

PROLOGO

Esta obra tiene como propósito primordial presentar los conceptos básicos sobre organización y arquitectura computacional para estudiantes del área de computación, partiendo desde las bases mismas. No se requieren conocimientos previos en el área. El libro está diseñado para ser utilizado como libro de texto básico y recopila la experiencia de varios años de impartir las materias correspondientes por parte de los autores..

Este libro está dividido en cuatro secciones principales. La primera de ellas comprende los capítulos del 2 al 6. En esta sección se describe la arquitectura básica de una computadora digital simple, sus diferentes componentes, la interacción entre estos componentes y las operaciones típicas que una computadora digital puede efectuar. Dentro de esta sección también se presentan los principales dispositivos periféricos de las computadoras digitales y sus principios de operación. Para dar un panorama más completo sobre los sistemas computacionales, se presenta la evolución histórica de los lenguajes de programación así como las diferencias entre los tipos de lenguajes computacionales existentes. Se incluye un capítulo que describe brevemente la función de un sistema operativo así como los servicios que brinda.

Todas las computadoras digitales trabajan internamente con el sistema binario. En este sistema de números solamente existen los dígitos binarios cero y uno. En la primera sección, se muestra la operación de una computadora digital hipotética que trabaja con el sistema de numeración decimal ya que se pretende ejemplificar su operación básica sin tener que aprender a utilizar el sistema binario.

La segunda sección del libro, que comprende los capítulos 7, 8 y 9, trata los sistemas de números posicionales científicos, utilizando diferentes bases. Se presentan las operaciones aritméticas básicas de suma y resta en los sistemas posicionales científicos y se definen los complementos a la base y a la base disminuida, así como su aplicación en las operaciones de resta.

Las computadoras digitales pueden manipular diferentes tipos de datos. En esta segunda sección también se describen las principales formas de representar cada uno de los diferentes tipos de datos internamente en las computadoras digitales, utilizando el sistema binario. Finalmente, se describe la forma en que se realizan las operaciones aritméticas, utilizando los diferentes tipos de datos y formatos de representación interna.

La tercera sección, que comprende los capítulos 10 a 16 inclusive, está dedicada a los circuitos lógicos combinatorios, que son la base de la unidad aritmética-lógica de las computadoras digitales. En esta sección se presentan los fundamentos del álgebra booleana, sus principales teoremas, las funciones

booleanas, los métodos de minimización de funciones booleanas y su aplicación en el diseño de circuitos lógicos combinatorios.

Se definen también las diferentes compuertas lógicas y la forma de construir funciones booleanas a partir de estas componentes, terminando con el diseño de una unidad aritmética-lógica simple, pero ya utilizando el sistema binario.

La última sección, que está formada por los capítulos 17 a 24, se enfoca a los circuitos lógicos secuenciales, que son la base para la unidad de control de las computadoras digitales. Esta sección inicia con las diferentes celdas básicas de memoria existentes y su utilización en la construcción de registros y unidades de memoria. Se procede posteriormente con el diseño de circuitos lógicos secuenciales hasta llegar al diseño completo de la unidad de control de una computadora digital simple. También se define el concepto de microprogramación y se usa para diseñar el juego básico de instrucciones de una computadora digital sencilla.

Se recomienda cubrir las primeras dos secciones (capítulos 1 a 9) en un curso durante el segundo o tercer semestre del plan de estudios, la tercera sección (capítulos 10 a 16) en un segundo curso y la cuarta sección (capítulos 17 a 24) en un tercer curso.

Si los cursos no están ubicados durante los primeros semestres del plan de estudios, se podría cubrir el material completo en dos cursos, incluyendo los capítulos 1 al 13 en un primer curso y los capítulos del 14 al 24 en un segundo curso, dado que los alumnos tendrían una mayor madurez y podrían asimilar el material más rápidamente.

Un punto que vale la pena mencionar es que todos los mnemónicos de las instrucciones de máquina así como las abreviaturas de los nombres de los componentes de las computadoras digitales y los nombres de las compuertas lógicas se utilizan en inglés. La razón por la cual se decidió usar esta nomenclatura es que los juegos de instrucciones de prácticamente todas las computadoras digitales están en inglés. Adicionalmente, los desarrollos en esta área se suceden de manera muy rápida y es necesario conocer los términos en inglés para que un profesional se mantenga actualizado, consultando los adelantos que surgen día a día.

Finalmente, queremos agradecer a todas las personas que contribuyeron a darle la forma final a este texto realizando lecturas de prueba y dando sus comentarios para mejorar la secuencia y presentación del material.

CAPITULO 1

INTRODUCCION

1.1. LA COMPUTADORA DIGITAL

A través de la historia, el hombre ha realizado descubrimientos científicos importantes tales como la electricidad y la energía nuclear, los cuales han permitido realizar avances tecnológicos que han transformado la forma de vida y la sociedad. Las máquinas de vapor, las máquinas de combustión interna, los automóviles, el telégrafo, el teléfono, la radio, los aviones, la televisión y los transbordadores espaciales entre otros, sin duda han tenido un fuerte impacto en la sociedad y han contribuido a mejorar la forma de vida en general.

En el presente siglo, los desarrollos tecnológicos se suceden uno tras otro a una velocidad nunca antes imaginable. Este rápido avance de las ciencias y la tecnología no sería posible sin la ayuda de las computadoras digitales. Debido a esto, muchas personas han afirmado que el siglo veinte es la era del procesador electrónico, o la era de la información.

Una computadora digital es un dispositivo electrónico que se utiliza para el procesamiento de datos. Las principales características que tiene una de estas máquinas son:

- Puede realizar operaciones aritméticas y lógicas a una gran velocidad y de manera confiable.
- Puede realizar comparaciones de datos con mucha rapidez, tomando acciones diferentes dependiendo del resultado de la comparación.
- Tiene una memoria en la cual puede almacenar tanto datos como instrucciones. Estas instrucciones le indican como procesar los datos.
- Puede ejecutar en secuencia un conjunto de instrucciones almacenadas en la memoria para realizar el procesamiento de datos. Este conjunto de instrucciones que definen como se procesan los datos recibe el nombre de programa.
- Puede transferir resultados de las operaciones realizadas a ciertos dispositivos que permiten que el usuario de la computadora pueda verlos. Algunos de estos dispositivos son una pantalla de vídeo, una impresora o un graficador. Los resultados de las operaciones también pueden ser transferidos a unidades de cinta magnética, discos magnéticos o disketes para guardarlos en forma permanente y utilizarlos posteriormente.
- Puede recibir tanto programas como datos de ciertos dispositivos como el teclado, el ratón, discos magnéticos, unidades de cinta, disketes y otros dispositivos.

Los diferentes dispositivos a los que se hizo mención anteriormente se conocen como dispositivos periféricos que pueden ser conectados a la computadora digital, pero no forman parte de ella.

El hecho de que se le puedan alimentar tanto los datos como las instrucciones a través de los diferentes dispositivos periféricos, hace que las computadoras digitales sean muy versátiles en cuanto al procesamiento de datos ya que se pueden alimentar programas para propósitos muy variados. Por esta razón se dice que las computadoras digitales son máquinas de propósito general.

Al ejecutarse un programa en una computadora digital, ésta puede interactuar con el usuario a través de los diferentes dispositivos periféricos, bajo el control del programa. Esto permite que la secuencia de operaciones que se ejecutan en la computadora dependa de estas interacciones y se tomen diferentes acciones, dependiendo de los datos que se le alimenten en forma interactiva.

La computadora digital también puede leer datos de dispositivos tales como sensores de temperatura, sensores de humedad, sensores de posición, etc. y enviar sus resultados a dispositivos conocidos como actuadores lo cual permite que estas máquinas puedan también ser utilizadas para controlar otros dispositivos.

Al conjunto de la computadora digital con sus periféricos, los programas y los datos almacenados en ellos se les conoce comúnmente como sistema computacional.

1.2. APLICACIONES DE LAS COMPUTADORAS DIGITALES

Como ya se mencionó anteriormente, las computadoras digitales son máquinas de propósito general que pueden ser utilizadas en diferentes tareas. A continuación se enumeran algunas de las principales aplicaciones que se ha dado a estas máquinas.

Sistemas administrativos.

La mayoría de los sistemas administrativos de las empresas actualmente se apoyan fuertemente en computadoras digitales para realizar el procesamiento y almacenamiento de datos en una manera rápida y confiable, lo cual les permite tener información veraz y oportuna para tomar decisiones. Algunos de estos sistemas son contabilidad, nómina, control de cartera, compras, cuentas por pagar, cuentas por cobrar, ventas, facturación, control de inventarios, inversiones, estudios financieros, estudios de mercado, etc.

Control de procesos.

En estas aplicaciones, las computadoras digitales interactúan con sensores y actuadores instalados en procesos industriales para controlarlos y elevar tanto la calidad de los productos como la eficiencia de los procesos. Pueden controlarse líneas de producción, máquinas

herramientas de control numérico, robots, reactores químicos, reactores nucleares y, en general, una gran variedad de procesos industriales.

Control de dispositivos específicos.

En forma similar a la aplicación de las computadoras digitales al control de procesos industriales, se han desarrollado computadoras digitales para controlar dispositivos específicos. Estas computadoras están basadas primordialmente en microprocesadores y microcontroladores. Las aplicaciones incluyen el control de inyección de gasolina y el control del sistema de frenado antibloqueante en los automóviles, el control de la operación de dispositivos domésticos como hornos de microondas, vídeograbadoras, lavadoras automáticas, sistemas de aire acondicionado, sistemas de calefacción, sistemas de alarma y conmutadores telefónicos pequeños entre otros. También son utilizadas en los sistemas de navegación de los aviones y del transbordador espacial, y en control de armamento en aviones, barcos, submarinos y tanques de guerra.

Diseño asistido por computadora.

El diseño mecánico puede ser apoyado fuertemente por las computadoras digitales mediante sistemas computacionales que permitan al diseñador definir los componentes del sistema mecánico, visualizarlos desde diferentes ángulos en una pantalla de vídeo, realizar cálculo de esfuerzos, cálculos cinemáticos, enviar las especificaciones del diseño a un graficador para obtener los dibujos y enviar las instrucciones necesarias a las máquinas herramientas y líneas de producción para su fabricación. Estos sistemas se conocen como sistemas de CAD/CAM (que significa Computer Aided Design/Computer Aided Manufacturing).

También existen sistemas de apoyo en otras áreas como arquitectura y diseño de componentes electrónicos.

Simulación.

Otra de las aplicaciones de las computadoras digitales es la simulación de procesos complejos. Estas simulaciones permiten realizar pronósticos de las condiciones atmosféricas, estudios de contaminación ambiental y predecir los resultados de reacciones químicas. También se utilizan para simular situaciones reales como en los simuladores de vuelo, que proporcionan a los pilotos de aviones un ambiente de práctica en lugar de utilizar un avión real, pero presentando las condiciones similares a las que se tendrían en la realidad. Otra de las aplicaciones que ha tomado mucho auge últimamente es la realidad virtual, en la cual se modelan situaciones supuestas que permiten que el usuario las explore sin haberlas construido físicamente.

Cálculos científicos.

En estas aplicaciones, las computadoras digitales permiten resolver modelos matemáticos muy complejos tales como la distribución de las temperaturas o los esfuerzos mecánicos a los que estará sometido el transbordador espacial cuando reingresa a la atmósfera. También los cálculos de la trayectoria de las sondas espaciales se realiza con la ayuda de estas máquinas. Sin el apoyo de las computadoras digitales, la solución de tales modelo matemáticos no sería posible.

Comunicaciones.

El auge que han tenido las comunicaciones se debe principalmente a la incorporación de las computadoras digitales en el área. Estas aplicaciones incluyen desde los conmutadores telefónicos hasta las redes computacionales mundiales como la INTERNET. Dentro de este vasto campo está el procesamiento de señales de audio y video de la televisión comercial distribuida mediante satélites geoestacionarios y la codificación de las señales de comunicación para darles seguridad.

Sistemas de seguridad.

Los sistemas de seguridad modernos tienen como parte central una computadora digital conectada a sensores y actuadores para activar alarmas, permitir el acceso a edificios y bóvedas mediante códigos de identificación y activar sistemas contra incendio entre otras funciones.

Otras aplicaciones.

Los videojuegos, las calculadoras electrónicas, las agendas electrónicas, los diccionarios electrónicos, las grabadoras y reproductoras de discos compactos de audio y vídeo, las enciclopedias electrónicas, el correo electrónico y los juegos computacionales son ejemplos de aplicaciones adicionales de las computadoras digitales.

1.3. TIPOS DE COMPUTADORAS.

De acuerdo con su capacidad, las computadoras digitales se han catalogado tradicionalmente como supercomputadoras, equipos principales (MAIN FRAMES), minicomputadoras y microcomputadoras.

Los equipos principales incluyen computadoras digitales con gran capacidad tanto de procesamiento como memoria principal y almacenamiento secundario. Estos sistemas computacionales son costosos y generalmente son utilizados como sistemas centrales corporativos.

Las minicomputadoras son equipos que tienen capacidad de procesamiento y almacenamiento en un rango intermedio. Originalmente, el término se usaba para describir computadoras digitales cuya unidad central de proceso, que es la que ejecuta las instrucciones y controla la operación de

la máquina, estaba contenida en una sola tarjeta de circuito impreso. Su costo es del orden de 10 veces menor que el de los equipos principales. Su aplicación es similar a los equipos centrales para empresas pequeñas o medianas.

Las microcomputadoras, en su definición original, son computadoras digitales con la característica de que toda la unidad central de proceso está contenida en un solo circuito integrado. El costo de estos equipos es del orden de 100 veces menor que el de un equipo principal pero su capacidad de cómputo y almacenamiento es menor que el de una minicomputadora.

Las supercomputadoras son las que tienen la mayor capacidad de cómputo y almacenamiento y su costo es del orden de 10 veces el de un equipo principal. Estas máquinas generalmente se utilizan en centros de investigación, algunas universidades y empresas que tienen necesidad de procesar grandes volúmenes de información. Las aplicaciones a que se dedican son cálculos científicos, diseño y simulación principalmente.

En la actualidad, las definiciones anteriores ya no son tan claras, principalmente entre minicomputadoras y microcomputadoras, dado que los avances en electrónica de estado sólido han permitido construir computadoras cuyo procesador está en un solo circuito integrado, pero que tienen una capacidad de cómputo similar a la que tenían los equipos principales de hace 10 o 15 años.

Actualmente tiene mayor sentido pensar en términos de otra clasificación, que incluye computadoras personales, estaciones de trabajo, equipos principales y supercomputadoras.

Las computadoras personales son microcomputadoras de bajo costo que incluyen computadoras de escritorio y computadoras portátiles. Tanto su capacidad de cómputo como de almacenamiento continúan creciendo a medida que avanza la tecnología, sin embargo, son las que tienen la menor capacidad de cómputo. Las principales aplicaciones que se usan en estos equipos son procesamiento de textos, hojas de cálculo, diseño de presentaciones, agendas personales, correo electrónico, pequeñas bases de datos, interconexión a redes de cómputo y aplicaciones de tipo personal

Las estaciones de trabajo son computadoras cuyo costo es del orden de 3 a 10 veces el de una computadora personal. Tienen mayor capacidad de cómputo y almacenamiento lo cual les permite realizar tareas más complejas. Estos equipos generalmente se enfocan a aplicaciones industriales o científicas y en la mayoría de las veces están conectados a una computadora principal. En las estaciones de trabajo se visualizan los resultados de las aplicaciones que se ejecutan en un equipo principal y que requieren gran capacidad de cómputo, tales como sistemas de diseño asistido por computadora. Una característica importante de las estaciones de trabajo es que cuentan con pantallas de video de alta resolución.

Actualmente, la definición de equipo principal y supercomputadora siguen siendo válidos, si bien es cierto que estas máquinas aumentan su capacidad de cómputo y almacenamiento a pasos agigantados.

Dentro de las supercomputadoras también existen dos tipos principales: las computadoras vectoriales y los multiprocesadores. Las computadoras vectoriales cuentan con procesadores vectoriales, los cuales pueden realizar una misma operación sobre un conjunto de datos en forma simultánea. Por otro lado, los multiprocesadores son computadoras que tienen varios procesadores independientes entre los cuales se distribuyen las operaciones que tienen que realizarse para resolver un mismo problema.

Finalmente, existen cierto tipo de computadoras de propósito especial, tales como las utilizadas en aplicaciones de control de dispositivos específicos y videojuegos.

1.4 RESUMEN

En este capítulo se mencionaron las principales características de las computadoras digitales, las cuales la han convertido en una poderosa herramienta de propósito general.

Se describieron brevemente las principales aplicaciones de las computadoras digitales, sin embargo, día a día estas máquinas se utilizan para realizar tareas nuevas y su campo de aplicación es sumamente amplio.

Se esbozó la clasificación de las computadoras digitales de acuerdo a su capacidad y construcción y se mencionaron los campos de aplicación de cada uno de los tipos de computadoras digitales.

Finalmente, se presentó un panorama general del texto, describiendo someramente las cuatro secciones de que consta.

1.5. PROBLEMAS

1. Mencione las principales características de las computadoras digitales y descríbalas brevemente.
2. Explique que se entiende por dispositivos periféricos.
3. ¿Qué se almacena en la memoria de las computadoras digitales?
4. Describa las diferentes clases de computadoras digitales de acuerdo a su capacidad, su costo y su construcción.
5. ¿Con cuál sistema de números trabajan internamente las computadoras digitales?

CAPITULO 2

ORGANIZACION DE LA COMPUTADORA DIGITAL

2.1. ARQUITECTURA BASICA

Una computadora digital, en su forma más simple, consta de una unidad central de proceso, una unidad de memoria y una unidad de entrada/salida. Estas unidades están interconectadas entre si mediante un conjunto de líneas de comunicación que recibe el nombre de BUS. La unidad central de proceso (CPU) internamente contiene la unidad de control (CU), la unidad aritmética lógica (ALU) y varios registros. Esta arquitectura básica se muestra en forma esquemática en la figura 2.1.

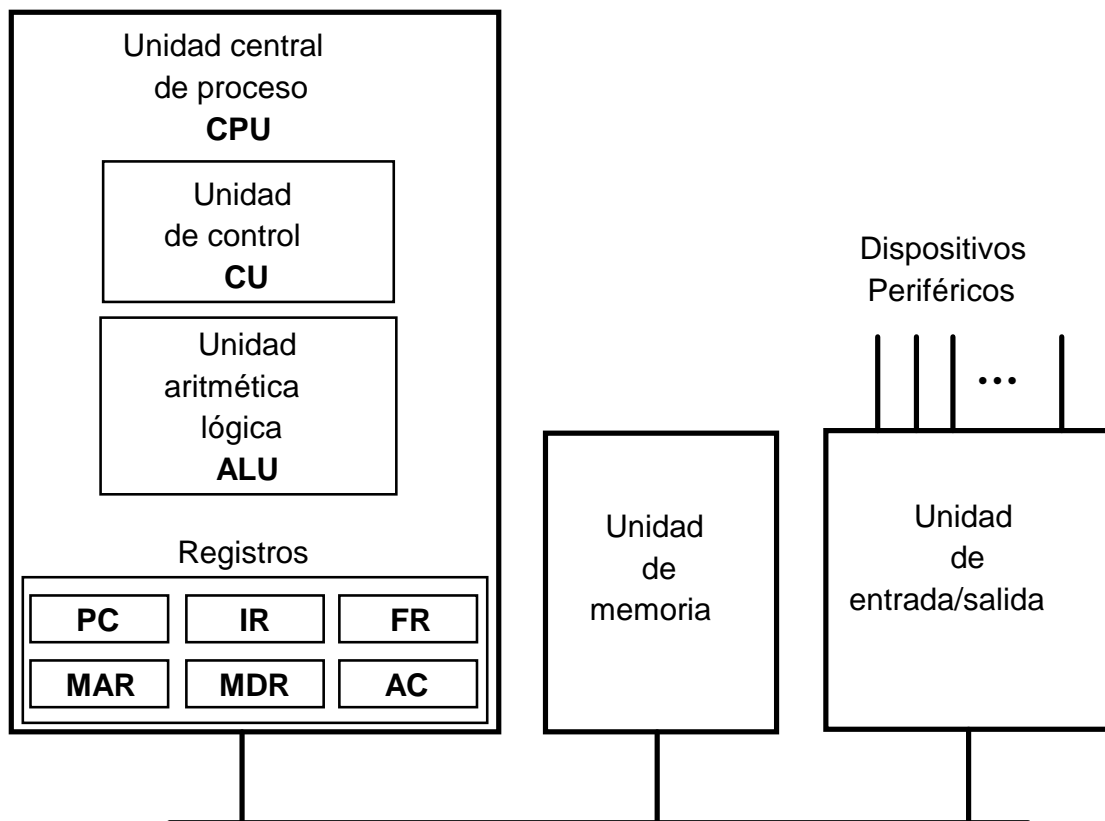


Figura 2.1 Arquitectura básica

Dentro del bus existen líneas para dirección, datos y señales de control. El número y la función de cada una de estas líneas se describirán posteriormente.

2.1.1. Unidad de memoria

La unidad de memoria está organizada como un conjunto de celdas, conocidas también como palabras, cada una de las cuales puede almacenar un dato o una instrucción. Las celdas de la memoria tienen asociada una dirección única. Las direcciones de memoria están asignadas en forma secuencial empezando con la dirección cero.

Se pueden realizar dos tipos de operaciones sobre la unidad de memoria. Estas operaciones son lectura de memoria y escritura a memoria. Para esto, la unidad de memoria tiene varias líneas mediante las cuales se conecta al bus. Un conjunto de líneas sirven para especificar la dirección de una celda, otro conjunto sirve para transferir el dato que se desea leer o escribir en esa celda y otra línea se usa para indicar el tipo de operación que se desea realizar. Existen además otras líneas que se utilizan para controlar y sincronizar la operación de la memoria con el resto de la computadora digital.

Una operación de lectura de la unidad de memoria permite obtener el dato o instrucción que se encuentra almacenado en una celda particular de la memoria. Cuando se desea leer un dato o una instrucción de una celda específica, las líneas de dirección deben tener la dirección de la celda de memoria cuyo contenido se desea obtener, la línea que indica el tipo de operación debe especificar una operación de lectura y las líneas de sincronización controlan el inicio de la operación. Cuando la unidad de memoria termina de realizar la operación de lectura, coloca en las líneas de transferencia de datos una copia del contenido de la celda de memoria cuya dirección se especificó mediante las líneas de dirección. El contenido de la celda de memoria no se modifica.

Una operación de escritura a la unidad de memoria permite almacenar un dato o una instrucción en una celda en particular de la unidad de memoria. El dato o instrucción que contiene la celda de memoria antes de realizar la operación de escritura será reemplazado con el dato o instrucción que se desea almacenar en dicha celda, perdiéndose el contenido original. Para realizar este tipo de operación sobre la memoria, las líneas de dirección deben tener la dirección de la celda en la cual se desea almacenar el dato o instrucción, las líneas de transferencia de datos deben tener el dato o instrucción que se desea guardar y la línea que indica el tipo de operación debe especificar que se trata de una operación de escritura. Cuando se inicia la operación, mediante las líneas de sincronización, la unidad de memoria almacena el dato o instrucción en la celda de memoria especificada.

2.1.2. Unidad central de proceso (CPU)

La función de la unidad central de proceso es ejecutar instrucciones para procesar datos y controlar toda la operación de la computadora digital. Para realizar las funciones de control, la unidad central de proceso cuenta precisamente con una unidad de control (**CU**) que es la que se encarga de coordinar la ejecución de las instrucciones y la transferencia de datos entre las diferentes unidades de la computadora.

Para realizar las operaciones aritméticas y lógicas, la unidad central de proceso contiene la unidad aritmética lógica (**ALU**). Para efectos de una arquitectura básica, las operaciones

aritméticas que se realizan son suma y resta de números enteros. Las operaciones lógicas se detallarán posteriormente al describir la operación de la computadora digital.

2.1.3. Registros

La unidad central de proceso también cuenta con una serie de registros utilizados para ciertos propósitos. Un registro es una celda similar a una de las celdas de memoria y puede almacenar un dato o una instrucción, dependiendo del registro de que se trate. Inicialmente se incluirán solamente seis registros en esta máquina y posteriormente se añadirán otros registros cuando se vaya haciendo más compleja su operación. A continuación se describe la función de cada uno de estos registros.

- **Registro de dirección de memoria (MAR).**

Este registro sirve para especificar la dirección de la celda de la memoria cuyo contenido se desea leer o escribir. Cuando se realiza una operación sobre la unidad de memoria, el contenido de este registro se conecta a las líneas de dirección del bus para indicar a la unidad de memoria la dirección de la celda involucrada en la operación.

- **Registro de datos de memoria (MDR).**

Este registro se utiliza para transferir datos entre la unidad de memoria y la unidad central de proceso. Cuando se desea guardar un dato en una celda de la memoria, este registro se conecta a las líneas de transferencia de datos del bus para que lo reciba la unidad de memoria. Si se desea obtener un dato de una celda de la memoria, se conecta este registro a las líneas de transferencia de datos del bus para recibir el dato que coloca la unidad de memoria en estas líneas y así, almacenar el dato en este registro.

La unidad de control, mediante las señales apropiadas, es la encargada de hacer que el dato que está en el registro de datos de memoria se envíe a través de las líneas de transferencia de datos del bus en una operación de escritura a memoria, o bien, se guarde el dato que está en estas líneas en el registro de datos de memoria en el caso de una operación de lectura de memoria.

- **Registro contador de programa (PC).**

Este registro contiene la dirección de la celda de memoria que tiene almacenada la siguiente instrucción a ejecutar. Cuando la unidad central de proceso necesita obtener la siguiente instrucción, transfiere el contenido de este registro al registro de dirección de memoria e inicia una operación de lectura de memoria. Cuando se termina la lectura y se coloca el contenido de la celda especificada en el registro de datos de memoria, se transfiere el contenido de dicho registro al registro de instrucción y se almacena la instrucción en él. Finalmente, se incrementa el contenido del registro contador de programa en una unidad para que ahora contenga la dirección de la siguiente instrucción a ejecutar.

- **Registro de instrucción (IR).**

Este registro contiene la instrucción que actualmente está ejecutando la unidad central de proceso. La unidad de control, dependiendo de la instrucción que está en este registro, activa las líneas de control necesarias en la secuencia adecuada para su ejecución.

- **Acumulador (AC).**

El acumulador es un registro en el cual se almacenan datos en forma temporal. La unidad aritmética lógica utiliza este registro en la ejecución de las operaciones aritméticas y lógicas. Por ejemplo, cuando en la computadora se realiza una operación de suma, la unidad aritmética lógica toma los datos que contienen el registro de datos de memoria y el acumulador, los suma, dejando el resultado de la operación en el acumulador. El dato que contenía originalmente el acumulador se pierde. También se pueden hacer transferencias de datos entre el acumulador y el registro de datos de memoria. Suponiendo que el resultado de una suma se quisiera guardar en memoria para utilizarse posteriormente, sería necesario transferir este resultado, que está en el acumulador, al registro de datos de memoria y ordenar una operación de escritura a la unidad de memoria.

- **Registro de banderas (FR).**

Las banderas son indicadores que solamente pueden tener dos valores: encendido (verdadero) y apagado (falso). Estos indicadores se modifican de acuerdo al resultado de cualquier operación aritmética o lógica. Para esta computadora simple, se definirán inicialmente las siguientes tres banderas:

- N** - indica que el valor almacenado en el acumulador es negativo.
- Z** - indica que el valor almacenado en el acumulador es cero.
- V** - indica que el resultado de la operación aritmética realizada excede la capacidad del registro acumulador.

Por ejemplo, si se realizara una operación de resta en la unidad aritmética lógica y su resultado fuera cero, entonces se encendería la bandera **Z** en este registro y se apagarían las banderas **N** y **V** si estaban encendidas. Si el resultado de otra operación de resta ejecutada inmediatamente después de la anterior no fuera cero, sino negativo, entonces se encendería la bandera **N** y se apagaría la bandera **Z**. Cualquier operación que modifique el valor almacenado en el registro acumulador, ocasiona que se enciendan o apaguen las banderas **Z** y **N** de acuerdo con el nuevo valor almacenado en ese registro.

El objetivo de tener estas banderas es permitir que la unidad de control pueda tomar acciones diferentes dependiendo del resultado de las operaciones.

2.1.4. Unidad de entrada/salida.

Para que un usuario de una computadora digital pueda ver y analizar los resultados de un programa, es necesario que se envíen a algún dispositivo periférico como una pantalla de vídeo o una impresora. Cada dispositivo periférico debe tener asignada una dirección única para poder identificarlo.

La forma en que estos datos serían enviados al dispositivo es similar a la forma en que se guardarían en la unidad de memoria. Para transferir un dato a un dispositivo periférico, se colocaría en el registro de datos de memoria la dirección del dispositivo, el dato que se desea enviar en el registro de datos de memoria y se indicaría una operación de escritura a la unidad de entrada/salida. En forma similar, si se desea obtener un dato de un dispositivo periférico, se ordenaría una operación de lectura a la unidad de entrada/salida. Dentro del bus existen líneas que indican si la operación de lectura o escritura debe ser realizada por la unidad de memoria o por la unidad de entrada/salida.

2.2. OPERACION DE LA COMPUTADORA DIGITAL.

Todas las computadoras digitales utilizan el sistema binario internamente para su operación en lugar del sistema decimal que es el que manejamos en la vida cotidiana. En el sistema decimal utilizamos los símbolos 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9 para representar números y cantidades, en el sistema binario solamente se utilizan los símbolos 0 y 1, que en computación reciben el nombre de bits (por abreviación de binary digit). Por ejemplo, la cantidad decimal 10 se representaría en binario como 1010. Para efectos de la operación básica de esta computadora digital, se usará el sistema decimal para facilitar la comprensión de su funcionamiento. En capítulos posteriores se introducirá el sistema binario y se estudiará la operación usando este sistema.

Las celdas de la unidad de memoria de una computadora tienen capacidad para un número fijo de dígitos binarios o bits. Esto hace que se tenga un límite en cuanto a los valores de los datos que se puedan representar en la máquina, similarmente a como estarían limitados los números decimales que pueden ser representados usando solamente 5 dígitos como máximo.

Los registros también tienen un número fijo de bits pero éste depende de la función de cada registro y, por lo tanto, no todos tienen el mismo número de bits.

Para presentar la operación de esta computadora digital usando el sistema decimal en lugar del binario, se supondrá que solamente pueden ser representados números enteros con signo usando 5 dígitos como máximo. Por consiguiente, tanto las celdas de memoria como el acumulador y el registro de datos de memoria deberán tener seis posiciones, asignando la primera al signo del número y las cinco restantes para la magnitud del número entero.

La computadora que se usa en este capítulo tiene una memoria con capacidad para 1000 datos o instrucciones; por consiguiente, las direcciones de memoria estarán numeradas del 000 al 999 y el registro de dirección de memoria así como el registro contador de programa tendrán 3 posiciones para contener números enteros en el rango de 000 a 999, ya que las direcciones no pueden ser negativas.

Las instrucciones también son almacenadas en la unidad de memoria de la computadora y estarán codificadas en seis posiciones, al igual que los datos. Esto indica que el registro de instrucción deberá tener también las mismas seis posiciones.

El registro de banderas que se utilizará en esta arquitectura tiene solamente las banderas **N**, **Z** y **V**. El significado de cada una es el descrito anteriormente.

En resumen, los registros **MAR**, **PC** y **FR** tendrán 3 posiciones y los registros **MDR**, **IR** y **AC**, 6 posiciones.

2.2.1. Juego básico de instrucciones.

El juego de instrucciones de esta computadora incluye las siguientes operaciones:

- **Inicializar el acumulador a cero (CLA).**
Esta instrucción permite poner el registro acumulador en cero. Al ejecutarse esta instrucción, se enciende la bandera **Z** y se apaga la bandera **N**.
- **Cambiar el signo al dato en el acumulador (NEG).**
La finalidad de esta instrucción es cambiarle el signo al dato que se encuentra en el registro acumulador. Al ejecutarse esta instrucción se modificarían las banderas **N** y **Z** de acuerdo con el resultado de la operación.
- **Cargar un dato en el acumulador (LDA).**
Mediante esta instrucción se puede poner un dato en el registro acumulador. El dato que estaba anteriormente en el registro se pierde. Esta instrucción cambia el valor contenido en el registro acumulador, por lo tanto, las banderas **N** y **Z** se modificarán de acuerdo al nuevo valor contenido en este registro.
- **Guardar el contenido del acumulador (STA).**
Esta instrucción se utiliza para guardar en la unidad de memoria el dato que actualmente está en el registro acumulador. El contenido del registro acumulador no se modifica al realizar esta instrucción.
- **Sumar un dato al contenido del acumulador (ADD).**
Esta instrucción permite sumar un dato al que actualmente se encuentra en el registro acumulador, el resultado de la operación se almacena en el acumulador. El dato que estaba anteriormente en el registro acumulador se pierde. Las banderas **N**, **Z** y **V** se modifican de acuerdo al resultado de la operación.
- **Restar un dato al contenido del acumulador (SUB).**
La instrucción de resta opera en forma similar a la de suma, permitiendo restarle un dato al contenido del acumulador y dejar el resultado en el acumulador. El dato que contiene el acumulador antes de la operación también se pierde. Esta instrucción modifica las banderas **N**, **Z** y **V** de acuerdo al resultado de la operación.
- **Transferir el control incondicionalmente (JMP).**
Cuando la computadora ejecuta un programa, toma las instrucciones en forma secuencial de la unidad de memoria. Recuérdese que el registro contador de programa (**PC**) contiene la

dirección en la cual se encuentra la siguiente instrucción a ejecutar en la unidad de memoria. El valor de este registro se incrementa en una unidad cada vez que se ejecuta una instrucción. Si fuere necesario que la siguiente instrucción no sea la que está en secuencia, se utilizaría esta instrucción para continuar con la ejecución de instrucciones, modificando el valor almacenado en el registro contador de programa, lo cual hará que se continúe tomando las instrucciones de otra dirección de memoria.

- **Transferir el control si la bandera Z está encendida (JMZ).**
Si la bandera **Z** estuviera encendida, debido al resultado de la última operación que la modificó, se transferiría el control de ejecución de instrucciones a la dirección de memoria indicada, es decir, la siguiente instrucción que ejecutaría la máquina sería la instrucción que se encuentra en la dirección de memoria indicada por la instrucción. Si por el contrario, la bandera **Z** no estuviere encendida, la ejecución de instrucciones continuaría de manera secuencial, como normalmente ocurre.
- **Transferir el control si la bandera N está encendida (JMN).**
La operación de esta instrucción es similar a la anterior, solamente que ahora se prueba la bandera **N** para tomar la decisión de transferir el control o continuar la ejecución en forma secuencial.
- **Transferir el control si la bandera V está encendida (JMV).**
Esta instrucción prueba la bandera **V** y, si está encendida, transfiere el control de la ejecución del programa a la dirección especificada por la instrucción. Si la bandera **V** no estuviere encendida, la ejecución continuaría en forma normal.
- **Detener la ejecución (HLT).**
Al ejecutarse esta instrucción se detendrá la operación de la computadora.
- **No operación (NOP).**
Esta instrucción no realiza ninguna operación, la computadora simplemente continúa con la instrucción que está en secuencia.

2.2.2. Formatos de instrucciones y datos.

Anteriormente se mencionó que los datos con que trabajará esta computadora simple son números enteros con signo. El formato para los datos tiene una posición para el signo y cinco posiciones para el número. Por convención, las posiciones se numerarán de izquierda a derecha, empezando con la posición cero. Este formato se muestra en la figura 2.2:

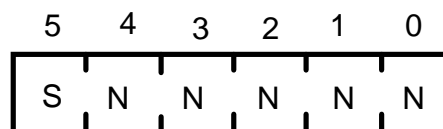


Figura 2.2. Formato para los datos

Las instrucciones constan de tres campos básicos que son:

C OP - Código de operación
TD - Tipo de direccionamiento
DIR - Dirección

La función de cada uno de estos campos se describirá posteriormente. El formato general de las instrucciones es:

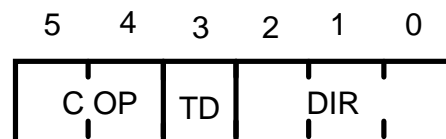


Figura 2.3. Formato para las instrucciones

Dentro de la computadora todas las instrucciones están codificadas en forma numérica y existe un código asignado a cada instrucción denominado código de operación. Las posiciones 4 y 5 contienen este código de operación para las instrucciones.

Las instrucciones **CLA** y **NEG** solamente necesitan el código de operación ya que operan solamente sobre el registro acumulador. Para estas instrucciones, las posiciones 0 a 3 no se utilizan y su contenido no es relevante. Las instrucciones **HLT** y **NOP** también utilizan solamente el código de operación.

Las demás instrucciones hacen referencia a un dato o una dirección de memoria y, por lo tanto, las posiciones 0 a 3 si son usadas para ejecutar la instrucción. A continuación se especifica el significado de las posiciones 0 a 3.

2.2.3. Tipos de direccionamiento.

Considérese la instrucción **LDA**, la cual permite cargar un dato en el registro acumulador. Una pregunta que surge al examinar esta instrucción es la siguiente: ¿Dónde está el dato que será cargado al registro acumulador?

Las posiciones 0 a 3 de la instrucción indican de dónde se obtendrá el dato que se almacenará en el registro acumulador de acuerdo al tipo de direccionamiento, especificado por la posición 3 de la instrucción.

Las computadoras digitales pueden tener varios tipos básicos de direccionamiento e incluso pueden tener combinaciones de tipos de direccionamiento. En esta computadora simple se incluirán inicialmente los siguientes tipos:

- Direccionamiento inmediato
 - Direccionamiento relativo
 - Direccionamiento absoluto
 - Direccionamiento indirecto
- **Direccionamiento inmediato.**
El direccionamiento inmediato se indicará con un cero en la posición 3, que es la que indica el tipo de direccionamiento. En este caso, el dato que se necesita para la instrucción se encuentra en la posiciones 0 a 2 de la misma instrucción, donde la posición 2 contiene el signo del dato y las posiciones 0 y 1 su magnitud. Con este tipo de direccionamiento solamente se pueden tener datos en el rango de -99 a +99. Las instrucciones que incluyen este tipo de direccionamiento son solamente **LDA, ADD y SUB**.
 - **Direccionamiento relativo.**
Este tipo de direccionamiento será indicado con un 1 en la posición 3 de la propia instrucción. El dato a que hace referencia la instrucción se encuentra en la unidad de memoria y su dirección se obtiene sumando el valor que se encuentra en las posiciones 0 a 2 de la instrucción al valor que contenga el registro contador de programa, que debe tener la dirección de la siguiente instrucción a ejecutar. El valor que está en las posiciones 0 a 2 es un número entero con signo en el rango de -99 a +99 en este caso. Las instrucciones que permiten este tipo de direccionamiento son **LDA, STA, ADD, SUB, JMP, JMZ, JMN y JMV**.
 - **Direccionamiento absoluto.**
Si el valor de la posición 3 de la instrucción fuere 2, se tendría este tipo de direccionamiento. En este caso, las posiciones 0 a 2 de la misma instrucción contienen un número entero sin signo en el rango 000 a 999. Este número representa la dirección de la celda en la unidad de memoria que contiene el dato para realizar la operación. Las instrucciones con las cuales se puede utilizar este tipo de direccionamiento son **LDA, STA, ADD, SUB, JMP, JMZ, JMN y JMV**.
 - **Direccionamiento indirecto.**
Para indicar este tipo de direccionamiento, la posición 3 de la instrucción tendría el valor 3. En las posiciones 0 a 2 de la misma instrucción se tiene un número entero sin signo en el rango de 000 a 999. Este número representa la dirección de la celda de la unidad de memoria cuyo contenido es la dirección de la celda que contiene el dato para realizar la operación. La computadora tiene que realizar una lectura de memoria para obtener la dirección donde se encuentra el dato. Una vez obtenida la dirección, es necesario realizar otra lectura a memoria para finalmente obtener el dato. Este tipo de direccionamiento es permitido en las instrucciones **LDA, STA, ADD, SUB, JMP, JMZ, JMN y JMV**.

2.2.4. Ejecución de las instrucciones.

Las instrucciones que se describieron anteriormente se conocen como instrucciones de máquina. La ejecución de estas instrucciones se realiza en base a micro operaciones. En esta sección

primero se describirán las micro operaciones que puede realizar la computadora básica que se está utilizando y posteriormente se definirá la ejecución de las instrucciones de máquina en base a estas micro operaciones.

Primeramente se definirán ciertas convenciones usadas en las micro operaciones:

Cuando se indica el nombre de un registro entre paréntesis rectangulares, significa que se hace referencia al contenido de dicho registro; por ejemplo, $[AC]$ indica el contenido del registro acumulador.

Si se especifica el nombre de un registro entre paréntesis rectangulares y un subíndice, se hace referencia al dato contenido en el registro y que está en la posición indicada por el subíndice. Por ejemplo, $[MDR]_3$ indica el número contenido en la posición 3 del registro **MDR**.

Si se especifica un rango como subíndice, se hace referencia al dato formado por el contenido de las posiciones indicadas en el rango. Por ejemplo, $[IR]_{2-0}$ indica que se hace referencia al dato contenido en las posiciones 0,1 y 2 del registro **IR**.

Las banderas **N**, **Z** y **V** solamente pueden tener los valores cero (apagada), o uno (encendida). Para referirse a estas banderas, solamente se usará la letra correspondiente, sin paréntesis cuadrados.

Cuando se indican varias micro operaciones encerradas entre corchetes, significa que se realizan simultáneamente.

Las micro operaciones definidas en esta computadora digital son las siguientes:

- Lectura de memoria: MMRead
- Escritura a memoria: MMWrite
- Inicializar el acumulador en cero: $[AC] \leftarrow +00000$
- Cambiar el signo del valor del acumulador: $[AC] \leftarrow -[AC]$
- Cargar un dato inmediato al acumulador: $\left\{ \begin{array}{l} [AC]_5 \leftarrow [IR]_2 \\ [AC]_{4-2} \leftarrow 000 \\ [AC]_{1-0} \leftarrow [IR]_{1-0} \end{array} \right\}$
- Incrementar el valor del PC: $[PC] \leftarrow [PC] + 1$
- Transferir el contenido del MDR al IR: $[IR] \leftarrow [MDR]$
- Transferir el contenido del PC al MAR: $[MAR] \leftarrow [PC]$
- Transferir al MAR una dirección relativa: $[MAR] \leftarrow [PC] + [IR]_{2-0}$
- Transferir al MAR una dirección absoluta: $[MAR] \leftarrow [IR]_{2-0}$

- Transferir una dirección del MDR al MAR: $[MAR] \leftarrow [MDR]_{2-0}$
- Transferir una dirección absoluta del IR al PC: $[PC] \leftarrow [IR]_{2-0}$
- Transferir una dirección relativa al PC: $[PC] \leftarrow [PC] + [IR]_{2-0}$
- Transferir una dirección del MDR al PC: $[PC] \leftarrow [MDR]_{2-0}$
- Transferir el contenido del MDR al acumulador: $[AC] \leftarrow [MDR]$
- Transferir el contenido del AC al MDR: $[MDR] \leftarrow [AC]$
- Sumar un dato inmediato al acumulador: $[AC] \leftarrow [AC] + [IR]_{2-0}$
- Restar un dato inmediato al acumulador: $[AC] \leftarrow [AC] - [IR]_{2-0}$
- Sumar el MDR al acumulador: $[AC] \leftarrow [AC] + [MDR]$
- Restar el MDR al acumulador: $[AC] \leftarrow [AC] - [MDR]$

La ejecución de las instrucciones de máquina se realiza en dos ciclos: El ciclo de búsqueda de la instrucción (**FETCH**), y el ciclo de ejecución de la instrucción (**EXECUTE**).

El ciclo de búsqueda es idéntico para todas las instrucciones y su objetivo es traer la siguiente instrucción a ejecutar, de la unidad de memoria, al registro de instrucción (**IR**). La dirección de la siguiente instrucción a ejecutar se encuentra en el registro contador de programa (**PC**). La unidad de control es la encargada de que se realice la secuencia de micro operaciones adecuada.

El ciclo de búsqueda de instrucciones está formado por las micro operaciones mostradas a continuación:

$$\begin{aligned}
 [MAR] &\leftarrow [PC] \\
 \text{MMRead} \\
 [PC] &\leftarrow [PC] + 1 \\
 [IR] &\leftarrow [MDR]
 \end{aligned}$$

El ciclo de ejecución es diferente para cada instrucción, ya que cada una realiza una operación distinta. De nuevo, la unidad de control, en base al código de operación y el tipo de direccionamiento, coordina la ejecución de las micro operaciones necesarias, en la secuencia adecuada, para ejecutar cada instrucción. A continuación se describe el ciclo de ejecución de las instrucciones de máquina en base a las micro operaciones.

- Instrucción **CLA**:
 $[AC] \leftarrow +00000$
- Instrucción **NEG**:
 $[AC] \leftarrow -[AC]$

• Instrucción **LDA**:

Con direccionamiento inmediato:

$$\left\{ \begin{array}{l} [AC]_5 \leftarrow [IR]_2 \\ [AC]_{4-2} \leftarrow 000 \\ [AC]_{1-0} \leftarrow [IR]_{1-0} \end{array} \right\}$$

Con direccionamiento relativo:

$$[MAR] \leftarrow [PC] + [IR]_{2-0}$$

MMRead

$$[AC] \leftarrow [MDR]$$

Con direccionamiento absoluto:

$$[MAR] \leftarrow [IR]_{2-0}$$

MMRead

$$[AC] \leftarrow [MDR]$$

Con direccionamiento indirecto:

$$[MAR] \leftarrow [IR]_{2-0}$$

MMRead

$$[MAR] \leftarrow [MDR]_{2-0}$$

MMRead

$$[AC] \leftarrow [MDR]$$

• Instrucción **STA**:

Con direccionamiento relativo:

$$[MAR] \leftarrow [PC] + [IR]_{2-0}$$

$$[MDR] \leftarrow [AC]$$

MMWrite

Con direccionamiento absoluto:

$$[MAR] \leftarrow [IR]_{2-0}$$

$$[MDR] \leftarrow [AC]$$

MMWrite

Con direccionamiento indirecto:

$$[MAR] \leftarrow [IR]_{2-0}$$

MMRead

$$[MAR] \leftarrow [MDR]_{2-0}$$

$$[MDR] \leftarrow [AC]$$

MMWrite

• Instrucción **ADD**:

Con direccionamiento inmediato:

$$[AC] \leftarrow [AC] + [IR]_2^{000} [IR]_{1-0}$$

Con direccionamiento relativo:

$$[MAR] \leftarrow [PC] + [IR]_{2-0}$$

MMRead

$$[AC] \leftarrow [AC] + [MDR]$$

Con direccionamiento absoluto:

$$[MAR] \leftarrow [IR]_{2-0}$$

MMRead

$$[AC] \leftarrow [AC] + [MDR]$$

Con direccionamiento indirecto:

$$[MAR] \leftarrow [IR]_{2-0}$$

MMRead

$$[MAR] \leftarrow [MDR]_{2-0}$$

MMRead

$$[AC] \leftarrow [AC] + [MDR]$$

• Instrucción **SUB**:

Con direccionamiento inmediato:

$$[AC] \leftarrow [AC] - [IR]_2^{000} [IR]_{1-0}$$

Con direccionamiento relativo:

$$[MAR] \leftarrow [PC] + [IR]_{2-0}$$

MMRead

$$[AC] \leftarrow [AC] - [MDR]$$

Con direccionamiento absoluto:

$$[MAR] \leftarrow [IR]_{2-0}$$

MMRead

$$[AC] \leftarrow [AC] - [MDR]$$

Con direccionamiento indirecto:

$$[MAR] \leftarrow [IR]_{2-0}$$

MMRead

$$[MAR] \leftarrow [MDR]_{2-0}$$

MMRead

$$[AC] \leftarrow [AC] - [MDR]$$

• Instrucción **JMP**:

Con direccionamiento relativo:

$$[PC] \leftarrow [PC] + [IR]_{2-0}$$

Con direccionamiento absoluto:

$$[PC] \leftarrow [IR]_{2-0}$$

Con direccionamiento indirecto:

$$[MAR] \leftarrow [IR]_{2-0}$$

MMRead

$$[PC] \leftarrow [MDR]_{2-0}$$

• Instrucción **JMZ**:

Con direccionamiento relativo:

$$\text{Si } Z = 1, \text{ entonces } [PC] \leftarrow [PC] + [IR]_{2-0}$$

Con direccionamiento absoluto:

$$\text{Si } Z = 1, \text{ entonces } [PC] \leftarrow [IR]_{2-0}$$

Con direccionamiento indirecto:

$$\text{Si } Z = 1, \text{ entonces } [MAR] \leftarrow [IR]_{2-0}$$

MMRead

$$[PC] \leftarrow [MDR]_{2-0}$$

• Instrucción **JMN**:

Con direccionamiento relativo:

$$\text{Si } N = 1, \text{ entonces } [PC] \leftarrow [PC] + [IR]_{2-0}$$

Con direccionamiento absoluto:

$$\text{Si } N = 1, \text{ entonces } [PC] \leftarrow [IR]_{2-0}$$

Con direccionamiento indirecto:

$$\text{Si } N = 1, \text{ entonces } [MAR] \leftarrow [IR]_{2-0}$$

$$\begin{array}{c} \text{MMRead} \\ [PC] \leftarrow [MDR]_{2-0} \end{array}$$

• Instrucción **JMV**:

Con direccionamiento relativo:

$$\text{Si } V = 1, \text{ entonces } [PC] \leftarrow [PC] + [IR]_{2-0}$$

Con direccionamiento absoluto:

$$\text{Si } V = 1, \text{ entonces } [PC] \leftarrow [IR]_{2-0}$$

Con direccionamiento indirecto:

$$\text{Si } V = 1, \text{ entonces } [MAR] \leftarrow [IR]_{2-0}$$

$$\begin{array}{c} \text{MMRead} \\ [PC] \leftarrow [MDR]_{2-0} \end{array}$$

• Instrucción **NOP**:

Esta instrucción no realiza ninguna micro operación.

2.2.5. Códigos de operación.

Las instrucciones dentro de la computadora están representadas por números y la operación que representa cada instrucción se puede saber analizando el código de operación y el tipo de direccionamiento, en caso de ser necesario.

Con dos posiciones para el código de operación se pueden tener hasta 100 instrucciones diferentes, desde el código 00 hasta el 99. En esta máquina hipotética no se utilizan todos los posibles códigos de operación, sino solamente los necesarios. Los códigos de operación correspondientes a cada una de las instrucciones de máquina descritas anteriormente, son los siguientes:

NOP	00
CLA	01
NEG	02
LDA	10
STA	11
ADD	20
SUB	21
JMP	30
JMZ	31
JMN	32
JMV	33
HLT	99

Posteriormente se añadirán otras instrucciones a medida que se vaya haciendo más compleja la operación de esta computadora y se asignarán otros códigos de operación.

2.3. EJEMPLOS DE PROGRAMAS.

A continuación se presentarán tres programas simples que ayudarán a comprender el funcionamiento de esta computadora digital.

Las instrucciones de máquina, dentro de la unidad de memoria de la computadora, están representadas por números. Programar usando los códigos de operación, los tipos de direccionamiento y las direcciones donde están los datos en la memoria es tedioso y, en cierta forma, complicado. Un programa que se desarrolla utilizando los códigos de operación, los tipos de direccionamiento y las direcciones reales en la unidad de memoria, está escrito en lenguaje maquina.

Para hacer que la programación a este nivel sea menos complicada, se ha desarrollado el lenguaje ensamblador. En este lenguaje se utilizan mnemónicos para cada código de operación ya que es más fácil recordar los mnemónicos **LDA**, **CLA**, **STA**, etc. que sus correspondientes códigos de operación. En forma similar, para cada tipo de direccionamiento también se utilizan letras en lugar de su codificación en lenguaje maquina. Respecto a las direcciones de memoria, se usan direcciones simbólicas representadas por nombres como **X**, **VARIABLE**, **NUMERO**, etc. La computadora no es capaz de ejecutar un programa en este lenguaje, sin embargo, para la persona que va a desarrollarlo, es más fácil escribirlo, probarlo y depurarlo en lenguaje ensamblador y posteriormente pasarlo al lenguaje maquina, que es el lenguaje que puede ejecutar la computadora. Cada instrucción en lenguaje ensamblador genera solamente una instrucción de lenguaje maquina.

Adicionalmente, en el lenguaje ensamblador se definen ciertas pseudoinstrucciones. Las pseudoinstrucciones no se traducen a una instrucción de máquina ya que no corresponden a éstas. Su uso permite controlar la generación del lenguaje maquina a partir del programa en lenguaje ensamblador. Por el momento solamente será necesario definir dos pseudoinstrucciones que son **ORG** y **DEF**.

La pseudoinstrucción **ORG** tiene como argumento una dirección de memoria y especifica que las instrucciones que están a continuación se deberán colocar a partir de dicha dirección de memoria.

La pseudoinstrucción **DEF** tiene como argumento un número y especifica que en el lugar de memoria correspondiente a la pseudoinstrucción deberá ponerse ese número. Esta pseudoinstrucción sirve para definir constantes en un programa en lenguaje ensamblador.

Las direcciones simbólicas usadas en el lenguaje ensamblador se denominan etiquetas. Una línea de programa en lenguaje ensamblador tiene tres componentes principales:

- El campo para etiquetas

- El campo para la instrucción y sus parámetros
- El campo para comentarios
- **Campo para etiquetas.**
Este es el primer campo de una línea de programa y puede usarse o dejarse en blanco. Cuando se utiliza, se asigna una dirección simbólica a la línea y, al momento de pasarse a lenguaje maquinal, el valor de la etiqueta será la dirección en memoria en la cual quedará la instrucción correspondiente a la línea de programa que tiene la etiqueta.
- **Campo para instrucciones y sus parámetros.**
En este campo se coloca el mnemónico para la instrucción, el tipo de direccionamiento y la dirección simbólica o numérica, separados por comas. En el caso de direccionamiento inmediato, se coloca el dato inmediato en el lugar de la dirección. Para las instrucciones que no tienen parámetros, solamente se coloca el mnemónico. Las pseudoinstrucciones con sus parámetros también se colocan en este campo.
- **Campo para comentarios.**
Este campo es el último de una línea de programa en lenguaje ensamblador y su contenido no se traduce a lenguaje maquinal. En este campo se colocarían, en forma opcional, los comentarios pertinentes para hacer el programa más entendible a una persona que lo analice.

En esta sección se presentarán los programas ejemplo tanto en lenguaje ensamblador como en lenguaje maquinal.

Los códigos de operación se definieron en la sección anterior. Los mnemónicos usados para los diferentes tipos de direccionamiento y su correspondiente representación en lenguaje maquinal, son los siguientes:

Inmediato	I	0
Relativo	R	1
Absoluto	A	2
Indirecto	D	3

Primeramente se presenta un programa que suma los números X, Y y Z y deja el resultado de la suma en W. Se supone que el valor de X está en la dirección de memoria 101, el de Y en la dirección 102 y el de Z en la dirección 103. El resultado W se dejará en la dirección de memoria 100. Se asignaron los valores de 250, 125 y -315 para X, Y y Z respectivamente. Para W se asignó el valor cero inicialmente.

La primera instrucción del programa estará en la dirección de memoria 000. Se supone que antes de iniciar la ejecución del programa, el registro contador de programa (**PC**) deberá inicializarse con la dirección 000.

ETIQUETA	INSTRUCCION	COMENTARIOS
	ORG, 000	LA SIG INSTR VA EN LA DIR 000

	LDA, A, X	CARGAR X EN EL AC
	ADD, A, Y	SUMARLE Y AL AC
	ADD, A, Z	SUMARLE Z AL AC
	STA, A, W	GUARDAR RESULTADO EN W
	HLT	DETENER LA EJECUCION
	ORG,100	LA SIG INST VA EN LA DIR 100
W	DEF, +0	DEFINIR EL VALOR +00000
X	DEF, +250	DEFINIR EL VALOR +00250
Y	DEF, +125	DEFINIR EL VALOR +00125
Z	DEF, -315	DEFINIR EL VALOR -00315

A continuación se muestra como quedaría el programa anterior al pasarlo a lenguaje maquina. En este caso se han suprimido los comentarios.

DIRECCION	CONTENIDO	ETIQUETA	INSTRUCCION
			ORG, 000
000	102101		LDA, A, X
001	202102		ADD, A, Y
002	202103		ADD, A, Z
003	112100		STA, A, W
004	990000		HLT
			ORG,100
100	+00000	W	DEF, +0
101	+00250	X	DEF, +250
102	+00125	Y	DEF, +125
103	-00315	Z	DEF, -315

Ahora se seguirá el programa anterior paso a paso para analizar como trabaja.

Inicialmente el contador de programa (**PC**) deberá contener la dirección 000. Al iniciar la ejecución se realiza el ciclo de búsqueda de la primera instrucción, que es la que está en la dirección 000 de la unidad de memoria. Esta instrucción se almacena en el registro de instrucción (**IR**) y el contenido del **PC** se incrementa en una unidad, conteniendo ahora la dirección 001. El registro **IR** contiene ahora la instrucción 102101. La unidad de control analiza el código de operación de la instrucción que es 10 y corresponde a la instrucción **LDA**. Ahora se analiza el tipo de direccionamiento que contiene un 2, lo cual indica direccionamiento absoluto y, por consiguiente, el dato que se deberá almacenar en el acumulador es el que está contenido en la dirección de memoria 101. Para ejecutar esta instrucción, la unidad de control ordena una transferencia del campo de dirección del registro **IR** (101) al registro **MAR**, seguida de una lectura de memoria. Como resultado de esta última micro operación, el registro **MDR** tendrá almacenado el valor +00250, que en lenguaje ensamblador representa el valor de X. Finalmente, la unidad de control ordena una transferencia del contenido del registro **MDR** al registro **AC**. En este momento ya se ha ejecutado la primera instrucción del programa y el registro **AC** contiene el valor de X, o sea, +00250.

La unidad de control inicia la búsqueda de la siguiente instrucción, cuya dirección en memoria es la 001. Esta dirección está en el registro **PC**. Al traer esta instrucción de memoria, mediante los pasos descritos anteriormente, el registro **IR** contendrá la instrucción 202102 y el registro **PC** tendrá almacenado el valor 002.

Para ejecutar esta instrucción, la unidad de control analiza el código de operación, que en este caso es un 20 y corresponde a la instrucción **ADD**. Inmediatamente después, la unidad de control analiza el tipo de direccionamiento que es un 2 y corresponde a un direccionamiento absoluto. La unidad de control transfiere la dirección absoluta que contiene la instrucción al registro **MAR**, que en este caso es la dirección 102. Se ordena una lectura de memoria y se almacena el valor que está en la dirección de memoria 102 en el registro **MDR**. Este valor es +00125 y corresponde al valor de Y. Para terminar con la ejecución de esta instrucción, se ordena sumar el valor contenido en el registro **MDR** al valor contenido en el registro **AC** y guardar el resultado de la suma en el registro **AC**. Ahora el registro **AC** contiene la suma de X y Y, lo cual es +00375.

La unidad de control inicia el ciclo de búsqueda de la siguiente instrucción, cuya dirección en la unidad de memoria está contenida en el registro **PC** y es la 002. Al terminar este ciclo, el registro **PC** tendrá almacenado la dirección 003 y el registro **IR** contendrá la instrucción 202103.

La unidad de memoria procede a ejecutar esta instrucción, analizando su código de operación, 20, que corresponde a la instrucción **ADD**. El tipo de direccionamiento es 2, correspondiente a un direccionamiento absoluto y la dirección del dato en memoria es la 103. Cuando termina la ejecución de esta instrucción, el registro acumulador contiene la suma de los valores +00375 y -00315, cuyo resultado es +00060.

La unidad de control procede ahora con la búsqueda de la siguiente instrucción, que está contenida en la celda de memoria cuya dirección está en el registro **PC**, que en este caso es la dirección 003. Al terminar el ciclo de búsqueda de la instrucción, el registro **PC** contendrá la dirección 004 y el registro **IR** contendrá la instrucción 112100.

Para ejecutar esta instrucción, la unidad de control decodifica el código de operación, que ahora es 11 y corresponde a la instrucción **STA**. El tipo de direccionamiento es 2, indicando un direccionamiento absoluto a la dirección 100. Al ejecutar la instrucción, la unidad de control ordena que se transfiera la dirección contenida en el campo de dirección del registro de instrucción, 100 en este caso, al registro **MAR**. Ordena que se transfiera el contenido del registro **AC** al registro **MDR** y, finalmente, se ordena una escritura a memoria. Esta última operación hace que se almacene el valor que contiene el registro **MDR**, que representa la suma de las tres cantidades X, Y y Z, en la dirección de memoria 100.

Se procede ahora con la búsqueda de la siguiente instrucción, cuya dirección en la unidad de memoria es la 004. Al terminar este ciclo, el registro **PC** contendrá la dirección 005 y el registro **IR** tendrá la instrucción 990000.

La unidad de control procede ahora con la ejecución de la instrucción que está contenida en el registro **IR**, analizando su código de operación. Este código es 99 y corresponde a la instrucción **HLT**, por consiguiente, la unidad de control detiene la operación, terminando el programa.

Al terminar la ejecución del programa anterior, la suma de las cantidades contenidas en las direcciones de memoria 101, 102 y 103 quedó almacenada en la dirección de memoria 100, la cual corresponde a W.

Este programa trabaja bien si se trata de sumar tres cantidades. Solo basta almacenar las cantidades en las direcciones de memoria 101, 102 y 103 y el resultado de la suma quedará almacenado en la dirección de memoria 100.

Una pregunta interesante es la siguiente: ¿Se puede diseñar un programa que realice la suma de N cantidades, donde N es un dato para el programa? La respuesta es sí. A continuación se presenta tal programa.

Antes de diseñar el programa es necesario definir dónde se van a almacenar ciertos valores en la unidad de memoria. El valor de N, que representa la cantidad de datos que se van a sumar, estará en la dirección 103. Los datos que serán sumados estarán a partir de la dirección de memoria 105, inclusive. En la dirección de memoria 104 se tendrá la dirección donde empiezan los datos que se van a sumar. Para este programa, en la dirección 104 se tendrá almacenado el valor 105.

En la dirección de memoria 100 se tendrá un número que servirá para contar el número de datos que se han sumado. A este número se le asignará inicialmente el valor de N y se irá decrementando cada vez que se sume un número hasta que su valor llegue a cero. Esto permitirá que se tenga control sobre la cantidad de valores que se llevan sumados. El valor contenido en la dirección de memoria 100 representará la cantidad de datos que faltan por ser sumados.

En la dirección de memoria 101 se almacenará la dirección del dato que sigue de ser sumado. Esta dirección almacenada en la celda de memoria 101 deberá irse incrementando para que contenga la dirección del siguiente dato a ser acumulado a la suma.

Por último, en la dirección de memoria 102 se almacenará el valor acumulado de la suma de los datos que se hayan sumado hasta el momento. Cuando se hayan sumado los N datos, en esta dirección se tendrá almacenado el valor de la suma de todos los datos.

Para el programa ejemplo se tomó el valor de N como 10 y los datos a ser sumados son 10, 20, 30, 40, 50, 60, 70, 80, 90 y -100. Si se cambian estos valores, se pueden sumar otros datos sin necesidad de modificar las instrucciones del programa.

ETIQUETA	INSTRUCCION	COMENTARIOS
	ORG, 000	LA SIG INST VA EN LA DIR 000
	LDA, A, N	CARGAR N EN EL AC
	STA, A, N1	GUARDAR EL VALOR EN N1
	LDA, A, DX	CARGAR EL CONT. DE DX EN EL AC
	STA, A, DX1	GUARDARLA EN DX1

	CLA	INICIALIZAR EL AC EN CERO
	STA, A, S	GUARDARLO EN S
CICLO	LDA, A, S	CARGAR EL VALOR DE S EN EL AC
	ADD, D, DX1	SUMARLE EL SIGUIENTE DATO
STA, A, S		GUARDAR LA SUMA ACUMULAA EN S
	LDA, A, N1	CARGAR EL VALOR DE N1
	SUB, I, 1	DECREMENTARLO EN 1
	STA, A, N1	GUARDARLO EN N1
	JMZ, A, FIN	SI RESULTA CERO IR A FIN
	LDA, A, DX1	CARGAR DX1
	ADD, I, +1	INCREMENTARLO EN 1
	STA, A, DX1	GUARDARLO EN DX1
	JMP, A, CICLO	REGRESAR A CICLO
FIN	HLT	DETENER LA EJECUCION
	ORG, 100	LA SIG INST VA EN LA DIR 100
N1	DEF, +0	INICIALIZAR N1 CON CERO
DX1	DEF, +0	INICIALIZAR DX1 CON CERO
S	DEF, +0	INICIALIZAR LA SUMA A CERO
N	DEF, +10	EL VALOR DE N ES 10
DX	DEF, X	LA DIR DEL PRIMER DATO ES X
X	DEF, +10	EL PRIMER DATO ES 10
	DEF, +20	EL SEGUNDO DATO ES 20
	DEF, +30	EL TERCER DATO ES 30
	DEF, +40	EL CUARTO DATO ES 40
	DEF, +50	EL QUINTO DATO ES 50
	DEF, +60	EL SEXTO DATO ES 60
	DEF, +70	EL SEPTIMO DATO ES 70
	DEF, +80	EL OCTAVO DATO ES 80
	DEF, +90	EL NOVENO DATO ES 90
	DEF, -100	EL DECIMO DATO ES -100

A continuación se muestra el programa anterior traducido al lenguaje maquina. De nuevo se han suprimido los comentarios.

DIRECCION	CONTENIDO	ETIQUETA	INSTRUCCION
			ORG, 000
000	102103		LDA, A, N
001	112100		STA, A, N1
002	102104		LDA, A, DX
003	112101		STA, A, DX1
004	010000		CLA
005	112102		STA, A, S
006	102102	CICLO	LDA, A, S
007	203101		ADD, D, DX1
008	112102		STA, A, S
009	102100		LDA, A, N1

010	210+01		SUB, I, 1
011	112100		STA, A, N1
012	312017		JMZ, A, FIN
013	102101		LDA, A, DX1
014	200+01		ADD, I, +1
015	112101		STA, A, DX1
016	302006		JMP, A, CICLO
017	990000	FIN	HLT
			ORG, 100
100	+00000	N1	DEF, +0
101	+00000	DX1	DEF, +0
102	+00000	S	DEF, +0
103	+00010	N	DEF, +10
104	+00105	DX	DEF, X
105	+00010	X	DEF, +10
106	+00020		DEF, +20
107	+00030		DEF, +30
108	+00040		DEF, +40
109	+00050		DEF, +50
110	+00060		DEF, +60
111	+00070		DEF, +70
112	+00080		DEF, +80
113	+00090		DEF, +90
114	-00100		DEF, -100

A continuación se analizará la operación del programa anterior, pero no se describirá a detalle la ejecución de cada instrucción como en el primer ejemplo. Se analizará a nivel de instrucción de máquina, mencionando los registros relevantes que cambian y las celdas de memoria que se modifican al ejecutar cada instrucción.

Inicialmente el registro contador de programa deberá contener la dirección 000 ya que la primera instrucción del programa se encuentra en dicha dirección.

La primera instrucción que se ejecuta es la que se encuentra en la dirección de memoria 000 y es 102103 (LDA, A, 103). Al terminar la ejecución de la instrucción, el registro PC contiene la dirección 001 y en el registro **AC** esta el valor de N, que es +00010.

Se ejecuta ahora la instrucción 112100 (STA, A, 100) que está en la dirección de memoria 001. Al final de su ejecución, el registro **PC** tiene la dirección 002 y el contenido de la celda de memoria cuya dirección es la 100 contendrá +00010, que es el valor de N. El registro **AC** no se modifica.

Continuando con el programa, se ejecuta la instrucción 102104 (LDA, A, 104), que está en la dirección de memoria 002. Esta instrucción carga en el registro **AC** el valor +00105, contenido en la celda de memoria 104 y que representa la dirección donde empiezan los datos a ser sumados. El registro **PC** tendrá la dirección 003.

La instrucción que está en la dirección de memoria 003 es 112101 (STA, A, 101), la cual almacena el contenido del registro **AC** en la dirección de memoria 101 e incrementa el valor el registro **PC** en 1, por lo que este registro contiene la dirección 004.

En la dirección de memoria 004 está la instrucción 010000 (CLA), la cual inicializa el registro **AC** con el valor de +00000 y el registro **PC** queda con la dirección 005.

La siguiente instrucción, que se encuentra en la dirección de memoria 005, es 112102 (STA, A, 102). Al terminar su ejecución, el contenido del registro **AC** se almacena en la dirección de memoria 102 y el registro **PC** tendrá la dirección 006. En la celda de memoria cuya dirección es la 102, se almacenará el valor acumulado de la suma. Inicialmente se pone la suma en cero.

El conjunto de instrucciones contenidas en las direcciones 006 a 016 inclusive, es el que realiza la suma de los N datos. Por lo tanto, este conjunto de instrucciones se ejecutará N veces.

La instrucción en la dirección 006 es 102102 (LDA, A, 102). Esta instrucción carga en el registro **AC** el contenido de la dirección de memoria 102, que contiene la suma acumulada hasta el momento. En esta ocasión, la suma es cero y el registro **AC** quedará con el valor +00000. El registro **PC** tendrá la dirección 007.

En la dirección de memoria 007 se encuentra la instrucción 203101 (ADD, D, 101). Al ejecutarse esta instrucción se suma al registro **AC** el dato cuya dirección se encuentra en la dirección de memoria 101, nótese que se está utilizando el direccionamiento indirecto. La dirección de memoria 101 contiene el valor +00105, que representa la dirección del primer dato a sumar. Por consiguiente, al registro **AC** se le sumará el valor contenido en la dirección de memoria 105 que es +00010. El registro acumulador contiene ahora el valor +00010, que corresponde a la suma acumulada hasta este momento, y el registro **PC** tiene la dirección 008.

Al ejecutar la instrucción en la dirección 008, que es 112102 (STA, A, 102), se guarda el valor de la suma acumulada hasta el momento, contenida en el registro **AC**, en la dirección de memoria 102. El valor del registro **AC** no se modifica y el registro **PC** tiene ahora 009.

La siguiente instrucción, contenida en la dirección de memoria 009, es 102100 (LDA, A, 100). Esta instrucción carga el contenido de la dirección de memoria 100 en el registro **AC**, quedando este registro con el valor +00010. El registro **PC** queda con la dirección 010. Recuérdese que el dato contenido en la dirección de memoria 100 es el que sirve para contar cuantos datos faltan por sumar.

Toca el turno ahora a la instrucción contenida en la dirección 010, que es 210+01 (SUB, I, 1). Esta instrucción le resta el valor +01 al registro **AC** en forma inmediata y el valor almacenado en este registro será +00009. Esto significa que ahora solamente falta sumar 9 datos. EL registro **PC** tiene la dirección 011.

La instrucción en la dirección 011 es 112100 (STA, A, 100). Al ejecutar esta instrucción se guarda el contenido del registro **AC** en la dirección de memoria 100, quedando almacenado el

valor +00009 en dicha dirección. El registro **PC** tendrá ahora la dirección 012 y el valor del registro **AC** no se modifica.

En la dirección de memoria 012 se encuentra la instrucción 312017 (JMZ, A, 017). Esta instrucción permite determinar si ya se han sumado todos los datos, realizando una transferencia de control de ejecución del programa a la dirección de memoria 017 si el resultado de la última operación aritmética fue cero. En este caso, el resultado fue +00009 y la transferencia no se realiza, sino que continúa la ejecución normal del programa, quedando el registro **PC** con la dirección 013 y el registro **AC** no se modifica.

Continuando la ejecución, la instrucción siguiente se encuentra en la dirección de memoria 013 y es 102101 (LDA, A, 101). Esta instrucción carga el contenido de la dirección de memoria 101, que es +00105, en el registro **AC**. El contenido del registro **PC** es 014.

La dirección 014 de memoria tiene la instrucción 200+01 (ADD, I, 1). Al ejecutarse la instrucción, se suma en forma inmediata el valor 1 al contenido del acumulador, quedando este registro con el valor +00106. El registro **PC** queda con la dirección 015.

La instrucción en la dirección 015 es 112101 (STA, A, 101). Esta instrucción guarda el contenido del acumulador, que es +00106, en la celda de memoria cuya dirección es la 101. El contenido del registro **AC** no se modifica y el registro **PC** queda con la dirección 016.

En la dirección de memoria 016 se encuentra la instrucción 302006 (JMP, A, 006). Esta instrucción transfiere el control de ejecución a la dirección de memoria 006 para ejecutar de nuevo el conjunto de instrucciones que realizan la suma de los datos. Cuando termina la ejecución de esta instrucción, el registro **PC** tiene la dirección 006 y el registro **AC** no se modifica. En este momento la celda de memoria cuya dirección es 100 contiene el valor +00009, lo cual significa que solo falta sumar 9 datos. La celda de memoria con dirección 101 contiene el valor +00106, que es la dirección en memoria del siguiente dato a sumar. El valor acumulado de la suma hasta el presente momento es +00010 ya que solo se ha sumado el primer dato. Este valor está en la dirección de memoria 102.

La siguiente vez que se ejecuta este grupo de instrucciones, las instrucciones contenidas de la dirección de memoria 006 a 016 inclusive, se acumula el valor del siguiente dato a la suma. La celda de memoria con dirección 100 tendrá ahora el valor de +00008, lo cual indica que solo faltan 8 datos por sumar. La celda de memoria cuya dirección es 101 tendrá el valor +00107, que es la dirección del siguiente dato a sumar. En la dirección de memoria 102 estará almacenado el valor +00030, que representa la suma acumulada hasta el momento, es decir, la suma de los primeros dos datos.

En la figura 2.4 se muestra una tabla con el contenido de estas tres celdas de memoria cuando el registro contador de programa tiene la dirección 006 y el grupo de instrucciones que realiza la suma se ha ejecutado *i* veces:

En este momento se han sumado 9 de los 10 datos, faltando solamente sumar el último. Se seguirá el programa para este último dato a partir de la instrucción en la dirección 006.

i	celda 100	celda 101	celda 102
0	+00010	+00105	+00000
1	+00009	+00106	+00010
2	+00008	+00107	+00030
3	+00007	+00108	+00060
4	+00006	+00109	+00100
5	+00005	+00110	+00150
6	+00004	+00111	+00210
7	+00003	+00112	+00280
8	+00002	+00113	+00360
9	+00001	+00114	+00450

Figura 2.4. Contenido de las posiciones de memoria

La instrucción en la dirección 006 es 102102 (LDA, A, 102). Esta instrucción carga en el registro **AC** el contenido de la dirección de memoria 102, que contiene la suma acumulada hasta el momento. En esta ocasión, el valor de la suma es +00450 y el registro **AC** quedará con este valor. El registro **PC** tendrá la dirección 007.

En la dirección de memoria 007 se encuentra la instrucción 2023101 (ADD, D, 101). EL ejecutarse esta instrucción se suma al registro **AC** el dato cuya dirección se encuentra en la dirección de memoria 101. La dirección de memoria 101 contiene el valor +00114, que representa la dirección del último dato a sumar. Por consiguiente, al registro **AC** se le sumará el valor contenido en la dirección de memoria 114 que es +00100. El registro acumulador contiene ahora el valor +00350, que representa la suma acumulada hasta este momento, y es la suma de los diez datos. El registro **PC** tiene la dirección 008.

Al ejecutar la instrucción en la dirección 008, que es 112102 (STA, A, 102), se guarda el valor de la suma acumulada, contenida en el registro **AC**, en la dirección de memoria 102. El valor del registro **AC** no se modifica y el registro **PC** tiene ahora 009. La dirección de memoria 102 tendrá almacenado el valor +00350.

La siguiente instrucción, contenida en la dirección de memoria 009, es 102100 (LDA, A, 100). Esta instrucción carga el contenido de la dirección de memoria 100 en el registro **AC**, quedando este registro con el valor +00001. El registro **PC** queda con la dirección 010. Recuérdese que el dato contenido en la dirección de memoria 100 es el que sirve para contar cuantos datos faltan por sumar.

La instrucción contenida en la dirección 010, que es 210+01 (SUB, I, 1), es ejecutada a continuación. Esta instrucción le resta el valor +01 al registro **AC** en forma inmediata y el valor almacenado en este registro será +00000. Esto significa que ahora ya no falta ningún dato por sumar. EL registro **PC** tiene la dirección 011.

La instrucción en la dirección 011 es 112100 (STA, A, 100). Al ejecutar esta instrucción se guarda el contenido del registro **AC** en la dirección de memoria 100, quedando almacenado el valor +00000 en dicha dirección. El registro **PC** tendrá ahora la dirección 012 y el valor del registro **AC** no se modifica.

En la dirección de memoria 012 se encuentra la instrucción 312017 (JMZ, A, 017). Esta instrucción sirve para determinar si ya se han sumado todos los datos, realizando una transferencia de control de la ejecución del programa a la dirección de memoria 017 si el resultado de la última operación aritmética fue cero. En este caso, el resultado efectivamente es cero y se transfiere control a la dirección de memoria 017. El registro **PC** tendrá ahora el valor 017 y el registro **AC** no se modifica.

Finalmente, toca el turno de ejecución a la instrucción contenida en la dirección de memoria 017, que es la instrucción 990000 (HLT). Al realizarse esta operación, se termina la ejecución del programa quedando el valor de la suma (+00350) en la dirección de memoria 102.

Con el análisis de este programa se deberá tener una buena idea de la operación de una computadora digital, sin considerar el uso de dispositivos periféricos. Se presentará un último programa en esta sección, pero ya no se realizará el análisis del mismo.

Ahora se diseñará un programa que determine el valor absoluto de una cantidad. Supóngase que la cantidad está en la dirección de memoria 010 y que el valor absoluto de esta cantidad se dejará en la dirección de memoria 011. EL programa inicia su ejecución en la dirección de memoria 000.

Este programa es el siguiente:

ETIQUETA	INSTRUCCION	COMENTARIOS
	ORG, 000	LA SIG INSTR VA EN LA DIR 000
	LDA, A, X	CARGAR EL DATO EN EL AC
	JMN, A, NEGAT	PROBAR EL RESULTADO
	JMP, A, SIG	NO ES NEGATIVO, IR A SIG
NEGAT	NEG	NEGATIVO, CAMBIARLE EL SIGNO
SIG	STA, A, RES	GUARDAR EL RESULTADO
	HLT	TERMINAR LA EJECUCION
	ORG, 010	LA SIG INSTR VA EN LA DIR 010
X	DEF, -125	DEFINIR EL VALOR -125
RES	DEF, 0	DEFINIR EL VALOR 0

Al traducir este programa a lenguaje maquinal, se tendría lo siguiente:

DIRECCION	CONTENIDO	ETIQUETA	INSTRUCCION
			ORG, 000
000	102010		LDA, A, X
001	322003		JMN, A, NEGAT

002	302004		JMP, A, SIG
003	020000	NEGAT	NEG
004	112011	SIG	STA, A, RES
005	990000		HLT
			ORG, 010
010	-00125	X	DEF, -125
011	+00250	RES	DEF, +0

La primera instrucción carga el dato cuyo valor absoluto se desea obtener en el registro acumulador. Esto ocasiona que la bandera **N** se encienda si el dato es negativo, o se apague si el dato no es negativo. Si el dato es negativo será necesario cambiarle el signo y almacenarlo en la dirección de memoria 011. Si no fuera negativo, simplemente se almacenaría en esa dirección.

Los programas anteriores pueden ejecutarse con el paquete de la computadora decimal que se encuentra en el diskette que acompaña al texto. Se sugiere al lector que los ejecute, cambiando los datos para comprobar el funcionamiento de estos tres programas.

2.4. LA PILA EN MEMORIA

La mayoría de los procesadores actuales incluyen instrucciones para el manejo de una pila en la unidad de memoria. Una pila es una estructura de datos en la cual, el primer dato que se obtiene de ella es el último dato que se colocó en la pila.

El concepto de una pila es realmente simple y puede ser entendido con un ejemplo sencillo. Suponga que en un escritorio se tiene una charola para poner documentos y que inicialmente está vacía. Si se coloca el documento A en la charola, luego el documento B, enseguida el documento C y finalmente el D; se tendrá una pila de documentos en la charola. Al retirar los documentos, el primero que se obtendría sería el documento D, luego el C, seguido del B y, por último, el documento A. Nótese que al retirar los documentos de la pila, se obtienen en el orden inverso en el que se colocaron.

Una estructura de datos con estas características puede ser manipulada en la unidad de memoria, utilizando un registro que contenga la dirección del siguiente lugar disponible en la pila. La unidad central de proceso tiene un registro denominado apuntador a la pila (**SP**), para estos fines. La forma en que se manipularía la pila es la siguiente:

- Para colocar un dato en la pila, se guardaría este dato en la celda de memoria cuya dirección está contenida en el registro apuntador a la pila. Luego se incrementaría el valor contenido en el registro apuntador a la pila en una unidad para que indique la siguiente dirección de memoria, que representa el siguiente lugar disponible en la pila.
- Para obtener un dato de la pila, primero se decrementaría el valor contenido en el registro apuntador a la pila en una unidad para que apunte a la dirección de memoria donde se colocó el último dato en la pila. El dato que se obtiene es el que está en la celda de memoria que

indica el registro apuntador a la pila. Al sacar este dato de la pila, el lugar queda disponible para guardar otro dato.

La dirección de memoria donde se coloque el inicio de la pila puede ser cualquier dirección, siendo responsabilidad del programador su manejo correcto.

Para ejemplificar lo anterior, supóngase que inicialmente el valor contenido en el apuntador a la pila es 100 y que la pila está vacía. Al colocar el dato +125 en la pila, se almacenaría el valor +125 en la celda de memoria cuya dirección es 100 y se incrementaría el valor contenido en el registro apuntador a la pila en una unidad, quedando almacenado el valor 101 en dicho registro.

Si ahora se pone el dato -315 en la pila, éste será depositado en la celda de memoria cuya dirección es 101 y el valor del registro apuntador a la pila será incrementado en una unidad, quedando con el valor 102.

En la figura 2.5 se muestra el estado de la pila y el valor contenido en el registro apuntador a la pila al inicio, después de poner al valor +125 y el estado final. El valor mostrado como XXXXXX puede ser cualquiera, pues no es relevante al proceso.

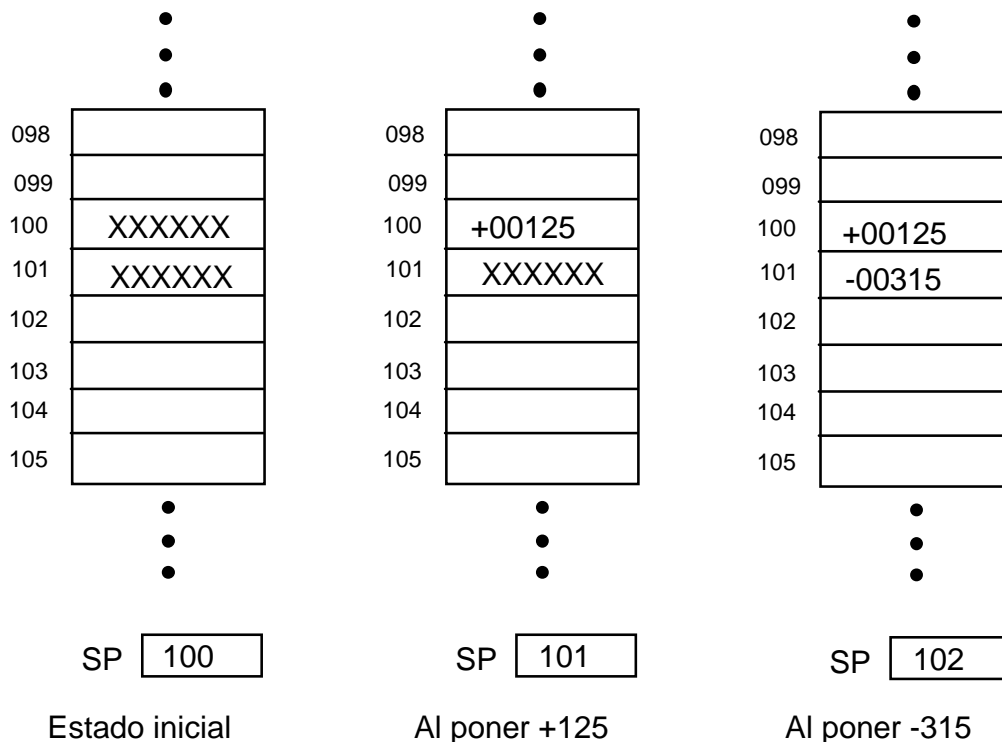


Figura 2.5. Estados de la pila

Suponiendo que ahora se deseara sacar un dato de la pila, primeramente se decrementaría el valor contenido en el registro apuntador a la pila, quedando éste con la dirección 101. El dato que se obtendría sería -315 y el estado de la pila sería el mismo que se muestra en la figura 2.5 al poner el dato +125 en la pila.

Si se sacara otro dato de la pila, se obtendría el valor +125. El estado de la pila y el registro apuntador a la pila sería el estado inicial de la figura 2.5.

Cuando se saca un dato de la pila, el valor del apuntador se modifica pero los datos que están en las celdas de memoria no se alteran. En este caso, en la dirección de memoria 100 continuaría existiendo el valor +125 y en la dirección de memoria 101 estaría el valor -315. Al introducir otro dato en la pila, se guardaría en la dirección de memoria 100 y en ese momento se perdería el valor +125 que estaba en dicha dirección.

Si se analiza la función que tiene el registro apuntador a la pila, se puede ver que su objetivo es guardar direcciones de memoria. El número de posiciones que debe tener este registro es el mismo número de posiciones que tiene el registro de dirección de memoria, es decir, tres posiciones.

Para manipular la pila en memoria es menester definir las instrucciones de máquina necesarias, además de añadir el registro apuntador a la pila. Estas nuevas instrucciones también requerirán de la definición de micro operaciones para su ejecución. Las micro operaciones necesarias son las siguientes:

- Transferir una dirección del IR al SP: $[SP] \leftarrow [IR]_{2-0}$
- Transferir una dirección del SP al MAR: $[MAR] \leftarrow [SP]$
- Transferir una dirección del SP al MDR: $\left\{ \begin{array}{l} [MDR]_{5-3} \leftarrow 000 \\ [MDR]_{2-0} \leftarrow [SP] \end{array} \right\}$
- Transferir una dirección del MDR al SP: $[SP] \leftarrow [MDR]_{2-0}$
- Incrementar el valor del registro SP: $[SP] \leftarrow [SP] + 1$
- Decrementar el valor del registro SP: $[SP] \leftarrow [SP] - 1$

Las instrucciones de máquina mínimas para poder manipular la pila en memoria son:

- **Cargar un dato en el registro apuntador a la pila (LDSP).**
Esta instrucción permite almacenar una dirección en el registro apuntador a la pila. La dirección que estaba anteriormente en este registro se pierde. Se permiten los mismos tipos de direccionamiento para esta instrucción que para la instrucción **LDA**, solamente que ahora el dato representa una dirección de memoria.
- **Guardar el contenido del apuntador a la pila (STSP).**
La función de esta instrucción es guardar en la unidad de memoria la dirección que tiene el registro apuntador a la pila. El contenido del registro no se modifica al realizar esta instrucción. La operación de esta instrucción es similar a la instrucción **STA** y permite los mismos tipos de direccionamiento.

- **Guardar el contenido del acumulador en la pila (PUSHA).**

Esta instrucción se utiliza para guardar, en la unidad de memoria, el dato que actualmente está en el registro acumulador. La dirección de la celda de memoria en la cual se depositará el contenido del acumulador es la dirección que tiene actualmente el registro apuntador a la pila (SP). El contenido del registro acumulador no se modifica al realizar esta instrucción, pero el valor que tiene el registro apuntador a la pila es incrementado en una unidad.

- **Cargar en el acumulador un dato de la pila (POPA).**

Esta instrucción sirve para cargar en el registro acumulador el dato que actualmente está en el tope de la pila, en la unidad de memoria. Al ejecutar esta operación, primeramente se decrementa el valor contenido en el registro apuntador a la pila en una unidad y luego se utiliza el nuevo valor como la dirección de la celda de memoria de la cual se obtiene el dato que se carga al acumulador.

El ciclo de ejecución de las instrucciones anteriores, en base a micro operaciones, se detalla a continuación:

- Instrucción **LDSP:**

Con direccionamiento inmediato:

$$[SP] \leftarrow [IR]_{2-0}$$

Con direccionamiento relativo:

$$[MAR] \leftarrow [PC] + [IR]_{2-0}$$

MMRead

$$[SP] \leftarrow [MDR]_{2-0}$$

Con direccionamiento absoluto:

$$[MAR] \leftarrow [IR]_{2-0}$$

MMRead

$$[SP] \leftarrow [MDR]_{2-0}$$

Con direccionamiento indirecto:

$$[MAR] \leftarrow [IR]_{2-0}$$

MMRead

$$[MAR] \leftarrow [MDR]_{2-0}$$

MMRead

$$[SP] \leftarrow [MDR]_{2-0}$$

- Instrucción **STSP:**

Con direccionamiento relativo:

$$[MAR] \leftarrow [PC] + [IR]_{2-0}$$

$$\left\{ \begin{array}{l} [\text{MDR}]_{5-3} \leftarrow 000 \\ [\text{MDR}]_{2-0} \leftarrow [\text{SP}] \end{array} \right\}$$

MMWrite

Con direccionamiento absoluto:

$$\begin{array}{l} [\text{MAR}] \leftarrow [\text{IR}]_{2-0} \\ \left\{ \begin{array}{l} [\text{MDR}]_{5-3} \leftarrow 000 \\ [\text{MDR}]_{2-0} \leftarrow [\text{SP}] \end{array} \right\} \end{array}$$

MMWrite

Con direccionamiento indirecto:

$$\begin{array}{l} [\text{MAR}] \leftarrow [\text{IR}]_{2-0} \\ \text{MMRead} \\ [\text{MAR}] \leftarrow [\text{MDR}]_{2-0} \\ \left\{ \begin{array}{l} [\text{MDR}]_{5-3} \leftarrow 000 \\ [\text{MDR}]_{2-0} \leftarrow [\text{SP}] \end{array} \right\} \end{array}$$

MMWrite

• Instrucción **PUSHA**:

$$\begin{array}{l} [\text{MAR}] \leftarrow [\text{SP}] \\ [\text{MDR}] \leftarrow [\text{AC}] \\ \text{MMWrite} \\ [\text{SP}] \leftarrow [\text{SP}] + 1 \end{array}$$

• Instrucción **POPA**:

$$\begin{array}{l} [\text{SP}] \leftarrow [\text{SP}] - 1 \\ [\text{MAR}] \leftarrow [\text{SP}] \\ \text{MMRead} \\ [\text{AC}] \leftarrow [\text{MDR}] \end{array}$$

Los códigos de operación asignados a las instrucciones anteriores son los siguientes:

LDSP	40
STSP	41
PUSHA	42
POPA	43

Las instrucciones **LDSP** y **STSP** utilizan también los campos para el tipo de direccionamiento y la dirección en el formato de instrucciones. Las instrucciones **PUSHA** y **POPA** solamente utilizan el campo para el código de operación ya que la dirección en memoria es la que indica el registro apuntador a la pila. En estas últimas instrucciones, los campos del formato que no se utilizan serán puestos en cero.

Revisando el ejemplo anterior sobre el manejo de la pila, las instrucciones de máquina necesarias para colocar los valores +125 y -315 en la pila, serían las siguientes:

```

:
:
LDSP, I, 100  INICIALIZAR EL REGISTRO SP CON 100
LDA, I, +125  ALMACENAR 125 EN EL AC
PUSHA                    PONERLO EN LA PILA
LDA, I, -315  ALMACENAR -315 EN EL AC
PUSHA                    PONERLO EN LA PILA
:
:

```

Para obtener los datos de la pila, se tendrían las siguientes instrucciones de máquina:

```

:
:
POPA                    OBTENER UN DATO DE LA PILA
STA, I, DATO1          GUARDARLO EN MEMORIA
POPA                    OBTENER OTRO DATO DE LA PILA
STA, I, DATO2          GUARDARLO EN MEMORIA
:
:

```

La pila puede ser utilizada para guardar datos en forma temporal; sin embargo, su utilización principal es en las llamadas a rutinas o subprogramas. Una rutina es un segmento de programa que realiza una función específica y que puede ser requerido en varias partes de un programa.

Por ejemplo, supóngase que se necesitara determinar el valor absoluto de una cantidad en varias partes de un programa. Las instrucciones necesarias para obtener el valor absoluto de una cantidad se escribieron en el último ejemplo de programa presentado al final de la sección anterior. Las preguntas que surgen en este caso son las siguientes: ¿Es necesario repetir este conjunto de instrucciones dentro de un programa cada vez que se necesite obtener el valor absoluto de una cantidad?, ¿Será posible escribir estas instrucciones solamente una vez y utilizarlas cada vez que se necesite determinar el valor absoluto de una cantidad?

Si el lector ha realizado ejercicios de programación en algún lenguaje computacional de alto nivel, sabrá que si es posible escribir las instrucciones solo una vez como un subprograma y utilizarlas cada vez que sea necesario, en varias partes de un programa.

Para permitir el diseño y la utilización de subprogramas en una computadora digital, será necesario contar al menos con dos instrucciones de máquina adicionales:

- **Transferir el control de la operación a un subprograma (JSR).**

Esta instrucción permite transferir la secuencia de ejecución de instrucciones a un subprograma. Para posteriormente poder regresar a la instrucción que está después de la llamada al subprograma es indispensable guardar en la unidad de memoria el valor que tiene el registro contador de programa al momento de iniciar la transferencia al subprograma. El valor del registro contador de programa se guarda automáticamente en la pila al momento de ejecutar esta instrucción y luego se almacena en el registro contador de programa la dirección de memoria donde reside la rutina. El contenido del registro apuntador a la pila es incrementado en una unidad.

- **Regresar de un subprograma al programa que lo llamó (RTN).**

Al ejecutar esta instrucción, se almacena en el registro contador de programa la dirección que está en el tope de la pila, en memoria. En esta operación, primeramente se decrementa el valor contenido en el registro apuntador a la pila en una unidad y luego se utiliza el nuevo valor como la dirección de la celda de memoria de la cual se obtiene la dirección que se almacena en el registro contador de programa. Este valor es la dirección de la siguiente instrucción en el programa que hizo la llamada al subprograma. Es importante que el programador se asegure que este valor se encuentre en el tope de la pila para poder regresar a la siguiente dirección desde donde fue llamada la rutina al ejecutar la instrucción RTN.

La instrucción **JSR** permite tener direccionamiento relativo, absoluto e indirecto. La instrucción **RTN** no utiliza el campo para tipo de direccionamiento ni el de dirección en el formato de instrucciones. Para realizar la ejecución de las instrucciones anteriores se necesita añadir una nueva micro operación a las ya existentes.

- Transferir una dirección del PC al MDR:
$$\left\{ \begin{array}{l} [\text{MDR}]_{5-3} \leftarrow 000 \\ [\text{MDR}]_{2-0} \leftarrow [\text{PC}] \end{array} \right\}$$

El ciclo de ejecución de las nuevas instrucciones de máquina es el siguiente:

- Instrucción **JSR**:

Con direccionamiento relativo:

$$\begin{array}{l} [\text{MAR}] \leftarrow [\text{SP}] \\ \left\{ \begin{array}{l} [\text{MDR}]_{5-3} \leftarrow 000 \\ [\text{MDR}]_{2-0} \leftarrow [\text{PC}] \end{array} \right\} \end{array}$$

MMWrite

$$\begin{array}{l} [\text{SP}] \leftarrow [\text{SP}] + 1 \\ [\text{PC}] \leftarrow [\text{PC}] + [\text{IR}]_{2-0} \end{array}$$

Con direccionamiento absoluto:

$$\begin{aligned} &[MAR] \leftarrow [SP] \\ &\left\{ \begin{array}{l} [MDR]_{5-3} \leftarrow 000 \\ [MDR]_{2-0} \leftarrow [PC] \end{array} \right\} \end{aligned}$$

MMWrite

$$[SP] \leftarrow [SP] + 1$$
$$[PC] \leftarrow [IR]_{2-0}$$

Con direccionamiento indirecto:

$$\begin{aligned} &[MAR] \leftarrow [SP] \\ &\left\{ \begin{array}{l} [MDR]_{5-3} \leftarrow 000 \\ [MDR]_{2-0} \leftarrow [PC] \end{array} \right\} \end{aligned}$$

MMWrite

$$[SP] \leftarrow [SP] + 1$$
$$[MAR] \leftarrow [IR]_{2-0}$$

MMRead

$$[PC] \leftarrow [MDR]_{2-0}$$

• Instrucción **RTN**:

$$[SP] \leftarrow [SP] - 1$$
$$[MAR] \leftarrow [SP]$$

MMRead

$$[PC] \leftarrow [MDR]_{2-0}$$

Los códigos de operación que se asignarán a estas nuevas instrucciones son:

JSR	50
RTN	51

Para apreciar la ventaja de contar con estas instrucciones se muestra un programa, denominado programa principal, para determinar el valor absoluto de tres cantidades, usando una rutina o subprograma que calcula el valor absoluto de la cantidad que contiene el registro acumulador.

ORG, 000	LA SIG INSTR VA EN LA DIR 000
LDA, A, A	CARGAR EL DATO A EN EL AC
JSR, A, VALABS	IR A LA RUTINA

	STA, A, ABSA	GUARDAR EL RESULTADO EN ABSA
	LDA, A, B	CARGAR EL DATO B EN EL AC
	JSR, A, VALABS	IR A LA RUTINA
	STA, A, ABSB	GUARDAR EL RESULTADO EN ABSB
	LDA, A, C	CARGAR EL DATO C EN EL AC
	JSR, A, VALABS	IR A LA RUTINA
	STA, A, ABSC	GUARDAR EL RESULTADO EN ABSC
	HLT	TERMINAR LA EJECUCION
	ORG, 010	LA SIG INSTR VA EN LA DIR 010
A	DEF, -125	DEFINIR EL VALOR -125
B	DEF, +138	DEFINIR EL VALOR +138
C	DEF, -112	DEFINIR EL VALOR -112
ABSA	DEF, 0	DEFINIR EL VALOR 0
ABSAB	DEF, 0	DEFINIR EL VALOR 0
ABASC	DEF, 0	DEFINIR EL VALOR 0
	ORG, 020	LA SIG INSTR VA EN LA DIR 010
VALABS	JMN, A, NEGAT	PROBAR EL RESULTADO
	JMP, A, SIG	NO ES NEGATIVO, IR A SIG
NEGAT	NEG	NEGATIVO, CAMBIARLE EL SIGNO
SIG	RTN	REGRESAR AL PROGRAMA

El programa principal empieza en la dirección de memoria 000 y calcula el valor absoluto de las cantidades A, B y C, que están en las direcciones de memoria 010, 011 y 012 respectivamente. Los valores absolutos de estos tres datos se guardan en las direcciones de memoria ABSA, ABSB y ABSC, localizadas en las direcciones de memoria 13, 14 y 15.

La rutina que determina el valor absoluto del dato que está contenido en el registro acumulador se encuentra en la dirección simbólica de memoria VALABS, que físicamente es la dirección 020.

En este programa se aprovechó esta rutina en tres ocasiones pero solamente se tuvo que codificar una sola vez. Las instrucciones que tiene esta rutina son pocas y si se hubieran puesto tres veces en el programa, éste no hubiera crecido mucho. Se tomó un caso simple para ejemplificar el uso de estas nuevas instrucciones, pero si la rutina hubiese tenido 200 instrucciones, se podría apreciar mejor los beneficios de tenerla que codificar una sola vez y poderla utilizar desde varios puntos del programa principal. Además, solamente ocuparía 200 posiciones de memoria para ejecutarla, independientemente del número de puntos diferentes de donde se llamara. Si se incluyeran todas las instrucciones cada vez que se necesitaran, se ocuparían 200 lugares de memoria para colocarlas en cada una de las ocasiones.

2.5. RESUMEN

En este capítulo se presentó la arquitectura básica de una computadora digital, explicando la función de cada una de sus partes principales y la relación existente entre estos componentes.

También se explicó la operación de esta computadora en base a micro operaciones e instrucciones de máquina, introduciendo los fundamentos del lenguaje ensamblador y mostrando pequeños programas de ejemplo.

Finalmente, se describió la operación de la pila en memoria, creando nuevas instrucciones para su manejo. Se introdujo el concepto de subrutinas o subprogramas, las cuales hacen uso de la pila en su ejecución y se mostró un programa principal que realiza llamadas a un subprograma para entender la operación conjunta de un programa principal y sus subprogramas.

2.6. PROBLEMAS

1. Describa brevemente como está organizada la unidad de memoria de la computadora descrita en este capítulo.
2. Se pueden realizar dos tipos de operaciones sobre la unidad de memoria de la computadora digital. Describa cada una de estas operaciones.
3. ¿Cuál es la función de la unidad de control de la computadora presentada en este capítulo?
4. ¿Cuál es la función de la unidad aritmética-lógica de la computadora presentada en este capítulo?
5. Describa para que se utilizan los registros MAR, MDR, IR, PC y A de la computadora digital presentada en este capítulo.
6. ¿Cuál es la diferencia entre una instrucción de máquina y una micro operación?
7. En la computadora digital utilizada en este capítulo se tienen cuatro tipos de direccionamiento. Describa cada uno de ellos.
8. ¿Para que sirve el ciclo de FETCH en la computadora digital?
9. Liste la secuencia de micro operaciones necesarias para realizar el ciclo de FETCH de una instrucción de máquina.
10. Liste la secuencia de micro operaciones necesarias para realizar la instrucción de máquina LDA para cada uno de los cuatro tipos de direccionamiento usados en este capítulo.
11. Describa la operación de la pila en memoria de la computadora digital presentada en este capítulo.
12. ¿Qué es el lenguaje ensamblador y para qué sirve?
13. ¿Qué ventajas tiene codificar un programa en lenguaje ensamblador?

14. Explique la operación de las instrucciones JSR y RTN.
15. Traduzca el último programa presentado en el capítulo a lenguaje maquinal.
16. Codifique un programa principal y un subprograma en lenguaje ensamblador que realicen lo siguiente:
El programa principal deberá poner dos datos en la pila, A y B y llamar al subprograma. Estos dos datos están en memoria.
Suponga que la pila empieza en la dirección de memoria 200, para lo cual es necesario cargar este valor en el registro SP antes de llamar al subprograma.
El subprograma deberá calcular el valor absoluto de la diferencia de los dos datos que el programa principal dejó en la pila y regresar este valor en el acumulador. Recuérdese que la dirección de retorno al programa principal también se deja en la pila.
Finalmente, el programa principal deberá dejar el resultado en el lugar de memoria asignado al dato C.
17. Traduzca el programa y subprograma del problema anterior a lenguaje maquinal.

CAPITULO 3

OPERACIONES DE ENTRADA Y SALIDA E INTERRUPCIONES

3.1. CONCEPTOS BASICOS

Las computadoras digitales necesitan tener interacción con el medio que las rodea para realizar las diferentes funciones en las que se utilizan. En las aplicaciones de procesamiento de datos, es necesario alimentar tanto los datos como los programas que establecen la forma de procesarlos. Adicionalmente, se necesita ver y analizar los resultados que generan estos programas. La forma más común de alimentar datos y programas a las computadoras digitales es el teclado. Los principales dispositivos que se usan para visualizar la información generada por las computadoras digitales son las pantallas de vídeo, las impresoras y los graficadores.

En las aplicaciones de control y automatización de procesos, la computadora digital interactúa con sensores para obtener información sobre el estado de los procesos. En base a estos datos, genera acciones de control, que a su vez deben ser comunicadas a los actuadores para llevar a cabo la función de control del procesos.

En muchas aplicaciones se requiere que la información sea guardada en algún medio de almacenamiento permanente para posteriormente ser consultada, actualizada o procesada. Los programas de aplicación también necesitan ser guardados para no tener que escribirlos de nuevo cada vez que se necesite utilizarlos. Los dispositivos más comúnmente usados para almacenar información y programas son los discos magnéticos, comúnmente denominados discos duros.

Todos estos equipos que interactúan con las computadoras digitales reciben el nombre genérico de dispositivos periféricos. La comunicación entre estos dispositivos y la computadora se realiza a través de la unidad de entrada salida.

Para que la computadora digital interactúe con un dispositivo periférico típico, es necesario proveer cuatro operaciones básicas:

- Transferir datos del periférico a la computadora.
- Transferir datos de la computadora al periférico.
- Conocer el estado del dispositivo.
- Realizar funciones de control sobre el dispositivo.

Para ejemplificar lo anterior, supóngase que se va a utilizar una computadora digital para transferir datos a otra computadora utilizando un módem. Este dispositivo permite que la computadora envíe los datos a través de la línea telefónica. Inicialmente, la computadora necesita saber si el módem está encendido y listo para operar. La forma en que se puede determinar lo anterior es realizar una lectura del estado del dispositivo. Si el módem está listo, es necesario indicarle a que velocidad deberá realizar la transferencia de datos por la línea telefónica. Esta operación se efectúa mediante una función de control sobre el dispositivo. Finalmente, será necesario enviarle los datos que se desean transferir. Estos se enviarán

realizando una transferencia de datos de la computadora al módem. La computadora que recibirá los datos necesita a su vez, transferirlos del módem que está en el otro extremo de la línea telefónica, a su unidad central de proceso. Existen otras operaciones adicionales que se involucran en este proceso; sin embargo, para los fines que se persiguen en el ejemplo, las operaciones mencionadas son suficientes.

Para realizar las operaciones mencionadas anteriormente, se asignan cuatro direcciones diferentes a cada dispositivo periférico. Desde el punto de vista de la computadora, dos de estas direcciones se utilizarán como entrada y dos como salida. Una de las direcciones que se usan como entrada permitirá obtener datos del dispositivo mientras que la otra se usará para leer su estado. Las direcciones que se usan como salida proveen la facilidad de enviar datos al dispositivo y realizar funciones de control sobre el mismo.

En el capítulo anterior se mostró la unidad de entrada salida y se mencionó que todos los dispositivos periféricos estaban conectados a ella. Hacer esto sería poco práctico debido a que esta unidad sería sumamente compleja, ya que tendría que realizar funciones para comunicarse con discos magnéticos, módems, impresoras, pantallas de vídeo, teclados, etc.

En la práctica, se tienen varias unidades de entrada salida, donde cada una de ellas se diseña para manejar un cierto tipo de dispositivos. Todas estas unidades están conectadas al bus de la computadora. Estas unidades especializadas de entrada salida se conocen comúnmente como controladores y son las que tienen asignadas las direcciones mediante las cuales se identifica el dispositivo periférico que tiene conectada cada una. En un controlador, cada una de estas cuatro direcciones está asociada a un registro en el controlador.

En la figura 3.1 se muestran esquemáticamente un controlador típico

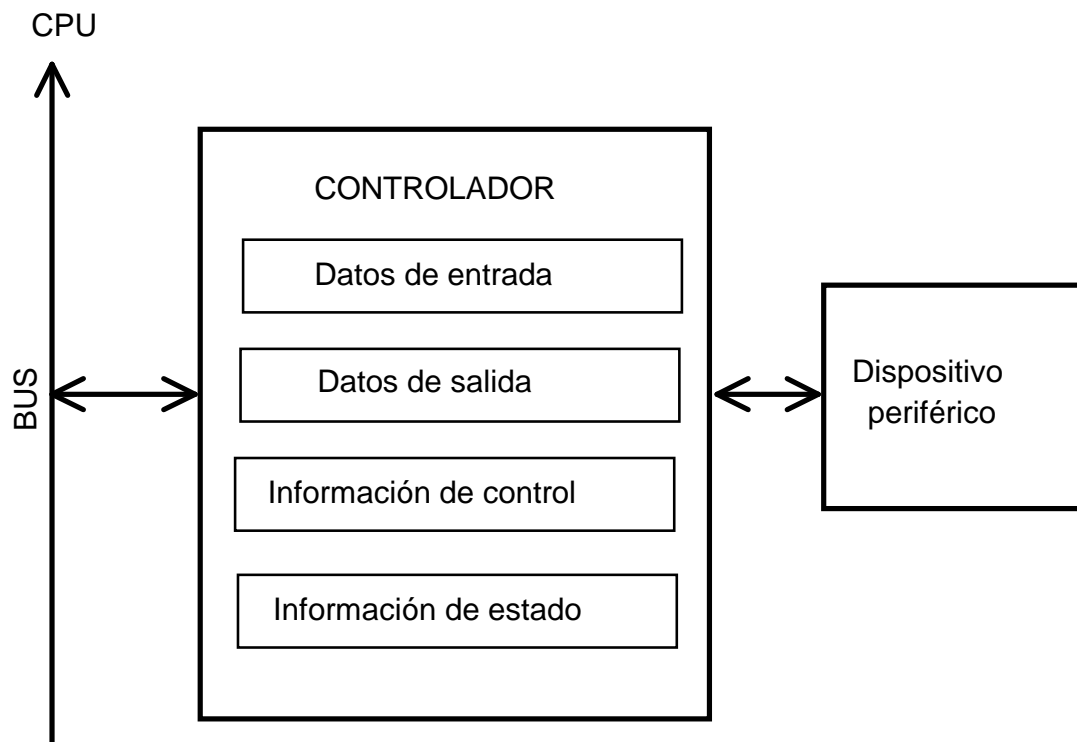


Figura 3.1 Comunicación con dispositivos periféricos

La computadora digital realiza la transferencia de datos hacia el dispositivo en forma similar a como se hace una escritura a memoria. La transferencia en sentido opuesto se efectúa en forma análoga a una lectura de memoria. Si se hace una operación de salida a la dirección asociada con la salida de datos, se enviará un dato al dispositivo periférico. Si se usa la dirección de salida asociada con el control del dispositivo, se enviará información de control al periférico. Para obtener un dato del dispositivo, se deberá realizar una operación de lectura a la dirección asociada con la entrada de datos, mientras que una operación de lectura a la dirección asociada con el estado del dispositivo, dará como resultado que la computadora obtenga el estado del periférico.

Existen dos formas básicas en las cuales una computadora digital puede operar los dispositivos periféricos. La primera es la operación bajo el control directo de la unidad central de proceso, la segunda es mediante la utilización de procesadores especiales de entrada salida.

Cuando los dispositivos periféricos son manejados directamente bajo el control de la unidad central de proceso, se han utilizado principalmente dos filosofías. En una de ellas, la unidad central de proceso opera los dispositivos periféricos exactamente igual a como realiza las operaciones de lectura y escritura a memoria. Si se usa este esquema, cada dispositivo periférico tendrá cuatro direcciones únicas, que deberán ser diferentes a cualquier dirección de memoria. Para enviar un dato o información de control a un dispositivo se utiliza la instrucción **STA**, solamente que la dirección usada es la dirección correspondiente del dispositivo (dirección de salida de datos o dirección de control). Una lectura de un dato del dispositivo o la obtención de

su estado, se realizaría con la instrucción **LDA**, empleando la dirección adecuada del dispositivo (dirección de entrada de datos o dirección de estado).

En la otra técnica, la computadora digital también realiza las operaciones sobre los dispositivos periféricos en forma similar a como se efectúan las operaciones de lectura y escritura a memoria, pero se tiene la instrucción especial **OUT**, la cual se usa para enviar datos o información de control. Para obtener el estado del dispositivo o para leer un dato del dispositivo se usa la instrucción especial **IN**. Ambas instrucciones tienen como parámetros una dirección. Bajo este esquema, las direcciones de los dispositivos periféricos no tienen que ser diferentes a las direcciones de memoria.

La forma de manejar los dispositivos periféricos mediante procesadores especiales de entrada salida se describirá posteriormente en este capítulo.

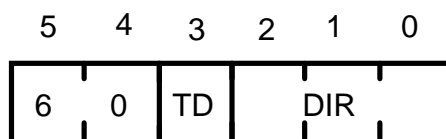
3.2. INSTRUCCIONES DE ENTRADA SALIDA

En esta sección se describirá el manejo de los dispositivos periféricos bajo el control de la unidad central de proceso, suponiendo que se tienen instrucciones especiales para enviar y recibir datos de estos dispositivos. En este caso, se tendrá una línea en el bus que indicará si la operación de lectura o escritura deberá ser atendida por la unidad de memoria o por la unidad de entrada salida. Esta línea no existe en las computadoras que utilizan la otra filosofía.

Las instrucciones de máquina que permitirán controlar y utilizar los dispositivos periféricos, como se mencionó anteriormente, son **IN** y **OUT**.

La instrucción **IN** es empleada para leer un dato de un dispositivo si se especifica la dirección de entrada de datos del periférico. Si se desea obtener información sobre el estado del dispositivo, deberá emplearse la dirección asociada con el estado del dispositivo.

El formato de esta instrucción es el siguiente:



Formato de la instrucción **IN**

El código de operación de la instrucción es 60, el tipo de direccionamiento se indica en la posición 3. El contenido de las posiciones 0 a 2 depende del tipo de direccionamiento utilizado en la instrucción. Las direcciones efectivas válidas para esta instrucción serán las direcciones asociadas con la entrada de datos de los dispositivos y las direcciones asociadas con el estado de los dispositivos solamente.

Esta instrucción trabaja solamente con los tipos de direccionamiento absoluto e indirecto. Cuando se utiliza el tipo de direccionamiento absoluto, las posiciones 0 a 2 deberán tener la dirección del dispositivo, que representa la dirección efectiva de la instrucción. Si se utiliza el tipo de direccionamiento indirecto, las posiciones 0 a 2 deberán contener la dirección de memoria en la cual se encuentra la dirección del dispositivo. La generación de la dirección efectiva se podrá apreciar mejor analizando el ciclo de ejecución de la instrucción.

Esta instrucción opera en forma similar a una lectura de memoria, con la diferencia de que una línea del bus indica que la operación debe ser atendida por la unidad de entrada salida y no por la unidad de memoria. En caso de una lectura de memoria, esta línea indicaría que la operación debe ser atendida por la unidad de memoria.

El ciclo de ejecución de esta instrucción está compuesto por las siguientes micro operaciones:

Con direccionamiento absoluto:

$$[MAR] \leftarrow [IR]_{2-0}$$

I / ORead

$$[AC] \leftarrow [MDR]$$

Con direccionamiento indirecto:

$$[MAR] \leftarrow [IR]_{2-0}$$

MMRead

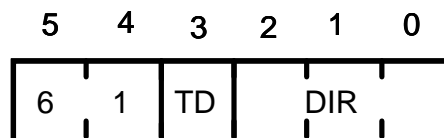
$$[MAR] \leftarrow [MDR]_{2-0}$$

I / ORead

$$[AC] \leftarrow [MDR]$$

La micro operación I / ORead opera en forma similar a una lectura de memoria, solamente que la línea correspondiente del bus indica que dicha operación debe ser realizada por la unidad de entrada salida y no por la unidad de memoria.

La instrucción **OUT** se usa para enviar un dato al dispositivo o para enviarle información de control, dependiendo de la dirección empleada en la instrucción. Su formato es el siguiente:



Formato de la instrucción **OUT**

El código de operación es 61. El tipo de direccionamiento funciona en forma similar a como opera la instrucción **IN**. El contenido del campo de dirección depende también del tipo de direccionamiento utilizado. La dirección efectiva deberá ser la dirección de salida de datos del

dispositivo si se desea enviarle un dato, o la dirección de control si se trata de enviar información de control.

EL ciclo de ejecución de esta instrucción es el siguiente:

Con direccionamiento absoluto:

$$[MAR] \leftarrow [IR]_{2-0}$$

$$[MDR] \leftarrow [AC]$$

I / OWrite

Con direccionamiento indirecto:

$$[MAR] \leftarrow [IR]_{2-0}$$

MMRead

$$[MAR] \leftarrow [MDR]_{2-0}$$

$$[MDR] \leftarrow [AC]$$

I / OWrite

La micro operación I / OWrite opera en forma similar a una escritura a memoria, activando la línea que indica que la operación debe ser atendida por la unidad de entrada salida y no por la unidad de memoria.

3.3. INTERRUPCIONES

Supóngase que se está ejecutando un programa que primero realiza una serie de cálculos durante un tiempo de 5 minutos y produce resultados. Estos resultados serán enviados a una impresora para posteriormente analizarlos. Una vez enviados a impresión, el programa realiza otra serie de cálculos durante otros cinco minutos, genera más resultados, los envía a impresión y termina su ejecución.

El estar enviando los resultados de la primera fase a la impresora, la unidad central de proceso envía un dato a impresión y se pone a leer el estado de la impresora para saber cuando se terminó de imprimir ese dato y poder enviarle el siguiente, hasta terminar de enviar todos los datos. La velocidad de cualquier impresora es mucho menor que la velocidad de un procesador y al realizar el proceso mencionado anteriormente, se desperdiciaría mucho tiempo de cómputo solamente en estar esperando que la impresora haya terminado de imprimir el dato que se le envió y esté disponible para recibir el siguiente dato.

Si la unidad central de proceso no tuviera que estar leyendo el estado de la impresora para saber cuando enviarle el siguiente dato, se podría aprovechar ese tiempo en empezar a realizar la siguiente serie de cálculos, pudiéndose ejecutar el programa completo en menor tiempo.

Para estar en posibilidades de aprovechar el tiempo del procesador de una manera más eficiente, se creó el concepto de interrupciones. Al utilizar interrupciones, la unidad central de proceso envía un dato a la impresora y puede continuar ejecutando otras instrucciones sin tener que estar leyendo el estado de la impresora para saber cuando puede enviarle el siguiente dato. Cuando la impresora termina de imprimir el dato que se le envió, genera una interrupción para indicarle a la unidad central de proceso que ya acabó de imprimirlo. La unidad central de proceso, al recibir la interrupción, suspende temporalmente el programa que estaba ejecutando, envía el siguiente dato a la impresora y continúa con el programa, donde se había quedado. Esta forma de operar permite aprovechar el procesador de una manera más eficiente.

Para permitir que la unidad central de proceso trabaje en la forma descrita con anterioridad se puede utilizar un esquema similar al empleado en la llamada a subprogramas o rutinas. Supóngase que cuando se generó una interrupción, el procesador estaba realizando la suma de una serie de números y en el registro acumulador estaba el valor acumulado de la suma hasta ese momento. Si al atender la interrupción se va a utilizar el registro acumulador para enviar el siguiente dato a la impresora, se perdería el valor acumulado de la suma que se tenía, a menos que ese valor sea guardado temporalmente en memoria. Además, el registro contador de programa tenía la dirección de la siguiente instrucción, pero al ejecutar las instrucciones necesarias para enviar el siguiente dato a la impresora, el valor de ese registro se modificaría y el procesador no sabría donde se había quedado en el momento en que fue interrumpido. Esto implica que también es necesario guardar temporalmente el valor del registro contador de programa en algún lugar de memoria, para posteriormente restaurar su valor y poder continuar donde fue interrumpido. El registro de banderas también puede ser modificado en el subprograma que atiende las interrupciones y es menester guardarlo para posteriormente restaurar su valor. En la siguiente sección se describe la forma en que se procesan las interrupciones.

3.4. PROCESAMIENTO DE INTERRUPCIONES

Las interrupciones indican a la unidad central de proceso que algún dispositivo periférico necesita ser atendido. Como se mencionó en el ejemplo anterior, puede ser debido a que la impresora ya terminó de imprimir el dato anterior y se lo indica al procesador mediante una interrupción. También se puede tener el caso en que el usuario utilizó el teclado y se generó una interrupción por cada tecla que se oprimió para indicarle al procesador que necesita atender el teclado, que es otro dispositivo periférico.

La unidad central de proceso puede estar en posición de aceptar o no aceptar interrupciones de dispositivos periféricos. Por ejemplo, si se está ejecutando un proceso que no debe ser interrumpido, el procesador deberá ejecutar una instrucción que deshabilite las interrupciones. En caso de que el procesador esté ejecutando un proceso que sí pueda ser interrumpido, se deberá ejecutar la instrucción que habilite las interrupciones. Se necesita una bandera adicional que indique si las interrupciones están habilitadas o deshabilitadas, esta bandera se denominará **I**.

La instrucción que habilita las interrupciones es **INTON**. Su función es encender la bandera **I**, indicando con ello que las interrupciones están habilitadas, es decir, que el procesador si acepta ser interrumpido. Esta instrucción solamente utiliza el código de operación.

En caso de que no se desee que el procesador sea interrumpido, se deberá ejecutar la instrucción **INTOFF**, cuya función es apagar la bandera **I**, lo cual deshabilita las interrupciones. Esto implica que aún cuando un dispositivo periférico genere una interrupción, el procesador no la atenderá. Esta instrucción, al igual que la anterior, solamente usa el código de operación.

Las micro operaciones necesarias para estas instrucciones son:

- Encender la bandera I: $I \leftarrow 1$ (encendida)
- Apagar la bandera I: $I \leftarrow 0$ (apagada)

La definición de las instrucciones para habilitar o deshabilitar las interrupciones, en base a micro operaciones, es muy simple:

- Instrucción **INTON**:
 $I \leftarrow 1$ (encendida)
- Instrucción **INTOFF**:
 $I \leftarrow 0$ (apagada)

Los códigos de operación para estas últimas instrucciones de máquina son:

INTON	03
INTOFF	04

Antes de proceder con la descripción de la forma en que se procesa una interrupción, se analizará con mayor detalle bajo cuales condiciones se generará una interrupción. Ya se mencionó que la unidad central de proceso puede estar en posición de aceptar o no aceptar interrupciones, dependiendo del estado de la bandera **I**, la cual se manipula mediante las instrucciones **INTON** e **INTOFF**. Si esta bandera está apagada, el procesador no atenderá ninguna interrupción de los periféricos.

Adicionalmente a la bandera **I**, el controlador de cada dispositivo periférico tiene otra bandera similar que indica si ese dispositivo en particular tiene permiso para interrumpir o no. Esta bandera propia de cada controlador se enciende o apaga mediante el envío de información de control al periférico, mediante la instrucción **OUT**, con la dirección de control del dispositivo . Al leer la información del estado de un dispositivo mediante la instrucción **IN**, con la dirección de estado del dispositivo, se obtiene información sobre el estado de la bandera particular del controlador.

Para que un dispositivo periférico específico pueda generar una interrupción, la bandera que le da capacidad para interrumpir debe estar encendida. Ahora bien, si un periférico genera una interrupción y la bandera **I** del procesador está apagada, esta interrupción no será atendida. La

unidad central de proceso aceptará y atenderá la interrupción solamente cuando su bandera **I** estuviere encendida.

Al presentarse una interrupción, el procesador la atiende cuando termina de ejecutar completamente la instrucción de máquina que está en proceso al momento de presentarse la interrupción. Recuérdese que en el ciclo de ejecución de una instrucción de máquina pueden estar involucradas varias micro operaciones, realizadas en una cierta secuencia. Si se atendiera una interrupción en el momento exacto en que ésta se presentara, podría quedar truncada la ejecución de una instrucción de máquina y el procesador no podría continuar con la ejecución de instrucciones en forma correcta después de atender la interrupción.

Para atender una interrupción, suponiendo que la unidad central de proceso tiene la bandera **I** encendida, se realizan los siguientes pasos en forma automática:

- Se termina de ejecutar la instrucción que está en proceso.
- Se guarda el estado del procesador en la pila.
- Se apaga la bandera **I** del procesador.
- Se realiza una transferencia incondicional a una dirección predeterminada donde está la rutina que atiende las interrupciones.

La rutina que atiende las interrupciones es un programa común y corriente. Esta rutina debe haber sido programada y almacenada previamente en la unidad de memoria a partir de la dirección específica a la cual se realiza la transferencia.

Una vez que se terminó de atender la interrupción, se debe restaurar el estado del procesador, que se había guardado en la pila, y se continúa con el proceso que se estaba ejecutando al momento de presentarse la interrupción.

El estado del procesador está formado por el contenido de los siguientes registros: **PC**, **FR** y **AC**.

Para guardar el estado del procesador se realizan, en forma automática, las micro operaciones necesarias para almacenar en la pila los contenidos de los registros que forman el estado del procesador:

$$\begin{aligned} & [\text{MAR}] \leftarrow [\text{SP}] \\ & \left\{ \begin{array}{l} [\text{MDR}]_{5-3} \leftarrow 000 \\ [\text{MDR}]_{2-0} \leftarrow [\text{PC}] \end{array} \right\} \end{aligned}$$

$$\begin{aligned} & \text{MMWrite} \\ & [\text{SP}] \leftarrow [\text{SP}] + 1 \end{aligned}$$

$$[\text{MAR}] \leftarrow [\text{SP}]$$

$$\left\{ \begin{array}{l} [\text{MDR}]_{5-4} \leftarrow 000 \\ [\text{MDR}]_{3-0} \leftarrow [\text{FR}] \end{array} \right\}$$

MMWrite

$$[\text{SP}] \leftarrow [\text{SP}] + 1$$

$$[\text{MAR}] \leftarrow [\text{SP}]$$

$$[\text{MDR}] \leftarrow [\text{AC}]$$

MMWrite

$$[\text{SP}] \leftarrow [\text{SP}] + 1$$

Todas las micro operaciones anteriores ya existían, excepto la que transfiere el registro de banderas al MDR:

$$\bullet \text{ Transferir el FR al MDR: } \left\{ \begin{array}{l} [\text{MDR}]_{5-4} \leftarrow 00 \\ [\text{MDR}]_{3-0} \leftarrow [\text{FR}] \end{array} \right\}$$

Una vez que se ha guardado el estado del procesador en la pila, se transfiere control a la rutina que atiende las interrupciones. Lo primero que debe hacerse en esta rutina es determinar cuál de todos los dispositivos periféricos generó la interrupción. Para esto, la rutina que atiende las interrupciones debe interrogar a cada periférico para conocer su estado y saber si está pidiendo ser atendido. Una vez que se determinó cual periférico originó la interrupción, debe atenderse este dispositivo y continuar con el programa que se estaba ejecutando al presentarse la interrupción. Para continuar con el programa que fue interrumpido basta restaurar el estado del procesador.

Este fue el primer enfoque que se utilizó para el procesamiento de interrupciones y presenta dos inconvenientes principales. El primero es que se pierde tiempo en determinar cual dispositivo periférico necesita ser atendido. El segundo problema es que la rutina que atiende interrupciones necesita ser subdividida en varios subprogramas que atiendan a cada uno de los diferentes periféricos de la computadora digital.

La filosofía que es más usada actualmente para el procesamiento de interrupciones permite identificar al dispositivo que generó la interrupción de una manera más rápida y en forma automática. La forma en que opera este esquema se describe a continuación.

Una vez guardado el estado del procesador en la pila, la unidad central de proceso activa una línea de control en el bus para indicarle al controlador del dispositivo que se aceptó su interrupción. Esta línea se denomina **INTA** (Interrupt acknowledge) y la micro operación que la activa será llamada igual que la línea.

Cuando el controlador del dispositivo periférico que generó la interrupción detecta que se activa la línea **INTA**, pone en las líneas de datos del bus la dirección asignada al dispositivo.

La unidad central de proceso lee estas líneas y usa el valor de la dirección del dispositivo para calcular la dirección de memoria donde se encuentra la dirección de la rutina que atiende la interrupción de ese dispositivo. Finalmente, se transfiere el control de ejecución a la dirección de la rutina que atiende la interrupción. Esto es muy similar, más no igual, a realizar una instrucción **JSR** con direccionamiento indirecto.

Los pasos que se realizan en forma automática bajo este esquema son los siguientes:

- Se termina de ejecutar la instrucción que está en proceso.
- Se guarda el estado del procesador en la pila.
- Se apaga la bandera **I** del procesador.
- Se obtiene la dirección del dispositivo y se calcula la dirección de memoria donde reside la rutina que atiende la interrupción
- Se realiza una transferencia incondicional a la dirección donde está la rutina que atiende la interrupción.

La celda de memoria cuya dirección es igual a la dirección del dispositivo que generó la interrupción deberá contener la dirección del subprograma que atiende las interrupciones de dicho dispositivo.

Las micro operaciones involucradas en este proceso son las siguientes:

$$\begin{array}{l} \text{INTA} \\ \text{[MAR]} \leftarrow [\text{líneas de datos del bus}]_{2-0} \\ \text{MMRead} \\ \text{[PC]} \leftarrow \text{[MDR]}_{2-0} \end{array}$$

El programador de la computadora deberá haber almacenado la dirección de la rutina que atiende la interrupción, en la dirección de memoria correspondiente así como las instrucciones de la rutina específica para procesar las interrupciones del periférico.

Por ejemplo, supóngase que el dispositivo cuya dirección es 21 genera una interrupción. La unidad central de proceso, después de haber terminado de ejecutar la instrucción que estaba en proceso, guarda el estado del procesador en la pila, apaga la bandera **I**, y obtiene, de la dirección de memoria 21, la dirección donde está la rutina que atiende esa interrupción. Supóngase que en la dirección de memoria 21 estaba almacenada la dirección 200. La ejecución de instrucciones continuaría en la dirección de memoria 200.

Las instrucciones de la rutina que atiende la interrupción deberán haber sido almacenadas a partir de la dirección de memoria 200 por el programador. En esta rutina se realizan las instrucciones necesarias para atender el dispositivo que generó la interrupción. Al terminar, deberá continuarse con la ejecución del programa que fue interrumpido.

Para regresar al programa que estaba siendo ejecutado al momento de presentarse la interrupción, el programador deberá ejecutar en la rutina de interrupción la siguiente instrucción:

RTI

La instrucción **RTI** restaura el estado que tenía el procesador al momento de presentarse la interrupción. Restaurar el estado del procesador implica poner en los registros de banderas, acumulador y contador de programa, los valores que tenían originalmente. Recuérdese que estos registros fueron almacenados en la pila al atenderse la interrupción. La instrucción que restaura el estado del procesador se describe a continuación:

Recuérdese que cuando se guardó el estado del procesador en la pila todavía estaba encendida la bandera **I**; por lo tanto, la instrucción **RTI** habilitará de nuevo las interrupciones.

- **Restaurar el estado del procesador para regresar de una interrupción (RTI).**
Esta instrucción restaura los valores que contenían los registros de banderas, acumulador y contador de programa antes de atender la interrupción. Al realizar esta operación, se obtienen de la pila, los valores originales que tenían estos registros, en el orden inverso en que se guardaron.

Para crear la instrucción de máquina **RTI** se necesita una nueva micro operación:

- Transferir el contenido del MDR al FR : $[FR] \leftarrow [MDR]_{3-0}$

La definición de la instrucción **RTI** en base a micro operaciones es la siguiente:

- Instrucción **RTI**:

$$[SP] \leftarrow [SP] - 1$$

$$[MAR] \leftarrow [SP]$$

$$\text{MMRead}$$

$$[AC] \leftarrow [MDR]$$

$$[SP] \leftarrow [SP] - 1$$

$$[MAR] \leftarrow [SP]$$

$$\text{MMRead}$$

$$[FR] \leftarrow [MDR]_{3-0}$$

$$[SP] \leftarrow [SP] - 1$$

$$[MAR] \leftarrow [SP]$$

$$\text{MMRead}$$

$$[PC] \leftarrow [MDR]_{2-0}$$

El código de operación que se asignará a la instrucción **RTI** es el 52. Esta instrucción solamente utiliza el campo para el código de operación. Los demás campos serán puestos en cero.

Hasta este momento, la computadora digital básica que se presentó en el capítulo 2, se ha ido haciendo cada vez más compleja y ahora tiene capacidad de manejar una pila en memoria y de aceptar y procesar interrupciones para dar mayor eficiencia al procesador.

Una pregunta que probablemente se haya hecho el lector es la siguiente: ¿Qué pasaría si mientras el procesador está atendiendo una interrupción de un dispositivo periférico, otro dispositivo termina lo que estaba haciendo y trata de generar una interrupción? ¿Se pierde esta interrupción?

Si esto ocurriera, la interrupción no sería atendida por la unidad central de proceso en ese momento debido a que cuando se atendió la primera interrupción, se deshabilitaron las interrupciones. Sin embargo, la interrupción queda pendiente y, al momento de habilitar las interrupciones de nuevo, se genera la interrupción. En ese momento se atiende al dispositivo.

También pudiera ocurrir que, mientras se atiende la interrupción de un dispositivo, dos o más dispositivos diferentes terminan de realizar sus tareas y tratan de generar interrupciones. En este caso, al habilitar de nuevo las interrupciones, se generaría una interrupción.

Existe una línea en el bus, denominada **IRQ**, por la cual le llegan las interrupciones al procesador. Esta línea puede ser activada por cualquier dispositivo que necesite generar una interrupción. La línea que utiliza el procesador para indicar a un dispositivo que su interrupción fue aceptada y que debe poner su dirección en las líneas correspondientes del bus trabaja en forma diferente. Cuando un dispositivo genera o trata de generar una interrupción, corta la línea **INTA** para que ésta no llegue a los dispositivos que están conectados después de él en el bus.

Entonces, si al momento de habilitar las interrupciones, existen dos o más dispositivos tratando de generar una interrupción, se generaría una interrupción al procesador por la línea **IRQ**. Cuando el procesador acepta la interrupción y activa la línea **INTA**, esta línea será detectada por el dispositivo que esté interrumpiendo y se encuentre conectado físicamente más cercano al procesador en el bus. Los demás dispositivos no reciben la señal de aceptación de la interrupción porque el periférico más cercano que estaba interrumpiendo la cortó. Los otros dispositivos continúan esperando a ser atendidos ya que no se enteran que ya se atendió a otro periférico. Cuando un dispositivo que estaba generando una interrupción es atendido, deja de interceptar la línea **INTA**, de suerte que la siguiente vez que sea activada, la reciba el siguiente dispositivo que esté interrumpiendo.

Este esquema de procesamiento de interrupciones es el más simple y constituye un ejemplo con un solo nivel de interrupciones. La prioridad de un dispositivo respecto a los demás está determinada por la posición donde se conecta el dispositivo en el bus. Entre más cerca del procesador esté conectado un dispositivo en el bus, mayor será su prioridad; entre más alejado se encuentre del procesador, tendrá una prioridad más baja.

Existen otros esquemas más complejos en los cuales se manejan varios niveles de interrupción, adicionalmente a las prioridades propias de cada nivel. Estos esquemas no serán tratados en este texto, debido a que se consideran solamente los conceptos fundamentales sobre las computadoras digitales.

3.5. PROCESADORES DE ENTRADA SALIDA

La introducción del concepto de interrupciones permite que el procesador pueda continuar ejecutando instrucciones mientras un dispositivo periférico está realizando su tarea. Esto permite que se aproveche mejor el tiempo del procesador, pero de todas formas, la unidad central de proceso tiene que continuar atendiendo los periféricos y se desperdicia tiempo en aceptar la interrupción, guardar el estado del procesador, ejecutar la rutina de atención al dispositivo, restaurar el estado del procesador y regresar al programa que estaba siendo ejecutado.

Tratando de aprovechar la unidad central de proceso de una manera más eficiente, se diseñaron los procesadores de entrada salida. Estos dispositivos son realmente procesadores especializados que ayudan a manejar los dispositivos periféricos de una manera más eficiente, liberando a la unidad central de proceso de muchas tareas de entrada y salida.

Existen varios tipos de procesadores de entrada salida. En la literatura también son conocidos como canales o controladores de acceso directo a memoria (**DMA**). En este texto se usará "controlador de acceso directo a memoria" para la versión más simple de un procesador de entrada salida y se reservará el término "canal" para los procesadores más sofisticados.

3.5.1. Controladores de acceso directo a memoria.

Un controlador de acceso directo a memoria trabaja simultáneamente con la unidad central de proceso, atendiendo a un dispositivo periférico. En el área de ciencias computacionales se utiliza el término "concurrente" para indicar que dos o más dispositivos trabajan en forma simultánea.

Cuando se necesita transferir un conjunto de datos de un área de memoria a un dispositivo periférico o viceversa, la unidad central de proceso envía la información de control al dispositivo para prepararlo para la transferencia de datos y le indica al controlador de acceso directo a memoria que se haga cargo de la transferencia. Para esto, le indica la dirección del dispositivo, la cantidad de datos que se habrán de transferir, el sentido de la transferencia (entrada o salida) y la dirección en la unidad de memoria a partir de la cual se van a tomar o dejar los datos. Las interrupciones que genera el dispositivo periférico serán ahora atendidas por el controlador de acceso directo a memoria en lugar de ser atendidas por el procesador. Al final de la operación, el controlador de acceso directo a memoria, indicará al procesador que se terminó la operación mediante una interrupción. De esta forma, la unidad central de proceso recibirá una sola interrupción al final de la transferencia de datos en lugar de recibir una interrupción por cada dato que se transfiera. El procesador le envía la información necesaria al controlador de acceso directo a memoria mediante instrucciones **OUT**, usando la dirección de control asignada al

procesador de entrada salida. También puede obtener el estado del controlador de acceso directo a memoria usando la instrucción **IN** con la dirección de estado.

Se describirá ahora la forma en la cual el procesador y el controlador de acceso directo a memoria trabajan en forma concurrente. Supóngase que el procesador ya ordenó una operación al controlador de acceso directo a memoria para transferir una cierta cantidad de datos a una impresora y que este dispositivo está listo para recibir datos. Tanto la unidad central de proceso como el controlador de acceso directo a memoria necesitan obtener datos de memoria para continuar trabajando, pero no pueden hacerlo exactamente al mismo tiempo. La unidad central de proceso continúa operando en forma normal y cuando el controlador de acceso directo a memoria necesita obtener un dato de memoria para enviarlo a la impresora, solicita a la unidad central de proceso el permiso para hacerlo mediante una línea del bus, denominada **HOLD-REQ** (hold request). Cuando el procesador ve que se activa esta línea, continúa hasta terminar la ejecución de la instrucción que estaba realizando y contesta con otra línea del bus, llamada **HOLD-ACK** (hold acknowledge) para indicarle al controlador de acceso directo a memoria que puede proceder a obtener el dato de memoria. Esta línea no se queda activada, sino que solamente se da un pulso para indicarle al controlador de acceso directo a memoria que su petición fue aceptada. El procesador no puede continuar ejecutando instrucciones mientras el controlador de acceso directo a memoria realiza la transferencia del dato. Cuando el controlador ya obtuvo el dato de memoria, desactiva la línea **HOLD-REQ**, permitiendo que el procesador continúe normalmente. El controlador de acceso directo a memoria envía el dato a la impresora y cuando la impresora termina de imprimirlo, genera una interrupción, solamente que ahora esta interrupción es atendida por el controlador de acceso directo a memoria y se repite el proceso descrito anteriormente.

Bajo este esquema, el controlador de acceso directo a memoria le quita tiempo al procesador mientras obtiene cada dato de memoria. Esta forma de operar se conoce en la literatura como operación de robo de ciclo (cycle stealing) debido a que el controlador de acceso directo a memoria le quita o "roba" ciclos de memoria al procesador. Es cierto que la operación del procesador también se degrada bajo este esquema, pero solamente pierde el tiempo de un ciclo de acceso a memoria cada vez que el controlador de acceso directo a memoria transfiere un dato. Si la unidad central de proceso tuviera que atender al dispositivo, tendría que aceptar la interrupción, guardar el estado del procesador en la pila, transferir la ejecución a la rutina que atiende al dispositivo, ejecutarla, restaurar el estado del procesador y, finalmente, continuar con el programa que estaba ejecutando. La pérdida de tiempo en este caso sería mucho mayor comparada con perder el tiempo de un ciclo de acceso a memoria.

La velocidad de los dispositivos periféricos es muy variada. Un disco magnético puede transferir datos a velocidades mucho mayores que la velocidad con que puede recibir e imprimir datos una impresora. Debido a esto, se tienen diferentes tipos de controladores de acceso directo a memoria y la utilización un tipo u otro depende del dispositivo periférico con el cual se va a trabajar.

Existen tres tipos básicos de controladores de acceso directo a memoria:

- Controlador dedicado

- Controlador selector
- Controlador compartido

Controlador dedicado

Este tipo de controlador de acceso directo a memoria trabaja, como su nombre lo indica, dedicado a atender a un solo dispositivo periférico. Se usaría para manejar periféricos que transfieren grandes volúmenes de información y se utilizan con mucha frecuencia, tales como discos duros.

La arquitectura básica definida en el capítulo anterior debe ser modificada para incluir los conceptos presentados en este capítulo. En la figura 3.2 se muestra una posible configuración donde un controlador de acceso directo a memoria (DMA) del tipo dedicado, atiende las operaciones de entrada y salida del dispositivo periférico X.

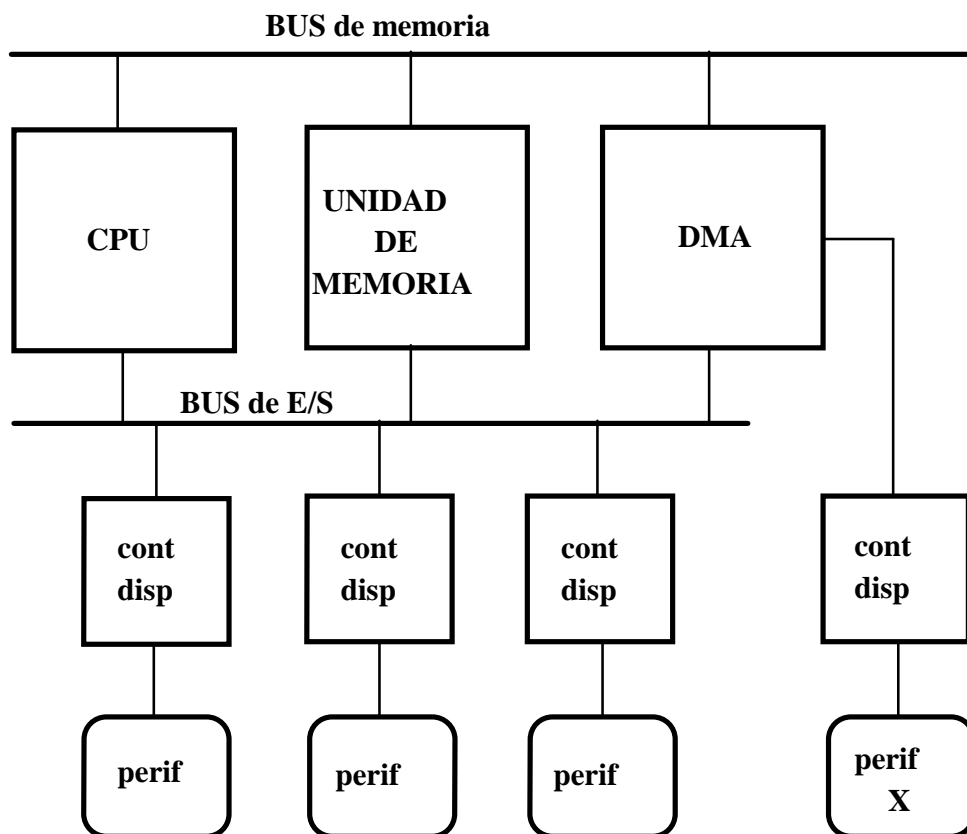


Figura 3.2. Controlador de acceso directo a memoria dedicado

En la arquitectura mostrada en la figura 3.2, la unidad central de proceso realiza directamente las operaciones de entrada y salida de los dispositivos periféricos, con excepción de las del periférico X, que es atendido por el controlador de acceso directo a memoria dedicado.

Es esta figura 3.2 se muestra claramente que la unidad de entrada salida no es un solo componente, sino que está realmente distribuida como lo indican los diferentes bloques denominados "controlador de dispositivo".

Controlador selector

Un controlador de acceso directo a memoria del tipo selector puede trabajar con varios dispositivos periféricos, pero puede atender solamente a uno a la vez. Cuando la unidad central de proceso prepara un dispositivo periférico para realizar una transferencia de datos a memoria, o de memoria; ordena al controlador de acceso directo a memoria que se haga cargo de dicha transferencia entre el periférico y la memoria. El controlador de acceso directo a memoria atiende solamente a ese dispositivo hasta que termina su transferencia de datos. En ese momento, genera una interrupción al procesador para indicarle que se terminó la operación e indicarle el estado de la misma, quedando liberado. El procesador puede asignarlo posteriormente a otro dispositivo para que controle la transferencia de información entre la unidad de memoria y el periférico.

Una posible arquitectura para una computadora con un controlador de acceso directo a memoria del tipo selector se muestra en la figura 3.3. En esta configuración, las operaciones de entrada y salida pueden ser realizadas bajo el control de la unidad central de proceso o bajo el control del controlador de acceso directo a memoria.

Controlador compartido

Un controlador de acceso directo a memoria de este tipo puede trabajar simultáneamente con varios dispositivos periféricos. La unidad central de proceso puede asignarle el control de varias transferencias de datos entre la unidad de memoria y varios dispositivos periféricos. El controlador de acceso directo a memoria coordina la transferencia de datos entre cada uno de los dispositivos periféricos y la unidad de memoria en forma independiente de las demás y generaría una interrupción cada vez que se termine una operación de transferencia de datos de uno de los periféricos. Este tipo de controlador es más versátil que los otros dos tipos.

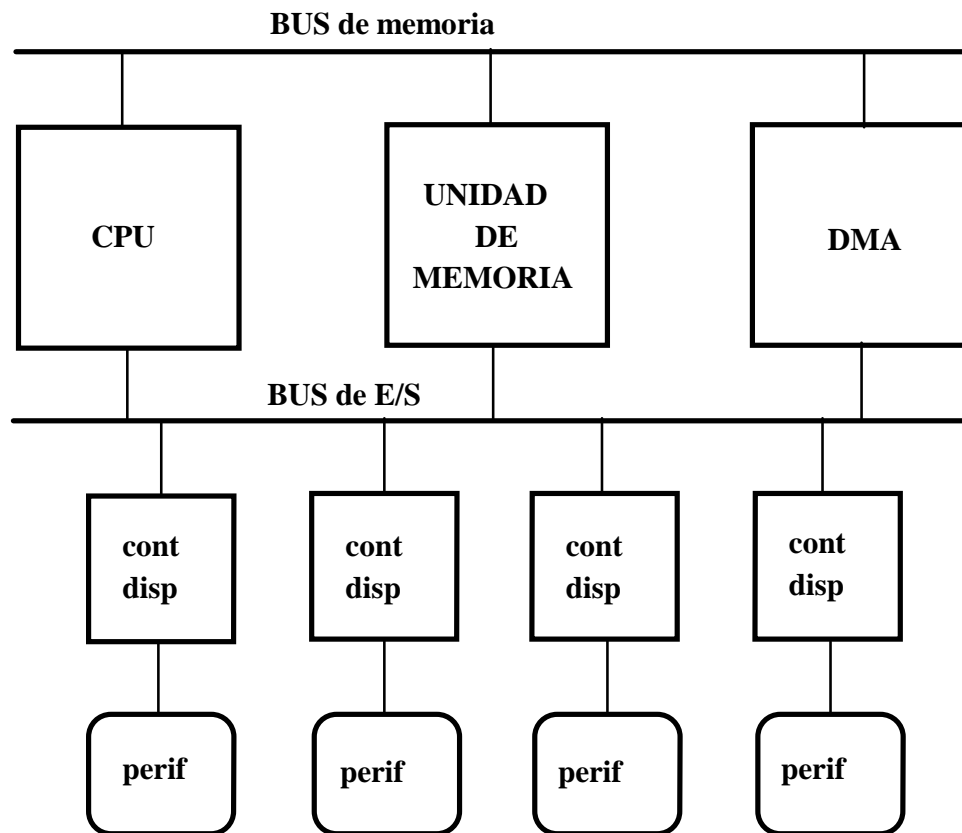


Figura 3.3. Controlador de acceso directo a memoria selector

La arquitectura mostrada en la figura 3.3 también puede ser usada para un controlador de acceso directo a memoria del tipo compartido y, en este caso, el controlador de acceso directo a memoria podría controlar varios dispositivos simultáneamente.

Un sistema computacional puede incluir controladores de los tres tipos y usar configuraciones muy variadas. En este capítulo solamente se presentan los conceptos básicos que permitirán entender el funcionamiento de arquitecturas más elaboradas.

3.5.2. Canales.

Un canal es un dispositivo similar a un controlador de acceso directo a memoria, pero con mayor capacidad para controlar las operaciones de entrada y salida. En el caso de un controlador de acceso directo a memoria, la unidad central de proceso necesita inicializar el controlador del dispositivo y luego indicar al controlador de acceso directo a memoria que se encargue de la transferencia de información. Estos procesadores de entrada y salida se utilizan típicamente en computadoras personales y minicomputadoras. En equipos principales se usan preferentemente los canales.

Cuando se tiene un canal, el procesador coloca las instrucciones necesarias para que el canal realice la operación de entrada o salida en un lugar de la unidad de memoria. Estas instrucciones se conocen comúnmente como "programa de canal". En este programa se le especifica al canal la dirección del dispositivo periférico, el tipo de operación que se desea realizar, la cantidad de información que se tiene que transferir y la dirección en memoria a partir de la cual se dejarán o tomarán los datos. El procesador da la orden de realizar la operación de entrada o salida al canal, indicándole la dirección de memoria a partir de la cual se encuentra el programa de canal y se dedica a ejecutar otros programas. El canal ejecuta su programa en forma concurrente con el procesador y controla toda la transferencia de información, desde la inicialización del dispositivo, la atención de las interrupciones y la terminación de la operación de entrada o salida. A final de la operación, o antes si se encontrara con algún problema especial, genera una interrupción al procesador, indicándole el estado de la operación.

Esta forma de manejar las operaciones de entrada y salida proporciona mayor independencia al procesador, permitiendo una mejor utilización del mismo.

Al igual que los controladores de acceso directo a memoria, los canales pueden ser canales dedicados, canales selectores o canales compartidos, dependiendo de su forma de operación.

Una posible arquitectura de un sistema computacional con canales de los tres tipos se muestra en la figura 3.4. En esta configuración, todos los dispositivos periféricos son manejados a través de canales. Los dispositivos más lentos, tales como terminales e impresoras generalmente se asignan a canales del tipo compartido. Los canales selectores se reservan para utilizarlos con dispositivos rápidos como discos duros o bien, con dispositivos que se utilicen para transferir grandes volúmenes de información como cintas magnéticas. Los canales dedicados se usan para dispositivos rápidos cuyo uso sea muy intenso como podría ser un disco duro usado para memoria virtual.

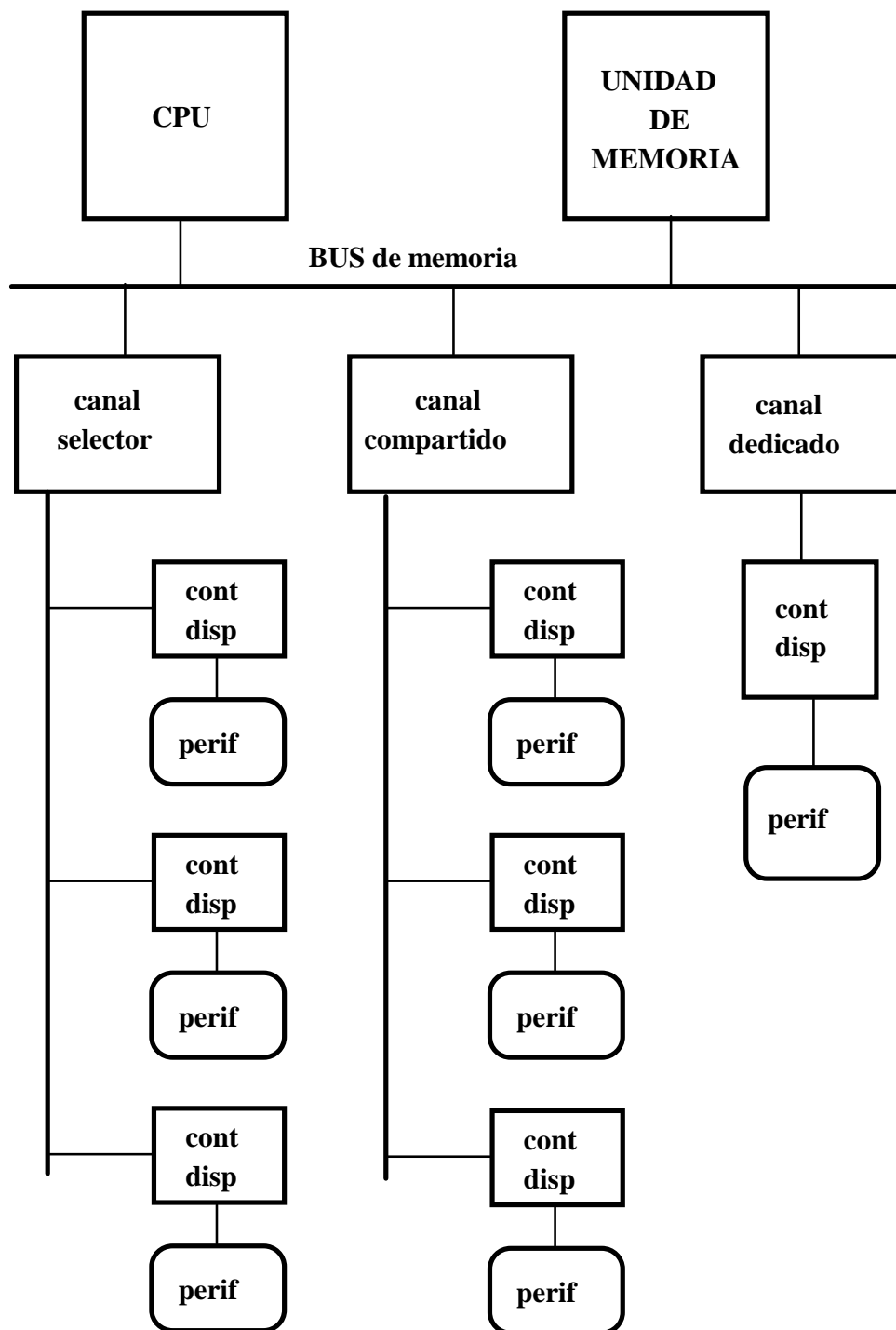


Figura 3.4. Sistema computacional con varios canales

3.6. RESUMEN

En este capítulo se describió la forma en que se realizan las operaciones de entrada y salida. Se explicó primeramente cómo se realizan estas operaciones bajo el control directo de la unidad central de proceso, destacando la ineficiencia de hacerlo de esta manera.

Se introdujo el concepto de interrupciones, cuyo objetivo inicial es liberar al procesador del proceso cíclico de interrogación a un dispositivo periférico para saber cuando terminó la operación que se le solicitó. Se analizó también la forma en que se procesan las interrupciones, añadiendo para ello nuevas micro operaciones e instrucciones de máquina.

Finalmente, se analizó la operación básica de los procesadores de entrada y salida, presentando los controladores de acceso directo a memoria y los canales. Se describieron los tres tipos básicos de procesadores de entrada y salida: procesadores de entrada y salida dedicados, selectores y compartidos. Esta clasificación se realiza de acuerdo al número de dispositivos que pueden atender, así como el número de dispositivos con que pueden trabajar simultáneamente.

Existen muchas variantes en los procesadores de entrada y salida; sin embargo, en este capítulo se presentaron solamente los modelos básicos, los cuales permiten entender el funcionamiento de las demás configuraciones.

3.7. PROBLEMAS

1. Explique como operan las dos filosofías de entrada y salida cuando la unidad central de proceso maneja directamente los dispositivos.
2. Defina que es un procesador de entrada y salida.
3. Explique para que sirven las interrupciones en una computadora digital,
4. ¿Bajo qué condiciones se acepta una interrupción de entrada y salida?
5. Describa para que se utiliza y como trabaja la instrucción INTA de esta computadora.
6. Describa lo que ocurre en forma automática cuando se presenta una interrupción de entrada y salida y la computadora tiene habilitadas las interrupciones.
7. Explique la operación de la instrucción RTI.
8. Describa la operación de un controlador de acceso directo a memoria.
9. De acuerdo a lo presentado en este capítulo, describa que es un canal.
10. ¿Cuáles son los tipos de canales que existen?
11. Explique la operación de cada uno de los res tipos de canales descritos en este capítulo.

12. Menciones las principales diferencias entre un controlador de acceso directo a memoria y un canal.
13. Explique para que sirven los registros de estado y control en un dispositivo periférico.
14. ¿Cuándo puede interrumpir un dispositivo periférico?
15. ¿A cuáles dispositivos se les debe dar la mayor prioridad de interrupción en un sistema computacional?

CAPITULO 4

DISPOSITIVOS PERIFERICOS

Los dispositivos periféricos interactúan con las computadoras digitales a través de la unidad de entrada y salida. Estos dispositivos son de naturaleza muy variada y cumplen funciones muy diversas.

Las cintas magnéticas y los discos magnéticos se utilizan para almacenamiento de información en forma permanente, permitiendo grabar información en ellos para posteriormente volver a utilizarla. En estos dispositivos la información puede grabarse y borrarse.

En los discos ópticos generalmente se graba la información una sola vez y puede ser utilizada las veces que se desee, pero la información ya no puede ser modificada. Existen algunos discos ópticos en los cuales si puede grabarse información y borrarse posteriormente para grabar nueva información; sin embargo, su uso todavía no está muy extendido.

Otros dispositivos periféricos se utilizan principalmente para alimentar información a las computadoras digitales siendo los más comunes el teclado, el ratón y los digitalizadores. Existen también ciertos dispositivos que permiten alimentar información a las computadoras digitales que reciben el nombre genérico de sensores. Estos se utilizan principalmente cuando la computadora realiza funciones de control de procesos.

Entre los dispositivos periféricos que se utilizan para visualizar resultados generados por las computadoras digitales están las pantallas de video, las impresoras y los graficadores. Las salidas de las computadoras también pueden ser dirigidas a actuadores para controlar algún proceso.

4.1. CINTAS MAGNETICAS

Las cintas magnéticas son medios que permiten almacenar información en forma permanente. Las cintas magnéticas están hechas de un material plástico cubierto con una película de material ferromagnético, que puede ser magnetizado en forma permanente. Los dispositivos que graban y leen información en cintas magnéticas tienen una cabeza lectora y una cabeza grabadora, similares a las que tiene una grabadora de audio.

Para grabar información, se pasa corriente por la cabeza grabadora lo cual genera un campo magnético que ocasiona que el material ferromagnético de la cinta se magnetice en forma permanente cuando la cinta pasa junto a ella. El sentido de la magnetización depende del sentido en que se pase la corriente en la bobina de la cabeza grabadora. Al grabar información en la cinta lo que realmente se graban son pequeños imanes con diferentes polaridades en la película ferromagnética de la cinta.

Para leer la información grabada en una cinta, se pasa la cinta junto a la cabeza lectora. Los pequeños imanes que contiene la cinta inducen corriente en la bobina de la cabeza lectora. El sentido de la corriente inducida depende de la polaridad del imán que está pasando junto a la cabeza lectora.

Existen diversos formatos utilizados para grabar información binaria en dispositivos magnéticos. Uno de los formatos más simples consiste en mantener una corriente $+I$ en la bobina de la cabeza grabadora durante un tiempo fijo, denominado tiempo de bit, para grabar un uno. Para grabar un cero, se mantiene una corriente $-I$ en la bobina de la cabeza grabadora durante el tiempo de bit. En este formato, la información quede grabada como pequeños imanes con diferente polaridad, dependiendo de la corriente que se mantuvo en la bobina de la cabeza grabadora.

Al pasar la cinta por la cabeza lectora, estos imanes inducirán una corriente $+I$ o $-I$ en la bobina de la cabeza, dependiendo de su polaridad. Con este procedimiento se puede leer la información que se grabó en la cinta.

Las unidades de cinta magnética tienen típicamente nueve cabezas grabadoras y nueve cabezas lectoras. La información es grabada por bytes, añadiendo un bit de paridad a cada byte. Cada byte es grabado en paralelo y los diferentes bytes son grabados en forma secuencial tal como se muestra en la figura 4.1.

El bit de paridad se añade como un noveno bit a cada byte para verificación de errores. Si se está utilizando paridad par, el noveno bit se pondría en cero si el número de bits que son uno en el byte es par. Si el número de bits que son uno en el byte es impar, este noveno bit se pondría en uno. En caso de usar paridad impar, el bit de paridad se pondría en cero o en uno para que el número total de bits que son uno, incluyendo el de paridad, fuera impar.

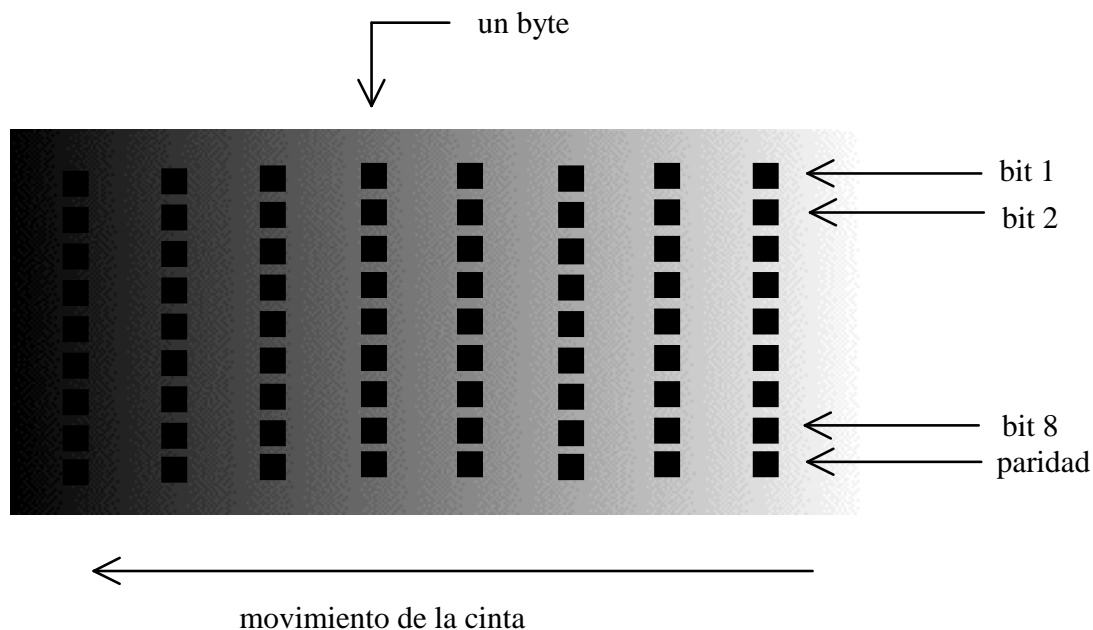


Figura 4.1

Este bit de paridad realmente no es parte de la información que se graba en la cinta. Su uso es solamente para verificar errores. Al leer la información grabada en la cinta, se verificaría la paridad de cada byte leído, incluyendo el bit de paridad, y se desecharía este bit una vez que se ha verificado la paridad del byte leído.

Debido a que los bytes de información son grabados en forma secuencial en las cintas magnéticas, estos dispositivos no son usadas actualmente para almacenar información a la que se necesita tener acceso en forma rápida. El principal uso de las cintas magnéticas es guardar información de respaldo. Por ejemplo, en un centro de cómputo típico se realizan respaldos a cinta magnética de la información contenida en los discos magnéticos una vez por semana. De esta forma, si la información contenida en los discos magnéticos llegara a dañarse, se copiaría la información del último respaldo en cinta magnética y se continuaría la operación del sistema computacional desde este punto.

Las cintas magnéticas permiten almacenar una gran cantidad de información. Las densidades de grabación actuales van de 1,600 bytes por pulgada en baja densidad, a 6,250 bytes por pulgada en alta densidad. Una cinta de alta densidad de 1,000 pies de longitud puede almacenar 75 megabytes de información.

4.2. DISCOS MAGNETICOS

Los discos magnéticos son platos circulares rígidos de metal, recubiertos con una película de material ferromagnético que permite grabar información en forma permanente en su superficie, por ambos lados del plato. Existe una cabeza lectora y otra grabadora para cada superficie.

Estas cabezas están unidas a un brazo que permite posicionarlas sobre la superficie del disco mediante movimientos hacia adentro o hacia afuera del disco. Las cabezas están a una distancia que oscila entre diez y veinte milésimas de pulgada de la superficie del plato y nunca deben tocar la superficie.

Una unidad de discos generalmente consta de varios platos paralelos que giran sincrónicamente. El brazo que posiciona las cabezas lectoras y grabadoras mueve todas las cabezas simultáneamente como se ilustra en la figura 4.2.

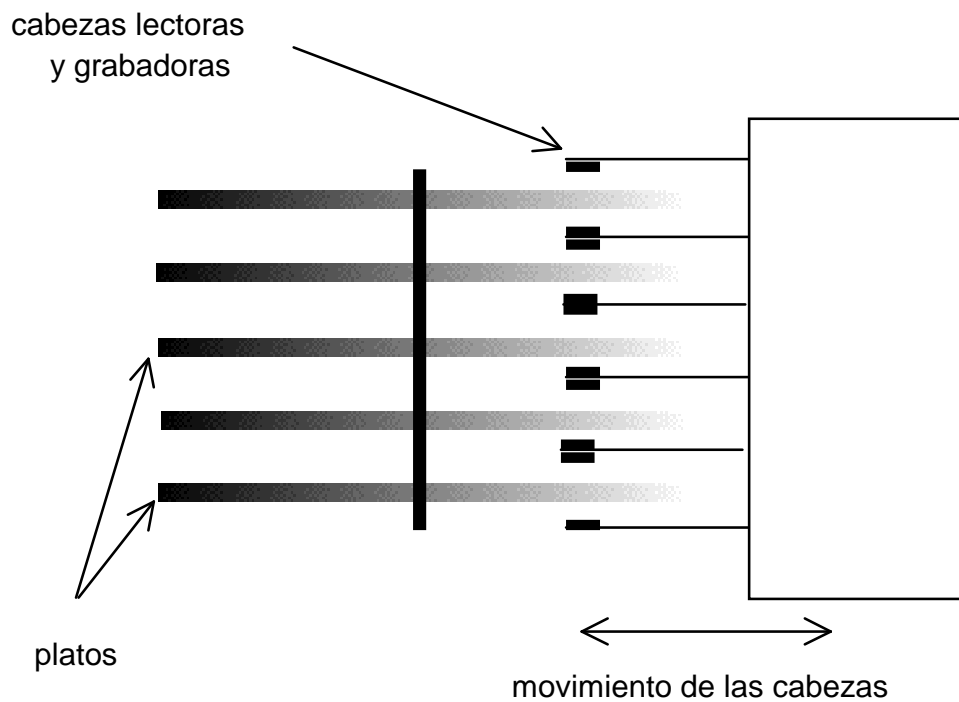


Figura 4.2. Unidad de Discos

La información en cada uno de los platos de una unidad de discos se organiza en segmentos circulares concéntricos denominados pistas. Cada pista está a su vez subdividida en sectores. Para identificar cada una de las superficies de los platos que componen la unidad de discos se utiliza el número de cabeza lectora asociada a dicha superficie. En la figura 4.3 se bosqueja la organización de la información en un plato.

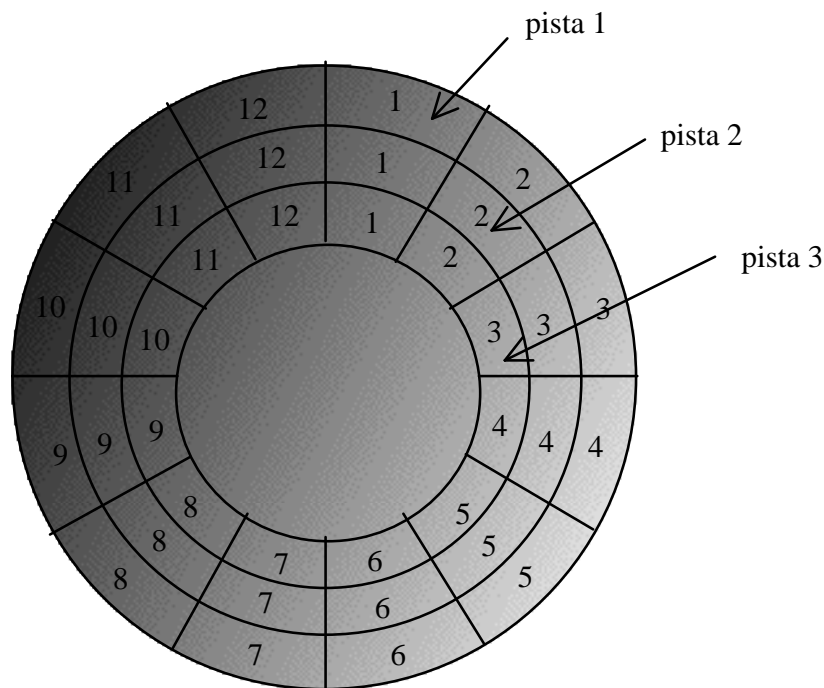


Figura 4.2. Organización de las pistas y los sectores

Al conjunto de todas las pistas con la misma posición relativa en cada plato se le conoce como cilindro. Para identificar un sector en particular en una unidad de discos es necesario especificar la pista (o el cilindro), el sector y el número de la cabeza lectora o grabadora.

En una unidad de discos solamente una cabeza está activa a la vez. La información se graba en forma secuencial a nivel bit. Se puede leer o grabar solamente sectores completos en una operación. No se puede grabar fragmentos de sectores. Si se desea modificar parcialmente la información en un sector, es necesario leer el sector completo a memoria, modificar lo que se desee cambiar, y grabar de nuevo el sector completo ya modificado.

Para leer o grabar información en un sector en particular en un unidad de discos es necesario primeramente posicionar las cabezas sobre la pista o cilindro en el cual se encuentra el sector. Esto se realiza mediante un movimiento (seek) del brazo que posiciona las cabezas. Luego se necesita seleccionar la cabeza que va a leer o grabar la información en la superficie adecuada. Por último, se tiene que esperar a que el sector deseado pase por donde está la cabeza para realizar la lectura o escritura de la información en ese sector.

Cada sector en la unidad de discos contiene información de identificación propia del sector y la información de datos grabada en él. tal como se muestra en la figura 4.4.

El campo que tiene la identificación del sector, contiene un patrón de bits para sincronización, el número de cilindro, el número de cabeza y el número de sector así como un campo para verificación de errores. Este último campo se conoce como CRC (Cyclic Redundancy Check) y

se calcula en base a los bits de la identificación de la cabeza, el cilindro y el sector. Cuando se graba la información de identificación en un sector, se calcula el CRC y se graba también en el disco. Al leer la información de identificación, se calcula de nuevo el CRC y se compara con el que está grabado en el disco. Si son iguales, la información se toma como buena. Si son diferentes, se marca un error.

Identificación del sector	Campo de datos
------------------------------	----------------

Formato de un sector

sincronización	cilindro	cabeza	sector	CRC
----------------	----------	--------	--------	-----

Campo de identificación del sector

sincronización	datos	CRC
----------------	-------	-----

Campo de datos del sector

Figura 4.4. Organización de un sector

El campo que contiene los datos grabados en el sector tiene a su vez tres campos. El primero tiene un patrón de sincronización, seguido del campo de datos y finalmente se encuentra otro CRC para detectar errores en los datos. Este CRC se calcula en base a los bits de los datos y se utiliza en forma similar al CRC del campo de identificación del sector.

Los discos flexibles, comúnmente conocidos como diskettes, son medios de almacenamiento permanente similares a los discos magnéticos. Estos discos tienen un plato circular de material plástico muy delgado y flexible, recubierto por una capa de material ferromagnético. Este plato viene en una cubierta de plástico que tiene una abertura para que las cabezas lectoras y grabadoras tengan acceso a la información en el medio. Antiguamente los discos flexibles venían en cubiertas de ocho pulgadas, posteriormente se fabricaron en cubiertas de cinco y un cuarto de pulgadas y actualmente vienen en cubiertas de tres y media pulgadas.

En estos discos generalmente se puede grabar información por ambos lados del plato circular, aunque los primeros discos flexibles solamente estaban preparados para grabarse por un solo lado. La información en el disco flexible se organiza en pistas, en forma similar a como se encuentra en los discos duros. Cada pista está dividida en sectores y éstos tienen un formato

similar al de los discos duros. La densidad de grabación en los discos flexibles es menor que en los discos duros.

Las cabezas en los discos flexibles si tocan la superficie magnética y eventualmente esta superficie se desgasta, haciendo que el disco flexible ya no sea útil.

Tanto en los discos duros como en los discos flexibles, los sistemas operativos crean una estructura de datos en la cual se mantiene la información referente a los diferentes archivos que existen en los discos. Esta estructura de datos es el directorio del disco. Para cada archivo creado en el disco, se mantiene en el directorio datos referentes al mismo tales como el nombre del archivo, la fecha de creación, la fecha de la última actualización del archivo, el nombre del usuario que es dueño del archivo, los permisos de acceso al archivo, el sector del disco donde inicia el archivo, etc. La información exacta que se mantiene para cada archivo cambia de un sistema operativo a otro.

4.3. DISCOS OPTICOS

Los discos ópticos mas comúnmente usados son dispositivos en los cuales se puede grabar información solamente una vez, pero puede ser leída cuantas veces se necesite. Estos discos son similares a los discos compactos utilizados en los equipos de audio. Actualmente hay discos ópticos en los cuales puede grabarse información, borrarse y volverse a grabar, pero no son los más comúnmente utilizados.

La información en los discos ópticos también está organizada en pistas y sectores, en forma análoga a como se encuentra en los discos magnéticos; sin embargo, las pistas no son concéntricas como en los discos duros. La información está organizada sobre una espiral desde la parte externa del disco hasta la parte interna.

La forma en que se graba y se lee la información también es diferente a como se realiza en los discos magnéticos. En los discos ópticos, la lectura de la información se realiza a velocidad lineal constante. Esto implica que la velocidad angular del disco óptico varía de acuerdo al lugar donde se está leyendo la información. La velocidad lineal o tangencial es igual al producto de la velocidad angular por el radio, donde el radio es la distancia del centro del disco al lugar donde se está leyendo la información. De acuerdo con esto, la velocidad angular del disco óptico es inversamente proporcional al radio para mantener la velocidad lineal constante.

Los discos ópticos tienen un material plástico que es capaz de reflejar un rayo láser de baja potencia. Para grabar información se utiliza un rayo láser de mayor potencia para quemar la película reflejante donde se desean grabar ceros, dejándola intacta donde se desean grabar unos. Para leer estos discos se utiliza el rayo láser de baja potencia, el cual es reflejado donde se dejó intacta la película reflejante y no es reflejado donde se quemó la película. Si se detecta la reflexión del rayo láser, se interpreta como un uno, si el rayo no es reflejado, se interpreta como un cero. Estos discos ópticos se conocen comúnmente como CD-ROMs, que significa Compact Disc Read-Only Memory. Los discos ópticos vírgenes, en los cuales se puede grabar la

información solamente una vez, se conocen comúnmente como WORMs, que significa Write Once, Read Many.

Los discos ópticos tienen una gran capacidad de almacenamiento. Un disco óptico de cuatro pulgadas y tres cuartos puede almacenar 600 Megabytes de información.

Recientemente se han construido discos magneto-ópticos en los cuales puede grabarse y borrarse la información. En estos dispositivos, se utiliza una película magnética, la cual puede ser magnetizada en dos direcciones mediante la aplicación de un campo magnético y el calor de un rayo láser. Para leer la información, se utiliza un rayo láser polarizado, que es reflejado en diferente forma, dependiendo de la magnetización de la película.

4.4. TECLADOS

Probablemente el dispositivo más conocido para alimentar datos e instrucciones a los sistemas computacionales es el teclado. Los teclados para computadoras tienen un arreglo de teclas similar al que tienen las máquinas de escribir, pero poseen algunas teclas adicionales que se usan para ciertas funciones específicas.

Cuando se presiona una tecla, se envía una serie de pulsos eléctricos a la unidad de entrada y salida a la que está conectado el teclado. Esta secuencia de pulsos representa el código de la tecla que se presionó y se traduce al carácter o comando apropiado al enviarse a la memoria de la computadora digital.

4.5. RATONES

Los ratones son dispositivos de entrada que tienen una pequeña bola en la parte de abajo y uno, dos o tres botones en la parte superior. Existen sensores que detectan el movimiento de la bola inferior cuando se mueve el ratón sobre una superficie plana. Este movimiento se traduce a un movimiento del cursor en la pantalla y permite apuntar a diferentes parte de la misma. El cursor generalmente se muestra como una pequeña flecha o como una pequeña línea parpadeante en la pantalla e indica el lugar en que está posicionado dicho cursor. Los botones, al oprimirse, sirven para seleccionar aquello a lo que se está apuntando con el curso, o especificar acciones que se desean realizar.

