UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES

''A R A G O N''

DESARROLLO AUTOMATICO DE MICROCONTROLADORES

TESIS PROFESIONAL

Que para obtener el Título de:

INGENIERO EN COMPUTACION

P r e s e n t a :

MIGUEL ANGEL LERA LOPEZ

México, D. F. 1994

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

A mis padres, la esencia y origen de todo lo que soy, es gracias a ellos.

A mis hermanos, por ayudarme a cumplir mis objetivos.

A mis maestros, por la experiencia transmitida en sus enseñanzas.

A mis amigos, por compartir conmigo el sendero de nuestro destino.

Todos ustedes han contribuido a formar en mi

más que a un buen alumno

a un ser humano

con una visión completamente especial

del complejo mundo que habitamos

y

de lo maravilloso que puede llegar a ser.

AGRADECIMIENTOS

Al Ing. Donaciano Jiménez Vázquez, director de mi tesis por su inapreciable ayuda, y por la paciencia y confianza depositada en mi.

A cada uno de los revisores de esta tesis, por sus valiosos comentarios y sugerencias, tendientes a mejorar su contenido.

Ing. Juan Castaldi Pérez

Ing. Fortunato Cerecedo Hernández

Ing. David Moisés Terán Pérez

Ing. Antonio Nieto Torres

Al futuro ingeniero Juan Salazar Alayón por su ayuda en la implementación física de los diagramas lógicos incluidos en este trabajo.

INDICE

PRÓLOGO

A través de los siglos el hombre ha buscado insistente el conocimiento y lo ha clasificado bajo diversas ciencias. Estas han crecido enormemente, día a día se suman a ellas nuevas teorías y resultados de investigaciones que rechazan ó confirman lo existente y desarrollan lo que hace tan sólo unas décadas pareciera imposible. La información acumulada como consecuencia del pensamiento es tan extensa que no existe persona alguna capaz de leerla y menos aún de comprenderla. La informática tiene por objetivo sistematizar los procesos de recolección, depuración, clasificación y recuperación de datos, así como su procesamiento en la generación de información. Una de las herramientas de mayor impacto inventada por el hombre ha sido la computadora, su rapidez para ejecutar operaciones lógicas y aritméticas la ha situado como elemento indispensable en casi todas las áreas del quehacer humano en que se requiera el procesamiento de datos, obtención de información, control de procesos, etc.

Unos cuantos años atrás se mencionaba que la computación marcaría la Era del Futuro, hoy la computación es una realidad, su uso se ha extendido formidablemente, la persona común se ha visto en la necesidad de aprender una terminología diferente, y adquirir nuevos conocimientos. Al mismo tiempo mucha gente se ha sentido desplazada por la computadora, siente que ésta le arrebata su trabajo, es decir, su modo de subsistencia. Esta actitud deberá cambiar, la computadora deberá verse como una valiosa herramienta que aprendiendo a usarla nos permitirá simplificar procesos mediante su sistematización, liberándonos del trabajo tedioso, para guiar nuestros pensamientos hacia estructuras del conocimiento más elevadas.

Capítulo Uno

Antecedentes

1.1. Introducción

1.2. Objetivos Generales

1.3. Objetivos Particulares

1.1. INTRODUCCIÓN

Prácticamente no existe hogar u oficina que carezca de algún dispositivo electrónico, desde un simple reloj de pulsera hasta un sofisticado equipo de cómputo, todos ellos funcionan gracias a la magia de la electrónica, el sólo mencionar artículos electrónicos nos llevaría varias páginas, lo cual nos demuestra que un artículo electrónico ha pasado de ser un objeto de lujo a una necesidad, el mundo sería completamente diferente si de repente careciéramos de equipos tan comunes como los radiorreceptores o los televisores.

La electrónica ha evolucionado y consecuentemente los equipos que la utilizan, es una tendencia actual sustituir elementos analógicos por elementos digitales, el uso de circuitos integrados se ha generalizado permitiendo que los equipos electrónicos reduzcan significativamente su costo y tamaño.

Los avances en Electrónica han constituido la base sustancial para el auge expansivo de las computadoras. La computadora digital se diferencia de cualquier otro aparato electrónico en que no constituye un equipo terminal. Todo equipo terminal tiene una función específica para la cual fue creado; por ejemplo un reloj digital nos sirve para medir el tiempo, un sintetizador para tocar una melodía, un radiorreceptor para recibir mensajes emitidos en puntos distantes al que estamos, etc. Ahora bien, sería usted capaz de nombrar la función asignada a una computadora ? Tal vez piense que una computadora tiene varias funciones ó que la función que se le confiere es demasiado amplia, ambos raciocinios son correctos, si nos inclinamos por lo primero listaríamos las actividades en que se hace uso de ella, presuponiendo que éstas fueron las funciones previstas en su diseño. La segunda respuesta nos conduciría a reconocer que la funcionalidad de la computadora ha sobrepasado las expectativas originales de su invención. Tomando ambas referencias, resulta que la computadora puede concebirse como un instrumento de propósito general que permite abordar problemas tan disímiles entre sí, que sólo poseen una característica común: todos ellos requieren una solución.

La propuesta de esta tesis es propiciar el desarrollo en cadena, es decir, aquel que se basa en generar elementos secundarios partiendo de los primariamente establecidos, hecho ésto los elementos resultantes contribuirán al desarrollo de sus antecesores, lo cual en síntesis enriquece al conocimiento. En

forma análoga si a partir de la electrónica ha evolucionado la computación, lo inverso puede ser factible. Gracias a las características no terminales de la computadora podemos crear soluciones a problemas concernientes a la electrónica digital. Este trabajo está destinado al estudio y sistematización de MicroControladores, buscando contribuir por medio de la computación al desarrollo de esta área tan importante del conocimiento humano.

1.2. Objetivos Generales

- Mostrar como la computadora puede ser empleada en la sistematización de procesos de todo tipo, incrementando la eficiencia y facilitando el trabajo a los diseñadores de sistemas.

- Contribuir por medio de la computación al desarrollo de la Electrónica Digital, específicamente en el área de MicroControladores.

1.3. OBJETIVOS PARTICULARES

- Discutir diversas metodologías utilizadas en el diseño de MicroControladores, analizarlas y sistematizarlas.

- Realizar una aplicación software que permita al usuario obtener la implementación lógica de un MicroControlador con el método deseado y requiriendo únicamente el autómata correspondiente al diseño.

Capítulo Dos

Introducción a los Autómatas

2.1. Definición de Autómata y Sistema

2.2. Características de un Sistema

2.3. Fases en el Desarrollo de un Sistema

2.4. Características de un Autómata

2.5. Cartas ASM

2.6. Diagramas MDS

2.1. DEFINICIÓN DE AUTÓMATA Y SISTEMA

Antes de que el lector especule respecto al tema a tratar en este capítulo, he creído prudente especificar lo que debemos entender por autómata. Seguramente para muchos de nosotros el concepto autómata generará en nuestra mente imágenes relacionadas con procesos industriales realizados por dispositivos mecánicos que simulan acciones ejercidas comúnmente por el cuerpo humano. Aunque este es el término más generalizado y conocido, existe otra acepción que es la que utilizaremos. A lo largo del presente libro definiremos a un autómata de la forma siguiente:

Autómata: Tipo de esquema que utilizando una simbología fija permite establecer y posteriormente entender el comportamiento preciso de un sistema.

Ahora bien, en nuestra definición previa ha surgido un concepto nuevo: *Sistema*. Prácticamente no existe disciplina que no utilice esta palabra, además de poseer muy diversas definiciones, cada una de ellas intenta apegarse a su objeto de estudio. En general por sistema se entiende a un conjunto de elementos que interactúan individualmente entre si, y como un todo con el medio que lo rodea.

Como puede observarse el término sistema es aplicable a un extenso caudal de formas y conceptos, de esta manera es posible encontrar diversos tipos de sistemas. El tipo de sistema estará dado por la naturaleza de sus componentes, por su funcionamiento (forma en que se relacionan sus componentes), y por su función (relación entre sistema y entorno circundante).

2.2. CARACTERÍSTICAS DE UN SISTEMA

Para comprender a un sistema se deben estudiar sus características, existen algunas que le son intrínsecas por naturaleza y otras de aspecto particular. Nos preocuparemos en mencionar brevemente las primeras.

Por definición enunciamos que todo sistema se encuentra formado por un conjunto de elementos, éstos reciben la denominación *subsistema*, físicamente se distinguen porque son parte del sistema, operacionalmente porque realizan una función específica dentro del mismo. De manera general todo

sistema resulta ser un subsistema y viceversa, un ejemplo clásico y evidente es el ser humano, se encuentra compuesto de una serie de sistemas (nervioso, muscular, óseo, etc.). Cada uno de estos sistemas cumple con los requerimientos para serlo, pero a su vez son subsistemas respecto al ser humano. A su vez, el ser humano es un subsistema del sistema seres vivos. Además un sistema puede pertenecer a varios tipos de sistema simultáneamente. Por ejemplo un tren podría ser englobado dentro de los sistemas mecánicos (por la naturaleza de sus componentes o como elemento del sistema de transporte (por su función).

En concreto, basta con fijar nuestra vista o pensamiento en un objeto o situación y habremos encontrado un sistema, si deseamos saber cuáles son sus subsistemas, simplemente observemos de que está formado (física o estructuralmente), si deseamos saber respecto a qué es un subsistema, bastará observar de qué forma parte.

2.3. FASES EN EL DESARROLLO DE SISTEMAS

Así como existen múltiples definiciones para el concepto sistema, también existe una gran cantidad de metodologías para desarrollo de sistemas, cualquiera de ellas comúnmente tiende a una de las posturas siguientes o a una mezcla de las mismas.

a) Visualiza al sistema como un todo, en esta forma el diseñador observa a su problema tal como fue planteado.

b) Visualiza al sistema en subsistemas aislados, el diseñador divide al sistema en subsistemas y busca soluciones individuales a cada uno de ellos, pensando así en solucionar al sistema total.

c) Visualiza al sistema como parte integrante de un todo, aquí el objetivo es ubicar al sistema por desarrollar dentro del ambiente en que se desea actúe.

La teoría moderna denominada enfoque de sistemas y la experiencia indican que el acoplamiento de todas las posturas anteriores desemboca en un desarrollo óptimo, el cual no carecerá de ningún punto relevante. El secreto

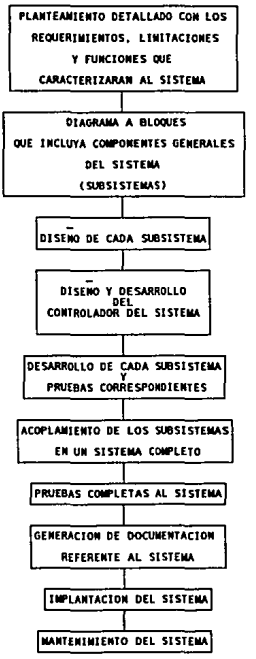


Fig. 2.1. FASES PARA EL DESARROLLO DE UN SISTEMA

de este enfoque es visualizar al sistema de la forma apropiada en cada una de las fases de su desarrollo. Por ejemplo, en su inicio lo mejor es partir de acuerdo al inciso *a*, con movimiento al inciso *c*, esto de manera cíclica hasta lograr la conjunción y comprensión en ambos marcos de referencia. Una vez que el sistema se encuentra completamente claro, se busca apoyo en el inciso *b* con cierta variación, en lugar de observar aislados a los subsistemas, se busca la forma en que se comunican recurriendo al inciso *a*. El inciso *b* en conjunción con el *c* resulta útil al planear la solución para cada subsistema, su implementación se basará exclusivamente en el inciso *b*, y el acoplamiento de todos los subsistemas en uno solo requerirá del uso de todos los incisos.

Tal vez el enfoque anterior suene un poco complicado, más aún cuando se nos ha enseñado a ser metódicos siguiendo una secuencia de pasos en busca de soluciones. Si este fuera su caso le aconsejo considerar el esquema de la figura 2.1. donde he incluido las fases más comunes que usted pudiera encontrar en una metodología cualquiera para el desarrollo de sistemas. Aunque cada fase se muestra en forma secuencial su seguimiento rara vez será en la forma indicada, al ubicarse en una fase, pudiera requerir retroceder a un punto previo o incluso saltar a uno más avanzado en busca de respuestas a sus expectativas actuales. A continuación presento algunas sugerencias que le serán de utilidad en cada fase de su desarrollo.

Planteamiento. En esta fase el diseñador debe familiarizarse con el problema, entender perfectamente las limitaciones y requerimientos del sistema. Debe preguntarse cuál y qué es su sistema ?, qué debe hacer ?, cómo lo debe hacer ?, dónde funcionará ? y cuándo se requerirá su utilización ?.

Diagrama a Bloques. Indique de forma gráfica a los subsistemas que visualiza como integrantes de su sistema, no hay reglas para comenzar, simplemente inicie el proceso, esquematice todo aquello que surja en su mente. El diagrama planteado originalmente difícilmente será el decisivo, continuamente requerirá de cambios en su estructura hasta que ésta se apegue completamente a lo que deseamos. El diagrama debe incluir también la relación entre sistema y su entorno operativo.

Diseño. En esta fase detalle cuidadosamente cada bloque dibujado en el diagrama previo. Si un bloque le parece complejo tal vez deba subdividirlo en pequeños bloques.

Desarrollo. Cuando cada bloque ha sido comprendido en su funcionamiento proceda a su desarrollo. Ubique su atención en cada subsistema sólo hasta que esté consciente de su relación dentro del sistema.

Pruebas parciales. Cada subsistema desarrollado debe ser sometido a prueba antes de integrarlo al sistema, esto le evitará efectuar rastreo innecesario en busca de errores.

Integración. Si está seguro de la funcionalidad de cada bloque por separado, es tiempo de integrarlos. Le sugiero -siempre que sea posible- realizar integración bloque a bloque y pruebas parciales a los módulos resultantes.

Pruebas finales. Al completar su sistema, sométalo a todo tipo de pruebas, considere principalmente las condiciones a que estará sujeto cotidianamente y no pase por alto aquellas que *difícilmente* pudieran presentarse.

Documentación. El que funcione el sistema no significa que lo haya concluido, siempre debe documentarlo, por su bien y por el de los demás. Recuerde realizar un manual dirigido al usuario final del sistema, en el cual le indique su objetivo, función, operación en detalle y como solucionar cualquier problema que pudiera ocurrir. La documentación no sólo es para el usuario, debe existir siempre un manual técnico dirigido a usted mismo o a futuras personas que tengan la necesidad de analizar en detalle el sistema realizado, éste segundo manual deberá incluir toda información concerniente al sistema. Siempre tenga en mente que un sistema bien documentado será más fácilmente analizarlo, operarlo, corregirlo en caso necesario e incluso mejorarlo.

Implantación. El siguiente paso es aquél que el diseñador ansía desde el momento en que emprende la realización de un sistema. Es aquí donde el sistema es ubicado en su espacio de acción y se le realizan sus últimas pruebas.

Mantenimiento. Ningún sistema realizado es perfecto, una vez que se ha implantado y se labora con él, se descubren carencias en ciertos aspectos ó bien se plantean nuevas sugerencias, ambos puntos requieren modificaciones

después de la implementación, a esta etapa se le conoce como mantenimiento. Prácticamente el mantenimiento del sistema es una etapa que se extiende durante toda su vida útil, ya que continuamente surgen nuevos elementos a incluir en él, cuando los requerimientos sobrepasan considerablemente a lo existente se ha llegado ya al deceso del sistema, es tiempo de cambiar y por tanto, el desarrollo de un nuevo sistema deberá iniciarse.

2.4. CARACTERÍSTICAS DE UN AUTÓMATA

Un autómata tal cual lo definimos debe poseer algunas características.

Simbología. Para cada elemento debe poseer un símbolo distinto que evite ambigüedades y al mismo tiempo sea flexible, permitiendo mediante su combinación la implementación de diversos tipos de operaciones de control.

Precisión. El autómata debe ser detallado en cuanto a las operaciones a realizar, se especificará claramente el comportamiento que asumirá el sistema bajo cualquier condición.

Nemónicos. Además de una simbología el autómata requiere el uso de nemónicos, los cuales son simplemente abreviaturas que representan señales, situaciones o instrucciones a fin de compactar el esquema realizado.

Documentación. Si bien el uso de nemónicos evita esquemas demasiado voluminosos, puede provocar ilegibilidad en los mismos, para aquellas personas que no hayan intervenido en el diseño. Por tal motivo es necesario documentar al autómata, explicando por escrito el significado de cada nemónico empleado.

Independencia. Al especificar un autómata, éste será independiente del hardware a utilizar. Es decir, el diseñador no debe preocuparse de la circuiteria a utilizar hasta que el autómata se ha concluido. Si la metodología empleada posteriormente impone algún tipo de restricción, existirán procedimientos para ajustarse a ellas.

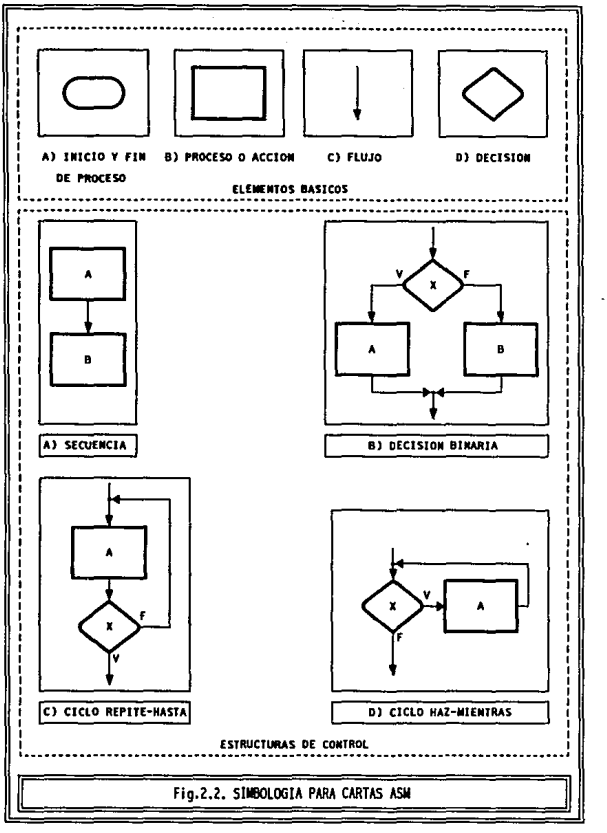


Fig.2.2. SIMBOLOGIA PARA CARTAS ASM

2.5. CARTAS ASM

La simbología correspondiente a este tipo de autómata se muestra en la fig.2.2. Existen cuatro elementos básicos que a continuación describiremos:

Inicio/Fin de Proceso: Especie de elipse horizontal. Tiene como función marcar el inicio del proceso del controlador así como el final del mismo, en el primer caso se coloca la palabra INICIO en el interior de la elipse, en el segundo caso la palabra a colocar será FIN. Existen muchos casos en que el inicio y final de un proceso corresponden a un mismo punto, es decir las condiciones al finalizar el proceso se restablecen a como estaban en su inicio, lo cual permitirá reanudar nuevamente el proceso. Si este fuera el caso, el bloque de Inicio/Fin aparecerá una sola vez y se marcará como I/F.

Proceso o Acción: Indica una acción o proceso a realizarse, la cual deberá ser escrita en el interior del rectángulo.

Flujo: Este símbolo siempre se encontrará uniendo a dos elementos. Tiene como función indicar el siguiente símbolo a procesar (al que apunta la flecha) a partir del símbolo actual (origen de la flecha).

Decisión: Cuando en la secuencia de operaciones del controlador se requiere tomar una decisión respecto al flujo del proceso, este será direccionado en base a una pregunta respecto al estado actual del sistema. La pregunta se coloca en el interior del rombo.

Combinando los cuatro elementos básicos es posible describir el flujo de operaciones que serán realizadas por un sistema, existe una serie de combinaciones denominadas *Estructuras de Control.*

Secuencia: Esta estructura es la más simple de todas, pues no incluye decisión alguna, simplemente se realiza una acción al término de otra. En la figura el controlador ejecuta la acción A y al finalizar procede con la B.

Decisión Binaria: Esta estructura es ampliamente utilizada, el proceso al encontrar el símbolo de decisión realiza una pregunta. Si la respuesta es afirmativa el flujo se dirige por la rama indicada con una V (verdadera), en caso contrario (respuesta negativa) el flujo se irá por la rama marcada con F (falso) y ejecutará el proceso B.

Ciclo Repite-Hasta: La estructura realiza la acción A, a continuación efectúa una pregunta, cuya respuesta en todo buen diseño debe depender de la acción A, para evitar que el controlador permanezca indefinidamente en este punto. La ejecución de A se realizará cíclicamente mientras la respuesta de la pregunta sea negativa.

Ciclo Haz-Mientras: Es una variación de la estructura anterior, aquí la decisión se realiza previa a la acción, la cual se ejecutará sólo y mientras la respuesta sea afirmativa. Cuando ya no lo sea, el control pasa al siguiente símbolo.

2.6. DIAGRAMAS MDS

Otro tipo de autómata es el llamado diagrama MDS, el cual permite describir el funcionamiento de un controlador en forma bastante clara. la simbología utilizada en su realización se encuentra especificada en la fig. 2.3. segmentada en elementos básicos y estructuras de control.

Elementos Básicos.

Estado: Fig. A. Define una situación particular del autómata. En su interior se coloca un nemónico que identifique al estado y al mismo tiempo lo distinga de los demás.

Flujo: Figs. B y C. Siempre se encuentra uniendo a dos estados. El estado señalado por la flecha se denomina *Estado Siguiente*, el estado origen será referenciado como *Estado Presente*. Tiene como función indicar el comportamiento del sistema a través de la transición de estados. El flujo podrá ser incondicional o condicional, en el primer caso la flecha aparecerá sola, lo cual indica que la transición del estado presente al estado siguiente ocurrirá automáticamente en el siguiente instante, sin depender de las condiciones externas al sistema. Si el flujo es condicional se deberá marcar por un nemónico, correspondiente a una señal que es recibida por el controlador. Así, la transición al estado siguiente sólo ocurrirá si en el estado actual el controlador recibe la señal adecuada.

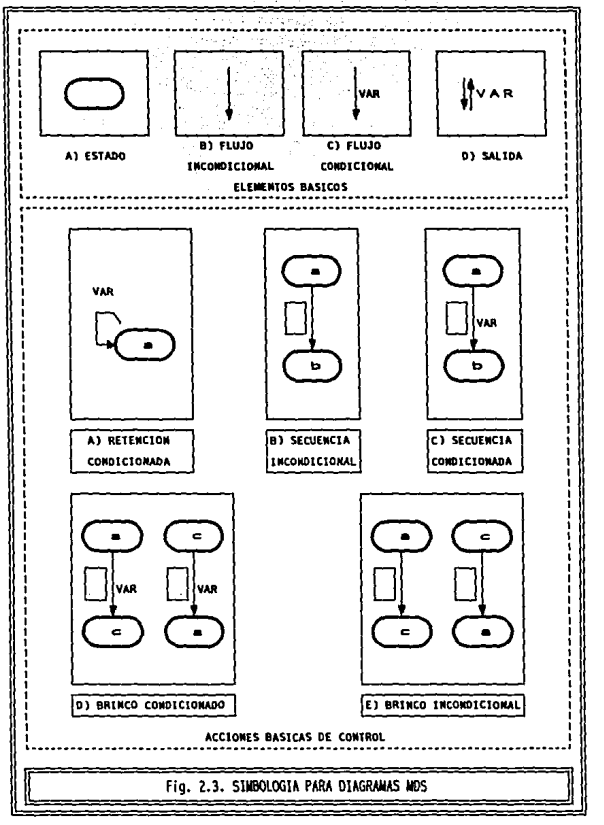


Fig. 2.3. SIMBOLOGIA PARA DIAGRAMAS MDS

Salida: Fig, D. Se indica por un par de flechas verticales con sentido contrario, con un nemónico asociado, el cual identifica una señal de salida generada por el controlador.

El símbolo se coloca junto al estado en que debe generarse la salida, así mismo un estado puede poseer varias salidas. La ausencia del símbolo en un estado indica la no generación de salidas.

Con esta breve simbología es posible la creación de diversas estructuras. Todas ellas pertenecerán o se formarán mediante la combinación de tres acciones básicas de control ilustradas en la fig. 2.3.

Un controlador siempre efectuará una de dos operaciones, puede permanecer en un estado o bien cambiar a otro, según convenga. La permanencia o *retención* en un estado siempre deberá ser condicionada, es decir dependiente de una señal externa, de lo contrario el controlador permanecerá indefinidamente en un estado, lo cual obviamente no resultaría útil. Cuando el controlador cambia de estado, se dice que ocurre una transición, esta a diferencia de la retención puede o no depender de una señal externa.

La transición se clasifica como secuencia o brinco, las figuras B y C muestran la estructura de una secuencia, incondicional y condicional respectivamente, el estado b debe ser continuo respecto al estado a, es decir deben ser colaterales en un marco de referencia específico. Más adelante detallaremos en profundidad el concepto de continuidad.

Si existen estados intermedios entre el estado presente y el siguiente no habiendo secuencia entre ambos, la transición se realiza por medio de un brinco, tal cual se ilustra en las figuras D y E.

Observe que al lado izquierdo del símbolo correspondiente al flujo en cada operación de transición se ha dibujado un recuadro vacío, el cual está destinado a contener un nemónico que indique el tipo de acción requerido. Su especificación no se ha realizado en razón a que cada metodología tiene la opción para definir sus propios nemónicos de acuerdo al conjunto de instrucciones que posee. Sin embargo, cualquiera que sea la instrucción, será posible su implementación a partir de las acciones básicas de control ya mencionadas.

