiene la disposición física de los pins y el diagrama lógico del C.I. 74121.

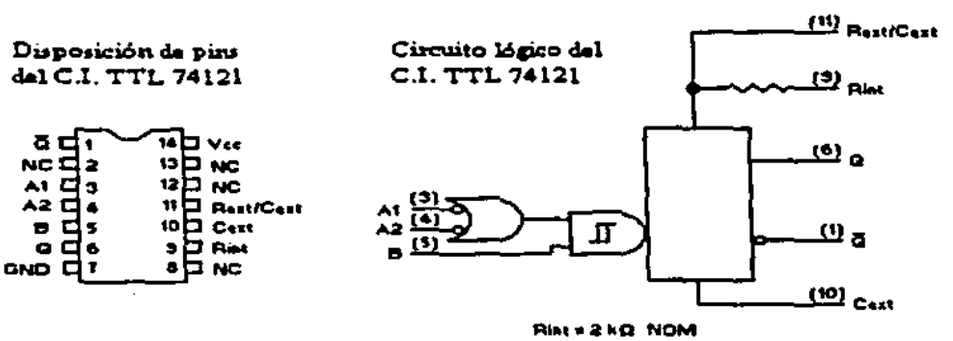


Figura 7-3

El ancho del pulso de salida bajo circunstancias definidas en el manual de usuario se define como:

$$t_W(out) = Cext * R_T * Ln2 = 0.7 * Cext * R_T$$

Las conexiones al μprocesador quedaron de acuerdo a la fig. 7-4.

7.4 Decodificación de memoria y Entrada/Salida.

Para decodificar la memoria y entrada salida hubo que tomar en cuenta el bus direcciones de16 bits y el bus de datos.

Por el bus de datos el μprocesador recibe información proveniente de la memoria y el dispositivo de e/s, al igual que les envía información. Dado que sólo se tiene un sólo bus, este se conecta en paralelo con la memoria y el dispositivo de e/s.

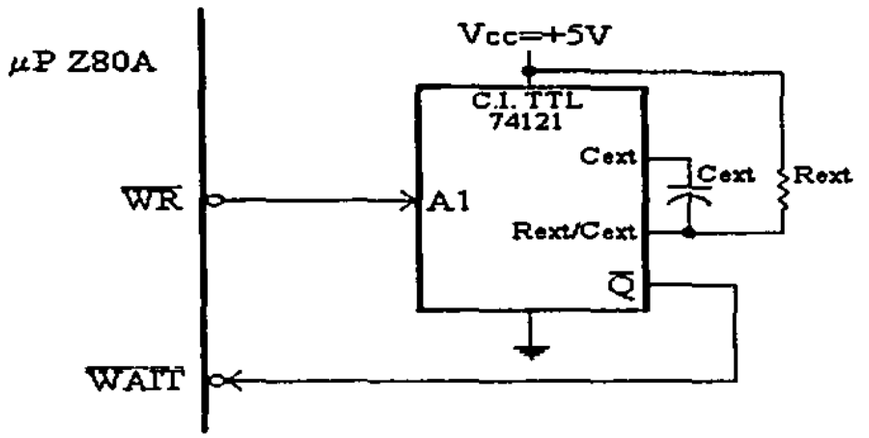


Figura 7-4

La decisión importante a tomar en este punto, es indicar quién debe enviar o leer información en el bus de datos. Para esto, se usa el bus de direcciones del μprocesador y un C.I.TTL decodificador 3 * 8 modelo 74138, cuyas características se muestran a continuación.

C.I. TTL 74138.

- Es programable para trabajar como decodificador o demultiplexor.
- Tiene 2 entradas de habilitación de chip, con lo cual es compatible con los μprocesadores.
- Al trabajar como decodificador sus 8 salidas son de lógica negativa, compatible con memorias y dispositivos de e/s.
- La fig. 7-5 presenta su disposición de pins y circuito lógico.

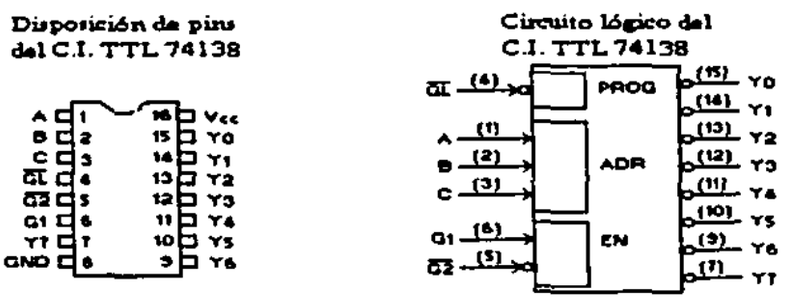


Figura 7-5

Para la solución de la decodificación, es importante conocer las características de las memorias y del PPI8255. Estas se muestran a continuación, ya que sirven para determinar las conexiones necesarias.

C.I.'s de memorias modelos 2764 y 2864.

En la memoria 2764 que a continuación se presenta (fig. 7-6), vemos como pins de control a: el bus de datos, de direcciones, de habilitación de lectura y chip.

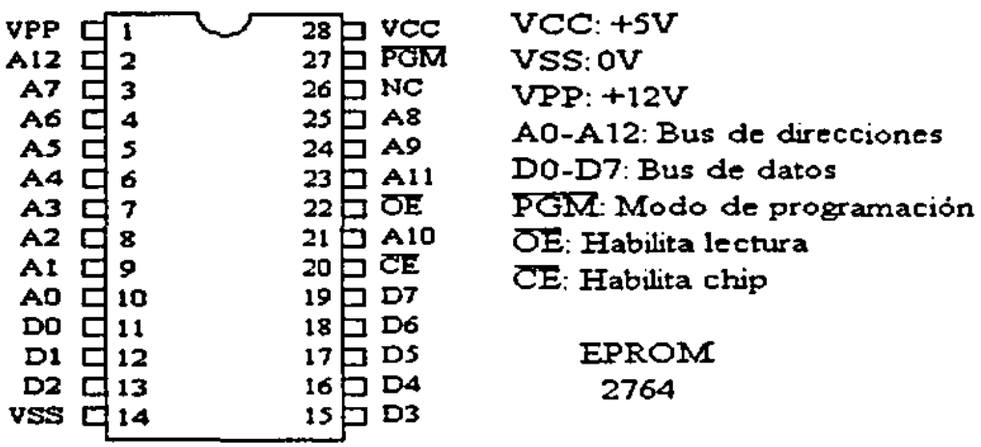


Figura 7-6

En la memoria 2864 que a continuación se presenta (fig. 7-7), vemos casi los mismos pins de control, sólo que ésta tiene también un pin de habilitación de escritura.