Los ratones permiten seleccionar información de una manera rápida, o bien alimentar comandos a la computadora cuando se selecciona algún comando de un menú de la pantalla.

4.6. DIGITALIZADORES OPTICOS

Existen dispositivos que transforman la luz reflejada por un documento en señales eléctricas, en forma similar a como que hace una copiadora. Estas señales eléctricas son alimentadas a la computadora digital para su posterior interpretación mediante programas especializados que les

asignan cierto significado. El nombre con el que comúnmente se conocen estos dispositivos es "scanners".

Existen digitalizadores ópticos especiales para leer códigos de barras, como los que comúnmente se encuentran en los comercios para leer los códigos de las mercancías.

Otro tipo de digitalizadores se utiliza para alimentar imágenes o fotografías a los sistemas computacionales. En estas aplicaciones, se guardan puntos de la imagen, denominados píxeles. Cada punto de la imagen tiene asociado un color y una luminosidad, lo cual permite reconstruir la imagen en una pantalla de video. La resolución del digitalizador determina cuantos puntos por pulgada se generan.

Otras aplicaciones permiten digitalizar texto y generar archivos en la computadora con el texto que se digitalizó; o bien, interpretar el texto leído para guardarlo o procesarlo.

4.7. PANTALLAS DE VIDEO

Entre los dispositivos de salida más comúnmente utilizados en la actualidad están las pantallas de vídeo. Existen pantallas de vídeo monocromáticas y con colores. Las pantallas de vídeo más comúnmente utilizadas se basan en la tecnología de tubo de rayos catódicos (CRT), siendo su operación muy similar a la de los aparatos de televisión.

Las pantallas de vídeo monocromáticas con tecnología de tubo de rayos catódicos constan de un tubo al vacío que tiene en su parte frontal una película interior de fósforo y en su parte posterior un cañón que emite un haz de electrones a alta velocidad. La parte frontal es donde se forma la imagen generada cuando el haz de electrones excita los átomos de fósforo y los hace emitir luz. A la salida del cañón se localiza un sistema de deflexión del haz de electrones que sirve para dirigirlo a un punto específico de la pantalla. Se puede regular la intensidad del haz de electrones para modular la luminosidad de la película de fósforo cuando este haz incide sobre ella. En la figura 4.5. se muestra esquemáticamente este arreglo de componentes.

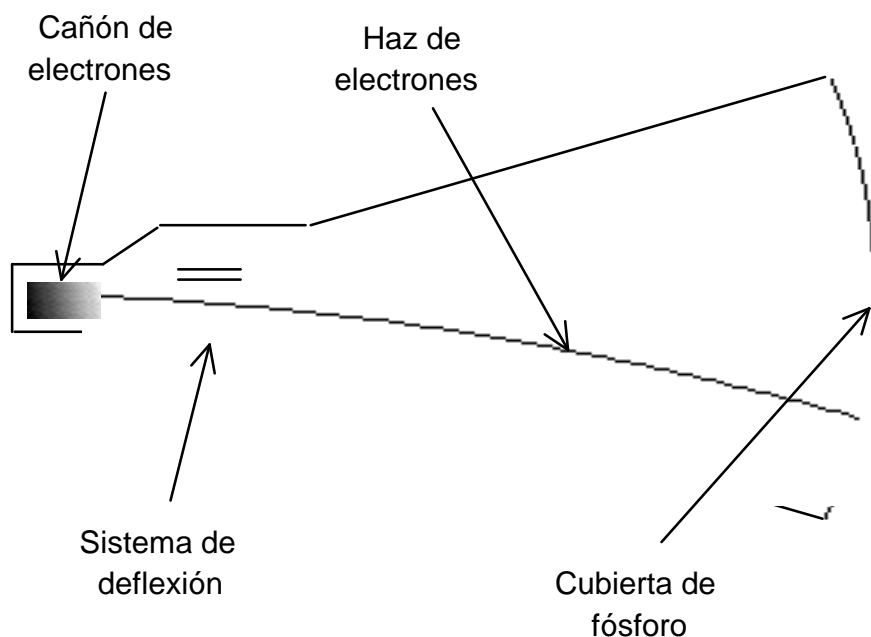


Figura 4.5. Pantalla de rayos catódicos

Para formar una imagen, el sistema de deflexión barre la pantalla al mismo tiempo que se regula la intensidad del haz de electrones, haciendo que se iluminen los diferentes puntos de la pantalla con la intensidad adecuada. Cada uno de estos puntos se denomina pixel y la resolución de la pantalla se mide por el número de pixeles que contiene en forma vertical y horizontal. Es necesario barrer la pantalla completa cuando menos 30 veces por segundo para que no se note un parpadeo de la imagen.

El controlador del dispositivo se encarga de traducir los datos que le envía el procesador a la pantalla de vídeo para controlar el haz de electrones y generar la imagen deseada.

En las pantallas de vídeo a colores, existen tres clases de fósforo en cada uno de los puntos o pixeles de la pantalla. Uno de ellos emite luz de color rojo, otro verde y el otro azul. Cuando se excitan los átomos de fósforo de cada una de las clases con la intensidad adecuada, se tiene una combinación de los colores rojo, verde y azul con diferente intensidad. Esto permite generar cualquier color que se desee en ese punto. En estas pantallas existen tres cañones de electrones así como tres sistemas de deflexión independientes para excitar los átomos de fósforo que emiten luz roja, verde y azul respectivamente. Es común denominar a estas pantallas como "monitores RGB" por la iniciales de Red, Green y Blue.

Las pantallas de tubo de rayos catódicos son voluminosas, pesadas y consumen mucha energía. Existen también pantallas planas para algunas aplicaciones que no pueden hacerse con las pantallas de tubos de rayos catódicos. Las más comunes son las de cristal líquido (LCD, por Liquid Crystal Display). Estas pantallas tienen dos cristales planos, en medio de los cuales se encuentra el cristal líquido, que es una sustancia con una consistencia aceitosa. El cristal líquido tiene la propiedad de orientar sus moléculas bajo la influencia de un campo eléctrico.

Con cierta orientación de las moléculas se permite el paso de la luz, mientras que otra orientación bloquea el paso de la luz, permitiendo formar imágenes en la pantalla. El campo eléctrico es generado y controlado en cada pixel de la pantalla. Para formar una imagen, se necesita controlar el campo eléctrico en cada uno de los pixeles de la pantalla. La mayoría de las calculadoras y los relojes digitales de pulso tienen pantallas de cristal líquido.

Otro tipo de pantallas planas tienen un gas inerte en medio de las placas de cristal planas. Este gas produce luminosidad cuando se ioniza mediante un campo eléctrico. Controlando el campo eléctrico en cada uno de los pixeles se generan las imágenes deseadas en la pantalla.

Las terminales de vídeo más simples incluyen una pantalla de vídeo y un teclado. Estas terminales no tienen capacidad de procesamiento local.

4.8. IMPRESORAS

Las impresoras son los dispositivos que nos permiten obtener información de las computadoras en forma impresa. Existen varios tipos de impresoras siendo las más comúnmente utilizadas las de impacto, matriz de puntos, de inyección de tinta y las impresoras láser.

Las impresoras de impacto de baja velocidad funcionan de manera similar a las máquinas de escribir con mecanismo de margarita. Estas impresoras se utilizan solamente para imprimir textos. Su funcionamiento se basa en un disco ranurado, denominado margarita, que tiene en el extremo de cada una de sus lengüetas uno de los diferentes caracteres que pueden ser impresos. El carácter que se desea imprimir se selecciona girando el disco de tal forma que dicho carácter quede enfrente de un martillo accionado por un electroimán. Una vez que dicho carácter se encuentra en la posición deseada, se dispara el martillo que presiona el carácter contra una cinta entintada y contra el papel, de manera similar a como se realiza en una máquina de escribir de este tipo.

Las impresoras de impacto de alta velocidad también son utilizadas solamente para imprimir texto. Su mecanismo se basa en una cadena sinfín que tiene resaltados los diferentes caracteres que se pueden imprimir y un martillo accionado por un electroimán por cada posición de impresión. La cadena gira constantemente y cada martillo se dispara cuando el carácter que se desea imprimir en esa posición pasa enfrente de él.

Las impresoras de matriz de puntos también son impresoras de impacto, pero son más versátiles que las de margarita y las de cadena ya que también pueden imprimir gráficas de baja resolución. Su mecanismo cuenta con una cabeza móvil con agujas accionadas por electroimanes. Estas agujas se encuentran dispuestas en una línea vertical y típicamente son nueve. La cabeza se mueve frente al papel en forma similar a como se mueve la cabeza de las máquinas de escribir eléctricas. Al irse moviendo la cabeza, se van disparando las agujas que presionan la cinta entintada contra el papel y de esta manera se puede imprimir textos o gráficas. La resolución típica de estas impresoras es de 72 puntos por pulgada.

Las impresoras de inyección de tinta operan en forma similar a las de matriz de puntos pero en lugar de tener agujas en la cabeza móvil, tienen inyectoros de tinta. Al irse desplazando la cabeza se van accionando los inyectoros en donde se desee imprimir los puntos y así se va formando la imagen en el papel. Estas impresoras pueden imprimir texto y gráficas de mejor resolución que las de matriz de puntos.

Las impresoras láser trabajan de manera similar a las copiadoras. La unidad de control de la impresora enciende y apaga un rayo láser mientras se va barriendo la superficie del tambor de impresión. En los puntos donde incide el rayo láser se adhiere el polvo de impresión al tambor. Cuando la hoja de papel es pasada sobre el tambor, se imprimen los puntos donde incidió el rayo láser y se depositó el polvo de impresión. Estas impresoras han tenido cada vez mayor aceptación y generan una impresión de muy alta calidad. Pueden imprimir tanto gráficas de alta resolución como texto.

4.9. GRAFICADORES

Los graficadores son dispositivos que producen dibujos y gráficas de alta calidad. En los graficadores pequeños, el papel está fijo sobre una superficie plana. Una barra mecánica horizontal se mueve de izquierda a derecha y de derecha a izquierda sobre la superficie donde está el papel. Sobre esta barra mecánica horizontal está montada una pluma que a su vez tiene movimiento sobre la barra y puede subirse para que no dibuje sobre el papel al moverse, o bajarse para que dibuje mientras se mueve. En la figura 4.6 Se muestra este arreglo en forma esquemática.

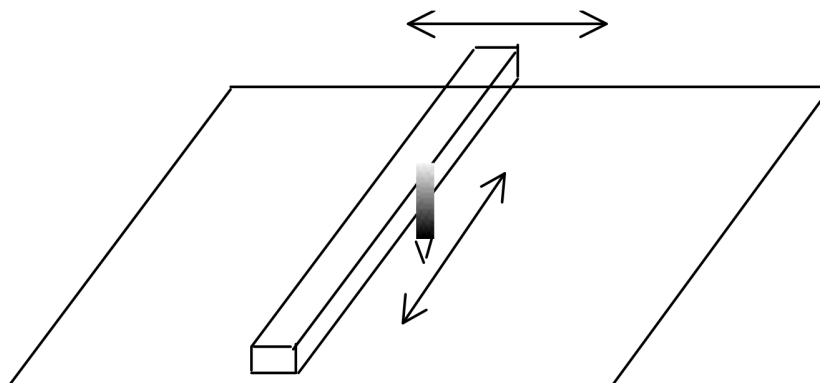


Figura 4.6. Componentes de un graficador.

En los graficadores grandes se tiene una barra horizontal fija sobre la cual se desplaza la pluma y el papel se monta sobre un tambor que gira hacia adelante y hacia atrás para moverlo. La pluma puede también subirse para que no dibuje o bajarse para que si lo haga. En ambos tipos de graficadores las plumas pueden ser de tinta o electrostáticas.

4.10. SENSORES Y ACTUADORES.

En las aplicaciones de control de procesos, los sensores hacen posible que las computadoras digitales puedan conocer el estado de los sistemas que está controlando. Existen dos formas básicas de señales que se pueden alimentar a una computadora digital mediante sensores: señales digitales y señales analógicas.

Las señales digitales solamente indican uno de dos posibles estados. Por ejemplo, pueden indicar si una puerta está abierta o está cerrada, si el nivel de un líquido en un recipiente está por encima o por debajo de un nivel de referencia, si una temperatura está por debajo o por encima de un cierto valor, etc. La computadora digital solamente necesita leer un bit para indicar uno de los dos posibles estados, indicados por el cero o el uno binarios. Los sensores utilizados son generalmente dispositivos mecánicos, eléctricos, electromecánicos o térmicos como los utilizados en la mayoría de los aparatos que operan encendidos o apagados.

Como ejemplo considérese el control de una unidad de aire acondicionado. El termostato que regula la operación del equipo simplemente envía la orden de encender el compresor cuando la temperatura sube de un cierto valor fijado por el usuario, o apagar el compresor cuando la temperatura desciende de otro valor. En el caso de que esta señal se necesitara alimentar a una computadora digital, se conectaría dicha señal a un punto en un módulo de entradas digitales. La computadora solamente leería el estado de dicho punto de entrada, donde un cero podría indicar que la temperatura está por debajo de un cierto valor y un uno indicaría que está por encima de dicho valor. El módulo de entradas digitales es visto por la computadora como un dispositivo periférico del cual puede leer datos digitales.

Las señales digitales indican el valor de una variable externa a la computadora con un cierto grado de precisión. Si en una aplicación particular se necesitara que la computadora pudiera conocer el valor de la temperatura ambiente y no solamente si está por encima o por debajo de un cierto valor de referencia, se usaría una entrada analógica. En este caso podría utilizarse un termopar que es un dispositivo que genera un voltaje entre sus terminales, proporcional a la temperatura a la que está sujeto. La señal generada por el termopar es una señal analógica ya que varía continuamente en el tiempo y puede tomar un número infinito de valores dentro de un cierto rango. Para que la computadora pudiera leer esta señal, es necesario alimentar la señal a un convertidor análogo-digital. El convertidor análogo digital es un dispositivo periférico que toma una muestra del valor de la señal analógica y la convierte en un valor digital, representada en binario (ceros y unos). Cuando la computadora digital lee el dato del convertidor, realmente recibe un valor digital proporcional a la temperatura a la que está sujeto el termopar.

Existen muchos dispositivos que pueden ser utilizados para alimentar señales analógicas a las computadoras digitales, pero siempre será a través de un convertidor análogo-digital. Pueden alimentarse señales provenientes de un micrófono, un equipo de audio, un equipo de video, una resistencia variable y, en general, cualquier señal eléctrica.

Cuando se requiere que una computadora digital tome una acción sobre el sistema que está controlando, se utiliza un actuador. En el caso de que la computadora digital sea la encargada de encender o apagar el compresor de un sistema de aire acondicionado, lo que tiene que hacer es enviar la señal para prenderlo o apagarlo tal y como la enviaría el termostato. En este caso se tiene una salida digital en la computadora. Las salidas digitales se utilizan cuando se desea tener

el sistema en uno de dos estados posibles como en el ejemplo anterior. En este caso, se tendría el compresor del equipo de aire acondicionado encendido, o apagado.

En otras ocasiones es necesario que la salida proveniente de la computadora digital sea una señal analógica. En este caso se utilizaría un convertidor digital-análogo. Los convertidores digital-análogos son dispositivos periféricos que reciben un valor digital de la computadora (ceros y unos) y lo convierten a una señal eléctrica, generalmente voltaje. Esta señal se alimenta ya sea directamente al sistema que se desea controlar, o a un amplificador que le de la potencia necesaria para realizar la labor requerida. El actuador final generalmente es un equipo electromecánico que recibe esta señal para controlar alguna variable del proceso. Por ejemplo, la señal analógica proveniente del convertidor digital-análogo puede ser alimentada a un amplificador conectado a una servoválvula para regular el flujo de gas a una turbina. La servoválvula deja pasar más o menos flujo de gas dependiendo del voltaje que se le aplique en su entrada de control.

4.11. RESUMEN

En este capítulo se describieron brevemente los principales dispositivos periféricos que se utilizan con las computadoras digitales tales como cintas magnéticas, discos magnéticos, discos ópticos, teclados, ratones, pantallas de video, impresoras, graficadores, sensores y actuadores. También se tocaron someramente los principios básicos en los cuales basan su operación.

Existen además otros dispositivos periféricos que no se incluyeron en este capítulo. Algunos de estos periféricos ya quedaron en desuso mientras que otros se utilizan en aplicaciones más especializadas como multimedios y realidad virtual.

Cada día se desarrollarán más dispositivos periféricos o se mejorarán algunos de los ya existentes. Otros quedarán en desuso por volverse obsoletos como los antiguos teletipos que se usaron en las décadas de los sesentas y setentas.

4.11. PROBLEMAS

1. Para grabar información digital en medios magnéticos se han utilizado diferentes formatos. Investigue cómo graban información los siguientes formatos:
 - (a) RZ (Return to Zero)
 - (b) NRZ (Non Return to Zero)
 - (c) NRZ-I (Non Return to Zero, Invert on ones)
 - (d) Manchester
 - (e) Manchester diferencial
2. Describa las principales ventajas y desventajas de los formatos de grabación mencionados en el problema 1.

3. Cuando se transmite información digital sobre líneas telefónicas se utilizan dispositivos denominados módems. Investigue que funciones realiza uno de estos dispositivos.
4. Las técnicas básicas de modulación utilizadas en los módems se conocen como:
 - ASK (Amplitude Shift Keying)
 - FSK (Frequency Shift Keying)
 - PSK (Phase Shift Keying)Investigue en qué consiste cada una de estas técnicas.
5. En la transmisión de información en redes locales también se utilizan formatos digitales similares a los utilizados para grabar información en medios magnéticos. Investigue cuáles formatos se utilizan en las redes locales con estándares Ethernet y Token Ring.
6. Para aplicaciones de control de procesos se utilizan convertidores análogo-digitales (ADC - Analog to Digital Converters) para obtener información de las variables de los procesos. Investigue que funciones realizan estos dispositivos.
7. Los convertidores digital-análogos (DAC - Digital to Analog Converters) se utilizan también en aplicaciones de control de procesos para dar información a los actuadores que realizan la acción de control en los procesos. Investigue las funciones que realizan estos dispositivos.

CAPITULO 5

LENGUAJES, COMPILADORES E INTERPRETADORES

5.1. Tipos de lenguajes

Las primeras computadoras se programaban en lenguaje maquinal a través de la consola. Los programas eran codificados directamente con ceros y unos, lo cual hacía que la programación fuera tediosa y era muy fácil que se cometieran errores. Posteriormente se desarrolló el lenguaje ensamblador. En este lenguaje, las instrucciones de máquina se codificaban con mnemónicos y direcciones simbólicas, lo que hacía más fácil la tarea de programación. Además, un programa hecho por una persona podía ser entendido y modificado por otra persona. El único lenguaje que puede ejecutar la computadora es el lenguaje maquinal, por lo cual, era necesario traducir este programa en lenguaje ensamblador al lenguaje maquinal. Para realizar esta tarea de traducción se desarrollaron programas que hacían la traducción del lenguaje ensamblador al lenguaje maquinal, substituyendo los mnemónicos por sus correspondientes patrones de ceros y unos y asignando direcciones físicas a las direcciones lógicas del programa en lenguaje ensamblador. Estos programas se conocen como ensambladores.

A finales de la década de los cincuentas se crearon los primeros lenguajes de alto nivel, también llamados lenguajes de tercera generación. En esta década surgió el FORTRAN (su nombre viene de FORMula TRANslation), que es un lenguaje orientado a desarrollar aplicaciones científicas e ingenieriles. En la siguiente década se desarrolló el lenguaje COBOL (su nombre viene de COMmon Business Oriented Language) cuyo propósito primordial era desarrollar aplicaciones administrativas. Posteriormente se desarrollaron otros lenguajes de programación de tercera generación como ALGOL, LISP, PL1, BASIC, APL, PASCAL, ADA, PROLOG, C, C++, etc.

Los esfuerzos iniciales en el área de lenguajes de programación se enfocaron a desarrollar lenguajes cada vez más potentes y con un mayor número de instrucciones; sin embargo, a finales de la década de los setentas esta tendencia se cambió y se trataron de desarrollar lenguajes con un juego reducido de instrucciones, pero bien estructurado, de tal forma que se pudieran desarrollar programas en forma estructurada, fáciles de entender y modificar.

Finalmente, a partir de la década de los ochentas se inició el desarrollo de los lenguajes de cuarta generación como Nomad, Focus, Passport, Oracle, dBASE, PARADOX, FoxPro, Clipper, SYMPHONY y DELPHI entre otros.

5.1.1. Lenguaje Maquinal

Todas las computadoras digitales trabajan internamente con el sistema binario. En los capítulos anteriores se utilizó una computadora hipotética que trabaja en sistema decimal. Esta máquina se creó solamente para presentar los conceptos básicos de la organización computacional sin

tener que estudiar el sistema de números binario para manejar tanto los datos como las instrucciones y direcciones en este sistema.

En el sistema binario solamente existen los dígitos binarios cero y uno. Todos los registros de la computadora pueden contener un cierto número fijo de dígitos binarios, conocidos comúnmente como bits (Binary digit). Todas las instrucciones en la computadora son secuencias de ceros y unos. Una parte de estas secuencias corresponde al código de operación, otra parte al tipo de direccionamiento y otra parte a la dirección. Las direcciones y los datos también están representados en el sistema binario. Cuando se analiza el contenido de una cierta dirección de memoria, no hay forma de saber si se trata una instrucción o de un dato ya que ambos se representan como secuencias de ceros y unos.

Los programas en lenguaje maquina son secuencias de instrucciones codificadas directamente en binario. Los datos y direcciones que se utilizan también se encuentran en binario. El único lenguaje que es directamente ejecutable en las computadoras digitales es el lenguaje maquina.

En capítulos subsiguientes de este libro se estudiará el sistema de números binario y se describirá cómo se representan los diferentes tipos de datos dentro de la computadora digital, utilizando el sistema binario. También se explicará cómo se realizan las operaciones aritméticas y lógicas en las diferentes representaciones de datos usando este sistema.

Posteriormente se presentará una computadora digital que opera con el sistema binario y se describirán sus diferentes instrucciones de máquina y los tipos de direccionamiento que utiliza.

5.1.2. Lenguaje ensamblador

La programación en el lenguaje maquina es muy tediosa y propensa a cometer errores, dado que en este lenguaje, todas las instrucciones y datos son secuencias de unos y ceros. Entender y modificar un programa en lenguaje maquina requiere de mucho tiempo y esfuerzo.

Para realizar la programación de una manera más sencilla y poder entender y modificar los programas de una manera más eficiente se desarrolló el lenguaje ensamblador. En este lenguaje se asigna un mnemónico a cada una de las diferentes instrucciones de máquina y se utilizan direcciones simbólicas para desarrollar el programa. Los datos generalmente se usan en decimal, aun cuando pueden expresarse en hexadecimal u octal, dependiendo del lenguaje específico que se utilice.

Cada una de las instrucciones del lenguaje ensamblador se traduce a una sola instrucción de máquina. También existen las pseudoinstrucciones, las cuales no son traducidas al lenguaje maquina porque no corresponden a una instrucción de máquina. Estas pseudoinstrucciones son realmente directivas para ser interpretadas por el programa que traduce el código en lenguaje ensamblador a código maquina.

En el capítulo dos de este libro se presentó una breve introducción al lenguaje ensamblador mediante tres ejemplos y se codificaron estos tres programas en lenguaje maquina, aunque para

ello se uso el sistema decimal en lugar del binario. Se utilizaron también algunas pseudoinstrucciones en esa sección

5.1.3. Lenguajes de alto nivel (tercera generación)

Programar en lenguaje ensamblador en lugar de programar en lenguaje maquinal representa un cierto avance en el desarrollo de programas computacionales; sin embargo, es todavía tedioso y susceptible a cometer errores fácilmente. El lenguaje maquinal también se conoce como lenguaje de primera generación y el lenguaje ensamblador como lenguaje de segunda generación.

Tratando de hacer que los programas se desarrollaran en una forma más natural y entendible para los programadores, se crearon los llamados lenguajes de programación de alto nivel, también conocidos como lenguajes de tercera generación. El primer lenguaje de este tipo que se diseñó fue el FORTRAN, cuyo nombre viene de FORMula TRANslation. Este lenguaje, como ya se ha mencionado, estaba orientado principalmente a desarrollar aplicaciones científicas o ingenieriles. Su estructura está basada en la representación de funciones matemáticas y palabras clave que forman las instrucciones del lenguaje.

En los lenguajes de programación de alto nivel se crearon instrucciones que realizaban operaciones más complejas que las instrucciones que puede ejecutar una computadora digital. Para que un programa desarrollado en un lenguaje de alto nivel pueda ser ejecutado en una computadora digital es necesario traducirlo al lenguaje maquinal, que es el único lenguaje que puede ejecutar la computadora digital.

Una instrucción de un lenguaje de alto nivel genera una o varias instrucciones de máquina al traducirse al lenguaje maquinal. Los datos en estos lenguajes se representan como constantes y variables, las cuales también se traducen a datos y direcciones en lenguaje maquinal.

La traducción de un lenguaje de alto nivel al lenguaje maquinal se realiza en forma automática mediante un programa traductor conocido como compilador.

Tanto el lenguaje maquinal como el lenguaje ensamblador son específicos para una máquina en particular. Si se desea desarrollar un mismo programa en dos máquinas diferentes usando cualquiera de estos dos lenguajes, deberá realizarse la programación específica para cada máquina.

En los lenguajes de tercera generación se busca que la programación sea independiente de la arquitectura de la máquina en que se programe. Un mismo programa desarrollado en un lenguaje de tercera generación puede ser compilado y ejecutado en diferentes máquinas. El compilador, que es el programa que traduce las instrucciones del programa de alto nivel al lenguaje maquinal, si es específico para cada máquina, pero el programa en el lenguaje de alto nivel es independiente de la máquina.

5.1.4. Lenguajes de cuarta generación

Con la evolución natural de las ciencias computacionales, aparecieron los lenguajes de cuarta generación en la década de los ochentas. Los primeros esfuerzos realizados en esta área se orientaron al diseño de los generadores de aplicaciones. Los desarrollos posteriores se enfocaron al desarrollo de lenguajes de muy alto nivel, que son los que comúnmente se conocen como lenguajes de cuarta generación. Todos estos esfuerzos se encaminaron al desarrollo de aplicaciones administrativas.

La principal característica de los lenguajes de cuarta generación es que son no procedurales. En un lenguaje no procedural, el programador especifica lo que debe realizar el programa sin definir el cómo deben realizarse las operaciones. En los lenguajes de tercera generación, el programador debe definir exactamente las instrucciones que debe ejecutar la computadora para procesar la información, es decir, el programador indica cómo debe procesarse la información.

Los generadores de aplicaciones constan de un lenguaje de alto nivel que permite especificar los diferentes programas que componen un sistema de información computarizado. En esta especificación, el programador se centra en lo que debe realizar el sistema en lugar de enfocarse en cómo debe realizarlo. Una vez que se ha especificado el sistema, se utiliza un generador de código, que toma como entrada las especificaciones del sistema y produce como salida programas en algún lenguaje de tercera generación, el cual generalmente es COBOL. Los programas generados en el lenguaje de tercera generación pueden ser modificados por el programador si se considera necesario. Finalmente, estos programas se compilan y se producen los programas en lenguaje maquina.

Los lenguajes de cuarta generación propiamente dichos constan de un manejador de bases de datos, un generador de reportes, un manejador de formas y un lenguaje de tercera generación.

El manejador de base de datos permite definir y crear las tablas (archivos) de que consta la base de datos del sistema que se está desarrollando, así como las relaciones que existen entre las diferentes entidades de la base de datos. Permite también definir ciertas validaciones en los campos de los archivos de la base de datos. Por ejemplo, puede especificarse que un cierto campo debe contener una fecha y no algún dato de otro tipo. La base de datos, sus relaciones y las validaciones necesarias se definen y crean de una manera simple y amigable. También es fácil realizar modificaciones a la base de datos.

El generador de reportes permite definir los diferentes reportes que debe producir el sistema de una manera sencilla. Al definir los reportes, se pueden combinar datos de diferentes archivos, de acuerdo a las relaciones existentes y a ciertas condiciones que deben cumplir los datos.

El manejador de formas se utiliza para generar formatos de pantallas para consulta y actualización de manera interactiva. También se pueden realizar validaciones en los diferentes campos que contiene la forma. Esto permite desarrollar y modificar los formatos de pantalla de una manera rápida y sencilla.

El lenguaje de tercera generación se utiliza para desarrollar programas para procesar la información de la base de datos, interactuar con las formas definidas y producir los reportes del sistema. El lenguaje de cuarta generación tiene muchas instrucciones que realizan operaciones sin necesidad de que el programador defina como deben realizarse; sin embargo, algunas operaciones particulares de un cierto sistema deben realizarse mediante programas desarrollados en el lenguaje de tercera generación.

5.1.5. Lenguajes orientados a objetos.

Paralelamente al desarrollo de lenguajes de programación, se crearon técnicas de programación, siendo la programación estructurada la que ocasionó el primer impacto fuerte en esta área.

Al evolucionar las técnicas de programación, se creó la programación orientada a objetos. La principal característica de esta técnica es que tanto los datos como los procedimientos para manipularlos se encapsulan como un todo denominado objeto. Para construir sistemas usando programación orientada a objetos se definen primeramente los diferentes objetos que intervienen en el sistema y posteriormente se utilizan dicho objetos para construir el sistema.

Esta técnica de programación requiere de un ambiente de programación adecuado para su implantación, lo cual dio origen a los lenguajes de programación orientados a objetos, cuyo objetivo primordial es permitir el desarrollo de este tipo de programas. El primer lenguaje orientado a objetos que se desarrolló es Smalltalk en 1970. Posteriormente se han diseñado otros lenguajes orientados a objetos como Hypertalk, Turbo Pascal, C++ y Java..

5.1.6. Lenguajes orientados a programación visual.

Una de las nuevas tendencias en el área de programación es la programación visual. Esta técnica incluye la programación orientada a objetos, pero adicionalmente permite desarrollar programas de manera gráfica, mediante dibujos.

Para construir los programas usando esta técnica, primeramente se definen los objetos y posteriormente se realizan las conexiones entre los objetos de manera gráfica para desarrollar los programas. El programador no tiene que aprender la sintaxis de un lenguaje de programación y codificar programas en la forma tradicional.

ObjectVision y VISUAL BASIC son dos de los lenguajes de programación visual desarrollados recientemente.

5.2. Ensambladores

Un ensamblador es un programa que toma como datos un programa escrito en lenguaje ensamblador y genera como resultado un programa en lenguaje maquinal. El programa original,

escrito en lenguaje ensamblador, se conoce como programa fuente. El programa generado en lenguaje maquina se conoce como programa objeto.

El programa objeto generalmente no es un programa ejecutable directamente debido a que las direcciones a las cuales hace referencia son direcciones relativas y se ensambla a partir de la dirección cero. También es factible que este programa utilice rutinas ya desarrolladas previamente y guardadas en la biblioteca del sistema computacional, las cuales tienen que ser añadidas y ligadas al programa objeto.

Cuando un programa objeto ya ha sido ligado con los subprogramas externos que utiliza, se convierte en un programa ejecutable, el cual puede ser cargado a memoria y ejecutado.

5.3. Compiladores

Un compilador es un programa que toma como entrada un programa desarrollado en un lenguaje de alto nivel y produce como salida un programa en lenguaje ensamblador. Este programa en lenguaje ensamblador representa la traducción de las instrucciones del programa escrito en el lenguaje de alto nivel, a las instrucciones de máquina necesarias para realizar las operaciones especificadas en el programa original. En este caso no se produce directamente un programa objeto, ya que las instrucciones de máquina están codificadas con mnemónicos y las direcciones son simbólicas. Este programa se alimenta al ensamblador y se tiene finalmente el programa objeto en lenguaje maquina.

El programa fuente en este caso es el desarrollado en el lenguaje de alto nivel.

Supóngase que se tiene la siguiente instrucción en un lenguaje de alto nivel:

$$C = A + B$$

Esta instrucción podría ser una instrucción en FORTRAN, BASIC, PASCAL o algún lenguaje similar. Lo que realiza esta instrucción es sumar el dato que se encuentra almacenado en la variable A con el dato que se encuentra almacenado en la variable B, guardando el resultado de la suma en la variable C.

La traducción de esta instrucción a lenguaje ensamblador, usando las instrucciones de máquina presentadas en el capítulo dos, sería la siguiente:

	LDA, A, A	CARGAR EN AC EL DATO A
	ADD, A, B	SUMARLE AL AC EL DATO B
	STA, A, C	GUARDAR LA SUMA EN C
	.	
	.	
	.	
A	DEF	+10
B	DEF	+2
C	DEF	0

En el capítulo dos se presentó como ejemplo un pequeño programa para sumar un conjunto de N números, guardados en la memoria a partir de una posición dada y guardar el resultado de la suma en otra posición de memoria.

Las instrucciones de un lenguaje de alto nivel que realizarían esta misma función, donde A representa la dirección a partir de la cual se encuentran almacenados los N números y S representa la dirección en la cual se dejaría el resultado de la suma, podrían ser:

En FORTRAN:

```
          S=0
          DO 10 I=1,N
10        S=S+A(I)
```

En BASIC:

```
          S=0
          FOR I=1 TO N
            S=S+A(I)
          NEXT I
```

En PASCAL:

```
          S:=0
          FOR I FROM 1 TO N
            S:=S+A[I];
```

Para este caso, compárese el número de instrucciones de los lenguajes de alto nivel con la cantidad de instrucciones necesarias en lenguaje maquina. Esto permitirá tener una idea de la facilidad de desarrollar programas en un lenguaje de alto nivel comparado con el desarrollo del mismo programa en lenguaje ensamblador.

5.4. Interpretadores

Algunos lenguajes de alto nivel no son traducidos al lenguaje maquina para ejecutarlos, sino que son interpretados directamente como están escritos. Por ejemplo, el lenguaje BASIC nació como un lenguaje interpretado. Posteriormente se desarrolló el compilador de BASIC, de tal forma que actualmente se puede trabajar en este lenguaje con el interpretador, o bien, con el compilador que sí lo traduce a lenguaje ensamblador y posteriormente se traduce a lenguaje maquina..

Cuando se tiene un lenguaje de alto nivel interpretado, se ejecuta un programa que interpreta las instrucciones del lenguaje de alto nivel directamente. Este programa, conocido como interpretador, tiene subprogramas para ejecutar las instrucciones de máquina necesarias para realizar cada una de las instrucciones del lenguaje de alto nivel.

El programa principal, que interpreta las instrucciones del lenguaje de alto nivel, llama a cada uno de los subprogramas necesarios para ejecutar las operaciones necesarias para cada instrucción del lenguaje de alto nivel. El programa fuente en este caso no se traduce a lenguaje maquina

Generalmente un programa interpretado se ejecuta más lentamente que el correspondiente programa compilado. A pesar de esta desventaja, los lenguajes interpretados ofrecen muchas ventajas de depuración y pruebas para desarrollo de programas tales como poder detener la ejecución del programa para investigar los valores de las variables en ese momento e incluso cambiar el valor de algunas variables para posteriormente continuar la ejecución en el mismo lugar en que se detuvo la ejecución.

Actualmente existen lenguajes que pueden ser interpretados durante las fases de desarrollo y pruebas de las aplicaciones para tener las ventajas que ofrece un interpretador. Cuando una aplicación ya ha sido probada y depurada, se puede compilar para dejar el programa en lenguaje maquina y así tener una ejecución más rápida de la aplicación final.

5.5. Resumen

Las primeras computadoras se programaban directamente en lenguaje maquina. En este lenguaje, tanto las instrucciones como los datos y las direcciones son secuencias de ceros y unos, ya que todas las computadoras digitales trabajan internamente con el sistema binario.

Posteriormente se desarrolló el lenguaje ensamblador. Es este lenguaje, las instrucciones de máquina son codificadas con mnemónicos y las direcciones del programa son direcciones simbólicas. Existen también las pseudoinstrucciones, que son directivas para el programa que traduce el código en lenguaje ensamblador al código en lenguaje maquina.

Subsecuentemente se desarrollaron los lenguajes de tercera generación en los cuales se tienen instrucciones de alto nivel, parecidas al lenguaje natural o al lenguaje algebraico. Cada una de las instrucciones de un lenguaje de alto nivel se traduce a una o más instrucciones de lenguaje maquina. Generalmente se requieren varias instrucciones de lenguaje maquina para traducir una sola instrucción de lenguaje de alto nivel.

El lenguaje maquina y el lenguaje ensamblador son específicos para una cierta máquina mientras que los lenguajes de tercera generación son independientes de la arquitectura de la máquina.

Con el avance de la computación surgieron los lenguajes de cuarta generación. Estos lenguajes son no procedurales en contraste con los de tercera generación. Un lenguaje de cuarta generación típico incluye un manejador de base de datos, un generador de reportes, un manejador de formas y un lenguaje de programación de tercera generación. El desarrollo de aplicaciones en los lenguajes de cuarta generación requiere de menor tiempo y esfuerzo que en los lenguajes de tercera generación.

Los lenguajes que han tenido mucho auge últimamente son los lenguajes orientados a objetos y los lenguajes orientados a la programación visual.

En los lenguajes orientados a objetos, se encapsulan los datos y los procedimientos para manipularlos en un todo denominado objeto. Se definen inicialmente todos los objetos de que consta un sistema y posteriormente se definen las interacciones entre estos objetos.

Los lenguajes orientados a la programación visual incluyen las operaciones de un lenguaje orientado a objetos, pero adicionalmente permiten que las relaciones entre los objetos sean determinadas de una manera gráfica, haciendo más fácil la tarea de programación.

Las computadoras digitales solamente pueden ejecutar programas en lenguaje maquinal. Para traducir las instrucciones en algún lenguaje de programación al lenguaje maquinal, se han desarrollado diferentes programas para realizar la traducción en forma automática. Los programas que traducen las instrucciones de un programa en lenguaje ensamblador al lenguaje maquinal son conocidos como ensambladores. Para traducir las instrucciones de un lenguaje de alto nivel al lenguaje maquinal se utilizan los programas conocidos como compiladores.

Existen algunos lenguajes de alto nivel en los cuales no se traducen las instrucciones del lenguaje al lenguaje maquinal, sino que estas instrucciones son interpretadas directamente por un programa conocido como interpretador. Generalmente los programas interpretados se ejecutan más lentamente que los programas compilados; sin embargo, se tienen ciertas ventajas durante las fases de desarrollo y depuración de aplicaciones que usualmente no se tienen cuando se trabaja con un lenguaje compilado.

5.6. Problemas

1. Explique brevemente que son el lenguaje maquinal y el lenguaje ensamblador.
2. ¿Qué ventajas proporciona un lenguaje de alto nivel?
3. Mencione las características de un lenguaje de tercera generación.
4. Explique que función realiza un compilador.
6. Explique que función realiza un interpretador.
7. Describa las características principales de un lenguaje de cuarta generación.
8. ¿Qué es un lenguaje de programación visual y qué ventajas proporciona?
9. Investigue las características de la programación estructurada.
10. Investigue las características de la programación orientada a objetos.

11. Java es un lenguaje orientado a objetos que puede ejecutarse en cualquier plataforma. Investigue como se compila y se ejecuta este lenguaje para que esto sea posible.

CAPITULO 6

SISTEMAS OPERATIVOS

6.1. ¿QUE ES UN SISTEMA OPERATIVO?

Un sistema operativo es un conjunto de programas cuya función principal es ayudar en la administración de los sistemas computacionales.

Un sistema computacional está compuesto por el equipo (hardware), los programas (software) y los datos. Los principales componentes del equipo son la unidad central de proceso, la unidad de memoria y los dispositivos periféricos tales como unidades de disco, impresoras y terminales. Los programas pueden ser clasificados como programas del sistema operativo, programas de utilería y programas de aplicación. Los programas de aplicación son los que en última instancia resuelven los problemas de los usuarios, algunos ejemplos de estos programas son los sistemas de nómina, contabilidad, inventarios, correo electrónico y los sistemas de apoyo al diseño mecánico entre otros. El objetivo de los programas de utilería es ayudar al desarrollo de aplicaciones. Los editores de texto, los compiladores y los editores de ligas son algunos ejemplos de este tipo de programas. Los programas del sistema operativo son los que controlan la operación del sistema computacional, asignan recursos para la ejecución de las aplicaciones, asignan áreas en disco para los archivos de datos, mantienen directorios de los archivos en disco, proveen la protección de los archivos en disco, permiten el acceso al sistema computacional solamente a los usuarios autorizados para hacerlo, y ayudan a mantener respaldos de la información y las aplicaciones que están almacenadas en el sistema computacional.

El sistema operativo también puede visualizarse como un componente que actúa como intermediario entre los usuarios de un sistema computacional y el propio equipo. Los programas que forman el núcleo del sistema operativo, conocido como kernel, siempre están ejecutándose y controlando la operación del sistema computacional. Los sistemas operativos también deben proveer un ambiente conveniente para desarrollar y ejecutar aplicaciones en forma eficiente.

En la siguiente sección se presentará un breve panorama que justifica la necesidad de tener sistemas operativos.

6.2. NECESIDAD DE LOS SISTEMAS OPERATIVOS

Los primeros sistemas computacionales que se utilizaron estaban dedicados a solucionar problemas de las áreas de ingeniería y ciencias. Cuando se utilizaban para resolver un cierto problema específico, se alimentaba el programa correspondiente y se ejecutaba solamente ese programa, hasta que finalizaba su operación. Posteriormente se alimentaba otro programa para resolver un problema diferente y solamente se ejecutaba ese nuevo programa. Estos sistemas estaban dedicados completamente a un solo usuario y éste tenía que cargar su programa directamente a la memoria, en lenguaje maquina, a través de los controles de la consola que eran principalmente interruptores y botones. Para examinar el contenido de los registros y la memoria

se disponía de focos que al estar encendidos representaban un uno, y al estar apagados representaban un cero.

Cuando se empezaron a añadir periféricos tales como unidades de cinta, teletipos, lectoras de papel perforado e impresoras, se empezaron a desarrollar programas en lenguaje ensamblador, lo cual contribuyó a incrementar la eficiencia de los programadores; sin embargo, el desarrollo de programas era todavía tedioso, propenso a errores y muy dependiente de la habilidad del programador. Existían muchas operaciones que se repetían de una aplicación a otra y que eran complejas de programar tales como las operaciones de lectura de datos y de impresión de resultados. Para mejorar el desarrollo de programas, se empezaron a diseñar subprogramas que proveían todas estas funciones rutinarias que se repetían constantemente de un programa a otro y que no tenía caso programar cada vez que se desarrollaba una aplicación nueva.

Posteriormente se desarrollaron los lenguajes de alto nivel como FORTRAN y COBOL. Para ejecutar un programa desarrollado en uno de estos lenguajes, se tenía que cargar el compilador, generalmente de una unidad de cintas o de una lectora de tarjetas perforadas, y luego alimentar el programa fuente para su compilación. El compilador generaba un programa en lenguaje ensamblador, el cual se almacenaba temporalmente en cinta magnética. Luego tenía que cargarse a memoria el ensamblador para que tomara el resultado de la compilación y generara un programa objeto, es decir, en lenguaje maquinal. Finalmente, se tenía que cargar el editor de ligas para que tomara las rutinas necesarias de la biblioteca y generara el módulo ejecutable. Todos estos pasos tenían que ser realizados por el programador para ejecutar un programa, dando como resultado una baja utilización del procesador.

Para mejorar la utilización de los equipos computacionales, primeramente se contrataron operadores especializados para que manejaran el sistema computacional en lugar de que los programadores lo hicieran. El operador del sistema tenía más práctica que los programadores para ejecutar los programas y se obtenía una mayor utilización de estos sistemas. Sin embargo; el aprovechamiento de los sistemas todavía era bajo debido al tiempo perdido entre la ejecución de un programa y otro, ya que era necesario cambiar cintas magnéticas, cargar compiladores a memoria, cargar el ensamblador, etc.

El segundo paso que se dio estuvo encaminado a tratar de realizar la mayoría de las operaciones que hacía el operador en forma automática, para evitar pérdidas de tiempo. Para lograr esto, se diseñaron subprogramas que ayudaban en estas tareas y, de paso contribuyeran a administrar las bibliotecas de subprogramas que para ese entonces se habían desarrollado. Este fue el nacimiento de los sistemas operativos, cuya función es ayudar en la administración de los sistemas computacionales, utilizándolos de la manera más eficiente posible.

Con el paso del tiempo y los avances tecnológicos, los sistemas computacionales incrementaron su poder de cómputo y tuvieron mayor capacidad de almacenamiento. Continuar dedicando estos sistemas a un solo usuario representaba una subutilización de los mismos. Entonces, se contempló la posibilidad de aumentar la utilización de estos sistemas, que representaban una inversión considerable de dinero, permitiendo que un usuario pudiera ejecutar sus programas simultáneamente con otros usuarios. Los sistemas operativos también controlan la compartición

de los recursos computacionales entre los diferentes usuarios y deben realizarlo de una manera eficiente.

Al permitir que varios usuarios utilizaran los sistemas computacionales en forma simultánea, se complicó más su administración ya que era necesario que el sistema operativo mantuviera los archivos de cada usuario en el almacenamiento secundario y brindara la protección adecuada para que un usuario no borrara, accidental o deliberadamente, los archivos de otro usuario. Asimismo, al ejecutar varios programas en forma simultánea, también era necesario saber en cuál área de memoria estaba el programa de un usuario y en cuál estaba el programa de otro usuario. Además, un programa de un usuario no debía almacenar datos en el área de memoria reservada para otro usuario.

Los sistemas operativos que apoyaban la administración de los sistemas computacionales empezaron a volverse cada vez más y más complejos, por lo cual se dedicó un gran esfuerzo a definirlos y diseñarlos de una manera mas estructurada. Actualmente, el diseño de los sistemas operativos sigue siendo motivo de mucha investigación.

Los primeros sistemas computacionales estaban confinados en un recinto denominado comúnmente centro de cómputo. Restringiendo el acceso a este centro, se aseguraba que estos sistemas computacionales no podían ser utilizados por usuarios no autorizados para hacerlo. Cuando se empezaron a utilizar terminales remotas que se localizaban fuera del centro de cómputo, el problema de controlar el acceso de usuarios no autorizados al sistema computacional se volvió mas complicado. Actualmente, con la proliferación de redes computacionales y la facilidad de tener acceso en forma remota a lugares muy distantes geográficamente, el problema de seguridad en los sistemas computacionales ha tomado mayores dimensiones. Esto crea un nuevo reto para los diseñadores de sistemas operativos ya que ahora, además de administrar el sistema computacional, deberían contemplar funciones relacionadas con la seguridad de la información y las aplicaciones contenidas en los sistemas computacionales y con la restricción del acceso por parte de usuarios no autorizados.

6.3. TIPOS DE SISTEMAS OPERATIVOS

Los primeros sistemas operativos estaban diseñados para ejecutar solamente un programa a la vez. Estos sistemas operativos se conocen como sistemas de procesamiento en lotes (BATCH). En los sistemas computacionales con estos sistemas operativos, se alimentaban los programas a la computadora y se almacenaban temporalmente en disco duro. El sistema operativo mantenía una lista de procesos a ejecutar, denominada lista de procesos pendientes, y se iban tomando los programas para ejecutarlos, uno a uno, de esta lista. Originalmente los programas se tomaban en el orden en que estaban en la lista, que era el mismo orden en que se habían alimentado al sistema computacional. Posteriormente se incluyó el concepto de prioridades y a cada programa se le asignaba una prioridad. Con este nuevo esquema, se iban tomando los procesos de mayor prioridad de la lista de procesos pendientes, lo que permitía ejecutar primeramente aquellos procesos más importantes, que eran los que tenían mayor prioridad.

Con el esquema de procesamiento en lotes, cada vez que un programa realizaba operaciones de entrada o salida, como escrituras a disco, impresión de resultados, o lecturas de disco; se desperdiciaba mucho tiempo de procesador. Para mejorar la utilización del procesador, se introdujo el concepto de multiprogramación. En los sistemas computacionales de procesamientos en lotes con multiprogramación, el sistema operativo mantenía otras dos listas. Una de ellas era la lista de procesos listos, que eran los programas que estaban siendo ejecutados en ese momento en el sistema computacional. La otra lista, denominada lista de procesos suspendidos, contenía los procesos que habían solicitado una operación de entrada o salida y estaban en espera de que esta operación terminara. Cuando el programa que estaba siendo ejecutado iba a iniciar alguna operación de entrada o salida, el sistema operativo añadía este programa a la lista de procesos suspendidos y ordenaba al canal que realizara la operación de entrada o salida. El programa permanecía en la lista de procesos suspendidos hasta que su operación de entrada o salida se terminaba. En ese momento, cuando el canal le indicaba al procesador que la operación de entrada o salida había terminado, se quitaba de la lista de procesos suspendidos y se pasaba a la lista de procesos listos. Mientras el canal realizaba la operación solicitada, se seleccionaba otro programa de la lista de procesos listos y se continuaba con sus ejecución hasta que terminara o hasta que tuviera que realizar una operación de entrada o salida. Cuando el número de programas en la lista de procesos listos disminuía, se traía un programa de la lista de procesos pendientes a la lista de procesos listos.

El concepto de multiprogramación permitió una mejor utilización del procesador, pero hizo más complicado el diseño de los sistemas operativos ya que no solamente involucraba mantener las listas adicionales de procesos suspendidos y procesos listos, sino que también era necesario administrar mejor la unidad de memoria de los sistemas computacionales y las unidades de disco.

Con la evolución de las ciencias computacionales y la electrónica digital, se introdujeron los sistema computacionales con terminales. Estos sistemas permitieron desarrollar aplicaciones interactivas, donde un usuario en una terminal podía ejecutar una aplicación en el sistema computacional a través de comandos enviados desde la terminal. Los resultados del programa podían ser visualizados en la misma terminal o enviados a impresión para su análisis posterior. La filosofía de los sistemas operativos para este nuevo tipo de aplicaciones tuvo que cambiar, ya que un sistema computacional podía tener conectadas diez o veinte terminales y, en cada una de ellas, podía haber un usuario ejecutando una aplicación. No era deseable que un usuario mandara ejecutar una aplicación desde una terminal y tuviera que esperar diez o quince minutos mientras se terminaban de procesar las aplicaciones que habían enviado otros usuarios antes que él. En estos sistemas computacionales, el sistema operativo tenía que asignar un cierto tiempo del procesador a cada uno de los programas que estaban siendo ejecutados, en una manera cíclica, para que cada usuario tuviera la impresión de que la computadora lo estaba atendiendo solamente a él. Estos sistemas operativos se conocen como sistemas de tiempo compartido. El tiempo que se le asigna a cada programa es pequeño y, al atender en forma cíclica a cada uno de los procesos, se da la impresión de que la computadora estaba dedicada solamente a una terminal. Este tiempo se denomina "quantum". Inicialmente, el quantum que se asignaba a cada programa era el mismo, pero posteriormente se introdujo el concepto de prioridades y, con este nuevo esquema, el quantum asignado a cada proceso dependía de su prioridad.

Hay ciertas aplicaciones que, por su naturaleza, no necesitan ser interactivas. Por ejemplo, el programa que genera la nómina de una empresa genera resultados impresos que son los que interesan y no se necesita interactuar con el programa mientras se está ejecutando.

Con la finalidad de hacer los sistemas computacionales más versátiles, se introdujeron los sistemas operativos que administraban tanto programas interactivos como procesamiento en lotes. En éstos, el sistema operativo mantenía una lista de procesos interactivos y otra lista para procesamiento en lotes, además de las listas de procesos suspendidos y procesos pendientes. El sistema operativo ejecutaba primeramente los procesos interactivos. Cuando no había ningún proceso interactivo que atender, ya sea porque todos habían terminado o porque estaban suspendidos debido a operaciones de entrada y salida, atendía a programas de procesamiento en lotes. Si esta lista también estaba vacía, traía un programa de la lista de procesos pendientes a la lista de procesamiento en lotes.

Antiguamente los sistemas operativos se diseñaban con el único propósito de administrar los recursos del sistema computacional de la manera más eficiente posible, sin prestar demasiada atención en su facilidad de operación por parte de los usuarios y los programadores. Actualmente se pone mucha atención a este último punto, sin descuidar la función primordial de un sistema operativo. Los sistemas operativos son cada día más amigables tanto para los usuarios como para los programadores, haciendo que los sistemas computacionales sean utilizados en una manera más conveniente.

La sofisticación de los sistemas computacionales actuales ha puesto nuevos retos a los diseñadores de sistemas operativos. Actualmente hay sistemas operativos diseñados específicamente para administrar recursos en redes computacionales que incluyen varios equipos tales como servidores de archivos, servidores de impresión, graficadores y estaciones de trabajo. Por otra parte, los multiprocesadores, que son sistemas computacionales con varios procesadores, necesitan sistemas operativos diseñados para administrar en forma eficiente este nuevo ambiente computacional.

La arquitectura de los sistemas computacionales ha influenciado definitivamente el diseño e implantación de los sistemas operativos puesto que su función es administrarlos. También es cierto que algunos cambios en la arquitectura de los sistemas computacionales han surgido de necesidades relacionadas con el diseño de los sistemas operativos.

La mayoría de los procesadores pueden estar ejecutando instrucciones en estado programa o estado supervisor. Al estar ejecutando en estado programa, solamente pueden realizarse ciertas instrucciones de máquina mientras que en estado supervisor pueden realizarse todas las instrucciones de máquina. El sistema operativo se ejecuta en estado supervisor y cuando se va a ceder control a un programa de aplicación se cambia a estado programa. Las instrucciones privilegiadas solamente se pueden ejecutar en estado supervisor y típicamente son todas las instrucciones de entrada y salida así como las instrucciones que encienden o apagan las interrupciones. Cuando se genera cualquier interrupción, el procesador automáticamente se pone en estado supervisor y las rutinas que atienden las interrupciones son parte del sistema operativo.

6.4. COMPONENTES Y SERVICIOS BASICOS DE UN SISTEMA OPERATIVO

En las secciones anteriores se presentaron de una manera muy somera algunas de las funciones y servicios que debe prestar un sistema operativo, así como las razones por las cuales son necesarias.

A manera de ejemplo, considérese el desarrollo de un programa de aplicación en un sistema computacional utilizando un lenguaje de tercera generación. Supóngase que el objetivo de este programa es mantener un archivo con los datos de un grupo de personas y que el programa deberá operar de manera interactiva con los usuarios.

La persona que va a desarrollar el programa, una vez que lo ha diseñado y codificado, se da de alta en el sistema computacional. Para esto, deberá proporcionar un nombre de usuario y una clave de acceso. El sistema operativo verificará que tanto el nombre del usuario como la clave de acceso proporcionada sean válidos, para permitir el acceso de este usuario al sistema computacional. Una vez que el usuario se ha dado de alta en el sistema, utiliza un editor de textos para alimentar el programa. Al utilizar el editor de textos, el sistema operativo localiza el archivo en disco que contiene el editor de textos, lo carga a memoria y empieza a ejecutar la aplicación. Cuando la persona que está desarrollando la nueva aplicación termina de alimentar las instrucciones de su programa, solicita al editor de textos que guarde su archivo en disco, proporcionando un nombre para dicho archivo. De nuevo, el sistema operativo asigna un área en disco para almacenar el archivo, actualiza el directorio para incluir el nombre del nuevo archivo y lo almacena en disco. El sistema operativo también es el encargado de saber que este nuevo archivo es propiedad del usuario que lo creó.

En este momento ya existe un archivo con el programa fuente en disco. Será necesario ahora compilar este programa para que quede en una forma entendible para la computadora digital. El usuario invoca al compilador del lenguaje utilizado para codificar el programa. El sistema operativo deberá localizar el archivo en disco que contiene el compilador, cargarlo a memoria y empezar su ejecución. Al ejecutarse el compilador, deberá de utilizar los servicios del sistema operativo para traer de disco el programa fuente que se desea compilar. Recuérdese que este programa fue creado con el editor de textos y guardado en un archivo en disco. Cuando termina la compilación, deberá guardarse el resultado de ésta en otro archivo en disco y de nuevo se solicitarán los servicios del sistema operativo. Indudablemente el programa que se está desarrollando utiliza algún archivo en disco para mantener los datos del grupo de personas mencionado. Además, utilizará seguramente el teclado de una terminal para alimentar los datos que se desean mantener y alguna terminal de video para desplegar estos datos al momento de su captura o para alguna consulta posterior. Es muy probable que se tenga la opción de generar algún reporte impreso con los datos almacenados en el archivo, para lo cual será necesario usar las rutinas para manejo de alguna impresora. Los subprogramas que manejan los dispositivos periféricos ya han sido desarrollados y probados, así como las rutinas que atienden las interrupciones de estos dispositivos. Para dejar la aplicación en forma ejecutable es necesario invocar al programa que toma el resultado de la compilación, que se dejó en un archivo en disco, y haga las ligas de esta compilación con las rutinas de manejo de periféricos necesarias y lo deje finalmente en un archivo que ya contenga la aplicación ejecutable.

Esta serie de eventos dan una idea de algunos de los servicios que debe proveer un sistema operativo, aunque no todos son muy obvios. En esta sección se describirán con mayor detalle estos servicios básicos que debe proveer un sistema operativo así como sus componentes principales.

Cualquier sistema operativo actual debe incluir los siguientes componentes:

- Administrador de procesos
- Administrador de memoria
- Administrador de almacenamiento secundario
- Administrador del sistema de entrada y salida
- Administrador de archivos
- Sistema de protección y seguridad
- Administrador de comunicaciones
- Intérprete de comandos
- Módulo de contabilidad

Dentro de los servicios que debe proveer un sistema operativo se tienen los siguientes:

- Ejecución de programas
- Manejo de operaciones de entrada y salida
- Manipulación de archivos
- Detección de errores
- Comunicaciones
- Protección
- Asignación de recursos
- Contabilidad

6.4.1. Administrador de procesos.

Cualquier programa que esté cargado en memoria principal representa un proceso. Los procesos pueden estar en uno de tres estados: listo, suspendido o activo. En cualquier sistema computacional existen procesos del sistema operativo y procesos de usuarios.

Cuando un programa solicita una operación de entrada o salida, debe esperar a que dicha operación sea realizada para continuar su ejecución; en este caso se tiene un proceso suspendido. También sería un proceso suspendido aquel que tuviera todo lo necesario para iniciar ejecución, excepto algún o algunos recursos que deben ser asignados en forma exclusiva, como podría ser una unidad de cinta.

Un proceso listo es aquel que tiene todo lo necesario para iniciar ejecución, pero que todavía no se le asigna el procesador. Un proceso suspendido porque estaba en espera de que se terminara una operación de entrada o salida se transforma en un proceso listo cuando esta operación

termina. Igualmente, un proceso suspendido por falta de asignación de algún recurso, se convierte en un proceso listo cuando dicho recurso se le asigna.

Cuando a un proceso listo se le asigna el procesador y empieza su ejecución, se convierte en un proceso activo o en ejecución.

El administrador de procesos realiza las siguientes funciones:

- Creación y supresión de procesos.
- Suspensión y reactivación de procesos.
- Sincronización de procesos.
- Comunicación de procesos.
- Asignación de recursos a procesos.

6.4.2. Administrador de memoria.

Los sistemas computacionales actuales utilizan multiprogramación para incrementar la utilización del procesador, manteniendo en memoria principal varios procesos en forma simultánea. El administrador de memoria debe realizar las siguientes funciones:

- Saber cuáles partes de la memoria están libres.
- Saber qué partes de la memoria están asignadas a cada uno de los programas cargados en memoria.
- Decidir cuál proceso debe ser cargado a memoria cuando haya espacio disponible.
- Asignar áreas de memoria a los procesos cuando sea necesario y liberarlas cuando ya no se necesiten.

6.4.3. Administrador de almacenamiento secundario.

Los sistemas computacionales actuales utilizan el almacenamiento secundario para almacenar datos y programas en forma permanente. Los dispositivos más comúnmente usados para este propósito son los discos magnéticos. El sistema operativo, los compiladores, los editores de texto y las rutinas de utilería también están almacenados en discos magnéticos.

El sistema operativo es responsable de:

- Administrar el espacio libre en disco.
- Asignar el espacio en disco para los archivos.
- Atender las solicitudes de acceso a las unidades de disco.

6.4.4. Administrador del sistema de entrada y salida.

Cada dispositivo de entrada y salida tiene sus propias características especiales y es utilizado a través de una rutina denominada manejador del dispositivo. Los usuarios no necesitan conocer todos los detalles referentes al manejo de un dispositivo. El sistema operativo se encarga de

manejarlos mediante sus rutinas de entrada y salida, así como de atender las interrupciones y ordenar a los procesadores de entrada y salida que realicen las operaciones necesarias.

6.4.5. Administrador de archivos.

Un archivo es una colección de información relacionada cuyas características son definidas por la persona que lo crea. Un archivo puede contener datos, programas fuentes o programas ejecutables. Los archivos en los sistemas computacionales se encuentran almacenados por lo general en discos magnéticos, cintas magnéticas y discos ópticos.

El sistema operativo, a través del sistema de archivos, permite realizar varias funciones relacionadas con los archivos, independientemente del medio de almacenamiento en que se encuentren. La mayoría de los sistemas operativos permiten manejar directorios para organizar mejor los archivos.

Las principales funciones del sistema de archivos son:

- Creación y supresión de archivos.
- Creación, supresión y mantenimiento de directorios.
- Actualización y mantenimiento de archivos.
- Localización física de los archivos en los dispositivos.
- Permitir el respaldo de la información.
- Restaurar los archivos respaldados.

6.4.6. Sistema de protección y seguridad.

La protección en los sistemas operativos está relacionada con el acceso de los procesos a los recursos del sistema computacional. Por ejemplo, el sistema operativo debe permitir el acceso a un archivo solamente a aquellos procesos o usuarios que tengan derecho a usarlo. Esta parte del sistema operativo debe proveer los mecanismos adecuados para proteger todos los recursos del sistema computacional, tanto recursos físicos como programas y datos.

La seguridad en los sistemas operativos está relacionada con el acceso de usuarios no autorizados al sistema computacional.

6.4.7. Administrador de comunicaciones.

Hoy en día, la mayoría de los sistemas computacionales proporcionan la facilidad de comunicarse con otros sistemas para consultar bibliotecas electrónicas e información disponible en los diferentes servidores de las redes computacionales. El subsistema que administra las comunicaciones es el responsable de establecer, mantener y terminar la comunicación de un sistema computacional con otros sistemas.

Para realizar sus funciones, el administrador de comunicaciones debe seleccionar el protocolo de comunicación adecuado para establecer un enlace con otro sistema. También es el encargado de recuperar los errores de transmisión que ocurran en las líneas de comunicación y, finalmente,

terminar el enlace cuando ya no se necesita. En muchos sistemas computacionales actuales es posible tener activos varios enlaces con diferentes sistemas, esto también es administrado por este subsistema del sistema operativo.

6.4.8. Intérprete de comandos.

Todos los sistemas operativos proporcionan un ambiente para que los usuarios y los programadores interactúen con los sistemas computacionales. En algunos equipos de cómputo, esta interacción se realiza mediante comandos que proporciona el usuario en forma de texto mediante un teclado. Por ejemplo, un usuario puede solicitar que se despliegue en la terminal la lista de los archivos que se encuentran almacenados bajo un cierto directorio; o bien, puede solicitar que se borre uno de los archivos que ya no desea guardar. También puede solicitar que se inicie la ejecución de un programa de aplicación para resolver un problema particular, o que se copie un archivo de un directorio a otro.

Todas estas solicitudes que hace un usuario al sistema operativo se realizan mediante comandos que son ejecutados por el intérprete de comandos. La tendencia actual es que la mayoría de los comandos se proporcionen al sistema operativo en forma gráfica, mediante un ratón, proporcionando un ambiente más amigable para los usuarios y los programadores.

6.4.9. Módulo de contabilidad.

Generalmente los sistemas operativos llevan estadísticas de uso de recursos por parte de los diferentes usuarios del sistema computacional. Estas estadísticas incluyen el tiempo de utilización del procesador, la cantidad de memoria usada, el área en disco asignada a los archivos de un usuario, el número de accesos a disco realizados por sus programas, el volumen de información enviada a impresión y el tiempo de conexión en caso de redes computacionales o servidores, entre otras. Algunas veces estos datos son utilizados solamente con fines informativos y, en otras ocasiones, se usan para facturar a los usuarios por el uso de estos recursos.

6.5. SINCRONIZACION DE PROCESOS

Un problema que se presenta en sistemas computacionales con multiprogramación es la sincronización de procesos. Considérese dos procesos que están actualizando un mismo archivo. Es cierto que los dos procesos nunca se encuentran en ejecución al mismo tiempo, pero aún así pueden presentarse problemas de sincronización. Supóngase que el proceso A inicia ejecución y empieza a actualizar el archivo X, pero se le acaba el tiempo que tiene asignado el procesador antes de terminar de actualizar el archivo. En ese momento se asigna el procesador al proceso B que también actualiza el archivo X. Si el proceso B realiza actualizaciones al archivo, se pueden originar conflictos ya que el proceso A estaba actualizando el mismo archivo pero no terminó de hacerlo. Para resolver el problema es necesario que cada uno de los procesos tenga acceso al archivo en forma exclusiva, y que el otro proceso no tenga acceso mientras el primero no haya terminado de realizar la actualización.

Este es solamente un ejemplo de dos procesos que necesitan ser sincronizados. En el área de sistemas operativos, los procesos que utilizan un recurso que debe asignarse en forma exclusiva mientras se realiza una operación, se denominan PROCESOS CONCURRENTES. La sección del proceso que debe tener acceso en forma exclusiva a un recurso se denomina SECCION CRITICA. Existen varias formas de sincronizar procesos concurrentes, en esta sección se presenta una de ellas basada en semáforos.

Un semáforo S es una variable entera que puede ser modificada solamente mediante las operaciones SIGNAL(S) y WAIT(S). Ambas operaciones están definidas como operaciones atómicas, es decir, tiene que ser ejecutadas como un todo y no pueden realizarse parcialmente. Existe una lista de procesos suspendidos asociada a cada semáforo. Los procesos suspendidos se añaden y suprimen de la lista de acuerdo a las operaciones anteriores.

La operación SIGNAL(S) toma el valor de la variable S y lo actualiza decrementándolo en una unidad. Si el resultado no es negativo, puede continuar su operación normal. Si el resultado es negativo, debe suspender su ejecución y se añade a la lista de procesos suspendidos de ese semáforo.

La operación WAIT(S) toma el valor de la variable S y lo actualiza incrementándolo en una unidad. El proceso que realiza la operación siempre continúa con su operación normal. Si el resultado de actualizar la variable S es negativo o cero, el sistema operativo debe reactivar el primer proceso de la lista de procesos suspendidos asociada al semáforo S.

Al definir las operaciones anteriores, se supone que inicialmente el valor de la variable S es uno. Considérese ahora la operación de los procesos A y B que actualizan el archivo X en forma concurrente, sincronizados por el semáforo S.

Cada uno de los procesos tiene que incluir una operación SIGNAL(S) antes de su sección crítica y una operación WAIT(S) después de su sección crítica como se muestra a continuación:

PROCESO A

-
-
-

SIGNAL(S)

- sección crítica de A
actualizar el archivo X
- fin de sección crítica

WAIT(S)

-
-
-

PROCESO B

-
-
-

SIGNAL(S)

- sección crítica de B
actualizar el archivo X
- fin de sección crítica

WAIT(S)

-
-
-

Para entender la operación de los semáforos, supóngase que el proceso A inicia operación y realiza la operación SIGNAL(S), dejando el valor del semáforo en cero. Este proceso continúa actualizando el archivo pero se le acaba el tiempo de procesador y se suspende sin terminar su

sección crítica. Ahora supóngase que inicia operación el proceso B y ejecuta la operación SIGNAL(S). El resultado de esta operación será de -1 y el proceso B se suspenderá, añadiéndolo a la lista de procesos suspendidos asociada al semáforo S. Este proceso ya no puede continuar ejecución.

Cuando el proceso A vuelve a activarse, termina de ejecutar su sección crítica y realiza la operación WAIT(S), dando como resultado un valor de cero para la variable S. El proceso A continúa ejecución, pero dado que el resultado de la operación fue cero, el sistema operativo quitaría el proceso B de la lista de procesos suspendidos por el semáforo S y lo dejaría en la lista de procesos listo. Cuando el proceso B sea reactivado, procederá a ejecutar su sección crítica sin problemas y al final realizará la operación WAIT(S) dejando el valor del semáforo nuevamente en uno.

Si al proceso B se le acabara su tiempo de procesador mientras estuviera ejecutando su sección crítica e iniciara de nuevo su ejecución el proceso A, éste se suspendería al realizar la operación SIGNAL(S) y se quitaría de la lista de procesos suspendidos del semáforo S hasta que el proceso B realizara la operación WAIT(S).

Los semáforos constituyen una importante herramienta de sincronización de procesos concurrentes y pueden utilizarse para sincronizar cualquier número de procesos. El lector podrá comprobar que este esquema también trabaja con tres o más procesos concurrentes.

6.6. MEMORIA VIRTUAL

La memoria virtual constituye un concepto sumamente importante dentro de los sistemas operativos. Cuando un programa está en ejecución, las instrucciones se ejecutan de manera secuencial y no es necesario que esté todo el programa cargado en memoria simultáneamente para su ejecución; solamente se necesita que estén en memoria principal las instrucciones que se están ejecutando en un momento dado, así como los datos a los que hacen referencia.

Para manejar el concepto de memoria virtual, la memoria de la computadora se divide lógicamente en páginas y los programas también se segmentan en páginas. Una página es un conjunto de celdas de memoria de tamaño fijo. Las direcciones a las que hacen referencia los programas se dividen lógicamente en dos partes: número de página y desplazamiento dentro de la página.

El sistema operativo mantiene en memoria una tabla con el número de página y el estado que tiene dicha página. Si la página se encuentra residente en memoria principal, se tiene la dirección a partir de la cual está cargada la página. Si la página no está residente en memoria principal, se tiene la dirección en la cual se encuentra en el almacenamiento secundario destinado para paginación. Generalmente se utiliza un disco magnético como medio de almacenamiento secundario para manejar la memoria virtual.

Cuando se inicia la ejecución de un programa en un ambiente de memoria virtual, el sistema operativo carga a memoria principal la página donde inicia ejecución el programa y actualiza su

tabla de páginas. Cuando el programa hace referencia a una dirección que se encuentra en una página que no está residente en memoria, el sistema operativo carga esa página a memoria principal, del disco de paginación. Probablemente necesite liberar una página de memoria principal para poder cargar la página necesaria para que el programa continúe su ejecución, en este caso, el sistema operativo copiaría al disco de paginación la página que se va a sacar de memoria y actualizaría su tabla de páginas para indicar que ahora se encuentra en almacenamiento secundario. Algunos sistemas operativos mantienen, para cada página residente en memoria principal, una bandera que se enciende cuando la página ha sido modificada. De esta manera, si se va a sacar de memoria principal una página que no ha sido modificada, no tiene que copiarse al disco de paginación.

La dirección física donde se localiza una dirección lógica a la que hace referencia un programa se obtiene sumando el desplazamiento de la dirección lógica a la dirección de inicio de página. Esta última se obtiene de la tabla de páginas.

Como ejemplo considérese una computadora con páginas formadas por 100 posiciones de memoria y una memoria real de 1000 celdas. El desplazamiento dentro de una página es un número entero en el rango de 00 a 99. La cantidad de páginas que caben simultáneamente en la memoria física de la computadora es 10. Las posiciones de memoria están numeradas consecutivamente de la posición 000 a la posición 999. Las páginas en la memoria física de la computadora iniciarían en las direcciones 000, 100, 200, 300, 400, 500, 600, 700, 800 y 900.

Las direcciones lógicas estarían formadas por el número de página y el desplazamiento. Para este ejemplo supóngase que el número de página en la dirección lógica está entre 00 y 99. En la figura 6.1. se muestra gráficamente cómo se calcularía una dirección física generada por un programa que tiene 30 páginas lógicas, numeradas de la 00 a la 29. La tabla de páginas tiene el número de página lógica, su dirección en memoria si está residente y un indicador de estado. El estado "p" indica que la página está residente en memoria y el estado "s" denota que la página está en almacenamiento secundario.

Como ejemplo, supóngase que un programa hace referencia la dirección lógica 2245. Esta dirección se encuentra en la página 22 y tiene un desplazamiento de 45 dentro de esa página. Para obtener la dirección física correspondiente a la dirección lógica 2245, se consulta la tabla de páginas en memoria y se obtiene la dirección de inicio de la página. En la tabla de páginas mostrada en la figura 6.1 se ve que la página 22 está cargada en memoria y que su dirección de inicio corresponde a la dirección física 800. Por lo tanto, la dirección física correspondiente a la dirección lógica 2245 se obtiene sumando el desplazamiento (45) a la dirección de inicio de la página (800), lo cual da como resultado la dirección física 845.

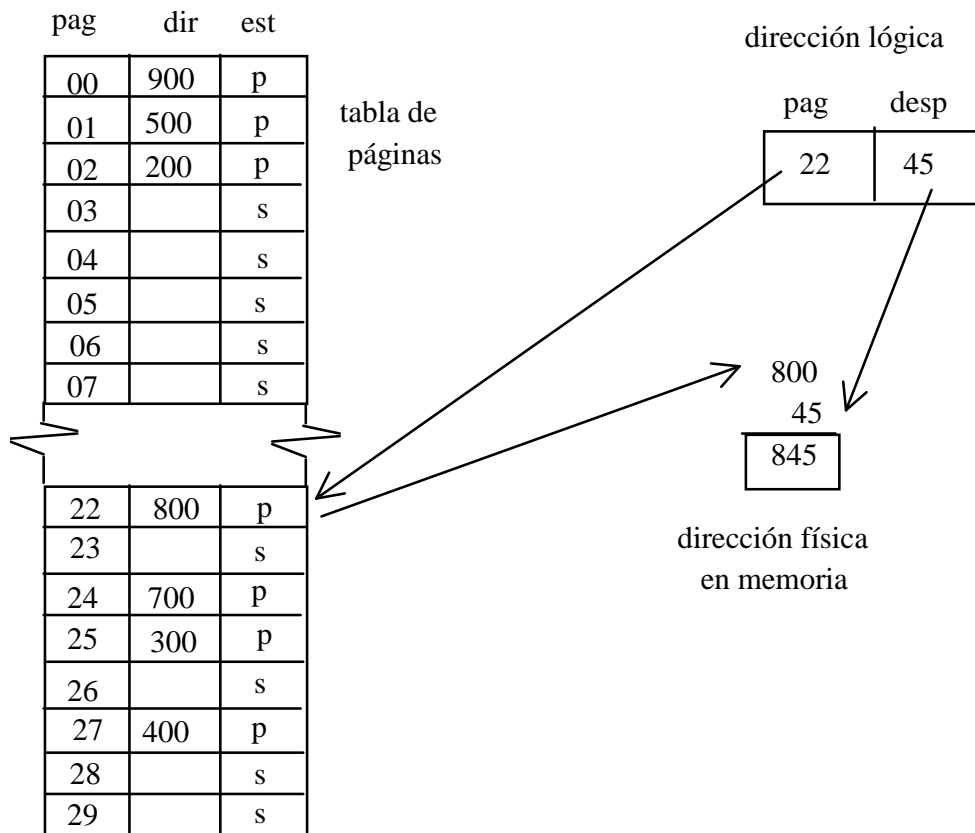


Figura 6.1. Cálculo de la dirección física

Nótese que en este caso un programa puede tener hasta 100 páginas. Si no se tuviera el esquema de paginación y memoria virtual, el tamaño de un programa estaría limitado a 10 páginas como máximo. Las direcciones lógicas pueden estar entre 0000 y 9999, pero las direcciones físicas estarán siempre limitadas a las direcciones reales de memoria; es decir, estarán entre 000 y 999.

La memoria virtual proporciona la ventaja de que se pueden ejecutar programas que no caben totalmente en la memoria principal de un sistema computacional. Cuando se tiene un ambiente con multiprogramación, la memoria virtual permite que se tengan más programas cargados en memoria principal, ya que no es necesario que tengan todas sus páginas residentes en memoria principal simultáneamente.

Para poder implantar la memoria virtual en un sistema computacional, es indispensable que la arquitectura del sistema esté preparada para soportar este ambiente ya que cuando se hace referencia a una página que no está residente en memoria principal, se debe generar una interrupción de falta de página para que el sistema operativo solucione el problema al procesar la interrupción. Este es un claro ejemplo de como los sistemas operativos han afectado la arquitectura de los sistemas computacionales.

6.7. RESUMEN

Un sistema operativo es un conjunto de subprogramas cuya función principal es apoyar la administración de un sistema computacional. También debe proporcionar un ambiente seguro, confiable y amigable para el desarrollo y la ejecución de las aplicaciones.

En este capítulo se presentó la evolución histórica de los sistemas operativos así como los diferentes tipos de sistemas operativos que se han desarrollado para satisfacer diferentes necesidades. Se resaltó el importante papel que los sistemas operativos juegan en un sistema computacional y la necesidad de diseñarlos y desarrollarlos adecuadamente, pensando en todas sus funciones.

Se describieron brevemente los componentes mínimos que debe tener todo sistema operativo y los servicios que debe proporcionar. Finalmente, se describieron de manera sencilla dos de los conceptos más importantes de los sistemas operativos que son la sincronización de procesos y el manejo de la memoria virtual. Para una mayor profundización en esta área se recomienda consultar uno de los muchos libros dedicado a los sistemas operativos

6.8. PROBLEMAS

1. Mencione tres problemas que resuelve un sistema operativo y justifiquen su existencia.
2. Diga cuáles son los tres estados en que puede estar un proceso y descríbalos brevemente.
3. Explique la diferencia entre multiprogramación y multiprocesamiento.
4. ¿Cuál es la función principal de un sistema operativo?
5. Mencione y describa brevemente tres servicios que debe proporcionar un sistema operativo.
6. Mencione tres tipos de sistemas operativos y describa sus características.
7. Explique la operación de los semáforos y mencione para que se utilizan.
8. Explique brevemente en qué consiste el esquema de memoria virtual y mencione las ventajas que proporciona.

CAPITULO 7

SISTEMAS DE NUMEROS

7.1. CONCEPTOS BASICOS

A través de la historia, el hombre ha creado diversos sistemas de números para ayudarse a representar cantidades en forma gráfica. Debido a las dificultades de su uso, muchos de estos sistemas han quedado en desuso, otros han subsistido y han llegado hasta la época actual.

Un sistema de números puede definirse como un conjunto de reglas y símbolos que sirven para representar cantidades. Un buen sistema de números no solamente debe servir para representar cantidades de una manera clara y concisa, sino que además debe facilitar la realización de las operaciones aritméticas. En la actualidad, los dos sistemas más conocidos son el sistema de numeración romano y el sistema decimal. Es obvio que este último tiene muchas ventajas sobre el primero, simplemente trátase de efectuar una suma o multiplicación en el sistema romano y se apreciará la diferencia.

Uno de los mayores logros de la humanidad ha sido el desarrollo de los sistemas de números posicionales. En el sistema romano ya se apreciaban ciertos indicios de un sistema posicional aún y cuando carecía de una estructura científica. Por otra parte, el sistema decimal está desarrollado sobre una sólida base científica y constituye un ejemplo de los sistemas posicionales científicos. El sistema de números maya también tiene una estructura similar a la del sistema decimal. Esta característica del sistema decimal para representar cantidades o valores es, en gran medida, lo que lo ha convertido en el sistema universal. En este capítulo, se estudiarán, los sistemas posicionales científicos, cuya estructura es igual a la del sistema decimal.

7.2. SISTEMAS POSICIONALES CIENTIFICOS

Un sistema de números posicional científico consta de una base, que se representará por el símbolo β , y β guarismos diferentes que se representarán por la letra **a**. β puede ser cualquier cantidad entera mayor que uno. Cada uno de los β guarismos diferentes es un símbolo que representa un valor entero entre cero y $\beta - 1$ inclusive.

El sistema de números también incluirá un elemento que sirva para separar la parte entera del número de la parte fraccionaria del mismo. El símbolo que se usará para este efecto es el punto. La representación de una cantidad en un sistema de números de este tipo se muestra a continuación:

$$A \equiv a_n a_{n-1} \dots a_2 a_1 a_0 . a_{-1} a_{-2} \dots a_{-m} \beta \quad (1-1)$$

La letra **A**, denota el valor de la cantidad. Cada una de las letras a_i , representa el guarismo que ocupa la posición i dentro del número y β representa la base del sistema de números.

El valor que representa el guarismo a_i dentro del número depende de la posición que ocupe, relativa al punto que separa la parte entera de la parte fraccionaria. Este valor está dado por:

$$V(a_i) = a_i \times \beta^i \quad (1-2)$$

Nótese que un guarismo a_i representa por sí solo un valor entero entre cero y $\beta - 1$, pero cuando forma parte de un número, representa ese mismo valor multiplicado por β^i , donde i es la posición que dicho guarismo ocupa dentro del número. La posición cero se encuentra inmediatamente a la izquierda del punto que separa la parte entera de la parte fraccionaria. Las posiciones 1, 2, 3, ..., n , siguen hacia la izquierda de la posición cero y las posiciones -1, -2, -3, ..., $-m$, siguen hacia la derecha de la posición cero. A esta forma de representar números se le conoce como representación yuxtaposicional.

El valor de A , se obtiene al representar al número en su forma polinomial, como se muestra a continuación:

$$V(A) = \sum_{i=-m}^n V(a_i) = \sum_{i=-m}^n a_i \beta^i \quad (1-3)$$

Los símbolos utilizados para representar los guarismos en sistemas numéricos con base menor o igual a 10 son los dígitos usados normalmente en el sistema decimal, en el rango de 0 a la base menos 1. En sistemas numéricos con base mayor que 10, se usan los dígitos para representar a los guarismos que indican valores menores que 10 y, por convención, se usan las letras en orden alfabético para representar guarismos que indican valores mayores a 9.

Como ejemplo de un sistema posicional científico, considérese el sistema decimal. En este sistema, la base β es 10 y los β guarismos diferentes son los símbolos 0, 1, 2, 3, 4, 5, 6, 7, 8, y 9. En el sistema decimal, los guarismos reciben el nombre de dígitos, y representan valores entre cero y nueve (0 a $\beta - 1$) inclusive. El punto que separa la parte entera de la parte fraccionaria se denomina punto decimal. El valor que representa el número 1345.82 (en el sistema decimal) está dado por la fórmula (1-3) donde :

$$\begin{array}{llll} a_3=1 & a_2=3 & a_1=4 & a_0=5 \\ a_{-1}=8 & a_{-2}=2 & \beta=10 & \end{array}$$

Desarrollando la sumatoria dada por (1-3), se obtiene lo siguiente:

$$\begin{array}{llll} a_{-2} \times \beta^{-2} & = & 2 \times \beta^{-2} & = & 0.02 \\ a_{-1} \times \beta^{-1} & = & 8 \times \beta^{-1} & = & 0.80 \\ a_0 \times \beta^0 & = & 5 \times \beta^0 & = & 5.00 \\ a_1 \times \beta^1 & = & 4 \times \beta^1 & = & 40.00 \end{array}$$

$$\begin{array}{rclclcl}
 a_2 \times \beta^2 & = & 3 \times \beta^2 & = & 300.00 \\
 a_3 \times \beta^3 & = & 1 \times \beta^3 & = & \frac{1,000.00}{1,345.82}
 \end{array}$$

Por convención, cuando se omite la base del sistema de números en la que se está representando una cantidad, se supondrá que se está utilizando el sistema decimal ($\beta=10$). Por ejemplo, el número 1345.82 es equivalente a 1345.82_{10} .

A continuación se muestran un conjunto de cruces y posteriormente se muestra la cantidad de cruces que tiene dicho conjunto, representada en sistemas de números con diferentes bases.

```

X      X      X      X      X      X
X      X      X      X      X      X
X      X      X      X      X      X
X      X      X      X      X      X
X      X      X      X      X      X

```

Conjunto de cruces

Base	Representación
10	30
16	1E ₁₆
8	36 ₈
5	110 ₅
4	132 ₄
3	1010 ₃
2	11110 ₂

7.3. CONVERSION DE BASE β A BASE 10

Para obtener la representación de una cantidad en el sistema decimal, dado que se conoce su representación en un sistema con base β , se aplica la fórmula (1-3), realizando las operaciones en el sistema decimal. Si deseara obtener la representación en el sistema decimal de la cantidad representada por 1AC2.D₁₆, se procedería como sigue:

$$V(A) = \sum_{i=-1}^3 a_i \beta^i \quad (1-4)$$

donde

$$a_3=1 \quad a_2=A \quad a_1=C$$

$$a_0=2 \qquad a_{-1}=D \qquad \beta=16$$

Desarrollando la sumatoria se obtiene lo siguiente:

$$\begin{array}{rclclcl}
 a_{-1} \times \beta^{-1} & = & D \times 16^{-1} & = & 0.8125 \\
 a_0 \times \beta^0 & = & 2 \times 16^0 & = & 2.0000 \\
 a_1 \times \beta^1 & = & C \times 16^1 & = & 192.0000 \\
 a_2 \times \beta^2 & = & A \times 16^2 & = & 2,560.0000 \\
 a_3 \times \beta^3 & = & 1 \times 16^3 & = & \underline{4,096.0000} \\
 & & & & 6,850.8125
 \end{array}$$

7.4 CONVERSION DE BASE 10 A BASE β .

Si se tiene la representación de un valor en el sistema decimal y se desea obtener su representación en un sistema de números con base β , es necesario obtener por separado la parte entera y la parte fraccionaria.

Supóngase que la representación de la cantidad deseada en el sistema de números con base β es la siguiente:

$$A = a_n a_{n-1} \dots a_2 a_1 a_0 . a_{-1} a_{-2} \dots a_{-m} \beta \quad (1-5)$$

El valor que representa el número A está dado por la fórmula (1-3):

$$\begin{aligned}
 V(A) = & a_n \times \beta^n + a_{n-1} \times \beta^{n-1} + \dots + a_1 \times \beta^1 + a_0 \times \beta^0 + \\
 & a_{-1} \times \beta^{-1} + a_{-2} \times \beta^{-2} + \dots + a_{-m} \times \beta^{-m}
 \end{aligned}$$

Para convertir un numero B de base 10 a base β , se considerará primero la parte entera del número:

$$B_e = b_n b_{n-1} \dots b_2 b_1 b_0$$

Si se divide B_e entre la base β , se obtiene un cociente C_0 y un residuo r_0 , donde:

$$B_e = C_0 \times \beta + r_0$$

Si ahora se procede a dividir el cociente C_0 entre la base β , se obtiene un cociente C_1 y un residuo r_1 , donde :

$$C_0 = C_1 \times \beta + r_1$$

$$B_e = (C_1 \times \beta + r_1) \times \beta + r_0$$

$$B_e = C_1 \times \beta^2 + r_1 \times \beta + r_0$$

Continuando con el mismo procedimiento hasta que se llegue a obtener un cociente cero, se tiene lo siguiente:

operación	cociente	residuo	B_e
B_e / β	C_0	r_0	$C_0 \times \beta + r_0$
C_0 / β	C_1	r_1	$C_1 \times \beta^2 + r_1 \times \beta + r_0$
C_1 / β	C_2	r_2	$C_2 \times \beta^3 + r_2 \times \beta^2 + r_1 \times \beta + r_0$
...
...
C_{n-2} / β	C_{n-1}	r_{n-1}	$C_{n-1} \times \beta^n + r_{n-1} \times \beta^{n-1} + \dots + r_1 \times \beta + r_0$
C_{n-1} / β	$C_n = 0$	r_n	$r_n \times \beta^n + r_{n-1} \times \beta^{n-1} + \dots + r_1 \times \beta + r_0$

Los residuos que se obtienen, r_0, r_1, \dots, r_n , son cantidades enteras dentro del rango de cero a $\beta - 1$. Cada uno de éstos corresponderá a uno de los β guarismos que tiene el sistema con base β .

Nótese que la expresión

$$B_e = r_n \times \beta^n + r_{n-1} \times \beta^{n-1} + \dots + r_2 \times \beta^2 + r_1 \times \beta + r_0$$

tiene la misma estructura que la parte entera del valor del número buscado; por lo tanto, los residuos r_0, r_1, \dots, r_n , son precisamente los guarismos que forman la parte entera del número a_0, a_1, \dots, a_n .

Para ilustrar el procedimiento anterior, se convertirá la parte entera del número 1487.52 a su representación en un sistema con base 7.

En el sistema con base 7 se tienen los guarismos 0, 1, 2, 3, 4, 5 y 6 que representan las cantidades enteras comprendidas entre cero y seis inclusive.

operación	cociente	residuo
1487 / 7	212	3
212 / 7	30	2
30 / 7	4	2
4 / 7	0	4

Por consiguiente, el valor que representa el número 1487, es el mismo que el representado por el número 4223₇.

Para convertir la parte fraccionaria B_f del número, se procede como sigue:

$$B_f = b_{-1} b_{-2} \dots b_{-m+1} b_{-m}$$

Si se multiplica B_f por la base β , se obtiene un parte entera E_1 y otra fraccionaria F_1 , donde:

$$B_f = \frac{E_1}{\beta} + \frac{F_1}{\beta}$$

Multiplicando la nueva parte fraccionaria F_1 por la base β , se obtiene un parte entera E_2 y una nueva parte fraccionaria F_2 , donde :

$$F_1 = \frac{E_2}{\beta} + \frac{F_2}{\beta}$$

$$B_f = \frac{E_1}{\beta} + \left(\frac{\frac{E_2}{\beta} + \frac{F_2}{\beta}}{\beta} \right) = \frac{E_1}{\beta} + \left(\frac{E_2}{\beta^2} + \frac{F_2}{\beta^2} \right)$$

Continuando con el mismo procedimiento, se obtiene lo siguiente:

operación	entero	fracción	B_f
$B_f \times \beta$	E_1	F_1	$E_1 \times \beta^{-1} + F_1 \times \beta^{-1}$
$F_1 \times \beta$	E_2	F_2	$E_1 \times \beta^{-1} + E_2 \times \beta^{-2} + F_2 \times \beta^{-2}$
$F_2 \times \beta$	E_3	F_3	$E_1 \times \beta^{-1} + E_2 \times \beta^{-2} + E_3 \times \beta^{-3} + F_3 \times \beta^{-3}$
...
...
$F_{m-1} \times \beta$	E_m	F_m	$E_1 \times \beta^{-1} + E_2 \times \beta^{-2} + E_m \times \beta^{-m} + F_m \times \beta^{-m}$

El procedimiento mostrado se continua hasta que la parte fraccionaria F_m sea cero; o bien, hasta que se hayan obtenido una cantidad de guarismos tal que proporcionen la precisión deseada.

Las cantidades $E_1, E_2, E_3, \dots, E_m$, que se obtienen son números enteros dentro del rango de cero a $\beta-1$. Cada uno de estos números enteros corresponderá a uno de los β guarismos que tiene el sistema con base β .

Nótese que la expresión

$$B_f = E_1 \times \beta^{-1} + E_2 \times \beta^{-2} + E_3 \times \beta^{-3} \dots + E_m \times \beta^{-m} + F_m \times \beta^{-m}$$

tiene la misma estructura que la parte fraccionaria del número buscado, al hacer $F_m = 0$. Los enteros $E_1, E_2, E_3, \dots, E_m$ son precisamente los guarismos que forman la parte fraccionaria del número $a_{-1}, a_{-2}, a_{-3}, \dots, a_{-m}$.

Ahora se procederá a convertir la parte fraccionaria del número 1487.52 a su representación en un sistema con base 7.

operación	parte entera	parte fraccionaria
0.52×7	3	0.64
0.64×7	4	0.48
0.48×7	3	0.36
0.36×7	2	0.52
0.52×7	3	0.64
...

Suponiendo que se deseara una precisión de 7^{-5} , se obtendrían solamente $a_{-1}, a_{-2}, a_{-3}, a_{-4}$ y a_{-5} . Por consiguiente:

$$0.52 \cong 0.34323_7$$

y el número 1487.52 equivaldría al número 4223.34323_7 con un error menor que 7^{-5} .

Si se desea la representación exacta del número en base 7, se tendría una serie infinita de guarismos dada por:

$$4223.3432343234323432...._7$$

Esto es similar a la representación en decimal de:

$$\frac{1}{30} = 0.0303030303 \dots$$

o de : $\frac{1}{3} = 0.33333333 \dots$

sin embargo, $1/3$ representado en el sistema con base 3, es simplemente 0.1_3 .

7.5. CONVERSION DE BASE β_1 A BASE β_2

El procedimiento que se siguió para convertir un número de base β a su representación en base 10 puede ser aplicado también para convertir un número en base β_1 a su equivalente en base β_2 , siempre y cuando se realicen todas las operaciones en el sistema de números con base β_2 .

Por ejemplo, para convertir el número 1342_5 a su equivalente en un sistema con base 4, se procedería como sigue:

Se convertirá simultáneamente a base 4 y a decimal, para que se pueda apreciar la equivalencia.

base 10			base 4		
1×5^3	=	125	$14 \times (114)^3$	=	1331_4
3×5^2	=	75	$34 \times (114)^2$	=	1023_4
4×5^1	=	20	$104 \times (114)^1$	=	110_4
2×5^0	=	2	$24 \times (114)^0$	=	21_4
suma		222	suma		3132_4

El problema práctico que presenta este procedimiento es que no se está acostumbrado a realizar operaciones aritméticas en un sistema de base β cuando β es diferente de 10. Por lo tanto, se recomienda convertir el número en base β_1 primero a base 10, y después hacer la conversión a la base β_2 aplicando los procedimientos vistos anteriormente.

Cuando β_1 y β_2 están en el conjunto de bases de potencias de 2 $\{2, 4, 8, 16, \dots\}$, se puede hacer la conversión en una forma directa, usando la relación que tienen estos sistemas con el sistema de base 2. Como se muestra a continuación, un guarismo en base 16, se puede representar por cuatro guarismos en base 2, un guarismo en base 8, se puede representar por tres guarismos en base 2 y un guarismo en base 4, se puede representar por dos guarismos en base 2.

decimal	base 16	base 2	base 8	base 2	base 4	base 2
0	0	0000	0	000	0	00
1	1	0001	1	001	1	01
2	2	0010	2	010	2	10
3	3	0011	3	011	3	11
4	4	0100	4	100		
5	5	0101	5	101		
6	6	0110	6	110		
7	7	0111	7	111		
8	8	1000				
9	9	1001				
10	A	1010				
11	B	1011				
12	C	1100				
13	D	1101				

14	E	1110				
15	F	1111				

Por ejemplo, para convertir el número 67352₈ a su equivalente en un sistema con base 16, se procedería como sigue:

El equivalente en el sistema de base 2 es: 110 111 011 101 010₂. Se agruparon los dígitos binarios de tres en tres para dar mayor claridad al ejemplo.

Agrupando ahora los dígitos binarios de cuatro en cuatro, a partir del punto binario, se tiene: 110 1110 1110 1010₂. El equivalente en el sistema con base 16 es: 6EEA₁₆.

Cuando se agrupa un número para encontrar su equivalente en otra base, la agrupación siempre se hace a partir del punto, hacia la izquierda para encontrar los guarismos de la parte entera y hacia la derecha para encontrar los guarismos de la parte fraccionaria. Si hace falta, se pueden rellenar ambos extremos del número con ceros.

7.6. SUMA DE NUMEROS EN BASE β

Sean $\mathbf{a_x}$ y $\mathbf{b_x}$ dos guarismos cualesquiera de un sistema de números con base β , y sea \mathbf{s} , el resultado que se obtiene al sumar $\mathbf{a_x}$ y $\mathbf{b_x}$.

$$\mathbf{s = a_x + b_x}$$

Dado que $\mathbf{a_x}$ y $\mathbf{b_x}$ son dos guarismos, se tiene que:

$$0 \leq \mathbf{a_x} \leq \beta - 1 \quad (1-6)$$

y,

$$0 \leq \mathbf{b_x} \leq \beta - 1 \quad (1-7)$$

Al sumar $\mathbf{a_x}$ y $\mathbf{b_x}$ se pueden presentar dos casos diferentes:

Caso i) $\mathbf{a_x + b_x < \beta}$

En este caso, el valor de la suma \mathbf{s} , corresponderá al valor que represente otro guarismo $\mathbf{c_x}$; por lo tanto, el resultado de la suma será simplemente $\mathbf{c_x}$.

Caso ii) $\mathbf{a_x + b_x > \beta - 1}$

Ahora la situación es un poco diferente, ya que no existe ningún guarismo que por sí solo represente un valor mayor que $\beta - 1$. La suma s siempre cumplirá la siguiente desigualdad:

$$s < 2\beta - 1$$

Lo anterior se deduce de (1-6) y (1-7). Por consiguiente, el resultado de la suma obviamente tendrá que ser representado por dos guarismos, tal como se muestra a continuación:

$$s = \mathbf{a_x} + \mathbf{b_x} = 1\mathbf{d_x} \quad (1-8)$$

$$0 \leq \mathbf{d_x} \leq \beta - 1 \quad (1-9)$$

donde $\mathbf{d_x}$ es un guarismo tal que:

$$1 \times \beta + \mathbf{d_x} = s$$

El uno a la izquierda de $\mathbf{d_x}$ en (1-8) representa un valor igual a β , y $\mathbf{d_x}$ representa el valor que es necesario sumar a β para obtener el valor de s . La desigualdad (1-9) se obtiene de (1-6) y (1-7) y simplemente permite asegurar que el valor que representa $\mathbf{d_x}$ corresponde a uno de los guarismos del sistema con base β .

A continuación se muestran tablas de suma en algunos sistemas de números con diferente bases. En la primera columna de la tabla se tienen los diferentes valores de $\mathbf{a_x}$, en el primer renglón de la tabla se tiene los diferentes valores de $\mathbf{b_x}$ y la suma de éstos se tiene en la posición de la tabla donde se intersectan las columnas y los renglones.

$\mathbf{a_x} + \mathbf{b_x}$ en base 2 (binario)

$\mathbf{a_x} \setminus \mathbf{b_x}$	0	1
0	0	1
1	1	10

$\mathbf{a_x} + \mathbf{b_x}$ en base 3 (ternario)

$\mathbf{a_x} \setminus \mathbf{b_x}$	0	1	2
0	0	1	2
1	1	2	10
2	2	10	11

$\mathbf{a_x} + \mathbf{b_x}$ en base 4

$\mathbf{a_x} \setminus \mathbf{b_x}$	0	1		2
---------------------------------------	---	---	--	---

0	0	1	2	3
1	1	2	3	10
2	2	3	10	11
3	3	10	11	12

$\mathbf{a_x + b_x}$ en base 5

$a_x \setminus b_x$	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	10
2	2	3	4	10	11
3	3	4	10	11	12
4	4	10	11	12	13

$\mathbf{a_x + b_x}$ en base 16 (hexadecimal)

$a_x \setminus b_x$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Para sumar números cuya representación incluya varios guarismos, se sigue un procedimiento similar.

Considérense los números

$$A \equiv a_n a_{n-1} \dots a_2 a_1 a_0 . a_{-1} a_{-2} \dots a_{-m} \beta$$

$$B \equiv b_n b_{n-1} \dots b_2 b_1 b_0 . b_{-1} b_{-2} \dots b_{-m} \beta$$

Sea S el resultado obtenido al sumar A y B .

$$S \equiv s_n s_{n-1} \dots s_2 s_1 s_0 . s_{-1} s_{-2} \dots s_{-m} \beta$$

El problema consiste en determinar todos y cada uno de los guarismos de S .

Para obtener s_{-m} simplemente basta con sumar a_{-m} y b_{-m} de acuerdo a las reglas anteriores. Como ya sabemos pueden presentarse dos casos:

Caso i) $a_{-m} + b_{-m} \leq \beta - 1$

En este caso $a_{-m} + b_{-m} = s_{-m}$

Caso ii) $a_{-m} + b_{-m} > \beta - 1$

Ahora se tiene que: $a_{-m} + b_{-m} = \beta + s_{-m}$. El resultado de la suma será s_{-m} , y β representará un acarreo para la posición $-(m-1)$.

Para determinar el resto de los guarismos, se procede a realizar las sumas de derecha a izquierda, con una pequeña variante. Esta consiste en contemplar la posibilidad de que al sumar los guarismos de la posición $i-1$, se genere un acarreo para la posición i , y dicho acarreo deberá ser considerado al sumar los guarismos de la posición i . Los dos posibles casos se muestran a continuación:

Caso ii.a) No se generó un acarreo en la posición $i-1$.

En este caso, se obtiene s_i al sumar a_i y b_i en la forma explicada anteriormente.

Caso ii.b)

Se generó un acarreo en la posición $i-1$.

Ahora es necesario obtener la suma de $a_i + b_i + 1$.

Si $a_i + b_i + 1 \leq \beta - 1$, el resultado es simplemente un guarismo s_i y no se genera acarreo para la posición $i + 1$.

Si $a_i + b_i + 1 > \beta - 1$, la suma puede ser expresada como $\beta + s_i$. Dado que a_i y b_i son ambos menores o iguales a $\beta - 1$, la suma $a_i + b_i + 1$ será siempre menor que 2β y; por lo tanto, existirá un guarismo s_i tal que al sumarse con β , dé la suma deseada. De nuevo, el guarismo s_i es el resultado de la suma y β , representa un acarreo para la posición $i + 1$.

A continuación se muestran varios ejemplos de sumas:

$$\begin{array}{r} 111 \ 1 \ 1 \quad <- \text{acarreos} \\ 1341.32_5 \\ \underline{3412.14_5} \\ 10404.01_5 \end{array}$$

$$\begin{array}{r} 1 \quad <- \text{acarreos} \\ 4317.10_9 \\ \underline{7381.46_9} \\ 12708.56_9 \end{array}$$

$$\begin{array}{r} 1 \ 11 \ 1 \quad <- \text{acarreos} \\ 1717.45_8 \\ \underline{3434.55_8} \\ 5354.22_8 \end{array}$$

$$\begin{array}{r} 111 \ 1 \quad <- \text{acarreos} \\ ABCD.18_{16} \\ \underline{CAFE.AB_{16}} \\ 176CB.C3_{16} \end{array}$$

$$\begin{array}{r} 1111111111 \quad <- \text{acarreos} \\ 10111010110_2 \\ \underline{01101101011_2} \\ 100101000001_2 \end{array}$$

$$\begin{array}{r} 11 \ 1 \quad <- \text{acarreos} \\ 123.321_7 \\ \underline{315.606_7} \\ 442.230_7 \end{array}$$

7.7. RESTA DE NUMEROS EN BASE β

Se analizará primeramente el caso de la resta de dos guarismos y posteriormente se tratará la resta de números con varios guarismos.

Sean $\mathbf{a_x}$ y $\mathbf{b_x}$ dos guarismos cualesquiera de un sistema de números con base β y sea \mathbf{r} el resultado que se obtiene al restar $\mathbf{b_x}$ de $\mathbf{a_x}$. Al efectuar la operación $\mathbf{a_x} - \mathbf{b_x}$ se pueden presentar dos casos:

Caso I) $\mathbf{a_x} \geq \mathbf{b_x}$

En este caso $0 \leq \mathbf{r} = \mathbf{a_x} - \mathbf{b_x} \leq \mathbf{a_x}$ y puede ser representado por un guarismo $\mathbf{c_x}$ tal que $\mathbf{c_x} + \mathbf{b_x} = \mathbf{a_x}$.

Caso ii) $\mathbf{a_x} < \mathbf{b_x}$

Ahora no es posible realizar la resta directamente; sin embargo, la operación puede efectuarse mediante un pequeño artificio. Si a $\mathbf{a_x}$ se le suma y se le resta β , su valor no se altera; por consiguiente, la resta puede expresarse como:

$$\mathbf{r} = \mathbf{a_x} - \mathbf{b_x} = (\mathbf{a_x} + \beta - \beta) - \mathbf{b_x} = -\beta + (\beta + \mathbf{a_x}) - \mathbf{b_x}$$

El valor de $\beta + \mathbf{a_x}$ (se representa como $1\mathbf{a_x}$) será siempre mayor que $\mathbf{b_x}$ y existe un guarismo $\mathbf{c_x}$ tal que:

$$\mathbf{c_x} + \mathbf{b_x} = \beta + \mathbf{a_x}$$

y \mathbf{r} puede expresarse como:

$$\mathbf{r} = -\beta + \mathbf{c_x}$$

El resultado de la resta es $\mathbf{c_x}$. La cantidad $-\beta$ representa un acarreo negativo para la siguiente posición a la izquierda.

El resultado se representará como:

$$\mathbf{r} = \mathbf{a_x} - \mathbf{b_x} = \overline{1} \mathbf{c_x}$$

donde $\overline{1}$ representa el acarreo negativo $-\beta$.

A continuación, se muestran tablas para resta de dos guarismos en sistemas con diferentes bases. En la primera columna de la tabla se tienen los diferentes valores de $\mathbf{a_x}$, en el primer renglón de la tabla se tiene los diferentes valores de $\mathbf{b_x}$ y la resta de éstos, se tiene en la posición de la tabla donde se intersectan los renglones y las columnas.

$\mathbf{a_x} - \mathbf{b_x}$ en base 2 (binario)

$\mathbf{a_x} \setminus \mathbf{b_x}$	0	1
0	0	$\overline{1}1$
1	1	0

$\mathbf{a_x} - \mathbf{b_x}$ en base 3 (ternario)

$\mathbf{a_x} \setminus \mathbf{b_x}$	0	1	2
0	0	$\overline{1}2$	$\overline{1}1$
1	1	0	$\overline{1}2$
2	2	1	0

$\mathbf{a_x} - \mathbf{b_x}$ en base 4

$\mathbf{a_x} \setminus \mathbf{b_x}$	0	1	3	3
0	0	$\overline{1}3$	$\overline{1}2$	$\overline{1}1$

1	1	0	$\overline{13}$	$\overline{12}$
2	2	1	0	$\overline{13}$
3	3	2	1	0

$\overline{a_x} - \overline{b_x}$ en base 5

1

$a_x \setminus b_x$	0	1	2	3	4
0	0	$\overline{14}$	$\overline{13}$	$\overline{12}$	$\overline{11}$
1	1	0	$\overline{14}$	$\overline{13}$	$\overline{12}$
2	2	1	0	$\overline{14}$	$\overline{13}$
3	3	2	1	0	$\overline{14}$
4	4	3	2	1	0

$\overline{a_x} - \overline{b_x}$ en base 16 (hexadecimal)

$a_x \setminus b_x$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	$\overline{1F}$	$\overline{1E}$	$\overline{1D}$	$\overline{1C}$	$\overline{1B}$	$\overline{1A}$	$\overline{19}$	$\overline{18}$	$\overline{17}$	$\overline{16}$	$\overline{15}$	$\overline{14}$	$\overline{13}$	$\overline{12}$	$\overline{11}$
1	1	0	$\overline{1F}$	$\overline{1E}$	$\overline{1D}$	$\overline{1C}$	$\overline{1B}$	$\overline{1A}$	$\overline{19}$	$\overline{18}$	$\overline{17}$	$\overline{16}$	$\overline{15}$	$\overline{14}$	$\overline{13}$	$\overline{12}$
2	2	1	0	$\overline{1F}$	$\overline{1E}$	$\overline{1D}$	$\overline{1C}$	$\overline{1B}$	$\overline{1A}$	$\overline{19}$	$\overline{18}$	$\overline{17}$	$\overline{16}$	$\overline{15}$	$\overline{14}$	$\overline{13}$
3	3	2	1	0	$\overline{1F}$	$\overline{1E}$	$\overline{1D}$	$\overline{1C}$	$\overline{1B}$	$\overline{1A}$	$\overline{19}$	$\overline{18}$	$\overline{17}$	$\overline{16}$	$\overline{15}$	$\overline{14}$
4	4	3	2	1	0	$\overline{1F}$	$\overline{1E}$	$\overline{1D}$	$\overline{1C}$	$\overline{1B}$	$\overline{1A}$	$\overline{19}$	$\overline{18}$	$\overline{17}$	$\overline{16}$	$\overline{15}$
5	5	4	3	2	1	0	$\overline{1F}$	$\overline{1E}$	$\overline{1D}$	$\overline{1C}$	$\overline{1B}$	$\overline{1A}$	$\overline{19}$	$\overline{18}$	$\overline{17}$	$\overline{16}$
6	6	5	4	3	2	1	0	$\overline{1F}$	$\overline{1E}$	$\overline{1D}$	$\overline{1C}$	$\overline{1B}$	$\overline{1A}$	$\overline{19}$	$\overline{18}$	$\overline{17}$
7	7	6	5	4	3	2	1	0	$\overline{1F}$	$\overline{1E}$	$\overline{1D}$	$\overline{1C}$	$\overline{1B}$	$\overline{1A}$	$\overline{19}$	$\overline{18}$
8	8	7	6	5	4	3	2	1	0	$\overline{1F}$	$\overline{1E}$	$\overline{1D}$	$\overline{1C}$	$\overline{1B}$	$\overline{1A}$	$\overline{19}$
9	9	8	7	6	5	4	3	2	1	0	$\overline{1F}$	$\overline{1E}$	$\overline{1D}$	$\overline{1C}$	$\overline{1B}$	$\overline{1A}$
A	A	9	8	7	6	5	4	3	2	1	0	$\overline{1F}$	$\overline{1E}$	$\overline{1D}$	$\overline{1C}$	$\overline{1B}$
B	B	A	9	8	7	6	5	4	3	2	1	0	$\overline{1F}$	$\overline{1E}$	$\overline{1D}$	$\overline{1C}$
C	C	B	A	9	8	7	6	5	4	3	2	1	0	$\overline{1F}$	$\overline{1E}$	$\overline{1D}$
D	D	C	B	A	9	8	7	6	5	4	3	2	1	0	$\overline{1F}$	$\overline{1E}$
E	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	$\overline{1F}$
F	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0

Para restar números cuya representación incluya varios guarismos, se sigue un procedimiento similar.

Considérense los números:

$$A \equiv a_n a_{n-1} \dots a_2 a_1 a_0 . a_{-1} a_{-2} \dots a_{-m} \beta$$

$$B \equiv b_n b_{n-1} \dots b_2 b_1 b_0 . b_{-1} b_{-2} \dots b_{-m} \beta$$

Sea R el resultado obtenido al restar A y B.

$$R \equiv r_n r_{n-1} \dots r_2 r_1 r_0 . r_{-1} r_{-2} \dots r_{-m} \beta$$

El problema consiste en la obtención de todos y cada uno de los guarismos de R.

Para determinar r_{-m} simplemente basta con efectuar la operación $a_{-m} - b_{-m}$ de acuerdo a las reglas anteriores. Pueden presentarse dos casos:

Caso i) $a_{-m} \geq b_{-m}$

En este caso $r_{-m} = a_{-m} - b_{-m}$

Caso ii) $a_{-m} < b_{-m}$

Ahora se tiene que: $a_{-m} - b_{-m} = -\beta + r_{-m}$. El resultado de la resta será r_{-m} y $-\beta$ representará un acarreo negativo para la posición $-(m-1)$.

Para determinar el resto de los guarismos, se procede a realizar las restas de derecha a izquierda, con una pequeña variante consistente en contemplar la posibilidad de que al restar los guarismos de la posición $i-1$, se genere un acarreo negativo para la posición i . Dicho acarreo negativo deberá ser considerado al restar los guarismos de la posición i .

Los dos casos posibles se muestran a continuación:

Caso ii.a) No se generó un acarreo negativo en la posición $i-1$.

En este caso se obtiene r_i efectuar la operación $a_{-m} - b_{-m}$, en la forma explicada anteriormente.

Caso ii.b) Se generó un acarreo negativo en la posición $i-1$.

Comúnmente se dice que se prestó una unidad a la posición $i-1$ (la cual vale β en la posición $i-1$) y ahora es necesario corregir este préstamo. La corrección de se puede hacer de dos formas: restándole uno al minuendo de la posición i , o sumándole uno al substraendo de la posición i .

Por lo tanto, el resultado de la posición i depende del resultado de las siguientes restas:

$$(a_i - 1) - b_i \text{ o } a_i - (b_i + 1)$$

Si $(a_i - 1) - b_i = a_i - (b_i + 1) \geq 0$, el resultado es simplemente un guarismo r_i . Dado que a_i y b_i son ambos menores o iguales a $\beta - 1$, la resta $a_i - (b_i + 1)$ será menor que $\beta - 1$. No se genera acarreo negativo para la posición $i+1$.

Si $(a_i - 1) - b_i = a_i - (b_i + 1) < 0$, la resta puede ser expresada como $-\beta + a_i$. Dado que a_i y b_i son ambos menores o iguales a $\beta - 1$, la resta $(a_i - 1) - b_i$ será mayor que $-\beta$ y; por lo tanto, existirá un guarismo r_i tal que al sumarse con $-\beta$, dé el resultado de la resta. El guarismo r_i es el resultado de la resta y $-\beta$ representa un acarreo negativo para la posición $i+1$.

A continuación se muestran varios ejemplos de restas.

$$\begin{array}{r} \bar{1} \quad \bar{1} \quad \bar{1} \quad <- \text{acarreos} \\ 4256.11_7 \\ \underline{1443.15_7} \\ 2512.63_7 \end{array}$$

$$\begin{array}{r} \bar{1} \quad \bar{1} \quad \bar{1} \quad <- \text{acarreos} \\ 1431.12_6 \\ \underline{0333.30_5} \\ 1042.32_5 \end{array}$$

$$\begin{array}{r} \bar{1} \quad \bar{1} \quad \bar{1} \quad \bar{1} \quad \bar{1} \quad \bar{1} \quad <- \text{acarreos} \\ 10110101.010_2 \\ \underline{1010101.101_2} \\ 01011111.101_2 \end{array}$$

$$\begin{array}{r} \bar{1} \quad <- \text{acarreos} \\ AB1F.C_{16} \\ \underline{1789.B_{16}} \\ 9396.1_{16} \end{array}$$

Nótese que para hacer una resta el substraendo debe ser menor que el minuendo.

7.8. RESTAS USANDO COMPLEMENTOS.

En esta sección se realizarán las restas por medio de sumas usando el concepto de complementos.

Para poder definir este concepto se debe limitar el valor de los números que se pueden representar. Para esto, se usarán números enteros en base β cuya representación estará restringida a usar n posiciones.

$$A \equiv a_{n-1} a_{n-2} \dots a_2 a_1 a_0 \beta$$

La menor cantidad que se puede representar usando n posiciones es:

$$A_m \equiv a_m a_m \dots a_m a_m a_m \beta \quad (n \text{ posiciones})$$

donde a_m es el guarismo que representa el valor de 0; por lo tanto, el valor de A_m es 0.

La mayor cantidad que se puede representar usando **n** posiciones es:

$$A_M \equiv a_M a_M \dots a_M a_M a_M \beta \quad (n \text{ posiciones})$$

donde a_M es el guarismo que representa el valor de $\beta - 1$.

El valor de A_M está dado por:

$$V(A_M) = \sum_{i=0}^{n-1} a_i \times \beta^i = \sum_{i=0}^{n-1} a_M \times \beta^i = a_M \times \sum_{i=0}^{n-1} \beta^i = a_M \left(\frac{1 - \beta^n}{1 - \beta} \right)$$

$$V(A_M) = (\beta - 1) \left(\frac{1 - \beta^n}{-(\beta - 1)} \right) = \beta^n - 1 \quad (1-10)$$

En la figura 7.1 se visualiza la gama de números que se pueden representar.

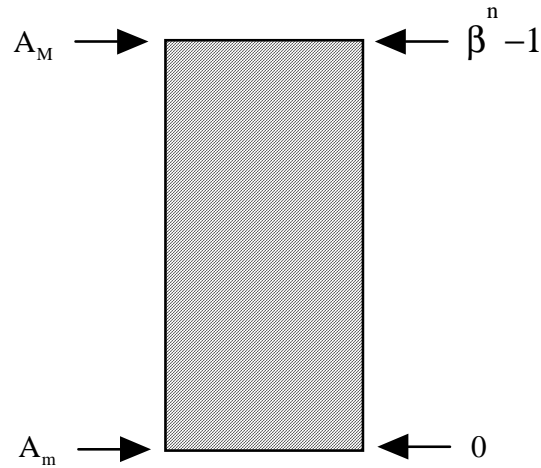


Figura 7.1 gama de números en n posiciones

El complemento de un número A es igual a un número A' , el cual tiene un valor igual a $A_M - A$, como se muestra en la figura 7.2.

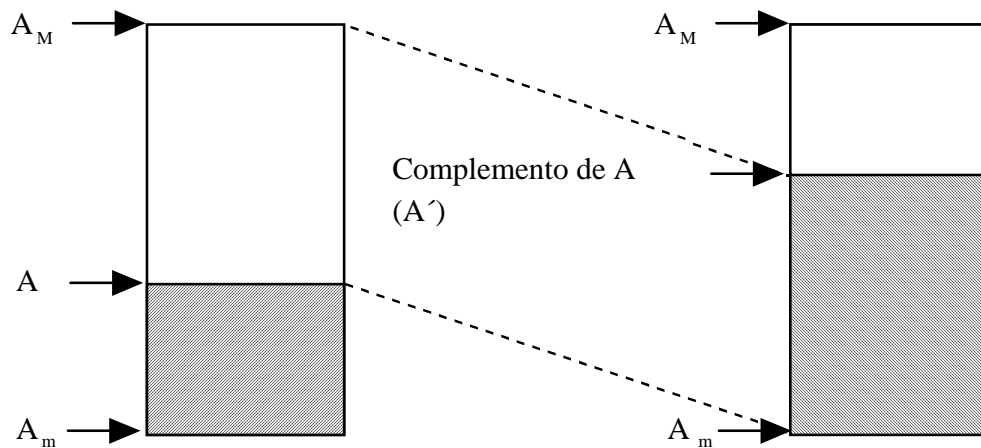


Figura 7.2. Complemento de un número

Al restar dos números, $(A-B)$ lo que se busca es la magnitud de la diferencia entre éstos, como se muestra en la figura 7.3

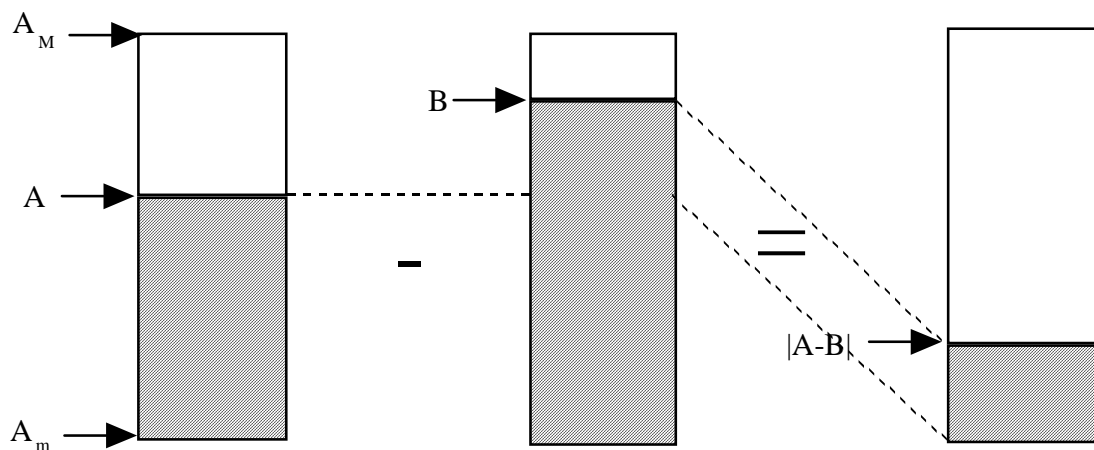


Figura 7.3. Magnitud de la diferencia de dos números

La magnitud de la diferencia también se puede obtener si se suma al número A, el complemento del número B. En la figura 7.4 se ejemplifica esto para el caso en que A es mayor que B.

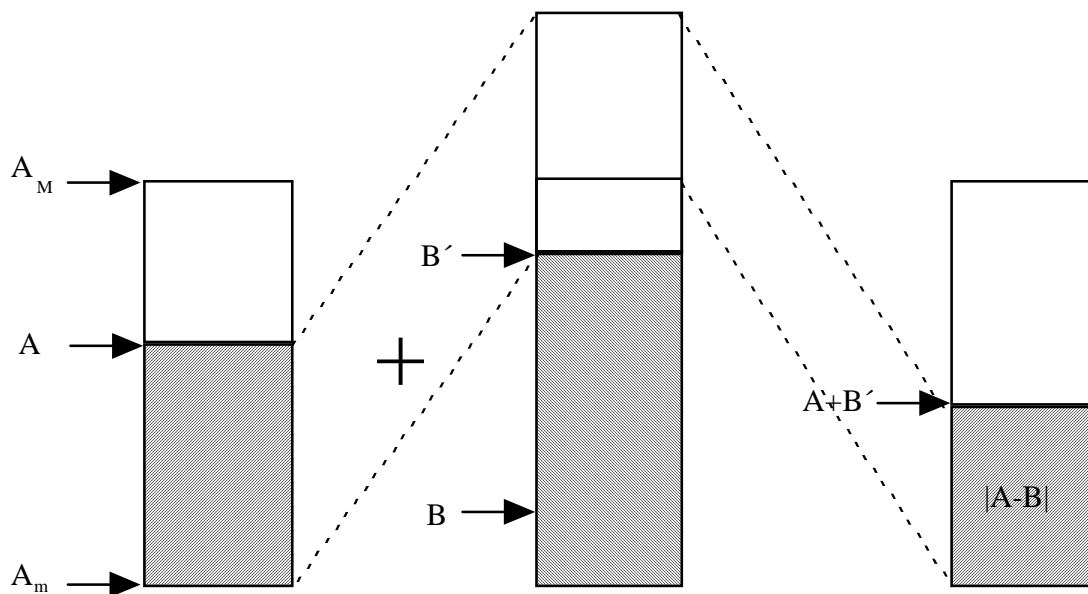


Figura 7.4 Resta $A - B$ usando complementos cuando $A > B$

Como se ve en la figura 7.4, la magnitud de la diferencia es lo que se excede al valor máximo que se puede representar. Al haber limitado los números a n posiciones, la magnitud de la diferencia quedará en las n posiciones y en la posición $n+1$ se tendrá un 1 indicando que ocurrió un desborde.

Para el caso en que B es mayor que A , la magnitud de la diferencia entre A y B es el complemento del resultado, como se observa en la figura 7.5.

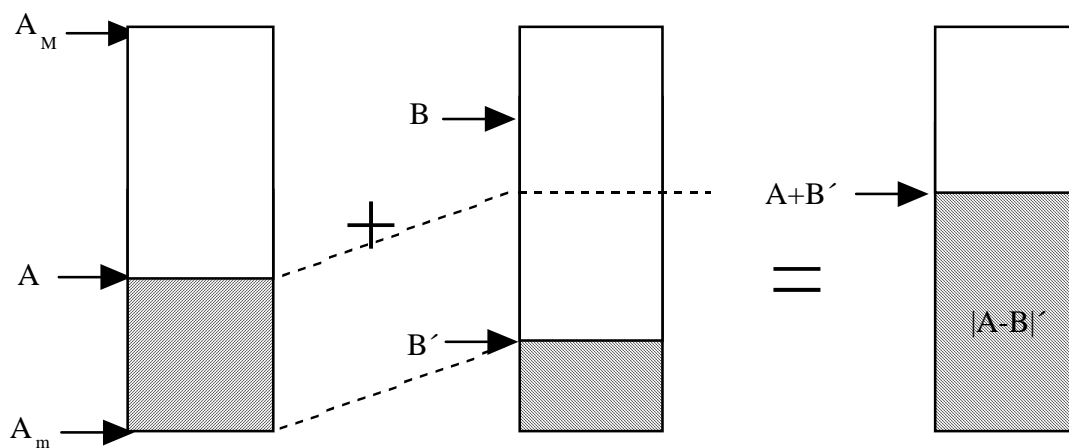


Figura 7.5. Resta $A - B$ usando complementos cuando $A < B$

Existen dos formas de definir un complemento: complementos a la base y complementos a la base disminuida.

El complemento a la base (β) de un número A , se define como:

$$A^* \equiv \beta^n - A \quad (1-11)$$

El complemento a la base disminuida ($\beta-1$) de un número A , se define como:

$$\overline{A} \equiv \beta^n - 1 - A \quad (1-12)$$

De (1-10), (1-11) y (1-12) se deduce que $A^* > 0$ y $\overline{A} \geq 0$.

Obsérvese que :

$$(A^*)^* = \beta^n - A^* = \beta^n - (\beta^n - A) = A$$

y

$$(\overline{A}) = \beta^n - 1 - \overline{A} = \beta^n - 1 - (\beta^n - 1 - A) = A$$

Lo anterior significa que el complemento a la base de A^* , es el número original A . Lo mismo se aplica a los complementos a la base disminuida.

7.8.1 RESTA USANDO COMPLEMENTOS A LA BASE (complementos a β).

Supóngase que se desea efectuar la resta de dos números en base β usando el complemento a la base para efectuar la operación.

Sean A y B dos números en base β , limitados a n posiciones.

Considérense ahora la siguiente operación:

$$R = A + B^* = A + (\beta^n - B)$$

Al efectuar esta operación, pueden presentarse dos casos, los cuales se analizan a continuación:

Caso i) $A \geq B$

Para este caso, el resultado de la resta es $(A-B)$.

El resultado obtenido en R es :

$$R = A + (\beta^n - B) = \beta^n + (A - B)$$

Dado que $A \geq B$, $A - B$ siempre será mayor o igual a cero y β^n representará un acarreo fuera de las n posiciones. Descartando dicho acarreo, el resultado obtenido es el correcto.

Caso ii) $A < B$

Ahora el resultado correcto debe ser $-(B - A)$.

El resultado obtenido en R es:

$$R = A + (\beta^n - B) = \beta^n - (B - A)$$

En este caso, $\beta^n - (B - A)$ es el complemento a la base de la magnitud de la resta. El resultado correcto se obtiene complementando R a la base, y anteponiéndole un signo negativo. Esto es:

$$-R^* = -(\beta^n - (\beta^n - (B - A))) = -(B - A)$$

Del análisis anterior para restar números usando complementos a la base se derivan las siguientes reglas:

- Para realizar la operación $C = A - B$, se efectúa la operación $R = A + B^*$.
- Si se produce un acarreo fuera de las n posiciones, dicho acarreo se desecha y en R queda el resultado correcto ($C = R$).
- Si no se produce un acarreo fuera de las n posiciones, en R queda el complemento a la base de la magnitud del resultado y el resultado es negativo. El resultado correcto es ($C = -R^*$).

Ejemplos:

Para efecto de estos ejemplos se consideran $n = 5$ y $\beta = 6$.

caso i) $A > B$

	1 11	<- acarreos	Resultado
43151 ₆	43151 ₆		correcto
- 31511 ₆	<u>24045₆</u>		C = R = 11240 ₆
	1) 11240 ₆		

Como se generó un acarreo fuera de las n posiciones, debe desecharse dicho acarreo. El resultado correcto es el obtenido de la suma ($C = R$).

Caso ii) $A < B$

$\begin{array}{r} 31511_6 \\ - 43151_6 \\ \hline \end{array}$	$\begin{array}{r} 1 \ 1 \quad \leftarrow \text{acarreo} \\ 31511_6 \\ \underline{12405_6} \\ 0)44320_6 \end{array}$	<p>Resultado correcto</p> <p>$C = - R^* = - 11240_6$</p>
---	---	---

En este caso no se generó un acarreo fuera de las n posiciones. El resultado correcto es negativo y su magnitud es el complemento a la base del resultado de la suma ($C = - R^*$).

7.8.1 RESTA USANDO COMPLEMENTOS A LA BASE DISMINUIDA (complementos a $\beta-1$).

Ahora supóngase que se desea efectuar la resta de dos números en base β usando el complemento a la base disminuida para efectuar la operación.

Sean A y B dos números en base β , limitados a n posiciones. Considérese ahora la siguiente operación:

$$R = A + \bar{B} = A + (\beta^n - 1 - B)$$

Al realizar esta operación pueden presentarse dos casos, los cuales se analizan a continuación:

Caso i) $A > B$

Para este caso, el resultado de la resta es $(A - B)$.

El resultado obtenido en R es :

$$R = A + \bar{B} = A + (\beta^n - 1 - B) = \beta^n + (A - B) - 1$$

Dado que $A > B$, $A - B - 1$ siempre será mayor o igual a cero y β^n representa un acarreo fuera de las n posiciones. Para obtener el resultado correcto es necesario descartar el acarreo y realizar una corrección a R , sumándole 1.

$$C = R + 1 = A - B$$

Caso ii) $A \leq B$

Ahora el resultado correcto debe ser $-(B - A)$.

El resultado obtenido en R es:

$$R = A + \bar{B} = A + (\beta^n - 1 - B) = \beta^n - 1 - (B - A)$$

En este caso, R contiene el complemento a la base disminuida de la magnitud del resultado. El resultado correcto se obtiene complementando R a la base disminuida, y anteponiéndole un signo negativo.

$$C = - \bar{R} = - (B - A)$$

Analizando los resultados anteriores, se pueden deducir las siguientes reglas para restar dos números usando complementos a la base disminuida:

- Para realizar la operación $C = A - B$ se efectúa la operación $R = A + \bar{B}$.
- Si se produce un acarreo fuera de las n posiciones, dicho acarreo se desecha y el resultado correcto es $C = R + 1$ (se efectúa la corrección).
- Si no se produce un acarreo fuera de las n posiciones, en R queda el complemento a la base disminuida de la magnitud del resultado y el resultado es negativo. El resultado correcto es $(C = - \bar{R})$.

Ejemplos:

Se considerarán los mismos ejemplos que en la sección anterior ($n = 5$ y $\beta = 6$).

caso i) $A > B$

	1 1	<- acarreos	Resultado
43151 ₆	43151 ₆		correcto
- 31511 ₆	<u>24044₆</u>		$C = R + 1 = 11240_{\underline{6}}$
	1)11235 ₆		

Como se generó un acarreo fuera de las n posiciones, debe desecharse dicho acarreo y realizar la corrección del resultado. El resultado correcto es el obtenido de la suma con la corrección ($C = R + 1$).

Caso ii) $A < B$

	1	<- acarreos	Resultado
31511 ₆	31511 ₆		correcto
- 43151 ₆	<u>12404₆</u>		$C = - \bar{R} = - 11240_{\underline{6}}$
	0)44315 ₆		

Dado que no se generó un acarreo fuera de las n posiciones, el resultado es negativo. En R está el complemento a la base disminuida de la magnitud del resultado. El resultado correcto se obtiene complementando R a la base disminuida, y anteponiéndole un signo negativo.

7.9. RESUMEN

En el presente capítulo se describen los fundamentos de los sistemas de números posicionales científicos así como las ventajas que se obtienen de un sistema de números con esta estructura.

Se describe como convertir un número representado en su sistema posicional científico con una base diferente a 10, a su representación en el sistema con base 10. También se muestra como representar un número en un sistema con cualquier base, dada su representación en el sistema con base 10.

A manera de ejemplo, se muestra el proceso general para representar un número en un sistema con una base, dada su representación en un sistema con otra base diferente, siendo ambas bases diferentes de la base 10.

Se realizaron operaciones de suma y resta de números en cualquier base. Para las operaciones de resta, se introdujeron los conceptos de complementos a la base y complementos a la base disminuida.

Los complementos permiten realizar las operaciones de resta como operaciones de suma que son, en principio, más simples de efectuar.

Para realizar una resta de números enteros con signo, restringidos a n posiciones, usando complementos a la base, se obtiene el complemento a la base del substraendo y se suma al minuendo. Si se produce un acarreo fuera de las n posiciones, dicho acarreo se desecha y el resultado obtenido es el correcto. Si no se produce acarreo fuera de las n posiciones, el resultado es obtenido es el complemento a la base de la magnitud del resultado, el resultado correcto es negativo y su magnitud se obtiene complementando a la base el resultado obtenido.

Para efectuar una resta de números enteros con signo, restringidos a n posiciones, usando complementos a la base disminuida, se obtiene el complemento a la base disminuida del substraendo y se suma al minuendo. Si se produce un acarreo fuera de las n posiciones, es necesario sumar este acarreo al resultado obtenido para tener el valor correcto de la resta. Si no se produce acarreo fuera de las n posiciones, el resultado es obtenido es el complemento a la base disminuida de la magnitud del resultado, el resultado correcto es negativo y su magnitud se obtiene complementando a la base disminuida el resultado obtenido.

7.10. PROBLEMAS

1. Convierta los siguientes números en base 10 a base β .

- | | |
|---------------------|----------------------|
| a) 1341.72 a base 6 | b) 4132.10 a base 16 |
| c) 12.8 a base 2 | d) 3148.1 a base 7 |

2. Convierta los siguientes números en base β a base 10.

- | | |
|---------------|----------------|
| a) 1341.7_8 | b) 131.311_4 |
|---------------|----------------|

c) $17AB.7_{12}$

d) $ABCD.E_{16}$

3. Convierta los siguientes números en base β_1 a base β_2 .

a) 1341.7_8 a base 4. b) $C0CA.CAFE_{16}$ a base 8

c) 0101.11_4 a base 12

4. Efectúe las siguientes sumas en la base β .

a) $101101.010_2 + 101111.101_2$ b) $1345.12_6 + 5114.55_6$

c) $1AB1.C_{16} + 3112.D_{16}$

d) $1212.11_3 + 221.22_3$

e) $14151.72_8 + 1617.15_8$

f) $437.82_9 + 781.37_9$

5. Efectúe las siguientes restas en la base β .

a) $101111.101_2 - 101101.010_2$ b) $5114.55_6 - 1345.12_6$

c) $3112.D_{16} - 1AB1.C_{16}$

d) $1212.11_3 - 0221.22_3$

e) $31617.15_8 - 14151.72_8$

f) $781.37_9 - 437.82_9$

6. Obtenga el complemento a la base y a la base disminuida de los siguientes números en base β . (use $n = 6$)

a) 101101_2

b) $AF1D17_{16}$

c) 144132_5

d) 462443_7

e) 678123_{11}

7. Efectúe las siguientes restas usando complementos a la base y complementos a la base disminuida. (use $n = 6$)

a) $C1C2AB_{16} - F1F2F3_{16}$

b) $132133_4 - 312121_4$

c) $101011_2 - 100111_2$

CAPITULO 8

REPRESENTACION DE DATOS.

Las computadoras digitales son máquinas capaces de almacenar en su memoria gran cantidad de datos e instrucciones que les indican cómo procesar esta información.

Para representar tanto los datos como las instrucciones, estas máquinas usan el sistema binario. El uso de este sistema no constituye un simple capricho de los diseñadores de computadoras, sino que obedece a ciertas razones que facilitan su construcción y proporcionan un alto grado de confiabilidad en su operación.

La principal razón de usar el sistema binario tiene que ver con la construcción de dispositivos electromecánicos y electrónicos usados en la fabricación de computadoras. Es mucho más sencillo construir un dispositivo que pueda encontrarse en uno de dos estados posibles, que construir otro que pueda encontrarse en uno de diez estados diferentes. Por ejemplo, es relativamente fácil construir un dispositivo que haga que un foco esté apagado o encendido (dos estados); pero construir un dispositivo que haga que un foco se encuentre en uno de diez estados posibles entre apagado y completamente encendido tiene mayor complicación.

La otra razón fuerte involucra la identificación del estado en que se encuentra un dispositivo. En el ejemplo anterior es muy fácil decir cuando un foco está apagado o cuando está encendido, pero es muy difícil decir en cuál estado se encuentra si puede estar en uno de diez estados diferentes. Esta última razón se relaciona directamente con la confiabilidad de las computadoras.

Este capítulo presentará la utilización del sistema binario en la representación de diferentes tipos de datos en las computadoras digitales.

8.1. TIPOS DE DATOS.

Las computadoras digitales manejan cuatro tipos básicos de datos:

- Números enteros con signo.
- Números reales con signo.
- Números decimales codificados en binario (BCD).
- Caracteres.

El número de bits que se usan para representar cada uno de los cuatro tipos básicos de datos y el significado que tiene cada bit varían de acuerdo al tipo de dato y el sistema computacional (hardware y software) que se esté usando.

Estos cuatro tipos básicos de datos pueden agruparse en estructuras más complejas que permitan manejar información de muy diversas clases.

En este capítulo se presentarán los formatos más comunes, que han sido usados para representar cada uno de los tipos de datos básicos.

8.2. REPRESENTACION DE NUMEROS ENTEROS CON SIGNO.

Existen tres representaciones que han sido usadas para manejar internamente números enteros con signos en las computadoras digitales.

Estas representaciones son:

- Magnitud y signo.
- Complementos a 2 (complementos a la base).
- Complementos a 1 (complementos a la base disminuida).

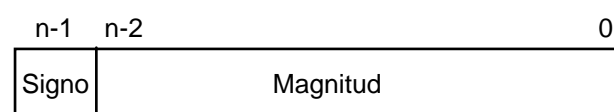
En todas las representaciones se restringe la cantidad de posiciones (bits) que se tienen para representar los números.

La más sencilla de las tres es la representación en magnitud y signo; sin embargo, es la que ocasiona mayores problemas al realizar operaciones aritméticas. Esta representación fue usada en las primeras computadoras digitales pero actualmente ha caído en desuso.

Las dos representaciones usando complementos permiten efectuar operaciones aritméticas con relativa facilidad, esto ha contribuido a que ambas hayan sido adoptadas en las computadoras actuales, siendo la representación en complementos a 2 la más usada. A continuación se detalla cada una de las tres formas de representar números enteros con signo, considerándose que se tienen solamente n bits para su representación.

8.2.1 Representación en magnitud y signo.

Esta forma de representar números enteros con signo usa un bit para el signo y $n-1$ bits para la magnitud. Los bits se numerarán de derecha a izquierda de 0 a $n-1$. El bit $n-1$ se utiliza para el signo y los bits 0 a $n-2$ para la magnitud del número. El formato general es el siguiente:



Un 0 en el bit de signo indica que el número es positivo y un 1 significa que es negativo. Los $n-1$ bits usados para la magnitud contiene el valor absoluto de la cantidad representada en el sistema binario.

El valor mayor que se puede representar con este formato es $2^{n-1} - 1$ y el menor, algebraicamente, es $-(2^{n-1} - 1)$.

En la figura 8.1 se muestra la distribución de los números usando este formato.

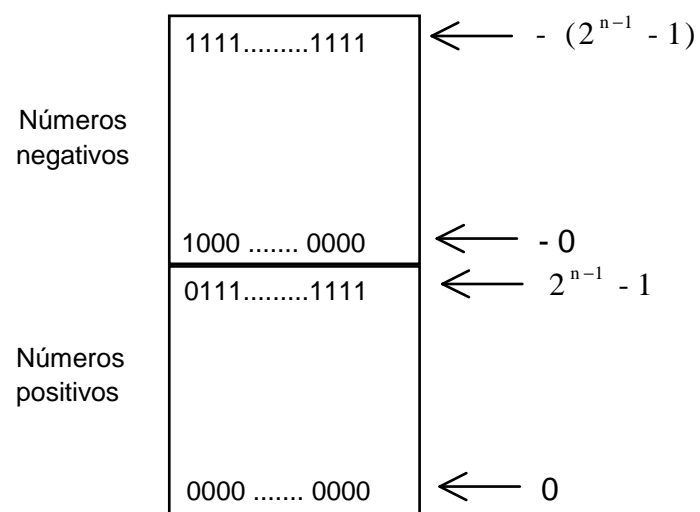


Figura 8.1 Distribución de los números usando signo y magnitud.

A continuación se dan algunos ejemplos:

Si se usan 3 bits ($n=3$)

Representación en magnitud y signo	Decimal
111	-3
110	-2
101	-1
100	-0
011	3
010	2
001	1
000	0

Si se usan 8 bits ($n=8$)

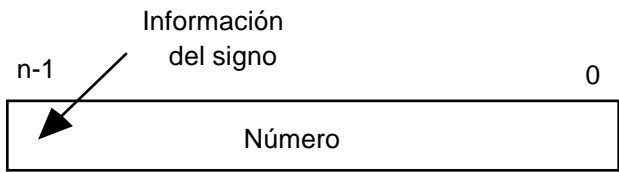
Representación en magnitud y signo	Decimal
11111111	-127
\vdots	\vdots
11001110	-78
\vdots	\vdots
10011001	-25
\vdots	\vdots
10000001	-1
10000000	-0
01111111	127
\vdots	\vdots
01001110	78

⋮	⋮
00011001	25
⋮	⋮
00000001	1
00000000	0

La cantidad cero no tiene signo; sin embargo, este formato permite dos representaciones equivalentes para dicho número.

8.2.2 Representación usando complementos a 2.

En esta forma de representación de números enteros con signo, el bit más significativo del número contiene la información del signo, como se muestra a continuación:



Un 0 en el bit más significativo indica que el número es positivo y un 1 significa que es negativo.

Los n bits se usan para representar el número. Si el número es positivo, los n bits contienen la representación binaria del valor deseado. Si el número es negativo, los n bits contienen el complemento a dos del valor deseado, representado en binario.

El valor mayor que puede ser representado en este formato es $2^{n-1} - 1$ y el menor, algebraicamente, es -2^{n-1} .

En la figura 8.2 se muestra la distribución de los números usando este formato.

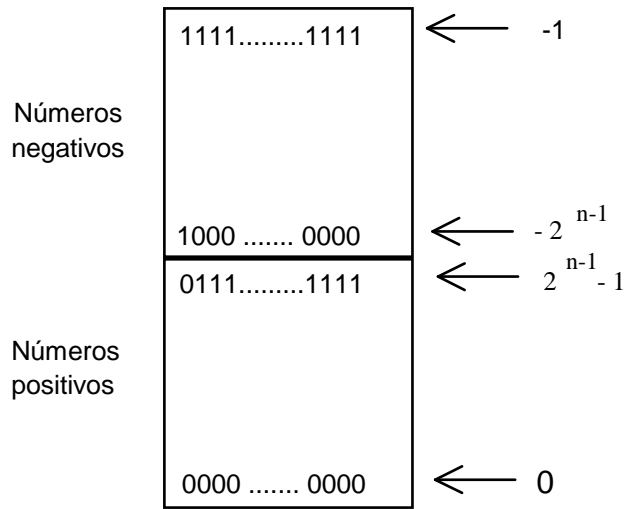


Figura 8.2 Distribución de los números usando complementos a 2.

A continuación se dan algunos ejemplos:

Si se usan 3 bits ($n=3$)

Representación en complementos a dos	Decimal
111	-1
110	-2
101	-3
100	-4
011	3
010	2
001	1
000	0

Si se usan 8 bits ($n=8$)

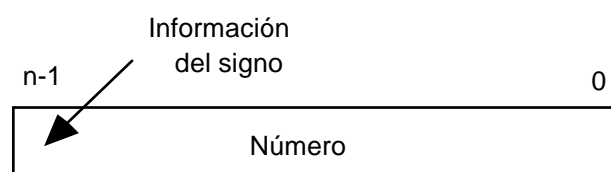
Representación en complementos a dos	Decimal
11111111	-1
11111110	-2
⋮	⋮
11100111	-25
⋮	⋮
10110010	-78
⋮	⋮
10000001	-127
10000000	-128
01111111	127
⋮	⋮
01001110	78
⋮	⋮
00011001	25
⋮	⋮
00000001	1
00000000	0

Nótese que el cero tiene una representación única en este formato y que el complementos a dos de un número negativo es el mismo número positivo.

Dado un número en binario se puede obtener su complemento a dos en forma directa si se aplica la siguiente regla: se analiza el número de derecha a izquierda, el número en complementos a dos es igual al número original hasta encontrar el primer 1 inclusive, después de éste, se invierten todos los bits (los unos a ceros y los ceros a unos).

8.2.3 Representación usando complementos a 1.

Al representar números enteros con signo en esta representación, el bit más significativo del número tiene información del signo, como se muestra a continuación:



Un 0 en el bit más significativo indica que el número es positivo y un 1 significa que es negativo. Los n bits se usan para representar el número. Si el número es positivo, los n bits contienen la representación binaria del valor deseado. Si el número es negativo, los n bits contienen el complemento a unos del valor deseado, representado en binario.

El mayor valor que puede ser representado en este formato es $2^{n-1} - 1$ y el menor, algebraicamente, es $-(2^{n-1} - 1)$.

En la figura 8.3 se muestra la distribución de los números usando este formato.

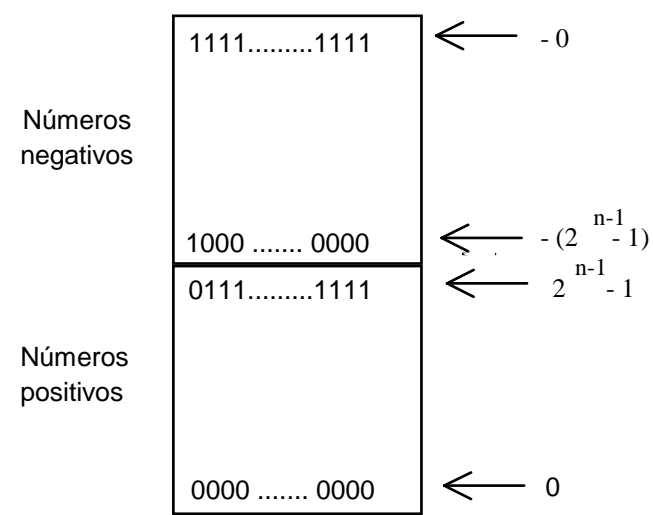


Figura 8.3 Distribución de los números usando complementos a 1.

A continuación se dan algunos ejemplos:

Si se usan 3 bits (n=3)

Representación en complementos a uno	Decimal
111	-0
110	-1
101	-2
100	-3
011	3
010	2
001	1
000	0

Si se usan 8 bits (n=8)

Representación en complementos a uno	Decimal
11111111	0
00000001	1
⋮	⋮
11001110	-49
⋮	⋮
10011001	-102

⋮	⋮
10000001	126
10000000	-127
01111111	127
⋮	⋮
01001110	78
⋮	⋮
00011001	25
⋮	⋮
00000000	0

Nótese que el cero también tiene dos representaciones en este formato y que el complementos a unos de un número negativo es el número positivo equivalente.

Dado un número en binario se puede obtener su complemento a uno en forma directa si se aplica la siguiente regla: el número en complementos a uno se obtiene si se invierten todos los bits del número original (los unos a ceros y los ceros a unos).

8.3. REPRESENTACION DE NUMEROS REALES.

Para poder indicar una forma conveniente de representar números reales en la computadora digital es necesario determinar el orden de magnitud de las cantidades con las que se desea trabajar y la precisión con la que se desea representar estas cantidades.

Muchas de las aplicaciones del área de ingeniería requieren manejar tanto cantidades del orden de 10^{30} como cantidades del orden de 10^{-30} , con un cierto grado de precisión. Esto da una idea del orden de magnitud de las cantidades que es necesario representar internamente en una computadora digital.

El problema que presenta el manejo de cantidades de muy diferente orden de magnitud ha sido resuelto en ingeniería y ciencias usando lo que se conoce como notación científica. Un número decimal expresado en esta notación consta de una parte entera, una parte fraccionaria y una potencia de diez por la cual están multiplicadas las partes entera y fraccionaria para obtener la cantidad deseada. La parte entera debe ser mayor que cero y menor que diez, el número de dígitos que contiene la parte fraccionaria depende de la precisión deseada y la potencia de diez debe ser tal que al multiplicar las partes entera y fraccionada por ella, se obtenga la cantidad que se desea representar. La parte entera más la parte fraccionaria se denomina mantisa.

El formato general de la notación científica (en decimal) es el siguiente:

$$D \equiv S \, d_0 . d_{-1} d_{-2} \cdots d_{-m} \times 10^E$$

Donde:

- D - Es la cantidad representada
- S - Es el signo del número (+,-).
- d_i - Son dígitos (guarismos del sistema decimal).
- E - Es un número entero con signo.

- m - Depende de la precisión deseada.

Sea C una cantidad real y $V(C)$ el valor que representa C . Sea D la representación de C en notación científica y $V(D)$ el valor que representa D .

Dado que D contiene un número finito de dígitos, es imposible representar exactamente la cantidad C en ciertos casos ($1/3$ por ejemplo), lo cual implica que existe un cierto error en la representación de la cantidad. A continuación se definen dos tipos de errores en los que se incurre al representar una cantidad:

$$\text{Error absoluto: } e_a \equiv |V(C) - V(D)|$$

$$\text{Error relativo: } e_r \equiv \frac{e_a}{|V(C)|}$$

El error absoluto no da ninguna idea acerca de la precisión de la representación. Por ejemplo, un error absoluto de una unidad en diez es muy grande, pero el mismo error absoluto en un millón de unidades en la mayoría de los casos es despreciable.

El error relativo da una clara idea de la precisión de una representación. Para ilustrar esto considérese el ejemplo anterior:

Cantidad C	Error absoluto (e_a)	Error relativo (e_r)
10	1	$0.1 = 10\%$
1000000	1	$0.000001 = 0.0001\%$

El error relativo en que se incurriría al representar una cantidad C en notación científica con m+1 dígitos será siempre menor que 10^{-m} . En este caso se dice que la representación tiene una precisión de 10^{-m} , o bien, que tiene m cifras significativas. El error relativo es una medida de la precisión de una representación y depende exclusivamente del número de cifras significativas que se usen.

Al obtener la notación científica de una cantidad real, existen dos formas para limitar el número de guarismos a m+1.

Estas formas son:

- Truncar: se ignoran los guarismos menos significativos.
- Redondear: Si el valor que representa los guarismos que no se pueden incluir es mayor o igual a $\frac{\beta^{-m}}{2}$ se suma β^{-m} a la cantidad representada en la notación científica.

Redondear es la forma más usada en la actualidad.

La notación científica se puede extender a cualquier sistema posicional científico, independiente de la base del mismo. El formato general es el siguiente:

$$A \equiv S a_0 . a_{-1} a_{-2} \cdots a_{-m} \beta \times \beta^E$$

Donde:

- A - Es la representación de la cantidad
- S - Es el signo del número (+,-).
- a_i - Son guarismos del sistema base β .
- E - Es un número entero con signo (exponente).
- m - Número de guarismos significativos.

El error relativo que tiene esta representación es menor que β^{-m} . Por consiguiente; la representación tendrá una precisión de β^{-m} o bien, m guarismos significativos.

Para representar números reales en las computadoras digitales se usan formas muy similares a la notación científica. En las siguientes secciones se definen los formatos más usados.

8.3.1 Formato con exponente polarizado y mantisa representada en signo y magnitud.

Una de las formas más usadas para representar números reales en este formato, toma como base la siguiente representación:

$$A \equiv S \ 0 . a_{-1} a_{-2} \cdots a_{-m} {}_2 \times 10_2^{E_2} \equiv S \ M_2 \times 10_2^{E_2} \quad (8-1)$$

La representación anterior consta de los siguientes elementos:

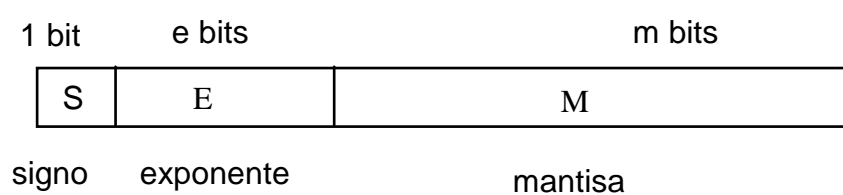
- Una mantisa, $(M_2 = 0 . a_{-1} a_{-2} \cdots a_{-m} {}_2)$, representada en binario y normalizada también en binario. Esto significa que M_2 es un numero binario mayor o igual que 2^{-1} y menor que 1 ($2^{-1} \leq M_2 < 1$).
- Un signo para la mantisa
- Un exponente entero con signo representado en binario (E_2), que indica, la potencia a la que tiene que elevarse la base (10_2).

A continuación se muestran dos números expresados en la forma anterior con 8 bits significativos:

$$125 = 1111101 {}_2 = +0.1111101 {}_2 \times 10_2^{+7} = 0.11111010 {}_2 \times 10_2^{+11} {}_2$$

$$-0.0625 = -0.0001 {}_2 = -0.1 {}_2 \times 10_2^{-3} = -0.10000000 {}_2 \times 10_2^{-11} {}_2$$

El formato que se utiliza en las computadoras digitales para representar números en la notación descrita anteriormente es el siguiente:



El bit de signo es uno cuando el número es negativo y cero en caso contrario.

Los m bits en M representan la parte fraccionaria de la mantisa normalizada en binario descrita anteriormente, la parte entera de la mantisa y el punto (0.) no ocupan un lugar en el formato ya que éstos son constantes y se supone que se tienen.

Los e bits de E representan el exponente en binario, polarizado en 2^{e-1} . La polarización se usa para poder representar números positivos y negativos usando solamente números positivos, lo cual facilitará la comparación de exponentes. Al polarizar un número, lo que se hace es darle el valor cero a la cantidad usada en la polarización (2^{e-1}); las cantidades mayores que ésta serán los números positivos, y las cantidades menores que ésta serán los números negativos. Esto se muestra en la figura 8.4.

Para encontrar el exponente polarizado E, se suma la polarización (2^{e-1}) al exponente. Nótese que si el exponente es negativo, realmente se efectúa una resta.

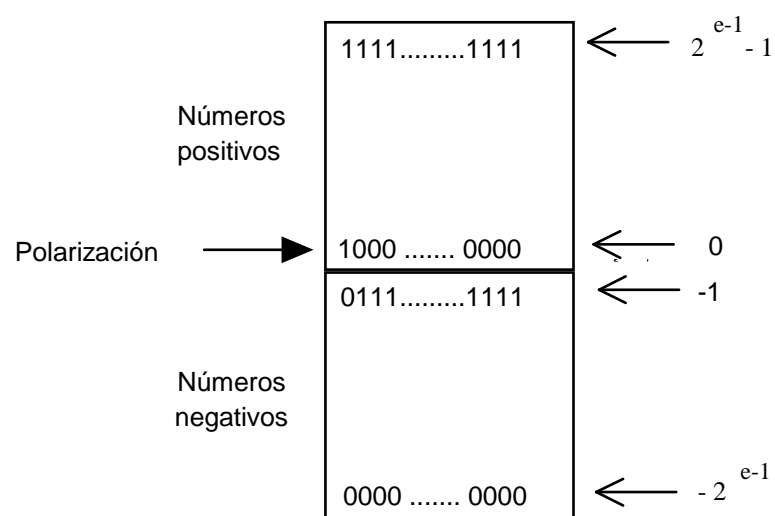


Figura 8.4 Exponente polarizado en 2^{e-1}

Para representar un número real en este formato es necesario realizar los siguientes pasos:

- Representar el número en binario.
- Expresar el número binario como se indica en (8-1)
- Obtener la mantisa M con m bits. Algunas veces será necesario añadir ceros a la mantisa hasta completar los m bits y otras veces se necesitará truncar o redondear la mantisa. Para redondear la mantisa M a m bits, se verifica el bit que ocupa la posición $-(m+1)$. Si éste es uno, se suma 2^{-m} a M ; si no es uno, la mantisa M es la correcta.
- Sumar la polarización (2^{e-1}) al exponente para obtener E .
- Determinar el bit del signo del número.
- Ensamblar el signo, exponente y mantisa en el formato.

Para representar el número -2441.5 en el formato anterior usando 1 bit para el signo, siete para el exponente ($e=7$), y 16 para la mantisa ($m=16$), es necesario hacer lo siguiente:

- Representar el número en binario:

$$-2441.5 = -100110001001 \dots .1_2$$

- Expresar el número binario como se indica en (8-1):

$$\begin{aligned} -100110001001 \dots .1_2 &= -0.1001100010011 \dots_2 \times 10_2^{12} \\ &= -0.1001100010011 \dots_2 \times 10_2^{1100} \end{aligned}$$

- Obtener la mantisa M con m bits:

$$M = 1001100010011000 \dots_2$$

- (d) Sumar la polarización (2^{e-1}) al exponente para obtener E:

$$E = 1000000_2 + 1100_2 = 1001100_2$$

- (e) Determinar el bit del signo del número:
Como el número es negativo el bit del signo es uno.
- (f) Ensamblar el signo, exponente y mantisa en el formato.

1	1001100	1001100010011000
signo	exponente	mantisa

Para representar el número real 0.2 en este formato se realizan los mismos pasos:

- (a) Representar el número en binario:

$$0.2 = 0.001100110011001100110011 \dots_2$$

- (b) Expresar el número binario como se indica en (8-1):

$$\begin{aligned} 0.001100110011 \dots_2 &= 0.1100110011 \dots_2 \times 10_2^{-2} \\ &= 0.1100110011 \dots_2 \times 10_2^{-10} \end{aligned}$$

- (c) Obtener la mantisa M con m bits:

$$M = 1100110011001100_2$$

- (d) Sumar la polarización (2^{e-1}) al exponente para obtener E:

$$E = 1000000_2 + (-10_2) = 0111110_2$$

- (e) Determinar el bit del signo del número:
Como el número es positivo el bit del signo es cero.
- (f) Ensamblar el signo, exponente y mantisa en el formato.

0	0111110	1100110011001100
signo	exponente	mantisa

El formato descrito anteriormente tiene limitaciones tanto en la precisión con la que se puede representar un número real, como en el rango de números que se pueden representar.

La precisión que tiene este formato es de 2^{-m+1} ya que el bit que ocupa la posición $-m$ puede tener error (puede haberse redondeado). Esto indica que se tienen $m-1$ bits significativos. Generalmente lo que interesa saber es el número de guarismos significativos en decimal, no en binario, si se tienen 16 bits para la mantisa se tendrá una precisión de 2^{-15} lo cual representa 3.05×10^{-5} . Esto implica que la quinta cifra en decimal podría tener error y proporciona solamente cuatro cifras significativas en decimal. Si se tuvieran 24 bits para la mantisa se tendrían 6 cifras significativas en decimal.

Nótese que la precisión depende exclusivamente del número de bits de que se disponga para representar la mantisa.

El mayor número positivo que puede representarse en este formato sería el siguiente:

$$\begin{aligned} \text{mantisa mayor} &= 0.1111 \dots 111_2 (m \text{ unos}) = 1 - 2^{-m} \\ \text{exp onente mayor} &= 2^{e-1} - 1 \\ \text{n úmero m a y o r} &= (1 - 2^{-m}) 2^{(2^{(e-1)}-1)} = 2^{(2^{(e-1)}-1)} - 2^{(2^{(e-1)}-1-m)}. \end{aligned}$$

El menor número positivo que se puede representar en este formato sería:

$$\begin{aligned} \text{mantisa m enor} &= 0.1000 \dots 000_2 (m - 1 \text{ ceros}) = 2^{-1} \\ \text{exp onente menor} &= -2^{e-1} \\ \text{n úmero m enor} &= 2^{-1} \times 2^{-2^{(e-1)}} = 2^{-(2^{(e-1)}+1)} \end{aligned}$$

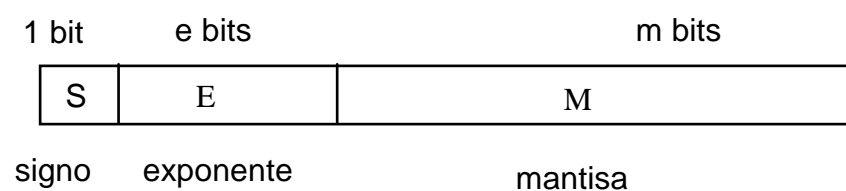
Si se usan 16 bits para la mantisa ($m=16$) y 7 bits para el exponente ($e=7$), el mayor número positivo y menor número positivo serían:

$$\begin{aligned} \text{n úmero mayor} &= (1 - 2^{-16}) \times 2^{2^{(7-1)}-1} = (1 - 2^{-16}) \times 2^{64-1} = 9.2234 \times 10^{18} \\ \text{n úmero m enor} &= 2^{-(2^{(7-1)}+1)} = 2^{-(64+1)} = 2.711 \times 10^{-20} \end{aligned}$$

Para números negativos se tiene el mismo rango que para números positivos con este formato.

En este formato, el cero es un caso especial ya que la mantisa del número cero no puede ser normalizada. Esta cantidad se representa con ceros en todos los bits.

Una variante del formato anterior que se usa con mucha frecuencia es la siguiente:



El bit de signo es uno cuando el número es negativo y cero en caso contrario.

Los m bits en M representa la parte fraccionaria de la mantisa normalizada en la base β . Los valores de β más comunes son 4, 8 y 16. Estos valores de β son potencias de 2. ($\beta^{-1} \leq M < 1$), la parte entera de la mantisa y el punto (0.) no ocupan un lugar en el formato ya que éstos son constantes y se puede suponer que existen.

Los e bits de E representan el exponente en binario polarizado en 2^{e-1} . El exponente es el número que indica la potencia a la que tiene que elevarse la base β para obtener el factor por el cual se tiene que multiplicar la mantisa para obtener la magnitud del número representado.

Esta variante permite representar un rango mayor de números reales que el formato anterior. Nótese que si se escoge la base β como 2, este formato es exactamente igual al anterior.

El mayor número positivo que se puede representar en este formato es:

$$(1 - 2^{-m}) \times \beta^{2^{(e-1)}-1}$$

En menor número positivo que puede ser representado en este formato es:

$$\beta^{-1} \times \beta^{-2^{e-1}} = \beta^{-(2^{(e-1)}+1)}$$

Los pasos que se siguen para representar un número real en este formato son exactamente los mismos que en el caso anterior, con una pequeña variante. Dado que la mantisa está representada en binario y debe normalizarse en base β ($\beta^{-1} \leq M < 1$), se debe recorrer el punto binario dos lugares a la vez si β es cuatro, tres lugares a la vez si β es ocho y cuatro lugares a la vez si β es dieciséis.

El exponente representa el número de veces que se realizaron corrimientos del punto binario y no el número de lugares que se corrió el punto binario. Por ejemplo, si β es ocho y se hacen dos corrimientos del punto binario (tres lugares a la vez), el punto binario se habrá recorrido seis lugares.

Como ejemplo de este formato, considérese la normalización del numero 117 en las diferentes bases:

$$117 = 1110101_2$$

$$1110101_2 = 0.1110101_2 \times 2^7 \text{ (normalizado en binario)}$$

$$1110101_2 = 0.01110101_2 \times 4^4 \text{ (normalizado en base 4)}$$

$$1110101_2 = 0.001110101_2 \times 8^3 \text{ (normalizado en base 8)}$$

$$1110101_2 = 0.01110101_2 \times 16^2 \text{ (normalizado en base 16)}$$

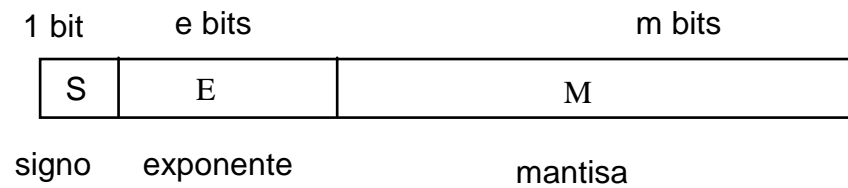
Si se analiza el caso cuando β es igual a 2, se puede deducir que es equivalente al primer formato que se definió. En ese formato, todas las mantisas normalizadas inician con 0.1_2 . Existen dos variantes a este formato por medio de las cuales se puede obtener un bit más de precisión, sin agregar más bits al formato.

La primera es que en lugar de suponer que la mantisa inicia con 0. se supone que inicia con 0.1_2 . La otra variante consiste en cambiar la forma de normalizar, dejando la mantisa en el rango: $1_2 \leq M_2 < 10_2$, donde todas las mantisas iniciarían con 1.2 .

Como ahora si es válida una mantisa de puros ceros en el formato, la representación que se tenía para el cero no puede seguirse usando.

Para representar el cero se restringe el uso del exponente menor y se usa sólo para la representación del cero. Esta cantidad se representa con cero en el bit de signo, en el exponente se coloca el exponente menor y la mantisa se rellena de ceros.

A continuación se da la especificación y un ejemplo de estos formatos.



El bit de signo es uno cuando el número es negativo y cero en caso contrario.

Los m bits de M representan la parte menos significativa de la mantisa fraccionaria. La parte entera de la mantisa, el punto y el primer bit de la fracción no ocupan un lugar en el formato, ya que éstos son constantes (0.1_2).

Los e bits de E representan el exponente en binario polarizado, en 2^{e-1} . El exponente más negativo se usa para representar el cero.

Para representar el número 0.2 en el formato anterior usando 7 bits para el signo (e=7) y 16 bits para la mantisa (m=16) es necesario hacer lo siguiente:

- (a) Representar el número en binario:

$$0.2 = 0.00110011001100110011 \dots_2$$

- (b) Expresar el número binario como se indica en (8-1):

$$\begin{aligned} 0.001100110011 \dots_2 &= 0.1100110011 \dots_2 \times 10_2^{-2} \\ &= 0.1100110011 \dots_2 \times 10_2^{-10} \end{aligned}$$

- (c) Obtener la mantisa M con m bits:

$$M = 1001100110011010_2$$

Note que al obtener M se hizo un redondeo.

- (d) Sumar la polarización (2^{e-1}) al exponente para obtener E:

$$E = 1000000_2 + (-10_2) = 0111110_2$$

- (e) Determinar el bit del signo del número:
Como el número es positivo el bit del signo es cero.

- (f) Ensamblar el signo, exponente y mantisa en el formato.

0	0111110	1001100110011010
signo	exponente	mantisa

El bit de signo es uno cuando el número es negativo y cero en caso contrario.

Los m bits en M representa la parte fraccionaria de la mantisa normalizada a $1_2 \leq M_2 < 10_2$.

Los e bits de E representa el exponente en binario polarizado en 2^{e-1} . El exponente más negativo se usa para representar el cero.

Para representar el número 0.2 en el formato anterior usando 7 bits para el exponente (e=7) y 16 bits para la mantisa (m=16), es necesario realizar lo siguiente:

- (a) Representar el número en binario:

$$0.2 = 0.00110011001100110011 \dots_2$$

- (b) Expresar el número binario en notación científica, con mantisa dentro del rango:
 $1_2 \leq M_2 < 10_2$.

$$\begin{aligned} 0.001100110011 \dots_2 &= 1.100110011 \dots_2 \times 10_2^{-3} \\ &= 1.100110011 \dots_2 \times 10_2^{-11} \end{aligned}$$

- (c) Obtener la mantisa M con m bits:

$$M = 1001100110011010_2$$

Note que al obtener M se hizo un redondeo.

- (d) Sumar la polarización (2^{e-1}) al exponente para obtener E:

$$E = 1000000_2 + (-11_2) = 0111101_2$$

- (e) Determinar el bit del signo del número:
Como el número es positivo el bit del signo es cero.
- (f) Ensamblar el signo, exponente y mantisa en el formato.

0	0111101	1001100110011010
---	---------	------------------

signo exponente mantisa

8.3.2 Formato con exponente y mantisa utilizando complementos a 2 para su representación.

Las formas más usadas para representar números reales en este formato toman como base la siguiente representación:

$$A \equiv 1.a_{-1}a_{-2} \cdots a_{-m_2} \times 10_2^{E_2} \equiv M_2 \times 10_2^{E_2} \quad (8-2)$$

si el número es positivo y

$$A \equiv 10.a_{-1}^*a_{-2}^* \cdots a_{-m_2}^* \times 10_2^{E_2} \equiv M_2^* \times 10_2^{E_2} \quad (8-3)$$

si el número es negativo.

La representación anterior consta de los siguientes elementos:

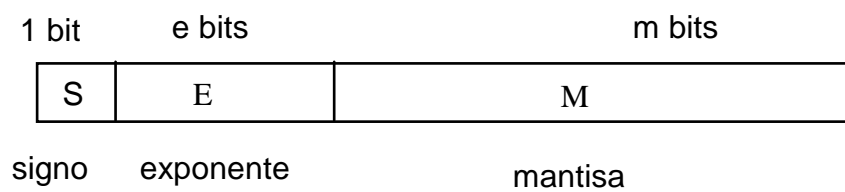
- Una mantisa binaria representada usando complementos a 2. Si el número es positivo, la mantisa ($M_2 = 1.a_{-1}a_{-2} \cdots a_{-m_2}$) esta normalizada a $1_2 \leq M_2 < 10_2$. Si el número es negativo, la mantisa ($M_2^* = 10.a_{-1}^*a_{-2}^* \cdots a_{-m_2}^*$) se obtiene al complementar a dos la mantisa que representa el número positivo ($M_2^* = 100_2 - M_2$).
- Un exponente entero con signo representado en binario (E_2), que indica la potencia a la que tiene que elevarse la base (10_2).

A continuación se muestran dos números expresados en la forma anterior con 8 bits significativos:

$$125_{10} = 1111101_2 = +1.111101_2 \times 10_2^{+6} = 1.111101_2 \times 10_2^{+110_2}$$

$$-0.1875_{10} = -0.0011_2 = -1.100000_2 \times 10_2^{-3} = 10.100000_2 \times 10_2^{-11_2}$$

El formato que se utiliza en las computadoras digitales para representar números en la notación descrita anteriormente es el siguiente:



El bit de signo es uno cuando el número es negativo y cero en caso contrario.

Los m bits en M representa la parte fraccionaria de la mantisa normalizada en binario descrita anteriormente, la parte entera de la mantisa y el punto (1. ó 10.) no ocupan un lugar en el formato ya que éstos son constantes y se puede suponer que se tienen. Los e bits de E representan el exponente en binario.

Para representar un número real en este formato es necesario realizar los siguientes pasos:

- (a) Representar el número en binario.
- (b) Expresar el número binario como se indica en (8-2) o (8-3) según sea el caso.
- (c) Obtener la mantisa M con m bits. Algunas veces será necesario añadir ceros a la mantisa hasta completar los m bits y otras veces se necesitará truncar o redondear la mantisa.
- (d) Expresar el exponente en e bits y, si éste es negativo complementarlo a dos para obtener E.
- (e) Determinar el bit del signo del número.
- (f) Ensamblar el signo, exponente y mantisa en el formato.

Para representar el número -2441.5 en el formato anterior usando 7 bits para el signo (e=7) y 16 bits para la mantisa (m=16) es necesario hacer lo siguiente:

- (a) Representar el número en binario:

$$-2441.5 = -100110001001.1_2$$

- (b) Expresar el número binario como se indica en (8-3):

$$\begin{aligned} -100110001001.1_2 &= -1.0011000100110000_2 \times 10_2^{11} \\ &= 10.1100111011010000_2 \times 10_2^{1011} \end{aligned}$$

- (c) Obtener la mantisa M :

$$M = 1100111011010000_2$$

- (d) Obtener E:

$$E = 0001011_2$$

- (e) Determinar el bit del signo del número:

Como el número es negativo el bit del signo es uno.

- (f) Ensamblar el signo, exponente y mantisa en el formato.

1	0001011	1100111011010000
---	---------	------------------

signo exponente mantisa

Para representar el número real 0.2 en este formato se realizan los mismos pasos:

- (a) Representar el número en binario:

$$0.2 = 0.001100110011001100110011 \dots_2$$

- (b) Expresar el número binario como se indica en (8-2):

$$\begin{aligned} 0.001100110011 \dots_2 &= 1.1001100110011001 \times 10_2^{-3} \\ &= 1.1001100110011001 \times 10_2^{-11} \end{aligned}$$

- (c) Obtener la mantisa M:

$$M = 1001100110011001_2$$

- (d) Obtener E:

$$E = -0000011_2 = 1111101_2$$

- (e) Determinar el bit del signo del número:

Como el número es positivo el bit del signo es cero.

- (f) Ensamblar el signo, exponente y mantisa en el formato.

0	1111101	1001100110011001
---	---------	------------------

signo exponente mantisa

El formato descrito anteriormente tiene limitaciones tanto en la precisión, con la que se puede representar un número real, como en rango de los números que permite representar.

La precisión que tiene este formato es de 2^{-m+1} ya que el bit que ocupa la posición -m puede tener error (puede haberse redondeado).

Esto indica que se tienen m-1 bits significativos. Generalmente lo que interesa saber es el número de guarismos significativos en decimal, no en binario, si se tienen 16 bits para la mantisa se tendrá una precisión de 2^{-15} lo cual representa 3.05×10^{-5} ; esto indica que la quinta cifra en decimal tendría

error y proporciona solamente cuatro cifras significativas en decimal. Si se tuvieran 24 bits para la mantisa se tendrían 6 cifras significativas en decimal.

Nótese que la precisión depende exclusivamente del número de bits de que se disponga para la mantisa.

El mayor número positivo que puede representarse en este formato sería el siguiente:

$$\begin{aligned} \text{mantisa mayor} &= 0.1111 \dots 1111_2 (m \text{ unos}) = 1 - 2^{-m} \\ \text{exp onente mayor} &= 2^{e-1} - 1 \\ \text{número mayor} &= (1 - 2^{-m}) 2^{(2^{(e-1)}+1)} = 2^{(2^{(e-1)}+1)} - 2^{(2^{(e-1)}+1-m)}. \end{aligned}$$

El menor número positivo que se puede representar en este formato sería:

$$\begin{aligned} \text{mantisa menor} &= 0.1000 \dots 000_2 (m - 1 \text{ ceros}) = 2^{-1} \\ \text{exp onente menor} &= -2^{e-1} \\ \text{número menor} &= 2^{-1} \times 2^{-2^{(e-1)}} = 2^{-(2^{(e-1)}+1)} \end{aligned}$$

Si se usa 16 bits para la mantisa (m=16) y 7 bits para el exponente (e=7), el número positivo mayor y menor serían:

$$\begin{aligned} \text{número mayor} &= (1 - 2^{-16}) \times 2^{2^{(7-1)}-1} = (1 - 2^{-16}) \times 2^{64-1} = 9.2234 \times 10^{18} \\ \text{número menor} &= 2^{-(2^{(7-1)}+1)} = 2^{-(64+1)} = 2.71 \times 10^{-20} \end{aligned}$$

Para números negativos se tiene el mismo rango que para números positivos con este formato.

En este formato el cero es un caso especial ya que la mantisa del cero no se puede normalizar. Esta cantidad se representa con ceros en todos los bits.

8.4. DECIMAL CODIFICADO EN BINARIO (BCD).

Algunas aplicaciones computacionales, normalmente sistemas administrativos, requieren manejar números reales dentro de un rango muy limitado comparado con los rangos que requieren las aplicaciones científicas o ingenieriles. Además, se requiere que los números representados sean exactos, con un cierto número de cifras decimales. Por ejemplo, un sistema de contabilidad requiere representar cantidades del orden de 10^9 exactas, con dos decimales (pesos y centavos). En la representación de números reales descrita en la sección anterior no es posible representar la cantidad de diez centavos (0.10 pesos) en forma exacta y esto no es conveniente para este tipo de aplicaciones.

La mayoría de las aplicaciones administrativas usan la representación BCD que significa "Binary-Code-Decimal" (Decimal codificado en binario).

Esta forma de representar números en la computadora digitales usa cuatro bits para representar o codificar cada uno de los dígitos del número decimal deseado.

La codificación de cada uno de los dígitos en DCB se muestra a continuación:

dígito	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Para representar números en BCD, se fija el número de dígitos enteros y decimales que se desea usar y se asignan cuatro bits por cada dígito.

Por ejemplo, si se usan cuatro dígitos enteros y dos decimales para representar una cantidad en BCD, el número 137.15 quedaría como sigue:

0000 0001 0011 0111 0001 0101

†

El punto decimal se supone entre la segunda y la tercera posiciones decimales, de izquierda a derecha.

Si se desea representar números en BCD con signo, sería necesario añadir un bit al menos para representar el signo.

8.5. REPRESENTACION DE CARACTERES.

En la sección anterior se usaron cuatro bits para representar o codificar los diez dígitos del sistema decimal. Para representar caracteres en computadoras digitales se usan códigos, que permiten representar los diferentes caracteres en binario.

Se han desarrollado varios códigos para representar caracteres y cada uno de ellos usa diferente números de bits para la representación. Algunos de estos códigos ya no se usan y otros están siendo desplazados por los que ofrecen un mayor número de caracteres y una mejor estructura.

Los dos códigos más usados en las computadoras digitales son el ASCII (American Standard Code for Information Interchange) y el EBCDIC (Extended Binary Code Decimal Interchange Code). Ambos códigos usan ocho bits para representar caracteres, los cuales incluyen las letras mayúsculas y minúsculas, los dígitos del 0 al 9, signos de puntuación (punto, coma, dos puntos, etc.), caracteres especiales (signo de suma, signo de menos, etc.) y caracteres de control.

Los caracteres de control realmente no representan un símbolo gráfico como los demás, sino que cumplen funciones específicas.

En los apéndices **I** y **II** se muestran los códigos ASCII y EBCDIC.