Todas las estructuras de control presentadas para las cartas ASM pueden ser esquematizadas mediante símbolos MDS (ver fig. 2.4.), debido a esta característica cuando se desea plantear la solución a un problema existe la opción de trazar una carta ASM y posteriormente trazar el diagrama MDS correspondiente, o bien cuando se tiene práctica suficiente dibujar directamente el diagrama MDS.

En general todo autómata por medio de su simbología puede representar el funcionamiento de un controlador digital, lo único que cambia es la forma de su representación. En los capítulos siguientes emplearemos extensivamente autómatas de tipo MDS en virtud de que pueden ser implementados en forma compacta y entendible. Al diseñar un diagrama MDS considere los puntos siguientes:

- No se permite que dos estados sean designados bajo un mismo nemónico.
- No se permite retención incondicional.
- Un estado presente puede tener más de un estado siguiente. Para cada uno de ellos se utilizará una ruta y un símbolo de flujo, así como un nemónico de acción.
- Un estado presente no puede acceder a más de un estado siguiente en forma incondicional.
- Un estado puede generar tantas salidas como se requieran.

A partir de la especificación del autómata, el siguiente paso a realizar es la elección y posterior aplicación de una metodología de diseño. El capítulo siguiente abarca una serie de métodos, todos ellos parten inicialmente del autómata en su forma MDS. Hasta ahora sólo hemos explicado los conceptos generales a todos los diagramas MDS, sólo resta mencionar que cada metodología realiza sobre el diagrama un proceso adicional llamado *Asignación de Nemónicos de Acción*, los códigos utilizados dependerán de la metodología y los mencionaremos en su momento. El proceso consiste en definir acciones a partir de los dos tipos de transición que hemos mencionado.

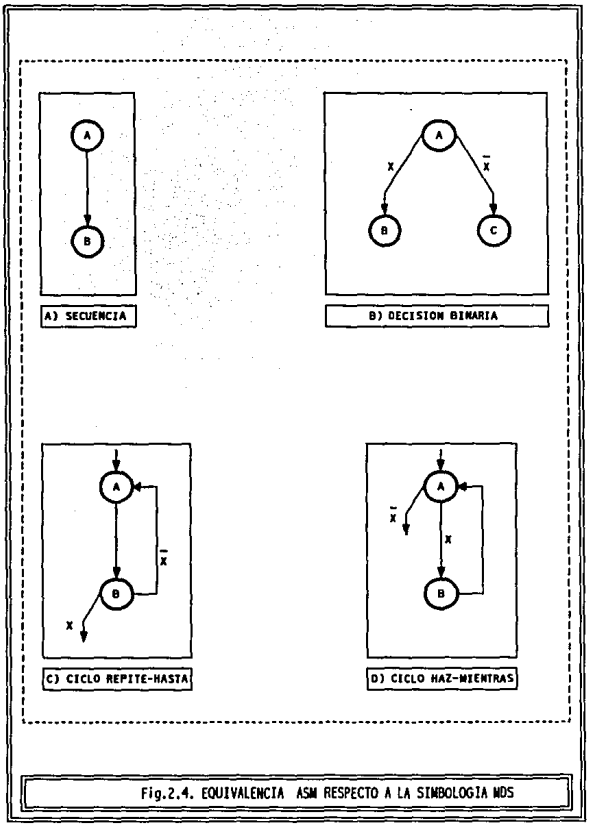


Fig.2.4. EQUIVALENCIA ASM RESPECTO A LA SIMBOLOGIA NDS

Capítulo Tres

Sistemas Automáticos

3.1. Clasificación de Sistemas

3.2. Características de un Sistema Automático

3.3. Definición y Características de un Controlador

3.4 Tipos de Controladores

3.5 Planteamiento de un Problema

y Obtención de su Autómata

3.1. CLASIFICACIÓN DE SISTEMAS

Con el paso del tiempo el hombre ha desarrollado y utilizado diversos tipos de sistemas, el surgimiento de ellos ha sido en respuesta a los problemas y necesidades que han aquejado al ser humano a través de su existencia. Cuando esto ha ocurrido, se ha tenido que modificar a los sistemas existentes, o crear sistemas nuevos. Generalmente la mayoría de los sistemas que dan solución completa a una necesidad no utilizan un solo tipo de componentes, y si una mezcla de ellos.

Las características presentadas por un sistema hacen posible su clasificación. Tal cual se menciono, un sistema al ser clasificado puede quedar incluido en más de un conjunto de sistemas. Para nuestro uso sólo nos interesa ubicar a un sistema bajo dos aspectos, primero de acuerdo al tipo de componentes que lo conforman y segundo por el grado de control que presenta. El inciso A) de la figura 3.1. muestra solo a tres tipos de componentes, son los más comunes más no los únicos. Nuestro campo de acción se encuentra enmarcado por un recuadro punteado y lo definimos a continuación.

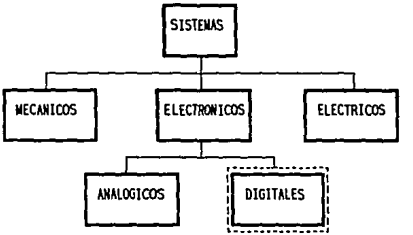
Sistema Electrónico Digital. Conjunto de dispositivos electrónicos que en forma individual realizan funciones digitales y que interconectados procesan información codificada en forma discreta.

Los primeros sistemas desarrollados fueron manuales, su principal característica es la intervención del ser humano para llevar a cabo un proceso, él es quien controla la forma de su operación. Un sistema semi automático distribuye su control externa e internamente, el primero corresponde al hombre, el segundo al mismo sistema. Un sistema automático es aquél en que no interviene la mano del hombre para desarrollar el proceso al cual fue destinado.

3.2. CARACTERÍSTICAS DE UN SISTEMA AUTOMÁTICO

a) La realización del proceso ejecutado por el sistema no requiere en absoluto la intervención del ser humano.

b) Su comportamiento está basado en un ciclo retroalimentado, es decir, la forma de operación actual determina su operación siguiente.



A) CLASIFICACION POR LA NATURALEZA DE SUS COMPONENTES

B) CLASIFICACION POR SU GRADO DE CONTROL

Fig. 3.1. CLASIFICACION DE SISTEMAS

c) Posee al menos un subsistema que ejerce control sobre los demás denominado *Controlador del Sistema.*

3.3 Definición y Características de un Controlador

Controlador. Conjunto de dispositivos integrantes de un subsistema que dirige el comportamiento general del sistema al cual pertenece.

El controlador posee características que lo diferencian de los demás subsistemas:

a) Un subsistema realiza una o varias acciones como respuesta a uno o varios estímulos ejercidos sobre él. Para una misma serie de estímulos siempre responderá de la misma forma. Un controlador ante un mismo estímulo puede generar diferentes respuestas, las cuales estarán en función de su comportamiento pasado.

b) Un subsistema por definición interactúa con los demás subsistemas, directa o indirectamente. En la forma directa no existe un control explícito en la acción ejercida por un subsistema sobre otro. En la forma indirecta, el control es quien decide como afecta la acción de un subsistema a otro.

Supongamos que el subsistema A es un interruptor de encendido y el subsistema B corresponde a un subsistema de riego para un jardín. En la acción directa el jardín será regado cada vez que el interruptor sea accionado. En la acción indirecta se podría intercalar un controlador que decida en base a la humedad de la tierra si es conveniente regar el jardín, lo cual evitará encharcamientos y deterioro del mismo. En este sencillo ejemplo, el subsistema A no ejerce la acción simplemente informa al controlador la situación o estado en que se encuentra y éste es quien decide la acción a ejercer.

c) El controlador debe saber como está operando cada subsistema, por medio de la señales recibidas, y así determinar como operará a continuación, por medio de la señales emitidas.

Para que un sistema opere adecuadamente requiere que su subsistema de control cumpla con varios requisitos:

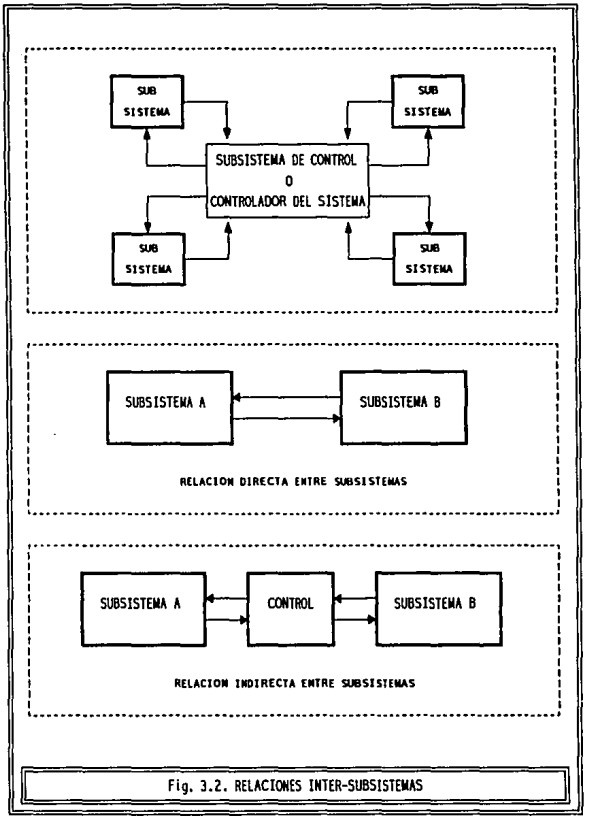


Fig. 3.2. RELACIONES INTER-SUBSISTEMAS

Confiabilidad: el controlador debe ser capaz de decidir en todo momento la acción que tomará cada uno de los subsistemas para que en conjunto operen adecuadamente.

Rapidez: debe tomar decisiones rápidas para que el sistema opere eficientemente.

Estabilidad: el sistema debe operar correctamente bajo una diversidad de ambientes diferentes o específicamente al que haya sido destinado.

Precisión: sus acciones deben estar en perfecta sincronía a fin de evitar ejecuciones a destiempo.

Los sistemas digitales son los que mejor cubren estos requisitos, motivo por el cual son los más utilizados. En la actualidad aunque los sistemas utilizan todo tipo de componentes existe la tendencia a diseñar el controlador mediante elementos electrónicos digitales.

De este punto en adelante concentraremos nuestra atención exclusivamente en los controladores digitales. Obviamente como su nombre lo indica, sus componentes y modo de operación tiene sus bases en la teoría de electrónica digital.

El desarrollo de un sistema dependiendo de su complejidad es un proceso bastante extenso, cada una de sus fases exige la dedicación y esfuerzo de una buena cantidad de personal y tiempo. Indudablemente todas las fases son importantes, sin embargo, y sin menospreciar a ninguna, el diseño del controlador del sistema es una parte vital, por ser aquí donde se decide la forma precisa en que operará todo el sistema, si este diseño se realiza inadecuadamente el sistema será inutilizable, podríamos decir que el controlador es al sistema como el cerebro al cuerpo humano. La fase de diseño y desarrollo del controlador exige también una serie de procedimientos. La estructura presentada en la fig.3.3. es general para todos los casos. Uno de los procedimientos se refiere a la elección de la metodología deseada para el diseño del controlador, esto es porque existe una amplia variedad de controladores, caracterizados por los elementos que los componen.

Además un subsistema de control puede clasificarse como microcontrolador, controlador o macrocontrolador, tomando en cuenta el

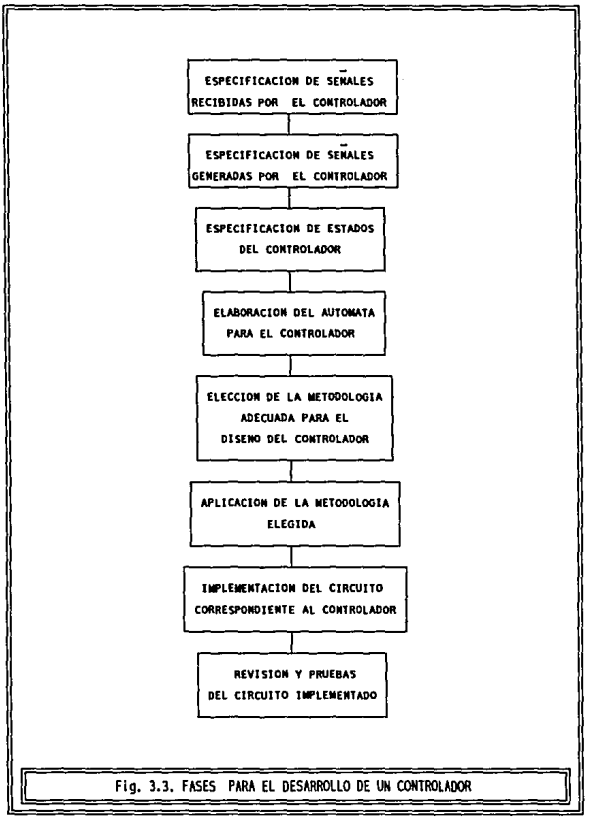


Fig. 3.3. FASES PARA EL DESARROLLO DE UN CONTROLADOR

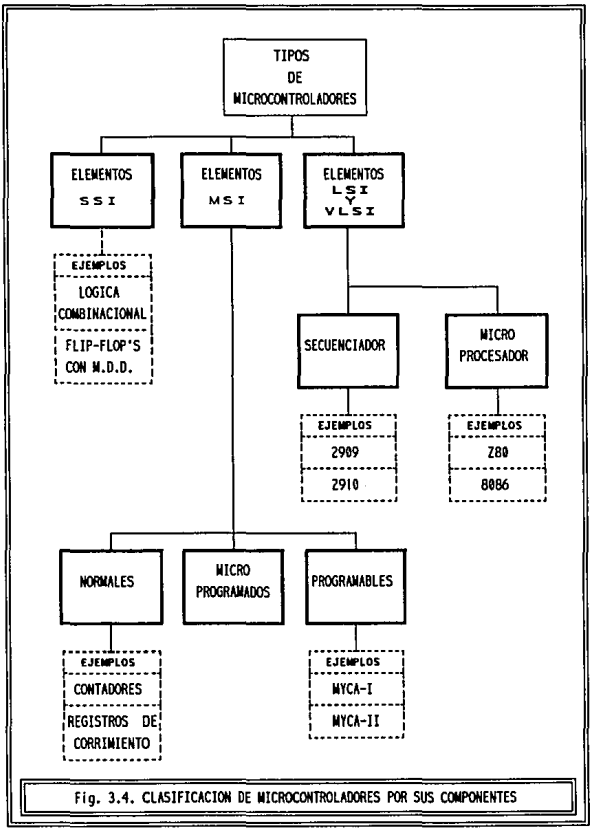


Fig. 3.4. CLASIFICACION DE MICROCONTROLADORES POR SUS COMPONENTES

número de estados que puede adoptar. Aunque en su aspecto básico la esencia de los controladores es la misma, enfocaremos nuestro estudio a microcontroladores digitales, entendiendo como tal a un conjunto de elementos electrónicos digitales interconectados, que tienen por función dirigir el comportamiento de un sistema en todo momento, mediante la adopción de una serie de estados. Esta serie de estados será perfectamente definida y limitada atendiendo a su carácter de microcontrolador.

3.4. Tipos de Microcontroladores

A un microcontrolador lo podemos clasificar de acuerdo al tipo de integración de sus componentes en SSI (Integración en Pequeña Escala), MSI (Integración a Mediana Escala) ó LSI (Integración a gran Escala). El esquema de la figura 3.4. nos proporciona por medio de un organigrama los tipos de microcontroladores que podemos encontrar. Los recuadros punteados indican algunos ejemplos para cada tipo.

Por ahora describiremos a grandes rasgos las características de cada arquitectura de acuerdo a sus elementos, para posteriormente en el capítulo cuatro, ejemplificarlas mediante metodologías representativas.

3.4.1. Microcontroladores SSI

Para la implementación de estos controladores se puede emplear única y exclusivamente elementos SSI o bien una combinación de SSI con MSI. Su clasificación bajo esta categoría se fundamenta en que su elemento de memoria está constituido por flip-flops, los cuales son capaces de almacenar una condición o estado y tumar a otro de manera síncrona a partir del actual.

En la fig.3.5. los decodificadores de entrada y de estados deben ser implementados mediante el uso exclusivo de compuertas, este será un controlador totalmente SSI. Una implementación alterna la constituye la fig 3.6. donde estos bloques son reemplazados por un conjunto de multiplexores a la entrada y un decodificador MSI a la salida. Aquí se requiere un bloque adicional, denominado Lógica Reducida que incluye el uso de compuertas para dirigir a los

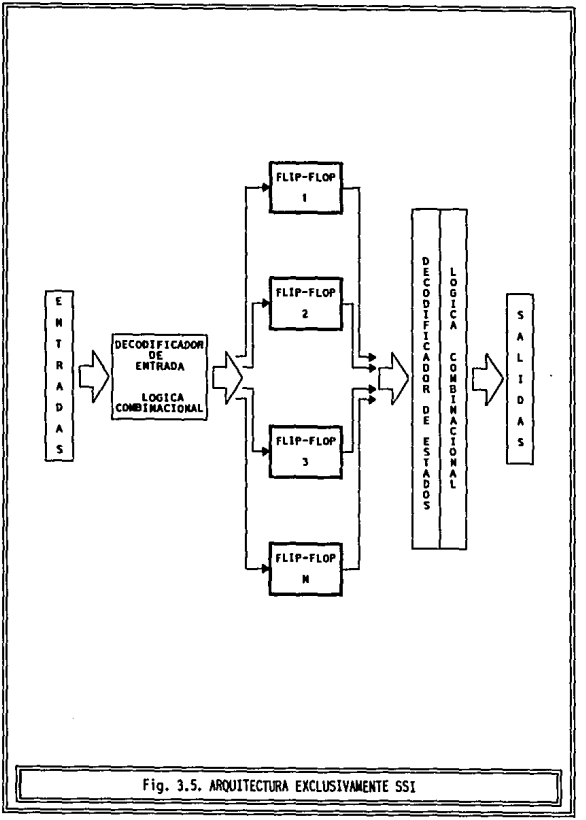


Fig. 3.5. ARQUITECTURA EXCLUSIVAMENTE SSI

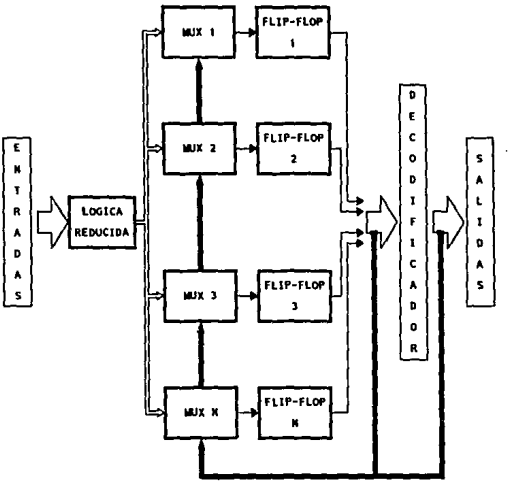


Fig. 3.6. ARQUITECTURA SSI COMPLEMENTADA CON MSI

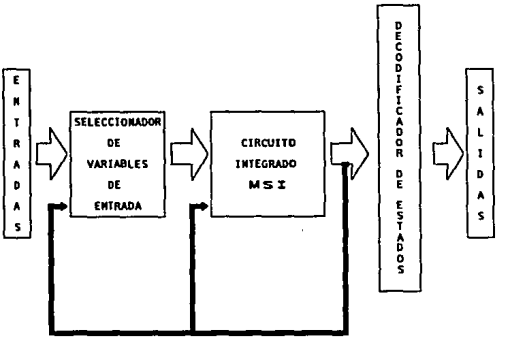


Fig. 3.7. ARQUITECTURA PARA MICROCONTROLADORES MSI

multiplexores los términos apropiados. La funcionalidad de ambos tipos de microcontroladores es similar, la diferencia radica en su implementación.

3.4.2. Microcontroladores MSI Normales

Su elemento principal es una pastilla MSI (un contador, un registro de corrimiento, etc.). El elemento utilizado le proporciona al controlador una serie de características simulables de instrucciones que controlan el flujo entre un estado y otro. En torno al elemento principal se colocan circuitos periféricos que sirven de interfaz para la recepción y generación de señales.

La fig. 3.7. muestra la arquitectura general para estos controladores, cada circuito MSI da lugar a una arquitectura particular, su discusión se realizará en el capítulo siguiente.

3.4.3. Microcontroladores Programados

Un controlador microprogramado es aquel en que el diseñador puede programar las operaciones de un dispositivo mediante el llenado de una tabla de memoria o mediante un lenguaje de máquina a partir de un diagrama MDS.

a) Componentes

- Una memoria RAM o ROM que permita el almacenamiento de instrucciones de operación básica y códigos de salida.
- Un contador de programa o un registro direccionador de memoria utilizado para seleccionar o direccionar las instrucciones almacenadas.

b) Características

- Habilidad para iniciar el proceso mediante la ejecución de la instrucción almacenada en una localidad arbitraria de memoria.
- Proceso secuencial de manera condicional o incondicional de las instrucciones almacenadas en localidades contiguas de memoria.

- Capacidad de procesamiento de la instrucción siguiente o del direccionamiento a otra localidad para la ejecución de la instrucción almacenada en ella.

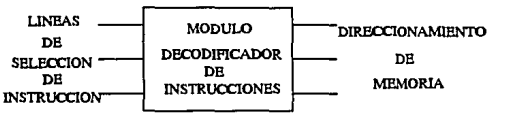
c) OPERACIÓN

La operación de estos controladores se basa en la ejecución de instrucciones almacenadas en una memoria. Para el adecuado funcionamiento del controlador se debe especificar la estructura con que serán almacenados los datos de la memoria, a este esquema se le denomina *Formato de Control*. Existen diversos tipos de formatos válidos, su estructura dependerá de sus elementos. De acuerdo a lo anterior, el esquema correspondiente a la arquitectura general para un controlador microprogramado se muestra en la fig. 3.8.

3.4.4. CONTROLADORES PROGRAMABLES CON CONJUNTO FIJO DE INSTRUCCIONES

Existen algunas diferencias entre este tipo de controlador programable con el mencionado previamente:

1) Antes de la realización del controlador se debe crear un módulo decodificador de instrucciones.



Los elementos que contenga el módulo así como sus conexiones dependerán de las instrucciones que quieran ser implementadas, las cuales pueden ser tan simples o complejas como uno quiera.

Una vez hecho este modulo existe una metodología general aplicable a su implantación dentro de un controlador.

2) Cada instrucción del controlador tiene asociado un código fijo, el cual se conoce comúnmente como código de operación u OPCODE.

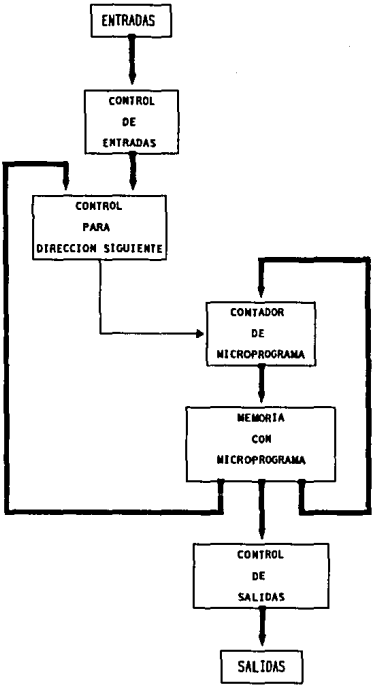


Fig. 3.8. ARQUITECTURA GENERAL PARA MSI MICROPROGRAMADO

Como ya se habrá dado cuenta el uso de estos controladores implica un doble trabajo:

a) Primeramente se debe diseñar el módulo decodificador de instrucciones.

b) y una vez hecho esto, diseñar el controlador deseado.

Este proceso es análogo a la creación de software para computadora. Anteriormente la creación de un programa se realizaba directamente en lenguaje de máquina o bien en lenguaje ensamblador, posteriormente se han desarrollado lenguajes de alto nivel que facilitan la escritura de programas poniendo a disposición del diseñador de software, un conjunto fijo de instrucciones que serán decodificadas por medio de un módulo especial llamado compilador.

Lo mismo ocurre en diseño de controladores de este tipo, el módulo decodificador deseado se diseña una sola vez y posteriormente solo se le utiliza, facilitando considerablemente el proceso de diseño.

Ahora bien, al igual que en el desarrollo de software existe una multitud de lenguajes de programación, cada uno de ellos con un conjunto fijo de instrucciones, en el desarrollo de controladores existen también múltiples diseños ya elaborados que nos proporcionan características diferentes.

En el siguiente capítulo estudiaremos dos tipos de decodificadores de instrucciones conocidos como MYCAI y MYCAII, se verá su configuración interna y su uso en el desarrollo de controladores.

3.4.5. MICROCONTROLADORES LSI

Su elemento principal es un circuito LSI, el cual proporciona a su arquitectura todos los elementos necesarios para implementación de una gran variedad de instrucciones. Su uso debiera ser exclusivo para aquellos casos cuya naturaleza no puede ser desarrollada o no es adecuada para arquitecturas SSI o MSI, en este trabajo solamente abordaremos un ejemplo con esta configuración basado en el circuito integrado MC2909.

La razón para no adentrarnos con estos elementos, es la intención de presentar formas alternativas al uso de microprocesadores, ya que no toda

aplicación requiere su empleo, el cual elevaría de manera innecesaria los costos del diseño.