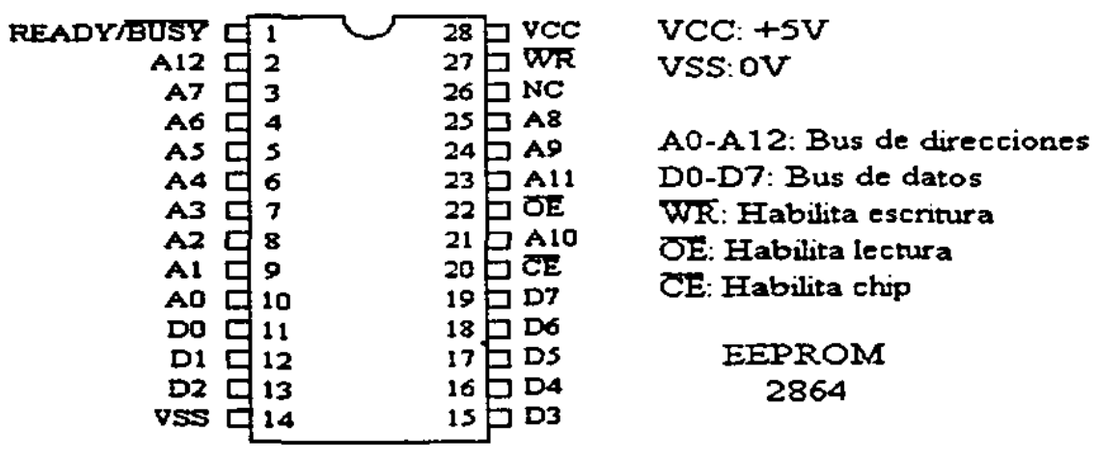


Figura 7-7

C.I. de dispositivo de entrada/salida modelo PPI8255A.

La fig. 7-8 presenta los pins de control del C.I. PPI8255A.

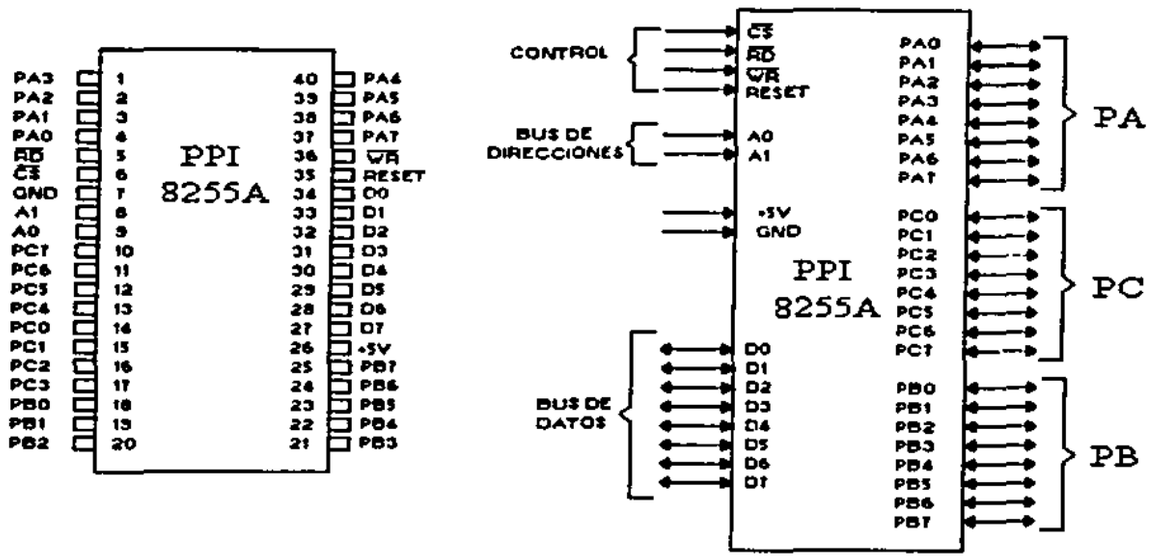


Figura 7-8

De las 3 figuras anteriores determinamos que los buses en común a los 3 circuitos son los siguientes:

- Bus de direcciones.
- Bus de datos.
- Habilitación de lectura.
- Habilitación de chip.

Y que además el C.I. 2864 y el PPI8255A tienen en común:

- Habilitación de escritura .

También podemos apreciar de la figura del µprocesador que con sus 13 primeros bits A0-A12 podemos direccionar $2^{13} = 8192 = 8$ KB. Teniendo en cuenta que nuestra última dirección de mapeo es la 4003H, observamos que el bit A15 no es necesario, y que además cuando A13 y A14 están desactivados quiere decir que el µprocesador desea acceder al primer bloque de 8 KB, cuando A13 está activado y A14 está desactivado el µprocesador desea acceder al segundo bloque de 8 KB, y finalmente cuando A13 está desactivado y A14 activado el µprocesador desea acceder al tercer bloque de 8 KB. Lo anterior se presenta gráficamente en la fig. 7-9.

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	DIR	DIR
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	2	2
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	3	3
0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	8191	1FFF
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	8192	2000
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	8193	2001
0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	8194	2002
0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	8195	2003
0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	16383	3FFF
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16384	4000
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	16385	4001
0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	16386	4002
0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	16387	4003
0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	24575	5FFF

Figura 7-9

De lo anterior se tiene que:

A14	A13	Habilita chip
0	0	2764
0	1	2864
1	0	8255A

La tabla anterior nos muestra que con el funcionamiento del C.I. 74138, podemos controlar la habilitación del chip deseado. La conexión se tiene en la fig. 7-10.

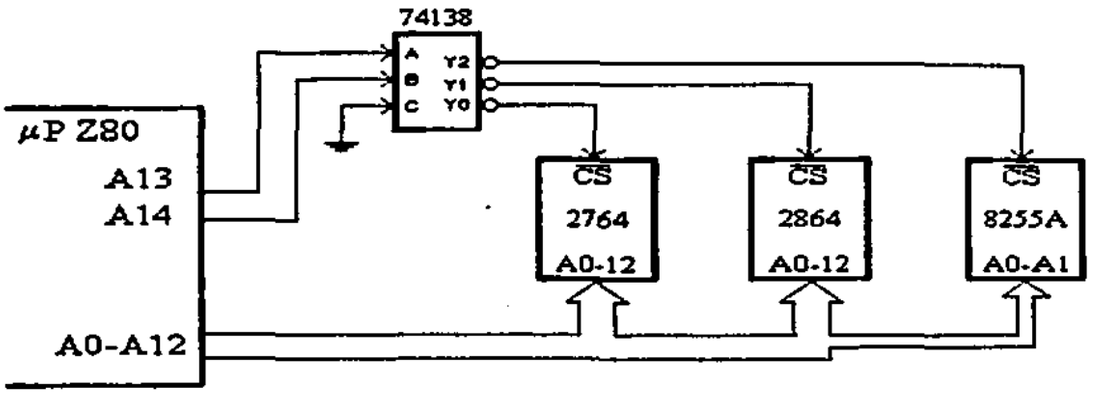


Figura 7-10

Una vez decidido lo anterior, se toma la configuración del PPI8255A (fig. 7-11), para la disposición de los periféricos (interfaz DNT, display y teclado).