8.6. RESUMEN

En este capítulo se describen los cuatro tipos básicos de datos que se utilizan en las computadoras digitales así como su representación interna usando el sistema binario.

Para representar números enteros con signo, se han utilizado tres formatos:

- Magnitud y signo
- Complementos a uno (a la base disminuida)
- Complementos a dos (a la base)

La representación en magnitud y signo fue la primera que se utilizó pero actualmente ha caído en desuso por los problemas que se tienen para realizar operaciones aritméticas con esta representación.

Las otras dos representaciones se usan actualmente ya que las operaciones aritméticas son hechas fácilmente con ambas, pero la tendencia es utilizar solamente la representación en complementos a dos debido a que presenta ciertas ventajas sobre la otra. En el siguiente capítulo se describirán las operaciones aritméticas con cada uno de los formatos.

Para representar números reales con signo se utilizan variantes de la notación científica. Se usa un bit para el signo del número, e bits para el exponente y m bits para la mantisa normalizada. Todas las representaciones de números reales tienen cierta precisión, la cual depende exclusivamente del número de bits que se usen para la mantisa y el formato en particular. El rango de números reales que pueden ser representados en las computadoras digitales depende de la base que se seleccione para el exponente, generalmente 2, 4, 8 ó 16, y del número de bits que se utilicen para representar este exponente

Con el formato anterior para números reales no es posible representar exactamente cantidades como 0.01, 0.1, 1/3, etc. ya que se tiene una precisión limitada. Para algunas aplicaciones administrativas que necesitan representar en forma exacta números reales dentro de un rango limitado, con un cierto número de cifras decimales, se utiliza la representación decimal codificada en binario (BCD). En este formato, cada dígito del sistema decimal se codifica con cuatro bits y el punto decimal se supone que está fijo, dependiendo del número de cifras decimales que se representen.

Los caracteres, tales como las letras mayúsculas y minúsculas, los signos de puntuación y algunos caracteres especiales, son representados utilizando códigos. Los códigos mas usados actualmente son el ASCII (American Standard Code for Information Interchange) y el EBCDIC (Extended Binary Code Decimal Interchange Code). Ambos códigos usan 8 bits para representar los diferentes caracteres y se muestran en los apéndices IV y V respectivamente.

8.7. PROBLEMAS.

- 1.- Una computadora usa la representación en complementos a 2 para manejar números enteros con signo y emplea 12 bits en total para la representación. Para esta máquina conteste lo siguiente:
 - a) ¿Cuál es el número mayor que se puede representar?

- b) ¿Cuál es el número menor que se puede representar?
- c) Represente los siguientes números enteros en el formato de la máquina.
 (i) 1345 (ii) -1718 (iii) 415 (iv) -123 (v) 5000
- 2.- Resuelva el problema 1 suponiendo que la máquina usara la representación en complementos a 1 con 12 bits.
- 3.- Resuelva el problema 1 suponiendo que la máquina usara la representación en magnitud y signo con 12 bits.
- 4.- Una computadora usa el siguiente formato para representar números reales:

1 bit	7 bits	24 bits
signo	exponente	mantisa

Mantisa - normalizada en binario, representada en magnitud y signo.
 Exponente - binario polarizado en 64.
 Signo - 0 para números positivos y 1 para números negativos.

- a) ¿Cuál es el número mayor que se puede representar?
- b) ¿Cuál es el número positivo menor que se puede representar?
- c) ¿Cuántas cifras significativas en decimal proporciona este formato?
- d) Represente los siguientes números en el formato:
 (i) 1317.75 (ii) -178.25 (iii) -0.09375
- e) Obtenga el número decimal que representa el siguiente número real:
 (i) 01010011110100000000000000000000₂
 (ii) 10011010010101000000000000000000₂
 (iii) FB701000₁₆
 (iv) 32CAFE00₁₆
- 5.- Resuelva el problema 4 suponiendo el siguiente formato para representar números reales:

1 bit	7 bits	24 bits
signo	exponente	mantisa

Mantisa - normalizada en hexadecimal, representada en magnitud y signo.
 Exponente - hexadecimal polarizado en 64.
 Signo - 0 para números positivos y 1 para números negativos.

CAPITULO 9

OPERACIONES ARITMETICAS EN DIFERENTES REPRESENTACIONES.

Dado que las computadoras digitales pueden usar diferentes formatos para representar cada uno de los tipos de números, existen diferentes algoritmos (secuencia de pasos bien definidos) para realizar las operaciones básicas de suma, resta, multiplicación y división, dependiendo del formato de representación utilizado. Con estas cuatro operaciones es posible definir otras funciones aritméticas más complejas usando métodos de análisis numérico.

En esta capítulo se presentaran diferentes algoritmos para realizar las operaciones de suma y resta con números enteros en sus diferentes representaciones.

9.1 OPERACIONES ARITMETICAS DE NUMEROS ENTEROS CON SIGNO.

Dado que el rango de números enteros en un computadora digital es finito, al realizar operaciones con éstos puede dar el caso que el resultado no pueda ser representado. Cuando esto ocurre, el resultado de la operación es incorrecto, ya que en la operación ocurrió un desborde (overflow).

9.1.1 Operaciones aritméticas de números enteros representados en magnitud y signo.

Se A y B dos cantidades enteras con signo que cumplan lo siguiente:

$$-(2^{n-1} - 1) \leq A \leq 2^{n-1} - 1$$

$$-(2^{n-1} - 1) \leq B \leq 2^{n-1} - 1$$

Sean a y b los valores absolutos de A y B respectivamente y A_{sm} y B_{sm} la representación de A y B en magnitud y signo en n bits. (formato especificado en la sección 8.2.1)

Sea C la cantidad que resulta al hacer una operación algebraica entre A y B y C_{sm} su representación en magnitud y signo.

Para realizar operaciones con números representados en este formato, hay que hacer una separación entre el signo y la magnitud.

Si X_{sm} es una cantidad representada en magnitud y signo en n bits, X_m representará la cantidad de los primeros n-1 bits (del 0 al n-2), y X_s representará el valor del bit n-1.

Sea R_m la cantidad que resulta al hacer una operación en binario entre las magnitudes A_m y B_m .

Operación de suma:

Al efectuar la operación $C = A + B$ se tiene realmente una operación de suma cuando A y B son positivos o A y B son negativos, si solamente uno de ellos es negativo, se tendrá una operación de resta como se presenta en los siguientes casos:

caso a) $A \geq 0$ y $B \geq 0$

$A_s = 0$, A_m = representación binaria de a .
 $B_s = 0$, B_m = representación binaria de b.
 $C_s = 0$, C_m = Los primeros n - 1 bits de R_m .

Si $a + b \leq 2^{n-1} - 1$, el resultado R_m ($R_m = A_m + B_m$) es el deseado y puede ser representado por C_{sm} . El resultado de la operación es positivo y la magnitud es dada por R_m ($C_s = 0$, $C_m = R_m$). La suma binaria se realiza como se define en la sección 7.5, (Suma de números en base β) usando la base 2.

Si $a + b > 2^{n-1} - 1$, el resultado R_m ($R_m = A_m + B_m$) requiere de **n** bits para su representación, por lo cual no puede ser representado por C_{sm} . El resultado de la operación es positivo y la magnitud es dada por $R_m - 2^{n-1}$ ($C_s = 0$, $C_m = R_m - 2^{n-1}$).

Note que al sumar $A_m + B_m$ ocurrirá un acarreo en la última posición, el cual no se podrá representar en C_{sm} . En este caso ocurre un desborde (overflow).

A continuación se muestran dos ejemplos de sumas donde se tienen ambos casos. Se usa $n = 6$.

Ejemplo 1. $(+18) + (+12) = (+30)$

$$\begin{array}{rcllcl} & 010010 & A_m & = & 10010 & \\ + & \underline{001100} & + B_m & = & \underline{01100} & \\ & & R_m & = & 11110 & C_{sm} = 011110 \end{array}$$

Ejemplo 2. $(+26) + (+13) = (+7)$ (ocurre desborde)

$$\begin{array}{rcllcl} & 011010 & A_m & = & \overset{1}{1}1010 & \\ + & \underline{001101} & + B_m & = & \underline{01101} & \\ & & R_m & = & 100111 & C_{sm} = 000111 \text{ (desborde)} \end{array}$$

caso b) $A \geq 0$ y $B < 0$

$A_s = 0$, A_m = representación binaria de a .
 $B_s = 1$, B_m = representación binaria de b.
 $C_s = 0$ o 1 , $C_m = R_m = A_m - B_m$.

La resta binaria se puede realizar de tres formas:

- En forma directa, como se define en la sección 7.6, (Resta de números en base β) usando la base 2.
Si $a \geq b$, $C_s = 0$, $C_m = R_m = A_m - B_m$,
y si $a < b$, $C_s = 1$, $C_m = R_m = B_m - A_m$.
- Usando complementos a dos, como se define en la sección 7.7.1, (Resta usando complementos a la base) usando la base 2.
Se efectúa la operación $R_m = A_m + B_m^*$, si se genera un acarreo en la última posición, dicho acarreo se desecha y $C_s = 0$, $C_m =$ los primeros $n - 1$ bits de R_m . Si en la última posición no se produce un acarreo $C_s = 1$, $C_m = (R_m)^*$.
- Usando complementos a uno, como se define en la sección 7.7.2, (Resta usando complementos a la base disminuida) usando la base 2.
Se efectúa la operación $A_m + \overline{B_m}$, si al final se produce un acarreo, dicho acarreo se desecha y $C_s = 0$, $C_m = A_m + \overline{B_m} + 1$. Si al final no se produce un acarreo $C_s = 1$, $C_m = \overline{A_m + B_m}$.

A continuación se muestran dos ejemplos de sumas donde se utilizan complementos a dos para realizarlas. Se usa $n = 6$.

Ejemplo 1. $(+18) + (-12) = (+6)$

$$\begin{array}{rclcl} 010010 & A_m & = & 10010 & \\ + \underline{101100} & + \underline{B_m^*} & = & \underline{10100} & \\ R_m & = & 100110 & C_{sm} = 000110 & \end{array}$$

Ejemplo 2. $(+16) + (-18) = (-2)$

$$\begin{array}{rclcl} 010000 & A_m & = & 10000 & \\ + \underline{110010} & + \underline{B_m^*} & = & \underline{01110} & \\ R_m & = & 11110 & C_{sm} = 100010 & \end{array}$$

caso c) $A < 0$ y $B \geq 0$

$A_s = 1$, $A_m =$ representación binaria de a .
 $B_s = 0$, $B_m =$ representación binaria de b .
 $C_s = 0$ o 1 , $C_m = R_m = B_m - A_m$.

La resta binaria se puede realizar de tres formas:

- En forma directa, como se define en la sección 7.6, (Resta de números en base β) usando la base 2.
Si $b \geq a$, $C_s = 0$, $C_m =$ resta binaria de $B_m - A_m$,
y si $b < a$, $C_s = 1$, $C_m =$ resta binaria de $A_m - B_m$.
- Usando complementos a dos, como se define en la sección 7.7.1, (Resta usando complementos a la base) usando la base 2.
Se efectúa la operación $B_m + A_m^*$, si al final se produce un acarreo, dicho acarreo se desecha y $C_s = 0$, $C_m = B_m + A_m^*$. Si al final no se produce un acarreo $C_s = 1$, $C_m = (B_m + A_m^*)^*$.
- Usando complementos a uno, como se define en la sección 7.7.2, (Resta usando complementos a la base disminuida) usando la base 2.

efectúa

A continuación se muestran dos ejemplos de sumas donde se utiliza complementos a dos para realizarlas. Se usa $n = 6$.

Ejemplo 1. $(-18) + (+20) = (+2)$

$$\begin{array}{r}
 110010 \\
 + 010100 \\
 \hline
 \end{array}
 \quad
 \begin{array}{r}
 11 \\
 A_m^* = 01110 \\
 + B_m = 10100 \\
 \hline
 R_m = 100010 \quad C_{sm} = 000010
 \end{array}$$

Ejemplo 2. $(-29) + (+23) = (-7)$

$$\begin{array}{r}
 111101 \\
 + 010111 \\
 \hline
 \end{array}
 \quad
 \begin{array}{r}
 111 \\
 A_m^* = 00011 \\
 + B_m = 10111 \\
 \hline
 R_m = 11010 \quad C_{sm} = 100110
 \end{array}$$

caso d) $A < 0$ y $B < 0$

$A_s = 1$, $A_m =$ representación binaria de a .
 $B_s = 1$, $B_m =$ representación binaria de b .
 $C_s = 1$, $C_m =$ Los primeros $n - 1$ bits de R_m .

Si $a + b \leq 2^{n-1} - 1$, el resultado C es el deseado y puede ser representado por C_{sm} .
 $C_s = 1$, $C_m =$ suma binaria de $A_m + B_m$. La suma binaria se realiza como se define en la sección 7.5, (Suma de números en base β) usando la base 2.

Si $a + b > 2^{n-1} - 1$, el resultado C no puede ser representado por C_{sm} .
 $C_s = 1$, $C_m =$ suma binaria de $A_m + B_m - 2^{n-1}$.

Note que al sumar $A_m + B_m$ ocurrirá un acarreo en la última posición, el cual no se podrá representar en C_{sm} . En este caso ocurre un desborde.

A continuación se muestran dos ejemplos de sumas donde se muestran estos casos. Se usa $n = 6$.

Ejemplo 1. $(-18) + (-12) = (-30)$

$$\begin{array}{rclclcl} & 110010 & & A_m & = & 10010 \\ + & \underline{101100} & & + \underline{B_m} & = & \underline{01100} \\ & & & R_m & = & 11110 & C_{sm} = 111110 \end{array}$$

Ejemplo 2. $(-26) + (-13) = (-7)$ (ocurre desborde)

$$\begin{array}{rclclcl} & 111010 & & A_m & = & \overset{1}{11010} \\ + & \underline{101101} & & + \underline{B_m} & = & \underline{01101} \\ & & & R_m & = & 100111 & C_{sm} = 100111 \text{ (desborde)} \end{array}$$

Operación de resta:

El análisis de la operación resta ($A - B$) tiene los mismos casos que la operación suma, ya que la resta se resuelve por medio de la suma de $A + (-B)$.

9.1.2 Operaciones aritméticas de números enteros en representación en complementos a 2.

Se A y B dos cantidades enteras con signo que cumplan lo siguiente:

$$\begin{aligned} -2^{n-1} &\leq A \leq 2^{n-1} - 1 \\ -2^{n-1} &\leq B \leq 2^{n-1} - 1 \end{aligned}$$

Sean a y b los valores absolutos de A y B respectivamente y A_2 y B_2 la representación usando complementos a 2 de A y B usando n bits. (formato especificado en la sección 8.2.2)

Sea C la cantidad que resulta al hacer una operación algebraica entre A y B y C_2 su representación usando complementos a 2 usando n bits.

Sea R_2 la cantidad que resulta al hacer una operación en binario entre A_2 y B_2 .

Operación de suma:

Para efectuar la operación $C = A + B$, se realizará la operación $R_2 = A_2 + B_2$. Dependiendo de los valores de A y B se pueden tener los siguientes casos:

Caso a) $A \geq 0$ y $B \geq 0$

A_2 = representación binaria de a.

B_2 = representación binaria de b.

Si $a + b \leq 2^{n-1} - 1$, el resultado R_2 es el deseado, éste es igual a C_2 .

Si $a + b > 2^{n-1} - 1$, el resultado R_2 no es el correcto ya que se produce un acarreo sobre el bit más significativo, lo cual introduce un uno en el bit más significativo (bit de información del signo) de R_2 , haciendo que R_2 se interprete como un número negativo. Ya que la cantidad C no puede ser representada por C_2 se dice que ocurre un desborde.

Note que para este caso no se puede generar un acarreo en la posición **n** y sólo se genera un acarreo en la posición **n-1** cuando ocurre un desborde.

A continuación se muestran dos ejemplos de sumas donde se muestran estos casos. Se usa $n = 6$.

Ejemplo 1. $(+18) + (+12) = (+30)$

$$\begin{array}{rcl} A_2 & = & 010010 \\ + B_2 & = & \underline{001100} \\ R_2 & = & 011110 \end{array} \quad C_2 = 011110$$

Ejemplo 2. $(+26) + (+13) = (-25)$ (ocurre desborde)

$$\begin{array}{rcl} & & 11 \\ A_2 & = & 011010 \\ + B_2 & = & \underline{001101} \\ R_2 & = & 100111 \end{array} \quad C_2 = 100111 \quad (\text{desborde})$$

Caso b) $A \geq 0$ y $B < 0$

A_2 = representación binaria de a.

B_2 = representación binaria de $(2^n - b)$.

R_2 = representación binaria de $a + (2^n - b) = 2^n + a - b$

Si $a \geq b$ en los primeros **n** bits (del bit 0 al n-1) del resultado R_2 se tiene la diferencia entre a y b, y son equivalentes a los **n** bits de C_2 . 2^n representa un acarreo en la posición **n**, el cual es desechado.

Al analizar la operación que se realiza en los primeros **n-1** bits (del bit 0 al n-2), se puede demostrar que siempre se genera un acarreo en la posición **n-1**.

Demostración: en los primeros **n-1** bits de A_2 tiene la representación binaria de a, en los primeros **n-1** bits de B_2 tiene la representación binaria de $(2^n - b) - 2^{n-1} = 2^{n-1} - b$, que al sumarlos ($a + 2^{n-1} - b = 2^{n-1} + (a - b)$) siempre genera un acarreo en la posición **n-1** ya que a-b es mayor que cero.

Si $a < b$ en R_2 se tiene el complemento a dos de la diferencia entre a y b ($a + 2^n - b = 2^n - (b - a)$). el resultado R_2 es correcto e igual a C_2 .

Al analizar la operación que se realiza en los primeros **n-1** bits (del bit 0 al n-2), se puede demostrar que nunca se genera un acarreo en la posición **n-1**, ni en la posición **n**.

Demostración: en los primeros **n-1** bits A_2 tiene la representación binaria de a, en los primeros **n-1** bits de B_2 tiene la representación binaria de $(2^n - b) - 2^{n-1} = 2^{n-1} - b$, que al sumarlos ($a + 2^{n-1} - b = 2^{n-1} + (a - b)$) nunca genera un acarreo en la posición **n-1** ya que a-b es menor que cero. Si no hay acarreo en la posición **n-1** no se puede generar acarreo en la posición **n**.

A continuación se muestran dos ejemplos de sumas donde se muestran estos casos. Se usa $n = 6$.

Ejemplo 1. $(+18) + (-12) = (+6)$

$$\begin{array}{rcl} & & 1 \\ A_2 & = & 010010 \\ + B_2 & = & \underline{110100} \\ R_2 & = & 1000110 \end{array} \quad C_2 = 000110$$

Ejemplo 2. $(+16) + (-18) = (-2)$

$$\begin{array}{rcl} A_2 & = & 010000 \\ + B_2 & = & \underline{101110} \\ R_2 & = & 111110 \end{array} \quad C_2 = 111110$$

Caso c) $A < 0$ y $B \geq 0$

A_2 = representación binaria de $(2^n - a)$.

B_2 = representación binaria de b.

R_2 = representación binaria de $(2^n - a) + b = 2^n + b - a$

Este caso se analiza igual que el caso anterior.

Caso d) $A < 0$ y $B < 0$

A_2 = representación binaria de $(2^n - a)$.

B_2 = representación binaria de $(2^n - b)$.

R_2 = representación binaria de $(2^n - a) + (2^n - b) = 2^n + 2^n - (a + b)$

Si $a + b \leq 2^{n-1}$, en los primeros n bits (del bit 0 al $n-1$) del resultado R_2 se tiene $2^n - (a + b)$ que es el complementos a dos de la magnitud del resultado, estos n bits son equivalentes a los n bits de C_2 . El segundo 2^n en R_2 representa un acarreo en la posición n , el cual es desechado.

Si $a + b > 2^{n-1}$, el resultado R_2 no es el correcto ya que no se produce un acarreo sobre el bit más significativo, lo cual introduce un cero en el bit más significativo (bit de información del signo) de R_2 , haciendo que R_2 se interprete como un número positivo. Ya que la cantidad C no puede ser representada por C_2 se dice que ocurre un desborde.

Note que para este caso siempre se genera un acarreo en la posición n y sólo se genera un acarreo en la posición $n-1$ cuando no ocurre un desborde, como se demuestra a continuación:

En los primeros $n-1$ bits A_2 tiene la representación binaria de $(2^n - a) - 2^{n-1} = 2^{n-1} - a$, en los primeros $n-1$ bits de B_2 tiene la representación binaria de $(2^n - b) - 2^{n-1} = 2^{n-1} - b$, que al sumarlos $((2^{n-1} - a) + (2^{n-1} - b) = 2^n - (a + b))$ sólo genera un acarreo en la posición $n-1$ cuando $2^n - (a + b) \geq 2^{n-1}$, que es cuando $(a + b) \leq 2^{n-1}$.

A continuación se muestran dos ejemplos de sumas donde se muestran estos casos. Se usa $n = 6$.

Ejemplo 1. $(-18) + (-12) = (-30)$

$$\begin{array}{rcl}
 & & 111 \\
 A_2 & = & 101110 \\
 + B_2 & = & \underline{110100} \\
 R_2 & = & 1100010 \qquad C_2 = 100010
 \end{array}$$

Ejemplo 2. $(-26) + (-13) = (+21)$ (ocurre desborde)

$$\begin{array}{rcl}
 & & 11 \\
 A_2 & = & 100110 \\
 + B_2 & = & \underline{110011} \\
 R_2 & = & 1011001 \qquad C_2 = 011001 \quad (\text{desborde})
 \end{array}$$

Analizando los resultados anteriores se puede deducir las siguientes reglas para sumar números enteros en representación en complementos a dos:

- Para sumar dos números enteros en representación en complementos a 2 se suman los n bits directamente.
- Si el acarreo sobre el bit más significativo y el acarreo fuera de los n bits son diferentes, ocurre un desborde y el resultado no es correcto.
- Si el acarreo sobre el bit más significativo y el acarreo fuera de los n bits son iguales, el resultado es correcto y queda en la representación en complementos a 2. Cualquier acarreo fuera de los n bits se desecha.

Otra forma equivalente de detectar un desborde es la siguiente:

- Si al sumar dos números positivos (negativos) el resultado da negativo (positivo), ocurre un desborde y el resultado obtenido no es correcto.

Operación de resta:

La operación de resta es igual a la de suma. Para hacer la resta de $A - B$ se hace la suma de $A + (-B)$. Si el número **B** es el más negativo esta operación no se puede realizar, ya que no se puede representar el número **-B**.

9.1.3 Operaciones aritméticas de números enteros en representación en complementos a 1.

Se A y B dos cantidades enteras con signo que cumplan lo siguiente:

$$\begin{aligned}
 -(2^{n-1} - 1) &\leq A \leq 2^{n-1} - 1 \\
 -(2^{n-1} - 1) &\leq B \leq 2^{n-1} - 1
 \end{aligned}$$

Sean a y b los valores absolutos de A y B respectivamente y A_1 y B_1 la representación usando complementos a 1 de A y B usando **n** bits. (formato especificado en la sección 8.2.3)

Sea C la cantidad que resulta al hacer una operación algebraica entre A y B y C_1 su representación usando complementos a 1 usando **n** bits.

Sea R_1 la cantidad que resulta al hacer una operación en binario entre A_1 y B_1 .

Operación de suma:

Para efectuar la operación $C = A + B$, se realizará la operación $R_1 = A_1 + B_1$. Dependiendo de los valores de A y B se pueden tener los siguientes casos:

Caso a) $A \geq 0$ y $B \geq 0$

A_1 = representación binaria de a .

B_1 = representación binaria de b .

Si $a + b \leq 2^{n-1} - 1$, el resultado R_1 es el deseado, éste es igual a C_1 .

Si $a + b > 2^{n-1} - 1$, el resultado R_1 no es el correcto ya que se produce un acarreo sobre el bit más significativo, lo cual introduce un uno en el bit más significativo (bit de información del signo) de R_1 , haciendo que R_1 se interprete como un número negativo. Ya que la cantidad C no puede ser representada por C_1 se dice que ocurre un desborde.

Note que para este caso no se puede generar un acarreo en la posición n y sólo se genera un acarreo en la posición $n-1$ cuando ocurre un desborde.

A continuación se muestran dos ejemplos de sumas donde se muestran estos casos. Se usa $n = 6$.

Ejemplo 1. $(+18) + (+12) = (+30)$

$$\begin{array}{rcl} A_1 & = & 010010 \\ + B_1 & = & \underline{001100} \\ R_1 & = & 011110 \end{array} \quad C_1 = 011110$$

Ejemplo 2. $(+26) + (+13) = (-24)$ (ocurre desborde)

$$\begin{array}{rcl} A_1 & = & \overset{11}{011010} \\ + B_1 & = & \underline{001101} \\ R_1 & = & 100111 \end{array} \quad C_1 = 100111 \text{ (desborde)}$$

Caso b) $A \geq 0$ y $B < 0$

A_1 = representación binaria de a .

B_1 = representación binaria de $(2^n - 1 - b)$.

R_1 = representación binaria de $a + (2^n - 1 - b) = 2^n - 1 + a - b$

Si $a > b$ en los primeros n bits (del bit 0 al $n-1$) del resultado R_1 se tiene la diferencia entre a y b menos 1, para obtener los n bits de C_1 se le suma 1 (se hace una corrección) a los primeros n bits de R_1 . El 2^n en R_1 representa un acarreo en la posición n , el cual es desechado.

Al analizar la operación que se realiza en los primeros $n-1$ bits (del bit 0 al $n-2$), se puede demostrar que siempre se genera un acarreo en la posición $n-1$.

Demostración: en los primeros $n-1$ bits de A_1 tiene la representación binaria de a , en los primeros $n-1$ bits de B_1 tiene la representación binaria de $(2^n - 1 - b) - 2^{n-1} = 2^{n-1} - 1 - b$, que al sumarlos y al hacer la corrección de sumar 1 se tiene la representación binaria de $(a + 2^{n-1} - 1 - b + 1 = 2^{n-1} + (a - b))$ siempre genera un acarreo en la posición $n-1$ ya que $a-b$ es mayor que cero.

Si $a \leq b$ en R_1 se tiene el complemento a uno de la diferencia entre a y b ($a + 2^n - 1 - b = 2^n - 1 - (b - a)$). el resultado R_1 es correcto e igual a C_1 .

Al analizar la operación que se realiza en los primeros $n-1$ bits (del bit 0 al $n-2$), se puede demostrar que nunca se genera un acarreo en la posición $n-1$, ni en la posición n .

Demostración: en los primeros $n-1$ bits A_1 tiene la representación binaria de a , en los primeros $n-1$ bits de B_1 tiene la representación binaria de $(2^n - 1 - b) - 2^{n-1} = 2^{n-1} - 1 - b$, que al sumarlos ($a + 2^{n-1} - 1 - b = 2^{n-1} - 1 + (a - b)$) nunca genera un acarreo en la posición $n-1$ ya que $a-b$ es menor o igual que cero. Si no hay acarreo en la posición $n-1$ no se puede generar acarreo en la posición n .

A continuación se muestran dos ejemplos de sumas donde se muestran estos casos. Se usa $n = 6$.

Ejemplo 1. $(+18) + (-12) = (+6)$

$$\begin{array}{rcl} & & 1 \quad 1 \\ A_1 & = & 010010 \\ + B_1 & = & \underline{110011} \\ R_1 & = & 1000101 \end{array} \quad C_1 = 000101 + 1 = 000110$$

Ejemplo 2. $(+16) + (-18) = (-2)$

$$\begin{array}{rcl} A_1 & = & 010000 \\ + B_1 & = & \underline{101101} \\ R_1 & = & 111101 \end{array} \quad C_1 = 111101$$

Caso c) $A < 0$ y $B \geq 0$

A_1 = representación binaria de $(2^n - 1 - a)$.

B_1 = representación binaria de b .

R_1 = representación binaria de $(2^n - 1 - a) + b = 2^n - 1 + b - a$

Este caso se analiza igual que el caso anterior.

Caso d) $A < 0$ y $B < 0$

A_1 = representación binaria de $(2^n - 1 - a)$.

B_1 = representación binaria de $(2^n - 1 - b)$.

R_1 = representación binaria de $(2^n - 1 - a) + (2^n - 1 - b) = 2^n - 1 + 2^n - 1 - (a + b)$

Si $a + b \leq 2^{n-1} - 1$, en los primeros n bits (del bit 0 al $n-1$) del resultado R_1 se tiene $2^n - 1 - (a + b) - 1$ que si hacemos una corrección de sumarle 1 se tiene el complemento a uno de la magnitud del resultado. Los n bits corregidos de R_1 son equivalentes a los n bits de C_1 . El segundo 2^n en R_1 representa un acarreo en la posición n , el cual es desechado.

Si $a + b > 2^{n-1} - 1$, el resultado R_1 no es el correcto ya que aún y cuando se haga la corrección de sumar 1, no se produce un acarreo sobre el bit más significativo, lo cual introduce un cero en el bit más significativo (bit de información del signo) de R_1 , haciendo que R_1 se interprete como un número positivo. Ya que la cantidad C no puede ser representada por C_1 se dice que ocurre un desborde.

Note que para este caso siempre se genera un acarreo en la posición n y sólo se genera un acarreo en la posición $n-1$ cuando no ocurre un desborde, como se demuestra a continuación:

En los primeros $n-1$ bits A_1 tiene la representación binaria de $(2^n - 1 - a) - 2^{n-1} = 2^{n-1} - 1 - a$, en los primeros $n-1$ bits de B_1 tiene la representación binaria de $(2^n - 1 - b) - 2^{n-1} = 2^{n-1} - 1 - b$, que al sumarlos y al hacer la corrección de sumar 1 tenemos la representación binaria de $((2^{n-1} - 1 - a) + (2^{n-1} - 1 - b) + 1 = 2^n - 1 - (a + b))$. Sólo genera un acarreo en la posición $n-1$ cuando $2^n - 1 - (a + b) \geq 2^{n-1}$, que es cuando $(a + b) \leq 2^{n-1} - 1$.

A continuación se muestran dos ejemplos de sumas donde se muestran estos casos. Se usa $n = 6$.

Ejemplo 1. $(-18) + (-12) = (-30)$

111111

$$\begin{array}{rcl}
 A_2 & = & 101101 \\
 + B_2 & = & \underline{110011} \\
 R_2 & = & 1100000 \qquad C_2 = 100000 + 1 = 100001
 \end{array}$$

Ejemplo 2. $(-26) + (-13) = (+24)$ (ocurre desborde)

$$\begin{array}{rcl}
 A_2 & = & 100101 \\
 + B_2 & = & \underline{110010} \\
 R_2 & = & 1010111 \qquad C_2 = 010111 + 1 = 011000 \quad (\text{desborde})
 \end{array}$$

Analizando los resultados anteriores se puede deducir las siguientes reglas para sumar números enteros en representación en complementos a uno:

- Para sumar dos números enteros en representación en complementos a 1 se suman los n bits directamente.
- Si se genera un acarreo fuera de los n bits, se desecha dicho acarreo y se efectúa una corrección sumándole uno al resultado obtenido.
- Si el acarreo sobre el bit más significativo y el acarreo fuera de los n bits que se generen al hacer la suma y la corrección en los casos necesarios, son diferentes, ocurre un desborde y el resultado no es correcto.
- Si el acarreo sobre el bit más significativo y el acarreo fuera de los n bits son iguales, el resultado es correcto y queda en la representación en complementos a 1.

Otra forma equivalente de detectar un desborde es la siguiente:

- Si al sumar dos números positivos (negativos) el resultado da negativo (positivo), ocurre un desborde y el resultado obtenido no es correcto.

Operación de resta:

La operación de resta es igual a la de suma. Para hacer la resta de $A - B$ se hace la suma de $A + (-B)$.

9.2 OPERACIONES DE NUMEROS EN DECIMAL CODIFICADO EN BINARIO (BCD).

Operación de suma:

Considérese la suma de dos números en BCD con un dígito:

Sean **a** y **b** dos números decimales de un dígito y sean **A** y **B** sus representaciones en BCD.

Sea **R** el resultado obtenido al sumar **A** y **B** como dos números binarios. Al realizar la suma pueden presentarse los siguientes casos:

caso i) $0000_2 \leq R \leq 1001_2$

En este caso la suma no excede a nueve y el resultado R es correcto.

caso ii) $1001_2 < R$

Ahora el resultado R no está correcto ya que una cantidad mayor a nueve no puede ser representada por un solo dígito BCD. Para que el resultado sea correcto es necesario restar diez (1010_2) a R o sumar seis (0110_2) que representa el complementos a 2 de diez y se genera un acarreo decimal.

A continuación se muestra varios ejemplos de suma de números en BCD con un solo dígito.

$\begin{array}{r} 2 \\ + 5 \\ \hline 7 \end{array}$	$\begin{array}{r} 0010 \\ + 0101 \\ \hline 0111 \end{array} \text{ correcto}$	$\begin{array}{r} 3 \\ + 6 \\ \hline 9 \end{array}$	$\begin{array}{r} 0011 \\ + 0110 \\ \hline 1001 \end{array} \text{ correcto}$
$\begin{array}{r} 7 \\ + 5 \\ (1) \ 7 \end{array}$	$\begin{array}{r} 0111 \\ + 0101 \\ \hline 1100 \\ + 0110 \text{ sumar 6} \\ (1) \ 0010 \text{ correcto} \end{array}$	$\begin{array}{r} 9 \\ + 8 \\ (1) \ 7 \end{array}$	$\begin{array}{r} 1001 \\ + 1000 \\ (1) \ 0001 \\ + 0110 \text{ sumar 6} \\ (1) \ 0111 \text{ correcto} \end{array}$

Dado que en los últimos dos ejemplos la suma se excede a nueve, se hizo una corrección y se generó un acarreo decimal.

Para sumar números de varios dígitos se sigue un procedimiento similar sumando también los acarreos decimales generados. A continuación se muestra un ejemplo.

$\begin{array}{r} 1 \\ 137 \\ + 381 \\ \hline 518 \end{array}$	$\begin{array}{r} 0001 \\ 0001 \ 0011 \ 0111 \\ + 0011 \ 1000 \ 0001 \\ \hline 0101 \ 1011 \ 1000 \\ + 0110 \\ (1) \ 0001 \end{array}$
Resultado :	0101 0001 1000

Operación de resta:

La resta de números en **BCD** se puede realizar por medio de una suma utilizando el concepto de complementos a la base o complementos a la base disminuida.

Sean **A** y **B** dos números BCD de **n** dígitos y **R** un número BCD de **n** dígitos que representa el resultado de la resta entre **A** y **B**. Para realizar la operación **A - B** por medio de una suma, utilizando el concepto de complementos a la base disminuida (complementos a nueve) se realizan los siguientes pasos :

- Se efectúa la operación $A + \bar{B}$ usando el procedimiento que se define en la sección anterior..
- Si se produce un acarreo fuera de las **n** dígitos, dicho acarreo se desecha y para obtener el resultado correcto se le suma uno al resultado de la operación.
- Si no se produce un acarreo fuera de las **n** dígitos, para obtener el resultado correcto se obtiene el complemento a la base disminuida de la magnitud del resultado y se indica que es negativo.

A continuación se muestra un ejemplo con número de 3 dígitos.

$$\begin{array}{r}
 137 \\
 - 381 \\
 \hline
 - 244
 \end{array}
 \qquad
 \begin{array}{r}
 1 \\
 137 \\
 + 618 \quad (\text{complemento a nueve de } 381) \\
 \hline
 755
 \end{array}$$

El resultado es negativo y es el complemento a nueve de 755 el cual es 244.

$$\begin{array}{r}
 0001 \\
 0001\ 0011\ 0111 \\
 + 0110\ 0001\ 1000 \\
 \hline
 0111\ 0101\ 1111 \\
 \qquad \qquad \qquad +0110 \\
 \qquad \qquad \qquad \hline
 \qquad \qquad \qquad 0101
 \end{array}$$

Resultado negativo y es el complemento a nueve de 0111 0101 0101

$$\begin{array}{r}
 1001\ 1001\ 1001 \\
 - 0111\ 0101\ 0101 \\
 \hline
 0010\ 0100\ 0100
 \end{array}$$

Resultado: - 0010 0100 0100.

Para realizar la operación **A - B** por medio de una suma, utilizando el concepto de complementos a la base se realizan una serie de pasos semejante.

9.3 RESUMEN.

En este capítulo se estudiaron las operaciones aritméticas de suma y resta de números enteros con signo en sus tres representaciones: Magnitud y signo, Complementos a uno y Complementos a 2. La representación más utilizada actualmente es esta última.

También se analizaron las operaciones con números representados en decimal codificado en binario (BCD).

9.4 PROBLEMAS.

- 1.- Efectúe las siguientes operaciones con números enteros con signo suponiendo que están en signo y magnitud usando 8 bits en total. Para cada caso obtenga el equivalente decimal del resultado.

$$\begin{array}{rcl} \text{(i)} & 01101001 & \text{(ii)} \quad 01101001 \quad \text{(iii)} \quad 01101001 \\ & + 00010101 & + 00101101 \quad + 11010111 \end{array}$$

$$\begin{array}{rcl} \text{(iv)} & 00011011 & \text{(v)} \quad 10110100 \quad \text{(vi)} \quad 11101010 \\ & + 10001011 & + 10001011 \quad + 11010110 \end{array}$$

- 2.- Resuelva el problema 1 suponiendo que los números están en la representación en complementos a 2 con 8 bits.
- 3.- Resuelva el problema 1 suponiendo que los números están en la representación en complementos a 1 con 8 bits.
- 4.- Realice las siguientes operaciones con números BCD y obtenga el resultado en decimal. Considere que los números BCD tienen 3 cifras en decimal.

$$\begin{array}{rcl} \text{(i)} & 0000 & 1001 & 0101 \\ & + 0100 & 0101 & 0111 \end{array} \qquad \begin{array}{rcl} \text{(ii)} & 1000 & 0001 & 0100 \\ & - 0100 & 1000 & 1001 \end{array}$$

$$\begin{array}{rcl} \text{(iii)} & 0100 & 1000 & 0010 \\ & + 0001 & 0101 & 0011 \end{array} \qquad \begin{array}{rcl} \text{(iv)} & 0110 & 0111 & 1000 \\ & - 0001 & 1000 & 0011 \end{array}$$

- 5.- Investigue que pasos se tienen que realizar para sumar números reales (números de punto flotante).

CAPITULO 10

FUNDAMENTOS DE ALGEBRA BOOLEANA.

El álgebra booleana es el álgebra de la lógica. Fue creada por el matemático inglés George Boole (1815-1864), quien dio a conocer su trabajo en su libro "An Investigation of the Law of Thought", en el año de 1854.

Aproximadamente en el año de 1940, Claude E. Shannon demuestra que las propiedades básicas de combinaciones en serie y paralelo de elementos biestables pueden ser representadas por esta álgebra.

El álgebra booleana es la mejor herramienta conocida para el análisis y diseño de sistemas lógicos, entre los cuales, las computadoras electrónicas digitales han llegado a ocupar un lugar preponderante. Esto debido a la gran ayuda que prestan al hombre y a la creciente aceptación que tienen en nuestra sociedad.

En este capítulo, se discutirán los conocimientos básicos del álgebra booleana.

10.1. VARIABLES BOOLEANAS.

Una variable booleana o lógica, puede tomar sólo dos valores diferentes que pueden representarse como "**0**" y "**1**"; "**falso**" y "**verdadero**"; "**ausencia**" y "**presencia**", etc. Para aquellos lectores que estén familiarizados con el álgebra de conjuntos, es conveniente visualizar el universo lógico como un universo que tiene un elemento; en dicho universo sólo pueden existir dos conjuntos: el conjunto vacío (\emptyset) y el conjunto universo (\cup).

De aquí en adelante, se utilizará el cero y el uno para representar los dos valores que puede tomar una variable booleana. El cero representa falso o conjunto vacío y el uno representa verdadero o el conjunto universo.

El nombre por medio del cual se referencia a una variable booleana es una letra mayúscula o minúscula, la cual puede tener un subíndice. A continuación se dan algunos ejemplos de nombres de variables booleanas:

A, a, Z, y, a₀, a₁.

10.2. OPERACIONES BASICAS.

Existen tres operaciones básicas en el álgebra booleana, dos de dichas operaciones actúan sobre dos operandos y la otra está definida sobre un solo operando. Estas operaciones son: **AND** (Y) , **OR** (O) y **NOT** (NO).

La operación **AND** actúa sobre dos operandos. Su definición se muestra en la tabla 10.1 y se representa como se indica a continuación:

$$A \bullet B = AB = A \wedge B = A \text{ AND } B \quad (\text{Léase "A" AND "B"})$$

A	B	A • B
0	0	0
0	1	0
1	0	0
1	1	1

Tabla 10.1 Operación **AND**

De la tabla se puede observar que para que el resultado de la operación **AND** sea uno, es necesario que ambos operandos tengan el valor de uno.

La operación **OR** también actúa sobre dos operandos, su definición se muestra en la tabla 10.2 y se representa como se indica a continuación:

$$A + B = A \vee B = A \text{ OR } B \quad (\text{Léase "A" OR "B"})$$

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

Tabla 10.2 Operación **OR**

La operación **OR** tomará el valor de uno, cuando cualquiera de sus dos operandos, o ambos, tengan el valor de uno.

La operación **NOT** está definida tal y como se muestra en la tabla 10.3 y actúa sobre un solo operando. Su representación es la siguiente:

$$A' = \overline{A} = \text{NOT } A \quad (\text{Léase NOT "A" o "A" complemento})$$

A	A'
0	1

1	0
---	---

Tabla 10.3 Operación **NOT**

La operación **NOT** tomará el valor de uno, cuando el operando tenga el valor de cero y viceversa.

Aquellos lectores que estén familiarizados con el álgebra de conjuntos, consideren de nuevo el universo con un solo elemento. Obsérvese que la operación **AND** equivale a la operación intersección, la operación **OR** equivale a la operación unión y la operación **NOT** equivale a la operación complemento. La tabla 10.4 muestra las tres operaciones. El cero es equivalente al conjunto vacío y el uno equivale al conjunto universo.

A	B	$A \wedge B$	$A \vee B$	A'
\emptyset	\emptyset	\emptyset	\emptyset	U
\emptyset	U	\emptyset	U	U
U	\emptyset	\emptyset	U	\emptyset
U	U	U	U	\emptyset

Tabla 10.4 Operación con conjuntos

En el libro usaremos la siguiente nomenclatura:

Operación	Nomenclatura
A AND B	A B
A OR B	A + B
NOT A	A'

10.3. JERARQUIA DE LAS OPERACIONES.

Al evaluar una expresión booleana, deben realizarse las operaciones de acuerdo a su jerarquía, realizando primero la de mayor jerarquía. Si existen paréntesis, se deben resolver primero los más internos y trabajar hacia afuera.

En ausencia de paréntesis, la jerarquía de las operaciones es, de mayor a menor, la siguiente:

1. Operación **NOT**
2. Operación **AND**
3. Operación **OR**

Si se tienen varias operaciones con la misma jerarquía, éstas pueden ser evaluadas de derecha a izquierda o de izquierda a derecha, el resultado será el mismo.

Como ejemplo, considérese la evaluación de las siguientes expresiones booleanas para A = 1, B = 0 y C = 0.

Expresión 1:

$$\begin{aligned}
 & A B + B C' + A B' \\
 & 1 0 + 0 0' + 1 0' \quad (\text{Sustitución de valores}) \\
 & 1 0 + 0 1 + 1 1 \quad (\text{Evaluación de los NOTs}) \\
 & 0 + 0 + 1 \quad (\text{Evaluación de los ANDs}) \\
 & 0 + 1 \\
 & 1 \quad (\text{Evaluación de los ORs})
 \end{aligned}$$

Expresión 2:

$$\begin{aligned}
 & A (B C' + C' (B' + A')) + A' B \\
 & 1 (0 0' + 0' (0' + 1')) + 1' 0 \quad (\text{Sustitución de valores}) \\
 & 1 (0 0' + 0' (1 + 0)) + 1' 0 \quad (\text{Evaluación paréntesis más interno}) \\
 & 1 (0 0' + 0' 1) + 1' 0 \quad (\text{Evaluación siguiente paréntesis}) \\
 & 1 (0 1 + 1 1) + 1' 0 \\
 & 1 (0 + 1) + 1' 0 \\
 & 1 1 + 1' 0 \quad (\text{Evaluación del NOT}) \\
 & 1 1 + 0 0 \quad (\text{Evaluación de los ANDs}) \\
 & 1 + 0 \quad (\text{Evaluación del OR}) \\
 & 1
 \end{aligned}$$

10.4. FUNCIONES BOOLEANAS.

Una función booleana de una o más variables representa una relación lógica entre dichas variables. La función booleana también es una variable booleana cuyo valor depende de los valores de las variables independientes y de la relación que exista entre ellas. A continuación se muestran ejemplos de funciones booleanas.

$$F(A, B) = A B' + A' B$$

$$Y(A, B, C) = A' (B' + C) + B' (A C + A')$$

10.4.1. Término

Un término consta de una o más variables unidas por la operación **AND**. Las variables pueden aparecer en su forma normal o complementadas. Algunos ejemplos de términos son:

$$A, A B, A' B C', X Y' Z' W$$

10.4.2. Minitérmino y maxitérmino.

Considere una función booleana de **n** variables $F(X_n, X_{n-1}, \dots, X_2, X_1)$. Dado que cada variable puede tomar dos valores, el conjunto de las **n** variables, puede tomar 2^n valores diferentes.

Un minitérmino es un término que contiene todas las variables de la función unidas por la operación **AND**. Por ejemplo, para $n = 3$, existen ocho minitérminos posibles que son:

$$\begin{array}{cccc} X_3' X_2' X_1' & X_3' X_2' X_1 & X_3' X_2 X_1' & X_3' X_2 X_1 \\ X_3 X_2' X_1' & X_3 X_2' X_1 & X_3 X_2 X_1' & X_3 X_2 X_1 \end{array}$$

Se le denomina minitérminos, porque solamente para una de las 2^n posibles combinaciones de los valores de las variables, toma el valor de 1. Por ejemplo, el minitérmino $X_3 X_2' X_1$ sólo toma el valor de 1 para $X_3 = 1$, $X_2 = 0$ y $X_1 = 1$; para cualquier otra combinación de los valores de las variables independientes, tomará el valor de 0.

Un maxitérmino, es realmente una operación **OR** de n términos, cada uno de los cuales, contiene solamente una de las variables de la función; todas las variables deberán estar presentes en el maxitérmino.

Por ejemplo, para $n=3$, existen ocho maxitérminos posibles que son:

$$\begin{array}{cccc} X_3 + X_2 + X_1 & X_3 + X_2 + X_1' & X_3 + X_2' + X_1 & X_3 + X_2' + X_1' \\ X_3' + X_2 + X_1 & X_3' + X_2 + X_1' & X_3' + X_2' + X_1 & X_3' + X_2' + X_1' \end{array}$$

Se les denominan maxitérminos porque solamente para una de las 2^n posibles combinaciones de los valores de las variables toma el valor de 0. Por ejemplo, el maxitérmino $(X_3' + X_2' + X_1)$ sólo toma el valor de 0 para $X_3 = 1$, $X_2 = 1$ y $X_1 = 0$; para cualquier otra combinación de valores de las variables tomará el valor de 1.

10.5 FORMAS CANONICAS DE FUNCIONES BOOLEANAS.

Existen dos formas canónicas únicas, en las cuales se puede representar una función booleana. En una de ellas se expresan los unos de la función (minitérminos) y en la otra los ceros de la función (maxitérminos).

Una de dichas formas, es llamada función canónica en minitérminos, o función canónica expresada como suma de productos. En esta forma, cada término es un minitérmino; por ejemplo:

$$F(A, B, C) = AB'C + A'BC' + A'BC + ABC$$

La otra forma es llamada función canónica en maxitérminos o función canónica expresada como productos de sumas. En esta forma, la función está expresada como una operación **AND** de maxitérminos; por ejemplo:

$$F(A,B,C) = (A + B + C) (A' + B + C) (A' + B' + C) (A + B + C')$$

La manera de expresar una función booleana en cualquiera de sus dos formas canónicas se describirá más adelante.

10.6. TABLA DE VERDAD.

Una tabla de verdad consiste en la enumeración de todas y cada una de las 2^n posibles combinaciones de los valores que pueden tomar las n variables independientes. A cada una de las 2^n combinaciones, se le añade el valor que toma la función booleana para esa combinación. La tabla de verdad es única para una función dada; es decir, una función booleana genera una y sólo una tabla de verdad.

Como ejemplo se muestra la tabla de verdad de la función siguiente:

$$F(A,B,C) = AB'C + A'BC' + A'BC + ABC$$

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Para construir la tabla de verdad es necesario generar todas y cada una de las 2^n posibles combinaciones de los valores de las variables independientes, y luego calcular el valor de la función booleana para cada una de estas combinaciones. Las 2^n combinaciones de valores de las variables no tienen que seguir ningún orden predeterminado; sin embargo, se ha hecho costumbre ver a cada una de estas combinaciones como un número binario y colocarlas en la tabla de verdad en forma ascendente.

Un procedimiento fácil y seguro de generar las 2^n posibles combinaciones de n variables ($X_n, X_{n-1}, \dots, X_2, X_1$) es el siguiente:

- En la variable X_1 alterna valores de cero y uno.
- En la variable X_2 alterna dos ceros y dos unos.
- En la variable X_i alternar 2^{i-1} ceros y 2^{i-1} unos.
- En la variable X_n colocar 2^{n-1} ceros seguidos de 2^{n-1} unos.

Este procedimiento coloca las combinaciones en el orden acostumbrado y evita la repetición u omisión de alguna combinación.

El valor de la función booleana, se obtiene al evaluar ésta con cada una de las combinaciones.

Si se examina la estructura de la función se pueden encontrar los valores que puede tomar de una forma más rápida que evaluando todas las combinaciones, como se describe a continuación:

Cuando la función se encuentra expresada como suma de productos, al garantizar que para determinada combinación de las variables de entrada, uno de los productos de la función vale uno, la función también tomará el valor de uno. Nótese que cada uno de los productos puede tomar el valor de uno para una (si es minitérmino) o varias combinaciones de los valores de las variables independientes. Al terminar de evaluar todos los productos las combinaciones que no tienen el valor de uno tomarán el valor de cero.

Cuando la función se encuentra expresada como producto de sumas, al garantizar que para determinada combinación de las variables de entrada, uno de las sumas de la función vale cero, la función también tomará el valor de cero. Nótese que cada una de las sumas puede tomar el valor de cero para una (si es maxitérmino) o varias combinaciones de los valores de las variables independientes. Al terminar de evaluar todas las sumas las combinaciones que no resultaron cero tomarán el valor de uno.

Como ejemplo, se obtendrá la tabla de verdad de las siguientes funciones:

$$G(A,B,C) = AB' + A'BC' + B'C$$

El producto AB' genera unos cuando $A = 1$ y $B = 0$.

El producto $A'BC'$ genera uno cuando $A = 0$, $B = 1$ y $C = 0$.

El producto $B'C$ genera unos cuando $B = 0$ y $C = 1$.

A	B	C	G
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Nótese que en una determinada combinación, más de un término puede generar un 1.

$$Y(A,B,C) = (A' + B')(A + B + C)(B' + C')$$

La suma $(A' + B')$ genera ceros cuando $A = 1$ y $B = 1$.

La suma $(A + B + C)$ genera cero cuando $A = 0$, $B = 0$ y $C = 0$.

La suma $(B' + C')$ genera ceros cuando $B = 1$ y $C = 1$.

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Nótese que en una determinada combinación, más de un suma puede generar un 0.

10.7. FUNCIONES EQUIVALENTES.

Una función booleana puede ser expresada en varias formas y aún representar la misma relación entre las variables independientes. Dos funciones booleanas, con las mismas variables, son equivalentes sí y sólo sí, generan la misma tabla de verdad. Por ejemplo, las funciones **G** y **Y** de la sección anterior son equivalentes ya que generan la misma tabla de verdad.

El lector podrá comprobar que las funciones:

$$F_1 = AB' + ABC + A'B'$$

$$F_2 = B' + ABC$$

$$F_3 = AC + B'$$

son equivalentes.

10.8. RESUMEN

En este capítulo se presentó una introducción al álgebra booleana, que es la mejor herramienta para diseñar funciones lógicas. Una variable booleana o lógica solamente puede tomar los valores **FALSO** y **VERDADERO**, que se representan como **cero** y **uno** respectivamente.

Se definieron también las operaciones del álgebra booleana que son la operación **AND** (Y), la operación **OR** (O) y la operación **NOT** (NO). Las primeras dos son operaciones binarias, es decir, actúan sobre dos operandos mientras que la tercera es una operación unaria que actúa sobre un solo operando.

En ausencia de paréntesis, la operación **NOT** tiene la mayor jerarquía, seguida por la operación **AND**, y la operación **OR** tiene la menor jerarquía. Al evaluar una expresión booleana deberán realizarse las operaciones de mayor jerarquía primero. Cuando existen paréntesis, deberán evaluarse primero los más internos y trabajar hacia afuera.

Una función booleana consta de una o más variables booleanas relacionadas mediante los operadores **NOT**, **AND** y **OR**. Las funciones booleanas son a su vez variables booleanas que representan una cierta relación lógica de sus variables independientes y sólo puede tomar los valores **FALSO** y **VERDADERO**. Para una función booleana de **n** variables independientes, existen 2^n combinaciones posibles de valores de éstas, dado que cada variable independiente solamente puede tomar dos valores diferentes.

Una función booleana puede representarse de muy diversas formas; sin embargo, existen dos formas canónicas para cualquier función booleana que son únicas. Estas son la forma canónica expresada en minitérminos y la forma canónica expresada en maxitérminos.

Un minitérmino tiene todas las variables independientes unidas por la operación **AND**, donde cada variable puede estar en su forma normal o complementada. Un minitérmino solamente toma el valor de uno para una de las 2^n combinaciones posibles de valores de las variables independientes.

Un maxitérmino contiene todas las variables independientes unidas por la operación **OR**, donde cada variable puede estar en su forma normal o complementada. Un maxitérmino solamente toma el valor de cero para una de las 2^n combinaciones posibles de valores de las variables independientes.

Una tabla de verdad muestra el valor que toma una función para cada una de las 2^n combinaciones posibles de valores de las variables independientes.

Dos funciones booleanas son equivalentes, sí y sólo sí, sus tablas de verdad son exactamente iguales.

10.9. PROBLEMAS

1. De las listas de funciones que se darán a continuación, diga cuales son equivalentes entre si:

- f(a,b) = $a + a \cdot b$
- g(a,b) = 1
- h(a,b) = $a \cdot a'$
- i(a,b) = a'
- j(a,b) = $a + b$
- k(a,b) = 0
- l(a,b) = $a \cdot b + a'$
- m(a,b) = $a' + a$
- n(a,b) = a
- p(a,b) = $ab + ab'$

2. Desarrolle las dos formas canónicas de cada una de las siguientes funciones:

a	b	c	f(a,b,c)	g(a,b,c)	h(a,b,c)	i(a,b,c)	j(a,b,c)	k(a,b,c)	l(a,b,c)	m(a,b,c)
0	0	0	0	0	1	1	1	0	0	0
0	0	1	0	0	1	1	1	1	1	1
0	1	0	1	0	1	1	1	1	0	0
0	1	1	1	0	1	1	1	1	1	1
1	0	0	0	0	1	1	1	1	1	1

1	0	1	0	0	1	1	0	1	0	1
1	1	0	1	1	0	0	0	0	0	0
1	1	1	0	0	1	0	1	1	1	1

3. Convierta las siguientes funciones a su forma canónica en minitérminos:

- a) $f(a,b,c) = ab+ac$
- b) $g(a,b,c) = 1$
- c) $h(a,b,c) = abc + ab + ac + a + ab' + ac' + aa'$
- d) $i(a,b,c) = (a+b+c)(a'+b'+c)(a+b+c')(a'+b+c')(a+b'+c)(a'+b+c)$
- e) $j(a,b,c) = (a+b)(a'+c)(a+b')(a'+b+c')$
- f) $k(a,b,c) = a(a'+b+c)$
- g) $l(a,b,c) = (a+b)(a+c)(a+b'+c')$
- h) $m(a,b,c) = (a+b)(b'+a+c)(c'+b)$
- i) $n(a,b,c) = (c+a+b')(b+a)(b+c')$
- j) $p(a,b,c,d) = (a+b+d)(a+c+d')(a'+b+c')(a'+c'+d)$

4. Convierta las siguientes funciones a su forma canónica en maxitérminos:

- a) $f(a,b,c) = a+b$
- b) $g(a,b,c) = 0$
- c) $h(a,b,c) = 1$
- d) $i(a,b,c) = (a+b+c)(a'+b'+c)(a+b+c')(a'+b+c')(a+b'+c)(a'+b+c)$
- e) $j(a,b,c) = abc+a'bc'+a'b'c$
- f) $k(a,b,c) = ab+ac+a'c+bc'$
- g) $l(a,b,c) = abc$
- h) $m(a,b,c) = abc'$
- i) $n(a,b,c) = ac'+abc+a$
- j) $p(a,b,c,d) = abd+ac'd'+a'b+cd$

CAPITULO 11

TEOREMAS DEL ALGEBRA BOOLEANA

11.1 TEOREMAS PRINCIPALES.

Existen varios teoremas en álgebra booleana, estos teoremas son de gran utilidad en la simplificación de funciones booleanas, o bien, al probar la equivalencia de dos funciones booleanas. A continuación se dan los principales teoremas del álgebra booleana.

Teoremas que definen el elemento **identidad**.

$$\begin{array}{ll} 1.a & A + 0 = A \\ 1.b & A \cdot 1 = A \end{array}$$

Teoremas de **idempotencia**.

$$\begin{array}{ll} 2.a & A + A = A \\ 2.b & A \cdot A = A \end{array}$$

Teoremas usando **ceros y unos**.

$$\begin{array}{ll} 3.a & A + 1 = 1 \\ 3.b & A \cdot 0 = 0 \end{array}$$

Teoremas en **complementación**.

$$\begin{array}{ll} 4.a & A + A' = 1 \\ 4.b & A \cdot A' = 0 \\ 5 & (A')' = A \end{array}$$

Teoremas de **De Morgan**.

$$\begin{array}{ll} 6.a & (A + B)' = A' \cdot B' \\ 6.b & (A \cdot B)' = A' + B' \end{array}$$

Generalización:

Para obtener el complemento de una función o expresión se hace lo siguiente:

Complementar las variables

Intercambiar los operadores **AND** por **OR** y los **OR** por **AND**.

Intercambiar los ceros por unos y los unos por ceros.

$$(F(A, B, \dots, Z, \bullet, +, 0, 1))' = F(A', B', \dots, Z', +, \bullet, 1, 0)$$

Teoremas de **conmutatividad**.

$$\begin{array}{ll} 7.a & A + B = B + A \\ 7.b & A \cdot B = B \cdot A \end{array}$$

Teoremas de **asociatividad**.

$$\begin{array}{ll} 8.a & A + (B + C) = (A + B) + C = A + B + C \\ 8.b & A \cdot (B \cdot C) = (A \cdot B) \cdot C = A \cdot B \cdot C \end{array}$$

Teoremas de **distributividad**.

$$\begin{array}{ll} 9.a & A + (B \cdot C) = (A + B) \cdot (A + C) \\ 9.b & A \cdot (B + C) = A \cdot B + A \cdot C \end{array}$$

Teoremas de **absorción**.

$$\begin{array}{ll} 10.a & A + A \cdot B = A \\ 10.b & A \cdot (A + B) = A \end{array}$$

Teoremas de **simplificación**.

$$\begin{array}{ll} 11.a & A \cdot B + A \cdot B' = A \\ 11.b & (A + B) \cdot (A + B') = A \\ 12.a & A + A' \cdot B = A + B \\ 12.b & A \cdot (A' + B) = A \cdot B \end{array}$$

Teoremas de **consenso**.

$$\begin{array}{ll} 13.a & A \cdot B + A' \cdot C + B \cdot C = A \cdot B + A' \cdot C \\ 13.b & (A + B) \cdot (A' + C) \cdot (B + C) = (A + B) \cdot (A' + C) \end{array}$$

11.2 DUALIDAD.

En álgebra booleana existe el principio de dualidad. Si se observan los teoremas anteriores se podrá ver que todos están dados en pares con la excepción del teorema 5. Cada uno de los teoremas es el dual del otro teorema del par.

Para formar el dual de un teorema o expresión se hace lo siguiente:

Las variables se dejan igual.
Intercambiar los operadores **AND** por **OR** y los **OR** por **AND**.
Intercambiar los ceros por unos y los unos por ceros.

$$(F(A, B, \dots, Z, \bullet, +, 0, 1))^D = F(A, B, \dots, Z, +, \bullet, 1, 0)$$

Lo importante del principio de dualidad es que si un teorema ha sido demostrado, su dual puede tomarse como cierto sin necesidad de demostrarlo.

11.3 PRUEBA POR INDUCCION PERFECTA.

La prueba por inducción perfecta, sirve para probar la equivalencia de dos expresiones booleanas, esto incluye tanto a las funciones como a los teoremas.

La prueba consiste en construir la tabla de verdad para cada una de las dos expresiones y compararlas. Si las dos tablas de verdad resultasen idénticas, las funciones son equivalentes. Cualquiera de los teoremas anteriores pueden ser demostrados por este método.

Como ejemplo considere el teorema 6.a y la tabla de verdad de cada una de las dos expresiones que involucra:

A	B	$(A + B)'$	$A' \cdot B'$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

Las tablas de verdad generadas por las expresiones $(A + B)'$ y $A' \cdot B'$ son idénticas; por lo tanto, el teorema 6.a ha sido probado y de acuerdo al principio de dualidad, el teorema 6.b, también puede considerarse como demostrado.

La prueba por inducción perfecta no es aconsejable cuando las expresiones tienen un número grande de variables, debido a que las tablas de verdad resultan muy grandes, por ejemplo, 512 combinaciones para 9 variables. Este tipo de pruebas puede usarse aún y cuando haya muchas variables, si se cuenta con la ayuda de una computadora digital.

11.4 PRUEBA POR DEDUCCION.

Este tipo de prueba consiste en la aplicación de los teoremas para demostrar la equivalencia de dos expresiones booleanas. Se pueden aplicar teoremas a una expresión hasta obtener la otra, o bien aplicar teoremas a ambas expresiones hasta obtener una tercera expresión equivalente para las dos.

Al aplicar un teorema, las variables que aparecen en éste pueden ser sustituidas por cualquier otra variable, o relación entre variables.

Como ejemplo suponga que los teoremas 1 al 9 han sido probados por inducción perfecta, y con ayuda de ellos se desea probar los teoremas 10.a y 11.a.

Teorema 10.a $A + A \cdot B = A$

Trabajando la expresión del lado izquierdo se tiene lo siguiente:

$$\begin{aligned}
A \cdot 1 + A \cdot B &= A && \text{(teorema 1.b)} \\
A \cdot (B + B') + A \cdot B &= A && \text{(teorema 4.a)} \\
A \cdot B + A \cdot B' + A \cdot B &= A && \text{(teorema 9.b)} \\
A \cdot B' + A \cdot B + A \cdot B &= A && \text{(teorema 7.a)} \\
A \cdot B' + A \cdot B &= A && \text{(teorema 2.a)} \\
A \cdot (B' + B) &= A && \text{(teorema 9.b)} \\
A \cdot 1 &= A && \text{(teorema 4.a)} \\
A &= A && \text{(teorema 1.b)}
\end{aligned}$$

Otra forma de probar este teorema es la siguiente:

$$\begin{aligned}
A \cdot 1 + A \cdot B &= A && \text{(teorema 1.b)} \\
A \cdot (1 + B) &= A && \text{(teorema 9.b)} \\
A \cdot 1 &= A && \text{(teorema 3.a)} \\
A &= A && \text{(teorema 1.b)}
\end{aligned}$$

Teorema 11.a $A \cdot B + A \cdot B' = A$

Trabajando la expresión del lado derecho se tiene lo siguiente:

$$\begin{aligned}
A \cdot B + A \cdot B' &= A \cdot 1 && \text{(teorema 1.b)} \\
A \cdot B + A \cdot B' &= A \cdot (B + B') && \text{(teorema 4.a)} \\
A \cdot B + A \cdot B' &= A \cdot B + A \cdot B' && \text{(teorema 9.b)}
\end{aligned}$$

Trabajando las dos expresiones se tiene lo siguiente:

$$\begin{aligned}
A \cdot (B + B') &= A \cdot 1 && \text{(teoremas 1.b y 9.b)} \\
A \cdot (B + B') &= A \cdot (B + B') && \text{(teoremas 4.a)} \\
\therefore A \cdot B + A \cdot B' &= A
\end{aligned}$$

Como se puede observar, el camino que se sigue para hacer la prueba por deducción no es único.

La demostración de los demás teoremas se puede realizar de una forma semejante. Se dejan como ejercicios para que el lector los compruebe.

11.5 APLICACION DE LOS TEOREMAS.

Los teoremas vistos anteriormente pueden ser usados para simplificar una función booleana, para probar la equivalencia de dos funciones booleanas, o bien para expresar una función booleana en cualquiera de sus formas canónicas. A continuación se muestran algunos ejemplos de la aplicación de los teoremas.

11.5.1 Simplificación de funciones usando teoremas.

La simplificación de funciones booleanas no constituye un ejercicio puramente académico, sino que es una forma de usar menos componentes físicos al hacer la implantación real de la función. La aplicación de los teoremas para simplificar funciones se verá a través de ejemplos.

Ejemplo 1. Simplificar a su mínima expresión la siguiente función:

$$F = X'Y'Z' + XYZ' + XYZ + X'Y'Z$$

$$F = X'Y'Z' + X'Y'Z + XYZ' + XYZ \quad (\text{teorema 7.a})$$

$$F = X'Y'(Z' + Z) + XY(Z' + Z) \quad (\text{teorema 9.b})$$

$$F = X'Y'(1) + XY(1) \quad (\text{teorema 4.a})$$

$$F = X'Y' + XY \quad (\text{teorema 1.b})$$

Ejemplo 2. Simplificar a su mínima expresión la siguiente función:

$$F = A'B'C + A'BC' + AB'C' + ABC'$$

$$F = A'B'C + A'BC' + AB'C' + ABC' + ABC' \quad (\text{teorema 2.a})$$

$$F = A'B'C + A'BC' + ABC' + AB'C' + ABC' \quad (\text{teorema 7.a})$$

$$F = A'B'C + (A' + A)BC' + AC'(B' + B) \quad (\text{teorema 9.b})$$

$$F = A'B'C + (1)BC' + AC'(1) \quad (\text{teorema 4.a})$$

$$F = A'B'C + BC' + AC' \quad (\text{teorema 1.b})$$

Ejemplo 3. Simplificar a su mínima expresión la siguiente función:

$$F = (X_1 + X_2 + X_3)(X_1 + X'_2 + X_3)(X'_1 + X_2 + X'_3)(X'_1 + X_2 + X_3)$$

$$F = (X_1 + X_3)(X'_1 + X_2) \quad (\text{teorema 11.b})$$

$$F = X_1X'_1 + X_1X_2 + X_3X'_1 + X_3X_2 \quad (\text{teorema 9.b})$$

$$F = X_1X_2 + X_3X'_1 + X_3X_2 \quad (\text{teoremas 4.b y 1.a})$$

$$F = X_1X_2 + X_3X'_1 \quad (\text{teoremas 13.a})$$

La forma $(X_1 + X_3)(X'_1 + X_2)$ también puede tomarse como una forma simplificada. Ya que una función booleana puede simplificarse y dejarse expresada como suma de productos o como producto de sumas, entonces la mínima expresión para esta función puede ser cualquiera de las siguientes formas:

$$F = (X_1 + X_3)(X'_1 + X_2) \quad \text{producto de sumas}$$

$$F = X_1X_2 + X_3X'_1 \quad \text{suma de productos}$$

Ejemplo 4. Simplificar a su mínima expresión la siguiente función:

$$F = AB + A'C + BCD$$

$$F = AB + A'C + BC + BCD \quad (\text{teorema 13.a})$$

$$F = AB + A'C + BC \quad (\text{teorema 10.a})$$

$$F = AB + A'C \quad (\text{teorema 13.a})$$

11.5.2 Expresar una función en forma canónica en minitérminos, mediante la aplicación de los teoremas.

La forma canónica en minitérminos es útil porque muestra en forma explícita y exhaustiva todas las combinaciones para las cuales la función vale uno. Recuérdese que cada minitérmino vale uno para una y sólo una de las 2^n combinaciones de valores de las variables.

Se trabajarán varios ejemplos para ilustrar la aplicación de los teoremas para obtener la forma canónica en minitérminos.

Ejemplo 1. Obtenga la forma canónica en minitérminos de la siguiente función:

$$F = XY' + YZ$$

$$\begin{aligned} F &= XY'(1) + YZ(1) && (\text{teorema 1.b}) \\ F &= XY'(Z + Z') + YZ(X + X') && (\text{teorema 4.a}) \\ F &= XY'Z + XY'Z' + XYZ + X'YZ && (\text{teoremas 9.b y 7.b}) \end{aligned}$$

Ejemplo 2. Obtenga la forma canónica en minitérminos de la siguiente función:

$$F = A'(BC + C') + (A + B)(B' + C)$$

$$\begin{aligned} F &= A'BC + A'C' + AB' + AC + BB' + BC && (\text{teorema 9.b}) \\ F &= A'BC + A'C'(1) + AB'(1) + AC(1) && \\ &\quad + BC(1) && (\text{teoremas 1.a, 1.b y 4.b}) \\ F &= A'BC + A'C'(B + B') + AB'(C + C') && \\ &\quad + AC(B + B') + BC(A + A') && (\text{teorema 4.a}) \\ F &= A'BC + A'BC' + A'B'C' + AB'C && \\ &\quad + AB'C' + ABC + AB'C + ABC + A'BC && (\text{teorema 9.b y 7.b}) \\ F &= A'BC + A'BC' + A'B'C' + AB'C && \\ &\quad + AB'C' + ABC && (\text{teorema 2.a}) \end{aligned}$$

Ejemplo 3. Obtenga la forma canónica en minitérminos de la siguiente función:

$$F = (A + B')(C' + AD')$$

$$\begin{aligned} F &= AC' + AAD' + B'C' + AB'D' && (\text{teoremas 9.b y 7.b}) \\ F &= AC' + AD' + B'C' + AB'D' && (\text{teorema 2.b}) \\ F &= AC'(1) + AD'(1) + B'C'(1) + AB'D'(1) && (\text{teorema 1.b}) \\ F &= AC'(B + B') + AD'(B + B') && \\ &\quad + B'C'(A + A') + AB'D'(C + C') && (\text{teorema 4.a}) \\ F &= ABC' + AB'C' + ABD' + AB'D' && \\ &\quad + AB'C' + A'B'C' + AB'CD' + AB'C'D' && (\text{teoremas 7.b y 9.b}) \\ F &= ABC' + AB'C' + ABD' + AB'D' && \end{aligned}$$

$$\begin{aligned}
& + A'B'C' + AB'CD' + AB'C'D' && \text{(teorema 2.a)} \\
F = & ABC'(1) + AB'C'(1) + ABD'(1) \\
& + AB'D'(1) + A'B'C'(1) + AB'CD' + AB'C'D' && \text{(teorema 1.b)} \\
F = & ABC'(D + D') + AB'C'(D + D') \\
& + ABD'(C + C') + AB'D'(C + C') \\
& + A'B'C'(D + D') + AB'CD' + AB'C'D' && \text{(teorema 4.a)} \\
F = & ABC'D + ABC'D' + AB'C'D + AB'C'D' \\
& + ABCD' + ABC'D' + AB'CD' + AB'C'D' \\
& + A'B'C'D + A'B'C'D' + AB'CD' + AB'C'D' && \text{(teoremas 7.b y 9.b)} \\
F = & ABC'D + ABC'D' + AB'C'D + AB'C'D' \\
& + ABCD' + AB'CD' + A'B'C'D + A'B'C'D' && \text{(teorema 2.a)}
\end{aligned}$$

De los ejemplos anteriores se puede observar que lo que se busca es que cada término contenga todas las variables. Para lograr esto se hace uso de los teoremas 9.b, 4.a y 1.b principalmente.

En muchas ocasiones se utiliza una forma condensada para expresar una función en su forma canónica en minitérminos. Esta forma se expresa como una sumatoria de los minitérminos, codificados por el valor decimal que representa la combinación de los valores de las variables independientes para cada minitérmino. La forma condensada para cada una de las funciones de los tres ejemplos anteriores es:

$$F(X, Y, Z) = \sum m(3, 4, 5, 7)$$

$$F(A, B, C) = \sum m(0, 2, 3, 4, 5, 7)$$

$$F(A, B, C, D) = \sum m(0, 1, 8, 9, 10, 11, 12, 13, 14)$$

11.5.3 Expresar una función en forma canónica en maxitérminos, mediante la aplicación de los teoremas.

La forma canónica en maxitérminos presenta en forma explícita y exhaustiva las combinaciones para las cuales la función vale cero. Recuérdese que cada maxitérmino vale cero para una y sólo una de las combinaciones posibles de valores de las variables.

Se trabajarán varios ejemplos para ilustrar la aplicación de los teoremas para obtener la forma canónica en maxitérminos.

Ejemplo 1. Obtenga la forma canónica en maxitérminos de la siguiente función:

$$F = (X' + Y')(X + Z)$$

$$\begin{aligned}
F = & ((X' + Y') + 0)((X + Z) + 0) && \text{(teoremas 1.a y 8.b)} \\
F = & ((X' + Y') + ZZ')((X + Z) + YY') && \text{(teorema 4.b)} \\
F = & ((X' + Y' + Z)(X' + Y' + Z')) \\
& ((X + Y + Z)(X + Y' + Z)) && \text{(teoremas 9a y 7a)} \\
F = & (X' + Y' + Z)(X' + Y' + Z')
\end{aligned}$$

$$(X + Y + Z) (X + Y' + Z) \quad (\text{teoremas 8.b})$$

Ejemplo 2. Obtenga la forma canónica en maxitérminos de la siguiente función:

$$F = (W + X' + Y) (X + Z') (W + X + Y + Z')$$

$$F = ((W + X' + Y) + 0) ((X + Z') + 0) (W + X + Y + Z') \quad (\text{teoremas 1.a y 8.b})$$

$$F = ((W + X' + Y) + ZZ') ((X + Z') + WW') (W + X + Y + Z') \quad (\text{teorema 4.b})$$

$$F = ((W + X' + Y + Z) (W + X' + Y + Z')) ((W + X + Z') (W' + X + Z')) (W + X + Y + Z') \quad (\text{teoremas 9a y 7a})$$

$$F = (W + X' + Y + Z) (W + X' + Y + Z') (W + X + Z') (W' + X + Z') (W + X + Y + Z') \quad (\text{teoremas 8.b})$$

$$F = (W + X' + Y + Z) (W + X' + Y + Z') ((W + X + Z') + 0) ((W' + X + Z') + 0) (W + X + Y + Z') \quad (\text{teoremas 1.a y 8.b})$$

$$F = (W + X' + Y + Z) (W + X' + Y + Z') ((W + X + Z') + YY') ((W' + X + Z') + YY') (W + X + Y + Z') \quad (\text{teorema 4.b})$$

$$F = (W + X' + Y + Z) (W + X' + Y + Z') ((W + X + Y + Z') (W + X + Y' + Z')) ((W' + X + Y + Z') (W' + X + Y' + Z')) (W + X + Y + Z') \quad (\text{teoremas 9a y 7a})$$

$$F = (W + X' + Y + Z) (W + X' + Y + Z') (W + X + Y + Z') (W + X + Y' + Z') (W' + X + Y + Z') (W' + X + Y' + Z') \quad (\text{teoremas 8.b y 4.b})$$

De los ejemplos anteriores se puede observar que si la función booleana está expresada en producto de sumas, lo que se debe de buscar es completar cada suma con todas las variables. Para lograrlo se hace uso de los teoremas 9.a, 4.b y 1.a.

Si la función booleana no se encuentra expresada en producto de sumas, el procedimiento para expresar ésta en su forma canónica en maxitérminos consiste en obtener el complemento de la función, expresarlo en minitérminos y finalmente, complementar para obtener la función expresada en máxitérminos. Se mostrará el procedimiento mediante ejemplos.

Ejemplo 3. Obtenga la forma canónica en maxitérminos de la siguiente función:

$$F = X'Y' + XZ$$

$$F' = (X'Y' + XZ)' = (X'Y')' (XZ)' \quad (\text{teorema 6.a})$$

$$F' = (X + Y) (X' + Z') \quad (\text{teorema 6.b})$$

$$F' = XX' + XZ' + X'Y + YZ' \quad (\text{teoremas 9.b y 7.b})$$

$$F' = XZ' + X'Y + YZ' \quad (\text{teoremas 4.b y 1.a})$$

$$F' = XZ'(1) + X'Y(1) + YZ'(1) \quad (\text{teorema 1.b})$$

$$F' = XZ'(Y + Y') + X'Y(Z + Z') + YZ'(X + X') \quad (\text{teorema 4.a})$$

$$\begin{aligned}
F' &= XYZ' + XY'Z' + X'YZ + X'YZ' + XYZ' + X'YZ' && \text{(teoremas 9.b y 7.b)} \\
F' &= XYZ' + XY'Z' + X'YZ + X'YZ' && \text{(teorema 2.a)} \\
F &= (XYZ' + XY'Z' + X'YZ + X'YZ')' \\
F &= (XYZ')' (XY'Z')' (X'YZ)' (X'YZ')' && \text{(teorema 6.a)} \\
F &= (X' + Y' + Z) (X' + Y + Z) (X + Y' + Z') \\
&\quad (X + Y' + Z) && \text{(teorema 6.b)}
\end{aligned}$$

Ejemplo 4. Obtenga la forma canónica en maxitérminos de la siguiente función:

$$F = (A' + BC') (A' + C)$$

$$\begin{aligned}
F' &= ((A' + BC') (A' + C))' \\
F' &= (A' + BC')' + (A' + C)' && \text{(teorema 6.b)} \\
F' &= A (BC')' + AC' && \text{(teorema 6.a)} \\
F' &= A (B' + C) + AC' && \text{(teorema 6.b)} \\
F' &= AB' + AC + AC' && \text{(teorema 9.b)} \\
F' &= AB'(1) + AC(1) + AC'(1) && \text{(teorema 1.b)} \\
F' &= AB'(C + C') + AC(B + B') + AC'(B + B') && \text{(teorema 4.a)} \\
F' &= AB'C + AB'C' + ABC + AB'C + ABC' + AB'C' && \text{(teoremas 9.b y 7.b)} \\
F' &= AB'C + AB'C' + ABC + ABC' && \text{(teorema 2.a)} \\
F &= (AB'C + AB'C' + ABC + ABC')' \\
F &= (AB'C)' (AB'C')' (ABC)' (ABC')' && \text{(teorema 6.a)} \\
F &= (A' + B + C') (A' + B + C) (A' + B' + C') \\
&\quad (A' + B' + C) && \text{(teorema 6.b)}
\end{aligned}$$

También existe una forma condensada para expresar una función en su forma canónica en maxitérminos. Esta forma se expresa como un producto de los maxitérminos, codificados por el valor decimal que representa la combinación de los valores de las variables independientes para cada maxitérmino. La forma condensada para cada una de las funciones de los cuatro ejemplos anteriores es:

$$F(X, Y, Z) = \prod M(0, 2, 6, 7)$$

$$F(W, X, Y, Z) = \prod M(1, 3, 4, 5, 9, 11)$$

$$F(X, Y, Z) = \prod M(2, 3, 4, 6)$$

$$F(A, B, C) = \prod M(4, 5, 6, 7)$$

11.6. RESUMEN

En este capítulo se presentaron los principales teoremas del álgebra booleana que sirven para simplificar funciones booleanas y para expresarlas en cualquiera de sus formas canónicas. También se definió el concepto de dualidad y su utilidad en el álgebra booleana

La prueba por inducción perfecta permite determinar la equivalencia de dos expresiones booleanas mediante la construcción de las tablas de verdad de ambas expresiones y la posterior comparación de estas tablas. La prueba por deducción consiste en aplicar los teoremas del álgebra booleana a una de las dos expresiones para llegar a la otra expresión, o bien aplicar teoremas a las dos expresiones para llegar a una tercera expresión equivalente a ambas.

Los teoremas del álgebra booleana pueden ser usados para simplificar funciones booleanas. Una función booleana puede expresarse en forma simplificada como suma de productos o como producto de sumas. La utilidad de representar una función booleana en su forma simplificada se verá en capítulos posteriores cuando se trate el tema de construcción de funciones booleanas. Lo que si se puede decir es que la construcción de una función booleana que esté expresada en cualquiera de sus formas mínimas o simplificadas requiere de menos componentes físicos para su implantación.

Otra de las muchas aplicaciones de los teoremas es la representación de una función booleana en cualquiera de sus formas canónicas. La forma canónica en minitérminos también se conoce como forma canónica expresada como suma de productos, mientras que la forma canónica en maxitérminos se denomina también como forma canónica expresada como producto de sumas.

11.7. PROBLEMAS.

1. Demuestre las siguientes equivalencias:

- a) $a' + ab' = a' + b'$
- b) $a + ab = ac + a$
- c) $(c+b)(b'+c) = (c+a')(a+c+d)(a+c+d')$
- d) $(a+c)(c'+d)(a+d') = abc' + ab'c' + ad$
- e) $xy' + w'y' + y'z + wy' + w = y' + wy$
- f) $((a+c)'(b'c'))' = a + bc + a'b'$
- g) $(c+d+e)(c'+f)(d'+f)(e+f) = (c+d+e)(c'+f)(d'+f)$
- h) $(a+c+e)(a'c'+e) = (e'(f+f'))'$
- i) $(c+d'+e')(c+d+e)(c+d+e')(c+d'+e) = (a+b+c)(a+b'+c)(a'+b'+c)(a'+b+c)$
- j) $abc' + a'c' + b'c' = c'$
- k) $((a+(bc'))' + (a+(a'b+bc')))' = (a+(bc'))'$
- l) $abcd + abc' + abd' = ab$
- m) $(xyz+xw)(xy'+xw') = x(w'yz+wy')$
- n) $(ac+a'c') + (a'c+c'a) = (c'd+c'd') + (cd'+cd)$

2. Simplifique las siguientes expresiones en su forma mínima como suma de productos:

- a) $bd + abcd + ad' + abc'd + abcd'$
- b) $a'b' + acd + bcd$
- c) $(acd+a'd'+c')(a'c'd'+ad+c)$
- d) $(abc)'(a'+b+c')$
- e) $ab'c + acd + bd' + ac'$
- f) $ab + ad + bc' + c'd + ab'd' + b'c'd'$
- g) $(a+b+c)(a+b+d)(a+b+c'+d')$
- h) $(b+c'+d)(a+b'+c'+d)(a'+c'+d)(b+c+d')$
- i) $abc + bd + a'bd' + bc'd'$

- j) $a + bd + a'b'c + a'cd'$
- k) $(ac' + bd)(a'b' + a'd' + c(b' + d'))$
- l) $a + b'c + a'b + a'bc$
- m) $abc' + bde' + a'bd' + a'be + bcd' + bce$
- n) $abc + a' + b' + c'$
- m) $f(a, b, c) = \sum m(0, 2, 4, 6, 7)$
- o) $f(a, b, c) = \prod M(0, 1, 2, 6)$

3. Simplifique las siguientes expresiones en su forma mínima como producto de sumas:

- a) $bd + abcd + ad' + abc'd + abcd'$
- b) $a'b' + acd + bcd$
- c) $(acd + a'd' + c')(a'c'd' + ad + c)$
- d) $(abc)'(a' + b + c')$
- e) $ab'c + acd + bd' + ac'$
- f) $ab + ad + bc' + c'd + ab'd' + b'c'd'$
- g) $(a + b + c)(a + b + d)(a + b + c' + d')$
- h) $(b + c' + d)(a + b' + c' + d)(a' + c' + d)(b + c + d')$
- i) $abc + bd + a'bd' + bc'd'$
- j) $a + bd + a'b'c + a'cd'$
- k) $(ac' + bd)(a'b' + a'd' + c(b' + d'))$
- l) $a + b'c + a'b + a'bc$
- m) $abc' + bde' + a'bd' + a'be + bcd' + bce$
- n) $abc + a' + b' + c'$
- m) $f(a, b, c) = \sum m(0, 2, 4, 6, 7)$
- o) $f(a, b, c) = \prod M(0, 1, 2, 6)$

4. Elabore las ecuaciones canónicas en minitérminos de las siguientes expresiones:

- a) $f(a,b,c) = a + b$
- b) $g(a,b,c) = ab + bc$
- c) $h(a,b,c,d) = bc' + bc$
- d) $i(a,b,c) = ab'c' + ab + ac$
- e) $j(a,b,c,d) = (ab + c)(a' + b'c')$
- f) $k(a,b,c,d) = (b'd + bc'd)(b + d')(b' + c + d')$
- g) $l(w,x,y,z) = (w' + y' + z)(w + y' + z)(w' + y + z)(w' + y' + z')$
- h) $m(a,b,c,d) = (a + b)'(c + d) + (a + b)'(c + d)'$
- i) $n(a,b,c,d) = ab + (a' + b')(c + d)'$
- j) $p(a,b,c,d) = abc + bcd + abd + acd$
- k) $q(a,b,c,d,e) = abcd + abce + abde + acde + bcde$
- l) $r(a,b,c,d,e) = a'(a + b' + c + d + e')(a + b' + c'd'e)(a + b)$
- m) $s(a,b,c,d,e) = (a + c')(a + c + b)(a + b')(c + b)'$
- n) $t(a,b,c,d,e,f) = (a'f' + be + cd')(b' + e')(a + f)(c' + d)(c + e'f + ab)$

5. Elabore las ecuaciones canónicas en maxitérminos de las siguientes expresiones:

- a) $f(a,b,c) = (a + b)(b + c')$
- b) $g(a,b,c) = (a + c)(b + c')(a' + b)$
- c) $h(a,b,c,d) = (a + b' + c + d)(a' + b' + c + d)$
- d) $i(a,b,c) = (a + bc)(a + b' + c')$
- e) $j(a,b,c,d) = (a + b + c)(b + c + d)$
- f) $k(a,b,c,d) = ac$
- g) $l(w,x,y,z) = wy' + w'xy' + w'x' + w'y + wy$
- h) $m(a,b,c,d) = (b + c' + d)(a'c + b)(a + b + c')$
- i) $n(a,b,c,d) = (ab + cd)(b' + c' + d)$
- j) $p(a,b,c,d) = a'bd' + bd' + a'd'$
- k) $q(a,b,c,d,e) = (a' + b' + c)(b + d' + e)(a' + b + c + e)$
- l) $r(a,b,c,d,e) = abc + abc' + ab'cd + ab'cd' + ab'c' + a'cd + a'cd' + a'c'$
- m) $s(a,b,c,d,e) = (a + c' + e)(a' + c + e')$
- n) $t(a,b,c,d,e) = (a' + d' + e')(a + b + c + d + e)(a' + b' + c' + d' + e')$

CAPITULO 12

OBTENCION DE FUNCIONES BOOLEANAS A PARTIR DE ESPECIFICACIONES LOGICAS

12.1 PASOS A SEGUIR PARA OBTENER LA FUNCION BOOLEANA.

En los capítulos anteriores se supone que la función booleana se conoce. Sin embargo, cuando se trata de resolver un problema real, el panorama es un poco diferente ya que la función booleana no se conoce. El principal problema se reduce a determinar la función booleana que resuelva el problema. Posteriormente esta función podrá ser simplificada o expresada en la forma que se desee.

Para obtener la función booleana que represente la solución a un problema real es necesario realizar los siguientes pasos:

- i) Determinar las variables involucradas, definiendo cuáles son de entrada (variables independientes) y cuáles son de salida (variables dependientes).
- ii) Codificar las variables en forma digital si es necesario. Esto consiste en asignarles valores de ceros y unos, que tengan relación con la situación que se desea resolver.
- iii) Establecer claramente las relaciones que existen entre entradas y salidas, de acuerdo con las especificaciones del problema.
- iv) Construir la tabla de verdad, usando todo lo definido en los pasos anteriores.
- v) Para cada una de las salidas, determinar su función booleana a partir de la tabla de verdad.
- vi) Por último, es conveniente expresar cada función booleana en la forma más apropiada para la solución del problema. Generalmente cada función se expresa en forma simplificada, ya sea como suma de productos o como producto de sumas; para que su construcción física sea más económica.

La mejor forma de comprender esta metodología es aplicarla a varios problemas específicos; por lo tanto, se resolverán varios ejemplos para ilustrar su aplicación.

Antes de pasar a resolver los ejemplos, es necesario estudiar cómo obtener una función booleana a partir de su tabla de verdad.

12.2 OBTENCION DE LA FUNCION BOOLEANA EN SU FORMA CANONICA EN MINITERMINOS A PARTIR DE SU TABLA DE VERDAD.

Para obtener una función booleana en forma canónica en minitérminos, a partir de su tabla de verdad, basta recordar lo que representa un minitérmino y aplicar este concepto. Un minitérmino vale uno

para una y sólo una de las 2^n posibles combinaciones de los valores de las variables independientes (variables de entrada); por consiguiente, cada combinación que tenga asignado el valor de uno en la tabla de verdad debe corresponder a un minitérmino de la función.

Una vez que se han obtenido todos los minitérminos, se realiza una operación **OR** de todos ellos, y se obtiene la función expresada en su forma canónica en minitérminos. La razón de ello, es que la función debe valer uno para cualquiera de las combinaciones que representan los minitérminos, lo cual sólo se logra con la operación **OR**.

12.3 OBTENCION DE LA FUNCION BOOLEANA EN SU FORMA CANONICA EN MAXITERMINOS A PARTIR DE SU TABLA DE VERDAD.

El procedimiento en este caso es similar al anterior. Un maxitérmino vale cero para una y sólo una de las combinaciones de las variables independientes (variables de entrada); por lo tanto, cada combinación que tenga asignado el valor de cero en la tabla de verdad debe corresponder a un maxitérmino de la función. Una vez que se han determinado todos los maxitérminos de la función, basta unirlos con una operación **AND** para obtener la función booleana expresada en su forma canónica en maxitérminos.

Los maxitérminos deben unirse por una operación **AND** porque la función debe valer cero para cualquiera de las combinaciones correspondientes a cada uno de los maxitérminos, y esto sólo se logra con la operación **AND**.

12.4 EJEMPLOS DE OBTENCION DE FUNCIONES BOOLEANAS A PARTIR DE ESPECIFICACIONES LOGICAS.

Se aplicará la metodología presentada anteriormente a la solución de cuatro problemas. Tres de ellos están directamente relacionados con aplicaciones computacionales y el último representa una situación un poco diferente.

Ejemplo1. Detector de números BCD.

Se tiene un número binario sin signo representado por cuatro bits, y se desea obtener una función booleana que indique si el número binario es un número BCD válido, o no.

Solución:

i) Identificación de variables:

Las variables independientes en este caso son los cuatro bits del número binario (entradas) y la variable dependiente es aquella, que indicará si el número binario es un número BCD válido o no. Se asignarán las variables a_3 , a_2 , a_1 y a_0 a los cuatro bits del número binario, siendo a_0 el menos significativo.

Se asignará la variable **z** a la salida o variable dependiente.

ii) Codificación en forma digital:

Las variables **a₃**, **a₂**, **a₁** y **a₀** ya están codificadas en forma digital; solamente queda por codificar la salida.

Se tomará la convención de que la variable **z** valga 1 si el número binario corresponde a un número BCD válido y cero en caso contrario.

iii) Relación entre entradas y salidas:

La salida **z** deberá ser uno, cuando el número binario representado por las variables **a₃**, **a₂**, **a₁** y **a₀** tenga un valor entre 0 y 9 inclusive y deberá valer cero, cuando el número binario represente un valor entre 10 y 15 inclusive.

iv) Construcción de la tabla de verdad:

Existen cuatro variables independientes; por lo tanto, la tabla de verdad constará de 16 renglones (2^4 combinaciones posibles).

En este caso se mostrará para cada combinación, el equivalente decimal del número binario. Esto con la finalidad de dar mayor claridad a la solución del problema.

a ₃	a ₂	a ₁	a ₀	z	Equivalente decimal
0	0	0	0	1	0
0	0	0	1	1	1
0	0	1	0	1	2
0	0	1	1	1	3
0	1	0	0	1	4
0	1	0	1	1	5
0	1	1	0	1	6
0	1	1	1	1	7
1	0	0	0	1	8
1	0	0	1	1	9
1	0	1	0	0	10
1	0	1	1	0	11
1	1	0	0	0	12
1	1	0	1	0	13
1	1	1	0	0	14
1	1	1	1	0	15

v) Determinación de la función booleana:

Se obtendrá la función booleana para **z** en sus dos formas canónicas.

Forma canónica en minitérminos:

$$\begin{aligned}
 z = & a_3' a_2' a_1' a_0' + a_3' a_2' a_1' a_0 + a_3' a_2' a_1 a_0' + a_3' a_2' a_1 a_0 \\
 & + a_3' a_2 a_1' a_0' + a_3' a_2 a_1' a_0 + a_3' a_2 a_1 a_0' + a_3' a_2 a_1 a_0 \\
 & + a_3 a_2' a_1' a_0' + a_3 a_2' a_1' a_0
 \end{aligned}$$

Forma canónica en maxitérminos:

$$\begin{aligned}
 z = & (a_3' + a_2 + a_1' + a_0) (a_3' + a_2 + a_1' + a_0') (a_3' + a_2' + a_1 + a_0) \\
 & (a_3' + a_2' + a_1 + a_0') (a_3' + a_2' + a_1' + a_0) (a_3' + a_2' + a_1' + a_0')
 \end{aligned}$$

vi) Simplificación de la función:

Se simplificarán ambas funciones para efectos de este ejemplo.

Simplificación de la forma canónica en minitérminos.

$$\begin{aligned}
 z = & a_3' a_2' a_1' a_0' + a_3' a_2' a_1' a_0 + a_3' a_2' a_1 a_0' + a_3' a_2' a_1 a_0 \\
 & + a_3' a_2 a_1' a_0' + a_3' a_2 a_1' a_0 + a_3' a_2 a_1 a_0' + a_3' a_2 a_1 a_0 \\
 & + a_3 a_2' a_1' a_0' + a_3 a_2' a_1' a_0
 \end{aligned}$$

Aplicando el teorema $AB + AB' = A$, a cada par de minitérminos agrupados según el orden en que se muestran, queda lo siguiente:

$$z = a_3' a_2' a_1' + a_3' a_2' a_1 + a_3' a_2 a_1' + a_3' a_2 a_1 + a_3 a_2' a_1'$$

Aplicando el mismo teorema a los términos primero y segundo, tercero y cuarto, y primero y quinto, se obtiene la forma mínima de la función expresada como suma de productos:

$$z = a_3' a_2' + a_3' a_2 + a_2' a_1'$$

Simplificación de la forma canónica en maxitérminos.

$$\begin{aligned}
 z = & (a_3' + a_2 + a_1' + a_0) (a_3' + a_2 + a_1' + a_0') (a_3' + a_2' + a_1 + a_0) \\
 & (a_3' + a_2' + a_1 + a_0') (a_3' + a_2' + a_1' + a_0) (a_3' + a_2' + a_1' + a_0')
 \end{aligned}$$

Aplicando el teorema $(A + B)(A + B') = A$, a cada par de maxitérminos agrupados en el orden como se muestran, queda lo siguiente:

$$z = (a_3' + a_2 + a_1') (a_3' + a_2' + a_1) (a_3' + a_2' + a_1')$$

Aplicando el mismo teorema al primer y tercer término y al segundo y tercero se obtiene la forma mínima de la función expresada como producto de sumas:

$$z = (a_3' + a_1') (a_3' + a_2')$$

Ejemplo2. Medio sumador.

Al sumar dos bits se obtiene un bit de suma y, algunas veces, otro bit de acarreo. Se desea obtener las funciones booleanas que represente la suma y el posible acarreo.

Solución:

i) Identificación de variables:

Las entradas (variables independientes) son los dos bits a sumar que se designarán por las variables **a** y **b**.

Las salidas son la suma que se designará por la variable **s** y el acarreo que se designará por la variable **c**.

ii) Codificación en forma digital:

Las variables ya están codificadas en forma digital.

iii) Relación entre entradas y salidas:

La variable **s** representa el bit resultante al sumar los bits representados por las variables **a** y **b**. La variable **c** representa el acarreo que se genera al realizar la suma.

iv) Construcción de la tabla de verdad:

La tabla de verdad consta de 4 renglones ya que solo hay 2 variables independientes.

a	b	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

v) Determinación de la función booleana:

Es necesario obtener dos funciones booleanas, una para la variable **s** y otra para **c**.

Forma canónica en minitérminos:

$$s = a'b + ab'$$
$$c = ab$$

Forma canónica en maxitérminos:

$$s = (a + b)(a' + b')$$
$$c = (a + b)(a + b')(a' + b)$$

vi) Simplificación de la función:

La única función que se puede simplificar es:

$$c = (a + b)(a + b')(a' + b)$$

Aplicando el teorema $(A + B)(A + B') = A$ al primero y segundo término y al primero con el tercer termino, queda la forma mínima de la función.

$$c = ab$$

Ejemplo3. Detector de números diferentes.

Se tienen dos números binarios sin signo representados en 2 bits cada uno y se desea obtener una función booleana que indique cuándo los dos números sean diferentes.

Solución:

i) Identificación de variables:

Para este problema se tienen cuatro entradas que representan los dos bits de cada uno de los números binarios, y una salida que indique cuando los números son diferentes.

Uno de los números binarios se representará por las variables $\mathbf{a_1}$ y $\mathbf{a_0}$, donde $\mathbf{a_0}$ es el bit menos significativo. El otro número se representará por las variables $\mathbf{b_1}$ y $\mathbf{b_0}$, siendo $\mathbf{b_0}$, el bit menos significativo.

La salida se representará por la variable \mathbf{f} .

ii) Codificación en forma digital:

Las entradas ya están codificadas en forma digital.

La salida se codificará asignándole el valor de uno si los números son diferentes, y cero si son iguales.

iii) Relación entre entradas y salidas:

La salida \mathbf{f} tomará el valor de uno si los números binarios representados por las variables $\mathbf{a_1, a_0}$ y $\mathbf{b_1, b_0}$ son diferentes, y cero cuando sean iguales.

iv) Construcción de la tabla de verdad:

Dado que hay cuatro entradas, la tabla de verdad constará de 16 renglones.

a ₁	a ₀	b ₁	b ₀	f
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

v) Determinación de la función booleana:

Se obtendrá la función booleana para **f** en sus dos formas canónicas.

Forma canónica en minitérminos:

$$f = a_1' a_0' b_1' b_0 + a_1' a_0' b_1 b_0' + a_1' a_0' b_1 b_0 + a_1' a_0 b_1' b_0' + a_1' a_0 b_1 b_0' + a_1' a_0 b_1 b_0 + a_1 a_0' b_1' b_0' + a_1 a_0' b_1' b_0 + a_1 a_0' b_1 b_0 + a_1 a_0 b_1' b_0' + a_1 a_0 b_1' b_0 + a_1 a_0 b_1 b_0'$$

Forma canónica en maxitérminos:

$$z = (a_1 + a_0 + b_1 + b_0) (a_1 + a_0' + b_1 + b_0') (a_1' + a_0 + b_1' + b_0) (a_1' + a_0' + b_1' + b_0')$$

vi) Simplificación de la función:

Se simplificarán ambas funciones para efectos de este ejemplo.

Simplificación de la forma canónica en minitérminos.

$$f = a_1' a_0' b_1' b_0 + a_1' a_0' b_1 b_0' + a_1' a_0' b_1 b_0 + a_1' a_0 b_1' b_0' + a_1' a_0 b_1 b_0' + a_1' a_0 b_1 b_0 + a_1 a_0' b_1' b_0' + a_1 a_0' b_1' b_0 + a_1 a_0' b_1 b_0 + a_1 a_0 b_1' b_0' + a_1 a_0 b_1' b_0 + a_1 a_0 b_1 b_0'$$

Aplicando el teorema $AB + AB' = A$, se obtiene lo siguiente:

$$f = a_1' a_0' b_0 + a_1' a_0' b_1 + a_1' a_0 b_0' + a_1' a_0 b_1 + a_1 a_0' b_1' + a_1 a_0' b_0 + a_1 a_0 b_1' + a_1 a_0 b_0'$$

Aplicando de nuevo el mismo teorema, se obtiene la forma mínima de la función expresada como suma de productos:

$$f = a_0' b_0 + a_1' b_1 + a_0 b_0' + a_1 b_1'$$

La forma mínima para **f** expresada como producto de sumas, es precisamente la forma canónica en maxitérminos.

Ejemplo 4.

El consejo directivo de una pequeña empresa está formado por tres personas. En la primera junta se decidió que la votaciones para decisiones importantes fueran secretas; sin embargo, existía el problema de que necesitaban que otra persona ajena contara los votos para que se mantuviera el secreto sobre cada voto.

Para evitar este problema, uno de ellos ideó el siguiente procedimiento para votar:

Se instalaría un botón debajo de la mesa en cada lugar y un pequeño foco en el centro de la mesa. Al momento de votar, cada una de las personas oprimiría el botón si estaba a favor, o no lo oprimiría si estaba en contra o se abstenía. El foco del centro de la mesa debería encenderse si la mayoría votaba a favor.

Solución:

i) Identificación de variables:

Existen tres entradas que representan la decisión de cada uno de los miembros del consejo, y una salida que indica el resultado de la votación. Las entradas se denominarán **a**, **b** y **c** y la salida **f**.

ii) Codificación en forma digital:

Se tomará la convención de que las entradas valdrán uno cuando se haya oprimido el botón correspondiente y cero en caso contrario. La salida **f** valdrá uno, cuando la decisión se haya aprobado por mayoría, y cero cuando se haya rechazado.

iii) Relación entre entradas y salidas:

La salida **f** será uno, cuando dos o más de las entradas valgan uno, y cero en caso contrario.

iv) Construcción de la tabla de verdad:

La tabla de verdad queda como sigue:

a	b	c	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

v) Determinación de la función booleana:

Se obtendrá la función booleana para **f** en sus dos formas canónicas.

Forma canónica en minitérminos:

$$f = a'bc + ab'c + abc' + abc$$

Forma canónica en maxitérminos:

$$f = (a + b + c)(a + b + c')(a + b' + c)(a' + b + c)$$

vi) Simplificación de la función:

Se simplificarán ambas funciones para efectos de este ejemplo.

Simplificación de la forma canónica en minitérminos.

$$f = a'bc + ab'c + abc' + abc$$

Aplicando el teorema $AB + AB' = A$, se obtiene la forma mínima de la función expresada como suma de productos:

$$f = bc + ac + ab$$

Simplificación de la forma canónica en maxitérminos.

$$f = (a + b + c)(a + b + c')(a + b' + c)(a' + b + c)$$

Aplicando el teorema $(A + B)(A + B') = A$, se obtiene la forma mínima de la función expresada como producto de sumas:

$$f = (a + b) (a + c) (b + c)$$

12.5 FUNCIONES NO ESPECIFICADAS COMPLETAMENTE

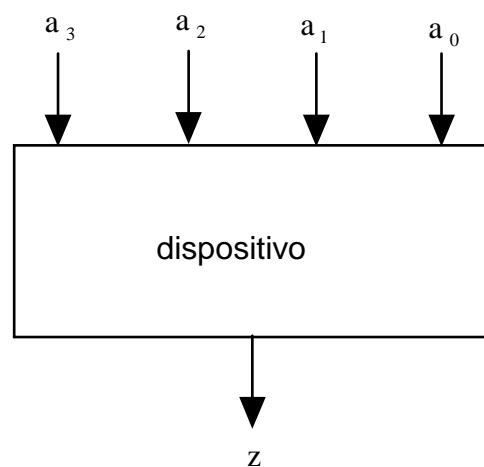
Hasta ahora se han tenido casos en los cuales las salidas están completamente definidas para todas las combinaciones posibles de las entradas; sin embargo, esto no ocurre en todos los diseños.

Existen algunas situaciones en las cuales las entradas no pueden tomar ciertas combinaciones o situaciones en las cuales para determinadas combinaciones de las variables de entrada no nos importa el valor que toma las funciones de salida. Estos valores de las variables de entrada deben determinarse a partir de las especificaciones lógicas del problema.

El problema que esto ocasiona es que el valor de las salidas para esas combinaciones no está definido. ¿Qué valor se le asignará a las salidas para estas combinaciones? Pensándolo detenidamente, el valor que se asigne a las salidas para estos casos es irrelevante para la solución del problema, dado que por definición, estas combinaciones de las entradas no pueden ocurrir.

Por otra parte, las ecuaciones booleanas para las salidas sí van a depender de los valores que se hayan asignado a las salidas para estos casos. En algunas ocasiones será conveniente asignarles el valor de uno y en otras el valor de cero para que la expresiones de las salidas se puedan simplificar mejor. Por lo pronto, el valor que se le asignará a las salidas para estos casos será **x** (**x** significa que el valor no importa). Posteriormente se mostrará cuando conviene que **x** valga cero o uno.

A manera de ejemplo considérese el diseño de un dispositivo que reciba como entrada un número BCD y que mediante una salida indique si dicho número es múltiplo de 3 o no.



Solución:

Entradas: a_3 , a_2 , a_1 y a_0 (representan un número BCD).

Salida: **z** indica si el número BCD es múltiplo de 3 o no.

Codificación de **z**: **z** será uno si el numero BCD es múltiplo de 3 y cero en caso contrario.

Tabla de verdad:

a ₃	a ₂	a ₁	a ₀	z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	x
1	0	1	1	x
1	1	0	0	x
1	1	0	1	x
1	1	1	0	x
1	1	1	1	x

Debido a que se especifica claramente que las entradas representan un número BCD, las combinaciones binarias que corresponden a los números del 10 al 15 inclusive no pueden ocurrir. El valor de la salida para estos casos no importa.

Si se asigna el valor de cero para la salida en todos los casos en la que ésta no importa se tendrá la siguiente función mínima expresada como suma de productos:

$$z = a_3'a_2'a_1a_0 + a_3'a_2a_1a_0' + a_3a_2'a_1'a_0$$

Si ahora se asignara el valor de uno para los casos en que la entrada representa los valores de 11 y 14, y cero para los valores de 10, 12, 13 y 15, se obtendría la siguiente función mínima expresada como suma de productos:

$$z = a_2'a_1a_0 + a_2a_1a_0' + a_3a_2'a_0$$

12.6. RESUMEN

En el capítulo 10 se introdujo el álgebra booleana y las funciones booleanas. Una función booleana representa una cierta relación lógica entre sus variables independientes. En el presente capítulo se

muestran los pasos a seguir para determinar una función booleana a partir de las especificaciones lógicas de un problema en particular.

Para obtener una función booleana a partir de las especificaciones lógicas en una situación dada, se realizan los siguientes pasos:

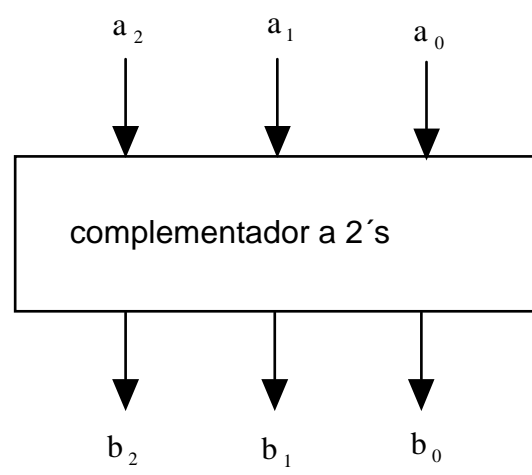
- i) Determinar las variables involucradas, definiendo cuáles son de entrada (variables independientes) y cuáles son de salida (variables dependientes).
- ii) Codificar las variables en forma digital si es necesario. Esto consiste en asignarles valores de ceros y unos, que tengan relación con la situación que se desea resolver.
- iii) Establecer claramente las relaciones que existen entre entradas y salidas, de acuerdo con las especificaciones del problema.
- iv) Construir la tabla de verdad, usando todo lo definido en los pasos anteriores.
- v) Para cada una de las salidas, determinar su función booleana a partir de la tabla de verdad.
- vi) Por último, es conveniente expresar cada función booleana en la forma más apropiada para la solución del problema. Generalmente cada función se expresa en forma simplificada, ya sea como suma de productos o como producto de sumas; para que su construcción física sea más económica.

Se presentaron varios ejemplos y se utilizaron también los teoremas para simplificar las funciones como suma de productos y como producto de sumas.

Por último, se discutieron algunos casos en los cuales las funciones booleanas no están especificadas completamente, debido a que sus entradas no pueden tomar todas las posibles combinaciones de valores. Para estos casos, es posible asignar el valor de cero o uno a la función para aquellas combinaciones que no son posibles. Una asignación adecuada permite obtener una forma más simplificada para la función. Este tema se volverá a retomar cuando se presenten otros métodos de simplificación de funciones booleanas. En éstos se definirá cómo se puede realizar la mejor asignación de valores de la función booleana para las combinaciones de valores de las entradas que no son posibles y así poder obtener la forma más simplificada posible.

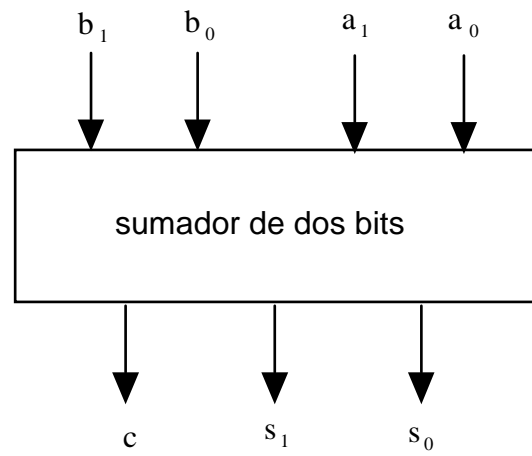
12.7. PROBLEMAS.

- 1.- Diseñe un dispositivo que obtenga el complemento a 2 de un número binario de 3 bits.



Obtenga las funciones mínimas para este dispositivo, en suma de productos y en producto de sumas.

2.- Diseñe un dispositivo que reciba como entrada dos números binarios de dos bits y dé como resultado la suma en dos bits y un posible acarreo.



Obtenga las funciones mínimas para este dispositivo, en suma de productos.

3.- Diseñe un dispositivo que tenga dos entradas de datos (**a** y **b**); y dos entradas de control (**c** y **d**). La salida deberá dar las operaciones siguientes dependiendo de las entradas de control:

c	d	z
0	0	$a + b$
0	1	$a \cdot b$
1	0	$(a + b)'$
1	1	$(ab)'$

Obtenga la función mínima para este dispositivo, como suma de productos y como producto de sumas.

CAPITULO 13

CIRCUITOS LOGICOS COMBINATORIOS

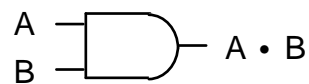
13.1 SIMBOLOS PARA LAS OPERACIONES LOGICAS

Las operaciones lógicas del algebra booleana y las funciones booleanas formadas a partir de las operaciones básicas, pueden ser construidas físicamente de muy diversas maneras. Pueden construirse a base de interruptores eléctricos, relevadores eléctricos, relevadores neumáticos, elementos mecánicos, elementos fluidicos, compuertas electrónicas o una combinación de varios tipos de elementos. Debido a la gran variedad de elementos físicos que pueden usarse para construir una función booleana, se ha pensado en usar una simbología que represente gráficamente el circuito lógico de la función, y que al mismo tiempo sea independiente de los elementos físicos que se desee utilizar, de tal forma que el circuito lógico de una función booleana, sea igual si se va a implementar con elementos fluidicos, o con elementos electrónicos.

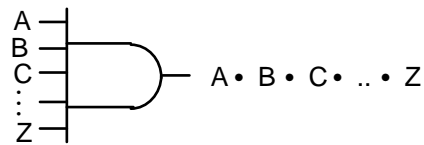
Existe una gran variedad de simbologías usadas para representar circuitos lógicos, pero afortunadamente una de estas formas ha tenido mucha aceptación entre las personas que trabajan en esta área y ha llegado a predominar sobre las otras. Esta simbología ha sido desarrollada por el servicio militar de los Estados Unidos, (MIL-STD-806B) y es la que se utilizará en el texto.

A continuación se describen los símbolos para las operaciones básicas.

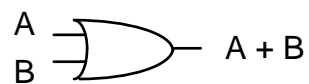
El símbolo usado para la operación **AND**, es el siguiente:



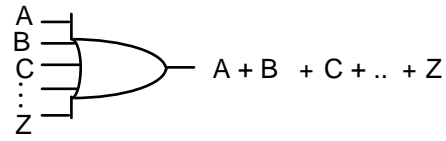
Este símbolo puede tener más de dos entradas, pero el número máximo de entradas depende del elemento físico que se vaya a usar. En este texto se supondrá que no se tiene un límite en el número de entradas.



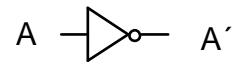
El símbolo usado para la operación **OR**, es el siguiente:



Este símbolo sigue exactamente las mismas reglas que el anterior en lo referente al número de entradas.



El símbolo usado para la operación **NOT**, es el siguiente:



Recuérdese que esta operación actúa sobre un sólo operando, por lo tanto, su símbolo sólo acepta una entrada.

Cada uno de los símbolos anteriores es conocido como bloque lógico o compuerta lógica.

13.2 CIRCUITOS LOGICOS DE FUNCIONES BOOLEANAS

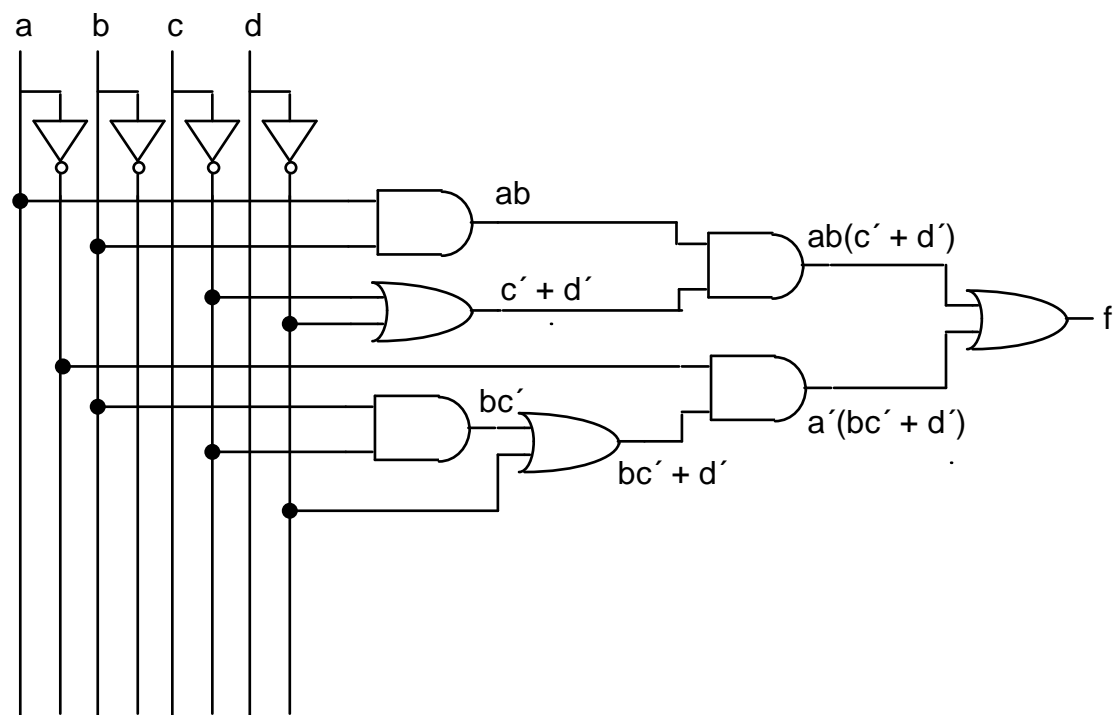
Dado que cada bloque lógico representa una operación básica y las funciones booleanas están construidas en base a operaciones básicas, cualquier función booleana puede ser representada por un circuito lógico. Dicho circuito estará compuesto por bloques lógicos.

Anteriormente se vio que una función booleana puede ser expresada en muy diferentes maneras sin alterar la relación que existe entre las variables. Por consiguiente, es lógico suponer que una misma función booleana puede ser representada de diversas formas usando bloques lógicos. El circuito lógico de una función booleana no es único, lo que sí es único es la relación que debe existir entre entradas y salidas (i.e. la tabla de verdad sí es única).

La manera de construir el circuito lógico de una función booleana se ilustrará a través del siguiente ejemplo.

Ejemplo: Construcción del circuito lógico para representar la función booleana dada por:

$$f = ab(c' + d') + a'(bc' + d')$$



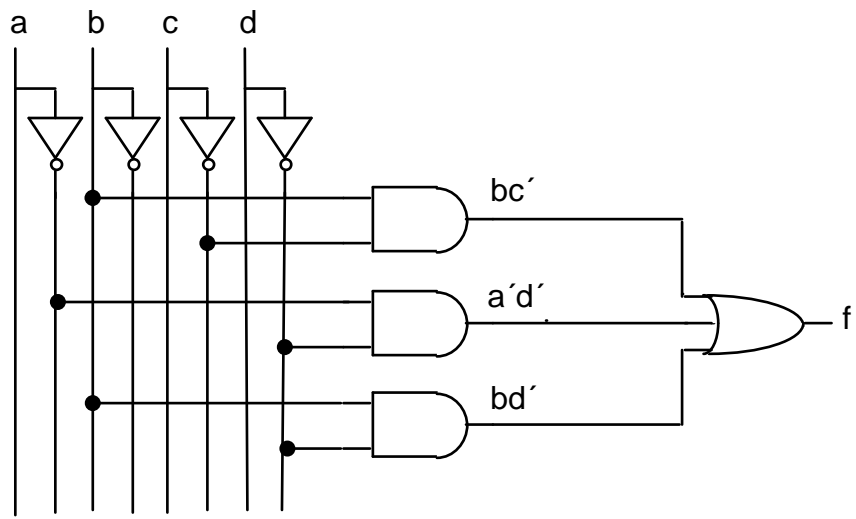
Nota: Si dos líneas se cruzan pero no están marcadas con un punto, esto significa que no están en contacto; si se cruzan y están marcadas con un punto (•), entonces significa que sí están conectadas.

Ahora se podrá valorar lo importante que es la simplificación de una función booleana, ya que entre menos bloques lógicos tenga el circuito, su construcción física será más económica y obviamente, entre más simplificada esté una función, su circuito lógico tendrá menos bloques lógicos. Se procederá a simplificar la función anterior y construir su circuito lógico para compararlo con el circuito anterior.

$$f = ab(c' + d') + a'(bc' + d') = abc' + abd' + a'bc' + a'd'$$

$$f = (a + a')bc' + d'(ab + a') = bc' + d'(a' + b)$$

$$f = bc' + a'd' + bd'$$



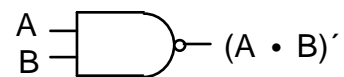
Otro factor importante en un circuito es el número de niveles, el cual está definido por el número máximo de compuertas que tiene que viajar una señal desde una variable independiente hasta la salida que representa la función. El número de niveles determina el retardo del circuito, esto es, el tiempo que tarda en ser válido el valor de la salida, una vez que los valores de las entradas se proporcionaron. Si se considera que el valor de las variables independientes y sus complementos se tienen como salidas del nivel cero, el primer circuito tiene cuatro niveles y el circuito simplificado solamente dos.

13.3 OPERACIONES LOGICAS ESPECIALES Y SUS CIRCUITOS.

Las tres operaciones lógicas que hasta ahora han sido utilizadas, son las operaciones lógicas básicas, las cuales son suficientes para construir cualquier función booleana. Sin embargo, se han agregado algunas otras operaciones lógicas, que pueden construirse a partir de las operaciones básicas, pero que han llegado a considerarse como bloques lógicos independientes por alguna u otra razón. Estos bloques lógicos son los correspondientes a las operaciones NAND (NO-Y) y NOR (NO-O), las cuales se describen a continuación.

13.3.1 Operación NAND (NO-Y).

La operación **NAND** es la negación de la operación **AND** y se representa de la siguiente forma:



La siguiente tabla de verdad define su funcionamiento.

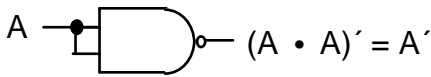
A	B	$(A \cdot B)'$
0	0	1
0	1	1

1	0	1
1	1	0

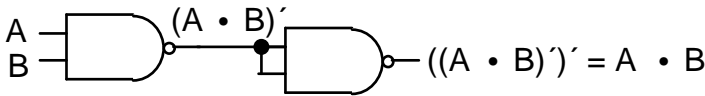
Este símbolo, al igual que el **AND**, puede tener más de dos entradas.

El bloque lógico **NAND** es un bloque funcionalmente completo, ya que con éste se pueden construir las tres operaciones básicas como se muestra a continuación:

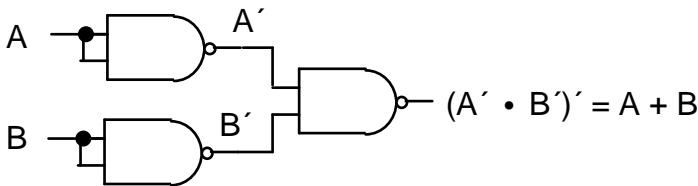
NOT



AND

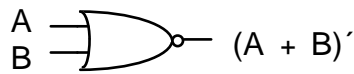


OR



13.3.2 Operación NOR (NO-O).

Esta operación es la negación de la operación **OR**, y se representa de la siguiente forma:



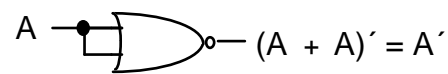
La siguiente tabla de verdad define su funcionamiento.

A	B	$(A + B)'$
0	0	1
0	1	0
1	0	0
1	1	0

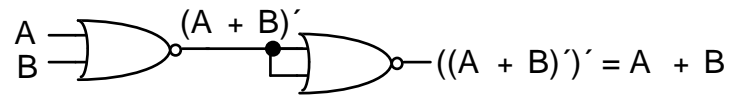
Este símbolo, al igual que el **OR** puede tener más de dos entradas.

El bloque lógico **NOR** también es un bloque funcionalmente completo, ya que con éste se pueden construir las tres operaciones básicas como se ilustra a continuación:

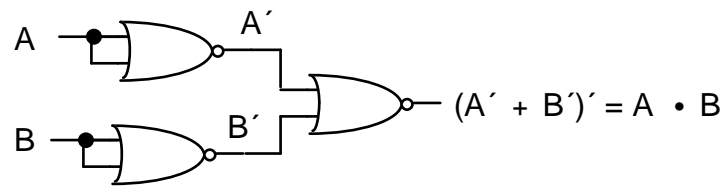
NOT



OR



AND



13.4 CONSTRUCCION DE CIRCUITOS LOGICOS UTILIZANDO BLOQUES NAND Y NOR.

Cualquier función booleana puede ser construida utilizando solamente bloques **NAND**, o bien, utilizando solamente bloques **NOR**. Existen fabricantes de bloques lógicos electrónicos que sólo construyen ya sea bloques **NAND** y bloques **NOT**, o bien bloques **NOR** y bloques **NOT**. La razón por la cual los fabricantes construyen sólo este tipo de bloques es evidente, se evitan el tener que construir varios tipos de bloques y, con los que construyen, se puede construir cualquier función booleana.

Por esta razón, es importante poder diseñar un circuito lógico usando solamente bloques **NAND** o bloques **NOR**. En las siguientes secciones se presentará la forma de diseñar circuitos lógicos utilizando solamente bloques de uno de estos tipos.

13.4.1 Construcción de circuitos lógicos usando bloques NAND.

Para construir el circuito más económico con bloques **NAND**, se necesita simplificar la función y dejarla expresada como suma de productos. Una vez que se ha hecho esto, se obtiene el doble complemento de la función (Recuérdese el teorema 5, $(A')' = A$), y la función quedará expresada con bloques **NAND** y **NOT**. A continuación se muestran tres ejemplos:

Ejemplo 1.

Construir el circuito lógico más económico de la función dada utilizando sólo bloques **NAND** y **NOT**.

$$f = a(b + a'c) + c(b + ab')$$

Simplificación:

$$f = a(b + a'c) + c(b + a)$$

$$f = ab + aa'c + bc + ac$$

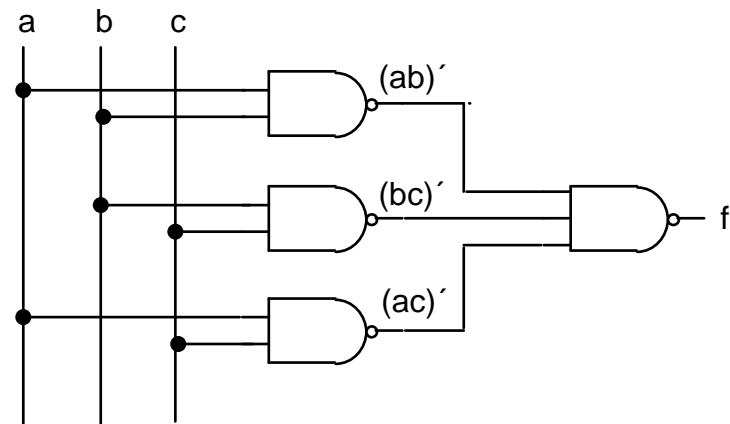
$$f = ab + bc + ac$$

Obtención del doble complemento:

$$f' = (ab)'(bc)'(ac)'$$

$$f = ((ab)'(bc)'(ac)')'$$

Como se puede observar, la función ha quedado expresada en base operaciones **NAND** solamente; su circuito lógico se muestra a continuación:



Ejemplo 2.

Para la función dada, desarrolle el circuito lógico más económico utilizando sólo bloques **NAND** y **NOT**.

$$f = abc' + a'(b + b'c) + ab'c$$

Simplificación:

$$f = abc' + a'(b + c) + ab'c$$

$$f = abc' + a'b + a'c + ab'c$$

$$f = abc' + a'b + a'c + ab'c + bc' + b'c$$

$$f = a'b + a'c + bc' + b'c$$

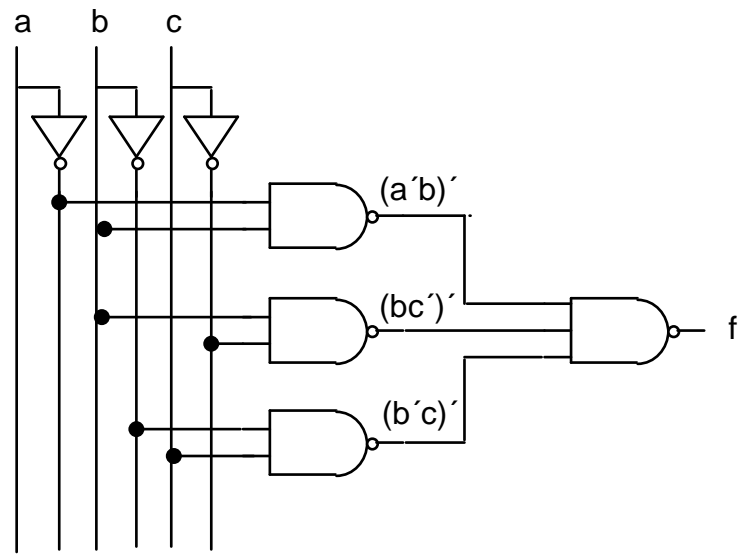
$$f = a'b + bc' + b'c$$

Obtención del doble complemento:

$$f' = (a'b)'(bc')'(b'c)'$$

$$f = ((a'b)'(bc')'(b'c)')'$$

La función está expresada con operaciones **NAND**, pero ahora se tienen variables complementadas. Para obtener los complementos de las variables se usan bloques **NOT**. El circuito lógico de la función se muestra a continuación:



Ejemplo 3.

Para la función que se muestra a continuación obtenga el circuito lógico más económico usando bloques **NAND** y **NOT**.

$$f = x + x'yz + y'z'$$

Simplificación:

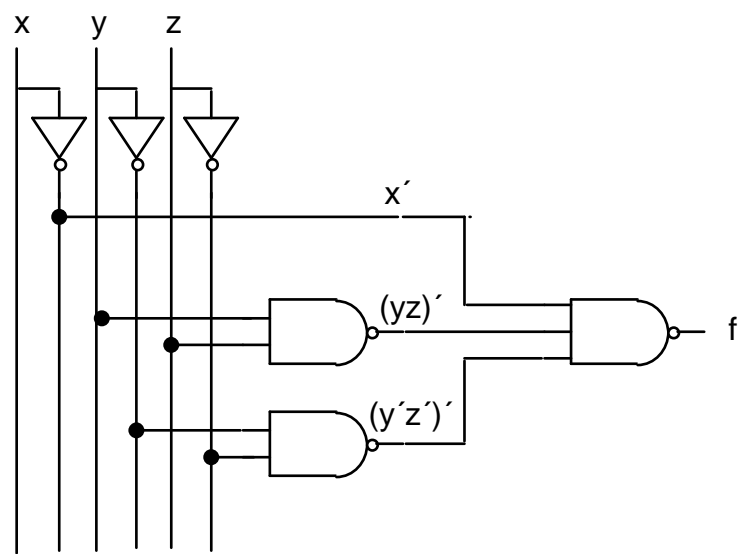
$$f = x + yz + y'z'$$

Obtención del doble complemento:

$$f' = x' (yz)' (y'z')'$$

$$f = (x' (yz)' (y'z')')'$$

Nótese la diferencia, en los casos anteriores las variables que se alimentan a cada bloque **NAND** son exactamente las mismas que las de la función expresada como suma de productos, en el caso de que exista una variable sólo, ésta deberá complementarse como se muestra en el circuito.



El lector podrá asegurar después de ver estos ejemplos, que la construcción del circuito lógico de una función booleana utilizando solamente bloques **NAND**, es relativamente sencilla, el único caso de cuidado es el que se mencionó en este último ejemplo. Adicionalmente, se tiene un circuito lógico de dos niveles, sin contar los bloques NOT para obtener los complementos de las variables independientes.

13.4.2 Construcción de circuitos lógicos usando bloques NOR.

La construcción del circuito lógico más económico usando bloques **NOR** es muy similar a la construcción utilizando bloques **NAND**. En este caso tiene que expresarse la forma simplificada de la función como productos de sumas y luego obtener el doble complemento.

Para encontrar la mínima expresión de la función como productos de sumas, lo más fácil de hacer cuando la función original no está expresada en producto de sumas, es complementar la función y simplificarla dejándola expresada como suma de productos. Después se complementa y esto da la función simplificada expresada como productos de sumas. Se trabajarán tres ejemplos para ilustrar el procedimiento.

Ejemplo 1.

Construir el circuito lógico más económico de la función dada utilizando sólo bloques **NOR** y **NOT**.

$$f = a(b + a'c) + c(b + ab')$$

Simplificación:

$$f' = (a' + (b'(a + c')))(c' + (b'(a' + b)))$$

$$f' = (a' + ab' + b'c')(c' + a'b' + b'b)$$

$$f' = (a' + b')(c' + a'b')$$

$$f' = a'c' + a'b' + b'c'$$

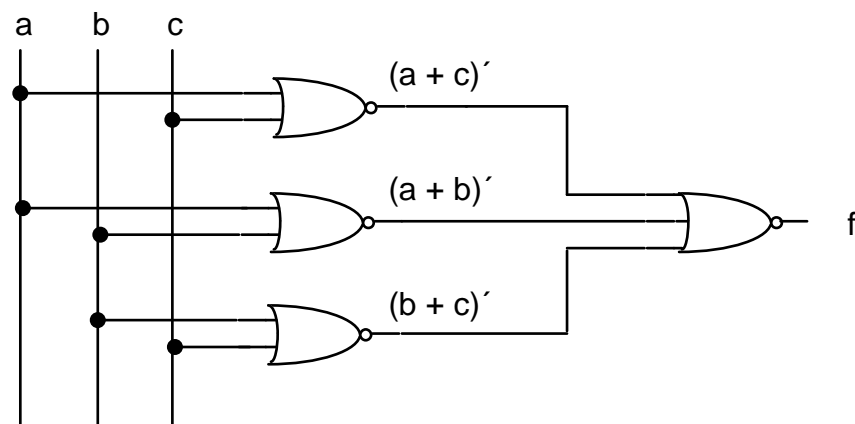
$$f = (a + c) (a + b) (b + c)$$

Obtención del doble complemento:

$$f' = (a + c)' + (a + b)' + (b + c)'$$

$$f = ((a + c)' + (a + b)' + (b + c)')'$$

De esta forma la función ha sido expresada con operaciones **NOR** solamente, su circuito lógico se muestra a continuación:



Ejemplo 2.

Usando bloques **NOR** y **NOT**, desarrolle el circuito lógico más económico de la función booleana que se muestra a continuación:

$$f = xy + yz'$$

Simplificación:

$$f' = (x' + y') (y' + z)$$

$$f' = x'y' + x'z + y'y' + y'z$$

$$f' = x'y' + x'z + y' + y'z$$

$$f' = y' + x'z$$

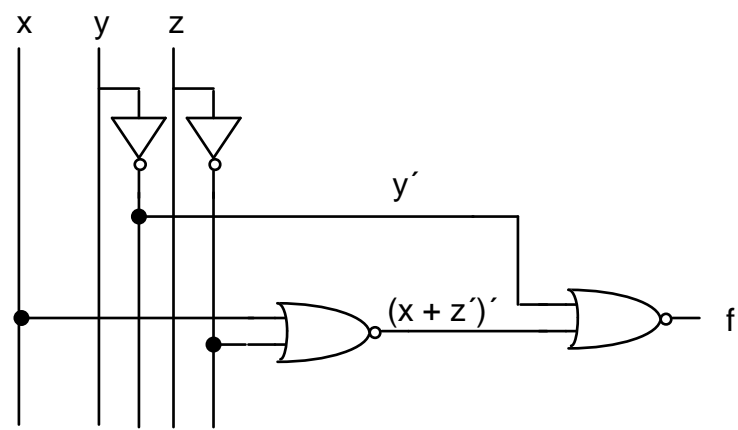
$$f = y (x + z')$$

Obtención del doble complemento:

$$f' = y' + (x + z')$$

$$f = (y' + (x + z'))'$$

Obsérvese que la variable **y** aparece complementada en la función con el doble complemento. A continuación se muestra el circuito lógico de la función.



Ejemplo 3.

Para la función dada obtenga el circuito lógico más económico usando bloques **NOR** y **NOT** solamente.

$$f = (a' + b') (a + b) (a' + c) (b' + c)$$

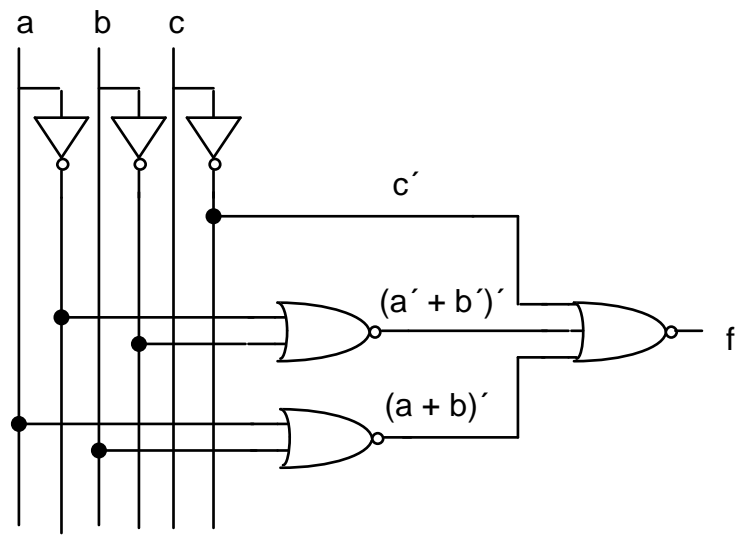
Simplificación:

$$\begin{aligned} f &= (a' + b') (a + b) (a' + c) (b' + c) (b + c) \\ f &= (a' + b') (a + b) (a' + c) c \\ f &= c (a' + b') (a + b) \end{aligned}$$

Obtención del doble complemento:

$$\begin{aligned} f' &= c' (a' + b')' (a + b)' \\ f &= (c' (a' + b')' (a + b)')' \end{aligned}$$

A continuación se muestra el circuito lógico de la función.



Obsérvese que en este caso también se obtienen circuitos lógicos de dos niveles, sin contar los bloques NOT utilizados para obtener los complementos de las variables de entrada.

13.5 CIRCUITOS COMBINATORIOS DE DOS NIVELES.

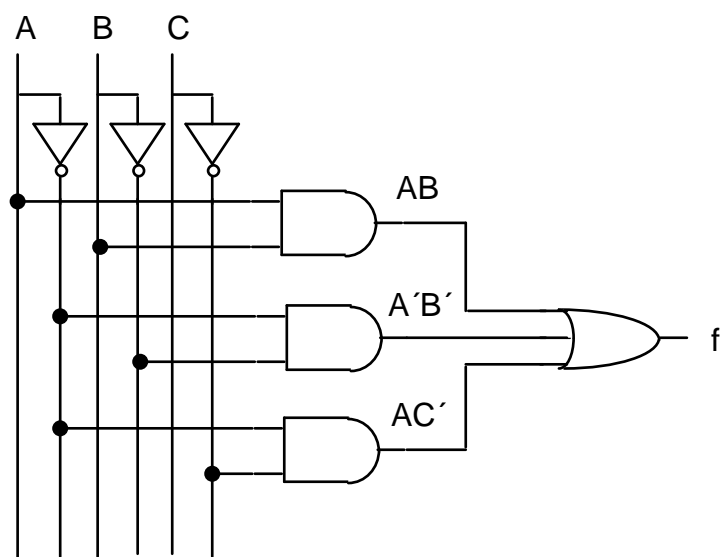
Dado que al simplificar una función ésta puede quedar expresada como suma de productos o como productos de sumas, existen dos circuitos combinatorios para construir cada una de estas expresiones.

Si la función se encuentra expresada en suma de productos, los circuitos combinatorios serían: el circuito **AND-OR** y el circuito **NAND-NAND**. El nombre del circuito se forma del nombre de los bloques que pertenecen al primer nivel junto con el nombre del bloque del segundo nivel. A continuación se da un ejemplo.

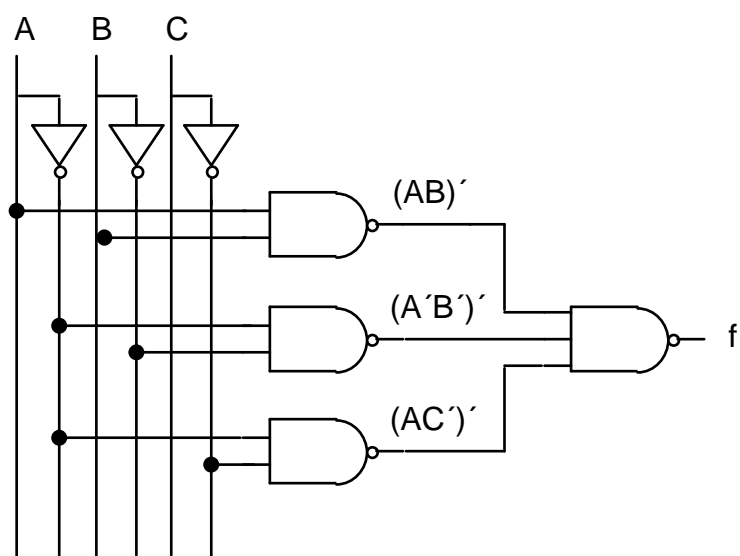
Función:

$$f = AB + A'B' + AC'$$

Circuito **AND-OR**:



Circuito **NAND-NAND**:

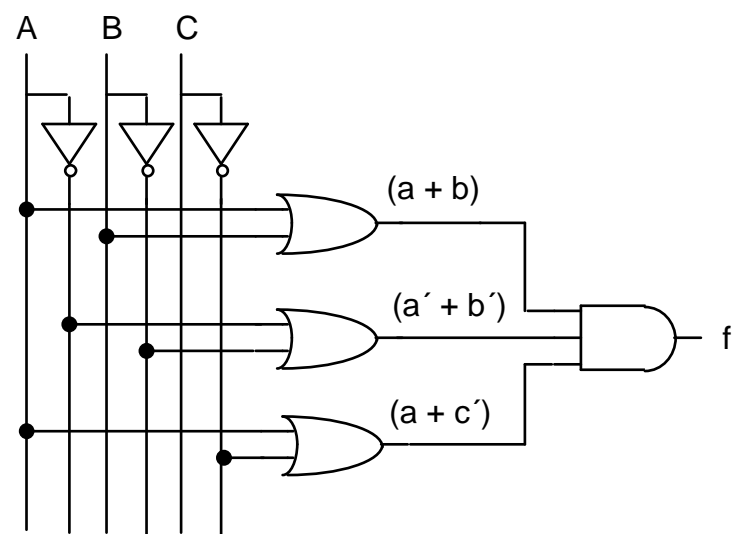


Si la función se encuentra expresada en producto de suma los circuitos combinatorios serian: el circuito **OR-AND** y el circuito **NOR-NOR**. A continuación se da un ejemplo.

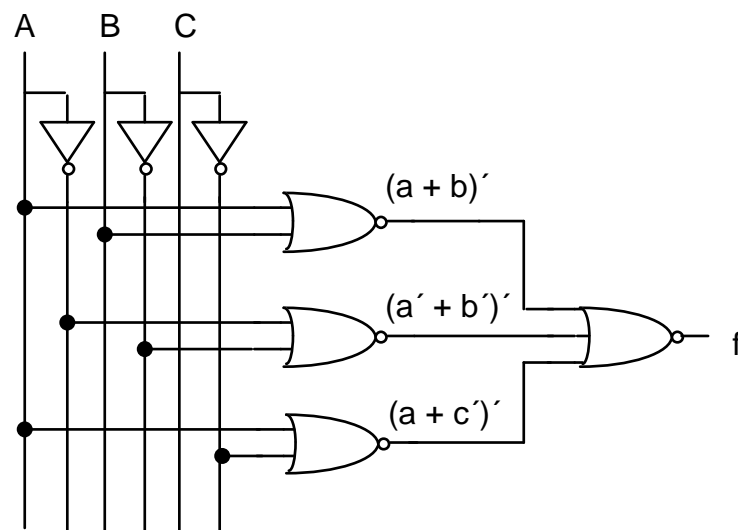
Función:

$$f = (a + b) (a' + b') (a + c')$$

Circuito **OR-AND**:



Circuito **NOR-NOR**:



Si se tiene una función que no está expresada como suma de productos, ni como productos de sumas, entonces no se puede construir un circuito de dos niveles.

El lector podrá comprobar que si una función expresada como suma de productos se construye utilizando bloques **NOR**, no se podrá construir un circuito de dos niveles ya que requiere de una negación al final del circuito. Lo mismo ocurre si se construye el circuito lógico de una función expresada como producto de sumas usando bloques **NAND**

13.6 RESUMEN.

En este capítulo se presentaron los símbolos de las operaciones lógicas elementales utilizados para construir circuitos de funciones lógicas y se mostró la construcción de circuitos lógicos utilizando estos símbolos.

Se introdujeron las operaciones lógicas NAND (NO-Y) y NOR (NO-O) así como los símbolos usados para representarlas. Estas operaciones son importantes ya que en el mercado se consiguen fácilmente circuitos integrados que las realizan. También se analizó la construcción de circuitos lógicos de dos niveles utilizando solamente bloques lógicos NAND y NOT, por lo cual es necesario representar la función lógica como suma de productos. Si la función lógica se representa como producto de sumas, se puede construir un circuito lógico de dos niveles utilizando solamente bloques lógicos NOR y NOT.

Los bloques lógicos reales usados para la construcción de circuitos lógicos tienen cierto tiempo de retardo desde que se presentan las entradas hasta que se tienen disponibles las salidas. Entre más niveles tenga un circuito lógico, mayor será el tiempo de retardo desde que se presentan las salidas hasta que se tiene disponible la salida. La ventaja de usar circuitos lógicos de dos niveles estriba en que se puede diseñar el circuito lógico con el menor retardo posible, lo cual permite construir dispositivos más rápidos.

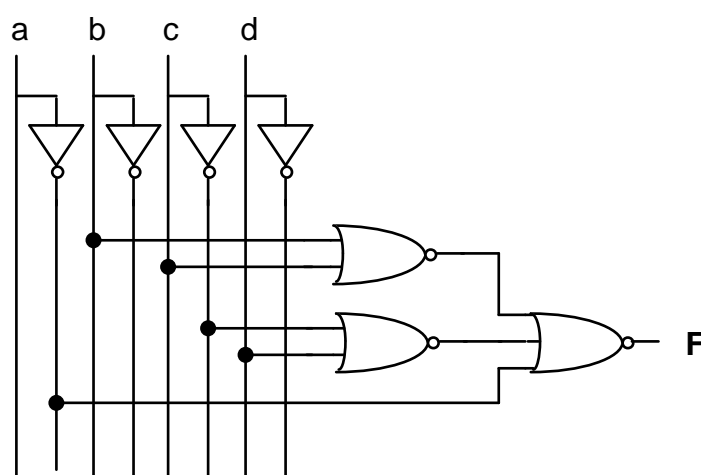
13.7 PROBLEMAS.

1.- Sin modificar las siguientes funciones, construya el circuito lógico usando bloques AND, OR y NOT.

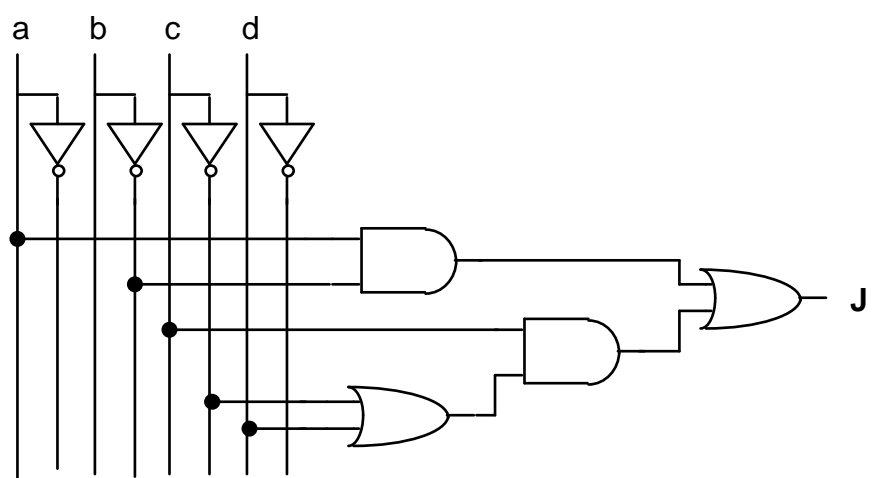
- a.- $f = ab' + a'(c' + bcd')$
- b.- $h = (d + ef'(a + b))c' + d(ab' + d'e)$
- c.- $g = (x + z'w)y + xy'(w + x'y)$

2.- Obtenga las funciones que realizan los siguientes circuitos lógicos, y construya el mínimo circuito lógico NAND-NAND.

a)



b)



CAPITULO 14

MINIMIZACION DE FUNCIONES BOOLEANAS

14.1. INTRODUCCION

En capítulos anteriores se ha enfatizado la necesidad de simplificar las funciones booleanas. Una función simplificada genera un circuito lógico sencillo y con pocos componentes, comparado con el circuito lógico de la misma función sin simplificar. Hasta ahora se ha hecho uso de los teoremas para simplificar las funciones booleanas, pero el lector se habrá dado cuenta de que al aplicar los teoremas no es fácil determinar si la expresión que se tiene en determinado momento es la forma más simple de la función. Esta forma de la función se denomina forma mínima y siempre es posible llegar a ella usando los teoremas.

Se han desarrollado varios métodos, basados en la aplicación sistemática de los teoremas del álgebra booleana, que permiten obtener la forma mínima de una función. Básicamente existen dos formas mínimas para cada función booleana, la forma mínima expresada como suma de productos y la forma mínima expresada como producto de sumas. Aún cuando ambas son formas mínimas, los circuitos lógicos no necesariamente contienen el mismo número de bloques lógicos, pero ambas son importantes dado que la primera de ellas produce directamente el circuito lógico NAND-NAND, mientras que la segunda produce el circuito lógico NOR-NOR, de tal suerte que la selección de una u otra forma mínima depende del tipo de bloques lógicos que se desee usar.

Cuando no se ha decidido qué tipo de bloques lógicos se va a usar, conviene obtener ambas formas mínimas y seleccionar aquella que de el circuito lógico con menos bloques lógicos.

En este capítulo se presentarán dos métodos para minimizar funciones booleanas: minimización usando mapas de Karnaugh y minimización usando el método tabular.

14.2. MINIMIZACION DE FUNCIONES BOOLEANAS USANDO MAPAS DE KARNAUGH

Este método se basa en la aplicación sistemática de los siguientes teoremas:

$$\begin{aligned}AB + AB' &= A \\A + A &= A \\(A + B)(A + B') &= A \\A \cdot A &= A\end{aligned}$$

Los primeros dos teoremas se aplican para obtener la forma mínima, expresada como suma de productos y los últimos dos, se aplican para obtener la forma mínima expresada como producto de sumas.

Este método parte de una de las dos formas canónicas de la función para construir el mapa de Karnaugh. Una vez construido el mapa, se puede obtener la forma mínima expresada como suma de productos si se trabaja con los unos (minitérminos) de la función. La forma mínima expresada como producto de sumas se obtiene trabajando con los ceros (maxitérminos) de la función.

El método consiste en hacer una representación condensada de la tabla de verdad, y luego, visualmente se obtiene la forma mínima que se desea. A la representación condensada de la tabla de verdad es lo que se conoce como mapa de Karnaugh.

Antes de explicar el método, se definirá qué es un mapa de Karnaugh y cómo se obtiene.

14.2.1. Mapas de Karnaugh

Recuérdese que para una función de n variables, existe 2^n posibles combinaciones de valores de las variables, y que la tabla de verdad incluye todas y cada una de ellas.

Un mapa de Karnaugh es simplemente otra forma de representar la tabla de verdad y consiste básicamente en un cuadrado o rectángulo dividido en 2^n casillas, cada una de las cuales tiene asociada una combinación de la tabla de verdad y el valor de la función para dicha combinación. Es importante aclarar que dichas casillas que constituyen el mapa de Karnaugh no pueden estar en cualquier orden, sino que existe cierta regla para ordenar las casillas para poder aplicar sistemáticamente los teoremas $AB + AB' = A$ o $(A + B)(A + B') = A$, dependiendo de la forma mínima que se desee obtener.

Antes de definir la regla para asignar las combinaciones de la tabla de verdad a las casillas se definen una serie de conceptos:

Para una combinación dada de las n variables de una función, existen n combinaciones con las cuales se puede aplicar el teorema: $AB + AB' = A$. Estas combinaciones son aquellas en las que solo cambia el valor de una de las n variables.

En la representación del mapa de Karnaugh de una función de n variables, cada casilla debe de tener n casillas contiguas para aplicar visualmente el teorema anterior. Una casilla es contigua de otra, si y solo si, de una a otra cambia solamente una variable. Si se desea localizar en forma visual las n casillas contiguas de una casilla dada en un mapa de Karnaugh, se deben asignar las combinaciones de la tabla de verdad a las casillas del mapa de acuerdo a la siguiente regla:

- Al asignar una combinación a una casilla, todas sus casillas contiguas deben tener una combinación donde sólo cambie el valor de una variable.

Al seguir la regla anterior, las casillas contiguas de una casilla dada son aquellas que se encuentran hacia la derecha, hacia la izquierda, hacia arriba y hacia abajo de la casilla en cuestión, es decir, son contiguas las casillas que están en dirección horizontal y vertical solamente.

Para identificar fácilmente las n casillas contiguas en un mapa de Karnaugh, se debe de suponer que el lado derecho del cuadrado o rectángulo está unido al lado izquierdo, y el lado superior, al lado inferior.

A continuación se definen los mapas de Karnaugh para funciones de 2, 3, 4 y 5 variables.

El Mapa de Karnaugh para funciones de 2 variables independientes constará de cuatro casillas. A continuación se muestran dos formas de asignar las combinaciones de la tabla de verdad en el mapa de Karnaugh para $F(A,B)$.

		A	
		0	1
B	0	00	10
	1	01	11

		B	
		0	1
A	0	00	01
	1	10	11

De los mapas de Karnaugh anteriores se puede observar que cada una de las variables independientes toma el valor de uno en la mitad del mapa y de cero en la otra mitad. Si marcamos el mapa por regiones, denominando con el nombre de la variable la región donde la variable toma el valor de uno y con el nombre de la variable complementa la región donde la variable toma el valor de cero, las regiones en los mapas anteriores serían las siguientes:

		A'	A
B'			
	B		

		B'	B
A'			
	A		

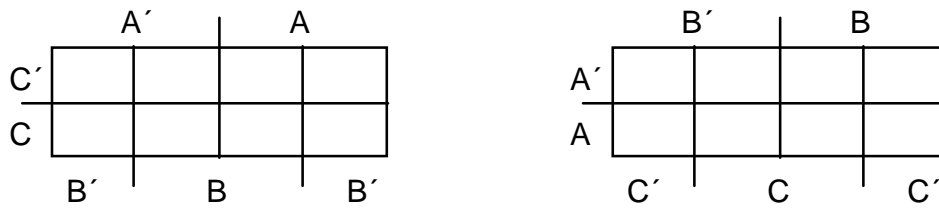
La combinación que se tiene asignada en cada casilla está dada por la intersección de las regiones que se tiene en la casilla. Ejemplo si una casilla está en la intersección de las regiones A' y B , la combinación de la tabla de verdad asignada a esa casilla es 01 ($A = 0$ y $B = 1$).

El Mapa de Karnaugh para funciones de 3 variables independientes constará de ocho casillas. A continuación se muestran dos formas de asignar las combinaciones de la tabla de verdad en el mapa de Karnaugh para $F(A,B,C)$.

		AB			
		00	01	11	10
C	0	000	010	110	100
	1	001	011	111	101

		BC			
		00	01	11	10
A	0	000	001	011	010
	1	100	101	111	110

Las regiones en los mapas anteriores serían las siguientes:



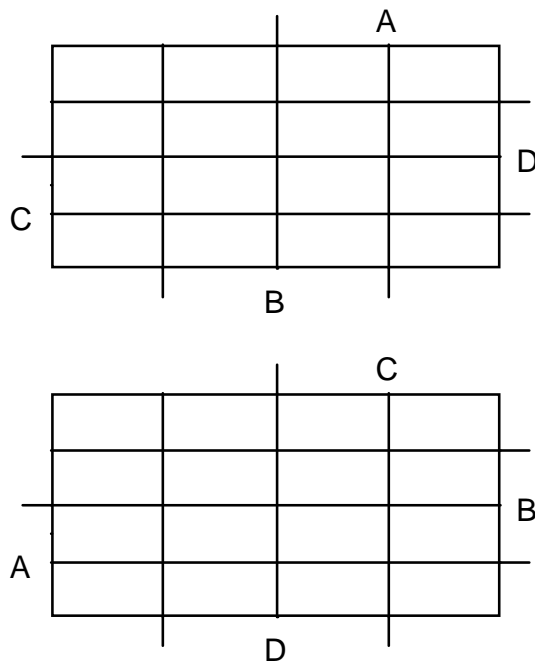
Recuerde que la combinación que se tiene asignada en cada casilla está dada por la intersección de las regiones en las que está la casilla. Por ejemplo, si una casilla está en la intersección de las regiones A' , B' y C la combinación de la tabla de verdad asignada a esa casilla es 001 ($A = 0$, $B = 0$ y $C = 1$).

El mapa de Karnaugh para funciones de 4 variables independientes constará de dieciséis casillas. A continuación se muestran dos posibles formas de asignar las combinaciones de la tabla de verdad en el mapa de Karnaugh para $F(A,B,C,D)$.

CD \ AB				
	00	01	11	10
00	0000	0100	1100	1000
01	0001	0101	1101	1001
11	0011	0111	1111	1011
10	0010	0110	1110	1010

AB \ CD				
	00	01	11	10
00	0000	0001	0011	0010
01	0100	0101	0111	0110
11	1100	1101	1111	1110
10	1000	1001	1011	1010

Las regiones en los mapas anteriores serían las siguientes :



Note que solo se indicaron las regiones de las variables y no el de las variables complementadas, es común usar este método ya que la región de la variable complementada se puede determinar fácilmente.

La combinación que se tiene asignada en cada casilla también está dada por la intersección de las regiones donde se encuentra la casilla. Por ejemplo, si una casilla está en la intersección de las regiones A' , B' , C y D , la combinación de la tabla de verdad asignada a esa casilla es 0011 ($A = 0$, $B = 0$, $C = 1$ y $D = 1$).

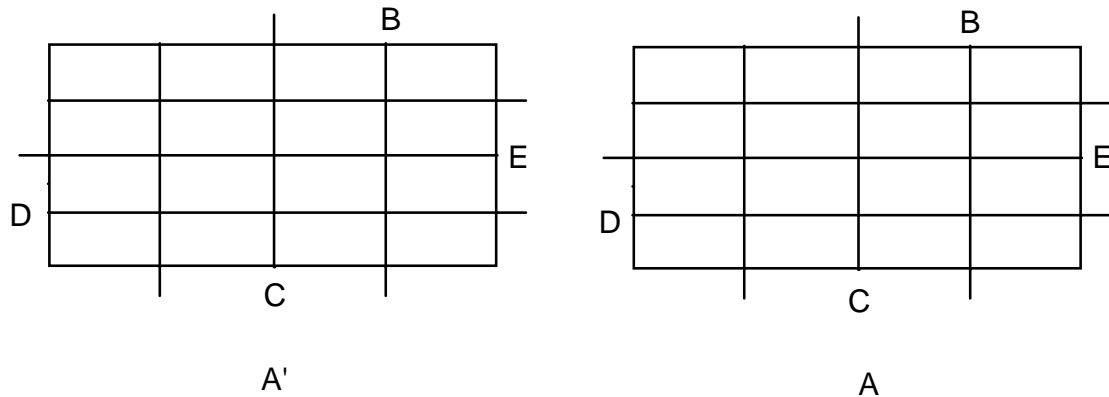
El Mapa de Karnaugh para funciones de 5 variables independientes constará de treinta y dos casillas. La construcción de este mapa se puede realizarse por medio de dos mapas de 4 variables. A continuación se muestra una forma de asignar las combinaciones de la tabla de verdad en el mapa de Karnaugh para $F(A,B,C,D,E)$.

DE \ BC	00	01	11	10	DE \ BC	00	01	11	10
00	00000	00100	01100	01000	00	10000	10100	11100	11000
01	00001	00101	01101	01001	01	10001	10101	11101	11001
11	00011	00111	01111	01011	11	10011	10111	11111	11011
10	00010	00110	01110	01010	10	10010	10110	11110	11010

$A = 0$
 $A = 1$

Nótese que para este caso, una casilla debe de tener cinco casillas contiguas. Las casillas contiguas a una casilla dada son las cuatro que se tienen en el mapa de cuatro variables y la

quinta es la casilla que ocupa la misma posición en el otro mapa. Debe suponerse que un mapa esta sobre el otro. Las regiones en los mapas anteriores serían las siguientes:



A continuación se ejemplificará la construcción del mapa de Karnaugh para cuatro funciones:

Considere la función booleana dada por:

$$F = AB' + A'$$

La tabla de verdad correspondiente a esta función es la siguiente:

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

Para representar la tabla de verdad (y por consiguiente la función) mediante el mapa de Karnaugh, en cada casilla se coloca el valor que toma la función para la combinación asignada a ésta.

		A
	1	1
B	1	0

Note que el mapa de Karnaugh tiene exactamente la misma información que la tabla de verdad, solamente que la información se presenta en una forma más condensada.

Considere ahora la siguiente función booleana:

$$F = AB'C + ABC' + B'C'$$

La tabla de verdad correspondiente a esta función es la siguiente:

A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0

A	B	C	F
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

El mapa de Karnaugh de la función se muestra a continuación:

				A
	1	0	1	1
C	0	0	0	1
				B

Considere ahora la siguiente función:

$$F = (W + X + Z') (X' + Y' + Z) (W' + Y')$$

W	X	Y	Z	F
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1

W	X	Y	Z	F
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

El correspondiente mapa de Karnaugh de esta función es el siguiente:

				W
	1	1	1	1
	0	1	1	1
Y	0	1	0	0
	1	0	0	0
				X

Por último, considere la función booleana:

$$F = A'BCD + CD'E' + B'DE + ABCDE + A'B'C$$

La tabla de verdad para esta función es:

A	B	C	D	E	F
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	1
0	0	1	0	0	1
0	0	1	0	1	1
0	0	1	1	0	1
0	0	1	1	1	1
0	1	0	0	0	0
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	1	1	0
0	1	1	0	0	1
0	1	1	0	1	0
0	1	1	1	0	1
0	1	1	1	1	1

A	B	C	D	E	F
1	0	0	0	0	0
1	0	0	0	1	0
1	0	0	1	0	0
1	0	0	1	1	1
1	0	1	0	0	1
1	0	1	0	1	0
1	0	1	1	0	0
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	0	1	0
1	1	0	1	0	0
1	1	0	1	1	0
1	1	1	0	0	1
1	1	1	0	1	0
1	1	1	1	0	0
1	1	1	1	1	1

El mapa de Karnaugh para esta función quedaría como sigue:

				B	
	0	1	1	0	
	0	1	0	0	
D	1	1	1	0	E
	0	1	1	0	
				C	
				A'	

				B	
	0	1	1	0	
	0	0	0	0	
D	1	1	1	0	E
	0	0	0	0	
				C	
				A	

Es conveniente notar que para construir el mapa de Karnaugh no es necesario construir la tabla de verdad, ya que el mapa puede ser construido directamente de la función como se indica a continuación.

Si la función está expresada en suma de productos, cada producto genera uno o varios unos en la casillas que se encuentran en la región dada por la intersección de las regiones de las variables del producto. A toda casilla que no tenga uno, se le asigna el valor de cero.

Si la función está expresada en producto de sumas, cada suma genera uno o varios ceros en la casillas que se encuentran en la región dada por la intersección de las regiones de las variables

complementadas de la suma. A toda casilla que no contenga un cero, se le asigna el valor de uno.

Si la función no está expresada en alguna de las formas anteriores, se recomienda primero construir la tabla de verdad.

14.2.2 Minimización como suma de productos.

Para obtener la forma mínima de una función booleana expresada como suma de productos usando mapas de Karnaugh se deben realizar agrupaciones de casillas que contienen unos. A continuación se ejemplificará el procedimiento para realizar la minimización.

Considere de nuevo la primera función booleana y su correspondiente mapa de Karnaugh.

$$F = AB' + A'$$

		A
	1	1
B	1	0

La forma canónica en minitérminos para esta función es la siguiente:

$$F = AB' + A'B + A'B'$$

Primero obsérvese que cada uno de los minitérminos de la forma canónica, corresponde a un uno en el mapa de Karnaugh.

Ahora, dado que la función es muy sencilla, se procederá a la minimización de la función usando teoremas, y se observará que estos teoremas se pueden aplicar visualmente en el mapa de Karnaugh.

$$F = AB' + A'B + A'B' = AB' + A'B' + A'B + A'B'$$

Hasta aquí solamente se repitió el término $A'B'$ usando el teorema que establece que $A + A = A$.

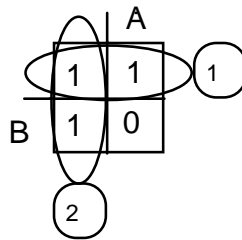
$$F = (A + A')B' + (B + B')A'$$

En este paso se aplicó el teorema de asociatividad.

Finalmente, usando los teoremas $A + A' = 1$ y $A \cdot 1 = A$ se obtiene la forma mínima expresada como suma de productos.

$$F = B' + A'$$

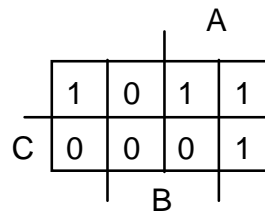
Ahora obsérvese el mapa y nótese que la primera columna contiene los términos $A'B'$ y $A'B$. Estos términos se simplifican para dar como resultado A' . Por otra parte, el primer renglón del mapa contiene los términos $A'B'$ y AB' , que al simplificarse dan como resultado B' . Por lo tanto, para este caso se puede concluir que dos términos que están en casillas contiguas en el mapa se pueden agrupar para simplificarse.



Nótese en el mapa las agrupaciones hechas, el grupo 1 está contenido en la región de B' y el grupo 2 está contenida en la región de A' , la función mínima expresada como suma de productos está definida por:

$$F = A' + B'$$

Considérese ahora la segunda función y su correspondiente mapa de Karnaugh.



$$F = AB'C + ABC' + B'C'$$

La forma canónica expresada en minitérminos es la siguiente:

$$F = AB'C + ABC' + AB'C' + A'B'C'$$

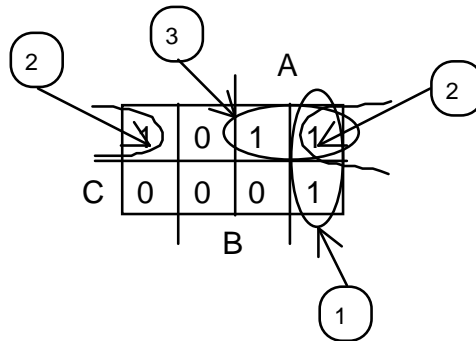
A continuación se procederá a simplificar la función usando teoremas. Los teoremas usados no se mencionarán explícitamente, ya que es fácil deducir qué teorema se ha aplicado en cada caso.

$$F = AB'C + AB'C' + ABC' + AB'C' + AB'C' + A'B'C'$$

$$F = AB'(C + C') + AC'(B + B') + B'C'(A + A')$$

$$F = AB' + AC' + B'C'$$

Ahora obsérvese la colocación en el mapa de los términos que se han simplificado.



Los términos $AB'C$ y $AB'C'$ se simplificaron para dar AB' , estos dos términos se agruparon en el grupo 1. Los términos $AB'C'$ y $A'B'C'$ se simplificaron para producir el término $B'C'$, estos dos términos corresponden al grupo 2. Finalmente, los términos ABC' y $AB'C'$ se simplificaron para generar el término AC' , los cuales corresponden al grupo 3.

Sigue siendo cierto para este caso que dos términos que están en casillas contiguas en el mapa se pueden agrupar para simplificarse. Recuerde que se debe de suponer que el lado derecho del mapa está unido al lado izquierdo y la parte superior a la inferior.

De todas las observaciones anteriores se puede deducir que para que dos términos se simplifiquen mediante la aplicación del teorema $AB + AB' = A$ es necesario que solo una variable cambie de un término a otro y, por consiguiente, para que la regla dada de agrupar en el mapa dos casillas que estén contiguas sea siempre aplicable es imprescindible que de una casilla a la contigua, solo cambie una variable. ¡He ahí el secreto de cómo deben ordenarse las casillas en el mapa de Karnaugh!

Volviendo al mapa donde se muestran las agrupaciones hechas, cada grupo será un término en la función mínima expresada como suma de productos. Obviamente, la función está definida por una operación OR de los tres términos, ya que debe valer uno cuando cualquiera de los tres términos valgan uno.

Considérese la tercer función y su mapa de Karnaugh.

				W
Y	1	1	1	1
	0	1	1	1
	0	1	0	0
	1	0	0	0
				X

$$F = (W + X + Z')(X' + Y' + Z)(W' + Y')$$

A continuación se muestra la función expresada en su forma canónica en minitérminos.

$$F = W'X'Y'Z' + W'X'YZ' + W'XY'Z' + W'XY'Z + W'XYZ + WX'Y'Z' + WX'YZ' + WXY'Z' + WXY'Z$$

Procediendo con la simplificación usando teoremas se tiene lo siguiente:

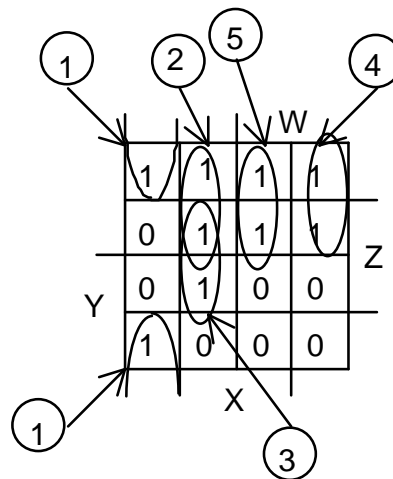
$$F = W'X'Y'Z' + W'X'YZ' + W'XY'Z' + W'XY'Z + W'XY'Z + W'XYZ + WX'Y'Z' + WX'YZ' + WXY'Z' + WXY'Z$$

Hasta ahora solamente se ha repetido un término y se han ordenado los minitérminos de una manera más cómoda para su simplificación.

$$F = W'X'Z'(Y' + Y) + W'XY'(Z' + Z) + W'XZ(Y' + Y) + WX'Y'(Z' + Z) + WXY'(Z' + Z)$$

$$F = \underset{1}{W'X'Z'} + \underset{2}{W'XY'} + \underset{3}{W'XZ} + \underset{4}{WX'Y'} + \underset{5}{WXY'}$$

Obsérvese lo que ocurrió en el mapa:



Hasta este punto todo va bien, la regla sigue funcionando. Se pueden agrupar casillas que sean contiguas; sin embargo, si se observa la función se verá que ésta aún puede ser simplificada.

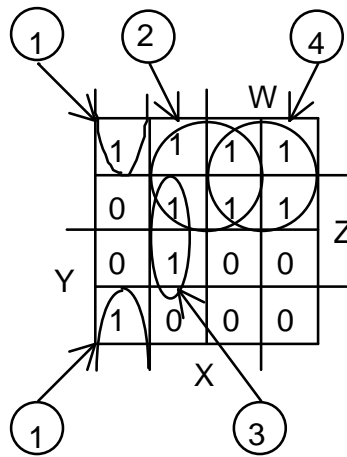
$$F = W'X'Z' + W'XY' + W'XZ + WX'Y' + WXY'$$

$$F = W'X'Z' + W'XY' + WXY' + W'XZ + WX'Y' + WXY'$$

$$F = W'X'Z' + XY'(W' + W) + W'XZ + WY'(X' + X)$$

$$F = W'X'Z' + XY' + W'XZ + WY'$$

Obsérvese ahora lo que ocurre en el mapa de Karnaugh. Se han reagrupado los grupos 2 y 5 y los grupos 4 y 5. Los grupos 1 y 3 no se pudieron reducir más. El mapa con la agrupación final y la función mínima se muestra a continuación.



$$F = \underset{1}{W'}\underset{2}{X'}\underset{3}{Z'} + \underset{4}{XY'} + \underset{5}{W'XZ} + WY'$$

Por lo tanto, también existen grupos de cuatro casillas (cuatro minitérminos), los cuales están definidos por dos variables en este caso.

Supóngase por un momento que se hubieran tenido los términos WY' y WY , estos dos términos podrían simplificarse para dar W simplemente. ¿Cuántas casillas se habrían agrupado en este caso? Es fácil deducir que, en este caso, se habrían agrupado las ocho casillas inferiores.

En general, se pueden agrupar 2^i casillas y cada grupo de 2^i casillas de un mapa de 2^n casillas estará definido con $(n-i)$ variables.

Para determinar si una agrupación es correcta se puede aplicar la siguiente regla:

- Al formar un grupo de 2^i casillas en un mapa de 2^n casillas, el grupo deberá estar contenido en la intersección de las regiones de $(n-i)$ variables, las cuales son las $(n-i)$ variables que lo definen. Dado que el objetivo es minimizar se debe de buscar el valor de i más grande posible.

Por último considérese la cuarta función y su mapa de Karnaugh.

$$F = A'B'CD + CD'E' + B'DE + ABCDE + A'B'C$$

14.2.3 Minimización como producto de sumas.

En la sección anterior se trabajó con los unos (minitérminos) de la función y se obtuvo la forma mínima expresada como suma de productos. Existe un proceso dual del anterior en el cual se toman los ceros (maxitérminos) de la función y se obtiene la forma mínima expresada como producto de sumas. Ambas formas mínimas son equivalentes. La selección de la forma más adecuada, depende del tipo de bloques lógicos que se desee utilizar; o bien, del número de bloques lógicos, que requiera cada forma, si no se ha decidido qué tipo de bloques utilizar (NANDs o NORs).

Este procedimiento se ilustrará mediante la simplificación de las funciones de tres, cuatro y cinco variables definidas anteriormente y el lector podrá comprobar que el procedimiento es el dual del descrito en la sección anterior.

Considere la función de tres variables y su correspondiente mapa de Karnaugh.

				A
	1	0	1	1
C	0	0	0	1
				B

$$F = AB'C + ABC' + B'C'$$

La forma canónica en maxitérminos para esta función es:

$$F = (A + B + C') (A + B' + C) (A + B' + C') (A' + B' + C')$$

Primero obsérvese que cada uno de los maxitérmino de la forma canónica, corresponde a un cero en el mapa de Karnaugh. Recuerde que los maxitérminos generan ceros en una sola combinación, por lo tanto el maxitérmino $(A + B' + C)$ genera cero cuando $A = 0$, $B = 1$ y $C = 0$, que es la intersección de las regiones A' , B y C' (es el cero de la parte superior del mapa de Karnaugh).

Ahora, se procederá a la minimización de la función usando teoremas.

Se repitará el termino $(A + B' + C')$ y se ordenarán los maxitérminos en una forma más conveniente (recuérdese que $X \cdot X = X$).

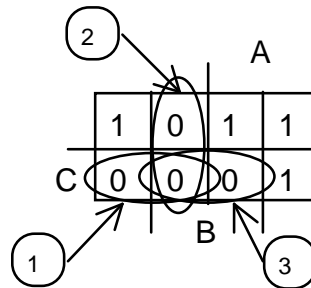
$$F = (A + B + C') (A + B' + C') (A + B' + C) (A + B' + C') \\ (A' + B' + C') (A + B' + C')$$

Se aplicara el teorema $(X + Y) (X + Y') = X$, a los pares de maxitérminos que resultan de separar en tres pares los seis maxitérminos ordenados.

$$F = (A + C') (A + B') (B' + C')$$

1 2 3

Ahora obsérvese la colocación en el mapa de cada uno de los términos.



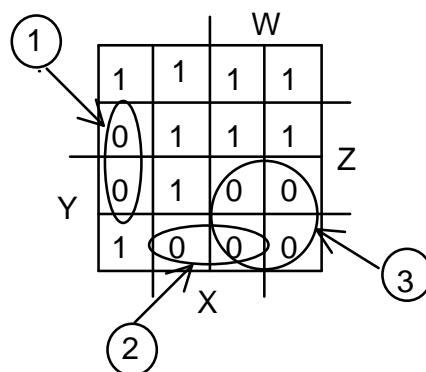
Como el lector podrá observar, ahora se trata de agrupar ceros siguiendo la misma regla de agrupación que se tenía. Cada grupo está definido por una operación OR de las variables y la función está definida por una operación AND de los términos que definen los grupos.

Considérese la función de cuatro variables y su mapa de Karnaugh.

		W			
		1	1	1	1
Y	Z	0	1	1	1
	0	0	1	0	0
	1	1	0	0	0
	X				

$$F = (W + X + Z') (X' + Y' + Z) (W' + Y')$$

A continuación se muestra el mapa de Karnaugh con las agrupaciones hechas.