3.5. Planteamiento de un Problema y Obtención de su Autómata

En esta sección escribiremos el planteamiento y solución a un problema. La intención principal es mostrar la metodología general explicada previamente para la generación de sistemas. No se pretende de ningún modo examinar exhaustivamente el planteamiento del problema, sólo se desea tomar de él sus características básicas que permitan la implementación didáctica de un autómata. Se debe tomar en cuenta que la resolución de un problema mediante controladores puede ser tan completa y compleja como se requiera o se desee, lo cual se verá reflejado en el número de estados e interconexiones en nuestro autómata. El problema que se presenta tiene por objetivo servir como ejemplo de utilización de diversas metodologías en el desarrollo de controladores.

3.5.1. Problema

Un banco desea implementar una serie de cajeros automáticos que proporcionen servicio a sus clientes las 24 horas del día. Cada cajero deberá tener las siguientes características:

- Funcionará con la introducción de una tarjeta especialmente magnetizada que le será entregada a cada cliente.
- Poseerá un teclado para que el usuario elija la operación deseada e introduzca las cantidades requeridas por el cajero.
- Los requerimientos del cajero serán presentados mediante un monitor.
- Aceptará tres tipos de operación: depósitos, retiros y fin de sesión.
- El depósito se hará mediante la introducción de un sobre con el dinero y los datos del cliente anotados en su exterior.

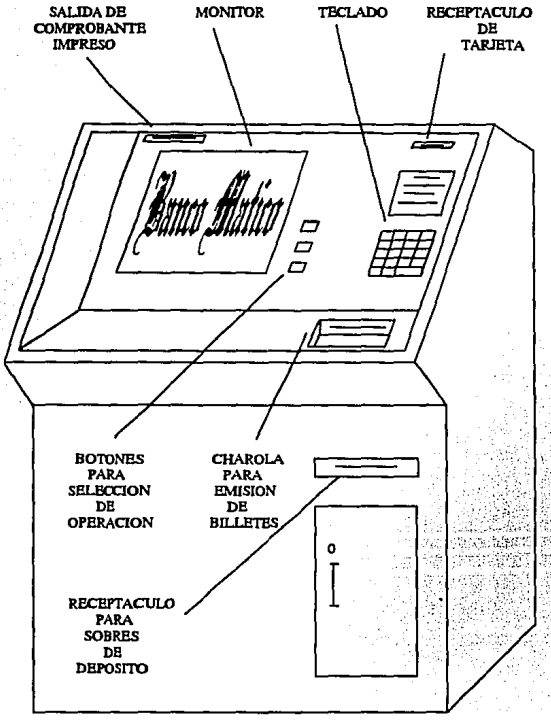


Fig. 3.9. IMPLEMENTACION FISICA DEL CAJERO AUTOMATICO

- Para retiros se verificará que la cantidad requerida por el usuario tenga fondos disponibles.
- Los retiros se harán en billetes de una sola denominación que serán expulsados por el cajero hacia una charola.
- Al finalizar las operaciones el cajero entregará al usuario su tarjeta y un comprobante impreso de las operaciones realizadas.

Como se mencionó, el planteamiento para este diseño no incluye la totalidad de funciones deseables. Por ejemplo no se han incluido opciones de validación con fines de seguridad, que una aplicación real debe considerar pero que para fines explicativos hemos omitido.

3.5.2. Algoritmo

Habiendo sido descrito el funcionamiento del problema procedemos a estructurar el diagrama a bloques del sistema. La fig. 3.10. nos muestra la interacción entre controlador y subsistemas del cajero. El diagrama muestra entradas y salidas referentes al controlador, para saber el orden en que ocurrirán se especifica el siguiente algoritmo.

a) El controlador solicita tarjeta.

b) Si la tarjeta es introducida (TAR) el receptáculo informa al controlador.

c) El controlador solicita tipo de operación (STO) al usuario a través del monitor.

d) El usuario elige operación por medio del teclado.

e) Si elige retiro (RET)

- El control solicita cantidad (STC) que se desea retirar.
- El usuario teclea la cantidad, si ocurre un error (ERR) por fondos insuficientes se solicita nuevamente la cantidad.

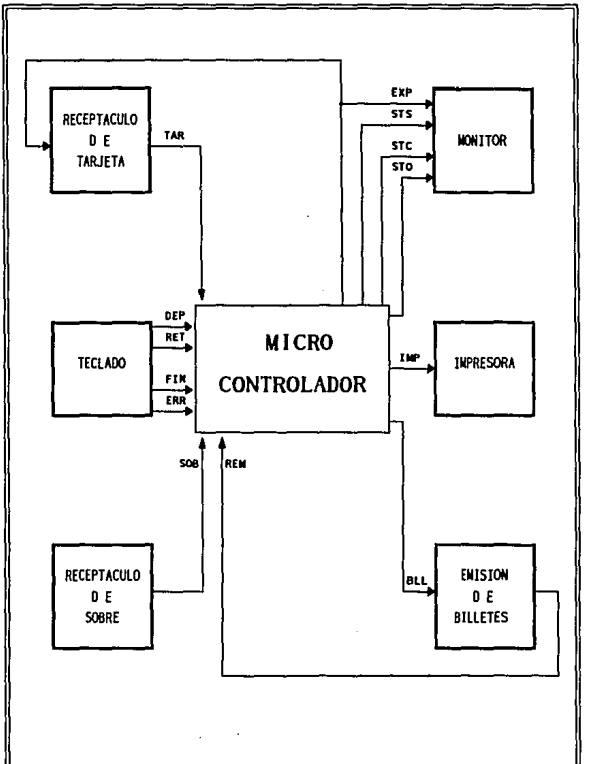


Fig. 3.10. DIAGRAMA A BLOQUES PARA EL CAJERO AUTOMATICO

- El control manda un pulso (BLL) por cada billete que debe expulsar el subsistema de emisión de billetes hasta que la cantidad requerida sea igual a la cantidad emitida (REM).
- Finaliza operación y solicita otra.

f) Si elige deposito (DEP)

- El control solicita sobre.
- El usuario debe depositar el sobre (SOB) con los datos y dinero correspondientes en el receptáculo de sobres.
- Finaliza operación y solicita otra.

g) Si elige fin de sesión

- El control indica al receptáculo de tarjetas que la expulse (EXP) y avisa al usuario con un mensaje en pantalla que retire su tarjeta.
- Imprime comprobante de operaciones realizadas y lo entrega al usuario.

Todo este proceso escrito permite la realización del autómata, la carta ASM correspondiente se muestra en la fig. 3.11, y el diagrama MDS en la fig.3.12. Ambos autómatas se implementan directamente a partir del algoritmo y son equivalentes.

El diagrama MDS resultante servirá como ejemplo para el capítulo siguiente, en el cual le aplicaremos una serie de metodologías, todas ellas tienen en común su requerimiento inicial: el planteamiento del autómata.

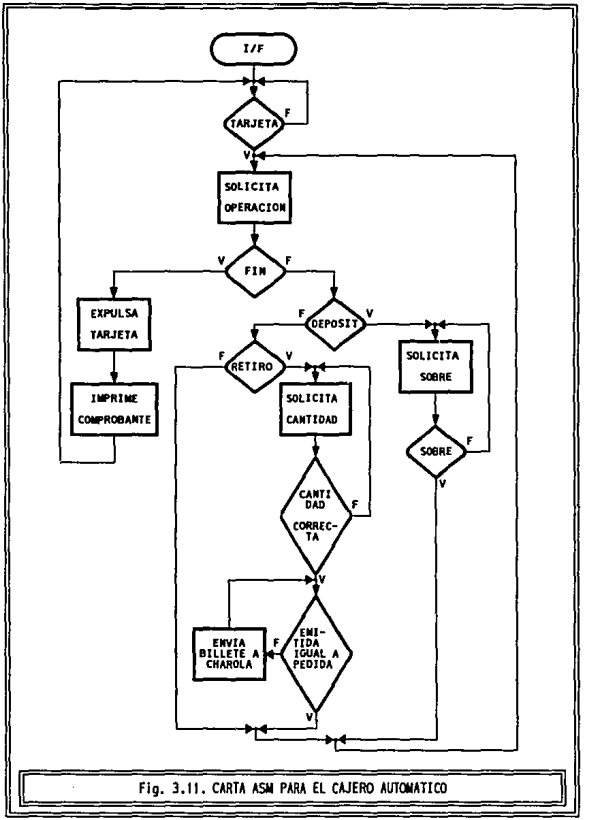
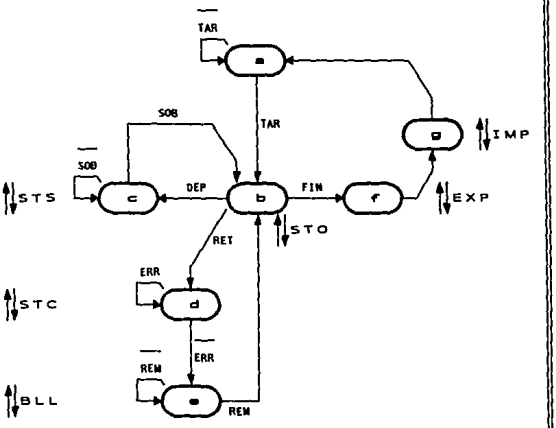


Fig. 3.11. CARTA ASM PARA EL CAJERO AUTOMATICO



DESCRIPCION DE NEMONICOS

ENTRADAS	SALIDAS	ESTADOS	
DEP: OPERACION DE DEPOSITO	BLL: ENVIA BILLETE A LA CHAROLA		C B A
ERR: EXISTE ERROR EN LA CANTIDAD INTRODUCIDA	EXP: CAJERO EXPULSA TARJETA	a	0 0 0
FIN: FIN DE SESION	IMP: IMPRIME COMPROBANTE DE OPERACION	b	0 0 1
REM: CANTIDAD REQUERIDA POR USUARIO IGUAL A EMITIDA POR CAJERO	STC: CAJERO SOLICITA CANTIDAD A RETIRAR POR EL USUARIO	c	0 1 0
SOB: FUE INSERTADO SOBRE CON CANTIDAD DEPOSITADA	STS: CAJERO SOLICITA CANTIDAD A DEPOSITAR POR USUARIO	d	0 1 1
TAR: TARJETA INSERTADA	STO: CAJERO SOLICITA TIPO DE OPERACION	e	1 0 0
		f	1 0 1
		g	1 1 0

Fig. 3.12. DIAGRAMA MDS PARA CAJERO AUTOMATICO

Capítulo Cuatro

Metodologías para la Implementación de Controladores Digitales

4.1. Lógica Combinacional

4.2. Multiplexores Directamente Direccionados

4.3. Contadores

4.4. Registros de Corrimiento

4.5. Microprogramado

4.6. Myca-1

4.7. Myca-2

4.8. MC2909

4.1. INTRODUCCIÓN

Este capítulo se integra por una serie de metodologías para el desarrollo de microcontroladores. Para cada una de ellas se han especificado las características, elementos requeridos, arquitectura y proceso detallado para su estructuración. Algunos de estos procesos son similares o idénticos en varias metodologías, motivo por el cual y para evitar repeticiones innecesarias previamente serán explicados en esta introducción y posteriormente sólo serán referenciados en las secciones adecuadas.

4.1.1. ASIGNACIÓN DE ESTADOS

Este proceso es común a toda metodología. Consiste en asignar un código binario por medio de un nemónico a cada estado del autómata. Generalmente se utilizan letras minúsculas individualmente para representar un código binario. Cada código binario utiliza un número fijo de posiciones que está en función del número de estados, el número de posiciones será aquél que sea suficiente para representar secuencialmente al número de estados. Por ejemplo, si nuestro autómata posee ocho estados, tendríamos una tabla como la siguiente:

NUMERO DE ESTADO	CÓDIGO BINARIO	NEMÓNICO
1	000	a
2	001	b
3	010	c
4	011	d
5	100	e
6	101	f
7	110	g
8	111	h

TABLA 4.1. ASIGNACION DE NEMONICOS PARA UN AUTOMATA DE OCHO ESTADOS

Se observa que requerimos tres bits para representar al código binario. A mayor número de estados corresponderá un mayor número de bits, lo cual se verá reflejado en un proceso más complejo y en mayor número de elementos al implementar al microcontrolador.

La tabla muestra códigos binarios en secuencia, cuando un número es secuencial a otro se dice que es *Continuo por Secuencia,* en la tabla el estado *c* (010) es continuo secuencialmente respecto al estado *b* (001) y discontinuo respecto al estado *a* (000).

Existe otro tipo de continuidad. Un número es *Continuo por Corrimiento* respecto a otro, si puede obtenerse este último mediante la inserción de un 0 o un 1 al código original; si la inserción se realiza por la izquierda todos los dígitos binarios se recorren una posición a la derecha eliminando el dígito extremo derecho, análogamente si la inserción se realiza por la derecha todos los dígitos se recorren una posición a la izquierda, siendo eliminado el dígito extremo izquierdo. Por ejemplo en la tabla, respecto al estado *c* (010) es continuo el estado *b* (001), el cual se forma insertando un 0 por la izquierda al estado original.

Un estado binario sólo puede ser continuo secuencialmente respecto a un sólo estado binario, en cambio ese mismo estado puede ser continuo por corrimiento respecto hasta cuatro estados. Por ejemplo el estado *d* (011) es continuo por secuencia respecto a *c* (010) y continuo por corrimiento respecto a los estados *b* (001), *f* (101), *e* (100) y *h* (111). Tal vez se pudiera pensar que el estado *d* (011) también es continuo respecto al estado *e* (100) , esto aunque aritméticamente es correcto, técnicamente no lo es, ambos estados son secuenciales, pero debido a que el estado *e* es ascendente del estado *d* , es continuo respecto a éste, pero lo contrario no es cierto.

A partir de estos tipos de continuidad obtenemos dos tipos de asignación de estados: por secuencia y por corrimiento. De las metodologías a discutir posteriormente, sólo una utilizará la *Asignación por Corrimiento* (el Método Registros de Corrimiento), las restantes harán uso de la *Asignación por Secuencia.*

La asignación de estados puede tomar diversas formas que culminarán en resultados funcionalmente iguales. El diseñador tiene la libertad de asignar los estados de la forma que lo desee, por lo que simplemente mencionaremos algunos puntos que le ayudarán a simplificar su trabajo.

- No debe asignarse un mismo código a más de un estado.
- Siempre que sea posible deben utilizarse códigos secuenciales para reducir el número de bits necesarios para representar a un autómata.
- Por estándar se acostumbra asignar el nemónico *a* correspondiente al código binario más bajo, al estado en que inicia el autómata del controlador. A partir de este estado se continua la asignación de los restantes.
- Si un estado presente parte a un solo estado futuro, debe asignarse a éste un código binario *continuo* respecto al estado presente. Al evitar discontinuidad entre estados se aprovecha la característica básica que nos proporciona el elemento principal utilizado en la implementación del microcontrolador.

4.1.2. Mapa de Estado Presente

Una vez que los estados del autómata han sido designados con un código binario y un nemónico se elabora un Mapa de Estado Presente, el cual sólo es un mapa de Karnaugh que tiene por objeto tabular al autómata.

Cada nemónico simplemente se ubica en aquella casilla cuya intersección corresponde a su código binario. Si para un código dado no se especifico un nemónico, en la casilla se coloca un asterisco, lo cual significa que ese estado no será utilizado por el autómata. El siguiente mapa de Estado Presente corresponde a la Tabla de Estados de la sección anterior.

A \ CB	00	01	11	10
0	a	c	g	e
1	b	d	h	f

4.1.3. Mapa de Salidas

Mapa análogo al de la figura anterior, se le utiliza para indicar aquellas señales que serán generadas por el controlador, deben colocarse en la casilla correspondiente al estado en que se desea aparezcan. Si un estado no posee salidas su casilla se deja en blanco o se le coloca un asterisco.

4.1.4. Tabla de Acción

Sirve para indicar la instrucción que determina el flujo de estados para un autómata dado.

En la columna Estado se coloca el nemónico correspondiente al estado presente y en la columna Acción la instrucción de transición entre el estado presente y el siguiente o siguientes, respetando la sintaxis de la metodología específica.

ESTADO	ACCION
a	Instruccion 1
b	Instruccion 2
.....	.....

TABLA 4.2. ESTRUCTURA PARA UNA TABLA DE ACCION

4.1.5. Arreglos Decisorios y Operación Columna

Para cambiar de un estado presente hacia un estado futuro, es necesario conocer el arreglo de variables adecuadas que producirán el cambio necesario en los bits que conforman al estado. Para ello se utiliza un *Arreglo Decisorio*, el cual se forma por una serie de renglones y columnas, un renglón por cada estado siguiente y una columna por cada bit de estado, cada renglón poseerá una variable asociada, la que en el autómata se haya marcado como variable condicionante al estado siguiente, cada columna del renglón indicará el dígito correspondiente al estado siguiente.

Implementado el arreglo, la definición del resultado que nos conduzca al estado apropiado de acuerdo al autómata se obtiene por medio de las reglas siguientes:

- Regla 1: Si todos los renglones de la columna contienen un cero, el resultado será también un cero.
- Regla 2: Si todos los renglones de la columna contienen un uno, el resultado será también un uno.
- Regla 3: Si en una columna un solo renglón contiene un cero y los demás contienen unos, el resultado será la negación de la variable asociada al renglón que contenga al cero.
- Regla 4: Si en una columna un solo renglón contiene un uno y los demás contienen ceros, el resultado será la variable asociada al renglón que contenga al cero.
- Regla 5: Si existe más de un renglón que contenga un cero, y más de un renglón que contenga un uno. El resultado será la suma de todas las variables asociadas a un renglón cuyo valor sea uno, o bien la negación de la suma de todas las variables asociadas a un renglón cuyo valor sea cero.

En cada metodología se indicará si es necesaria la implementación de Arreglos de Decisión, así como sus componentes y la ubicación adecuada de los resultados obtenidos mediante la Operación Columna. A través del capítulo se mostrará extensivamente este proceso por medio de ejemplos, por ahora sólo

bastará recordar las reglas presentadas, que serán de gran utilidad y aplicables a cualquier arreglo.

Nota: En ocasiones podrá aplicar más de una regla a una situación en particular. Todas serán válidas, pero es preferible elegir aquella que produzca el resultado más compacto.

4.1.6. Implementación

Toda metodología parte de un autómata y finaliza en el diagrama correspondiente al microcontrolador. Siempre que fue posible se evitó introducir en los diagramas a circuitos integrados específicos, en su lugar se prefirió mostrar el tipo genérico del elemento y las características que debe poseer, esto le da mayor libertad al diseñador que no se verá limitado a la adquisición de ciertos componentes, teniendo la oportunidad de utilizar elementos que cumplan con las características mencionadas.

4.2. LÓGICA COMBINACIONAL

4.2.1. ELEMENTOS

Utiliza como elementos exclusivamente a componentes MSI.

- Flip Flops: Serán utilizados como elementos principales de memoria. Se requiere un flip-flop por cada bit de estado.
- LÓGICA COMBINACIONAL: Requerida para el diseño del decodificador de señales de entrada y para el decodificador de estados para la obtención de señales de salida.

4.2.2. METODOLOGÍA

a) ASIGNACIÓN DE ESTADOS

A partir del autómata asignar estados por secuencia y asignar el nemónico correspondiente.

b) MAPA DE ESTADO SIGUIENTE

Elaborarlo a partir del inciso anterior.

c) MAPA DE ESTADO FUTURO

Este mapa es similar en su forma al Mapa de Estado Presente y contiene un mismo número de casillas. Cada estado presente deriva en uno o más estados siguientes, la transición a estos estados puede ser condicionada o incondicional. Si uno de los estados siguientes es igual al estado presente -efecto de *Retención*- no será tabulado a menos que existan tres o más estados siguientes. Las diferentes situaciones que pueden presentarse se ilustran a continuación.

1ER. CASO: Un estado presente da origen a un estado siguiente de manera incondicional. Ejemplo: el estado presente *a* da origen al estado *b*.

A \ B	0	1
0	a	
1		

MAPA DE ESTADO PRESENTE

A \ B	0	1
0	b	
1		

MAPA DE ESTADO SIGUIENTE

2do. caso: **Un estado presente da origen a un estado siguiente de manera condicional. Ejemplo: el estado presente *a* da origen al estado *b* por medio de la variable** VAR.

A \ B	0	1
0	a	
1		

MAPA DE ESTADO PRESENTE

A \ B	0	1
0	VAR→ b	
1		

MAPA DE ESTADO SIGUIENTE

3er. caso: **Un estado presente da origen a más de un estado siguiente de manera condicional. Cada estado siguiente es indicado en la misma casilla. Ejemplo: el estado presente *a* da origen al estado *b* por medio de la variable** V1 **y al estado *c* por medio de la variable** V2.

A \ B	0	1
0	a	
1		

MAPA DE ESTADO PRESENTE

A \ B	0	1
0	V1 →b V2 →c	
1		

MAPA DE ESTADO SIGUIENTE

4to. caso: **Un casilla del mapa de estado presente no es utilizada, contiene el símbolo no importa (asterisco), la casilla del mapa de estado siguiente tampoco importará.**

A \ B	0	1
0	*	
1		

MAPA DE ESTADO PRESENTE

A \ B	0	1
0	*	
1		

MAPA DE ESTADO SIGUIENTE

d) Elaborar Mapa de Salidas

e) Mapas de Entrada a Flip Flops

Por cada bit de estado requeriremos un flip-flop y llenar su mapa correspondiente. Sin importar el número de bits de estado el proceso siempre es similar. Para ejemplificar utilizaremos tres bits de estado referenciados como C, B y A, donde A es el bit menos significativo. Las anotaciones en cada mapa de flip-flop se realizan en la casilla análoga al mapa de estado siguiente.

1er. caso: si el estado siguiente posee un estado incondicional, sus bits serán colocados en los mapas de flip-flop respetando la secuencia CBA.

Ejemplo: Si el estado *b* representa al código 001.

A \ CB	00	01	11	10
0	b			
1				

MAPA DE ESTADO SIGUIENTE

A \ B	0	1
0	0	
1		

MAPA DE FLIP FLOP C

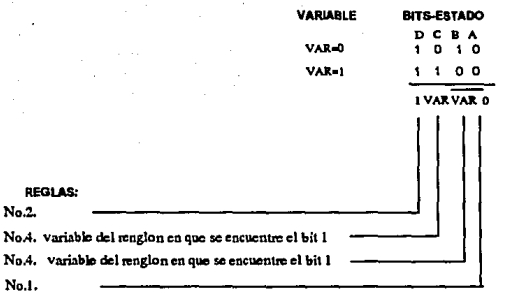
A \ B	0	1
0	0	
1		

MAPA DE FLIP FLOP B

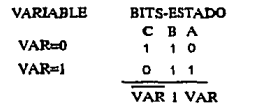
A \ B	0	1
0	1	
1		

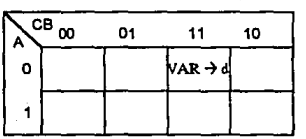
MAPA DE FLIP FLOP A

2do. caso: Si el estado siguiente posee un solo estado condicional, se realiza un Arreglo de Decisión como el siguiente, en un renglón se coloca el estado siguiente si la variable condicionante es afirmada (igual a 1), y en otro renglón el estado siguiente si la variable es negada (igual a cero). Posteriormente se realiza una operación bit a bit por columna respetando las reglas ya vistas.

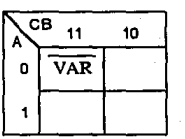


Ejemplo: Si en la localidad 110 tenemos como estado siguiente al nemónico *d* con código binario 011, tabulamos de la siguiente forma:

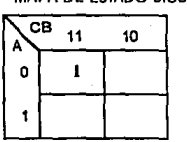




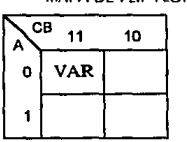
MAPA DE ESTADO SIGUIENTE



MAPA DE FLIP FLOP C



MAPA DE FLIP FLOP B



MAPA DE FLIP FLOP A

3ER. CASO: Si el estado siguiente posee dos estados condicionales a diferentes variables, se realiza un arreglo similar al caso dos, sustituyendo la variable afirmada y negada por las dos variables condicionantes, y realizando para cada bit de estado una operación columna.

Ejemplo: En la casilla se encuentran tres estados siguientes. Con V1 se pasa al estado f (101) con V2 al estado h (111) y con V3 al estado g (110).

VARIABLE	BITS-ESTADO C B A
V1	1 0 0
V2	0 1 0
V3	1 1 1

$\overline{V2}\ \overline{V1}\ V3$

A \ CB	00	01	11	10
0			V1 → e V2 → c V3 → b	
1				

MAPA DE ESTADO SIGUIENTE

A \ CB	11	10
0	V2	
1		

MAPA DE FLIP FLOP C

A \ CB	11	10
0	V1	
1		

MAPA DE FLIP FLOP B

A \ CB	11	10
0	V3	
1		

MAPA DE FLIP FLOP A

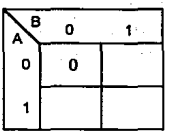
4TO. CASO: Si la casilla es un estado no importa (contiene un asterisco), las casillas análogas a todos los mapas se llenan con un cero.