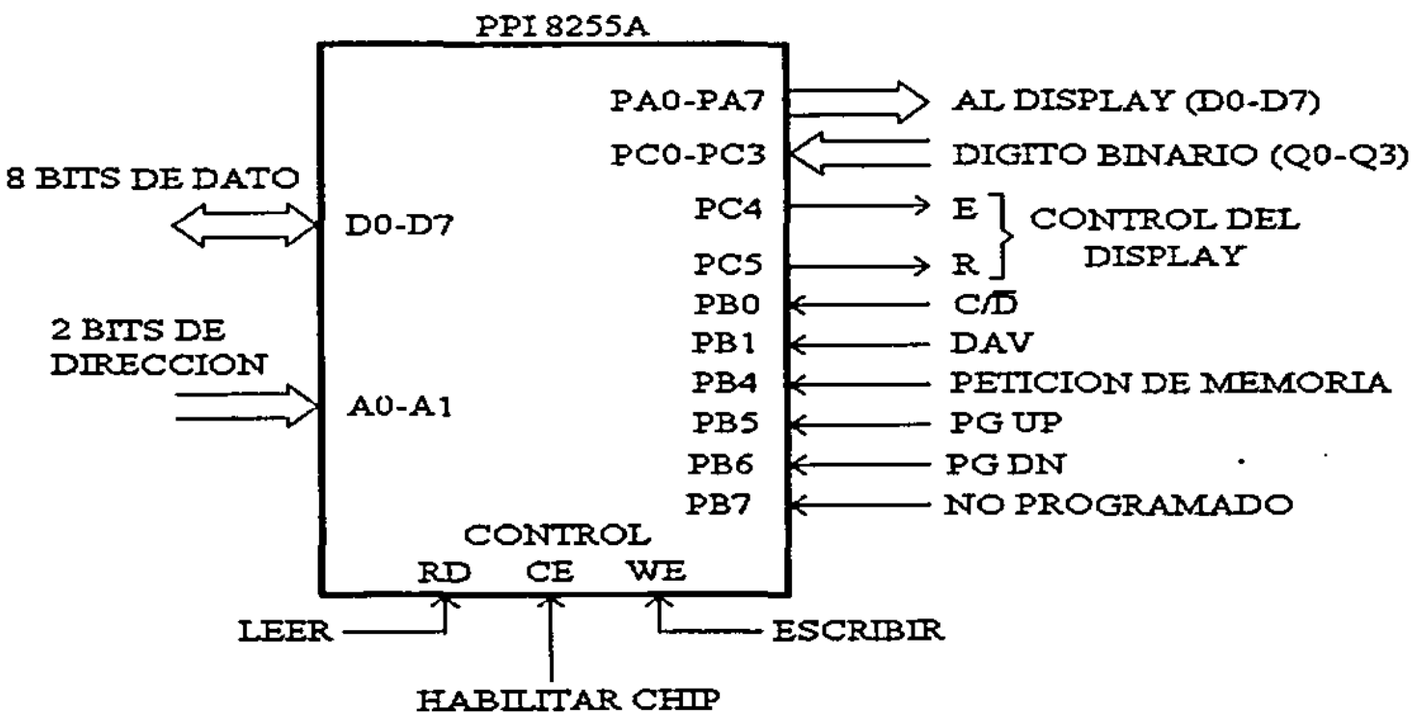


Figura 7-11

7.5 Mapeo del teclado.

Según la figura, el teclado está formado por 3 botones que se encuentran mapeados en la localidad de memoria 4001, y que viene siendo el registro de datos del puerto B del PPI8255A.

La configuración del teclado es de señal con lógica positiva, como se muestra en la figura 7-12.

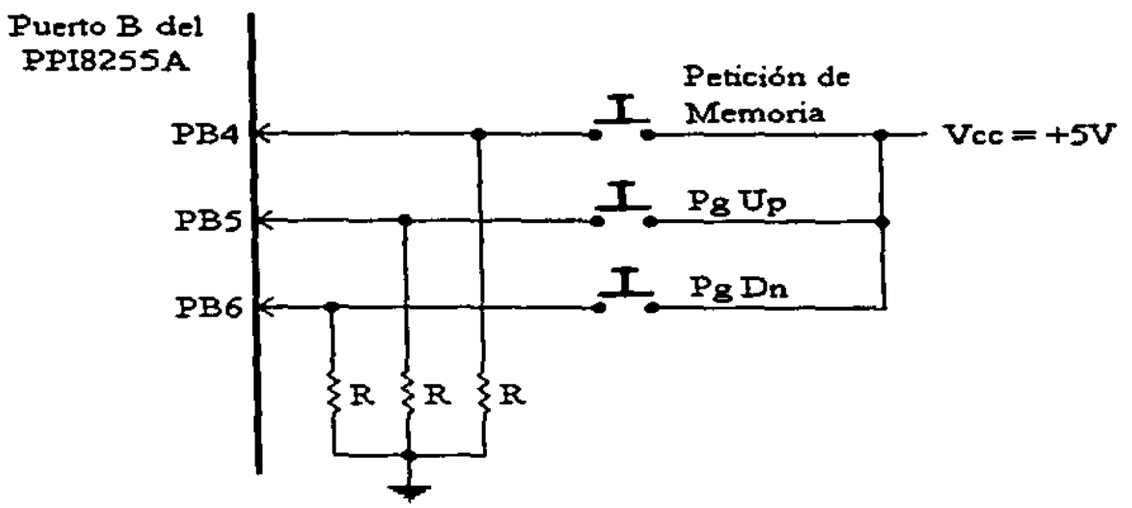


Figura 7-12

La figura anterior muestra que cuando se oprima algún botón, la señal de entrada será de +5V (1 lógico) y cuando éste no se oprima, la señal será de 0V (0 lógico).

7.6 Mapeo de la interfaz DNT.

Según la fig. 7-11, la interfaz esta mapeada en la localidad de memoria 4001 (Puerto B) en los bits 0 y 1, para la señal de colgado/descolgado y dato valido respectivamente. Y en la localidad de memoria 4002 (Puerto C) bits 0-3 para el dato en BCD del digito marcado.

7.7 Mapeo del LCD.

Según la fig. 7-11, el display esta mapeado en la localidad de memoria 4000 (Puerto A), para el bus de datos del display. Y en la localidad de memoria 4002 (Puerto C) bits 4 y 5 para la señal de habilitación de display y habilitación de dato/comando respectivamente.

7.8 Circuito final.

La fig. 7-13 presenta todas las conexiones del circuito final.