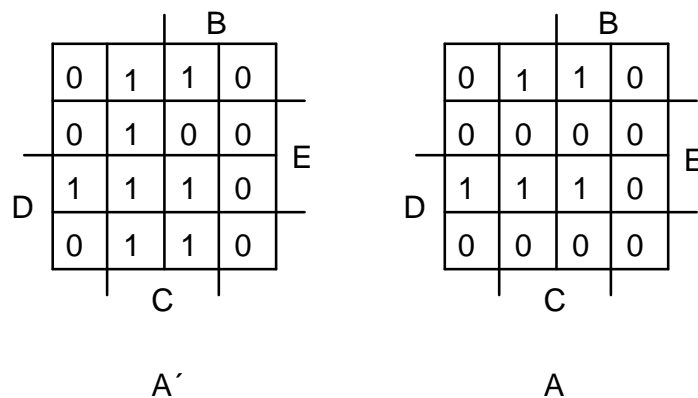
La función mínima es:

$$F = \underset{1}{(W + X + Z')} \underset{2}{(X' + Y' + Z)} \underset{3}{(W' + Y')}$$

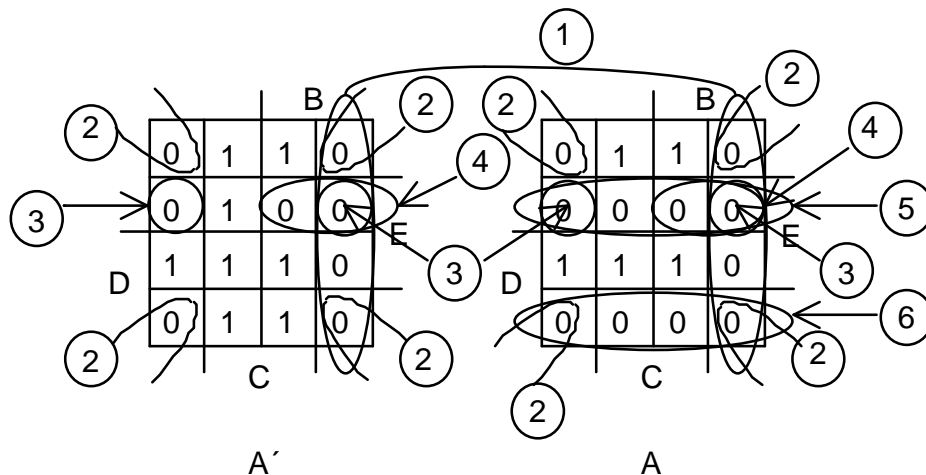
Esta función es igual a la función original, lo cual indica que ésta ya se encontraba expresada en su forma mínima como producto de sumas.

Por último, considérese la función de cinco variables y su mapa de Karnaugh.

$$F = A'BCD + CD'E' + B'DE + ABCDE + A'B'C$$



El mapa de Karnaugh con las agrupación final y la función mínima se muestran a continuación..



$$F = \underset{1}{(B' + C)} \underset{2}{(C + E)} \underset{3}{(C + D + E')} \underset{4}{(B' + D + E')} \underset{5}{(A' + D + E')} \underset{6}{(A' + D' + E)}$$

Como el lector puede comprobar, el proceso es el dual del anterior. Con estos ejemplos debe ser suficiente para comprender el método.

14.2.4 Minimización de funciones usando implicantes primos.

Dado que al minimizar funciones usando el método de mapas de Karnaugh se hacen las agrupaciones visualmente, algunas veces la función que se obtiene no es la que da la mínima expresión para la función.

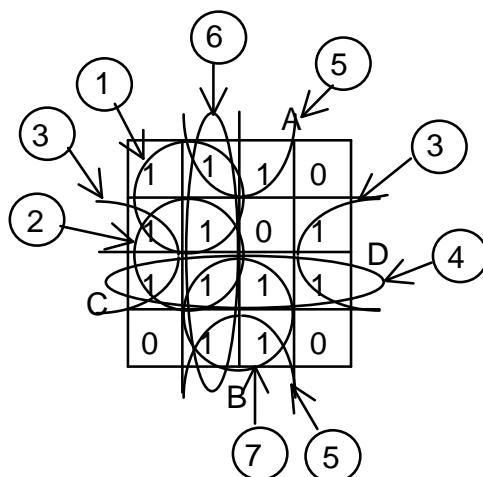
Un procedimiento que ayuda a realizar las agrupaciones correctas es la identificación de todos los grupos posibles que tiene la función, los cuales se denominan implicantes primos de la función. Al tener todos los implicantes primos se puede identificar cuales son los que deben estar en la función mínima.

Para encontrar todos los implicantes primos de una función, se recomienda analizar cada uno de los unos si se está minimizando como suma de productos; o cada uno de los ceros si se está minimizando como producto de sumas. Se deben encontrar todos los grupos que se puedan generar, de acuerdo con la regla de agrupación dada anteriormente.

Para ejemplificar este procedimiento, supóngase que se tiene el siguiente mapa de Karnaugh.

		A		
	1	1	1	0
	1	1	0	1
C	1	1	1	1
	0	1	1	0
	B			
				D

A continuación se muestran todas las agrupaciones posibles si se quiere obtener la forma mínima expresada como suma de productos.



Los implicantes primos se dividen en implicantes primos esenciales e implicantes primos no esenciales. Si un implicante primo contiene a un uno (cero) que sólo pertenece a él, es un implicante primo esencial; si no, es un implicante primo no esencial.

Se denominan implicantes primos esenciales porque deben aparecer en la expresión de la función mínima. Los implicantes primos no esenciales pueden aparecer en la función mínima o pueden no aparecer en ella.

En el ejemplo anterior, los implicantes primos 1, 3 y 5 son esenciales.

La función mínima esta compuesta por los implicantes primos esenciales y, si con ellos no se agrupan todos los unos (ceros) de la función, se debe añadir la mínima cantidad de implicantes primos no esenciales para incluir todos los unos (ceros) de la función.

En el ejemplo anterior, con los implicantes primos esenciales no se agrupan todos los unos, se debe de añadir el implicante primo no esencial 4 para incluir todos los unos de la función.

14.2.5 Minimización de funciones no especificadas completamente.

En el capítulo 12 se definieron los casos en que una función booleana no quedaba especificada completamente y se recomendó asignar el valor de x a la función cuando ésta estuviera indefinida.

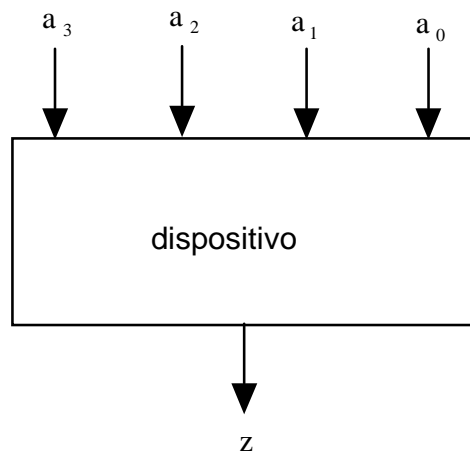
Se mencionó también que x podía tomar el valor de cero o de uno indistintamente. También se afirmó que si estos valores se seleccionaban adecuadamente, se podía obtener una función más simplificada que si simplemente se les asignaba el valor de cero.

Los valores más convenientes para cada una de estas combinaciones pueden ser determinadas de una manera sencilla utilizando mapas de Karnaugh.

El procedimiento que debe seguirse es el siguiente:

- Construir el mapa de Karnaugh dejando el valor de x para cada una de las combinaciones no definidas.
- Si se está minimizando como suma de productos se deberán realizar las agrupaciones de unos tomando los valores de x como unos siempre y cuando éstos contribuyan a formar grupos con un mayor número de unos. Todos los unos deberán quedar incluidos en un grupo al menos. Los valores de x no necesariamente deberán quedar incluidos en algún grupo.
- Si se está minimizando como productos de suma se asignará el valor de cero a las x que permitan realizar agrupaciones mayores de ceros.
- La función mínima estará definida por el mínimo número de grupos que incluyan todos los unos (ceros) del mapa.

Para ejemplificar este procedimiento se tomará el mismo caso que se uso en el capítulo 12. Se tratara de diseñar un dispositivo que reciba como entrada un número DCB e indique si dicho número es múltiplo de 3 o no. Se había tomado la convención de que la salida fuera uno cuando el número BCD fuera múltiplo de 3.



La tabla de verdad para este dispositivo es la siguiente:

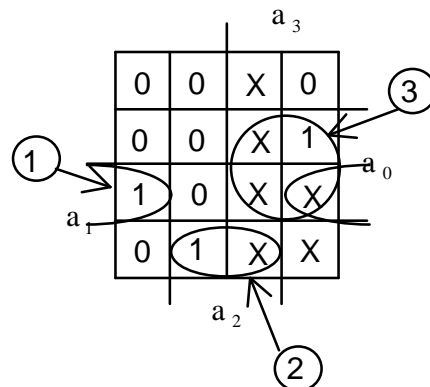
a_3	a_2	a_1	a_0	z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1

a_3	a_2	a_1	a_0	z
1	0	0	0	0
1	0	0	1	1
1	0	1	0	x
1	0	1	1	x

0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0

1	1	0	0	x
1	1	0	1	x
1	1	1	0	x
1	1	1	1	x

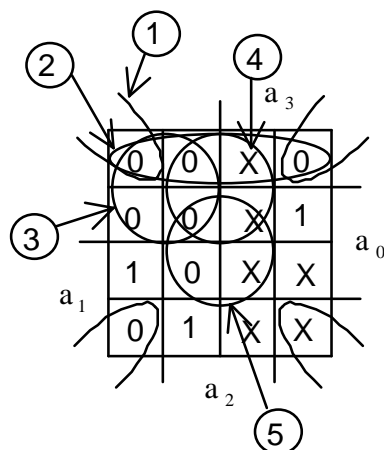
El mapa de Karnaugh, agrupando como suma de productos queda como sigue:



La función mínima es la siguiente:

$$z = \underset{1}{a_2'} \underset{2}{a_1} \underset{3}{a_0} + a_2 a_1 a_0' + a_3 a_0$$

El mapa de Karnaugh, agrupando como producto de sumas es el siguiente:



Se agruparon todos los implicantes primos, en los cuales se puede identificar que los implicantes primos 1, 3 y 5 son esenciales y con ellos se agrupan todos los ceros de la función. Por lo tanto, los implicantes primos no esenciales no aparecen en la expresión de la función mínima.

La función mínima es la siguiente:

$$z = (a_2 + a_0) (a_3 + a_1) (a_2' + a_0')$$

Nótese que ambas funciones generan exactamente la misma tabla de verdad para las combinaciones del cero al nueve, pero los valores asignados a las combinaciones del 10 al 15 son diferentes. Esto no debe considerarse como un error ya que el valor asignado a estas combinaciones es irrelevante.

14.3. MINIMIZACION DE FUNCIONES BOOLEANAS USANDO EL METODO TABULAR.

Este método tiene la ventaja de que puede ser programado en una computadora digital y puede aplicarse a funciones de n variables independientes. La complicación del método no se incrementa cuando el número de variables aumenta.

Este método de minimización, consiste en la aplicación sistemática de los teoremas:

$$\begin{aligned} A + A &= A \\ A \cdot B + A \cdot B' &= A \\ A + A \cdot B &= A \end{aligned}$$

cuando se desea obtener la forma mínima expresada como suma de productos.

Si se desea obtener la forma mínima expresada como productos de sumas se aplican los teoremas:

$$\begin{aligned} A \cdot A &= A \\ (A + B)(A + B') &= A \\ A \cdot (A + B) &= A \end{aligned}$$

Ambas variantes del método se presentarán usando un ejemplo en las secciones siguientes.

14.3.1. Minimización como suma de productos

Para obtener la forma mínima de una función expresada como suma de productos mediante este método es necesario expresar la función booleana en su forma canónica en minitérminos. Esto se puede hacer obteniendo la tabla de verdad de la función. Recuérdese que cada uno de la función corresponde a un minitérmino.

Una vez que se han obtenido todos los minitérminos de la función, se codifican en ceros y unos; por ejemplo, el minitérmino $ab'cd$ se codifica como 1001. Si se construyó la tabla de verdad, ya se tienen los minitérminos codificados.

Cuando se han obtenido todos los minitérminos codificados, se deben agrupar de acuerdo al número de unos que contenga la codificación de cada minitérmino.

Considérese por el momento la función booleana definida por la siguiente tabla de verdad:

a	b	c	d	e	f
0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	0
0	0	1	0	0	0
0	0	1	0	1	1
0	0	1	1	0	1
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	0	1	1
0	1	1	1	0	1
0	1	1	1	1	1

a	b	c	d	e	f
1	0	0	0	0	1
1	0	0	0	1	0
1	0	0	1	0	1
1	0	0	1	1	0
1	0	1	0	0	0
1	0	1	0	1	1
1	0	1	1	0	1
1	0	1	1	1	1
1	1	0	0	0	1
1	1	0	0	1	0
1	1	0	1	0	1
1	1	0	1	1	0
1	1	1	0	0	0
1	1	1	0	1	1
1	1	1	1	0	1
1	1	1	1	1	1

Dado que ya se tienen los minitérminos codificados, se procederá a agruparlos de acuerdo con el número de unos que contengan.

<u>0 0 0 0 0</u>	cero unos
0 1 0 0 0	
<u>1 0 0 0 0</u>	Un uno
0 0 1 0 1	
0 0 1 1 0	
1 0 0 1 0	
<u>1 1 0 0 0</u>	dos unos
0 0 1 1 1	
0 1 1 0 1	
0 1 1 1 0	
1 0 1 0 1	
1 0 1 1 0	
<u>1 1 0 1 0</u>	tres unos
0 1 1 1 1	
1 0 1 1 1	
1 1 1 0 1	
<u>1 1 1 1 0</u>	cuatro unos
<u>1 1 1 1 1</u>	cinco unos

Ahora se debe proceder a la aplicación sistemática del teorema:

$$A \cdot B + A \cdot B' = A.$$

Nótese que para que dicho teorema se pueda aplicar a dos minitérminos, es necesario que solamente cambie una variable de uno a otro. Como ejemplo considérese los minitérminos 00000 y 01000.

Dichos minitérminos se pueden simplificar como sigue:

$$a'b'c'd'e' + a'bc'd'e' = a'c'd'e'$$

El resultado de la simplificación se puede representar codificado como:

$$00000 + 01000 = 0X000$$

La X en el segundo lugar, implica que la variable se simplificó y, por lo tanto, su valor no importa.

Ahora se podrá ver la razón por la cuál se agruparon los minitérminos de acuerdo al número de unos que contenían. Los minitérminos con un uno se podrán simplificar con el que tiene cero unos, o bien con los que tienen dos unos, pero no con los que tienen tres unos, dado que de uno a otro cambiarían dos variables cuando menos.

El procedimiento a seguir para la simplificación consiste en tomar el minitérmino con cero unos (si es que existe como en el caso presente) y tratar de simplificarlo con todos y cada uno de los minitérminos con un uno, anotando el resultado de la simplificación en una columna a la

derecha. Dicho resultado deberá estar codificado con ceros y una X. Cada minitermino que se haya simplificado deberá marcarse (✓).

El minitérmino con cero unos, no podrá simplificarse con los minitérminos que contengan más de un uno, dado que cambian dos o más variables, por lo tanto dicho minitérmino ya se simplificó con todos los que se podía al terminar esta etapa.

Se procederá después a simplificar cada minitérmino de los del grupo de un uno con todos los que sea posible de los del grupo con dos unos. El resultado de las simplificaciones se anotará en la columna a la derecha y se marcará cada minitérmino que se haya podido simplificar (✓), excepto cuando ya estuviera marcado. Dicho resultado deberá estar codificado con ceros, un uno y una X. Al terminar esta etapa, todos los minitérminos con un uno ya se simplificaron con todos los que se podía, ya que se simplificó con todos los miniterminos posibles con cero unos y con dos unos.

Se procede después a simplificar cada minitérmino del grupo de dos unos, con todos aquellos que se pueda simplificar con los del grupo de tres unos, haciendo lo mismo que en el caso anterior.

Este procedimiento se sigue para cada uno de los grupos hasta agotar la tabla de minitérminos.

Al final del proceso se tendrá una nueva lista de términos que contienen una X (están definidos por 4 variables en el caso presente). Si algún minitérmino de la lista original no se pudo simplificar no deberá estar marcado (✓), en este caso, dicho minitérmino se marcará con un asterisco (*).

A continuación se aplicará el proceso a los minitérminos de la función en cuestión.

Miniterminos

0 0 0 0 0✓
0 1 0 0 0✓
1 0 0 0 0✓
0 0 1 0 1✓
0 0 1 1 0✓
1 0 0 1 0✓
1 1 0 0 0✓
0 0 1 1 1✓
0 1 1 0 1✓
0 1 1 1 0✓
1 0 1 0 1✓
1 0 1 1 0✓
1 1 0 1 0✓
0 1 1 1 1✓
1 0 1 1 1✓
1 1 1 0 1✓
1 1 1 1 0✓
1 1 1 1 1✓

Términos simplificados

0 X 0 0 0
X 0 0 0 0
 X 1 0 0 0
 1 0 0 X 0
1 X 0 0 0
 0 0 1 X 1
 0 X 1 0 1
 X 0 1 0 1
 0 0 1 1 X
 0 X 1 1 0
 X 0 1 1 0
 1 0 X 1 0
 1 X 0 1 0
1 1 0 X 0
 0 X 1 1 1
 X 0 1 1 1
 0 1 1 X 1
 X 1 1 0 1
 0 1 1 1 X
 X 1 1 1 0
 1 0 1 X 1
 1 X 1 0 1
 1 0 1 1 X
 1 X 1 1 0
1 1 X 1 0
 X 1 1 1 1
 1 X 1 1 1
 1 1 1 X 1
1 1 1 1 X

Para este caso, todos los miniterminos quedaron marcados (✓).

considérese por un momento los siguientes dos términos que resultaron de la simplificación anterior:

$$0X000 \text{ y } 1X000$$

Nótese que dichos términos todavía pueden simplificarse:

$$a'c'd'e' + ac'd'e' = c'd'e'$$

o bien:

$$0X000 + 1X000 = XX000$$

por lo tanto, los términos resultantes de esta primera simplificación todavía pueden ser simplificados con la condición de que la X esté en el mismo lugar y de que de un término al otro solamente cambie el valor de una variable.

Se debe proceder de igual manera a tratar de simplificar los términos resultantes de la primera simplificación. Se realiza el mismo procedimiento ya que la tabla que se obtuvo está ordenada de acuerdo al número de unos que contiene cada término.

Al hacer la segunda simplificación puede ser que los nuevos términos, estén repetidos como sucede con las siguientes simplificaciones:

$$\begin{aligned} 0X000 + 1X000 &= XX000 \\ X0000 + X1000 &= XX000 \end{aligned}$$

Si en la nueva simplificación resultaren términos repetidos, se deberán desechar todos menos uno. Recuerdese que:

$$A + A = A$$

Los nuevos términos obtenidos mediante la simplificación se vuelven a simplificar dando como resultado una nueva tabla (ahora con 3X cada término) y así sucesivamente hasta que ya no sea posible ninguna simplificación posterior.

Al final, los términos que no se hayan podido simplificar estarán marcados con un asterisco. Estos términos reciben el nombre de implicantes primos y corresponden a los obtenidos con las agrupaciones hechas en el mapa de Karnaugh.

A continuación se muestra el proceso completo de simplificación:

Minitérminos	primera	segunda	
	<u>simplificación</u>	<u>simplificación</u>	
<u>0 0 0 0 0</u> ✓	0 X 0 0 0✓	<u>X X 0 0 0</u>	<u>X X 0 0 0</u> *
0 1 0 0 0✓	<u>X 0 0 0 0</u> ✓	<u>1 X 0 X 0</u>	<u>1 X 0 X 0</u> *
<u>1 0 0 0 0</u> ✓	X 1 0 0 0✓	0 X 1 X 1	0 X 1 X 1✓
0 0 1 0 1✓	1 0 0 X 0✓	X 0 1 X 1	X 0 1 X 1✓
0 0 1 1 0✓	<u>1 X 0 0 0</u> ✓	X X 1 0 1	X X 1 0 1✓
1 0 0 1 0✓	0 0 1 X 1✓	0 X 1 1 X	0 X 1 1 X✓
<u>1 1 0 0 0</u> ✓	0 X 1 0 1✓	X 0 1 1 X	X 0 1 1 X✓
0 0 1 1 1✓	X 0 1 0 1✓	X X 1 1 0	X X 1 1 0✓
0 1 1 0 1✓	0 0 1 1 X✓	<u>1 X X 1 0</u>	<u>1 X X 1 0</u> *
0 1 1 1 0✓	0 X 1 1 0✓	X X 1 1 1	X X 1 1 1✓
1 0 1 0 1✓	X 0 1 1 0✓	X 1 1 X 1	X 1 1 X 1✓
1 0 1 1 0✓	1 0 X 1 0✓	X 1 1 1 X	X 1 1 1 X✓
<u>1 1 0 1 0</u> ✓	1 X 0 1 0✓	1 X 1 X 1	1 X 1 X 1✓
0 1 1 1 1✓	<u>1 1 0 X 0</u> ✓	<u>1 X 1 1 X</u>	<u>1 X 1 1 X</u> ✓
1 0 1 1 1✓	0 X 1 1 1✓	X 0 1 1 X	
1 1 1 0 1✓	X 0 1 1 1✓	X X 1 1 0	
<u>1 1 1 1 0</u> ✓	0 1 1 X 1✓	1 X X 1 0	
<u>1 1 1 1 1</u> ✓	X 1 1 0 1✓	<u>1 X X 1 0</u>	
	0 1 1 1 X✓	X X 1 1 1	
	X 1 1 1 0✓	X X 1 1 1	
	1 0 1 X 1✓	X 1 1 X 1	
	1 X 1 0 1✓	X 1 1 X 1	
	1 0 1 1 X✓	X 1 1 1 X	
	1 X 1 1 0✓	X 1 1 1 X	
	<u>1 1 X 1 0</u> ✓	1 X 1 X 1	
	X 1 1 1 1✓	1 X 1 X 1	
	1 X 1 1 1✓	1 X 1 1 X	
	1 1 1 X 1✓	<u>1 X 1 1 X</u>	
	<u>1 1 1 1 X</u> ✓		
tercera		implicantes	
<u>simplificación</u>		<u>primos</u>	
X X 1 X 1	X X 1 X 1 *	X X 0 0 0	
X X 1 X 1	<u>X X 1 1 X</u> *	1 X 0 X 0	
X X 1 X 1		1 X X 1 0	
X X 1 1 X		X X 1 X 1	
X X 1 1 X		X X 1 1 X	
<u>X X 1 1 X</u>			

Una vez que se han determinado los implicantes primos de la función, se procede a investigar cuáles de ellos son implicantes primos esenciales. Para esto se construye una tabla, donde se tendrá un renglón para cada implicante primo y una columna para cada minitérmino de la función.

		Minitérminos																	
		00000	00101	00110	00111	01000	01101	01110	01111	10000	10010	10101	10110	10111	11000	11010	11101	11110	11111
Implicantes primos	* XX000	✓				✓				✓					✓				
	1X0X0								✓	✓					✓				
	1XX10										✓	✓				✓			
*	XX1X1		✓		✓		✓				✓		✓				✓		✓
	XX11X			✓	✓			✓				✓	✓				✓		✓
		↑	↑	↑	↑	↑	↑	↑			↑						↑		

Figura 14.1 Tabla inicial

Dentro de la tabla se marca cuáles minitérminos están cubiertos por cada implicante primo (teorema $A + A \cdot B = A$). La figura 14.1 muestra esta tabla.

Cuando ya se tiene la tabla anterior, se procede a determinar cuales implicantes primos son esenciales. Para esto se analizan las columnas de la tabla, las cuales corresponden a los minitérminos.

Si un minitérmino está cubierto solamente por un implicante primo, dicho implicante primo es esencial, ya que si este implicante primo no se incluyera en la función, ningún otro implicante primo cubriría ese minitérmino.

En la misma tabla se muestran los minitérminos que están cubiertos por un solo implicante primo y se muestran los implicantes primos esenciales marcados con un asterisco (*).

Cuando se han determinado todos los implicantes primos esenciales, se eliminan de la tabla todos los minitérminos que están cubiertos por los implicantes primos esenciales, como se muestra en la tabla de la figura 14.2.

Si todavía quedan minitérminos que no están cubiertos por los implicantes primos esenciales, se procede a formar una tabla reducida. En esta tabla se incluyen solamente los implicantes primos no esenciales y los minitérminos que no están cubiertos por los implicantes primos esenciales.

Implicantes primos	Minitérminos																	
	00000	00101	00110	00111	01000	01101	01110	01111	10000	10010	10101	10110	10111	11000	11010	11101	11110	11111
* XX000	X				X				X					X				
1X0X0					X				X					X				
1XX10										X					X			
* XX1X1											X		X			X		X
* XX11X												X	X			X	X	X

Figura 14.2 Tabla inicial con simplificaciones

A continuación se muestra la tabla reducida para el ejemplo en cuestión.

Implicantes primos no esenciales	Minitérminos	
	10010	11010
1X0X0	✓	✓
1XX10	✓	✓

Por último, se tratará de cubrir los minitérminos de la tabla reducida con el menor número de implicates primos.

En este caso ambos implicantes primos cubren los minitérminos restantes y además, ambos tienen el mismo número de variables (1X0X0 y 1XX10). Por lo tanto, se puede tomar cualquiera de ellos.

La función mínima estará definida por los implicantes primos esenciales más los implicantes primos seleccionados de la tabla reducida.

Para este ejemplo, la función mínima estará definida por los siguientes implicantes primos:

- Implicantes primos esenciales : XX000, XX1X1 y XX11X.
- Implicante primo no esencial : 1X0X0 (se escogió éste).

Y queda expresada de la siguiente forma:

$$f = c'd'e' + ce + cd + ac'e'$$

Notése que para este caso la función dada por:

$$f = c'd'e' + ce + cd + ade'$$

es equivalente, ya que de la tabla reducida se pudo haber escogido el implicante primo 1XX10 de la misma forma. Esta función en particular tiene dos formas mínimas equivalentes.

A continuación se resolverá otro ejemplo mediante este método.

Obtenga la forma mínima expresada como suma de productos de la función definida por la siguiente tabla de verdad:

a	b	c	d	e	f
---	---	---	---	---	---

a	b	c	d	e	f
---	---	---	---	---	---

0	0	0	0	0	1
0	0	0	0	1	1
0	0	0	1	0	1
0	0	0	1	1	0
0	0	1	0	0	1
0	0	1	0	1	1
0	0	1	1	0	1
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	0	1	1
0	1	0	1	0	1
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	0	1	1
0	1	1	1	0	0
0	1	1	1	1	1

1	0	0	0	0	0
1	0	0	0	1	0
1	0	0	1	0	0
1	0	0	1	1	0
1	0	1	0	0	1
1	0	1	0	1	1
1	0	1	1	0	1
1	0	1	1	1	1
1	1	0	0	0	1
1	1	0	0	1	1
1	1	0	1	0	1
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	0	1	1
1	1	1	1	0	0
1	1	1	1	1	1

A continuación se muestra el proceso completo de simplificación:

Minitérminos	primera	segunda	
	<u>simplificación</u>	<u>simplificación</u>	
<u>00000</u> ✓	0000X✓	00X0X	00X0X*
00001✓	000X0✓	0X00X	0X00X*
00010✓	00X00✓	00XX0	00XX0*
00100✓	<u>0X000</u> ✓	0X0X0	<u>0X0X0</u> *
<u>01000</u> ✓	00X01✓	00X0X	0XX01*
00101✓	0X001✓	00XX0	001XX✓
00110✓	00X10✓	0X00X	X010X✓
01001✓	0X010✓	<u>0X0X0</u>	X01X0✓
01010✓	0010X✓	0XX01	X100X*
10100✓	001X0✓	0XX01	<u>X10X0</u> *
<u>11000</u> ✓	X0100✓	001XX	<u>0X1X1</u> ✓
00111✓	0100X✓	X010X	X01X1✓
01101✓	010X0✓	001XX	XX101✓
10101✓	<u>X1000</u> ✓	X01X0	X011X✓
10110✓	001X1✓	X010X	X1X01*
11001✓	0X101✓	X01X0	101XX✓
<u>11010</u> ✓	X0101✓	X100X	<u>110XX</u> *
01111✓	0011X✓	X10X0	XX111✓
10111✓	X0110✓	X100X	X11X1✓
11011✓	01X01✓	<u>X10X0</u>	1X1X1✓
<u>11101</u> ✓	X1001✓	0X1X1	<u>11XX1</u> *
11111✓	X1010✓	X01X1	
	1010X✓	0X1X1	
	101X0✓	XX101	
	1100X✓	X01X1	
	<u>110X0</u> ✓	XX101	
	0X111✓	X011X	
	X0111✓	X011X	
	011X1✓	X1X01	
	X1101✓	X1X01	
	101X1✓	101XX	
	1X101✓	101XX	
	1011X✓	110XX	
	110X1✓	<u>110XX</u>	
	11X01✓	XX111	
	<u>1101X</u> ✓	XX111	
	X1111✓	X11X1	
	1X111✓	X11X1	
	11X11✓	1X1X1	
	<u>111X1</u> ✓	1X1X1	
		11XX1	
		11XX1	

tercera
simplificación

implicantes
primos

X 0 1 X X	<u>X 0 1 X X</u> *	0 0 X 0 X
X 0 1 X X	<u>X X 1 X 1</u> *	0 X 0 0 X
<u>X 0 1 X X</u>		0 0 X X 0
X X 1 X 1		0 X 0 X 0
X X 1 X 1		0 X X 0 1
<u>X X 1 X 1</u>		X 1 0 0 X
		X 1 0 X 0
		X 1 X 0 1
		1 1 0 X X
		1 1 X X 1
		X 0 1 X X
		X X 1 X 1

En la figura 14.3 se muestra la tabla inicial y además se marcan (*) los implicantes primos esenciales.

Minitérminos	1 1 1 1 1										✓		✓
	1 1 1 0 1								✓		✓		✓
	1 1 0 1 1									✓	✓		
	1 1 0 1 0							✓			✓		
	1 1 0 0 1						✓		✓	✓	✓		
	1 1 0 0 0						✓	✓		✓			
	1 0 1 1 1											✓	✓
	1 0 1 1 0											✓	
	1 0 1 0 1											✓	✓
	1 0 1 0 0											✓	
	0 1 1 1 1												✓
	0 1 1 0 1					✓			✓				✓
	0 1 0 1 0				✓			✓					
	0 1 0 0 1		✓			✓	✓		✓				
	0 1 0 0 0		✓		✓		✓	✓					
	0 0 1 1 1											✓	✓
	0 0 1 1 0			✓								✓	
	0 0 1 0 1	✓				✓						✓	✓
	0 0 1 0 0	✓		✓								✓	
	0 0 0 1 0			✓	✓								
	0 0 0 0 1	✓	✓			✓							
	0 0 0 0 0	✓	✓	✓	✓								
abcde													
Implicantes primos	00X0X												
	0X00X												
	00XX0												
	0X0X0												
	0XX01												
	X100X												
	X10X0												
	X1X01												
	110XX												
	11XX1												
	* X01XX												
	* XX1X1												

Figura 14.3 Tabla inicial

Al quitar los implicantes primos esenciales y los minitérminos que están cubiertos por los mismos, se obtiene la siguiente tabla reducida:

Tabala reducida

Implicantes primos no esenciales	abcde /	Minitérminos									
		00000	00001	00010	01000	01001	01010	11000	11001	11010	11011
00X0X		✓	✓								
0X00X		✓	✓		✓	✓					
00XX0		✓		✓							
0X0X0		✓		✓	✓		✓				
0XX01			✓			✓					
X100X					✓	✓			✓		
X10X0					✓		✓	✓		✓	
X1X01						✓			✓		
110XX								✓	✓	✓	✓
11XX1									✓		✓

Ahora se tratará de cubrir todos los minitérminos de la tabla reducida con el menor número de implicantes primos no esenciales. Este paso se puede simplificar si se aplican las siguientes reducciones a la tabla reducida:

- Un renglón **i** se puede eliminar de la tabla si existe un renglón **j** que contiene una marca (✓) en cada columna donde el renglón **i** contiene marcas (✓). Si existen varios renglones iguales se eliminan todos menos uno, se debe dejar el renglón del implicante primo que contenga el mayor número de X, ya que éste representa un término con menos variables. Esta reducción puede realizarse ya que al tomar el implicante primo que se deja, se está agrupando a todos los minitérminos que cubren los implicantes primos que se eliminaron.
- Un columna **i** se puede eliminar de la tabla si existe una columna **j** que en todos los renglones donde esta columna tiene marcas (✓), la columna **i** también las tiene (✓). Si existen varias columnas iguales, se eliminan todas menos una. La justificación de esta reducción es que al agrupar el minitérmino de la columna que se deja, el

implicante primo que lo cubre también agruparía a todos los minitérminos que se eliminan de la tabla.

Las reducciones que se pueden aplicar en la tabla reducida serían las siguientes:

- El renglón del implicante primo 0X00X elimina a los renglones de los implicantes primos 00X0X Y 0XX01.
- El renglón del implicante primo 0X0X0 elimina al renglón del implicante primo 00XX0.
- El renglón del implicante primo X100X elimina al renglón del implicante primo X1X01.
- El renglón del implicante primo 110XX elimina al renglón del implicante primo 11XX1.
- La columna del minitérmino 00010 elimina a la columna del minitermino 00000.
- La columna del minitérmino 01010 elimina a la columna del minitermino 01000.
- La columna del minitérmino 11010 elimina a la columna del minitermino 11000.
- La columna del minitérmino 11011 elimina a la columna del minitermino 11001.

A continuación se muestra la tabla resultante.

Implicantes primos no esenciales	Minitérminos						
	abcde	00001	00010	01001	01010	11010	11011
0X00X		✓		✓			
0X0X0			✓		✓		
X100X				✓			
X10X0					✓	✓	
110XX						✓	✓

En esta tabla se puede ver que los implicantes primos 0X00X, 0X0X0 y 110XX deben aparecer en la expresión de la función mínima, ya que hay minitérminos que solamente son agrupados por

alguno de ellos. Si se eliminan estos implicantes primos no esenciales y los minitérminos que cubre, se puede ver que ya no quedan minitérminos por agrupar.

La expresión de la función mínima está formada por los implicantes primos esenciales (X01XX y XX1X1) más los implicantes primos no esenciales (0X00X, 0X0X0 y 110XX).

$$F = b'c + ce + a'c'd' + a'c'e' + abc'$$

El procedimiento para obtener la función mínima expresada como suma de productos se puede resumir en los siguientes pasos:

1. Expresar la función que se desea minimizar en su forma canónica en minitérminos y codificar cada minitérmino con ceros y unos.
2. Agrupar los minitérminos de acuerdo al número de unos que se tengan.
3. Encontrar los implicantes primos de la función aplicando los teoremas $A \cdot B + A \cdot B' = A$ y $A + A = A$, todas las veces que sea posible.
4. Determinar los implicantes primos esenciales de la función. Para obtenerlos se construye una tabla donde se tiene un renglón para cada implicante primo y una columna para cada minitérmino de la función. Dentro de la tabla se marcan cuáles minitérminos están cubiertos por cada implicante primo. Si un minitérmino está cubierto por un sólo implicante primo, dicho implicante primo es esencial.
5. Si los implicantes primos esenciales no cubren todos los minitérminos de la función, se procede a encontrar la mínima cantidad de implicantes primos no esenciales que deben aparecer en la forma mínima para incluir todos los minitérminos. Para cubrir los minitérminos que faltan con la menor cantidad de implicantes primos no esenciales, se repite el siguiente paso tantas veces como sea necesario.
6. Se construye una tabla reducida donde se eliminan los implicantes primos que ya se tomaron y los minitérminos agrupados por ellos, y se aplican las siguientes reducciones:
 - a. Un renglón **i** se puede eliminar de la tabla, si existe un renglón **j** que contiene una marca (✓) en cada columna donde el renglón **i** contiene marcas (✓). Si existen varios renglones iguales se eliminan todos menos uno, se debe de dejar el renglón del implicante primo que contenga más X.
 - b. Un columna **i** se puede eliminar de la tabla, si existe una columna **j** que en todos los renglones donde contiene marcas (✓), la columna **i** también contiene marcas (✓). Si existen varias columnas iguales se eliminan todas menos una.
 - c. Si se eliminó un renglón o una columna, se construye una nueva tabla y se identifican los implicantes primos no esenciales que deben de ir en la forma mínima de la misma

manera en que se identificaron a los implicantes primos esenciales. Si no se puede identificar algún implicante primo se toma en forma arbitraria cualquiera de ellos (se recomienda seleccionar aquel que agrupe la mayor cantidad de minitérminos, si hay empate se selecciona el que tenga la mayor cantidad de X).

14.3.2. Minimización como producto de sumas.

Cuando se explicó la simplificación usando mapas de Karnaugh se obtuvieron formas mínimas de funciones booleanas expresadas como suma de productos, agrupando los unos (minitérminos) en el mapa de Karnaugh. También se obtuvieron formas mínimas expresadas como producto de sumas agrupando los ceros (maxitérminos) de la función.

Con el método tabular también se es posible obtener la forma mínima expresada como producto de sumas si en lugar de tomar los miniterminos de la función se toman los maxitérminos de la misma. Los teoremas que se aplican en este caso son los siguientes:

$$\begin{aligned}A \cdot A &= A \\(A + B) \cdot (A + B') &= A \\A + (A + B) &= A\end{aligned}$$

El resto del procedimiento es exactamente igual. Para aclararlo, se obtendrá la forma mínima de la función booleana que se usó en el segundo ejemplo de la sección anterior, expresada como productos de sumas.

A continuación se muestran las simplificaciones:

Maxitérminos	primera	segunda	
	<u>simplificación</u>	<u>simplificación</u>	
<u>1 0 0 0 0</u> ✓	<u>1 0 0 0 X</u> ✓	<u>1 0 0 X X</u>	<u>1 0 0 X X</u> *
<u>0 0 0 1 1</u> ✓	<u>1 0 0 X 0</u> ✓	<u>1 0 0 X X</u>	<u>X 1 1 X 0</u> *
<u>0 1 1 0 0</u> ✓	<u>0 X 0 1 1</u> *	<u>X 1 1 X 0</u>	
<u>1 0 0 0 1</u> ✓	<u>X 0 0 1 1</u> *	<u>X 1 1 X 0</u>	
<u>1 0 0 1 0</u> ✓	<u>0 1 1 X 0</u> ✓		
<u>0 1 0 1 1</u> ✓	<u>X 1 1 0 0</u> ✓		
<u>0 1 1 1 0</u> ✓	<u>1 0 0 X 1</u> ✓		
<u>1 0 0 1 1</u> ✓	<u>1 0 0 1 X</u> ✓		
<u>1 1 1 0 0</u> ✓	<u>X 1 1 1 0</u> ✓		
<u>1 1 1 1 0</u> ✓	<u>1 1 1 X 0</u> ✓		

A continuación se muestra la tabla inicial, donde se marcan (*) los implicantes primos esenciales.

Tabla inicial

Implicantes primos	abcde	Maxitérminos									
		00011	01011	01100	01110	10000	10001	10010	10011	11100	11110
* 0X011		✓	✓								
X0011		✓							✓		
* 100XX						✓	✓	✓	✓		
* X11X0				✓	✓					✓	✓

En este caso, los tres implicantes primos esenciales (0X011, 100XX y X11X0) cubren todos los maxiterminos. Para este ejemplo, no existe tabla reducida.

La función mínima expresada como productos de sumas queda como sigue:

$$f = (a + c + d' + e') (a' + b + c) (b' + c' + e)$$

14.3.3. Minimización de funciones no especificadas completamente.

El procedimiento para aplicar el método tabular a las funciones no especificadas completamente presenta las siguientes variantes:

- Considerar los valores de **x** como unos (ceros) al realizar la simplificación de los minitérminos (maxitérminos) para obtener los implicantes primos.
- Al formar la tabla con los minitérminos (maxitérminos) y los implicantes primos, no deberán incluirse las combinaciones para las cuales la función no está definida (**x**). Tampoco se deben incluir en la tabla reducida.

Por lo que concierne al resto del procedimiento, no presenta ninguna variación.

Esta modificación al procedimiento se aplicará para reducir la función del ejemplo que se utilizó en el capítulo 12 para funciones no especificadas completamente.

En este ejemplo se trata de diseñar un dispositivo que recibe como entrada un número DCB e indica en la salida si dicho número es múltiplo de 3 o no. Se había tomado la convención de que la salida fuera uno cuando el número BCD fuera múltiplo de 3.

La tabla de verdad para este dispositivo es la siguiente:

a ₃	a ₂	a ₁	a ₀	z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0

a ₃	a ₂	a ₁	a ₀	z
1	0	0	0	0
1	0	0	1	1
1	0	1	0	x
1	0	1	1	x
1	1	0	0	x
1	1	0	1	x
1	1	1	0	x
1	1	1	1	x

Minimización como suma de productos.

A continuación se muestran las simplificaciones:

Minitérminos	primera	segunda	
	<u>simplificación</u>	<u>simplificación</u>	
0 0 1 1 ✓	X 0 1 1 *	1 X X 1	1 X X 1 *
0 1 1 0 ✓	X 1 1 0 *	1 X X 1	1 X 1 X *
1 0 0 1 ✓	1 0 X 1 ✓	1 X 1 X	<u>1 1 X X</u> *
1 0 1 0 ✓	1 X 0 1 ✓	1 X 1 X	
<u>1 1 0 0</u> ✓	1 0 1 X ✓	1 1 X X	
1 0 1 1 ✓	1 X 1 0 ✓	<u>1 1 X X</u>	
1 1 0 1 ✓	1 1 0 X ✓		
<u>1 1 1 0</u> ✓	<u>1 1 X 0</u> ✓		
<u>1 1 1 1</u> ✓	1 X 1 1 ✓		
	1 1 X 1 ✓		
	<u>1 1 1 X</u> ✓		

La tabla inicial se muestra a continuación, donde se han marcado (*) los implicants primos esenciales.

Tabla inicial

Implicantes primos	Minitérminos		
	$a_3 a_2 a_1 a_0$		
	0 0 1 1	0 1 1 0	1 0 0 1
* X011	✓		
* X110		✓	
* 1XX1			✓
1X1X			
11XX			

Se tienen tres implicantes primos esenciales que cubren todos los minitérminos de la función. La función mínima expresada como suma de productos es la siguiente:

$$z = a_2' a_1 a_0 + a_2 a_1 a_0' + a_3 a_0$$

Minimización como producto de sumas.

A continuación se muestran las simplificaciones:

Maxitérminos	primera	segunda	
	<u>simplificación</u>	<u>simplificación</u>	
<u>0 0 0 0</u> ✓	0 0 0 X✓	0 X 0 X	0 X 0 X *
0 0 0 1✓	0 0 X 0✓	X 0 X 0	X 0 X 0 *
0 0 1 0✓	0 X 0 0✓	0 X 0 X	<u>X X 0 0</u> *
0 1 0 0✓	<u>X 0 0 0</u> ✓	X X 0 0	X 1 0 X *
<u>1 0 0 0</u> ✓	0 X 0 1✓	X 0 X 0	<u>1 X X 0</u> *
0 1 0 1✓	X 0 1 0✓	<u>X X 0 0</u>	X 1 X 1 *
1 0 1 0✓	0 1 0 X✓	X 1 0 X	1 X 1 X *
<u>1 1 0 0</u> ✓	X 1 0 0✓	X 1 0 X	<u>1 1 X X</u> *
0 1 1 1✓	1 0 X 0✓	1 X X 0	
1 0 1 1	<u>1 X 0 0</u> ✓	<u>1 X X 0</u>	
1 1 0 1	0 1 X 1✓	X 1 X 1	
<u>1 1 1 0</u>	X 1 0 1✓	X 1 X 1	
<u>1 1 1 1</u>	1 0 1 X✓	1 X 1 X	
	1 X 1 0✓	1 X 1 X	
	1 1 0 X✓	1 1 X X	
	<u>1 1 X 0</u> ✓	1 1 X X	
	X 1 1 1✓		

$1\ X\ 1\ 1\ \checkmark$
 $1\ 1\ X\ 1\ \checkmark$
 $1\ 1\ 1\ X\ \checkmark$

La tabla inicial, donde se marcan (*) los implicantes primos esenciales es la siguiente:

		Maxitérminos						
		$a_3 a_2 a_1 a_0$	0000	0001	0010	0100	0101	0111
Implicantes primos								
	* 0X0X		✓	✓		✓	✓	
	* X0X0		✓		✓			✓
	XX00		✓			✓		✓
	X10X					✓	✓	
	1XX0							✓
	* X1X1						✓	✓
	1X1X							
	11XX							

Se tienen tres implicantes primos esenciales que cubren todos los maxitérminos de la función. La función mínima expresada como producto de sumas es la siguiente:

$$Z = (a_3 + a_1) (a_2 + a_0) (a_2' + a_0')$$

14.4 MINIMIZACION DE CIRCUITOS DE VARIAS SALIDAS.

La mayoría de los dispositivos lógicos que se usan en aplicaciones reales tienen varias entradas y varias salidas. Cada una de las salidas está representada por una función lógica que depende de las entradas.

En los capítulos anteriores se han presentado diversos dispositivos que tienen varias salidas. Para cada una de las salidas se ha determinado su función lógica y se ha procedido a simplificar cada una de estas funciones tomadas por separado. Una pregunta que surge respecto al procedimiento anterior es la siguiente: ¿Se obtiene el circuito lógico con la menor cantidad de bloques lógicos para el dispositivo?. La respuesta en general es no.

El problema de minimizar la cantidad de bloques que contiene el circuito lógico de un dispositivo de varias salidas no es fácil de resolver. Se han desarrollado varios métodos de minimización pero no todos obtienen el circuito mínimo. Sin embargo, todos estos métodos proporcionan una solución mejor, en la mayoría de los casos, a la obtenida al minimizar cada función de salida por separado.

En esta sección se presenta un método de minimización basado en el método tabular, que proporciona una solución bastante aceptable mediante un procedimiento sencillo y fácil de aplicar. El circuito que se obtiene es de dos niveles.

Para obtener los implicantes primos se toman todos los minitérminos (maxitérminos) de las funciones simultáneamente y a cada uno se le pone etiquetas que indican a cuál o cuáles funciones corresponde. Si un minitérmino corresponde a tres funciones deberá llevar la etiqueta de cada una de las funciones. Si existen combinaciones que no importan, deberán también incluirse en el proceso de obtención de implicantes primos.

Durante el proceso de simplificación, se podrán simplificar términos siempre y cuando tengan al menos una etiqueta en común. El término resultante de la simplificación llevará aquellas etiquetas que sean comunes a ambos términos que se simplificaron.

Una vez que se han obtenido los implicantes primos, se procede a formar la tabla con los minitérminos (maxitérminos) de las funciones y los implicantes primos con sus etiquetas, marcando los minitérminos que cubre cada implicante. Un implicante primo no podrá cubrir minitérminos de una función de la cual no tiene etiqueta.

De esta tabla se obtienen los implicantes primos esenciales y se obtiene la tabla reducida. Finalmente, se cubre la tabla reducida con el menor número de implicantes primos.

Se aplicará el método descrito anteriormente a la minimización de dos circuitos de tres salidas, en el primero se obtienen las funciones simplificadas expresadas como suma de productos y en el segundo como producto de sumas.

Minimización como suma de productos.

Considérese la siguiente tabla de verdad con cuatro entradas (**a, b, c y d**) y tres salidas (**f₁, f₂ y f₃**).

a	b	c	d	f₁	f₂	f₃
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	1	1	X

0	0	1	1	0	1	1
0	1	0	0	0	1	0
0	1	0	1	X	1	1
0	1	1	0	0	0	0
0	1	1	1	1	1	X
1	0	0	0	1	0	1
1	0	0	1	X	0	1
1	0	1	0	1	X	1
1	0	1	1	0	0	0
1	1	0	0	0	0	0
1	1	0	1	X	0	X
1	1	1	0	0	0	0
1	1	1	1	1	X	0

Para obtener la forma mínima expresada como suma de productos se ordenan los minitérminos de las funciones de salida de acuerdo al número de unos que contienen, indicando mediante etiquetas a cuál o cuáles salidas corresponden. Para este ejemplo se usarán como etiquetas los números 1, 2 y 3 para indicar las salidas f_1 , f_2 y f_3 . Las etiquetas se muestran entre paréntesis a la derecha de cada término.

En el proceso de simplificación, cuando dos términos se simplifican se marcarán (✓) solamente las etiquetas comunes a ambos términos. Si un término al final tuviera las etiquetas 1, 2 y 3 y solamente estuviera marcada la 2 como simplificada, dicho término sería implicante primo de la función 1 y 3 pero no necesariamente de la 2, ya que se simplificó con otro de la misma función 2 y quedó englobado en otro implicante primo para dicha función.

Por otro lado, un término que tenga marcadas todas sus etiquetas no es implicante primo de ninguna función ya que quedó englobado en algún implicante de cada una de las funciones de salida.

El proceso de simplificación sería el siguiente:

Minitérminos	primer simplificación	segunda simplificación
<u>0 0 1 0</u> (1✓, 2✓, 3✓)	0 0 1 X (2, 3)	X 1 X 1 (1) <u>X 1 X 1</u> (1)
0 1 0 0 (2✓)	X 0 1 0 (1, 2, 3)	<u>X 1 X 1</u> (1)
<u>1 0 0 0</u> (1✓, 3✓)	0 1 0 X (2)	
0 0 1 1 (2✓, 3✓)	1 0 0 X (1, 3)	
0 1 0 1 (1✓, 2✓, 3✓)	<u>1 0 X 0</u> (1, 3)	
1 0 0 1 (1✓, 3✓)	0 X 1 1 (2, 3)	
<u>1 0 1 0</u> (1✓, 2✓, 3✓)	0 1 X 1 (1✓, 2, 3)	
0 1 1 1 (1✓, 2✓, 3✓)	X 1 0 1 (1✓, 3)	
<u>1 1 0 1</u> (1✓, 3✓)	<u>1 X 0 1</u> (1, 3)	
<u>1 1 1 1</u> (1✓, 2✓)	X 1 1 1 (1✓, 2)	
	<u>1 1 X 1</u> (1✓)	

Una vez obtenidos los implicantes primos se procede a formar la tabla inicial. En dicha tabla se incluyen todos los implicantes primos con todas sus etiquetas aún y cuando alguna de ellas esté marcada como simplificada y los minitérminos de cada una de las funciones. Recuérdese que en esta tabla no se incluyen condiciones que no importan. La razón por la cual se ponen todos los implicantes primos con todas sus etiquetas es que pudiera darse el caso de que un implicante primo de una función pueda formarse con una operación **OR** de dos implicantes primos de otras dos funciones diferentes. Si estos implicantes primos se tomaran para las otras funciones, ya no tendría caso generar mediante un bloque lógico **AND** el implicante que englobara a ambos.

A continuación se muestra la tabla inicial donde se marcan (*) los implicantes primos esenciales.

Tabla Inicial

Minitérminos

Implicantes primos	abcd /	Función F1					Función F2					Función F3				
		0010	0111	1000	1010	1111	0010	0011	0100	0101	0111	0011	0101	1000	1001	1010
001X (2,3)							✓	✓				✓				
* X010 (1,2,3)		✓			✓		✓									✓
* 010X (2)									✓	✓						
100X (1,3)				✓										✓	✓	
10X0 (1,3)				✓	✓									✓		✓
0X11 (2,3)							✓				✓	✓				
01X1 (1,2,3)			✓							✓	✓		✓			
X101 (1,3)													✓			
1X01 (1,3)															✓	
X111 (1,2)			✓			✓					✓					
X1X1 (1)			✓			✓										

↑

↑

Nótese que el implicante primo esencial X010 formará parte de las tres funciones, ya que agrupa minitérminos en cada una de ellas.

Si eliminamos de la tabla todos los implicantes primos esenciales y los minitérminos que están cubiertos por los mismos, se obtiene la siguiente tabla reducida:

Tabla reducida

		Minitérminos								
Implicantes primos no esenciales	abcd	Función F1			F2		Función F3			
		0 1 1 1	1 0 0 0	1 1 1 1	0 0 1 1	0 1 1 1	0 0 1 1	0 1 0 1	1 0 0 0	1 0 0 1
001X (2,3)					✓		✓			
100X (1,3)			✓						✓	✓
10X0 (1,3)			✓						✓	
0X11 (2,3)					✓	✓	✓			
01X1 (1,2,3)		✓				✓		✓		
X101 (1,3)								✓		
1X01 (1,3)										✓
X111 (1,2)		✓		✓		✓				
X1X1 (1)		✓		✓						

Ahora se tratará de cubrir todos los minitérminos de la tabla reducida con el menor número de implicantes primos no esenciales. Este paso se puede simplificar eliminando renglones de igual forma en como se eliminaron cuando se minimizó una sola función. En este ejemplo se pueden aplicar las siguientes simplificaciones:

- El renglón del implicante primo 100X elimina a los renglones de los implicantes primos 10X0 Y 1X01.
- El renglón del implicante primo 0X11 elimina a renglón de el implicante primo 001X.
- El renglón del implicante primo 01X1 elimina al renglón de el implicante primo X101.
- El renglón del implicante primo X111 elimina al renglón de el implicante primo X1X1.

No se pueden eliminar columnas ya que al tomar un implicante primo no esencial, no se sabría si pertenece a todas las funciones cuyas etiquetas lo indica, como sucede con el implicante primo no esencial 01X1. Para este caso, aún y cuando tiene las etiquetas de las funciones 1, 2 y 3, no es necesario incluirlo en las funciones mínimas 1 y 2. Con el implicante primo X111 ocurre algo semejante.

A continuación se muestra la tabla resultante.

		Minitérminos								
Implicantes primos no esenciales	abcd	Función F1			F2		Función F3			
		0111	1000	1111	0011	0111	0011	0101	1000	1001
100X (1,3)		✓							✓	✓
0X11 (2,3)					✓	✓	✓			
01X1 (1,2,3)		✓				✓		✓		
X111 (1,2)		✓		✓		✓				

Si se puede hacer el proceso de eliminación de columnas cuando el análisis se realiza con las columnas de una misma función. Par este caso se podría eliminar la columna del minitérmino 0111 de la función 1 y la columna 0111 de la función 2.

La opción que cubre a todos los minitérminos con el menor número de implicantes es la que se muestra marcada en la tabla.

Las funciones mínimas que se obtienen están formadas como sigue:

Función f_1 —→ X010, 100X, X111

Función f_2 —→ X010, 010X, 0X11

Función f_3 —→ X010, 100X, 0X11, 01X1

Puesto en otra forma, se tiene lo siguiente:

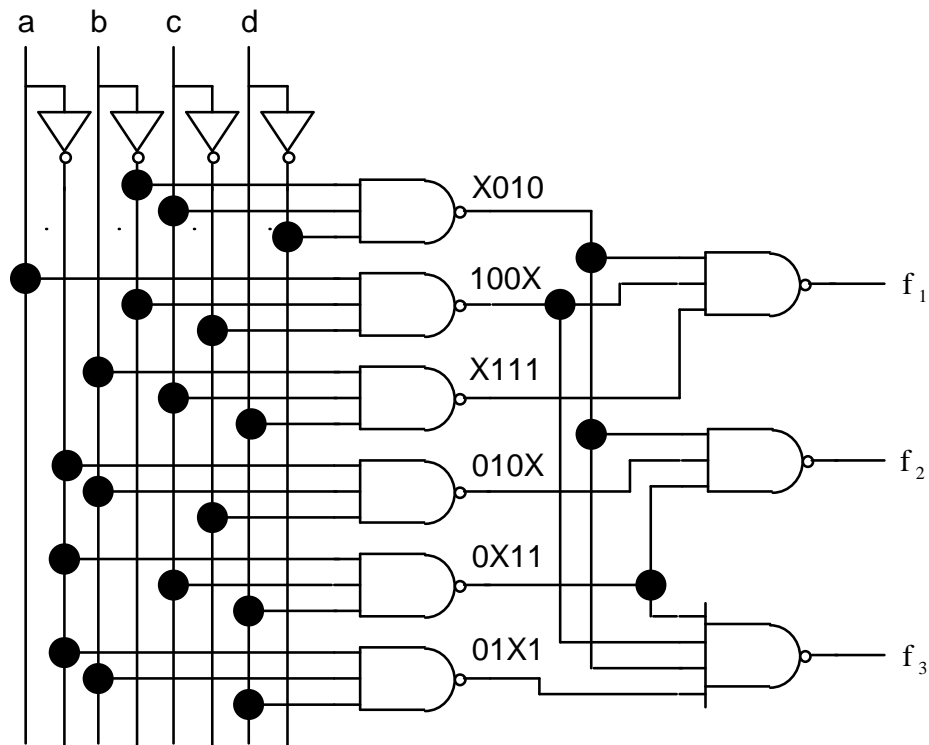
$$f_1 = b'cd' + ab'c' + bcd$$

$$f_2 = b'cd' + a'bc' + a'cd$$

$$f_3 = b'cd' + ab'c' + a'cd + a'bd$$

Obsérvese que con solamente seis implicantes primos se pueden construir las tres funciones.

El circuito lógico final se muestra a continuación construido con bloques **NAND**.



Si se hubiesen minimizado cada una de las funciones por separado, se hubieran necesitado ocho términos en el mejor de los casos para construir el circuito lógico correspondiente.

En algunos casos es posible obtener circuitos lógicos con menos bloques lógicos que los obtenidos por el método presentado, pero es necesario usar más de dos niveles para construir el circuito lo cual no es muy deseable. Recuerde que entre más niveles contenga un circuito lógico, el tiempo de retraso será mayor.

Minimización como producto de sumas.

La minimización de un circuito de varias salidas como producto de sumas es similar al proceso de minimización como suma de productos solamente que en este caso se trabaja con los maxitérminos.

El procedimiento se mostrará a través del ejemplo que se presenta a continuación:

Considérese la siguiente tabla de verdad con cuatro entradas (**a, b, c y d**) y tres salidas (**f₁, f₂ y f₃**).

a	b	c	d	f ₁	f ₂	f ₃
0	0	0	0	0	0	X

0	0	0	1	1	1	1
0	0	1	0	0	0	0
0	0	1	1	1	1	1
0	1	0	0	1	1	1
0	1	0	1	0	0	X
0	1	1	0	0	1	1
0	1	1	1	X	0	1
1	0	0	0	X	0	0
1	0	0	1	1	1	1
1	0	1	0	X	0	0
1	0	1	1	1	1	1
1	1	0	0	1	1	1
1	1	0	1	0	X	0
1	1	1	0	X	0	X
1	1	1	1	0	X	0

El proceso de simplificación sería el siguiente:

<u>Maxitérminos</u>	<u>primer simplificación</u>
<u>0 0 0 0</u> (1✓, 2✓, 3✓)	0 0 X 0 (1✓, 2✓, 3✓)
0 0 1 0 (1✓, 2✓, 3✓)	<u>X 0 0 0</u> (1✓, 2✓, 3✓)
<u>1 0 0 0</u> (1✓, 2✓, 3✓)	0 X 1 0 (1✓)
0 1 0 1 (1✓, 2✓, 3✓)	X 0 1 0 (1✓, 2✓, 3✓)
0 1 1 0 (1✓)	<u>1 0 X 0</u> (1✓, 2✓, 3✓)
<u>1 0 1 0</u> (1✓, 2✓, 3✓)	0 1 X 1 (1✓, 2✓)
0 1 1 1 (1✓, 2✓)	X 1 0 1 (1✓, 2✓, 3)
1 1 0 1 (1✓, 2✓, 3✓)	0 1 1 X (1✓)
<u>1 1 1 0</u> (1✓, 2✓, 3✓)	X 1 1 0 (1✓)
<u>1 1 1 1</u> (1✓, 2✓, 3✓)	<u>1 X 1 0</u> (1✓, 2, 3)
	X 1 1 1 (1✓, 2✓)
	1 1 X 1 (1✓, 2✓, 3)
	<u>1 1 1 X</u> (1✓, 2, 3)

<u>segunda simplificación</u>
X0X0 (1, 2, 3) <u>X0X0</u> (1, 2, 3)
<u>X0X0</u> (1, 2, 3) <u>XX10</u> (1)
XX10 (1) X1X1 (1, 2)
<u>XX10</u> (1) <u>X11X</u> (1)
X1X1 (1, 2)
X1X1 (1, 2)
X11X (1)
<u>X11X</u> (1)

Una vez obtenidos los implicantes primos, se procede a formar la tabla inicial. Recuerde que en esta tabla se incluyen todos los implicantes primos con todas sus etiquetas, aún y cuando alguna de ellas esté marcada como simplificada así como los maxitérminos de cada una de las funciones.

Tabla Inicial

		Maxitérminos																	
Implicantes primos	abcd /	Función F1					Función F2					Función F3							
		0000	0010	0101	0110	1101	1111	0000	0010	0101	0111	1000	1010	1110	0010	1000	1010	1101	1111
1X10 (1,2,3)													✓	✓			✓		
* 11X1 (1,2,3)						✓	✓											✓	✓
111X (1,2,3)							✓							✓					✓
* X0X0 (1,2,3)		✓	✓					✓	✓			✓	✓		✓	✓	✓		
XX10 (1)			✓		✓														
* X1X1 (1,2)				✓		✓	✓			✓	✓								
X11X (1)					✓		✓												
		↑		↑				↑	↑	↑	↑	↑			↑	↑			↑

Si se eliminan de la tabla todos los implicantes primos esenciales y los minitérminos que están cubiertos por los mismos, se obtiene la siguiente tabla reducida:

		Maxitérminos	
Implicantes primos no esenciales	abcd /	F1	F2
		0110	1110
1X10 (1,2,3)			✓
111X (1,2,3)			✓
XX10 (1)		✓	
X11X (1)		✓	

Ahora se tratará de cubrir todos los maxitérminos de la tabla reducida con el menor número de implicantes primos no esenciales. El primero o segundo renglón puede ser eliminado, al igual que el tercero o cuarto renglón.

Si eliminamos el segundo y tercer renglón, las funciones mínimas que se obtienen están formadas como sigue:

Función $f_1 \longrightarrow 11X1, X0X0, X1X1, X11X$

Función $f_2 \longrightarrow X0X0, X1X1, 1X10$

Función $f_3 \longrightarrow 11X1, X0X0$

Puesto en otra forma se tiene lo siguiente:

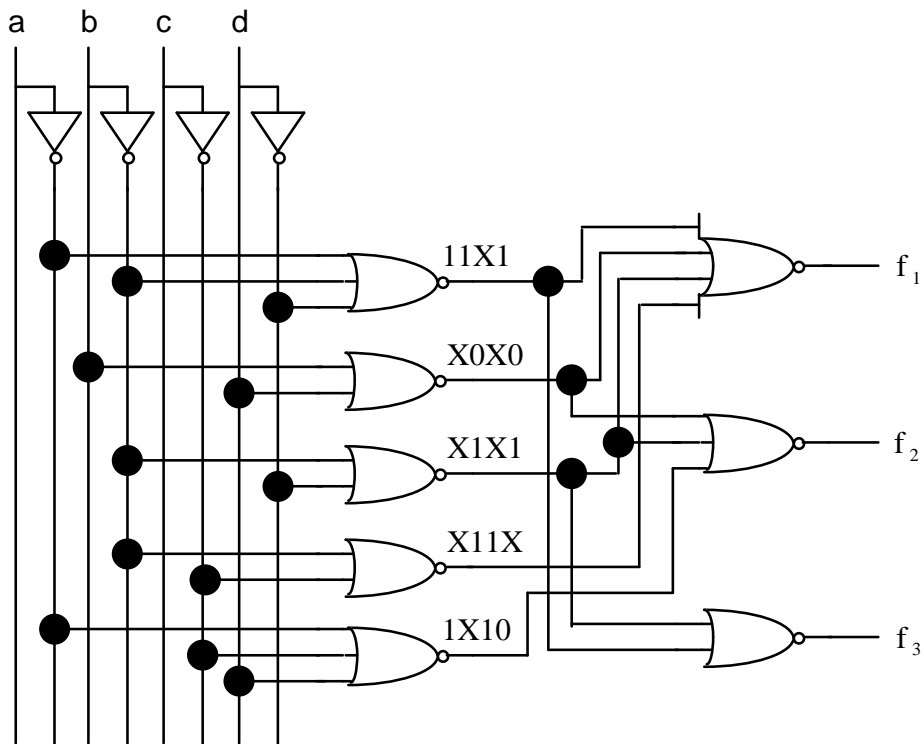
$$f_1 = (a' + b' + d')(b + d)(b' + d')(b' + c')$$

$$f_2 = (b + d)(b' + d')(a' + c' + d)$$

$$f_3 = (a' + b' + d')(b + d)$$

Obsérvese que el implicante primo esencial 11X1 no está en la función dos ya que no agrupo a maxitérminos en ella.

El circuito lógico final se muestra a continuación, construido con bloques **NOR**.



14.5. RESUMEN

La minimización de funciones booleanas es un tema importante debido a que una función booleana que está representada en su forma mínima puede ser construida con un circuito lógico de dos niveles que necesita menos bloques lógicos que el circuito de la misma función expresada en otras formas diferentes. Un circuito lógico con menos bloques permite ser construido de una manera más económica y además proporciona mayor confiabilidad ya que al tener menos bloques, también se reduce la posibilidad de falla en el circuito.

En este capítulo se presentó el método de minimización de funciones lógicas utilizando mapas de Karnaugh, el cual es adecuado para minimizar funciones con 5 variables o menos de una manera visual. El método puede ser aplicado para obtener la forma mínima de una función expresada como suma de productos o como producto de sumas.

Para funciones de seis o más variables se presentó el método de minimización tabular, que también puede ser utilizado para minimizar funciones de cinco variables o menos. Este último método puede ser computarizado fácilmente y aplicado a funciones con cualquier número de variables independientes, permitiendo obtener las formas mínimas de las funciones expresadas como producto de sumas y como suma de productos.

Se presentó también la manera de aplicar el método de minimización tabular a circuitos lógicos de varias salidas, resaltando el hecho de que si se minimizan cada una de las funciones de salida en forma individual, no necesariamente se obtiene el circuito lógico con menos bloques.

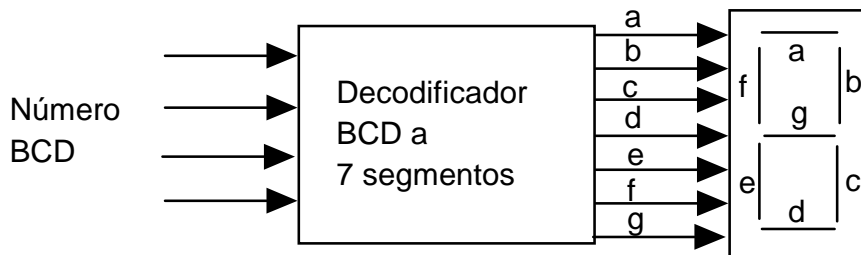
14.6 PROBLEMAS.

- 1.- Obtenga las formas mínimas como suma de productos y como producto de sumas de las siguientes funciones usando mapas de Karnaugh.
 - a) $f = ab + a'(bc + b')$
 - b) $f = (a + b)(a + c')(a' + b' + c)$
 - c) $f = xy' + x'yz + xy'z' + x'y'z$
 - d) $x = a'c'd'e' + a'bd + ace + ab'c'd'e' + a'bd'e + a'cde' + a'b'ce + ac'd'e'$
- 2.- Obtenga las funciones mínimas como suma de productos y como producto de sumas de las funciones del problema 1 usando el método tabular.
- 3.- A continuación se muestra la tabla de verdad para las funciones f, g y h. Obtenga ambas formas mínimas usando mapas de Karnaugh.

a	b	c	d	f	g	h
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	1	1	X
0	0	1	1	0	1	1
0	1	0	0	0	1	0

0	1	0	1	X	1	1
0	1	1	0	0	0	0
0	1	1	1	1	1	X
1	0	0	0	1	0	1
1	0	0	1	X	0	1
1	0	1	0	1	X	1
1	0	1	1	0	0	0
1	1	0	0	0	0	0
1	1	0	1	X	0	X
1	1	1	0	0	0	0
1	1	1	1	1	X	0

- 4.- Obtenga las formas mínimas expresadas como suma de productos y como producto de sumas para las funciones del problema 3 usando el método tabular.
5. Obtenga las formas mínimas expresadas como suma de productos y como producto de sumas para la siguiente función usando mapas de Karnaugh.
- $$F(A, B, C, D, E) = \sum m(2, 3, 5, 7, 11, 12, 13, 16, 17, 24, 25, 26, 27, 28, 29, 30, 31) + x(0, 1, 8, 9, 18)$$
6. Obtenga las formas mínimas expresadas como suma de productos y como productos de suma para las funciones del problema 5 usando el método tabular
- 7.- Diseña y construye el mínimo decodificador de BCD a 7 segmentos.



Las funciones **a, b, c** y **d** deben ser construidas con el mínimo circuito **NAND-NAND**.

Las funciones **e, f** y **g** deben ser construidas con el mínimo circuito **NOR-NOR**.

Simplifica las funciones **a, b, d, e** y **g** usando mapas de Karnaugh y con el método tabular simplifica **c** y **f**.

Nota :

El número 6 puede prender el segmento **a** o no prenderlo.

El número 9 puede prender el segmento **d** o no prenderlo.

El número 7 no prende el segmento **g**.

8. Usando el método de minimización de circuitos de varias salidas, encuentre el mínimo circuito **NAND-NAND** para los segmentos **a,b,c,d,e,f** y **g** del problema 7.
9. Usando el método de minimización de circuitos de varias salidas, encuentre el mínimo circuito **NOR-NOR** para los segmentos **a,b,c,d,e,f** y **g** del problema 7.
10. Demuestre que la solución dada por el método de minimización de circuitos de varias salidas, no es mejor que la dada al minimizar en forma independiente cada una de las siguientes funciones en suma de productos:

$$F_1(A, B, C, D) = \sum m(0, 1, 5, 6, 10, 11, 12, 14)$$

$$F_2(A, B, C, D) = \sum m(2, 3, 4, 5, 7, 8, 9, 15)$$

$$F_3(A, B, C, D) = \sum m(0, 1, 2, 3, 6, 7, 8, 9, 10, 11, 14, 15)$$

CAPITULO 15

CIRCUITOS LOGICOS COMBINATORIOS QUE REALIZAN FUNCIONES COMUNMENTE UTILIZADAS.

En los capítulos anteriores se han presentado las bases para el análisis y diseño de circuitos lógicos combinatorios. En este capítulo se desarrollarán algunos circuitos que realizan funciones comúnmente usadas en el área de diseño de circuitos lógicos computacionales. Los circuitos que se mostrarán se consiguen fácilmente en un sólo circuito integrado y, cuando es necesario usarlos, basta conseguir el circuito integrado correspondiente. Sin embargo, es conveniente saber como están contruidos para hacer uso de ellos en forma eficiente.

15.1 OPERACIONES EX-OR (O EXCLUSIVO) Y COIN (EQUIVALENCIA)

La operación **EX-OR** actúa sobre dos operandos. Su definición se muestra en la tabla 15.1 y se representa como se indica a continuación:

$$A \oplus B = AB' + A'B \quad (\text{Léase "A o exclusivo B"})$$

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Tabla 15.1 Operación **EX-OR**

De la tabla se puede observar que para que el resultado de la operación **EX-OR** sea uno, es necesario que sólo uno de los dos operadores tenga el valor de uno.

A continuación se presentan una serie de propiedades del EX-OR.

$$\begin{aligned} X \oplus 0 &= X \\ X \oplus 1 &= X' \\ X \oplus X &= 0 \\ X \oplus X' &= 1 \\ X \oplus Y &= Y \oplus X \\ (X \oplus Y)' &= X' \oplus Y = X \oplus Y' \end{aligned}$$

Se deja como ejercicio para el lector la comprobación de estas propiedades.

La operación **COIN** (del inglés "coincidence"), conocida también como operación equivalencia, actúa sobre dos operandos. Su definición se muestra en la tabla 15.2 y se representa como se indica a continuación:

$$A \equiv B = A'B' + AB = (A \oplus B)' \quad (\text{Léase "A equivalencia B"})$$

A	B	$A \equiv B$
0	0	1
0	1	0
1	0	0
1	1	1

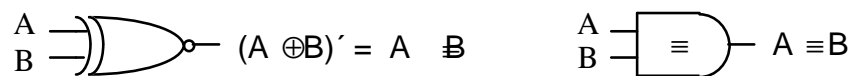
Tabla 15.2 Operación **COIN**

La operación **COIN** tomará el valor de uno cuando los dos operandos tengan el mismo valor.

En la figura 15.1 se muestran los símbolos utilizados para los bloques lógicos que representan estas operaciones.



bloque **EX-OR**.



bloque **COIN**

Figura 15.1. Bloques lógicos **EX-OR** y **COIN**

Al igual que los bloques lógicos **AND**, **OR**, **NAND** y **NOR**, estos bloques también pueden tener más de dos entradas. La forma de analizar el comportamiento de un bloque lógico de más de dos entradas se muestra por medio de un ejemplo con un **EX-OR** de 3 entradas.

$$A \oplus B \oplus C = (A \oplus B) \oplus C = (AB' + A'B) \oplus C$$

$$\begin{aligned}
&= (AB' + A'B)C' + (AB' + A'B)'C \\
&= AB'C' + A'BC' + (A' + B)(A + B')C \\
&= AB'C' + A'BC' + (A'A + A'B' + AB + BB')C \\
&= AB'C' + A'BC' + (A'B' + AB)C \\
&= AB'C' + A'BC' + A'B'C + ABC
\end{aligned}$$

15.2 DECODIFICADORES.

Un decodificador es un circuito lógico que tiene n entradas y 2^n salidas. Para una combinación dada de las n entradas, solamente una salida tomará el valor de uno y todas las demás tomarán el valor de cero. Si se tienen n entradas para un circuito lógico, existirán 2^n combinaciones posibles de los valores de las variables independientes, cada una de las cuales corresponderá a un minitérmino de las n variables independientes. Por consiguiente cada minitérmino está asociado a una de las 2^n salidas. El nombre que se le da a este dispositivo es decodificador de $n \times 2^n$.

Estos dispositivos normalmente cuentan con una entrada adicional denominada habilitadora. Cuando esta entrada vale cero, todas las salidas del decodificador son cero. Cuando la entrada habilitadora vale uno, la salida correspondiente al minitérmino formado por la combinación presente en las n entradas tomará el valor de uno y las demás tomarán en valor de cero.

El esquema y funcionamiento de un decodificador de 2×4 se muestra en la figura 15.2:

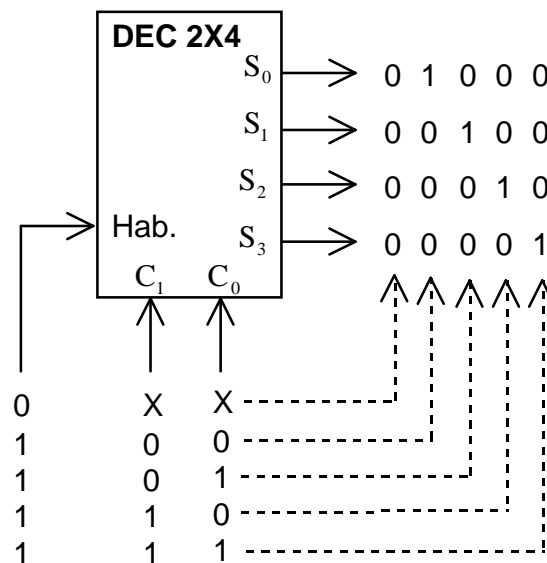


Figura 15.2. Decodificador de 2×4

Un valor de **X** en las entradas **C₁** y **C₂** indica que puede tomar el valor de uno o cero.

En la figura 15.3. se muestra el circuito lógico de un decodificador de 2×4 :

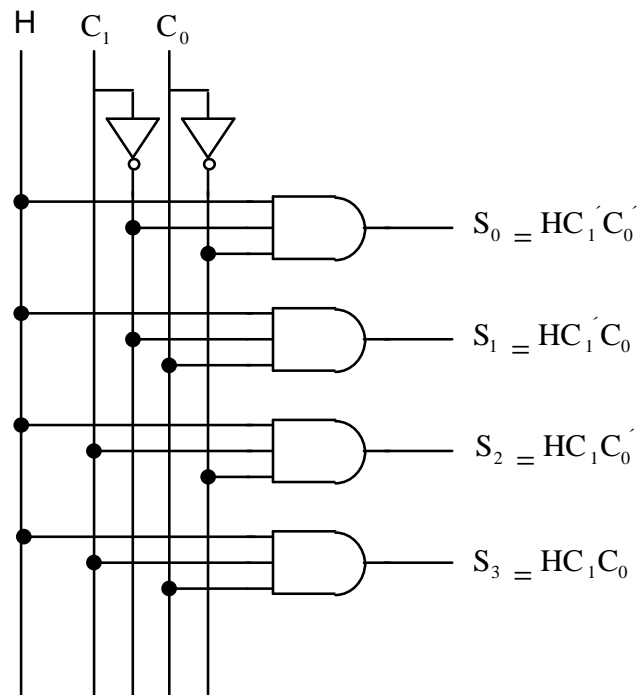


Figura 15.3. Circuito lógico de un decodificador de **2X4**

De forma semejante a como se define el decodificador de **2 X 4**, se pueden definir decodificadores de **3 X 8**, **4 X 16**, **5 X 32** y en forma general de **n X 2ⁿ**.

La principal utilización de este dispositivo es cuando se tienen **N** alternativas que se pueden seleccionar, pero se desea seleccionar solamente una de ellas. Un decodificador también puede ser utilizado para construir funciones lógicas como se muestra en el siguiente ejemplo.

Ejemplo 15.1. Dada la función representada en la siguiente tabla de verdad, construya su circuito usando un decodificador de **3 X 8**.

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Tabla 15.3. Tabla de verdad para el ejemplo 15.1

Recuérdese que cada una de las salidas de un decodificador corresponde a un minitérmino. El circuito lógico se muestra en la figura 15.3.

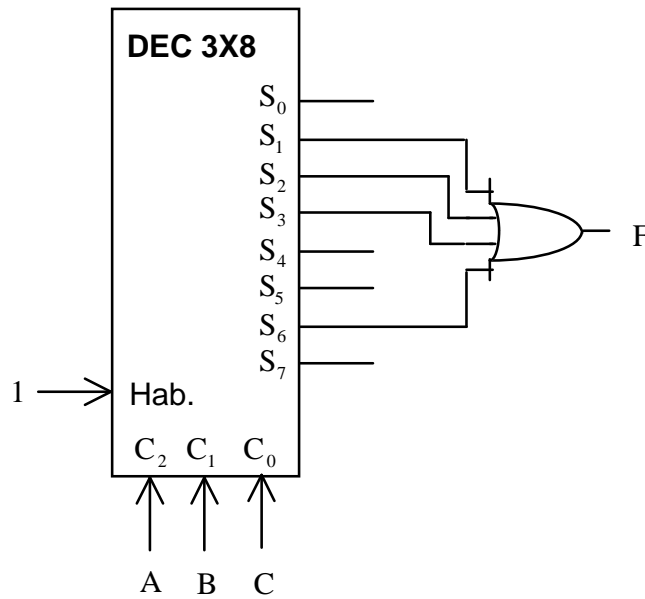


Figura 15.3. Circuito lógico para el ejemplo 15.1

Nótese que al poner un 1 en la entrada habilitadora, en las ocho salidas se tienen los ocho minterminos posibles y con el bloque **OR** se seleccionan sólo los minterminos pertenecientes a la función.

15.3 SELECTORES.

Los selectores son dispositivos lógicos que tienen un conjunto de entradas de control además de las entradas y salidas de datos. Este conjunto de entradas de control es el que rige la operación del dispositivo.

Existen dos tipos básicos de selectores:

- De varias entradas a una salida, llamados selectores de 2^n a 1 o simplemente **MUX** (del inglés "multiplexer") de 2^n a 1.
- De una entrada a varias salidas, llamados selectores de 1 a 2^n o simplemente **DEMUX** (del inglés "demultiplexer") de 1 a 2^n .

En ambos tipo de selectores se tienen **n** entradas de control. La operación de cada uno se describe a continuación.

15.3.1 Selectores de 2^n entradas a 1 salida.

Este tipo de selector tiene n entradas de control, 2^n entradas de datos y una salida. Al tener n entradas de control, se tienen 2^n posibles combinaciones de valores de las entradas de control, cada una de los cuales puede asociarse con el minitérmino correspondiente, en forma similar a lo que ocurre en un decodificador.

El valor de la salida es el correspondiente a la entrada que se selecciona mediante las entradas de control. En otras palabras, las entradas de control sirven para seleccionar cual de las entradas de datos aparece en la salida, como se muestra en el siguiente diagrama esquemático de un selector de **4:1** (de 4 entradas a 1 salida) en la figura 15.4.

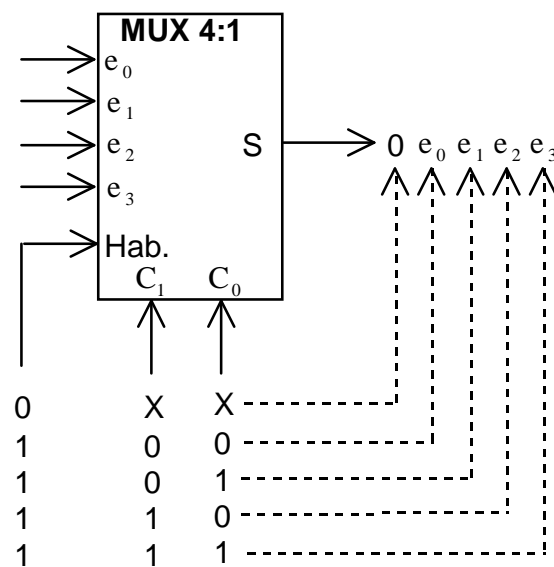


Figura 15.4. Selector de **4:1**

El valor de **X** en las entradas C_1 y C_2 indica que puede tomar el valor de uno o cero.

Este dispositivo también tiene un entrada para habilitarlo, la cual debe tener el valor de uno para que en la salida se obtenga el valor de la entrada seleccionada por las entradas de control. Si la entrada habilitadora vale cero, la salida tomará el valor de cero, independientemente de los valores presentes en las otras entradas.

La tabla de verdad para la salida **S** en función de las entradas de control (C_1 y C_0) y la entrada habilitadora (**H**) es la siguiente:

H	C_1	C_0	S
0	0	0	0
0	0	1	0

0	1	0	0
0	1	1	0
1	0	0	e ₀
1	0	1	e ₁
1	1	0	e ₂
1	1	1	e ₃

Tabla 15.4. Tabla de verdad para el selector de **4:1**

La función booleana para **S** es:

$$S = H C_1' C_0' e_0 + H C_1' C_0 e_1 + H C_1 C_0' e_2 + H C_1 C_0 e_3$$

El circuito lógico para este selector se muestra en la figura 15.5.

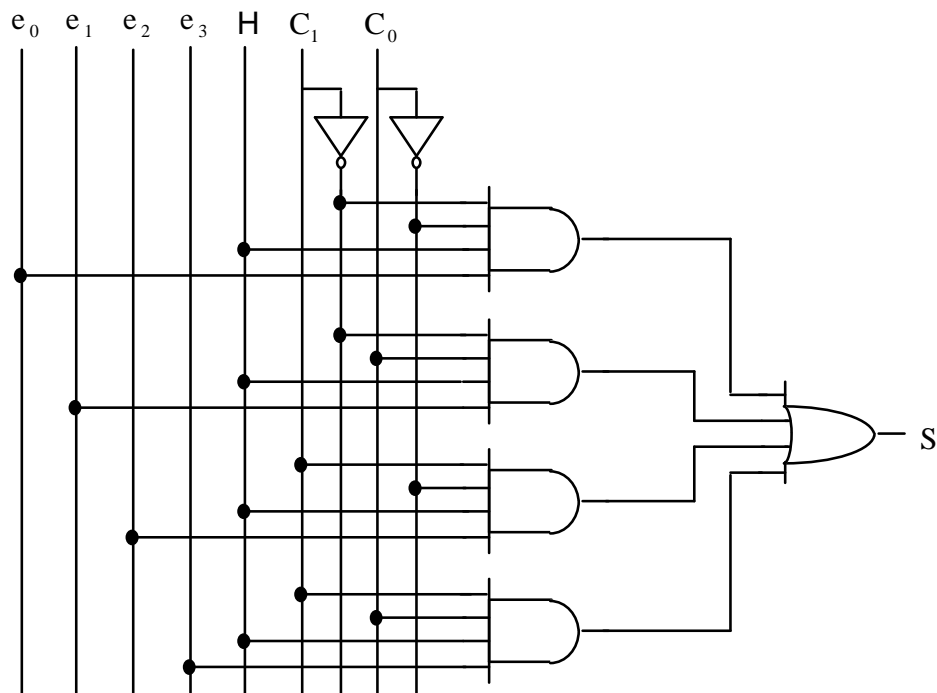


Figura 15.5. Circuito lógico de un selector de **4:1**

De forma semejante se pueden definir selectores de **8:1**, **16:1**, **32:1** y en forma general de **2ⁿ:1**.

La principal utilización de este dispositivo es cuando se tienen N entradas posibles, pero sólo se desea seleccionar una de ellas. Este dispositivo también puede ser usado para construir funciones como se muestra en el ejemplo 15.2.

Ejemplo 15.2: Dada función representada en la siguiente tabla de verdad, construya su circuito usando un selector de 8:1.

A	B	C	F
----------	----------	----------	----------

0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Tabla 15.5. Tabla de verdad para el ejemplo 15.2

El circuito lógico se muestra en la figura 15.6.

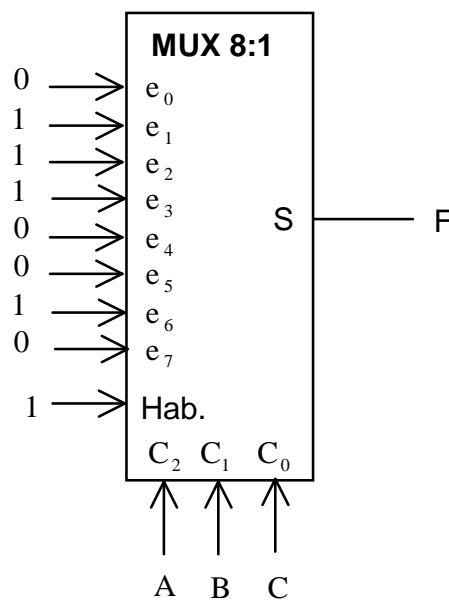


Figura 15.6. Circuito lógico del ejemplo 15.2.

Nótese que al poner un 1 en el habilitador y los valores en cada una de las entradas de control, la salida tomará el valor de la función deseada. Este circuito también puede ser construido con selectores con menos entradas como se muestra en la figura 15.7.

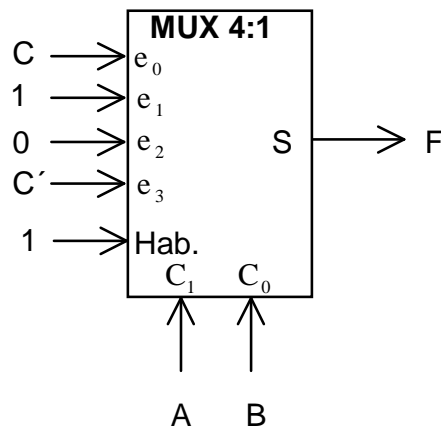


Figura 15.7. Circuito lógico del ejemplo 15.2 con selectores de 4:1.

15.3.2 Selectores de 1 entrada a 2ⁿ salidas.

Este dispositivo tiene n entradas de control, una entrada habilitadora, una sola entrada de datos y 2^n salidas. Las entradas de control sirven para seleccionar a cual de las 2^n salidas de datos se conectará la entrada. En otras palabras, las entradas de control sirven para seleccionar en cual de las salidas de datos aparece la entrada e , como se muestra en el diagrama esquemático para un selector de **1:4** (de 1 entrada a 4 salidas) de la figura 15.8.

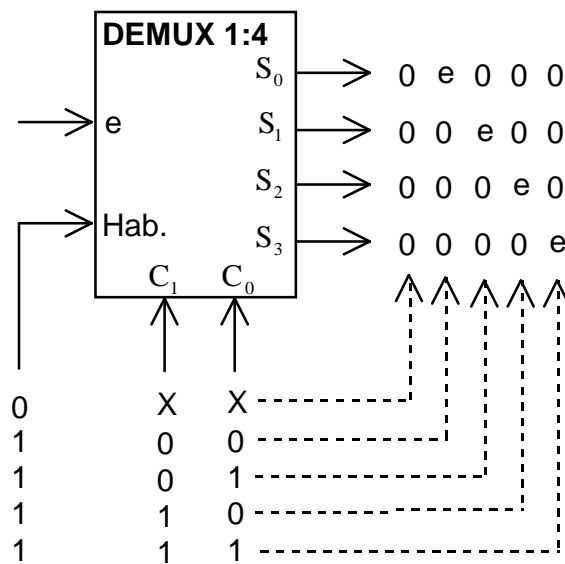


Figura 15.8. Circuito lógico de un selector de **1:4**.

El valor de **X** en las entradas C_1 y C_2 indica que puede tomar el valor de uno o cero.

Las funciones lógicas para las salidas del selector de **1:4** son:

$$\begin{aligned}
 S_0 &= H C_1' C_0' e \\
 S_1 &= H C_1' C_0 e \\
 S_2 &= H C_1 C_0' e \\
 S_3 &= H C_1 C_0 e
 \end{aligned}$$

El circuito lógico para este selector se muestra en la figura 15.9.

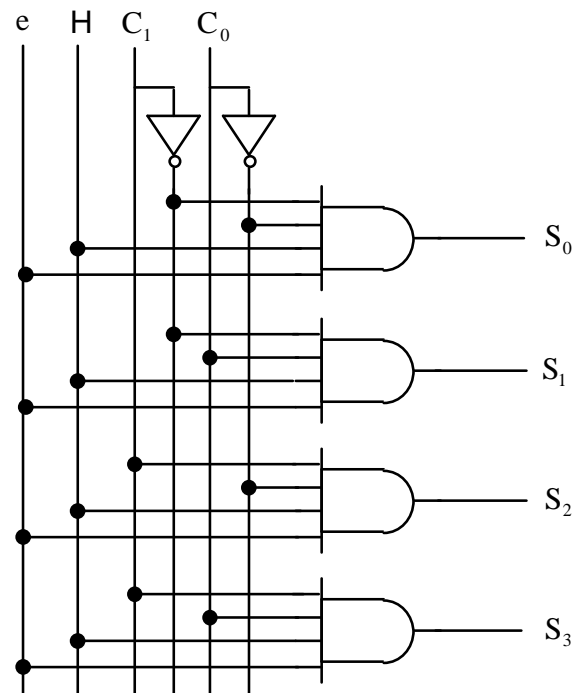


Figura 15.9. Circuito lógico de un selector de **1:4**.

De forma semejante, se pueden definir selectores de **1:8**, **1:16**, **1:32** y, en forma general, de **1:2ⁿ**.

La principal utilización de este dispositivo es cuando se tiene una sola entrada que puede ser dirigida a solamente una de N salidas posibles.

15.4. CONSTRUCCION DE FUNCIONES LOGICAS CON DISPOSITIVOS PROGRAMABLES.

La idea básica en un dispositivo programable es proporcionar un arreglo de ANDs cuyas salidas puedan ser conectadas a las entradas de un arreglo de ORs y con esto poder construir cualquier función lógica. La figura 15.10 ejemplifica esta idea, e_i son las variables de entrada y f_i son las funciones de salida.

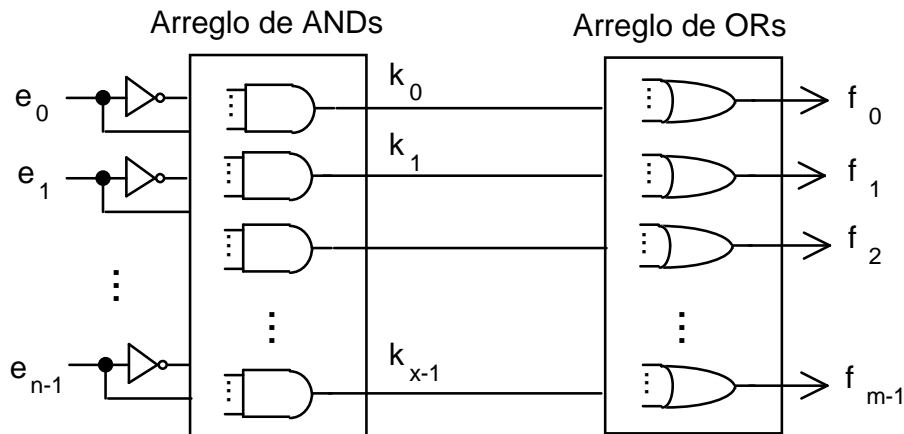


Figura 15.10 Idea básica en un dispositivo programable.

Dependiendo del dispositivo que se utilice, se puede controlar o no la conexión de las entradas externas (e_i) a las entradas de los ANDs, y poder controlar la conexión de las salidas de los ANDs (k_i) a cada una de las entradas de los ORs. Sí se puede tener un control en las conexiones se dice que puede ser programado.

En esta sección se presentan dos dispositivos de este tipo. Uno es la memoria estática en la cual sólo se puede programar el arreglo de ORs y el otro es el arreglo lógico programable en el cual se puede controlar el arreglo de ANDs y el arreglo de ORs.

15.4.1 Memorias estáticas.

Una memoria estática puede considerarse como un conjunto de posiciones de memoria en los cuales está almacenada cierta información. Cada una de estas posiciones de memoria se identifica por una dirección única, la cual recibe el nombre de palabra.

En cada palabra se puede almacenar un número fijo de bits, los cuales pueden representar un dato o una instrucción.

Una memoria estática tiene un conjunto de entradas que sirven para proporcionar la dirección de la palabra cuya información se desea conocer, un conjunto de salidas, que corresponden a cada uno de los bits de la palabra que se ha seleccionado mediante la dirección, y una entrada de control para habilitar o deshabilitar la memoria.

Al conjunto de entradas donde se indica la dirección se les conoce como bus de direcciones, la entrada de control se le llama selector del circuito y al conjunto de líneas de salida se le conoce como bus de datos.

En la figura 15.11 se ejemplifica una memoria estática de n entradas para dirección y palabras de m bits. En esta memoria se tendrán 2^n palabras de m bits, ya que con n líneas de dirección se puede indicar un valor entre 0 y $2^n - 1$ y cada una de estas direcciones contiene una palabra de m bits.

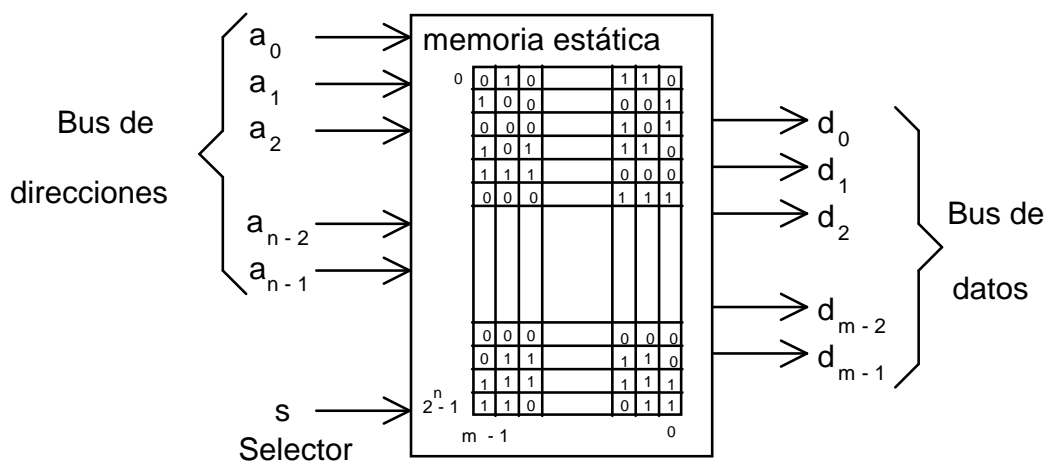


Figura 15.11. Memoria estática de $2^n \times m$ bits

La principal característica de las memorias estáticas es que no son volátiles, por lo cual son muy útiles en un sistema computacional, para que al momento de encenderlo éste tenga algunas rutinas y datos en su memoria o para tener cargadas en memoria principal rutinas de uso frecuente y así no tener que cargarlas de memoria secundaria.

Las primeras memorias estáticas que se construyeron son las conocidas como memorias **ROM** (del inglés "Read Only Memory"), la información que está almacenada en cada una de las palabras se fija al momento de construir la memoria y ya no puede ser cambiada posteriormente. Lógicamente, su construcción esta hecha en base a un decodificador y un bloque **OR** por cada uno de los bits de la palabra, los cuales tienen una entrada por cada palabra.

La construcción lógica de un ROM de 4 palabras de 3 bits con la información mostrada en la tabla 15.6 se muestra en la figura 15.12.

Dirección	Contenido		
00	0	1	1
01	0	1	0
10	1	0	0

11	1	1	1
----	---	---	---

Tabla 15.6 Contenido de la memoria ROM

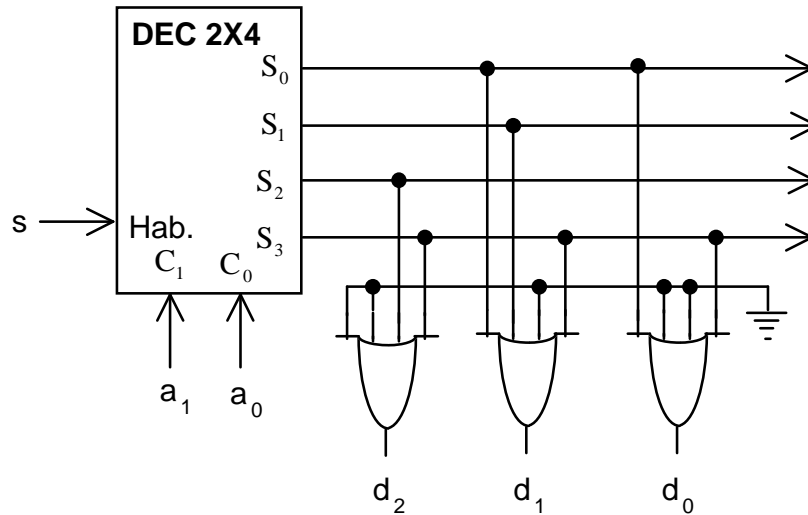


Figura 15.12. Construcción lógica de una memoria ROM

Para analizar la construcción lógica de un ROM tome encuentra lo siguiente: cada salida del decodificador identifica a una palabra del ROM y cada una de las entradas de los **ORs** corresponde a una palabra del ROM. Las conexiones se realizan de la siguiente forma: las primeras entradas de cada uno de los **ORs** pertenecen a la palabra cero, éstas se conectan a la salida cero del decodificador si el contenido en el bit correspondiente en el ROM es uno, si no, se conectan a tierra. Se repite esto con cada una de las líneas.

El lector puede comprobar que al seleccionar el decodificador y poner una dirección en el bus de direcciones, el bus de datos tendrá la información correspondiente a la palabra seleccionada.

En la memoria ROM, el arreglo de **ANDs** formado por el decodificador está fijo, esto es, las entradas a cada uno de los **ANDs** ya está definido. El arreglo de **ORs** no está definido, se programa con la información que debe contener el ROM.

Si se tienen **m** funciones booleanas que dependen de **n** variables, éstas pueden ser construidas con un ROM de 2^n palabras de **m** bits. El bus de direcciones sería controlado por las variables de entrada y en el bus de datos se tendrían las funciones. Cada una de las palabras es un minitérmino de las funciones y deben tener el valor de uno solo en los bits correspondientes a las funciones que contienen este minitérmino.

En la memoria ROM de la figura 15.12 se tiene la construcción de las siguientes tres funciones:

$$\begin{aligned} d_2 &= a_1 a_0' + a_1 a_0 \\ d_1 &= a_1' a_0' + a_1' a_0 + a_1 a_0 \\ d_0 &= a_1' a_0' + a_1 a_0 \end{aligned}$$

El circuito de una memoria ROM puede ser construido con un decodificador y una matriz de diodos. El diodo es un elemento analógico pero en su comportamiento ideal se puede utilizar como un elemento digital. El símbolo utilizado para su representación es el mostrado en la figura 15.13.

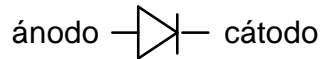


figura 15.13. Símbolo utilizado para representar un diodo.

El funcionamiento ideal del diodo es el siguiente: si en el ánodo se tiene un voltaje mayor que en el cátodo, el diodo se comporta como un corto circuito permitiendo el paso de la corriente. En caso contrario el diodo se comporta como un circuito abierto como se muestra en la figura 15.14. Recuerde que en los circuitos digitales, el uno lógico es representado por 5 volts y el cero lógico por 0 volts.

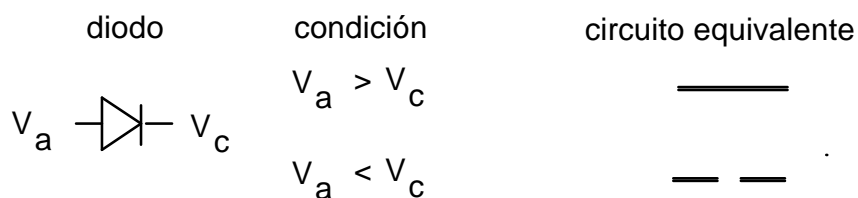



Figura 15.14. Funcionamiento ideal de un diodo

Primero se analizara un ROM con una matriz de diodos completa, el cual se muestra en la figura 15.15.

El símbolo  indica una conexión a tierra, que equivale a tener un cero lógico en la entrada.

La resistencia (R) es un elemento pasivo, cuya única función en el circuito es evitar que se produzca un corto circuito al poner un uno lógico con un cero lógico en la misma línea.

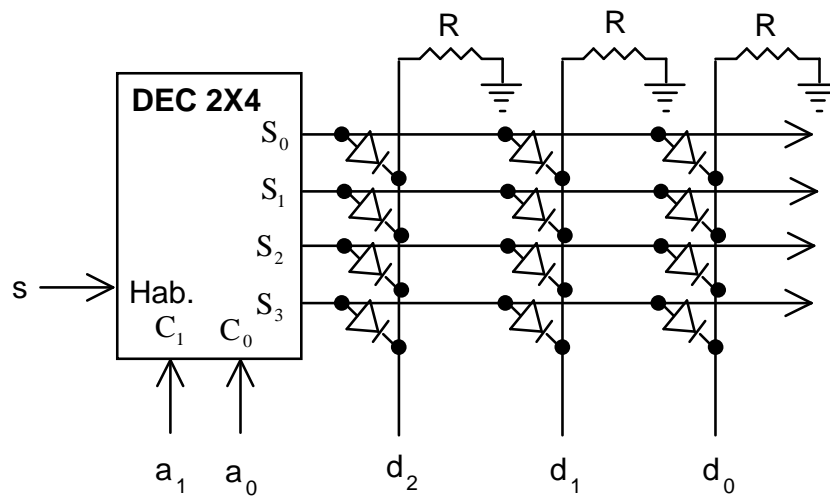


figura 15.15. ROM con una matriz de diodos completa

Primero suponga que se tiene un cero en la entrada s , todas las salidas del decodificador son cero. Por lo tanto, los ánodos de todos los diodos tienen un cero, haciéndolos que se comporte como circuito abierto. Esto ocasiona que todas las líneas de salida d_i estén conectadas a tierra por medio de una resistencia, lo que se debe interpretar como una salida lógica de cero.

Si la entrada s toma el valor de uno, y en las entradas a_1 y a_0 se tiene 00, solo la salida s_0 del decodificador se tendrá el valor de uno, haciendo que los diodos conectados a esta salida se pongan en corto ya que el votaje que tienen en el ánodo es mayor que el votaje del cátodo. Los demás diodos se encuentran en circuito abierto y las salidas d_2 , d_1 y d_0 se tendrá 111. Noté que la resistencia (R) evita que se produzca un corto circuito.

De la misma forma el lector puede comprobar que si cambian los valores de a_1 y a_0 las salidas d_2 , d_1 y d_0 siguen teniendo el mismo valor.

Para hacer que este ROM tenga la información de la tabla 15.6 lo que se tiene que hacer es eliminar los diodos en la posición donde se quiera tener un valor de cero, la figura 15.16 muestra el ROM construido con diodos, el cual almacena la información de la tabla 15.6.

Los ROM tienen dos problemas; el primero es que para que su costo de fabricación no sea muy elevado es necesario que se fabrique una gran cantidad de estos circuitos integrados. El segundo es que si después de construido se detecta un error en la información que se tienen, todos estos circuitos no podrán ser corregidos, teniendo que construir un nuevo circuito con la información correcta.

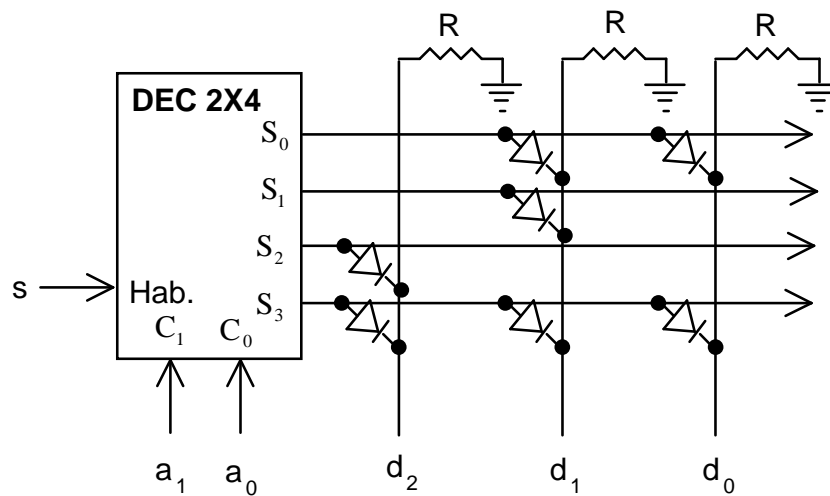


figura 15.16. ROM construido con diodos.

Existen otras formas de tener memorias estáticas, una de ellas son las memorias ROM programables (PROM del inglés "Programmable Read-Only Memory"), en este tipo de memorias el proceso de escribir (programar) la información no se hace al momento de fabricación. Cuando se fabrica un PROM, se construye con la matriz de diodos completa, y el usuario tendrá la facilidad de poder eliminar eléctricamente cualquier diodo de la matriz. Esto es lo que se conoce como programar un PROM, para lo cual se requiere de un equipo especial.

Si al programar un PROM se escribe uno o más bits con error, este circuito ya no será útil. Existe otra memoria estática que permite hacer correcciones como el ROM programable y borrable (EPROM del inglés "Erasable Programmable Read-Only Memory"), en la cual se programa eléctricamente igual que el PROM, pero si se expone éste a luz ultravioleta se borra todo, quedando la matriz de diodos otra vez completa. Estos circuitos integrados se pueden identificar, ya que tienen una ventana para permitir el paso de la luz ultravioleta.

Otra memoria semejante al EPROM es EEPROM (EEPROM del inglés "Electrically Erasable Programmable Read-Only Memory") la diferencia es que ésta se borra y se escribe eléctricamente y normalmente se hace esta operación a nivel de un byte.

15.4.2. Arreglos lógicos programados.

Un arreglo lógico programado o PLA (del inglés "Programmable Array Logic") es un circuito con varias entradas y varias salidas construido dentro de un sólo circuito integrado, por medio de un arreglo de **ANDs** y un arreglo de **ORs**, los cuales se pueden programar.

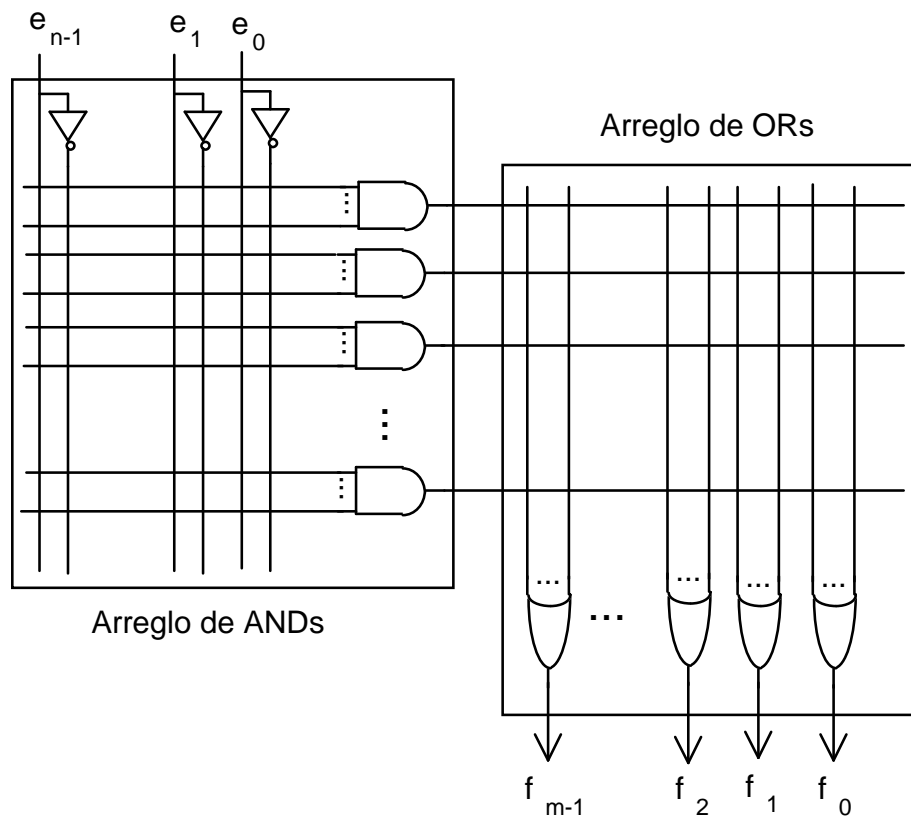


Figura 15.17. Circuito básico de un PLA

La figura 15.17 muestra el circuito básico de un PLA. Al programar cada uno de los arreglos, lo que se hace es definir el número de entradas y que valor de todas las posibles tiene cada uno de los **ANDs** y cada uno de los **ORs**.

Considérese el ejemplo que se presentó en la sección 14.4, donde se definió la siguiente tabla de verdad mostrada de nuevo en la tabla 15.7, con cuatro entradas (**a**, **b**, **c** y **d**) y tres salidas (**f₁**, **f₂** y **f₃**). Para este ejemplo se obtuvieron las siguientes funciones mínima expresadas como suma de productos:

$$\begin{aligned} f_1 &= b'cd' + ab'c' + bcd \\ f_2 &= b'cd' + a'bc' + a'cd \\ f_3 &= b'cd' + ab'c' + a'cd + a'bd \end{aligned}$$

La construcción del PLA con estas funciones requiere de un arreglo de 6 **ANDs** y un arreglo de 3 **ORs** con las conexiones mostradas en la figura 15.18

a	b	c	d	f₁	f₂	f₃
0	0	0	0	0	0	0
0	0	0	1	0	0	0

0	0	1	0	1	1	X
0	0	1	1	0	1	1
0	1	0	0	0	1	0
0	1	0	1	X	1	1
0	1	1	0	0	0	0
0	1	1	1	1	1	X
1	0	0	0	1	0	1
1	0	0	1	X	0	1
1	0	1	0	1	X	1
1	0	1	1	0	0	0
1	1	0	0	0	0	0
1	1	0	1	X	0	X
1	1	1	0	0	0	0
1	1	1	1	1	X	0

Tabla 15.7. Tabla de verdad del ejemplo de la sección 14.4.

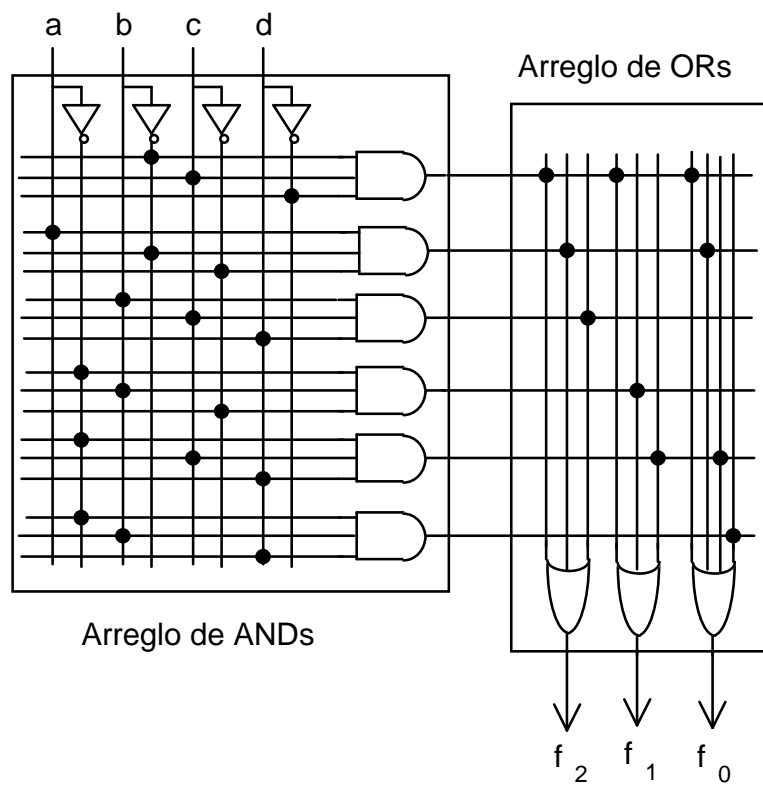


Figura 15.18. Circuito esquemático de un arreglo lógico programado.

El circuito lógico para estas funciones puede ser construido por medio de un circuito NAND-NAND o un circuito AND-OR como se hizo anteriormente, o bien mediante un arreglo lógico programado como se define en esta sección.

Un arreglo lógico programado consiste realmente de un arreglo de bloques **AND** y otro arreglo de bloques **OR** que permiten construir funciones de dos niveles cuando éstas están expresadas como suma de productos. El circuito lógico quedaría como se muestra en la figura 15.18.

La forma de construir el circuito mostrado en la figura 15.18 no tiene ninguna diferencia con la construcción del circuito AND-OR. En efecto, no existe ninguna diferencia lógica pero si existe diferencia en la construcción física.

Un arreglo lógico programado se construye en un sólo circuito integrado usando internamente un arreglo de bloques **AND** y otro de bloques **OR**. La construcción del circuito AND-OR requiere de varios circuitos integrados.

Para indicar que un circuito lógico está construido en base a un arreglo lógico programado se usa una simbología especial para diferenciarlo de un circuito AND-OR. El arreglo lógico programado para este ejemplo se muestra en la figura 15.19. En este circuito las líneas horizontales representan los ANDs, indicado por medio de una conexión (•) las líneas de entrada que tiene cada uno de éstos. Las líneas verticales representan los ORs y de la misma forma se indican las entradas de éstos.

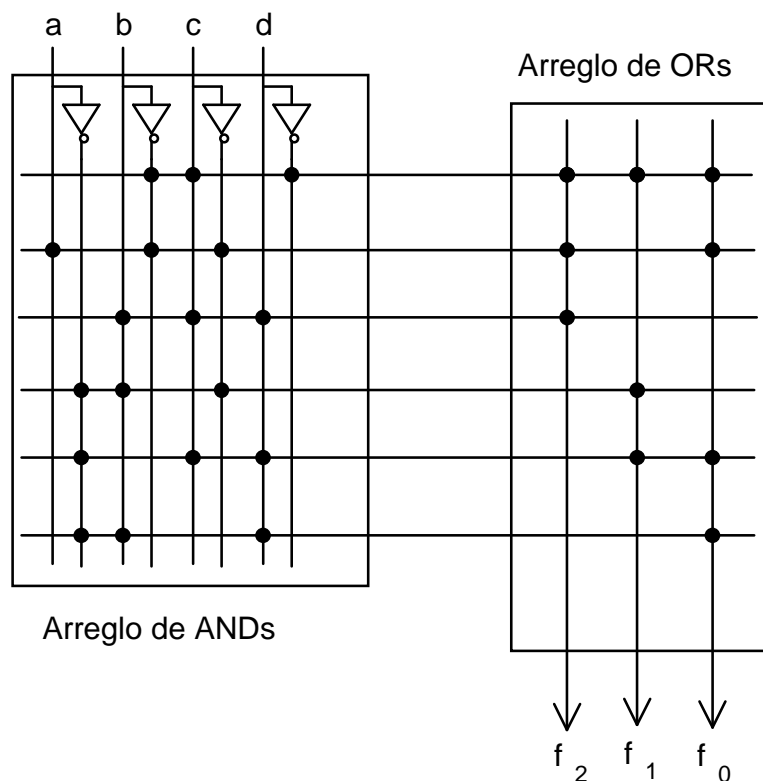


Figura 15.19. Arreglo lógico programado

15.5. RESUMEN

En este capítulo se presentaron algunos de los circuitos lógicos combinatorios más comúnmente utilizados como son los bloques lógicos **EX-OR** (O exclusivo) y **COIN** (o Equivalencia), y los decodificadores y selectores.

También se describió el funcionamiento y la construcción interna de los selectores y decodificadores así como las aplicaciones de estos circuitos y la forma de construir funciones lógicas usando selectores y decodificadores.

Se presentó la forma de usar memorias estáticas para construir funciones lógicas y se describieron las memorias ROM, PROM, EPROM y EEPROM.

Finalmente, se presentaron los circuitos lógicos programables así como las bases de su funcionamiento y construcción. Se trabajaron algunos ejemplos previamente resueltos, construyéndolos con estos circuitos.

15.6. PROBLEMAS

1. Comprobar cada una de las propiedades del **EX-OR** mencionadas en el punto 15.1
2. Construya un decodificador de **3X8** usando dos decodificadores de **2X4** y bloques lógicos.
3. Construya un decodificador de **4X16** usando cinco decodificadores de **2X4**.
4. Construya un selector de **8:1** usando dos selectores de **4:1** y bloques lógicos.
5. Construya un selector de **16:1** usando cinco selectores de **4:1**.
6. Construya el circuito lógico de la siguiente función:

$$f = a'b + ac' + bc$$

usando un decodificador de **3X8** y bloques lógicos **OR**.

7. A continuación se muestra la tabla de verdad para la función F:

W	X	Y	Z	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0

W	X	Y	Z	F
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

- a) Construya el circuito lógico de F usando un selector de **16:1**.
b) Construya el circuito lógico de F usando un selector de **8:1** y bloques lógicos **NOT**.
c) Construya el circuito lógico de F usando un selector de **4:1** y bloques lógicos.
8. Construya con un ROM el decodificador de BCD a 7 segmentos presentado en el problema 7 del capítulo 14.
9. Construya con un PLA las funciones que se minimizaron como suma de productos en la sección 14.4.
10. Construya con un PLA las funciones obtenidas en el problema 9 del capítulo 14.

CAPITULO 16

UNIDAD ARITMÉTICA Y LÓGICA.

Un componente muy importante de la unidad central de proceso es la unidad aritmética y lógica. Esta unidad es un circuito combinatorio que permite realizar las operaciones aritméticas básicas como suma, resta, multiplicación y división, así como operaciones lógicas simples tales como **NOT**, **AND** y **OR**.

En este capítulo se analiza y diseña una unidad aritmética y lógica elemental, la cual no podrá realizar las operaciones de multiplicación y división ya que el diseño del circuito que realiza estas operaciones queda fuera del alcance de este capítulo. Es conveniente aclarar que hay unidades aritméticas que no realizan las operaciones de multiplicación y división mediante circuitos, siendo necesario desarrollar rutinas que las efectúen.

16.1 SUMADOR BÁSICO.

Un sumador básico es un circuito combinatorio que tiene como entrada dos números binarios de **n** bits y da como salida la suma binaria de estos números y un posible acarreo.

Sea **A** y **B** los dos números a sumar, la operación a realizar es la siguiente:

$$\begin{array}{rcccccccc} & c_{n-1} & c_{n-2} & \dots & c_1 & & & \text{(acarreos)} \\ & a_{n-1} & a_{n-2} & \dots & a_1 & a_0 & & \\ + & b_{n-1} & b_{n-2} & \dots & b_1 & b_0 & & \\ \hline c_n & \sum_{n-1} & \sum_{n-2} & \dots & \sum_1 & \sum_0 & & \end{array}$$

Si se analiza la suma de cada uno de los bits en forma independiente, se tendrá que al sumar los bits menos significativos ($a_0 + b_0$) se requiere de un circuito que solamente sume dos bit, el cual genera dos salidas; una para dar la suma de estos bits (\sum_0) y la otra para indicar el posible acarreo que se pueda generar (c_1). Para sumar los bits restantes se requiere de un circuito que pueda sumar tres bits ($a_i + b_i + c_i$) el cual genera como salida la suma (\sum_i) y el posible acarreo (c_{i+1}).

Para llegar al diseño de este sumador, primero se diseñará un circuito que sume dos números de un bit, el cual se conoce como medio sumador, después se diseñará el sumador completo, el cual suma tres bits y, por último, se diseñará el sumador binario de **n** bits.

16.1.1 Medio sumador.

Un medio sumador, mostrado en la figura 16.1, es un circuito que tiene como entradas dos números de un bit y produce como salida la suma y el acarreo de estos números.

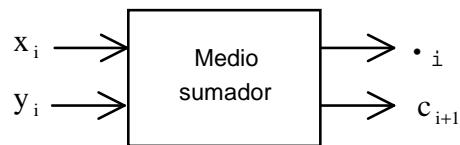


Figura 16.1. Medio sumador

La tabla de verdad de este dispositivo es la siguiente:

x_i	y_i	c_{i+1}	\sum_i
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Las funciones para el acarreo y la suma son las siguientes:

$$c_{i+1} = x_i y_i$$

$$\sum_i = x_i' y_i + x_i y_i' = x_i \oplus y_i$$

Circuito del medio sumador se muestra en la figura 16.2.

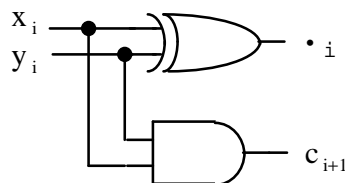


Figura 16.2. Circuito lógico de un medio sumador

Se usará el símbolo mostrado en la figura 16.3 para hacer referencia a un medio sumador, se usan las iniciales HA (del inglés "half adder") para designarlo.

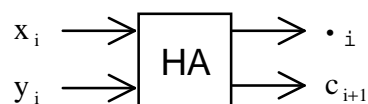


Figura 16.3. Símbolo para un medio sumador

16.1.2. Sumador completo.

Un sumador completo es un circuito que tiene como entradas tres números de un bit y produce como salida la suma y el acarreo de estos números, tal como se muestra en la figura 16.4.

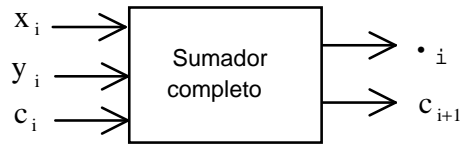


Figura 16.4. Sumador completo

La tabla de verdad de este dispositivo es la siguiente:

x_i	y_i	c_i	c_{i+1}	Σ_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

La función para el acarreo es:

$$c_{i+1} = x_i' y_i c_i + x_i y_i' c_i + x_i y_i c_i' + x_i y_i c_i$$

Expresada usando **EX-OR**.

$$c_{i+1} = (x_i' y_i + x_i y_i') c_i + x_i y_i = (x_i \oplus y_i) c_i + x_i y_i$$

Simplificada.

$$c_{i+1} = y_i c_i + x_i c_i + x_i y_i$$

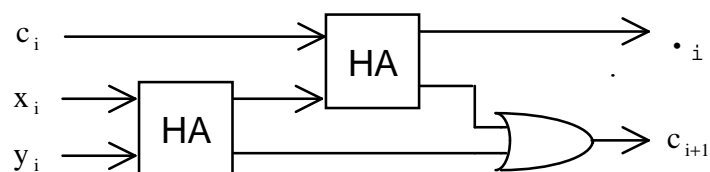
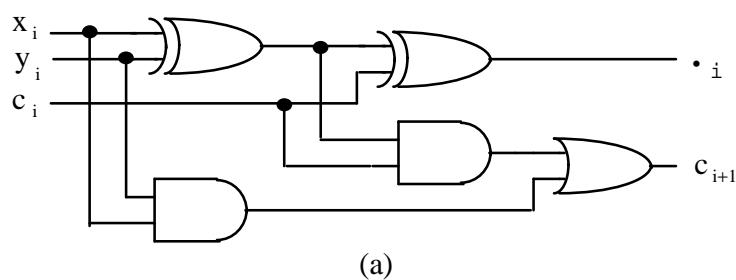
Y la función para la suma es:

$$\Sigma_i = x_i' y_i' c_i + x_i' y_i c_i' + x_i y_i' c_i' + x_i y_i c_i$$

Expresada usando **EX-OR**.

$$\Sigma_i = x_i \oplus y_i \oplus c_i$$

El circuito se puede construir con dos medios sumadores y un OR como se muestra en la figura 16.5 (a). En la figura 16.5 (b) se muestra el circuito usando el símbolo para el medio sumador.



(b)

Figura 16.5. Circuito lógico de un sumador completo

Nótese, que el circuito queda de tres niveles. También se puede construir un circuito de dos niveles como se muestra en la figura 16.6.

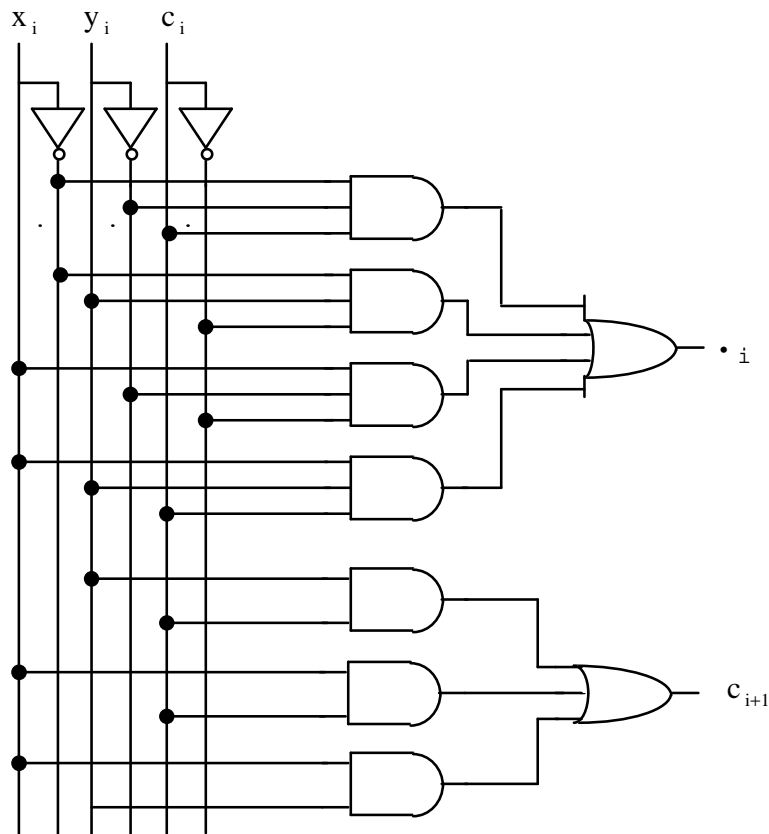


Figura 16.6. Circuito lógico de dos niveles para un sumador completo

Se usara el símbolo mostrado en la figura 16.7 para identificar un sumador completo, se usan las iniciales FA (del inglés "full adder") para designarlo.

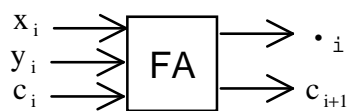


Figura 16.7. Símbolo para un sumador completo

16.1.3 Sumador binario de n bits.

Un sumador binario de n bits se puede construir al unir un medio sumador con $n-1$ sumadores completos. En la figura 16.8 se da un ejemplo de un sumador binario de 4 bits.

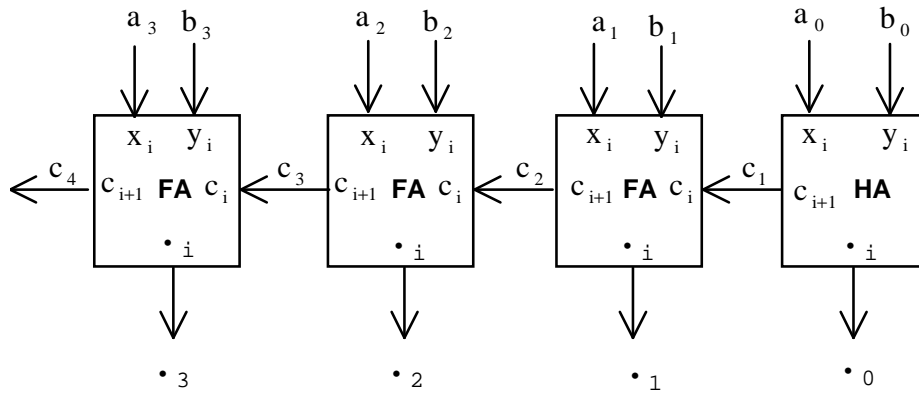


Figura 16.8. Sumador binario de 4 bits

Este sumador también puede ser construido con cuatro sumadores completos como se muestra en la figura 16.9.

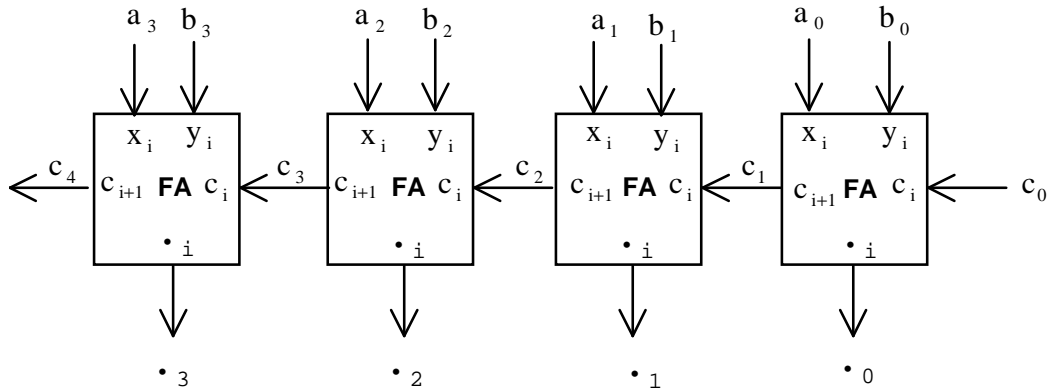


Figura 16.9. Sumador binario de 4 bits con 4 FA

Si a la entrada c_0 se le da el valor de cero, se tendrá el mismo funcionamiento que en el sumador anterior.

Este circuito se puede representar de la forma mostrada en la figura 16.10, como un solo bloque.

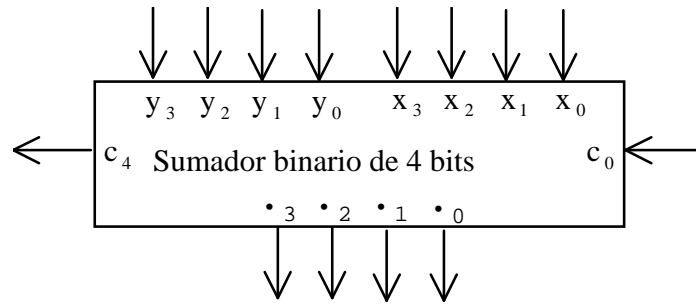


Figura 16.10. Sumador binario de 4 bits como un bloque

Bajo este mismo esquema se pueden construir sumadores binarios de n bits. Observe que se puede construir un sumador binario de **8** bits con dos sumadores binarios de **4** bits, sólo se tendría que conectar la salida c_4 de uno de ellos a la entrada c_0 del otro.

A esta forma de construirlos se les llama sumadores en cascada, ya que, como se puede observar, los acarrees se van propagando de un sumador a otro.

Los sumadores binarios de n bits en cascada tienen el problema de que el número de niveles totales en el circuito aumenta cuando el número de bits aumenta, lo que hace que el tiempo de retardo sea mayor a medida que n crece.

Existe una forma de construir sumadores de n bits donde el tiempo de retardo del circuito no depende de n , esto se logra haciendo que los acarrees no se propaguen, sino que se generen por medio de una función que dependa de las variables de entrada como se describe a continuación. A estos sumadores binarios se les llama sumadores binarios con "look ahead" (del inglés "mirar hacia adelante").

Si se considera que las entradas a un sumador de n bits son el acarreo inicial (c_0), un número **A** de n bits (a_0, a_1, \dots, a_{n-1}) y un número **B** de n bits (b_0, b_1, \dots, b_{n-1}), los acarrees que se generan en cada uno de los sumadores se pueden obtener bajo la siguiente forma:

El acarreo de salida en cada uno de los sumadores completos es igual a uno si el acarreo se genera internamente en el sumador, o si existe un acarreo en la entrada y éste es propagado a la salida.

La generación de un acarreo en la posición i está definida por la función :

$$G_i = a_i b_i$$

La propagación de un acarreo en la posición i , dado que existía un acarreo a la entrada, está definida por la función:

$$P_i = a_i \oplus b_i$$

Por lo tanto el acarreo de salida de la posición i estará definido por la siguiente función:

$$c_{i+1} = G_i + P_i c_i$$

De tal forma que se pueden obtener la función para c_1 de la siguiente forma:

$$c_1 = G_0 + P_0 c_0 = a_0 b_0 + (a_0 \oplus b_0) c_0 = a_0 b_0 + a_0 \bar{b}_0 c_0 + a_0 b_0 \bar{c}_0$$

Simplificando:

$$c_1 = a_0 b_0 + b_0 c_0 + a_0 c_0$$

La función para c_2 se obtiene de la siguiente manera:

$$c_2 = G_1 + P_1 c_1 = a_1 b_1 + (a_1 \oplus b_1) c_1 = a_1 b_1 + a_1 \bar{b}_1 c_1 + a_1 b_1 \bar{c}_1$$

Dado que c_1 no es una variable de entrada, se sustituye c_1 por la función definida anteriormente que si depende de las variables de entrada.

$$c_2 = a_1 b_1 + a_1 \bar{b}_1 (a_0 b_0 + b_0 c_0 + a_0 c_0) + a_1 b_1 \bar{c}_1 (a_0 b_0 + b_0 c_0 + a_0 c_0) =$$

$$a_1 b_1 + a_1 \bar{b}_1 a_0 b_0 + a_1 \bar{b}_1 b_0 c_0 + a_1 \bar{b}_1 a_0 c_0 + a_1 b_1 \bar{a}_0 b_0 + a_1 b_1 \bar{b}_0 c_0 + a_1 b_1 \bar{a}_0 c_0$$

Simplificando:

$$c_2 = a_1 b_1 + b_1 a_0 b_0 + b_1 b_0 c_0 + b_1 a_0 c_0 + a_1 a_0 b_0 + a_1 b_0 c_0 + a_1 a_0 c_0$$

De las dos funciones anteriores se puede observar que si se redefine la función de propagación a $P_i = a_i + b_i$, al hacer el desarrollo de una función, ésta ya queda simplificada, lo cual nos reduce un paso al obtener la función.

Si aplicamos esta nueva definición para la propagación, la función para c_3 se obtiene de la siguiente manera:

$$c_3 = G_2 + P_2 c_2 = a_2 b_2 + (a_2 + b_2) c_2 = a_2 b_2 + a_2 c_2 + b_2 c_2$$

Dado que c_2 no es una variable de entrada, se sustituye c_2 por la función definida anteriormente que si depende de variables de entrada.

$$c_3 = a_2 b_2 + a_2 (a_1 b_1 + b_1 a_0 b_0 + b_1 b_0 c_0 + b_1 a_0 c_0 + a_1 a_0 b_0 + a_1 b_0 c_0 + a_1 a_0 c_0) + b_2 (a_1 b_1 + b_1 a_0 b_0 + b_1 b_0 c_0 + b_1 a_0 c_0 + a_1 a_0 b_0 + a_1 b_0 c_0 + a_1 a_0 c_0)$$

$$c_3 = a_2 b_2 + a_2 a_1 b_1 + a_2 b_1 a_0 b_0 + a_2 b_1 b_0 c_0 + a_2 b_1 a_0 c_0 + a_2 a_1 a_0 b_0 + a_2 a_1 b_0 c_0 + a_2 a_1 a_0 c_0 + b_2 a_1 b_1 + b_2 b_1 a_0 b_0 + b_2 b_1 b_0 c_0 + b_2 b_1 a_0 c_0 + b_2 a_1 a_0 b_0 + b_2 a_1 b_0 c_0 + b_2 a_1 a_0 c_0$$

De igual forma se puede definir la función para cualquiera de los demás acarreos. Como se puede observar, cualquier acarreo se puede obtener con un circuito de dos niveles. Por lo tanto un sumador de n bits se puede construir con un circuito de tres niveles, ya que el primer nivel de la función Σ en un sumador sólo depende de variables de entrada.

16.2. Sumadores/restadores binarios.

Como se demostró en los capítulos 7 y 9, la resta de dos números binarios en cualquiera de sus representaciones se puede obtener por medio de una suma. Por lo tanto, un sumador/restador binario se construye en base a un sumador pero se controlan sus entradas para que realice la operación que se desee.

Antes de diseñar un sumador/restador se analizarán diferentes circuitos para obtener el complemento a uno y complemento a dos de un número binario.

El circuito para obtener el complemento a uno de un número binario A de n bits es muy simple, ya que todos los bits cambian de valor. El circuito que obtiene el complemento a uno se muestra en la figura 16.11.

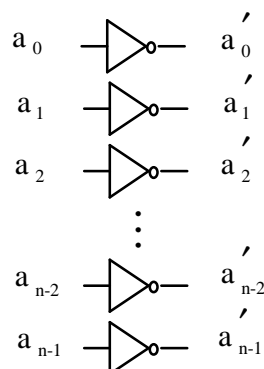


Figura 16.11. Circuito para obtener el complemento a uno.

Considere un circuito que tiene como entradas un número A de n bits ($a_{n-1} a_{n-2} \dots a_2 a_1 a_0$) y una entrada de control m . La salida es otro número S de n bits ($s_{n-1} s_{n-2} \dots s_2 s_1 s_0$). Este número de salida S puede ser el mismo número A que se tiene en la entrada, o bien el complemento a uno de dicho número A , dependiendo del valor de la entrada de control (m). Para diseñar este circuito basta con analizar un sólo bit, ya que todos los bits se comportan de la misma forma.

La tabla de verdad para el bit i sería la siguiente:

a_i	m	s_i
0	0	0
0	1	1
1	0	1

1	1	0
---	---	---

Donde se obtiene que $s_i = m \oplus a_i$.

El circuito que realiza esta función se muestra en la figura 17.12.

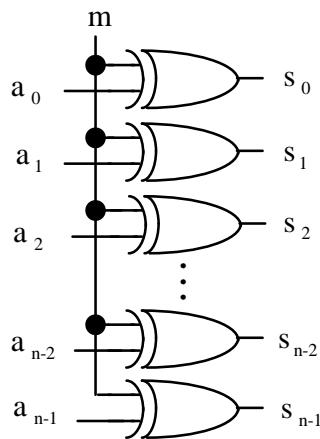


Figura 17.12. Circuito que obtiene el mismo número que se le alimenta a la entrada, o su complemento a uno.

Para el diseño del circuito que obtiene el complemento a dos de un número no se puede analizar cada bit en forma independiente ya que, recordando la regla que se aplica para encontrar el complemento a dos de un número, se debe analizar este número de derecha a izquierda y sólo cambian los bits después del primer uno que se encuentra.

Se puede diseñar un circuito donde se aplique la regla anterior, tal como se muestra en la figura 16.13.

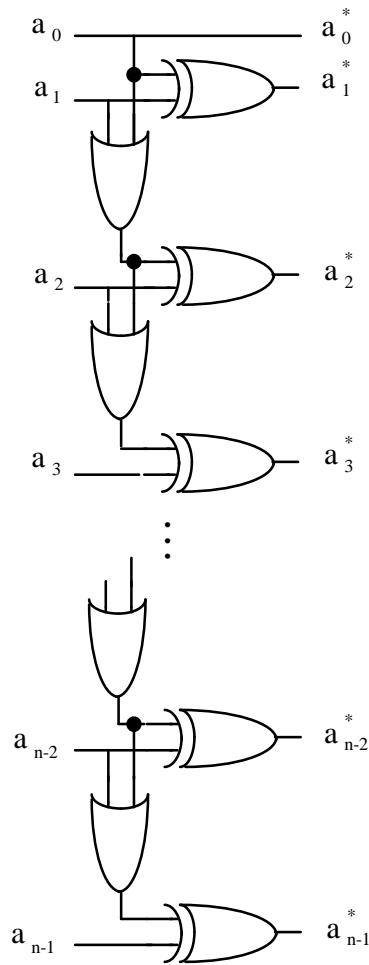


Figura 16.13. Circuito que obtiene el complemento a dos de un número.

Como se puede observar en el circuito, a_0 nunca cambia, a_1 invierte su valor solamente si a_0 tiene el valor de uno, a_2 invierte su valor si a_0 o a_1 valen uno y, en forma general, a_i invierte su valor siempre y cuando exista al menos un bit que valga uno en $a_0 \dots a_{i-1}$.

El número de niveles de este circuito aumenta al aumentar n . El circuito se puede construir con dos niveles si se sustituyen los **ORs** que se forman con más de un **OR** por **ORs** de más entradas. Otra forma de construir un circuito equivalente de dos niveles es obtener las funciones para cada una de las salidas a partir de la tabla de verdad y simplificarlas.

Recuérdese que el complemento a dos de un número también puede ser obtenido sumándole uno al complemento a uno de dicho número.

Se deja como ejercicio para el lector el diseño de un circuito que reciba como entradas un número binario A de n bits y una entrada de control m , y genere como salida un número binario S de n bits que contenga ya sea el mismo número A que se tiene en las entradas, o bien, el complemento a dos del número A .

16.2.1 Sumador/restador de números positivos sin signo.

Para diseñar un sumador/restador de números positivos sin signo se revisarán los resultados que se obtuvieron en el capítulo 7 para sumar o restar dos números positivos.

Sean A y B dos números positivos de n bits y R el número que resulta al hacer una operación con estos números. El número R se limitará también a n bits.

Si al realizar la operación $A + B$ resulta un acarreo en c_n , se indicará como un desborde; si no se genera un acarreo en c_n , en R se tendrá el resultado correcto.

La resta se puede hacer de dos formas:

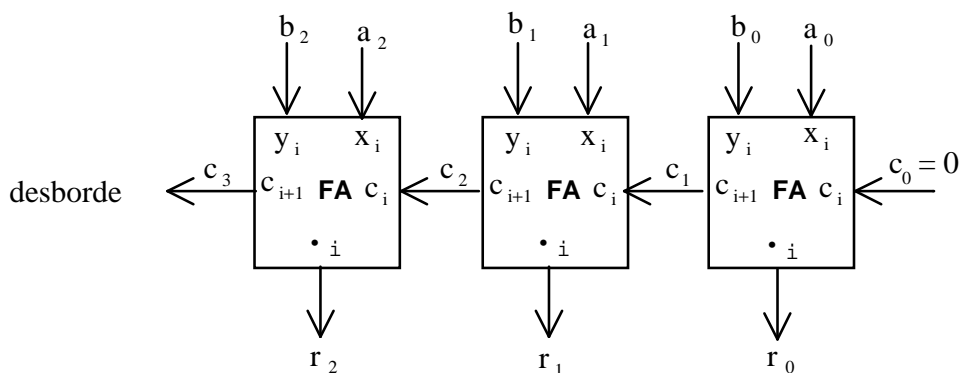
Si se usan complementos a dos para hacerla, se tienen las siguientes reglas:

- Para realizar la operación $R = A - B$, se efectúa la operación $A + B^*$.
- Si se produce un acarreo fuera de las n posiciones, dicho acarreo se desecha y en R queda el resultado correcto.
- Si no se produce un acarreo fuera de las n posiciones, para obtener el resultado correcto se obtiene el complemento a la base del resultado y se indica que es negativo.

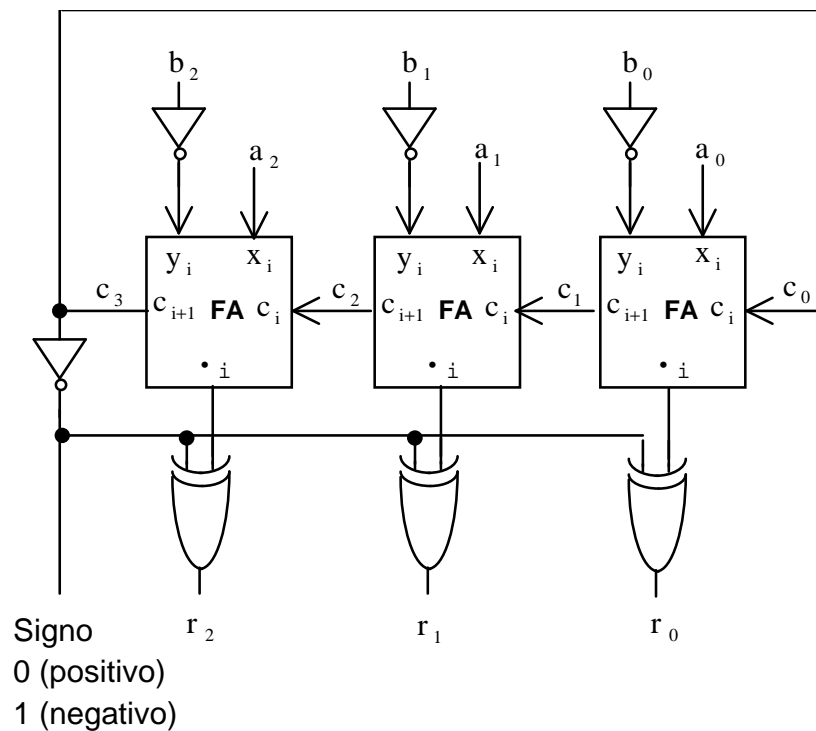
Si se usan complementos a uno para efectuarla, se tienen las siguientes reglas:

- Para realizar la operación $A - B$, se efectúa la operación $A + \bar{B}$.
- Si se produce un acarreo fuera de las n posiciones, dicho acarreo se desecha y, para obtener el resultado correcto se le suma uno al resultado de la operación.
- Si no se produce un acarreo fuera de las n posiciones, el resultado correcto se obtiene sacando el complemento a la base disminuida de la magnitud del resultado, indicando que es negativo.

En la figura 16.14 se presentan los circuitos para sumar (a) y restar (b) números positivos sin signo con 3 bits.



(a) sumador de 3 bits.



(b) restador de 3 bits utilizando complementos a uno.

Figura 16.14. Circuitos para sumar y restar números positivos sin signo, d 3 bits.

Obsérvese que al generarse un acarreo de salida, éste pasa al acarreo de entrada para hacer la corrección y no se complementa el resultado. Al no generarse un acarreo de salida el resultado se complementa a uno.

Si se construyen un sumador y un restador en el mismo circuito, se tendría el diseño mostrado en la figura 17.15.

16.2.2 Sumador /restador de números enteros en representación de magnitud y signo.

Al tener números positivos y negativos, la suma se puede convertir en una resta y la resta se puede convertir en una suma, ya que esto depende de los signos de los números.

Para diseñar un sumador/restador de números enteros en representación de magnitud y signo se usarán los resultados que se obtuvieron en el capítulo 9. Se usarán complementos a uno para realizar las restas.

Sean **A** y **B** dos números enteros de **5** bits representados en magnitud y signo, por lo tanto los bits más significativos (**a₄** y **b₄**) corresponden a los signos de dichos números.

Sea **R** un número entero de **5** bits, que representa el resultado de la operación entre **A** y **B**.

Primero se diseñará un sumador. La operación aritmética a realizar depende de los signos de los números **A** y **B** tal como se muestra en la tabla 16.1.

a₄	b₄	Operación
0	0	$R = A + B$
0	1	$R = A - B$
1	0	$R = B - A$
1	1	$R = A + B$

Tabla 16.1. Operación a realizar dependiendo de los signos de los números.

Para simplificar el circuito que se diseñará, sólo se realizará la operación **A - B**; por lo tanto, para realizar la operación **B - A** se efectuará la operación **A - B**, invirtiendo después el signo del resultado.

Dependiendo de los signos de los números **A** y **B**, y del acarreo de salida, se puede determinar cuando se deben complementar **B** y **R** (**B** y **R**). También se puede determinar el signo del resultado (**r₄**); cuando se debe realizar una corrección (**c₀**), y cuando ocurrirá un desborde (**V**), tal como se muestra en la tabla 16.2.

a₄	b₄	c₄	Operación	\overline{B}	\overline{R}	r₄	c₀	V
0	0	0	$A + B$	0	0	0	0	0
0	0	1	$A + B$	0	0	0	0	1
0	1	0	$A - B$	1	1	1	0	0
0	1	1	$A - B$	1	0	0	1	0
1	0	0	$A - B$	1	1	0	0	0
1	0	1	$A - B$	1	0	1	1	0
1	1	0	$A + B$	0	0	1	0	0
1	1	1	$A + B$	0	0	1	0	1

Tabla 16.2. operación a realizar, complementar B y R, signo del resultado, corrección y desborde, en función de los signos de **A** y **B** y del acarreo de salida.

Las funciones son las siguientes:

$$\begin{aligned}\bar{B} &= a_4 \oplus b_4 \\ \bar{R} &= a_4' b_4 c_4' + a_4 b_4' c_4' = (a_4 \oplus b_4) c_4' \\ r_4 &= a_4 c_4 + b_4 c_4' \\ c_0 &= a_4' b_4 c_4 + a_4 b_4' c_4 = (a_4 \oplus b_4) c_4 \\ V &= a_4' b_4' c_4 + a_4 b_4 c_4 = (a_4 \oplus b_4)' c_4\end{aligned}$$

El circuito presentado en la figura 16.16 suma dos números enteros de **5** bits, representados en magnitud y signo.

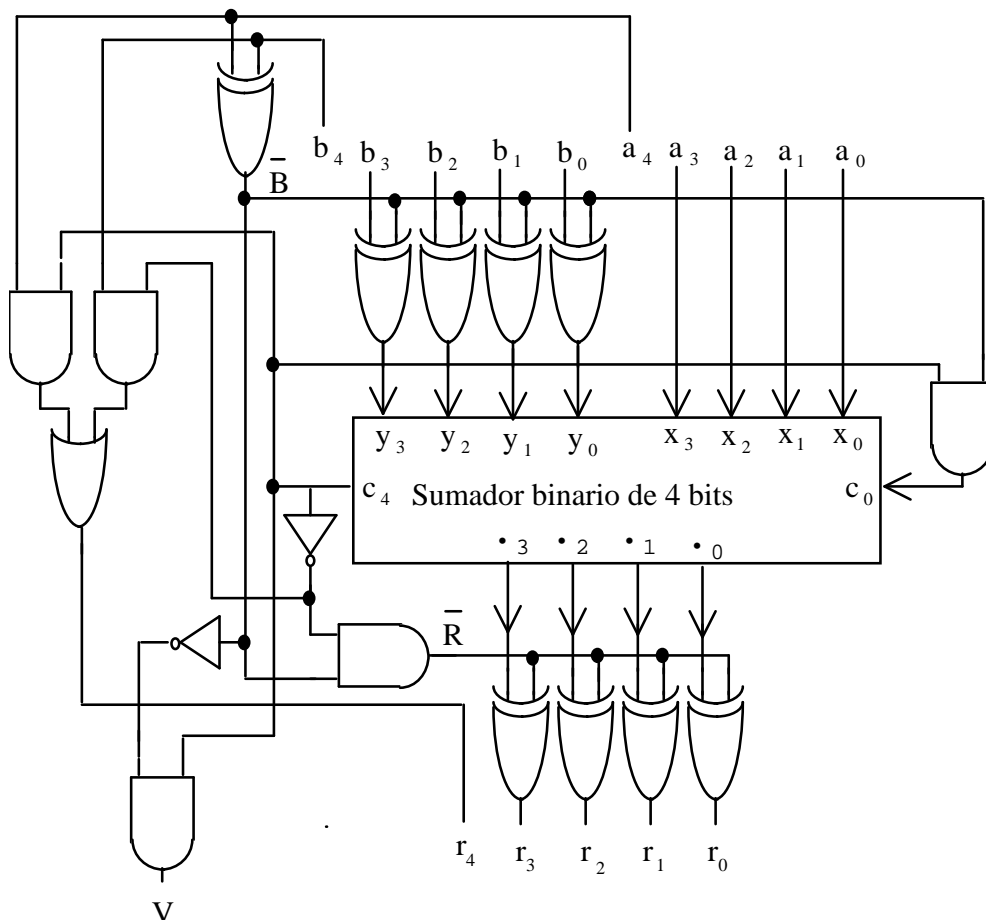


Figura 16.16. Sumador para dos números enteros de **5** bits, representados en magnitud y signo.

Se puede construir un circuito sumador/restador de números enteros de 5 bits en representación de magnitud y signo añadiendo una entrada de control, denominada **T**, para indicar la operación a realizar (**T** = 0 indica operación de suma y **T** = 1, operación de resta). La única modificación que requiere el circuito anterior será complementar el signo de **B** cuando se efectúe una operación de resta, como se muestra en la figura 16.17.

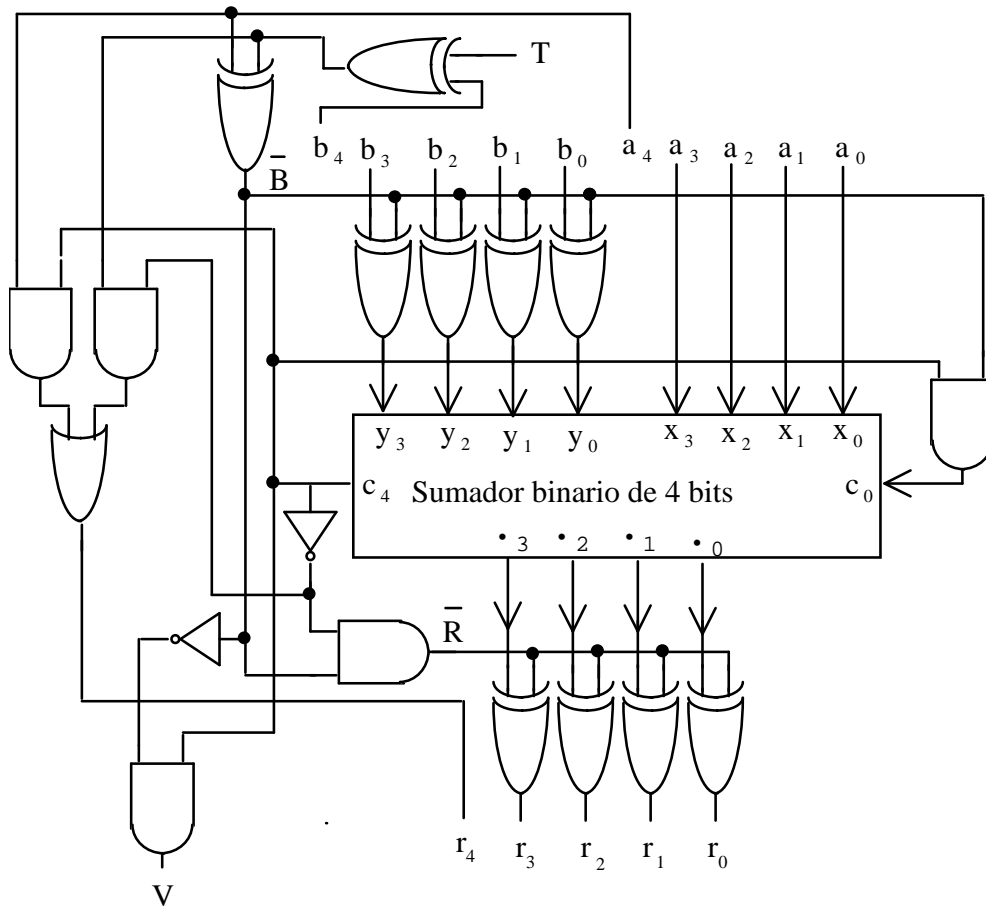


Figura 16.17. Sumador/restador para dos números enteros de 5 bits, representados en magnitud y signo.

Otra forma de construir este circuito es determinar cuando se deben complementar **B** y **R** (\bar{B} y \bar{R}), cómo obtener el signo del resultado (r_4), cuando se debe efectuar una corrección (c_0), y cuando ocurre un desborde (V), tal como se muestra en la siguiente tabla 16.3.

T	a ₄	b ₄	c ₄	Operación	\overline{B}	\overline{R}	r ₄	c ₀	V
0	0	0	0	A + B	0	0	0	0	0
0	0	0	1	A + B	0	0	0	0	1
0	0	1	0	A - B	1	1	1	0	0
0	0	1	1	A - B	1	0	0	1	0
0	1	0	0	A - B	1	1	0	0	0
0	1	0	1	A - B	1	0	1	1	0
0	1	1	0	A + B	0	0	1	0	0
0	1	1	1	A + B	0	0	1	0	1
1	0	0	0	A - B	1	1	1	0	0
1	0	0	1	A - B	1	0	0	1	0
1	0	1	0	A + B	0	0	0	0	0
1	0	1	1	A + B	0	0	0	0	1
1	1	0	0	A + B	0	0	1	0	0
1	1	0	1	A + B	0	0	1	0	1
1	1	1	0	A - B	1	1	0	0	0
1	1	1	1	A - B	1	0	1	1	0

Tabla 16.3. operación a realizar, complementar B y R, signo del resultado, corrección y desborde, en función de los signos de **A** y **B**, del acarreo de salida y la entrada T.

De igual forma que en el caso anterior, se determinan las funciones y se construye el circuito.

16.2.3 Sumador/restador de números enteros en representación de complementos a dos

Para diseñar el sumador/restador de números enteros en representación de complementos a dos, considérese lo siguiente:

Sean **A** y **B** dos números enteros de **4** bits, representados en complementos a dos.

Sea **R** un número entero de **4** bits que representa el resultado de la operación entre **A** y **B**.

En el capítulo 9 se obtuvieron las siguientes reglas para sumar y restar números enteros en representación en complementos a dos:

- Para sumar dos números enteros en representación en complementos a dos, se suman los n bits directamente.
- Si el acarreo sobre el bit más significativo y el acarreo fuera de los n bits son diferentes, ocurre un desborde y el resultado no es correcto.

- Si el acarreo sobre el bit más significativo y el acarreo fuera de los n bits son iguales, el resultado es correcto y queda en la representación en complementos a 2. Cualquier acarreo fuera de los n bits se deshecha.
- Para hacer la resta de $A - B$ se hace la suma de $A + (-B)$. Si el número B es el más negativo esta operación no se puede realizar ya que no se puede representar el número $-B$.

En la figura 16.18 se muestra el circuito para sumar/restar números de 4 bits.

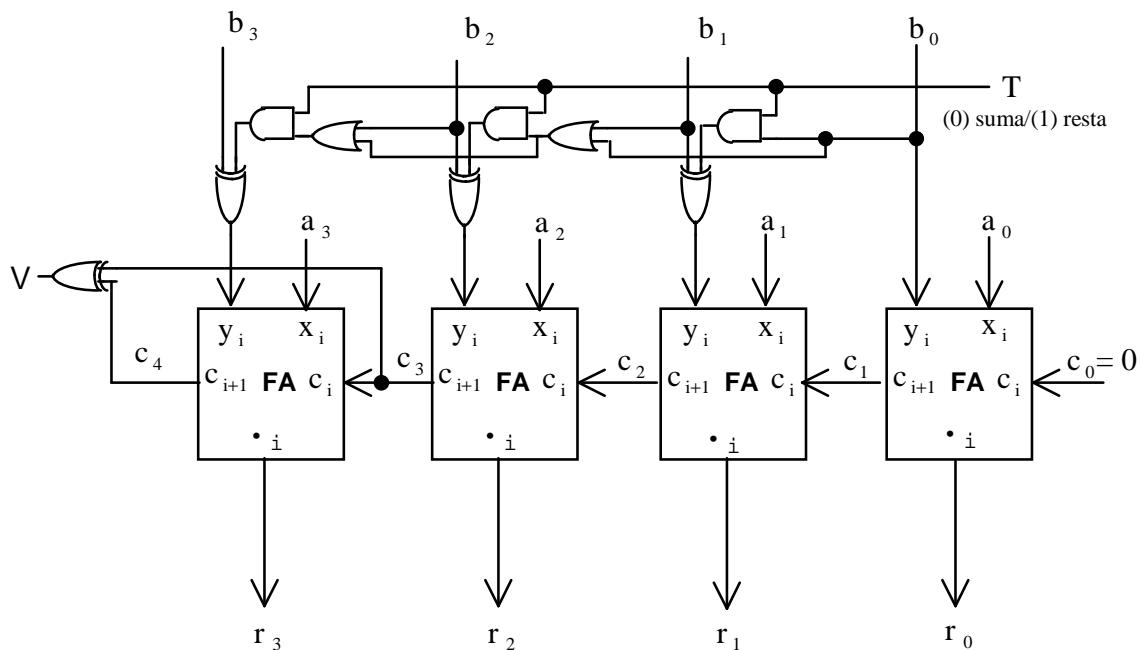


Figura 16.18. Sumador/restador de números de 4 bits en representación de complementos a dos.

Obsérvese que cuando la entrada T tiene el valor de uno, se obtiene el complemento a dos de B , esto se realiza aplicando la regla para obtener el complemento a dos.

Otra forma de determinar el complemento a dos de B es obtener primero el complemento a uno de éste y sumarle uno. El siguiente circuito es un sumador/restador diseñado con esta idea.

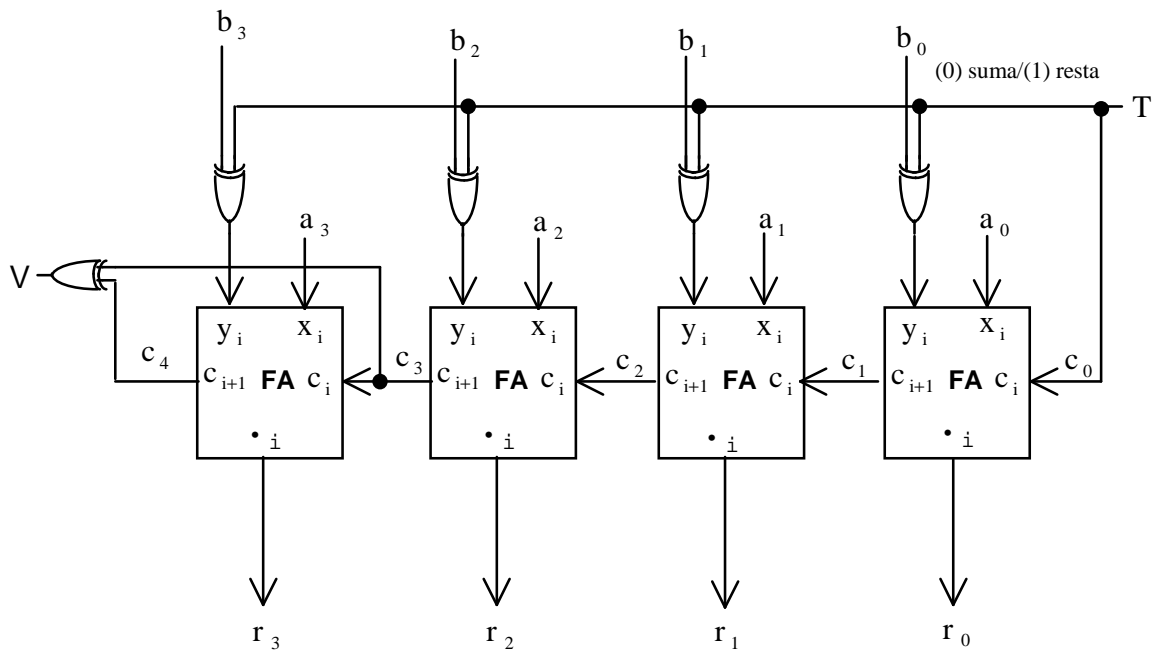


Figura 16.19. Otro circuito sumador/restador de números de 4 bits en representación de complementos a dos.

Nótese que el acarreo de entrada (c_0) tiene el valor de uno cuando la entrada **T** tiene el valor de uno, lo cual indica que se quiere realizar una resta y sirve para sumar uno al complemento a uno que se obtiene de **B**.

16.2.4 Sumador/restador de números enteros en representación de complemento a uno

Para diseñar el sumador/restador de números enteros en representación de complementos a uno, considérese lo siguiente:

Sean **A** y **B** dos números enteros de 4 bits, representados en complementos a uno.

Sea **R** un número entero de 4 bits que representa el resultado de la operación entre **A** y **B**.

En el capítulo 9 se dedujeron las siguientes reglas para sumar y restar números enteros en representación en complementos a uno:

- Para sumar dos números enteros en representación en complementos a 1 se suman los n bits directamente.
- Si se genera un acarreo fuera de los n bits, se desecha dicho acarreo y se efectúa una corrección sumándole uno al resultado obtenido.

- Si el acarreo sobre el bit más significativo y el acarreo fuera de los n bits que se generen al hacer la suma y la corrección en los casos necesarios son diferentes, ocurre un desborde y el resultado no es correcto.
- Si el acarreo sobre el bit más significativo y el acarreo fuera de los n bits son iguales, el resultado es correcto y queda en la representación en complementos a 1.
- Para hacer la resta de $A - B$ se hace la suma de $A + (-B)$.

Un circuito para sumar/restar números de 4 bits en representación de complementos a uno se muestra en la figura 16.20.

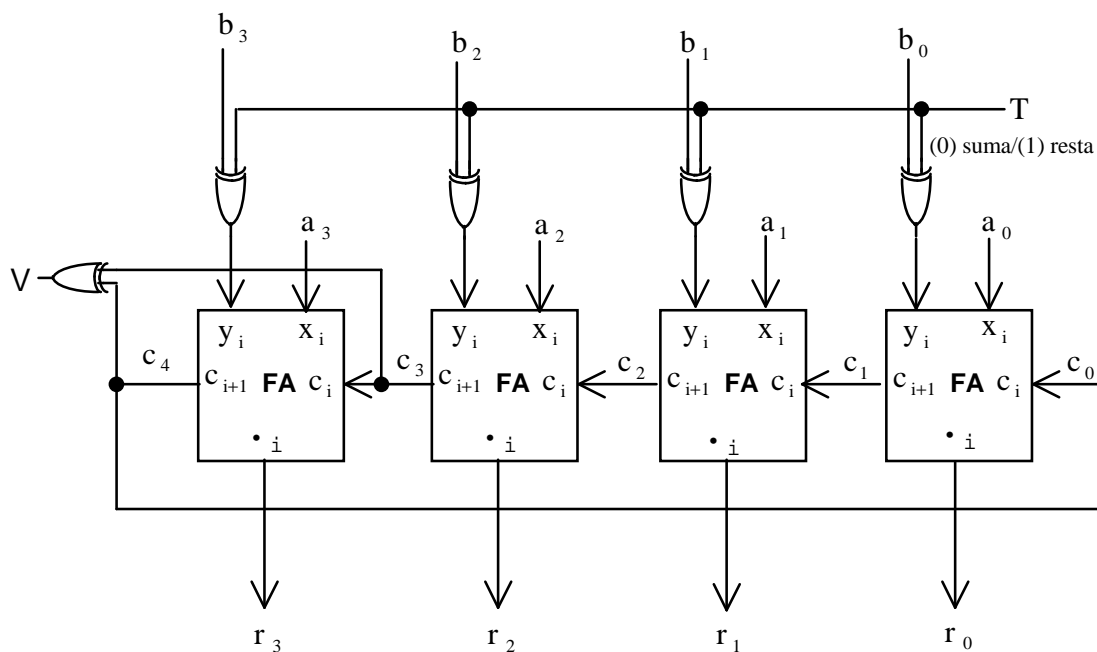


Figura 16.20. Circuito sumador/restador de números de 4 bits en representación de complementos a uno.

16.2.5 Sumador/restador de números enteros representados en

BCD

Para diseñar este sumador/restador primero se construirá un circuito que sume dos números **BCD** de un sólo dígito (**A** y **B**). La suma de estos dos números se puede realizar con un sumador binario de 4 bits, teniendo en cuenta que cuando la suma exceda de 9 se deberá realizar una corrección y se generará un acarreo decimal como se explicó en el capítulo 9. Este circuito se muestra en la figura 16.21.

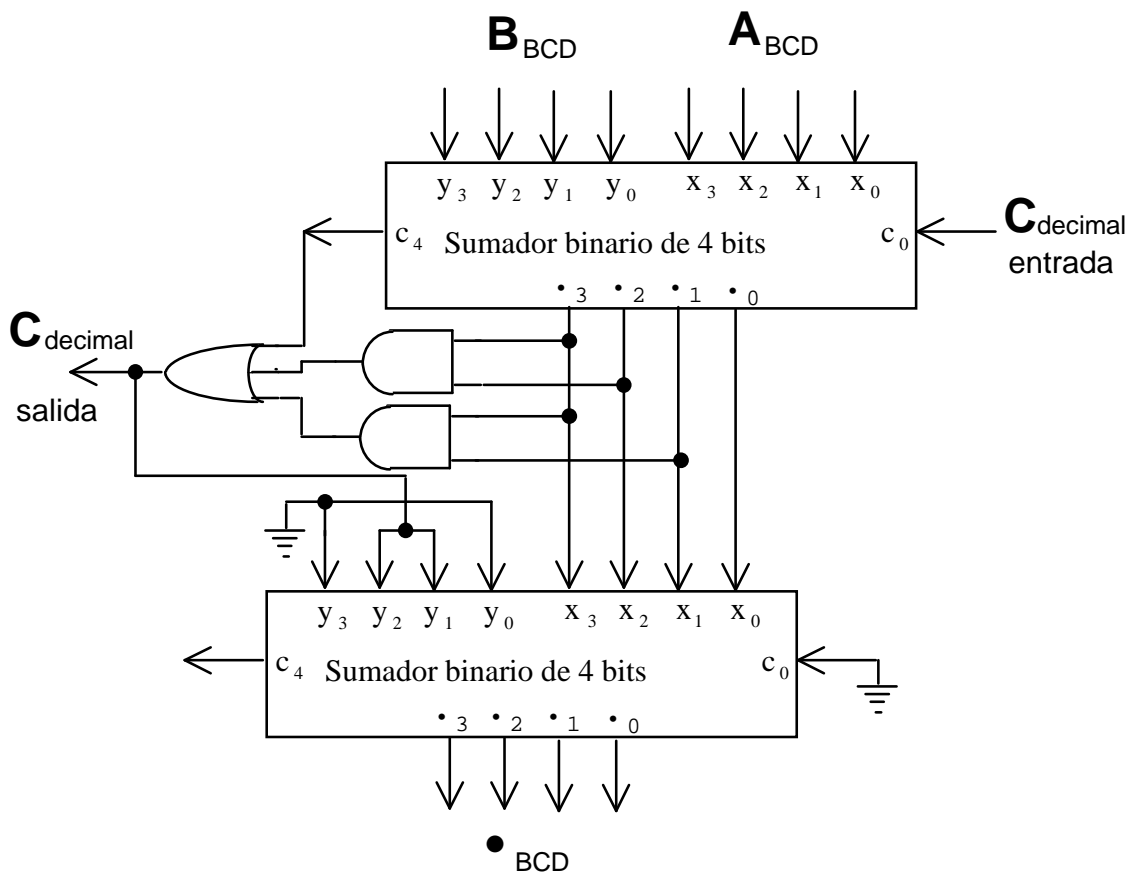


Figura 16.21. Sumador de dos números BCD de un dígito.

El símbolo $\text{---}\text{---}\text{---}$ indica una conexión a tierra, que equivale a tener un cero lógico en la entrada.

El sumador de dos dígitos BCD de la figura 16.21 se representará como un solo bloque tal como se muestra en la figura 16.22.

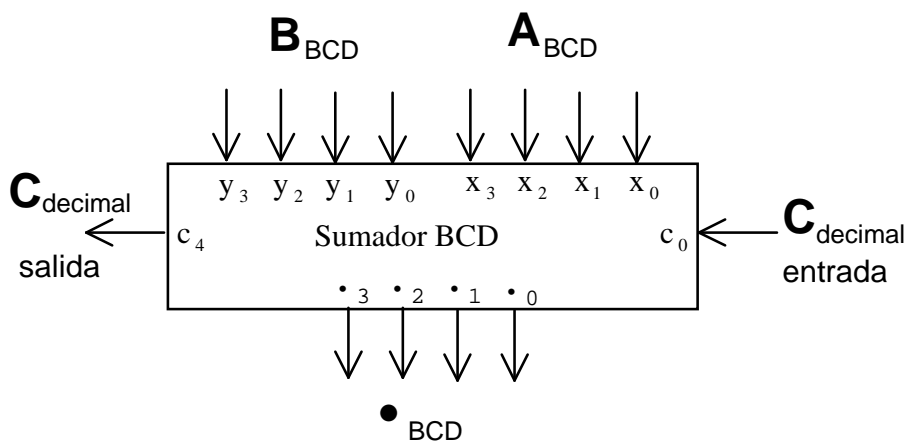


Figura 16.22. Representación del sumador de dos números BCD de un dígito.

Para diseñar el restador se usarán complementos a la base disminuida; por lo tanto, primero se diseñará un circuito que permita obtener el complemento a nueve de un número BCD.

Si se quiere construir un circuito donde se tenga como entrada un número **N** en **BCD** de **n** dígitos y en la salida otro número **X** de **n** dígitos que, dependiendo de una entrada de control (**m**) contenga el complemento a nueve de **N** (si **m=1**) o el número **N** (si **m=0**), basta con analizar un sólo dígito, ya que todos los dígitos se comportarán de la misma forma.

El complemento a nueve de un dígito se puede hacer con un sumador binario de cuatro bits ($9+N^*$) como se muestra en la figura 16.23.

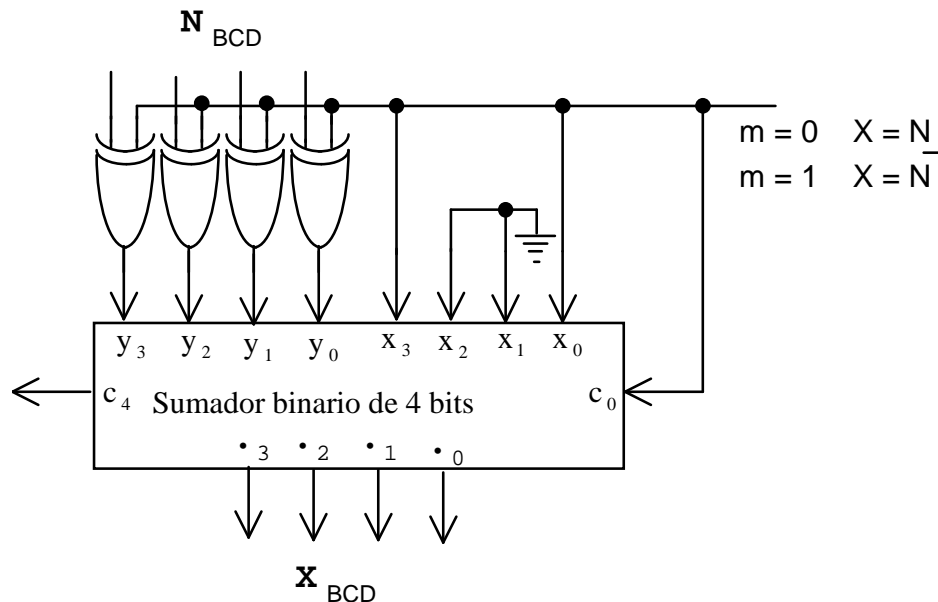


Figura 16.23. Circuito para obtener el complemento a 9 de un número BCD de un dígito, o el mismo número.

Observe que, dependiendo de la señal **m**, se obtendrá el resultado de la operación $(0+N)$, o $(9+N^* = 9-N)$ en la salida del sumador.

Para diseñar el restador de números BCD considérese lo siguiente:

Sean **A** y **B** dos números BCD de 2 dígitos.

Sea **R** un número BCD de dos dígitos que representa el resultado de la resta entre **A** y **B**.

En el capítulo 9 se determinaron las siguientes reglas para restar números BCD:

- Para realizar la operación $A - B$, se efectúa la operación $A + \bar{B}$.
- Si se produce un acarreo fuera de las n dígitos, dicho acarreo se desecha y para obtener el resultado correcto se le suma uno al resultado de la operación.
- Si no se produce un acarreo fuera de las n dígitos, para obtener el resultado correcto se obtiene el complemento a la base disminuida de la magnitud del resultado y se indica que es negativo.

Un posible circuito para efectuar la resta se presenta en la figura 16.24.