A \ CB	00	01	11	10
0	*			
1				

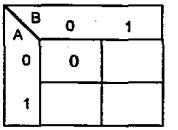
MAPA DE ESTADO SIGUIENTE

A \ B	0	1
0	0	
1		

MAPA DE FLIP FLOP C



MAPA DE FLIP FLOP B



MAPA DE FLIP FLOP A

f) **Encontrar la** EXPRESIÓN RESULTANTE **para cada Mapa de flip-flop mediante su resolución.**

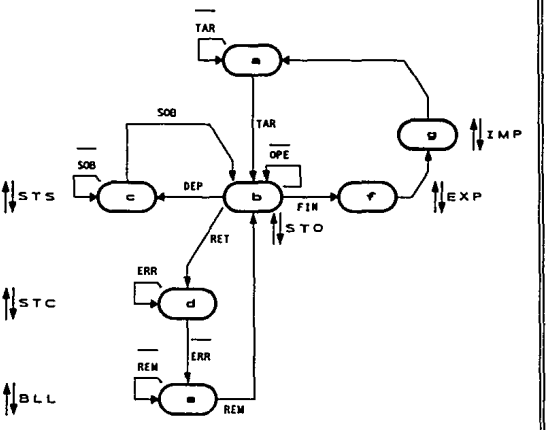
g) IMPLEMENTACIÓN

Implementación lógica de nuestro controlador de acuerdo a la arquitectura general. Las expresiones obtenidas en el punto anterior se implementan por medio de compuertas y se dirigen hacia cada uno de los flip-flops utilizados. Los estados que se obtienen a la salida de los flip-flops son decodificados también por medio de lógica combinacional de acuerdo al Mapa de Salidas.

4.2.3. SOLUCIÓN AL CAJERO AUTOMÁTICO

Las páginas siguientes muestran por incisos la solución al cajero automático mediante Lógica Combinacional. Nótese que se incluye en recuadros la obtención de cada estado de los Mapas de Entrada para cada Flip-Flop.

Como se verá más adelante esta metodología implica una serie de desventajas en relación a otras, ya que utiliza un gran número de componentes y su desarrollo es más tedioso.



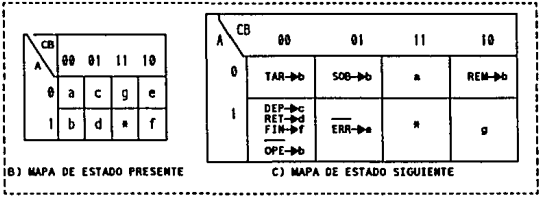
DESCRIPCION DE NEMONICOS

ENTRADAS	SALIDAS
DEP: OPERACION DE DEPOSITO	BLL: ENVIA BILLETE A LA CHAROLA
ERR: EXISTE ERROR EN LA CANTIDAD INTRODUCIDA	EXP: CAJERO EXPULSA TARJETA
FIN: FIN DE SESION	IMP: IMPRIME COMPROBANTE DE OPERACION
REM: CANTIDAD REQUERIDA POR USUARIO IGUAL A EMITIDA POR CAJERO	STC: CAJERO SOLICITA CANTIDAD A RETIRAR POR EL USUARIO
SOB: FUE INSERTADO SOBRE CON CANTIDAD DEPOSITADA	STS: CAJERO SOLICITA CANTIDAD A DEPOSITAR POR USUARIO
TAR: TARJETA INSERTADA	STO: CAJERO SOLICITA TIPO DE OPERACION
OPE: YA SE ELIGIO OPERACION	

ESTADOS	C	B	A
a	0	0	0
b	0	0	1
c	0	1	0
d	0	1	1
e	1	0	0
f	1	0	1
g	1	1	0

A) ASIGNACION DE ESTADOS POR SECUENCIA

Fig. 4.1. SOLUCION AL C.A. POR LOGICA COMBINACIONAL Y MULTIPLEXORES (1a. Parte)



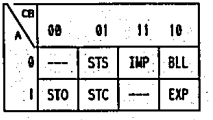


FIG. 4.2. SOLUCION A C.A. POR LOGICA COMBINACIONAL Y MULTIPLEXORES (2a. Parte)

A \ CB	00	01	11	10
0	0	0	0	$\overline{REM}$
1	FIN	$\overline{ERR}$	0	1

MAPA C

A \ CB	00	01	11	10
0	0	$\overline{SOB}$	0	0
1	RET+ DEP	ERR	0	1

MAPA B

A \ CB	00	01	11	10
0	TAR	SOB	0	REM
1	$\overline{DEP}$	ERR	0	0

MAPA A

E) MAPAS DE ENTRADA A FLIP-FLOPS

$C = \overline{REM}[\overline{CB}]+FIN[\overline{CBA}]$ $+\overline{ERR}[\overline{CBA}]+\overline{CBA}$

$B = \overline{SOB}[\overline{CBA}]+[RET+DEP][\overline{BA}]$ $+ERR[\overline{CBA}]+\overline{CBA}$

$A = TAR[\overline{CBA}]+SOB[\overline{CBA}]+$ $\overline{REM}[\overline{CBA}]+ERR[\overline{CBA}]+$ $\overline{DEP}[\overline{CBA}]$

F) RESOLUCION DE MAPAS DE ENTRADA A FLIP-FLOPS

ESTADO A

	C	B	A	
$\overline{TAR}$	0	0	0	
TAR	0	0	1	
	0	0	TAR	

ESTADO B

	C	B	A	
DEP	0	1	0	
RET	0	1	1	
$\overline{FIN}$	1	0	1	
$\overline{OPE}$	0	0	1	
	FIN	RET+ DEP	$\overline{DEP}$	

ESTADO C

	C	B	A	
$\overline{SOB}$	0	1	0	
SOB	0	0	1	
	0	$\overline{SOB}$	SOB	

ESTADO D

	C	B	A	
$\overline{ERR}$	1	0	0	
ERR	0	1	1	
	$\overline{ERR}$	ERR	ERR	

ESTADO E

	C	B	A	
$\overline{REM}$	1	0	0	
REM	0	0	1	
	$\overline{REM}$	0	REM	

ESTADO F

INCONDICIONAL

C B A

g → 1 1 0

ESTADO G

INCONDICIONAL

C B A

a → 0 0 0

ESTADO *

NO IMPORTA

LLENAR CON CEROS

C B A

0 0 0

CUADROS PARA OBTENCION DE MAPAS DE ENTRADA A FLIP-FLOPS

FIG. 4.3. SOLUCION A C.A. POR LOGICA COMBINACIONAL Y MULTIPLEXORES (3a. Parte)

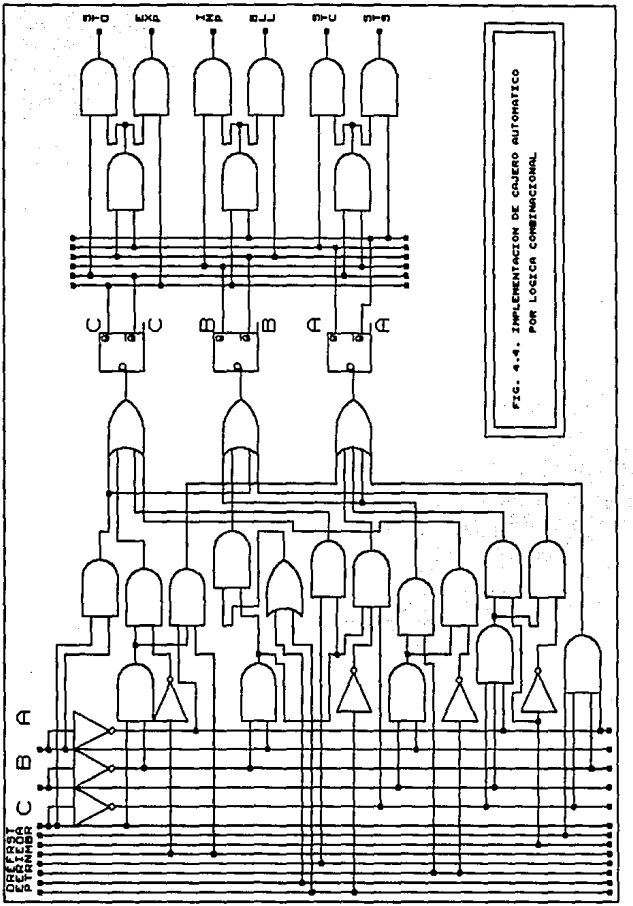


FIG. 4.4. IMPLEMENTACION DE CAJERO AUTOMATICO POR LOGICA COMBINACIONAL

4.3. Multiplexores Direccionados

Esta metodología constituye una alternativa más óptima en relación al método de Lógica Combinacional. Sólo mencionaremos los cambios necesarios haciendo referencia a la sección previa.

4.3.1. Elementos

- Flip Flops: Serán utilizados como elementos principales de memoria. Se requiere un flip-flop por cada bit de estado.
- Multiplexores: Se requiere un multiplexor por cada flip-flop para dirigir hacia él la entrada adecuada.
- Decodificador: Reemplaza a las compuertas en la decodificación de estados para obtener las señales deseadas.

4.3.2. Metodología

La metodología empleada es prácticamente la misma que se utilizó en el método anterior. Para su aplicación utilice los incisos a, b, c, d y e.

a) Asignación de Estados
b) Mapa de Estado Presente
c) Mapa de Estado Futuro
d) Elaborar Mapa de Salidas
e) Mapas de Entrada a Flip Flops

Las reglas para su aplicación son idénticas a las ya mencionadas. El inciso f)que hace referencia a la resolución de mapas no se requiere, ya que con el uso de multiplexores la tarea se facilita enormemente, en su lugar se ejecuta el paso siguiente:

f) Cada mapa obtenido servirá para asociar a un mux las entradas requeridas. Simplemente se asocia el código binario o variable ubicado en cada casilla a la terminal del multiplexor que posea el mismo número a la intersección en el mapa.

Ejemplo: Si tenemos un mapa de flip-flop como el siguiente, tendremos que aplicar al multiplexor en su terminal 3 (11 en binario) la variable V1, y en su terminal 1 (01 en binario) un bajo voltaje (indicado con 0).

A \ B	0	1
0		
1	0	V1

g) Implementación

Aplicar todas las entradas requeridas a los multiplexores dirigiendo la salida de cada uno de ellos a su flip-flop correspondiente. Todos los multiplexores se conectan en paralelo y son direccionados por las salidas de los flip-flops. Estas salidas son decodificadas directamente, asociando cada terminal del decodificador de acuerdo a las localidades del Mapa de Salidas.

Es preciso observar que en algunos casos para la obtención de las entradas dirigidas a los multiplexores se requerirá el uso de algunas compuertas, a este conjunto de circuitos SSI se le denomina *Lógica Reducida.*

4.3.3. Solución al Cajero Automático

En la página siguiente se muestra la implementación para el cajero autómatico, compárela con la solución implementada en la sección anterior, a fin de notar aquellos puntos que se han modificado. Sin duda notará como se ha simplificado nuestro trabajo.

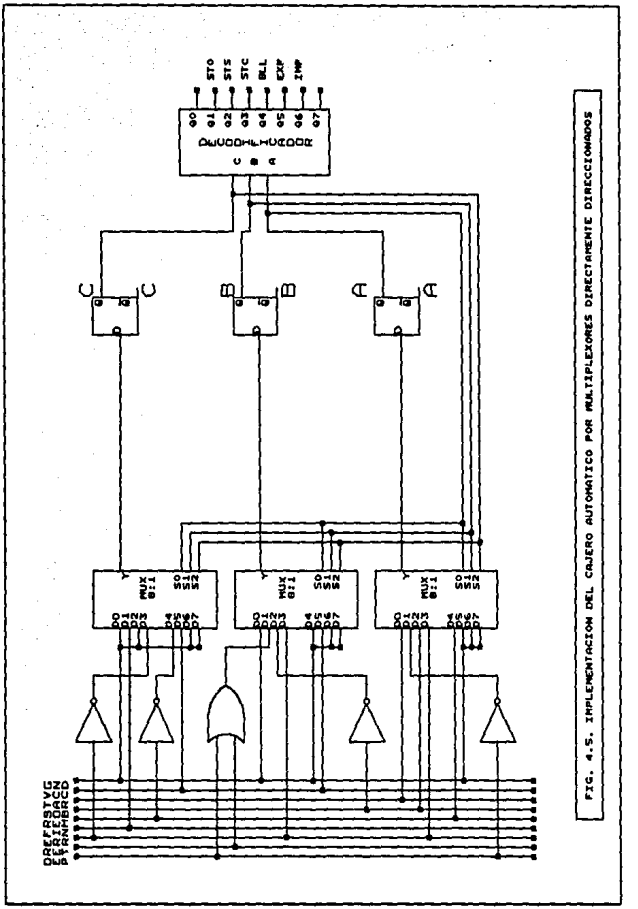


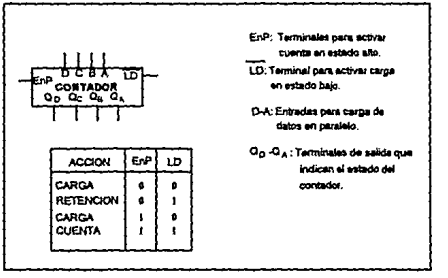
FIG. 4.5. IMPLEMENTACION DEL CAJERO AUTOMATICO POR MULTIPLEXORES DIRECTAMENTE DIRECCIONADOS

4.4. CONTADORES

4.4.1. ELEMENTOS

- CONTADOR: Para la implementación de este tipo de controlador se requiere un contador que permita carga de bits en paralelo y que posea entradas de habilitación tanto para contar, como para carga de datos. La simple característica de cuenta nos permite emplear acciones de secuencia y la opción de carga de datos en paralelo nos permitirá ejecutar acciones de brinco.

Nuestro contador dependiendo de su tipo puede poseer algunas entradas extras de control, las cuales deben ser verificadas en el manual correspondiente. Por ejemplo algunos contadores poseen dos entradas para habilitación de cuenta, en tal caso ambas deben interconectarse o bien una de ellas fijarse a estado de habilitación permanente.



ACCION	EnP	LD
CARGA	0	0
RETENCION	0	1
CARGA	1	0
CUENTA	1	1

TABLA 4.3. DIAGRAMA LOGICO Y TABLA DE ACCION CORRESPONDIENTE A NUESTRO CONTADOR

- MULTIPLEXORES: Se requieren para dirigir las variables de entrada hacia las terminales de control del contador.

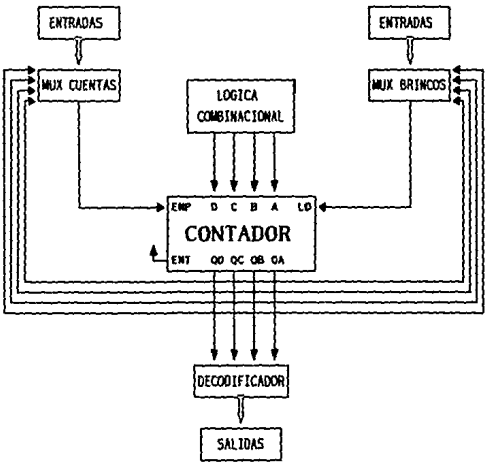


Fig. 4.6. ARQUITECTURA GENERAL PARA EL METODO CONTADORES

- Decodificador: Las salidas del contador (estados del controlador) serán decodificadas para obtener las salidas.

4.4.2. Instrucciones

Con las características de nuestro contador es posible implementar el siguiente cuadro de instrucciones o acciones.

INSTRUCCIÓN	DESCRIPCIÓN
CC (VAR)	CUENTA CONDICIONADA a la variable VAR
CI	CUENTA INCONDICIONAL
BC (VAR) [DIR]	BRINCO CONDICIONADO a la variable VAR ydirigido hacia la dirección DIR
BI	BRINCO INCONDICIONAL
C/B (V1)(V2)[DIR]	CUENTA-BRINCO.Si la variable V1 es verdadera se realiza una cuenta. Si la variable V2 es verdadera se realiza un brinco hacia la dirección DIR.
RT (VAR)	RETENCION CONDICIONADA

TABLA 4.4. INSTRUCCIONES SOPORTADAS POR NUESTRO CONTADOR

4.4.3. Metodología

a) Asignación de Estados

A partir del autómata asignar estados por secuencia y asignar el nemónico correspondiente.

b) Mapa de Estado Presente

Tabularlo a partir del inciso anterior.

c) Asignación de acciones Inter Estados

De acuerdo a los códigos asignados a cada estado se deberá establecer el tipo de acción adecuada para llegar de un estado presente a un estado futuro. La elección se realizará de la forma siguiente:

- Si el estado siguiente es igual al presente, la acción será Retención (RT).
- Si el estado siguiente es continuo al estado actual y existe una variable entre ambos, la acción será Cuenta Condicional (CC).
- Si el estado siguiente es continuo al estado actual y no existe una variable entre ambos, la acción será Cuenta Incondicional (CI).
- Si el estado siguiente no es continuo al estado actual y existe una variable entre ambos, la acción será Brinco Condicional (BC).
- Si el estado siguiente no es continuo al estado actual y no existe una variable entre ambos, la acción será Brinco Incondicional (BI).

Tabular esta información en el Mapa de Estados Siguientes. La retención sólo se tabula si para un estado presente existen tres o más estados siguientes.

d) Elaborar Mapa de Salidas

e) Llenar Mapas de Control

Cada mapa de control tendrá un número de localidades igual al mapa de estados. Para cada localidad del Mapa de Acciones hacemos lo siguiente:

1er. caso: Si la acción es CC colocamos la variable de transición en la localidad análoga del Mapa de Cuenta y un uno en el Mapa de Carga.

A \ B	0	1
0	CC VR→g	
1		

MAPA DE ACCION

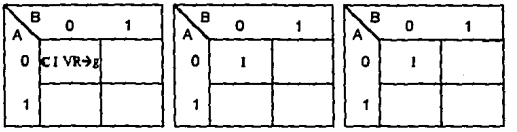
A \ B	0	1
0	VR	
1		

MAPA DE CUENTA

A \ B	0	1
0	1	
1		

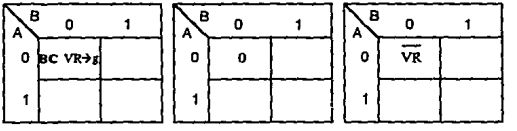
MAPA DE CARGA

2do. caso: **Si la acción es CI colocamos un uno en la localidad del Mapa de Cuenta y un uno en el Mapa de Carga.**



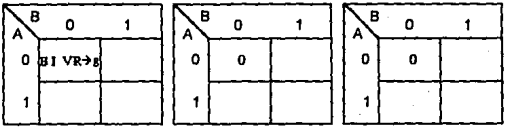
MAPA DE ACCION MAPA DE CUENTA MAPA DE CARGA

3er. caso: **Si la acción es BC colocamos un cero en la localidad del Mapa de Cuenta y la variable negada en el Mapa de Carga.**



MAPA DE ACCION MAPA DE CUENTA MAPA DE CARGA

4to. caso: **Si la acción es BI colocamos un cero en el Mapa de Cuenta y un cero en el Mapa de Carga.**



MAPA DE ACCION MAPA DE CUENTA MAPA DE CARGA

5to. caso: **Si existe más de una acción en una sola casilla, se realiza un Arreglo de Decisión y la correspondiente operación columna para cada bit de control.**

Ejemplo: Si para una casilla tenemos una cuenta condicionada a la variable V1 y un brinco condicionado a la variable VR.

A \ B	0	1
0	CC V1→a BC VR→g	
1		

MAPA DE ACCION

	CUENTA	CARGA
V1	1	1
VR	0	0
	V1	V1

A \ B	0	1
0	V1	
1		

MAPA DE CUENTA

A \ B	0	1
0	V1	
1		

MAPA DE CARGA

6TO. CASO: Si la casilla contiene asteriscos (estado no utilizado), por convención ambos mapas se llenarán con ceros.

A \ B	0	1
0	*	
1		*

MAPA DE ACCION

A \ B	0	1
0	0	
1		0

MAPA DE CUENTA

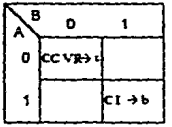
A \ B	0	1
0	0	
1		0

MAPA DE CARGA

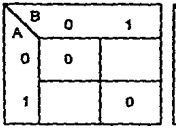
f) LLENAR MAPAS DE CARGA EN PARALELO

Por cada bit de entrada tendremos un mapa. Para cada estado se realiza el proceso siguiente:

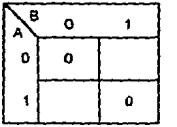
1ER. CASO: Si el estado siguiente es una cuenta (condicional o incodicional) se coloca un cero en las localidades análogas de todos los mapas.



MAPA DE ACCION

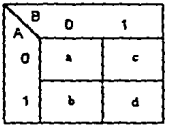


MAPA DE CARGA B

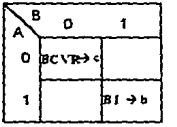


MAPA DE CARGA A

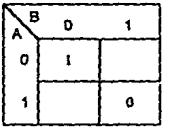
2do. caso. Si la acción entre estado presente y el siguiente es un brinco (condicional o incodicional) se coloca en cada mapa el bit correspondiente al estado siguiente.



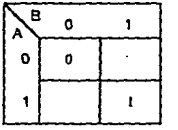
MAPA ESTADO PRESENTE



MAPA DE ACCION



MAPA DE CARGA B

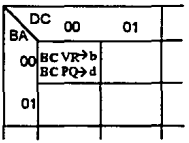


MAPA DE CARGA A

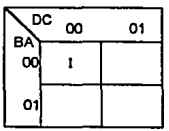
3er. caso. Si en una misma casilla hay una cuenta y un brinco, simplemente se toma la acción del brinco.

4to. caso: Si en una misma casilla hay más de un brinco, se realiza un arreglo de decisión y su operación columna.

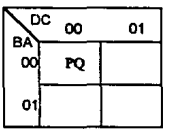
Ejemplo: Si *b* representa al estado 1010 y *d* al estado 1100, tendremos el arreglo siguiente:



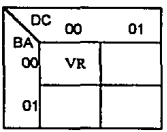
VARIABLE	ESTADO	BITS-ESTADO
		D C B A
VR	b	1 0 1 0
PQ	d	1 1 0 0
		1 PQ VR 0



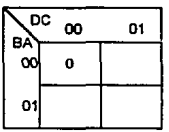
MAPA DE CARGA D



MAPA DE CARGA C



MAPA DE CARGA B



MAPA DE CARGA A

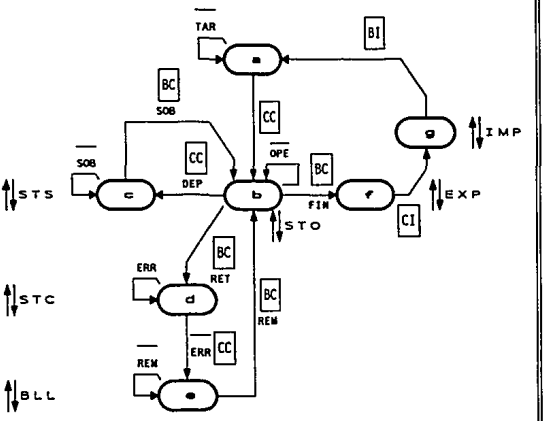
g) **Encontrar la** EXPRESIÓN RESULTANTE **para cada mapa de carga en paralelo.**

h) IMPLEMENTACIÓN

Implementación lógica de nuestro controlador de acuerdo a la arquitectura general.

4.4.4. Solución al Cajero Automático

Las páginas siguientes muestran por incisos la solución al cajero automático mediante contadores. Nótese que se incluye en recuadros la obtención de cada estado de los mapas de carga en paralelo.