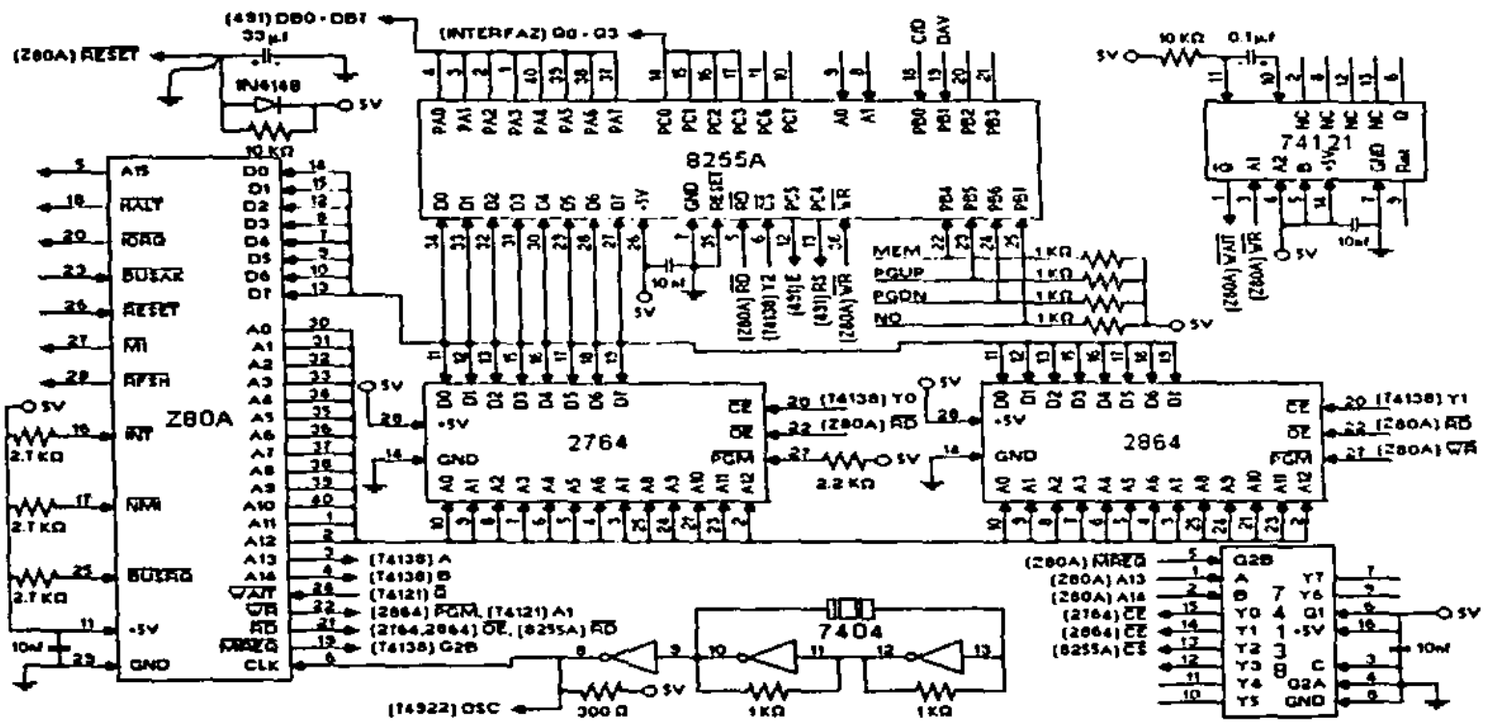


Figura 7-13

CAPITULO 8

DISEÑO DE LOS PROGRAMAS DE CONTROL.

De acuerdo al capítulo 7, la configuración basada en el μprocesador Z80A es la siguiente:

1 Microprocesador Z80A (4 MHz).

1 Memoria 2764, la cual contiene el programa que controlaría al sistema y a la interfaz detectora.

1 Memoria 2864, que almacena las llamadas telefónicas detectadas, sirviendo también de pila (o stack) al microprocesador.

1 Circuito de puerto entradas/salidas 8255, para la comunicación con la IDNT el display y el teclado.

1 Decodificador de memoria (basado en C.I. TTL 74138).

1 Circuito de reloj (basado en cristal y C.I. TTL 7404).

1 Circuito controlador de escritura (basado en C.I. TTL 74121).

1 Circuito de reset (basado en una red RC).

Con lo anterior trabajamos hasta obtener la lógica de los programas de detección de llamada y almacenamiento en memoria, los cuales fueron obtenidos por medio de un sistema de microcomputador que podimos usar gracias a que en épocas de la Universidad, diseñamos y desarrollamos para utilizarlo como herramienta de programación y prueba de sistemas basados en el microprocesador Z80A.

Así que hicimos la programación de detección de llamada y despliegue, desarrollándose dos modos:

1) El modo de detección.
2) El modo memoria.

El primero se encuentra detectando las llamadas telefónicas y grabándolas. El segundo permite ver las llamadas grabadas en memoria.

Estos modos se programaron exclusivos, aunque podría cambiarse la programación para hacerlos paralelos en el modo 2.

Se determinó que el número máximo de digitos de una llamada sería de 13, ya que se necesitaban 2 dígitos para indicar el número de llamada cuando se tuviese en memoria, y una más para indicar el fin del número telefónico.

8.1 Configuración del PPI8255 y del LCD.

Subrutina INIT, inicializa el display y configura el puerto 8255.

a) Configuración de puertos.

De acuerdo al mapa de memoria diseñado las direcciones del 8255A quedan:

```
PA    En la dirección X'4000        ;Puerto A
PB    En la dirección X'4001        ;Puerto B
PC    En la dirección X'4002        ;Puerto C
RC    En la dirección X'4003        ;Puerto de Configuración.
```

Lo que se desea es la siguiente configuración de I/O:

- PA0-PA7 Salidas
- PB0-PB7 Entradas
- PC0-PC3 Entradas
- PC4-PC7 Salidas

De acuerdo a la sección 7.2, el dato que se debe guardar en el registro de configuración es 1 0 0 0 0 0 1 1 B = 83 H.

b) Configuración del LCD.

- Longitud de dato = 8 bits, comando al display = 38
- 2 líneas de display, comando al display = 0F
- Encendido de display con parpadeo de cursor, comando al display = 06
- Limpiar display, comando al display = 01

```
INIT    LD A,83          ;Prepara el dato de configuración del puerto
        LD (4003),A      ;y lo guarda en RC.

        LD A,00          ;Prepara el dato en nivel lógico 0, para
        LD (4002),A      ;generación de escritura de comando al display.

        LD A,38          ;Envía
        LD (4000),A      ;el comando
```

```
        CALL RETA       ;de longitud de dato = 8 bits.

        LD A,0F         ;Envía
        LD (4000),A     ;el comando
        CALL RETA       ;de selección de 2 líneas de display.

        LD A,06         ;Envía
        LD (4000),A     ;el comando de encendido
        CALL RETA       ;del display con parpadeo del cursor.

        LD A,01         ;Envía
        LD (4000),A     ;el comando
        CALL RETA       ;de limpiar display.

        RET             :Regresa al programa principal.
```

8.2 Subrutinas.

Estas subrutinas fueron creadas poco a poco, conforme se fue realizando el programa principal, y se ponen todas de una vez para no ser repetitivos.

Subrutina LIMP, limpia display.

```
LIMP    PUSH AF
        LD A,01
        LD (4000),A
        CALL RETA
        POP AF
        RET
```

Subrutina RETA, genera pulso para E del display en modo comando.

Modo comando RS = 0

```
RETA    PUSH AF
        PUSH BC
        LD A,10
        LD (4002),A
        LD A,00
        LD (4002),A
        POP BC
        POP AF
        RET
```

Subrutina RETB, retardo y generación de pulso E del display en modo dato.