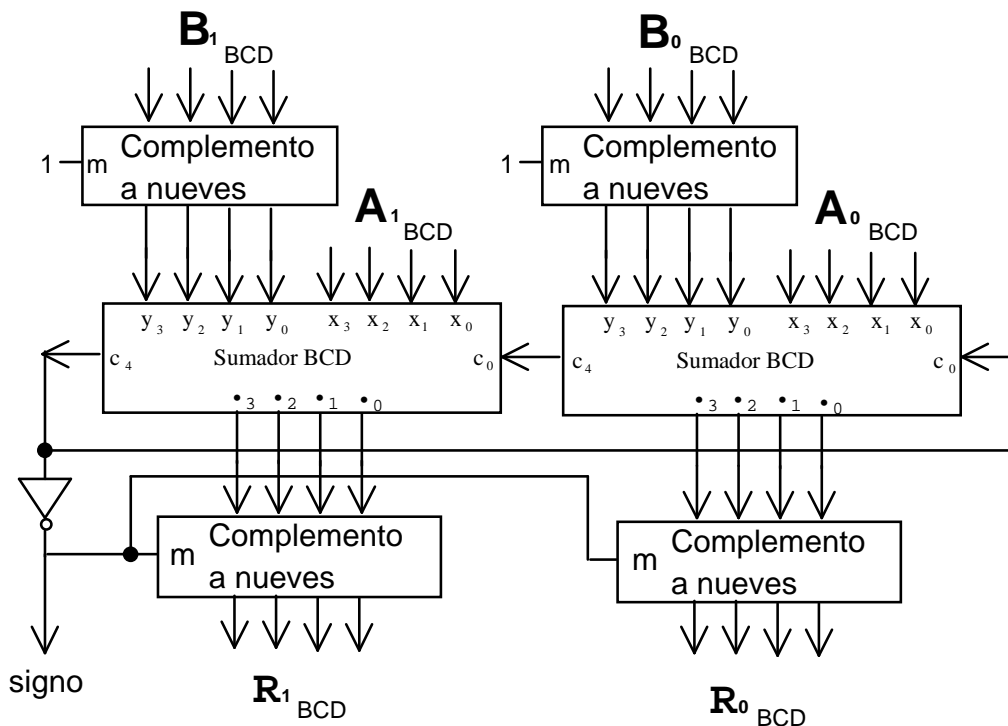


Figura 16.24. Circuito para restar dos números BCD de dos dígitos.

16.3 Diseño de la unidad aritmética

La unidad aritmética es un circuito combinatorio que tiene como entradas dos números binarios de n bits y como salida otro número binario de n bits que se obtiene mediante una operación aritmética entre los números de entrada. La operación a realizar se indica por medio de líneas de control en las entradas.

Sean A y B los dos números de entrada y R el número de salida. Se considera que los números pueden ser positivos o negativos y se utiliza la representación en complementos a dos.

El circuito mostrado en la figura 16.25 es una unidad aritmética que, dependiendo de las entradas de control Z_0 y Z_1 , puede realizar diferentes operaciones.

En las entradas y_i puede estar presente un valor de cero, el valor de **B**, el complemento a uno de **B** (que se debe de interpretar como **-B - 1**, ya que se está trabajando con números representados en complementos a dos), o un valor de puros unos, que representa al valor de **-1**.

Las operaciones que realiza esta unidad aritmética dependen de los valores que se tengan en las entradas de control y en el acarreo de entrada. Estas operaciones se indican en la tabla 16.5.

Z_1	Z_0	c_0	Operación
0	0	0	$R = A$
0	0	1	$R = A + 1$
0	1	0	$R = A + B$
0	1	1	$R = A + B + 1$
1	0	0	$R = A - B - 1$
1	0	1	$R = A - B$
1	1	0	$R = A - 1$
1	1	1	$R = A$

Tabl1 16.5. Operaciones que realiza la unidad aritmética.

Recuérdese que si el acarreo de entrada vale uno, el resultado de la operación se incrementa en uno.

Otras salidas de la unidad aritmética son una serie de bits que se llaman banderas, los cuales indican las condiciones obtenidas al efectuar la operación. Las banderas mas comunes son las siguientes:

- cero (Z), vale uno si el resultado es cero
- negativo (N), vale uno si el resultado es negativo
- acarreo final (C), vale uno si hay un uno en el acarreo final
- desborde (V), vale uno si en la operación ocurrió desborde.

Si se ponen estas salidas en la unidad aritmética que se analizó, el circuito quedará como se indica en la figura 16.26.

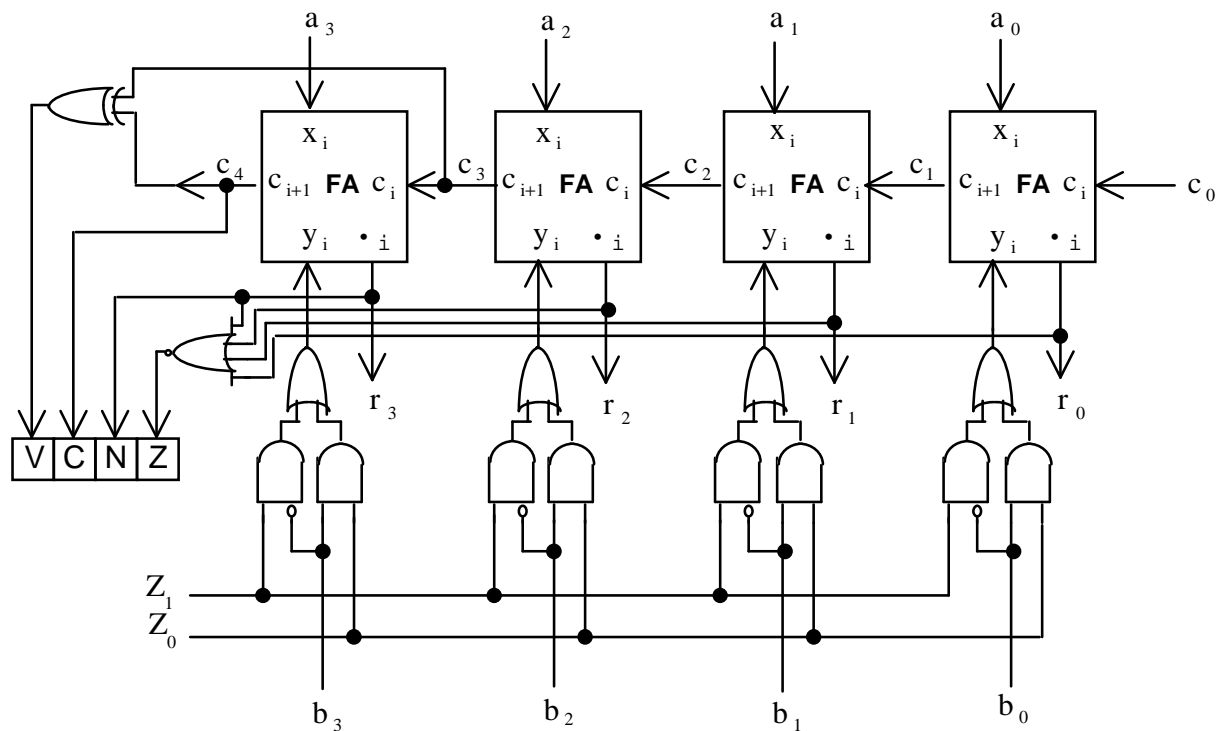


Figura 16.26. Unidad aritmética descrita en la tabla 16.5.

16.4 Diseño de la unidad lógica.

La unidad lógica es un circuito combinatorio que tiene como entradas dos números binarios de n bits y como salida otro número binario de n bits que se obtiene mediante una operación lógica entre los números de entrada. La operación lógica a realizar se indica por medio de líneas de control en las entradas.

Sean **A** y **B** los dos números de entrada y sea **R** el número que representa el resultado de la operación lógica en la salida. La definición de las operaciones básicas podría ser la siguiente:

NOT	:	$r_i = \text{NOT } a_i$
AND	:	$r_i = a_i \text{ AND } b_i$
OR	:	$r_i = a_i \text{ OR } b_i$

Nótese que cada bit del número de salida se obtiene mediante la operación lógica realizada con los bits correspondientes de los números de entrada.

Otra operación lógica muy común es el **EX-OR**, su definición sería la siguiente:

EX-OR	:	$r_i = a_i \text{ EX-OR } b_i$
--------------	---	--------------------------------

Se definirá una unidad lógica cuyo esquema general se muestra en la figura 16.27. Las operaciones que debe realizar son las descritas por la tabla 16.6.

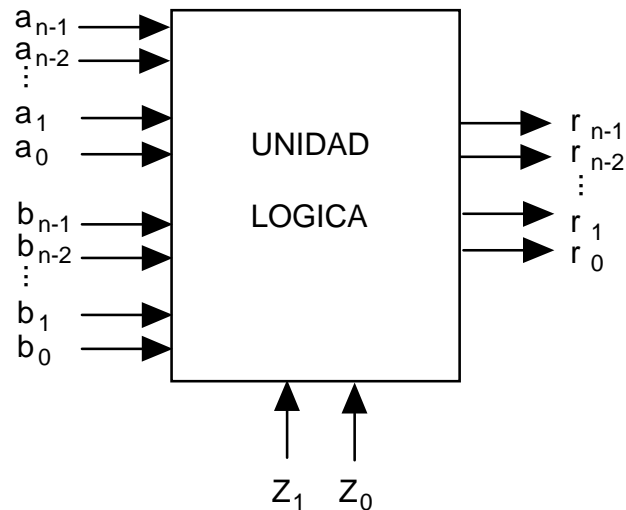


Figura 16.27 Esquema de la unidad lógica.

Dependiendo de las dos entradas de control, las funciones que debe realizar son las siguientes:

Z_1	Z_0	Función
0	0	NOT A
0	1	A AND B
1	0	A OR B
1	1	A EX-OR B

Tabla 16.6. Operaciones de la unidad lógica, dependiendo de sus entradas de control.

Un posible circuito de esta unidad lógica para $n = 3$, se presenta en la figura 16.28.

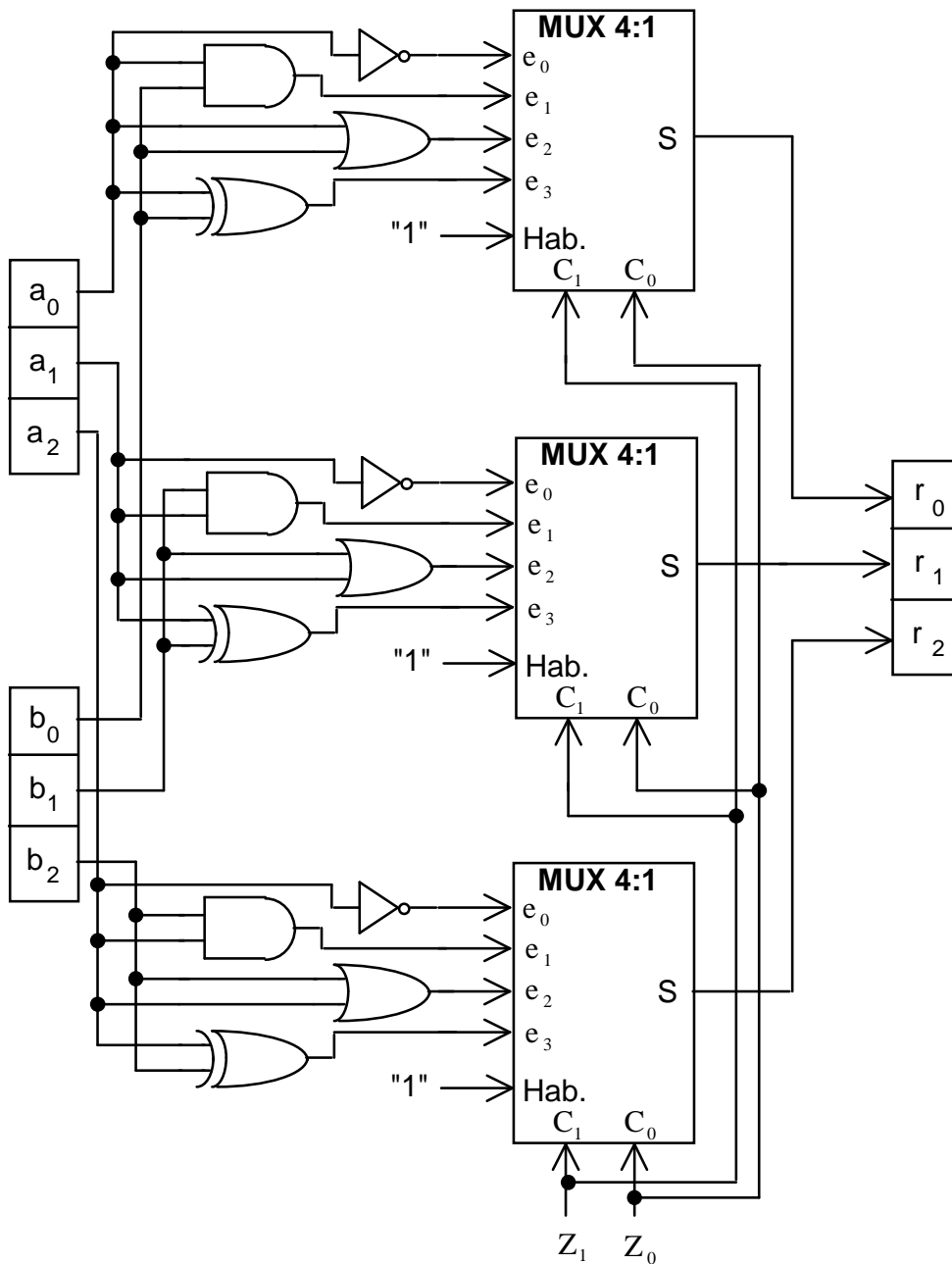


Figura 16.28. Circuito de la unidad lógica descrita en la tabla 16.6.

16.5 Diseño de la unidad aritmética y lógica (ALU)

La unidad aritmética y lógica es un circuito que realiza tanto operaciones aritméticas como lógicas. Una forma simple de construirla es diseñar en forma independiente la unidad aritmética y la unidad lógica como se explicó en los puntos anteriores, y por medio de una línea de control externa indicar el resultado que se desea. Esta idea se bosqueja en la figura 16.29.

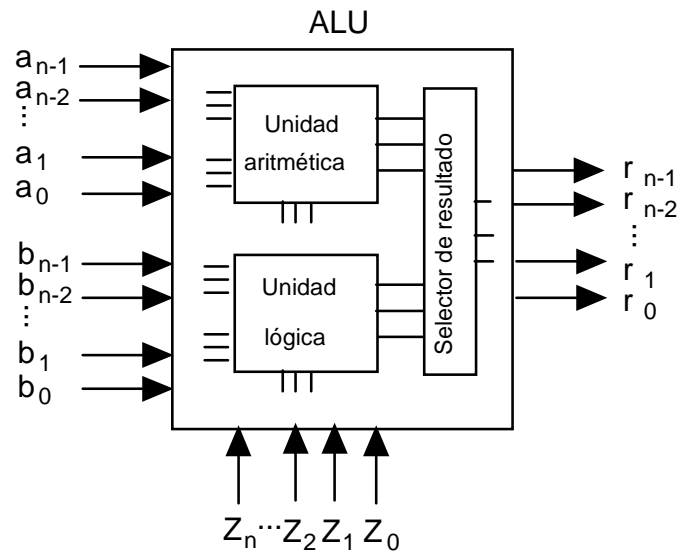


Figura 16.29. Unidad aritmética y lógica construida por separado.

En este tipo de diseño ambas unidades realizan la operación, pero sólo uno de los resultados se obtiene en la salida.

Otra forma de construir la unidad aritmética y lógica es tomar como base la unidad aritmética haciéndole una modificación para evitar la propagación de acarreo entre los sumadores al efectuara una operación lógica. Sean **A** y **B** los dos números de entrada y **R** el número de salida. Para las operaciones aritméticas se considera que los números pueden ser positivos o negativos y se utiliza la representación en complementos a dos. Si los números fueran de 2 bits, el circuito base de la unidad aritmética y lógica sería el mostrado en la figura 16.30.

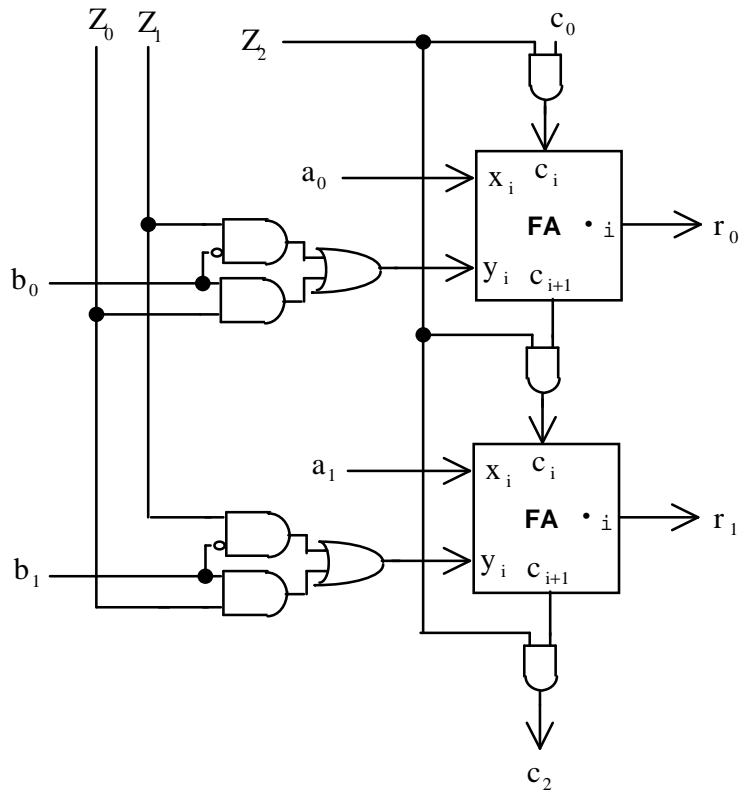


Figura 16.30. Unidad aritmética y lógica de dos bits.

Nótese que cuando la entrada de control Z_2 tiene el valor de 1, este circuito realiza las mismas operaciones que la unidad aritmética descrita en el punto 16.3. Pero cuando la entrada de control Z_2 tiene el valor de 0, todos los acarrees de entrada en los **FA** tienen el valor de 0, haciendo que el circuito se comporte como unidad lógica, ya que el resultado de r_i sólo depende de los valores de a_i y b_i , ya que para este caso el valor de r_i es igual a $a_i \oplus b_i \oplus 0 = a_i \oplus b_i$.

Para analizar el funcionamiento del circuito cuando Z_2 tiene el valor de 0, es necesario determinar los valores que se tendrán en las entradas x_i y y_i , e interpretar el resultado que se obtiene en cada una de las combinaciones de las entradas de control Z_1 y Z_0 . En la tabla 16.7 se muestra este análisis.

Z_1	Z_0	x_i	y_i	Salida	Operación
0	0	a_i	0	$r_i = a_i \oplus 0$	$R = A$
0	1	a_i	b_i	$r_i = a_i \oplus b_i$	$R = A \text{ EX-OR } B$
1	0	a_i	b_i'	$r_i = a_i \oplus b_i'$	$R = A \text{ EX-OR } B'$
1	1	a_i	1	$r_i = a_i \oplus 1$	$R = \text{NOT } A$

Tabla 16.7. Operaciones de la unidad aritmética y lógica.

Como se puede observar, las operaciones lógicas **A EX-OR B** y **NOT A** se obtienen sin necesidad de modificar mucho el circuito de la unidad aritmética. Para que este circuito realice las operaciones lógicas **A OR B** y **A AND B** se deben modificar las entradas que se generan en las x_i y y_i en las combinaciones de las variables de control donde se quiera que se efectúen estas operaciones.

Si en la combinación donde $Z_2=0$, $Z_1=0$ y $Z_0=0$ se quiere tener la operación lógica **A OR B**, es necesario determinar los valores que deben tener x_i y y_i para que se realice esta operación.

Los valores que se tienen en las entradas x_i y y_i para esta combinación son:

$$x_i = a_i$$

$$y_i = 0$$

éstos pueden ser modificados si se les agrega un termino a las funciones de x_i y y_i donde pueda tener un valor diferente de cero para esta combinación como se indica a continuación:

$$x_i = a_i + Z_2'Z_1'Z_0'k_1$$

$$y_i = Z_0b_i + Z_1b_i' + Z_2'Z_1'Z_0'k_2$$

nótese que los valores de x_i y y_i sólo se modifican en la combinación de las variables de control que se está analizando, los nuevos valores que se tienen son:

$$x_i = a_i + k_1$$

$$y_i = 0 + k_2 = k_2.$$

Ya que se tiene la forma de poder modificar las entradas de x_i y y_i , el siguiente paso será determinar que valor deben tener estas entradas para realizar la operación.

Primero se analizará si se puede realizar la operación lógica **OR** modificando sólo x_i . Haciendo esto, los valores que se tienen son:

$$x_i = a_i + k_1$$

$$y_i = 0$$

$$r_i = (a_i + k_1) \oplus 0 = a_i + k_1$$

si se hace $k_1 = b_i$ se tiene la operación **OR**.

Por lo tanto, si se cambian las funciones de \mathbf{x}_i por $\mathbf{x}_i = \mathbf{a}_i + \mathbf{Z}_2' \mathbf{Z}_1' \mathbf{Z}_0' \mathbf{b}_i$, la operación **A OR B** ya se puede efectuar.

Si se analiza el caso donde se modifica sólo \mathbf{y}_i el lector puede comprobar que no hay un valor de \mathbf{k}_2 para el cual se realice la operación **OR**.

Si en la combinación donde $\mathbf{Z}_2=0$, $\mathbf{Z}_1=1$ y $\mathbf{Z}_0=0$ se quiere tener la operación lógica **A AND B**, se deben determinar los valores de \mathbf{x}_i y \mathbf{y}_i que es necesario tener para que se efectúe esta operación.

Los valores que se tienen en las entradas \mathbf{x}_i y \mathbf{y}_i para esta combinación son:

$$\begin{aligned}\mathbf{x}_i &= \mathbf{a}_i \\ \mathbf{y}_i &= \mathbf{b}_i'\end{aligned}$$

al igual que en el caso anterior, éstos pueden ser modificados si se les agrega un término a las funciones de \mathbf{x}_i y \mathbf{y}_i donde pueda tener un valor diferente de cero para esta combinación tal como se indica a continuación:

$$\begin{aligned}\mathbf{x}_i &= \mathbf{a}_i + \mathbf{Z}_2' \mathbf{Z}_1' \mathbf{Z}_0' \mathbf{b}_i + \mathbf{Z}_2' \mathbf{Z}_1 \mathbf{Z}_0' \mathbf{k}_1 \\ \mathbf{y}_i &= \mathbf{Z}_0 \mathbf{b}_i + \mathbf{Z}_1 \mathbf{b}_i' + \mathbf{Z}_2' \mathbf{Z}_1 \mathbf{Z}_0' \mathbf{k}_2\end{aligned}$$

los nuevos valores que se tienen para esta combinación son:

$$\begin{aligned}\mathbf{x}_i &= \mathbf{a}_i + \mathbf{k}_1 \\ \mathbf{y}_i &= \mathbf{b}_i' + \mathbf{k}_2.\end{aligned}$$

Si se analiza el caso donde sólo se modifica \mathbf{y}_i , se tendrá:

$$\begin{aligned}\mathbf{x}_i &= \mathbf{a}_i \\ \mathbf{y}_i &= \mathbf{b}_i' + \mathbf{k}_2 \\ \mathbf{r}_i &= (\mathbf{a}_i) \oplus (\mathbf{b}_i' + \mathbf{k}_2) = \mathbf{a}_i' \mathbf{b}_i' + \mathbf{a}_i' \mathbf{k}_2 + \mathbf{a}_i \mathbf{b}_i \mathbf{k}_2'\end{aligned}$$

no hay un valor de \mathbf{k}_2 que haga que en \mathbf{r}_i se tenga la operación **AND**.

Para el caso donde sólo se modifica \mathbf{x}_i , se obtendría:

$$\begin{aligned}\mathbf{x}_i &= \mathbf{a}_i + \mathbf{k}_1 \\ \mathbf{y}_i &= \mathbf{b}_i' \\ \mathbf{r}_i &= (\mathbf{a}_i + \mathbf{k}_1) \oplus (\mathbf{b}_i') = \mathbf{a}_i' \mathbf{b}_i' \mathbf{k}_1' + \mathbf{a}_i \mathbf{b}_i + \mathbf{b}_i \mathbf{k}_1\end{aligned}$$

si \mathbf{k}_2 vale \mathbf{b}_i' , en \mathbf{r}_i se tendrá la operación **AND**.

Por lo tanto, la función de \mathbf{x}_i debe ser igual a $\mathbf{x}_i = \mathbf{a}_i + \mathbf{Z}_2' \mathbf{Z}_1' \mathbf{Z}_0' \mathbf{b}_i + \mathbf{Z}_2' \mathbf{Z}_1 \mathbf{Z}_0' \mathbf{b}_i'$ para que además se realicen las operaciones que se tenían: la operación **A OR B** y la operación **A AND B**.

En la figura 16.31 se presenta el circuito completo. En la tabla 16.8 se indican todas las operaciones que realiza.

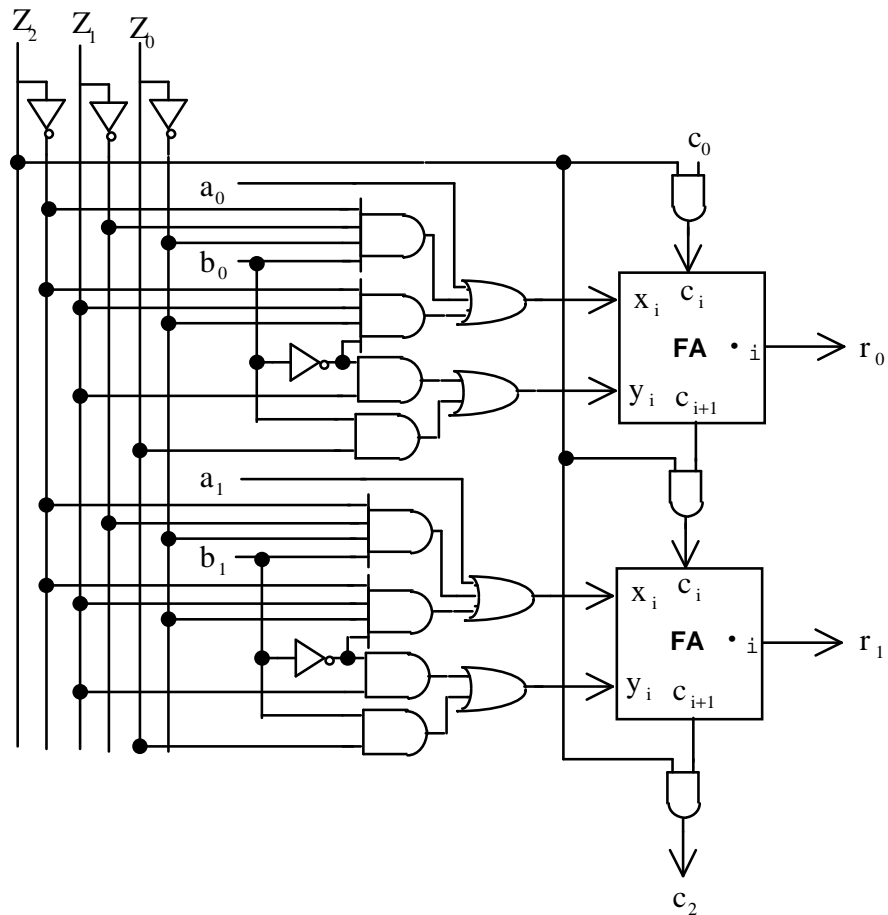


Figura 16.31. Circuito completo de la unidad aritmética y lógica.

Z_2	Z_1	Z_0	c_0	Operación
0	0	0	X	$R = A \text{ OR } B$
0	0	1	X	$R = A \text{ EX-OR } B$
0	1	0	X	$R = A \text{ AND } B$
0	1	1	X	$R = \text{NOT } A$
1	0	0	0	$R = A$
1	0	0	1	$R = A + 1$
1	0	1	0	$R = A + B$
1	0	1	1	$R = A + B + 1$
1	1	0	0	$R = A - B - 1$
1	1	0	1	$R = A - B$
1	1	1	0	$R = A - 1$

1	1	1	1	$R = A$
---	---	---	---	---------

Tabla 16.8. Operaciones de la unidad aritmética y lógica de la figura 16.31.

Como se puede observar en la tabla, las primeras cuatro combinaciones son operaciones lógicas, donde la variable c_0 tiene el valor de **X**, lo que indica que puede tomar el valor de cero o uno, y se realiza la misma operación. Las últimas ocho combinaciones corresponden a operaciones aritméticas.

Otra forma de diseñar la ALU es primero determinar las operaciones que se desean tener y posteriormente determinar los valores que deben tener las entradas x_i y y_i . En la tabla 16.9 se muestran las operaciones.

Z_2	Z_1	Z_0	Operación
0	0	0	$R = A \text{ OR } B$
0	0	1	$R = A \text{ EX-OR } B$
0	1	0	$R = A \text{ AND } B$
0	1	1	$R = \text{NOT } A$
1	0	0	$R = A + 1$
1	0	1	$R = A + B$
1	1	0	$R = A - B$
1	1	1	$R = A - 1$

Tabla 16.9. Operaciones de la unidad aritmética y lógica.

En la tabla 16.10 se indican los valores que se deben tener en x_i , y_i y c_0 para que se realicen estas operaciones. Se supone que cuando Z_2 es cero no se tiene propagación de acarreo, y cuando éste es uno, si se tiene.

Z_2	Z_1	Z_0	Operación	x_i	y_i	c_0
0	0	0	$R = A \text{ OR } B$	$a_i + b_i$	0	X
0	0	1	$R = A \text{ EX-OR } B$	a_i	b_i	X
0	1	0	$R = A \text{ AND } B$	$a_i b_i$	0	X
0	1	1	$R = \text{NOT } A$	a_i	1	X
1	0	0	$R = A + 1$	a_i	0	1
1	0	1	$R = A + B$	a_i	b_i	0
1	1	0	$R = A - B$	a_i	b_i'	1
1	1	1	$R = A - 1$	a_i	1	0

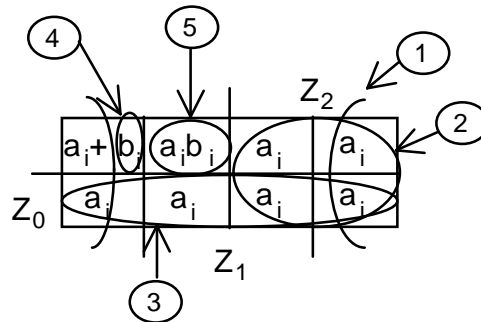
Tabla 16.10. Valores de x_i , y_i y c_0 en función de las operaciones.

Recuerde que la operación $R = A - B$ se hace sumándole al número **A** el complemento a dos de **B**, el cual se obtiene sumándole uno al complemento a uno de **B**. La operación $R = A \text{ AND } B$

puede realizarse de dos formas, una es como se indica en esta tabla 16.10, $r_i = (a_i b_i) \oplus 0$, la otra es como se obtiene en el ALU anterior: $r_i = (a_i + b_i)' \oplus b_i$.

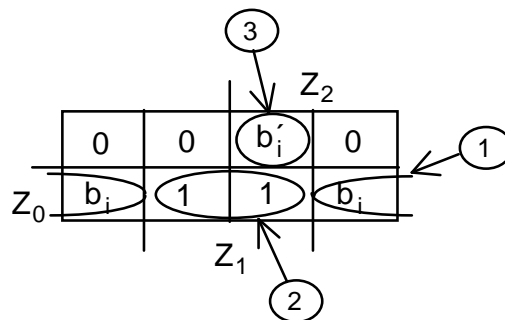
Las funciones mínimas para x_i , y_i y c_0 se pueden obtener con la ayuda de mapas de Karnaugh como se indican a continuación:

Para x_i



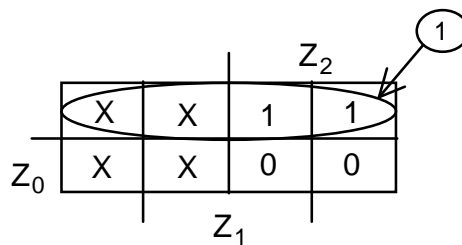
$$x_i = Z_1' a_i + Z_2 a_i + Z_0 a_i + Z_2' Z_1' Z_0' b_i + Z_2' Z_1 Z_0' a_i b_i$$

Para y_i



$$y_i = Z_1' Z_0 b_i + Z_1 Z_0 + Z_2 Z_1 Z_0' b_i'$$

Y para c_0



$$c_0 = Z_0'$$

Al tener las funciones lo único que falta por hacer es construirlas en la ALU. Esto se presenta en la figura 16.32.

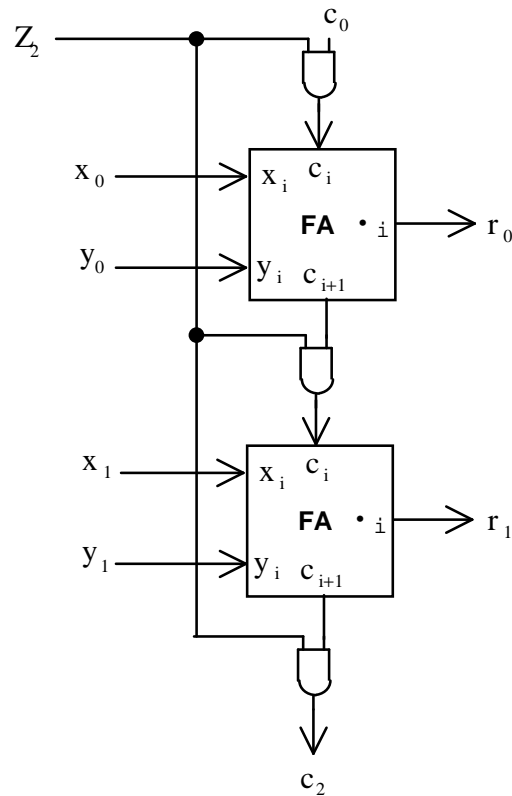


Figura 16.32. Unidad aritmética y lógica

16.6 RESUMEN

En esta sección se presentó inicialmente el medio sumador que permite realizar la suma de dos bits, dando como salidas la suma de estos bits y un posible acarreo para la siguiente posición. También se realizó el diseño del circuito para construirlo. Después se diseñó el sumador completo, el cual permite sumar dos bits más un posible acarreo de la posición anterior, dando como resultado la suma y el posible acarreo para la siguiente posición.

Se presentó la manera de usar un medio sumador y varios sumadores completos para sumar números de varios bits. Usando este análisis, se diseñaron posteriormente sumadores/restadores de números en diferentes formatos (magnitud y signo, complementos a uno y complementos a dos).

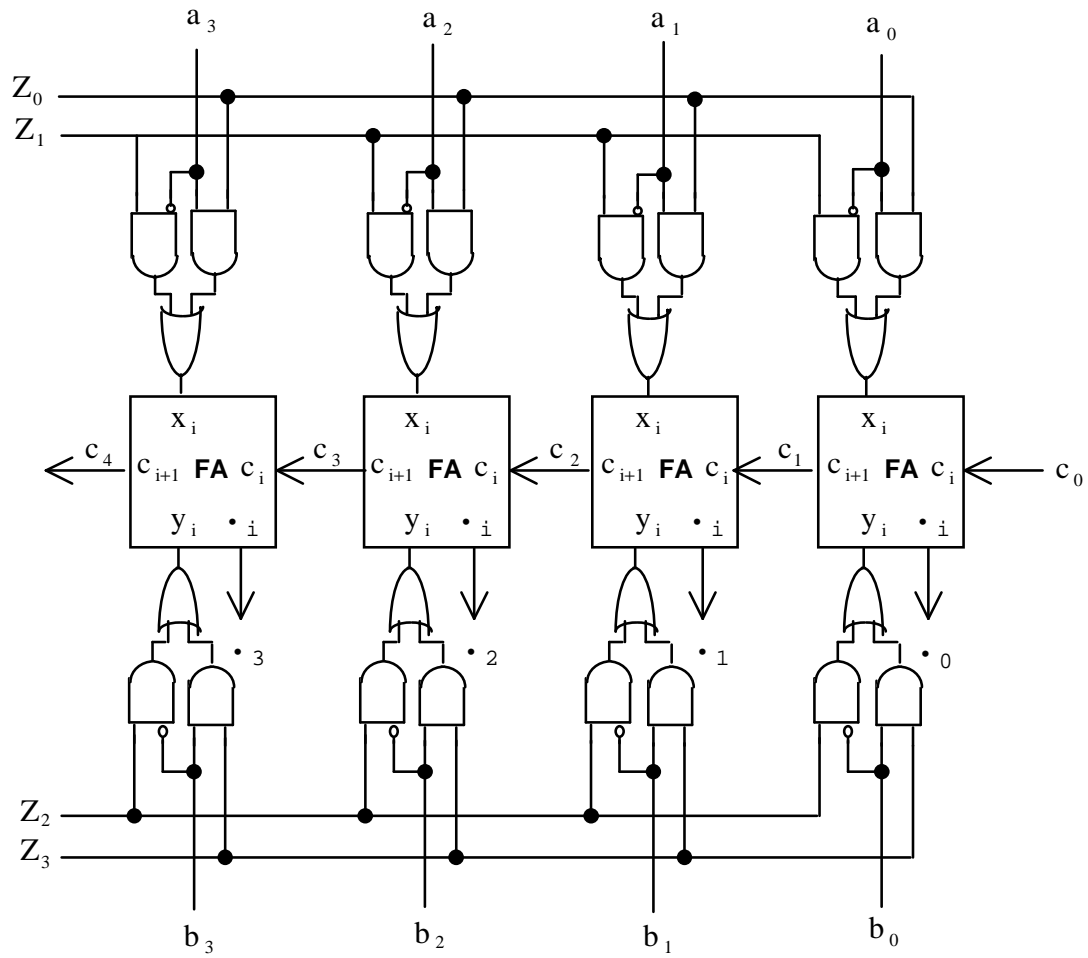
Se definieron varias operaciones lógicas y se construyeron unidades lógicas que permiten realizar varias operaciones lógicas entre números de varios bits.

La unidad aritmética/lógica contiene la integración de los dos conceptos anteriores en una sola unidad. Se realizó un diseño de esta unidad en el cual se incluyeron tanto operaciones aritméticas como lógicas.

Finalmente, se presentó una unidad aritmética que permite realizar sumas y restas de números BCD de varios dígitos.

16.7 Problemas

1. Diseñe un circuito lógico que realice la operación $| (A - B) + (C - D) |$ donde A, B, C y D son números positivos de tres bits, sin signo. El circuito debe indicar cuando ocurra un desborde.
2. Resuelva el problema anterior suponiendo que los números A, B, C y D son números enteros con signo en representación de complementos a 1, usando tres bits en total para cada número.
3. Resuelva el problema 1 suponiendo ahora que los números A, B, C y D son números enteros con signo en representación de complementos a 2, usando tres bits en total para cada número.
4. Resuelva el problema 1 suponiendo ahora que los números A, B, C y D son números enteros sin signo en representación BCD, usando dos dígitos decimales para representar cada número.
5. Sean A y B dos números enteros (se usan complementos a dos para representar los números negativos).
¿Qué funciones realiza la siguiente unidad aritmética ?

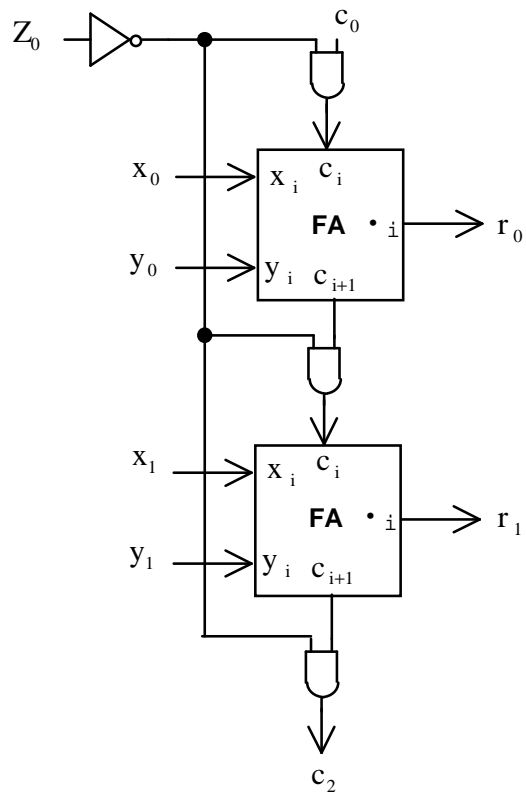


6. Sean A y B dos números enteros (se usan complementos a dos para representar los números negativos). Indique las funciones que realiza la siguiente ALU si

$$c_0 = Z_1$$

$$y_i = b_i Z_0' Z_1' + b_i' Z_1$$

$$x_i = a_i + b_i Z_0 Z_1' + b_i' Z_1 Z_0$$



Modifíquela para que realice las siguientes funciones:

z_0	z_1	Función
0	0	$A + B$
0	1	$A - 1$
1	0	$A \text{ OR } B$
1	1	$\text{NOT } A$

CAPITULO 17

INTRODUCCION A LOS CIRCUITOS SECUENCIALES.

Los circuitos lógicos que se han analizado y diseñado anteriormente han sido circuitos lógicos combinatorios. En este tipo de circuitos, las salidas en cualquier momento dependen exclusivamente de los valores de las entradas presentes. Los valores pasados de las entradas no influyen sobre los valores de las salidas.

Existen también los circuitos lógicos secuenciales, en los cuales, los valores de las salidas dependen tanto de las entradas presentes como del estado interno del circuito. Una pregunta obligada es: ¿Qué es el estado interno del circuito? Los circuitos lógicos secuenciales tiene la característica de almacenar valores booleanos internamente. Estos valores se conservan aún cuando las entradas al circuito ya no estén presentes. El estado interno del circuito está determinado por estos valores que se conservan internamente.

En la figura 17.1 se muestran las partes de que consta un circuito lógico secuencial.

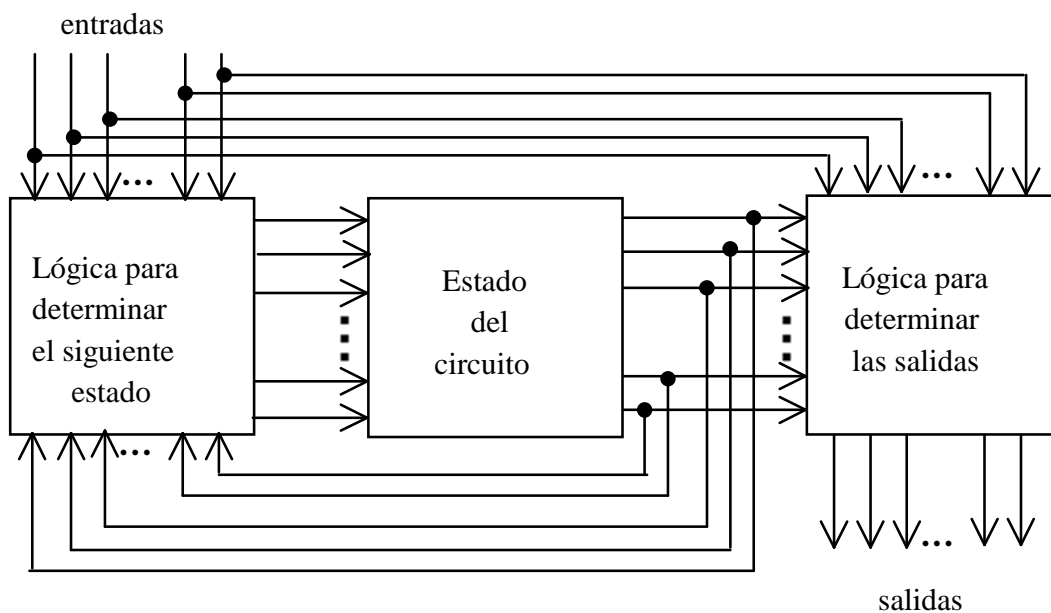


Figura 17.1 Circuito lógico secuencial

Los bloques de la figura 17.1 denominados “Lógica para determinar el siguiente estado” y “Lógica para determinar las salidas”, son circuitos lógicos combinatorios similares a los analizados en los capítulos anteriores. El bloque “Estado del circuito” es el que precisamente tiene los elementos que almacenan el estado interno del circuito. Este bloque no es un circuito combinatorio.

El estado futuro del circuito secuencial se determina en base al estado actual y la combinación de valores presentes en las entradas. En la figura se puede ver que las entradas al bloque llamado

“Lógica para determinar el siguiente estrado” están compuestas por entradas propias del circuito secuencial y el estado actual del sistema, el cual se retroalimenta.

En la misma figura se puede apreciar que las entradas al bloque “Lógica para determinar las salidas” también están formadas por el estado actual del sistema y las entradas externas del circuito secuencial.

Muchas aplicaciones necesitan ser desarrolladas con circuitos secuenciales debido a que demandan que se sucedan varios eventos en un cierto orden. Por ejemplo, considérese el control de un elevador y suponga que éste se encuentra en el nivel 1. Una persona que está en el nivel 7 desea bajar y oprime el botón del nivel 7 que indica que alguien desea descender. El elevador iniciará el ascenso al nivel 7. Si en ese momento otra persona que se encuentra en el nivel 3 desea subir, oprimirá el botón correspondiente en el nivel 3. El elevador se detendrá en el nivel 3 para que la persona suba. Si esta persona oprime el botón que está dentro del elevador correspondiente al nivel 10, el ascensor iniciará su recorrido hacia el nivel 10 y no se detendrá en el nivel 7 dado que la persona que está en ese nivel indicó que quería descender, no subir. Una vez que el elevador llega al nivel 10 y la persona que iba en él lo abandona, iniciará el descenso hacia el nivel 7 y se detendrá en ese nivel para que la primera persona que deseaba bajar lo aborde.

Repasando la serie de eventos anteriores, es fácil deducir que es necesario que el control del elevador “recuerde” que una persona que se encuentra en el nivel 7 desea bajar, para que una vez que atendió a la persona del nivel 3, proceda con el siguiente servicio. En este ejemplo, los botones que se encuentran en cada uno de los pisos así como los botones que están dentro del mismo elevador, representan entradas externas al circuito secuencial que lo controla. Cada vez que se oprime alguno de estos botones, el estado interno del circuito secuencial se modifica para que la unidad de control del elevador “se acuerde” de las solicitudes que tiene pendientes y las atienda en el orden más eficiente. Debido a esto, el elevador del ejemplo anterior no se detuvo en el nivel 7 cuando iba de subida, sino hasta que llevó a la persona al nivel 10 y emprendió su carrera de bajada.

En este capítulo se estudiarán las celdas biestables que son los elementos básicos que permiten almacenar el estado interno de los circuitos secuenciales.

17.1 Celdas biestables o LATCHES

Una celda biestable, comúnmente denominada LATCH, es un circuito lógico que puede almacenar un valor booleano o un bit.

En la figura 17.2 se muestra una celda biestable construida en base a bloques lógicos NOR, llamada LATCH S-R.

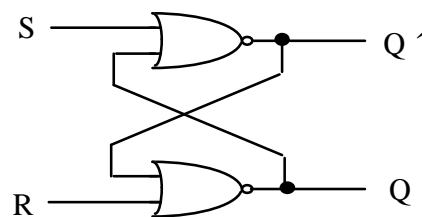


Figura 17.2 LATCH S-R con bloques NOR

La salida Q se conoce como la salida normal de la celda y la salida Q' , en operación normal, es el complemento de la salida Q y se conoce como la salida complementada. Las entradas S y R reciben sus nombres por SET (activar) y RESET (desactivar).

Al analizar la operación de un LATCH debe recordarse que cuando a una compuerta lógica se le ponen las entradas, la salida no aparece inmediatamente ya que transcurre cierto tiempo en que las señales se propaguen hasta la salida. El tiempo de propagación es aproximadamente el mismo para todas las compuertas lógicas combinatorias. A continuación se analizará la operación de este LATCH.

Al proceder con el análisis, es necesario suponer ciertos valores iniciales para las salidas Q y Q' . Asimismo, deben tomarse en cuenta los tiempos de propagación de las entradas a la salida en cada compuerta. Al seguir las explicaciones que describen la operación de los LATCHES se hace referencia a los diagramas de tiempo, que muestran como cambian las diferentes señales, tomando en cuenta los tiempos de propagación en las compuertas. En estos diagramas se utiliza el tiempo de propagación en una compuerta como unidad.

Bajo ciertas condiciones, existen estados transitorios que reflejan los cambios en las salidas de las diferentes compuertas lógicas, pero debe continuarse con el análisis hasta que todas las señales se hayan estabilizado. Se supone que durante todo este tiempo, las entradas externas al LATCH deben permanecer constantes.

Supóngase que inicialmente el valor de Q es 1, el valor de Q' es cero, y los valores de las entradas S y R son cero. El bloque NOR superior tiene como entradas $S=0$ y $Q=1$; por lo tanto, la salida de este bloque es 0, lo cual concuerda con el valor de Q' . Las entradas al bloque NOR inferior son $R=0$ y $Q'=0$, lo cual hace que la salida de este bloque sea 1, que coincide con el valor de Q . Bajo estas condiciones, las salidas del LATCH no cambian.

Ahora supóngase que Q tiene el valor de 0, Q' tiene un valor de 1, y los valores de S y R también son cero. Las entradas al bloque NOR superior son $S=0$ y $Q=0$, lo cual da como salida el valor de 1, que concuerda con el valor de Q' . Las entradas al bloque NOR inferior son $R=0$ y $Q'=1$, lo cual da como resultado que su salida sea 0, que es el mismo valor que ya tenía Q . En este caso, el LATCH tiene sus salidas estables.

De los dos casos analizados anteriormente se puede concluir que el LATCH es capaz de mantener ya sea un cero o un uno en su salida Q , cuando sus entradas son ambas cero. Este

circuito lógico puede estar en uno de estos dos estado posibles, de ahí que se denomine celda biestable.

Lo que haría que este LATCH realmente fuera útil sería poder cambiarlo de un estado a otro a voluntad, para poder almacenar en él un valor booleano o un bit (cero o uno). Se analizará a continuación la forma de hacerlo.

Supóngase ahora que $Q=0$ y $Q'=1$. Si las entradas S y R son ambas cero, el valor de Q se mantiene como se analizó anteriormente. Esto corresponde al tiempo cero en el diagrama de tiempos de la figura 17.3. Estando en el estado descrito, supóngase que en el tiempo 1 se pone un uno en la entrada S , manteniendo la entrada R en cero. Las entradas al bloque NOR superior son $S=1$ y $Q=0$, dando como salida un cero en el tiempo 2, que modifica el valor de Q' . La salida del otro bloque NOR todavía no se modifica.

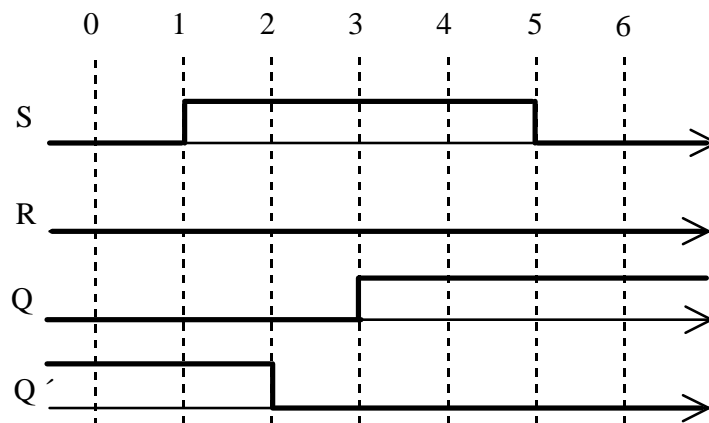
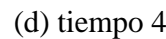


Figura 17.3 Diagrama de tiempos para activar el LATCH S-R

Las entradas al bloque NOR inferior ahora son $R=0$ y $Q'=0$, lo cual produce una salida de 1 en el tiempo 3, modificando el valor de Q , por lo que ahora $Q=1$. Esto hace que las entradas al bloque NOR superior sean ahora $S=1$ y $Q=1$, dando como salida un cero en el tiempo 4, que no modifica el valor de Q' , manteniéndolo en cero. En este momento todas las señales del circuito se han estabilizado. Si en el tiempo 5 se pone un cero en la entrada S , la salida del bloque NOR superior continuará en cero, lo cual se muestra en el tiempo 6. El nuevo estado de la celda binaria se mantendrá mientras no cambien las entradas S y R , dado que las señales ya están estables.

En la figura 17.4 se muestran las entradas y salidas de los bloques lógicos correspondientes a los tiempos 1, 2, 3 y 4 del diagrama de tiempos de la figura 17.3. Nótese que si se analizan estáticamente cada uno de las figuras, las salidas no siempre parecen corresponder con las entradas debido a que éstas últimas aún no se han propagado hasta las salidas.



5

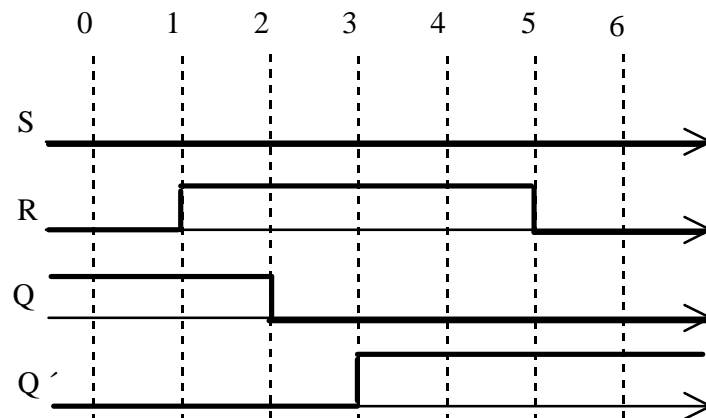


Figura 17.5 Diagrama de tiempos para desactivar el LATCH S-R

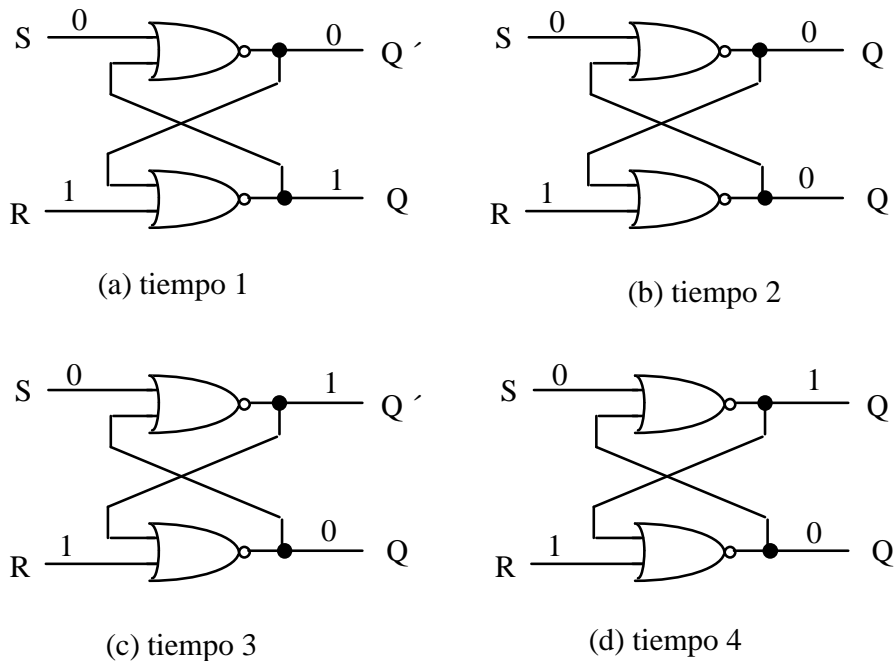


Figura 17.6 Cuatro instantes de tiempo correspondientes al diagrama de la figura 17.5

La figura 17.6 muestra las entradas y salidas del LATCH en cuatro instantes de tiempo correspondientes al diagrama 17.5.

Analizando ahora el caso en que $Q=0$ y $Q'=1$, ya se sabe que si las entradas S y R son ambas cero, el valor de Q se mantendrá. Si ahora se cambia el valor de la entrada R a uno, las entradas al bloque NOR inferior serán $R=1$ y $Q'=1$, dando como resultado un cero, que concuerda con el valor de Q . Las entradas al bloque NOR superior son $S=0$ y $Q=0$, dando como salida un uno, que es precisamente el valor de Q' . Al poner de nuevo la entrada R en cero, el estado del LATCH permanecerá igual al que tenía inicialmente.

Como conclusión de los últimos cuatro casos analizados, se puede afirmar que si en la entrada S se pone el valor de 1, manteniendo la entrada R en 0, el LATCH quedará en el estado $Q=1$ y $Q'=0$, independientemente del estado en que se encontraba. Al poner de nuevo un cero en la entrada S, este estado permanecerá.

Si se pone un 1 en la entrada R, manteniendo la entrada S en 0, el LATCH quedará en el estado $Q=0$ y $Q'=1$, independientemente del estado en que se encontraba inicialmente. Poniendo de nuevo un cero en la entrada R, este estado será conservado.

Por consiguiente, si se desea almacenar el valor de 1 en la salida Q del LATCH S-R, deberá ponerse un 1 en la entrada S y un 0 en la entrada R. Al quitar el 1 de la entrada S, este estado ($Q=1$) se mantendrá. Si lo que se quiere es almacenar un 0 en la salida Q, debe ponerse un 1 en la entrada R y un 0 en la entrada S. Al regresar el valor de la entrada R a 0, este estado será conservado.

El único caso que faltaría analizar es lo que ocurriría si se ponen en 1 ambas entradas, S y R. Mientras se mantengan ambas entradas en 1, las salidas de los dos bloques lógicos NOR serán 0, haciendo una inconsistencia ya que $Q=Q'$. Sin embargo, este es un estado que no puede ser mantenido por el LATCH al poner de nuevo las entradas en cero. El estado final que se obtendría depende de cual de las entradas, S o R, se haga cero primero, ya que físicamente es imposible hacerlas cero al mismo tiempo. Si se hace cero primero la entrada R, el LATCH quedará en el estado $Q=1$. Si se hiciera cero primero la entrada S, se quedaría en el estado $Q=0$. Para este caso ($S=R=1$), se dice que el estado en que queda el LATCH no está determinado. El LATCH S-R no está diseñado para trabajar con esta condición, por lo cual debe evitarse.

Otra forma de construir un LATCH S-R que se comporte de igual manera que la anterior es utilizar dos bloques NAND y dos bloques NOT, tal como se muestra en la figura 17.7.

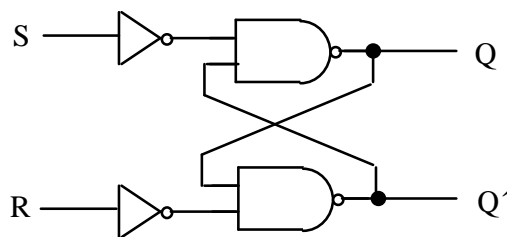


Figura 17.7 LATCH S-R con bloques NAND

Cuando ambas entradas son cero, este LATCH también mantiene su estado actual. Por ejemplo, suponiendo que $S=0$, $R=0$, $Q=1$ y $Q'=0$, las entradas al bloque NAND superior son $S'=1$ y $Q'=0$, lo cual da como salida el valor de 1, que coincide con el valor de Q. Las entradas al bloque NAND inferior son $R'=1$ y $Q=1$, dando como salida el valor de 0, que es el de Q' . Esto indica que el estado $Q=1$ se preserva cuando ambas entradas son cero.

Analizando el caso en que $S=0$, $R=0$, $Q=0$ y $Q'=1$, se puede ver que el bloque NAND superior tiene como entradas $S'=1$ y $Q'=1$, dando como salida el valor de 0, lo cual corresponde al valor de Q . Las entradas al bloque NAND inferior son $R'=1$ y $Q=0$, lo cual da como salida el valor de 1, que corresponde al valor de Q' . Por lo tanto, el LATCH conservará este estado.

Si se considera ahora el caso en que $S=1$ y $R=0$, se verá que el LATCH quedará en el estado $Q=1$ al hacer de nuevo $S=0$, independientemente del valor que tenga Q inicialmente. Si $S=1$, $R=0$, $Q=1$ y $Q'=0$, las entradas al bloque NAND superior son $S'=0$ y $Q'=0$, lo cual hace que su salida sea 1, coincidiendo con el valor de Q . Las entradas al bloque inferior son $R'=1$ y $Q=1$, haciendo que su salida sea 0, manteniendo el valor de Q' . Al hacer cero de nuevo la entrada R , el estado $Q=1$ se mantendrá.

Se analizará ahora el caso en que el LATCH se encuentra en el estado $Q=0$ y se pone un uno en su entrada S , manteniendo la entrada R en cero. En el diagrama de tiempos de la figura 17.8 se muestra la condición $S=0$, $R=0$, $Q=0$ y $Q'=1$ en el tiempo 0, suponiendo que dicha condición ha estado durante más de un tiempo de propagación. Debido a que las entradas llegan primero a los dos bloques NOT antes de alimentarse a los bloques NAND, se mostrarán también las salidas de estos bloques como variables adicionales, llamando X a la salida del bloque NOT a donde llega la entrada S , y Y a la salida del bloque NOT a donde llega R .

Si la entrada S se cambia al valor de 1 en el tiempo 1, la salida X del bloque NOT a donde llega S no aparecerá sino hasta el tiempo 2, manteniéndose todas las demás variables sin cambio. En este instante, las entradas al bloque NAND superior cambian a $X=0$ y $Q'=1$, pero la salida de este bloque cambia al valor de 1 hasta el tiempo 3, haciendo $Q=1$. Esto afecta las entradas al bloque NAND inferior que ahora son $Q=1$ y $Y=1$, por consiguiente, su salida cambiará al valor de 0 en el tiempo 4, haciendo que Q' valga 0. Ahora las entradas al bloque NAND superior son $X=0$ y $Q'=0$, pero esto no afecta su salida que ya tenía el valor de 1. En este instante de tiempo, los valores de las variables ya están estabilizados. Si en el tiempo 5 se cambia de nuevo el valor de la entrada S a cero, la salida X del correspondiente bloque NOT cambiará a 1 en el tiempo 6. Este nuevo valor de X no afecta la salida del bloque NAND inferior que ya tenía el valor de 1 ($Q=1$). En este momento el circuito ya se ha estabilizado y el LATCH mantendrá su nuevo estado $Q=1$.

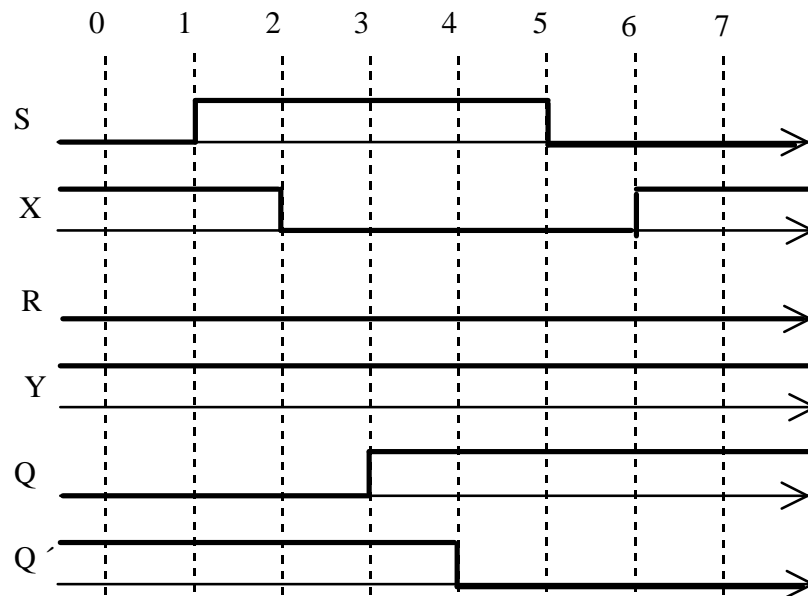


Figura 17.8 Diagrama de tiempos para activar el LATCH S-R

Si se considera ahora el caso en que el LATCH se encuentre en el estado $Q=1$ ($Q'=0$) y se cambia el valor de la entrada R a 1, manteniendo el valor de la otra entrada en 0, se comprobará que el estado final del LATCH al volver de nuevo el valor de la entrada R a 0, será $Q=0$. En la figura 17.9 se muestra el diagrama de tiempos para este caso.

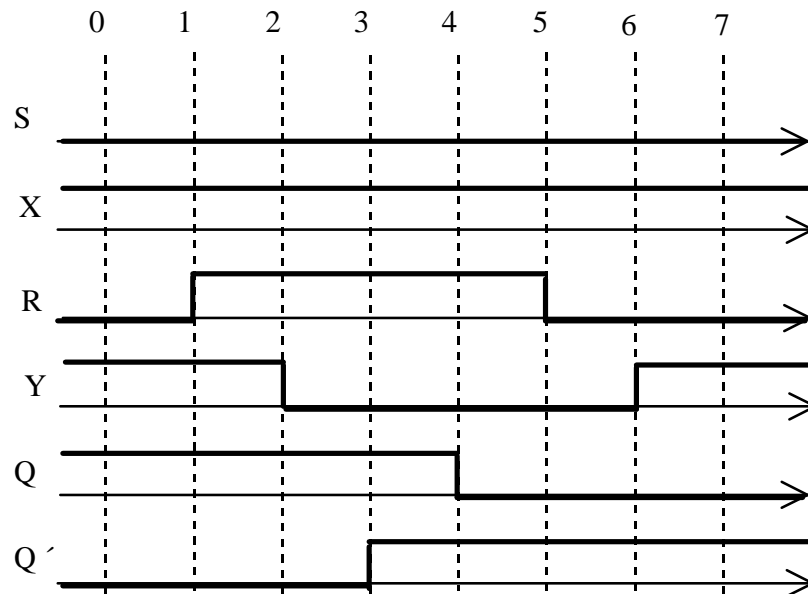


Figura 17.9 Diagrama de tiempos para desactivar el LATCH S-R

Finalmente, el lector podrá comprobar que si se colocan ambas entradas, S y R, en 1, se genera una inconsistencia ya que tanto Q como Q' tomarán el valor de 1. Este estado no se puede

mantener cuando las entradas se hacen cero y el LATCH quedará en uno de los dos estados estables ($Q=0$ o $Q=1$), pero no se puede predecir en cual estado quedará, por lo cual se debe de evitar colocar estas entradas.

Existen todavía más formas de construir un LATCH cuyo comportamiento sea igual al de los dos descritos en esta sección. De aquí en adelante se utilizará el símbolo mostrado en la figura 17.10 para representar estos LATCHES, independientemente del circuito específico que se utilice internamente para construirlos.

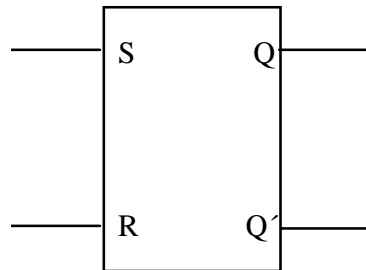


Figura 17.10 Símbolo para el LATCH S-R

En la tabla 17.1 se presenta un resumen de la operación del LATCH S-R.

S	R	estado futuro
0	0	conserva su estado
0	1	$Q=0$
1	0	$Q=1$
1	1	indeterminado

Tabla 17.1. Operación del LATCH S-R

Si se modifica el LATCH presentado en la figura 17.7 quitándole los bloques NOT, se obtiene el LATCH mostrado en la figura 17.11.

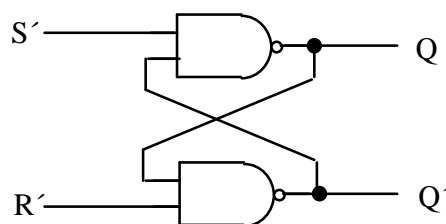


Figura 17.11. LATCH S-R con entradas invertidas

La operación de este LATCH es similar al mostrado en la figura 17.7, excepto que, como se han quitado los bloques NOT en las entradas, opera cuando las entradas son los complementos de las correspondientes entradas del anterior. El símbolo que se usa para este tipo de LATCH S-R es el

mostrado en la figura 17.12. Nótese que la diferencia son los dos círculos mostrados en las entradas.

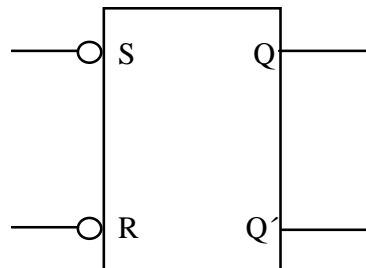


Figura 17.12. Símbolo para el LATCH S-R con entradas invertidas

La operación de un LATCH S-R con entradas invertidas se resume en la tabla 17.2. Nótese que conserva su estado cuando ambas entradas valen 1.

S'	R'	estado futuro
1	1	conserva su estado
1	0	$Q=0$
0	1	$Q=1$
0	0	indeterminado

Tabla 17.2. Operación del LATCH S-R con entradas invertidas

Si a las entradas del LATCH S-R mostrado en la figura 17.2 se les colocan bloques NOT, también se obtendrá un LATCH S-R con entradas invertidas, cuya operación es igual a la del último LATCH presentado.

17.2 Señales de reloj

La mayoría de los circuitos secuenciales se diseñan para operar en forma sincrónica. Esto significa que los cambios en los estados internos de los circuitos solamente ocurren en instantes predefinidos de tiempo, los cuales son controlados por una señal maestra de reloj.

La señal de reloj generalmente es una forma de onda cuadrada que alterna su valor entre los niveles de voltaje correspondientes al cero y al uno lógicos. En la figura 17.13 se muestra la forma de onda de una señal típica de reloj.

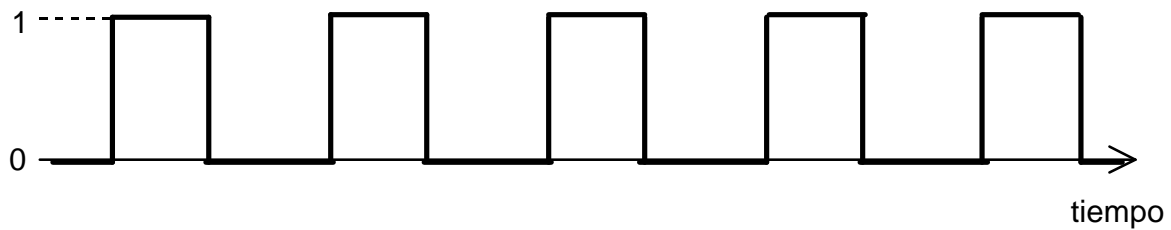


Figura 17.13. Señal típica de reloj

El estado de un circuito secuencial está formado por el conjunto de valores booleanos almacenados internamente mediante LATCHES. Para que un circuito secuencial opere en forma sincrónica es necesario contar con celdas biestables que tengan una entrada de reloj, además de las entradas S y R. En la entrada de reloj de cada una de las celdas S-R se alimentaría la señal maestra de reloj, la cual sincronizaría todas las celdas biestables para que cambiaran al mismo tiempo. Las celdas biestables con entrada de reloj se llamarán celdas de memoria o FLIP FLOPS.

17.3. FLIP FLOP S-R (Set-Reset Flip Flop)

Se pueden diseñar celdas biestables S-R con una entrada de reloj que cambien su estado solamente cuando la señal de reloj tenga el valor de uno. Estas celdas se conocen como FLIP FLOPS S-R que operan con nivel de reloj alto. En la figura 17.14 se muestran dos FLIP FLOPS S-R de este tipo, contruidos utilizando bloques NAND. La entrada denominada C (CLOCK), corresponde a la entrada de la señal de reloj. En el segundo circuito se unieron los bloques AND y NOT en un solo bloque NAND. También puede construirse basándose en el LATCH que utiliza bloques NOR, o usando bloques AND y NOT, o bloques OR y NOT.

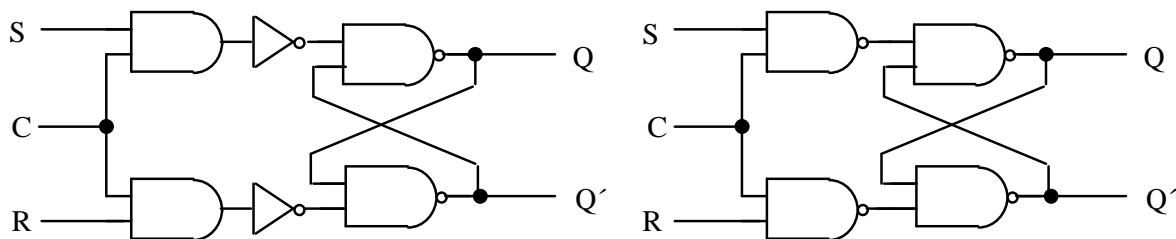


Figura 17.14. FLIP FLOPS S-R con nivel de reloj alto

En la figura 17.15 se muestra el símbolo que se utilizará para representar un FLIP FLOP S-R, sin importar su construcción interna. Nótese que el símbolo es similar al del LATCH S-R, la diferencia es que el FLIP FLOP S-R tiene la entrada de reloj adicional.

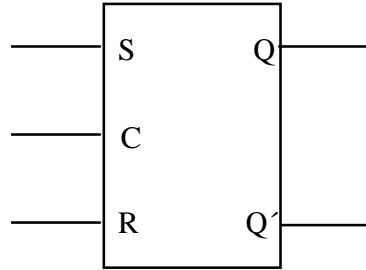


Figura 17.15. Símbolo para el FLIP FLOP S-R con nivel de reloj alto

Analizando el primer circuito presentado en la figura 17.14 se puede ver que cuando la señal de reloj (la entrada C) vale cero, la salida de los bloques AND serán cero. Esta condición es análoga a cuando se tenían ceros en las entradas S y R de un LATCH S-R, la cual originaba que conservara su estado. Cuando la señal de reloj vale uno, las entradas S y R se aplican a los bloques NAND y la celda funciona exactamente igual al LATCH S-R analizado inicialmente. El estado de este FLIP FLOP cambia solamente cuando la señal de reloj vale uno, por lo tanto, opera con nivel de reloj alto.

Si se añade un bloque lógico NOT en la entrada de reloj, se tendrá un FLIP FLOP S-R con nivel de reloj bajo, similar al mostrado en la figura 17.16.

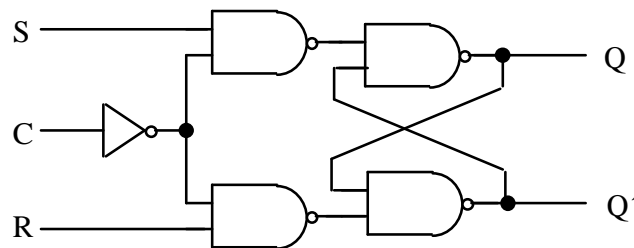


Figura 17.16. FLIP FLOP S-R con nivel de reloj bajo

Nótese que en este FLIP FLOP, las entradas S y R se aplicarán a los bloques NAND solamente cuando la señal de reloj valga cero. Por consiguiente, el estado cambiará sólo cuando la señal de reloj tome este valor. El símbolo utilizado para un FLIP FLOP S-R que opera con nivel de reloj bajo se muestra en la figura 17.17. Nótese que la entrada de reloj se encuentra complementada.

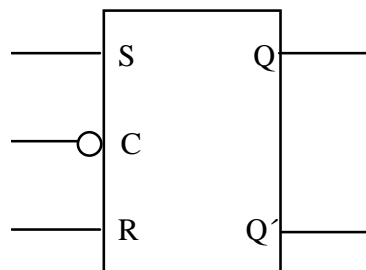


Figura 17.17. Símbolo del FLIP FLOP S-R con nivel de reloj bajo

Se puede utilizar una tabla de verdad para representar el comportamiento del FLIP FLOP S-R como se muestra en la tabla 17.1. En ésta aparece la variable Q que es el valor del FLIP FLOP en un instante de tiempo dado, el cual se conoce como el estado actual y, como función de salida, Q^+ que representa el siguiente estado. Este es el valor que tomará el FLIP FLOP después de que ocurra el evento que ocasiona el cambio de estado, o sea, el nivel del reloj alto o bajo según sea el caso.

S	R	Q	Q^+
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	NO
1	1	1	NO

Tabla 17.1 Tabla de funcionamiento de un FLIP FLOP S-R

Si se considera que las entradas S y R nunca tomaran simultáneamente el valor de 1, se puede obtener la siguiente función para el FLIP FLOP S-R:

$$Q^+ = S + R'Q$$

La cual nos indica que el siguiente estado del FLIP FLOP será 1 cuando $S=1$ o cuando el estado actual es 1 y $R=0$.

17.4. EL FLIP FLOP D (Delay Flip Flop)

Dado que un FLIP FLOP puede guardar un valor booleano, se pensó en crear uno que tuviera una sola entrada en la cual se pusiera el valor que se deseara guardar, adicionalmente a la entrada de reloj para controlar el momento en que cambie su estado. Para lograr esto, se diseñó el FLIP FLOP D, el cual tiene una entrada que se denomina D, y la entrada de reloj C.

La construcción de este FLIP FLOP se basa en el LATCH S-R. Una posible configuración se muestra en la figura 17.18, la cual usa un LATCH S-R construido con bloques NAND y que opera con nivel de reloj alto.

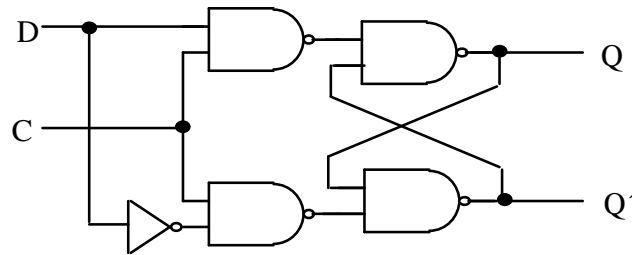


Figura 17.18 FLIP FLOP D construido con bloques NAND con nivel de reloj alto

El funcionamiento de este FLIP FLOP es muy simple. Cuando la señal de reloj toma el nivel de uno, el valor que está en la entrada D se almacena en su salida Q. Para analizarlo, supóngase que en D está un uno. Cuando la señal de reloj cambia de cero a uno, el bloque superior NAND del LATCH recibe un cero en la entrada que proviene del bloque NAND superior y el bloque NAND inferior del LATCH recibe un uno del bloque NAND inferior. Esto equivale en que en un LATCH S-R se tengan las entradas $S=1$ y $R=0$, lo cual ocasiona que se cambie al estado $Q=1$.

Si ahora se pone el valor de cero en la entrada D, cuando la señal de reloj pasa de cero a uno, el bloque NAND superior del LATCH recibe un uno de la salida del bloque NAND superior y el bloque NAND inferior del LATCH recibe un cero del bloque NAND inferior. Esto equivale a que en un LATCH S-R se tenga $S=0$ y $R=1$, lo cual lo deja en el estado $Q=0$.

En la figura 17.19 se muestra un FLIP FLOP D construido con bloques NAND y que opera con nivel de reloj bajo.

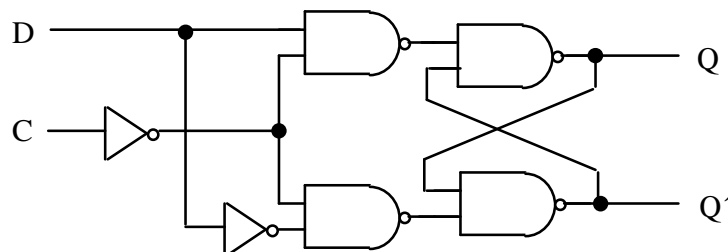


Figura 17.19 FLIP FLOP D construido con bloques NAND con nivel de reloj bajo

En la figura 17.20 se muestran los símbolos que se utilizarán para los FLIP FLOPS D, sin importar su construcción interna.

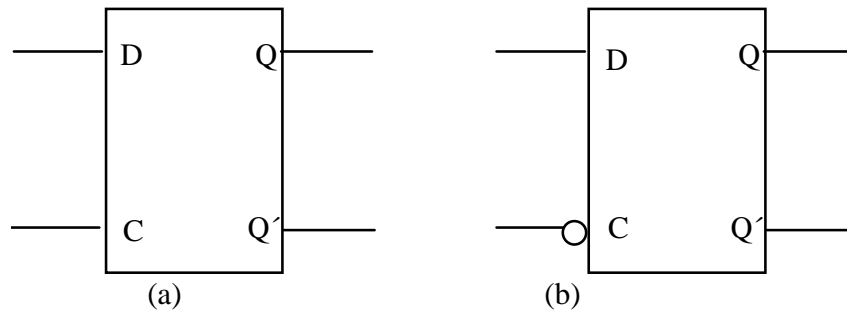


Figura 17.20 Símbolos para FLIP FLOPS D
(a) con nivel de reloj alto, y (b) con nivel de reloj bajo

La tabla 17.2 muestra el funcionamiento del FLIP FLOP D.

D	Q	Q ⁺
0	0	0
0	1	0
1	0	1
1	1	1

Tabla 17.2 Tabla de funcionamiento de un FLIP FLOP D
La función para el FLIP FLOP D es la siguiente:

$$Q^+ = D$$

La cual nos indica que el siguiente estado del FLIP FLOP será igual a D.

17.5. El FLIP FLOP T (Trigger Flip Flop)

Este FLIP FLOP tiene solamente dos entradas: la entrada de reloj (C) y la entrada que controla su operación (T). Cuando la entrada de reloj es cero, el estado del FLIP FLOP no cambia. Cuando la entrada de reloj vale uno, el comportamiento del FLIP FLOP depende del valor que tenga la entrada T. Si esta entrada es cero, el estado se conservará. En cambio, si la entrada T vale uno, el FLIP FLOP cambiará siempre su estado. Una forma de construirlo se muestra en la figura 17.21.

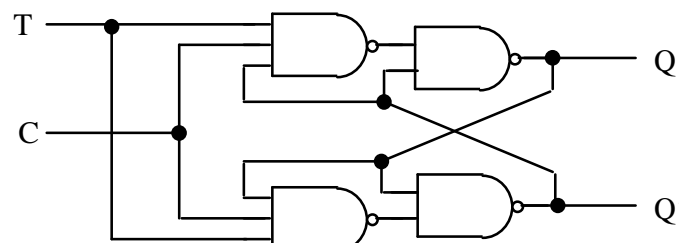


Figura 17.21. FLIP FLOP T construido con bloques NAND con nivel de reloj alto

Cuando la señal de reloj vale cero, las salidas de los dos bloques NAND de la izquierda valen uno, haciendo que la celda conserve su estado (revísese el LATCH de la figura 17.11).

Suponiendo que la señal de reloj vale uno y la entrada T vale cero, las salidas de los dos bloques NAND de la izquierda valen uno; por consiguiente, la celda conservará su estado.

Si tanto la señal de reloj como la entrada T valen uno, el FLIP FLOP cambiará de estado. Las salidas de los dos bloques NAND de la izquierda dependerán del estado en que se encuentre ya que ambas salidas, Q y Q', se retroalimentan a estos bloques. Este es el caso que realmente vale la pena analizar.

Se mostrará un diagramas de tiempo (figuras 17.22) para analizar la operación de este FLIP FLOP. La salida del bloque NAND superior de la izquierda se denominará X y la salida del bloque NAND inferior de la izquierda se identifica como Y en el diagrama.

En la figura 17.22 se muestra el diagrama de tiempos para la siguiente serie de eventos:

- Se supone que el FLIP FLOP se encuentra en el estado $Q=1$ y tanto la entrada T como el reloj están en 0.
- En el tiempo 1 se pone un 1 en la entrada T.
- En el tiempo 2 se pone un 1 en la entrada de reloj.

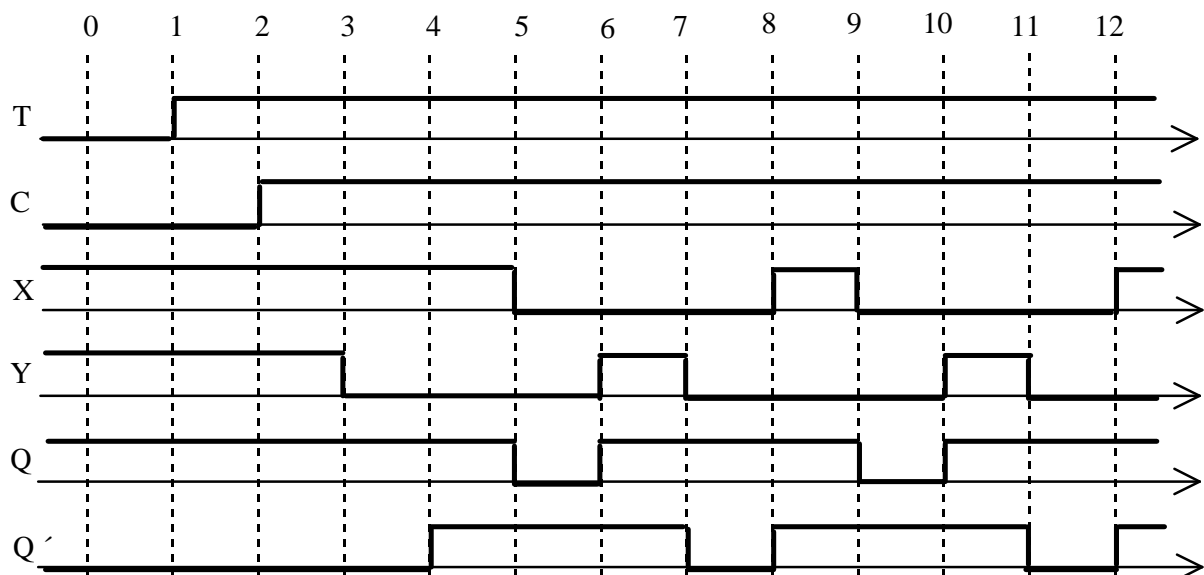


Figura 17.22. Diagrama de tiempos del FLIP FLOP T

En el diagrama de tiempos se puede observar que inicialmente el FLIP FLOP estaba en el estado $Q=1$ y en el tiempo 5 se encuentra en el estado $Q=0$, con $Q'=1$; sin embargo, en el tiempo 7 se encuentra de nuevo en el estado inicial y en el tiempo 9 vuelve a encontrarse en $Q=0$ y $Q'=1$, repitiéndose el ciclo mientras la señal de reloj continúe en el valor de 1.

Si la señal de reloj se hubiese puesto en 0 en el tiempo 4, el FLIP FLOP solamente hubiera cambiado de estado, quedando en $Q=0$ y $Q'=1$, pero al mantener la señal de reloj en 1 ocasiona que este FLIP FLOP oscile, cambiando constantemente de estado.

Este problema se origina porque las salidas del FLIP FLOP están retroalimentadas a las entradas y, al cambiar las salidas, también cambian las entradas. La configuración de “maestro-esclavo” (Master-Slave) para FLIP FLOPS permitirá resolver este inconveniente.

Para ser consistente con la representación de las memorias anteriores, el símbolo que se utilizará para este FLIP FLOP se muestra en la figura 17.23.

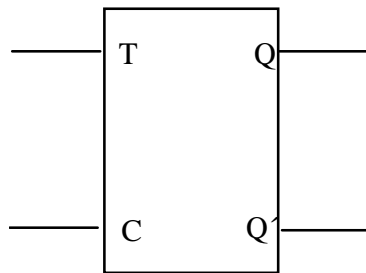


Figura 17.23. Símbolo para el FLIP FLOP T con nivel de reloj alto

La tabla 17.3 muestra el funcionamiento del FLIP FLOP T.

T	Q	Q ⁺
0	0	0
0	1	1
1	0	1
1	1	0

Tabla 17.3 Tabla de funcionamiento de un FLIP FLOP T

La función para el FLIP FLOP T es la siguiente:

$$Q^+ = T \oplus Q$$

La cual nos indica que el siguiente estado del FLIP FLOP se mantendrá si T es 0 o se complementará si T = 1.

17.6. El FLIP FLOP J-K (J-K Flip Flop)

Este FLIP FLOP es el más versátil de todos ya que puede trabajar como la S-R o T. Añadiendo bloques lógicos externos también puede hacerse funcionar como FLIP FLOP D. Este dispositivo tiene tres entradas. Dos de ellas controlan los cambios de estado y se denominan J y K. La tercera, denominada C, es la entrada de reloj.

Cuando las dos entradas J y K valen uno, este FLIP FLOP opera como un FLIP FLOP T, cambiando de estado cuando la señal de reloj vale uno. El diseño que se muestra en esta sección también sufre del mismo problema que el FLIP FLOP T. Si la señal de reloj no dura un tiempo muy pequeño en el valor de uno, el estado continuará cambiando mientras la señal de reloj no tome el valor de cero. La solución a este problema también se basa en los FLIP FLOP del tipo “maestro-esclavo”.

Suponiendo que las entradas J y K no pueden valer uno simultáneamente, el FLIP FLOP operará como un S-R, donde J hace las veces de la entrada S y K las veces de R. En la figura 17.24 se muestra una posible configuración para el FLIP FLOP J-K. El símbolo que se utilizará para representarlo se muestra en la figura 17.25.

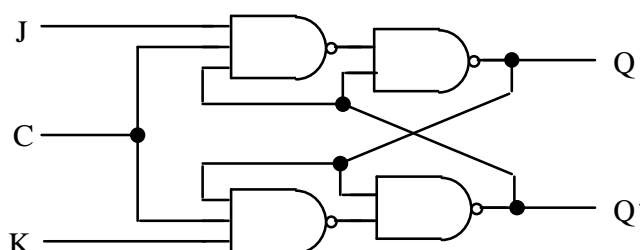


Figura 17.24 FLIP FLOP J-K construido con bloques NAND con nivel de reloj alto

Analizando la figura 17.24, se puede ver fácilmente que si las entradas J y K valen cero al mismo tiempo, las salidas de los bloques NAND de la izquierda tendrán el valor de uno, lo que ocasionará que la celda mantenga su estado cuando C tome el valor de uno.

Cuando las entradas J y K valen uno simultáneamente, comparando las figuras 17.21 y 17.24, se podrá deducir esta celda trabajará igual que la memoria T.

Cuando la entrada J vale uno y la entrada K vale cero, al tomar el valor de uno la señal de reloj, la celda cambiará al estado $Q=1$ si estaba en el estado $Q=0$, o permanecerá en el estado $Q=1$ si ya se encontraba en él.

Si la entrada J vale cero y la entrada K vale uno, al presentarse la señal de reloj (al tomar el valor de 1), la celda permanecerá en el estado $Q=0$ si ya estaba en él, o cambiará a este estado si se encontraba en el estado $Q=1$. Se invita al lector a comprobar estos dos últimos casos.

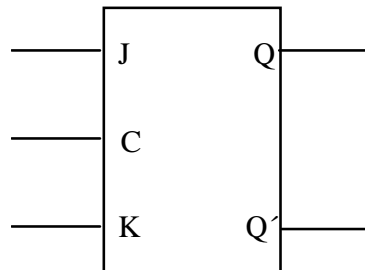


Figura 17.25. Símbolo para el FLIP FLOP J-K con nivel de reloj alto

La tabla 17.4 muestra el funcionamiento del FLIP FLOP J-K.

J	K	Q	Q ⁺
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Tabla 17.1 Tabla de funcionamiento de un FLIP FLOP J-K

La función para el FLIP FLOP J-K es la siguiente:

$$Q^+ = JQ' + R'Q$$

También se mencionó que es posible hacer que un FLIP FLOP J-K funcione como uno D añadiéndole bloques lógicos. Una posible forma de hacerlo se muestra en la figura 17.26. Como se puede observar se hizo $J=D$ y $K=D'$ que al sustituirlos en la función del FLIP FLOP J-K nos queda lo siguiente:

$$Q^+ = DQ' + (D')'Q = D$$

La cual es igual a la función de un FLIP FLOP D.

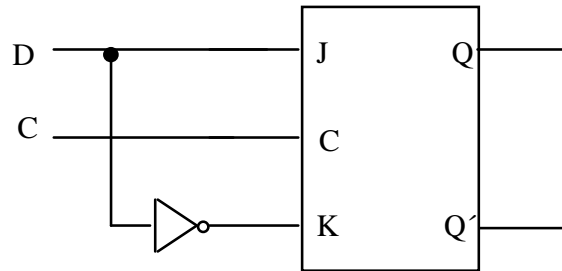


Figura 17.26. FLIP FLOP D construido con un J-K con nivel de reloj alto

17.7. FLIP FLOP S-R maestro-esclavo (Master-Slave S-R Flip Flop)

En las secciones anteriores se comentó el problema que tiene el FLIP FLOP T cuando el pulso de reloj no tiene una duración muy pequeña en el valor de uno. Lo mismo ocurre con el FLIP FLOP J-K cuando ambas entradas, J y K, valen uno y se presenta el pulso de reloj. Esto se debe a que en ambos tipos de celda biestable se retroalimentan las salidas Q y Q' a la entrada, cosa que no ocurre con los FLIP FLOPS S-R y D.

En la introducción a este capítulo se describió la arquitectura general de un circuito lógico secuencial. El estado interno del circuito está formado por un conjunto de FLIP FLOPS, que generalmente son del mismo tipo. Suponiendo que se tuvieran 4 FLIP FLOPS J-K para el estado interno de un circuito secuencial, se tendrían 16 posibles estados, correspondientes a las 16 combinaciones diferentes de las salidas Q de cada celda de memoria.

Para el caso anterior, las entradas al circuito lógico combinatorio que sirve para determinar el siguiente estado (Lógica para determinar el siguiente estado) serían las entradas externas al circuito secuencial más las salidas Q de cada uno de los FLIP FLOPS J-K que forman el estado interno del circuito. La salidas de este circuito combinatorio sería precisamente las entradas J y K de cada uno de los cuatro FLIP FLOPS que forman el estado. Si el pulso del reloj dura mucho tiempo en el valor de uno, al cambiar el estado interno del circuito cambiarán también las entradas al circuito lógico combinatorio que sirve para determinar el siguiente estado, lo que ocasionaría que las salidas de este circuito también cambiaran y, por consiguiente, también cambiaría de nuevo el estado interno del circuito secuencial. Esto continuaría ocurriendo mientras la señal de reloj permaneciera en el valor de uno.

Para evitar estos problemas se diseñaron las celdas biestable del tipo maestro-esclavo. En la figura 17.27 se muestra la configuración interna de un FLIP FLOP S-R maestro-esclavo. Obsérvese que internamente tiene dos FLIP FLOPS S-R. El de la izquierda se conoce como el maestro y el de la derecha es el esclavo. La razón de esta nomenclatura es que cuando el pulso de reloj cambia de cero a uno, el FLIP FLOP maestro cambia de estado de acuerdo a lo que tengan sus entradas S y R, pero el FLIP FLOP esclavo no cambia debido a que la señal de reloj le llega complementada. Cuando la señal de reloj regresa al valor de cero, el FLIP FLOP maestro ya no cambiará, pero en la entrada de reloj del esclavo aparecerá un uno y su estado

cambiará. Las entradas S y R del FLIP FLOP esclavo son precisamente las salidas Q y Q' del maestro, de tal forma que su estado cambiará al estado que tiene el maestro.

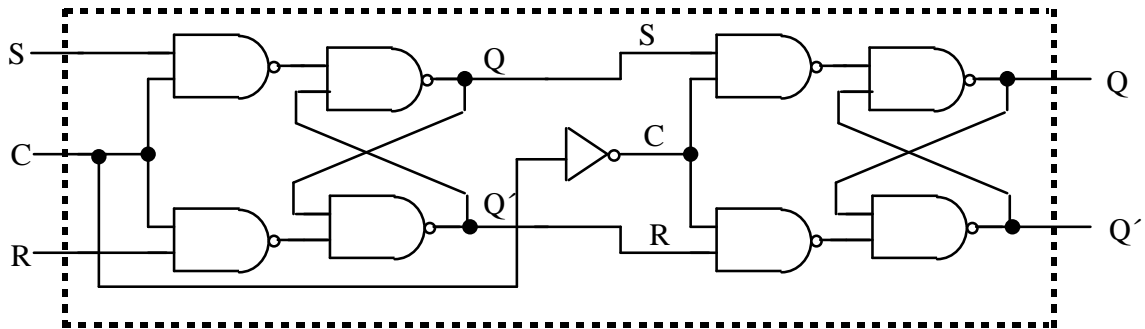


Figura 17.27. FLIP FLOP S-R maestro-esclavo activado con pulso positivo

Estos FLIP FLOPS no se activan de acuerdo al nivel que tenga la entrada de reloj como los anteriores, realmente se activan cuando se presenta un pulso completo en su entrada de reloj. En la figura 17.28 se muestra como opera este FLIP FLOP. Cuando la señal de reloj pasa de 0 a 1, se activa la operación del FLIP FLOP maestro y tarda un tiempo T_1 en estabilizarse el circuito, pero las salidas externas del FLIP FLOP no cambian. Al cambiar la señal de reloj de 1 a 0, inicia la operación del FLIP FLOP esclavo y el maestro ya no cambia. Después de un tiempo T_2 se estabiliza el circuito y se modifican las salidas externas del FLIP FLOP.

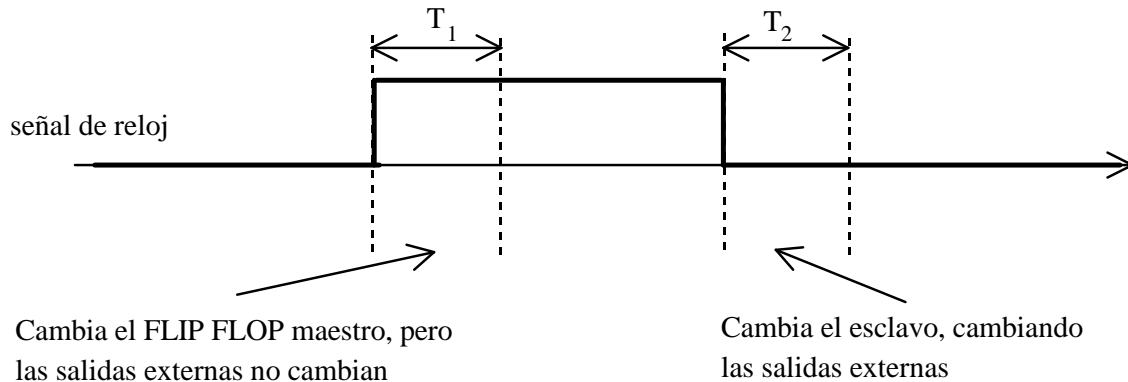


Figura 17.28. Operación de un FLIP FLOP maestro-esclavo al recibir un pulso en su entrada de reloj

Para que un FLIP FLOP de este tipo opere adecuadamente es necesario que sus entradas externas permanezcan estables durante todo el tiempo que dura el pulso de reloj para que el maestro cambie al estado que se desea. También es indispensable que el pulso de reloj tenga una duración igual o mayor al tiempo T_1 para que el maestro se haya estabilizado antes de que la señal de reloj regrese al valor de cero.

En la figura 17.29 se muestra de nuevo la estructura interna del FLIP FLOP S-R maestro esclavo, en la cual se han dado nombres diferentes a las variables internas del circuito para que se puedan identificar en el diagrama de tiempos de la figura 17.30.

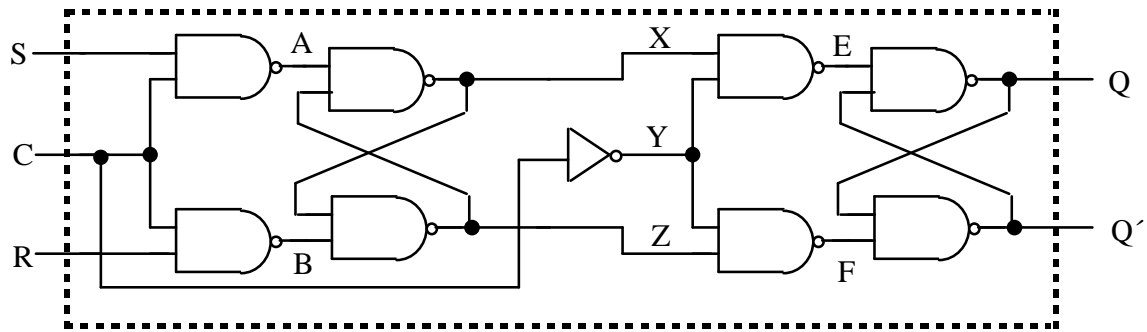


Figura 17.29. FLIP FLOP S-R maestro-esclavo en el cual se han cambiado los nombres de las variables internas.

El diagrama de tiempos mostrado en la figura 17.30 se supone que el FLIP FLOP se encuentra en el estado $Q=0$ ($Q'=1$) y las entradas externas, S y R, tienen el valor de cero. Se muestra como cambian todas las variables cuando ocurre la siguiente serie de eventos:

- La entrada S cambia al valor de 1 en el tiempo 1, manteniendo la entrada R en 0.
- La señal de reloj cambia al valor de 1 en el tiempo 2.
- La señal de reloj cambia de nuevo a 0 en el tiempo 8.

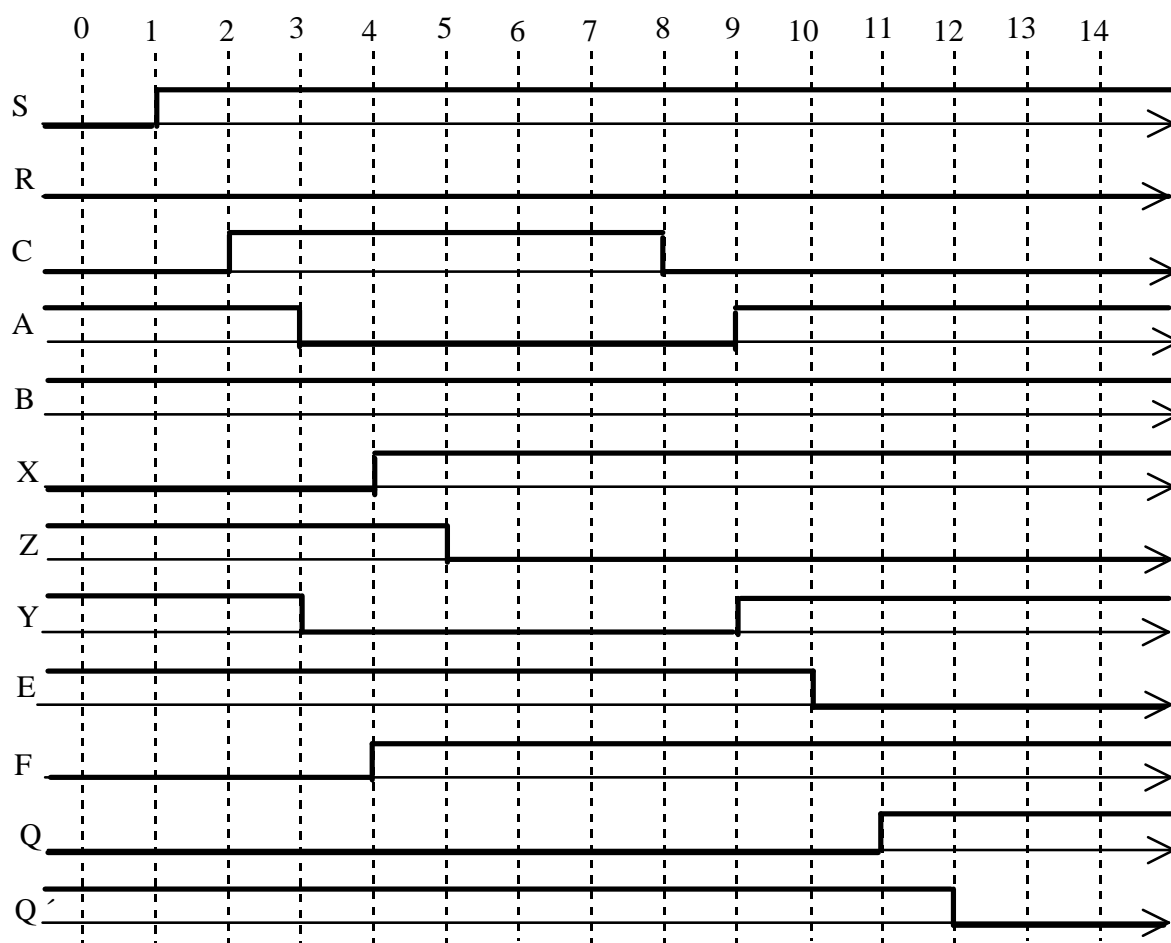


Figura 17.30. Diagrama de tiempos obtenido al activar el FLIP FLOP S-R maestro esclavo

En el diagrama de tiempos de la figura 17.30 se puede apreciar que el FLIP FLOP maestro tarda en estabilizarse desde el tiempo 2 en que aparece la señal de reloj, hasta el tiempo 4 en el cual se han estabilizado sus salidas, X y Y. Este lapso de tiempo corresponde al tiempo T_1 de la figura 17.28.

El pulso de reloj vuelve al valor de cero en el tiempo 8 y las salidas del FLIP FLOP esclavo se estabilizan hasta el tiempo 12. Por lo tanto, el tiempo T_2 de la figura 17.28 corresponde al intervalo de tiempo comprendido desde el tiempo 8 hasta el tiempo 12. Nótese que estos dos tiempos, T_1 y T_2 , no son iguales para este caso.

Para que se asegure la activación correcta del FLIP FLOP S-R maestro-esclavo es menester que sus entradas externas, S y R, permanezcan estables durante todo el tiempo que dura el pulso de reloj, es decir, desde el tiempo 2 hasta el tiempo 8 en el diagrama de tiempos. Si las entradas externas cambian del tiempo 9 en adelante, el lector podrá comprobar que no originan ningún cambio en ninguna de las variables internas ya que las señales denominadas A y B permanecerán en el valor de 1 siempre que la señal de reloj sea cero.

En la figura 17.31 se muestra en forma simplificada la estructura interna del FLIP FLOP S-R maestro-esclavo. En ésta, se han usado los símbolos para los FLIP FLOPS S-R en lugar de mostrar su arquitectura interna detallada.

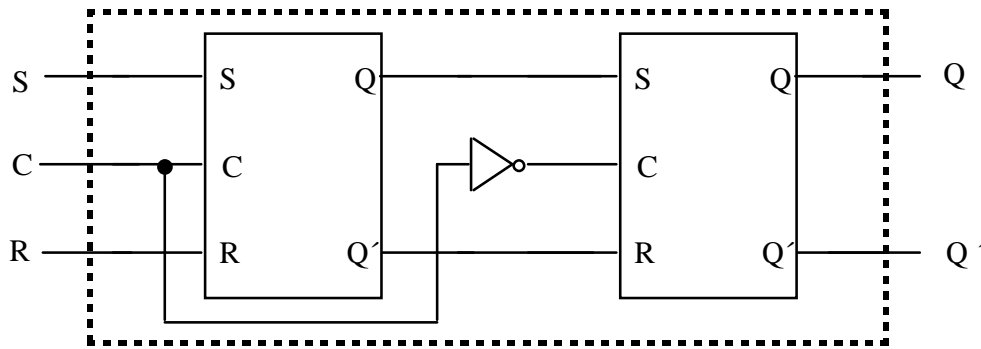


Figura 17.31. FLIP FLOP S-R maestro-esclavo activado con pulso positivo

El símbolo que se utilizará para representar un FLIP FLOP S-R maestro-esclavo activado con pulso de reloj positivo se muestra en la figura 17.32(a). El ángulo que se muestra al centro en este caso indica que las salidas cambiarán hasta que el pulso de reloj regrese al valor de 0. En la figura 17.32(b) se muestra el símbolo de un FLIP FLOP S-R maestro esclavo que trabaja con pulso de reloj negativo. Ahora el ángulo del centro indica que las salidas cambiarán hasta que el pulso de reloj regrese al valor de 1. Esto se logra colocando un bloque NOT en la entrada de reloj del maestro y suprimiendo el NOT en la entrada de reloj del esclavo.

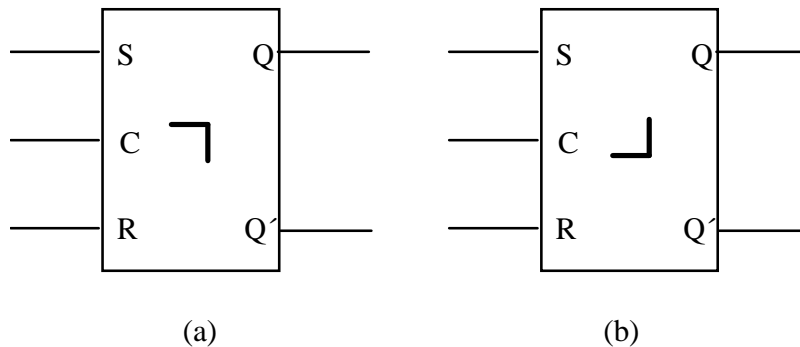


Figura 17.32. Símbolos para los FLIP FLOPS S-R maestro-esclavo.
(a) activado con pulso positivo y (b) activado con pulso negativo.

17.8. FLIP FLOP D maestro-esclavo (Master-Slave Delay Flip Flop)

La configuración interna para un FLIP FLOP D maestro-esclavo se muestra en la figura 17.33. En la figura 17.34 se muestran los símbolos para representar estos tipos de FLIP FLOP. La

operación de este FLIP FLOP es muy similar a la del FLIP FLOP S-R correspondiendo el valor de $D=0$ al caso en que se tiene $S=0$ y $R=1$, y el caso en que $D=1$ a las entradas $S=1$ y $R=0$.

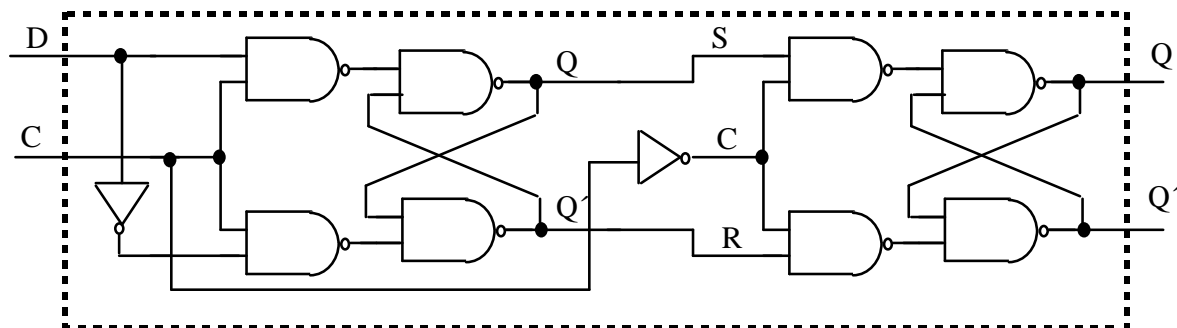


Figura 17.33. FLIP FLOP D maestro-esclavo.

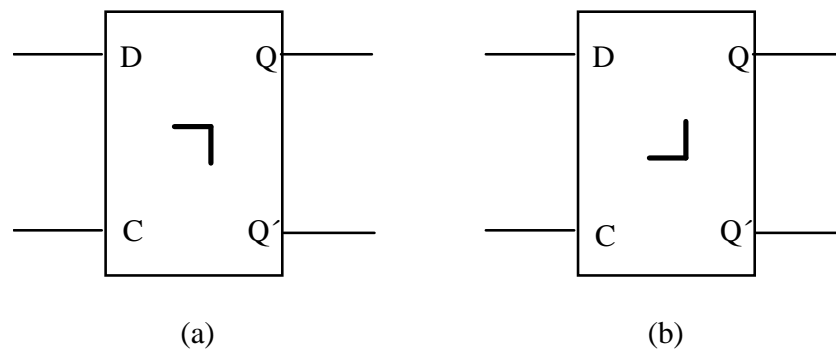


Figura 17.34. Símbolos para los FLIP FLOPS D maestro-esclavo.
(a) activado con pulso positivo y (b) activado con pulso negativo.

17.9. FLIP FLOP T maestro-esclavo (Master-Slave Trigger Flip Flop)

La configuración maestro-esclavo soluciona el problema de oscilación de las salidas del FLIP FLOP T cuando la entrada T vale 1 y el pulso de reloj tiene una duración grande. La arquitectura interna mostrada en la figura 17.35 corresponde a un FLIP FLOP T maestro-esclavo activado por un pulso de reloj positivo.

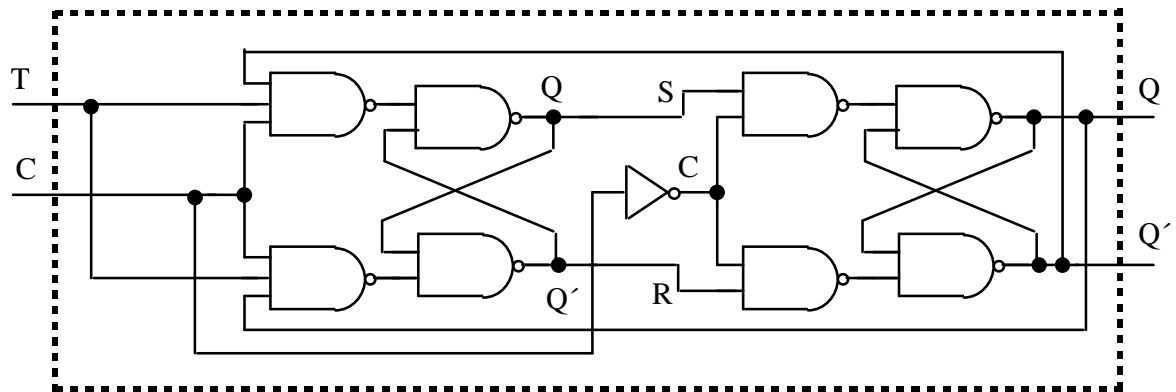


Figura 17.35. Configuración de un FLIP FLOP T maestro-esclavo

En la figura 17.37 se muestra el diagrama de tiempos para cambiar al estado de este FLIP FLOP de $Q=0$ a $Q=1$. Los nombres de algunas variables se han cambiado y se han identificado variables adicionales tal como se muestra en la figura 17.36.

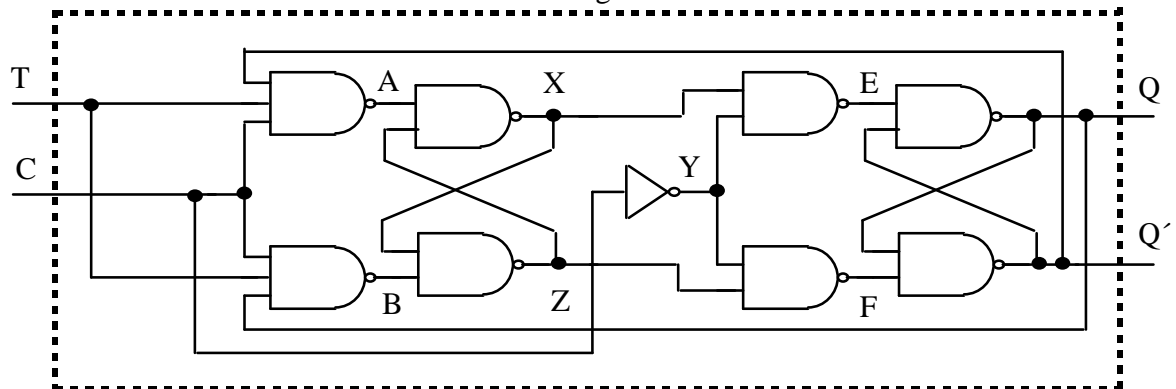


Figura 17.36. Identificación de variables del FLIP FLOP T maestro-esclavo para el diagrama de tiempos de la figura 17.37.

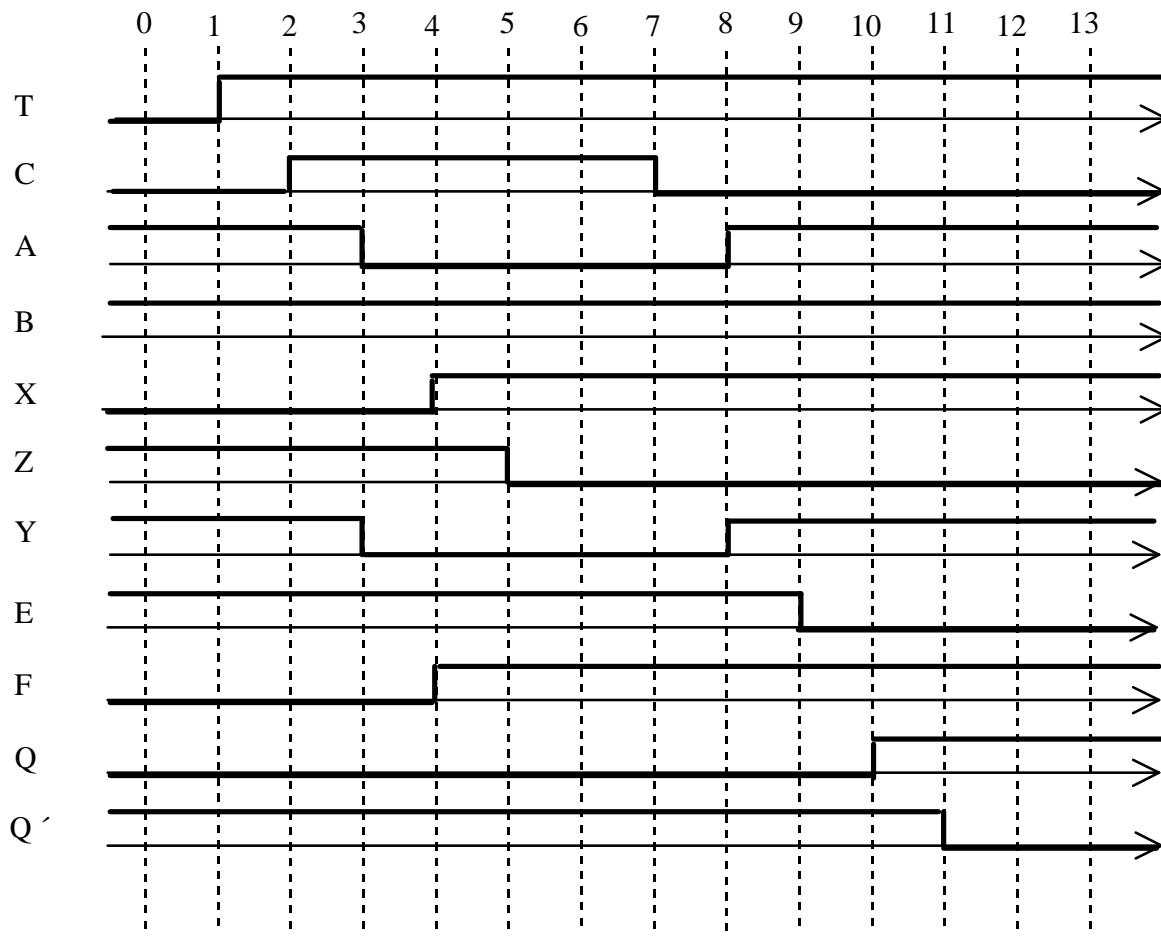


Figura 17.37. Diagrama de tiempos al cambiar de estado el FLIP FLOP T maestro-esclavo

Analizando el diagrama de tiempos de la figura 17.37 se puede observar que la entrada T vale 1 una unidad de tiempo antes de que se active la señal de reloj en el tiempo 2 y, en este mismo momento, inicia su operación del FLIP FLOP maestro. Las salidas del maestro, identificadas como X y Z, se encuentran estables en el tiempo 6. Obsérvese que hasta ahora no se ha modificado ninguna señal del FLIP FLOP esclavo.

EL pulso de reloj regresa a cero en el tiempo 7, pero el FLIP FLOP esclavo inicia su operación hasta el tiempo 8, después de que la señal de reloj se ha propagado a través de la compuerta NOT cuya salida está identificada como Y. También en este tiempo cambia de nuevo la señal A al valor de 1. De este instante en adelante, el FLIP FLOP maestro ya no cambiará porque las señales A y B permanecerán en el valor de uno.

Las salidas del esclavo se estabilizan hasta el tiempo 11 y hasta entonces estarían disponibles en la salida. Si se modifica el valor de la entrada T después de que la señal de reloj regresó al valor de cero, se puede comprobar que el FLIP FLOP maestro ya no cambiará, pues las señales A y B permanecerán en el valor de 1, independientemente del valor de T.

El símbolo que se usará para representar este FLIP FLOP se muestra en la figura 17.38.

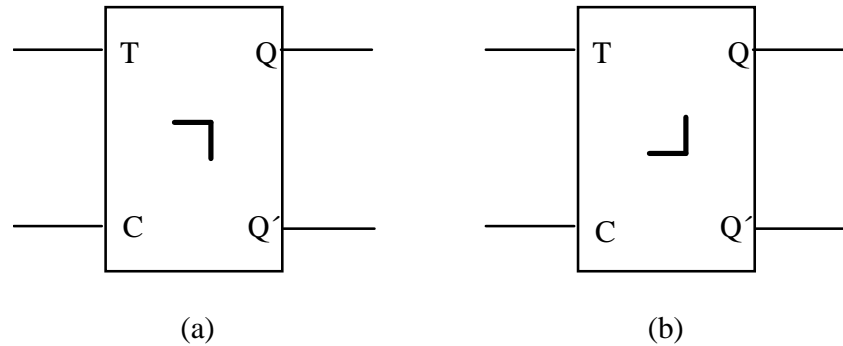


Figura 17.38. Símbolos para los FLIP FLOPS T maestro-esclavo.
(a) activado con pulso positivo y (b) activado con pulso negativo.

17.10. FLIP FLOP J-K maestro-esclavo (Master-Slave J-K Flip Flop)

La configuración maestro-esclavo también resuelve el problema que presentaba el FLIP FLOP J-K cuando ambas entradas valen 1 y se presentaba el pulso de reloj, haciendo que las salidas oscilaran mientras se mantuviera el pulso. La arquitectura interna de este FLIP FLOP se muestra en la figura 17.39.

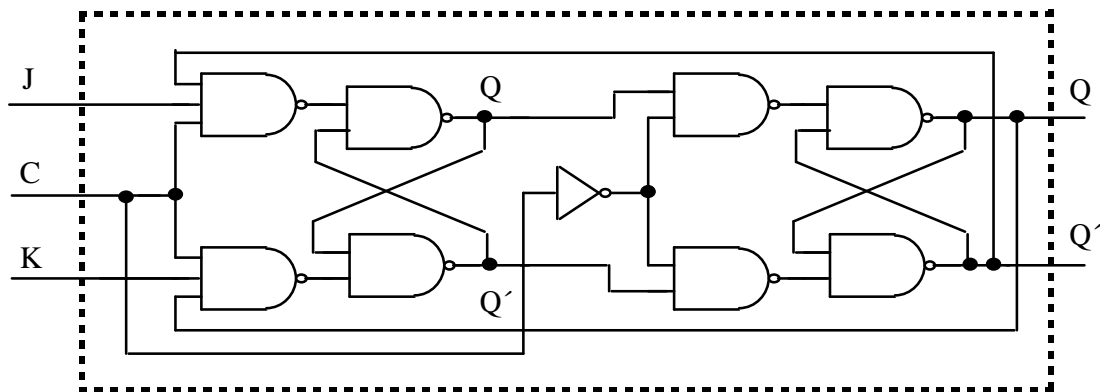


Figura 17.39. Diagrama interno de un FLIP FLOP J-K maestro-esclavo

En la figura 17.40 se presenta de nuevo el circuito interno de este FLIP FLOP, dándole nombres a todas las variables internas para poder identificarlas en los dos diagramas de tiempos que ejemplifican su operación.

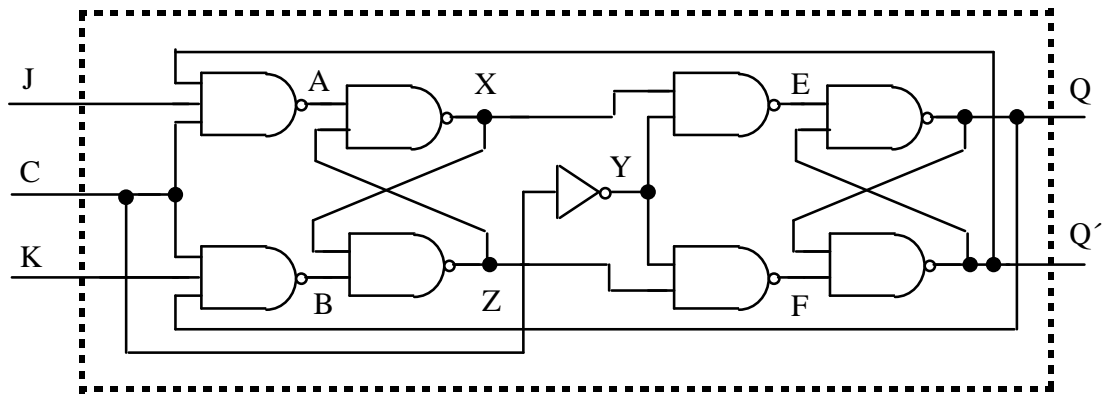


Figura17.40. Circuito del FLIP FLOP J-K maestro- esclavo con las variables mostradas en los diagramas de tiempos.

En el diagrama de tiempos de la figura 17.41 se muestra como cambian las señales al poner las entradas J y K en el valor de uno (tiempo 1) y después dar un pulso en la señal de reloj (del tiempo 2 al 7). Nótese que el FLIP FLOP maestro cambia del tiempo 2 al 5, donde se estabiliza. El FLIP FLOP esclavo todavía no ha sufrido ningún cambio. En el tiempo 8, un tiempo de propagación después de que ha regresado la señal de reloj a cero, inicia su operación el esclavo, estabilizándose hasta el tiempo 11.

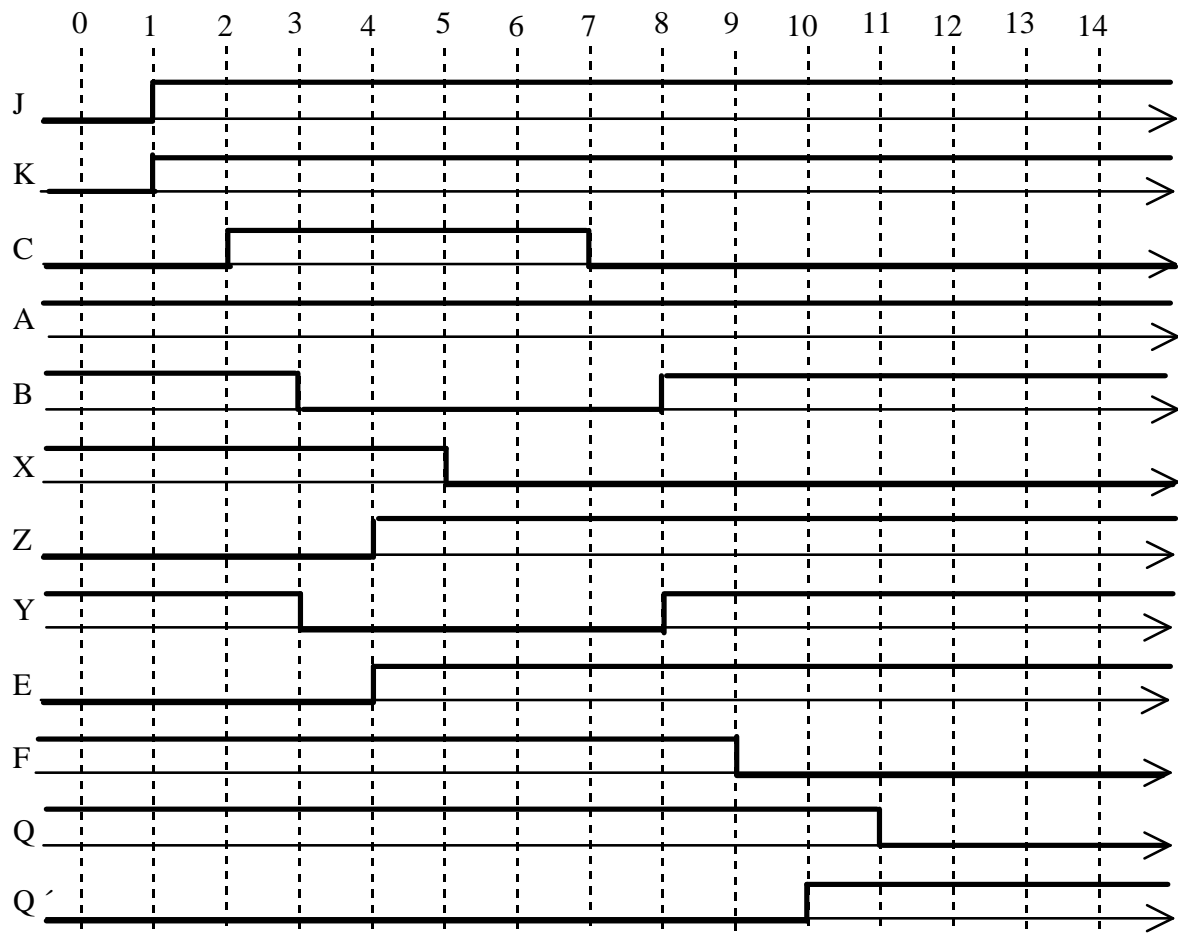


Figura 17.41. Diagrama de tiempos al cambiar el FLIP FLOP J-K maestro-esclavo de $Q=1$ a $Q=0$ poniendo las entradas J y K en 1.

Si después de que el pulso de reloj regresa a cero se modifican los valores de las entradas J y K, esto no afectará al circuito ya que las señales A y B permanecerán en el valor de 1, obligadas por la entrada de reloj en cero.

Nótese que con esta configuración se evita que las salidas del FLIP FLOP oscilen, manteniendo las salidas del esclavo constantes mientras cambia el maestro, impidiendo que las entradas al maestro se alteren. Cuando entra en operación el esclavo, las salidas del maestro ya no cambian.

En la figura 17.42 se muestra el diagrama de tiempos para el caso en que se cambia el estado de este FLIP FLOP de $Q=1$ a $Q=0$, pero ahora poniendo en uno la entrada K mientras se mantiene en cero la entrada J. Note la similitud entre ambos diagramas de tiempos. Realmente la única diferencia es que en el segundo, la señal J se mantiene todo el tiempo en cero, mientras que en el primero esta señal se cambia al valor de uno en el tiempo 1.

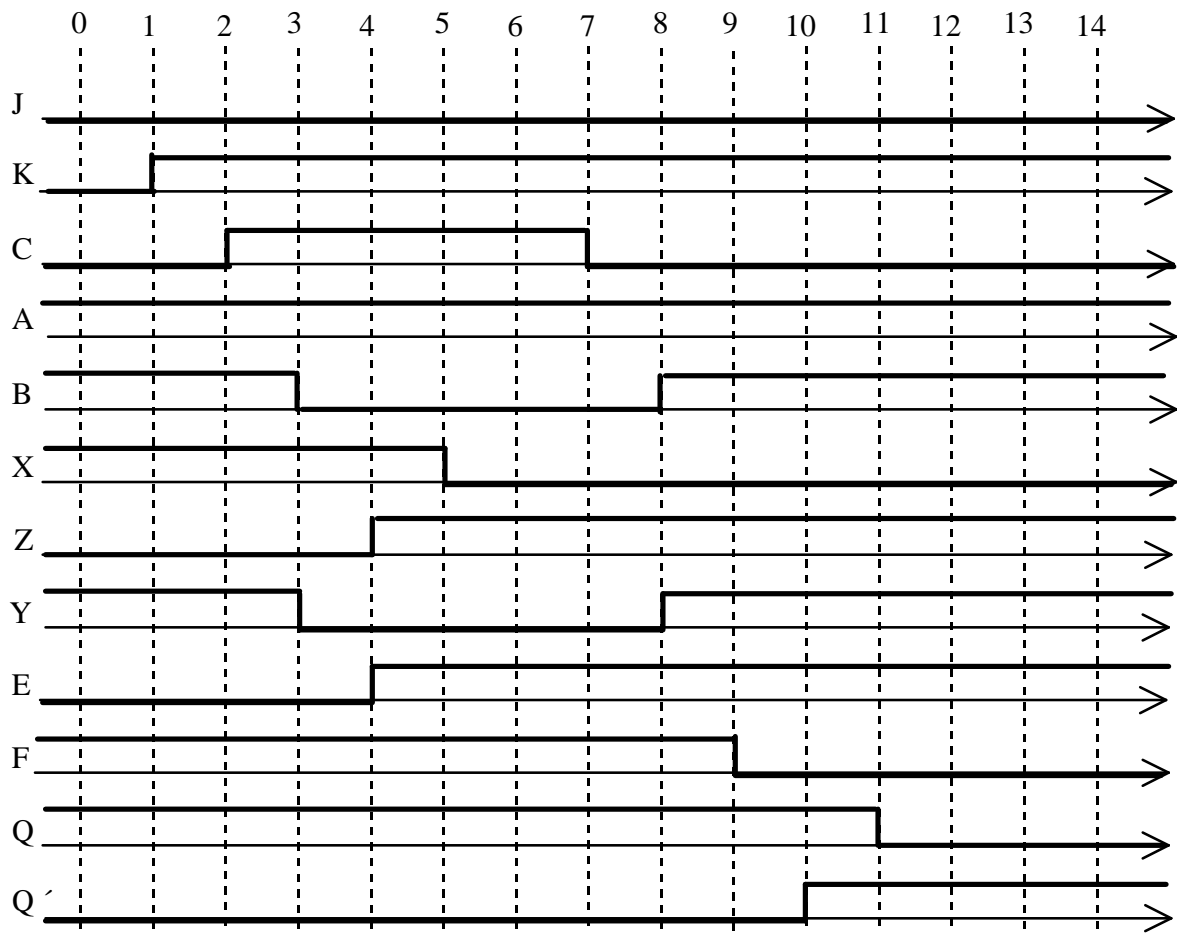


Figura 17.42. Diagrama de tiempos al cambiar el FLIP FLOP J-K maestro-esclavo de $Q=1$ a $Q=0$ poniendo las entrada K en 1 y manteniendo la entrada J en 0.

Las forma de representar un FLIP FLOP J-K maestro-esclavo se muestran en la figura 17.43.

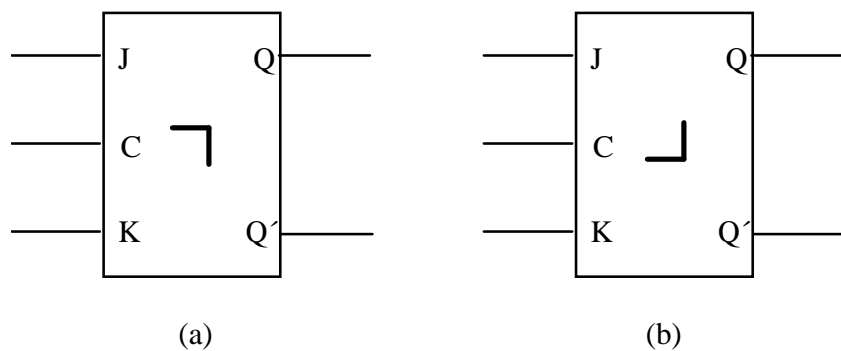


Figura 17.43. Símbolos para los FLIP FLOPS J-K maestro-esclavo. (a) activado con pulso positivo y (b) activado con pulso negativo.

17.11. FLIP FLOPS activados por transición (Edge Triggered Flip Flops).

Los FLIP FLOPS del tipo maestro-esclavo, también conocidos como FLIP FLOPS activados por pulsos, almacenan la información en el maestro cuando el pulso de reloj toma un valor y, cuando el reloj regresa al otro valor, la información es guardada en el esclavo. Esta es una desventaja cuando se desean circuitos lógicos de alta velocidad, ya que es menester esperar el tiempo necesario para que las salidas del FLIP FLOP esclavo se estabilicen antes de poderlas utilizar como entradas a otros circuitos.

Para resolver este problema se diseñaron los FLIP FLOPS activados por transiciones. Estos dispositivos están contruidos para que operen ya sea cuando el pulso de reloj pasa de 0 a 1; o bien, cuando el pulso de reloj pasa de 1 a 0. Los primeros se conocen como FLIP FLOPS activados por transición positiva y los segundos como activados por transición negativa.

Para ambos tipos de FLIP FLOPS se necesita que las entradas permanezcan estables durante un cierto tiempo antes de que se presente la transición en la señal de reloj y además, continúen estables hasta que las salidas del circuito hayan llegado al estado estable. Las salidas se estabilizarán hasta que las entradas terminen de propagarse por todas las compuertas lógicas después de que ocurrió la transición de la señal de reloj. Cuando el reloj regresa a su valor original, éstas ya no cambian. En la figura 17.44, se muestra los tiempos que importan para un FLIP FLOP activado en transición positiva (a) y para un FLIP FLOP activado en transición negativa (b).

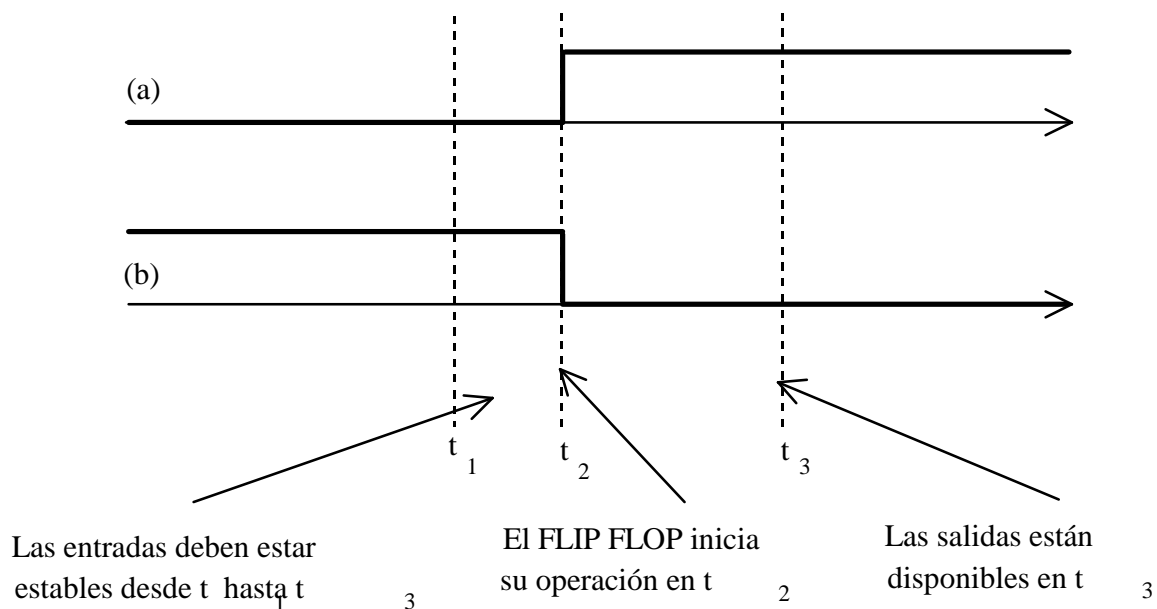


Figura 17.44. Tiempos que importan en FLIP FLOPS activados por transiciones.

En la figura 17.45 se muestran los símbolos utilizados para representar un FLIP FLOP D activado por transiciones. El circuito lógico interno se muestra en la figura 17.46.

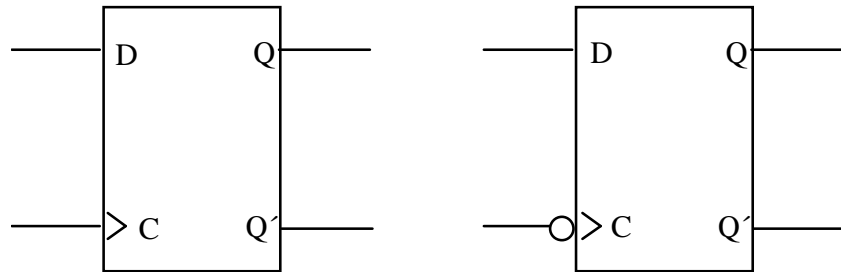


Figura 17.45. Símbolos para FLIP FLOPS D activados por (a) transición positiva, y (b) transición negativa

Este tipo de FLIP FLOPS basa su operación precisamente en el hecho de que todas las compuertas lógicas tienen aproximadamente el mismo tiempo de propagación de las señales eléctricas desde sus entradas hasta su salida. Aprovechando lo anterior y usando un ingenioso diseño, se evita la inestabilidad de las salidas, aún cuando éstas se retroalimentan a la entrada en algunos casos.

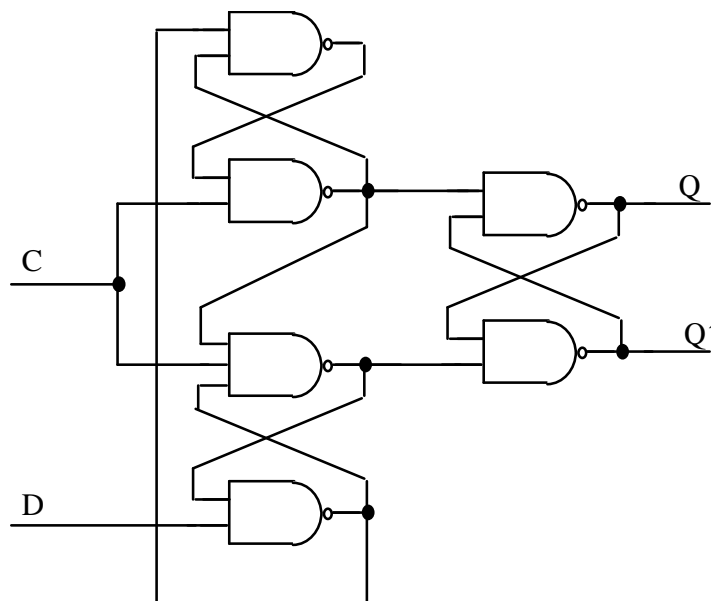


Figura 17.46. FLIP FLOP D activado por transición positiva.

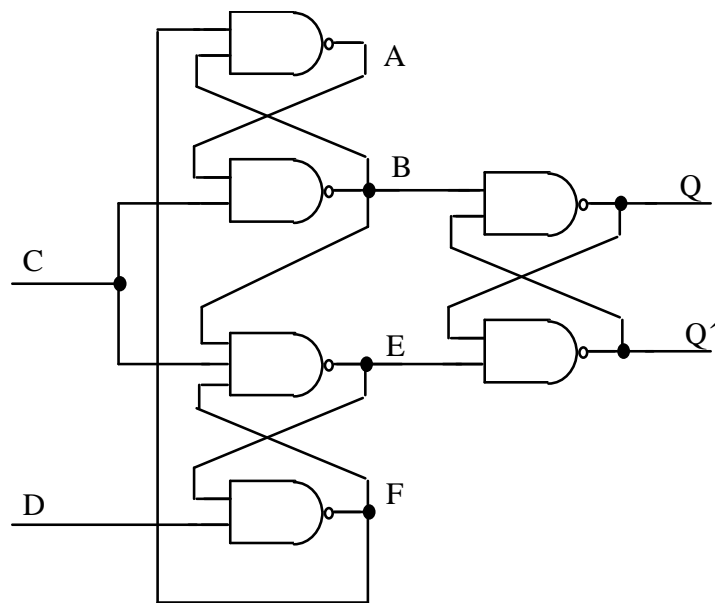


Figura 17.47. Identificación de señales para el diagrama de tiempos mostrado en la figura 17.48.

La mejor forma de entender la operación de este tipo de FLIP FLOPS es construir y analizar los diagramas de tiempos. En la figura 17.48 se muestra el diagrama generado al cambiar un FLIP FLOP D, activado por transición positiva, de $Q=0$ a $Q=1$. Los nombres asignados a las señales internas del FLIP FLOP se muestran en el circuito de la figura 17.47.

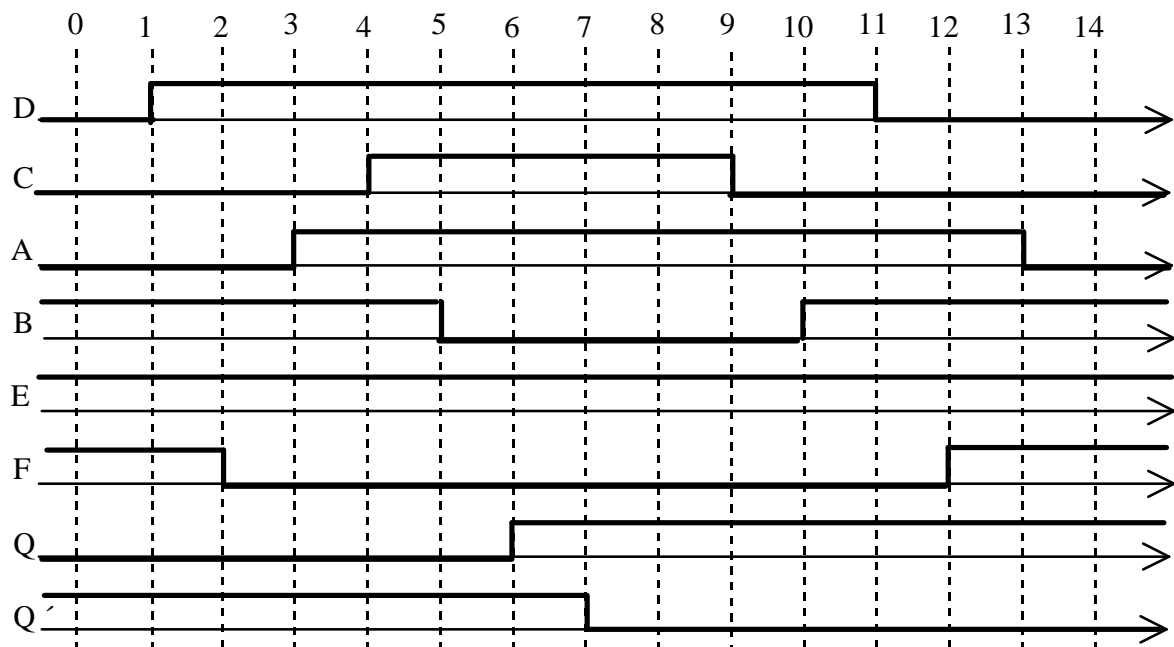


Figura 17.48. Diagrama de tiempos obtenido al cambiar el FLIP FLOP D

de la figura 17.47 del estado $Q=0$ al estado $Q=1$.

En este diagrama de tiempos se supone que todas las señales están estables en el tiempo 0. La entrada D cambia de cero a uno en el tiempo 1, manteniéndose la señal de reloj en cero. El circuito empieza a reaccionar (para estar listo cuando se presente la transición) aún cuando la señal de reloj todavía está en cero, y se estabiliza hasta el tiempo 3. Para este FLIP FLOP, es necesario que la señal D se mantenga estable al menos durante dos tiempos de propagación antes de que se presenta la transición de la señal de reloj (revítese también la figura 17.44).

La señal de reloj realiza la transición positiva (de 0 a 1) en el tiempo 4 y el circuito empieza de nuevo a reaccionar, estabilizándose en el tiempo 7. Por lo tanto, las salidas de este FLIP FLOP estarán disponibles tres tiempos de propagación después de que la señal de reloj realice una transición positiva. A partir de este instante, si la señal de reloj y la entrada D continúan en 1, las salidas del FLIP FLOP ya no cambian.

Tampoco cambiarán las salidas del circuito si la entrada D cambia ni cuando la señal de reloj regrese de nuevo a cero. En el diagrama de tiempos, la señal de reloj cambia a cero en el tiempo 9 y, aunque la señal interna A cambia de cero a uno en el tiempo 10, las salidas del circuito ya no cambian.

En el tiempo 11, la entrada D cambia a cero, lo cual ocasiona que la señal interna F cambie a uno en el tiempo 12. Sin embargo, las salidas ya no cambian y el FLIP FLOP se mantendrá en este estado hasta que la señal de reloj realice de nuevo una transición positiva.

Para el FLIP FLOP analizado, la entrada D debe mantenerse estable al menos dos tiempos de propagación antes de que ocurra la transición, hasta tres tiempos de propagación después de que ha ocurrido la transición. Si no es así, no se puede asegurar que la operación del circuito sea la correcta.

En la figura 17.49 se muestra un FLIP FLOP J-K activado por transiciones negativas. Nótese que en este caso las salidas están retroalimentadas hacia la entrada del circuito.

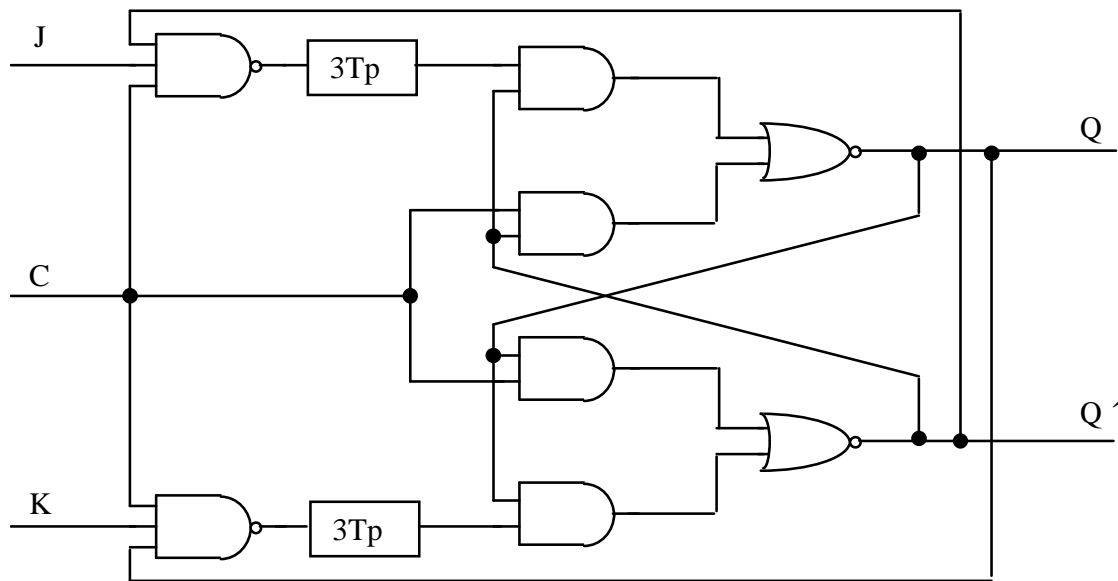


Figura 17.49. FLIP FLOP J-K activado por transición negativa.

El circuito de la figura 17.49 se muestran dos rectángulos que tienen en el interior la leyenda “3Tp”. Estos bloques representan un retardo en la señal, añadido en forma intencional, equivalente a tres veces el tiempo de propagación en una compuerta lógica para que el FLIP FLOP J-K opere correctamente al recibir una transición negativa en su entrada de reloj.

Los símbolos utilizados para representar estos FLIP FLOPS son los mostrados en la figura 17.50.

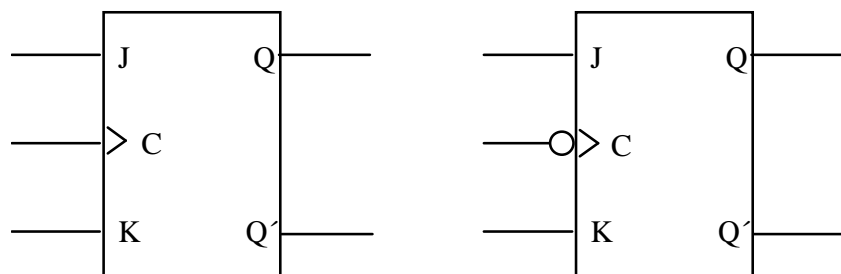


Figura 17.50. Símbolos para FLIP FLOPS J-K activados por
(a) transición positiva, y (b) transición negativa

En la figura 17.51 se muestran los nombres asignados a las variables en el circuito para relacionarlas con los diagramas de tiempos presentados en las figuras 17.52, 17.53 y 17.54.

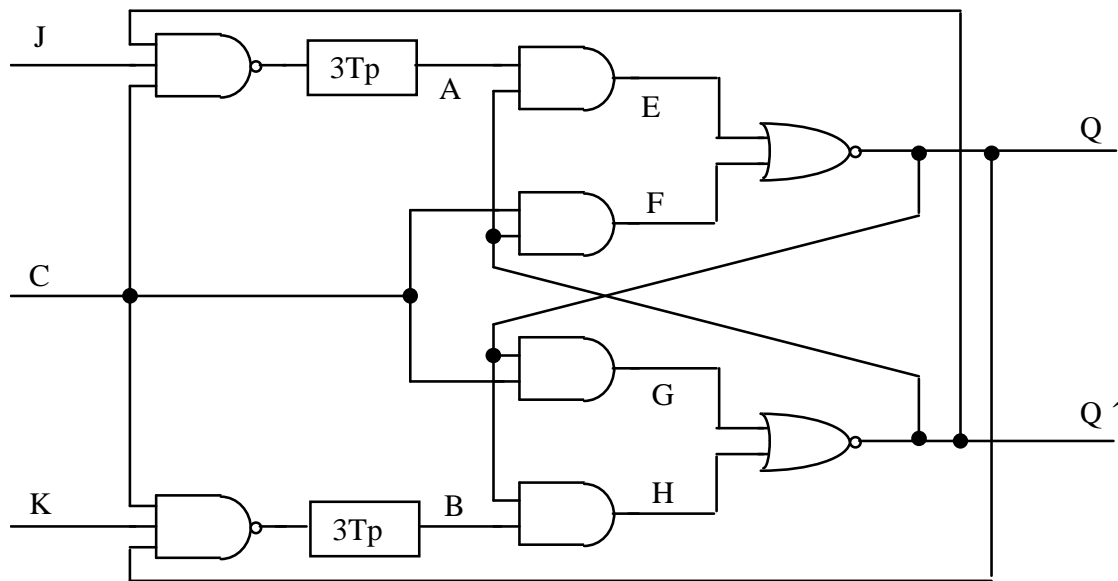


Figura 17.51. Variables internas mostradas en los diagramas de tiempos de las figuras 17.52, 17.53 y 17.54

En la figura 17.52 se muestra el diagrama de tiempos al cambiar el estado del FLIP FLOP de $Q=0$ a $Q=1$, poniendo las entradas J y K en uno y realizando una transición negativa en la entrada de reloj. Este caso es el que presentaba oscilación de las salidas en el FLIP FLOP J-K activado por nivel de la señal de reloj.

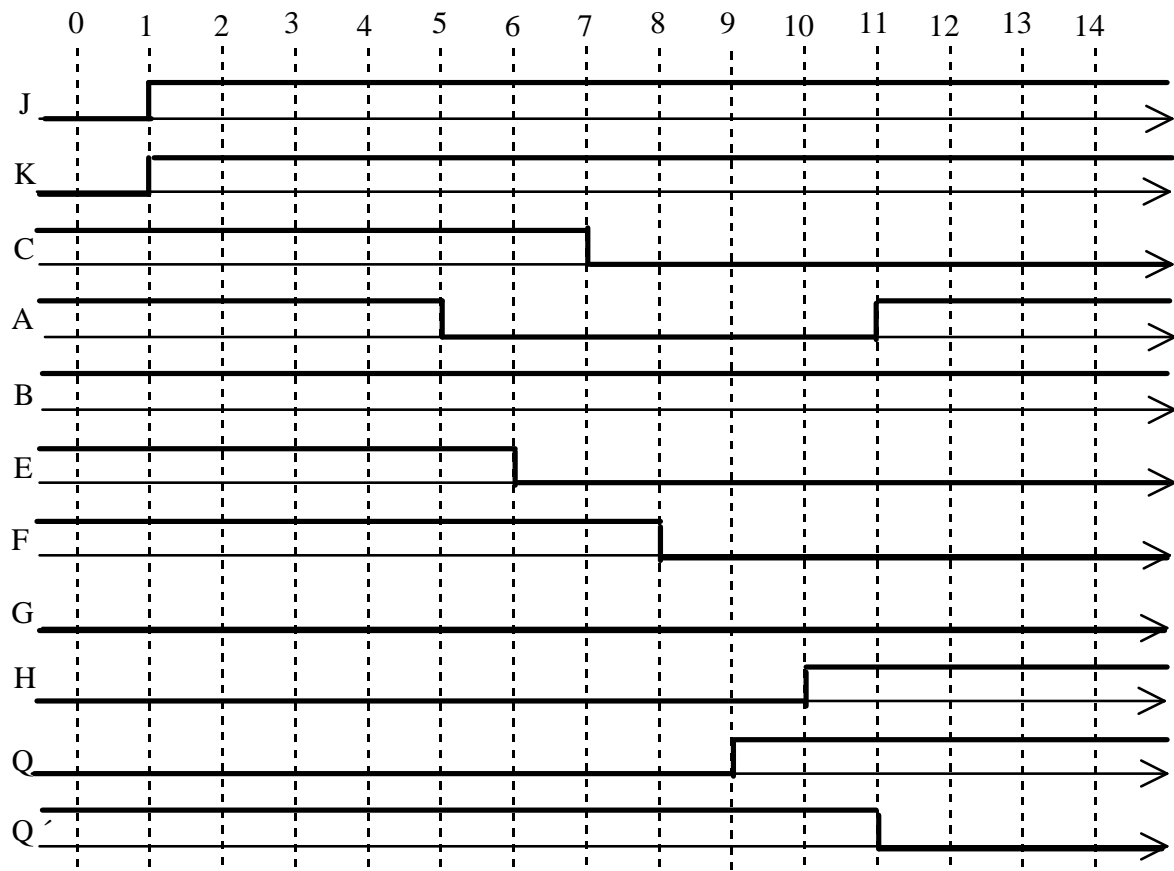


Figura 17.52. Diagrama de tiempos obtenido al cambiar el FLIP FLOP J-K de la figura 17.51 del estado $Q=0$ al estado $Q=1$, con $J=1$ y $K=1$.

Observando el diagrama de la figura 17.52, se puede ver que las entradas J y K cambian en el tiempo 1, pero las señales se estabilizan hasta el tiempo 6. La señal de reloj realiza una transición negativa en el tiempo 7 y el FLIP FLOP se estabiliza hasta el tiempo 11. A partir de este momento las salidas están disponibles y ya no cambiarán al regresar la señal de reloj nuevamente al valor de uno, ya que esto haría que la señal G tomara el valor de uno en un tiempo de propagación y que la señal B tomara ahora el valor de cero, en cuatro tiempos de propagación, lo cual no cambiaría el valor de Q' .

Si después del tiempo 11 cambian los valores de las señales J y K, las salidas del FLIP FLOP también se mantendrán.

En la figura 17.53 se muestra el diagrama de tiempos obtenido al cambiar el valor de Q de 0 a 1 mediante las entradas $J=1$ y $K=0$. Nótese que este diagrama es muy similar al anterior.

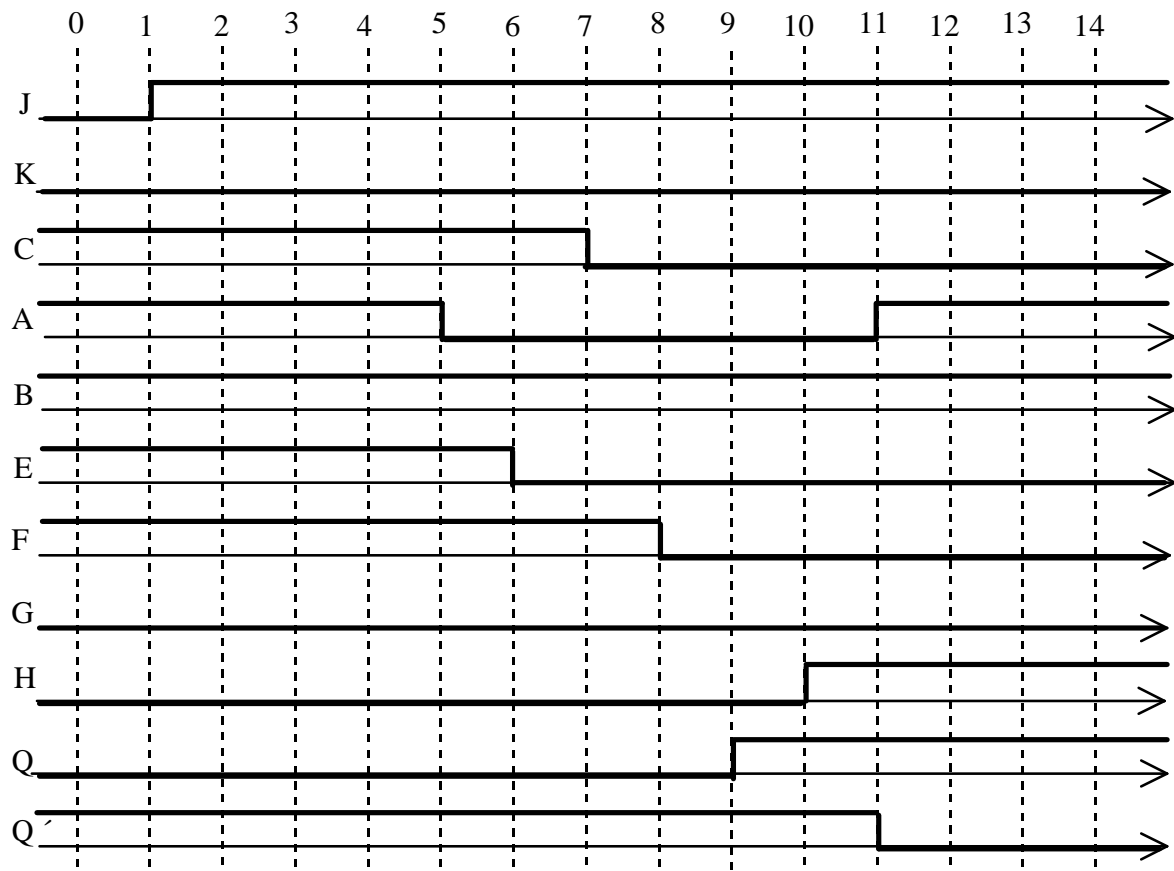


Figura 17.53. Diagrama de tiempos obtenido al cambiar el FLIP FLOP J-K de la figura 17.51 del estado $Q=0$ al estado $Q=1$, con $J=1$ y $K=0$.

En la figura 17.54 se muestra el diagrama de tiempos obtenido al cambiar el valor de Q de 1 a 0 mediante las entradas $J=0$ y $K=1$. Nótese que este diagrama es muy similar al anterior, solamente que están intercambiados los valores de A y B , E y H , F y G , y Q y Q' . La razón de esto es la simetría del circuito.

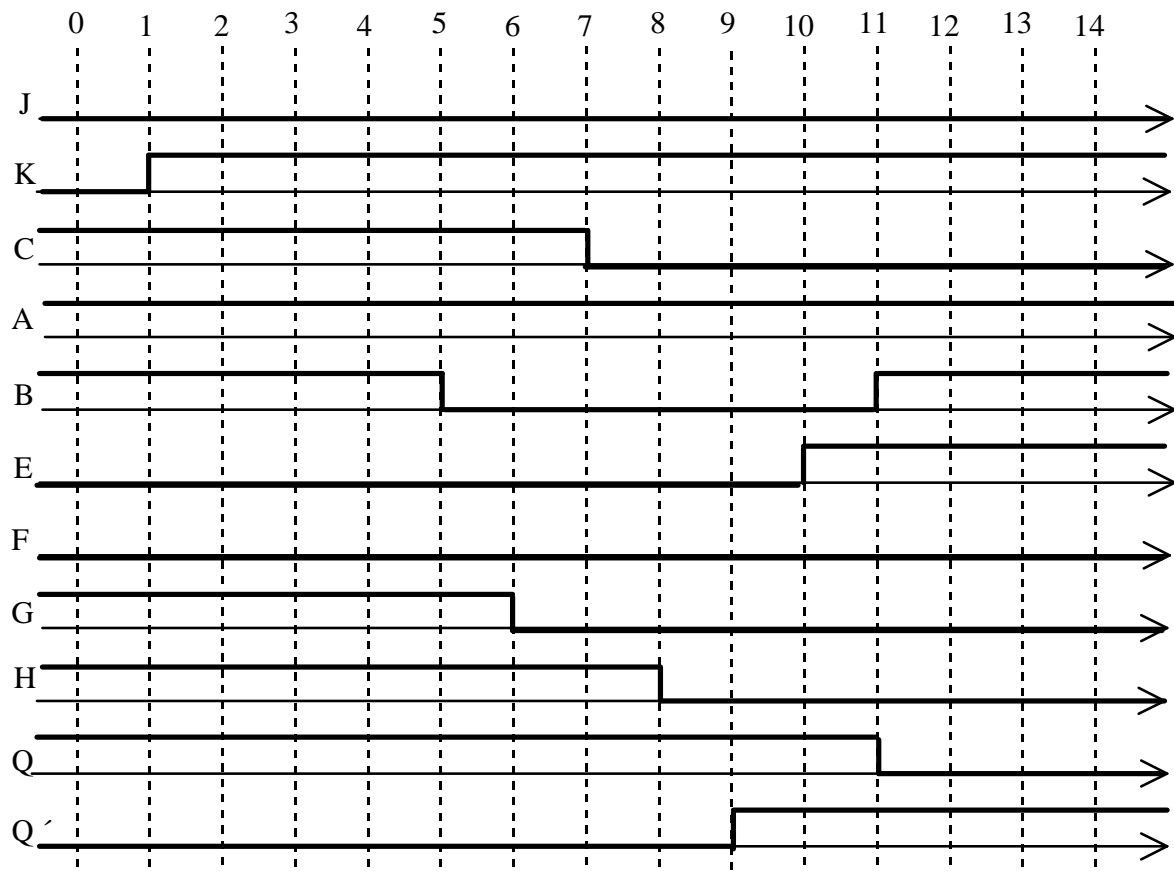


Figura 17.54. Diagrama de tiempos obtenido al cambiar el FLIP FLOP J-K de la figura 17.51 del estado $Q=1$ al estado $Q=0$, con $J=0$ y $K=1$.

Los FLIP FLOPS S-R y T, activados por transiciones, son difíciles de encontrar; sin embargo, un FLIP FLOP J-K puede trabajar como cualquiera de los dos.

17.12. Entradas asincrónicas Clear y Preset

Cuando se energiza un circuito secuencial, cada uno de los FLIP FLOPS que forman el estado queda en alguno de sus dos posibles estados estables. Debido a esto, el circuito secuencial queda en un estado aleatorio. Para que un circuito trabaje correctamente, es necesario que inicie su operación en un cierto estado predefinido y no en cualquier estado. Las entradas Clear y Preset se diseñaron para poder poner un FLIP FLOP en un estado determinado y resolver este problema.

La entrada Clear pone al FLIP FLOP en el estado $Q=0$ en el momento que se activa, independientemente del valor que tengan la señal de reloj y las demás entradas. La entrada Preset opera en forma similar a la anterior solamente que pone al FLIP FLOP en el estado $Q=1$. En la mayoría de los FLIP FLOPS, las entradas asincrónicas se accionan al ponerlas en cero. Si

se activan ambas entradas, Clear y Preset simultáneamente, el FLIP FLOP quedará en alguno de sus dos posibles estados, pero no se sabe en cual.

En la figura 17.55 se muestra el símbolo de un FLIP FLOP S-R del tipo maestro-esclavo, activado por pulso de reloj positivo y con entradas asincrónicas Clear y Preset que se activan al ponerlas en cero. La arquitectura interna de este FLIP FLOP se muestra en la figura 17.56.

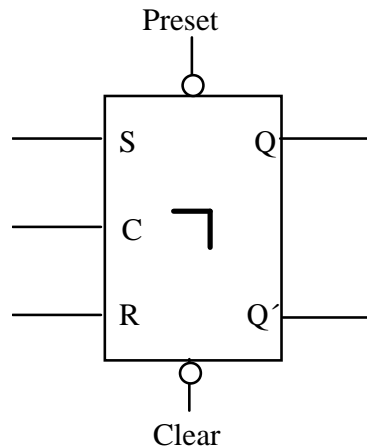


Figura 17.55. FLIP FLOP S-R maestro esclavo con entradas Clear y Preset

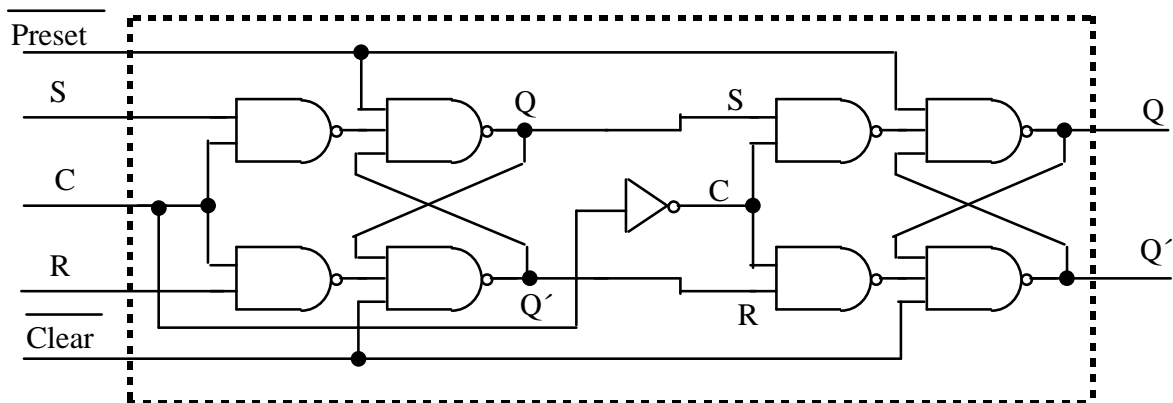


Figura 17.56. Circuito interno de un FLIP FLOP S-R maestro-esclavo activado por pulso de reloj positivo, con entradas Clear y Preset activadas con nivel lógico cero.

En la figura 17.57 se muestra la estructura interna de un FLIP FLOP J-K del tipo maestro-esclavo, activado por pulso positivo, con entradas asincrónicas Clear y Preset. EL símbolo usado para este FLIP FLOP se presenta en la figura 17.58.

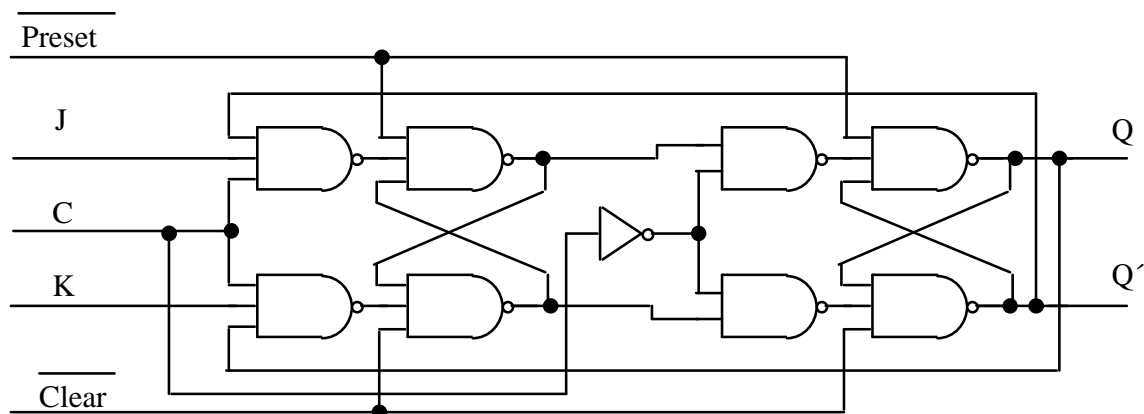


Figura 17.57. Circuito interno de un FLIP FLOP J-K maestro-esclavo, activado por pulso de reloj positivo, con entradas Clear y Preset activadas con nivel lógico cero.

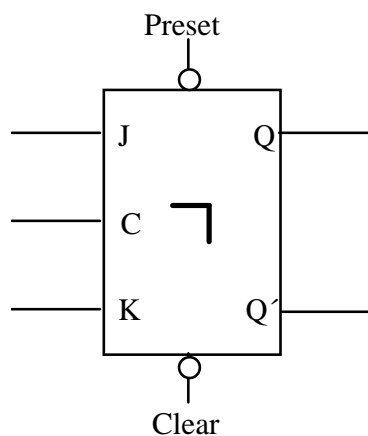


Figura 17.58. FLIP FLOP J-K maestro esclavo con entradas Clear y Preset

Por lo general, los FLIP FLOPS activados por transiciones en su entrada de reloj también cuentan con entradas asincrónicas. En las figuras 17.59 y 17.60 se han añadido las entradas Clear y Preset a los dos FLIP FLOPS activados por transiciones que se presentaron anteriormente. Obsérvese que no es tan obvio la forma de añadir las entradas asincrónicas para estos casos.

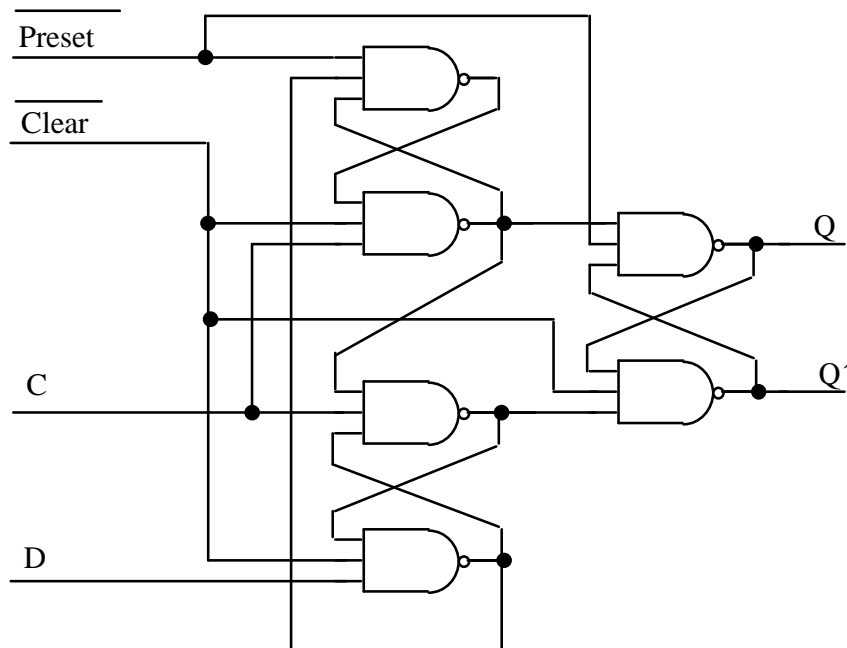


Figura 17.59. Circuito interno de un FLIP FLOP D, activado por transición positiva, con entradas Clear y Preset activadas con nivel lógico cero.

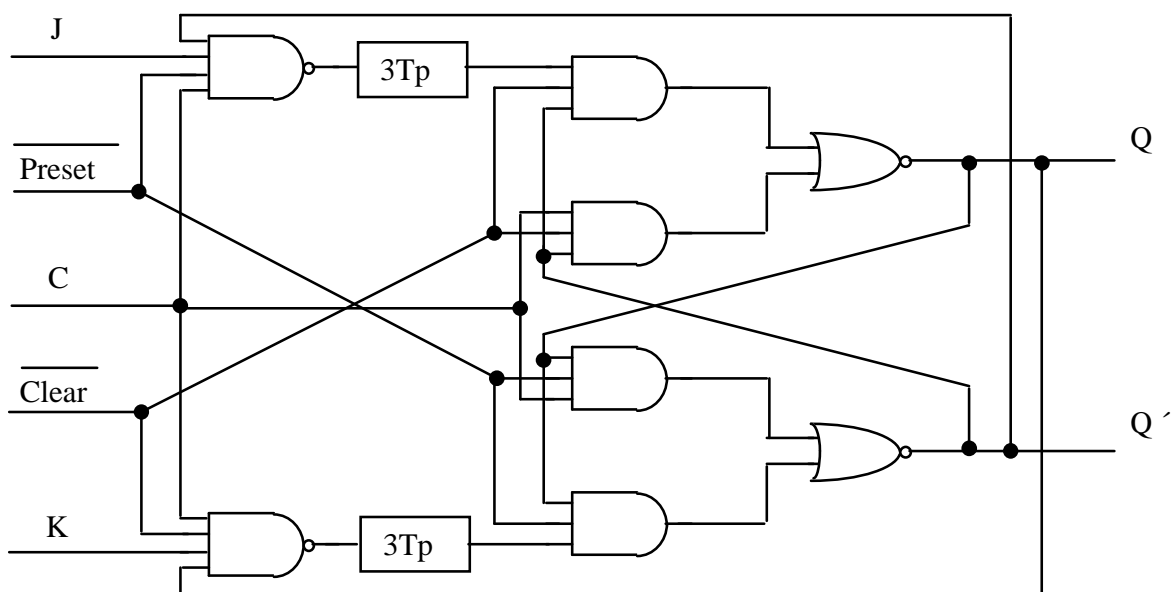


Figura 17.60. Circuito interno de un FLIP FLOP J-K, activado por transición negativa, con entradas Clear y Preset activadas con nivel lógico cero.