DESCRIPCION DE NEMONICOS

ENTRADAS

- DEP: OPERACION DE DEPOSITO
- ERR: EXISTE ERROR EN LA CANTIDAD INTRODUCIDA
- FIN: FIN DE SESION
- REM: CANTIDAD REQUERIDA POR USUARIO IGUAL A EMITIDA POR CAJERO
- SOB: FUE INSERTADO SOBRE CON CANTIDAD DEPOSITADA
- TAR: TARJETA INSERTADA
- OPE: YA SE ELIGIO OPERACION

SALIDAS

- BLL: ENVIA BILLETE A LA CHAROLA
- EXP: CAJERO EXPULSA TARJETA
- IMP: IMPRIME COMPROBANTE DE OPERACION
- STC: CAJERO SOLICITA CANTIDAD A RETIRAR POR EL USUARIO
- STS: CAJERO SOLICITA CANTIDAD A DEPOSITAR POR USUARIO
- STO: CAJERO SOLICITA TIPO DE OPERACION

ESTADOS

	C	B	A
a	0	0	0
b	0	0	1
c	0	1	0
d	0	1	1
e	1	0	0
f	1	0	1
g	1	1	0

A) ASIGNACION DE ESTADOS POR SECUENCIA

Fig. 4.7. SOLUCION PARA CAJERO AUTOMATICO POR CONTADORES (1a. Parte)

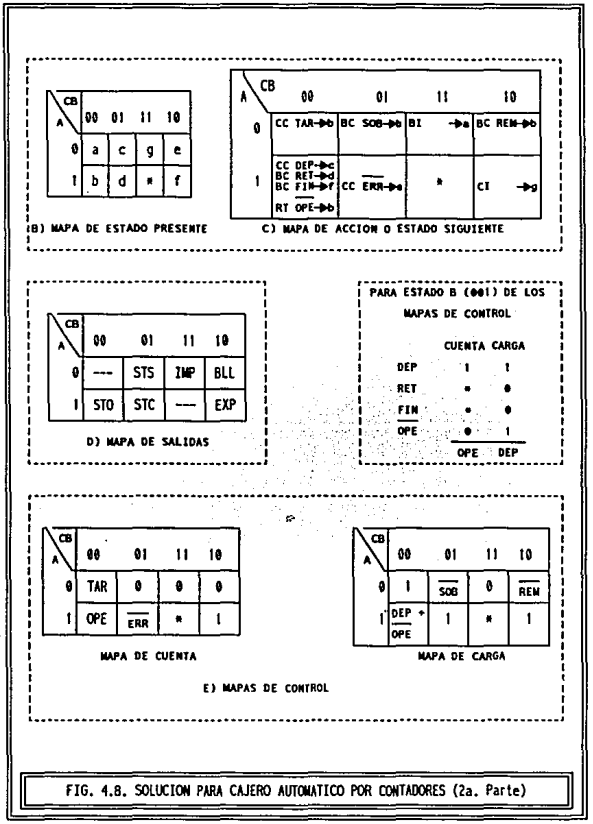
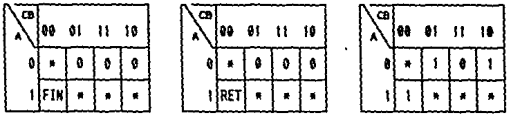


FIG. 4.8. SOLUCION PARA CAJERO AUTOMATICO POR CONTADORES (2a. Parte)



E) MAPAS DE CARGA EN PARALELO

C = FIN [A] B = RET [A] $A = \overline{C} + \overline{B}$

F) RESOLUCION DE MAPAS DE CARGA EN PARALELO

ESTADO A	ESTADO B	ESTADO C	ESTADO D
CUENTA BITS : C B A * * *	C B A BC RET 0 1 1 BC FIN 1 0 1 FIN RET 1	BRINCO : b BITS : C B A ESTADO b: 0 0 1	CUENTA BITS : C B A * * *
ESTADO E	**ESTADO F**	**ESTADO G**	**ESTADO ***
BRINCO : b BITS : C B A ESTADO b: 0 0 1	CUENTA BITS : C B A * * *	BRINCO : a BITS : C B A ESTADO a: 0 0 0	NO IMPORTA BITS : C B A 0 0 0

CUADROS PARA OBTENCION DE MAPAS DE CARGA EN PARALELO

FIG. 4.9. SOLUCION PARA CAJERO AUTOMATICO POR CONTADORES (3a. Parte)

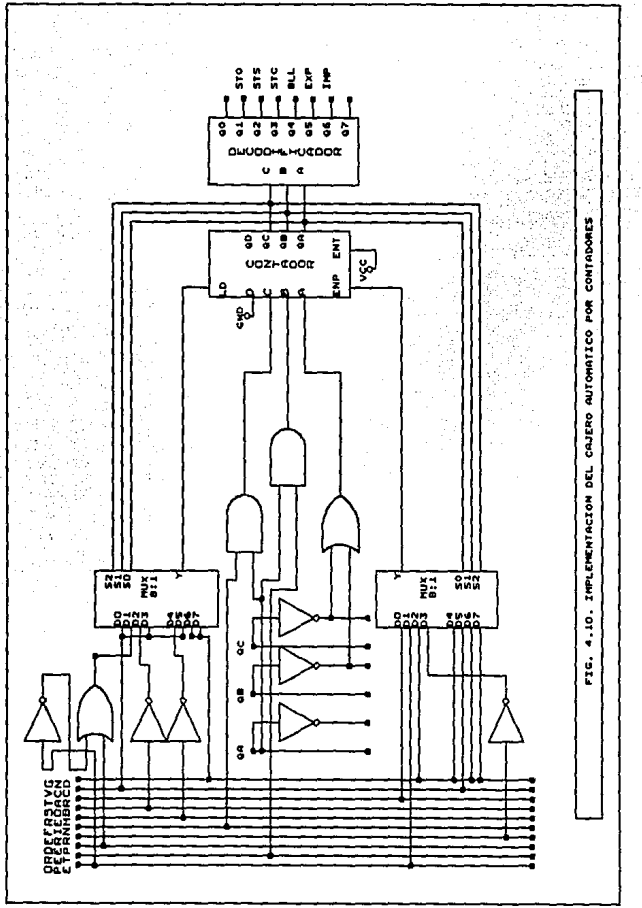
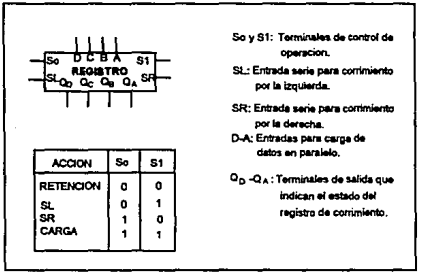


FIG. 4.10. IMPLEMENTACION DEL CAJERO AUTOMATICO POR CONTADORES

4.5. REGISTROS DE CORRIMIENTO

4.5.1. ELEMENTOS

- REGISTRO DE CORRIMIENTO: **Para la implementación de este tipo de controlador se requiere un dispositivo que permita corrimiento de bits en ambos sentidos, así como su carga en paralelo. Además debe poseer entradas de habilitación para el tipo de acción a ejecutar.**



ACCION	So	S1
RETENCION	0	0
SL	0	1
SR	1	0
CARGA	1	1

TABLA 4.5. DIAGRAMA LOGICO Y TABLA DE ACCION PARA NUESTRO REGISTRO DE CORRIMIENTO

- MULTIPLEXORES: **Se requieren para dirigir las variables de entrada hacia las terminales de control So y S1 del registro de corrimiento.**
- DECODIFICADOR: **Las salidas del contador (estados del controlador) serán decodificadas para obtener las salidas.**
- LÓGICA COMBINACIONAL: **Para fijar los bits que serán cargados en serie y en paralelo.**

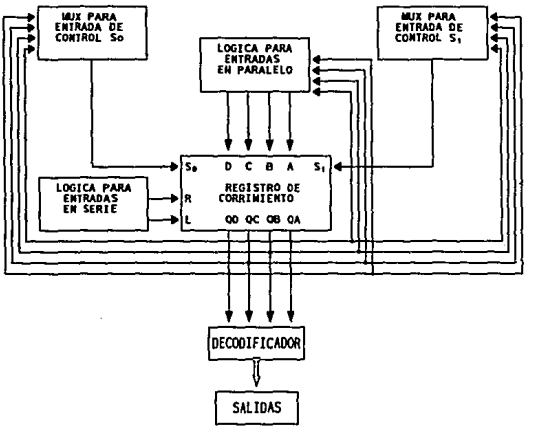


Fig. 4.11. ARQUITECURA GENERAL PARA EL METODO REGISTROS DE CORRIMIENTO

4.5.2. Instrucciones

Con las características de un registro de corrimiento es posible implementar el siguiente cuadro de instrucciones.

INSTRUCCIÓN	DESCRIPCIÓN
SR0 VR	Desplazamiento con cero a la derecha condicionado a la variable VR
SR0	Desplazamiento con cero a la derecha incondicional
SR1 VR	Desplazamiento con uno a la derecha condicionado a la variable VR
SR1	Desplazamiento con uno a la derecha incondicional
SL0 VR	Desplazamiento con cero a la izquierda condicionado a la variable VR
SL0	Desplazamiento con cero a la izquierda incondicional
SL1 VR	Desplazamiento con uno a la izquierda condicionado a la variable VR
SL1	Desplazamiento con uno a la izquierda incondicional
BC VR	Brinco Condicionado a la variable VR
BI	Brinco Incondicional
RT VR	Retención Condicionada a la variable VR

TABLA 4.8. INSTRUCCIONES SOPORTADAS POR UN REGISTRO DE CORRIMIENTO

4.5.3. Metodología

a) Asignación de Estados

A partir del autómata asignar estados por corrimiento y sus nemónicos correspondientes.

b) Mapa de Estado Presente

Tabularlo a partir del inciso anterior.

c) Elaborar Mapa de Salidas

d) Mapa de Acción

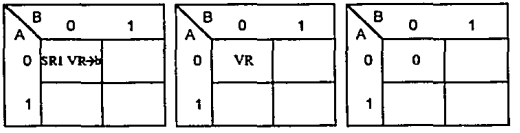
De acuerdo a los códigos asignados a cada estado se deberá establecer el tipo de acción adecuada para llegar de un estado presente a un estado futuro. Las instrucciones seleccionadas así como el estado asociado al que conducen se tabulan en un Mapa de Acción.

Si el estado siguiente es igual al presente, la acción será Retención (RT), la cual sólo se tabulará si el estado presente tiene tres o más estados siguientes.

e) Llenar Mapas de Control

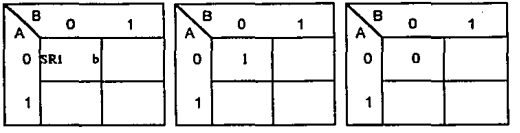
Se requiere la tabulación de dos mapas correspondientes a las terminales de control So y S1 del registro de corrimiento. Cada casilla de ambos mapas se llenará de acuerdo a la instrucción ubicada en la casilla análoga del Mapa de Acción.

1er. caso: Si la acción es SR0 o SR1 condicionada, colocamos la variable de transición en el Mapa So y un cero en el Mapa S1.



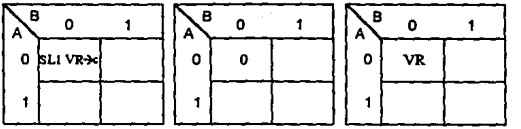
MAPA DE ACCIÓN MAPA So MAPA S1

2do. caso: Si la acción es SR0 o SR1 incondicional, colocamos un uno en la localidad del Mapa So y un cero en el Mapa S1.



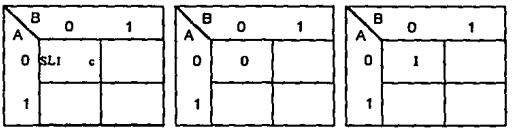
MAPA DE ACCIÓN MAPA So MAPA S1

3er. caso: Si la acción es SL0 o SL1 condicionada, colocamos un cero en la localidad del Mapa So y la variable en el Mapa S1.



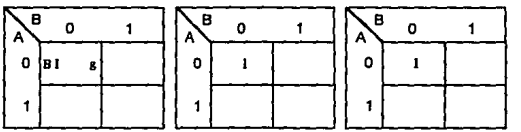
MAPA DE ACCIÓN MAPA So MAPA S1

4TO. CASO: **Si la acción es SLO o SLI incondicional, colocamos un cero en el Mapa So y un uno en el Mapa S1.**



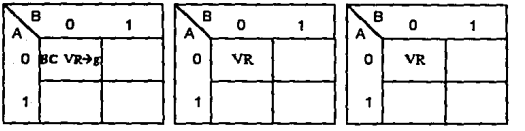
MAPA DE ACCIÓN MAPA So MAPA S1

5TO. CASO: **Si la acción es BI colocamos un uno en ambos mapas.**



MAPA DE ACCIÓN MAPA So MAPA S1

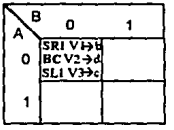
6TO. CASO: **Si la acción es BC colocamos la variable de transición en ambos mapas.**



MAPA DE ACCIÓN MAPA So MAPA S1

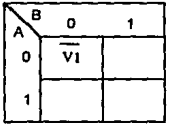
7MO. CASO: **Si existe más de una acción en una sola casilla, se realiza un Arreglo de Decisión y la correspondiente operación columna para cada bit de control.**

Ejemplo: Si para una casilla tenemos dos desplazamientos y un brinco condicionados. Donde *a* representa al estado 01, *c* al estado 10 y *d* al estado 11. Resolvemos de la siguiente manera:

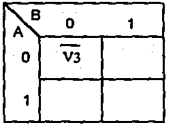


MAPA DE ACCIÓN

	So	S1
V1	0	1
V2	1	1
V3	1	0
	$\overline{V1}$	$\overline{V3}$

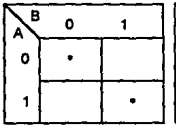


MAPA So



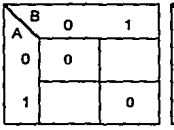
MAPA S1

8VO. CASO: **Si la casilla contiene asteriscos (estados no utilizados), por convención ambos mapas se llenarán con ceros.**



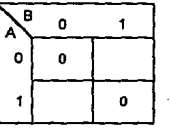
A \ B	0	1
0	*	
1		*

MAPA DE ACCIÓN



A \ B	0	1
0	0	
1		0

MAPA DE CUENTA



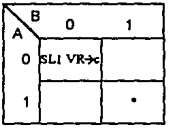
A \ B	0	1
0	0	
1		0

MAPA DE CARGA

f) LLENAR MAPAS DE CARGA EN PARALELO

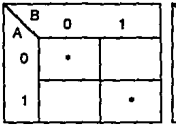
Se requiere un mapa por cada bit de estado, su tabulación indica los brincos necesarios por el autómata, cada casilla se llenará de acuerdo al Mapa de Acción.

1ER. CASO: Si la casilla contiene estados no utilizados o solamente instrucciones de corrimiento, los mapas serán llenados con asteriscos.



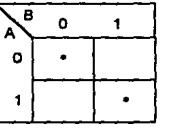
A \ B	0	1
0	SLI VR→c	
1		*

MAPA DE ACCIÓN



A \ B	0	1
0	*	
1		*

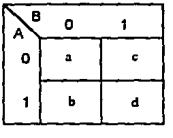
MAPA DE CARGA B



A \ B	0	1
0	*	
1		*

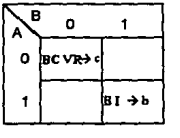
MAPA DE CARGA A

2do. CASO: Si la acción entre estado presente y el siguiente es un brinco (condicional o incondicional) se coloca en cada mapa el bit correspondiente al estado siguiente.



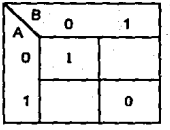
A \ B	0	1
0	a	c
1	b	d

MAPA ESTADO PRESENTE

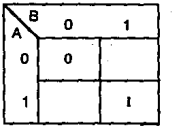


A \ B	0	1
0	BC VR→ c	
1		BI → b

MAPA DE ACCIÓN



MAPA DE CARGA B

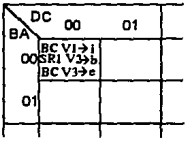


MAPA DE CARGA A

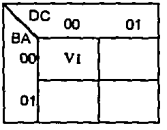
3ER. CASO: Si la casilla contiene brincos y corrimientos, sólo se toman en cuenta los primeros.

4TO. CASO: Si en una misma casilla hay más de un brinco, se realiza un arreglo de decisión y su operación columna.

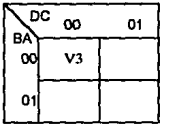
Ejemplo: Si *i* representa al estado 1000 y *e* al estado 0100, tendremos el arreglo siguiente:



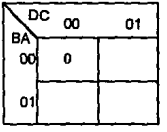
VARIABLE	ESTADO	BITS-ESTADO
		D C B A
V1	i	1 0 0 0
V3	e	0 1 0 0
		V1 V3 0 0



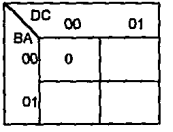
MAPA DE CARGA D



MAPA DE CARGA C



MAPA DE CARGA B



MAPA DE CARGA A

g) Mapas de Carga en Serie

Para controlar los corrimientos del autómata, se utilizan dos mapas para tabular las entradas requeridas por las terminales de control SL (corrimiento a la izquierda) y SR (corrimiento a la derecha).

1er. caso: Si la casilla contiene estados no utilizados o brincos, ambos mapas se llenan con ceros.

2do. caso: Si la instrucción es SLO condicionada o incondicional, se coloca cero en SL y asterisco en SR.

3er. caso: Si la instrucción es SL1 condicionada o incondicional, se coloca uno en SL y asterisco en SR.

4to. caso: Si la instrucción es SR0 condicionada o incondicional, se coloca cero en SR y asterisco en SL.

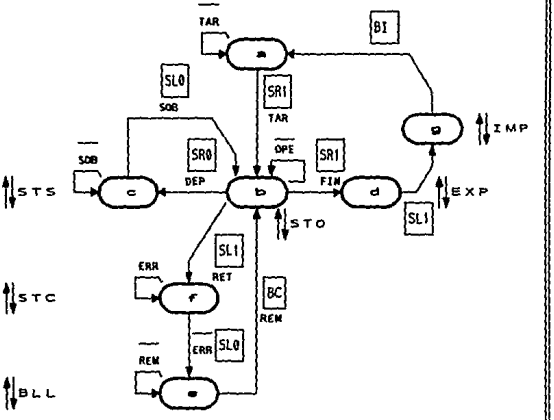
5to. caso: Si la instrucción es SR1 condicionada o incondicional, se coloca uno en SR y asterisco en SL.

h) Implementación

Implementación lógica de nuestro controlador de acuerdo a la arquitectura general.

4.5.4. Solución al Cajero Automático

Las páginas siguientes muestran por incisos la solución al cajero automático mediante registros de corrimiento. Hemos utilizado dos métodos, en el primer caso utilizamos los cuatro bits del registro de corrimiento, en el segundo caso sólo utilizamos a tres de ellos, dejando libre al bit menos significativo. Es posible la implementación de esta alternativa porque el número de estados del autómata lo permite, su uso nos limita corrimientos en un sólo sentido, pero a cambio obtenemos una simplificación en el proceso al reducir el número de mapas necesarios y su complejidad.



DESCRIPCION DE NEMONICOS

ENTRADAS	SALIDAS
DEP: OPERACION DE DEPOSITO	BLL: ENVIA BILLETE A LA CHAROLA
ERR: EXISTE ERROR EN LA CANTIDAD INTRODUCIDA	EXP: CAJERO EXPULSA TARJETA
FIN: FIN DE SESION	IMP: IMPRIME COMPROBANTE DE OPERACION
REM: CANTIDAD REQUERIDA POR USUARIO IGUAL A EMITIDA POR CAJERO	STC: CAJERO SOLICITA CANTIDAD A RETIRAR POR EL USUARIO
SOB: FUE INSERTADO SOBRE CON CANTIDAD DEPOSITADA	STS: CAJERO SOLICITA CANTIDAD A DEPOSITAR POR USUARIO
TAR: TARJETA INSERTADA	STO: CAJERO SOLICITA TIPO DE OPERACION
OPE: YA SE ELIGIO OPERACION	

ESTADOS

	C	B	A
a	0	0	0
b	0	0	1
c	0	1	0
d	0	1	1
e	1	0	0
f	1	0	1
g	1	1	1

A) ASIGNACION DE ESTADOS POR CORRIMIENTO

Fig. 4.12. SOLUCION PARA C.A. POR REGISTROS DE CORRIMIENTO (1a.Forma)(1a. Parte)

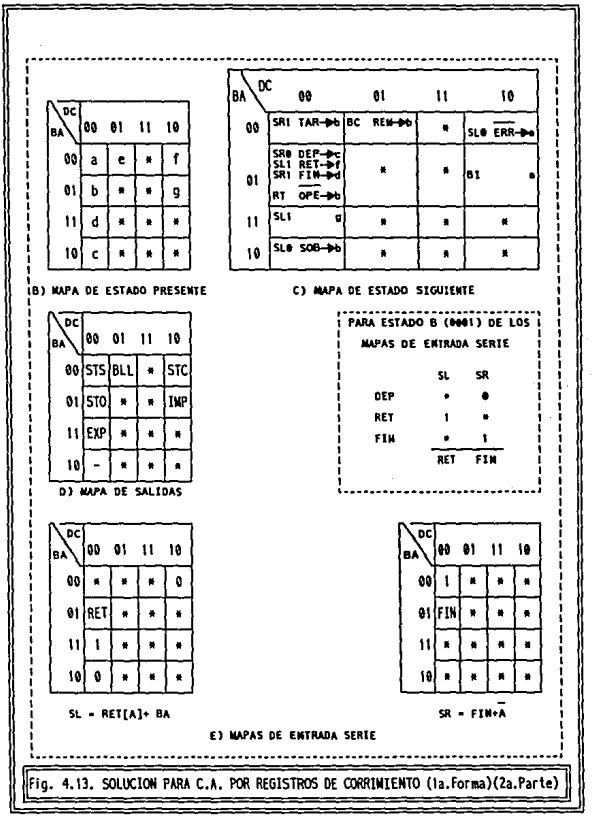


Fig. 4.13. SOLUCION PARA C.A. POR REGISTROS DE CORRIMIENTO (1a.Forma)(2a.Parte)

BA \ DC	00	01	11	10
00	TAR	REM	*	0
01	DEP+FIN	*	*	1
11	0	*	*	*
10	0	*	*	*

$S_0 = TAR\overline{[DCBA]} + REM[C] + [DEP+FIN]\overline{[BA]} + DA$

BA \ DC	00	01	11	10
00	0	REM	*	$\overline{ERR}$
01	RET	*	*	1
11	1	*	*	*
10	SOB	*	*	*

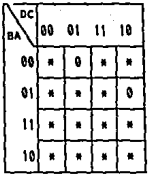
$S_1 = REM[C] + RET[A] + \overline{ERR}[D] + SOB[B] + BA$

F) MAPAS DE CONTROL

ESTADO a (0000)	ESTADO b (0001)	ESTADO c (0010)	ESTADO d (0011)
SR CONDICIONADO $S_0 = TAR$ $S_1 = 0$	S_0 S_1 DEP 1 0 RET 0 1 FIN 1 0 $\overline{OPE}$ 0 0 $\overline{DEP+FIN\ RET}$	SL CONDICIONADO $S_0 = 0$ $S_1 = SOB$	SL INCONDICIONAL $S_0 = 0$ $S_1 = 1$
ESTADO e (0100)	ESTADO f (1000)	ESTADO g (1001)	ESTADOS *
BRINCO CONDIC. $S_0 = REM$ $S_1 = REM$	SL CONDICIONADO $S_0 = 0$ $S_1 = ERR$	BRINCO INCONDIC. $S_0 = 1$ $S_1 = 1$	NO IMPORTA LLENAR CON ASTERISCOS

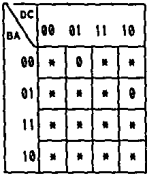
CUADROS PARA OBTENCION DE MAPAS DE CONTROL

Fig. 4.14. SOLUCION PARA C.A. POR REGISTROS DE CORRIMIENTO (1a.Forma)(3a.Parte)



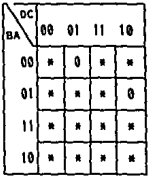
BA \ DC	00	01	11	10
00	*	0	*	*
01	*	*	*	0
11	*	*	*	*
10	*	*	*	*

D = GND



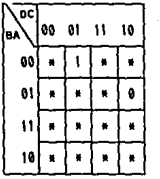
BA \ DC	00	01	11	10
00	*	0	*	*
01	*	*	*	0
11	*	*	*	*
10	*	*	*	*

C = GND



BA \ DC	00	01	11	10
00	*	0	*	*
01	*	*	*	0
11	*	*	*	*
10	*	*	*	*

B = GND



BA \ DC	00	01	11	10
00	*	1	*	*
01	*	*	*	0
11	*	*	*	*
10	*	*	*	*

A = $\bar{A}$

G) MAPAS DE CARGA EN PARALELO

Fig. 4.15. SOLUCION PARA C.A. POR REGISTROS DE CORRIMIENTO (1a.Forma)(4a.Parte)

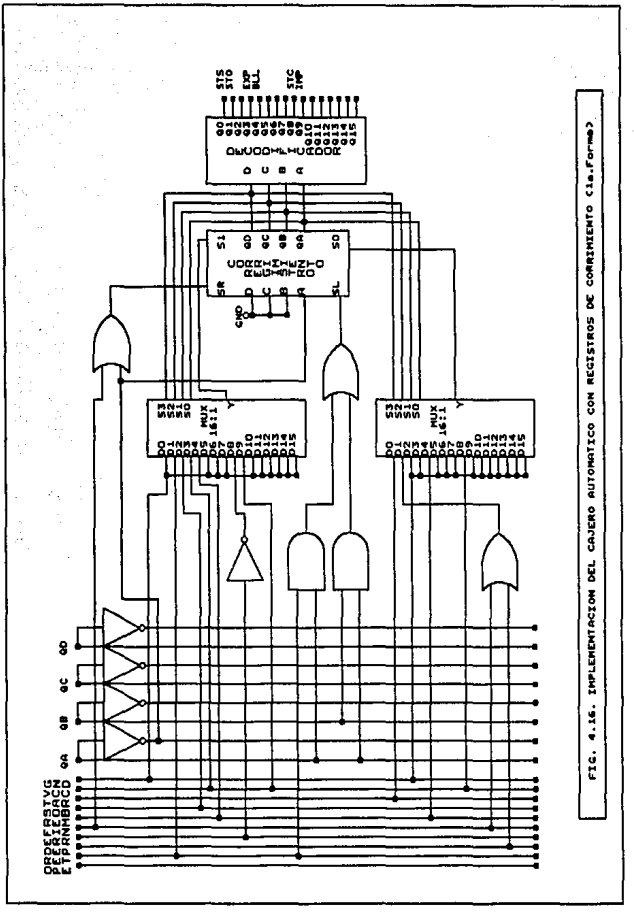
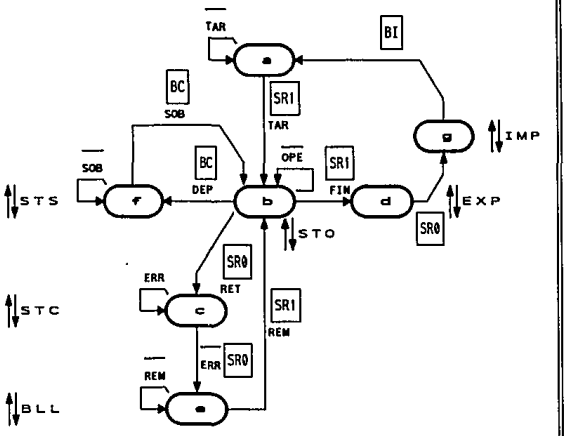