Modo dato RS = 1

```
RETB    PUSH AF
        PUSH BC
        LD A,30
        LD (4002),A
        LD B,FF
ETJ0    LD C,FF
ETJ1    DEC C
        JP NZ,ETJ1
        DEC B
        JP NZ,ETJ0
        LD A,20
        LD (4002),A
        POP BC
        POP AF
        RET
```

Subrutina WAIT, retardo que genera un tiempo de espera entre dígito y dígito detectados.

```
WAIT    PUSH AF
        PUSH BC
        LD B,FF
WAI0    LD C,FF
WAI1    DEC C
        JP NZ,WAI1
        DEC B
        JP NZ,WAI0
        POP BC
        POP AF
        RET
```

Subrutina INTE, detecta si se ha oprimido la tecla de petición de memoria.

```
INTE    LD A,(4001)     ;Detecta si se ha oprimido la tecla de petición
        AND F0
        CP E0
        JP NZ,NUEV      ;Si no es la tecla de petición, continúa en modo
                        ;de detección
```

```
ESTA    LD A,(4001)     ;Si fue la tecla de petición, espera a que la
        AND F0          ;suelten
        CP F0
        JP NZ,ESTA
        JP LECT         ;y después ejecuta el modo de memoria.
```

Subrutina SACH, despliega en el display el número contenido en HL.

```
SACH    LD A,20
        LD (4002),A
        LD A,(HL)
        LD (4000),A
        CALL RETB
        RET
```

Subrutina DESP, despliega una llamada cuando se ha oprimido la tecla PGUP (botón 2).

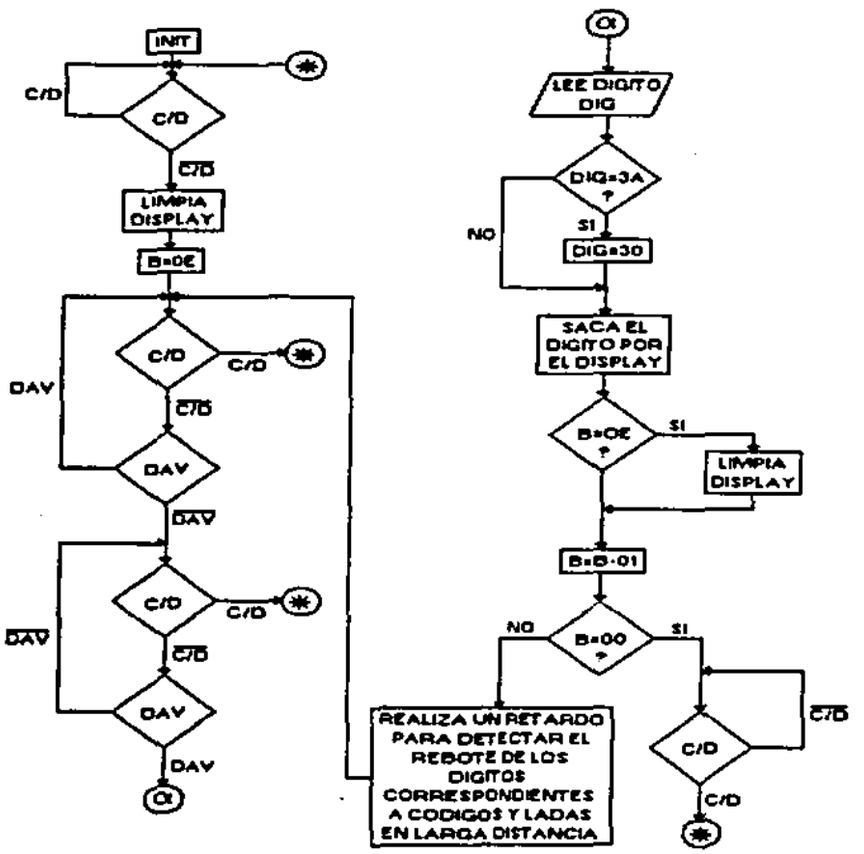
```
DESP    CALL LIMP       ;Limpia el display.
        LD E,11
DUMY    LD A,2A         ;DUMY manda uno a uno los dígitos
        CP (HL)         ;de una llamada, hasta encontrar
        JP Z,FINI       ;el valor Dumy = '*', después del
        CALL SACH       ;cual enviará espacios.
        INC HL
        DEC E
        LD A,E
        CP 03
        JP Z,FINJ
        JP DUMY
FINI    LD A,20         ;En esta parte
        LD (4002),A     ;envía los
        LD A,20         ;espacios al display.
        LD (4000),A
        CALL RETB
        INC HL
        DEC E
        LD A,E
        CP 03
        JP NZ,FINI
FINJ    CALL SACH       ;En esta parte
        INC HL          ;envía los
        CALL SACH       ;números al display.
        INC HL
```

RET

8.3 Programa para detección del número y presentación visual.

El listado que se presenta a continuación es parte de todo el programa, para ver el código refierase al listado del programa principal. Este listado sólo nos da idea de cómo se vería el programa principal, si sólo detectara llamadas y presentación visual; a continuación lo veremos:

Diagrama de flujo.



Programa.

```
NUEV    LD A,(4001)      ;Detecta si se ha descolgado el teléfono.
        AND 01
        CP 00
        JP NZ,NUEV
        CALL LIMP
        LD B,0E

ETIO    LD A,(4001)      ;Detecta si se ha colgado el teléfono.
        AND 01
        CP 01
        JP OZ,****

        LD A,(4001)      ;Detecta si el dato válido está en bajo.
        AND 02
        CP 00
        JP NZ,ETIO

ETIP    LD A,(4001)      ;Detecta si se ha colgado el teléfono.
        AND 01
        CP 01
        JP Z,****

        LD A,(4001)      ;Detecta si el dato válido está en alto.
        AND 02
        CP 02
        JP NZ,ETIP
```

ETIO y ETIP, conjuntamente detectan si el teléfono fue colgado, y si se activó la entrada de dato válido (DAV), al pasar éste de bajo a alto (transición positiva) validando el dígito detectado.

```
        LD A,20          ;Prepara a E y RS para escribir dato en
        LD (4002),A      ;display (E=0 RS=1).

        LD A,(4002)      :Lee dígito y compara con 10
        AND 0F
        OR 30
        CP 3A
        JP NZ,SACA
        LD A,30          ;Si el dígito es igual a 10, lo convierte a 0.
```

```
SACA    LD (4000),A     ;Revisa el valor de B, el cual nos dice el
        CALL RETB       ;# de dígitos que se se llevan marcados en
        LD C,A          ;una llamada. Si B=0E, entonces se acaba
        LD A,B          ;de descolgar, y como en ese momento
        CP 0E           ;manda basura al display, hay que limpiarlo.
        JP NZ,NINH
        CALL LIMP

NINH    DEC B           ;Se lleva la cuenta de números marcados:
        JP Z,BAJO       ;si es mayor de 13, los siguientes dígitos no
        CALL WAIT       ;se toman en cuenta.
        JP ETIO

BAJO    LD A,(4001)     ;Esta es la rutina que ignora los dígitos
        AND 01          ;posteriores a los 13 primeros, y detecta si
        CP 00           ;se colgó el teléfono.
        JP Z,BAJO
        JP NUEV
```