Las figuras 17.61 y 17.62 muestran los símbolos utilizados para representar estos dos FLIP FLOPS.

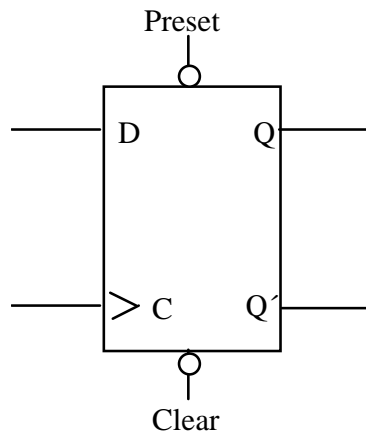


Figura 17.61. FLIP FLOP D activado por transición positiva de reloj, con entradas Clear y Preset

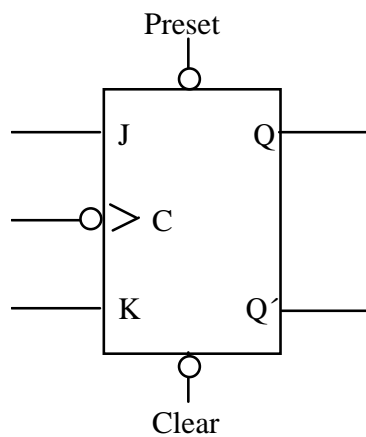


Figura 17.62. FLIP FLOP J-K activado por transición negativa de reloj, con entradas Clear y Preset

17.13. Resumen.

En este capítulo se presentó una breve introducción a los circuitos secuenciales para resaltar la necesidad de contar con circuitos lógicos que tengan memoria.

Se introdujo el concepto de la celda biestable y se analizó su construcción utilizando compuertas lógicas combinatorias. Para analizar su funcionamiento, se introdujeron los diagramas de tiempo y se presentaron algunos ejemplos del análisis de la operación de varios LATCHES.

Posteriormente se comentó la principal característica de los circuitos secuenciales sincrónicos, que contempla la existencia de una señal maestra de reloj que es utilizada para sincronizar el momento en que cambia cada una de las celdas biestables que forman el estado interno de un

circuito secuencial. Para lograr esta sincronización, se añadió la entrada de reloj a los elementos biestables, teniéndose las celdas de memorias o FLIP FLOPS.

Se analizó el funcionamiento de los FLIP FLOPS S-R, D, T y J-K, utilizando diagramas de tiempo. Se destacó que estos dispositivos operan de acuerdo al nivel de la señal de reloj. Se hizo patente el problema de oscilación que presenta el FLIP FLOP T cuando la entrada de reloj toma el valor de uno y la entrada T vale también uno. El mismo problema se encontró en el FLIP FLOP J-K cuando sus entradas valen ambas uno y la señal de reloj toma el valor de uno.

Este problema se presenta siempre que se utilizan FLIP FLOPS activados por nivel de la señal de reloj y se retroalimentan los estados para determinar el siguiente estado. Para resolverlo, se diseñaron los FLIP FLOPS del tipo maestro-esclavo. En éstos, se tienen internamente dos LATCHES. El maestro cambia cuando la señal de reloj toma uno de los valores, pero no cambia el esclavo. Cuando la señal de reloj regresa al otro valor, el esclavo cambia, modificando las salidas del FLIP FLOP, pero el maestro ya no cambia. Estos FLIP FLOPS son activados por pulsos en la señal de reloj.

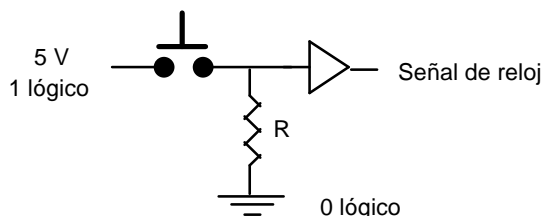
Aún cuando los FLIP FLOPS maestro-esclavo resuelven efectivamente los problemas de la retroalimentación, su operación está sujeta a un pulso completo de la señal de reloj. Esta característica los hace inapropiados para circuitos secuenciales de alta velocidad. Si se desea un mejor tiempo de respuesta de los FLIP FLOPS, se deben utilizar los diseñados para operar con transiciones en la señal de reloj, ya que éstos no necesitan que la señal de un pulso completo., sino que se activan cuando la señal de reloj cambia de nivel. Se presentaron solamente un FLIP FLOP D, activado por transición positiva, y un FLIP FLOP J-K, activado por transición negativa.

Finalmente, se añadieron las entradas asincrónicas Clear y Preset, cuya operación no está sujeta a la entrada de reloj y que permiten poner un FLIP FLOP en un estado determinado. La necesidad de estas entradas estriba en que los circuitos secuenciales deben iniciar su operación en un estado específico, no en cualquier estado. Por lo general, al energizar un circuito, se activan secuencias que lo colocan en el estado en que debe iniciar su operación, aprovechando estas entradas asincrónicas.

17.14. Problemas.

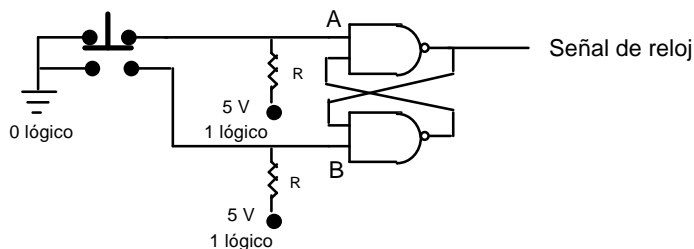
1. Mencione la principal diferencia entre un circuito combinatorio y un circuito secuencial.
2. Construya un FLIP-FLOP T a partir de un FLIP-FLOP D, usando la mínima cantidad de bloques lógicos adicionales al FLIP-FLOP D.
3. Construya un FLIP-FLOP D a partir de un FLIP-FLOP T, usando la mínima cantidad de bloques lógicos adicionales al FLIP-FLOP T.
4. Construya un FLIP-FLOP JK a partir de un FLIP-FLOP T, usando la mínima cantidad de bloques lógicos adicionales al FLIP-FLOP T.

5. Construya un FLIP-FLOP JK a partir de un FLIP-FLOP D, usando la mínima cantidad de bloques lógicos adicionales al FLIP-FLOP D.
6. Si se utiliza un PUSH-BUTTON para generar una señal de reloj como se muestra en la siguiente figura, ¿Qué problemas se generan?



Cuando no se oprime el botón, la entrada de la compuerta se conecta a tierra por medio de la resistencia, se recomienda un valor de R entre $1\text{ K}\Omega$ a $10\text{ K}\Omega$ dependiendo de la fuente de poder que se tenga. Para analizar el circuito debe considerarse que entra un cero lógico cuando el botón no está oprimido (0 Volts). Al oprimir el botón se conectan los 5 Volts (1 lógico) en la entrada de la compuerta, no se produce un corto circuito debido a la resistencia por la cual fluye una corriente que depende del valor de la resistencia que se haya seleccionado.

7. Una posible circuito para generar una señal de reloj en forma manual es el siguiente:



Obtenga un diagrama de tiempos donde se muestre la señal de reloj así como las señales marcadas como A y B al momento de oprimir y soltar el botón.

Cuando no se oprime el botón la entrada B de la compuerta inferior se conecta a 5 Volts (1 lógico) por medio de la resistencia (se recomienda un valor de R entre $1\text{ K}\Omega$ a $10\text{ K}\Omega$ dependiendo de la fuente de poder que se tenga), la entrada A tendría una entrada de 0 Volts (0 lógico). Al oprimir el botón, la entrada B se conectan a tierra (0 lógico) y, no se produce un corto circuito debido a la resistencia donde fluye una corriente que depende del valor de la resistencia que se haya seleccionado. La entrada B tendría una entrada de 5 Volts (1 lógico).

CAPITULO 18

DISEÑO DEL SISTEMA DE MEMORIA

18.1 Introducción.

La memoria principal de una computadora de propósito general normalmente está formada en gran parte por circuitos de memoria **RAM** (del inglés Random Access Memory), y solo una pequeña parte por circuitos de memoria **ROM**. Originalmente se utilizaba el termino **RAM** para hacer referencia a memorias de acceso aleatorio, pero ahora se utiliza para identificar a las memorias volátiles donde se puede leer y escribir información, y así diferenciarla de una memoria **ROM**. Nótese que ambas memorias son de acceso aleatorio.

En este capítulo se presenta la construcción interna de una memoria **RAM**, así como el esquema de decodificación que se utilizaría para construir un sistema de memoria, partiendo de una serie de circuitos de memoria **RAM** y **ROM**. En el capítulo 15 se presenta la construcción interna de una memoria **ROM**.

18.2 Memoria RAM

Al igual que una memoria **ROM**, la memoria **RAM** puede ser considerada como un conjunto de posiciones de memoria, solo que en cada una de éstas se puede leer información y también se puede escribir. Cada una de estas posiciones de memoria se identifica por una dirección única, la cual recibe el nombre de palabra. En cada palabra se puede leer o almacenar un número fijo de bits, los cuales pueden representar una instrucción o dato.

El número de palabras que contiene una memoria siempre es una potencia de 2. Además existen varias abreviaciones para indicar algunas potencias de 2, la letra **K** (de Kilo) equivale a $2^{10} = 1,024$ palabras, la letra **M** (de Mega) equivale a $2^{20} = 1,048,576$ palabras, la letra **G** (de Giga) equivale a $2^{30} = 1,073,741,824$ palabras y la letra **T** (de Tera) equivale a 2^{40} palabras.

Una memoria **RAM** tiene como entradas; un bus de direcciones, para especificar la palabra con la cual se desea trabajar, y dos líneas de control; una para habilitar o deshabilitar la memoria, llamada selector de circuito o habilitador del circuito y la otra para indicar si se desea realizar una lectura o una escritura llamada R/\overline{W} (del inglés Read/Write, se usa \overline{W} para indicar que la operación de escritura se realiza cuando en esta línea se tiene un cero.). Además cuenta con un bus de datos el cual es normalmente bidireccional, se comporta como bus de entrada en la operación de escritura y como bus de salida en la operación de lectura. La principal razón de este bus bidireccional es reducir el número de conexiones externas en el circuito integrado. La figura 18.1 ejemplifica una memoria **RAM** de n líneas en el bus de dirección y de m líneas en el bus de datos.

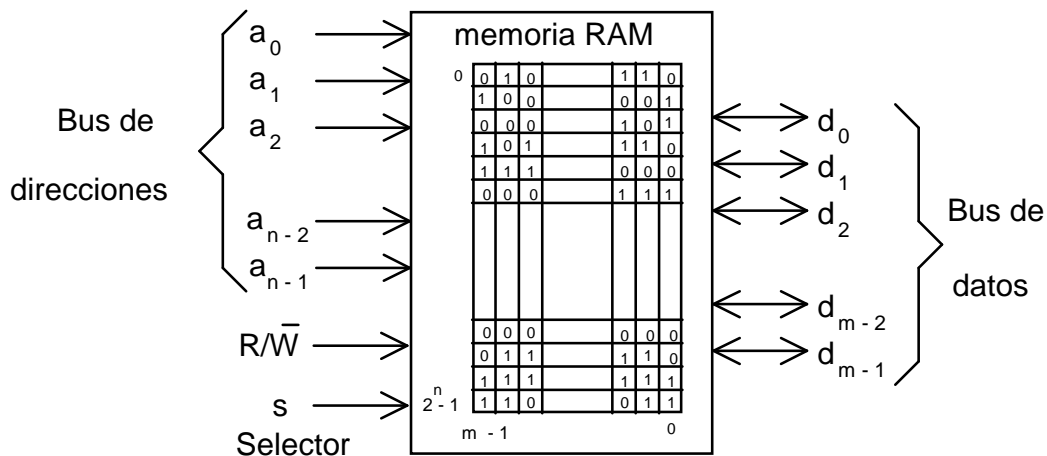


Figura 18.1. Memoria **RAM** de $2^n \times m$ bits

La construcción interna de una memoria **RAM** de 2^n palabras de m bits consta de una lógica de direccionamiento y $2^n \times m$ celdas binarias (**cb**), en cada una de estas celdas binarias se puede almacenar un bit de información y constituye el bloque básico de una memoria **RAM**.

18.2.1 Celda binaria

Existen dos modos de construir este bloque básico, estático y dinámico.

En las memorias **RAM** estáticas, la celda binaria se construye esencialmente con un LATCH S-R, la información no se pierde mientras el circuito permanece encendido.

Las celdas binarias de la memoria **RAM** dinámica están construidas básicamente por medio de un capacitor que almacena la información y un transistor. El transistor es un dispositivo electrónico que permite cargar o descargar el capacitor, así como poder leer el valor que tiene éste. La carga almacenada en el capacitor tiende a desaparecer al momento de leerla, por lo cual siempre que se lee una celda binaria con valor de uno, debe ser seguida de una operación de escritura de un uno y así restaurar la carga en el capacitor, esta operación de escritura la realiza en forma automática los circuitos de control de la **RAM**.

Incluso si la celda binaria nunca se lee, la carga almacenada tiende a desvanecerse con el tiempo, típicamente en pocos milisegundos. Para evitar que se pierda la información, las memorias dinámicas deben de ser restauradas a intervalos regulares. A este proceso se le conoce como refresco de la memoria. Una celda se puede refrescar simplemente ejecutando una operación de lectura.

Las memorias **RAM** dinámicas se pueden refrescar mediante un circuito externo, que lea una secuencia de direcciones. Normalmente no es necesario leer todas las direcciones ya que al leer una dirección se realiza el refresco a una serie de direcciones. En la actualidad existen memorias

RAM dinámicas que realizan el refresco en forma transparente, a estas memorias se les llama **RAM** dinámica sincrónica. En una computadora el refresco de la memoria **RAM** se realiza mientras el **RAM** está inactivo.

Al ser la celda binaria dinámica más sencilla en su construcción se pueden tener circuitos integrados con mayor capacidad y menor consumo de energía, su principal desventaja es que tienen un mayor tiempo de acceso.

El funcionamiento lógico de una celda binaria estática o dinámica es el mismo, la figura 18.2 muestra la lógica equivalente de una celda binaria.

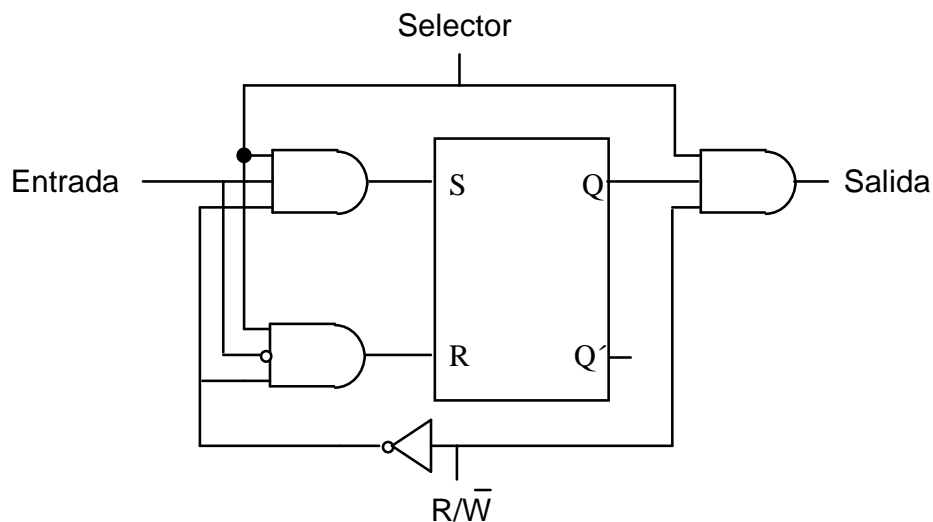


Figura 18.2. Lógica equivalente de una celda binaria.

Nótese que para poder hacer una escritura en la celda es necesario poner en la línea de R/\overline{W} un 0, en el selector un 1 y el dato que se desea escribir en la entrada. Si el dato que se desea escribir es un 1 se dará un SET en el LATCH y si el dato es un 0 se dará un RESET en el LATCH.

Para tener el valor de la celda en la línea de salida, es necesario poner un 1 en la línea de R/\overline{W} y un 1 en el selector.

La figura 18.3 muestra el diagrama de bloque de una celda binaria que se usará en la siguiente sección.

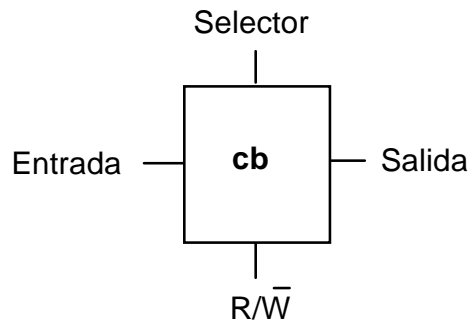


Figura 18.3. Diagrama de bloque de una celda binaria.

18.2.2 Memoria RAM de $N \times M$

La construcción interna de una memoria **RAM** de 2^n palabras de m bits básicamente consta de un decodificador de $n \times 2^n$ y de $2^n \times m$ celdas binarias.

La construcción lógica de un **RAM** pequeño se muestra en la figura 18.4. Esta memoria cuenta con 4 palabras de 4 bits. Para seleccionar una de las cuatro palabras se requiere de un bus de direcciones de 2 líneas, las cuales entran al decodificador. Al estar habilitado el decodificador seleccionará las celdas binarias de una de las cuatro palabras, dependiendo del contenido en el bus de direcciones. Si la operación que se desea realizar es una escritura, se pone en la línea de $\overline{R/W}$ un 0 y las celdas binarias seleccionadas guardarán los bits que se encuentran en el bus de entrada. Para realizar una operación de lectura, se pone en la línea de $\overline{R/W}$ un 1, lo que hace que los bits de las celdas binarias seleccionadas pasen por los bloques **ORs** al bus de salida. Las celdas binarias de las palabras no seleccionadas mantienen su contenido.

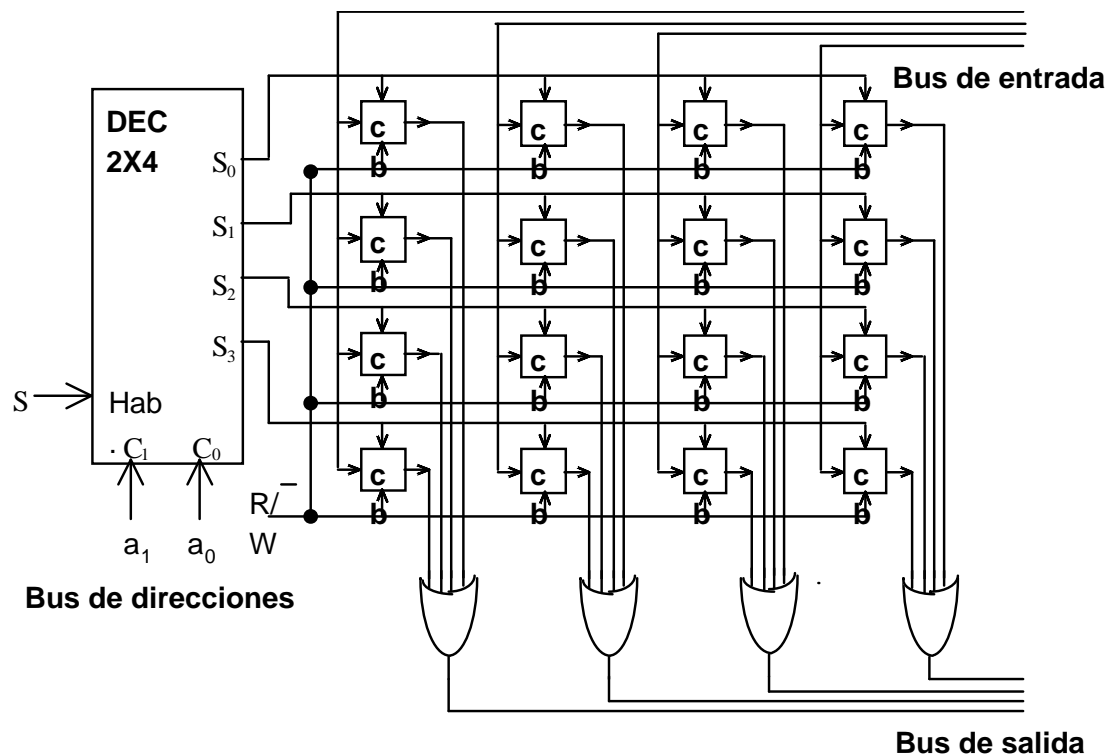


Figura 18.4. Memoria RAM de 4 palabras de 4 bits.

Para poder unir el bus de entrada y de salida en uno sólo es necesario introducir una nueva compuerta, esta compuerta es el buffer de tres estados. En la figura 18.5 muestra el símbolo usado para esta compuerta, así como su funcionamiento.

Esta compuerta tiene una entrada de control y una entrada norma, cuando la entrada de control tiene el valor de uno la salida de la compuerta tiene el valor que se encuentra en la entrada, el cual puede ser un uno o un cero. Si la entrada de control vale cero, la salida pasa a un estado de alta impedancia, la cual equivale a desaparecer la compuerta, separando la entrada de la salida. El estado de alta impedancia que puede tener la salida es la característica que hace especial a esta compuerta.

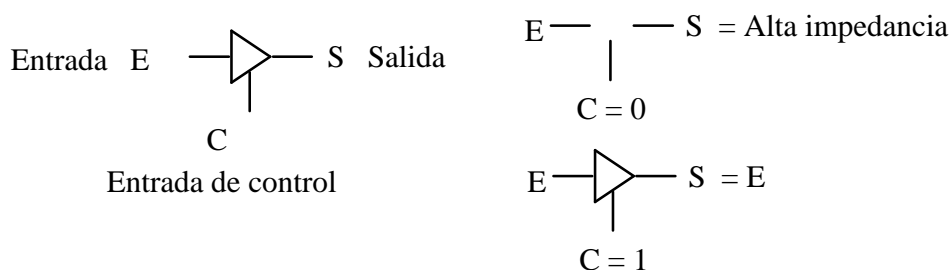


Figura 18.5. Símbolo y funcionamiento del buffer de tres estados.

En la figura 18.6 muestra el circuito usado para hacer que una línea de un bus se pueda comportar como línea de entrada o como línea de salida dependiendo de una señal de control.

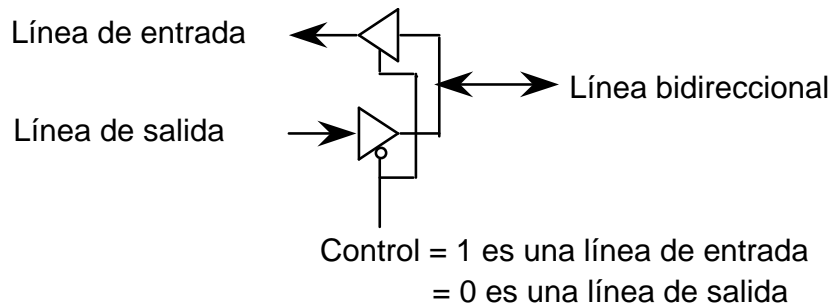


Figura 18.5. Línea bidireccional.

En la figura 18.6 la memoria RAM de 4 palabras de 4 bits con bus de datos bidireccional, nótese que si la señal selectora S vale cero el bus de datos se encuentra en alta impedancia. Al tener esta construcción el bus de datos se dice que es bidireccional y de tres estados. Una construcción semejante a esta tienen las memorias ROM donde su bus de datos es unidireccional y de tres estados. El tener bus de datos de tres estados será muy útil cuando se intente conectar varios circuitos de memoria a un mismo bus de datos.

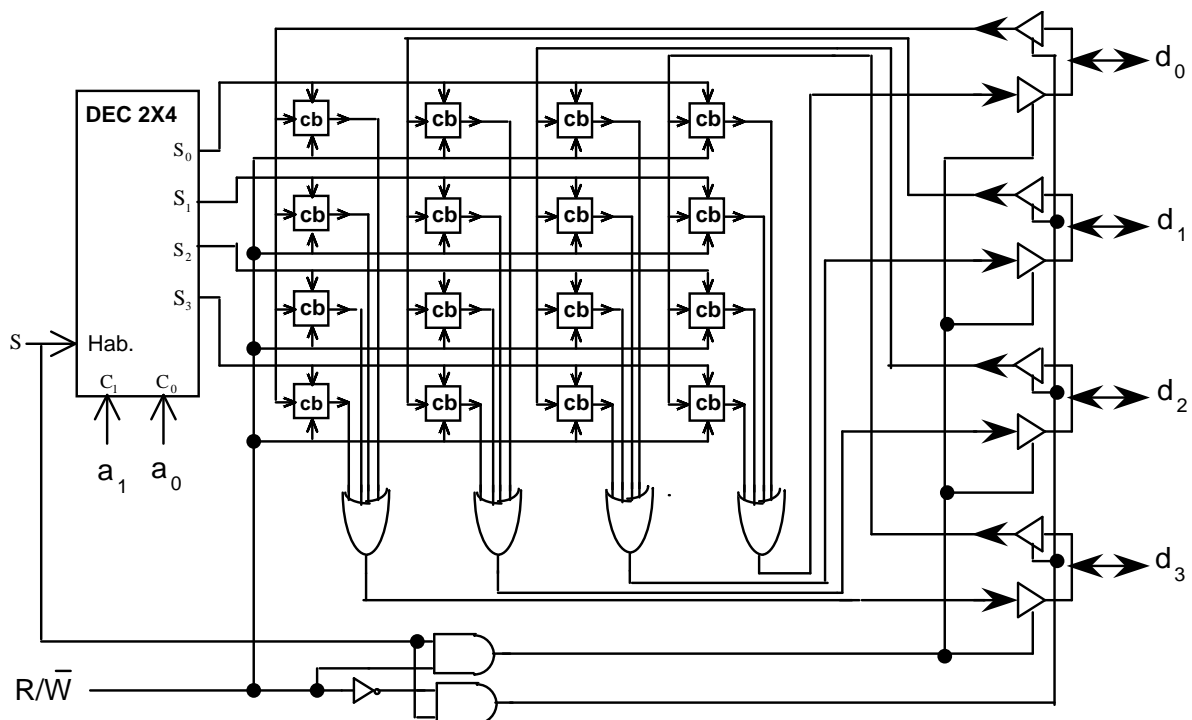


Figura 18.6. Memoria RAM de 4 palabras de 4 bits con bus de datos bidireccional

La construcción lógica de una memoria con mayor capacidad, es una extensión directa de la memoria que aquí se presenta.

La figura 18.7 muestra el bloque que se usa para representar a esta memoria RAM, así como el bloque de una memoria RAM de 1K palabras de 8 bits. Con frecuencia en las memorias comerciales se encuentra más de una línea de selección, que están unidas a un AND internamente y la salida de éste es la que habilita al decodificador. La razón de esto es que muchas veces facilita la conexión de varios circuitos de memoria y que en el circuito integrado quedaban conexiones externas sin uso. Para el caso de la RAM de 1K se debe de poner la línea CS₀ (CS del inglés Chip Select) en 1 y la línea CS₁ en 0, para habilitar al circuito. Las líneas mas gruesas representa los buses y se acostumbra a indicar el número de líneas que contiene encima del símbolo /.

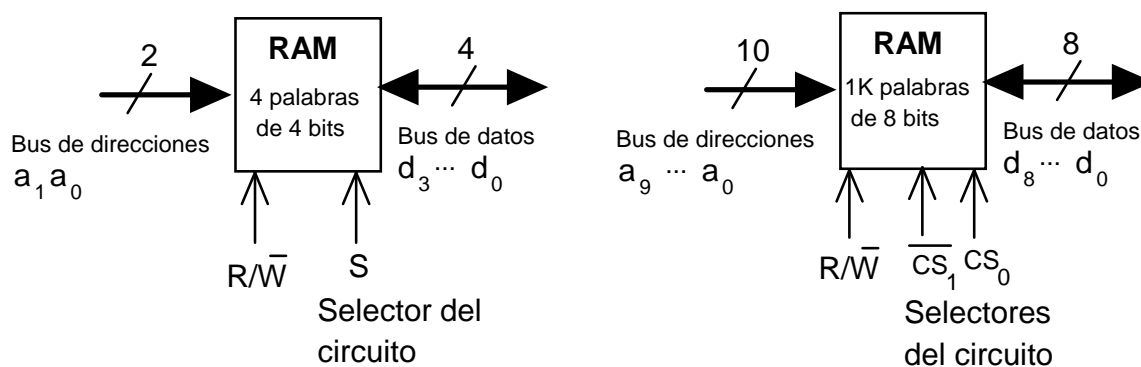


Figura 18.7. Bloques de memorias **RAM**

Existe otra forma de hacer la decodificación de la dirección para seleccionar las celdas binarias de la palabra direccionada, esta es dividiendo el decodificador de $n \times 2^n$ en dos decodificadores de la mitad de tamaño, los $n/2$ bits más significativos de la dirección serian la entrada a un decodificador y los restantes $n/2$ serian la entrada al otro decodificador. Las salidas de los dos decodificadores se colocan en una forma matricial y con el uso de una bloque lógico AND en cada uno de los puntos de la matriz se formara un direccionamiento coincidente para cada una de las 2^n palabras. La figura 18.8 muestra esta forma de direccionamiento para una memoria de 64 palabras de 8 bits. La palabra direccionada es la que se encuentra en la posición 27 (0110112).

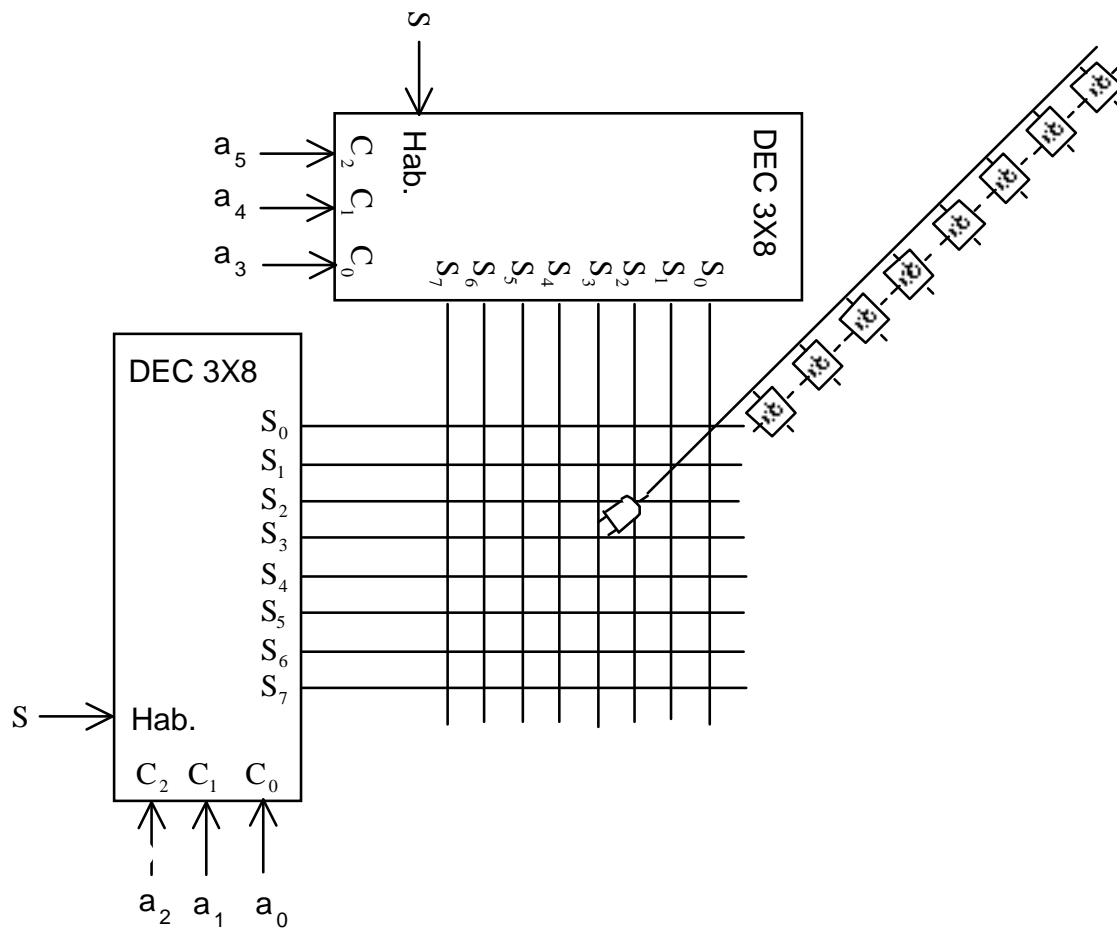


Figura 18.8. RAM de 64 palabras de 8 bits con direccionamiento coincidente.

La ventaja que se tiene con este esquema de direccionamiento es que reduce la complejidad de la lógica de direccionamiento. Por ejemplo en una memoria de 1 Mega palabras si se usa un solo decodificador se requieren de $2^{20} = 1,048,576$ compuertas AND de 20 entradas (no se toma en cuenta la entrada de habilitador) para poder seleccionar cada una de las palabras. Si se usa este esquema se requieren $2(2^{10}) = 2,048$ compuertas AND de 10 entradas más 1,048,576 compuertas AND de dos entradas para hacer el direccionamiento coincidente. Este tipo de direccionamiento normalmente se usa en memorias con palabra de un sólo bit.

Para hacer una operación correcta en una memoria RAM o ROM se requiere de un dispositivo externo como un CPU, para que provea la secuencia de señales en el tiempo preciso para que se pueda hacer la operación. En la figura 18.9 muestra el diagramas de tiempos para el ciclo de lectura y la figura 18.10 para el ciclo de escritura de una RAM Hipotética.

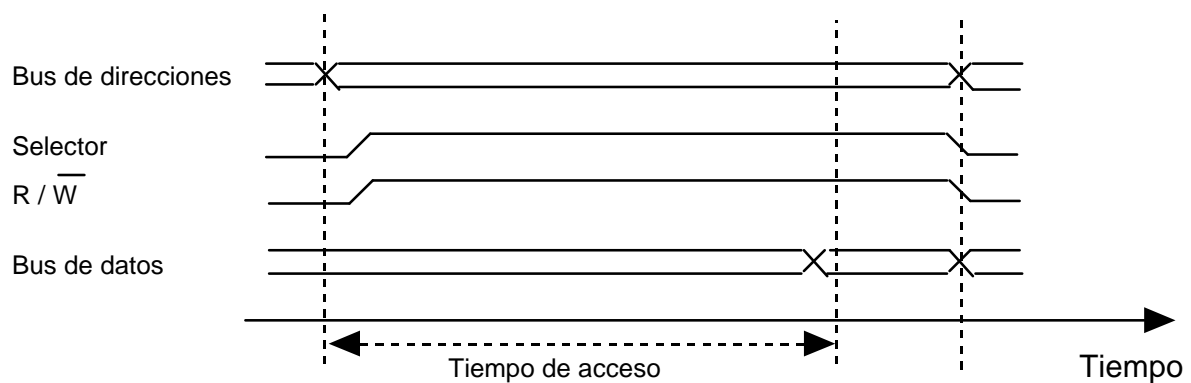


Figura 18.9. Diagrama de tiempos para el ciclo de lectura.

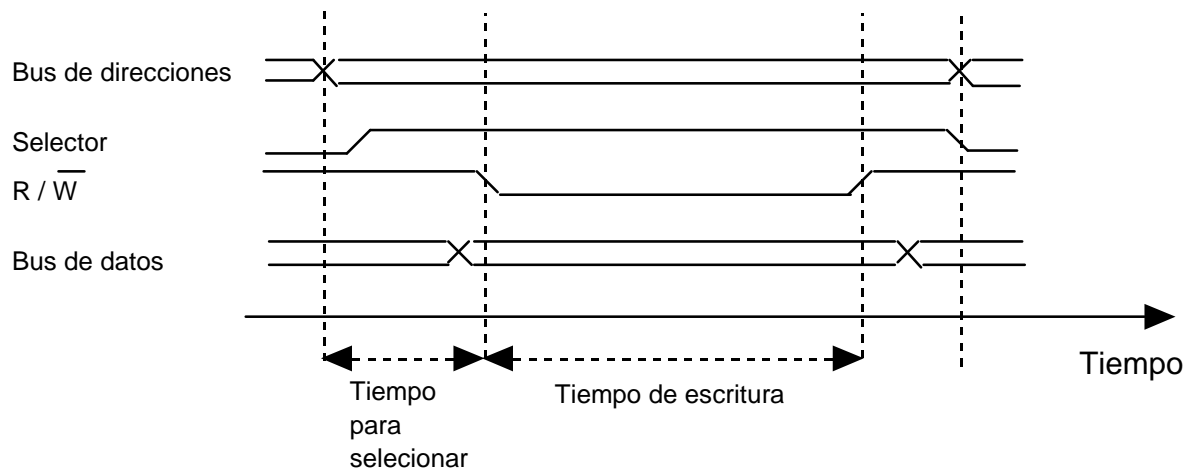


Figura 18.10. Diagrama de tiempos para el ciclo de escritura.

Para poder leer un dato de la memoria primero se pone en el bus de direcciones una dirección válida, posteriormente se activa la línea de selección y se indica una lectura, transcurrido el tiempo de acceso, en el bus de datos tienen disponibles los datos.

El proceso de escritura en memoria es un poco más complicado, ya que se pone la dirección en el bus de direcciones y se activa la línea de selección, se debe de esperar un tiempo para que la palabra se seleccione y así poder indicar la escritura. La señal de escritura debe mantenerse por el tiempo necesario para que internamente se realice. Nótese en la figura 18.10 que los datos que se desean escribir, deben de estar estables un tiempo antes que se realice la escritura, durante todo el tiempo de la escritura y un tiempo después.

En la bibliografía de cada memoria se tienen disponibles estos diagramas con la especificación de todos los tiempos que se deben de cumplir para cada uno de las operaciones.

Dado que el **CPU** envía todas estas señales de control a la memoria, para realizar la sincronización entre estos dos, los tiempos de cada uno de los ciclos se deben de ajustarse a un

número fijo de ciclos del **CPU**. Si la velocidad del CPU es mayor que la de la memoria, éste tendrá que perder una serie de ciclos de reloj en cada acceso a memoria para poderse sincroniza.

18.3 Sistema de memoria

Con frecuencia se usan varios circuitos de memoria para formar memorias más grandes, puede ser que se quiera una memoria con más palabras o una memoria que contenga con un tamaño de palabra mayor. Suponga que se tienen varios circuitos de memoria de 1K palabras de 8 bits como el mostrado en la figura 18.7.

La figura 18.11 muestra como se pueden interconectar dos memorias de éstas para formar una memoria de 2K palabras de 8 bits.

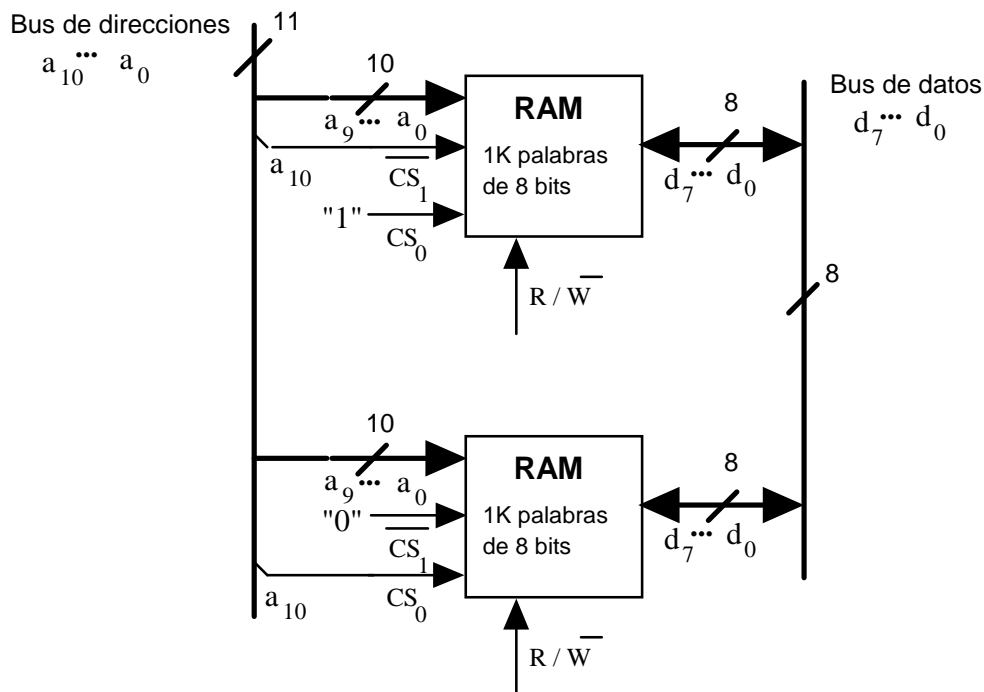


Figura 18.11. RAM de 2K palabras de 8 bits.

La línea de dirección más significativa a_{10} determina cual de los dos circuitos se selecciona, si esta línea vale cero selecciona el circuito superior, el segundo K de memoria se encuentra en el circuito inferior ya que éste se selecciona cuando a_{10} vale uno. De las dos líneas de selección solo se necesita una de ellas, la otra se fija a uno o a cero según el caso. Recuerda que el bus de datos es bidireccional y de tres estados por lo cual se pueden juntar las líneas correspondientes en cada uno de circuitos sin producir un corto, ya que solo uno de éstos puede seleccionar a la vez, lo que hace que el otro bus se encuentre en su tercer estado.

Diagrama de un sistema de memoria con dos RAM de 1K palabras de 8 bits cada una, conectadas a un bus de direcciones de 10 bits y un bus de datos de 16 bits.

Bus de direcciones (10 bits): $a_9 \dots a_0$

RAM 1 (Izquierda): 1K palabras de 8 bits. Se conecta al bus de direcciones a través de un multiplexor de 10 bits. Las señales de selección de chip (CS_1 y CS_0) se activan con "0" y "1" respectivamente. El control de lectura/escritura (R/\bar{W}) se conecta directamente al bus de direcciones.

RAM 2 (Derecha): 1K palabras de 8 bits. Se conecta al bus de direcciones a través de un multiplexor de 10 bits. Las señales de selección de chip (CS_1 y CS_0) se activan con "0" y "1" respectivamente. El control de lectura/escritura (R/\bar{W}) se conecta directamente al bus de direcciones.

Bus de datos (16 bits): $d_{15} \dots d_0$

El diagrama muestra la conexión de los buses de direcciones y datos a las RAM, así como las señales de selección de chip y control de lectura/escritura.

En este circuito los dos circuitos siempre se seleccionan, y cada uno de los circuitos proporcionan 8 líneas para formar el bus de datos de 16 líneas.

Para mostrar un ejemplo particular, considérese el diseño del siguiente sistema de memoria. Se quiere diseñar un sistema de memoria de 64K bytes, donde del K0 al K7 y del K24 al K31 son de memoria ROM y el resto es de memoria RAM. Se tienen dos ROM de 8k bytes (ROM-1 y ROM-2), una memoria RAM de 16k bytes (RAM-1) y ocho memorias RAM de 32k nibble, un nibble son 4 bits (RAM-2 y RAM-3). Todos los circuitos cuentan con dos líneas de selección y sus buses de datos son de tres estados.

El bus de direcciones para cada una de las memoria depende del número de palabras que tenga, la memoria de 8k palabras su bus de direcciones es de 13 líneas, para la de 16k palabras es de 14 líneas y para la de 32k palabras es de 15 líneas. Para construir el sistema de memoria a cada circuito se le conectan las líneas que requiera menos significativas del bus de direcciones del

sistema y se usan las líneas más significativas para construir la lógica de selección de cada uno de los circuitos.

Para determinar la lógica de decodificación a usar en la selección de cada uno de los circuitos de memoria, se construye el mapa de dirección de memoria como se muestra a continuación:

K de memoria	direcciones hexadecimal	bus de direcciones a15 a14 a13	Circuito de memoria
0 al 7	0000 - 1FFF	0 0 0	ROM - 1
8 al 15	2000 - 3FFF	0 0 1	RAM - 1
16 al 23	4000 - 5FFF	0 1 0	RAM - 1
24 al 31	6000 - 7FFF	0 1 1	ROM - 2
32 al 39	8000 - 9FFF	1 0 0	RAM - 2 y 3
40 al 47	A000 - BFFF	1 0 1	RAM - 2 y 3
48 al 55	C000 - DFFF	1 1 0	RAM - 2 y 3
56 al 63	E000 - FFFF	1 1 1	RAM - 2 y 3

Del mapa de memoria se puede determinar la lógica de decodificación para cada uno de los circuitos. La siguiente tabla indica la función lógica a usar en cada uno de los selectores de los circuitos de memoria.

Circuito de memoria	selector CS_0	selector $\overline{CS_1}$
ROM - 1	$\overline{a_{14}} + \overline{a_{13}}$	a_{15}
ROM - 2	$a_{14} \cdot a_{13}$	a_{15}
RAM - 1	$a_{14} \oplus a_{13}$	a_{15}
RAM - 2 y 3	a_{15}	0

La figura 18.13 muestra el diseño de este sistema de memoria. Como puede comprobarse, dependiendo de los bits más significativos de la dirección solo uno de los circuitos se selecciona y solo este puede poner o tomar la información que se encuentra en el bus de datos.

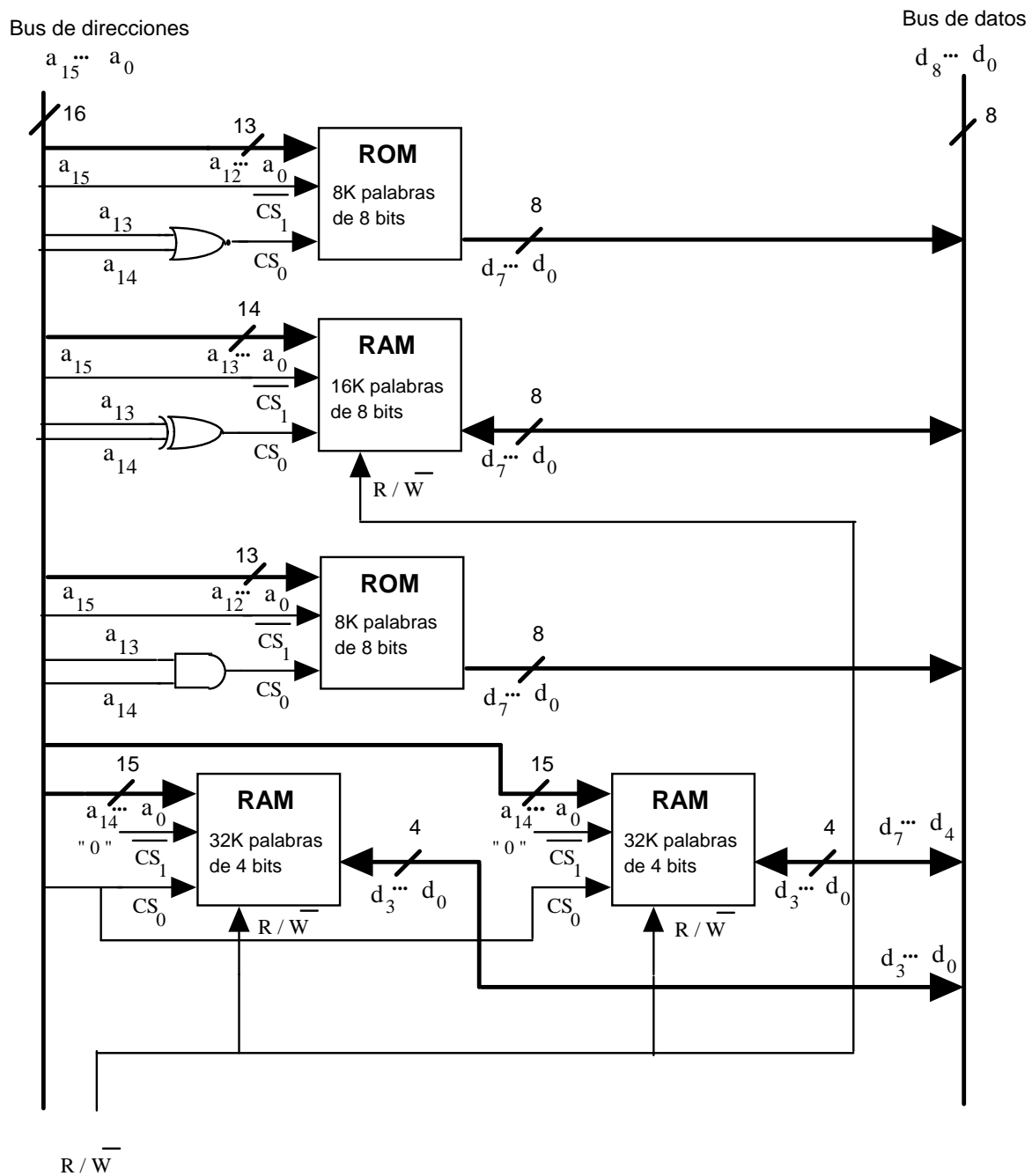


Figura 18.16. Sistema de memoria de 64K palabras de 8 bits.

Cada circuito tiene sus propias características y tiempos de acceso, en forma general podemos indicar las siguientes características:

Carcterística	ROM	RAM estático	RAM dinámico
---------------	-----	-----------------	-----------------

Capacidad de almacenamiento por circuito	Muy alta	Alta	Muy alta
Tiempo de acceso	El más bajo	Muy bajo	Bajo
Volátil	No	Sí	Sí

Las características particulares de cada circuito deben ser consideradas al momento de hacer el diseño del sistema de memoria, y también por el dispositivo que genera las señales de control a ésta.

18.4 Resumen.

La memoria de una computadora está formada por una combinación de memorias RAM y ROM. Este último tipo de memorias se había presentado en capítulos anteriores, no así la organización de la memoria RAM.

Se presentó la construcción de una unidad simple de memoria RAM a partir de celdas binarias y circuitos lógicos que permiten realizar la lectura y escritura de una dirección de memoria en particular. La mayoría de las unidades de memoria simples contienen palabras de 8 bits.

La memoria RAM de una computadora está formada por varias unidades de memoria simples, organizadas de cierta manera para formar el sistema de memoria de la computadora. Generalmente, a palabra de memoria de una computadora contiene un múltiplo de entero de un byte (8 bits). En este capítulo se presenta también como está organizado el sistema de memoria de una computadora.

18.5 PROBLEMAS

1. Construya una celda binaria usando solo bloques lógicos NAND.
2. Construya una memoria RAM de 4 palabras de 8 bits con bus de datos bidireccional, de tres estados, utilizando celdas binarias.
3. Construya una memoria RAM de 8K palabras de 8 bits, utilizando memorias RAM de 4K palabras de 8 bits.
4. Construya una memoria RAM de 4K palabras de 8 bits, usando memorias RAM de 4K palabras de 1 bits.
5. Construya una memoria RAM de 16K palabras de 8 bits, a partir de memorias RAM de 8K palabras de 4 bits.
6. Diseñe un sistema de memoria de 64K bytes, donde los primeros 48K bytes son de RAM y el resto es de ROM. Se cuenta con dos memorias RAMs de 16K bytes, dos memorias RAMs de 16K palabras de 4 bits y una memoria ROM de 16K bytes.

7. Diseñe un sistema de memoria de 8K bytes, donde los primeros 6K bytes son de RAM y el resto es de ROM. Se cuenta con una memoria RAM de 4K bytes, una memoria RAM de 2K bytes y una memoria ROMs de 2K palabras de 4 bits.
8. Diseñe un sistema de memoria de 1M bytes, donde los primeros 512K bytes son de RAM y el resto es de ROM. Se dispone de memorias RAMs de 512K palabras de 1 bit, y memorias ROMs de 256K bytes.
9. Diseñe un sistema de memoria de 64K bytes, donde los primeros 8K bytes y los últimos 8K son de ROM y el resto es de RAM. Utilice memorias RAMs de 16K de 2 bits y memorias ROMs de 4K palabras de 4 bits.
10. Construya una memoria ROM de 8 palabras de 4 bits con bus de datos de tres estados. La información que debe de contener el ROM es la siguiente:

Dirección	Contenido
0	1100
1	1111
2	1010
3	0000
4	1100
5	0001
6	1001
7	0110

CAPITULO 19

DISEÑO DE CIRCUITOS SECUENCIALES.

En el capítulo 17 se presentó una breve introducción a los circuitos secuenciales, destacando también la necesidad de contar con ellos. La arquitectura de una computadora simple, como la presentada en el capítulo 2, incluía una unidad aritmética y lógica (ALU) así como una unidad de control (CU).

El capítulo 16 presentó el diseño de una unidad aritmética y lógica básica, para lo cual se requirieron todos los conceptos tratados en los capítulos 7 a 15. Esta unidad es puramente combinatoria. Por otra parte, la unidad de control debe ser capaz de realizar una serie de operaciones en una determinada secuencia para poder llevar a cabo su función. Esta unidad es la responsable de controlar los ciclos de búsqueda (FETCH) y ejecución (EXECUTE) de cada una de las instrucciones de que consta un programa, así como controlar la operación de la unidad de entrada y salida. La unidad de control no puede ser construída como un circuito lógico puramente combinatorio, realmente tiene que ser diseñada como un circuito lógico secuencial.

En este capítulo primeramente se presentará el análisis de varios circuitos lógicos secuenciales simples y posteriormente se presentarán los principios básicos para el diseño de circuitos secuenciales. Con estos conceptos, más los presentados en los capítulos 20, 21 y 22, se realizará al diseño de una unidad de control básica en el capítulo 23.

19.1. Clases de circuitos secuenciales.

En la figura 19.1 se muestra de nuevo la arquitectura básica de un circuito secuencial, la cual fue presentada en el capítulo 17. El estado interno de un circuito secuencial está formado por la combinación de los estados de un conjunto de celdas biestables. Dado que cada celda puede estar solamente en uno de dos estados posibles, si se tienen N celdas para formar el estado de un circuito secuencial, se podrán tener, como máximo, 2^N estados diferentes.

Los circuitos secuenciales pueden clasificarse en circuitos sincrónicos y circuitos asincrónicos. La forma de analizar y diseñar estas dos clases de circuitos secuenciales siguen enfoques diferentes, aunque comparten muchos principios básicos. Por lo general, el tipo de celdas biestables que se utilizan para mantener el estado de los circuitos secuenciales sincrónicos son FLIP FLOPS, es decir, celdas biestables que tienen entrada de reloj. Para circuitos asincrónicos se pueden utilizar LATCHES, que no tienen entrada de reloj, FLIP FLOPS, o una combinación de ambos tipos.

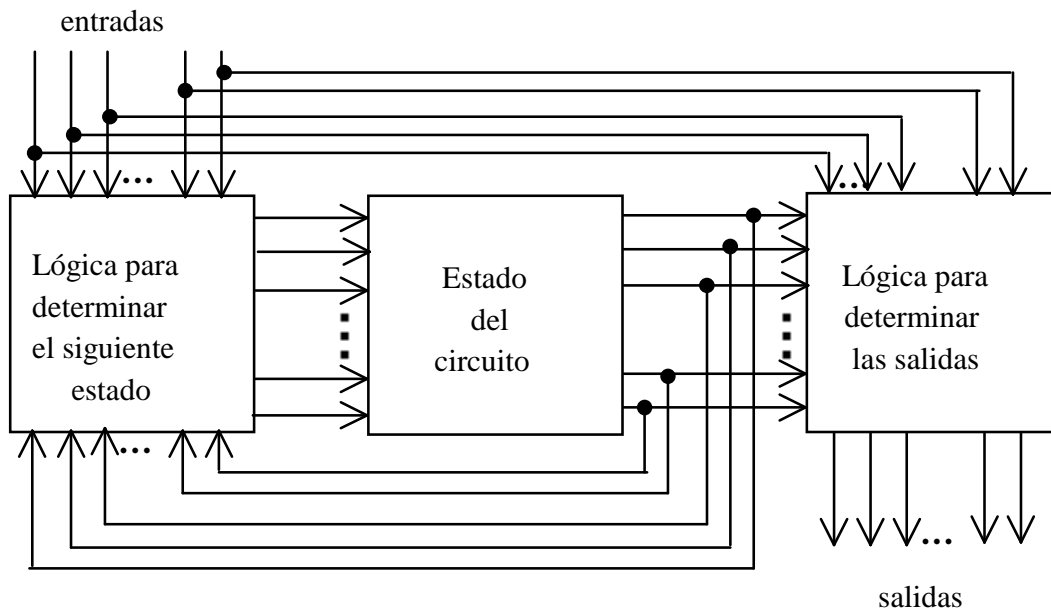


Figura 19.1 Arquitectura de un circuito lógico secuencial

La característica que identifica a los circuitos secuenciales síncronos es que todas las celdas biestables que forman el estado interno cambian al mismo tiempo, sincronizadas por una misma señal. En cambio, en los circuitos asíncronos, cada una de las celdas que forman el estado pueden cambiar en instantes diferentes de tiempo.

La señal que activa el cambio de estado en circuitos síncronos puede ser una señal maestra de reloj, la cual es una señal periódica. En este caso, el estado interno del circuito puede cambiar solamente en instantes predefinidos de tiempo, determinados por la señal maestra de reloj. Sin embargo, también puede utilizarse una señal externa para sincronizar los cambios de estado, en cuyo caso, se tendrá una misma señal aperiódica en las entradas de reloj de todos los FLIP FLOPS, los cuales pueden cambiar en cualquier momento, pero todos tienen que cambiar simultáneamente.

Los circuitos secuenciales asíncronos pueden contar con una señal de reloj o prescindir de ella. En los que carecen de señal de reloj, los cambios de estado se activan por eventos externos, los cuales llegan al circuito a través de las entradas. En este caso se pueden utilizar simplemente LATCHES para mantener el estado del circuito, aunque también podrían ser utilizados FLIP FLOPS, o una combinación de los dos tipos de celdas. En los circuitos asíncronos que tienen una señal de reloj, ésta no controla el cambio de estado de todas las celdas que forman el estado, solamente controla los cambios de estado de alguna o algunas de ellas, utilizándose combinaciones de entradas externas y estados para activar los cambios de estado del resto de las celdas.

Algunos autores definen como circuitos síncronos solamente aquellos en los cuales se tiene una señal maestra de reloj que controla el cambio de estado de todas las celdas biestables, considerando como circuitos asíncronos todos los demás. En este libro se utilizará la

definición dada anteriormente para circuitos sincrónicos, que implica que todas las celdas biestables cambien al mismo tiempo, independientemente de que la señal de sincronización sea una señal maestra de reloj o una entrada externa.

Una de las principales herramientas para el análisis y diseño de circuitos secuenciales, tanto sincrónicos como asincrónicos, son los diagramas de estado. Inicialmente en estos diagramas se mostrarán solamente todos los estados en que puede encontrarse el circuito secuencial y todas las transiciones que puedan existir de un estado a otro. Los estados se representan por medio de un círculo que contienen internamente el valor del estado y, para indicar las transición de un estado a otro, se usa un arco dirigido. La figura 19.2 muestra la forma de definir un diagrama de estados, $Q_n Q_{n-1} \dots Q_0$ representa el estado actual del circuito y $Q_n^+ Q_{n-1}^+ \dots Q_0^+$ representa el siguiente estado. A medida que se vaya necesitando mayor información, se irá añadiendo ésta a los diagramas. Otra importante herramienta es el diagrama de tiempos. Esta última ya fue utilizada en el capítulo 17 y se continuará usando en este capítulo.

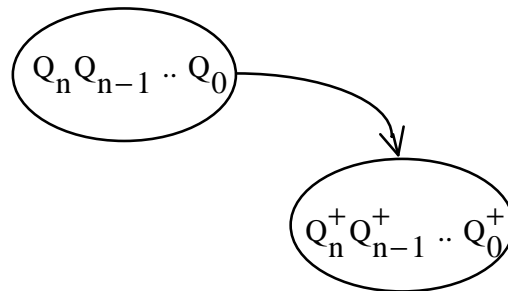


Figura 19.2. Definición de diagrama de estados

19.2. Ejemplos simples de circuitos secuenciales.

Como primer ejemplo de un circuito secuencial se presentará un contador binario ascendente de 0 a 7. Este circuito no tendrá ninguna entrada y la salida del circuito será directamente su estado. Se analizará primeramente el circuito sincrónico de este contador y posteriormente su forma asincrónica.

El circuito sincrónico para este contador se muestra en la figura 19.3. Para poder tener ocho estados diferentes es necesario utilizar tres celdas binarias ($2^3=8$), para lo cual se escogieron tres FLIP FLOPS T, activados por transición positiva en la señal de reloj. Las salidas de estos FLIP FLOPS se definieron como Q_2 , Q_1 y Q_0 . En este ejemplo, cada una de las salidas del circuito corresponde a una de las salidas de cada FLIP FLOP. Se supone que se tiene algún dispositivo que genera la señal de reloj.

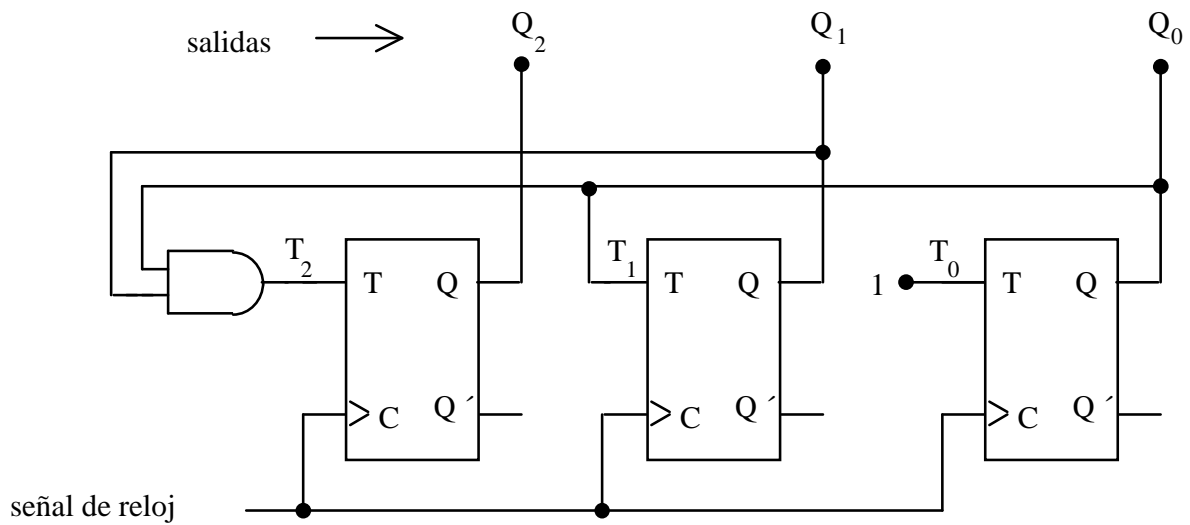


Figura 19.3. Circuito para el contador binario sincrónico de 0 a 7.

El diagrama de estados de este circuito se muestra en la figura 19.4. Cada estado está representado por un círculo que contiene internamente los valores de Q_2 , Q_1 y Q_0 que lo identifican. Los estados son 000, 001, 010, 011, 100, 101, 110 y 111, correspondientes a los números decimales del 0 al 7 inclusive. La secuencia de estados de este circuito es 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, ..., y así sucesivamente.

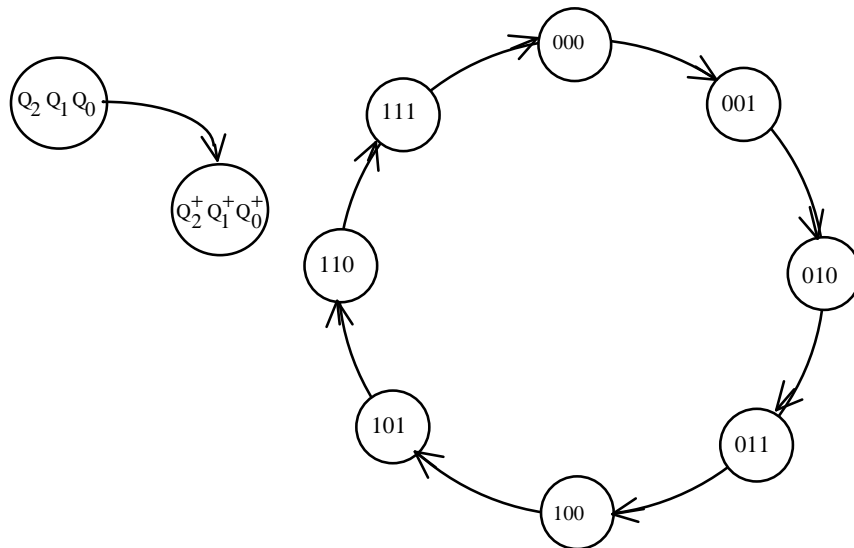


Figura 19.4. Diagrama de estados del contador sincrónico de 0 a 7

Para analizar la operación de este circuito, supóngase que se encuentra en el estado 000. Las entradas de control de los FLIP FLOPS son:

$$T_0 = 1$$

$$T_1 = Q_0 = 0$$

$$T_2 = Q_1 Q_0 = 0$$

Cuando el pulso de reloj efectúe la transición de 0 a 1, el único FLIP FLOP que cambiará será el identificado como Q_0 , quedando el circuito en el estado 001, lo cual coincide con la transición en el diagrama de estados.

Ahora, las entradas de control de los FLIP FLOPS serán:

$$T_0 = 1$$

$$T_1 = Q_0 = 1$$

$$T_2 = Q_1 Q_0 = 0$$

Cuando ocurra la transición positiva en la señal de reloj, cambiarán los dos FLIP FLOPS que tienen la entrada de control en 1 y el circuito quedará en el estado 010.

Analizando lo que ocurre en este estado, se tienen las siguientes entradas de control a los FLIP FLOPS:

$$T_0 = 1$$

$$T_1 = Q_0 = 0$$

$$T_2 = Q_1 Q_0 = 0$$

Al presentarse la transición positiva de la señal de reloj, cambiarán solamente los FLIP FLOP que tengan un 1 en su entrada de control, quedando el circuito en el estado 011.

El lector podrá comprobar que para los demás casos, el circuito cambiará de estado de acuerdo a la secuencia mostrada en el diagrama de estados.

El diagrama de tiempos para este circuito se muestra en la figura 19.5

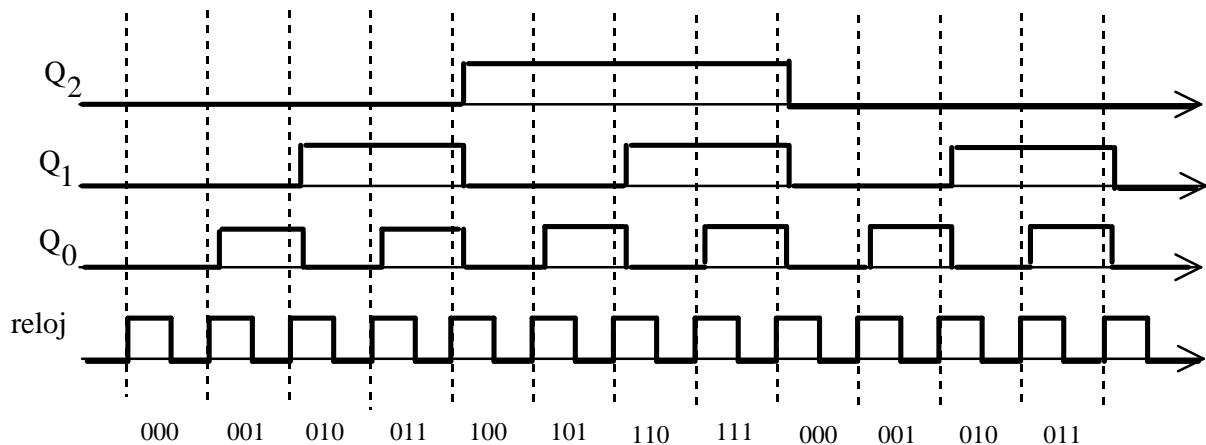


Figura 19.5. Diagrama de tiempos del contador sincrónico de 0 a 7.

Nótese que las salidas de los tres FLIP FLOPS cambian al mismo tiempo, cuando cambian. El desfase que se muestra entre la señal de reloj y los cambios en las salidas de los FLIP FLOPS corresponde al tiempo que tarda en cambiar y estabilizarse la salida de cada celda. Revísese el capítulo 17 si es necesario.

El circuito mostrado en la figura 19.6 es una versión asincrónica del contador anterior. Obsérvese que la entrada de reloj sólo controla el cambio de estado del FLIP FLOP Q_0 . Los otros FLIP FLOPS se activan en ciertos estados solamente, mediante transiciones negativas en sus entradas de reloj.

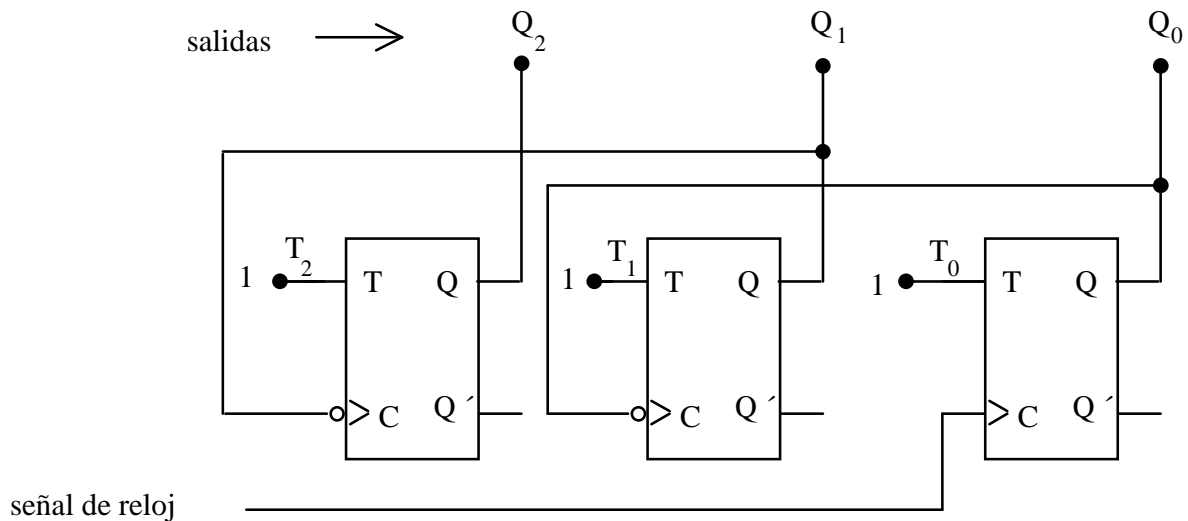


Figura 19.6. Circuito para el contador binario asincrónico de 0 a 7.

El diagrama de tiempos para este circuito se muestra en la figura 19.7. El FLIP FLOP Q_0 siempre cambia de estado cuando se presenta la transición positiva del pulso de reloj. EL FLIP FLOP Q_1 cambia de estado cuando Q_0 realiza una transición negativa (de 1 a 0). El tercer FLIP FLOP cambia su estado cuando Q_1 realiza una transición negativa.

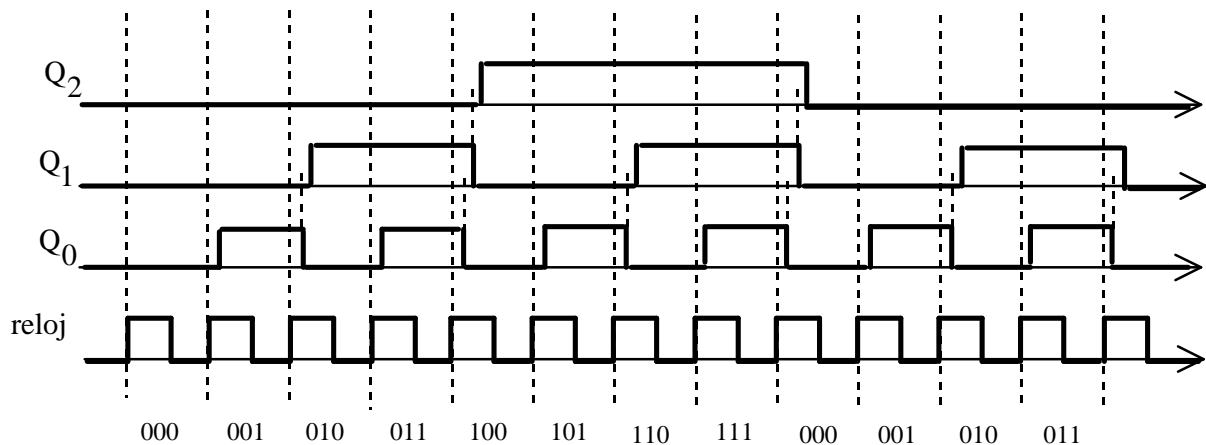


Figura 19.7. Diagrama de tiempos del contador asincrónico de 0 a 7.

Si se pone atención al diagrama de tiempos se podrá ver que los tres FLIP FLOPS cambian en diferentes instantes de tiempo. Por ejemplo, al cambiar del estado 011 al 100, el FLIP FLOP Q_0 cambia un cierto tiempo después de que se presenta la transición positiva del pulso de reloj. El FLIP FLOP Q_1 cambia después de que cambia Q_0 de 1 a 0. El FLIP FLOP Q_2 cambia un tiempo después de que cambia Q_1 . Por lo tanto al cambiar del estado 011 al 100 se pasa por una serie de estados falsos como son los estados 010 y 000. En el diagrama de estados de la figura 19.8 se muestran todos los estados falsos que tiene este circuito, que aunque tiene una señal de reloj, es un circuito asincrónico.

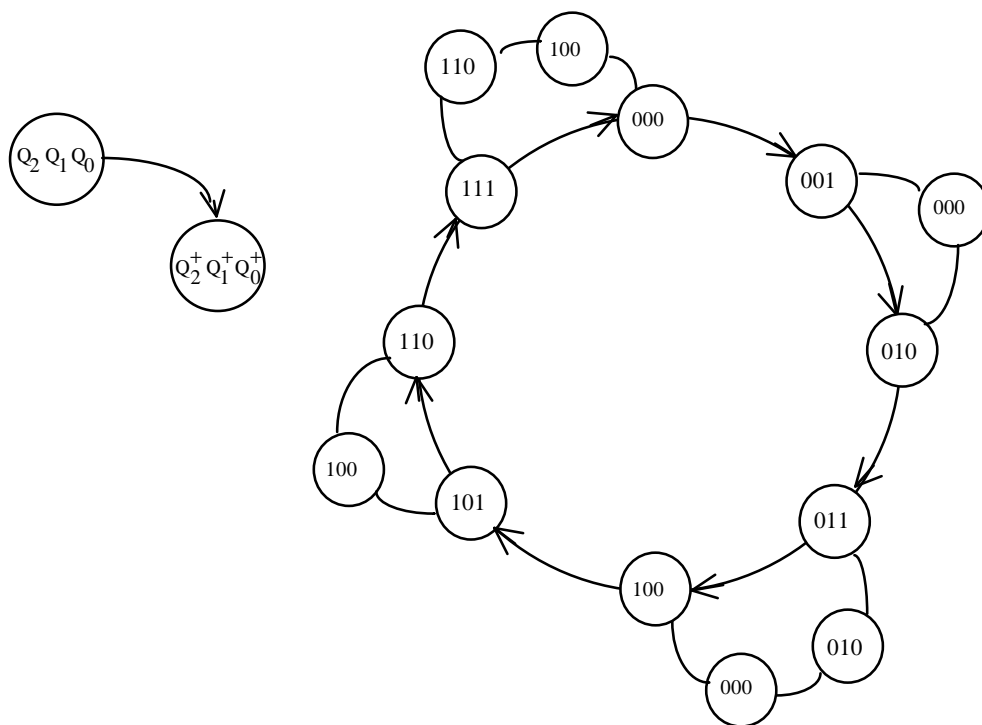


Figura 19.8. Diagrama de estados del contador asincrónico de 0 a 7

Los circuitos sincrónicos no tienen estados falsos y funcionan más rápido que los circuitos asincrónicos, pero estos últimos normalmente requieren de menos elementos en su construcción, como en este caso, o no se requiere de la compuerta AND, lo cual lo hace más económico.

Se analizará ahora otro circuito secuencial sincrónico con una entrada y dos salidas. Este nuevo circuito es un contador binario de cero a tres, que puede contar hacia arriba o hacia abajo, dependiendo del valor de su entrada. Las salidas del circuito serán los dos bits que se necesitan para representar un número binario entre cero y tres inclusive.

Es fácil adivinar que se necesitan dos FLIP FLOPS para mantener el estado del circuito, ya que con dos bits se pueden hacer cuatro combinaciones diferentes que corresponderán a cada uno de los cuatro estados.

El circuito lógico de este contador binario se muestra en la figura 19.9. Se utilizaron dos FLIP FLOPS T del tipo maestro-esclavo para construirlo. El estado del circuito está definido por los estados de los dos FLIP FLIPS T, denominadas Q_1 y Q_0 . La entrada al circuito se muestra como X y las salidas del circuito son las señales Y_1 y Y_0 . Si la entrada X vale uno, el circuito contará en forma ascendente (0, 1, 2, 3, 0, 1, 2,...). Si la entrada X vale cero, contará en forma descendente (3, 2, 1, 0, 3, 2, 1, ...). Las salidas del circuito tomarán los valores 00, 01, 10 y 11 en secuencia, hacia arriba o hacia abajo, dependiendo de la entrada X

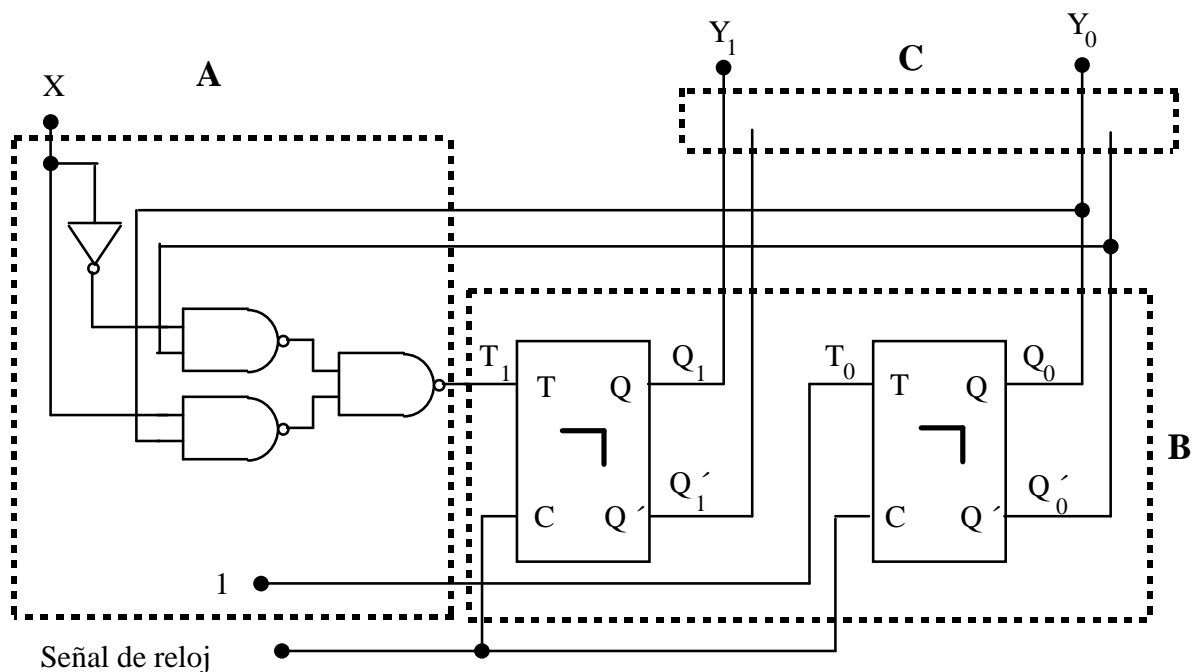


Figura 19.9. Contador binario de 0 a 3, hacia arriba y hacia abajo

En esta figura se identifican tres partes del circuito enmarcadas por los rectángulos A, B y C, que corresponden a cada una de las partes identificadas en la figura 19.1.

Los dos FLIP FLOPS que están dentro del rectángulo B forman el estado interno del circuito. Las variables Q_1 y Q_0 pueden tomar cuatro combinaciones (00, 01, 10 y 11) que corresponden a cada uno de los cuatro posibles estados del sistema.

La parte del circuito comprendida dentro del rectángulo A corresponde al bloque “Lógica para determinar el siguiente estado” de la figura 19.1. Las entradas a esta parte del circuito están formadas comúnmente por el estado del circuito y las entradas externas. Para este caso, las entradas son las dos salidas del FLIP FLOP identificado como Q_0 y Q_0' y la entrada externa X . En general, las entradas a este bloque serían las variables X , Q_0 , Q_0' , Q_1 y Q_1' , solamente que

para este circuito en particular no se necesitan Q_1 y Q'_1 . Las salidas de este bloque se conocen comúnmente como excitación y son las entradas de control de los FLIP FLOPS que forman el estado del circuito secuencial. Para el caso presente, las salidas de este bloque son T_1 y T_0 , que controlan el cambio de estado del circuito secuencial cuando se presenta el pulso de la señal de reloj. La lógica para determinar el siguiente estado está definida por las ecuaciones:

$$\begin{aligned} T_0 &= 1 \\ T_1 &= Q_0 X + Q_0' X' \end{aligned}$$

El rectángulo identificado como C corresponde al bloque “Lógica para determinar las salidas” de la figura 19.2. Las entradas a este bloque son las salidas de los FLIP FLOPS que forman el estado del circuito y las entradas externas. Sus salidas son precisamente las salidas del circuito secuencial. En el caso presente, la lógica de salida es sumamente simple y está definida por las ecuaciones siguientes:

$$\begin{aligned} Y_0 &= Q_0 \\ Y_1 &= Q_1 \end{aligned}$$

Se procederá ahora a analizar este circuito secuencial utilizando el diagrama de estados mostrado en la figura 19.10. Al igual que en el caso anterior, cada estado del circuito se muestra como un círculo identificado por el valor de las variables Q_1 y Q_0 que determinen el estado del circuito. Las salidas del circuito corresponden al estado del mismo.

Los arcos dirigidos representan las transiciones que existen de un estado a otro. Nótese que hay transiciones del estado 00 a los estados 01 y 11, pero no existe una transición del estado 00 al 10. Algo similar ocurre con los otros tres estados.

Junto a estos arcos dirigidos ahora se muestra el valor de la entrada externa X que, junto con el estado presente, determina a cual estado se realizará la siguiente transición. En el primer ejemplo no se tenía esta información en el diagrama de estados porque no era necesaria. El contador anterior siempre seguía la misma secuencia, pero éste puede contar en forma ascendente o descendente, dependiendo del valor de su entrada externa.

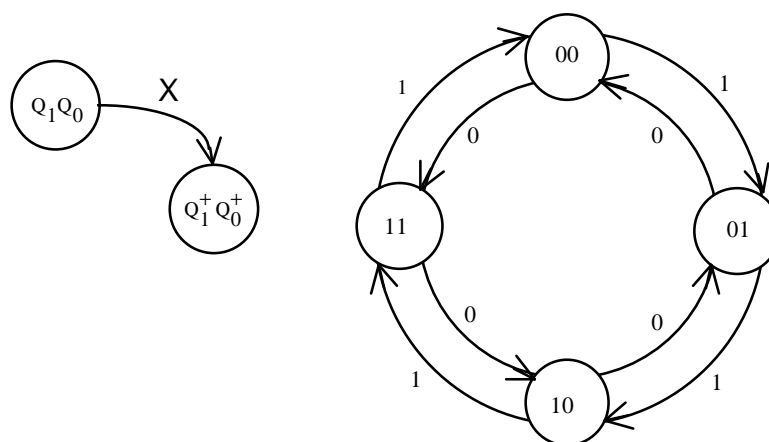


Figura 19.10. Diagrama de estados del contador binario de 0 a 3, hacia arriba y hacia abajo.

De acuerdo a las convenciones establecidas, se puede deducir que el circuito hará una transición del estado 00 al estado 01 cuando se presente el pulso en la señal de reloj y en la entrada externa X se tenga el valor de 1. Si se encontrara en el estado 00 y la señal externa X tuviera el valor de 0, la transición se haría al estado 11.

Según lo indicado en el diagrama de estados, si el circuito se encuentra en el estado 10 y la entrada X tiene el valor de 1, el siguiente estado será el 11, al cual cambiará cuando se presente el pulso de reloj. Si se encontrara en el mismo estado 10 y la entrada X tuviera el valor de 0, el siguiente estado sería el 01.

El circuito mostrado en la figura 19.9 se comportará de acuerdo a lo indicado en el diagrama de estados. El lector puede realizar un análisis similar al efectuado en el ejemplo anterior para comprobarlo.

En la figura 19.11 se muestra el diagrama de tiempos para el circuito de la figura 9.9. Se supuso que el estado inicial para este diagrama es el 00 y que la entrada externa X vale 1. Obsérvese que ambos FLIP FLOPS cambian al mismo tiempo, cuando la señal de reloj regresa al valor de cero. Recuérdese que estos FLIP FLOPS son activados con el pulso positivo y su salida cambia cuando la señal del reloj llega a cero.

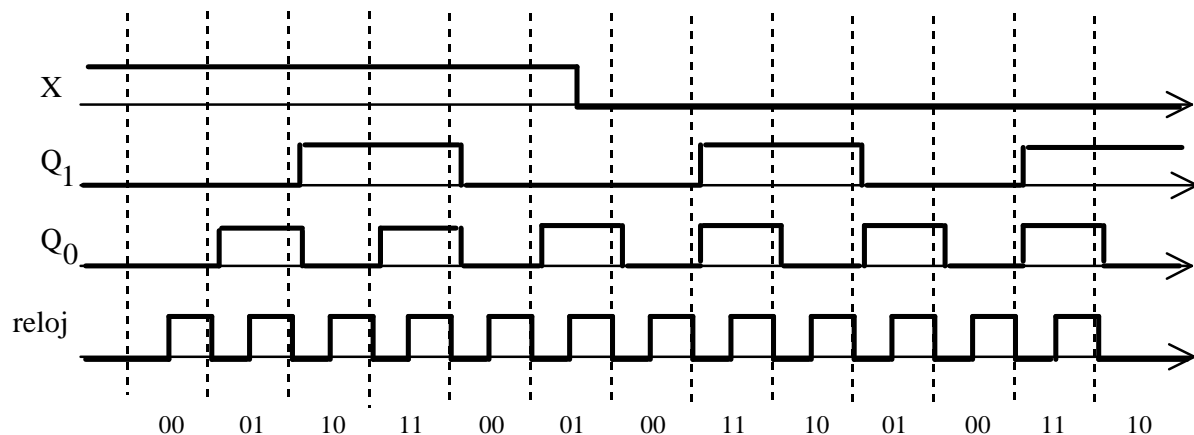


Figura 19.11. Diagrama de tiempos para el contador binario de 0 a 3, hacia arriba y hacia abajo.

En la segunda parte del diagrama de tiempos de la figura 19.11 muestra lo que se obtendría cuando la entrada externa tuviera el valor de 0.

19.3. Modelos de Mealy y de Moore.

Existen dos modelos básicos de circuitos secuenciales, el modelo de Mealy y el de Moore, los cuales han recibido sus nombres en honor de sus creadores G. H. Mealy y E. F. Moore. El modelo de Mealy es el más general ya que en éste, las salidas del circuito secuencial dependen tanto del estado actual como de las entradas externas. La arquitectura mostrada en la figura 19.1 representa este modelo. Estos circuitos son conocidos como máquinas de Mealy.

En el modelo de Moore, las salidas dependen solamente del estado presente, sin importar los valores de las entradas. Los circuitos diseñados de acuerdo a este modelo se conocen como máquinas de Moore. Todos los ejemplos de circuitos secuenciales que se han presentado en este capítulo son máquinas de Moore. En la figura 19.12 se ilustra este modelo.

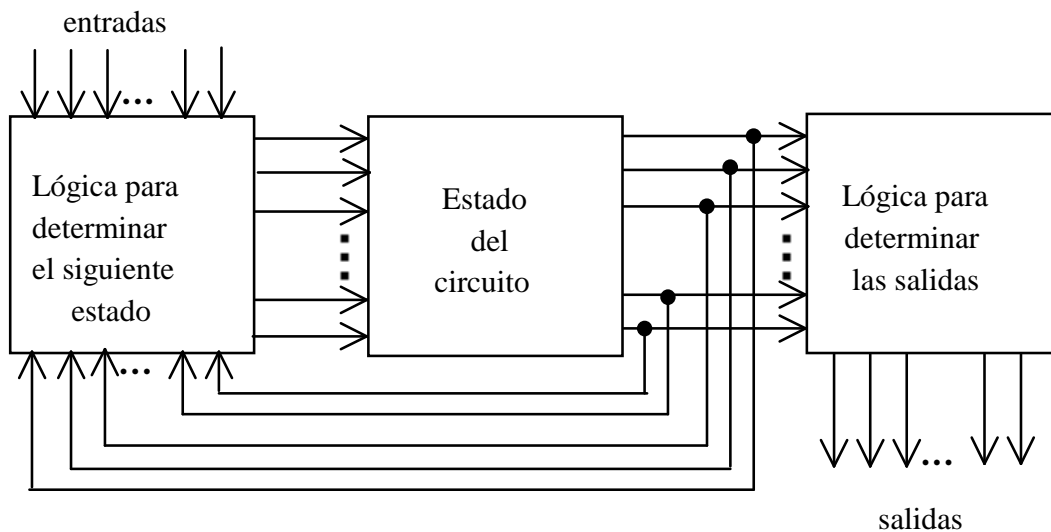


Figura 19.12. Circuito secuencial construido con el modelo de Moore.

Los diagramas de estado de una máquina de Mealy requieren más información que los de una máquina de Moore. El diagrama de estados de un circuito secuencial construido de acuerdo al modelo de Mealy también muestra en los estados la combinación de variables de estado que los identifica. En los arcos dirigidos, que representan las transiciones, es necesario añadir a los valores de las variables de entrada y los valores de las variables de salida para el estado presente, ya que éstos dependen de las entradas. En la figura 19.13 se muestra la definición y un fragmento de un diagrama de estado para una máquina de Mealy que tiene cuatro variables de estado ($Q_3Q_2Q_1Q_0$), dos entradas (E_1E_0) y tres salidas ($S_2S_1S_0$).

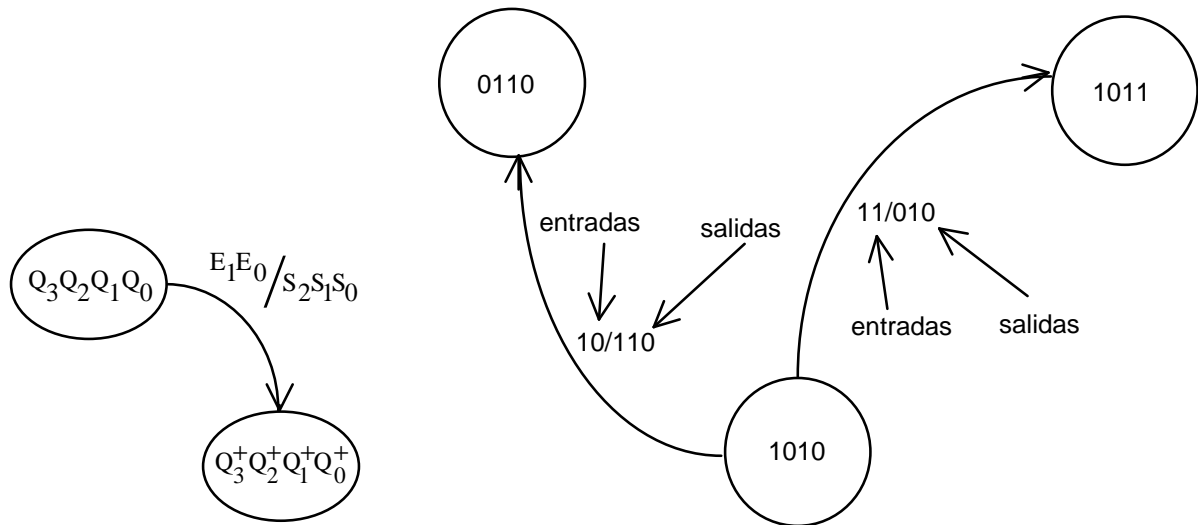


Figura 19.13. Definición y ejemplo de un fragmento de diagrama de estado para una máquina de Mealy

En el fragmento de diagrama de la figura 19.13 se indica que si el circuito se encuentra en el estado 1010 y las entradas son 10, la transición debe realizarse hacia el estado 0110. Adicionalmente se especifica que las salidas deben ser 110 para el caso de estar en este estado y tener las entradas 10.

Si el sistema estuviera en el mismo estado 1010 pero las entradas fueran 11, las salidas que debería dar son 010 y el siguiente estado sería 1011.

En las máquinas de Moore las salidas dependen solamente del estado actual y no de las entradas. En los ejemplos presentados anteriormente se especificó que las salidas de los circuitos estaban dadas directamente por el estado y no se indicaron en los diagramas de estado. En general, las salidas de un circuito de este tipo se indican en los estados tal como se ejemplifica en la figura 19.14.

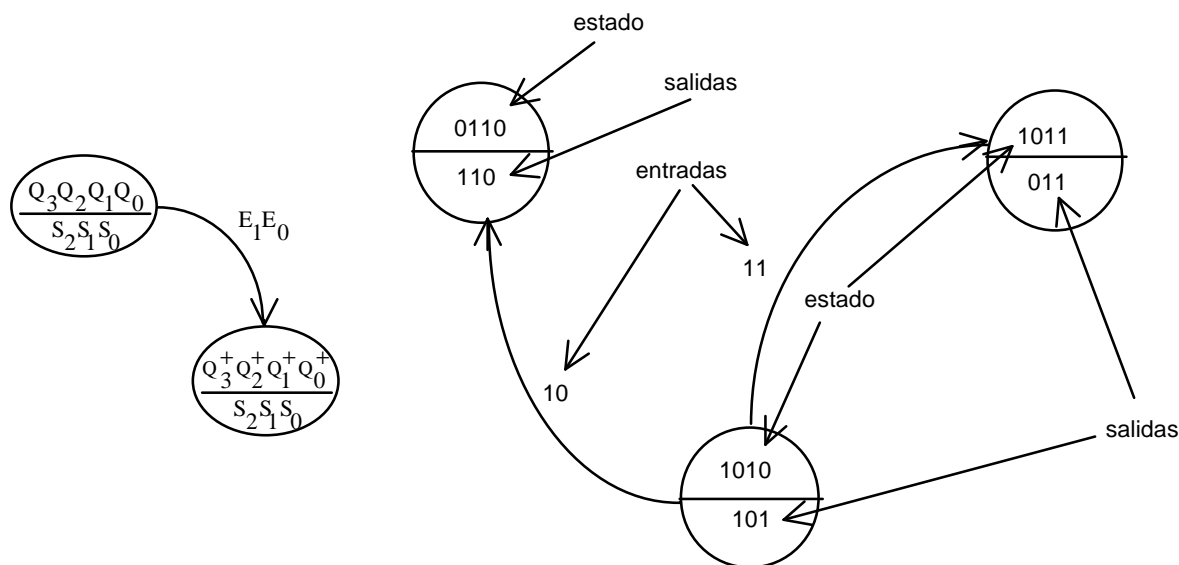


Figura 19.14. Definición y ejemplo de un fragmento de diagrama de estado para una máquina de Moore

En el fragmento de diagrama de la figura 19.14 se indica que si el circuito está en el estado 1010, las salidas que debe dar son 101, independientemente de los valores de las entradas. Si las entradas son 10, el siguiente estado será el 0110. Si las entradas son 11, deberá ir al estado 1011.

Cuando la salida de un circuito secuencial sea el estado mismo, se omitirá la indicación de las salidas en el diagrama de estados.

19.4. Tablas de excitación para FLIP FLOPS.

En los ejemplos anteriores se comentó que las salidas del bloque identificado como “Lógica para determinar el siguiente estado” reciben comúnmente el nombre de excitación. En los circuitos secuenciales, estas señales sirven como entradas de control para los FLIP FLOPS que forman el estado del circuito y determinan el estado futuro del mismo.

Si se utilizan FLIP FLOPS T como en los ejemplos anteriores, las salidas de la lógica para determinar el siguiente estado serán las señales que se alimentan a las entradas de control T de los FLIP FLOPS. Si se utilizaran FLIP FLOPS J-K, estas salidas serían obviamente las señales que se alimentarían a las entradas de control J y K de cada uno de los FLIP FLIPS J-K. Algo equivalente ocurriría si se tuvieran FLIP FLOPS S-R o D.

En última instancia, la excitación se utiliza en la entrada o entradas de control de cada uno de los FLIP FLOPS que definen el estado del circuito y lo que se necesita determinar es el valor que deben tener estas entradas, ya sea para cambiar el estado del FLIP FLOP, o para dejarlo tal como está. Los tipos de cambio o transiciones que pueden ocurrir en un FLIP FLOP se muestran en la tabla 19.1. En ésta se muestra el estado presente del FLIP FLOP, identificado como Q; su estado

futuro (el estado al cual se desea cambiar), denotado por Q^+ , y el nombre que se le dará a la transición. Cuando en Q^+ se indica una X significa que no nos interesa el valor que tome, puede ser 0 o 1.

Q	Q^+	Transición (t)
0	0	0
0	1	∞
0	X	X
1	0	β
1	1	1
1	X	X

Tabla 19.1 Tabla de transiciones

En la tabla 19.2 se muestra la tabla de excitación para cada uno de los diferentes tipos de FLIP FLOP. En ésta se muestra la transición que se quiere que ocurra, y los valores que deben tomar cada una de las entradas de control de los FLIP FLOP para que realicen el cambio especificado.

Transición (t)	D	T	S	R	J	K
0	0	0	0	X	0	X
1	1	0	X	0	X	0
∞	1	1	1	0	1	X
β	0	1	0	1	X	1
X	X	X	X	X	X	X

Tabla 19.2. Tabla de excitación para los FLIP FLOPS

Esta tabla se deduce de las tablas de funcionamiento de cada uno de los FLIP FLOPS que se obtuvieron en el capítulo 17. Se tomarán algunos casos para interpretar la tabla de excitación. Supóngase que se tiene un FLIP FLOP D y que se encuentra el estado 0 ($Q=0$). Si se quiere que cuando se presente la señal de reloj este FLIP FLOP permanezca en el estado 0 (transición 0) la entrada D deberá tener el valor de 0. En cambio, si se quiere que cambie al estado 1 (transición ∞), su entrada D deberá tener el valor de 1.

Ahora suponga que se tiene un FLIP FLOP J-K y se encuentra en el estado 1. Si se desea que permanezca en este estado cuando se presente la señal de reloj (transición 1), sus entradas pueden valer $J=0$ y $K=0$. También permanecerá en el estado 1 si se ponen sus entradas como $J=1$ y $K=0$. En la tabla de excitación se puede ver que para este caso se marca $J=X$ y $K=0$, es decir, K debe ser cero pero el valor de J no importa (puede ser 0 o 1).

Si se tiene un FLIP FLOP S-R en el estado 1 y se desea que cambie al estado 0 (transición β), se deben tener las entradas $S=0$ y $R=1$. Si lo que se desea es que permanezca en el estado 1 (transición 1), se puede lograr con $S=0$ y $R=0$, o bien, con $S=1$ y $R=0$. En la tabla se indica $S=X$ y $R=0$.

El lector podrá comprobar que todos los valores de las entradas de control para cada una de las transiciones indicadas en la tabla de excitación son correctas. Esta tabla se utilizará posteriormente en el procedimiento de diseño de circuitos secuenciales.

19.5. Diseño de circuitos secuenciales sincrónicos.

El diseño de los circuitos secuenciales sincrónicos realmente no es tan difícil como pudiera parecer a primera instancia. Para diseñarlos es necesario realizar los siguientes pasos:

1. Definir y construir el diagrama de estado.
2. Realizar la reducción de estados, si es factible.
3. Determinar el número de FLIP FLOPS requeridos.
4. Dar nombre a las variables de estado y asignar códigos a cada estado.
5. Escoger el tipo de FLIP FLOPS a utilizar.
6. Construir las tablas de: entradas, estado futuro, transiciones, excitaciones y de salida.
7. Obtener las ecuaciones para determinar la excitación y las salidas
8. Construir el circuito.

Para explicar el procedimiento de diseño se utilizarán varios ejemplos.

Ejemplo 1.

El primer ejemplo será el contador binario sincrónico de 0 a 7 cuyo circuito (figura 19.3) se analizó al inicio de este capítulo.

Paso1. Definir y construir el diagrama de estado.

En la figura 19.15 se muestra la definición y el diagrama de estados del contador binario de 0 a 7. Para este ejemplo, la salida del circuito será el estado del mismo.

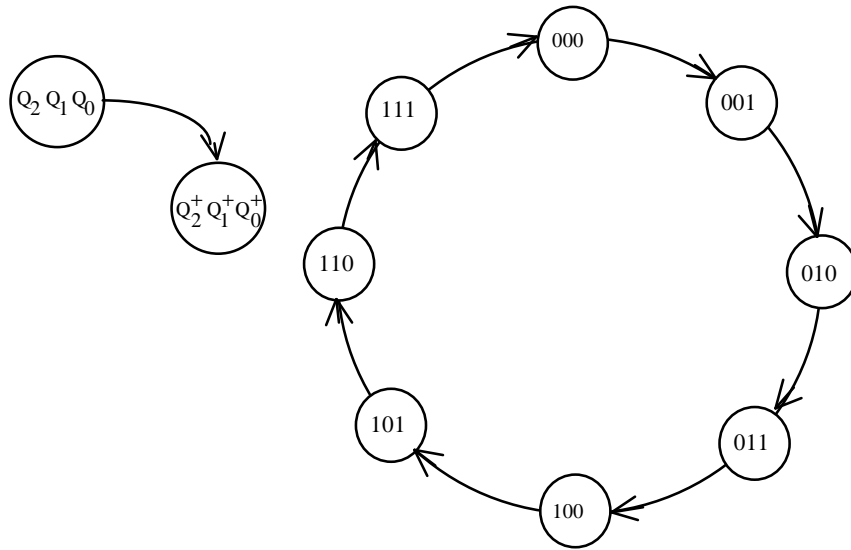


Figura 19.15. Diagrama de estado del ejemplo1.

Paso 2. Realizar la reducción de estados.

La reducción de estados consiste en revisar los diferentes estados del sistema y ver si existen estados equivalentes, es decir, que tengan las mismas salidas y realicen las mismas transiciones hacia otros estados o que, bajo ciertas consideraciones, puedan ser considerados como estados equivalentes (ver ejemplo 3). El tener menos estados permite utilizar menos FLIP FLOPS para construir el circuito, resultando en un diseño más simple. Para el ejemplo que se está diseñando no existen estados equivalentes y no se pueden reducir.

Paso 3. Determinar el número de FLIP FLOPS requeridos.

Recuérdese que con N FLIP FLOPS se pueden tener como máximo 2^N estados diferentes. Si se tiene E estados, se necesitarán $\log_2 E$ FLIP FLOPS. Si se obtiene un número fraccionario, deberá redondearse al entero superior. Por ejemplo, si se tuvieran 12 estados, se obtendrían $\log_2 12 = 3.585$ FLIP FLOPS, que se deben redondear a 4. Para el ejemplo actual, se requieren $\log_2 8 = 3$ FLIP FLOPS.

Paso 4. Dar nombre a las variables de estado y asignar códigos a cada estado.

Este paso sólo se realiza cuando la definición que se realiza del diagrama de estados en el paso 1, no se puede utilizar en el diseño del circuito. En este ejemplo se utilizará la definición que se dio en el diagrama de estados.

Paso 5. Escoger el tipo de FLIP FLOPS a utilizar.

Para poderlo comparar con el circuito presentado, se utilizarán FLIP FLOPS del tipo T. En el proceso de diseño de circuitos secuenciales sincrónicos, no influye si los FLIP FLOPS son maestro-esclavo o activados por transición, siempre y cuando todos los FLIP FLOPS sean del mismo tipo y la frecuencia del reloj que se utilice respete los tiempos de operación de éstos.

Paso 6. Construir las tablas de: entradas, estado futuro, transiciones, excitaciones y salida.

La tabla de entradas y estado futuro muestra el comportamiento que tendrá el circuito, tiene la información que se indica en el diagrama de estados. Se acostumbra a colocar en la tabla de entradas, todos los estados en orden ascendente (tabla 19.3 a) y en la parte de estado futuro, el estado al que se quiere que cambie el circuito al encontrarse en el estado actual correspondiente y, llegue la señal del reloj (tabla 19.3 b).

La tabla de transiciones representa en forma condensada la información de las tablas anteriores (tabla 19.3 c). Se usa la nomenclatura definida en la tabla 19.1.

La tabla de excitaciones muestra las entradas de control que deben tener cada uno de los FLIP FLOPS para realizar las transiciones que se indican en la tabla anterior (tabla

19.3 d). Se usa la información de la tabla 19.2 para encontrar el valor de excitación requerido para realizar cada una de las transiciones.

Par este ejemplo no se tendrá una tabla de salida, ya que la salida del circuito es el estado mismo.

Q_2	Q_1	Q_0	Q_2^+	Q_1^+	Q_0^+	t_2	t_1	t_0	T_2	T_1	T_0
0	0	0	0	0	1	0	0	∞	0	0	1
0	0	1	0	1	0	0	∞	β	0	1	1
0	1	0	0	1	1	0	1	∞	0	0	1
0	1	1	1	0	0	∞	β	β	1	1	1
1	0	0	1	0	1	1	0	∞	0	0	1
1	0	1	1	1	0	1	∞	β	0	1	1
1	1	0	1	1	1	1	1	∞	0	0	1
1	1	1	0	0	0	β	β	β	1	1	1

(a)
(b)
(c)
(d)

Tabla 19.3. tablas del ejemplo 1. (a) tabla de entradas, (b) tabla de estado futuro, (c) tabla de transiciones, (d) tabla de excitaciones.

Paso 7. Obtener las ecuaciones para determinar la excitación y las salidas.

Las variables de control de los FLIP FLOPS en este caso son T_2 , T_1 y T_0 y dependen del estado actual del circuito. De la tabla de excitación se puede ver que $T_0 = 1$. T_2 y T_1 se pueden obtener mediante mapas de Karnaugh:

$Q_1 \backslash Q_0$		00	01	11	10
		0	1	1	0
Q_2	0	0	1	1	0
	1	0	1	1	0

Del mapa de Karnaugh se obtiene $T_1 = Q_0$

$Q_1 \backslash Q_0$		00	01	11	10
		0	0	1	0
Q_2	0	0	0	1	0
	1	0	0	1	0

Del mapa de Karnaugh se obtiene $T_2 = Q_1 Q_0$

Las salidas en este caso son: Q_2 , Q_1 y Q_0 .

Paso 8. Construir el circuito.

El circuito es exactamente el mismo que el mostrado en la figura 19.3.

Ejemplo 2.

El siguiente ejemplo es un contador binario, que sigue una determinada secuencia y no usa todos los estados posibles. Una posible definición del circuito secuencial se presenta a continuación:

“Diseñe el mínimo circuito secuencial síncronico que siga la siguiente secuencia: 0, 1, 4, 7, 3, 0, 1, 4, ...”.

Paso1. Definir y construir el diagrama de estado.

En la figura 19.16 se muestra la definición y el diagrama de estados. También en este ejemplo la salida del circuito será el estado de éste.

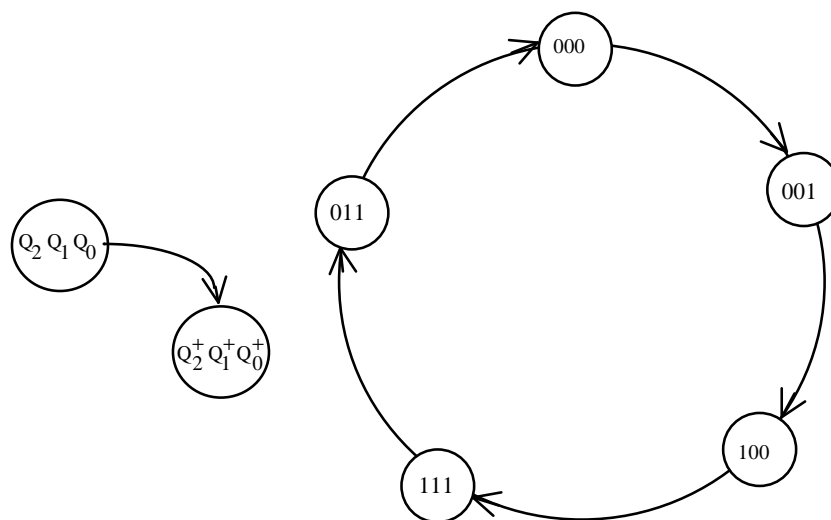


Figura 19.16. Diagrama de estado del ejemplo 2 .

Paso 2. Realizar la reducción de estados.

Para el ejemplo que se está diseñando no existen estados equivalentes y no se pueden reducir.

Paso 3. Determinar el número de FLIP FLOPS requeridos.

Para el ejemplo actual, se requieren $\log_2 5 = 2.32$, por lo cual se requieren 3 FLIP FLOPS.

Paso 4. Dar nombres a las variables de estado y asignar códigos a cada estado.

En este ejemplo se utilizará la definición que se da en el diagrama de estados.

Paso 5. Escoger el tipo de FLIP FLOPS a utilizar.

Se utilizará un FLIP FLOP del tipo S-R para Q_2 , un FLIP FLOP del tipo D para Q_1 , y un FLIP FLOP del tipo T para Q_0 .

Paso 6. Construir las tablas de: entradas, estado futuro, transiciones, excitaciones y salida.

Para nuestro ejemplo, las tablas de: entradas, estado futuro, transiciones, excitaciones se muestra en la tabla 19.4. Las tablas se forman de la misma forma que en el ejemplo anterior, con excepción de la tabla de estado futuro, donde se puso una X, si la entrada correspondiente no esta en la secuencia del contador, para poder obtener el circuito mínimo.

Par este ejemplo no se tendrá una tabla de salida, ya que la salida del circuito es el estado mismo.

Q_2	Q_1	Q_0	Q_2^+	Q_1^+	Q_0^+	t_2	t_1	t_0	S_2	R_2	D_1	T_0
0	0	0	0	0	1	0	0	∞	0	X	0	1
0	0	1	1	0	0	∞	0	β	1	0	0	1
0	1	0	X	X	X	X	X	X	X	X	X	X
0	1	1	0	0	0	0	β	β	0	X	0	1
1	0	0	1	1	1	1	∞	∞	X	0	1	1
1	0	1	X	X	X	X	X	X	X	X	X	X
1	1	0	X	X	X	X	X	X	X	X	X	X
1	1	1	0	1	1	β	1	1	0	1	1	0
(a)			(b)			(c)			(d)			

Tabla 19.4. tablas del ejemplo 2. (a) tabla de entradas, (b) tabla de estado futuro, (c) tabla de transiciones, (d) tabla de excitaciones.

Paso 7. Obtener las ecuaciones para determinar la

excitación y las salidas.

Las variables de control de los FLIP FLOPS en este caso son S_2 , R_2 , D_1 y T_0 . De la tabla de excitación se puede obtener mediante mapas de Karnaugh estas variables.

$Q_1 Q_0$		Q_2			
		00	01	11	10
Q_2	0	0	1	0	X
	1	X	X	0	X

mapa para S_2

$Q_1 Q_0$		Q_2			
		00	01	11	10
Q_2	0	X	0	X	X
	1	0	X	1	X

mapa para R_2

De los mapas de Karnaugh se obtiene $S_2 = Q_1 \cdot Q_0$ y que $R_2 = Q_1$.

$Q_1 Q_0$		Q_2			
		00	01	11	10
Q_2	0	0	0	0	X
	1	1	X	1	X

mapa para D_1

La ecuación para D_1 es: $D_1 = Q_2$.

$Q_1 Q_0$		Q_2			
		00	01	11	10
Q_2	0	1	1	1	X
	1	1	X	0	X

mapa para T_0

La ecuación para T_0 es: $T_0 = Q_2 \cdot Q_1$

Paso 8. Construir el circuito.

El circuito se muestra en la figura 19.17.

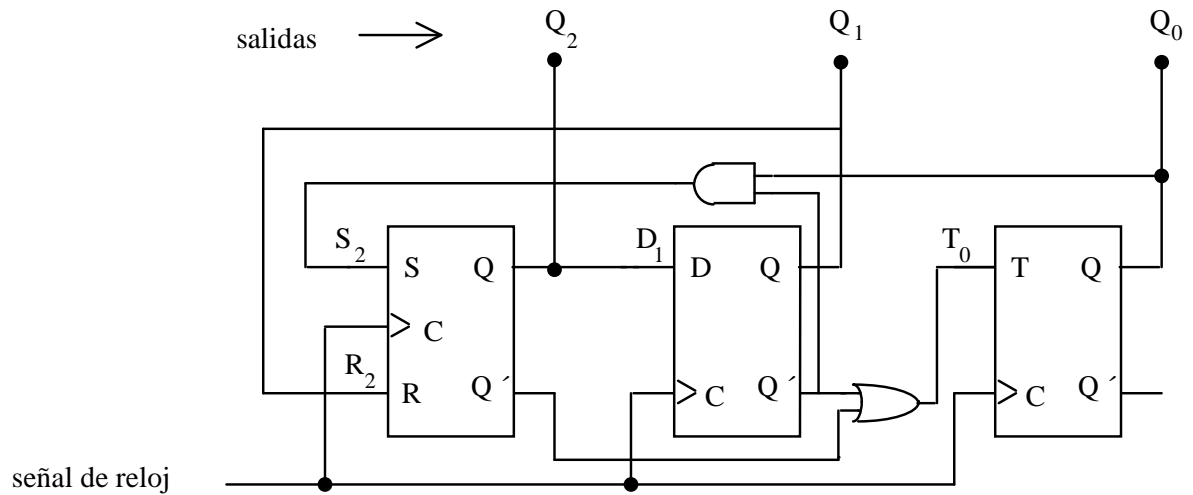


Figura 19.18. circuito del contador binario del ejemplo 2.

La figura 19.18 muestra el diagrama de estados completo, ya que en el diagrama de la figura 19.16 so se sabía que pasaba si el circuito se enciende en alguno de los estados que no están presentes. El diagrama de estados completo se obtiene analizando en el circuito anterior lo que sucede al suponer que el circuito se encuentra en uno de estos estados y llega la señal del reloj.

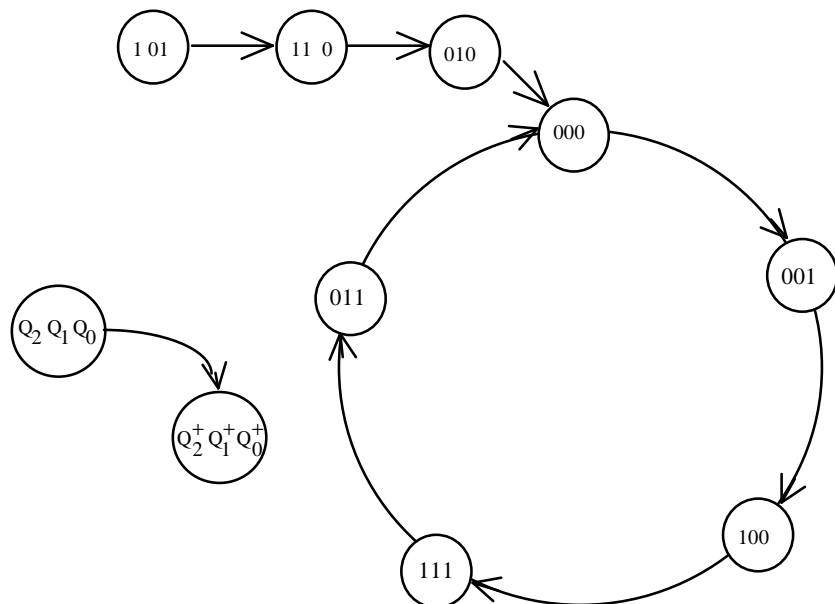


Figura 19.18. Diagrama de estado completo del contador binario del ejemplo 2

Ejemplo 3.

El siguiente ejemplo es el contador binario de 0 a 3 que se presentó anteriormente. Una posible definición del circuito secuencial se presenta a continuación:

“Diseñe un contador binario de 0 a 3 que pueda contar hacia arriba o hacia abajo, dependiendo del valor de una entrada externa. Si la entrada externa vale uno, deberá contar hacia arriba; si esta entrada vale cero, deberá contar hacia abajo.”

Paso1. Definir y construir el diagrama de estado.

La definición específica que debe contar de 0 a 3, lo cual indica que debe tener 4 estados, si la entrada externa, que se denotará por X, tiene el valor de uno, la secuencia de estados debe ser 0,1,2,3,0, ... y así sucesivamente. Si la entrada externa vale cero, la secuencia debe ser 0,3,2,1,0, ..., etc. En la figura 19.19 se muestra la definición y el diagrama de estados. También en este ejemplo la salida del circuito será el estado de éste.

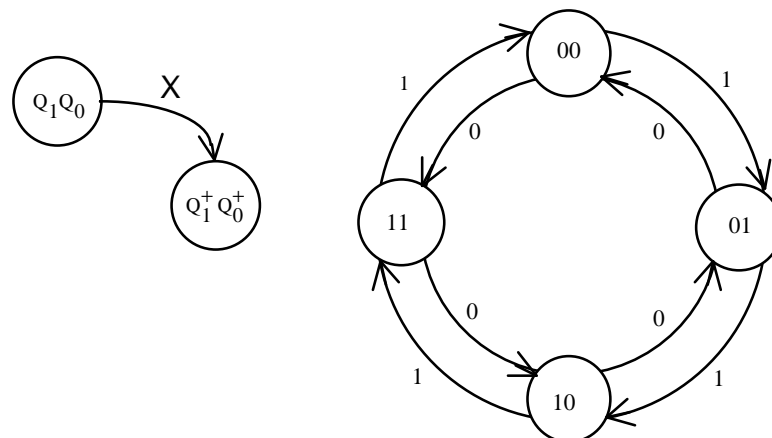


Figura 19.19. Diagrama de estado del ejemplo 3.

Paso 2. Realizar la reducción de estados.

Para el ejemplo que se está diseñando no existen estados equivalentes y no se pueden reducir.

Paso 3. Determinar el número de FLIP FLOPS requeridos.

Para el ejemplo actual, se requieren $\log_2 4=2$ FLIP FLOPS.

Paso 4. Dar nombres a las variables de estado y asignar códigos a cada estado.

En este ejemplo se utilizará la definición que se da en el diagrama de estados.

Paso 5. Escoger el tipo de FLIP FLOPS a utilizar.

Para poderlo comparar con el circuito presentado, se utilizarán FLIP FLOPS del tipo T.

Paso 6. Construir las tablas de: entradas, estado futuro, transiciones, excitaciones y salida.

En este ejemplo la tabla de entradas se forma con la enumeración de todas las combinaciones que se pueden formar con la variable de entrada X y los estados que puede tener el circuito. Las demás tablas se forman de la misma forma que en el ejemplo anterior.

Para nuestro ejemplo, las tablas de entradas, estado futuro, transiciones y excitaciones se muestran en la tabla 19.5. En este caso no se tendrá una tabla de salida, ya que la salida del circuito es el estado mismo.

X	Q ₁	Q ₀	Q ₁ ⁺	Q ₀ ⁺	t ₁	t ₀	T ₁	T ₀
0	0	0	1	1	∞	∞	1	1
0	0	1	0	0	0	β	0	1
0	1	0	0	1	β	∞	1	1
0	1	1	1	0	1	β	0	1
1	0	0	0	1	0	∞	0	1
1	0	1	1	0	∞	β	1	1
1	1	0	1	1	1	∞	0	1
1	1	1	0	0	β	β	1	1

(a)
(b)
(c)
(d)

Tabla 19.5. tablas del ejemplo 3. (a) tabla de entradas, (b) tabla de estado futuro, (c) tabla de transiciones, (d) tabla de excitaciones.

Paso 7. Obtener las ecuaciones para determinar la excitación y las salidas.

Las variables de control de los FLIP FLOPS en este caso son T₁ y T₀ y dependen, en general, del estado presente y de la entrada. De la tabla de excitación se puede ver que T₀ = 1. T₁ se puede obtener mediante un mapa de Karnaugh:

$Q_1 Q_0$ X		00	01	11	10
		0	1	0	0
	1	0	1	1	0

Del mapa de Karnaugh se obtiene $T_1 = X' Q_0' + X Q_0$

Paso 8. Construir el circuito.

El circuito es exactamente el mismo que el mostrado en la figura 19.9.

Se trabajará el mismo ejemplo suponiendo que en el paso 5 se hubiesen escogido FLIP FLOPS D. Las nuevas tablas que obtienen en el paso 6 sería la mostrada en la tabla 19.6, donde las entradas de control a los FLIP FLOPS ahora son D_1 y D_0 .

X	Q_1	Q_0	Q_1^+	Q_0^+	t_1	t_0	D_1	D_0
0	0	0	1	1	∞	∞	1	1
0	0	1	0	0	0	β	0	0
0	1	0	0	1	β	∞	0	1
0	1	1	1	0	1	β	1	0
1	0	0	0	1	0	∞	0	1
1	0	1	1	0	∞	β	1	0
1	1	0	1	1	1	∞	1	1
1	1	1	0	0	β	β	0	0

(a)

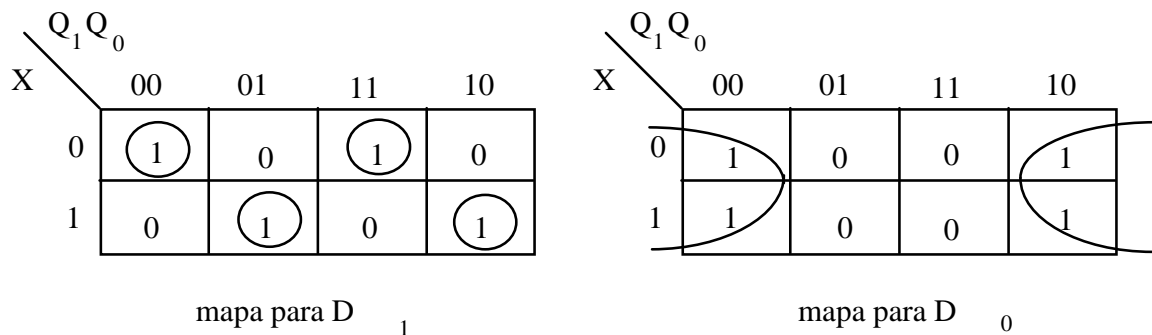
(b)

(c)

(d)

Tabla 19.6. tablas nuevas para el ejemplo 3. (a) tabla de entradas, (b) tabla de estado futuro, (c) tabla de transiciones, (d) tabla de excitaciones

Las ecuaciones para las entradas de control a los FLIP FLOPS se pueden obtener usando los mapas de Karnaugh.



La función para D_1 no se puede simplificar y es equivalente a un EX-OR de las tres variables. Las funciones de la lógica para determinar el siguiente estado son:

$$D_1 = X \oplus Q_1 \oplus Q_0 \quad \text{y} \quad D_0 = Q_0'$$

El circuito para este caso se muestra en la figura 19.20. Al cambiar el tipo de FLIP FLOPS que se utilizan, cambia la lógica para determinar el siguiente estado.

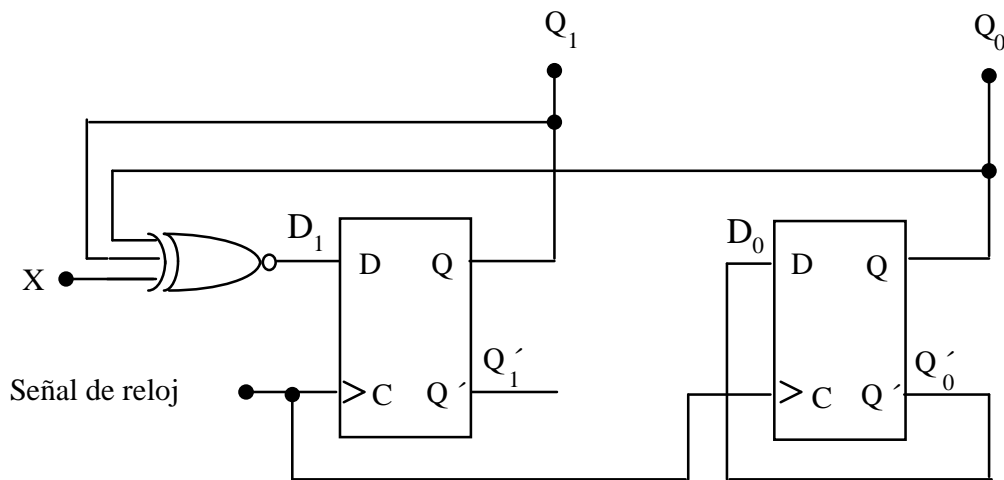


Figura 19.20. Circuito lógico del ejemplo 3, construido con FLIP FLOPS D.

Ejemplo 4.

El siguiente ejemplo es un circuito secuencial que tiene una entrada X y una salida Z y realiza lo siguiente: en cada transición positiva del reloj, se muestrea la señal de entrada X . La salida Z será igual a 1, sólo cuando, el número de unos y ceros que se han detectado en X , sean múltiplo de 2.

Paso1. Definir y construir el diagrama de estado.

En este tipo de problemas el principal problema que se tiene es la definición del estado, es decir, que significado se le va a dar a cada uno de los estados. Por ejemplo si se define cada estado como se indica en la figura 19.21, se tendrá el problema que el diagrama de estados nunca terminaría de construirse y no es posible obtener una solución viable.

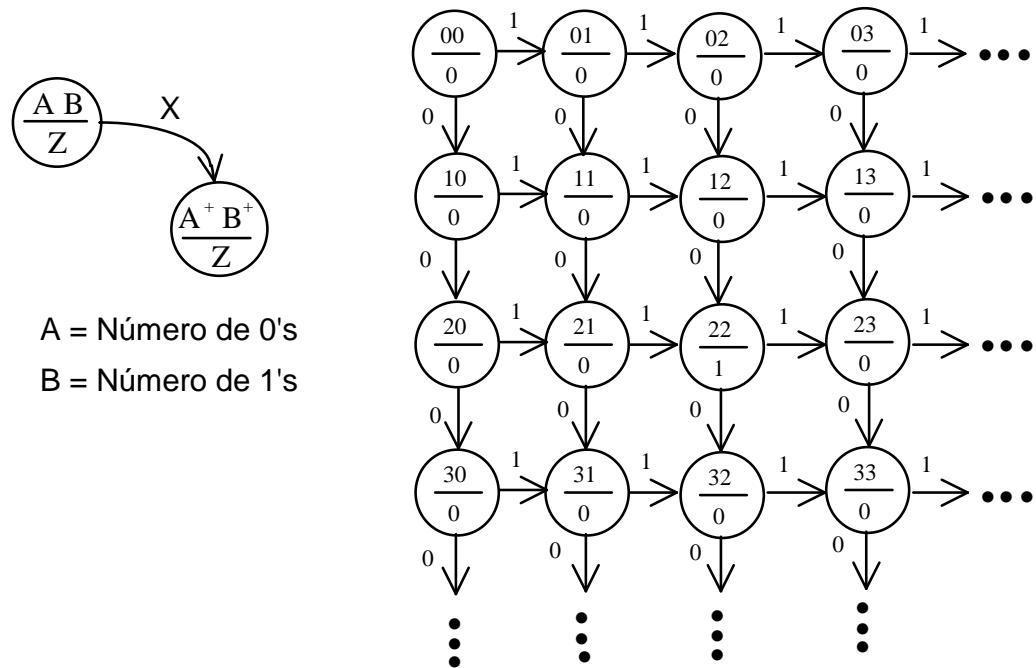


Figura 19.21. Posible diagrama de estados para el ejemplo 4

Si se usa la definición que se da en la figura 19.22, este diagrama de estados si tiene una solución ya que si tiene un número finito de estados. En realidad lo único que necesita recordar el circuito es; si la cantidad de unos y ceros son pares, y no la cantidad exacta de unos y ceros.

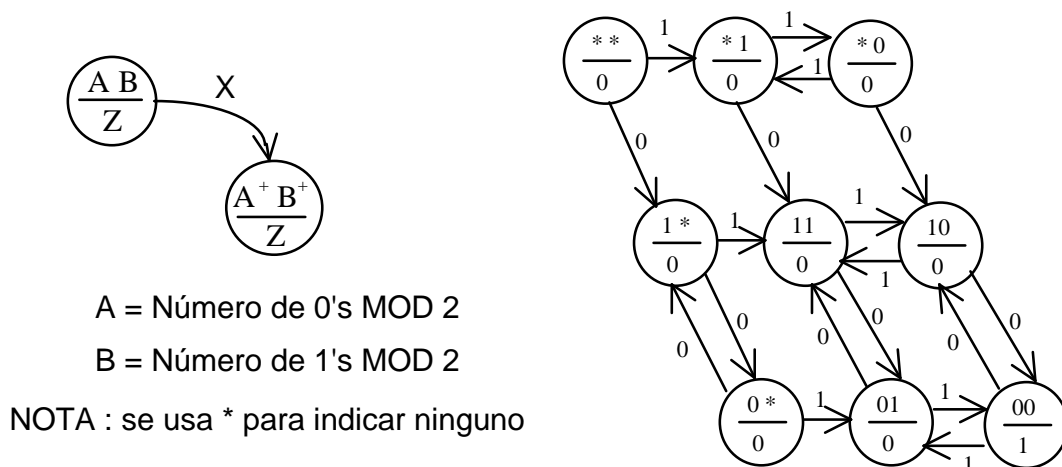


Figura 19.22. Diagrama de estado del ejemplo 4.

Dado que las salida no depende de la entrada, ambos diagramas de estados utilizan el modelo de Moore.

Paso 2. Realizar la reducción de estados.

Al observar el diagrama de estados de este ejemplo se puede observar que existen 5 estados que solo se utilizan al principio (estados con uno o dos *). Se considera que, cuando no se a muestreado la señal X, la cantidad de unos y ceros son pares ya que son ambas cero, entonces, el circuito puede iniciar en el estado 00 y se pueden eliminar estos cinco estados. El diagrama reducido se muestra en la figura 19.23

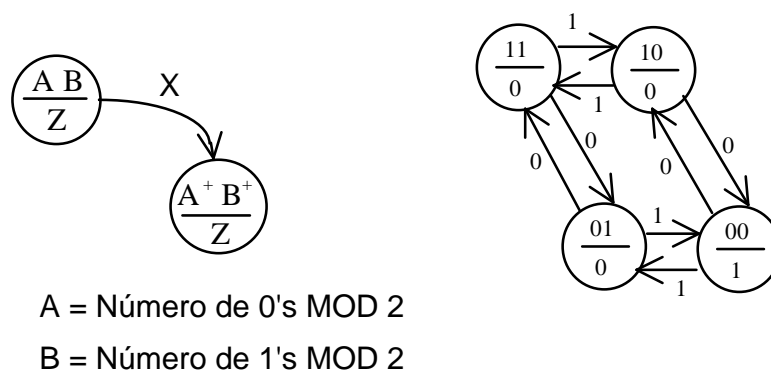


Figura 19.23. Diagrama de estado reducido del ejemplo 4.

Paso 3. Determinar el número de FLIP FLOPS requeridos.

Para el ejemplo actual, se requieren $\log_2 4=2$ FLIP FLOPS.

Paso 4. Dar nombres a las variables de estado y asignar

códigos a cada estado.

Se usarán las variables Q_1 y Q_0 para identificar a los dos FLIP FLOPS y se usarán en este orden para identificar los estados. Se asignarán los códigos 00 al estado donde A y B toman el valor de 00, el código 01 al estado donde A y B toman el valor de 01, el código 10 al estado donde A y B toman el valor de 10, y el código 11 al estado donde A y B toman el valor de 11. El nuevo diagrama de estados se muestra en la figura 19.24.

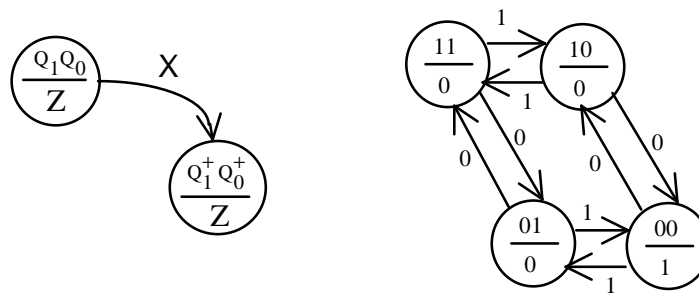


Figura 19.24. Nuevo diagrama de estado del ejemplo 4.

Paso 5. Escoger el tipo de FLIP FLOPS a utilizar.

Se utilizarán FLIP FLOPS del tipo J-K.

Paso 6. Construir las tablas de: entrada, estado futuro, transiciones, excitaciones y salida.

En este ejemplo la tabla se forman igual que en el ejemplo anterior, solo que en este ejemplo se tendrá que formar la tabla de salida, ya que el circuito si tiene una salida diferente a la del estado del circuito la cual es Z. La tabla de salida se construye aparte ya que este circuito representa a un modelo de Moore, en que la salida no depende de la entrada, como se muestra en la tabla 19.7

X	Q_1	Q_0	Q_1^+	Q_0^+	t_1	t_0	J_1	K_1	J_0	K_0
0	0	0	1	0	∞	0	1	X	0	X
0	0	1	1	1	∞	1	1	X	X	0
0	1	0	0	0	β	0	X	1	0	X
0	1	1	0	1	β	1	X	1	X	0
1	0	0	0	1	0	∞	0	X	1	X
1	0	1	0	0	0	β	0	X	X	1
1	1	0	1	1	1	∞	X	0	1	X
1	1	1	1	0	1	β	X	0	X	1

(a)

(b)

(c)

(d)

Q_1	Q_0	Z
0	0	1
0	1	0
1	0	0
1	1	0

(e)

Tabla 19.7. tablas del ejemplo 4. (a) tabla de entradas, (b) tabla de estado futuro, (c) tabla de transiciones, (d) tabla de excitaciones, (e) tabla de salida.

Paso 7. Obtener las ecuaciones para determinar la excitación y las salidas.

Las variables de control de los FLIP FLOPS en este caso son J_1 , K_1 , J_0 y K_0 y dependen, en general, del estado presente y de la entrada X , las ecuaciones de cada uno de éstas se pueden obtener mediante mapas de Karnaugh:

$Q_1 Q_0$ X		00	01	11	10
		0	1	X	X
X	0	1	1	X	X
	1	0	0	X	X

mapa para J_1

$Q_1 Q_0$ X		00	01	11	10
0	X	X	1	1	
1	X	X	0	0	

mapa para K_1

De los mapa de Karnaugh se obtiene $J_1 = K_1 = X'$.

$Q_1 Q_0$ X		00	01	11	10
		0	0	X	X
1	1	1	X	X	1

mapa para J_0

$Q_1 Q_0$ X		00	01	11	10
0	X	0	0	X	
1	X	1	1	X	

mapa para K_0

De los mapa de Karnaugh se obtiene $J_0 = K_0 = X$.

De la tabla de salida se puede observar que, $Z = Q_1' Q_0'$.

Si se hubieran asignado los códigos de manera diferente, se hubieran tenido ecuaciones diferentes para la salidas. Las ecuaciones para la excitación también cambiarían si se asignaran otros códigos a los estados.

Paso 8. Construir el circuito.

El la figura 19.25 se muestra el circuito lógico.

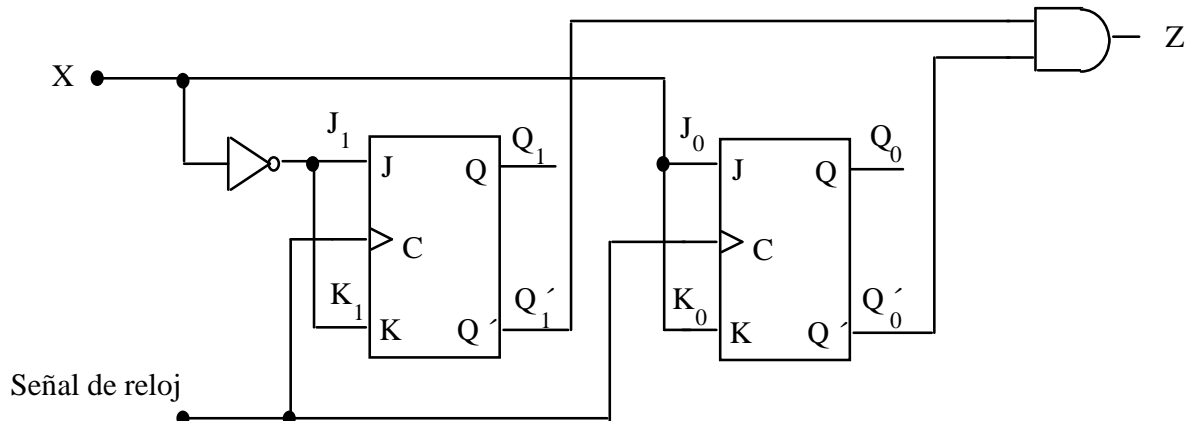


Figura 19.25. Circuito lógico del ejemplo 4.

Ejemplo 5.

Se diseñará otro circuito secuencial síncrono para reafirmar la metodología de diseño. La operación de este circuito se describe a continuación:

“Diseñe un contador binario síncrono ascendente de 0 a 3 que tenga una entrada externa. Si el valor de la entrada es cero, la salida del circuito debe ser el número binario correspondiente a su estado. Si el valor de la entrada es uno, la salida debe ser el complemento a dos del número binario correspondiente a su estado”.

Paso1. Definir y construir el diagrama de estado.

La definición específica que debe contar de 0 a 3, lo cual indica que debe tener 4 estados. Si la entrada externa, que se denotará por X, tiene el valor de cero, las salidas serán directamente el valor binario del estado. Si la entrada externa vale uno, las salidas deberán ser el complemento a dos del estado. El diagrama de estados se muestra en la figura 19.26.

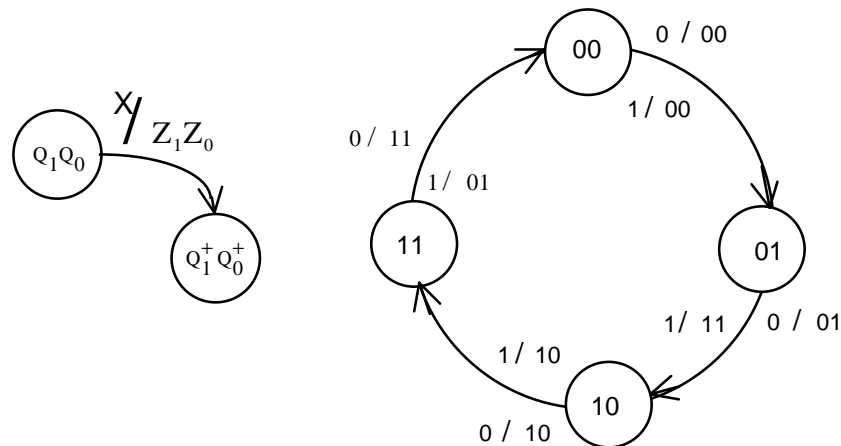


Figura 19.26. Diagrama de estado del ejemplo 5.

En este caso las salidas dependen de la entrada, por lo tanto, se utilizó el modelo de Mealy. Por ejemplo, si el circuito se encuentra en el estado 01 y la entrada vale cero, las salidas deben ser 01 y el siguiente estado será el 10. Si la entrada vale uno y está en el mismo estado 01, las salidas serán 11 y el siguiente estado será el 10. Las variables de salida se identificarán como Z_1 y Z_0 .

Paso 2. Realizar la reducción de estados.

En este caso no existen dos estados que tengan las mismas salidas y las mismas transiciones, por consiguiente, no se puede reducir el número de estados.

Paso 3. Determinar el número de FLIP FLOPS requeridos.

Para este caso se requieren $\log_2 4 = 2$ FLIP FLOPS.

Paso 4. Dar nombres a las variables de estado y asignar códigos a cada estado.

En este ejemplo se utilizará la definición que se dio en el diagrama de estados.

Paso 5. Escoger el tipo de FLIP FLOPS a utilizar.

Se utilizarán un FLIP FLOP J-K para Q_1 y un FLIP FLOP S-R para Q_0 .

Paso 6. Construir las tablas de: entrada, estado futuro, transiciones, excitaciones y salida.

Las variables independientes para este ejemplo son las dos variables de estado Q_1 y Q_0 , y la variables externa X . Las ecuaciones de la lógica para determinar el siguiente estado corresponden a las entradas de control para los dos FLIP FLOPS J-K S-R. Es necesario determinar ecuaciones para J_1 , K_1 , S_0 y R_0 . Estas son las ecuaciones de la

excitación. Las ecuaciones de salida son las que proporcionan los valores de Z_1 y Z_0 , en función del estado presente, Q_1 y Q_0 , y la entrada externa X . Las tablas para este ejemplo se muestra en la tabla 19.8.

X	Q_1	Q_0	Q_1^+	Q_0^+	t_1	t_0	J_1	K_1	S_0	R_0	Z_1	Z_0
0	0	0	0	1	0	∞	0	X	1	0	0	0
0	0	1	1	0	∞	β	1	X	0	1	0	1
0	1	0	1	1	1	∞	X	0	1	0	1	0
0	1	1	0	0	β	β	X	1	0	1	1	1
1	0	0	0	1	0	∞	0	X	1	0	0	0
1	0	1	1	0	∞	β	1	X	0	1	1	1
1	1	0	1	1	1	∞	X	0	1	0	1	0
1	1	1	0	0	β	β	X	1	0	1	0	1

(a)
(b)
(c)
(d)
(e)

Tabla 19.8. Tablas del ejemplo 5. (a) tabla de entradas, (b) tabla de estado futuro, (c) tabla de transiciones, (d) tabla de excitaciones, (e) tabla de salida.

Paso 7. Obtener las ecuaciones para determinar la excitación y las salidas.

Se utilizarán mapas de Karnaugh para obtener las ecuaciones para J_1 , K_1 , J_0 , K_0 , Y_1 y Y_0 a partir de la tabla de excitación y salidas.

		$Q_1 Q_0$			
		00	01	11	10
X	0	0	1	X	X
	1	0	1	X	X

mapa para J_1

		$Q_1 Q_0$			
		00	01	11	10
X	0	X	X	1	0
	1	X	X	1	0

mapa para K_1

De los mapas de Karnaugh se obtiene $J_1 = K_1 = Q_0$.

		$Q_1 Q_0$			
X		00	01	11	10
	0	1	0	0	1
	1	1	0	0	1

mapa para S_0

La ecuación para S_0 es: $S_0 = Q_0'$

La ecuación para R_0 es: $R_0 = Q_0$

		$Q_1 Q_0$			
X		00	01	11	10
	0	0	1	1	0
	1	0	1	1	0

mapa para R_0

		$Q_1 Q_0$			
X		00	01	11	10
	0	0	0	1	1
	1	0	1	0	1

mapa para Z_1

		$Q_1 Q_0$			
X		00	01	11	10
	0	0	1	1	0
	1	0	1	1	0

mapa para Z_0

La ecuación para Z_1 es: $Z_1 = X' Q_1 + Q_1 Q_0' + X Q_1' Q_0$

La ecuación para Z_0 es: $Z_0 = Q_0$

Paso 8. Construir el circuito.

En la figura 19.27 se muestra el circuito lógico construido con arreglos de ANDS y ORS.

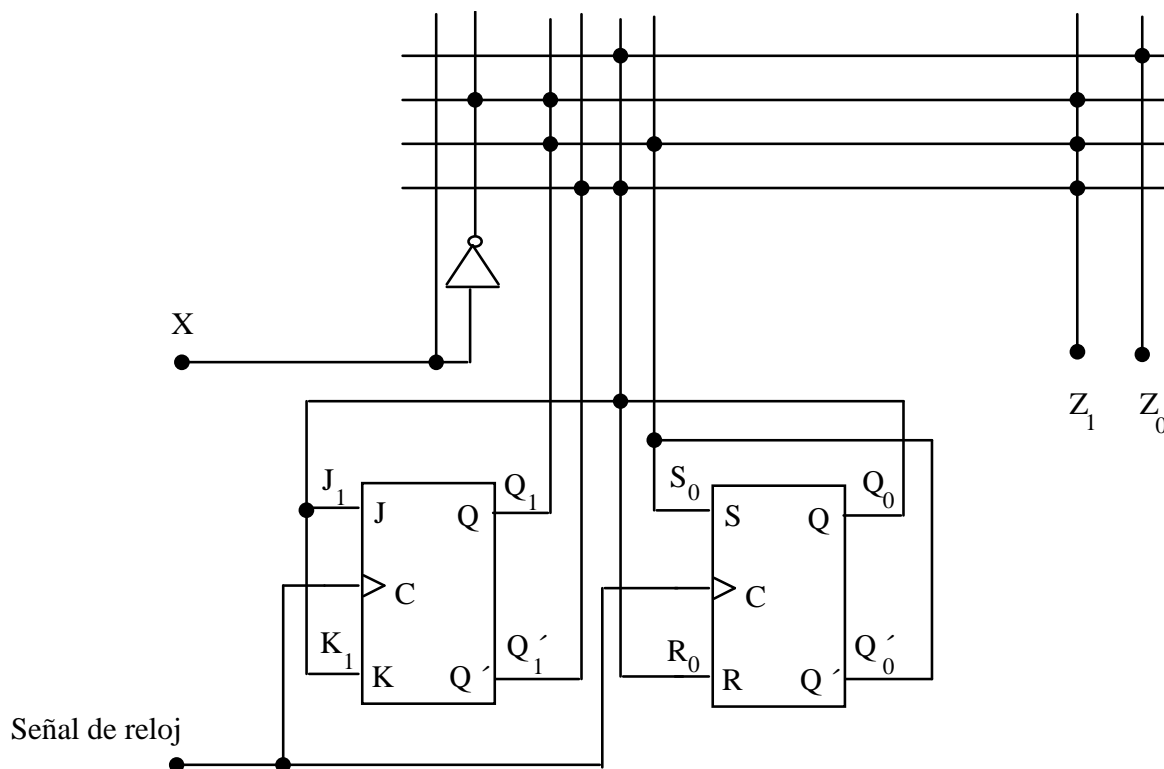


Figura 19.23. Circuito lógico con arreglos de ANDS y ORS del ejemplo 5.

El arreglo de la izquierda es un arreglo de ANDS y el de la derecha es un arreglo de ORS. Esta forma de dibujarlo es conveniente cuando el circuito se va a construir utilizando arreglos lógicos programables (ver capítulo 15).

19.6. Diseño de circuitos secuenciales asincrónicos.

El procedimiento de diseño de los circuitos secuenciales asincrónicos, con una señal de reloj que controla a una parte de los FLIP FLOPS del circuito es muy parecido al diseño de circuitos secuenciales sincrónicos, solo se tiene un paso adicional que viene después de obtener la tabla de transiciones. Es en este momento cuando se puede determinar si se puede construir un circuito asincrónico.

Para poder construir un circuito asincrónico, por lo menos se tiene que conectar la salida de un FLIP FLOP a la entrada de reloj de otro FLIP FLOP. En este caso, el FLIP FLOP cuya entrada de reloj está controlada por la salida del otro se denominará FLIP FLOP dependiente y el FLIP FLOP cuya salida se alimenta a la entrada de reloj del dependiente se denominará FLIP FLOP controlador. Esto se puede hacer siempre y cuando se cumpla la siguiente condición: cuando en el FLIP FLOP dependiente se requiera realizar un cambio (transición α o β), en el FLIP FLOP controlador debe presentarse siempre una misma transición, independientemente de que sea α o β . Puede ser que en el FLIP FLOP controlador haya más transiciones del mismo tipo y que en el

dependiente no existan transiciones. Si no existen dependencias de este tipo, el circuito no puede ser diseñado en forma asincrónica.

Para diseñarlos circuitos secuenciales asincrónicos se deben realizar los siguientes pasos:

1. Definir y construir el diagrama de estado.
2. Realizar la reducción de estados, si es factible.
3. Determinar el número de FLIP FLOPS requeridos.
4. Dar nombre a las variables de estado y asignar códigos a cada estado.
5. Escoger el tipo de FLIP FLOPS a utilizar.
6. Construir las tablas de entradas, estado futuro, transiciones y de salida.
7. Determinar dependencias entre los FLIP FLOP.
8. Construir la nueva tabla de transiciones y la tabla de excitaciones.
9. Obtener las ecuaciones para determinar la excitación y las salidas
10. Construir el circuito.

Para explicar el procedimiento de diseño se utilizarán varios ejemplos.

Ejemplo 1.

El primer ejemplo será el contador binario asincrónico de 0 a 7 cuyo circuito (figura 19.6) se analizó al inicio de este capítulo.

Paso1. Definir y construir el diagrama de estado.

En la figura 19.24 se muestra la definición y el diagrama de estados del contador binario de 0 a 7. Par este ejemplo la salida del circuito será el estado de éste.

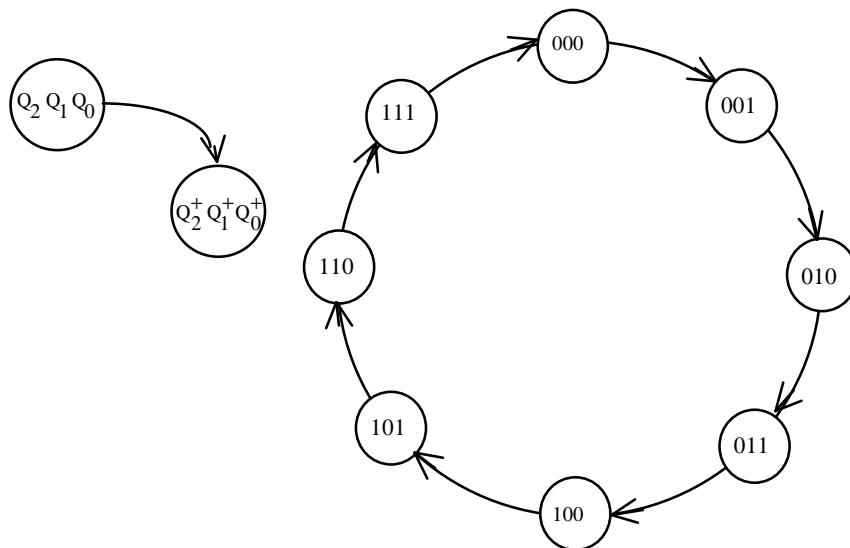


Figura 19.24. Diagrama de estado del ejemplo1.

Paso 2. Realizar la reducción de estados.

Para el ejemplo que se está diseñando no existen estados equivalentes y no se pueden reducir.

Paso 3. Determinar el número de FLIP FLOPS requeridos.

Para el ejemplo actual, se requieren $\log_2 8=3$ FLIP FLOPS.

Paso 4. Dar nombre a las variables de estado y asignar códigos a cada estado.

En este ejemplo se utilizará la definición que se da en el diagrama de estados.

Paso 5. Escoger el tipo de FLIP FLOPS a utilizar.

Para poder compararlo con el circuito presentado, se utilizarán FLIP FLOPS del tipo T. En el proceso de diseño de circuitos secuenciales asincrónicos si influye la transición en la que cambia la salida de los FLIP FLOPS, esto se explicará en el paso 10.

Paso 6. Construir las tablas de entradas, estado futuro, transiciones y salida.

Estas tablas se construyen de la misma forma que en el diseño de circuitos secuenciales sincrónicos y se muestran en la tabla 19.9. Para este ejemplo no se tendrá una tabla de salida, ya que la salida del circuito es el estado mismo.

Q ₂	Q ₁	Q ₀	Q ₂ ⁺	Q ₁ ⁺	Q ₀ ⁺	t ₂	t ₁	t ₀
0	0	0	0	0	1	0	0	∞
0	0	1	0	1	0	0	∞	β
0	1	0	0	1	1	0	1	∞
0	1	1	1	0	0	∞	β	β
1	0	0	1	0	1	1	0	∞
1	0	1	1	1	0	1	∞	β
1	1	0	1	1	1	1	1	∞
1	1	1	0	0	0	β	β	β

(a)
(b)
(c)

Tabla 19.9. Tablas del ejemplo 1. (a) tabla de entradas, (b) tabla de estado futuro, (c) tabla de transiciones.

Paso 7. Determinar dependencias entre los FLIP FLOP.

Para encontrar las posibles dependencias se realiza el siguiente procedimiento: analizar la tabla de transiciones para determinar si existe un FLIP FLOP “X” que siempre

realice alguna transición (transición ∞ o β) cuando otro FLIP FLOP “Y” realiza una misma transición. Si se presenta esta condición, el FLIP FLOP “Y” puede controlar al FLIP FLOP “X”.

En nuestro ejemplo, el FLIP FLOP 2 depende de las transiciones β del FLIP FLOP 1 ya que siempre que cambia el FLIP FLOP 2 existen transiciones β del FLIP FLOP 1. También depende de las transiciones β del FLIP FLOP 0 aún y cuando el FLIP FLOP 0 presenta otras transiciones β en las cuales el FLIP FLOP 2 no cambia. El FLIP FLOP 1 depende de las transiciones β del FLIP FLOP 0. El FLIP FLOP 0 no puede ser controlado por otro, por lo tanto debe ser controlado por la señal de reloj del circuito.

Paso 8. Construir la nueva tabla de transiciones y la tabla de excitaciones.

La nueva tabla de transiciones se forma de la siguiente manera:

La tabla de transiciones para los FLIP FLOPS que serán controlados por la señal de reloj del circuito, quedan igual.

La tabla de transiciones para los FLIP FLOPS que son controlados por otro se va a simplificar, ya que sólo es necesario preparar las entradas de control cuando ocurran las transiciones adecuadas en la entrada del reloj del FLIP FLOP dependiente. Esto no ocurre en todas las combinaciones. En aquellas combinaciones en que no ocurra la transición adecuada en la entrada de reloj se podrá colocar una X, ya que no importa el valor que se tenga en las variables de control.

La tabla de excitaciones se llena de la misma forma que en los ejemplos anteriores.

Si usamos las siguientes dependencias:

FLIP FLOP 0 depende del reloj principal.

FLIP FLOP 1 depende de las transiciones β del FLIP FLOP 0.

FLIP FLOP 2 depende de las transiciones β del FLIP FLOP 1.

Las nueva tabla de transiciones y tabla de excitaciones se muestran en la tabla 19.10.

Q_2	Q_1	Q_0	Q_2^+	Q_1^+	Q_0^+	t_2	t_1	t_0	nt_2	nt_1	nt_0	T_2	T_1	T_0
0	0	0	0	0	1	0	0	∞	X	X	∞	X	X	1
0	0	1	0	1	0	0	∞	β	X	∞	β	X	1	1
0	1	0	0	1	1	0	1	∞	X	X	∞	X	X	1
0	1	1	1	0	0	∞	β	β	∞	β	β	1	1	1
1	0	0	1	0	1	1	0	∞	X	X	∞	X	X	1
1	0	1	1	1	0	1	∞	β	X	∞	β	X	1	1
1	1	0	1	1	1	1	1	∞	X	X	∞	X	X	1
1	1	1	0	0	0	β	β	β	β	β	β	1	1	1
(a)			(b)			(c)			(d)			(e)		

Tabla 19.10. tablas del ejemplo 1. (a) tabla de entradas, (b) tabla de estado futuro, (c) tabla de transiciones, (d) nueva tabla de transiciones, (e) tabla de excitaciones.

Paso 9. Obtener las ecuaciones para determinar la excitación y las salidas.

Las variables de control de los FLIP FLOPS en este caso son T_2, T_1 y T_0 , también dependen del estado actual del circuito y de las entradas externas, si se tuvieran. De la tabla de excitación se puede ver que T_0, T_1 y T_2 son igual a uno.

Las salidas en este caso son: Q_2, Q_1 y Q_0 .

Paso 10. Construir el circuito.

Par realizar la conexión a la entrada de reloj en los FLIP FLOPS dependientes es necesario tomar en cuenta la transición en la que puede cambiar su salida (transición en la que trabaja el FLIP FLOP) así como la transición del FLIP FLOP controlador de la cual depende. En la tabla 19.11 se muestra la forma de realizar la conexión de la entrada del reloj del FLIP FLOP dependiente. Se usa ∞ para indicar que el FLIP FLOP trabaja cuando la entrada de reloj pasa de 0 a 1 y β para indicar que el FLIP FLOP trabaja cuando la entrada de reloj pasa de 1 a 0.

Transición en la que trabaja el FLIP FLOP dependiente	Transición de la que depende	
	∞	β
∞	Conectar reloj del FLIP FLOP dependiente a Q del FLIP FLOP controlador.	Conectar reloj del FLIP FLOP dependiente a Q' del FLIP FLOP controlador.
β	Conectar reloj del FLIP FLOP dependiente a Q' del FLIP FLOP controlador.	Conectar reloj del FLIP FLOP dependiente a Q del FLIP FLOP controlador.

tabal 19.11 Forma de conectar el reloj de los FLIP FLOPS dependientes.

El circuito que se obtiene es exactamente el mismo que el mostrado en la figura 19.6. Note que el reloj del FLIP FLOP 1 se conecta a Q_0 , ya que éste trabaja en transiciones negativas y depende de las transiciones negativas del FLIP FLOP 0 y el reloj del FLIP

FLOP 2 se conecta a Q_1 , ya que éste también trabaja en transiciones negativas y depende de las transiciones negativas del FLIP FLOP 1.

Los FLIP FLOPS que se conectan a la señal de reloj de circuito, pueden trabajar en la transición positiva o negativa del reloj, siempre y cuando todos sean del mismo tipo.

Se trabajará el mismo ejemplo suponiendo que en el paso 8 se hubiesen escogido las siguientes dependencias:

FLIP FLOP 0 depende del reloj principal.

FLIP FLOP 1 depende de las transiciones β del FLIP FLOP 0.

FLIP FLOP 2 depende de las transiciones β del FLIP FLOP 0.

Las nueva tabla de transiciones y tabla de excitaciones se muestran en la tabla 19.12.

Q_2	Q_1	Q_0	Q_2^+	Q_1^+	Q_0^+	t_2	t_1	t_0	nt_2	nt_1	nt_0	T_2	T_1	T_0
0	0	0	0	0	1	0	0	∞	X	X	∞	X	X	1
0	0	1	0	1	0	0	∞	β	0	∞	β	0	1	1
0	1	0	0	1	1	0	1	∞	X	X	∞	X	X	1
0	1	1	1	0	0	∞	β	β	∞	β	β	1	1	1
1	0	0	1	0	1	1	0	∞	X	X	∞	X	X	1
1	0	1	1	1	0	1	∞	β	1	∞	β	0	1	1
1	1	0	1	1	1	1	1	∞	X	X	∞	X	X	1
1	1	1	0	0	0	β	β	β	β	β	β	1	1	1
(a)			(b)			(c)			(d)			(e)		

Tabla 19.12. tablas del ejemplo 1 modificado. (a) tabla de entradas, (b) tabla de estado futuro, (c) tabla de transiciones, (d) nueva tabla de transiciones, (e) tabla de excitaciones.

Las nuevas variables de control de los FLIP FLOPS son:

$$T_0 = 1 \quad T_1 = 1 \quad T_2 = Q_1$$

En la figura 19.25 se muestra el nuevo circuito secuencial asincrónico. Todos los FLIP FLOP operan en la transición positiva, por lo tanto la conexión de la señal de reloj de los FLIP FLOPS 1 y 2, está conectada a Q_0' .

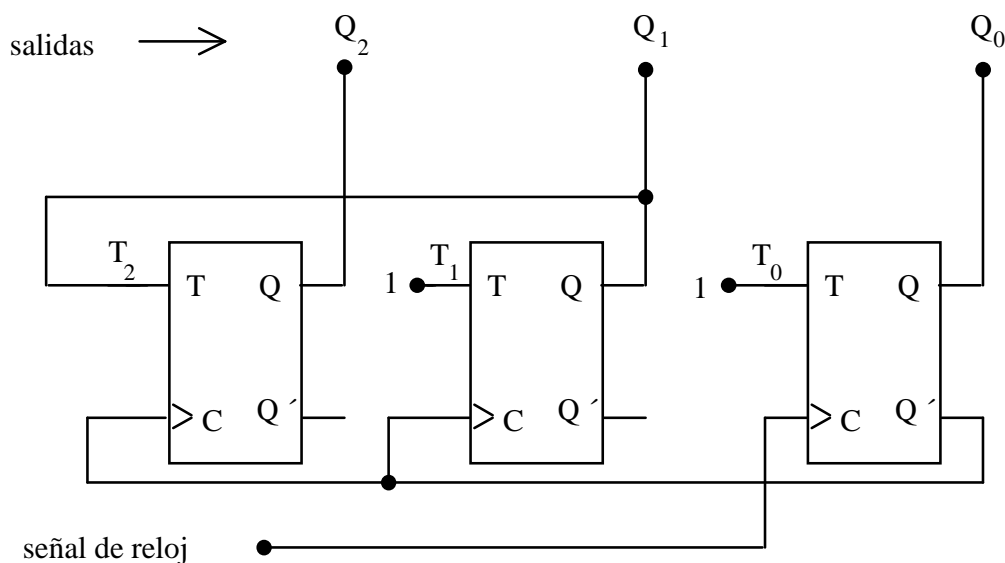


Figura 19.25. Circuito lógico del ejemplo1 modificado.

El lector podrá comprobar que este nuevo diseño tiene menos estados falsos que el anterior; por consiguiente, al momento de cambiar de estado, lo hace más rápido.

Observación: Cuando la entrada de control de un FLIP FLOP depende de la salida de otro, el valor que deben de estar presente en esta entrada al momento en que se pueda producir un cambio en el FLIP FLOP dependiente, debe de ser la correspondiente al estado actual del circuito, no la del estado futuro. Por ejemplo, en el caso anterior, cuando se puede producir un cambio en el FLIP FLOP 2, el valor que se tiene presente en T_2 es el valor Q_1 del estado actual, ya que ambos cambian en el mismo instante de tiempo. Normalmente se tiene esta situación, pero si no se tuviera, se deben introducir los retrasos necesarios para lograrlo.

Ejemplo 2.

El siguiente ejemplo es un contador binario que sigue una determinada secuencia y no usa todos los estados posibles. Una posible definición del circuito secuencial se presenta a continuación:

“Diseñe el mínimo circuito secuencial asincrónico que siga la siguiente secuencia: 0, 2, 5, 7, 4, 6, 3, 0, ...”.

.

Paso1. Definir y construir el diagrama de estado.

En la figura 19.26 se muestra la definición y el diagrama de estados para este contador. Par este ejemplo la salida del circuito será el estado del mismo.

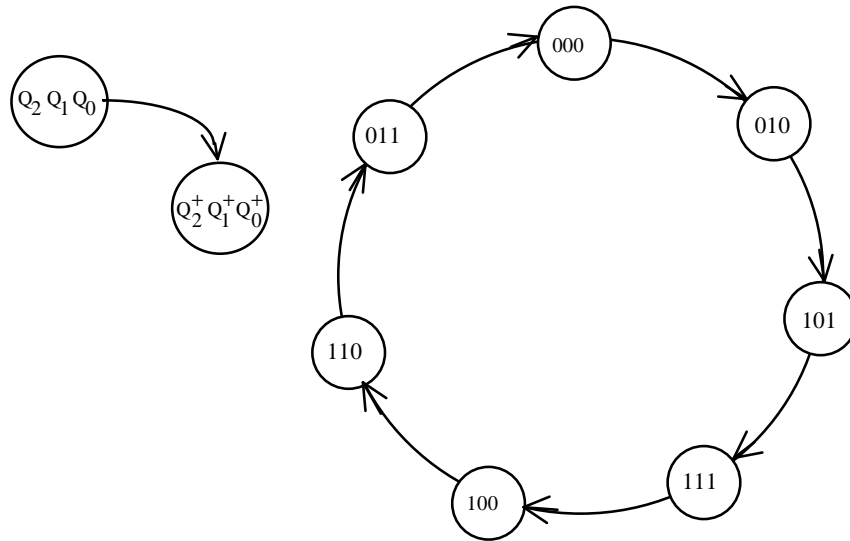


Figura 19.26. Diagrama de estado del ejemplo 2.

Paso 2. Realizar la reducción de estados.

Para el ejemplo que se está diseñando no existen estados equivalentes y no se pueden reducir.

Paso 3. Determinar el número de FLIP FLOPS requeridos.

Para el ejemplo actual, se requieren $\log_2 7 = 2.874$, se usaran 3 FLIP FLOPS.

Paso 4. Dar nombre a las variables de estado y asignar códigos a cada estado.

En este ejemplo se utilizará la definición que se da en el diagrama de estados.

Paso 5. Escoger el tipo de FLIP FLOPS a utilizar.

Se usaran FLIP FLOP J-K que operan en transición positiva.

Paso 6. Construir las tablas de entradas, estado futuro, transiciones y salida.

Las tablas para este ejemplo se muestra en la tabla 19.13. Para este ejemplo no se tendrá una tabla de salida, ya que la salida del circuito es el estado mismo.

Q_2	Q_1	Q_0	Q_2^+	Q_1^+	Q_0^+	t_2	t_1	t_0
0	0	0	0	1	0	0	∞	0
0	0	1	X	X	X	X	X	X

0	1	0	1	0	1	∞	β	∞
0	1	1	0	0	0	0	β	β
1	0	0	1	1	0	1	∞	0
1	0	1	1	1	1	1	∞	1
1	1	0	0	1	1	β	1	∞
1	1	1	1	0	0	1	β	β
(a)			(b)			(c)		

Tabla 19.13. Tablas del ejemplo 2. (a) tabla de entradas, (b) tabla de estado futuro, (c) tabla de transiciones.

Paso 7. Determinar dependencias entre los FLIP FLOP.

En este ejemplo, el FLIP FLOP 2 depende de las transiciones ∞ del FLIP FLOP 0. El FLIP FLOP 0 y 1 no puede ser controlado por otro, por lo tanto debe ser controlado por la señal de reloj del circuito.

Paso 8. Construir la nueva tabla de transiciones y la tabla de excitaciones.

Si se usan las siguientes dependencias:

FLIP FLOP 0 depende del reloj principal.

FLIP FLOP 1 depende del reloj principal.

FLIP FLOP 2 depende de las transiciones ∞ del FLIP FLOP 0.

Las nuevas tablas de transiciones y excitaciones se muestran en la tabla 19.14.

Q_2	Q_1	Q_0	Q_2^+	Q_1^+	Q_0^+	t_2	t_1	t_0	nt_2	nt_1	nt_0	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	1	0	0	∞	0	X	∞	0	X	X	1	X	0	X
0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
0	1	0	1	0	1	∞	β	∞	∞	β	∞	1	X	X	1	1	X
0	1	1	0	0	0	0	β	β	X	β	β	X	X	X	1	X	1
1	0	0	1	1	0	1	∞	0	X	∞	0	X	X	1	X	0	X
1	0	1	1	1	1	1	∞	1	X	∞	1	X	X	1	X	X	0
1	1	0	0	1	1	β	1	∞	β	1	∞	X	1	X	0	1	X
1	1	1	1	0	0	1	β	β	X	β	β	X	X	X	1	X	1
(a)			(b)			(c)			(d)			(e)					

Tabla 19.14. tablas del ejemplo 2. (a) tabla de entradas, (b) tabla de estado futuro, (c) tabla de transiciones, (d) nueva tabla de transiciones, (e) tabla de excitaciones.

Paso 9. Obtener las ecuaciones para determinar la excitación y las salidas.

Partiendo de la tabla de excitación se obtienen las expresiones mínimas para las ecuaciones de control J_2 , K_2 , J_1 , K_1 , J_0 , y K_0 .

$$\begin{array}{lll} J_2 = 1 & J_1 = 1 & J_0 = Q_1 \\ K_2 = 1 & K_1 = Q_2' + Q_0 & K_0 = Q_1 \end{array}$$

Las salidas en este caso son: Q_2 , Q_1 y Q_0 .

Paso 10. Construir el circuito.

El circuito que se obtiene se muestra en la figura 19.27. Note que el reloj del FLIP FLOP 2 se conecta a Q_0 , ya que éste trabaja en transiciones positivas y depende de las transiciones positivas del FLIP FLOP 0.

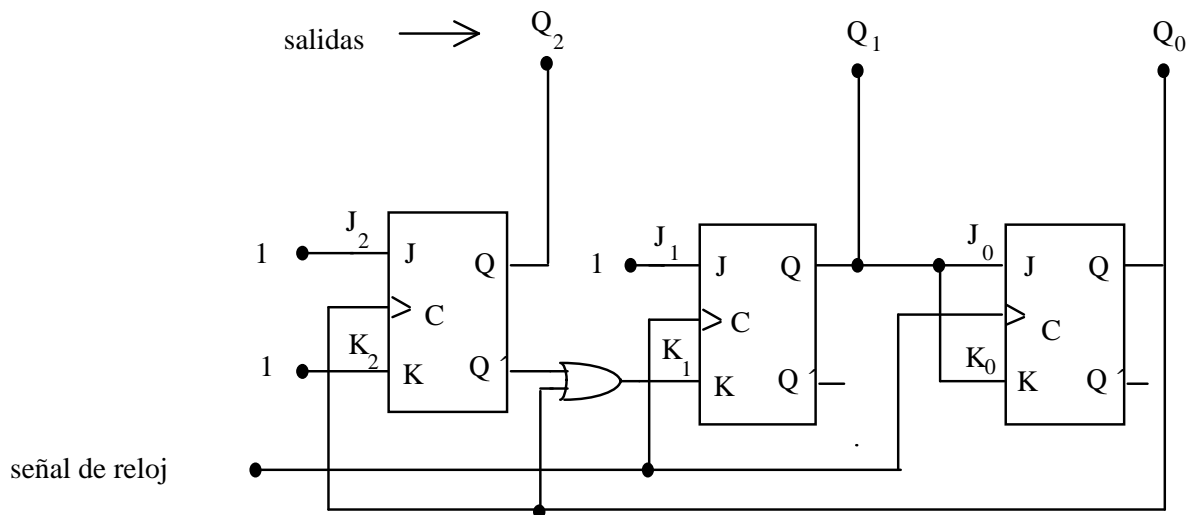


Figura 19.27. Circuito lógico del ejemplo 2.

El lector podrá comprobar que, si el circuito inicia en el estado 1 (001) al llegar la señal de reloj este pasará al estado 3 (011).

Los circuitos secuenciales asincrónicos normalmente no son utilizados en el diseño de una computadora, por tal motivo hasta aquí se dejará el análisis de éstos. No se analizará el diseño de circuitos secuenciales asincrónicos sin señal de reloj.

19.7. RESUMEN.

En este capítulo se presentaron las características principales de los circuitos lógicos secuenciales así como el tipo de aplicaciones que requieren ser construidas utilizando estos circuitos. Se

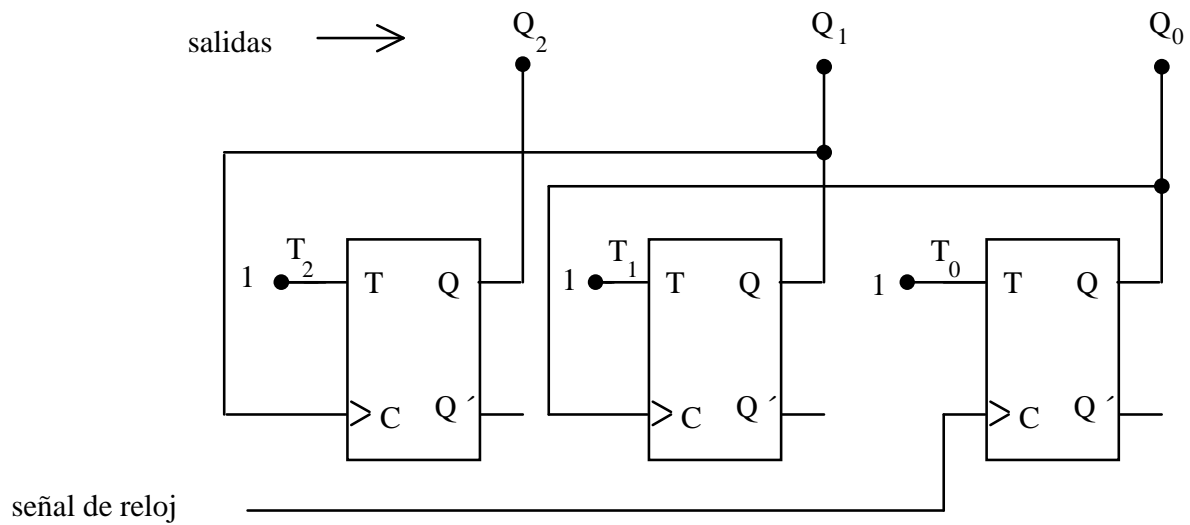
Se definieron las características de los dos tipos de circuitos secuenciales existentes, de acuerdo al momento en que cambia su estado interno, presentando un ejemplo sencillo de un circuito lógico síncrono y otro asíncrono.

Para integrar todos los conceptos anteriores, se presentó una metodología para el diseño de circuitos secuenciales sincrónicos, ejemplificándola con varios ejemplos. Se presentó también el diseño de circuitos secuenciales asíncrónicos trabajando un ejemplo sencillo de un circuito.

19.8. PROBLEMAS.

-

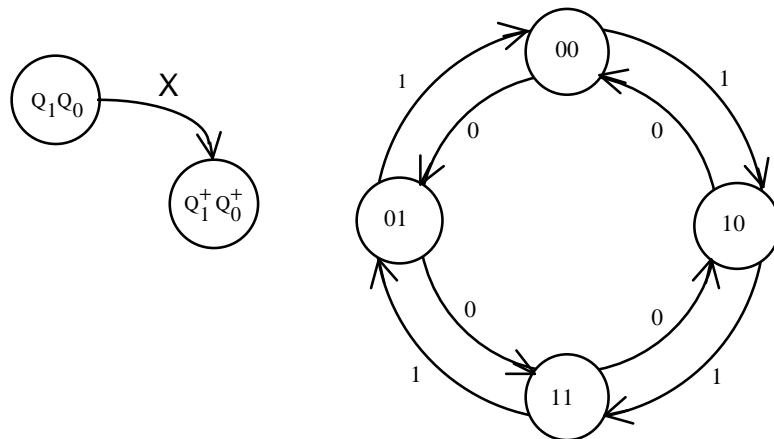
- 45



3. Diseñe y construya un circuito secuencial síncrono que siga la siguiente secuencia:
000, 010, 100, 110, 001, 011, 101, 111, 000, 010
 - a) Usando FLIP-FLOPs JK
 - b) Usando FLIP-FLOPs T
 - c) Usando FLIP-FLOPs D
 - d) Usando FLIP-FLOPs SR

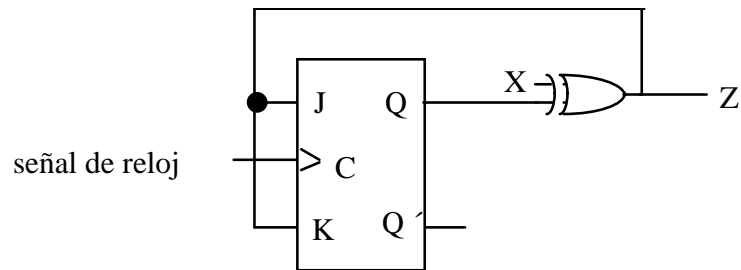
4. Diseñe y construya un circuito secuencial síncrono que siga la siguiente secuencia:
000, 011, 101, 110, 001, 111, 000, 011
Además obtenga el diagrama de estados completo.
 - a) Usando FLIP-FLOPs JK
 - b) Usando FLIP-FLOPs T
 - c) Usando FLIP-FLOPs D
 - d) Usando FLIP-FLOPs SR

5. Diseñe y construya un circuito secuencial síncrono de acuerdo al siguiente diagrama de estados.

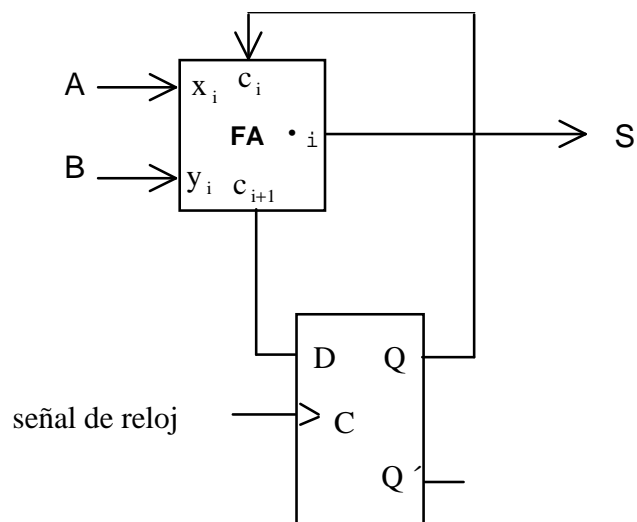


- a) Usando FLIP-FLOPs JK
 - b) Usando FLIP-FLOPs T
 - c) Usando FLIP-FLOPs D
 - d) Usando FLIP-FLOPs SR
6. Diseñe y construya un circuito secuencial sincrónico que tenga una entrada X y una salida Z y realice lo siguiente: en cada transición positiva del reloj se muestrea la señal de entrada X. La salida Z será igual a 1, sólo cuando, el número de unos o ceros que se han detectado en X, sean múltiplo de 2.
 - a) Usando FLIP-FLOPs JK
 - b) Usando FLIP-FLOPs T
 - c) Usando FLIP-FLOPs D
 - d) Usando FLIP-FLOPs SR
 7. Diseñe un diagrama que realice lo siguiente: en cada transición positiva del reloj se muestrea la señal de entrada X y, dependiendo de los valores que se muestreen, genere una salida Z la cual será igual a 1, sólo cuando, el número de unos que se hayan muestreado sea múltiplo de 3 y el número de ceros que se han detectado en X, sean múltiplo de 2.
 8. Diseñe un diagrama que realice lo siguiente: en cada transición positiva del reloj se muestrea la señal de entrada X y, dependiendo de los valores que se muestreen, genere una salida Z la cual será igual a 1, sólo cuando en los tres últimos muestreos la cantidad de uno sea mayor o igual a dos.
 9. Diseñe un diagrama que realice lo siguiente: en cada transición positiva del reloj se muestrea la señal de entrada X y, dependiendo de los valores que se muestreen, genere una salida Z la cual será igual a 1, sólo cuando en los tres últimos muestreos la cantidad de uno sea mayor o igual a dos y la cantidad de ceros que se han muestreado durante todo el tiempo es múltiplo de dos.