FIG. 4.16. IMPLEMENTACION DEL CAJERO AUTOMATICO CON REGISTROS DE CORRIMIENTO (1a. Forma)



DESCRIPCION DE MEMONICOS

ENTRADAS	SALIDAS
DEP: OPERACION DE DEPOSITO	BLL: ENVIA BILLETE A LA CHAROLA
ERR: EXISTE ERROR EN LA CANTIDAD INTRODUCIDA	EXP: CAJERO EXPULSA TARJETA
FIN: FIN DE SESION	IMP: IMPRIME COMPROBANTE DE OPERACION
REM: CANTIDAD REQUERIDA POR USUARIO IGUAL A EMITIDA POR CAJERO	STC: CAJERO SOLICITA CANTIDAD A RETIRAR POR EL USUARIO
SOB: FUE INSERTADO SOBRE CON CANTIDAD DEPOSITADA	STS: CAJERO SOLICITA CANTIDAD A DEPOSITAR POR USUARIO
TAR: TARJETA INSERTADA	STO: CAJERO SOLICITA TIPO DE OPERACION
OPE: YA SE ELIGIO OPERACION	

ESTADOS	C B A
a	0 0 0
b	0 0 1
c	0 1 0
d	0 1 1
e	1 0 0
f	1 0 1
g	1 1 0

A) ASIGNACION DE ESTADOS POR CORRIMIENTO

Fig. 4.17. SOLUCION PARA C.A. POR REGISTROS DE CORRIMIENTO (2a.Forma)(1a. Parte)

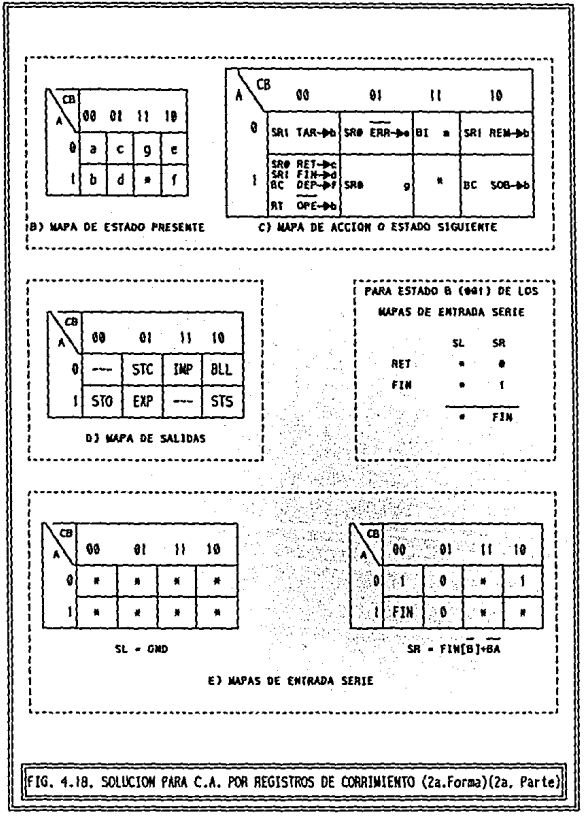


FIG. 4.18. SOLUCION PARA C.A. POR REGISTROS DE CORRIMIENTO (2a.Forma)(2a. Parte)

A \ CB	00	01	11	10
0	TAR	$\overline{ERR}$	1	REM
1	OPE	1	*	SOB

$S_0 = TAR[\overline{CBA}]+\overline{ERR}[B]+REM[\overline{CA}]+OPE[\overline{CA}]+SOB[CA]+CB+BA$

A \ CB	00	01	11	10
0	0	0	1	0
1	DEP	0	*	SOB

$S_1 = DEP[\overline{CBA}]+SOB[CA]+CB$

F) MAPAS DE CONTROL

PARA ESTADO B (001) DE LOS MAPAS DE CONTROL

	S_0	S_1
RET	1	0
FIN	1	0
DEP	1	1
$\overline{OPE}$	0	0
	OPE	DEP

A \ CB	00	01	11	10
0	*	*	0	*
1	1	*	*	0

MAPA C = $\overline{C}$

A \ CB	00	01	11	10
0	*	*	0	*
1	0	*	*	0

MAPA B = GND

A \ CB	00	01	11	10
0	*	*	0	*
1	1	*	*	1

MAPA A = A

G) MAPAS DE CARGA EN PARALELO

FIG. 4.19. SOLUCION PARA C.A. POR REGISTROS DE CORRIMIENTO (2a.Forma)(3a.Parte)

FIG. 4.20. IMPLEMENTACION DEL CAJERO AUTOMATICO CON REGISTROS DE CORRIMIENTO (2a.Forma)

4.6. CONTROLADOR MICROPROGRAMADO BÁSICO

En la sección 1.2 se establecieron tanto las características como la arquitectura generales para un controlador programable. Ahora en esta sección describiremos un tipo de metodología que se ajuste a los requerimientos generales.

A partir de la fig 1.2.a. y siguiendo el esquema análogo al presentado a través de diversas metodologías, la configuración para nuestra estructura se muestra en la fig. 4.21.

4.6.1. ELEMENTOS

- MEMORIA RAM: Almacenará el microprograma.
- CONTADOR: Con capacidad de carga en paralelo y entradas de control para secuencia y carga, actuará como contador de programa, indicando la localidad de memoria que será procesada.
- MULTIPLEXORES: Los cuales controlarán el tipo de instrucción a ser ejecutada (brinco o cuenta) mediante la habilitación de las entradas de control del contador.

4.6.2. RESTRICCIONES

- Ningún estado del autómata deberá poseer más de dos estados siguientes.
- Si un estado posee dos estados siguientes, a uno se llegará por medio de una acción de cuenta y al otro por una acción de brinco. Lo cual significa que no se permite la especificación de dos cuentas o de dos brincos simultáneamente.

4.6.3. INSTRUCCIONES

Con la arquitectura presentada es posible el uso de las siguientes operaciones que nos permitirán elaborar el subsecuente cuadro de instrucciones:

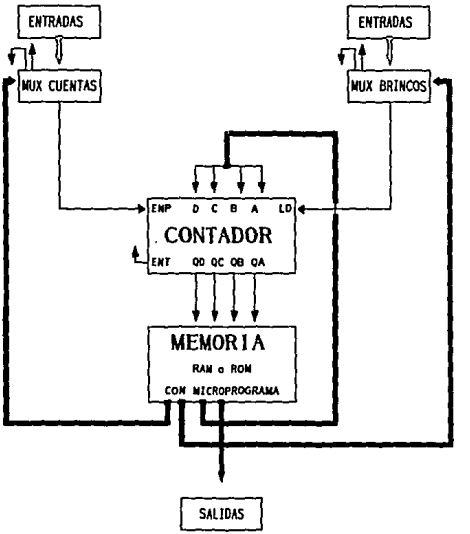


Fig. 4.21. ARQUITECTURA BASICA PARA EL METODO MICROPROGRAMADO

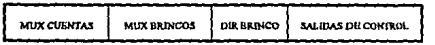
- INCREMENTO CONDICIONAL del contador de programa a la siguiente instrucción. Se procesará la siguiente localidad de acuerdo a una o más condiciones de entrada.
- INCREMENTO INCONDICIONAL del contador de programa.
- DIRECCIONAMIENTO CONDICIONAL del contador de programa a otra localidad de memoria para su procesamiento.
- DIRECCIONAMIENTO INCONDICIONAL del contador de programa a otra localidad de memoria.

INSTRUCCIÓN	DESCRIPCIÓN
CC (VAR)	CUENTA CONDICIONADA a la variable VAR.
CI	CUENTA INCONDICIONAL.
BC (VAR) [DIR]	BRINCO CONDICIONADO a la variable VAR y dirigido hacia la dirección DIR.
BI	BRINCO INCONDICIONAL.
C/B (V1)(V2)[DIR]	CUENTA-BRINCO.Si la variable V1 es verdadera se realiza una cuenta. Si la variable V2 es verdadera se realiza un brinco hacia la dirección DIR.

TABLA 4.7. INSTRUCCIONES PARA EL METODO MICROPROGRAMADO

4.6.4 FORMATO DE CONTROL

Ya que contamos con cinco instrucciones y sus nemónicos correspondientes debemos establecer la forma en que las almacenaremos dentro de la memoria. Nuestro formato de control será el siguiente:



- MUX CUENTAS: Indica si la instrucción a procesar involucra una cuenta, si es incondicional o condicional, y en este último caso de qué variable depende.
- MUX BRINCOS: Indica si la instrucción a procesar involucra un brinco, si es incondicional o condicional, y de ser así de qué variable depende.
- DIR BRINCO: Indica la localidad de memoria a la que será transferido el contador de programa, en caso de que la instrucción a procesar involucre un brinco.
- SALIDAS DE CONTROL: Indica para cada salida del controlador si estará o no activa en el estado presente.

Es necesario recalcar que el formato de control puede tomar diversas formas e incluso arbitrariamente al gusto del diseñador, esto será válido siempre y cuando él mismo sea consistente con su formato durante todo el proceso de desarrollo del controlador.

La metodología presentada a continuación es idónea para la arquitectura y formato de control definidos en esta sección. Sin embargo el lector una vez que haya entendido completamente el proceso deberá si así lo desea ser capaz de implementar cambios tanto en la arquitectura como en el formato de control, cambios que obviamente deben ser considerados en la metodología aquí propuesta.

4.6.5 METODOLOGÍA

a) VERIFICACIÓN DE RESTRICCIONES Y ASIGNACIÓN DE ESTADOS

Antes de cualquier paso, es necesario comprobar que cada estado de nuestro diagrama cumple con las restricciones mencionadas anteriormente. En caso de que no sea así, deberá modificarse el diagrama original en aquellos estados problemáticos.

Si un estado posee más de dos estados siguientes (lo cual ocurre con frecuencia), simplemente ejecute los pasos siguientes:

- Incluimos un nuevo estado al autómata por cada estado siguiente de más que posea el estado presente. El primer estado incluido ocupará la posición en el autómata del estado presente original, los demás serán secuenciales a éste, quedando al final de la secuencia el estado previamente problemático.
- Cada estado incluido poseerá dos estados siguientes dependientes de una misma variable (en forma afirmada y en forma negada).
- Para cada acción extra en el estado presente, tomamos su variable afirmada y la dirigimos al estado que apuntaba originalmente, la variable negada la dirigimos hacia el siguiente estado incluido.

Hecho lo anterior se realiza nuevamente la asignación de estados por secuencia.

b) Asignación de Acciones Inter Estados

Para cada estado se debe asignar una instrucción adecuada que permita el cambio al estado o estados siguientes. Al mismo tiempo se llevará a cabo el llenado de la Tabla de Acción.

c) Tabla de Entradas a Mux

Se debe establecer el número de bits que poseerá cada uno de los campos de nuestro formato de control.

Mux Cuentas: Enumeramos el número de variables diferentes que condicionan una instrucción de cuenta. Si una variable condiciona mas de una cuenta en igual número de estados, sólo la contaremos una vez. A este número que representaremos como X le sumaremos el valor (X+2) correspondiente a dos lineas de mux reservadas, para inhabilitación de cuenta y para cuenta incondicional respectivamente.

El número de bits necesarios para este campo estará dado por n en la expresión $2^n = X+2$, es decir es el equivalente al número de líneas de selección necesarias para direccionar X+2 entradas hacia una salida.

Mux Brincos: Contamos el número de variables diferentes que condicionan una instrucción de brinco. Si una variable condiciona más de un brinco en igual número de estados, sólo la contaremos una vez. A este número que representaremos como Y le sumaremos el valor (Y+2) correspondiente a dos líneas de mux reservadas, para inhabilitación de brinco y para brinco incondicional respectivamente.

El número de bits necesarios para este campo estará dado por n en la expresión $2^n = Y+2$, es decir es el equivalente al número de líneas de selección necesarias para direccionar Y+2 entradas hacia una salida.

Dir Brinco: El número de bits será equivalente al número de bits con que representamos a un estado en nuestro controlador.

Salidas: Habrá un bit para cada salida generada por el controlador.

d) Tabla de Microcódigo

La Tabla de Microcódigo es un arreglo de todos los bits de nuestra memoria separados en campos y mostrando su contenido (1 o 0) en cada uno de los estados de nuestro controlador. La tabla contiene cinco columnas fijas (Estado, Mux Cuentas, Mux Brincos, Dir Brinco y Salidas) más una columna por cada memoria a utilizar (HexN a Hex1). El número de renglones de la tabla será igual al número de estados de nuestro controlador.

ESTADO	MUX CUENTAS	MUX BRINCOS	DIR BRINCO	SALIDAS	HEXN	. . .	HEX1
a . b ...							

TABLA 4.8. ESTRUCTURA DE LA TABLA DE MICROCODIGO PARA EL METODO MICROPROGRAMADO

A continuación describiremos de manera general el llenado de la tabla y más adelante se describe mediante un ejemplo.

La primera columna: Estado contiene el nemónico correspondiente al estado del controlador.

Como sabemos a cada estado corresponde una acción. El llenado de la tabla de microcódigo depende de la tabla de acción y del diagrama implementado. Para cada estado las columnas Mux Cuentas, Mux Brincos y Dir Brinco se llenan de acuerdo a la instrucción.

1ER. CASO: Si la instrucción es C.C. (Cuenta Condicional)

MUX CUENTAS: Colocar en binario el número de entrada al mux de cuentas asociado a la variable condicionante.

MUX BRINCO: Llenar con ceros (Inhabilitación de brinco)

DIR BRINCO: Puede colocarse cualquier código puesto que no habrá brinco. Para nuestro método lo llenaremos con ceros.

2do. CASO: Si la instrucción es C.I. (Cuenta Incondicional)

MUX CUENTAS: Colocar un uno en binario (habilitación de cuenta incondicional.

MUX BRINCO: Llenar con ceros (Inhabilitación de brinco)

DIR BRINCO: Llenar con ceros.

3ER. CASO: Si la instrucción es B.C. (Brinco Condicional)

MUX CUENTAS: Llenar con ceros (Inhabilitación de brinco)

MUX BRINCO: Colocar en binario el número de entradas al mux de brincos asociado a la variable condicionante.

DIR BRINCO: Colocar en binario la dirección a la cual se transferirá el control del programa.

4TO. CASO: Si la instrucción es B.I (Brinco Incondicional)

MUX CUENTAS: Llenar con ceros (Inhabilitación de brinco)

MUX BRINCO: Colocar un uno en binario invertido.

DIR BRINCO: Colocar en binario la dirección a la cual se transferirá el control del programa.

5TO. CASO: Si la instrucción es C/B (Cuenta Brinco)

MUX CUENTAS: Colocar en binario el número de entradas al mux de cuentas asociado a la variable condicionante de cuentas.

MUX BRINCO: Colocar en binario el número de entradas al mux de brincos asociado a la variable condicionante.

DIR BRINCO: Colocar en binario la dirección a la cual se transferirá el control del programa.

El campo SALIDAS contiene un bit para cada salida, todas las salidas que sean generadas en el estado presente serán asignadas con un 1, las restantes con un 0.

Las columnas restantes HEX1 A HEXN dependerán tanto en número de ellas como en su contenido de las columnas anteriores. Cada columna representa una memoria, el número de memorias requeridas será igual a:

Num_Memorias = INT (Y/X)+1

donde X: Longitud de palabra de la memoria a utilizar

Y: Número de bits del formato de control

INT(Y/X): Parte entera del cociente de Y/X.

- Agrupamos del bit menos significativo al más significativo en secuencias de X bits (longitud de palabra de la memoria), los campos de nuestro formato de control. Cada grupo corresponderá a cada memoria.
- Convertimos a código hexadecimal el contenido de cada grupo binario.
- Estos códigos hexadecimales son los que utilizaremos finalmente, los debemos obtener porque al grabar una memoria se requiere que el contenido de sus localidades sea expresado en forma hexadecimal.
- Una vez grabadas las memorias simplemente se insertan en el alambrado del circuito y se habrá completado el controlador.

e) Dibujo de Arquitectura

Nuestra arquitectura siempre será similar a la presentada previamente en la figura 4.21.

Los parámetros que varían son:

- Número de bits que parten de la memoria hacia el mux de cuentas.
- Número de bits que parten de la memoria hacia el mux de brincos.
- Número de bits que parten de la memoria hacia el contador de programa.
- Número de salidas.

Como puede observarse todos estos parámetros son los que se calculan en el paso anterior, siendo equivalentes a la longitud en bits de los campos que integran al formato de control.

Ahora se realizan los pasos siguientes:

- Fijar entrada cero (de los mux de cuentas y brincos) a bajo voltaje, lo que representará inhabilitación de cuenta y brinco respectivamente.
- Fijar uno de los mux de cuentas y brincos a alto voltaje, lo que representará habilitación de cuenta y brinco incondicionales respectivamente.
- Cada variable de cuenta será asociada al mux de cuentas y posteriormente será referida bajo el código binario correspondiente a esa entrada.
- Cada variable de brinco será asociada al mux de brinco y posteriormente será referida bajo el código binario correspondiente a esa entrada.

4.6.6. Solución al Cajero Automático

El punto relevante en cuanto al uso de la metodología microprogramada en nuestro caso es el replanteamiento del autómata original.

Verificando el diagrama de la fig.4.1. se puede apreciar que el estado *b* es catalogable como problemático por poseer cuatro estados siguientes, la metodología nos indica en este caso incluir dos estado más al autómata, el diagrama modificado se muestra en la figura 4.22. Cada variable sobrante es separada en un estado. Ahora el estado problemático posee sólo dos acciones, siendo completamente equivalentes ambos autómatas, con la ventaja que ya es posible aplicar la metodología a nuestro nuevo autómata. Observe que el nuevo diagrama posee nueve estados y que algunos de ellos han sido reasignados por secuencia.

A partir del autómata hemos propuesto dos métodos ligeramente diferentes pero con igual resultado funcional. Primeramente se resuelve por el método *Normal* utilizando introducción binaria directa, y posteriormente se resuelve por el método *Invertido* utilizando introducción binaria invertida.

La lógica en la solución por ambos métodos se apega a los pasos previamente anotados, se observará que los pasos son idénticos a excepción del llenado de la tabla de microcódigo y la implementación. La diferencia básica radica en la asignación de bits en el formato de control. En el método microprogramado normal la asignación de bits así como su agrupación es de derecha a izquierda y la introducción de cualquier valor a la tabla es en forma directa. En el método invertido la asignación de bits y su agrupación es de izquierda a derecha, sin olvidar que todo valor introducido en la tabla será en binario invertido.

Nótese también que las columnas correspondientes al campo Salidas no varían en absoluto en ambos métodos.

Para ejemplificar el llenado de la tabla utilizaremos el estado *c*. La tabla de acciones nos señala una acción cuenta-brinco, la cuenta depende de la variable FIN negada que corresponde a la terminal 4 del mux de cuentas en la tabla de entradas a mux. El valor 4 se introduce en la tabla normal como 100 y en la invertida como 001. El brinco está condicionado a FIN afirmada que corresponde al valor 2 del mux de brincos en la tabla de entradas a mux. El valor 3 se introduce en la tabla normal como 010 y en la invertida como 010, ambos en el campo mux brincos. La dirección del brinco nos señala la localidad 0111, así es introducida en la tabla normal y como 1110 en la tabla invertida. Como

este estado no posee ninguna salida todos los bits de salida se llenan con ceros. Agrupando los bits de cada estado en secuencias de ocho, en forma directa tenemos para el estado *c*:

1000 1001 equivalente a 8 9

1100 0000 equivalente a C 0

y en forma invertida:

0010 1011 equivalente a D 4

1000 0000 equivalente a 0 1

Cada estado se introduce análogamente al explicado.

La agrupación de bits se realiza en series de ocho debido a que este es el valor correspondiente a la longitud de palabra de la memoria a utilizar.

A continuación se desarrolla la solución completa por los métodos Microprogramado Normal y Microprogramado Invertido.

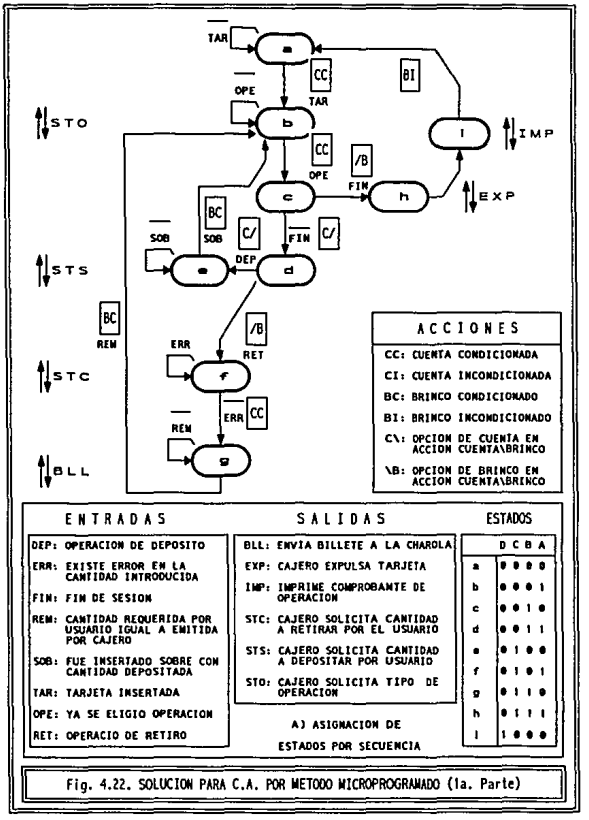


Fig. 4.22. SOLUCION PARA C.A. POR METODO MICROPROGRAMADO (1a. Parte)

ESTADO	ACCION
a	CC (TAR)
b	CC (OPE) <STO>
c	C/B (FIN)(FIN) [0111]
d	C/B (DEP)(RET) [0101]
e	BC (SOB) [0001]<STS>
f	CC (ERR) <STC>
g	BC (REM) [0001]<BLL>
h	CI <EXP>
i	BI [0000]<IMP>

B) TABLA DE ACCIONES

TERMINAL	MUX CUENTAS	MUX BRINCOS
0	NO CUENTA	NO BRINCO
1	INCONDICIONAL	INCONDICIONAL
2	TAR	FIN
3	OPE	RET
4	FIN	SOB
5	DEP	REM
6	ERR	
SE REQUIEREN 3 LINEAS DE SELECCION PARA CADA MUX		

C) TABLA DE ENTRADAS A MUX

MSB LSB

MUX CUENTAS	MUX BRINCOS	DIR BRINCO	SALIDAS

FORMATO DE CONTROL

ESTADO	MUX CUENTAS D_{15} D_{14} D_{13}	MUX BRINCOS D_{12} D_{11} D_{10}	DIR BRINCO D_9 D_8 D_7 D_6	SALIDAS D_5 D_4 D_3 D_2 D_1 D_0	HEX2	HEX1
a	0 1 0	0 0 0	0 0 0 0	0 0 0 0 0 0	40	00
b	0 1 1	0 0 0	0 0 0 0	1 0 0 0 0 0	60	20
c	1 0 0	0 1 0	0 1 1 1	0 0 0 0 0 0	89	C0
d	1 0 1	0 1 1	0 1 0 1	0 0 0 0 0 0	AD	40
e	0 0 0	1 0 0	0 0 0 1	0 1 0 0 0 0	10	50
f	1 1 0	0 0 0	0 0 0 0	0 0 1 0 0 0	C0	08
g	0 0 0	1 0 1	0 0 0 1	0 0 0 1 0 0	14	44
h	0 0 1	0 0 0	0 0 0 0	0 0 0 0 1 0	20	02
i	0 0 0	0 0 1	0 0 0 0	0 0 0 0 0 1	04	01

HEX2 HEX1

D) TABLA DE MICROCODIGO EN MODO NORMAL

Fig. 4.23. SOLUCION A CAJERO AUTOMATICO POR MICROPROGRAMADO NORMAL (2a. Parte)

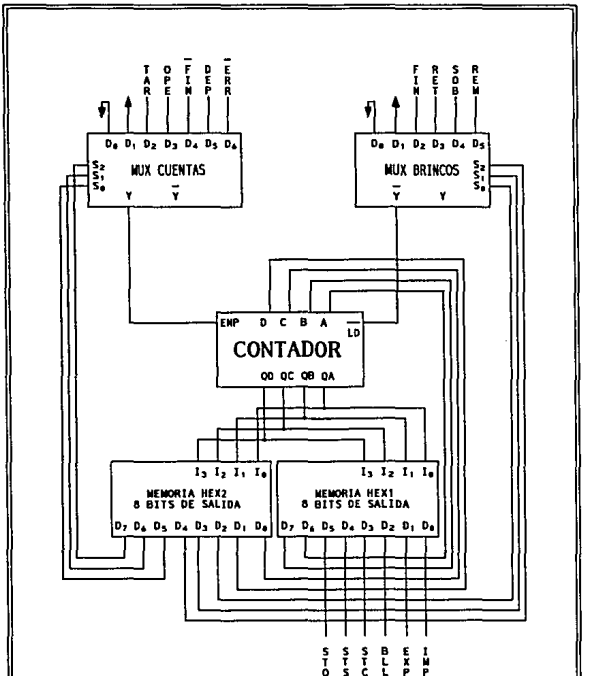


Fig. 4.24. IMPLEMENTACION DEL CAJERO AUTOMATICO POR MICROPROGRAMADO NORMAL

ESTADO	ACCION
a	CC (TAR)
b	CC (OPE) <STO>
c	C/B (FIN)(FIN) [0111]
d	C/B (DEP)(RET) [0101]
e	BC (SOB) [0001]<STS>
f	CC (ERR) <STC>
g	BC (REM) [0001]<BLL>
h	CI <EXP>
i	BI [0000]<IMP>