8.4 Programa para escritura en memoria del número detectado.

Ya se tienen reservadas las localidades para las llamadas. La localidad 2000 es un puntero que indica cuál es la localidad de memoria en la cual ha de empezar a grabarse la llamada. Como a continuación se muestra, la dirección donde ha de empezar la siguiente llamada es en 2010.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2000	10	20	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
2010	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	30	30
2020	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	30	31
2030	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	30	32
2040	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	30	33
2050	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	30	34
2060	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	30	35
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
2660	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	39	39
2670																

Si se marcara el teléfono 7033881, la memoria quedará de la siguiente forma (hay que apreciar como el puntero 2000 se ha incrementado a 2020):

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2000	20	20	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
2010	37	30	33	33	38	38	31	2A	A0	A0	A0	A0	A0	A0	30	30
2020	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	30	31
2030	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	30	32
2040	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	30	33
2050	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	30	34
2060	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	30	35
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
2660	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	39	39
2670																

Si se marcara después el teléfono 9054041093, la memoria quedará de la siguiente forma (hay que apreciar cómo el puntero 2000 se incrementa a 2030):

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2000	30	20	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
2010	37	30	33	33	38	38	31	2A	A0	A0	A0	A0	A0	A0	30	30
2020	39	30	35	34	30	34	31	30	39	33	2A	A0	A0	A0	30	31
2030	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	30	32
2040	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	30	33
2050	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	30	34
2060	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	30	35
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
2660	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	39	39
2670																

Grabación de llamadas.

Las líneas de programa que se presentan a continuación forman parte del programa principal, aquí sólo se muestran para explicarlas e identificarlas; para un análisis general, ver el diagrama de flujo principal.

```
ETI2    LD HL,(2000)    ;La dirección 2000 es el puntero que nos va
                        ;indicando cuál dirección de memoria es en la
                        ;que ha de empezar a grabarse la siguiente
                        ;llamada (la llamada 0 empieza en la dirección
                        ;2010).

DETB    LD A,C          ;Una vez mandado el dígito al display, se
```

```
                LD (HL),A       ;guarda en memoria mediante el apuntador HL.
                INC HL          ;Se incrementa HL para el siguiente dígito.

****            LD A,B          ;Líneas de programa que invalidan llamadas
                CP 0C           ;con uno o menos dígitos; es decir, si la gente
                JP NC,CARG      ;marca un sólo dígito y cuelga, éste no se
                                ;tomará como llamada y no se almacenará.
                                ;Tampoco se almacenará cuando sólo se
                                ;descuelgue y cuelgue el teléfono sin haber
                                ;marcado dígitos,lo anterior se cumple cuando
                                ;B>=0C.
                                ;Si fue válida la llamada, entonces se mantiene
                                ;en memoria, y la siguiente rutina incrementa a
                                ;HL para que apunte la dirección de inicio de la
                                ;próxima llamada

                LD (HL),2A      ;Si fue llamada válida, se pone un * para saber
                                ;dónde termina.

                LD HL,(2000)    ;Detecta que el número de llamadas no sea
                LD E,10         ;mayor de 100; si es así, se sigue el curso
INC1            INC HL          ;normal del programa.
                DEC E
                JP NZ,INC1
                LD (2000),HL
                LD A,70
                CP L
                JP NZ,ETI3
                LD A,26
                CP H
                JP NZ,ETI3

                LD A,10         ;Si el número de llamadas es mayor de 100, el
                LD (2000),A     ;puntero 2000 se ajusta a 2010, con lo cual es
                LD A,20         ;como si borráramos las cien llamadas y
                LD (2001),A     ;empezáramos a llenarlas otra vez.
                JP ETI2

CARG            LD HL,(2000)    ;Mantiene sin cambios al apuntador 2000,
                JP ETI3         ;cuando la llamada no fue validada.
```

8.5 Programa para lectura y presentación visual.

Programación de botones 1 y 2.

Las líneas de programa que se presentan a continuación forman parte del programa principal, aquí sólo se muestran para explicarlas e identificarlas, para un análisis general ver el diagrama de flujo principal.

```
ETI3    JP INTE          ;Llamado a la subrutina INTE, ver subrutina 6.

LECT    LD HL,2010       ;Ajusta puntero HL a llamada 0.
INIQ    CALL DESP        ;Envía al display la llamada apuntada por HL.

DECB    LD A,(4001)      ;Detecta si no se ha oprimido ninguna tecla; si
        AND F0           ;es así, se mantiene en el bucle DECB
        CP F0
        JP Z,DECB

DECC    LD C,A           ;Si se oprimió la tecla, esta rutina detecta que la
        LD A,(4001)      ;misma sea soltada, para continuar. Si no deja
        AND F0           ;de ser oprimida, se mantiene la secuencia del
        CP F0            ;programa en el bucle DECC
        JP NZ,DECC

        LD A,C           ;Al ser soltada la tecla se verifica si fue la de
        CP E0            ;petición, con lo cual se regresa al modo de
        JP Z,SUBR        ;detección.

        CP D0            ;Si fue la tecla de avance, llama a la rutina de
        JP Z,INCR        ;avance INCR

        CP B0            ;Si fue la tecla de retroceso, llama a la rutina de
        JP Z,DECR        ;retroceso DECR

        JP DECB          ;Si no fue ninguna, explora al teclado.

INCR    LD A,(2001)      ;Rutina de avance:
        CP H             ;incrementa el puntero 2000, y si no sobrepasa
        JP Z,IIII        ;el número de llamadas realizadas,
        JP C,LECT        ;la envía al display (INIQ).
        JP INIQ          ;Si sobrepasa el número de llamadas

IIII    LD A,(2000)      ;realizadas vuelve a la llamada cero y la envía
```

```
                CP L            ;al display (LECT).
                JP Z,LECT
                JP C,LECT
                JP INIQ

DECR            LD A,20         ;Rutina de retroceso:
                CP H            ;verifica que el byte alto del puntero 2000
                JP Z,CMPL       ;se encuentre en el rango 2000-2670, de no
                JP NC,REIN      ;ser así, reinicializarlo a 2010 (REIN)
                LD A,26
                CP H
                JP Z,CPL70
                JP NC,MINHL
                JP REIN

CPL70           LD A,70
                CP L
                JP Z,MINHL
                JP NC,MINHL
                JP REIN

MINHL           LD C,20         ;Si se encuentra en el rango,
CONHL           DEC HL          ;decrementa y
                DEC C
                JP NZ,CONHL
                JP INIQ         ;despliega.

CMPL            CP L
                JP Z,BEGIN
                JP NC,REIN
                JP MINHL

REIN            LD A,10
                LD (2000),A
                LD A,20
                LD (2001),A
                JP INIQ

BEGIN           LD HL,(2000)
                JP INIQ

SUBR            CALL LIMP
                JP ETI2
```