10. Obtenga el diagrama de estados para el siguiente circuito (modelo Mealy). X es una entrada externa y Z es una salida del circuito.



11. Obtenga el diagrama de estados para el siguiente circuito (modelo Mealy). A y B son entradas externas y S es la salida del circuito.



12. Diseñe y construya un circuito secuencial asincrónico que tenga la siguiente secuencia: 000, 111, 101, 110, 100, 011, 001, 010, 000, 111

Además, obtenga el diagrama de estados donde se indiquen los estados falsos.

- a) Usando FLIP-FLOPs JK
- b) Usando FLIP-FLOPs T
- c) Usando FLIP-FLOPs D
- d) Usando FLIP-FLOPs SR

CAPITULO 20

DISEÑO DE REGISTROS.

20.1. REGISTROS

En el capítulo 2 se presentó la arquitectura de una computadora simple y, dentro de la unidad central de proceso, se tenían los siguientes registros: el registro acumulador (**AC**), el registro contador de programa (**PC**), el registro de instrucción (**IR**), el registro de dirección de memoria (**MAR**), el registro de datos de memoria (**MDR**) y el registro de banderas (**FR**).

La función de los registros es guardar datos, instrucciones o banderas en forma temporal para que la unidad central de proceso realice sus funciones. El tamaño de los registros depende del propósito de cada uno de ellos. Por ejemplo, el registro de datos de memoria generalmente es del mismo tamaño que una celda o palabra de memoria ya que a través de él se guardarán datos en memoria y se recuperarán los datos almacenados en la unidad de memoria.

El registro contador de programa debe ser de un tamaño tal que permita contener en él cualquier dirección de memoria de la computadora ya que es utilizado para determinar la dirección de donde se traerá la siguiente instrucción, que se encuentra en la unidad de memoria, al registro de instrucción.

De forma similar, el registro de dirección de memoria también debe ser del mismo tamaño que el registro contador de programa ya que debe contener la dirección de memoria de donde se leerá un dato o en la cual se guardará un dato en la unidad de memoria.

En los ejemplos mencionados anteriormente la información se carga en los registros en forma paralela, es decir, todos los bits se almacenan en forma simultánea. La obtención de la información también se realiza en paralelo.

Cuando se transmite información de una computadora a una red de comunicaciones, se envían los bits uno tras otro en forma serial hacia la red. Para estos casos es conveniente tener registros en los cuales la información se carga en paralelo pero la obtención de la información se realiza en serie. Por otra parte, si una computadora está obteniendo información de una red de comunicaciones, recibirá los bits en forma serial y, una vez recibidos en un registro, se obtendrán en forma paralela. En este último caso, es conveniente tener registros en los cuales se cargue la información en serie y se pueda obtener en paralelo.

Finalmente, puede ocurrir que se necesiten registros en los cuales tanto la carga de la información como la obtención de la misma se realice en forma serial. Este sería el caso de un registro intermedio utilizado en una red de comunicaciones.

Para construir los registros se pueden utilizar cualquiera de las celdas biestables (FLIP FLOPS) descritas en el capítulo 17, donde cada celda almacena un bit de la información contenida en el registro. La forma de interconectar las celdas biestables para construir un registro depende de la

función del registro y de la forma en que se desee guardar la información en él y la forma en que se obtenga la información guardada en el mismo.

20.2 REGISTROS DE CARGA Y SALIDA EN PARALELO.

En la figura 20.1 se muestra un registro de 3 bits construido con FLIP FLOPS D en el cual la carga de la información se realiza en paralelo en forma sincrónica; es decir, cada una de los tres bits que conforman la información que se desea guardar se almacenan simultáneamente en el registro cuando se presenta el pulso de reloj y la señal de carga (load) está en uno lógico. La obtención de la información almacenada en el registro también se realiza en paralelo.

La entrada de información al registro se realiza a través de las líneas E_2 , E_1 y E_0 . La salida se obtiene de las líneas S_2 , S_1 y S_0 . La operación del registro se describe a continuación solamente para uno de los bits del mismo. Los demás bits se comportan de la manera similar.

Sea Q_i el bit i del registro y sea L la línea de carga (load) del registro. La ecuación para el estado futuro del bit i , después de presentarse el pulso de reloj, está dada por:

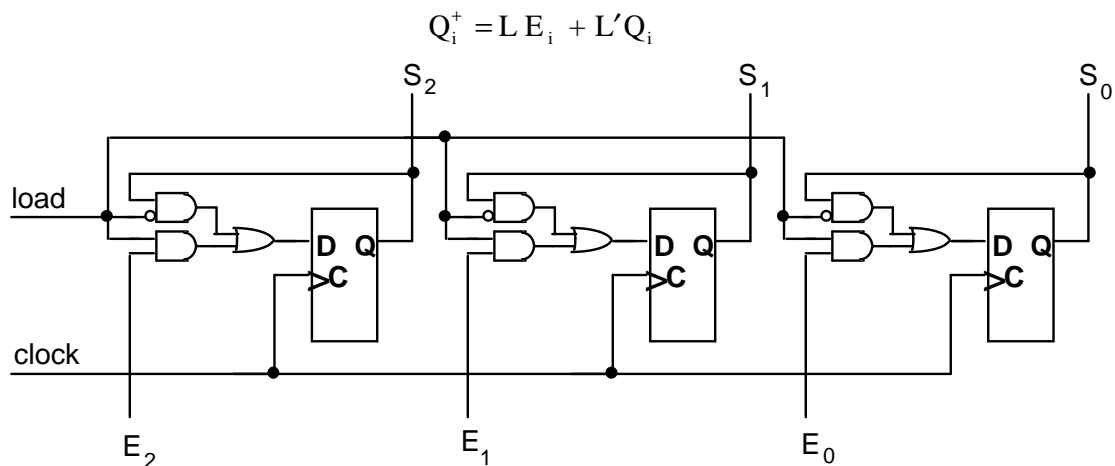


Figura 20.1. Registro de tres bits con carga sincrónica en paralelo y salida en paralelo.

La tabla de verdad para el estado futuro del FLIP FLOP i , en función de su estado actual Q_i , la línea de carga L y la entrada E_i se muestra en la tabla 20.1.

L	E_i	Q_i	Q_i^+
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0

1	1	0	1
1	1	1	1

Tabla 20.1. Tabla de verdad para el bit i del registro de la figura 20.1.

Obsérvese que cuando la línea de carga L vale cero, el estado futuro del bit i es igual a su estado actual. Cuando la línea de carga L vale uno, el estado futuro del bit i es igual a la entrada correspondiente E_i independientemente de cual sea su estado actual.

De acuerdo a la operación descrita anteriormente, se pueden almacenar tres bits de información en este registro poniendo la información deseada en las entradas y poniendo un uno la línea de carga. Cuando se presente el pulso de reloj se almacenará esta información en las celdas biestables y estará disponible en las salidas del registro.

Extendiendo la configuración mostrada en la figura 20.1 se pueden construir registro con los bits necesarios para almacenar información.

En la figura 20.2 se muestra un registro de tres bits con carga en paralelo asincrónica y salida en paralelo. En este caso no se utiliza señal de reloj para realizar la carga del registro, sino que se usa la línea de carga directamente para hacerlo. La construcción del registro está hecha con FLIP FLOPS D con entradas Clear (CL) y Preset (PR) activadas por el nivel lógico cero.

Recuérdese que en este tipo de celdas biestables cuando tanto la entrada Clear como la entrada Preset valen uno, la celda conserva su estado. Si la entrada Clear vale cero y la entrada Preset vale uno, la celda se cambiará al estado cero, independientemente del estado en que se encontraba. Si la entrada Clear vale uno y la entrada Preset vale cero, la celda cambiará al estado uno, independientemente del estado en que se encuentre. Finalmente, cuando ambas entradas valen cero, no se sabe en cual estado quedará la celda.

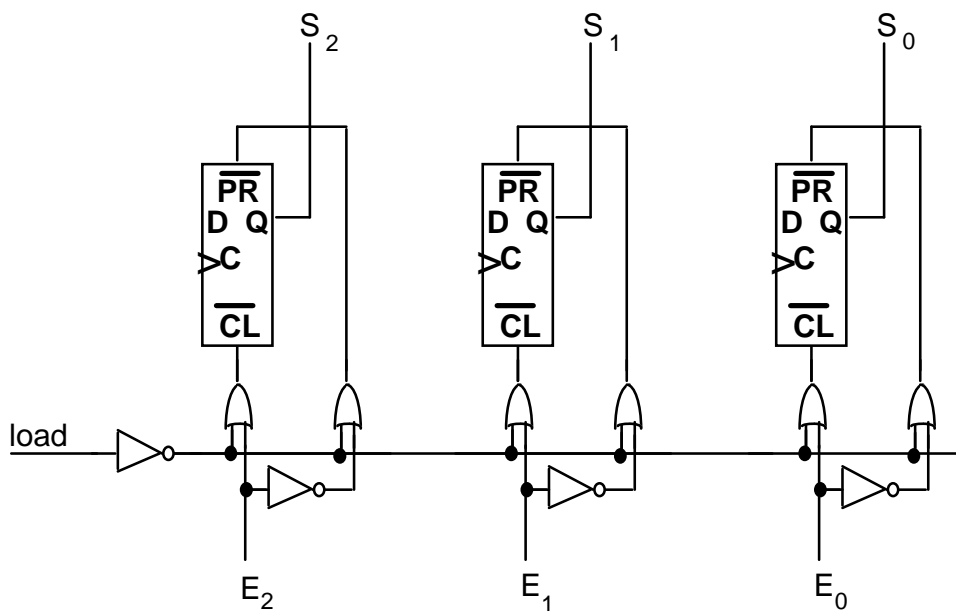


Figura 20.2. Registro de tres bits con carga asincrónica en paralelo y salida en paralelo.

Para analizar la operación de la celda obsérvese que cuando la línea de carga (load) vale cero, las entradas Clear y Preset valen uno; por lo tanto, las celdas conservarán su estado. Cuando la línea de carga vale uno, la entrada Clear valdrá cero solamente si la correspondiente entrada E_i vale cero y la entrada Preset valdrá uno, haciendo que se almacene un cero en la celda. Si la línea de carga vale uno pero la entrada E_i vale también uno, la entrada de la celda que tomará el valor de cero será Preset mientras que la entrada Clear valdrá uno, haciendo que se almacene el valor uno en la celda.

20.3. REGISTROS DE CARGA EN PARALELO Y SALIDA EN SERIE.

Un registro de tres bits cuya carga se realiza en paralelo en forma asincrónica pero su salida se obtiene en serie en forma sincrónica se muestra en la figura 20.3.

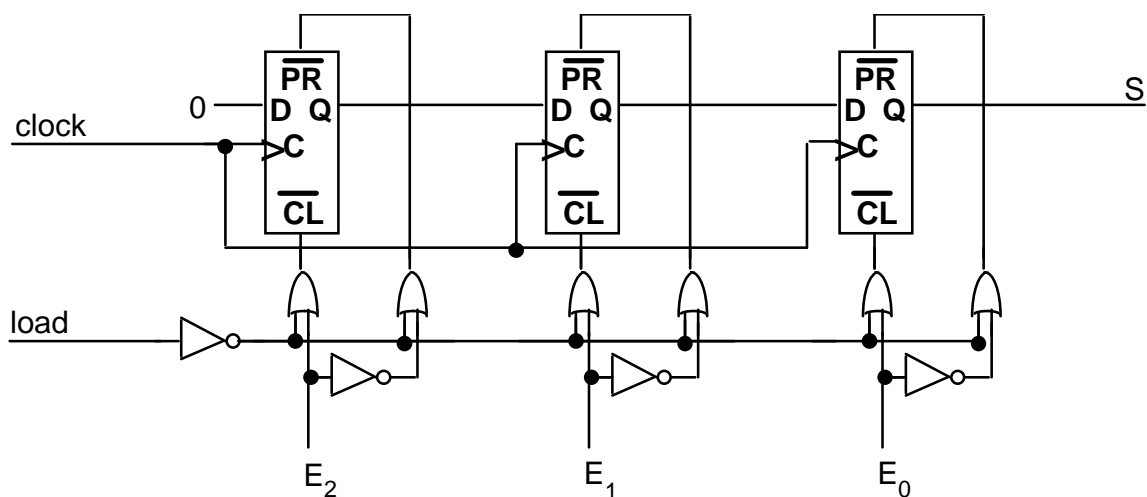


Figura 20.3. Registro de tres bits con carga asincrónica en paralelo y salida sincrónica en serie.

Analizando el diagrama de este registro se puede ver que la señal de carga (load) hace que el registro pueda ser cargado en paralelo en forma asincrónica desde las entradas E_2 , E_1 y E_0 , tal como ocurre con el segundo registro presentado en la sección anterior. Las entradas Clear y Preset cambian de estado a las celdas biestables sin importar cuanto valga la señal de reloj, por esta razón se denominan entradas asincrónicas.

Cuando la señal de carga está en cero lógico y se alimenta la señal de reloj, la información que está en las celdas biestables se recorre en el registro hacia la derecha con cada pulso de reloj. El primer pulso de reloj hace que el bit que está en la celda del extremo izquierdo pase a la celda central al mismo tiempo que el bit de esta celda pasa a la celda de la derecha y el cero que está en

la entrada D de la celda de la izquierda se almacene en ella. El siguiente pulso de reloj hará exactamente lo mismo con la información que ahora contienen las celdas. La información que queda en este caso después del tercer pulso de reloj será 000, ya que se está alimentando un cero en la entrada D de la celda de la izquierda. Si se alimentara siempre un uno lógico en esta entrada, quedaría la información 111 después del tercer pulso de reloj.

Si se denomina a la información contenida en los tres bits del registro como $b_2 b_1 b_0$, el primer pulso de reloj hace que la información contenida en el registro sea $0b_2b_1$. El siguiente pulso de reloj deja la información como $00b_2$ y un tercer pulso de reloj deja la información 000 en el registro. Si se analiza lo que se encuentra en la salida serial S del registro, se verá que inicialmente estaba el bit b_0 . Después del primer pulso de reloj estaba b_1 ; mientras que después del segundo pulso de reloj el bit que se encontraba es la salida era b_2 . Estos bits representan la salida serial del registro.

Para un registro de n bits se necesitarán $n-1$ pulsos de reloj para poder obtener los n bits del registro en la salida S, uno antes de cada pulso de reloj. Inicialmente se presenta el bit menos significativo del registro en la salida S y, en cada pulso de reloj aparece en esta salida el siguiente bit menos significativo. Un último pulso de reloj dejará completamente borrada la información en el registro, es decir, quedará con todos sus bits en cero.

Si se desea que la información contenida en el registro permanezca sin alterarse, se debe dejar de alimentar la señal de reloj al registro. Esto se puede hacer mediante una compuerta AND a la cual se alimente la señal de reloj y otra señal que controle cuando se deja pasar la señal de reloj y cuando se interrumpe el paso de esta señal.

20.4. REGISTROS DE CARGA EN SERIE Y SALIDA EN PARALELO.

Se analizará ahora un registro cuya carga se realiza en forma sincrónica serial pero su salida es obtenida en forma paralela. La información es almacenada en el registro mediante la entrada E, un bit a la vez, con cada pulso de reloj. Una configuración simple para este registro se muestra en la figura 20.4. La información contenida en el registro está disponible en paralelo en las salidas S_2 , S_1 y S_0 .

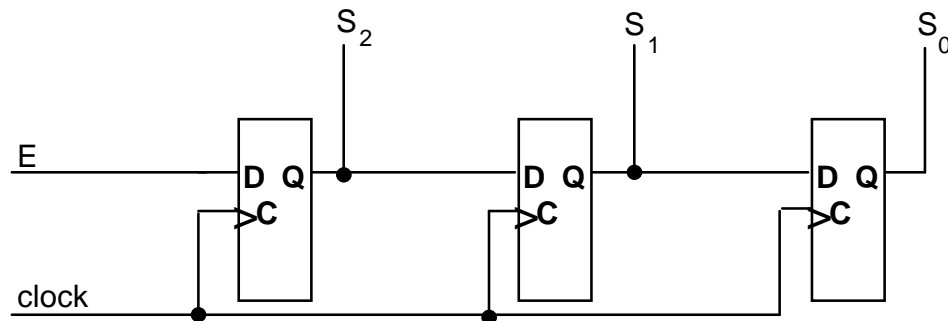


Figura 20.4. Registro de tres bits con carga sincrónica en serie y salida en paralelo.

Observando el circuito para este registro, se puede ver que si la señal de reloj no se está alimentando, la información que está en el registro permanece sin alterarse. Cuando se alimenta la señal de reloj, el primer pulso en esta entrada hará que la información presente en ese momento en la entrada E se almacene en la celda biestable de la izquierda. El siguiente pulso de reloj ocasionará que el bit que esté en la celda de la izquierda se almacene en la celda biestable del centro al mismo tiempo que se almacena en la celda biestable de la izquierda el bit que se encuentre presente en la entrada E. Por último, un tercer pulso hará que simultáneamente se almacene en la celda biestable de la derecha el bit que se encuentre en la celda central, el bit que está en la celda biestable de la izquierda se almacene en la celda central y el bit que esté presente en la entrada E se almacene en la celda izquierda. Esto termina la carga en serie para este registro. La salida está disponible en paralelo.

Si se construye un registro de este tipo con n bits, serán necesarios n pulsos de reloj para realizar la carga de la información (compuesta de n bits) alimentando un bit en cada pulso de reloj iniciando con el menos significativo y terminando con el más significativo.

Una configuración alterna para un registro que se cargue sincrónicamente en serie y su salida pueda obtenerse en paralelo se muestra en la figura 20.5. Nótese que en esta configuración la información permanece sin modificarse cuando la señal de carga (load) está en el nivel lógico cero aún cuando la señal de reloj se esté alimentando.

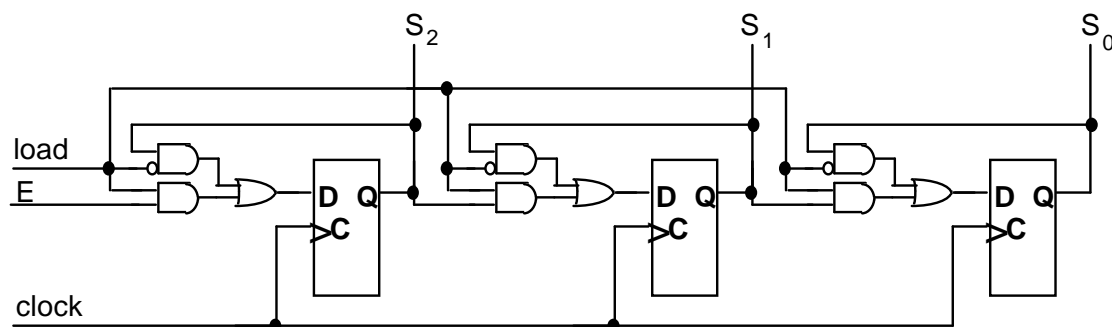


Figura 20.5. Configuración alterna para un registro de tres bits con carga sincrónica en serie y salida en paralelo.

Si se activa la señal de carga, el primer pulso de reloj hace que el bit presente en ese momento en la entrada E se almacene en la celda biestable de la izquierda. El siguiente pulso de reloj ocasiona que el bit que se almacenó en la celda izquierda sea ahora almacenado en la celda del centro al mismo tiempo que el bit que ahora esté presente en la entrada E sea almacenado en la celda izquierda. Un tercer pulso de reloj hará que simultáneamente se almacene el bit de la celda del centro en la celda derecha, el bit de la celda izquierda en la celda central y, el bit presente en la entrada E en la celda izquierda. Se supone que antes de cada pulso de reloj debe estar presente en la entrada E el correspondiente bit de la información que desea guardarse en el registro, empezando con el bit menos significativo y terminando con el más significativo.

Si se quiere construir un registro de n bits, basta extender la configuración mostrada para que el registro contenga n celdas biestables. En este caso serán necesarios n pulsos de reloj para almacenar un dato de n bits en el registro.

20.5. REGISTROS DE CARGA Y SALIDA EN SERIE.

La configuración más simple para un registro con carga y salida sincrónicas en serie se presenta en la figura 20.6. Esta configuración utiliza FLIP FLOPS del tipo D.

Cuando la señal de reloj no está presente, la información que se encuentra almacenada en el registro permanece sin modificarse y el bit menos significativo está presente en la salida S. Al alimentarse la señal de reloj, la información contenida en el registro se recorre una posición a la derecha, perdiéndose el bit menos significativo y almacenándose en el bit más significativo el valor que esté presente en la entrada E.

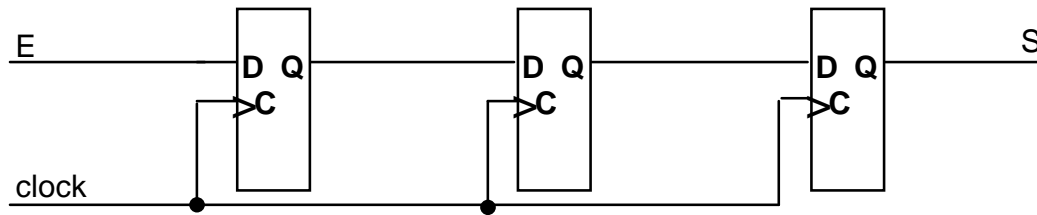


Figura 20.6. Configuración simple para un registro con carga y salida sincrónicas en serie.

En este registro, siempre que la señal de reloj esté activa, se está alimentando al registro el bit que esté presente en la entrada E y se está desechando el bit menos significativo del registro, recorriéndose todos los demás bits una posición a la derecha.

Una configuración más elaborada permite construir un registro que puede cargarse en forma sincrónica en serie o en forma asincrónica en paralelo y, la salida puede ser obtenida en paralelo o en forma sincrónica en serial. Este registro se muestra en la figura 20.7.

Si la señal de carga (load) está en cero lógico, los valores presentes en las entradas asincrónicas Clear y Preset de las celda biestables son todos uno y la operación de los FLIP FLOPS está controlada por la señal de reloj y los valores presentes en cada una de las entradas D. El lector podrá verificar que en este caso el registro opera exactamente igual al registro básico de la figura 20.6.

Si se activa la señal de carga (load), la operación del registro es igual a la del registro de carga asincrónica en paralelo presentado en la figura 20.2 ya que las entradas asincrónicas Clear y Preset operan sin importar lo que esté presente en la señal de reloj.

El valor almacenado en el registro permanece sin alterarse cuando la señal de carga está en cero lógico y la señal de reloj no se está alimentando al registro.

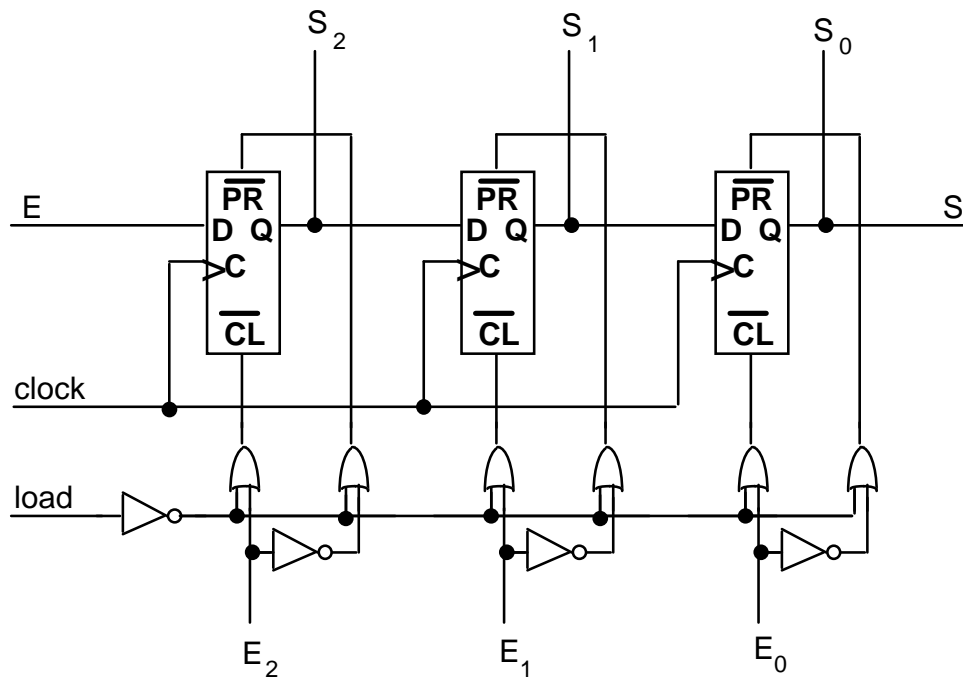


Figura 20.7. Registro con carga sincrónica en serie y asincrónica en paralelo, y salida sincrónica en serie y salida en paralelo.

Este registro es muy versátil ya que, en principio, puede operar como cualquiera de los registros descritos anteriormente. Existen registros que pueden realizar aún más operaciones que éste, algunos de ellos se presentarán en las secciones siguientes.

20.6. REGISTROS DE CORRIMIENTOS (Shift Registers).

Los registros de corrimientos permiten realizar varias funciones. Una de estas operaciones básicas es dividir un número entero entre una potencia de dos. Si se tiene un número entero sin signo en binario y se desea dividir entre 2, basta recorrer el punto binario un lugar hacia la izquierda y, si se desea conservar solamente la parte entera, se desecha el bit que está hacia la derecha del punto binario.

Supóngase que se tiene un número binario entero en un registro y se desea dividirlo entre dos. Bastará hacer un corrimiento de los bits del número un lugar hacia la derecha, desechando el bit menos significativo y alimentando un cero en el bit más significativo (el del extremo izquierdo). Por ejemplo, si se tiene un registro de 8 bits que contiene el número binario 01101100 (que representa el número 108 en decimal) y se le hace la operación descrita anteriormente, se obtendrá el número binario 00110110 en el registro (que representa el número 54 en decimal). Si se desea dividir el número original entre cuatro, será necesario recorrerlo dos veces hacia la derecha, obteniéndose el número binario 00011011 (que representa el número 27 en decimal).

El registro mostrado en la figura 20.3 puede realizar perfectamente la operación descrita anteriormente.

Supóngase ahora que lo que contiene el registro es un número binario con signo en la representación de complementos a la base. Si se desea dividir el número entre dos y éste es positivo, basta realizar la operación anterior; sin embargo, si el número es negativo, debe alimentarse un uno en la entrada D del FLIP FLOP que contiene el bit de signo para que la operación sea correcta. Como ejemplo considérese un registro de 8 bits que contiene el número -108 en representación de complementos a la base, que corresponde a 10010100. Al realizar la operación indicada se obtendrá el número binario 11001010, que representa el número decimal -54 en este formato.

Cuando se alimenta siempre un cero en el bit más significativo al realizar un corrimiento hacia la derecha en un registro, se dice que se realiza un corrimiento lógico. Si lo que se alimenta en el bit más significativo es el signo del número binario que contiene el registro, ya sea 0 (positivo) o 1 (negativo), se efectúa un corrimiento aritmético hacia la derecha en el registro.

El diseño de un registro que pueda realizar corrimientos lógicos o aritméticos hacia la derecha puede hacerse modificando ligeramente el registro de la figura 20.3. La carga del registro se hace en paralelo en forma asincrónica y su salida también es en paralelo. El diseño de este registro se muestra en la figura 20.8.

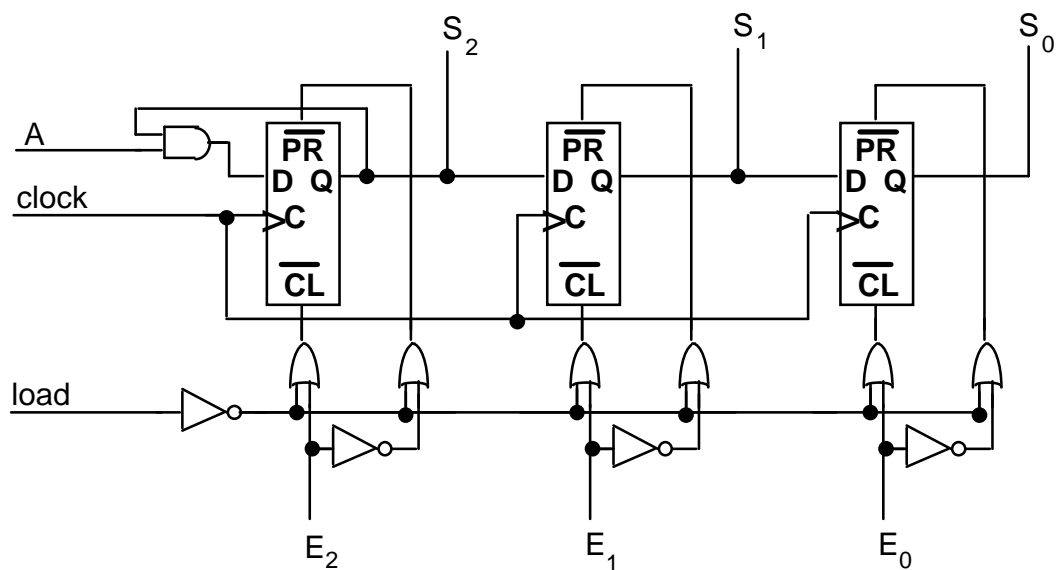


Figura 20.8. Registro de corrimientos con carga asincrónica en paralelo y salida en paralelo.

La entrada A determina si se realiza un corrimiento lógico ($A=0$), o aritmético ($A=1$). Cuando A vale cero, la entrada D del FLIP FLOP de la izquierda siempre vale cero y, en cada corrimiento hacia la derecha, se alimenta un cero en el bit más significativo. Cuando la entrada A vale uno, la salida Q del FLIP FLOP de la izquierda se retroalimenta hacia su entrada D, lo cual hace que el signo del número binario se propague al recorrerlo hacia la derecha.

También es útil en ciertas circunstancias realizar un corrimiento circular en un registro. Al realizar un corrimiento circular a la derecha, el bit menos significativo no se desecha, pues se alimenta en el bit más significativo. Si el registro es de n bits y se efectúan n corrimientos circulares, se obtendrá de nuevo el valor original en el registro.

En la figura 20.9 se muestra una modificación del registro de corrimientos de la figura 20.8 al cual se le ha añadido otra entrada de control R que indica si se realiza un corrimiento simple (shift) cuando $R=0$, o un corrimiento circular (rotate) cuando $R=1$. Cuando se efectúa un corrimiento simple, la entrada de control A determina si el corrimiento es aritmético ($A=1$) o lógico ($A=0$).

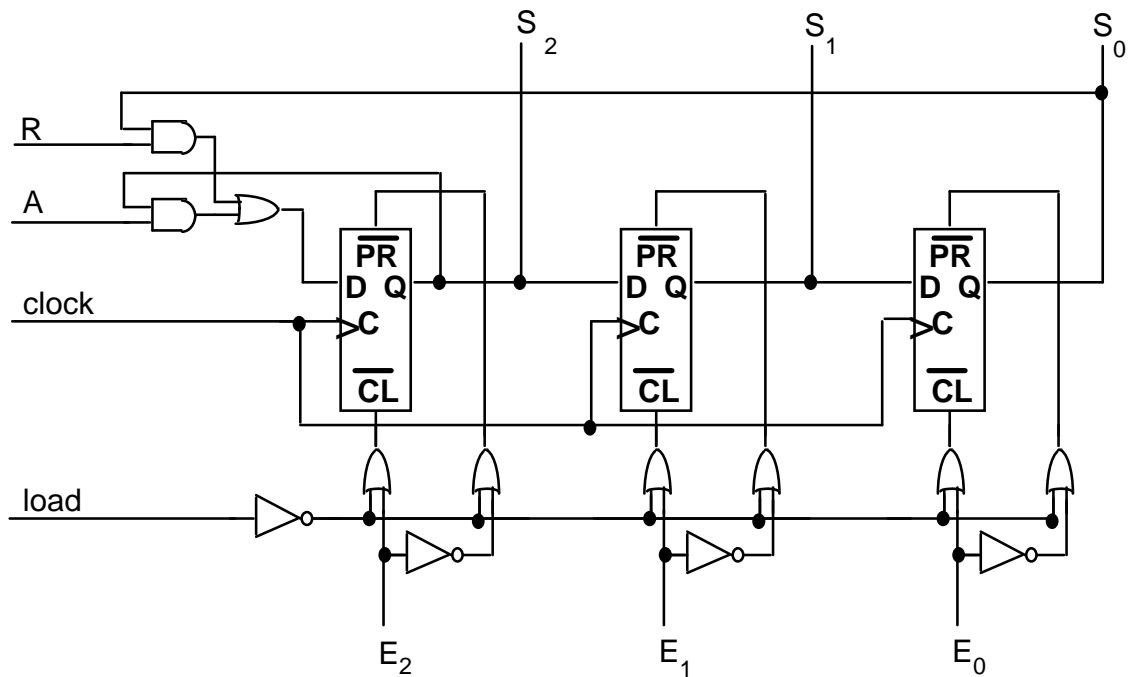


Figura 20.9. Registro de corrimientos con carga asincrónica en paralelo y salida en paralelo, que puede realizar corrimientos circulares.

Lo que no ha sido previsto en el diseño del registro anterior es la posibilidad de que tanto la entrada de control A como la R valgan uno simultáneamente. Se supone que esta condición no debe presentarse ya que la operación del registro bajo esta situación no es correcta.

Hasta ahora se ha hablado solamente de corrimientos hacia la derecha y se han mencionado los corrimientos lógicos y aritméticos así como los circulares. En el caso de corrimientos hacia la izquierda solamente se tienen los corrimientos lógicos y los circulares ya que al no existir ningún signo que propagar, no se presentan los corrimientos aritméticos. Al recorrer un registro hacia la izquierda, el valor del número binario que contiene se multiplica por dos en cada corrimiento.

Existen registros que pueden realizar corrimientos hacia la derecha o hacia la izquierda, dependiendo de valor de una entrada de control. Otra entrada de control indica si el corrimiento es circular o simple y, en el caso de corrimientos simples hacia la derecha, una tercera entrada de control determinaría si el corrimiento se lógico o aritmético. El diseño de estos registros es muy similar a los presentados en esta sección.

20.7. REGISTROS CONTADORES (Counters).

Se presentará ahora un registro que pueda ser cargado en forma asincrónica en paralelo, que tenga salida en paralelo, y que además permita incrementar su contenido en forma sincrónica, es decir, mediante pulsos en la entrada de reloj. Probablemente el lector se esté preguntando sobre la utilidad de un registro con estas características. Recordando lo que sucede durante el ciclo de

El registro contador de programa debe tener salida en paralelo para poder transferirlo al registro de dirección de memoria y debe poder incrementarse en una unidad cada vez que se realiza la búsqueda de una instrucción. Además, al realizar una transferencia incondicional a otra dirección de memoria, debe cargarse la dirección efectiva en el registro contador de programa. Para poder hacer esto, se debe poder cargar en paralelo.

[illegible]

Los registros presentados en secciones anteriores tenían diseños más simples que este último registro y no era difícil construirlos, pero el diseño de este registro contador no se ve tan obvio y probablemente el lector se estará preguntando de dónde salió el circuito de la figura 20.10.

Para el diseño de este registro se puede aprovechar la carga asincrónica en paralelo del registro de la figura 20.2 y utilizar los conocimientos sobre diseño de circuitos secuenciales sincrónicos del capítulo 19 para diseñar la parte sincrónica, que simplemente es un contador binario hacia arriba de 0 a 7.

Para diseñar un registro de tres bits que cuente hacia arriba se utilizará el modelo de Moore, ya que las salidas (S_2 , S_1 y S_0) dependen en este caso solamente del estado. La única entrada es la señal de reloj. Se necesitarán 8 estados para contar de 0 a 7, éstos se denominarán de acuerdo al número del estado en binario (000, 001, 010, 011, 100, 101, 110 y 111). La salida del circuito corresponderá el estado y se utilizarán FLIP FLOPS J-K para este diseño. El diagrama de estados se muestra en la figura 21.11. En este caso no es posible realizar ninguna simplificación de estados y se procederá a derivar las ecuaciones para la excitación del circuito secuencial usando la tabla acostumbrada. Se asignarán las variables Q_2 , Q_1 y Q_0 para los FLIP FLOPS del circuito. La tabla para este circuito se presenta en la tabla 20.2.

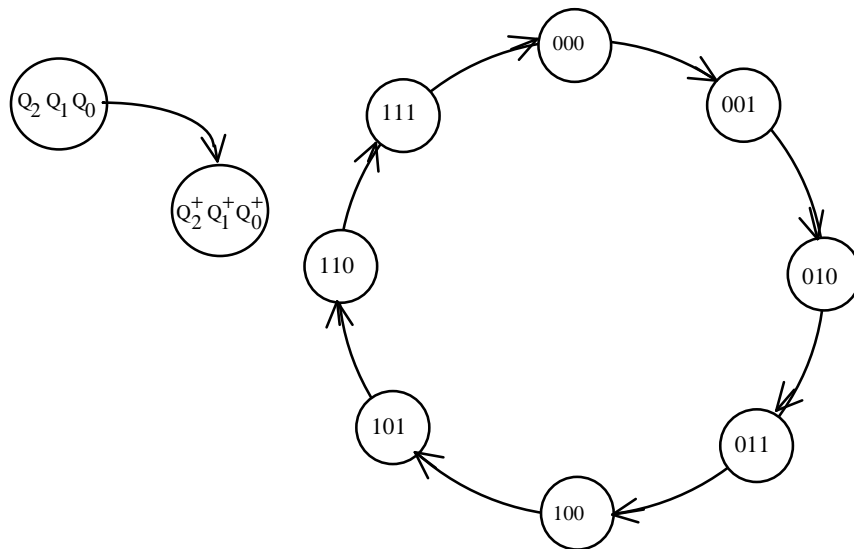


Figura 20.11. Diagrama del estados de un registro contador hacia arriba de tres bits.

Q_2	Q_1	Q_0	Q_2^+	Q_1^+	Q_0^+	t_2	t_1	t_0	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	0	1	0	0	∞	0	X	0	X	1	X
0	0	1	0	1	0	0	∞	β	0	X	1	X	X	1
0	1	0	0	1	1	0	1	∞	0	X	X	0	1	X
0	1	1	1	0	0	∞	β	β	1	X	X	1	X	1
1	0	0	1	0	1	1	0	∞	X	0	0	X	1	X
1	0	1	1	1	0	1	∞	β	X	0	1	X	X	1
1	1	0	1	1	1	1	1	∞	X	0	X	0	1	X
1	1	1	0	0	0	β	β	β	X	1	X	1	X	1

(a) (b) (c) (d)

Tabla 20.2. Tablas del contador de 0 a 7. (a) tabla de entradas, (b) tabla de estado futuro, (c) tabla de transiciones, (d) tabla de excitaciones

En la figura 20.12 se muestran los mapas de Karnaugh usados para obtener las ecuaciones de excitación.

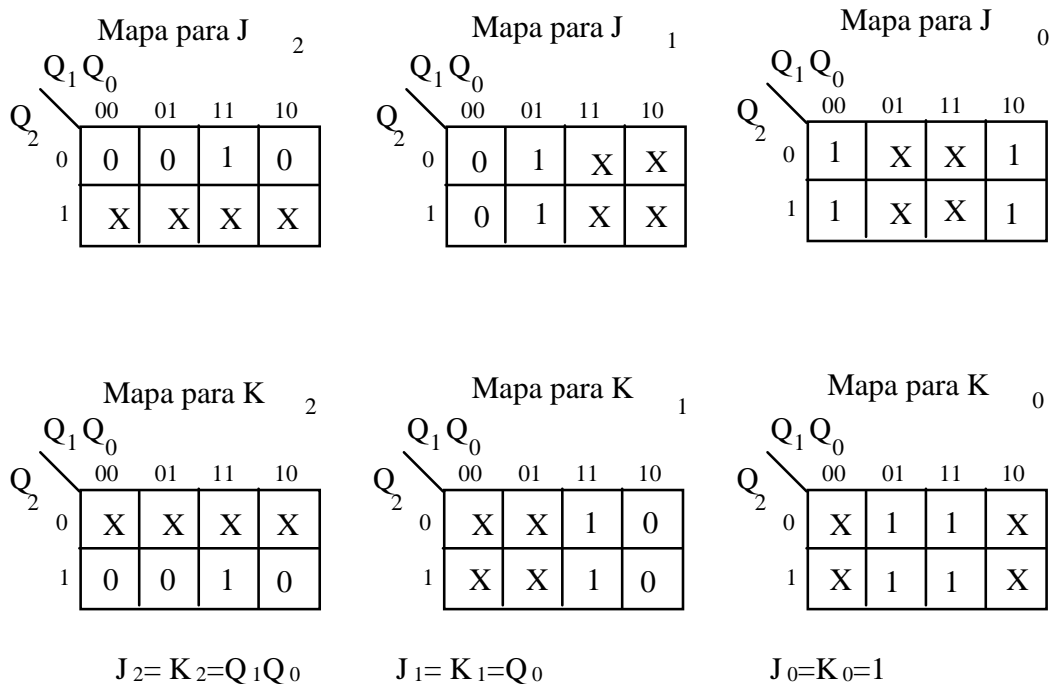


Figura 20.12. Mapas de Karnaugh para la excitación.

Las salidas (S_2 , S_1 y S_0) están dadas directamente por la salida de cada uno de los FLIP FLOPS que forman el estado del circuito secuencial (Q_2 , Q_1 y Q_0).

En este caso el uso de FLIP FLOPS J-K simplifica mucho el diseño. Pudieran haberse utilizado FLIP FLOPS del tipo D como en los casos anteriores pero el diseño del circuito lógico que alimenta la entrada D de estos FLIP FLOPS sería mas complejo.

El circuito de la figura 20.10 está construido precisamente con las ecuaciones obtenidas en este diseño del circuito secuencial para el contador binario ascendente de 0 a 7. En el capítulo 19 se diseñó un contador similar pero se usaron FLIP FLOPS T. El incremento del valor contenido en el registro se realiza cada vez que se le da un pulso en la entrada de reloj.

También es útil tener registros que permitan decrementar su contenido en forma sincrónica y que tengan carga asincrónica y salida en paralelo. La construcción de un registro con estas características es muy similar al registro anterior ya que la única diferencia es que ahora su

contenido se decrementa en una unidad con cada pulso de reloj en lugar de incrementarse. En la figura 20.13 se presenta un posible diseño para este registro en el cual se han utilizado FLIP FLOPS J-K con entradas Clear y Preset asincrónicas. Se deja como ejercicio para el lector verificar las ecuaciones para la excitación.

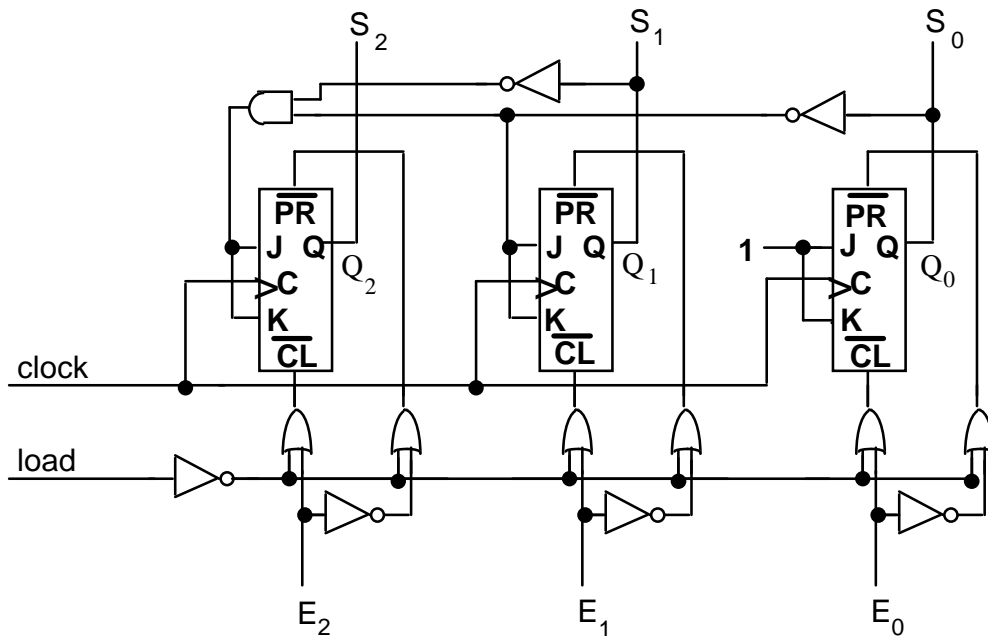


Figura 20.13. Registro con carga asincrónica en paralelo y salida en paralelo, que puede ser decrementado sincrónicamente.

Los dos diseños anteriores pueden ser combinados en uno solo que incluya una entrada de control U , cuya función sea indicar si el contenido del registro será incrementado ($U=1$) o decrementado ($U=0$) cuando se presenta el pulso de reloj. Este diseño se presenta en la figura 20.14 para un registro de tres bits.

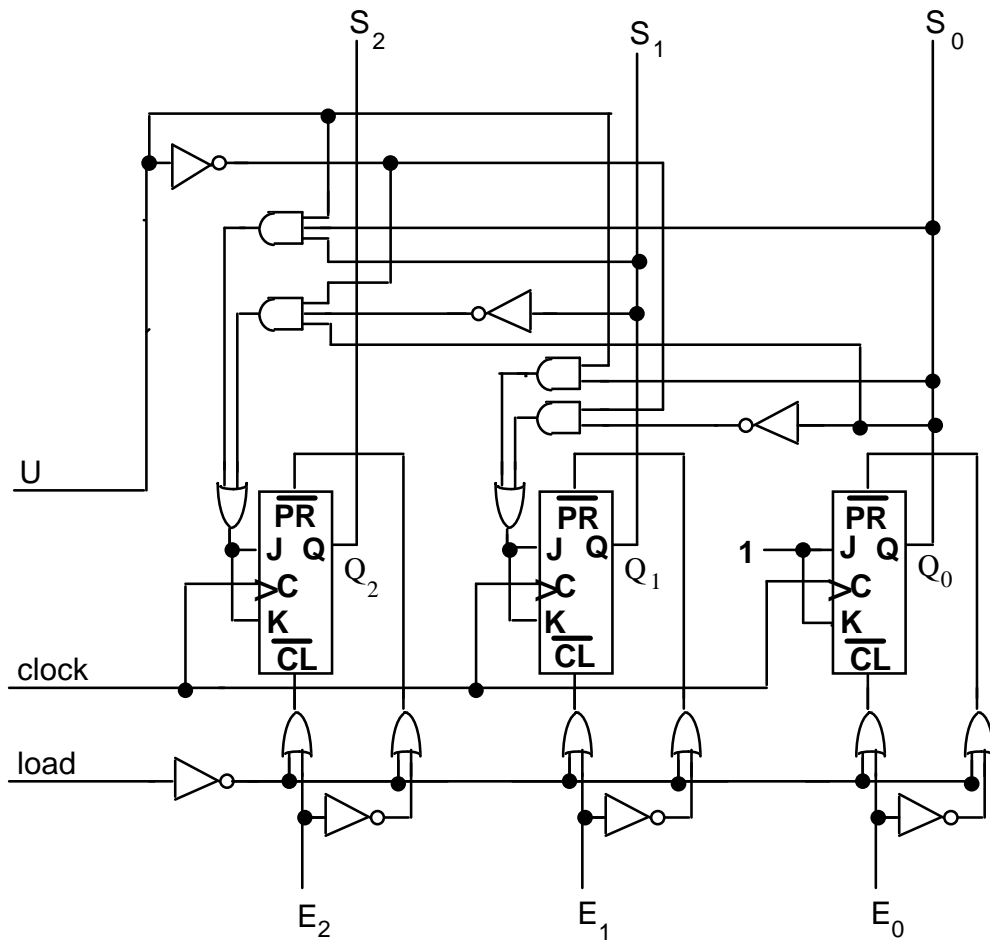


Figura 20.14. Registro con carga asincrónica en paralelo y salida en paralelo, que puede ser incrementado o decrementado sincrónicamente.

Se realizará el diseño de este registro que puede incrementarse o decrementarse en forma sincrónica además de tener carga asincrónica en serie y salida en paralelo.

La parte del circuito lógico para la carga asincrónica es igual que la utilizada en el registro de la figura 20.2. Las funciones de incremento y decremento en forma sincrónicas corresponden a un contador de tres bits que pueda contar tanto hacia arriba como hacia abajo, similar al diseñado en el capítulo 19, solamente que ahora es de ocho estados y se utilizarán FLIP FLOPS del tipo J-K en lugar de FLIP FLOPS T.

Se requieren tres celdas biestables para construir el contador y se asignarán los códigos 000, 001, 010, 011, 100, 101, 110 y 111 a los estados.

La entrada U si influye para determinar el siguiente estado pero no toma parte en la determinación de la salida del circuito secuencial. Dada esta observación, se usará el modelo de

Moore para el diseño del circuito secuencial. Se usarán las variables Q_2 , Q_1 y Q_0 para la salida de cada uno de los FLIP FLOPS que forman el estado del circuito secuencial. La salidas están representadas por el mismo estado. El diagrama de estados se presenta en la figura 20.15.

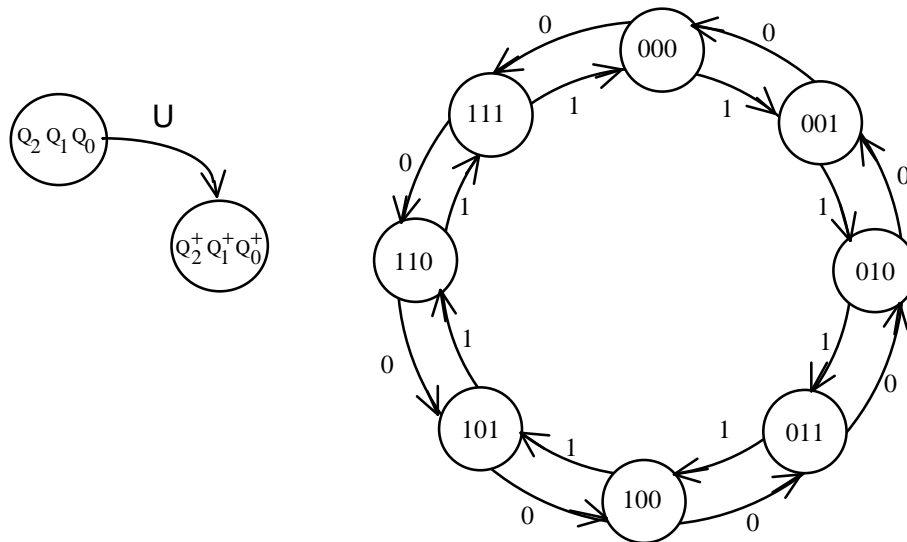


Figura 20.15. Diagrama de estados del contador de 0 a 7 y de 7 a 0.

Analizando el diagrama se puede ver que no es posible reducir el número de estados y solamente falta construir la tablas de entradas, estado futuro, transiciones y de excitaciones para determinar las ecuaciones para las entradas J y K de cada uno de los tres FILP FLOPS que forman el estado del circuito.

La tablas se muestra en la tabla 20.3.

U	Q_2	Q_1	Q_0	Q_2^+	Q_1^+	Q_0^+	t_2	t_1	t_0	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	1	1	1	∞	∞	∞	1	X	1	X	1	X
0	0	0	1	0	0	0	0	0	β	0	X	0	X	X	1
0	0	1	0	0	0	1	0	β	∞	0	X	X	1	1	X
0	0	1	1	0	1	0	0	1	β	0	X	X	0	X	1
0	1	0	0	0	1	1	β	∞	∞	X	1	1	X	1	X
0	1	0	1	1	0	0	1	0	β	X	0	0	X	X	1
0	1	1	0	1	0	1	1	β	∞	X	0	X	1	1	X
0	1	1	1	1	1	0	1	1	β	X	0	X	0	X	1
1	0	0	0	0	0	1	0	0	∞	0	X	0	X	1	X
1	0	0	1	0	1	0	0	∞	β	0	X	1	X	X	1
1	0	1	0	0	1	1	0	1	∞	0	X	X	0	1	X
1	0	1	1	1	0	0	∞	β	β	1	X	X	1	X	1
1	1	0	0	1	0	1	1	0	∞	X	0	0	X	1	X
1	1	0	1	1	1	0	1	∞	β	X	0	1	X	X	1

1	1	1	0	1	1	1	1	1	∞	X	0	X	0	1	X
1	1	1	1	0	0	0	β	β	β	X	1	X	1	X	1
(a)				(b)				(c)				(d)			

Tabla 20.3. Tablas del contador de 0 a 7 y de 7 a 0.

(a) tabla de entradas, (b) tabla de estado futuro, (c) tabla de transiciones, (d) tabla de excitaciones.

Los mapas de Karnaugh y las ecuaciones mínimas para la excitación se muestran en la figura 20.16.

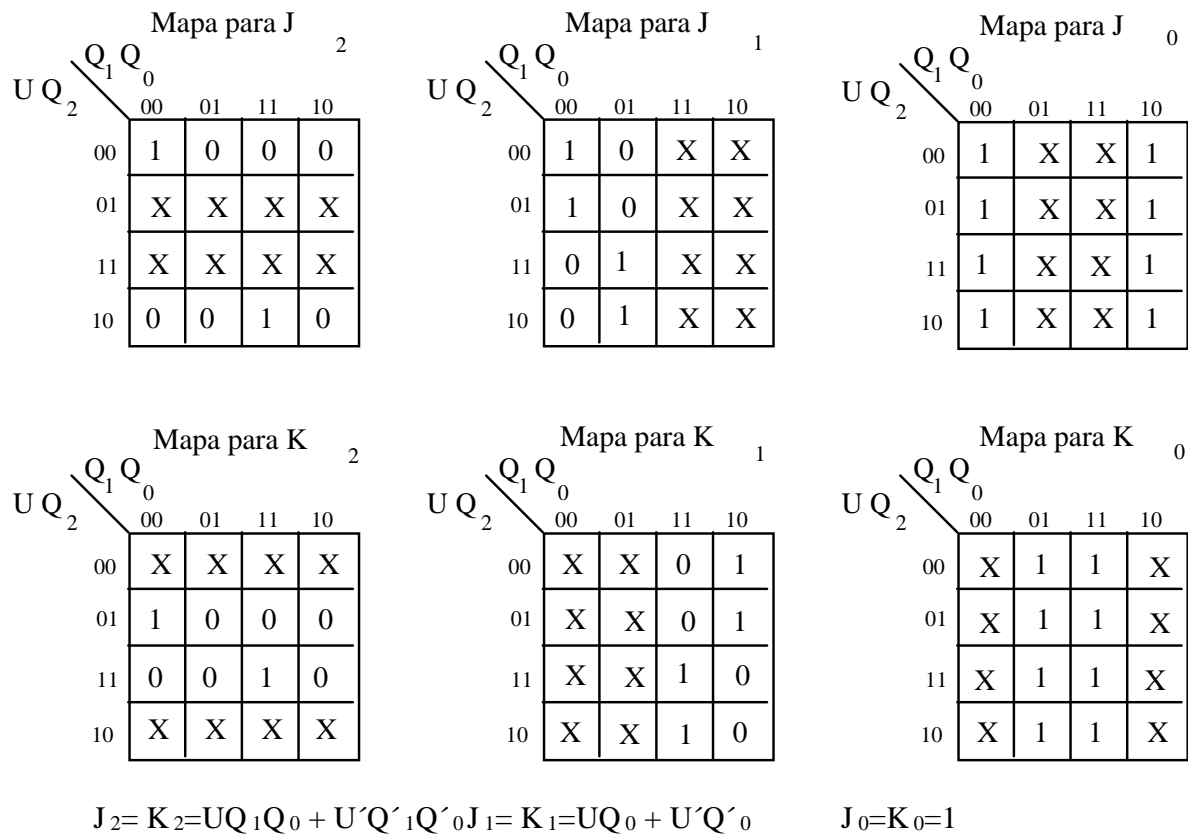


Figura 20.16. Mapas de Karnaugh y ecuaciones mínimas para la excitación.

Las salidas (S_2 , S_1 y S_0) están dadas directamente por la salida de cada uno de los FLIP FLOPS que forman el estado del circuito secuencial (Q_2 , Q_1 y Q_0) como se muestra en la figura 20.14.

Por último se diseñará un registro de tres bits, con carga sincrónico en paralelo y salida en paralelo, además podrá incrementar y decrementar. Se tendrá dos entradas de control, las cuales indican la función a realizar, como se especifica en la tabla 20.4.

C ₁	C ₀	Función
0	0	Mantiene su valor
0	1	Carga en paralelo
1	0	Incrementa
1	1	Decrementa

Tabla 20.4 Funcionamiento del registro síncronico de 3 bits

Se utilizará FLIP FLOPS T para el diseño de este registro.

La entrada de información al registro se realiza a través de las líneas E₂, E₁ y E₀. La salida se obtiene de las líneas S₂, S₁ y S₀, las cuales están dadas directamente por la salida de cada uno de los FLIP FLOPS. Se usarán las variables Q₂, Q₁ y Q₀ para la salida de cada uno de los FLIP FLOPS que forman el estado del circuito secuencial

Este registro se puede diseñar de la misma forma en que se hizo en los casos anteriores, esto es, definiendo un diagrama de estados donde se consideren todas las opciones y, de éste se obtengan las tablas, las funciones de excitación y posteriormente el circuito correspondiente. El diagrama de estados que se obtiene quedaría complicado ya que las transiciones de un estado a otro dependen de las dos variables de control del registro y de las tres entradas de datos del registro. La obtención de este diagrama se deja como ejercicio para el lector. Para simplificar el diseño se utilizará selectores, por medio de los cuales se dividirá el problema, ya que con éstos se puede controlar el valor que se tenga presente en las entradas de control de los FLIP FLOPS. El esquema general de este circuito se muestra en la figura 20.17.

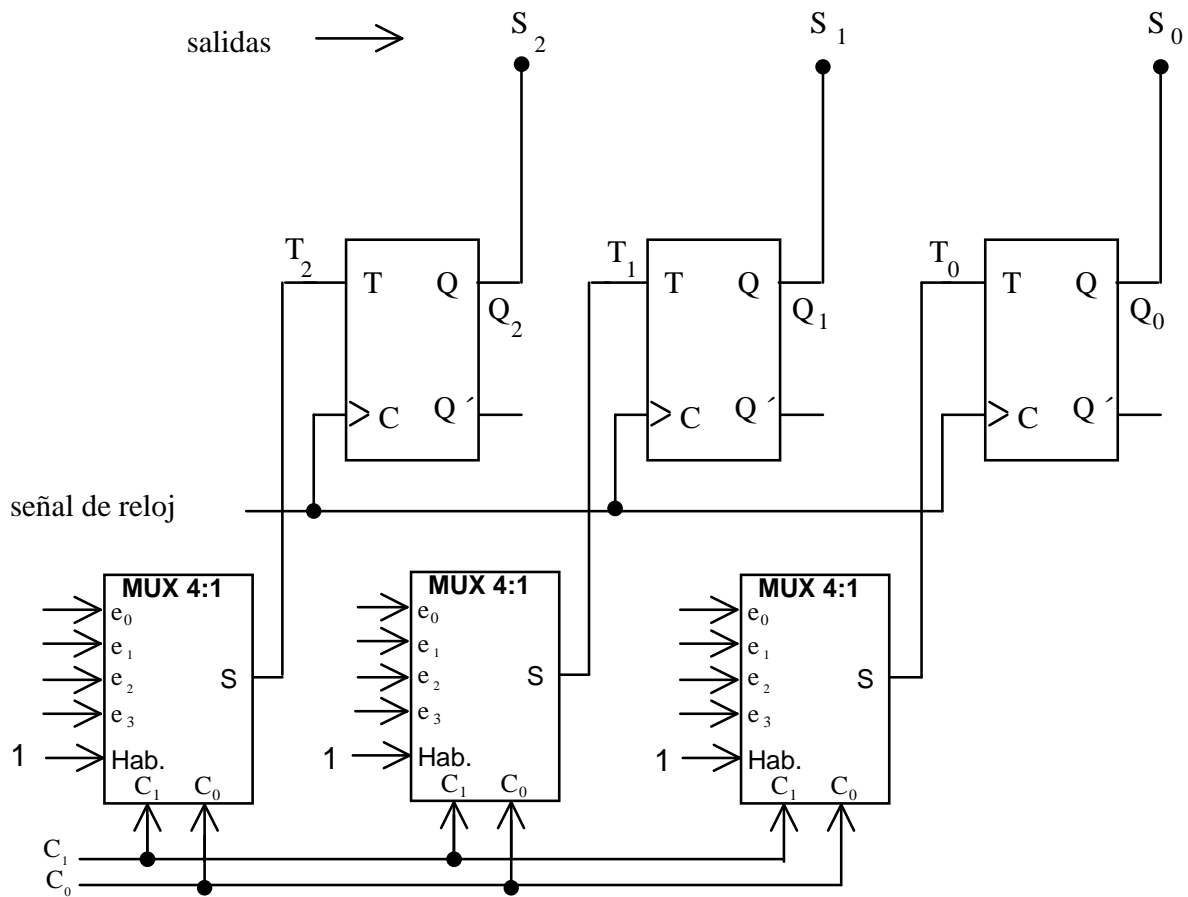


Figura 20.17. Registro sincrónico construido con ayuda de selectores.

Como se puede observar en la figura 20.17 el valor que se tiene en las entradas de los FLIP FLOPS es determinado por las entradas de control del registro. Si en las entradas de control del registro (C_1 y C_0) se tiene 0 y 0, los valores que se tienen en las entradas cero de los selectores, serán los que estén presentes en las entradas de los FLIP FLOPS. Y así sucesivamente para los demás valores de las variables de control del registro.

Al tener este esquema de circuito el diseño se reduce a determinar los valores que se deben tener en cada una de las entradas de los selectores. Para lo cual se analizará en forma independiente cada una de las cuatro funciones que realiza el registro.

La primera función, en la cual el registro mantiene su valor, basta con colocar ceros en las entradas de los FLIP FLOPS, ya que un FLIP FLOP T mantiene su valor si su entrada es cero. Por lo tanto todas las entradas cero de los selectores tendrán un valor de cero.

Para obtener los valores que deben tener las entradas uno de los selectores, se tiene que determinar el valor que debe tener la entrada de un FLIP FLOP T para que este almacene el valor de una entrada externa (E_i).

Las tablas de entradas, estado futuro, transiciones, y excitaciones para este caso se muestran en la tabla 20.5.

E_i	Q_i	Q_i^+	t_i	T_i
0	0	0	0	0
0	1	0	∞	1
1	0	1	β	1
1	1	1	1	0

(a) (b) (c) (d)

Tabla 20.5. Tablas para la función dos.

(a) tabla de entradas, (b) tabla de estado futuro, (c) tabla de transiciones, (d) tabla de excitaciones.

De la tabla de excitaciones se obtiene el valor de la entrada del FLIP FLOP T, el cual es $E_i \oplus Q_i$, donde E_i es la entrada externa y Q_i es la salida del FLIP FLOP.

Par la función de incremento, se obtienen los valores que deben tener las entradas de los 3 FLIP FLOP T, para que trabajen como un circuito síncronico que cuenta de 0 a 7. El contador síncronico de 0 a 7, es el primer ejemplo de diseño de circuitos síncrónicos, que se realiza en el capítulo 19. Las funciones que se obtienen para las entradas de control de los FLIP FLOP son las siguientes:

$$\begin{aligned} T_0 &= 1. \\ T_1 &= Q_0 \\ T_2 &= Q_2 Q_1 \end{aligned}$$

Por lo tanto la entrada 2 de sector tendrán estos valores.

Par determinar los valores de las entradas 3 de los selectores se deben de obtener los valores de las entradas de control de los FLIP FLOPS T para que operen como un circuito síncronico que cuente de 7 a 0. Si se usa el procedimiento para diseñar circuitos síncrónicos, las funciones que se obtienen para las entradas de control de los FLIP FLOP T son las siguientes:

$$\begin{aligned} T_0 &= 1. \\ T_1 &= Q_0' \\ T_2 &= Q_2' Q_1' \end{aligned}$$

Por lo tanto la entrada 3 de sector tendrán estos valores.

El circuito para este registro se muestra en la figura 20.18

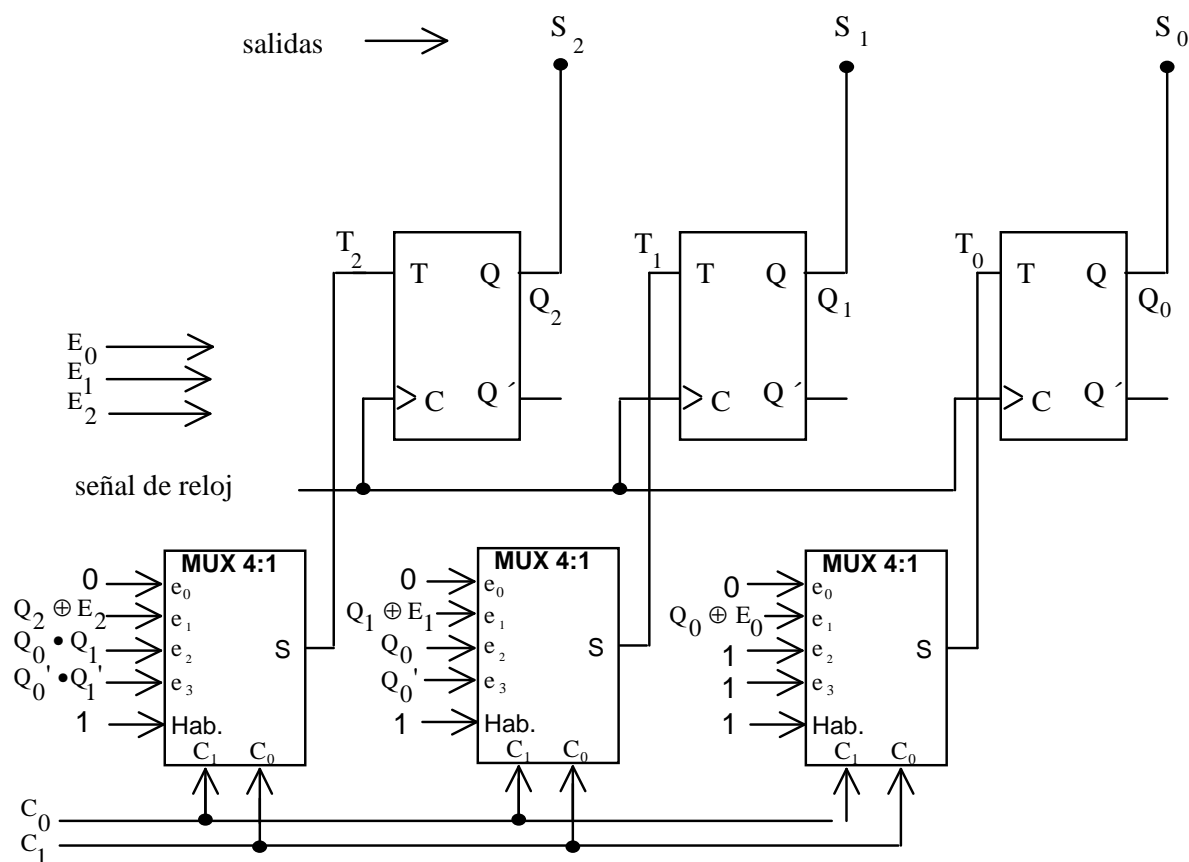


Figura 20.17. Registro sincrónico con cuatro funciones.

En este último circuito, solo se indico por medio de las funciones el circuito que deben tener las entradas los selectores. No se construyo el circuito completo para facilitar su comprensión.

20.8. TRANSFERENCIAS ENTRE REGISTROS.

Dado que dentro de una computadora hay más de un registro, se diseñará un circuito que permita la transferencia del contenido de cualquier registro a cualquier otro registro.

En la figura 20.18 se encuentra un circuito que permite transferir el contenido de un registro (registro fuente) a otro registro (registro destino). Si se analiza el circuito se puede observar que al indicar el registro fuente (cero, uno, dos o tres) en la entrada de control de los selectores, éstos pondrán el contenido del registro fuente en todas las entradas de datos de todos los registros. El registro destino se indica en las entradas de control del decodificador, el cual pondrá en uno la señal de Load del registro destino, y en cero en todas las señales de Load de los demás registros. Si los registros son de carga sincrónica, al momento en que se produzca la señal de reloj adecuada, el registro de destino se carga con el valor del registro fuente.

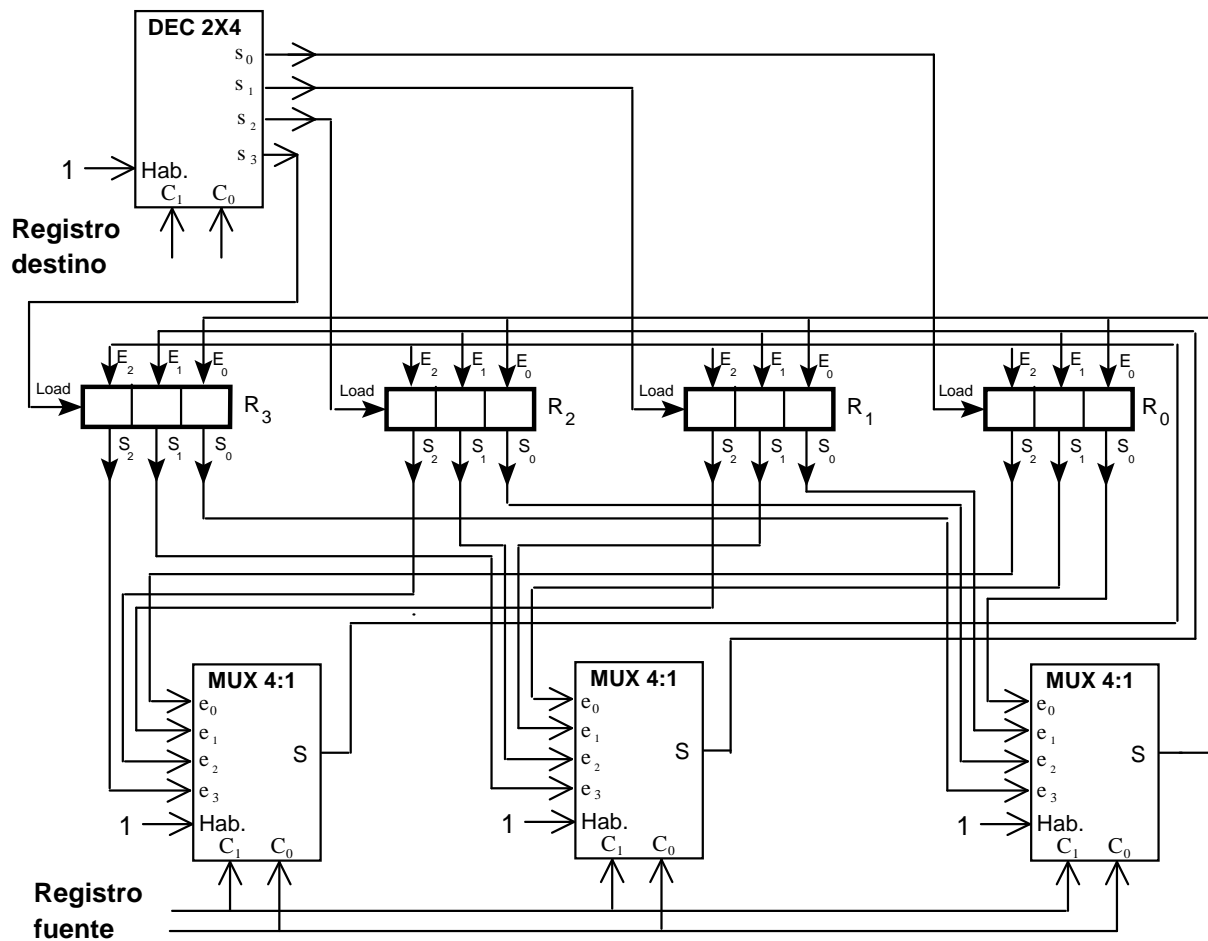


Figura 20.18. Transferencia entre registro usando lógica combinatoria.

El circuito mostrado en la figura 20.18 se puede construir de una forma más simple si se usa un BUS en donde todos los registros pueden colocar su contenido, o realizar una carga con el contenido del BUS. En la figura 20.19 se muestra este circuito que permite hacer transferencia entre registro, nótese que solo el registro indicado en las entradas de control del decodificador inferior (registro fuente) puede colocar su información en el BUS, dado que solo sus buffers de 3 estados se habilitan.

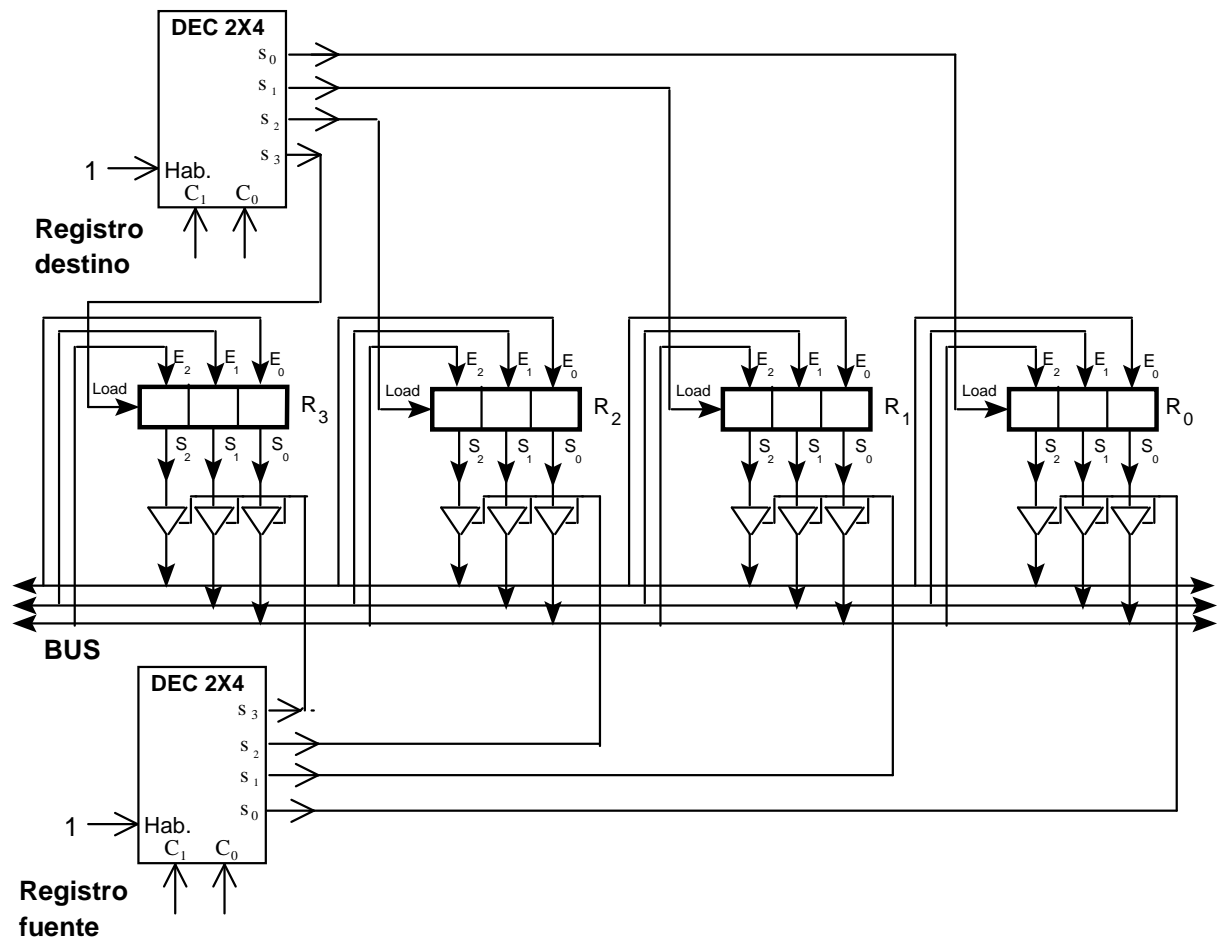


Figura 20.19. Transferencia entre registro usando un BUS.

Todos los registros pueden tomar la información del bus, pero solo el registro de destino, que se indica en las entradas de control del decodificador superior, almacenará esta información, ya que es el único con la señal de Load en uno. Si los registros son de carga sincrónica, al momento en que se produzca la señal de reloj adecuada, el registro de destino cargará el contenido del registro fuente.

20.9. RESUMEN.

Los registros son utilizados principalmente para almacenar datos o instrucciones dentro de una computadora digital en forma temporal. Adicionalmente, pueden realizar otras funciones tales como incrementar o decrementar su valor, efectuar corrimientos lógicos y aritméticos, servir como contadores, etc.

La carga de los registros puede realizarse en forma paralela o forma serial. La salida de los valores almacenados en los registros también puede obtenerse de las dos formas mencionadas

anteriormente. En esta sección se presentaron varios registros cuyo valor puede cargarse en paralelo y en serie, y la salida puede ser obtenida en serie o en paralelo.

Se utilizaron los conocimientos de los capítulos anteriores para diseñar registros que puedan realizar corrimientos lógicos y aritméticos, así como registros contadores. También se efectuó el diseño de registros que puedan realizar varias funciones, dependiendo de un conjunto de entradas de control.

Finalmente, se describieron las operaciones de transferencias entre registros, las cuales son indispensables para que las computadoras digitales puedan efectuar ciertas micro operaciones necesarias para su operación. En esta sección también se introdujo un bus para simplificar el diseño del circuito.

20.10. PROBLEMAS.

1. ¿Qué tipo de registros se utilizan en la unidad central de proceso de una computadora?
2. Si dos computadoras se quieren comunicar a través de una línea de comunicación donde se puede enviar un solo bit a la vez, qué tipo de registro seria el más recomendado en la computadora fuente y cuál en la computadora destino?
3. Construya un circuito que pueda sumar dos registros de 8 bits con entrada y salida en serie. El resultado se debe de almacenar en el primer registro. El segundo registro no se debe de modificar. Los registros contienen números positivos sin signo y el circuito no debe de detectar OVERFLOW. Se cuenta con los dos registros de 8 bits con entrada y salida en serie, un sumador completo (FA), un FLIP-FLOP D y un generador de 8 pulsos.
4. Construya el registro que se muestra en la figura 20.14 usando FLIP FLOPS D.
5. Construya un registro de tres bits con carga sincrónico en paralelo y salida en paralelo, que pueda incrementarse en una unidad y que permita realizar un corrimiento circular a la derecha. Se tendrán dos entradas de control, las cuales indican la función sincrónica a realizar, de acuerdo a lo especificado en la siguiente tabla;

C ₁	C ₀	Función
0	0	Mantiene su valor
0	1	Carga en paralelo
1	0	Incrementa
1	1	Corrimiento circular a la derecha

Utilice FLIP FLOPS D para el diseño de este registro. La entrada de información al registro se realiza a través de las líneas E₂, E₁ y E₀ y la salida se obtiene de las líneas S₂, S₁ y S₀

6. Diseñe un registro de tres bits con carga asincrónica en paralelo y salida en paralelo, además contará con dos entradas de control para poder realizar las funciones sincrónicas que se indican en la siguiente tabla:

C ₁	C ₀	Función
0	0	Mantiene su valor
0	1	Corrimiento a la derecha aritmético
1	0	Incrementa
1	1	Complemento a uno

Utilice FLIP FLOPS T para el diseño de este registro. La entrada de información al registro se realiza a través de las líneas E₂, E₁ y E₀ y la salida se obtiene de las líneas S₂, S₁ y S₀

7. Diseñe un registro de tres bits con carga en paralelo y salida en paralelo. El registro tendrá dos funciones asincrónicas controladas por las señales C y L. Si C = 1 el registro se debe de borrar y si L = 1 se hace una carga en paralelo. Además contará con dos entradas de control para poder realizar las funciones sincrónicas que se indican en la siguiente tabla.

C ₁	C ₀	Función
0	0	Mantiene su valor
0	1	Corrimiento a la Izquierda
1	0	Decrementa en 1
1	1	Complemento a dos

Utilice FLIP FLOPS T para el diseño de este registro. La entrada de información al registro se realiza a través de las líneas E₂, E₁ y E₀ y la salida se obtiene de las líneas S₂, S₁ y S₀

CAPITULO 21

EJEMPLOS DE CIRCUITOS SECUENCIALES.

En este capítulo se usarán los conceptos definidos en los capítulos anteriores para diseñar circuitos secuenciales que no están relacionados directamente con la arquitectura de una computadora. Además, se diseñarán circuitos con hardware y software, lo cual se conoce como firmware y se mostrarán las ventajas de esta forma de construir circuitos.

21.1. Diseño de un reloj digital.

En la figura 21.1 se muestra, mediante una serie de bloques, el diseño general de un reloj digital. En ésta, se muestran dos contadores de 0 a 59, uno de ellos es usado para llevar la cuenta de los segundos y el otro para los minutos. El tercer contador solo cuenta de 1 a 12 ya que se usa para llevar el registro de las horas. Obsérvese que las salidas de cada uno de los contadores son entradas a dos convertidores BCD a 7 segmentos, por lo cual, los contadores deben proporcionar salidas BCD.

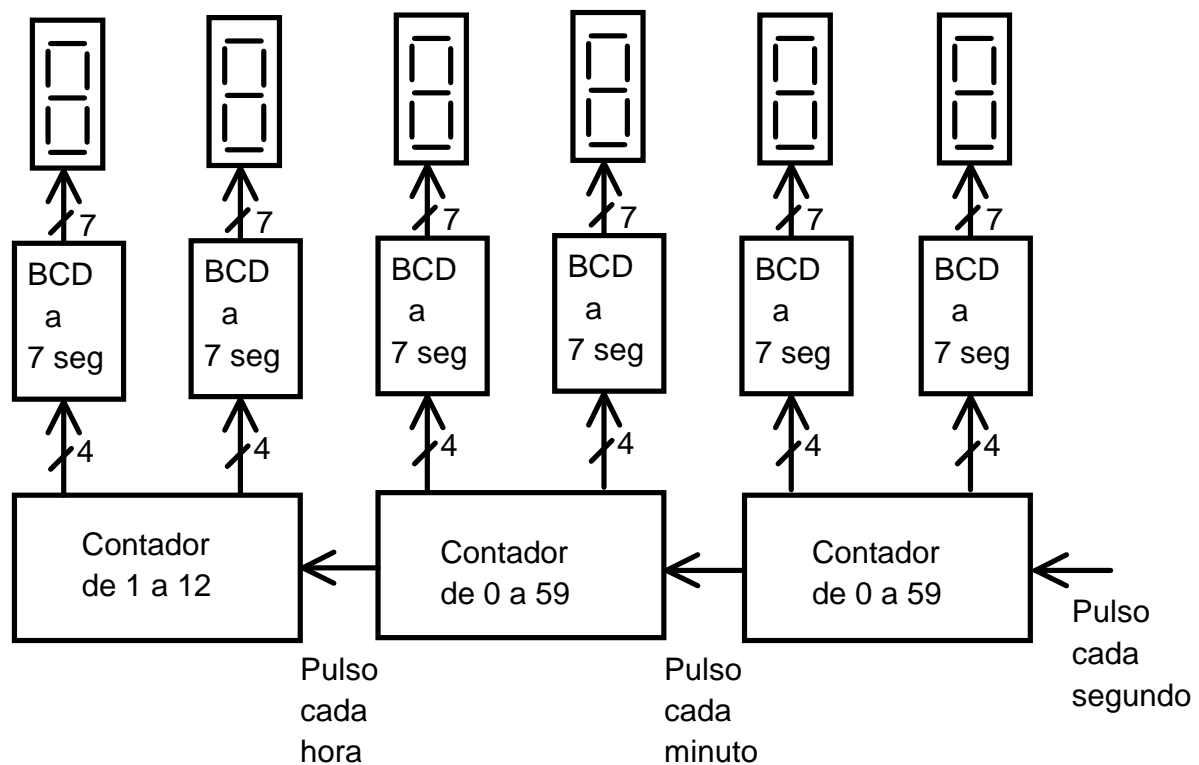


Figura 21.1 Diseño en bloques de un reloj digital.

En el diseño del contador de 0 a 59 se tendrán dos contadores, uno para llevar las unidades y el otro para las decenas. El contador de unidades será construido utilizando un contador sincrónico de 0 a 15, (este contador se encuentra integrado en un solo circuito (chip) y es muy fácil de conseguir). Si se construye este circuito con 4 FLIP FLOPS T con CLEAR, usando el procedimiento para diseñar circuitos secuenciales sincrónicos, se pueden encontrar las ecuaciones de excitación de los FLIP FLOPS T que son:

$$T_3 = Q_2 Q_1 Q_0$$

$$T_2 = Q_2 Q_1$$

$$T_1 = Q_1$$

$$T_0 = 1$$

donde T_i es la entrada y Q_i es la salida de cada uno de los FLIP FLOPS, T_0 es la entrada y Q_0 es la salida del FLIP FLOP menos significativo. La figura 21.2 muestra este contador.

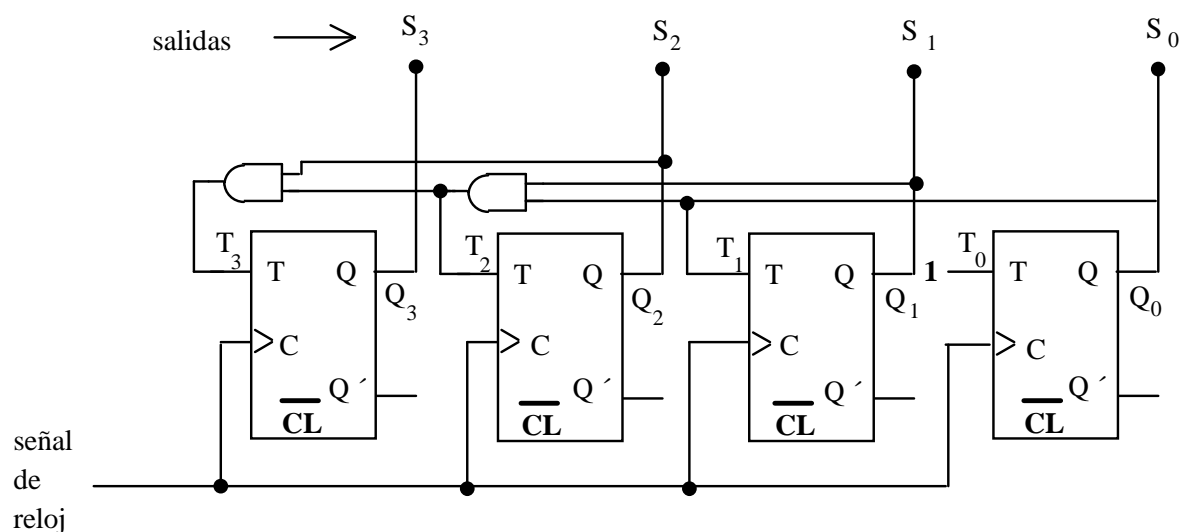


Figura 21.2 Contador sincrónico de 0 a 15.

Si se modifica el circuito anterior agregando una lógica que detecte cuando el contador llega a 10, para que en ese preciso momento se de un Clear a todos los FLIP FLOP, se tendrá un contador que cuente solamente de 0 a 9. La figura 21.3 muestra este contador. Observe que si el contador inicia su secuencia en el estado 0, la salida del NAND tendrá el valor de 1, hasta que se llegue al estado 10, y solo se encontrará en éste estado por un instante de tiempo muy pequeño, ya que el contador regresará al estado 0.

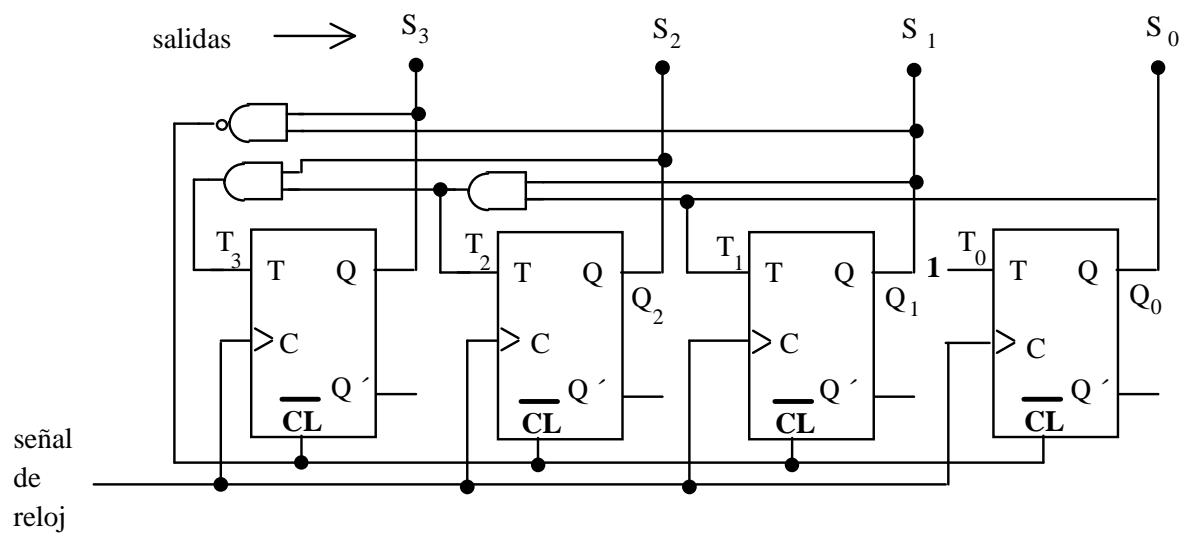


Figura 21.3 Contador sincrónico de 0 a 9.

El contador de decenas será construido de la misma manera, solo que se utilizara un contador de 0 a 7 (figura 19.3) y se dará el Clear a todos los FLIP FLOP cuando se detecte que el contador llega a 6. En la figura 21.4 se muestra el contador de 0 a 59 con salidas en BCD.

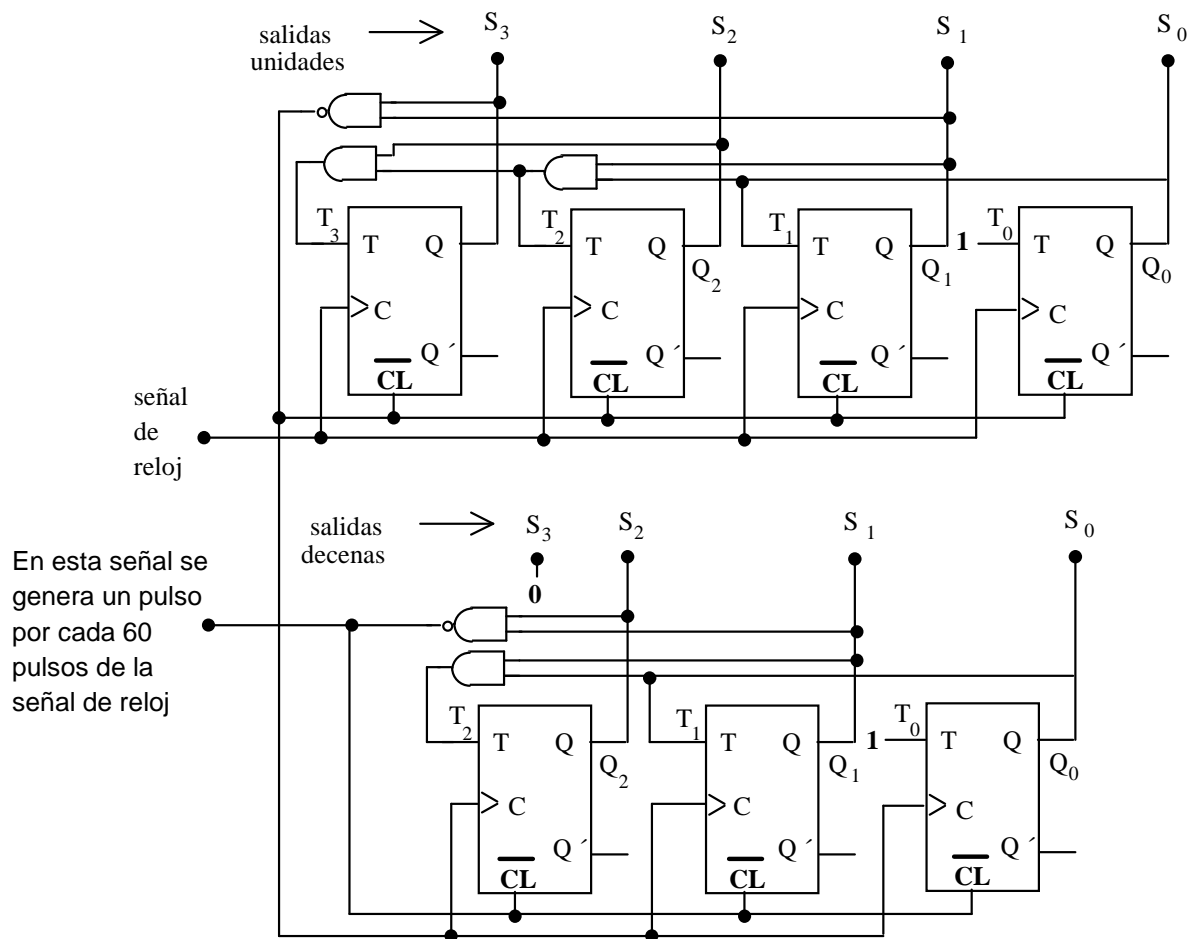


Figura 21.4 Contador de 0 a 59 con salidas BCD.

Observe en la figura 21.4 que la salida del NAND del contador de unidades se usa como señal de reloj del contador de decenas, ya que cuando el contador de unidades llega a 10, la salida del NAND se pasa a 0, ocasionando que a todos los FLIP FLOPS de este contador se les de un clear. Al estar este contador en el estado 0, la salida del NAND toma otra vez el valor de uno, generándose la transición positiva en esta señal. En el NAND del contador de decenas, se genera un pulso por cada 60 transiciones positivas de la señal de reloj.

Para que el contador de la figura 21.4 funcione adecuadamente todos los FLIP FLOPS deben iniciar en estado cero.

El contador de 1 a 12 es construido de una forma semejante. Se modificara el contador de 0 a 16 para que solamente cuente de 0 a 9 y se utilizará un FLIP FLOP para que se lleve registro de las decenas. Se usará una lógica adicional para detectar cuando el contador llega a 13 (el FLIP FLOP de las decenas encendido y el contador de unidades en 3) para borrar el FLIP FLOP indicador de decenas y el segundo bit menos significativo del contador de unidades. Esto ocasionará que nuestro contador reinicie en 1. El contador de 1 a 12 se muestra en la figura 21.5.

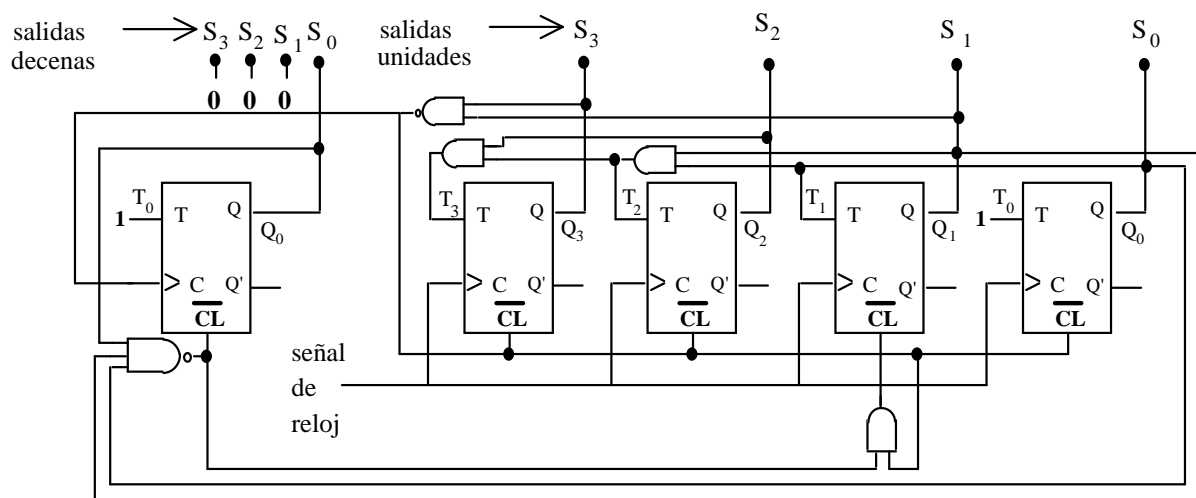


Figura 21.5 Contador de 0 a 12 con salidas BCD.

Par que el contador de la figura 21.5 funcione adecuadamente, todos los FLIP FLOPS deben iniciar en estado cero con excepción del bit menos significativo del contador de unidades.

Al sustituir los bloques de los contadores que aparecen en la figura 21.1 por el contador de 0 a 59 y el contador de 1 a 12 que se definieron, se tiene el diseño completo del reloj digital.

La señal de reloj que llega al contador de segundos es externa y debe tener una frecuencia de un segundo. La señal de reloj que llega al contador de minutos es generada por el contador de segundos, en esta señal se genera un pulso cada 60 segundos. El contador de minutos genera la señal de reloj para el contador de horas, la cual es una señal donde se produce un pulso cada 60 minutos.

21.2. Diseño de un semáforo para el cruce de dos calles, de un solo sentido.

En la figura 21.6 se muestra el cruce para el cual se quiere diseñar el semáforo.

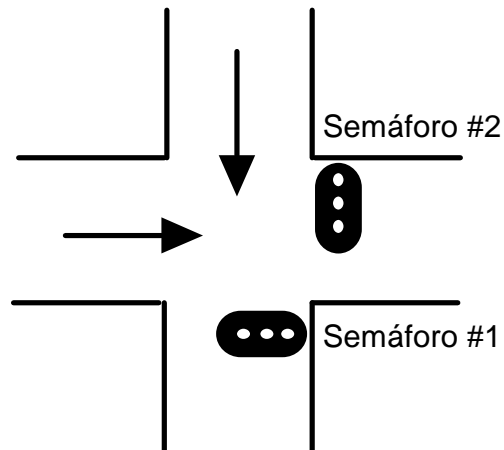


Figura 21.6 Cruce de dos calles de un solo sentido

Llamaremos R1, A1 y V1 a los focos rojo, amarillo y verde del semáforo número uno y, R2, A2 y V2 los focos del semáforo número dos. Al igual que un semáforo real, se desea que el tiempo que permanece encendido cada uno de los focos sea diferente. Para este caso daremos un tiempo de 3 segundos al verde del semáforo número uno, 6 segundos al verde del semáforo número dos y los amarillos tendrán una duración de 1 segundo.

En la figura 21.7 se muestra el diagrama de estados para este circuito. En cada uno de los estados se indica el tiempo que debe de permanecer.

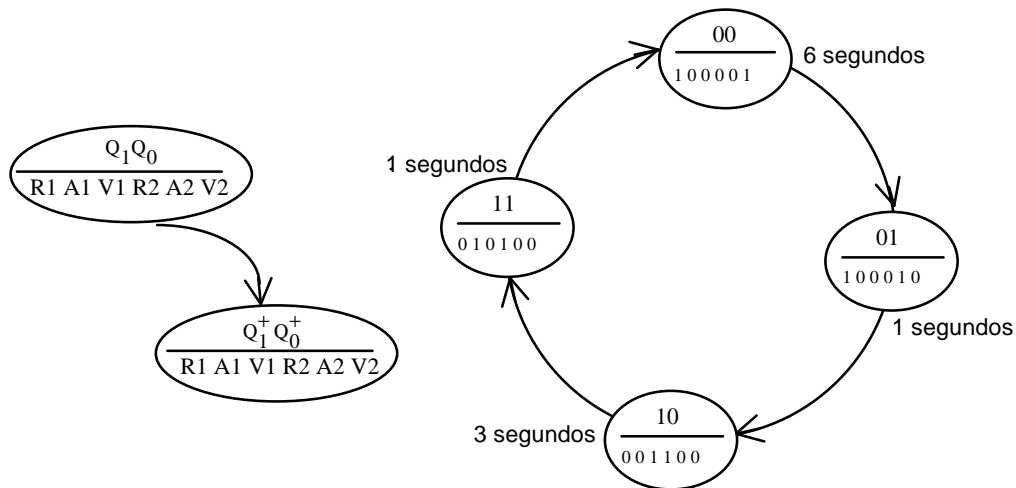


Figura 21.7 Diagrama de estados para el semáforo.

El diseño del semáforo a nivel bloques se muestra en la figura 21.8.

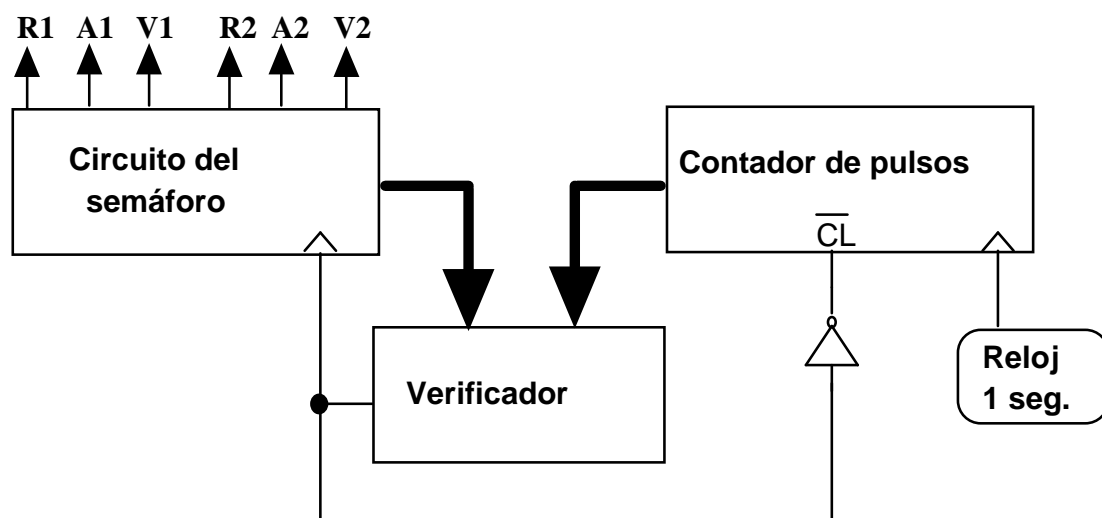


Figura 21.8 Diseño del semáforo a nivel bloques.

El circuito del semáforo se obtiene al diseñar un circuito secuencial síncronico para el diagrama 21.7 sin tener en cuenta que cada estado puede durar un tiempo diferente. La señal de reloj para el circuito del semáforo es generada por el circuito verificador, el cual analiza el estado en que se encuentra el semáforo y la cantidad de pulsos que se han registrado de un segundo, al cumplirse el tiempo, éste envía un pulso al circuito del semáforo y da un CLEAR al contador de pulsos.

Dado que la cantidad máxima de pulsos que se tiene que contar es 6, se usará un contador síncronico de 0 a 7 con CLEAR, para registrar la cantidad de pulsos que han llegado del reloj de un segundo.

Par diseñar el circuito del semáforo se usarán dos FLIP FLOPS T (Q_1 y Q_0) dado que el diagrama de estados del semáforo solamente tiene 4 estados. La tabla 21.1 muestra la tablas de entradas, estados futuros, transiciones, excitaciones, y salidas, usadas para el diseño de este circuito.

Q_1	Q_0	Q_1^+	Q_0^+	t_1	t_0	T_1	T_0	R_1	A_1	V_1	R_2	A_2	V_2
0	0	0	1	0	∞	0	1	1	0	0	0	0	1
0	1	1	0	∞	β	1	1	1	0	0	0	1	0
1	0	1	1	1	∞	0	1	0	0	1	1	0	0
1	1	0	0	β	β	1	1	0	1	0	1	0	0
(a)		(b)		(c)		(d)		(e)					

Tabla 21.1. Tablas del circuito del semáforo. (a) tabla de entradas, (b) tabla de estado futuro, (c) tabla de transiciones, (d) tabla de excitaciones, (e) tabla de salidas.

Las variables de control de los FLIP FLOPS en este caso son T_1 , T_0 . De la tabla de excitación se puede obtener que $T_1 = Q_0$ y $T_0 = 1$.

De la tabla de salidas se obtienen las siguientes funciones de salida:

$$\begin{aligned} R_1 &= Q_1' \\ A_1 &= Q_1 Q_0 \\ V_1 &= Q_1' Q_0' \\ R_2 &= Q_1 \\ A_2 &= Q_1' Q_0 \\ V_2 &= Q_1' Q_0' \end{aligned}$$

La figura 21.9 muestra el circuito del semáforo. Obsérvese que las salidas dependen solamente del estado presente, por lo cual este circuito es un modelo de Moore.

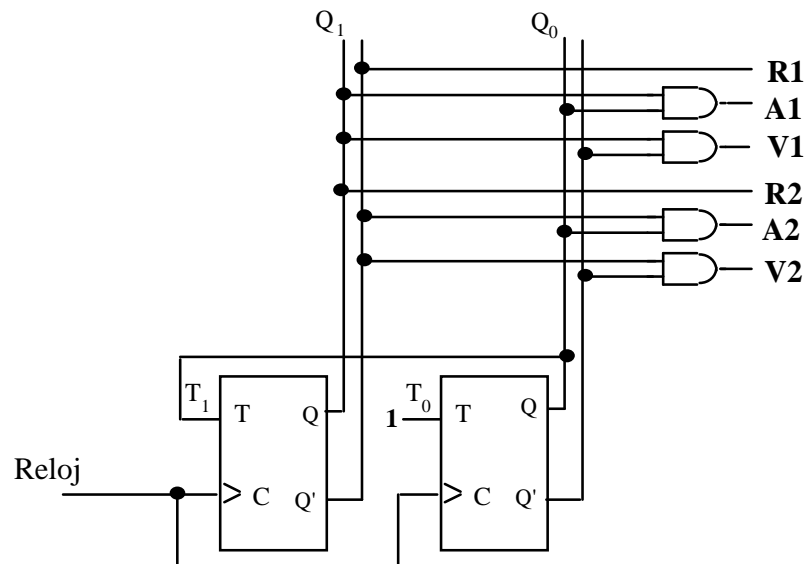


Figura 21.9 Circuito el semáforo.

El circuito verificador se forma con dos decodificadores, una para identificar el estado en que se encuentra el semáforo y el otro para determinar la cantidad de tiempo que se ha permanecido en el estado. Adicionalmente, se tiene una lógica formada por compuertas con la cual se detectan las condiciones en las que debe cambiar el estado al semáforo y reiniciar el contador de tiempo.

La figura 21.10 muestra el diseño completo del semáforo. Para analizar el circuito, suponga que el semáforo se encuentra en el estado cero (00), el cuál enciende las salidas del semáforo R1 y V2, y que el contador de pulsos se encuentra en el estado cero (000). Como podrá comprobarse, la salida del verificador es cero en este momento, ya que la salidas de los cuatro AND tienen un cero. Cada vez que el reloj genere un pulso (se genera un pulso cada segundo), el contador de pulsos se incrementará. La salida del verificador permanecerá en 0 hasta que el contador de pulsos llegue al estado 6 (110), ya que la salida del AND marcado con el numero 1 tomará el valor de uno, indicando que el circuito del semáforo está en el estado cero y que ya han

transcurrido 6 segundos. La salida del OR en el verificador tomará el valor de uno, generando dos acciones:

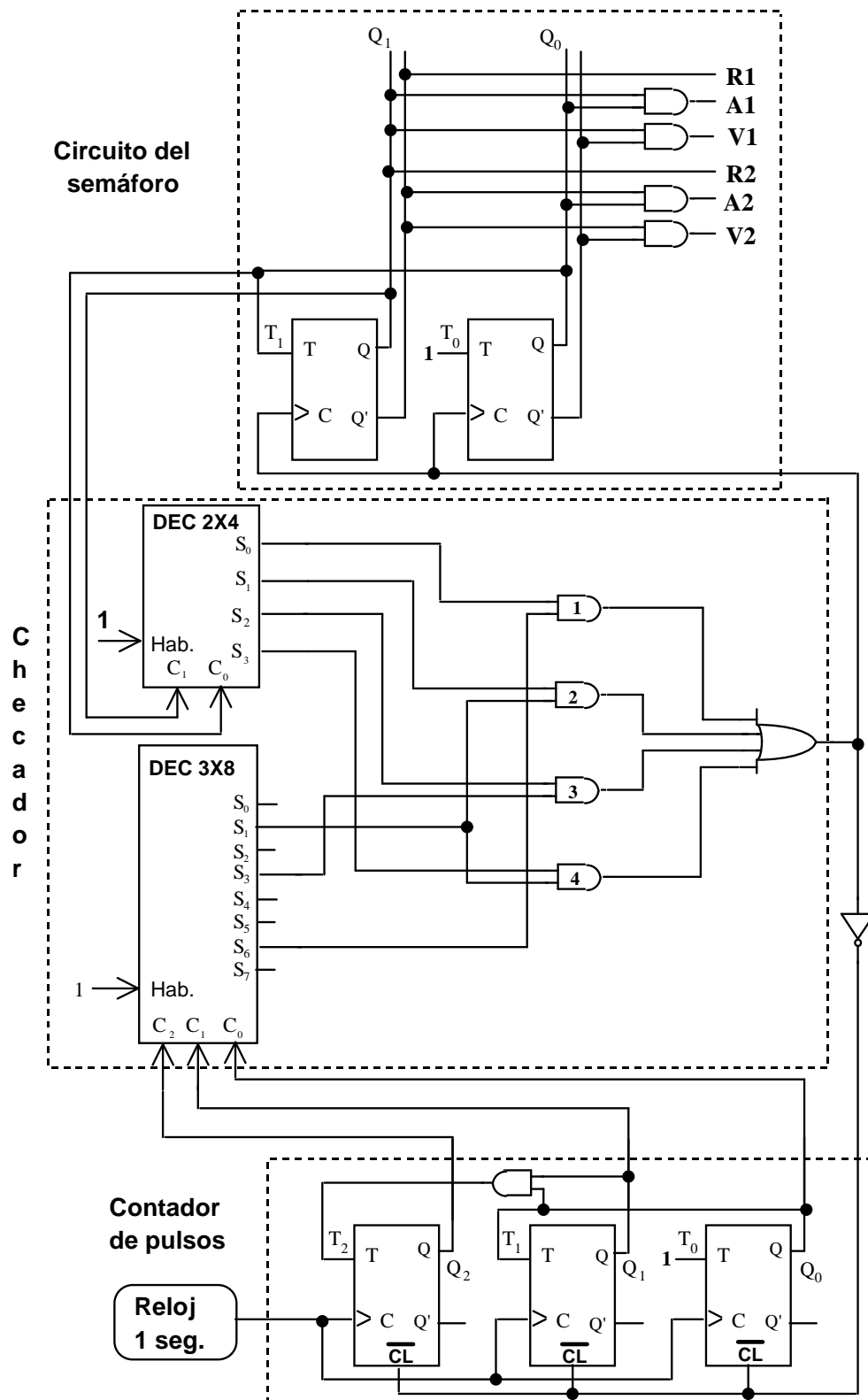


Figura 21.10 Circuito del semáforo completo.

La primera es hacer cero todas las entradas CLEAR de los FLIP FLOPS del contador de pulsos, lo cual hará que el contador se reinicialice, haciendo cero las salida del AND marcado con el número 1 y el OR del verificador.

La otra acción ocurre en el circuito del semáforo al pasar la salida del verificador de 0 a 1 y a 0 otra vez. En las entradas de reloj de los FLIP FLOPS del circuito se genera la transición para que éste cambie al estado 01, encendiendo las salidas del semáforo R1 y A2. Los AND marcados con los números 2, 3 y 4 son usados para identificar la condiciones de cambio para los estados 01, 10 y 11 del circuito del semáforo.

21.3. Construcción de una máquina Moore con firmware.

Firmware, también llamado microcódigo, son rutinas de software que se almacenan en ROM y dirigen el funcionamiento del hardware. El circuito básico para construir un diagrama de estados con frimware se muestra en la figura 21.11.

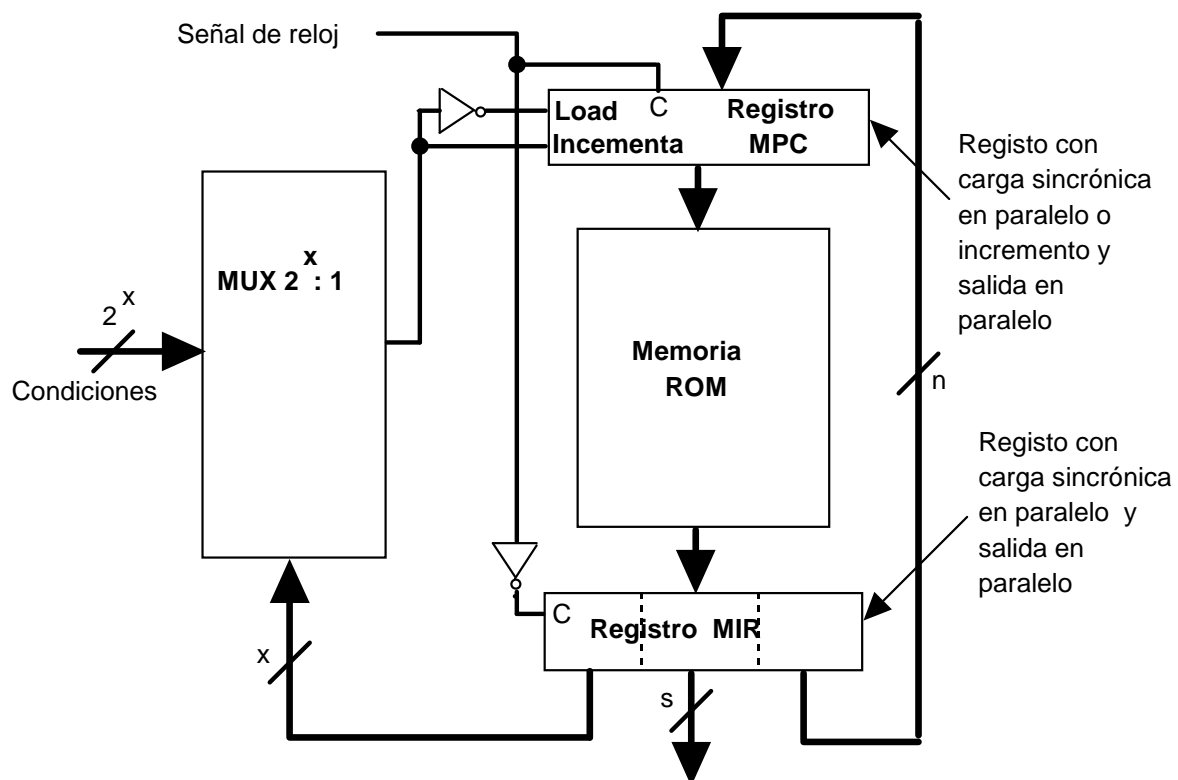


Figura 21.11 Circuito básico para construir un diagrama de estados con frimware.

En la memoria ROM se encuentran almacenadas las microinstrucciones que controlan el funcionamiento del circuito. El registro que contiene la dirección del ROM de la microinstrucción a ejecutar, se llama MPC (Micro Program Counter). Este es un registro que puede incrementar su valor en uno o cargar un nuevo valor en cada transición positiva en la señal

del reloj. El registro que almacena la microinstrucción que se está ejecutando se llama MIR (Micro Instruction Register). Este registro carga un nuevo valor en todas las transiciones negativas en la señal del reloj.

El formato de la micro-nstrucción es el siguiente:

X bits	S bits	N bits
Condición	Salidas externas	Siguiente dirección

El campo de condición indica la entrada al selector que se tomará en cuenta para determinar la siguiente instrucción a ejecutar.

El campo de salidas externas, indica los valores que tomarán las salidas del circuito.

El campo de siguiente dirección contiene la dirección de la memoria ROM de la siguiente microinstrucción a ejecutar cuando la entrada del selector que se indica tiene el valor de cero.

Cada microinstrucción produce dos acciones. Una es dar valores a las salidas externas y la otra es indicar la condición (entrada al selector) que se tomará en cuenta para determinar si la siguiente microinstrucción a ejecutar es la que sigue en forma secuencial o la que se encuentra en la dirección indicada por el campo de siguiente dirección.

Para ejemplificar el funcionamiento del firmware se utilizará el diagrama de estados que se muestra en La figura 21.12

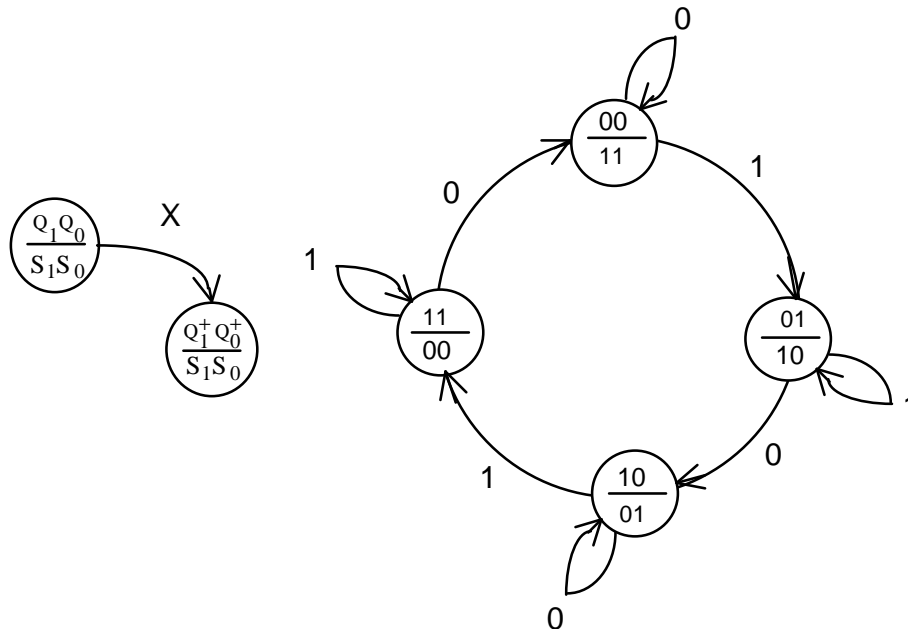


Figura 21.12 Diagrama de estados de un modelo Moore.

La construcción de esta máquina de Moore se encuentra en la figura 21.13. En el diseño se utilizan dos señales de reloj, la que llamamos reloj-1, que es la señal de reloj utilizada para

indicar el cambio de estado en la máquina de Moore. Y la señal de reloj-2, utilizada para sincronizar el hardware que ejecuta el microcódigo. Observe que cuando se genere una transición positiva en ésta señal se hace una carga o un incremento en el MPC, dependiendo de la condición que se esté indicando, y cuando ocurra una transición negativa se hace la carga en el MIR lo cual inicia la ejecución de una nueva microinstrucción. La señal de reloj-2 debe tener una frecuencia mucho mayor a la del reloj-1, ya que un estado de la máquina de Moore se representa en un conjunto de microinstrucciones.

El tamaño del MPC es de cuatro bits y el tamaño del MIR es de nueve bits.

En el microcódigo la microinstrucción definida por E XX es donde inicia el estado XX del diagrama de estados de la figura 21.12.

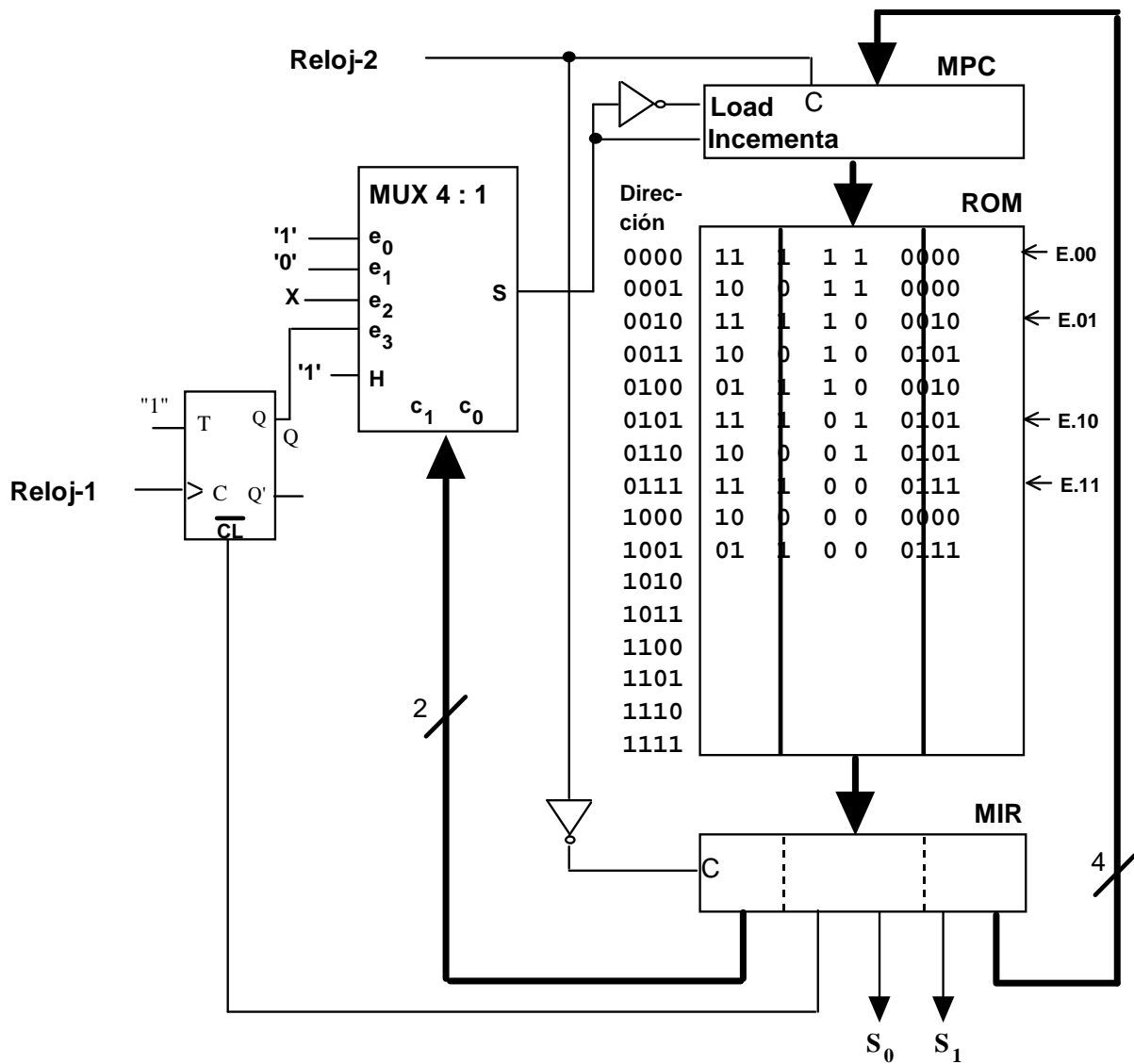


Figura 21.13 Modelo Moore con firmware.

Las condiciones que se tienen en el selector son las siguientes: condición 0 es un uno lógico, esta condición es utilizada cuando se quiere ejecutar una serie de instrucciones en forma secuencial (no usada en este ejemplo). La condición 1 es un cero lógico, esta condición es utilizada para hacer un salto incondicionado. La condición 2, se tiene la entrada externa X, al cuestionar esta condición dependerá del valor de X la acción a tomar, si $X=1$ la siguiente microinstrucción a ejecutar es la que se encuentra en forma secuencial y si $x=0$ se realizará un salto. Y la condición 3 sirve para detectar cuando se ha generado una transición positiva en la señal de reloj, note que la microinstrucción que se ejecuta inmediatamente después de detectar este evento regresa el valor de cero al flip flop.

Par analizar el funcionamiento del circuito suponga que el MPC tiene un cero, que el flip flop vale cero y que se genera la transición negativa en la señal de reloj-2. En el MIR se carga la palabra cero del ROM, en la cual se encuentra la primer microinstrucción del estado cero. Esta microinstrucción produce las salidas externas del estado cero (11) y no borra el flip flop, además indica la condición tres (11) la cual sirve para detectar cuando ha llegado la transición positiva en la señal de reloj-1. Cuando se genere la transición positiva en la señal de reloj-2 el MPC cargará la dirección cero (0000) si no ha llegado la transición en la señal de reloj-1 o se incrementará si ya llegó.

Note que la microinstrucción que se encuentra en la dirección 0001 no se ejecuta hasta que llegue la transición positiva en la señal de reloj-1. Al ejecutarse esta microinstrucción, borra el flip-flop y cuestiona el valor de X (condición 10), si $X=0$ se ejecuta un salto a la dirección 0000 lo cual hace que nos regresemos al estado cero, como lo indica el diagrama de estados. Y si $X=1$ el MPC se incrementa, haciendo que la siguiente microinstrucción a ejecutar sea la que se encuentra en la dirección 0010, que es donde inicia el estado uno.

Al ejecutar la microinstrucción que se encuentra en la dirección 0010 se producen las salidas del estado uno (10) y se utiliza la condición 2 la cual nos sirve para esperar a que se genere la transición positiva en la señal de reloj-1, como en la microinstrucción de la dirección 0000.

Al llegar la transición positiva en la señal de reloj-1 se ejecutará la microinstrucción de la dirección 0011, la cual borra el flip flop y cuestiona el valor de X. Ahora si $X = 0$ se realizará un salto a la dirección 0101, que es donde inicia el estado dos. Y si $X=1$ solo se incrementará el MPC haciendo que se ejecute la microinstrucción de la dirección 0100, la cual realiza un salto incondicional a la dirección 0010, que es donde inicia el estado uno.

El lector podrá continuar con el seguimiento de este microcódigo y comprobar que efectivamente realiza el diagrama de estados mostrado en la figura 21.12

21.4. Diseño de un semáforo para el cruce de dos calles de un solo sentido, usando firmware.

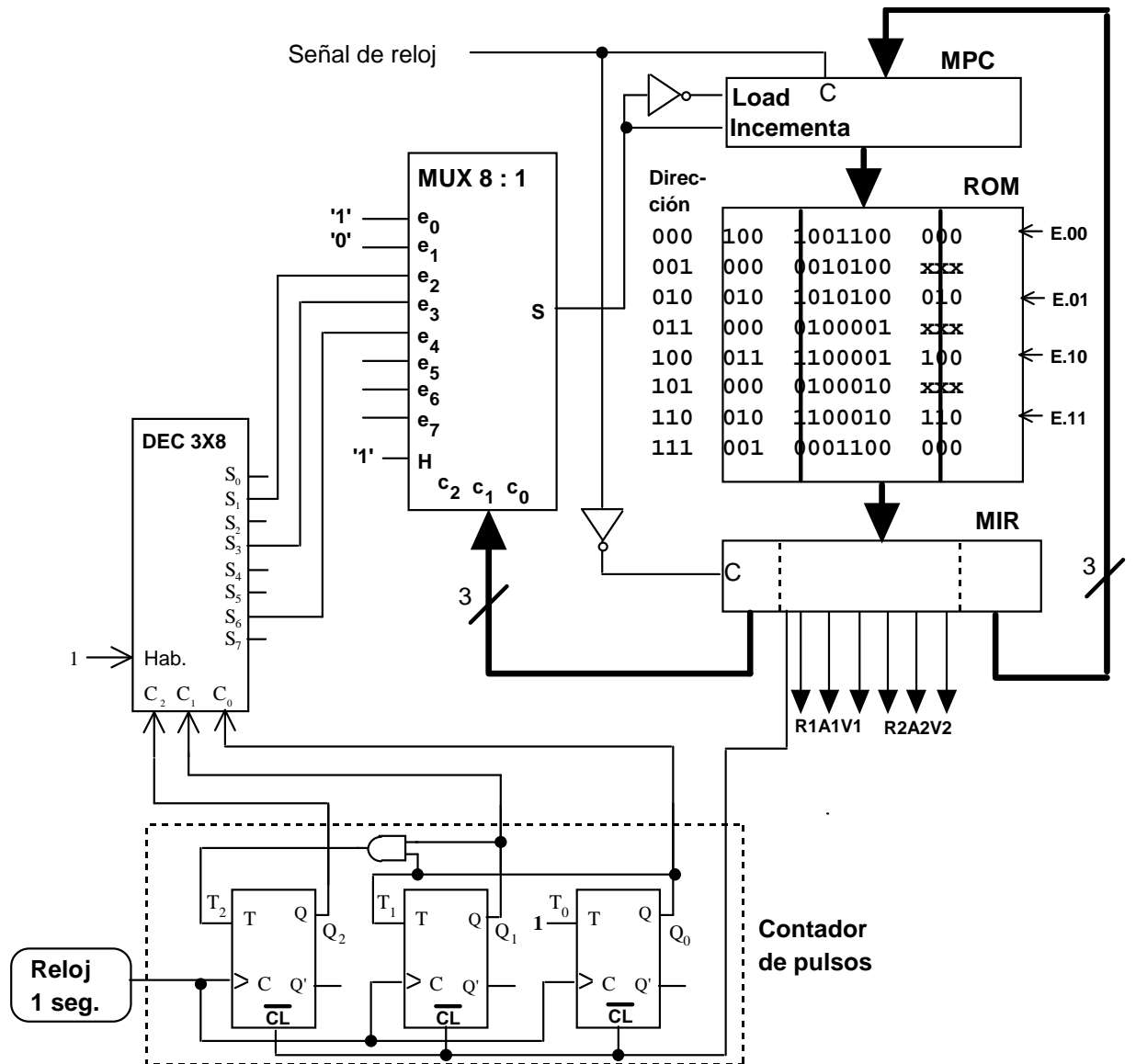


Figura 21.14 Circuito del semáforo.

Las salidas del circuito son las seis funciones para los focos de los dos semáforos y una salida adicional que se utiliza para borrar el contador de pulsos.

El contador de pulsos es el mismo que se utilizó en el ejemplo anterior para identificar cuando han transcurrido 1, 3 ó 6 segundos. Se usa un selector de 8:1 para cuestionar cada una de las 5 condiciones que manejan las microinstrucciones. La condición 0 y 1 se usan igual que en el

ejemplo anterior tiene y las condiciones 2,3 y 4 son usadas para preguntar si ha transcurrido el tiempo de 1, 3 ó 6 segundos respectivamente.

El tamaño del MPC es de tres bits y el tamaño del MIR es de 13 bits. Recuerda que en cada transición positiva del reloj se carga o se incrementa el MPC dependiendo de la condición indicada y en cada transición negativa se carga el MIR.

Al igual que en el ejemplo anterior la microinstrucción definida por E XX es donde inicia el estado XX del diagrama de estados de la figura 21.7.

Par analizar el circuito, suponga que el contador de pulsos tiene el valor de cero; que el MPC tiene cargado un cero, lo cual hace que en las entradas del MIR tengan los valores que se encuentra almacenada en la palabra cero del ROM, y que se genera una transición negativa en la señal de reloj, haciendo que se cargue esta información en el MIR.

En este momento se enciende el verde del semáforo 1 y el rojo del semáforo 2 y no se borra el contador de tiempo. Se cuestiona la entrada 4 del selector, la cual indica que han transcurrido 6 pulsos del reloj de un segundo. Si esta entrada tiene el valor de cero, hará que en la transición positiva del reloj se cargue la dirección cero en el MPC y que en la siguiente transición negativa se ejecute lo mismo. Este ciclo se repite hasta que en la entrada 4 del selector se encuentre un uno, ya que el MPC se incrementará y, al llegar la transición negativa del reloj, se cargará la información que se encuentra en la dirección uno del ROM en el MIR. En este momento el MIR hará que se encienda el amarillo del semáforo 1 y el rojo del semáforo 2 sin borrar el contador de pulsos. Dado que se indica la condición cero, al llegar la siguiente transición positiva en la señal del reloj, el MPC se incrementará.

Al generarse la siguiente transición negativa en la señal de reloj, la microinstrucción que se encuentra en la dirección dos del ROM se cargará en el MIR, haciendo que los semáforos permanezcan en el mismo estado, sin borrar el contador de tiempo. El MPC se incrementará cuando en la entrada 2 del selector se encuentre un uno, haciendo que en las salidas se tengan las correspondientes al estado tres del diagrama de estados de la figura 21.7.

Como el lector podrá comprobar, el análisis del resto de las microinstrucciones es muy semejante. El proceso continuará hasta que se llega a la dirección siete donde, por medio de la condición 1, se realiza un salto incondicional a la dirección cero, iniciando el ciclo de nuevo.

Si la señal de reloj tiene una frecuencia mucho mayor que un segundo, este circuito genera las mismas salidas y las mantendrá por el tiempo que se indicada en el diagrama de estados de la figura 21.7

Se deja como ejercicio el modificar el firmware para que el control de pulsos se haga con un solo flip flop como en el ejemplo anterior, en lugar del contador de pulsos.

21.5 Resumen

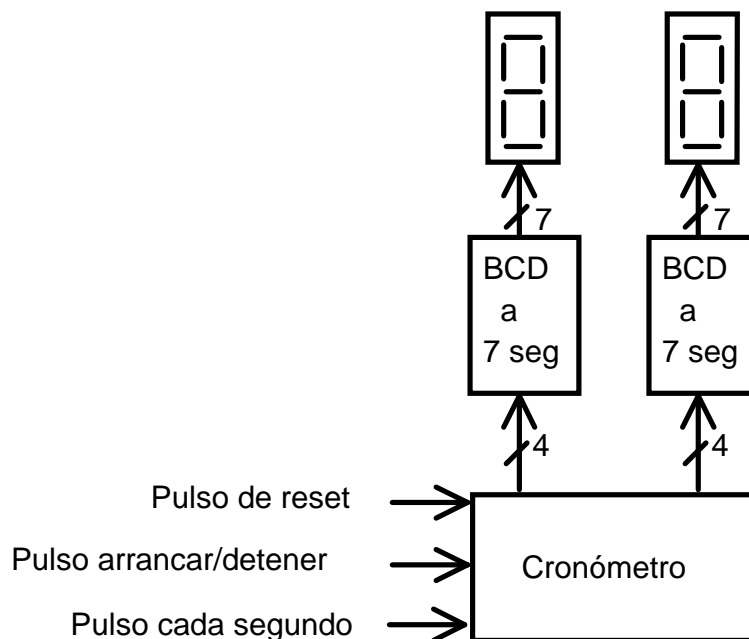
En este capítulo se presenta el diseño de circuitos secuenciales usando contadores para su construcción. Se aplican los conceptos aprendidos en los capítulos anteriores y se refuerza el diseño de circuitos secuenciales.

Se introduce el concepto de microprogramación, el cual es ampliamente usado para la construcción de las unidades de control de las computadoras digitales, dada su gran flexibilidad. En la microprogramación se definen microinstrucciones y los programas almacenados en las memorias ROM, que se conocen generalmente como microcódigo, controlan la secuencia en la que se ejecutan estas microinstrucciones para la construcción de circuitos lógicos secuenciales. Esta forma de construir circuitos secuenciales se conoce como “firmware”.

Para ejemplificar el uso de la microprogramación, se diseñaron y se construyeron dos circuitos secuenciales utilizando estos conceptos, los cuales se aplicarán al diseño de la unidad de control de una computadora digital simple en el capítulo 23.

21.6 Problemas

1. Modifique el reloj digital que se diseña en la sección 21.1 para que ahora incluya una señal de alarma.
2. Construye un cronómetro de 60 segundos, el cual tiene como entradas una señal de reloj de un segundo, una señal de reset y una señal para arrancar y detener como se muestra en la figura.

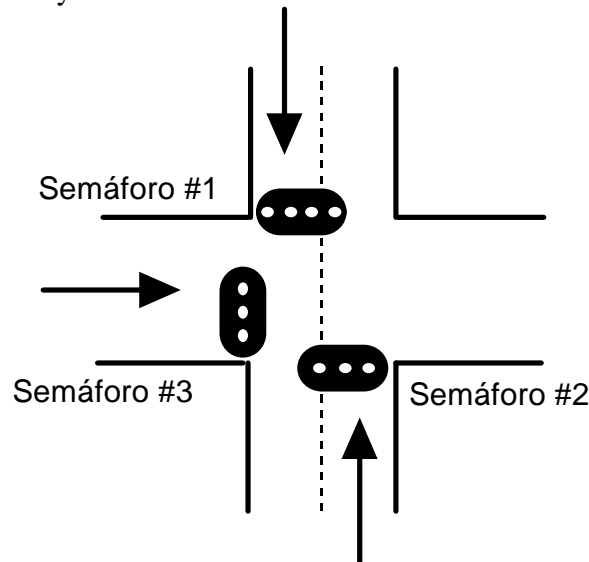


La señal de reset pone a ceros el cronómetro.

La señal de arrancar/detener inicia el conteo si el cronómetro estaba detenido, o lo detiene en el caso contrario.

Si el cronometro llega a 60, inicia otra vez el conteo.

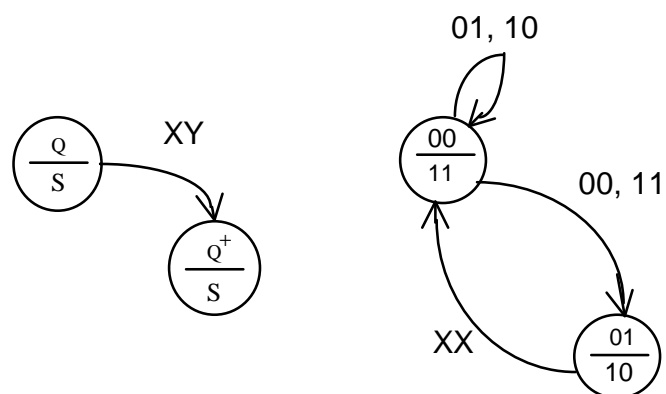
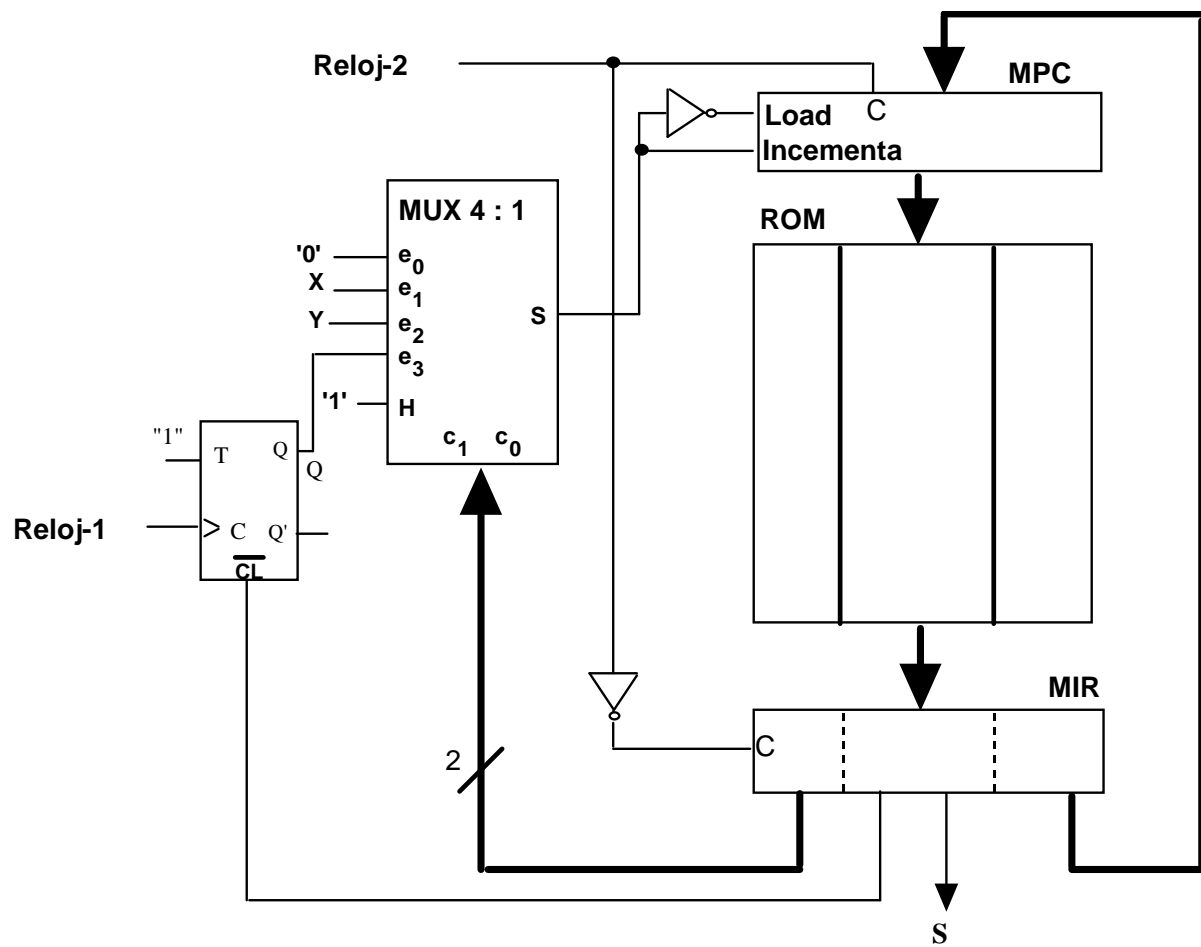
3. Diseña y construya un circuito secuencial para un semáforo de un cruce de dos calles, una de dos sentidos y la otra de un solo sentido como se muestra en la figura.



Llama R1, A1, F1 y V1 a los focos rojo, amarillo, verde para dar vuelta y verde del semáforo número uno, R2, A2 y V2 los focos del semáforo número dos y R3, A3 y V3 los focos del semáforo número tres.

Al igual que un semáforo real, se desea que el tiempo que permanece encendido cada uno de los focos sea diferente. Para este caso daremos un tiempo de 7 segundos al verde del semáforo número uno, de los cuales solo 2 segundos se enciende la luz para dar vuelta (F1) y un segundo después se enciende el verde del semáforo número dos. 4 segundos al verde del semáforo número tres y los amarillos tendrán una duración de 1 segundo

- 4.- Programe el siguiente circuito para que realice el diagrama de estados que se muestra (XX significa cualquier valor). Par cambiar de un estado se requiere que llegue una transición positiva en la señal de reloj-1. La frecuencia del reloj-2 es mucho mayor que la del reloj-1



5. Modificar el frimware del semáforo que se muestra en la figura 21.14 para que el control de pulsos se haga con un solo flip flop, en lugar del contador de pulsos.

CAPITULO 22

ARQUITECTURA BASICA DE UNA COMPUTADORA

22.1. INTRODUCCION.

Las arquitecturas de las computadoras digitales simples pueden clasificarse en arquitecturas de 0, 1, 2 ó 3 direcciones, de acuerdo a los operandos que utilizan en las operaciones aritméticas y lógicas. En forma general, una operación aritmética o lógica puede definirse como:

$$\text{resultado} = \text{operando1} \text{ operación } \text{operando2}$$

Donde la operación puede ser cualquier operación aritmética o lógica que realice la computadora a nivel de instrucción de máquina. Típicamente, todas las computadoras digitales efectúan operaciones de suma, resta y operaciones lógicas. Algunas computadoras tienen también operaciones de multiplicación y división como parte de su repertorio de instrucciones de máquina.

En las arquitecturas de cero dirección, los operandos se toman siempre de una pila y el resultado de la operación se deja en la pila. Por lo tanto, en las instrucciones aritméticas y lógicas de estas máquinas no se especifica ninguna dirección para los operandos ni para el resultado.

En las máquinas que tienen arquitectura de una dirección, uno de los operandos es siempre el registro acumulador y el otro corresponde a un operando en memoria. El resultado se deja en el registro acumulador. Estas máquinas tienen solamente un acumulador y las instrucciones aritméticas y lógicas solamente especifican la dirección en memoria del segundo operando.

Las máquinas que tienen una arquitectura de dos direcciones cuentan con instrucciones aritméticas en las cuales se especifica de donde se van a tomar cada uno de los dos operandos de la instrucción. El resultado se deja en el lugar de donde se tomó el primer operando. Por ejemplo, en una máquina que tenga varios registros acumuladores, se puede especificar como primer operando uno de estos registros acumuladores y, como segundo operando, alguna dirección de memoria. El resultado se deja en el acumulador de donde se toma el primer operando.

Por último, en las máquinas con arquitectura de tres direcciones, se pueden especificar tanto de donde se tomarán los operandos como el lugar donde se va a dejar el resultado. Cualquiera de las direcciones pueden corresponder a una dirección de memoria o un registro del procesador.

En este capítulo se definirá una máquina con arquitectura de una dirección como ejemplo de una arquitectura más real que la presentada en el capítulo 2. En el capítulo 23 se trabajará en el diseño de la unidad de control de esta máquina.

22.2. ARQUITECTURA BASICA.

La computadora que se utilizará como ejemplo en este capítulo usa palabras de 8 bits. La memoria está organizada en bytes y puede tener hasta 65,536 bytes, numerados del 0 al 65,535. Para poder seleccionar cualquier byte de la memoria se necesitan 16 bits.

En la figura 22.1 se presenta la arquitectura de la máquina utilizada en este capítulo. A continuación se describen algunas generalidades de la misma

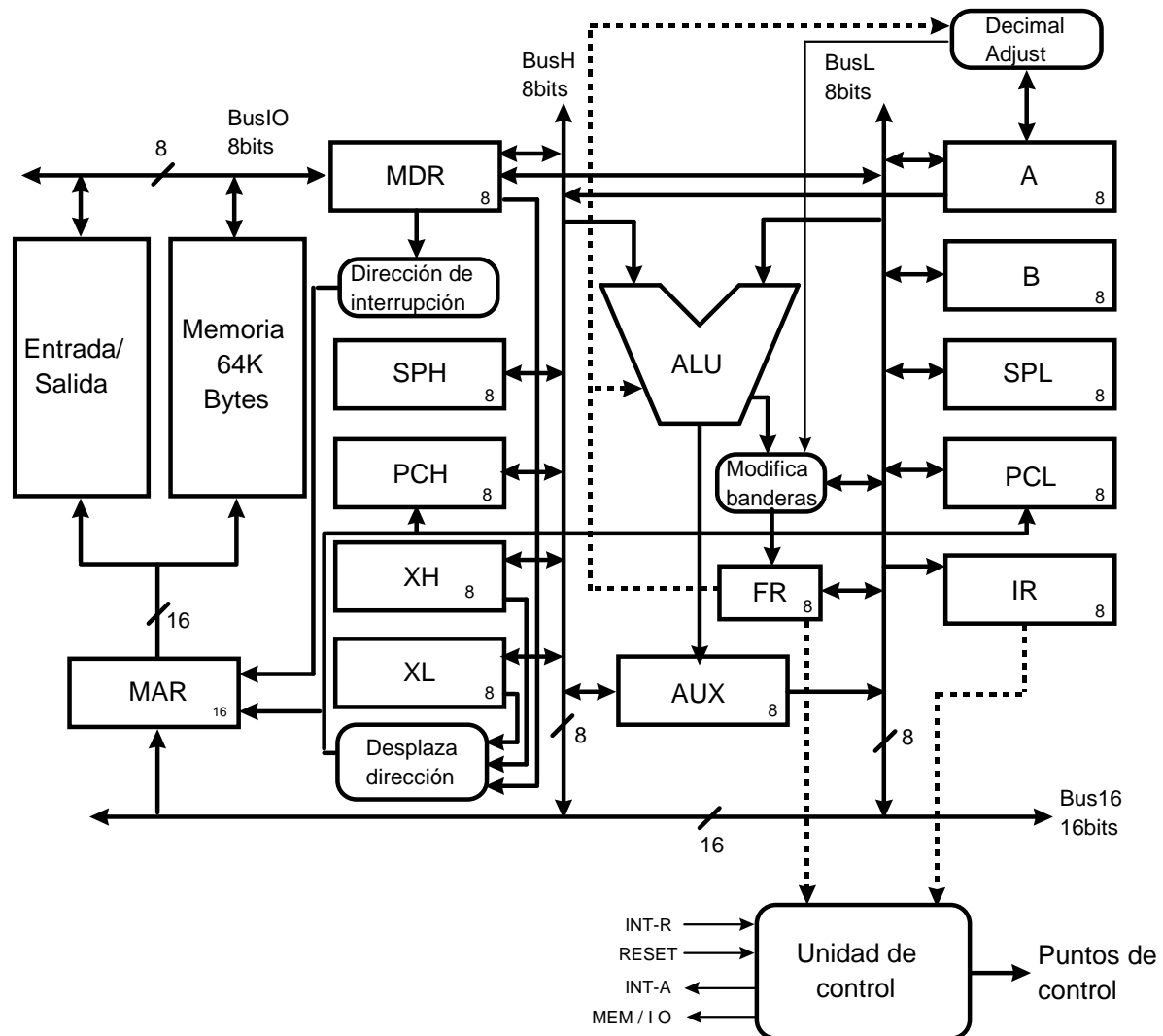


Figura 22.1 Arquitectura de la computadora digital

Esta unidad central de procesamiento (CPU) tiene dos buses externos que se conectan a los registros MDR y MAR. El bus conectado al registro MAR es de 16 bits, lo cual permite que la memoria tenga como máximo 65,546 palabras. El bus conectado al registro MDR es de 8 bits, que corresponden al tamaño de una palabra de memoria.

La máquina también cuenta con instrucciones de entrada y salida y puede hacer referencia a 256 direcciones de dispositivos periféricos, comprendidas de 0 a 255, para lo cual se usan 8 bits. Sin embargo, el número de dispositivos periféricos que se pueden manejar es menor ya que cada dispositivo típicamente utiliza cuatro direcciones para entrada y salida.

Internamente se tienen varios buses. El BusH es de 8 bits, los cuales corresponden a los 8 bits más significativos del Bus16. El BusL también es de 8 bits y éstos corresponden a los 8 bits menos significativos del Bus16.

Los registros que contiene su unidad central de procesamiento se describen a continuación.

22.2.1. Registros de la unidad central de proceso.

- **Acumulador A (A)**
Este es un registro de propósito general de 8 bits que se utiliza para realizar operaciones aritméticas y lógicas.
- **Acumulador B (B)**
Este es un segundo registro de propósito general de 8 bits. También se utiliza para realizar operaciones aritméticas y lógicas.
- **Registro índice (X)**
Este registro se utiliza en algunas instrucciones para generar la dirección efectiva del operando. Tiene 16 bits.
- **Registro apuntador a la pila (SP)**
La función de este registro es mantener la dirección del siguiente lugar de memoria disponible en la pila. Es de 16 bits
- **Registro contador de programa (PC)**
Este registro contiene la dirección de la siguiente instrucción a ejecutar. Tiene 16 bits para poder seleccionar cualquier dirección de memoria.
- **Registro de instrucción (IR)**
Este registro recibe la instrucción que se va a ejecutar y tiene 8 bits.
- **Registro de dirección de memoria (MAR)**
Este registro contiene la dirección de memoria a la cual se hace referencia durante las operaciones de lectura o escritura a memoria o a la dirección de un dispositivo periférico cuando se trata de una operación de entrada/salida. Tiene 16 bits.
- **Registro de datos de memoria (MDR)**
En este registro queda el dato obtenido de la unidad de memoria durante una operación de lectura a memoria o el dato obtenido de un dispositivo periférico durante una operación de entrada y debe contener el dato que se desea guardar en la unidad de memoria durante una

operación de escritura a memoria o el dato a enviar a un periférico durante una operación de salida. Este registro tiene 8 bits

- **Registro de banderas (FR)**

Este registro es de 8 bits y contiene las seis banderas que utiliza esta máquina: I (Interrupt), Z (Zero), N (Negative), C (Carry), H (Half carry) y V (Overflow). El formato del registro es 11IVCHNZ. Nótese que sus dos bits más significativos siempre tienen el valor de 1. La bandera I permite interrupciones cuando está encendida y no se permiten interrupciones cuando está apagada. Las demás banderas reflejan el estado de la última operación que se ejecutó y las modificó. Una bandera que no se había utilizado anteriormente es el Half Carry. Esta bandera se enciende cuando se realiza alguna operación aritmética sobre algún acumulador y se genera un acarreo sobre el bit 4. El bit menos significativo es el 0 y el más significativo es el bit 7. La bandera H se utiliza cuando se realizan operaciones con números en formato BCD, su función se detallará posteriormente. En la descripción de cada instrucción se indica cuales banderas se modifican durante su ciclo de ejecución.

- **Registro auxiliar (AUX)**

Este registro contiene 8 bits y es utilizado internamente por el procesador para realizar algunas operaciones. El registro AUX no está disponible para el programador pero si existen micro operaciones que lo utilizan en la ejecución de las instrucciones.

La unidad aritmética actúa sobre operandos de 8 bits que representan números enteros con signo en representación de complementos a 2. Es posible realizar operaciones aritméticas sobre operandos de más bits, pero se tiene que realizar mediante la programación de rutinas que lo efectúen.

En ciertas ocasiones, los registros de 16 bytes se representarán en dos partes: los 8 bits más significativos, que se identificarán con la letra H, y los 8 bits menos significativos, para los cuales se utilizará la letra L. Por ejemplo, el registro índice (X) se identificará como XH y XL. El registro contador de programa (PC) se denotará como PCH y PCL. Lo mismo se tendrá para el apuntador a la pila, SPH y SPL y para el registro de dirección de memoria MARH y MARL. Cuando se haga referencia a todo el registro completo no se usarán las letras H y L. Por ejemplo, X solamente implica XH y XL.

Esta máquina cuenta con un esquema vectorial de interrupciones, el cual se describe a detalle en la sección 22.6.

22.2.2. Tipos de direccionamiento.

Los tipos de direccionamiento que tiene esta máquina son los siguientes:

Implícito

Estas instrucciones son de un byte. La dirección del operando está implícita en la instrucción.

Inmediato

Estas instrucciones pueden ser de dos o tres bytes, dependiendo del operando necesario. Cuando el operando es de 8 bits, éste se encuentra en el segundo byte de la instrucción en formato de complementos a 2. Obsérvese que el operando es un número entero con signo que está en el rango de -128 a $+127$

En caso de ser un operando de 16 bits, éste se encuentra en el segundo y tercer byte de la instrucción y representa un número entero sin signo en el rango de 0 a 65,535. El segundo byte de la instrucción contiene los 8 bits más significativos y el tercer byte los 8 bits menos significativos del operando.

Directo

Estas instrucciones son de dos bytes de longitud. La dirección del operando se encuentra en la dirección especificada por el segundo byte. Nótese que las direcciones efectivas posibles son de 0 a 255 solamente. Si la instrucción hace referencia a memoria, el segundo byte de la instrucción corresponderá a una dirección de memoria. Cuando la instrucción sea de entrada o salida, el segundo byte representará la dirección de un dispositivo periférico.

La dirección efectiva para este caso es:

$MARH \leftarrow 0$

$MARL \leftarrow \text{dirección (segundo byte de la instrucción)}$

Absoluto

Las instrucciones con este tipo de direccionamiento son de 3 bytes. El primer byte tiene el código de operación y el tipo de direccionamiento mientras que el segundo y tercer bytes tienen la dirección absoluta del operando, la cual representa la dirección efectiva. Con los 16 bits de dirección se puede seleccionar cualquiera de las 65,536 direcciones de memoria. El segundo byte contiene los 8 bits más significativos y el tercero contiene los 8 bits menos significativos de la dirección.

La dirección efectiva en este caso es:

$MARH \leftarrow \text{dirección H (segundo byte de la instrucción)}$

$MARL \leftarrow \text{dirección L (tercer byte de la instrucción)}$

Indexado

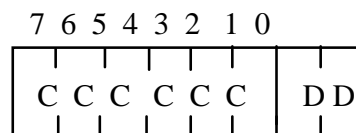
Las instrucciones que usan este tipo de direccionamiento son de 2 bytes. En el primer byte de la instrucción se tiene el código de operación y el tipo de direccionamiento. En el segundo byte se tiene un número entero con signo representado en complementos a 2 que se denomina desplazamiento (este desplazamiento está en el rango de -128 a $+127$). La dirección efectiva se calcula sumando el desplazamiento al contenido del registro índice (X). Para realizar la suma se dispone de un sumador de 16 bits que recibe el registro índice (X) como uno de los operandos y, por otro lado, un desplazamiento de 8 bits. El resultado de la suma constituye la dirección efectiva. Para calcular la dirección efectiva, se propaga el signo del desplazamiento de 8 bits para obtener un desplazamiento de 16 bits y sumarlo al contenido del registro índice.

Por ejemplo, supóngase que en el desplazamiento de 8 bits se tiene el número 33. Este número en binario es 0010 0001. El desplazamiento de 16 bits sería 0000 0000 0010 0001. Si en el desplazamiento de 8 bits se tuviera el número –33, que en binario es 1101 0001, el desplazamiento de 16 bits sería 1111 1111 1101 0001.

22.2.3. Formato de instrucciones

Las instrucciones de esta máquina, como se mencionó anteriormente, pueden ser de 8, 16 ó 24 bits, dependiendo del tipo de direccionamiento que se utilice. El primer byte de las instrucciones contiene el código de operación en los 6 bits más significativos y el tipo de direccionamiento, en aquellas instrucciones que los utilizan, en los dos bits menos significativos. Cuando los bits del tipo de direccionamiento no se usan, se ponen en cero.

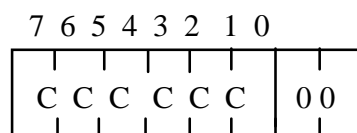
El formato para el primer byte de todas las instrucciones es el siguiente:



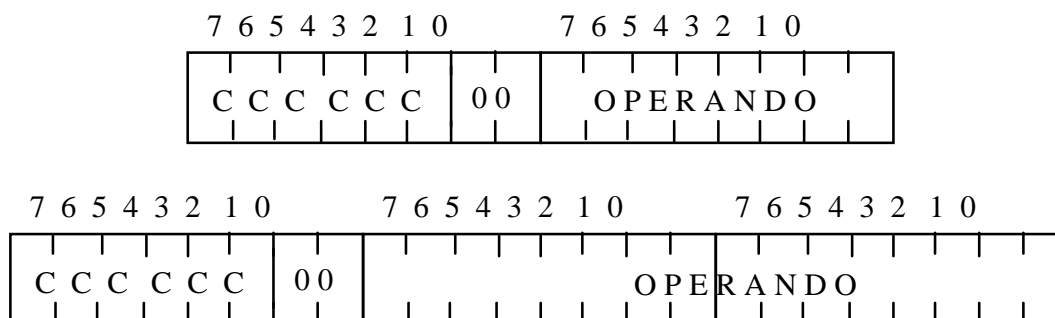
Los 6 bits más significativos tiene el código de operación (CCCCCC), el cual permitiría tener como máximo 64 instrucciones. En los dos bits menos significativos se tiene el tipo de direccionamiento (DD) para aquellas instrucciones que hacen referencia a memoria. El significado de estos bits es el siguiente:

- 00 – direccionamiento inmediato
- 01 – direccionamiento directo
- 10 – direccionamiento absoluto
- 11 – direccionamiento indexado

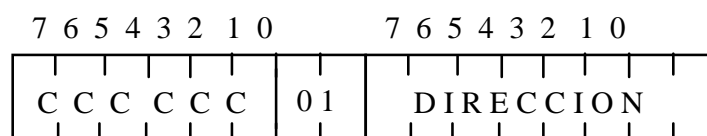
Algunas instrucciones usan direccionamiento implícito y no hacen referencia a memoria. Estas instrucciones son de un byte y los dos bits del tipo de direccionamiento no se utilizan. Estos dos bits se ponen en cero. El formato para estas instrucciones es:



Las instrucciones que usan direccionamiento inmediato tampoco hacen referencia a memoria y pueden ser de dos o tres bytes, dependiendo de si el operando que utilizan es de 8 ó 16 bits. Los dos bits para el tipo de direccionamiento se ponen en 00 para indicar este direccionamiento. Los formatos para estas instrucciones son los siguientes:

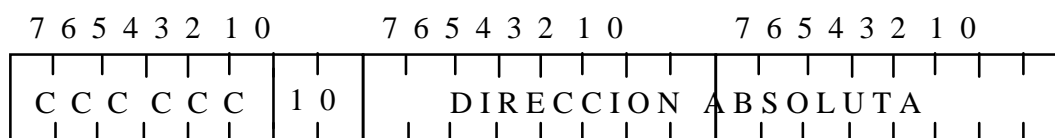


Las instrucciones que hacen referencia a memoria con direccionamiento directo son de dos bytes. El primer byte de estas instrucciones tiene el código de operación y los bits que indican el tipo de direccionamiento que en este caso son 01. El segundo byte de la instrucción contiene la dirección de memoria donde se encuentra el operando. Nótese que solamente se puede hacer referencia a las primeras 256 posiciones de memoria (de 0 a 255) con este tipo de direccionamiento. El formato de las instrucciones es:

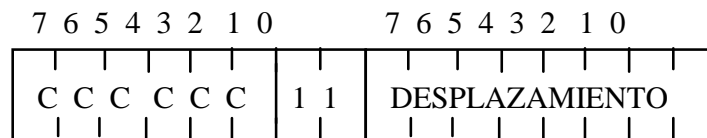


Las dos instrucciones de entrada y salida de esta máquina también utilizan direccionamiento directo, solamente que la dirección a la cual hacen referencia no es una dirección de memoria, sino la dirección del dispositivo correspondiente. Esta dirección puede ser de 0 a 255 ya que solo se usan 8 bits.

Las instrucciones que usan direccionamiento absoluto son de tres bytes. El primer byte contiene el código de operación y el tipo de direccionamiento. En este caso los bits para el tipo de direccionamiento son 10. La dirección absoluta del operando se encuentra en el segundo y tercer bytes de la instrucción, estando los bits más significativos de la dirección en el segundo byte y los menos significativos en el tercero. El formato para estas instrucciones es el siguiente:



Por último, las instrucciones que hacen referencia a memoria con tipo de direccionamiento indexado constan de dos bytes. En el primer byte tienen el código de operación y los dos bits para el tipo de direccionamiento que en este caso son 11. En el segundo byte se encuentra un desplazamiento que represente un número entero con signo en representación de complementos a 2, en el rango de -128 a +127. EL formato para estas instrucciones se muestra a continuación:



22.3. CONJUNTO DE INSTRUCCIONES

Esta máquina incluye instrucciones de carga y almacenamiento de registros, instrucciones aritméticas, instrucciones lógicas, instrucciones para el manejo de una pila en memoria, instrucciones de transferencia de control, instrucciones para transferencia a subrutinas, instrucciones de entrada y salida, e instrucciones para prender o apagar algunos bits del registro de banderas, las cuales incluyen encender o apagar la bandera que habilita las interrupciones.

Cada una de las instrucciones se muestran en forma de resumen en tablas en las cuales se incluye la descripción de la instrucción, su mnemónico, el código de operación (6 bits) y los 8 bits del primer byte de la instrucción para cada uno de los tipos de direccionamiento que permita la instrucción, en hexadecimal. Cuando alguna de estas casillas esté en blanco, implica que ese tipo de direccionamiento no aplica para la instrucción. También se muestran las banderas que se modifican al ejecutar la instrucción.

En el apéndice cuatro se presenta un resumen de las instrucciones de esta computadora.

22.3.1. Instrucciones de carga de registros.

Estas instrucciones permiten cargar operandos en los registros acumuladores, el registro índice y el apuntador a la pila. En la tabla 22.1 se muestra un resumen de estas instrucciones.

Instrucción	Mnemónico	código de operación	Tipo de direccionamiento					Banderas					
			IMP	INM	DIR	ABS	IND	I	V	C	H	N	Z
Load A	LDA	010000		40	41	42	43					X	X
Load B	LDB	010001		44	45	46	47						
Load SP	LDSP	010010		48	49	4A	4B						
Load X	LDX	010011		4C	4D	4E							

Tabla 22.1. Instrucciones de carga de registros.

22.3.2. Instrucciones de almacenamiento de registros.

Este conjunto de instrucciones permite guardar en memoria los registros acumuladores, el registro índice y el registro apuntador a la pila. El resumen de estas instrucciones se muestra en la tabla 22.2.

Instrucción	Mnemónico	código de operación	Tipo de direccionamiento					Banderas					
			IMP	INM	DIR	ABS	IND	I	V	C	H	N	Z
Store A	STA	010100		50	51	52	53						
Store B	STB	010101		54	55	56	57						
Store SP	STSP	010110		58	59	5A	5B						
Store X	STX	010111		5C	5D	5E							

Tabla 22.2. Instrucciones de almacenamiento de registros.

22.3.3. Instrucciones de incremento y decremento de registros.

Las instrucciones dentro de este grupo permiten incrementar o decrementos los registros en una unidad. En la tabla 22.3. se muestra un resumen de estas instrucciones.

Instrucción	Mnemónico	código de operación	Tipo de direccionamiento					Banderas					
			IMP	INM	DIR	ABS	IND	I	V	C	H	N	Z
Increment A	INA	011000	60						X	X	X	X	X
Increment B	INB	011001	64						X	X	X	X	X
Increment X	INX	011010	68										
Decrement A	DCA	011011	6C						X	X	X	X	X
Decrement B	DCB	011100	70						X	X	X	X	X
Decrement X	DCX	011101	74										

Tabla 22.3. Instrucciones de incremento y decremento de registros.

22.3.4. Instrucciones de suma y resta.

Algunas de estas instrucciones operan sobre un dato contenido en uno de los acumuladores y un operando en memoria, dejando el resultado en el acumulador correspondiente. Otras operan sobre los datos contenidos en los dos acumuladores, dejando el resultado en el acumulador A.

Instrucción	Mnemónico	código de operación	Tipo de direccionamiento					Banderas					
			IMP	INM	DIR	ABS	IND	I	V	C	H	N	Z
Add to A	ADDA	000101		14	15	16	17		X	X	X	X	X
Add to B	ADDB	000110		18	19	1A	1B		X	X	X	X	X
Add accumulators	ADDAB	000111	1C						X	X	X	X	X
Add to A with carry	ADDC	001000		20	21	22	23		X	X	X	X	X
Subtract from A	SUBA	001001		24	25	26	27		X	X	X	X	X
Subtract from B	SUBB	001010		28	29	2A	2B		X	X	X	X	X
Subtract accumulators	SUBAB	001011	2C						X	X	X	X	X

Subtract from A with carry	SUBC	001100		30	31	32	33		X	X	X	X	X
----------------------------	------	--------	--	----	----	----	----	--	---	---	---	---	---

Tabla 22.4. Instrucciones de suma y resta.

22.3.5. Instrucciones para el manejo de la pila en memoria.

Esta computadora también permite manipular una pila en memoria. El registro apuntador a la pila indica la dirección de la siguiente posición (byte) libre de la pila en memoria. La pila crece en sentido inverso a como crecen las direcciones de memoria. Cuando se guarda un registro en la pila que tiene 16 bits, primero se guarda el byte menos significativo y luego el byte más significativo. Al obtener este registro de la pila, se obtienen los dos bytes en sentido inverso, es decir, primero se obtiene el byte más significativo y luego el menos significativo. Las instrucciones con que cuenta esta máquina para el manejo de la pila se muestran en la tabla 22.5.

Instrucción	Mnemónico	código de operación	Tipo de direccionamiento					Banderas					
			IMP	INM	DIR	ABS	IND	I	V	C	H	N	Z
Push A	PSHA	011110	78										
Push B	PSHB	011111	7C										
Push X	PSHX	100000	80										
Pop A	POPA	100001	84									X	X
Pop B	POPB	100010	88										
Pop X	POPX	100011	8C										

Tabla 22.5. Instrucciones para el manejo de la pila en memoria.

22.3.6. Instrucciones lógicas y de corrimientos.

Las instrucciones lógicas permiten realizar operaciones de corrimientos y rotaciones sobre el acumulador A, así como operaciones lógicas entre el acumulador A y un operando en memoria o entre los dos acumuladores. Un resumen de estas instrucciones se muestra en la tabla 22.6.

Instrucción	Mnemónico	código de operación	Tipo de direccionamiento					Banderas						
			IMP	INM	DIR	ABS	IND	I	V	C	H	N	Z	
Shift right A arithmetically	SHRAA	100100	90											X
Shift right A logically	SHRAL	100101	94										X	X
Shift left A	SHLA	100110	98										X	X
Rotate left A	ROLA	100111	9C										X	X
Not A	NOTA	101000	A0										X	X
And to A	ANDA	101001		A4	A5	A6	A7						X	X
And accumulators	ANDAB	101010	A8										X	X
Or to A	ORA	101011		AC	AD	AE	AF						X	X
Or accumulators	ORAB	101100	B0										X	X
Exclusive or to A	XORA	101101		B4	B5	B6	B7						X	X

Tabla 22.6. Instrucciones lógicas.

22.3.7. Instrucciones de comparación.

Este juego de instrucciones permiten realizar operaciones de resta entre uno de los acumuladores y un operando en memoria pero el resultado no se almacena en ningún lugar, pero si se modifican las banderas V, C, H, N y Z para reflejar el resultado de dicha operación. Es por esta razón que se denominan instrucciones de comparación ya que permiten comparar dos números sin modificar los acumuladores. También se puede realizar una comparación entre los dos registros acumuladores. Nótese que las banderas modificadas corresponden a el resultado de realizar una operación de resta. La tabla 22.7 contiene el resumen de estas instrucciones.

Instrucción	Mnemónico	código de operación	Tipo de direccionamiento					Banderas					
			IMP	INM	DIR	ABS	IND	I	V	C	H	N	Z
Compare A with memory	CMPA	001101		34	35	36	37		X	X	X	X	X
Compare B with memory	CMPB	001110		38	39	3A	3B		X	X	X	X	X
Compare accumulators	CMPAB	001111	3C						X	X	X	X	X

Tabla 22.7. Instrucciones de comparación.

22.3.8. Instrucciones de transferencia de control.

Las instrucciones en este grupo permiten alterar la secuencia normal de ejecución transfiriendo el control a otra parte del programa. Estas transferencias pueden ser incondicionales o dependientes de ciertas condiciones presentes en el registro de banderas. También se incluye una instrucción para transferir el control a un subprograma y para regresar al punto desde donde se invocó dicho subprograma. Estas últimas dos instrucciones utilizan la pila en memoria para guardar el valor del registro contador de programa al momento de la llamada al subprograma y recuperar su valor para regresar.

Instrucción	Mnemónico	código de operación	Tipo de direccionamiento					Banderas					
			IMP	INM	DIR	ABS	IND	I	V	C	H	N	Z
Jump	JMP	101110			B9	BA	BB						
Jump on zero	JMZ	101111			BD	BE	BF						
Jump on negative	JMN	110000			C1	C2	C3						
Jump on carry	JMC	110001			C5	C6	C7						
Jump on overflow	JMV	110010			C9	CA	CB						
Jump subroutine	JSR	110011			CD	CE	CF						
Return from subroutine	RTN	110100	D0										

Tabla 22.8. Instrucciones de transferencia de control.

22.3.9. Instrucciones de entrada y salida.

Existen dos instrucciones para realizar la entrada y salida de datos a dispositivos periféricos bajo el control de la unidad central de proceso. Las direcciones de los periféricos son de 8 bits (están en el rango de 0 a 255) y son diferentes de las direcciones de memoria. La tabla 22.9 contiene el resumen de estas instrucciones.

Instrucción	Mnemónico	código de operación	Tipo de direccionamiento					Banderas					
			IMP	INM	DIR	ABS	IND	I	V	C	H	N	Z
Input A	INPA	110111			DD							X	X
Output A	OUTA	111000			E1								

Tabla 22.9. Instrucciones de entrada y salida.

22.3.10. Instrucciones para manejo de interrupciones y registro de banderas.

Se tienen instrucciones para habilitar y deshabilitar las interrupciones así como para regresar al punto en que se encontraba la ejecución del programa al momento de presentarse una interrupción y restaurar el estado del procesador. La máquina cuenta con una instrucción que permite generar una interrupción por programa y también se cuenta con dos instrucciones para poder manipular el registro de banderas. A continuación se muestran estas instrucciones en la tabla 22.10.

Instrucción	Mnemónico	código de operación	Tipo de direccionamiento					Banderas					
			IMP	INM	DIR	ABS	IND	I	V	C	H	N	Z
Interrupts on	INTON	111001	E4					X					
Interrupts off	INTOFF	111010	E8					X					
Return from interrupt	RTI	111011	EC					X	X	X	X	X	X
Software interrupt	SWI	111100	F0					X					
Copy flag register to A	CFRA	111101	F4									X	X
Set flag register from A	SFRA	111110	F8					X	X	X	X	X	X

Tabla 22.10. Instrucciones para manejo de interrupciones y registro de banderas.

22.3.11. Instrucciones misceláneas.

Por último, existen algunas instrucciones que permiten borrar los acumuladores, obtener su complemento a la base, intercambiar el contenido de ambos acumuladores y realizar el ajuste

necesario sobre el acumulador A cuando se están sumando números en formato BCD. Estas instrucciones se muestran en la tabla 22.11.

Instrucción	Mnemónico	código de operación	Tipo de direccionamiento					Banderas						
			IMP	INM	DIR	ABS	IND	I	V	C	H	N	Z	
No operation	NOP	000000	00											
Clear A	CLA	000001	04									X	X	
Clear B	CLB	000010	08											
Negate A	NEGA	000011	0C									X	X	
Negate B	NEGB	000100	10											
Swap accumulators	SWPAB	110110	D8									X	X	
Decimal Adjust A	DAA	110101	D4							X	X	X	X	
Halt	HLT	111111	FC											

Tabla 22.11. Instrucciones misceláneas.

22.4. PSEUDO MICRO OPERACIONES DE LA COMPUTADORA.

Las micro operaciones, como se describió en el capítulo 2, son operaciones básicas que se utilizan en la secuencia adecuada para la búsqueda y ejecución de las instrucciones de máquina. Algunas instrucciones de máquina corresponden directamente a una micro operación mientras que otras instrucciones requieren una serie de micro operaciones para su ejecución. En esta sección se utilizan pseudo micro operaciones para posteriormente describir las instrucciones de máquina en base a ellas. Estas pseudo micro operaciones, en la mayoría de los casos, corresponden a micro operaciones de la máquina; sin embargo, en algunos casos, se requieren varias micro operaciones para realizar una pseudo micro operación. En el siguiente capítulo, cuando se diseñe la unidad de control de esta computadora, quedará perfectamente clara la relación de micro operaciones y pseudo micro operaciones.

Esta arquitectura cuenta con las pseudo micro operaciones que se listan a continuación. En la descripción se incluyen las banderas que se modifican, incluyéndolas al final de la pseudo micro operación entre paréntesis. Si no se incluye ninguna bandera, implica que la pseudo micro operación no modifica ninguna de las banderas.