B) TABLA DE ACCIONES

TERMINAL	MUX CUENTAS	MUX BRINCOS
0	NO CUENTA	NO BRINCO
1	INCONDICIONAL	INCONDICIONAL
2	TAR	FIN
3	OPE	RET
4	FIN	SOB
5	DEP	REM
6	ERR	

SE REQUIEREN 3 LINEAS DE SELECCION PARA CADA MUX

C) TABLA DE ENTRADAS A MUX

LSB MSB

MUX CUENTAS	MUX BRINCOS	DIR BRINCO	SALIDAS

FORMATO DE CONTROL

ESTADO	MUX CUENTAS D_0 D_1 D_2	MUX BRINCOS D_3 D_4 D_5	DIR BRINCO D_6 D_7 D_8 D_9	SALIDAS $D_{10}D_{11}D_{12}D_{13}D_{14}D_{15}$	HEX1	HEX2
a	0 1 0	0 0 0	0 0 0 0	0 0 0 0 0 0	02	00
b	1 1 0	0 0 0	0 0 0 0	1 0 0 0 0 0	03	04
c	0 0 1	0 1 0	1 1 1 0	0 0 0 0 0 0	D4	01
d	1 0 1	1 1 0	1 0 1 0	0 0 0 0 0 0	5D	01
e	0 0 0	0 0 1	1 0 0 0	0 1 0 0 0 0	60	08
f	0 1 1	0 0 0	0 0 0 0	0 0 1 0 0 0	06	10
g	0 0 0	1 0 1	1 0 0 0	0 0 0 1 0 0	68	20
h	1 0 0	0 0 0	0 0 0 0	0 0 0 0 1 0	01	40
i	0 0 0	1 0 0	0 0 0 0	0 0 0 0 0 1	08	80

HEX1 HEX2

D) TABLA DE MICROCODIGO EN MODO NORMAL

Fig. 4.25. SOLUCION A CAJERO AUTOMATICO POR MICROPROGRAMADO INVERTIDO (2a.Parte)

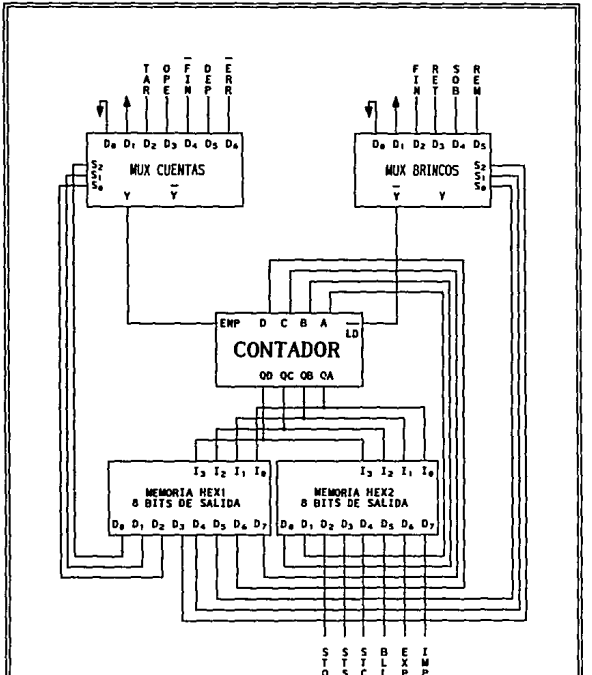


Fig. 4.26. IMPLEMENTACION DEL CAJERO AUTOMATICO POR MICROPROGRAMADO INVERTIDO

4.7. MICROCONTROLADORES MYCA-I

4.7.1. INSTRUCCIONES

Como habíamos mencionado, antes de aplicar esta metodología se requiere desarrollar un módulo, lo cual se hace en base a las características que deseamos nos proporcione. Por tal motivo, como primer paso se debe especificar el conjunto de instrucciones que incluirá el módulo y posteriormente implementarlas.

OPCODE C_2 C_1 C_0	INSTRUCCIÓN	DESCRIPCIÓN
0 0 0	CC (VAR)	CUENTA CONDICIONADA a la variable VAR
0 0 1	CI	CUENTA INCONDICIONAL
0 1 0	BC (VAR) [DIR]	BRINCO CONDICIONADO a la variable VAR y dirigido hacia la dirección DIR
0 1 1	BI	BRINCO INCONDICIONAL
1 0 0	C/BC (VAR)[DIR]	CUENTA-BRINCO.Si la variable VAR es negada se realiza una cuenta. Si es afirmada se realiza un brinco hacia la dirección DIR.

TABLA 4.9. INSTRUCCIONES SOPORTADAS POR EL MODULO MYCA-I

4.7.2. ARQUITECTURA DEL MODULO MYCA-I

El conjunto de instrucciones será implementado en torno a un contador, con entradas de control (ENP: Habilitación de Cuenta y LD: Habilitación de Carga). La combinación de estas dos entradas nos permitirá realizar acciones básicas de Retención, Cuenta y Brinco. La instrucción requerida por el usuario será direccionada por medio de cuatro bits,

ACCION	EnP	LD
RETENCION	0	0
CUENTA	1	0
BRINCO	*	1

MAPA DE ACCION

PARA NUESTRO CONTADOR

OPCODE CO_2 CO_1 CO_0	VAR	EnP	LD
0 0 0	0	0	0
0 0 0	1	1	0
0 0 1	0	1	0
0 0 1	1	1	0
0 1 0	0	0	0
0 1 0	1	*	1
0 1 1	0	*	1
0 1 1	1	*	1
1 0 0	0	1	0
1 0 0	1	*	1

TABLA DE RELACION

PARA EL MODULO MYCA-I

CO_0 \ CO_2CO_1	00	01	11	10
0	VAR	0	*	1
1	1	*	*	*

MAPA EnP

CO_0 \ CO_2CO_1	00	01	11	10
0	0	VAR	1	VAR
1	0	1	1	1

MAPA LD

FIG. 4.27. PROCESO PARA OBTENCION DEL MODULO MYCA-I

correspondientes al código de operación (OPCODE) y al estado de la variable condicionante VAR.

Si tabulamos las salidas deseadas para cada secuencia posible de los bits de entrada obtendremos la Tabla de Relación mostrada en la fig. 4.27. Su obtención se realiza a partir de la Tabla de Instrucciones requerida por Myca-1 y del Mapa de Acción del Contador.

- Para OPCODE (000) Y VAR(0): La instrucción para OPCODE es Retención, ya que la variable está negada. Esto significa EnP=0 y LD=0.
- Para OPCODE (000) Y VAR(1): La instrucción para OPCODE es Cuenta, ya que la variable está afirmada. Esto significa EnP=1 y LD=0.
- Para OPCODE (001) : La instrucción para OPCODE es Cuenta Incondicional, no importa el estado de la variable VAR(0 o 1) Esto significa EnP=1 y LD=0.
- Para OPCODE (010) Y VAR(0): La instrucción para OPCODE es Retención, ya que la variable está negada. Esto significa EnP=0 y LD=0.
- Para OPCODE (010) Y VAR(1): La instrucción para OPCODE es Brinco, ya que la variable está afirmada. Esto significa EnP=* y LD=1.
- Para OPCODE (011) : La instrucción para OPCODE es Brinco Incondicional, no importa el estado de la variable VAR(0 o 1). Esto significa EnP=* y LD=1.
- Para OPCODE (100) Y VAR(0): La instrucción para OPCODE es Cuenta, ya que la variable está negada. Esto significa EnP=1 y LD=0.
- Para OPCODE (100) Y VAR(1): La instrucción para OPCODE es Brinco, ya que la variable está afirmada. Esto significa EnP=* y LD=1.

Cada una de las dos salidas (EnP y VAR) de la Tabla de Relación puede ser ubicada en un Mapa de Karnaugh.

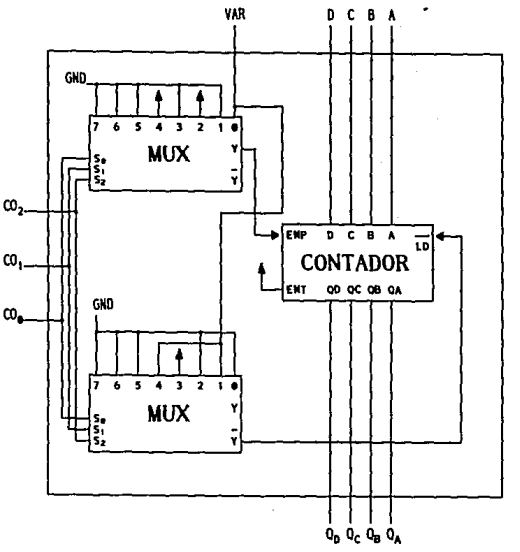


Fig. 4.28. ESTRUCTURA INTERNA DEL MODULO MYCA-I

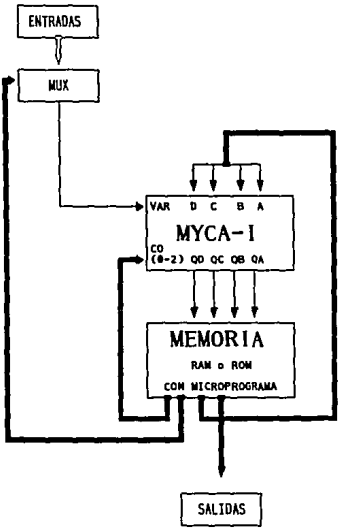


Fig. 4.29. ARQUITECTURA GENERAL PARA EL METODO MYCA-I

La implementación del módulo es ya bastante simple, sólo basta direccionar los valores requeridos en cada estado -ubicados en los mapas- hacia las entradas de control del contador, y lo hacemos sencillamente con un par de multiplexores.

El diagrama interno que hemos obtenido para nuestro módulo MYCA I se localiza en la fig.4.28. Ahora que lo hemos diseñado lo podremos utilizar tantas veces como sea requerido, por lo pronto veremos su aplicación.

4.7.3. Elementos

Para la implementación de este microcontrolador se requiere además del módulo MYCA I -desarrollado en la sección previa- los elementos siguientes:

- Memoria RAM: Almacenará el microprograma.
- Multiplexor: El cual conducirá hacia la entrada de control VAR del módulo MYCA-II la variable requerida por la instrucción a procesarse.

4.7.4. Restricciones

- Ningún estado del autómata deberá poseer más de dos estados siguientes.
- No se permite la especificación de dos cuentas o de dos brincos simultáneamente a partir de un estado presente.

4.7.5. Formato de Control

El formato de control para esta metodología será el siguiente:

OPCODE	MULTIPLEXOR	DIR BRINCO	SALIDAS DE CONTROL

- Opcode: Indica el código de operación asociado a la instrucción que será ejecutada.

- Multiplexor: Indica la terminal del multiplexor asociada a la variable que condiciona la instrucción que requiere ejecutarse.
- Dir Brinco: Indica la localidad de memoria a la que será transferido el contador de programa, en caso de que la instrucción a procesar involucre un brinco.
- Salidas de Control: Indica para cada salida del controlador si estará o no activa en el estado presente.

4.7.6. Metodología

a) Verificación de Restricciones y Asignación de Estados

Verificar que cada estado del autómata cumple con las restricciones mencionadas anteriormente y de ser necesario modificarlo en los estados requeridos. Realizar la asignación de estados por secuencia.

b) Asignación de Acciones Inter-Estados

Para cada estado se debe asignar una instrucción adecuada que permita el cambio al estado o estados siguientes. Al mismo tiempo se llevará a cabo el llenado de la Tabla de Acción.

c) Tabla de Entradas a Mux

Establecer el número de bits que poseerá cada uno de los campos de nuestro formato de control.

Multiplexor: Enumerar el número de variables diferentes que condicionan a una instrucción a partir de la tabla de acciones y asociar una terminal diferente del multiplexor a cada una de ellas. Si una variable condiciona a más de una instrucción, sólo se cuenta una vez. El número de bits necesarios para este campo estará dado por n en la expresión $2^n = X$, es decir es el equivalente al número de líneas de selección necesarias para direccionar X entradas hacia una salida.

Dir Brinco: El número de bits será equivalente al número de bits con que representamos a un estado en nuestro controlador.

Salidas: Habrá un bit para cada salida generada por el controlador.

d) Tabla de Microcódigo

La Tabla de Microcódigo es un arreglo de todos los bits de nuestra memoria separados en campos y mostrando su contenido (1 o 0) en cada uno de los estados de nuestro controlador. La tabla contiene cinco columnas fijas (Estado, Opcode, Multiplexor, Dir Brinco y Salidas) más una columna por cada memoria a utilizar (HexN a Hex1). El número de renglones de la tabla será igual al número de estados de nuestro controlador.

ESTADO	OPCODE	MULTIPLEXOR	DIR BRINCO	SALIDAS	HEXN	.. .	HEX1
a							
b							
...							

TABLA 4.10. ESTRUCTURA DE LA TABLA DE MICROCODIGO PARA EL METODO MYCA-I

La primera columna (Estado) contiene el nemónico correspondiente al **estado del controlador.**

Como sabemos a cada estado corresponde una acción. El llenado de la Tabla de Microcódigo depende de la Tabla de Acción y del diagrama implementado. Para cada estado las columnas Opcode, Multiplexor y Dir Brinco **se llenan de acuerdo a la instrucción.**

1er. caso: **Si la instrucción es CC (Cuenta Condicional)**

Opcode: **Código binario 0.**

Multiplexor: **Colocar en binario el número de entrada al mux asociado a la variable condicionante.**

Dir Brinco: **Puede colocarse cualquier código puesto que no habrá brinco. Para nuestro método lo llenaremos con ceros.**

2do. caso: Si la instrucción es CI (Cuenta Incondicional)

Opcode: Código binario 1.

Multiplexor: Llenar con ceros.

Dir Brinco: Llenar con ceros.

3er. caso: Si la instrucción es BC (Brinco Condicional)

Opcode: Código binario 10.

Multiplexor: Colocar en binario el número de entrada al mux asociado a la variable condicionante.

Dir Brinco: Colocar en binario la dirección a la cual se transferirá el control del programa.

4to. caso: Si la instrucción es BI (Brinco Incondicional)

Opcode: Código binario 11.

Multiplexor : Llenar con ceros.

Dir Brinco: Colocar en binario la dirección a la cual se transferirá el control del programa.

5to. caso: Si la instrucción es C/B (Cuenta Brinco)

Opcode: Código binario 100.

Multiplexor: Colocar en binario el número de entrada al mux asociado a la variable condicionante.

Dir Brinco: Colocar en binario la dirección a la cual se transferirá el control del programa.

El campo Salidas contiene un bit para cada salida, todas las salidas que sean generadas en el estado presente serán asignadas con un 1, las restantes con un 0.

Las columnas restantes Hex1 a HexN dependerán tanto en número de ellas como en su contenido de las columnas anteriores. Su número se calcula con las fórmula: Num_Memorias = INT (Y/X)+1 referida anteriormente.

- Agrupar del bit menos significativo al más significativo en secuencias de X bits (longitud de palabra de la memoria), los campos de nuestro formato de control. Cada grupo corresponderá a cada memoria.
- Convertir a código hexadecimal el contenido de cada grupo binario para el grabado posterior de las memorias.
- Insertar las memorias en el alambrado del circuito.

e) Dibujo de Arquitectura

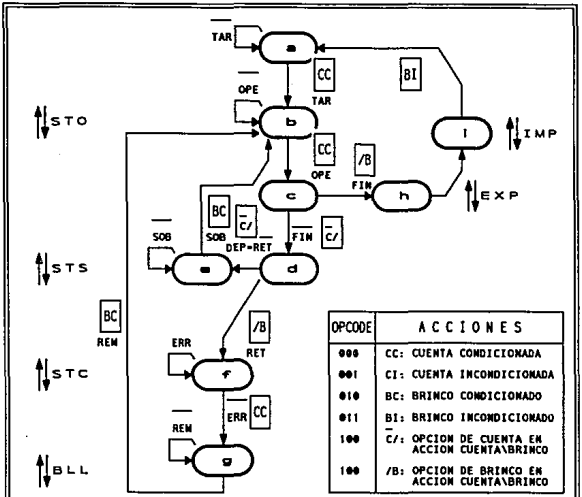
El esquema lógico de nuestra arquitectura siempre será similar en aspecto a la fig. 4.29. Lo único que varía de un microcontrolador a otro es el número de bits que parten de la memoria hacia el multiplexor y hacia los bits de entrada al módulo MYCA I, así como el número de entradas recibidas y salidas generadas.

Al mux de entradas se le asocian las variables condicionantes de acuerdo a la Tabla de Entradas a Mux llenada anteriormente.

4.7.7. Solución al Cajero Automático

El autómata modificado que cumple las restricciones de la metodología es similar al obtenido con el método microprogramado.

La solución presentada a continuación es por Introducción Tabular Normal, pero también es válida la Introducción Invertida.



OPCODE	ACCIONES
000	CC: CUENTA CONDICIONADA
001	CI: CUENTA INCONDICIONADA
010	BC: BRINCO CONDICIONADO
011	BI: BRINCO INCONDICIONADO
100	$\overline{C}$/: OPCION DE CUENTA EN ACCION CUENTA\BRINCO
100	/B: OPCION DE BRINCO EN ACCION CUENTA\BRINCO

ENTRADAS

- DEP: OPERACION DE DEPOSITO
- ERR: EXISTE ERROR EN LA CANTIDAD INTRODUCIDA
- FIN: FIN DE SESION
- REM: CANTIDAD REQUERIDA POR USUARIO IGUAL A EMITIDA POR CAJERO
- SOB: FUE INSERTADO SOBRE CON CANTIDAD DEPOSITADA
- TAR: TARJETA INSERTADA
- OPE: YA SE ELIGIO OPERACION
- RET: OPERACION DE RETIRO

SALIDAS

- BLL: ENVIA BILLETE A LA CHAROLA
- EXP: CAJERO EXPULSA TARJETA
- IMP: IMPRIME COMPROBANTE DE OPERACION
- STC: CAJERO SOLICITA CANTIDAD A RETIRAR POR EL USUARIO
- STS: CAJERO SOLICITA CANTIDAD A DEPOSITAR POR USUARIO
- STO: CAJERO SOLICITA TIPO DE OPERACION

A) ASIGNACION DE ESTADOS POR SECUENCIA

ESTADOS	D	C	B	A
a	0	0	0	0
b	0	0	0	1
c	0	0	1	0
d	0	0	1	1
e	0	1	0	0
f	0	1	0	1
g	0	1	1	0
h	0	1	1	1
I	1	0	0	0

Fig. 4.30. SOLUCION PARA CAJERO AUTOMATICO POR METODO MYCA-I (1a. Parte)

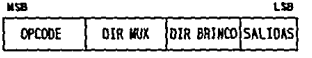
ESTADO	ACCION
a	CC (TAR)
b	CC (OPE) <STO>
c	C/B (FIN) [0111]
d	C/B (RET) [0101]
e	BC (SOB) [0001]<STS>
f	CC (ERR) <STC>
g	BC (REM) [0001]<BLL>
h	CI <EXP>
i	BI [0000]<INP>

B) TABLA DE ACCIONES

TERMINAL	MUX ENTRADAS
0	TAR
1	OPE
2	FIN
3	RET
4	SOB
5	ERR
6	REM

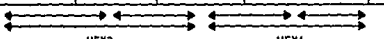
7 TERMINALES REQUIEREN 3 LINEAS DE SELECCION PARA EL MUX

C) TABLA DE ENTRADAS A MUX



FORMATO DE CONTROL

ESTADO	OPCODE D_{15} D_{14} D_{13}	DIR MUX D_{12} D_{11} D_{10}	DIR BRINCO D_9 D_8 D_7 D_6	SALIDAS D_5 D_4 D_3 D_2 D_1 D_0	HEX2	HEX1
a	0 0 0	0 0 0	0 0 0 0	0 0 0 0 0 0	00	00
b	0 0 0	0 0 1	0 0 0 0	1 0 0 0 0 0	04	20
c	1 0 0	0 1 0	0 1 1 1	0 0 0 0 0 0	B9	C0
d	1 0 0	0 1 1	0 1 0 1	0 0 0 0 0 0	BD	40
e	0 1 0	1 0 0	0 0 0 1	0 1 0 0 0 0	50	50
f	0 0 0	1 0 1	0 0 0 0	0 0 1 0 0 0	14	08
g	0 1 0	1 1 0	0 0 0 1	0 0 0 1 0 0	58	44
h	0 0 1	0 0 0	0 0 0 0	0 0 0 0 1 0	20	02
i	0 1 1	0 0 0	0 0 0 0	0 0 0 0 0 1	60	01



D) TABLA DE MICROCODIGO EN MODO NORMAL

Fig. 4.31. SOLUCION A CAJERO AUTOMATICO POR MYCA-I NORMAL (2a. Parte)

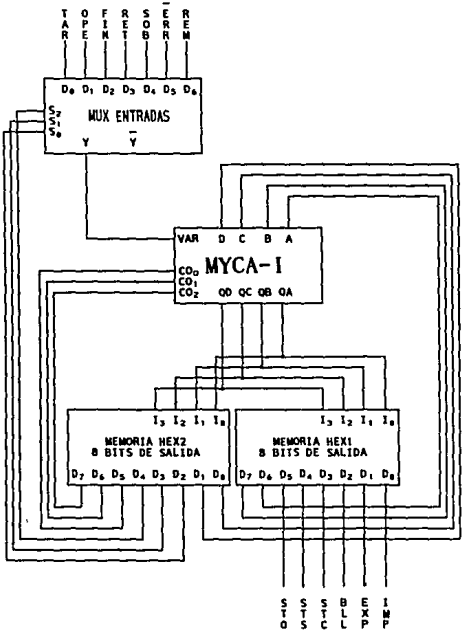


Fig. 4.32. IMPLEMENTACION AL CAJERO AUTOMATICO POR MYCA-I NORMAL

4.8. MICROCONTROLADORES MYCA-II

4.8.1. ARQUITECTURA DEL MODULO MYCA-II

El módulo denominado MYCA II se compone de una serie de elementos MSI controlados por el código grabado en una memoria ROM. A diferencia de la sección anterior en que indicamos de manera completa el desarrollo del módulo MYCA I, ahora sólo mostraremos su estructuración interna y las características que nos proporciona. El diagrama interno se muestra en la fig.4.33, se observa que quien controla a todo el módulo es una memoria ROM, esta memoria solo requiere el uso de 7 bits de salida, uno para cada linea de control, y sólo hemos utilizado 16 localidades (2 por cada instrucción a implementar, para negación y afirmación), esto da una idea del número de instrucciones diferentes que pueden implementarse, por supuesto que el funcionamiento de ellas estará basado en los circuitos periféricos a utilizarse. Para fines didácticos sólo haremos uso de 16 localidades direccionadas por medio de 4 lineas de selección.