8.6 Programa final.

```
Dir.   Código      Etiqueta  Mnemónico
0000   31 FF 3F             LD SP,(X'3FFF)
0003   C3 20 01             JP PRIN
0006   3E 83       INIT     LD A,X'83
0008   32 03 40             LD (X'4003),A
000B   3E 00                LD A,X'00
000D   32 02 40             LD (X'4002),A
0010   3E 38                LD A,X'38
0012   32 00 40             LD (X'4000),A
0015   CD 3C 00             CALL RETA
0018   3E 0F                LD A,X'0F
001A   32 00 40             LD (X'4000),A
001D   CD 3C 00             CALL RETA
0020   3E 06                LD A,X'06
0022   32 00 40             LD (X'4000),A
0025   CD 3C 00             CALL RETA
0028   3E 01                LD A,X'01
002A   32 00 40             LD (X'4000),A
002D   CD 3C 00             CALL RETA
0030   C9                   RET

0031   F5          LIMP     PUSH AF
0032   3E 01                LD A,X'01
0034   32 00 40             LD (X'4000),A
0037   CD 3C 00             CALL RETA
003A   F1                   POP AF
003B   C9                   RET

003C   F5          RETA     PUSH AF
003D   C5                   PUSH BC
003E   3E 10                LD A,X'10
0040   32 02 40             LD (X'4002),A
0043   3E 00                LD A,X'00
0045   32 02 40             LD (X'4002),A
0048   C1                   POP BC
0049   F1                   POP AF
004A   C9                   RET

004B   F5          RETB     PUSH AF
004C   C5                   PUSH BC
004D   3E 30                LD A,X'30
004F   32 02 40             LD (X'4002),A
0052   06 FF                LD B,X'FF
0054   0E FF       ETJ0     LD C,X'FF
0056   0D         ETJ1     DEC C
0057   C2 56 00             JP NZ,ETJ1
```

```
005A 05                  DEC B
005B C2 54 00            JP NZ,ETJ0
005E 3E 20               LD A,X'20
0060 32 02 40            LD (X'4002),A
0063 C1                  POP BC
0064 F1                  POP AF
0065 C9                  RET

0066 F5            WAIT  PUSH AF
0067 C5                  PUSH BC
0068 06 FF               LD B,X'FF
006A 0E FF         WAI0  LD C,X'FF
006C 0D            WAI1  DEC C
006D C2 6C 00            JP NZ,WAI1
0070 05                  DEC B
0071 C2 6A 00            JP NZ,WAI0
0074 C1                  POP BC
0075 F1                  POP AF
0076 C9                  RET

0077 3A 01 40      INTE  LD A,(X'4001)
007A E6 F0               AND X'F0
007C FE E0               CP X'E0
007E C2 29 01            JP NZ,NUEV
0081 3A 01 40      ESTA  LD A,(X'4001)
0084 E6 F0               AND X'F0
0086 FE F0               CP X'F0
0088 C2 81 00            JP NZ,ESTA
008B C3 00 02            JP LECT

0098 3E 20         SACH  LD A,X'20
009A 32 02 40            LD (X'4002),A
009D 7E                  LD A,(HL)
009E 32 00 40            LD (X'4000),A
00A1 CD 4B 00            CALL RETB
00A4 C9                  RET

00A5 CD 31 00      DESP  CALL LIMP
00A8 1E 11               LD E,X'11
00AA 3E 2A         DUMY  LD A,X'24
00AC BE                  CP (HL)
00AD CA BE 00            JP Z,FINI
00B0 CD 98 00            CALL SACH
00B3 23                  INC HL
00B4 1D                  DEC E
00B5 7B                  LD A,E
00B6 FE 03               CP X'03
00B8 CA D3 00            JP Z,FINJ
```

```
00BB C3 AA 00          JP DUMY
00BE 3E 20       FINI  LD A,X'20
00C0 32 02 40          LD (X'4002),A
00C3 3E 20             LD A,X'20
00C5 32 00 40          LD (X'4000),A
00C8 CD 4B 00          CALL RETB
00CB 23                INC HL
00CC 1D                DEC E
00CD 7B                LD A,E
00CE FE 03             CP X'03
00D0 C2 BC 00          JP NZ,FINI
00D3 CD 98 00    FINJ  CALL SACH
00D6 23                INC HL
00D7 CD 98 00          CALL SACH
00DA 23                INC HL
00DB C9                RET

0120 CD 06 00    PRIN  CALL INIT
0123 2A 00 20    ETI2  LD HL,(X'2000)
0126 C3 77 00    ETI3  JP INTE
0129 3A 01 40    NUEV  LD A,(X'4001)
012C E6 01             AND X'01
012E FE 00             CP X'00
0130 C2 26 01          JP NZ,ETI3
0133 CD 31 00          CALL LIMP
0136 06 0E             LD B,X'0E
0138 3A 01 40    ETIO  LD A,(X'4001)
013B E6 01             AND X'01
013D FE 01             CP X'01
013F CA A0 01          JP Z,****
0142 3A 01 40          LD A,(X'4001)
0145 E6 02             AND X'02
0147 FE 00             CP X'00
0149 C2 38 01          JP NZ,ETIO
014C 3A 01 40    ETIP  LD A,(X'4001)
014F E6 01             AND X'01
0151 FE 01             CP X'01
0153 CA A0 01          JP Z,****
0156 3A 01 40          LD A,(X'4001)
0159 E6 02             AND X'02
015B FE 02             CP X'02
015D C2 4C 01          JP NZ,ETIP
0160 3E 20             LD A,X'20
0162 32 02 40          LD (X'4002),A
0165 3A 02 40          LD A,(X'4002)
0168 E6 0F             AND X'0F
016A F6 30             OR X'30
016C FE 3A             CP X'3A
```

```
016E  C2 73 01          JP NZ,SACA
0171  3E 30             LD A,X'30
0173  32 00 40  SACA    LD (X'4000),A
0176  CD 4B 00          CALL RETB
0179  4F                LD C,A
017A  78                LD A,B
017B  FE 0E             CP X'0E
017D  C2 86 01          JP NZ,DETB
0180  CD 31 00          CALL LIMP
0183  C3 89 01          JP NINH
0186  79          DETB  LD A,C
0187  77                LD (HL),A
0188  23                INC HL
0189  05          NINH  DEC B
018A  CA 93 01          JP Z,BAJO
018D  CD 66 00          CALL WAIT
0190  C3 38 01          JP ETI0
0193  3A 01 40  BAJO    LD A,(X'4001)
0196  E6 01             AND X'01
0198  FE 00             CP X'00
019A  CA 93 01          JP Z,BAJO
019D  C3 A0 01          JP ****

01A0  78          ****  LD A,B
01A1  FE 0C             CP X'0C
01A3  D2 F0 01          JP NC,CARG
01A6  36 2A             LD (HL),X'2A
01A8  2A 00 20          LD HL,(X'2000)
01AB  1E 10             LD E,X'10
01AD  23          INC1  INC HL
01AE  1D                DEC E
01AF  C2 AD 01          JP NZ,INC1
01B2  22 00 20          LD (X'2000),HL
01B5  3E 70             LD A,X'70
01B7  BD                CP L
01B8  C2 26 01          JP NZ,ETI3
01BB  3E 26             LD A,X'26
01BD  BC                CP H
01BE  C2 26 01          JP NZ,ETI3
01C1  00                NOP
01C2  00                NOP
01C3  3E 10             LD A,X'10
01C5  32 00 20          LD (X'2000),A
01C8  3E 20             LD A,X'20
01CA  32 01 20          LD (X'2001),A
01CD  C3 23 01          JP ETI2
01F0  2A 00 20  CARG    LD HL,(X'2000)
01F3  C3 26 01          JP ETI3
```

```
0200 21 10 20  LECT  LD HL,X'2010
0203 CD A5 00  INIQ  CALL DESP

0206 3A 01 40  DECB  LD A,(X'4001)
0209 E6 F0           AND X'F0
020B FE F0           CP X'F0
020D CA 06 02        JP Z,DECB

0210 4F        DECC  LD C,A
0211 3A 01 40        LD A,(X'4001)
0214 E6 F0           AND X'F0
0216 FE F0           CP X'F0
0218 C2 10 02        JP NZ,DECC

021B 79              LD A,C
021C FE E0           CP X'E0
021E CA A0 02        JP Z,SUBR

0221 FE D0           CP X'D0
0223 CA 2E 02        JP Z,INCR

0226 FE B0           CP X'B0
0228 CA 48 02        JP Z,DECR

022B C3 06 02        JP DECB

022E 3A 01 20  INCR  LD A,(X'2001)
0231 BC              CP H
0232 CA 3B 02        JP Z,IIII
0235 DA 00 02        JP C,LECT
0238 C3 03 02        JP INIQ

023B 3A 00 20  IIII  LD A,(X'2000)
023E BD              CP L
023F CA 00 02        JP Z,LECT
0242 DA 00 02        JP C,LECT
0245 C3 03 02        JP INIQ

0248 3E 20     DECR  LD A,X'20
024A BC              CP H
024B CA 73 02        JP Z,CMPL
024E D2 7D 02        JP NC,REIN
0251 3E 26           LD A,X'26
0253 BC              CP H
0254 CA 5D 02        JP Z,CPL70
0257 D2 69 02        JP NC,MINHL
025A C3 7D 02        JP REIN
```

```
025D 3E 70            CPL70 LD A,X'70
025F BD               CP L
0260 CA 69 02         JP Z,MINHL
0263 D2 69 02         JP NC,MINHL
0266 C3 7D 02         JP REIN

0269 0E 20      MINHL LD C,X'20
026B 2B         CONHL DEC HL
026C 0D               DEC C
026D C2 6B 02         JP NZ,CONHL
0270 C3 03 02         JP INIQ

0273 BD         CMPL  CP L
0274 CA 8A 02         JP Z,BEGIN
0277 D2 7D 02         JP NC,REIN
027A C3 69 02         JP MINHL

027D 3E 10      REIN  LD A,X'10
027F 32 00 20         LD (X'2000),A
0282 3E 20            LD A,X'20
0284 32 01 20         LD (X'2001),A
0287 C3 03 02         JP INIQ

028A 2A 00 20  BEGIN LD HL,(X'2000)
028D C3 03 02         JP INIQ

02A0 CD 31 00  SUBR   CALL LIMP
02A3 C3 23 01         JP ETI2
```