22.4.1 Pseudo Micro operaciones lógicas.

Estas pseudo micro operaciones solamente alteran el contenido de un registro.

- $[A] \leftarrow 0$

Pone el contenido del acumulador A en cero (NZ).

- $[B] \leftarrow 0$

Pone el contenido del acumulador B en cero.

- $[MARH] \leftarrow 0$

Pone el contenido del registro MARH en cero.

- $[SPH] \leftarrow 0$

Pone el contenido del registro SPH en cero.

- $[A] \leftarrow [A]^*$

Obtiene el complemento a la base del acumulador A (NZ).

- $[B] \leftarrow [B]^*$

Obtiene el complemento a la base del acumulador B.

- $[A] \leftarrow \text{Shral}$

Esta pseudo micro operación realiza un corrimiento lógico del acumulador A hacia la derecha. El bit menos significativo se pierde y en el lugar del bit más significativo entra un cero. Esto es equivalente a dividir el acumulador entre 2 suponiendo que tiene un número entero sin signo (NZ).

- $[A] \leftarrow \text{Shraa}$

Esta pseudo micro operación realiza un corrimiento aritmético del acumulador A hacia la derecha. El bit menos significativo se pierde y en el lugar del bit más significativo permanece el valor que tenía antes de la operación. La instrucción divide el contenido del acumulador entre 2 suponiendo que se tiene un número entero con signo en representación de complementos a la base (Z).

- $[A] \leftarrow \text{Shla}$

La pseudo micro operación realiza un corrimiento del acumulador A hacia la izquierda. El bit más significativo se pierde y en el lugar del bit menos significativo se alimenta un cero. La operación es equivalente a multiplicar el contenido del acumulador por 2, pero si ocurre un desborde, las banderas V y C no lo reflejan (NZ).

- $[A] \leftarrow \text{Rola}$

La pseudo micro operación realiza un corrimiento circular del acumulador A hacia la izquierda. El bit más significativo se alimenta en el lugar del bit menos significativo (NZ).

- $[A] \leftarrow \text{Nota}$

Esta pseudo micro operación obtiene el complemento de cada uno de los bit del acumulador A, dejando el resultado en el acumulador A (NZ).

- $[A] \leftarrow [A] \text{ AND } [MDR]$

Realiza la operación AND, bit a bit entre los registros A y MDR dejando el resultado de la operación en el registro A (NZ).

- $[A] \leftarrow [A] \text{ AND } [B]$

Realiza la operación AND, bit a bit entre los registros A y B dejando el resultado de la operación en el registro A (NZ).

• $[A] \leftarrow [A] \text{ OR } [MDR]$

Realiza la operación OR, bit a bit entre los registros A y MDR dejando el resultado de la operación en el registro A (NZ).

• $[A] \leftarrow [A] \text{ OR } [B]$

Realiza la operación OR, bit a bit entre los registros A y B dejando el resultado de la operación en el registro A (NZ).

• $[A] \leftarrow [A] \text{ XOR } [MDR]$

Realiza la operación XOR, bit a bit entre los registros A y MDR dejando el resultado de la operación en el registro A (NZ).

• $[A] \leftarrow [A] \text{ XOR } [B]$

Realiza la operación XOR, bit a bit entre los registros A y B dejando el resultado de la operación en el registro A (NZ).

22.4.2 Pseudo Micro operaciones de incremento de registros.

Este conjunto de pseudo micro operaciones permiten incrementar el contenido de un registro en una unidad.

• $[A] \leftarrow [A] + 1$

Incrementa el contenido del acumulador A en una unidad (CHNZ).

• $[B] \leftarrow [B] + 1$

Incrementa el contenido del acumulador B en una unidad (CHNZ).

• $[PC] \leftarrow [PC] + 1$

Incrementa el contenido del registro contador de programa en una unidad.

• $[X] \leftarrow [X] + 1$

Incrementa el contenido del registro índice en una unidad.

• $[MAR] \leftarrow [MAR] + 1$

Incrementa el contenido del registro de dirección de memoria en una unidad.

• $[SP] \leftarrow [SP] + 1$

Incrementa el contenido del registro apuntador a la pila en una unidad.

22.4.2 Pseudo Micro operaciones de decremento de registros.

Este conjunto de pseudo micro operaciones permiten decrementar el contenido de un registro en una unidad.

• $[A] \leftarrow [A] - 1$

Decrementa el contenido del acumulador A en una unidad (CHNZ).

$$\bullet [B] \leftarrow [B] - 1$$

Decrementa el contenido del acumulador B en una unidad (CHNZ).

$$\bullet [X] \leftarrow [X] - 1$$

Decrementa el contenido del registro índice en una unidad.

$$\bullet [SP] \leftarrow [SP] - 1$$

Decrementa el contenido del registro apuntador a la pila en una unidad.

22.4.3 Pseudo Micro operaciones aritméticas.

$$\bullet [A] \leftarrow [A] + [MDR]$$

Suma los contenidos de A y MDR dejando el resultado en A (CHNZ).

$$\bullet [A] \leftarrow [A] + [MDR] + C$$

Suma los contenidos de A, MDR y la bandera C dejando el resultado en A (CHNZ).

$$\bullet [B] \leftarrow [B] + [MDR]$$

Suma los contenidos de B y MDR dejando el resultado en B (CHNZ).

$$\bullet [A] \leftarrow [A] + [B]$$

Suma los contenidos de A y B dejando el resultado en A (CHNZ).

$$\bullet [A] \leftarrow [A] - [MDR]$$

Resta los contenidos de A y MDR dejando el resultado en A (CHNZ).

$$\bullet [A] \leftarrow [A] - [MDR] - C$$

Resta los contenidos de MDR y la bandera C al contenido de A dejando el resultado en A (CHNZ).

$$\bullet [B] \leftarrow [B] - [MDR]$$

Resta los contenidos de B y MDR dejando el resultado en B (CHNZ).

$$\bullet [MAR] \leftarrow [X] + [MDR]$$

Suma los contenidos de X y MDR dejando el resultado en X.

$$\bullet [MAR] \leftarrow [X] + [MDR]$$

Suma los contenidos del registro índice y el MDR dejando el resultado en X.

$$\bullet [A] \leftarrow [A] - [B]$$

Resta los contenidos de A y B dejando el resultado en A (CHNZ).

$$\bullet [A] - [MDR]$$

Resta el contenido de los registros A y MDR pero no guarda el resultado (CHNZ).

- $[B] - [MDR]$

Resta el contenido de los registros B y MDR pero no guarda el resultado (CHNZ).

- $[A] - [B]$

Resta el contenido de los registros A y B pero no guarda el resultado (CHNZ).

- $[A] \leftarrow Da([A])$

Realiza un ajuste al acumulador A para ponerlo en formato BCD cuando es necesario. Esta micro operación se utiliza cuando se está realizando aritmética BCD. Su operación se describirá posteriormente al discutir la instrucción DAA.

23.4.4 Pseudo Micro operaciones de transferencias entre registros.

Estas pseudo micro operaciones solamente realizan transferencias entre registros. Su operación se detallará solamente cuando no sea muy obvia, pero si se indicará cuales banderas se modifican.

- $[A] \leftarrow [MDR]$ (NZ)

- $[B] \leftarrow [MDR]$

- $[A] \leftrightarrow [B]$ (NZ)

Intercambia los contenidos de los registros A y B.

- $[MDR] \leftrightarrow [PCL]$

Intercambia los contenidos de los registros MDR y PCL (este intercambio solamente se utiliza en la instrucción JSR con direccionamiento absoluto).

- $[MDR] \leftarrow [A]$

- $[MDR] \leftarrow [B]$

- $[MDR] \leftarrow [XL]$

- $[MDR] \leftarrow [AUX]$

- $[MDR] \leftarrow [XH]$

- $[MDR] \leftarrow [PCL]$

- $[MDR] \leftarrow [PCH]$

- $[MAR] \leftarrow [PC]$

- $[\text{MARL}] \leftarrow [\text{AUX}]$
- $[\text{MARH}] \leftarrow 0$
- $[\text{MARH}] \leftarrow [\text{AUX}]$
- $[\text{MARL}] \leftarrow [\text{MDR}]$
- $[\text{MAR}] \leftarrow [\text{SP}]$
- $[\text{MAR}] \leftarrow [\text{X}] + [\text{MDR}]$
- $[\text{MAR}] \leftarrow \text{F000} + 2 * [\text{MDR}]$
- $[\text{MAR}] \leftarrow \text{FFFE}$
- $[\text{SPH}] \leftarrow [\text{MDR}]$
- $[\text{SPL}] \leftarrow [\text{MDR}]$
- $[\text{XH}] \leftarrow [\text{MDR}]$
- $[\text{XL}] \leftarrow [\text{MDR}]$
- $[\text{PCH}] \leftarrow [\text{MDR}]$
- $[\text{PCL}] \leftarrow [\text{MDR}]$
- $[\text{PC}] \leftarrow [\text{X}] + [\text{MDR}]$
- $[\text{IR}] \leftarrow [\text{MDR}]$
- $[\text{A}] \leftarrow [\text{FR}]$
- $[\text{FR}] \leftarrow [\text{A}]$

Recuérdese que los dos bits más significativos del registro de banderas siempre quedan con el valor de 1, independientemente de lo que tengan en el acumulador A.

23.4.5 Pseudo Micro operaciones de lectura y escritura a memoria.

Solamente existen dos pseudo micro operaciones que permiten leer o almacenar datos de la unidad de memoria.

- Mread

Copia el contenido de la posición de memoria especificada por el registro MAR al registro MDR.

- Mwrite

Copia el contenido del registro MDR a la posición de memoria especificada por el registro MAR.

23.4.6 Pseudo Micro operaciones de entrada y salida.

Solamente existen dos micro operaciones que permiten leer o almacenar datos de la unidad de entrada y salida.

- IOread

Copia el contenido del registro del periférico especificado por el registro MAR al registro A.

- IOwrite

Copia el contenido del registro A al registro del periférico especificado por el registro MAR.

23.4.7 Pseudo Micro operaciones para habilitar o deshabilitar las interrupciones.

Solamente existen dos pseudo micro operaciones que permiten habilitar o deshabilitar las interrupciones.

- $I \leftarrow 1$

- $I \leftarrow 0$

23.5. DESCRIPCION DE LAS INSTRUCCIONES EN BASE A LAS PSEUDO MICRO OPERACIONES.

En esta sección se analizarán las pseudo micro operaciones necesarias para realizar el ciclo de búsqueda de las instrucciones (FETCH) así como la secuencia de pseudo micro operaciones involucradas en el ciclo de ejecución (EXECUTE) de algunas instrucciones de máquina típicas ya que muchas tienen un ciclo de ejecución similar. Cuando se muestren dos o más pseudo micro operaciones en el mismo renglón significa que se ejecutan simultáneamente.

• Ciclo de búsqueda

Aún cuando existen instrucciones de 1 byte, 2 bytes y 3 bytes, las pseudo micro operaciones necesarias para el ciclo de búsqueda de las instrucciones son independientes de la longitud de la instrucción. Lo único que se necesita es cargar el primer byte de la instrucción en el registro de instrucción. En la fase de ejecución se generaría la dirección efectiva del operando y se obtendrían los bytes adicionales necesarios.

Las pseudo micro operaciones necesarias para el ciclo de búsqueda son:

```
[ MAR ] ← [ PC ]
Mread y [ PC ] ← [ PC ] + 1
[ IR ] ← [ MDR ]
```

• NOP (No Operation)

El ciclo de ejecución de esta instrucción no implica ninguna pseudo micro operación.

- **CLA (Clear A)**

La única pseudo micro operación para esta instrucción es:

$$[A] \leftarrow 0$$

- **NEGA (Negate A)**

Para negar el valor del acumulador es necesario cambiarle el signo. Dado que la representación de los números enteros se hace en complementos a la base, solamente tiene que obtenerse el complemento a la base del valor que está en el acumulador. La pseudo micro operación involucrada es:

$$[A] \leftarrow [A]^*$$

- **ADDA (Add to A)**

Para direccionamiento inmediato, las pseudo micro operaciones son:

$$\begin{aligned} [MAR] &\leftarrow [PC] \\ \text{Mread } y [PC] &\leftarrow [PC] + 1 \\ [A] &\leftarrow [A] + [MDR] \end{aligned}$$

Para direccionamiento directo se tienen las siguientes pseudo micro operaciones:

$$\begin{aligned} [MAR] &\leftarrow [PC] \\ \text{Mread } y [PC] &\leftarrow [PC] + 1 \\ [MARH] &\leftarrow 0 \text{ y } [MARL] \leftarrow [MDR] \\ \text{Mread} \\ [A] &\leftarrow [A] + [MDR] \end{aligned}$$

Las pseudo micro operaciones involucradas en direccionamiento absoluto son:

$$\begin{aligned} [MAR] &\leftarrow [PC] \\ \text{Mread } y [PC] &\leftarrow [PC] + 1 \\ [AUX] &\leftarrow [MDR] \\ [MAR] &\leftarrow [PC] \\ \text{Mread } y [PC] &\leftarrow [PC] + 1 \\ [MARH] &\leftarrow [AUX] \text{ y } [MARL] \leftarrow [MDR] \\ \text{Mread} \\ [A] &\leftarrow [A] + [MDR] \end{aligned}$$

Con direccionamiento indexado, las pseudo micro operaciones son:

```
[ MAR ] ← [ PC ]  
Mread y [ PC ] ← [ PC ] + 1  
[ MAR ] ← [ X ] + [ MDR ]  
Mread  
[ A ] ← [ A ] + [ MDR ]
```

• **CMPA (Compare A with memory)**

Esta instrucción realiza una resta entre el acumulador y el operando especificado pero el resultado no se conserva, solamente se modifican las banderas correspondientes.

Para direccionamiento inmediato se tiene las pseudo micro operaciones siguientes:

```
[ MAR ] ← [ PC ]  
Mread y [ PC ] ← [ PC ] + 1  
[ A ] ← [ MDR ]
```

EL direccionamiento directo involucra las siguientes pseudo micro operaciones:

```
[ MAR ] ← [ PC ]  
Mread y [ PC ] ← [ PC ] + 1  
[ MARH ] ← 0 y [ MARL ] ← [ MDR ]  
Mread  
[ A ] ← [ MDR ]
```

Las pseudo micro operaciones necesarias para direccionamiento absoluto son:

```
[ MAR ] ← [ PC ]  
Mread y [ PC ] ← [ PC ] + 1  
[ AUX ] ← [ MDR ]  
[ MAR ] ← [ PC ]  
Mread y [ PC ] ← [ PC ] + 1  
[ MARH ] ← [ AUX ] y [ MARL ] ← [ MDR ]  
Mread  
[ A ] ← [ MDR ]
```

Con direccionamiento indexado, las pseudo micro operaciones son:

```
[ MAR ] ← [ PC ]  
Mread y [ PC ] ← [ PC ] + 1  
[ MAR ] ← [ X ] + [ MDR ]  
Mread  
[ A ] ← [ MDR ]
```

• LDA (Load A)

Para direccionamiento inmediato, las micro pseudo operaciones son:

$$\begin{aligned} [MAR] &\leftarrow [PC] \\ \text{Mread y } [PC] &\leftarrow [PC] + 1 \\ [A] &\leftarrow [MDR] \end{aligned}$$

Para direccionamiento directo se tienen las siguientes pseudo micro operaciones:

$$\begin{aligned} [MAR] &\leftarrow [PC] \\ \text{Mread y } [PC] &\leftarrow [PC] + 1 \\ [MARH] &\leftarrow 0 \text{ y } [MARL] \leftarrow [MDR] \\ \text{Mread} \\ [A] &\leftarrow [MDR] \end{aligned}$$

Las pseudo micro operaciones involucradas en direccionamiento absoluto son:

$$\begin{aligned} [MAR] &\leftarrow [PC] \\ \text{Mread y } [PC] &\leftarrow [PC] + 1 \\ [AUX] &\leftarrow [MDR] \\ [MAR] &\leftarrow [PC] \\ \text{Mread y } [PC] &\leftarrow [PC] + 1 \\ [MARH] &\leftarrow [AUX] \text{ y } [MARL] \leftarrow [MDR] \\ \text{Mread} \\ [A] &\leftarrow [MDR] \end{aligned}$$

Las pseudo micro operaciones necesarias para direccionamiento indexado son:

$$\begin{aligned} [MAR] &\leftarrow [PC] \\ \text{Mread y } [PC] &\leftarrow [PC] + 1 \\ [MAR] &\leftarrow [X] + [MDR] \\ \text{Mread} \\ [A] &\leftarrow [MDR] \end{aligned}$$

• LDSP (Load Stack Pointer)

El registro apuntador a la pila es de 16 bits y su carga implica más micro operaciones que la carga de algún acumulador.

Con direccionamiento inmediato se tiene un operando de 16 bits y las pseudo micro operaciones son:

$$[MAR] \leftarrow [PC]$$


```

Mread y [ PC ]  $\leftarrow$  [ PC ] + 1
[ SPH ]  $\leftarrow$  [ MDR ]
[ MAR ]  $\leftarrow$  [ PC ]
Mread y [ PC ]  $\leftarrow$  [ PC ] + 1
[ SPL ]  $\leftarrow$  [ MDR ]

```

La instrucción con direccionamiento directo implica una dirección de 8 bits para obtener el operando, pero la dirección que se carga en el registro apuntador a la pila es de 16 bits. Este caso comprende las siguientes pseudo micro operaciones:

```

[ MAR ]  $\leftarrow$  [ PC ]
Mread y [ PC ]  $\leftarrow$  [ PC ] + 1
[ MARL ]  $\leftarrow$  [ MDR ] y [ MARH ]  $\leftarrow$  0
Mread
[ MAR ]  $\leftarrow$  [ MAR ] + 1
[ SPH ]  $\leftarrow$  [ MDR ]
Mread
[ SPHL ]  $\leftarrow$  [ MDR ]

```

Para direccionamiento absoluto tanto la dirección donde se encuentra el dato (dirección) a cargar en el registro apuntador a la pila como la dirección misma son de 16 bits. El conjunto de pseudo micro operaciones necesarias en este caso se muestra a continuación:

```

[ MAR ]  $\leftarrow$  [ PC ]
Mread y [ PC ]  $\leftarrow$  [ PC ] + 1
[ AUX ]  $\leftarrow$  [ MDR ]
[ MAR ]  $\leftarrow$  [ PC ]
Mread y [ PC ]  $\leftarrow$  [ PC ] + 1
[ MARH ]  $\leftarrow$  [ AUX ] y [ MARL ]  $\leftarrow$  [ MDR ]
Mread
[ MAR ]  $\leftarrow$  [ MAR ] + 1
[ SPH ]  $\leftarrow$  [ MDR ]
Mread
[ SPHL ]  $\leftarrow$  [ MDR ]

```

Con direccionamiento indexado se necesitan las siguientes pseudo micro operaciones:

```

[ MAR ]  $\leftarrow$  [ PC ]
Mread y [ PC ]  $\leftarrow$  [ PC ] + 1
[ MAR ]  $\leftarrow$  [ X ] + [ MDR ]
Mread
[ SPH ]  $\leftarrow$  [ MDR ] y [ MAR ]  $\leftarrow$  [ MAR ] + 1
Mread
[ SPL ]  $\leftarrow$  [ MDR ]

```

- **STA (Store A)**

Para direccionamiento directo se tienen las siguientes pseudo micro operaciones:

```
[ MAR ] ← [ PC ]  
Mread  
[ MARH ] ← 0 y [ MARL ] ← [ MDR ]  
[ MDR ] ← [ A ]  
Mwrite
```

Las pseudo micro operaciones necesarias con direccionamiento absoluto son:

```
[ MAR ] ← [ PC ]  
Mread y [ PC ] ← [ PC ] + 1  
[ AUX ] ← [ MDR ]  
[ MAR ] ← [ PC ]  
Mread y [ PC ] ← [ PC ] + 1  
[ MARH ] ← [ AUX ] y [ MARL ] ← [ MDR ]  
[ MDR ] ← [ A ]  
Mwrite
```

Las pseudo micro operaciones necesarias para direccionamiento indexado son:

```
[ MAR ] ← [ PC ]  
Mread y [ PC ] ← [ PC ] + 1  
[ MAR ] ← [ X ] + [ MDR ] y [ MDR ] ← [ A ]  
Mwrite
```

- **STSP (Store Stack Pointer)**

En este caso, es necesario guardar en la memoria 16 bits. Con direccionamiento directo se tienen las siguientes pseudo micro operaciones:

```
[ MAR ] ← [ PC ]  
Mread  
[ MARH ] ← 0 y [ MARL ] ← [ MDR ]  
[ MDR ] ← [ SPH ]  
Mwrite  
[ MAR ] ← [ MAR ] + 1 y [ MDR ] ← [ SPL ]  
Mwrite
```

Las pseudo micro operaciones involucradas para direccionamiento absoluto son:

```
[ MAR ] ← [ PC ]  
Mread y [ PC ] ← [ PC ] + 1
```

```

[ AUX ] ← [ MDR ]
[ MAR ] ← [ PC ]
Mread y [ PC ] ← [ PC ] + 1
[ MARH ] ← [ AUX ] y [ MARL ] ← [ MDR ]
[ MDR ] ← [ SPH ]
Mwrite
[ MAR ] ← [ MAR ] + 1 y [ MDR ] ← [ SPL ]
Mwrite

```

Las pseudo micro operaciones necesarias para direccionamiento indexado son:

```

[ MAR ] ← [ PC ]
Mread y [ PC ] ← [ PC ] + 1
[ MAR ] ← [ X ] + [ MDR ] y [ MDR ] ← [ SPH ]
Mwrite
[ MAR ] ← [ MAR ] + 1
[ MDR ] ← [ SPL ]
Mwrite

```

• INPA (Input A)

Esta instrucción solamente tiene direccionamiento directo. Las pseudo micro operaciones necesarias son las siguientes:

```

[ MAR ] ← [ PC ]
Mread y [ PC ] ← [ PC ] + 1
[ MARH ] ← 0 y [ MARL ] ← [ MDR ]
IOread
[ A ] ← [ MDR ]

```

• OUTA (Output A)

Las pseudo micro operaciones que se realizan para esta instrucción, la cual solamente tiene direccionamiento directo, son:

```

[ MAR ] ← [ PC ]
Mread y [ PC ] ← [ PC ] + 1
[ MAR0 ] ← 0 y [ MARL ] ← [ MDR ]
[ MDR ] ← [ A ]
IOWrite

```

• INA (Increment A)

Esta instrucción solamente requiere una pseudo micro operación:

$$[A] \leftarrow [A] + 1$$

• PSHA (Push A)

Las pseudo micro operaciones involucradas son:

$$\begin{aligned} [MAR] &\leftarrow [SP] \\ [MDR] &\leftarrow [A] \\ \text{Mwrite } y \quad [SP] &\leftarrow [SP] - 1 \end{aligned}$$

• PSHX (Push Index Register)

Esta instrucción guarda en la pila un dato de 16 bits, por lo tanto, el apuntador a la pila debe disminuirse en dos unidades al ejecutar la instrucción. Para realizarla se deben llevar a cabo las siguientes pseudo micro operaciones:

$$\begin{aligned} [MAR] &\leftarrow [SP] \\ [MDR] &\leftarrow [XL] \\ \text{Mwrite } y \quad [SP] &\leftarrow [SP] - 1 \\ [MAR] &\leftarrow [SP] \\ [MDR] &\leftarrow [XH] \\ \text{Mwrite } y \quad [SP] &\leftarrow [SP] - 1 \end{aligned}$$

• POPA (Pop A)

Las pseudo micro operaciones involucradas son:

$$\begin{aligned} [SP] &\leftarrow [SP] + 1 \\ [MAR] &\leftarrow [SP] \\ \text{Mread} \\ [A] &\leftarrow [MDR] \end{aligned}$$

• POPX (Pop Index Register)

Esta instrucción obtiene un dato de 16 bits de la pila y es necesario realizar la siguiente secuencia de pseudo micro operaciones para ejecutarla:

$$\begin{aligned} [SP] &\leftarrow [SP] + 1 \\ [MAR] &\leftarrow [SP] \\ \text{Mread} \\ [XH] &\leftarrow [MDR] \quad y \quad [SP] \leftarrow [SP] + 1 \\ [MAR] &\leftarrow [SP] \\ \text{Mread} \\ [XL] &\leftarrow [MDR] \end{aligned}$$

- **SHRAL (Shift Right A Logically)**

La pseudo micro operación que se realiza es:

$$[A] \leftarrow \text{Shral}$$

- **SHRAA (Shift Right A Arithmetically)**

La única pseudo micro operación que se ejecuta es:

$$[A] \leftarrow \text{Shraa}$$

- **SHLA (Shift Left A)**

La pseudo micro operación necesaria es:

$$[A] \leftarrow \text{Shla}$$

- **ROLA (Rotate Left A)**

La pseudo micro operación ejecutada es:

$$[A] \leftarrow \text{Rola}$$

- **NOTA (Not A)**

La única pseudo micro operación necesaria es:

$$[A] \leftarrow \text{Nota}$$

- **ANDA (And to A)**

Para direccionamiento inmediato, las micro pseudo operaciones son:

$$\begin{aligned} [\text{MAR}] &\leftarrow [\text{PC}] \\ \text{Mread y } [\text{PC}] &\leftarrow [\text{PC}] + 1 \\ [A] &\leftarrow [A] \text{ AND } [\text{MDR}] \end{aligned}$$

Para direccionamiento directo se tienen las siguientes pseudo micro operaciones:

$$\begin{aligned} [\text{MAR}] &\leftarrow [\text{PC}] \\ \text{Mread y } [\text{PC}] &\leftarrow [\text{PC}] + 1 \\ [\text{MARH}] &\leftarrow 0 \text{ y } [\text{MARL}] \leftarrow [\text{MDR}] \\ \text{Mread} & \\ [A] &\leftarrow [A] \text{ AND } [\text{MDR}] \end{aligned}$$

Las pseudo micro operaciones involucradas en direccionamiento absoluto son:

```
[ MAR ] ← [ PC ]
Mread y [ PC ] ← [ PC ] + 1
[ AUX ] ← [ MDR ]
[ MAR ] ← [ PC ]
Mread y [ PC ] ← [ PC ] + 1
[ MARH ] ← [ AUX ] y [ MARL ] ← [ MDR ]
Mread
[ A ] ← [ A ] AND [ MDR ]
```

Con direccionamiento indexado, las pseudo micro operaciones son:

```
[ MAR ] ← [ PC ]
Mread y [ PC ] ← [ PC ] + 1
[ MAR ] ← [ X ] + [ MDR ]
Mread
[ A ] ← [ A ] AND [ MDR ]
```

• **JMP (Jump)**

La instrucción de transferencia incondicional con direccionamiento directo implica las siguientes pseudo micro operaciones:

```
[ MAR ] ← [ PC ]
Mread
[ PCH ] ← 0 y [ PCL ] ← [ MDR ]
```

Con direccionamiento absoluto se requiere la siguiente secuencia de pseudo micro operaciones:

```
[ MAR ] ← [ PC ]
Mread y [ PC ] ← [ PC ] + 1
[ AUX ] ← [ MDR ]
[ MAR ] ← [ PC ]
Mread
[ PCH ] ← [ AUX ] y [ PCL ] ← [ MDR ]
```

La secuencia de pseudo micro operaciones con direccionamiento indexado es:

```
[ MAR ] ← [ PC ]
Mread
[ PC ] ← [ X ] + [ MDR ]
```

• **JMZ (Jump on Zero)**

Para ejecutar esta instrucción de transferencia condicional cuando la bandera Z está apagada no se requiere realizar ninguna pseudo micro operación.

Cuando la bandera Z bandera está encendida se requiere realizar las siguientes pseudo micro operaciones con direccionamiento directo:

```
[ MAR ] ← [ PC ]  
Mread  
[ PCH ] ← 0 y [ PCL ] ← [ MDR ]
```

Si la bandera Z está encendida y se tiene direccionamiento directo, se necesita la siguiente secuencia de pseudo micro operaciones:

```
[ MAR ] ← [ PC ]  
Mread y [ PC ] ← [ PC ] + 1  
[ AUX ] ← [ MDR ]  
[ MAR ] ← [ PC ]  
Mread  
[ PCH ] ← [ AUX ] y [ PCL ] ← [ MDR ]
```

La secuencia de pseudo micro operaciones con direccionamiento indexado cuando la bandera Z está encendida es:

```
[ MAR ] ← [ PC ]  
Mread  
[ PC ] ← [ X ] + [ MDR ]
```

•JSR (Jump Subroutine)

Esta instrucción permite transferir el control de ejecución a un subprograma para posteriormente regresarlo a la siguiente instrucción después de la llamada al subprograma. Es necesario guardar en la pila el valor del registro contador de programa para poder regresar.

Si se tiene direccionamiento directo se deben realizar las siguientes pseudo micro operaciones:

```
[ MAR ] ← [ PC ]  
Mread y [ PC ] ← [ PC ] + 1  
[ AUX ] ← [ MDR ]  
[ MAR ] ← [ SP ]  
[ MDR ] ← [ PCL ]  
Mwrite y [ SP ] ← [ SP ] -1  
[ MAR ] ← [ SP ]  
[ MDR ] ← [ PCH ]  
Mwrite y [ SP ] ← [ SP ] -1  
[ PCH ] ← 0 y [ PCL ] ← [ AUX ]
```

Con direccionamiento absoluto se deben realizar las siguientes pseudo micro operaciones:

```
[ MAR ] ← [ PC ]  
Mread y [ PC ] ← [ PC ] + 1  
[ AUX ] ← [ MDR ]  
[ MAR ] ← [ PC ]  
Mread y [ PC ] ← [ PC ] + 1  
[ PCL ] ↔ [ MDR ]  
[ MAR ] ← [ SP ]  
Mwrite y [ SP ] ← [ SP ] - 1  
[ MAR ] ← [ SP ]  
[ MDR ] ← [ PCH ]  
Mwrite y [ SP ] ← [ SP ] - 1  
[ PCH ] ← [ AUX ]
```

Con direccionamiento indexado se deben realizar las siguientes pseudo micro operaciones:

```
[ MAR ] ← [ PC ]  
Mread y [ PC ] ← [ PC ] + 1  
[ AUX ] ← [ MDR ]  
[ MDR ] ← [ PCL ]  
[ MAR ] ← [ SP ]  
Mwrite y [ SP ] ← [ SP ] - 1  
[ MAR ] ← [ SP ]  
[ MDR ] ← [ PCH ]  
Mwrite y [ SP ] ← [ SP ] - 1  
[ MDR ] ← [ AUX ]  
[ PC ] ← [ X ] + [ MDR ]
```

• RTN (Return from Subroutine)

Ahora es necesario obtener la dirección de retorno de la pila. Esta instrucción solamente tiene direccionamiento implícito. Para ejecutarla se realizan las siguientes pseudo micro operaciones:

```
[ SP ] ← [ SP ] + 1  
[ MAR ] ← [ SP ]  
Mread  
[ PCH ] ← [ MDR ] y [ SP ] ← [ SP ] + 1  
[ MAR ] ← [ SP ]  
Mread  
[ PCL ] ← [ MDR ]
```

• DAA (Decimal Adjust A)

Esta instrucción efectúa el ajuste necesario para realizar operaciones aritméticas sobre números en formato BCD. Antes de definir la pseudo micro operación para esta instrucción se darán ejemplos de su operación. Se recomienda revisar la sección 9.3 para entender la instrucción DAA.

Supóngase que en el acumulador A se tiene el número BCD 25 en formato BCD (0010 0101) y que en el acumulador B se tiene el número BCD 32 en el mismo formato (0011 0010). Si se efectúa la suma de los acumuladores A y B, dejando el resultado en A mediante la instrucción ADDAB (Add accumulators), el acumulador A tendría el valor 0101 0111 que es el resultado de sumarlos como números binarios. Las banderas involucradas serían V=0, C=0, H=0, N=0 y Z=0. En este caso el valor contenido en el acumulador A está correcto de acuerdo al formato BCD y no están encendidas las banderas H y C. Este valor representa el número 57 en decimal que es el resultado correcto de la operación. La operación DAA no haría nada en esta ocasión. Para operaciones en BCD las banderas V, N y Z si se modifican al realizar la suma de los acumuladores pero no se utilizan para hacer el ajuste decimal del acumulador A (DAA).

Supóngase ahora que el acumulador A tiene el mismo valor de 25 pero el acumulador B tiene 67 (0110 0111 en formato BCD). Si se suman ambos acumuladores directamente mediante la instrucción ADDAB, el acumulador A tendría como resultado 1000 1100 que no corresponde a ningún número en formato BCD. Las banderas H y C quedarían igual que en el ejemplo anterior. Para dejar el acumulador en el formato adecuado se debe ejecutar la instrucción DAA. Esta instrucción, mediante un circuito especial, analiza primeramente si la bandera H está encendida, lo cual no es cierto. Entonces se procede a analizar los cuatro bits menos significativo del acumulador A (1100) para ver si corresponden a un número BCD. En este caso los cuatro bits no representan un número BCD y es necesario realizar un ajuste. Dicho ajuste se hace sumando el número binario 0110 a estos cuatro bits lo cual da como resultado 0010 y se genera un acarreo fuera de los cuatro bits, el cual se suma a los cuatro bits más significativos del acumulador A, dado que representa un acarreo decimal. Se obtiene 1001 en estos bits al realizar la suma y no se genera un acarreo fuera de estos cuatro bits. Una vez realizada la suma, se procede a analizar los cuatro bits más significativos del acumulador A. Primero se ve si está encendida la bandera C, que en este caso está en cero. Posteriormente se analiza si el número 1001 corresponde a un número BCD, lo cual es cierto y no se realiza ninguna corrección. El resultado final obtenido en A es 1001 0010 y representa el número 92 en BCD, siendo este el resultado correcto (25 + 67) y las banderas C y H quedan en cero.

Ahora supóngase que el acumulador A contiene 0010 1001 (29 en BCD) y el acumulador B contiene 0100 1000 (48 en BCD). Al sumar los dos acumuladores se obtendría 0111 0001 y se encendería la bandera H ya que hubo un acarreo sobre el bit 4. El valor que existe en el acumulador A puede interpretarse como un número BCD que corresponde al 71 en decimal; sin embargo, este resultado no es correcto ya que la suma debe ser 77. Al ejecutar la instrucción DAA se analiza primero la bandera H, la cual ahora si está encendida. Por consiguiente, se procede a realizar la corrección sumando 0110 a los cuatro bits menos significativos del acumulador, obteniéndose 0111 en estas cuatro posiciones sin apagar la bandera H. Después, dado que la bandera H estaba encendida, no se necesita sumar un 1 a los cuatro bits más significativos, pero si se deben analizar para determinar si estos cuatro bits (0111) corresponden a un número BCD. Primero se analiza la bandera C, la cual no está encendida y se procede a

revisar los cuatro bits más significativos de A. Al analizarlos, se ve que si corresponden a un número BCD y no se necesita realizar corrección alguna, obteniéndose como resultado final 0111 0111 en formato BCD, que corresponde al 77 en decimal y representa el resultado correcto.

Suponiendo ahora que el acumulador A tiene el número BCD 0011 0100, que representa el 34 en decimal y el acumulador B tiene el número BCD 0110 1000, que corresponde al 68 en decimal. Al sumar ambos acumuladores se obtiene 1001 1100 y no se encienden ninguna de las banderas C ni H. Al realizar la instrucción DAA, se analiza primero la bandera H, la cual está apagada y luego se analizan los cuatro bits menos significativos de A que son 1100 y no corresponden a ningún número en BCD. Se procede a realizar la corrección sumando 0110 a estos cuatro bits y obteniéndose como resultado 0010 y un medio acarreo que se suma a los cuatro bits más significativos de A, obteniéndose 1010 en estos bits. Ahora se analiza la bandera C que está apagada y se procede entonces a ver si los cuatro bits más significativos corresponden a un número BCD. Estos cuatro bits tampoco corresponden a ningún número BCD y es necesario realizar otra corrección sumándoles 0110. Se obtiene 0000 y un acarreo que enciende en la bandera C. El resultado final en el acumulador A es 0000 0010 que representa el 2 en decimal pero se tiene un acarreo en la bandera C que corresponde a un acarreo decimal que representa el 100, obteniéndose el resultado correcto.

Finalmente, se verá el caso en que el acumulador A tiene el número BCD 98 (1001 1000) y el acumulador B contiene 79 (0111 1001). Al sumar los acumuladores se obtiene 0001 0001 y tanto la bandera C como la H está encendidas. Al realizar la instrucción DAA, se analiza primero la bandera H, que está encendida y se procede a realizar la corrección de los cuatro bits menos significativos de A sumándoles 0110 sin apagar la bandera H. El resultado obtenido en estos bits es 0111. Ahora se analiza la bandera C, que también está encendida por lo que se procede a realizar la corrección de los cuatro bits más significativos sumándole también 0110, lo cual da como resultado 0111, pero la bandera C no se apaga. El resultado final en el acumulador A es 0111 0111 que corresponde al número decimal 77 y se tiene un acarreo decimal (la bandera C) que representa 100 y el resultado es correcto.

En resumen, la instrucción DAA realiza lo siguiente:

- Analiza primero la bandera H. Si está encendida procede a realizar la corrección de los cuatro bits menos significativos de A sumándole 0110 al acumulador, manteniéndose el valor de la bandera H.
- Si la bandera H no está encendida, analiza si los cuatro bits menos significativos de A corresponden a un número BCD, si es así, no se hace nada. Si éstos no representan un número BCD se corrigen sumándoles 0110 y sumando el acarreo generado a los cuatro bits más significativos de A (esta operación enciende la bandera H y podría encender la bandera C).
- Se analiza ahora la bandera C. Si está encendida se suma 0110 a los cuatro bits más significativos de A y se termina el proceso, sin modificar la bandera C.

- Si no está encendida, se analiza si estos bits corresponden a un número BCD. Si corresponden, no se realiza corrección y se termina el proceso; si no, se les suma 0110 y se enciende la bandera C para indicar un acarreo decimal y se termina el proceso.

Se espera que ahora el lector pueda entender la operación de la instrucción DAA. Dado que todo esto se realiza mediante un circuito especial, la única pseudo micro operación necesaria para esta instrucción es precisamente:

$$[A] \leftarrow Da([A])$$

•INTON (Interrupts On)

Esta instrucción habilita las interrupciones encendiendo la bandera I. La única pseudo micro operación que se realiza es:

$$I \leftarrow 1$$

•RTI (Return from Interrupt)

Esta instrucción se debe ejecutar al terminar de atender una interrupción para restaurar el estado del procesador al momento en que se presentó la interrupción. Al restaurar el estado del procesador, la única diferencia es que la bandera que habilita las interrupciones, I, estará siempre apagada. El programador deberá encenderla después de restaurar el estado del procesador si desea habilitar las interrupciones.

Las pseudo micro operaciones necesarias son:

```
[ SP ] ← [ SP ] +1
[ MAR ] ← [ SP ]
Mread y [ SP ] ← [ SP ] +1
[ PCH ] ← [ MDR ]
[ MAR ] ← [ SP ]
Mread y [ SP ] ← [ SP ] +1
[ PCL ] ← [ MDR ]
[ MAR ] ← [ SP ]
Mread y [ SP ] ← [ SP ] +1
[ XH ] ← [ MDR ]
[ MAR ] ← [ SP ]
Mread y [ SP ] ← [ SP ] +1
[ XL ] ← [ MDR ]
[ MAR ] ← [ SP ]
Mread y [ SP ] ← [ SP ] +1
[ FR ] ← [ MDR ]
[ MAR ] ← [ SP ]
```

```

Mread y [ SP ] ← [ SP ] +1
[ B ] ← [ MDR ]
[ MAR ] ← [ SP ]
Mread
[ A ] ← [ MDR ]

```

• SWI (Software Interrupt)

Al ejecutar esta instrucción se genera una interrupción por programa, independientemente de si la bandera I está encendida o apagada. Esto origina que se apague la bandera I si estaba encendida, se guarde el estado del procesador en la pila (los registros A, B, X, PC y FR) y se transfiera control a la dirección que está almacenada en las posiciones de memoria FFFC y FFFD hexadecimal. Las pseudo micro operaciones necesarias son:

```

I ← 0
[ MAR ] ← [ SP ]
[ MDR ] ← [ A ]
Mwrite y [ SP ] ← [ SP ] -1
[ MAR ] ← [ SP ]
[ MDR ] ← [ B ]
Mwrite y [ SP ] ← [ SP ] -1
[ MAR ] ← [ SP ]
[ MDR ] ← [ FR ]
Mwrite y [ SP ] ← [ SP ] -1
[ MAR ] ← [ SP ]
[ MDR ] ← [ XL ]
Mwrite y [ SP ] ← [ SP ] -1
[ MAR ] ← [ SP ]
[ MDR ] ← [ XH ]
Mwrite y [ SP ] ← [ SP ] -1
[ MAR ] ← [ SP ]
[ MDR ] ← [ PCL ]
Mwrite y [ SP ] ← [ SP ] -1
[ MAR ] ← [ SP ]
[ MDR ] ← [ PCH ]
Mwrite y [ SP ] ← [ SP ] -1
[ MAR ] ← FFFE
Mread
[ PCH ] ← [ MDR ] y [ MAR ] ← [ MAR ] + 1
Mread
[ PCL ] ← [ MDR ]

```

Esta instrucción permite copiar el registro de banderas al acumulador A, donde se puede manipular. Recuérdese que este registro también es de 8 bits, pero sus dos bits más significativos siempre valen 1. La pseudo micro operación que se realiza es:

$$[A] \leftarrow [FR]$$

• SFRA (Set Flag Register from A)

Esta instrucción copia el contenido del acumulador A al registro de banderas, pero los dos bits más significativos del registro de banderas siempre quedarán con el valor de 1, independientemente de lo que contengan en el acumulador A. La pseudo micro operación correspondiente es:

$$[FR] \leftarrow [A]$$

22.6. INTERRUPTACIONES.

Las interrupciones se generan cuando algún dispositivo periférico necesita ser atendido. En esta máquina se pueden tener hasta 256 direcciones de dispositivos periféricos, numeradas del 0 al 255.

Esta máquina utiliza un esquema de interrupciones vectorial, donde la dirección de la rutina que atiende a un dispositivo se obtiene de la dirección de memoria que se calcula en base a la dirección de dicho dispositivo. La forma de calcular la posición de memoria donde se encuentra la dirección de la rutina que atiende una interrupción es la siguiente:

- La dirección del dispositivo que generó la interrupción se almacena en el registro de datos de memoria (MDR) mediante la micro operación INTA (Interrupt acknowledge).
- Se suma el contenido del registro MDR multiplicado por dos con el valor hexadecimal F000 mediante un circuito especial, dejando el resultado en el registro de dirección de memoria (MAR), después de haber guardado el estado del procesador en la pila. Esta es la dirección donde se encuentran los 8 bits más significativos de la dirección de la rutina que atiende dicha interrupción. Los 8 bits menos significativos están en la siguiente posición de memoria.

Para ejemplificar el procedimiento anterior, supóngase que un dispositivo periférico cuya dirección es 100 (64 en hexadecimal) genera una interrupción. En el MDR se cargaría el valor hexadecimal 64.

Se sumarían el valor hexadecimal F000 con el doble del contenido del MDR quedando el resultado en MAR ($F000 + 2 * 64 = F0C8$). La dirección de la rutina que atiende las interrupciones del dispositivo 64 estaría en las posiciones de memoria F0C8 (los 8 bits más significativos) y F0C9 (los 8 bits menos significativos).

Cuando se presenta una interrupción y es aceptada (la bandera I debe estar en 1), se realiza un conjunto de pseudo micro operaciones que guardan el estado del procesador automáticamente en la pila y transfieren el control a la rutina que atiende la interrupción. Las micro operaciones realizadas son:

```

I ← 0
[ MAR ] ← [ SP ]
[ MDR ] ← [ A ]
Mwrite y [ SP ] ← [ SP ] -1
[ MAR ] ← [ SP ]
[ MDR ] ← [ B ]
Mwrite y [ SP ] ← [ SP ] -1
[ MAR ] ← [ SP ]
[ MDR ] ← [ FR ]
Mwrite y [ SP ] ← [ SP ] -1
[ MAR ] ← [ SP ]
[ MDR ] ← [ XL ]
Mwrite y [ SP ] ← [ SP ] -1
[ MAR ] ← [ SP ]
[ MDR ] ← [ XH ]
Mwrite y [ SP ] ← [ SP ] -1
[ MAR ] ← [ SP ]
[ MDR ] ← [ PCL ]
Mwrite y [ SP ] ← [ SP ] -1
[ MAR ] ← [ SP ]
[ MDR ] ← [ PCH ]
Mwrite y [ SP ] ← [ SP ] -1
INTA
[ MAR ] ← F000 + 2 * [ MDR ]
Mread
[ PCH ] ← [ MDR ]
[ MAR ] ← [ MAR ] + 1
Mread
[ PCL ] ← [ MDR ]

```

22.7. OPERACION DE RESET.

El procesador tiene una línea llamada RESET cuya finalidad es reiniciar la operación de la máquina. Cuando esta línea realiza una transición de 0 volts a 5 volts se ejecutan automáticamente las siguientes pseudo micro operaciones:

```

I ← 0
[ MAR ] ← FF00
Mread y [ PC ] ← [ PC ] + 1

```

[IR] ← [MDR]

Esta secuencia de pseudo micro operaciones realiza el ciclo de búsqueda de la instrucción que se encuentra en la dirección de memoria FF00 y la máquina empezaría a ejecutar esta instrucción. Típicamente las direcciones de la FF00 a la FFFF corresponderían a una memoria ROM en la cual reside el programa de arranque de la máquina.

Cuando se energiza la computadora, la línea de RESET automáticamente ejecutaría una transición de 0 volts a 5 volts, ocasionando que se inicie la ejecución del programa que se tenga a partir de la dirección de memoria FF00. Si en cualquier momento se oprimiera el botón de RESET, al soltarlo, la máquina reiniciaría su operación ejecutando este programa.

22.8. EJEMPLOS DE PROGRAMAS.

A continuación se presentarán cuatro programas simples que ayudarán a comprender el funcionamiento de esta computadora digital.

Al igual que en capítulo 2, se desarrollarán los programas en lenguaje ensamblador y luego se mostrará como quedan en lenguaje maquinal, pero ahora se mostrarán direcciones, instrucciones y datos en hexadecimal como una abreviación del binario. Recuérdese que dentro de la máquina todo está en binario.

Adicionalmente, se utilizarán pseudoinstrucciones equivalentes a las definidas en el capítulo 2, las cuales no se traducen a una instrucción de máquina ya que no corresponden a éstas. Su propósito es controlar la generación del lenguaje maquinal a partir del programa en lenguaje ensamblador. Se usarán tres pseudoinstrucciones **ORG**, **DEF8** y **DEF16**.

La pseudoinstrucción **ORG** tiene como argumento una dirección de memoria y especifica que las instrucciones o datos que se definan a continuación se deberán colocar a partir de dicha dirección de memoria. Se puede especificar la dirección en decimal o en hexadecimal, añadiendo la letra H al final de la constante. Por ejemplo, los valores 127 y 7FH son equivalentes.

Las pseudoinstrucciones **DEF8** y **DEF16** sirven para definir constantes en un programa en lenguaje ensamblador. La pseudoinstrucción **DEF8** tiene como argumento un número y especifica que en el lugar de memoria correspondiente a la pseudoinstrucción deberá colocarse dicho número en 8 bits. La pseudoinstrucción **DEF16** indica que debe colocarse el valor dado pero ahora en formato de 16 bits. El byte más significativo se guarda primero y el byte menos significativo se guarda en la siguiente dirección de memoria. También pueden especificarse constantes en decimal o hexadecimal para estas pseudoinstrucciones.

Los programas mostrados tienen los mismos tres campos que los programas del capítulo 2:

- El campo para etiquetas
- El campo para la instrucción y sus parámetros

- El campo para comentarios
- **Campo para etiquetas.**
Este es el primer campo de una línea de programa y puede usarse o dejarse en blanco. Cuando se utiliza, se asigna una dirección simbólica a la línea y, al momento de pasarse a lenguaje maquinal, el valor de la etiqueta será la dirección en memoria en la cual quedará la instrucción correspondiente a la línea de programa que tiene la etiqueta.
- **Campo para instrucciones y sus parámetros.**
En este campo se coloca el mnemónico para la instrucción, el tipo de direccionamiento y la dirección simbólica o numérica, separados por comas. En el caso de direccionamiento inmediato, se coloca el dato inmediato en el lugar de la dirección. Para las instrucciones que no tienen parámetros, solamente se coloca el mnemónico. Las pseudoinstrucciones con sus parámetros también se colocan en este campo.
- **Campo para comentarios.**
Este campo es el último de una línea de programa en lenguaje ensamblador y su contenido no se traduce a lenguaje maquinal. En este campo se colocarían, en forma opcional, los comentarios pertinentes para hacer el programa más entendible a otra persona que lo analice.

Los códigos de operación para la computadora de ejemplo se definieron anteriormente así como sus mnemónicos. La simbología usada para los diferentes tipos de direccionamiento y su correspondiente representación en lenguaje maquinal, son los siguientes:

Inmediato	I	00
Directo	D	01
Absoluto	A	10
Indexado	X	11

Primeramente se presenta un programa que suma los números X, Y y Z y deja el resultado de la suma en W. Se supone que el valor de X está en la dirección de memoria 301, el de Y en la dirección 302 y el de Z en la dirección 303. El resultado W se dejará en la dirección de memoria 300. Se asignaron los valores de 25, 35 y -31 para X, Y y Z respectivamente. Para W se asignó el valor cero inicialmente.

La primera instrucción del programa estará en la dirección de memoria 256. Se supone que antes de iniciar la ejecución del programa, el registro contador de programa (**PC**) deberá inicializarse con la dirección 256.

ETIQUETA	INSTRUCCION	COMENTARIOS
	ORG, 256	LA SIG INSTR VA EN LA DIR 256
	LDA, A, X	CARGAR X EN EL AC
	ADD, A, Y	SUMARLE Y AL AC
	ADD, A, Z	SUMARLE Z AL AC

	STA, A,W	GUARDAR RESULTADO EN W
	HLT	DETENER LA EJECUCION
	ORG, 300	LA SIG INST VA EN LA DIR 300
W	DEF8, 0	DEFINIR EL VALOR 0
X	DEF8, 25	DEFINIR EL VALOR 25
Y	DEF8, 35	DEFINIR EL VALOR 35
Z	DEF8,-31	DEFINIR EL VALOR -31

A continuación se muestra como quedaría el programa anterior al pasarlo a lenguaje maquina. Ahora, dado que existen instrucciones de 1, 2 y 3 bytes, se listan todos los bytes de una instrucción en una misma línea y la dirección de memoria que se muestra a continuación es la que sigue de dicha instrucción. El contenido de memoria y las direcciones se muestran en hexadecimal y se han suprimido los comentarios.

DIRECCION	CONTENIDO	ETIQUETA	INSTRUCCION
			ORG, 256
0100	42 012D		LDA, A, X
0103	16 012E		ADD, A, Y
0106	16 012F		ADD, A, Z
0109	11 012C		STA, A, W
010C	FF		HLT
			ORG, 300
012C	0	W	DEF8, 0
012D	19	X	DEF8, 25
012E	23	Y	DEF8, 35
021F	E1	Z	DEF8, -31

Por lo general, los compiladores incluyen código para detectar overflow al realizar operaciones aritméticas. En estos ejemplos deliberadamente no se incluye la verificación por overflow para no complicarlos.

El siguiente programa que se presentará suma un grupo de N números enteros de 8 bits, los cuales se encuentran contiguos a partir de una posición de memoria dada. El valor de N se almacenará en la posición de memoria 301 y en este caso tendrá el valor de 10 para sumar 10 números enteros con signo de 8 bits. En las posiciones de memoria 302 Y 303 se almacenará la dirección donde está el primero de los números a sumar. Los números que se van a sumar estarán a partir de la posición de memoria 310 inclusive. El valor de la suma se almacenará en la posición de memoria 300. Esta última posición de memoria se inicializará con cero aún cuando no es necesario. El programa inicia en la posición de memoria 256.

En este programa se ejemplificará el uso del registro índice y se podrá apreciar su utilidad así como la ventaja de tener dos registros acumuladores. El lector puede comparar este programa con el presentado en el capítulo 2, donde no se contaba con un registro índice y solamente se tenía un registro acumulador. Los datos a sumar son: 10, 15, -30, 40, 18, 23, -44, 3, -64 y 28.

ETIQUETA	INSTRUCCION	COMENTARIOS
	ORG, 256	LA SIG INST VA EN LA DIR 256
	LDX, A, DX	CARGAR LA DIRECCION DE INICIO EN X
	LDB, A, N	CARGAR N EN B
	LDA, X, 0	CARGAR EL PRIMER DATO EN A
	DCB	DECREMENTAR EL REGISTRO B
CICLO	INX	INCREMENTAR EL REGISTRO X
	ADDA, X, S	SUMAR EL SIGUIENTE DATO A A
	DCB	DECREMENTAR EL REGISTRO B
	JMZ, A, FIN	SI RESULTA CERO IR A FIN
	JMP, A, CICLO	REGRESAR A CICLO
	STA, A, SUMA	GUARDAR LA SUMA
FIN	HLT	DETENER LA EJECUCION
	ORG, 300	LA SIG INST VA EN LA DIR 300
SUMA	DEF8, 0	INICIALIZAR LA SUMA A CERO
N	DEF8, 10	EL VALOR DE N ES 10
DX	DEF16, DIR	LA DIR DEL PRIMER DATO ES DIR
	ORG, 310	LA SIG DIRECCION ES LA 310
DIR	DEF8, 10	EL PRIMER DATO ES 10
	DEF8, 15	EL SEGUNDO DATO ES 15
	DEF8, -30	EL TERCER DATO ES -30
	DEF8, +40	EL CUARTO DATO ES 40
	DEF8, 18	EL QUINTO DATO ES 18
	DEF8, 23	EL SEXTO DATO ES 23
	DEF8, -44	EL SEPTIMO DATO ES -44
	DEF8, 3	EL OCTAVO DATO ES 3
	DEF8, -64	EL NOVENO DATO ES -64
	DEF8, 28	EL DECIMO DATO ES 28

A continuación se muestra el programa anterior traducido al lenguaje maquina. De nuevo se han suprimido los comentarios.

DIRECCION	CONTENIDO	ETIQUETA	INSTRUCCION
			ORG, 256
0100	4C 0302		LDX, A, DX
0103	46 0301		LDB, A, N
0106	43 00		LDA, X, 0
0108	70		DCB
0109	68	CICLO	INX
010A	17 00		ADDA, X, S
010C	70		DCB
010D	BE 0117		JMZ, A, FIN
0110	BA 010A		JMP, A, CICLO
0113	52 0300		STA, A, SUMA
0116	FF	FIN	HLT

			ORG, 300
012C	00	SUMA	DEF8, 0
012D	0A	N	DEF8, 10
012E	0304	DX	DEF16, DIR
			ORG, 310
0136	0A	DIR	DEF8, 10
0137	0F		DEF8, 15
0138	E2		DEF8, -30
0139	28		DEF8, +40
013A	12		DEF8, 18
013B	17		DEF8, 23
013C	D4		DEF8, -44
013D	03		DEF8, 3
013E	C0		DEF8, -64
013F	1C		DEF8, 28

El siguiente programa realiza la suma de dos números enteros de 16 bits en formato de complementos a la base. Para diseñarlo se utilizará un subprograma que realiza la suma al cual se le pasarán como parámetros las direcciones de memoria donde inicia cada uno de los datos así como la dirección donde deberá dejarse el resultado. En este caso no es necesario hacerlo mediante un subprograma, pero se aprovechará para ejemplificar también el uso de subprogramas y paso de parámetros a través de la pila.

El programa inicia en la dirección 512 y los datos a sumar, A y B, estarán en las direcciones AA y BB de memoria. El resultado se dejará en la dirección CC. El subprograma que realiza la suma inicia en la dirección 768.

El programa principal pone las direcciones de AA, BB y CC en la pila y luego llama al subprograma que efectúa la suma. Este subprograma saca de la pila las direcciones de los datos a sumar así como la dirección donde se dejará el resultado. Cuando el programa principal llama al subprograma, la dirección de retorno también se guarda en la pila y el subprograma debe diseñarse de tal forma que esta dirección no se pierda.

Para sumar dos números enteros de 16 bits en formato de complementos a la base primero se suman los 8 bits menos significativos. Si hubiere un acarreo, éste se guarda en la bandera C automáticamente. Luego es necesario sumar los 8 bits más significativos de los operandos y el posible acarreo para lo cual se utiliza la instrucción ADDC en este caso.

A continuación se muestra este programa.

ETIQUETA	INSTRUCCION	COMENTARIOS
	ORG, 512	LA SIG INST VA EN LA DIR 512
	LDX, A, DIRC	CARGAR LA DIRECCION DE CC EN X
	PSHX	GUARDARLA EN LA PILA
	LDX, A, DIRB	CARGAR LA DIRECCION DE BB EN X

	PSHX	GUARDARLA EN LA PILA
	LDX, A, DIRA	CARGAR LA DIRECCION DE AA EN X
	PSHX	GUARDARLA EN LA PILA
	JSR, A, SUMA	LLAMAR A LA RUTINA PARA SUMAR
	HLT	DETENER LA EJECUCION
AA	DEF16, 125	EL VALOR DE A ES 125
BB	DEF16, -547	EL VALOR DE B ES -547
CC	DEF16, 0	EL VALOR DE C INICIALMENTE ES 0
DIRA	DEF16, AA	DIRA TIENE LA DIRECCION DE AA
DIRB	DEF16, BB	DIRA TIENE LA DIRECCION DE BB
DIRC	DEF16, CC	DIRA TIENE LA DIRECCION DE CC
	ORG, 300H	LA SIG INST VA EN LA DIR 300H
SUMA POPX		SACAR LA DIRECCION DE RETORNO
	STX, A, DRET	GUARDARLA EN DRET
	POPX	SACAR LA DIRECCION DE AA
	STX, A, DA	GUARDARLA EN DA
	POPX	SACAR LA DIRECCION DE BB
	STX, A, DB	GUARDARLA EN DB
	POPX	SACAR LA DIRECCION DE CC
	STX, A, DC	GUARDARLA EN DA
	LDX, A, DRET	CARGAR LA DIRECCION DE RETORNO EN X
	PSHX	GUARDARLA EN LA PILA DE NUEVO
	LDX, A, DA	CARGAR LA DIRECCION DE AA EN X
	LDA, X, 1	CARGAR LA PARTE BAJA DE AA EN A
	LDX, A, DB	CARGAR LA DIRECCION DE BB EN X
	ADD, X, 1	SUMAR LA PARTE BAJA DE BB A A
	LDX, A, DC	CARGAR LA DIRECCION DE CC EN X
	STA, X, 1	GUARDAR LA PARTE BAJA DE LA SUMA
	LDX, A, DA	CARGAR LA DIRECCION DE AA EN X
	LDA, X, 0	CARGAR LA PARTE ALTA DE AA EN A
	LDX, A, DB	CARGAR LA DIRECCION DE BB EN X
	ADDC, X, 0	SUMAR LA PARTE ALTA DE BB A A
	LDX, A, DC	CARGAR LA DIRECCION DE CC EN X
	STA, X, 0	GUARDAR LA PARTE ALTA DE LA SUMA
	RTN	REGRESAR AL PROGRAMA PRINCIPAL
DRET	DEF16, 0	LUGAR PARA LA DIR. DE RETORNO
DA	DEF16, 0	LUGAR PARA LA DIRECCION DE AA
DB	DEF16, 0	LUGAR PARA LA DIRECCION DE BB
DC	DEF16, 0	LUGAR PARA LA DIRECCION DE CC

DIRECCION	CONTENIDO	ETIQUETA	INSTRUCCION
			ORG, 512
0200	4E 0219		LDX, A, DIRC
0203	80		PSHX
0204	4E 00217		LDX, A, DIRB

0207	80		PSHX
0208	4E 0215		LDX, A, DIRA
020B	80		PSHX
020C	CE 0300		JSR, A, SUMA
020F	FC		HLT
0210	007D	A	DEF16, 125
0212	FDDD	B	DEF16, -547
0214	0000	C	DEF16, 0
0216	020F	DIRA	DEF16, A
0218	0211	DIRB	DEF16, B
021A	0213	DIRC	DEF16, C
			ORG, 300H
0300	8C	SUMA POPX	
0301	5E 0333		STX, A, DRET
0304	8C		POPX
0305	5E 0335		STX, A, DA
0308	8C		POPX
0309	8C 0337		STX, A, DB
030C	8C		POPX
030D	5E 0339		STX, A, DC
0310	4E 0333		LDX, A, DRET
0313	80		PSHX
0314	4E 0335		LDX, A, DA
0317	43 01	LDA, X, 1	
0319	4E 0337		LDX, A, DB
031C	16 01	ADD, X, 1	
031E	4E 0339		LDX, A, DC
0321	53 01	STA, X, 1	
0323	4E 0335		LDX, A, DA
0326	4E 00		LDA, X, 0
0328	4E 0037		LDX, A, DB
032B	23 00	ADDC, X, 0	
032D	4E 0039		LDX, A, DC
0330	53 00	STA, X, 0	
0332	D0		RTN
0333	0000	DRET	DEF16, 0
0335	0000	DA	DEF16, 0
0337	0000	DB	DEF16, 0
0339	0000	DC	DEF16, 0

El último programa que se mostrará suma dos números BCD de 4 dígitos y deja el resultado de esta suma también con cuatro dígitos. Los números a sumar inician en las direcciones de memoria AA y BB. El resultado se dejará en la dirección de memoria CC.

ETIQUETA	INSTRUCCION	COMENTARIOS
	ORG, 256	LA SIG INSTR VA EN LA DIR 256
	LDX, A, DA	CARGAR LA DIRECCION DE AA EN X
	LDA, X, 1	CARGAR LA PARTE BAJA DE AA EN A
	LDX, A, DB	CARGAR LA DIRECCION DE BB EN X
	ADD, X, 1	SUMAR LA PARTE BAJA DE BB A A
	DAA	AJUSTAR LA SUMA A FORMATO BCD
	LDX, A, DC	CARGAR LA DIRECCION DE CC EN X
	STA, X, 1	GUARDAR LA PARTE BAJA DE LA SUMA
	LDX, A, DA	CARGAR LA DIRECCION DE AA EN X
	LDA, X, 0	CARGAR LA PARTE ALTA DE AA EN A
	LDX, A, DB	CARGAR LA DIRECCION DE BB EN X
	ADDC, X, 0	SUMAR LA PARTE ALTA DE BB A A
	DAA	AJUSTAR LA SUMA A FORMATO BCD
	LDX, A, DC	CARGAR LA DIRECCION DE CC EN X
	STA, X, 0	GUARDAR LA PARTE ALTA DE LA SUMA
	HLT	DETENER LA EJECUCION
AA	DEF16, 0001H	EL VALOR DE A ES 0125 (BCD)
	DEF16, 0205H	
BB	DEF16, 0002H	EL VALOR DE B ES 0255 (BCD)
	DEF16, 0505H	
CC	DEF16, 0	EL VALOR DE C INICIALMENTE ES 0
	DEF16, 0	
DA	DEF16, AA	DA TIENE LA DIRECCION DE AA
DB	DEF16, BB	DA TIENE LA DIRECCION DE BB
DC	DEF16, CC	DA TIENE LA DIRECCION DE CC

Este programa quedaría, en lenguaje maquina, como se muestra.

DIRECCION	CONTENIDO	ETIQUETA	INSTRUCCION
			ORG, 256
			LDX, A, DA
0100	4E 013D		
0103	43 01	LDA, X, 1	
0105	4E 013F		LDX, A, DB
0108	16 01	ADD, X, 1	
010A	D4		DAA
010B	4E 0141		LDX, A, DC
010E	53 01	STA, X, 1	
0120	4E 013D		LDX, A, DA
0123	43 00	LDA, X, 0	
0125	4E 013F		LDX, A, DB
0128	23 00	ADDC, X, 0	
012A	D4		DAA
012B	4E 0141		LDX, A, DC

012E	53 00		STA, X, 0
0130	FC		HLT
0131	0001	AA	DEF16, 0001H
0133	0205		DEF16, 0205H
0135	0002	BB	DEF16, 0002H
0137	0505		DEF16, 0505H
0139	0000	CC	DEF16, 0
013B	0000		DEF16, 0
013D	0127	DA	DEF16, AA
013F	0129	DB	DEF16, BB
0141	012B	DC	DEF16, CC

22.9. RESUMEN.

En este capítulo se presentó la arquitectura de una máquina hipotética que cuenta con un juego de instrucciones completo y que perfectamente podría representar el juego de instrucciones de un procesador real.

Esta arquitectura, como la mayoría de las actuales, incluye instrucciones para el manejo de una pila en memoria, la cual también es utilizada para las llamadas a subprogramas y manejo de interrupciones. Esta máquina también incluye operaciones de entrada y salida con el acumulador A. Se cuenta con un esquema vectorial de interrupciones en el cual la dirección que contiene la dirección de inicio de la rutina que atiende una interrupción en particular se calcula en base a la dirección del dispositivo periférico que originó la interrupción.

Adicionalmente se tiene una instrucción que genera una interrupción, la cual se procesa en forma similar a las demás interrupciones. Para iniciar la operación de la máquina, se tiene la señal de RESET, la cual ocasiona que el procesador empiece a ejecutar un programa en una dirección específica.

22.10. PROBLEMAS

1. Compare las arquitecturas de las computadoras digitales simples de 0, 1, 2 ó 3 direcciones, escribiendo un programa para calcular la siguiente operación:

$$x = \frac{A + B - C}{D + E - F}$$

Las instrucciones disponibles para cada computadora son las siguientes:

0 direcciones	1 dirección	2 direcciones	3 direcciones
Push M	LDA M	LD OP ₁ ,OP ₂	LD OP ₁ ,OP ₂ ,OP ₃
Pop M	STA M	ST OP ₁ ,OP ₂	ST OP ₁ ,OP ₂ ,OP ₃
ADD	ADD M	ADD OP ₁ ,OP ₂	ADD OP ₁ ,OP ₂ ,OP ₃
SUB	SUB M	SUB OP ₁ ,OP ₂	SUB OP ₁ ,OP ₂ ,OP ₃
DIV	DIV M	DIV OP ₁ ,OP ₂	DIV OP ₁ ,OP ₂ ,OP ₃

M siempre es una dirección de memoria y OP_x puede ser un número de registro o una dirección de memoria.

2. Par la máquina definida en este capítulo, indique la secuencias de Pseudo micro operaciones necesarios para realizar el ciclo de ejecución de las siguientes instrucciones maquinales:

- a) LDX con direccionamiento inmediato.
- b) LDX con direccionamiento directo.
- c) LDX con direccionamiento absoluto.
- d) LDB con direccionamiento indexado.
- e) STX con direccionamiento inmediato.
- f) STX con direccionamiento directo.
- g) STX con direccionamiento absoluto.
- h) STB con direccionamiento indexado.
- i) SUBAB.
- j) SUBC con direccionamiento inmediato.
- k) SUBC con direccionamiento directo.
- l) SUBC con direccionamiento absoluto.
- m) SUBC con direccionamiento indexado.
- n) JMV con direccionamiento directo
- ñ) JMV con direccionamiento absoluto
- o) JMV con direccionamiento indexado
- p) CFRA
- q) PSHB
- r) POPB
- s) SWPAB

3. Codifique un programa principal y un subprograma en lenguaje ensamblador de la máquina definida en este capítulo, que realicen los siguiente:

El programa principal deberá poner dos datos en la pila, A y B y llamar al subprograma. Estos dos datos están en memoria.

Suponga que la pila empieza en la dirección de memoria 200, para lo cual es necesario cargar este valor en el registro SP antes de llamar al subprograma.

El subprograma deberá calcular el valor absoluto de la diferencia de los dos datos que el programa principal dejó en la pila y regresar este valor en el acumulador. Recuérdese que la dirección de retorno al programa principal también se deja en la pila.

Finalmente, el programa principal deberá dejar el resultado en el lugar de memoria asignado al dato C.

4. Traduzca el programa y subprograma del problema anterior a lenguaje maquinaal.

5. Codifique un programa en lenguaje ensamblador de la máquina definida en este capítulo, que realicen la siguiente instrucción de un lenguaje de nivel superior.

IF $X < Y$ THEN GOTO 500 (SALTA A LA DIRECCION 500 SI $X < Y$)

X y Y son números positivos o negativos en representación de complementos a dos.

Recuerde que al realizar la comparación puede ocurrir un desborde, lo cual debe de evitarlo.

CAPITULO 23

UNIDAD DE CONTROL

23.1. INTRODUCCION.

La unidad de control, que es parte de la unidad central de proceso (CPU), tiene como función producir las señales necesarias para que se lleven a cabo todas las operaciones necesarias para la ejecución de las instrucciones, incluyendo el control del flujo de información requerido para realizarlas. Estas señales se conocen como señales de control.

Al momento en que el CPU esta ejecutando un programa, la unidad de control es responsable de producir las señales de control adecuadas para que se realicen en una forma repetitiva los dos ciclos requeridos para ejecutar las instrucciones: el ciclo de búsqueda (FETCH) y el ciclo de ejecución.

Existen dos formas de construir una unidad de control, una es mediante una máquina de Moore o de Mealy, donde las salidas de cada estado serían las señales de control que indicarían las operaciones y/o transferencias de información a realizar en ese momento. La otra forma es utilizar firmware para su construcción donde la memoria ROM del firmware contendría lo que se conoce como microinstrucciones, las cuales indican las operaciones y/o la transferencias de información a realizar para ejecutar los ciclos de búsqueda y ejecución de las instrucciones. A éstas últimas se les conoce como unidades de control microprogramables.

La unidad de control actúa como un intérprete de instrucciones de máquina, ya que cada vez que se ejecuta una instrucción maquina, se genera una serie de eventos en la computadora para que se lleve a cabo la instrucción.

En este capítulo se utilizará la computadora definida en el capítulo anterior, utilizando las diferentes formas de construir la unidad de control.

23.2. CONCEPTOS BÁSICOS DE CONTROL.

Antes de diseñar una unidad de control se debe conocer exactamente la arquitectura de la computadora y las señales de control que ésta requerirá. En esta sección se utilizará una versión simplificada de la arquitectura presentada en el capítulo anterior para definir los conceptos básicos de la unidad de control. La figura 23.1 muestra esta arquitectura simplificada.

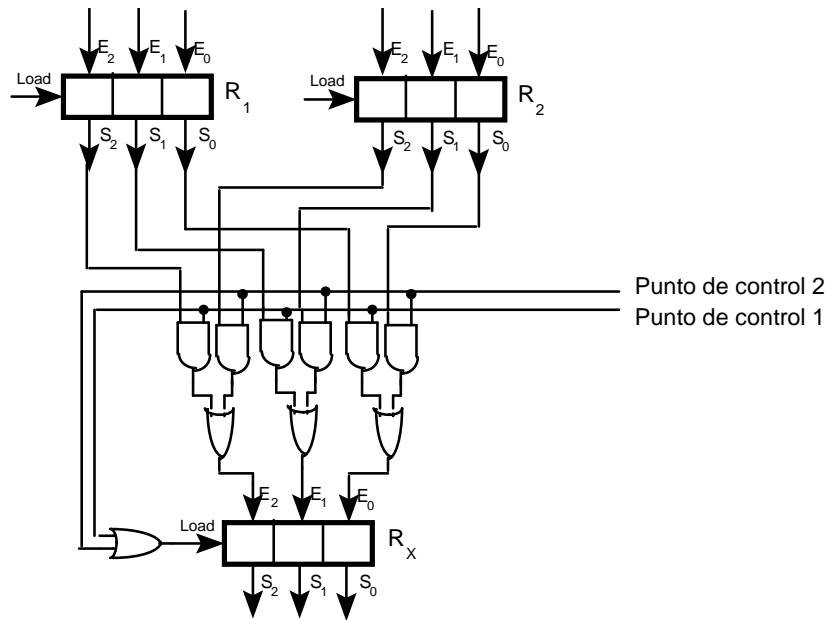


Figura 32.2 Ejemplo de puntos de control

En la computadora que se utilizará, la transferencia de información entre registros se realiza a través de un bus usando un esquema semejante al de la figura 20.19 y se controla por puntos de control como se indicó en el ejemplo anterior. Al diseñar la unidad de control se debe de evitar el caso en que dos registros pongan información en el mismo bus al mismo tiempo, ya que se produciría un corto. Se considerará que el bus tiene puros ceros cuando no se ponga información en él.

Además, existen puntos de control para indicar que se realice una operación en algún elemento. Por ejemplo, en el acumulador pueden existir puntos de control para indicar que su valor se incremente en una unidad, que se complemente a dos su contenido, o que se haga un corrimiento en la información que contiene. En la ALU existen puntos de control para indicar la operación aritmética o lógica a realizar en cada caso particular. En la memoria existen puntos de control para indicar si se desea efectuar una operación de lectura o de escritura.

La unidad llamada “modifica banderas” opera siempre que se cargue un nuevo valor en el acumulador A o se realice una operación en la unidad aritmética-lógica que modifique el valor del acumulador A. No se requiere de un punto de control adicional.

La computadora que se analizará tiene definidos 23 puntos de control (**pc**) los cuales se listan en la tabla 23.1

pc	Acción	Comentario
0	$[PCL] \leftarrow [MDR]$	El contenido del MDR pasa al BUSL y es almacenado en el PCL.
1	$[PCH] \leftarrow [MDR]$	El contenido del MDR pasa al BUSH y es almacenado en el PCH.
2	$[A] \leftarrow [MDR]$	El contenido del MDR pasa al BUSL y es almacenado en el acumulador A.
3	$[AUX] \leftarrow [MDR]$	El contenido del MDR pasa al BUSL y es almacenado en AUX.
4	$[IR] \leftarrow [MDR]$	El contenido del MDR pasa al BUSL y es almacenado en IR.
5	$[AUX] \leftarrow ALU \leftarrow [MDR]$	El contenido del MDR pasa al BUSH, se realiza una operación con el valor que contiene BUSL y es almacenado en el AUX.
6	$[AUX] \leftarrow ALU_1 \leftarrow [A]$	El contenido del acumulador A pasa al BUSL, es sumado con el valor que contiene BUSH y es almacenado en el AUX.
7	$[AUX] \leftarrow ALU_2 \leftarrow [A]$	El contenido del acumulador A pasa al BUSL, es sumado usando el carry con el valor que contiene BUSH y es almacenado en el AUX.
8	$[AUX] \leftarrow ALU_3 \leftarrow [A]$	El contenido del acumulador A pasa al BUSL, es restado con el valor que contiene BUSH y es almacenado en el AUX.
9	$[AUX] \leftarrow ALU_4 \leftarrow [A]$	El contenido del acumulador A pasa al BUSL, es restado usando el carry con el valor que contiene BUSH y es almacenado en el AUX.
10	$[MDR] \leftarrow [A]$	El contenido del acumulador A pasa al BUSL y es almacenado en MDR.
11	$[A] \leftarrow [A] + 1$	Se incrementa el contenido del acumulador A.
12	$[A] \leftarrow 0$	Se borra el contenido del acumulador A.
13	$[A] \leftarrow [A]^*$	Se complementa a dos el contenido del acumulador A.
14	$[A] \leftarrow [AUX]$	El contenido de AUX pasa al BUSL y es almacenado en el acumulador A.
15	$[PC] \leftarrow [PC] + 1$	Se incrementa el contenido del PC.
16	$[PCH] \leftarrow 0$	Se borra el contenido del acumulador PCH.
17	$[MAR] \leftarrow [MAR] + 1$	Se incrementa el contenido del MAR.
18	$[MAR] \leftarrow [PCH], [PCL]$ o $[MAR] \leftarrow [PC]$	El contenido de PCH pasa al BUSH, además el contenido de PCL pasa al BUSL y es almacenado en el MAR.
19	$[MAR] \leftarrow [AUX], [MDR]$	El contenido de MDR pasa al BUSH, además el contenido de AUX pasa al BUSL y es almacenado en el MAR.
20	$[MAR] \leftarrow 0, [MDR]$	El contenido de MDR pasa al BUSL y es almacenado en el MAR. Dado que no se puso información en el BUSH este contiene puros ceros y se cargan 8 ceros en

		la parte alta del MAR.
21	Mread	Lectura de memoria. Carga el byte que se lee en el registro MDR
22	Mwrite	Escritura en memoria. Toma del registro MDR el byte a escribir.

Tabla 23.1 Puntos de control

Como se puede deducir de los puntos de control listados, la ALU de esta computadora realiza las operaciones aritméticas de suma y resta, incluyendo o no el carry.

Para que una instrucción maquina pueda ser ejecutada por la computadora, ésta se debe definir por una secuencia de acciones basadas en activaciones de **pc**. Por ejemplo, la secuencia de puntos de control para el ciclo de búsqueda (FETCH) y ejecución de la instrucción ADDA (Add to A) con direccionamiento inmediato sería la siguiente:

Ciclo de búsqueda:	18	$[MAR] \leftarrow [PC]$
	21,15	Mread, $[PC] \leftarrow [PC] + 1$
	4	$[IR] \leftarrow [MDR]$
Ciclo de ejecución :	18	$[MAR] \leftarrow [PC]$
	21,15	Mread, $[PC] \leftarrow [PC] + 1$
	5,6	$[AUX] \leftarrow ALU_1 \leftarrow [A], [AUX] \leftarrow ALU \leftarrow [MDR]$
	14	$[A] \leftarrow [AUX]$

La secuencia de **pc** que deben activarse para realizar el ciclo de búsqueda es la misma para todas las instrucciones; sin embargo, la secuencia de **pc** necesaria para el ciclo de ejecución depende de la instrucción a ejecutar, por lo cual, una de las entradas a la unidad de control es el código de operación de la instrucción. Dado que la ejecución de algunas instrucciones dependen del estado de alguna bandera, éstas también actúan como entradas de la unidad de control.

La velocidad a la que opera una computadora está relacionada con la frecuencia de la señal de reloj que tiene como entrada la unidad central de proceso. En cada pulso de reloj que ocurra, la computadora debe realizar una o varias acciones. El orden en que deben ocurrir las acciones en cada ciclo del reloj es indicado por la unidad de control a través de sus líneas de salida, que corresponden a los diferentes puntos de control de la computadora. El número máximo de acciones a realizar en cada ciclo de reloj depende de la arquitectura particular que se tenga y de la operación en específica a realizar. Para que una instrucción maquina se ejecute en el mínimo tiempo posible, se debe maximizar este número de acciones.

La unidad de control deberá proporcionar los valores a los **pc** un tiempo antes que las acción se realice. En la figura 23.3 se muestra una posible alternativa para realizar esta sincronización. En ésta, la unidad de control proporciona los valores a los **pc** en la transición positiva de la señal de reloj. En la transición negativa de la señal de reloj se modifican los registros involucrados. Al interpretar el diagrama de la figura 23.3, se debe considerar que el valor de cada una de las líneas es constante (1 o 0) cuando se tienen dos líneas paralelas. Los valores pueden estar modificándose cuando se tienen dos líneas cruzadas.

Recuerde que los registros internos en un CPU son sincrónicos y tienen entradas y salidas en paralelo.

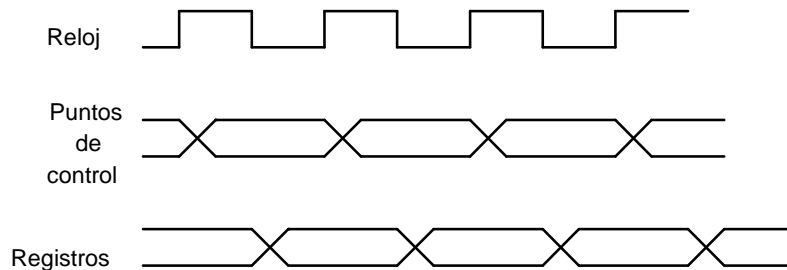


Figura 23.3 Sincronización entre la unidad de control y los registros

23.3. Diseño de la unidad de control para la computadora simplificada usando solo hardware.

Existen dos formas de construir la unidad de control usando solo hardware, una es usando el modelo de Moore y la otra es usando el modelo de Mealy.

Si se usa un modelo de Moore para construirla, se debe obtener un diagrama de estados donde se indique la secuencia de pasos a realizar para cada una de las instrucciones. En la figura 23.4 se muestra una fracción de este diagrama, donde se indican los pasos para ejecutar la instrucción ADDA con direccionamiento inmediato (código de operación 14) y la instrucción NOP (código de operación 00). (X puede ser cualquier valor)

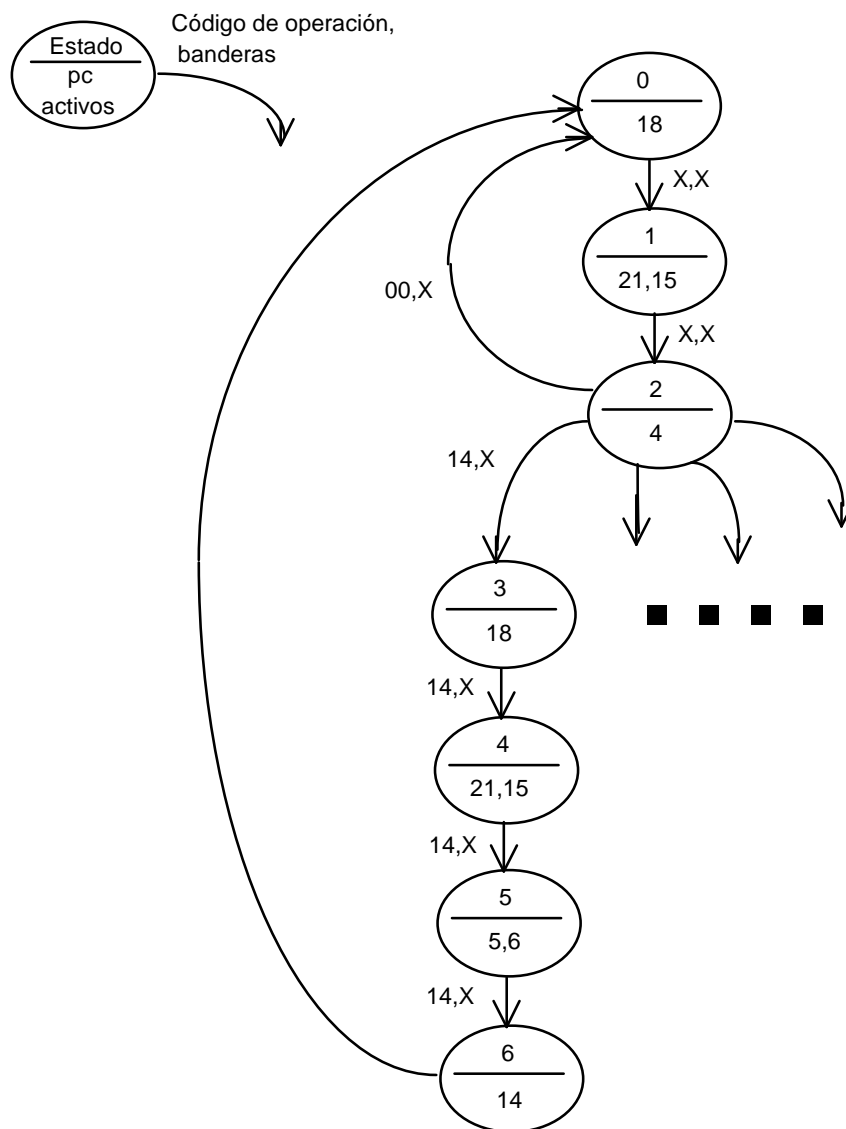


Figura 23.3 Fracción del diagrama de estados para el modelo de Moore.

Los estados 0, 1 y 2 corresponden a la búsqueda del código de operación y son los mismos para todas las instrucciones. El siguiente estado depende del código de operación que se tiene en ese momento. Si el código de operación corresponde a la instrucción NOP (00), se regresaría al estado 0 para realizar otra búsqueda, pero si el código de operación es el de la instrucción ADDA con direccionamiento inmediato (14), se ejecutarían los estados 3, 4, 5 y 6, los cuales realizan el ciclo de ejecución de esta instrucción. Al terminar se regresa al estado 0 para realizar el ciclo de búsqueda de la siguiente instrucción.

Como se muestra en el diagrama, del estado 2 debe salir una trayectoria para cada instrucción que esta computadora pueda ejecutar. El lector podrá comprobar que parte de estas trayectorias son compartidas por varias instrucciones; por ejemplo, los estados 3 y 4 son comunes a todas las

instrucciones con direccionamiento inmediato. Los valores de las banderas solo se cuestionan al ejecutar una instrucción que depende de éstas, tales como las transferencias condicionales.

Al tener el diagrama de estados completo se puede construir un circuito secuencial síncrono que se comporte de acuerdo al diagrama. La unidad de control consistiría en el circuito secuencial resultante.

Cuando se efectúa un RESET, las acciones que se realizan son cargar un valor determinado ($FF00_{16}$) en el PC, borrar todos los registros de la computadora e iniciar la ejecución del diagrama de estados en el estado 0.

Otra forma de construir la unidad de control con solo hardware es utilizando el modelo de Mealy como lo indica el diagrama de la figura 23.4. En los primeros tres estados del diagrama siempre se activarán los puntos de control necesarios para realizar el ciclo de búsqueda. En los siguientes estados, los puntos de control que se activen dependerán del valor que se tengan en el código de operación y en las banderas. El número de estados requeridos después de haber hecho el ciclo de búsqueda, depende de la instrucción específica que se ejecute.

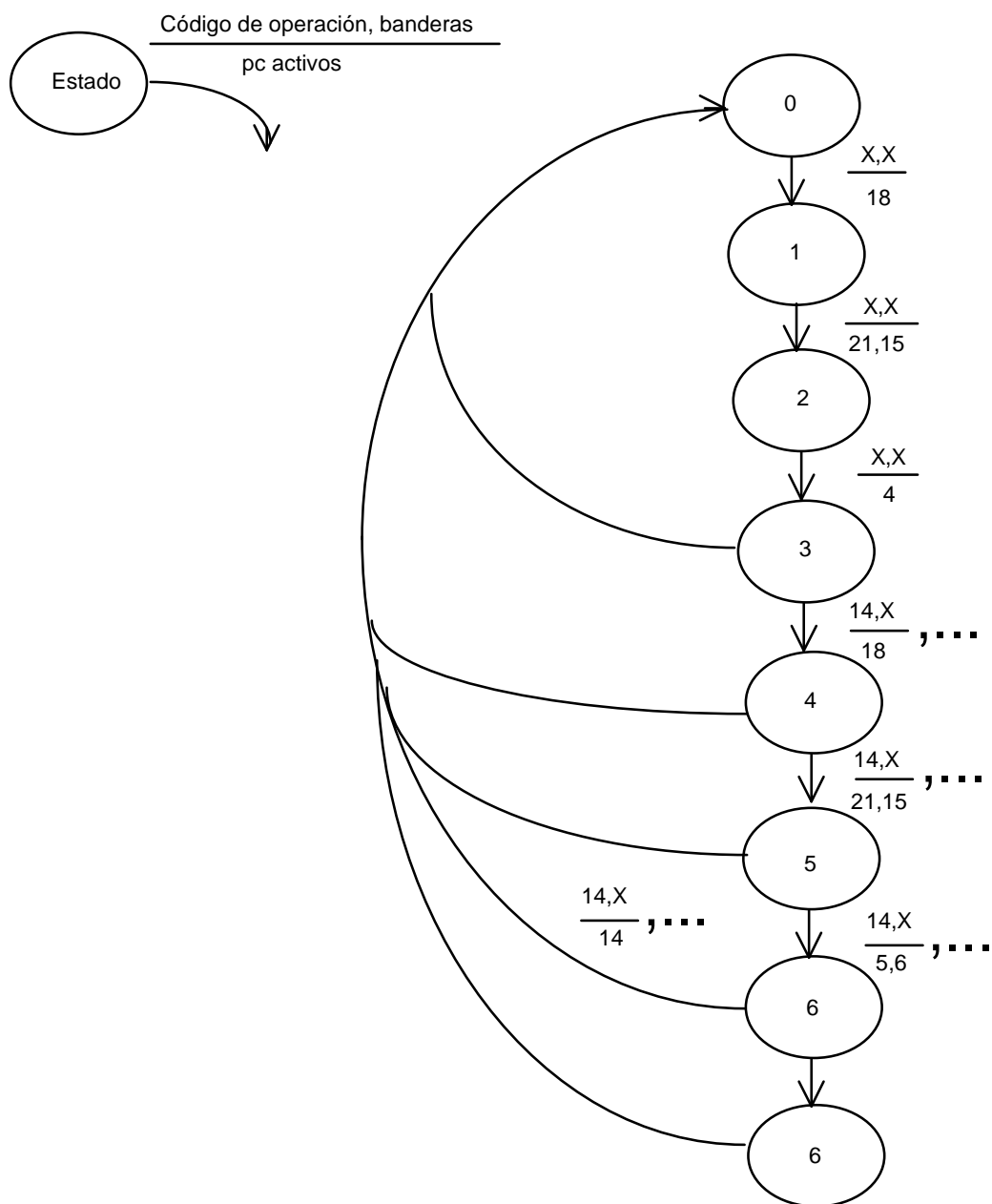


Figura 23.4 Fracción del diagrama de estados para el modelo de Mealy.

El diagrama de estados de la figura 23.4 puede ser construido como muestra en la figura 23.5.

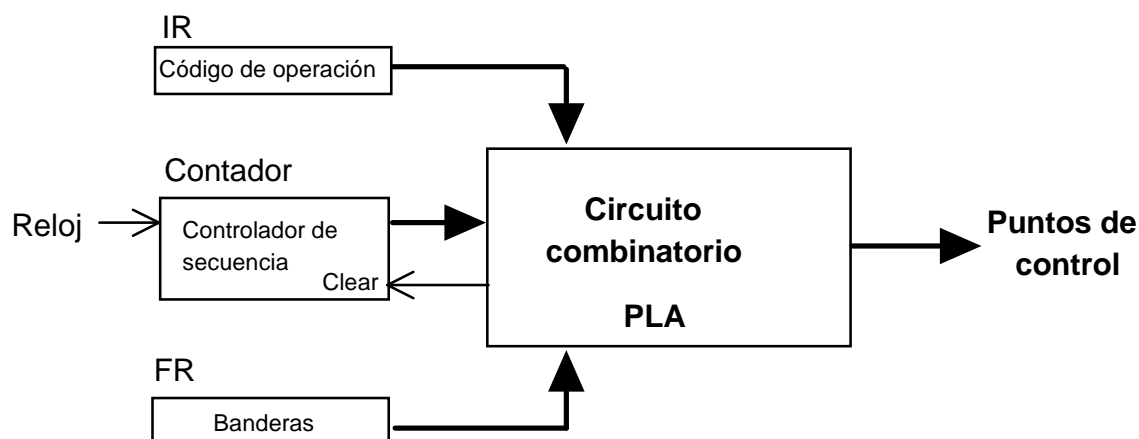


Fig 23.5 Unidad de control construida con solo hardware con el modelo de Mealy

La forma de operar de esta unidad de control es la siguiente: cuando el contador de secuencia se encuentra en el estado cero se inicia el ciclo de búsqueda, activándose el punto de control 18 ($[MAR] \leftarrow [PC]$). Al pasar el contador de secuencia al estado uno, se activan los puntos de control 21 y 15 ($Mread$ y $[PC] \leftarrow [PC] + 1$). Finalmente, al pasar al estado dos, se activa el punto de control 4 ($[IR] \leftarrow [MDR]$).

Al tener un código de operación en el IR, la lógica combinatoria generará la secuencia adecuada de puntos de control para ejecutar la instrucción, al ritmo indicado por el contador de secuencia y, al terminar el ciclo de ejecución, se borra el contador de secuencia para iniciar el ciclo de búsqueda de la siguiente instrucción.

Para ejemplificar un ciclo de ejecución, suponga que en el IR se tiene la instrucción maquina ADDA con direccionamiento inmediato (Código de operación 14) y el contador de secuencia está en el estado tres. Con estas entradas, el circuito combinatorio debe de activar el punto de control 18 ($[MAR] \leftarrow [PC]$). Al llegar un pulso en la señal de reloj el contador de secuencia se incrementará y ahora tendrá el valor de cuatro, lo que ocasiona que el circuito combinatorio active los puntos de control 21 y 15 ($Mread$ y $[PC] \leftarrow [PC] + 1$). Al incrementarse de nuevo el contador de secuencia, el circuito combinatorio activará los puntos de control 6 y 5 ($[AUX] \leftarrow ALU_1 \leftarrow [A]$, $[AUX] \leftarrow ALU \leftarrow [MDR]$), y al incrementarse a seis, se activará el punto de control 14 ($[A] \leftarrow [AUX]$) para terminar el ciclo de ejecución. En este momento se borra el contador de secuencia, lo que ocasiona que se inicie el ciclo de búsqueda de la siguiente instrucción.

El circuito combinatorio normalmente es construido con un arreglo lógico programado (PLA) como los definidos en el capítulo 15. El controlador de secuencia es simplemente un contador síncronico de cero al número máximo de estados requerido, como los definidos en el capítulo 19.

23.4. Diseño de la unidad de control para la computadora simplificada usando solo firmware.

La unidad de control también puede ser construida usando firmware. Para ésto se utiliza un circuito semejante al definido en el capítulo 21. Este tipo de unidades de control se conocen como unidades microprogramadas. En la figura 23.6 se tiene el circuito básico para su construcción.

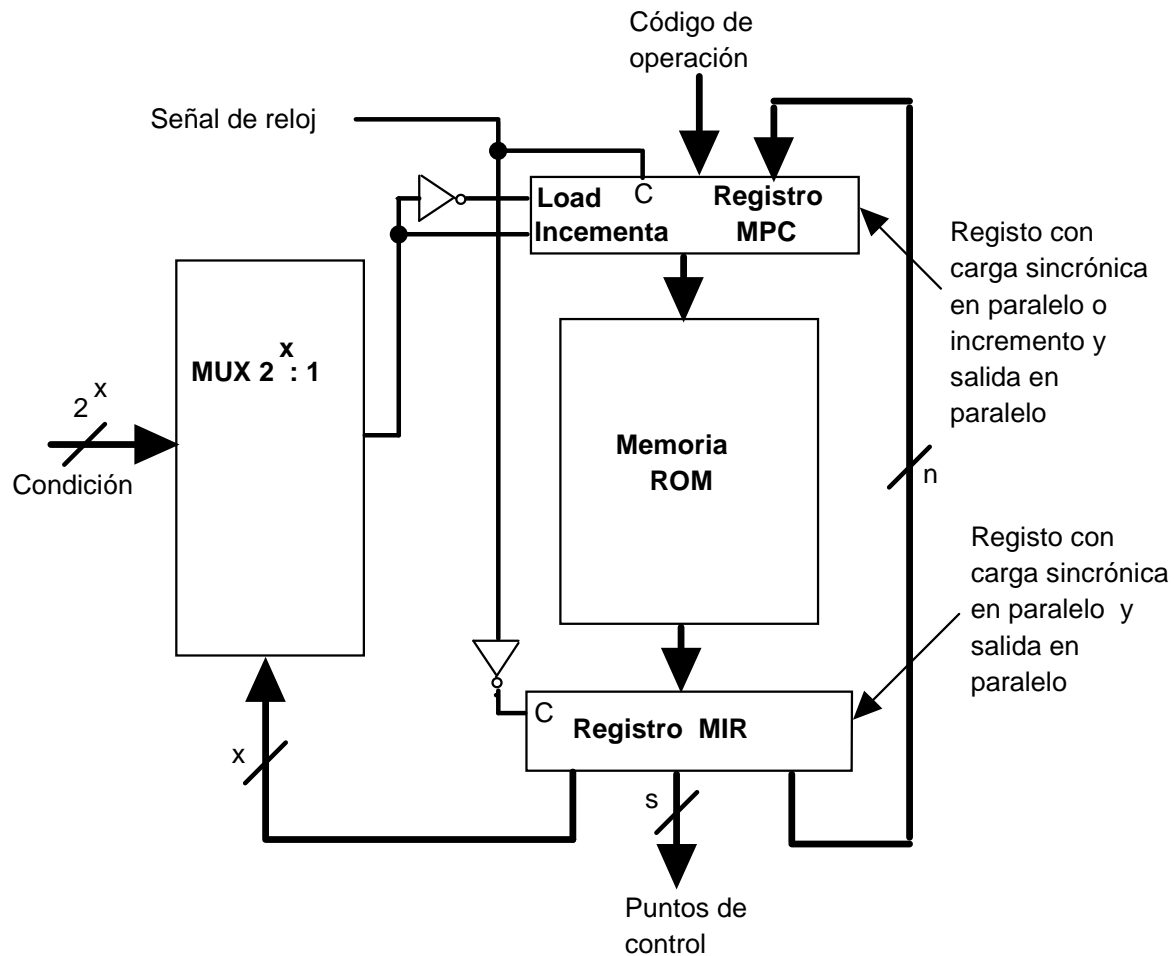


Figura 23.6 Circuito Básico de la unidad de control microprogramada

En la memoria ROM se encuentran almacenadas las microinstrucciones que controlan el funcionamiento del circuito y producen los valores a asignar en los puntos de control. El registro que contiene la dirección de la microinstrucción a ejecutar, se llama MPC (Micro Program Counter). Este es un registro que puede incrementar su valor en uno o cargar un nuevo valor en cada transición positiva en la señal del reloj. El registro que almacena la microinstrucción que se está ejecutando se llama MIR (Micro Instruction Register). Este registro carga un nuevo valor en todas las transiciones negativas en la señal del reloj.

Existen diferentes formatos de microinstrucción. En cada uno de éstos se define la forma de generar los valores de los puntos de control y la forma de indicar la dirección de la siguiente microinstrucción a ejecutar. A continuación se analizarán algunos de estos formatos.

Formato #1:

X bits	S bits	N bits
Condición	Puntos de control	Siguiente dirección

En este formato, el campo de condición sirve para indicar la acción a realizarse en el MPC, la cual puede ser:

- Incrementar su valor.
- Cargar el valor que se encuentra en el campo de siguiente dirección bajo una condición (bandera del registro FR), o siempre.
- Cargar el valor que se encuentra en el registro IR (código de operación).

El cargar en el MPC el código de operación permite a la unidad de control conocerlo y poder realizar acciones diferentes para cada instrucción de máquina.

En el campo de puntos de control se encuentran un bit por cada punto de control que tenga la arquitectura de la computadora. En cada bit se tiene un 1 o un 0 para activar o no activar el punto de control correspondiente durante el tiempo que la instrucción se encuentra en el MIR.

Formato #2:

X bits	s bits	N bits
Condición	Número de pc	Siguiente dirección

Los campos de condición y siguiente dirección tienen la misma función que en el formato #1. En el campo número de pc se indica el número en binario del punto de control a activarse durante el tiempo que la instrucción se encuentra en el MIR. En este formato solo se puede activar un punto de control a la vez.

Formato #3:

X bits	d bits	N bits
Condición	Dirección	Siguiente dirección

Los campos de condición y siguiente dirección tienen la misma función que en el formato #1. En el campo de dirección se tienen una dirección a otra memoria ROM de M palabras de S bits. Cada palabra de esta ROM tiene un bit por cada punto de control que tenga la arquitectura de la computadora, en cada bit se tiene un 1 o un 0 para activar o desactivar el punto de control correspondiente durante el tiempo que la microinstrucción que hace referencia a esta palabra se encuentra en el MIR. A cada palabra de la nueva ROM se les llama nonoinstrucción.

Formato #4:

1 bits	N bits	
tipo ₁	Puntos de control	
tipo ₀	condición	Siguiente dirección

En este formato se tienen dos tipos de microinstrucciones, una para aplicar puntos de control y el otro para hacer micros saltos incondicionales o condicionados al valor de una de las banderas del registro FR.

Cuando se utiliza una microinstrucción con el formato #1 se dice que la microprogramación es horizontal o sin empacar. Se conoce como microprogramación vertical o empacada cuando se utiliza una microinstrucción con el formato #2. Puedne existir microinstrucciones con un formato donde se tenga una mezcla de ambas.

El formato #3 es muy semejante al formato #1. El propósito de éste es reducir el tamaño de la memoria ROM, ya que en la ROM que se agrega no se repiten las nanoinstrucciones.

El formato #4 es semejante al formato #1, solo que en este caso no se pueden realizar dos acciones al mismo tiempo. En un instante determinado de tiempo se da valor a los puntos de control o se hace un salto. Este formato reduce el tamaño de la microinstrucción pero aumenta el tiempo de ejecución de una instrucción maquina, ya que todos los micros saltos consumirán tiempo adicional.

Se utilizará el formato #4 para construir la unidad de control microprogramada de la computadora simplificada, aún y cuando este formato no es el más eficiente. Sin embargo, se considera el más dicático para presentar los conceptos de microprogramación. Los ejemplos de firmware que se dan en el capítulo 21 y el diseño de esta unidad de control pueden servir de guía si se quiere construir una unidad de control con alguno de los otros formatos de microinstrucciones.

A continuación se define el formato de la microinstrucción que se utilizará.

23	22	20	0
1	Puntos de control (22-0)		
0	condición	00000000	Siguiente dirección (12 bits)

El tamaño de la microinstrucción es de 24 bits (23-0), si el bit 23 de la microinstrucción es 1, en los bits del 0 al 22 se tienen los valores que se aplican a los puntos de control en el orden definido en la tabla 23.1. Después de aplicar los valores a los puntos de control, se incrementa el contenido del MPC para ejecutar la siguiente microinstrucción.

Si el bit 23 vale 0, la microinstrucción corresponde a un salto. Dependiendo de los 3 bits de condición, se hacen las siguientes acciones:

Condición	Acción
000	Se incrementa el MPC. Condición usada cuando se aplican puntos de control para ejecutar las microinstrucciones en forma secuencial.
001	El MPC se carga con los 12 bits menos significativos del MIR.
010	Si la bandera Z = 0, el MPC se carga con los 12 bits menos significativos del MIR. Si no el MPC solo se incrementa.
011	Si la bandera N = 0, el MPC se carga con los 12 bits menos significativos del MIR. Si no el MPC solo se incrementa.
100	Si la bandera V = 0, el MPC se carga con los 12 bits menos significativos del MIR. Si no el MPC solo se incrementa.
101	Si la bandera C = 0, el MPC se carga con los 12 bits menos significativos del MIR. Si no el MPC solo se incrementa.
110	No definida
111	Los 8 bits menos significativos del MPC se carga con el código de operación que se encuentre en el IR. Los 4 bits más significativos se cargan con ceros.

La condición 001 es utilizada para realizar micros saltos incondicionales. Las condiciones 010, 011, 100 y 101 se usan para realizar micros saltos condicionados al valor de una bandera. Si no se cumple la condición, el micros salto no se realiza y se incrementa el valor del MPC.

La condición 111 es un salto especial que representa el mecanismo que se tiene para que la unidad de control conozca el código de operación de la instrucción que se tiene que ejecutar. Al realizarse esta microinstrucción quedará en el MPC una dirección entre 000_{16} a $0FF_{16}$ inclusive, dependiendo del código de operación que se encuentre en el IR. Este conjunto de direcciones deben ser reservadas, para colocar en cada una de ellas una microinstrucción de salto incondicional a la dirección donde inicia el microcódigo del ciclo de ejecución para cada una de las instrucciones de máquina.

En la figura 23.7 se muestra el circuito de la unidad de control. La memoria ROM es de 4K palabras de 24 bits. Esta memoria se conoce como memoria de control porque es donde se encuentran almacenadas las microinstrucciones que controlan el funcionamiento de la computadora. El registro MPC es un registro que puede incrementar su valor en uno o cargar un nuevo valor en cada transición positiva en la señal del reloj. Las fuentes para el valor a cargar en el MPC pueden ser dos. El modulo denominado “selector de fuente” en el circuito es el que selecciona la fuente. Este módulo verifica si la condición que se encuentra en el MIR es 111, en

cuyo caso selecciona como fuente el código de operación concatenándole 4 bits de ceros en la parte más significativa. Si se encuentra otra condición, la fuente serán los 12 bits menos significativos del MIR.

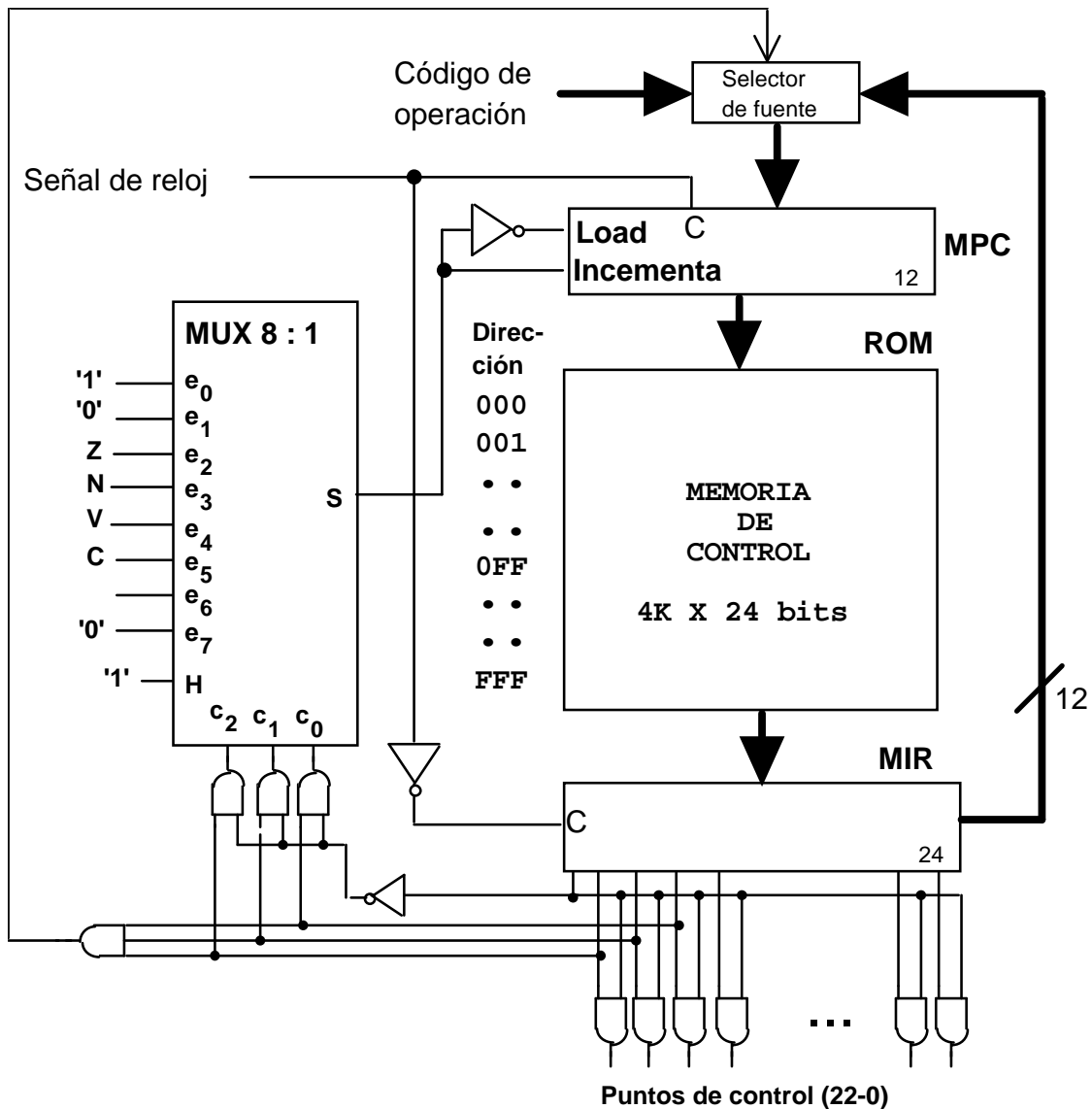


Figura 23.7 Circuito de la unidad de control microprogramada

El registro MIR carga un nuevo valor en todas las transiciones negativas en la señal del reloj y, dependiendo del bit más significativo de éste, se tomará la decisión de aplicar valores a los puntos de control o la posibilidad de realizar un microsalto.

La figura 23.8 muestra el esquema general de la memoria de control. En ésta, se puede observar que las primeras 255 direcciones (000_{16} a la $0FF_{16}$) se encuentran reservadas, una para cada

instrucción maquina. En cada una de éstas se coloca un microsalto incondicional a la dirección donde inicia el código para el ciclo de ejecución. Las direcciones correspondientes a los códigos de operación no definidos no se usan.

A partir de la dirección 256 (100_{16}) se encuentran las microrrutinas para ejecutar cada una de las instrucciones.

Memoria de control																															
Dirección ₁₆	2 2 2 2 1																														

Figura 23.8 Esquema general de la memoria de control.

En la figura 23.9 se muestra las microrrutinas del NOP, ADDA, con direccionamiento inmediato, JMP con direccionamiento directo y JMN con direccionamiento absoluto.

Memoria de control																									
Dirección ₁₆	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	
000	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	Salto incondicional a 100 (instrucción NOP)
001																									No utilizada, no hay código de operación 01
:																									
014	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	Salto incondicional a 200 (instrucción ADDA)
:																									
0B9	0	0	0	1	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	Salto incondicional a E00 (instrucción JMP)
:																									
0C3	0	0	0	1	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	Salto incondicional a F00 (instrucción JMN)
:																									
0FF																									
100	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Inicio instrucción NOP (solo FETCH), [MAR]← [PC]
101	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Mread, [PC] ← [PC]+1
102	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	[IR]← [MDR]
103	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Salto especial ([MPC] ← Código de operación)
:																									
1FF																									
200	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Inicio instruc. ADDA (dir. inmediato), [MAR]← [PC]
201	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Mread, [PC] ← [PC]+1
202	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	[AUX] ← ALU ₁ ← [A], [AUX] ← ALU ← [MDR]
203	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	[A] ← [AUX]
204	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	Salto incondicional a 100 (ciclo de FETCH)
:																									
DFF																									
E00	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Inicio instruc. JMP (dir. directo), [MAR]← [PC]
E01	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Mread
E02	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	[PCH] ← 0, [PCL] ← [MDR]
E03	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	Salto incondicional a 100 (ciclo de FETCH)
:																									
EFF																									
F00	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	Inicio instruc. JMN (dir. absoluto) Si N =0 salta a 100 (ciclo de FETCH)
F01	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	[MAR]← [PC]
F02	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Mread
F03	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	[MAR]← [MAR] + 1, [PCH] ← [MDR],
F04	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Mread
F05	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	[PCL] ← [MDR]
F06	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	Salto incondicional a 100 (ciclo de FETCH)
:																									
FFF																									

Figura 23.9 Ejemplo de memoria de control

Se analizarán primero las acciones que se llevan a cabo para ejecutar el NOP. La última microinstrucción del ciclo de FETCH que se realiza para traer el código de operación de esta instrucción maquina debe de ser un salto especial ([IR] → [MPC]). Dado que el código de operación de esta instrucción es 00₁₆, el MPC se carga con la dirección 000₁₆ la cual será la dirección de la primera microinstrucción a ejecutar. La microinstrucción almacenada en la

dirección 000_{16} es un salto incondicional a la dirección 100_{16} , por lo cual, se transferirá el control a esa dirección. Dado que la instrucción NOP no tiene ciclo de ejecución, a partir de la dirección 100_{16} iniciará el ciclo de búsqueda. En esta dirección se encuentra una microinstrucción para asignar valores a los puntos de control, en la cual se activa el punto de control 18, el cual indica que se pase el contenido del PC al MAR. Al ejecutarse este tipo de microinstrucciones el MPC se incrementa (la salida del MUX en la unidad de control es 1 lógico). Después de un ciclo del reloj, se ejecutará la microinstrucción en la dirección 101_{16} , que es del mismo tipo que la anterior, solo que en ésta se activan los puntos de control 21 y 15, los cuales indican una orden de lectura a memoria para transferir el código de operación de la siguiente instrucción al MDR e incrementar el contenido del PC. En el siguiente ciclo de reloj se ejecuta la microinstrucción en la dirección 102_{16} , que también es del mismo tipo, solo que ahora se activa el punto de control 4 para que se pase el contenido del MDR al IR. Al terminar la ejecución de esta microinstrucción, se termina el ciclo de búsqueda. En el siguiente ciclo de reloj se ejecutará la microinstrucción que se encuentra en la dirección 103_{16} , la cual es un salto especial, haciendo que el MPC se cargue con un valor entre 000_{16} y $0FF_{16}$ dependiendo del código de operación de instrucción maquina que se haya traído en el ciclo de búsqueda.

Suponiendo que se hizo la búsqueda de un ADDA con direccionamiento inmediato (código de operación 14_{16}), el MPC tendrá en este momento la dirección 014_{16} . En esta dirección se encuentra una microinstrucción de salto incondicional a la dirección 200_{16} , por lo cual se transferirá el control a dicha dirección.

A partir de la dirección 200_{16} se encuentra el ciclo de ejecución para la instrucción ADDA con direccionamiento inmediato. La primera microinstrucción activa el punto de control 18, el cual pasa el contenido del PC al MAR. En el siguiente ciclo de reloj se ejecuta la microinstrucción en la dirección 201_{16} , la cual activa los puntos de control 21 y 15, los cuales indican una orden de lectura a memoria para transferir el dato a sumar al MDR e incrementar el contenido del PC para que apunte al inicio de la siguiente instrucción maquina. En la dirección 202_{16} se encuentra la microinstrucción a ejecutar en el siguiente ciclo de reloj. En esta se activan los puntos de control 6 y 5, los cuales guardan en el registro AUX, el resultado de la suma del valor del acumulado y el dato que se acaba de leer. En el siguiente ciclo de reloj se ejecuta la microinstrucción de la dirección 203_{16} , la cual solo activa el punto de control 14, para guardar el resultado de la suma, que se encuentra en el registro AUX, en el acumulador A. Al terminar la ejecución de esta microinstrucción, se concluye el ciclo de ejecución. La siguiente actividad a realizar por la unidad de control sería un ciclo de búsqueda para lo cual se tienen dos opciones: una es remicroprogramar el ciclo de búsqueda a partir de la dirección 204_{16} , y la otra es realizar un salto incondicional al inicio del ciclo de búsqueda que ya se microprogramó, como lo hace la microinstrucción que se encuentra en la dirección 204_{16} .

Las demás microrutinas de los ciclos de ejecución de saltos se analizan de la misma forma. La microrutina de la instrucción JMP con direccionamiento directo inicia en la dirección $E00_{16}$. En ésta solo se obtiene de memoria la parte baja de la dirección a donde se va a transferir la ejecución y se guarda en el PCL, en el PCH se guardan ceros. Por último, se pasa al ciclo de búsqueda. La microrutina de la instrucción JMN con direccionamiento absoluto inicia en la dirección $F00_{16}$. En ésta, lo primero que se hace es preguntar por la bandera N; si no está encendida, el salto se realiza al inicio del ciclo de búsqueda. Si la bandera está encendida,

primero se obtiene de memoria la parte alta de la dirección a donde se va a transferir la ejecución y se almacena en el PCH; después se obtiene la parte baja y se almacena en PCL. Por último, se realiza un salto incondicional al inicio del ciclo de búsqueda.

En la figura 23.10 se muestra otra parte de la memoria de control en la que se tienen las microrutinas para la instrucción ADDA con direccionamiento directo y con direccionamiento absoluto.

Memoria de control		Comentario
Dirección ₁₆		
000	0 0 0 1 0	
:		
014	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Salto incondicional a 200 (instrucción ADDA-I)
015	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0	Salto incondicional a 205 (instrucción ADDA-D)
016	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Salto incondicional a 20C (instrucción ADDA-A)
:		
0FF		
100	1 0 0 0 0 0 1 0	Inicio instrucción NOP (solo FETCH), [MAR]← [PC]
:		
205	1 0 0 0 0 0 1 0	Inicio instruc. ADDA (dir. directo), [MAR]← [PC]
206	1 0 1 0 0 0 0 0 0 1 0	Mread, [PC]← [PC] + 1
207	1 0 0 1 0	[MAR]← 0,[MDR]
208	1 0 1 0	Mread
209	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0	[AUX] ← ALU ₁ ← [A], [AUX] ← ALU ← [MDR]
20A	1 0 0 0 0 0 0 0 0 0 1 0	[A] ← [AUX]
20B	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Salto incondicional a 100 (ciclo de FETCH)
20C	1 0 0 0 0 0 1 0	Inicio instruc. ADDA (dir. absoluto), [MAR]← [PC]
20D	1 0 1 0 0 0 0 0 0 1 0	Mread, [PC]← [PC] + 1
20E	1 0 1 0 0 0 0 0 0 0 0	[AUX]← [MDR]
20F	1 0 0 0 0 0 1 0	[MAR]← [PC]
210	1 0 1 0 0 0 0 0 0 1 0	Mread, [PC]← [PC] + 1
211	1 0 0 0 1 0	[MAR]← [AUX],[MDR]
212	1 0 1 0	Mread
213	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0	[AUX] ← ALU ₁ ← [A], [AUX] ← ALU ← [MDR]
214	1 0 0 0 0 0 0 0 0 0 1 0	[A] ← [AUX]
215	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Salto incondicional a 100 (ciclo de FETCH)
:		
FFF		

Figura 23.10 Ejemplo de microrutinas

La acción que se realiza al llegar un RESET es cargar un valor determinado (FF00₁₆) en el PC, borrar todos los registros de la computadora, incluyendo los de la unidad de control, lo cual obliga a realizar una búsqueda de la primera instrucción maquina de la rutina del RESET.

23.5. Diseño de la unidad de control microprogramada para la computadora definida en el capítulo anterior.

La arquitectura de la computadora que se definió en el capítulo anterior se muestra en la figura 23.11.

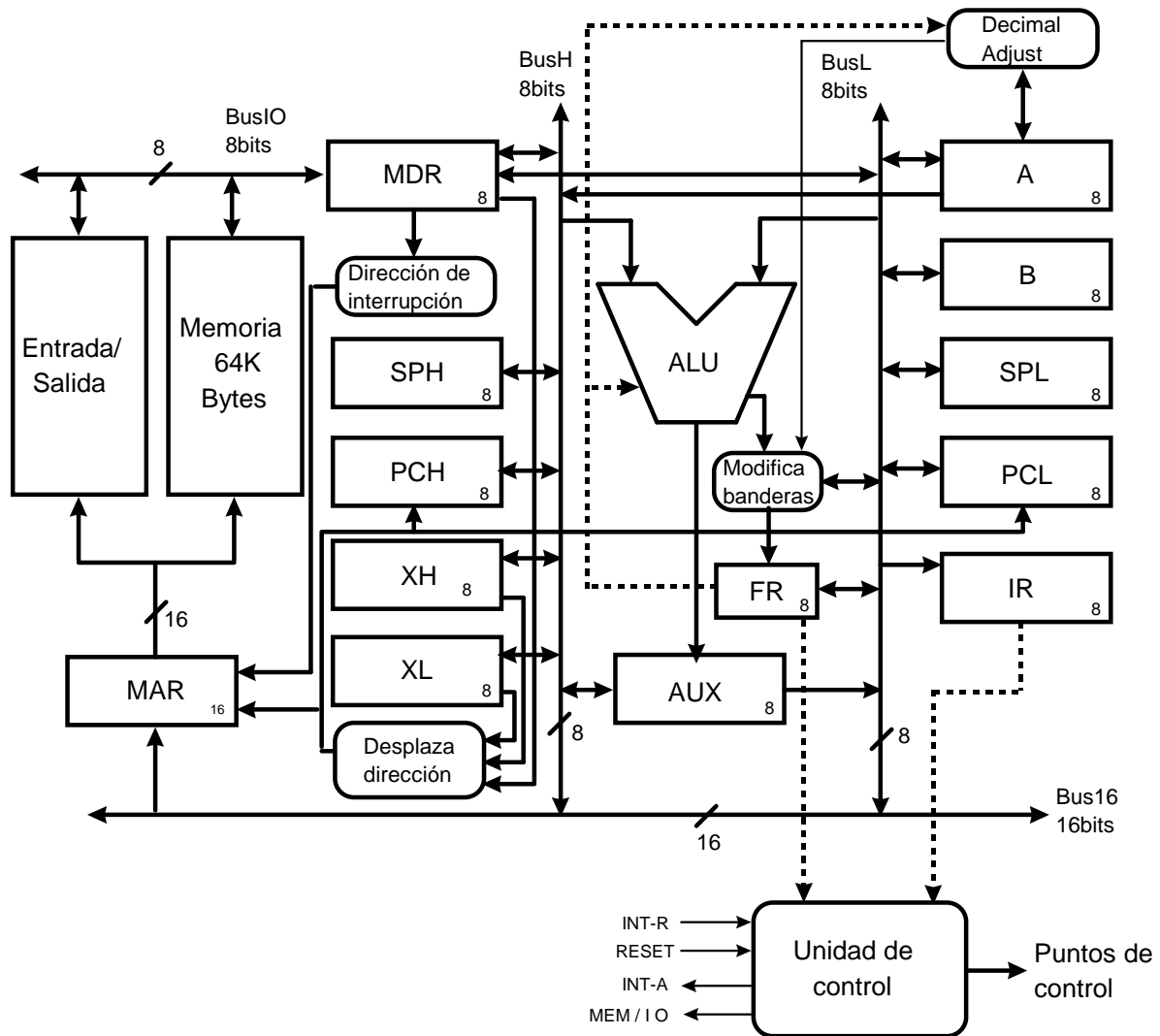


Figura 23.11 Arquitectura de la computadora.

En el diseño de la unidad de control microprogramada para esta computadora se utilizarán microinstrucciones con el formato #4 como en la sección anterior, solo que la microprogramación será de una forma más vertical (empacada).

El formato de la microinstrucción:

36															0				
1	BusH	BusL	A	B	SPH	SPL	PCH	PCL	XH	XL	FR	IR	AUX	MAR	MDR	ALU	ESP		
x	xxx	xxx	xxx	xx	x	xx	x	xx	x	x	xx	xxx	xx	xxxx	xxx				
1	Condición		000000000000000000000000												Siguiete dirección				
x	xxx														xxxxxxxxxxxx				

El tamaño de la microinstrucción es de 37 bits (36-0), si el bit 36 de la microinstrucción es 1, en los bits del 0 al 35 se tienen los siguientes campos:

Campo **BusH**, en éste se indica el registro que pone información en el bus, los valores para este campo son los siguientes:

Valor	Fuente
000	Ninguno
001	MDR
010	A
011	SPH
100	PCH
101	AUX
110	XH
111	XL

Campo **BusL**, en éste se indica el registro que pone la información en el bus, los valores para este campo son los siguientes:

Valor	Fuente
000	Ninguno
001	MDR
010	A
011	SPL
100	PCL
101	AUX
110	B
111	FR

Recuerde que la información que se encuentra en el Bus16 en todo momento, es la concatenación de las informaciones del BusH y BusL. Adicionalmente, si no se pone información en el BusH o BusL éstos contienen puros ceros lógicos.

Campo A: en éste se especifican las funciones sincrónicas que realiza el acumulador A. Los valores para este campo son los siguientes:

Valor	Función
000	Ninguno

001	Carga de BusL
010	Complemento a dos
011	Carga de DA (decimal Adjust)
100	Corrimiento Lógico hacia la derecha
101	Corrimiento Aritmetico hacia la derecha
110	Corrimiento Lógico hacia la izquierda
111	Corrimiento circular hacia la derecha

Campo **B**: en éste se especifican las funciones sincrónicas que realiza el acumulador B. Los valores para este campo son los siguientes:

Valor	Función
00	Ninguno
01	Carga de BusL
10	Complemento a dos

Campo **SPH**: en éste se especifican las funciones sincrónicas que realiza el registro SPH. Los valores para este campo son los siguientes:

Valor	Función
0	Ninguno
1	Carga de BusH

Campo **SPL**: en éste se especifican las funciones sincrónicas que realiza el registro SPL. Los valores para este campo son los siguientes:

Valor	Función
00	Ninguno
01	Carga de BusL
10	Incrementar SP
11	Decrementar SP

El incremento y el decremento se hacen a todo el registro.

Campo **PCH**: en éste se especifican las funciones sincrónicas que realiza el registro PCH. Los valores para este campo son los siguientes:

Valor	Función
-------	---------

0	Ninguno
1	Carga de BusH

Campo **PCL**: en éste se especifican las funciones sincrónicas que realiza el registro PCL. Los valores para este campo son los siguientes:

Valor	Función
00	Ninguno
01	Carga de BusL
10	Incrementar PC
11	Carga del modulo desplaza dirección

Las operaciones de incremento y carga del modulo desplaza dirección, se realizan en todo el registro.

Campo **XH**, en éste se especifican las funciones sincrónicas que realiza el registro XH. Los valores para este campo son los siguientes:

Valor	Función
0	Ninguno
1	Carga de BusH

Campo **XL**, en éste se especifican las funciones sincrónicas que realiza el registro XL, los valores para este campo son los siguientes:

Valor	Función
00	Ninguno
01	Carga de BusH
10	Incrementar X
11	Decrementar X

El incremento y el decremento se hace a todo el registro.

Campo **FR**, en éste se especifican las funciones sincrónicas que realiza el registro FR, los valores para este campo son los siguientes:

Valor	Función
0	Ninguno
1	Carga de BusL

Campo **IR**, en éste se especifican las funciones sincrónicas que realiza el registro IR. Los valores para este campo son los siguientes:

Valor	Función
0	Ninguno
1	Carga de BusL

Campo **AUX**, en éste se especifican las funciones sincrónicas que realiza el registro AUX. Los valores para este campo son los siguientes:

Valor	Función
00	Ninguno
01	Carga de ALU
10	Carga de BusH
11	Carga de BusL

Campo **MAR**, en éste se especifican las funciones sincrónicas que realiza el registro MAR. Los valores para este campo son los siguientes:

Valor	Función
000	Ninguno
001	Carga de Bus16
010	Carga del modulo desplaza dirección
011	Carga del modulo dirección de interrupción
100	Incrementar
101	Carga el valor $FFFE_{16}$

Campo **MDR**, en éste se especifican las funciones sincrónicas que realiza el registro MDR. Los valores para este campo son los siguientes:

Valor	Función
00	Ninguno
01	Carga de BusH
10	Carga de BusL
11	Carga de BusIO

Campo **ALU**: en éste se especifican las operaciones que realiza la ALU. Los operandos son tomados del BusH y/o BusL. Los valores para este campo son los siguientes:

Valor	Operación
0000	Ninguna

0001	Incrementa el valor de BUSL
0010	Decrementa el valor de BUSL
0011	Suma
0100	Suma con carry
0101	Resta
0110	Resta con carry
0111	NOT el valor de BUSL
1000	AND
1001	OR
1010	EX-OR

Campo **Esp**: en éste se especifican las señales de coordinación con memoria o dispositivos así como el encendido y apagado de la bandera I. Los valores para este campo son los siguientes:

Valor	Función
000	Ninguno
001	Mread
010	Mwrite
011	IOrread
100	IOwrite
101	Bandera I = 1
110	Bandera I = 0
111	Señal INT-A = 1

En este tipo de microinstrucción cada campo es independiente y puede activarse más de un campo a la vez.

El otro tipo de microinstrucción es para realizar saltos,. El bit 36 de esta microinstrucción vale 0 y, dependiendo de los 3 bits de condición, se realizan las siguientes acciones:

Condición	Acción
000	Se incrementa el MPC. Condición usada cuando se aplican puntos de control para ejecutar las microinstrucciones en forma secuencial.
001	El MPC se carga con los 12 bits menos significativos del MIR.
010	Si la bandera Z = 0, el MPC se carga con los 12 bits menos significativos del MIR. Si no se el MPC se solo se incrementa.
011	Si la bandera N = 0, el MPC se carga con los 12 bits menos significativos del MIR. Si no se el MPC se solo se incrementa.
100	Si la bandera V = 0, el MPC se carga con los 12 bits menos significativos del MIR. Si no se el MPC se solo se incrementa.

101	Si la bandera C = 0, el MPC se carga con los 12 bits menos significativos del MIR. Si no se el MPC se solo se incrementa.
110	Si la bandera I = 0, el MPC se carga con los 12 bits menos significativos del MIR. Si no se el MPC se solo se incrementa
111	Los 8 bits menos significativos del MPC se carga con el código de operación que se encuentre en el IR. Los 4 bits más significativos se cargan con ceros.

En la figura 23.12 se muestra el circuito de la unidad de control. El funcionamiento de ésta es muy semejante a la unidad de control de la sección anterior. Se agregaron la posibilidad de preguntar por el AND entre la bandera I y la señal de interrupción y el módulo de decodificadores para generar las señales de los puntos de control.

ARCHIVO EN EXCEL (MCONTROL)

Figura 23.12 Ejemplo de memoria de control

Se analizarán primero las acciones que se llevan a cabo para ejecutar la instrucción NOP. La última microinstrucción del ciclo de búsqueda que se realiza para traer el código de operación de esta instrucción maquina debe de ser un salto especial ($[IR] \rightarrow [MPC]$). Dado que el código de operación de esta instrucción es 00, el MPC se carga con la dirección 000_{16} y será la microinstrucción almacenada en esta dirección la primera a ejecutarse. Si se analiza la microinstrucción almacenada en la dirección 000_{16} , se verá que corresponde a un salto incondicional a la dirección 100_{16} , por lo cual se transferirá el control a esa dirección.

Dado que la instrucción NOP no tiene ciclo de ejecución, a partir de la dirección 100_{16} se iniciará el ciclo de búsqueda de la siguiente instrucción. Antes de realizar las acciones correspondientes al ciclo de búsqueda, se pregunta si hay una interrupción pendiente. Si no la hay, se transfiere el control a la dirección $11F_{16}$ donde se inicia la búsqueda de la siguiente instrucción. Si hay una interrupción pendiente, antes de pasar el control a la rutina que la atiende se debe de guardar el estado de la máquina como se indica en el capítulo anterior, las microinstrucciones que se encuentran de la dirección 101_{16} a la 118_{16} guardan el estado de la máquina.

La microinstrucción de la dirección 101_{16} realiza las siguientes acciones: se pone en el BusH el contenido del SPH; se pone el contenido del SPL en el BusL; en el MAR se hace una carga del Bus16 (carga el contenido del SP), y se apaga la bandera que habilita interrupciones. Al ejecutarse este tipo de microinstrucciones, el MPC se incrementa (la salida del MUX en la unidad de control es 1 lógico). Después de un ciclo de reloj, la microinstrucción de la dirección 102_{16} se ejecutará. Esta es del mismo tipo que la anterior, solo que ahora se realizan las siguientes acciones: poner el contenido del acumulador A en el BusL y hacer una carga del contenido que se encuentra en el BusL en el MDR. Al ejecutar la microinstrucción de la dirección 103_{16} se escribe en memoria y se decrementa el contenido del SP. De la misma forma se guardan los registros B, FR, XL, XH, PCL y PCH en las microinstrucciones que se encuentran de la dirección 104 a la 118 .

Al llegar a la microinstrucción de la dirección 119_{16} , se activa la señal de reconocimiento de interrupción y se guarda la dirección del dispositivo en el MDR. En la microinstrucción de la dirección $11A_{16}$ se da la orden de cargar en el MAR la salida del modulo de dirección de interrupción, el cual realiza el calculo de $F000_{16} + 2_{16} * [MDR]$, que es la localidad de memoria donde se encuentra la dirección de inicio de la rutina de atención para esa interrupción. La siguiente microinstrucción lee la parte alta de la dirección de la rutina de atención y la microinstrucción de la dirección $11C_{16}$ la guarda en el PCH e incrementa el MAR. De la misma forma, las dos siguientes microinstrucciones guardan en el PCL la parte baja de la dirección de la rutina de atención.

De la dirección $11F_{16}$ a la 122_{16} se realiza el ciclo de búsqueda de la siguiente instrucción. En la primera microinstrucción se pone en el BusH el contenido de PCH; en el BusL el contenido de PCL, y en el MAR se carga el contenido del Bus16. Después de un ciclo de reloj se da la ordenes de lectura a la memoria, de cargar el MDR con el contenido del BusIO y de incrementar el PC. En el siguiente ciclo de reloj se pone en el BusL el contenido del MDR y en el IR se carga la información de este bus. Por último, al ejecutar la microinstrucción de la dirección 122_{16} , se efectúa un salto especial cargando en el MPC un valor entre 000_{16} y $0FF_{16}$,

dependiendo del código de operación de instrucción maquina que se haya traído en el ciclo de búsqueda.

En la dirección 200_{16} inicia el ciclo de ejecución de la instrucción ADD con direccionamiento inmediato. En la primera microinstrucción se pone en el BusH el contenido de PCH; en el BusL el contenido de PCL, y se indica al MAR que cargue el contenido del Bus16. Después de un ciclo de reloj se da la ordenes de lectura a la memoria, de cargar el MDR con el contenido del BusIO y de incrementa el PC. En el siguiente ciclo de reloj se pone en el BusH el contenido del MDR; en el BusL el contenido del acumulador A; se le indica al ALU que sume, y que se cargue el resultado de la ALU en el registro AUX. En la siguiente microinstrucción se pasa el contenido del registro AUX al BusL y se indica al acumulador A que guarde el contenido de este bus. Finalmente, se hace un salto incondicional a la dirección 100_{16} para que inicie un nuevo ciclo de búsqueda. Recuerde que siempre se pregunta por interrupciones pendientes antes de realizar el ciclo de búsqueda y, si existen interrupciones pendientes, se guarda el estado de la computadora y se carga en el PC la dirección de la rutina de atención de la interrupción.

La microrutina de la instrucción JMP con direccionamiento directo inicia en la dirección $E00_{16}$. En ésta solo se obtiene de memoria la parte baja de la dirección a donde se va a saltar y se guarda en el PCL, en el PCH se guardan ceros, ya que se indica al PCH que cargue la información del BusH y no se coloquen información en éste. Por último se pasa al ciclo de búsqueda. La microrutina de la instrucción JMN con direccionamiento absoluto inicia en la dirección $F00_{16}$. Lo primero que se hace es preguntar por la bandera N. Si no está encendida, el salto se realiza y se pasa al ciclo de búsqueda. Si la bandera está encendida, el salto no se realiza y se pasa a las siguientes microinstrucciones. En las primeras dos se obtiene de memoria la parte alta de la dirección a donde se va a saltar y se almacena en el PCH, en las siguientes dos se obtiene la parte baja y se almacena en PCL. Por último se pasa al ciclo de búsqueda.

En la figura 23.13 se muestra otra parte de la memoria de control la cual contiene las microrutinas para la instrucción ADDA con direccionamiento directo, con direccionamiento absoluto y con direccionamiento indexado, así como la microrutina para cargar el registro índice (X) con direccionamiento absoluto.

ARCHIVO EN EXCEL (MCONTROL2)

Figura 23.13 Ejemplo de mirorutinas

23.6. Sincronización de eventos.

La sincronización de todos los eventos que ocurren en la computadora está dada por la señal de reloj. En cada transición de la señal de reloj se pueden realizar una o varias acciones. Los registros internos en un CPU son sincrónicos y tienen entradas y salidas en paralelo, éstos pueden trabajar en la transición positiva o negativa de la señal de reloj. La figura 23.14 muestra una posible sincronización entre los diferentes elementos de la computadora.

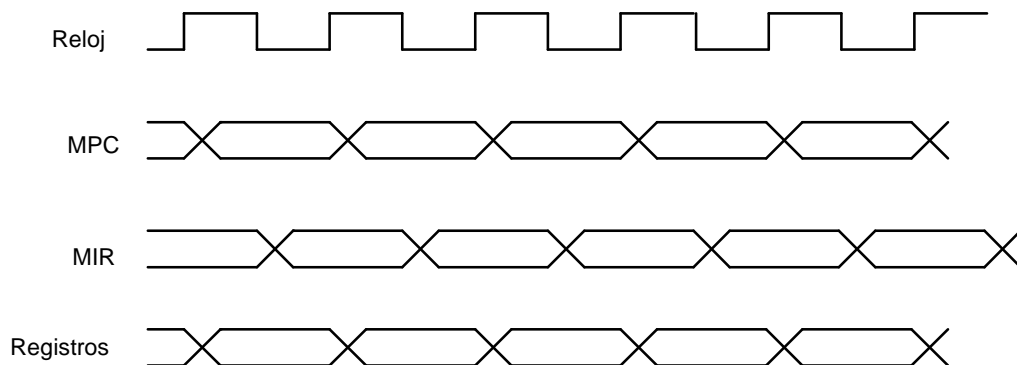


Fig 23.3 Sincronización entre la unidad de control y los registros

En el diagrama de tiempos de la figura 23.3 se indica que el MPC carga la dirección de una microinstrucción en la transición positiva. La microinstrucción correspondiente se cargará en el MIR en la siguiente transición negativa de la señal de reloj. En este momento se aplican los puntos de control de la computadora y es en la siguiente transición positiva cuando se realiza la acción. En esta misma transición el MPC está cargando un nuevo valor.

También debe existir una sincronización con la entrada y salida de información de la computadora, ya que cuando la unidad de control aplica los puntos de control para leer o escribir, en memoria o en un dispositivo periférico, ésta supone que la operación ya se realizó en el siguiente transición del reloj, de lo contrario, se deberá insertar un número fijo de ciclos de reloj para garantizar esta sincronización.

Par conocer la velocidad con la que se procesan las instrucciones en una computadora, se debe dividir la frecuencia de la señal de reloj entre el número promedio de ciclos de reloj que requiere una instrucción para ejecutarse.

Par lograr aumentar la velocidad de procesamiento de una computadora solo hay dos alternativas; una es aumentar la frecuencia de la señal de reloj, lo cual requiere tener tecnología que soporte una mayor velocidad de operación. La otra alternativa es disminuir el número de ciclos de reloj por instrucción, para lo cual se debe tener una arquitectura computacional más compleja donde se permita realizar más acciones en cada ciclo de reloj.

23.7. Resumen.

23.8. Problemas.

1. Par la máquina simplificada definida en la sección 23.2, indique la secuencias de puntos de control necesarios para realizar el ciclo de ejecución de las siguientes instrucciones maquinales:
 - a) LDA con direccionamiento inmediato.
 - b) LDA con direccionamiento directo.
 - c) LDA con direccionamiento absoluto.
 - d) STA con direccionamiento inmediato.
 - e) STA con direccionamiento directo.
 - f) STA con direccionamiento absoluto.
 - g) INA.
 - h) SUBC con direccionamiento inmediato.
 - i) SUBC con direccionamiento directo.
 - j) SUBC con direccionamiento absoluto.
 - k) NEGA
 - l) JMV con direccionamiento directo
 - m) JMV con direccionamiento absoluto
 - n) CLA

2. Modifique el diagrama de estados de la figura 23.3, para que se consideren los ciclos de ejecución de las siguientes instrucciones:
 - a) LDA con direccionamiento inmediato.
 - b) LDA con direccionamiento directo.
 - c) STA con direccionamiento inmediato.
 - d) INA.
 - e) SUBC con direccionamiento inmediato.
 - f) SUBC con direccionamiento directo.
 - g) SUBC con direccionamiento absoluto.
 - h) NEGA
 - i) JMV con direccionamiento directo
 - j) JMV con direccionamiento absoluto
 - k) CLA

3. Modifique el diagrama de estados de la figura 23.4, para que se consideren los ciclos de ejecución de las siguientes instrucciones:
 - a) LDA con direccionamiento inmediato.
 - b) LDA con direccionamiento directo.
 - c) STA con direccionamiento inmediato.
 - d) INA.
 - e) SUBC con direccionamiento inmediato.
 - f) SUBC con direccionamiento directo.

- g) SUBC con direccionamiento absoluto.
 - h) NEGA
 - i) JMV con direccionamiento directo
 - j) JMV con direccionamiento absoluto
 - k) CLA
4. Para la unidad de control microprogramada definida en la sección 23.5, microprogramela para que pueda realizar las siguientes instrucciones maquinales:
- a) LDA con direccionamiento inmediato.
 - b) LDA con direccionamiento directo.
 - c) LDA con direccionamiento absoluto.
 - d) LDA con direccionamiento indexado.
 - e) STA con direccionamiento inmediato.
 - f) STA con direccionamiento directo.
 - g) STA con direccionamiento absoluto.
 - h) STA con direccionamiento indexado.
 - i) INA.
 - j) SUBC con direccionamiento inmediato.
 - k) SUBC con direccionamiento directo.
 - l) SUBC con direccionamiento absoluto.
 - m) SUBC con direccionamiento indexado.
 - n) JMV con direccionamiento directo
 - ñ) JMV con direccionamiento absoluto
 - o) JMV con direccionamiento indexado
 - p) JSR con direccionamiento directo
 - q) JSR con direccionamiento absoluto
 - r) JSR con direccionamiento indexado
 - s) INPA
 - t) OUTA
 - u) HLT (una interrupción debe ser aceptada en este estado)

CAPITULO 24

ARQUITECTURAS COMPUTACIONALES AVANZADAS

24.1. INTRODUCCION.

De la década de los años cincuenta a la fecha, las computadoras digitales han incrementado tanto su capacidad de cómputo como de almacenamiento más allá de lo que se vislumbraba en ese entonces. Además, su costo, como su tamaño y la potencia eléctrica requerida para su operación se han reducido considerablemente. En las primeras computadoras se utilizaban bulbos y relevadores electromecánicos para la construcción de sus circuitos lógicos y se usaban núcleos de ferrita para la construcción de su memoria principal. Estas primeras computadoras eran voluminosas, súmamente caras y requerían mucha energía para operar. Gracias a los avances logrados en la electrónica, primero con la invención del transistor y posteriormente con la creación de los circuitos integrados y la microelectónica, se logró aumentar su capacidad y abatir tanto su costo como su tamaño. Sin embargo, el increíble aumento en su velocidad no solamente se debe a estos desarrollos en el área de la electrónica, sino también a las muchas innovaciones hechas a la arquitectura de las computadoras digitales como se mencionó en el capítulo anterior.

En este capítulo se presenta un panorama somero con los principales conceptos de arquitectura computacional que han contribuido a esta revolución del siglo 20. Los temas que se tratan incluyen la utilización de memoria de caché en los procesadores, el concepto de “pipeline”, las arquitecturas RISC, los procesadores vectoriales y los multiprocesadores. Dado que solamente se pretende dar un panorama de las arquitecturas avanzadas, no se verán los detalles de la construcción en esta sección. Existen varios libros que se dedican por completo a estos temas y el lector interesado puede consultarlos para lograr un mayor conocimiento referente a estos temas.

24.2. Memoria de caché.

Frecuentemente se ha dicho que el principal cuello de botella en las computadoras digitales es la memoria principal ya que el procesador tiene que traer tanto datos como instrucciones de memoria principal para la ejecución de los programas. Generalmente no se pueden traer estas instrucciones y datos de la memoria con la velocidad necesaria para que el procesador opere a su máxima velocidad, lo cual ocasiona que el procesador tenga que esperar por los datos y las instrucciones almacenadas en la memoria y trabaje a una menor velocidad de la que podría hacerlo si se tuviera una unidad de memoria más rápida. Es cierto que la velocidad de las unidades de memoria ha aumentado considerablemente, pero nunca ha alcanzado la velocidad necesaria para que los procesadores logran trabajar a su máxima velocidad.

Los investigadores en el área de arquitectura computacional observaron que durante la ejecución de un programa existen ciertas direcciones de memoria a las cuales se hace referencia mucho más frecuentemente que a otras para traer datos. Si el contenido de estas direcciones de memoria a las cuales se hace referencia con mayor frecuencia se tuvieran en una unidad de memoria más rápida, se podría lograr una mayor velocidad de ejecución de los programas. Las memorias de caché se crearon precisamente para lograr lo anterior.

Es posible construir una memoria más rápida que la memoria principal de una computadora pero esta ganancia en velocidad se obtiene a costa de su capacidad y un costo mucho mayor. Por ejemplo, las memorias de caché son de 4 a 20 veces más rápidas que la memoria principal pero su capacidad es mucho menor y su costo más alto.

Un problema que se tuvo que resolver al diseñar las memorias de caché es que las direcciones de los datos que se guardan en ellas no siempre corresponden a direcciones consecutivas de memoria principal. Probablemente se tengan varios grupos pequeños de direcciones consecutivas de memoria de memoria principal, pero las direcciones de inicio de cada uno de estos grupos están dispersas en la memoria principal. Por esta razón, las memorias de cache son memorias asociativas.

Una memoria de caché está organizada de la forma mostrada en la figura 24.1. Se tiene un campo para una etiqueta, la cual corresponde a un grupo de direcciones de memoria principal. Para el ejemplo mostrado, una etiqueta corresponde a un grupo de 8 direcciones de memoria principal. La etiqueta 173X, en octal, corresponde a las ocho direcciones de memoria principal 1730, 1731, 1732, 1733, 1734, 1735, 1736 y 1737 en octal. Asociados a esta etiqueta, se tienen los contenidos de las ocho direcciones de memoria principal mencionadas. De la misma forma, la etiqueta octal 771X corresponde a las direcciones octales 7710 a 7717.

etiqueta	datos
0173X	25, 33, 44, 55, 90, 78, 66, 88
0771X	34, 55, 66, 12, 39, 18, 22, 23
0234X	33, 44, 55, 39, 18, 22, 15, 10
-----	-----
-----	-----
0451X	11, 17, 81, 99, 92, 25, 17, 47

Figura 24.1. Organización de una memoria de caché.

Al ejecutar un programa, cada referencia hecha a una dirección de memoria se presenta primero a la memoria de caché. La unidad de control del caché busca en sus etiquetas para ver si tiene el contenido de la dirección deseada. Si la tiene, proporciona el dato al procesador, logrando un tiempo de acceso mucho menor que si se hubiera tenido que traer de la memoria principal. Si no la tiene, se produce una falta

en el caché y se trae de memoria el contenido del grupo de 8 direcciones de memoria principal al que corresponda la dirección en cuestión y se guardan en el caché, poniéndole la etiqueta correspondiente. La próxima vez que se haga referencia a alguna de estas direcciones, ya se tendrá la información en caché. Por ejemplo, si se hace referencia a la dirección octal de memoria principal 1734, el grupo de direcciones octales de 1730 a 1737 está en caché y el tiempo en que se tiene disponible el dato es pequeño. Si se hiciera referencia a la dirección octal 4415, tomado los datos mostrados en la figura 24.1, se produciría una falta en el caché y se traerían los datos correspondientes a las direcciones de memoria principal de la 4410 a la 4417, poniéndoles la etiqueta 441X.

Eventualmente, la memoria de caché se llena de información y, al haber una falta en el caché, será necesario descartar alguno de los grupos existentes para hacer espacio al nuevo grupo. La decisión de cual grupo descartar es crucial y se han utilizado varios algoritmos para decidir cual grupo quitar de la memoria de caché. Una de las políticas mas comunes es desechar el grupo de direcciones de memoria que tiene más tiempo sin ser utilizado. Al desechar un grupo de direcciones de memoria del caché cuyos valores no han sido modificados no hay que hacer nada adicional; sin embargo, si alguno de los valores ha sido modificado después de que se trajo al caché, será necesario actualizar los contenidos en la memoria principal.

Cuando se empezó a utilizar la memoria de caché, ésta se conectaba a los procesadores como una unidad independiente. Actualmente la mayoría de los procesadores que se construyen en un solo circuito integrado cuentan internamente con memoria de caché.

Inicialmente, en la memoria de caché se guardaban solamente datos; sin embargo, estos conceptos son también válidos para guardar aquellas instrucciones que se utilizan frecuentemente durante la ejecución de un programa y así poder bajar el tiempo de búsqueda de estas instrucciones. En algunos procesadores se guardan en el caché tanto datos como instrucciones. En otros, se cuenta con una memoria de caché para datos y otra para instrucciones. Finalmente, en algunos sistemas se cuenta con memorias de caché de varios niveles, con diferentes tiempos de acceso y diferentes capacidades.

24.3. Pipeline.

El concepto de pipeline es algo similar a una línea de ensamble en la cual se tienen varias estaciones y cada una realiza una operación diferente sobre los productos que se están fabricando, pero todas estas estaciones trabajan simultáneamente sobre distintos productos iguales.

Para ejemplificar el concepto, considérese la operación de multiplicación de dos números reales (números de punto flotante). Para llevar a cabo la operación es necesario realizar los siguientes pasos:

1. Suma de los exponentes.
2. Multiplicación de las mantisas.
3. Normalizar la mantisa del resultado.
4. Redondear la mantisa del resultado.

Suponiendo que cada una de las operaciones durara una unidad de tiempo, el resultado de la multiplicación de dos números de punto flotante tardaría cuatro unidades de tiempo. Si se tuviera una unidad que solamente pudiera operar sobre un par de números a la vez, se obtendría una multiplicación cada cuatro unidades de tiempo en el mejor de los casos.

Si se deseara obtener un resultado por unidad de tiempo en promedio para aumentar la velocidad de la computadora, se podrían poner cuatro unidades de multiplicación de punto flotante independientes que trabajaran simultáneamente. Estas unidades producirían cuatro resultados en cuatro unidades de tiempo, pero el costo de la multiplicación de punto flotante se cuadruplicaría.

Otra posible solución consisten en dividir la unidad de multiplicación en cuatro etapas independientes, cada una de las cuales realiza solamente uno de los pasos necesarios para multiplicar dos números de punto flotante. Si se alimentan dos números reales a la etapa 1, después de una unidad de tiempo se habría terminado la suma de los exponentes. En este momento se pasaría la operación a la etapa 2 para que realizara la multiplicación de las mantisas, pero la etapa 1 quedaría libre. Se pueden alimentar otros dos números reales a la etapa 1 para que efectúe la suma de los exponentes mientras la etapa 2 realiza la multiplicación de las mantisas de primer par de números.

Transcurrida otra unidad de tiempo, la etapa 2 habría terminado de realizar la multiplicación de las mantisas del primer par de números reales y la etapa 1 habría concluido su trabajos sobre el segundo par de números reales. Ahora se puede alimentar un tercer par de números reales a la etapa 1 mientras la etapa 2 trabaja sobre el segundo par de números reales y la etapa tres trabaja en la normalización de la mantisa del resultado del primer para de números.

Al transcurrir otra unidad de tiempo, se puede alimentar un cuarto par de números a la etapa 1 mientras la etapa 2 trabaja sobre el tercer par de números, la etapa 3 sobre el segundo y la etapa cuarto realiza el redondeo del resultado del primer par de números.

A partir de este momento, si se continúa alimentando un par de números reales a la etapa 1, se obtendría un resultado de multiplicación en cada unidad de tiempo, lográndose el objetivo deseado. Esta segunda solución constituye una unidad de multiplicación de punto flotante en pipeline. El costo de la unidad no es cuatro veces el de la unidad original aunque si es mayor debido a que se necesita hardware adicional para sincronizar las etapas así como registros adicionales para guardar resultados intermedios. En la figura 24.1 se muestra esquemáticamente la operación de esta unidad de multiplicación de punto flotante en pipeline con cuatro etapas.

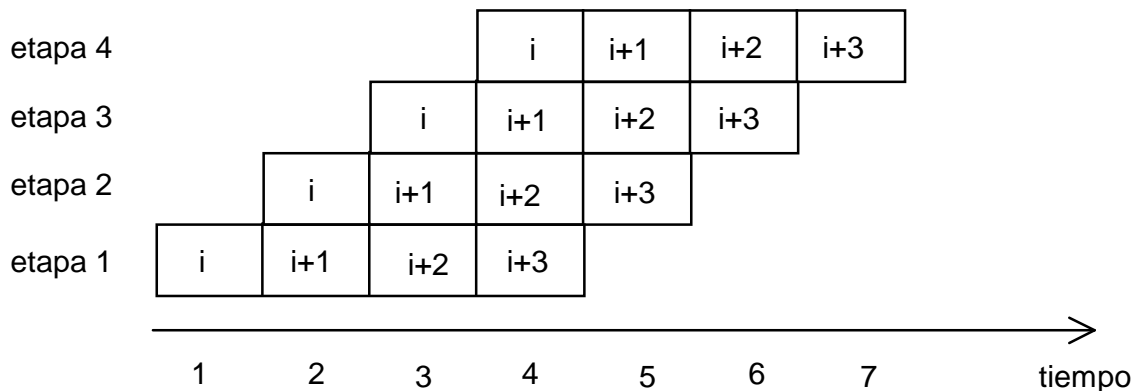


Figura 24.2. Pipeline de 4 etapas.

En la figura 24.2 se muestra que en el tiempo 1, la etapa está trabajando con el par de operandos i . En el tiempo 2, la etapa 1 trabaja sobre el par de operandos $i+1$ mientras que ahora la etapa 2 trabaja sobre el par de operandos i . En el tiempo 3, la etapa 1 trabaja sobre el par de operandos $i+2$, la etapa 2 sobre el par de operandos $i+1$ y la etapa 3 trabaja sobre el par de operandos i . Durante el tiempo 4, la etapa 1 trabaja sobre el par de operandos $i+3$, la etapa 2 sobre el par de operandos $i+2$, la etapa 3 sobre el par de operandos $i+1$ y la etapa 4 sobre el par de operandos i .

En el caso general, si se tiene una operación que tarda T segundos en efectuarse y se divide esta operación en N etapas secuenciales en pipeline con la misma duración de T/N segundos cada una, se produciría un resultado cada T/N segundos cuando el pipeline se haya llenado. Suponiendo que el pipeline está ocioso (vacío), tardaría $(N-1)T/N$ segundos en llenarse y empezar a producir un resultado cada T/N segundos. Similarmente, tardaría $(N-1)T/N$ segundos en vaciarse una vez que se dejan de alimentar datos en la primer etapa.

Al crear una unidad en pipeline para efectuar una operación es necesario dividirla en N etapas independientes con la misma duración, lo cual tiene sus complicaciones. Si las etapas no realizan su trabajo en intervalos de tiempos iguales, una etapa tendría que esperar por los resultados de la anterior y la eficiencia del pipeline bajaría. Además es necesario añadir mecanismos de sincronización en las etapas y registros adicionales para guardar resultados intermedios.

Inicialmente el concepto de pipeline se utilizó en las unidades aritméticas de las computadoras. Con el paso del tiempo se extendió su uso para realizar accesos a memoria y logran una mayor velocidad en la obtención de datos e instrucciones de memoria principal. Actualmente también se utiliza en los ciclos de búsqueda y ejecución de las instrucciones lo cual ha logrado aumentar la velocidad de los procesadores. Nótese que este concepto contribuye a aumentar la velocidad con la cual se realizan las operaciones aunque la velocidad del hardware no aumente.

Un ejemplo del uso de pipeline en la operación de una máquina es el procesador TMS320C3x de Texas Instruments. En su arquitectura, se tiene un pipeline con cuatro etapas:

- Fetch (F) Esta etapa realiza el ciclo de búsqueda de instrucciones en memoria y actualiza el registro contador de programa.
- Decode (D) Esta etapa efectúa la decodificación de la instrucción y realiza el cálculo de la dirección efectiva. También controla las modificaciones al registro apuntador a la pila y los registros auxiliares.
- Read (R) Esta etapa lee los operandos de memoria cuando se requieren.
- Execute (E) Esta etapa ejecuta la instrucción obteniendo los datos de los registros auxiliares cuando se requieren y actualizando su contenido en caso necesario. También escribe a memoria resultados de operaciones previas.

En este procesador se hace uso del concepto de pipeline realizando en paralelo las operaciones de búsqueda de instrucciones, decodificación y generación de direcciones efectivas, obtención de operandos, y ejecución de instrucciones.

No siempre es posible obtener el máximo rendimiento de este procesador debido a que pueden generarse conflictos que obligan a vaciar el pipeline y reducen su utilización. Estos conflictos pueden agruparse en tres tipos principales:

Conflictos de transferencias de control.

Conflictos con los registros auxiliares.

Conflictos de accesos a memoria.

Los conflictos de transferencias de control ocurren cuando se ejecuta una instrucción que ocasiona que se cambie el valor del registro contador de programa (PC). Esta puede ser una transferencia incondicional, una transferencia condicional que si se efectúa o una llamada a una subrutina. En cualquiera de estos casos, el contador de programa se modifica al momento de ejecución y tanto la búsqueda de las siguientes instrucciones como su decodificación, la generación de la dirección efectiva y la lectura de los operandos resultan inútiles ya que la instrucción a ejecutar está en otra dirección de memoria. En estos casos el pipeline se vacía y empieza a utilizarse eficientemente hasta que se llena.

Los registros auxiliares se utilizan para guardar resultados intermedios y para la generación de las direcciones efectivas. Pueden generarse conflictos cuando una etapa desea guardar un resultado intermedio en uno de ellos pero el registro está

siendo utilizado para la generación de la dirección efectiva. En este caso, la etapa tiene que esperar bajando la eficiencia del procesador.

Por último, pueden existir conflictos de acceso a memoria si la etapa está de ejecución está actualizando un dato en memoria y la etapa de búsqueda de instrucciones desea traer una instrucción de memoria. En este caso la etapa de búsqueda de instrucciones debe esperar, lo cual hace que baje el rendimiento del procesador.

Sin embargo, a pesar de que en ciertas ocasiones se originan conflictos, en la mayoría de los casos estos conflictos no ocurren y el procesador TMS320C3x logra un alto rendimiento.

24.4. Arquitecturas RISC.

El concepto de arquitecturas RISC (Restricted Instruction Set Computers) apareció a mediados de la década de los setentas en IBM, culminando con la construcción de una máquina llamada IBM 801. La idea original consistió en desarrollar una arquitectura que pudiera ejecutar todas sus instrucciones en un ciclo de reloj, lo cual la haría más rápida. Recuérdese que en el capítulo 23 se mostró que algunas instrucciones de una máquina convencional (denominadas CISC por Complex Instruction Set Computers) tardaban varios ciclos de reloj en ejecutarse.

Para lograr el objetivo original, se redujo el número de instrucciones de la computadora, eliminando aquellas que tomaban mucho tiempo. Sin embargo, el juego de instrucciones era completo y bien pensado de tal forma que se pudieran realizar todas las operaciones necesarias con la arquitectura. Otra característica de esta máquina era que todas sus instrucciones tenían la misma longitud en contraste con la arquitectura presentada en los capítulos 22 y 23 que incluye instrucciones de 1, 2 y 3 bytes de longitud. Esto permitiría hacer más eficiente el ciclo de búsqueda de las instrucciones.

Dado que se requería que cada instrucción tardara solamente un ciclo de reloj para realizarse, la arquitectura hacía uso extensivo del concepto de pipeline en todas sus componentes para lograr el objetivo.

Las arquitecturas RISC originales tiene sus desventajas respecto a las arquitecturas CISC. Las principales son:

- Dado que las arquitecturas RISC puras carecen de instrucciones complejas en comparación con las arquitecturas CISC, es necesario ejecutar varias instrucciones para realizar lo mismo que haría una sola instrucción de una arquitectura CISC.
- Al ejecutar más instrucciones para realizar una operación compleja, el número de accesos a memoria para los ciclos de búsqueda de las instrucciones es mayor que en una arquitectura CISC, lo cual baja su rendimiento ya que, como se mencionó al

inicio de este capítulo, los accesos a memoria representan el principal cuello de botella de las computadoras digitales.

Con el tiempo, las arquitecturas RISC han evolucionado y actualmente el término es engañoso, ya que los conceptos usados en las arquitecturas RISC hoy en día no son exactamente los mismos que las ideas que originaron su aparición.

Las arquitecturas RISC actuales si incluyen operaciones complejas en su juego de instrucciones; sin embargo, el objetivo de ejecutar una instrucción por ciclo de reloj se mantiene. También se utiliza ampliamente el concepto de pipeline en todas sus componentes tales como accesos a memoria, unidad aritmética y lógica, ciclos de búsqueda de instrucciones y en el ciclo de ejecución en forma similar a como está estructurada la arquitectura del procesador TMS320C3x descrito someramente en la sección anterior.

24.5. Arquitecturas vectoriales.

Las primeras computadoras eran utilizadas exclusivamente para resolver modelos matemáticos de sistemas físicos, estos problemas requieren hacer un cantidad muy grande de cálculos numéricos. Aunque actualmente las computadoras digitales se utilizan en prácticamente todos los ámbitos de la vida diaria, la resolución de grandes modelos de sistemas físicos sigue siendo importante para los científicos e investigadores y contribuyen en gran medida al avance de la ciencia y la tecnología.

Muchos de estos modelos matemáticos requieren que se realice una misma operación repetitivamente sobre un varios grupos de datos diferentes y, si programa en una máquina convencional, se tiene que realizar la misma operación sobre los diferentes datos en forma secuencial.

Para dar un ejemplo típico de tales modelos considere la ecuación de Poisson en dos dimensiones para calcular el potencial eléctrico en una región plana en función de la densidad de carga presente en dicha región:

$$\frac{\partial^2 V(x, y)}{\partial x^2} + \frac{\partial^2 V(x, y)}{\partial y^2} = -C(x, y)$$

En esta ecuación, $V(x,y)$ representa el poencial eléctrico en el punto cuyas coordenadas son (x,y) y $C(x,y)$ representa la carga presente en el mismo punto. El potencial eléctrico en un punto dado depende tanto de la carga presente en dicho punto como de las cargas presentes en los puntos vecinos.

Esta ecuación diferencial parcial represente un sistema continuo que necesita ser digitalizado para resolverse en una computadora digital. La distancia en la dirección x se divide en N intervalos iguales que representan las posiciones $x_0, x_1, x_2, \dots, x_N$. Lo mismo tiene que hacerse con la distancia en la dirección y . Sin entrar en detalles

respecto a su solución, se necesita resolver el sistema de ecuaciones lineales dado por:

$$\frac{V_{i+1,j} + V_{i-1,j} + V_{i,j+1} + V_{i,j-1}}{4} = V_{i,j} - \frac{C_{i,j}}{4N^2}$$

donde $V_{i,j}$ representa el potencial eléctrico en el punto (x_i, y_i) y $C_{i,j}$ representa la carga eléctrica en el mismo punto. Tanto los valores de i como los de j están en el rango de 0 a N inclusive.

Podría pensarse en utilizar un método de solución numérica del conjunto de ecuaciones basado en eliminación Gaussiana, pero este sistema de ecuaciones es en cierta forma especial ya que su representación matricial genera una matriz dispersa y los métodos de eliminación Gaussiana no son los más adecuados.

El modelo se resuelve generalmente mediante un método recursivo de solución en el cual se realizan iteraciones hasta que se obtiene la convergencia de todos los valores del potencial eléctrico en todos los puntos. En una máquina convencional se tendrían que realizar estas iteraciones en forma secuencial. Pensando en una forma de acelerar el proceso, se puede tener un arreglo de procesadores tal como se muestra en la figura 24.3, donde cada procesador realiza el mismo cálculo sobre un conjunto diferente de datos.

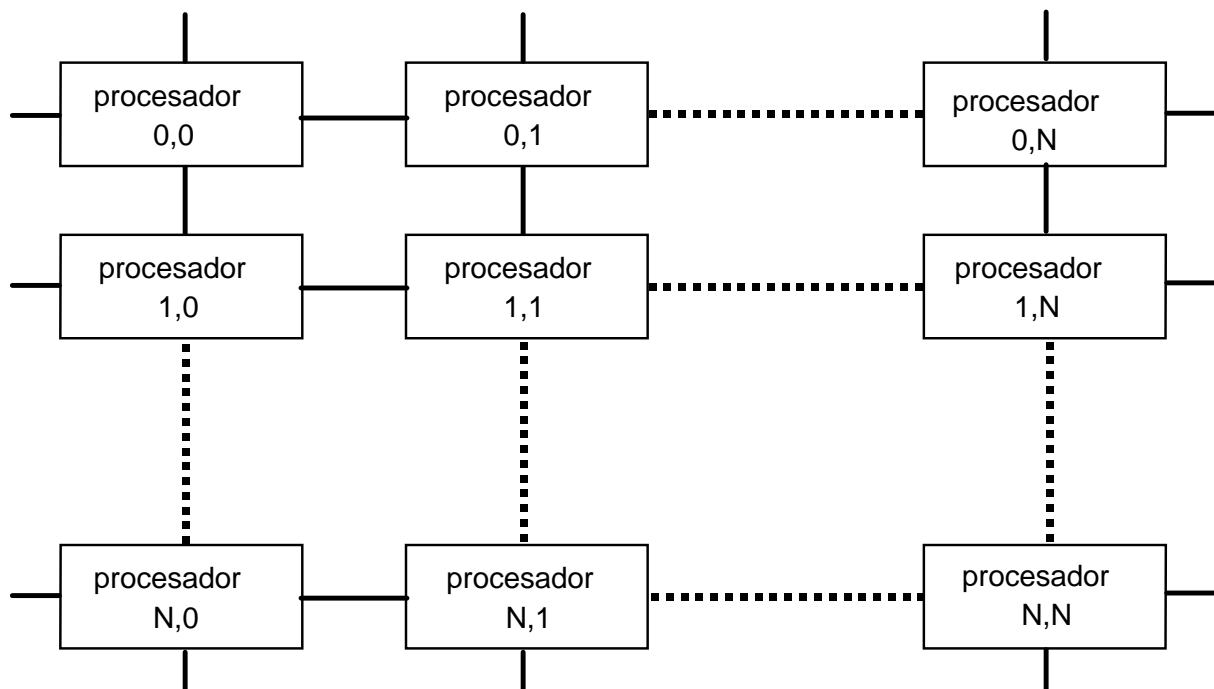


Figura 24.3. Arreglo de procesadores.

Este último método de solución dio origen a las arquitecturas vectoriales. Estas máquinas tienen muchos procesadores pero todos ellos realizan la misma operación en forma paralela sobre un conjunto de datos diferentes. Estas arquitecturas se conocen en la literatura como arquitecturas SIMD, que significa “Single Instruction stream, Multiple Data stream” ya que la secuencia de instrucciones que realiza cada procesador es la misma pero los datos sobre los que operan son diferentes.

Adicionalmente a las arquitecturas vectoriales se han realizado muchas investigaciones sobre la mejor forma de utilizar estas arquitecturas en la resolución de problemas numéricos que requieren un gran cantidad de cómputo tales como la solución de la ecuación de Poisson. Se han desarrollado compiladores que, partiendo de un programa desarrollado para una máquina convencional (también conocida como SISD por “Single Instruction stream, Single Data stream”), obtienen el código eficiente para una máquina vectorial.

Las computadoras vectoriales tienen un área de aplicación muy específica que es la solución numérica de grandes sistemas de ecuaciones en forma iterativa. Estos sistemas generalmente corresponden a soluciones numéricas de sistemas de ecuaciones diferenciales parciales. Por ejemplo, cuando el transbordador espacial reingresa a la atmósfera, se produce un calentamiento por la fricción del transbordador con el aire. Esto produce muy altas temperaturas que a su vez producen esfuerzos mecánicos en su cubierta por la expansión térmica. Para estudiar tanto la distribución de temperaturas como los esfuerzos mecánicos producidos es necesario resolver grandes sistemas de ecuaciones lineales similares a las del ejemplo presentado en esta sección.

24.6. Multiprocesadores.

Las arquitecturas vectoriales permiten lograr altas velocidades de cómputo para resolver una clase particular de problemas, pero su aplicación para otros casos no es posible. Los multiprocesadores son máquinas cuyo propósito es más general que el de los procesadores vectoriales y también fueron concebidas para lograr una mayor velocidad de cómputo.

Un multiprocesador consta de varios procesadores independientes que pueden ejecutar un conjunto de instrucciones diferente en forma paralela sobre diferentes datos. Para que estos procesadores puedan cooperar en la resolución de un problema específico, deben tener la capacidad de comunicarse entre ellos.

Existen dos filosofías principales en las arquitecturas de los multiprocesadores las cuales se ejemplifican en la figura 24.4. En el caso de memoria compartida, se tienen varios módulos de memoria y unidades de entrada y salida que son compartidos por todos los procesadores a través de la red de interconexión. Cualquier procesador puede tener acceso a cualquier módulo de memoria o unidad de entrada y salida; sin embargo, no es posible que dos procesadores tengan acceso a un mismo módulo de

memoria exactamente al mismo tiempo. La comunicación entre los procesadores se realiza a través de la memoria compartida.

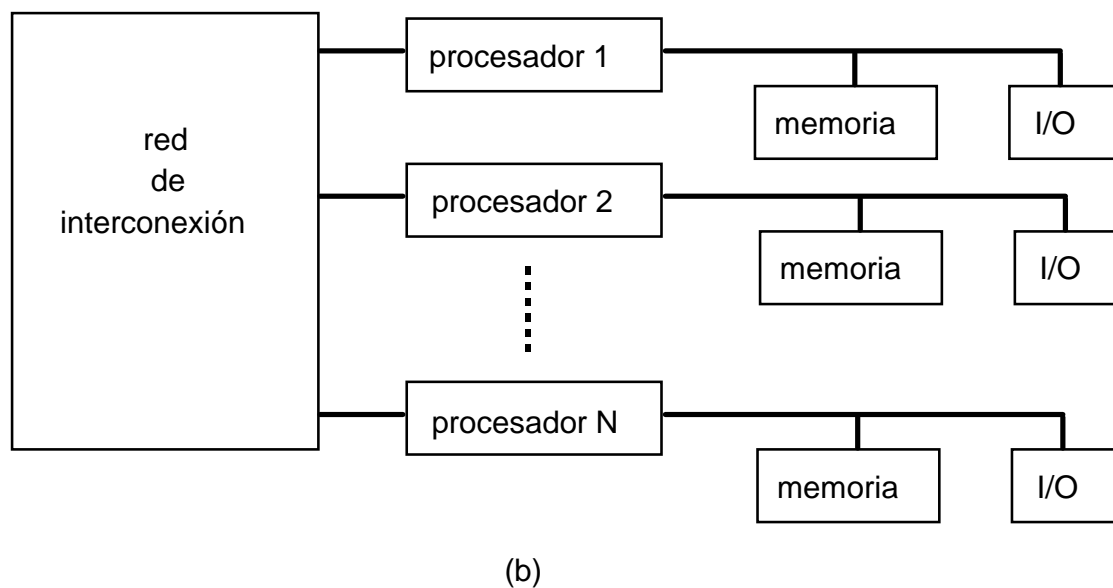
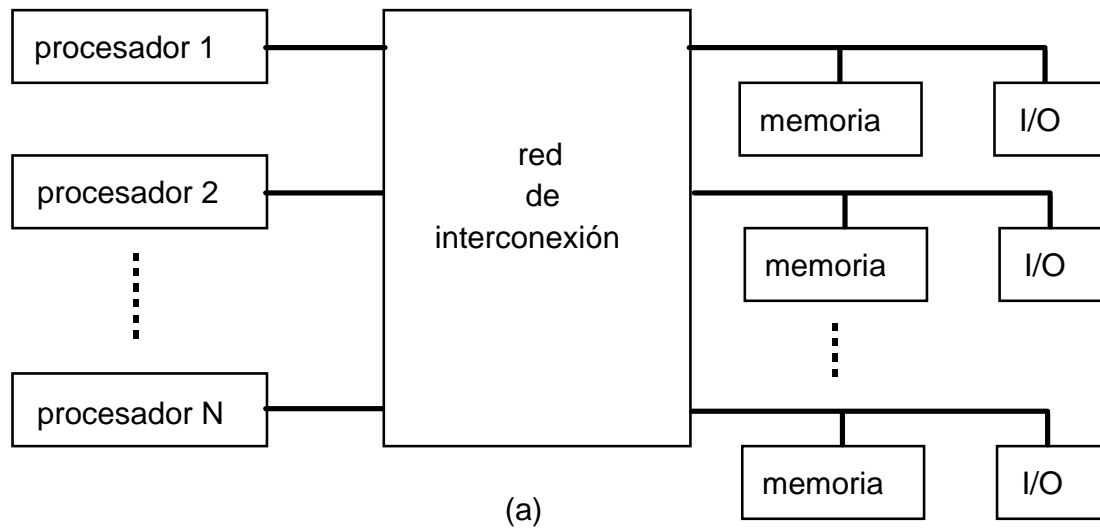


Figura 24.4. Arquitecturas de multiprocesadores.
(a) con memoria compartida y (b) con memoria local

En la arquitectura con memoria local, cada procesador tiene su propia memoria y su propia unidad de entrada y salida, solamente él puede hacer uso de ellas. La comunicación entre los procesadores se produce a través de la red de interconexión entre los mismos.

Para poder resolver un problema dado en un multiprocesador es necesario diseñar un algoritmo paralelo para su solución. Este es uno de los problemas de mayor complejidad al utilizar un multiprocesador. No todos los problemas permiten la creación de un algoritmo paralelo eficiente.

Otros de los problemas que se presentan en los multiprocesadores son:

- Los tiempos involucrados en la comunicación entre los procesadores.
- El trabajo que tiene que realizarse para la sincronización de los procesadores.
- El tiempo perdido cuando algún procesador se quede inactivo porque no tiene trabajo que hacer.
- El tiempo invertido en el control del sistema y la asignación de trabajos a cada procesador.

Estos problemas no se tienen en una arquitectura vectorial, pero como se mencionó anteriormente, el tipo de problemas que se pueden resolver en ellas es restringido.

24.7. Resumen.

En este capítulo se presentó un breve panorama de los conceptos utilizados en arquitecturas computacionales avanzadas para que el lector no se quede con la idea de que la arquitectura computacional básica que se trató a lo largo de los primeros 23 capítulos del libro constituye todo lo que se ha desarrollado en el área.

El concepto de memoria de caché fue desarrollado para lograr una mayor velocidad de cómputo aliviando parcialmente el problema de la velocidad de acceso a memoria que es el principal cuello de botella de las computadoras digitales.

La velocidad de las componentes electrónicas se ha incrementado más allá de lo que se vislumbraba hace treinta años pero este logro tecnológico no es la única fuente del incremento de la velocidad de las computadoras. El concepto de pipeline permite lograr velocidades de cómputo mayores con la misma velocidad de los componentes electrónicos. En este capítulo se presentaron los principios básicos de pipeline y su utilización en las arquitecturas computacionales.

Un cierto tipo de aplicaciones numéricas para solución de problemas científicos en las áreas de ingeniería y ciencias presentan características adecuadas para ser resueltos en arquitecturas con procesadores vectoriales. En este capítulo se presentó un ejemplo de esta tipo de aplicaciones y los fundamentos de los procesadores vectoriales.

Por último, se presentó una breve introducción a los multiprocesadores, distinguiendo los dos tipos principales de arquitecturas existentes y se mencionaron algunos de los principales problemas que presentan estas arquitecturas así como los retos con los que se enfrentan los programadores al desarrollar aplicaciones en este tipo de máquinas.

La finalidad de incluir este capítulo al final del libro es solamente para dar una idea de los principios utilizados en las arquitecturas avanzadas. Los lectores interesados deberán referirse a libros que se dedican especialmente a tratar estos temas con lujo de detalle.

24.8. Problemas.

1. Supóngase que se desea construir una unidad para sumar números de punto flotante utilizando un pipeline. Determine las etapas de que constaría esta unidad.
2. Si se tiene una unidad para multiplicación de números de punto flotante de 4 etapas similar a la descrita en la sección 24.3 y cada etapa opera en T segundos, conteste lo siguiente:
 - (a) calcule el tiempo necesario para realizar la multiplicación de un par de números de punto flotante.
 - (b) calcule el tiempo necesario para realizar la multiplicación de 10 pares de números de punto flotante.
 - (c) calcule el tiempo necesario para realizar la multiplicación de 50 pares de números de punto flotante.
 - (d) calcule el tiempo necesario para realizar la multiplicación de 100 pares de números de punto flotante.
3. Definiendo la eficiencia de una unidad en pipeline como la suma de las etapas utilizadas para realizar un trabajo entre la suma de las etapas utilizadas más las ociosas durante el mismo período de tiempo, calcule la eficiencia lograda en la unidad del problema anterior para cada uno de los cuatro casos mencionados. ¿Puede concluir algo al respecto?
4. A continuación se lista una serie de máquinas o procesadores que existen o han existido en la historia de la computación. Clasifíquelas como máquinas convencionales, máquinas con arquitecturas de pipeline, arquitecturas RISC, procesadores vectoriales o multiprocesadores.
 - (a) IBM RS/6000
 - (b) Procesador Pentium II
 - (c) Craig
 - (d) Cosmic cube
 - (e) ILLIAC IV
 - (f) IBM ES/9000
 - (g) IBM SP/2
 - (h) CDC 6600
 - (i) Motorola 68000
 - (j) INTEL 8051

5. Son similares los principios que llevaron a la creación de la memoria virtual a los utilizados en la memoria de caché? ¿Por qué?