CONCLUSIONES.

Las conclusiones dan respuesta a las siguientes preguntas.

1.- ¿Cuál es el aspecto que se debe tomar en cuenta para crear la solución a un problema?

Un aspecto importante del actual trabajo de tesis nos hace ver que el conocimiento y la imaginación de un ingeniero debe ir de la mano con las necesidades de la sociedad para obtener resultados prácticos y útiles.

Dicho aspecto es el de aprender a vender un producto, basado en un análisis de beneficios técnicos, pero más importante de su funcionalidad e impacto económico. Este último debe ser inmediato desde el punto de vista comercial.

2.- ¿Cuando, un ingeniero en México, debe desarrollar controles basados en microprocesador ?

El desarrollo de los circuitos integrados a revolucionado el campo de aplicación de la electrónica y debe ser explotado en México en mercados en los que las computadoras (PC'S) y los controles lógicos programables (PLC'S) no puedan ser empleados por requerimientos de bajo precio.

Las condiciones de uso es el número de piezas a producir. Cuanto mayor sea el volumen, más viable es el uso de técnicas para desarrollo de controles, dado que la inversión para el proceso de producción se recupera rápidamente.

De otra forma, al no existir un mercado con amplia repetibilidad, habrá que usar equipos ya existentes como los mencionados anteriormente, que son de propósito general.

3.- ¿Cuales son la industrias y en que forma ha de poderse usar el diseño por microprocesador en México?

Industrias en que hay posibilidad para diseño de controles se mencionan a continuación.

1. Industria del Plástico.
2. Industria Textil.
3. Industria del Papel.
4. Industria del Agro.
5. Industria de Linea blanca y Electrodomésticos.
6. Industria del Acero.
7. Industria de la Publicidad.
8. Industria de Telecomunicaciones.
9. Industria Farmacéutica.

Su uso puede ser variado, vea a continuación algunas recomendaciones.

Equipos de aplicación del diseño por microprocesador.

- Controles de tiempo (Temporizadores), para diversos procesos.
- Controladores de Temperatura, para diversos procesos.
- Controladores de par y velocidad en motores de extrusoras y laminadoras.
- Control de proceso en inyectoras.
- Controles remotos (radio e infrarrojos).
- Controles de seguridad.
- Indexadores para posicionado y dosificado.
- Controles de intensidad de luz.
- Colectores de Datos.

4.- ¿Cuál es la posibilidad de diseñar equipo basado en microprocesador, en México?

Las posibilidades son grandes, ya que en la actualidad hay equipos que se importan de Estados Unidos (algunas marcas son: Microbit, SMT) y nos ofrecen un controlador con interfaz para usuario, basados en microprocesadores y microcontroladores (Intel, Microchips, Motorola), que nosotros conocemos y con los cuales podemos desarrollar hardware con la misma capacidad y económicos (eliminando a un gran número de intermediarios del producto final), y desarrollar un poderoso software de programación vía puerto serial, empleando herramientas como Visual C++, Visual Basic, etc. desde una PC.

CONCLUSIONES

BIBLIOGRAFÍA.

1. Z80 Family Data Book(January 1989).
   Zilog.

2. Microprocesador Z80, Programación e Interfaces.
   Nichols, Joseph c. et al
   Publicaciones Marcombo S.A.

3. Diseño con circuitos integrados TTL.
   Morris L. Robert, John R. Miller.
   C.E.C.S.A.

4. Lógica Digital y Diseño de Computadores.
   Morris Mano, M.
   Prentice Hall.

5. Manual de Semiconductores de Silicio.
   Texas Instruments.
   Edición Técnica 82/83.

6. Construya una Microcomputadora basado en el Z80.
   (Guía de Diseño y Funcionamiento).
   Ciarcia, Steve.
   McGraw-Hill.