Obsérvese que el conjunto de instrucciones planteado incluye el utilizado a través de este capítulo, más la incorporación de dos instrucciones adicionales, ISR y RTS, este par de instrucciones nunca aparece aisladamente en un autómata, el uso de una de ellas requiere el de la otra. Su acción conjunta produce una estructura de control denominada *Subrutina*, la cual se define como un proceso modularizado que puede ejecutarse cíclicamente de manera condicionada. El diagrama MDS para este par de instrucciones se muestra en la fig.4.34. Existe un estado presente *a* con dos estados siguientes *b* y *c*, continuo y discontinuo respectivamente, al primero se llega por medio de una cuenta y al segundo por medio de un brinco, el estado *c* debe ser el inicio de una malla (secuencia cerrada de estados), la malla debe poseer un estado final *d*, que será el único que poseerá dos estado siguientes, un brinco hacia el estado inicial de la malla (*c*) y un brinco hacia el estado continuo al original (*b*). Una restricción a

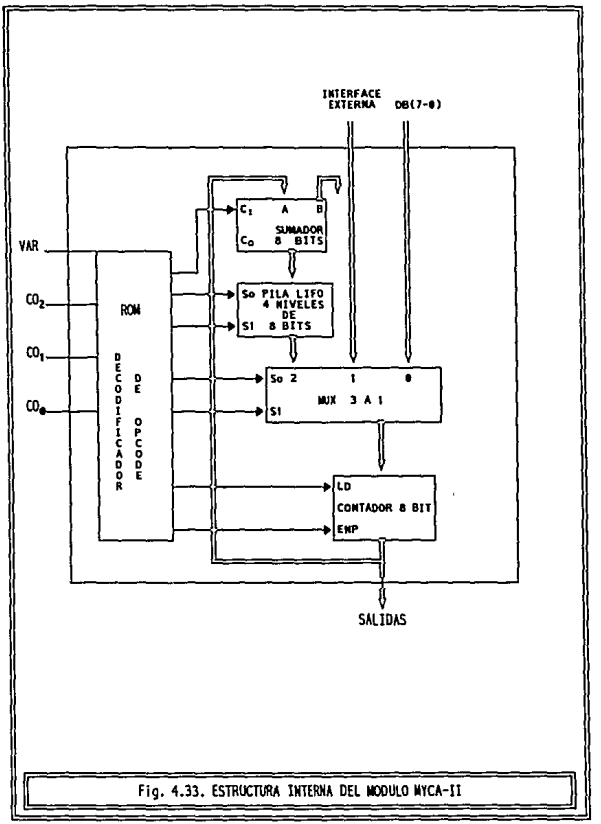


Fig. 4.33. ESTRUCTURA INTERNA DEL MODULO MYCA-II

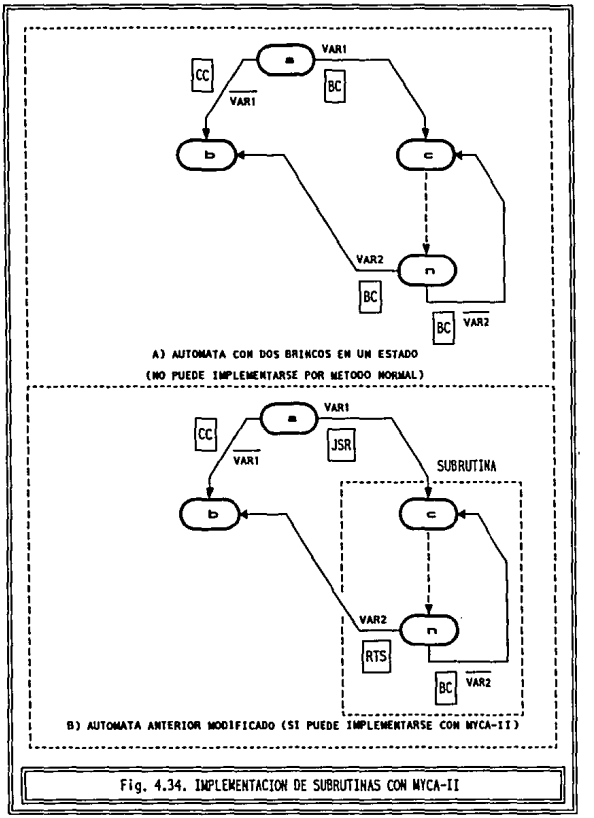


Fig. 4.34. IMPLEMENTACION DE SUBRUTINAS CON MYCA-II

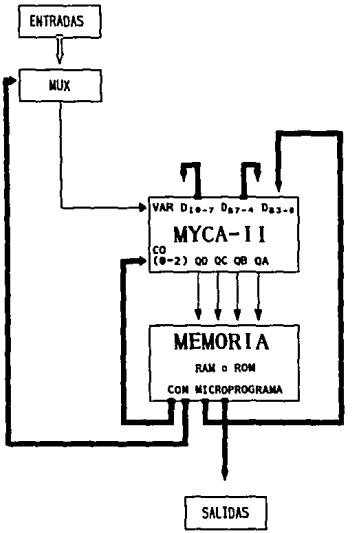


Fig. 4.35. ARQUITECTURA GENERAL PARA EL METODO MYCA-II

la metodología es que un estado no puede poseer dos brincos, de manera que la estructura discutida no podría ser implementada, sin embargo el módulo MYCA II nos proporciona dos instrucciones válidas para ello: la primera JSR, implica un brinco hacia el inicio de la subrutina, la segunda RTS sustituye a uno de los brincos e implica un retorno del final de la subrutina hacia el estado siguiente al que se transfirió el control.

4.8.2. Elementos

Para la implementación de este microcontrolador se requiere además del módulo MYCA II -desarrollado en la sección previa- los siguientes elementos:

- Memoria RAM: Almacenará el microprograma.
- Multiplexor: El cual conducirá hacia la entrada de control VAR del módulo MYCA-II la variable requerida por la instrucción a procesarse.

4.8.3. Restricciones

- Ningún estado del autómata deberá poseer más de dos estados siguientes.
- No se permite la especificación de dos cuentas o de dos brincos simultáneamente a partir de un estado presente.

4.8.4. Instrucciones

Con la arquitectura presentada es posible el uso de un mayor número de instrucciones, cada una se encuentra asociada a un código de operación denominado OPCODE. En el cuadro siguiente resalta la presencia de las instrucciones JSR y RTS que permiten el manejo de subrutinas en la especificación de un autómata.

OPCODE $C0_2$ $C0_1$ $C0_0$	INSTRUCCIÓN	DESCRIPCIÓN
0 0 0	H/CC (VAR)	CUENTA CONDICIONADA a la variable VAR.
0 0 1	CI	CUENTA INCONDICIONAL.
0 1 0	H/BC (VAR) [DIR]	BRINCO CONDICIONADO a la variable VAR y dirigido hacia la dirección DIR.
0 1 1	BI	BRINCO INCONDICIONAL.
1 0 0	C/BC (VAR)[DIR]	CUENTA-BRINCO.Si la variable VAR es negada se realiza una cuenta. Si es afirmada se realiza un brinco hacia la dirección DIR.
1 0 1	C/JSR (VAR) [DIR]	CUENTA-SALTO A SUBRUTINA. Si VAR es negada se realiza una cuenta. Si es afirmada se realiza un salto a la dirección DIR que marca el inicio de una subrutina.
1 1 0	B/RTS (VAR) [DIR]	BRINCO-RETORNO. Si VAR es negada se realiza un brinco a la dirección DIR. Si es afirmada se retorna el control al estado continuo al que llamó a la subrutina.

TABLA 4.11. INSTRUCCIONES IMPLEMENTADAS PARA EL MODULO MYCA-II

4.8.5. FORMATO DE CONTROL

Nuestro formato de control será el siguiente:

OPCODE	MULTIPLEXOR	DIR BRINCO	SALIDAS DE CONTROL

- Opcode: Indica el código de operación asociado a la instrucción que será ejecutada.

- Multiplexor: Indica la terminal del multiplexor asociada a la variable que condiciona la instrucción que requiere ejecutarse.
- Dir Brinco: Indica la localidad de memoria a la que será transferido el contador de programa, en caso de que la instrucción a procesar involucre un brinco.
- Salidas de Control: Indica para cada salida del controlador si estará o no activa en el estado presente.

4.8.6. Metodología

a) Verificación de Restricciones y Asignación de Estados

Comprobar que cada estado del autómata cumpla con las restricciones mencionadas anteriormente. En caso de que no se cumplan, se deberá modificar el diagrama original en aquellos estados problemáticos.

Hechos los cambios necesarios reasignar los estados nuevamente.

b) Asignación de Acciones Inter Estados

Para cada estado se debe asignar una instrucción adecuada que permita el cambio al estado o estados siguientes. Al mismo tiempo se llevará a cabo el llenado de la Tabla de Acción.

c) Tabla de Entradas a Mux

Establecer el número de bits que poseerá cada uno de los campos de nuestro formato de control.

Multiplexor: Enumerar el número de variables diferentes que condicionan a una instrucción a partir de la tabla de acciones y asociar una terminal diferente del multiplexor a cada una de ellas. Si una variable condiciona a más de una instrucción, sólo se cuenta una vez. El número de bits necesarios para este campo estará dado por n en la expresión $2^n = X$, es decir es el equivalente al número de líneas de selección necesarias para direccionar X entradas hacia una salida.

Dir Brinco: El número de bits será equivalente al número de bits con que representamos a un estado en nuestro controlador.

Salidas: Habrá un bit para cada salida generada por el controlador.

d) Tabla de Microcódigo

La Tabla de Microcódigo es un arreglo de todos los bits de nuestra memoria separados en campos y mostrando su contenido (1 o 0) en cada uno de los estados de nuestro controlador. La tabla contiene cinco columnas fijas (Estado, Opcode, Multiplexor, Dir Brinco y Salidas) más una columna por cada memoria a utilizar (HexN a Hex1). El número de renglones de la tabla será igual al número de estados de nuestro controlador.

ESTADO	OPCODE	MULTIPLEXOR	DIR BRINCO	SALIDAS	HEXN	..	HEX1
a							
b							
...							

TABLA 4.12. ESTRUCTURA DE LA TABLA DE MICROCODIGO PARA EL METODO MYCA-II

La primera columna (Estado) contiene el nemónico correspondiente al estado del controlador.

Como sabemos a cada estado corresponde una acción. El llenado de la Tabla de Microcódigo depende de la Tabla de Acción y del diagrama implementado. Para cada estado las columnas Opcode, Multiplexor y Dir Brinco se llenan de acuerdo a la instrucción.

1er. caso: Si la instrucción es CC (Cuenta Condicional)

Opcode: Código binario 0.

Multiplexor: Colocar en binario el número de entrada al mux asociado a la variable condicionante.

Dir Brinco: Puede colocarse cualquier código puesto que no habrá brinco. Para nuestro método lo llenaremos con ceros.

2do. caso: Si la instrucción es CI (Cuenta Incondicional)

Opcode: Código binario 1.

Multiplexor: Llenar con ceros.

Dir Brinco: Llenar con ceros.

3er. caso: Si la instrucción es BC (Brinco Condicional)

Opcode: Código binario 10.

Multiplexor: Colocar en binario el número de entrada al mux asociado a la variable condicionante.

Dir Brinco: Colocar en binario la dirección a la cual se transferirá el control del programa.

4to. caso: Si la instrucción es BI (Brinco Incondicional)

Opcode: Código binario 11.

Multiplexor : Llenar con ceros.

Dir Brinco: Colocar en binario la dirección a la cual se transferirá el control del programa.

5to. caso: Si la instrucción es C/B (Cuenta Brinco)

Opcode: Código binario 100.

Multiplexor: Colocar en binario el número de entrada al mux asociado a la variable condicionante.

Dir Brinco: Colocar en binario la dirección a la cual se transferirá el control del programa.

6to. caso: Si la instrucción es C/JSR (Cuenta-Salto a Subrutina)

Opcode: Código binario 101.

Multiplexor: Colocar en binario el número de entrada al mux asociado a la variable condicionante.

Dir Brinco: Colocar en binario la dirección en la cual inicia la subrutina.

7mo. caso: Si la instrucción es B/RTS (Brinco-Retorno de Subrutina)

Opcode: Código binario 110.

MULTIPLEXOR: Colocar en binario el número de entrada al mux asociado a la variable condicionante.

DIR BRINCO: Colocar en binario la dirección del ciclo que se ejecutará mientras el control no sea regresado al punto en que fue llamada la subrutina.

El campo SALIDAS contiene un bit para cada salida, todas las salidas que sean generadas en el estado presente serán asignadas con un 1, las restantes con un 0.

Las columnas restantes HEX1 A HEXN dependerán tanto en número de ellas como en su contenido de las columnas anteriores. Su número se calcula con las fórmula: Num_Memorias = INT (Y/X)+1 referida anteriormente.

- Agrupar del bit menos significativo al más significativo en secuencias de X bits (longitud de palabra de la memoria), los campos de nuestro formato de control. Cada grupo corresponderá a cada memoria.
- Convertir a código hexadecimal el contenido de cada grupo binario para el grabado posterior de las memorias.
- Insertar las memorias en el alambrado del circuito.

c) DIBUJO DE ARQUITECTURA

El esquema lógico de nuestra arquitectura siempre será similar en aspecto a la fig. 4.35. Su estructuración es relativamente sencilla dado que el módulo MYCA II incluye a varios de los componentes requeridos. Lo único que varía de un microcontrolador a otro es el número de bits que parten de la memoria hacia el multiplexor y hacia los bits de entrada al módulo MYCA II, así como el número de entradas recibidas y salidas generadas.

Al mux de entradas se le asocian las variables condicionantes de acuerdo a la Tabla de Entradas a Mux llenada anteriormente.

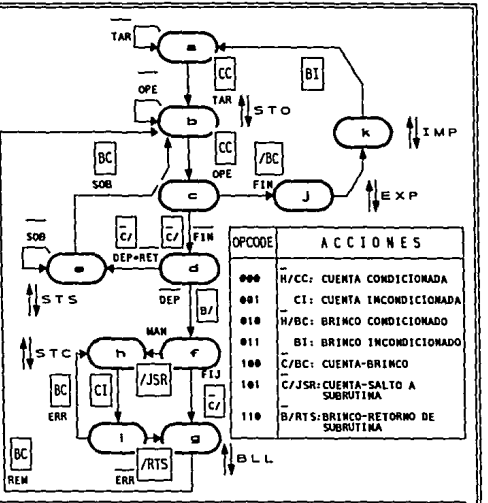
4.8.7. SOLUCIÓN AL CAJERO AUTOMÁTICO

Partiendo del aútomata original de la fig. 4.1., y al verificar restricciones es evidente que requiere el replanteamiento del autómata, como

resultado obtendremos un diagrama idéntico al que obtuvimos por los métodos Microprogramado y MYCA-I. Aunque podríamos solucionar este autómata por MYCA-II, realmente no se vería el uso de las instrucciones adicionales; por lo cual replantearemos una vez más al autómata. El autómata de la fig.4.30. es funcionalmente equivalente al de la fig.4.36., aunque a primera vista parezca lo contrario. Para ejemplificar el uso de subrutinas se ha incluido un nuevo tipo de operación, si el usuario elige un retiro, el cajero automático le permitirá seleccionar entre retiro fijo (FIJ), lo cual representa una cantidad ya establecida, y retiro manual (MAN), en el cual el usuario especifica la cantidad a retirar.

Recuerde que lo ideal al aplicar MYCA-II sería que el diagrama original lo requiriera, si este no fuera el caso aconsejamos utilizar otro tipo de metodología más simple.

La solución presentada a continuación es por Introducción Tabular Normal, pero también es válida la Introducción Invertida.



OPCODE	ACCIONES
000	H/CC: CUENTA CONDICIONADA
001	CI: CUENTA INCONDICIONADA
010	H/BC: BRINCO CONDICIONADO
011	BI: BRINCO INCONDICIONADO
100	C/BC: CUENTA-BRINCO
101	C/JSR: CUENTA-SALTO A SUBRUTINA
110	B/RTS: BRINCO-RETORNO DE SUBRUTINA

ENTRADAS

- DEP: OPERACION DE DEPOSITO
- ERR: ERROR EN CANTIDAD PEDIDA
- FIN: FIN DE SESION
- REM: CANTIDAD PEDIDA IGUAL A EMITIDA POR CAJERO
- SOB: FUE INSERTADO SOBRE CON CANTIDAD DEPOSITADA
- TAR: TARJETA INSERTADA
- OPE: YA SE ELIGIO OPERACION
- FIJ: OPERACION DE RETIRO FIJO
- MAN: OPERACION RETIRO MANUAL

SALIDAS

- BLL: ENVIA BILLETE A LA CHAROLA
- EXP: CAJERO EXPULSA TARJETA
- IMP: IMPRIME COMPROBANTE DE OPERACION
- STC: CAJERO SOLICITA CANTIDAD A RETIRAR POR EL USUARIO
- STS: CAJERO SOLICITA CANTIDAD A DEPOSITAR POR USUARIO
- STO: CAJERO SOLICITA TIPO DE OPERACION

ESTADOS

	D	C	B	A
a	0	0	0	0
b	0	0	0	1
c	0	0	1	0
d	0	0	1	1
e	0	1	0	0
f	0	1	0	1
g	0	1	1	0
h	0	1	1	1
i	1	0	0	0
j	1	0	0	1
k	1	0	1	0

A) ASIGNACION DE ESTADOS POR SECUENCIA

Fig. 4.36. SOLUCION para CAJERO AUTOMATICO por METODO MYCA-II (1a. Parte)

ESTADO	ACCION
a	$\overline{H}$/CC (TAR)
b	$\overline{H}$/CC (OPE) <STO>
c	$\overline{C}$/BC (FIN) [0111]
d	$\overline{C}$/BC ($\overline{DEP}$) [0101]
e	$\overline{H}$/BC (SOB) [0011] <STS>
f	$\overline{C}$/JS (MAN)
g	$\overline{H}$/BC (REM) [0101] <BLL>
h	CI <STC>
i	B/RT <$\overline{ERR}$> [0000]
j	CI <EXP>
k	BI <IMP>

B) TABLA DE ACCIONES

TERMINAL	MUX ENTRADAS
0	TAR
1	OPE
2	FIN
3	$\overline{DEP}$
4	SOB
5	MAN
6	REM
7	$\overline{ERR}$
8 TERMINALES REQUIEREN 3 LINEAS DE SELECCION	

C) TABLA DE ENTRADAS A MUX

MSB LSB

OPCODE	DIR MUX	DIR BRINCO	SALIDAS

FORMATO DE CONTROL

ESTADO	OPCODE D_{15} D_{14} D_{13}	DIR MUX D_{12} D_{11} D_{10}	DIR BRINCO D_9 D_8 D_7 D_6	SALIDAS D_5 D_4 D_3 D_2 D_1 D_0	HEX2	HEX1
a	0 0 0	0 0 0	0 0 0 0	0 0 0 0 0 0	00	00
b	0 0 0	0 0 1	0 0 0 0	1 0 0 0 0 0	04	20
c	1 0 0	0 1 0	1 0 0 1	0 0 0 0 0 0	8A	40
d	1 0 0	0 1 1	0 1 0 1	0 0 0 0 0 0	8D	40
e	0 1 0	1 0 0	0 0 0 1	0 1 0 0 0 0	50	50
f	1 0 1	1 0 1	0 1 1 1	0 0 0 0 0 0	B5	C0
g	0 1 0	1 1 0	0 0 0 1	0 0 1 0 0 0	58	48
h	0 0 1	0 0 0	0 0 0 0	0 0 0 1 0 0	20	04
i	1 1 0	1 1 1	0 1 1 1	0 0 0 0 0 0	DD	C0
j	0 0 1	0 0 0	0 0 0 0	0 0 0 0 1 0	20	02
k	0 1 1	0 0 0	0 0 0 0	0 0 0 0 0 1	60	01

HEX2 HEX1

D) TABLA DE MICROCODIGO EN MODO NORMAL

Fig. 4.37. SOLUCION A CAJERO AUTOMATICO POR MYCA-II NORMAL (2a. Parte)

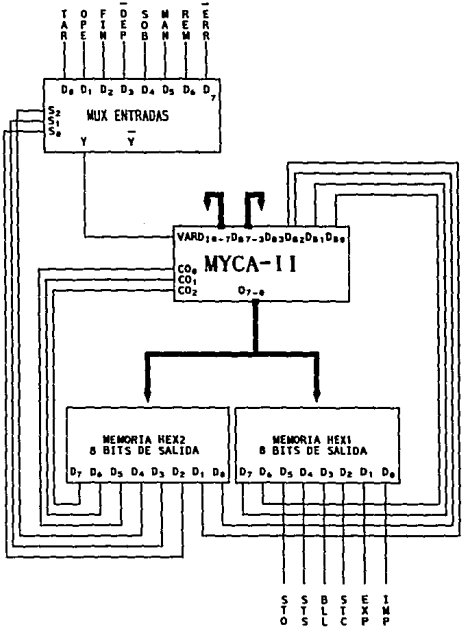


Fig. 4.38. IMPLEMENTACION PARA CAJERO AUTOMATICO POR MYCA-II NORMAL

4.9. MICROCONTROLADORES CON EL C.I. MC2909

Para finalizar este capítulo se ha incluido una metodología que utiliza como elemento principal a una pastilla LSI, a diferencia de secciones anteriores aquí especificaremos el circuito integrado a utilizar, estudiaremos su configuración y su modo de empleo para una implementación particular

4.9.1. ESTRUCTURA Y DESCRIPCIÓN DEL C.I. MC2909

El C.I. MC2909 es un controlador de direcciones de 4 bits, aplicable al secuenciamiento de microinstrucciones almacenadas en una memoria ROM o PROM.

El MC2909 posee cuatro fuentes para seleccionar una dirección:

1) Entradas externas directas.

2) Datos externos almacenados en un registro interno.

3) Pila de cuatro palabras, permite ejecución de subrutinas.

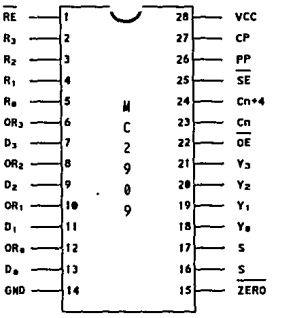
4) Registro contador de programa.

Por medio de la elección de una fuente apropiada y dirigiendo los bits necesarios hacia las terminales de control es posible simular diversas instrucciones, como veremos en la sección siguiente.

La fig. 4.39. muestra la estructura y descripción de las terminales correspondientes al MC2909. Las tablas de acción necesarias para la implementación de instrucciones se muestran en la fig. 4.40.

4.9.2. ELEMENTOS

- C.I. MC2909: Autosecuenciador Microprogramable y elemento principal.
- MEMORIA RAM: Almacenará el microprograma.
- MULTIPLEXORES: Los cuales conducirán las entradas necesarias hacia las entradas de control So y S1.



ENTRADAS

S_0-S_1 : LINEAS DE CONTROL DE SELECCION DE FUENTE PARA DIRECCION

$\overline{RE}$: HABILITACION PARA REGISTRO DIRECCIONADOR INTERNO

$\overline{OE}$: HABILITACION DE SALIDAS

$\overline{SE}$: HABILITACION DE PILA

PP : INDICACION DE OPERACION DE PILA

Cn : ACARREO DE ENTRADA AL INCREMENTADOR

R_0-R_3 : ENTRADAS HACIA EL REGISTRO DIRECCIONADOR INTERNO

D_0-D_3 : ENTRADAS DIRECTAS HACIA EL MULTIPLEXOR

OR_0-OR_3: ENTRADAS LOGICAS OR SOBRE CADA LINEA DE SALIDA

$\overline{ZERO}$: DIRECCION CERO A LA SALIDA

CP : PULSO DE RELOJ

SALIDAS

Cn+4 : ACARREO DE SALIDA DEL INCREMENTADOR

Y_0-Y_3 : DIRECCIONES DE SALIDA

Fig.4.39. DESCRIPCION DE TERMINALES DEL CIRCUITO INTEGRADO MC2909

S_0	S_1	DESCRIPCION
0	0	ACTIVA CONTADOR DE PROGRAMA
0	1	ACTIVA OPERACION DE PILA
1	0	ACTIVA REGISTRO DIRECCIONADOR
1	1	ACTIVA ENTRADAS DIRECTAS

TABLA A) SELECCION DE FUENTE DE DIRECCION

OR	$\overline{ZERO}$	$\overline{OE}$	YI
*	*	1	ALTA IMPEDANCIA
*	0	0	0
1	1	0	1
0	1	0	FUENTE SELECCIONADA POR So Y S1

TABLA B) SELECCION PARA CONTROL DE SALIDAS

INSTRUCCION	S_0	S_1	CI	$\overline{RE}$	$\overline{ZE}$	$\overline{SE}$	P/P
CI	0	0	1	*	1	*	*
$\overline{RT/CC}$	X	0	1	0	1	*	*
BI	1	1	0	*	1	*	*
$\overline{RT/BC}$	X	X	0	*	1	*	*
$\overline{C/B}$	X	X	1	*	1	*	*

TABLA C) IMPLEMENTACION DE INSTRUCCIONES

Fig. 4.40. TABLAS DE SELECCION E INSTRUCCIONES PARA EL C.I. WC2909

4.9.3. RESTRICCIONES

- Ningún estado del autómata deberá poseer más de dos estados siguientes.
- No se permite la especificación de dos cuentas o de dos brincos simultáneamente a partir de un estado presente.

4.9.4. INSTRUCCIONES

El cuadro de instrucciones a utilizar será análogo al que hemos utilizado anteriormente, para la implementación de cada instrucción con el MC2909, solamente debemos aplicar los bits requeridos a sus terminales de control. Los valores requeridos se muestran en la fig. 4.40. en su inciso C) y se obtienen a partir de las tablas de los incisos A) y B).

INSTRUCCIÓN	DESCRIPCIÓN
CC (VAR)	CUENTA CONDICIONADA a la variable VAR.
CI	CUENTA INCONDICIONAL.
BC (VAR) [DIR]	BRINCO CONDICIONADO a la variable VAR y dirigido hacia la dirección DIR.
BI	BRINCO INCONDICIONAL.
C/BC (VAR)[DIR]	CUENTA-BRINCO. Si la variable VAR es verdadera se realiza un brinco a la dirección DIR. Si es negada se realiza una cuenta.

TABLA 4.13. INSTRUCCIONES INCLUIDAS PARA LA METODOLOGIA MC2909

A continuación se anota la obtención de cada instrucción:

CI: Al ubicar un cero en las terminales So y S1 seleccionamos el contador de programa y el incremento se consigue colocando un uno en la terminal de acarreo de entrada Ci.

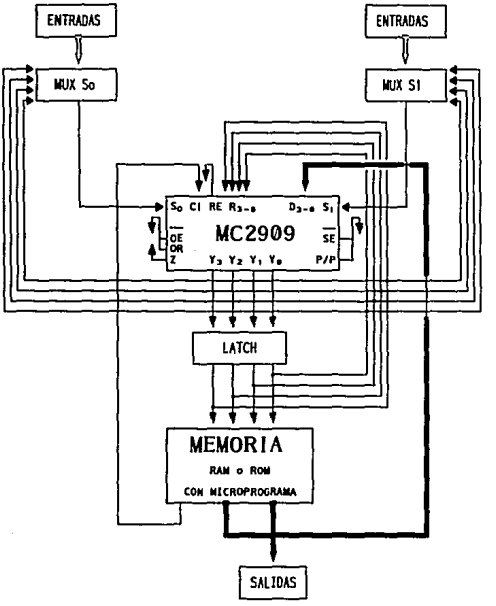


Fig. 4.41. ARQUITECTURA PARA EL METODO SECUENCIADOR MC2909

CC: Fijamos S1 a cero y So a X. Si la variable condicional X está afirmada se activa el registro direccionador, el cual contiene la dirección actual ocurriendo un efecto de retención; si la variable está negada ocurre una selección de contador de programa y un incremento en el estado al haber un uno en el estado Ci.

BI: Al seleccionar un uno en So y S1 activamos la entrada directa, la cual conducirá a la salida la dirección del estado deseado almacenada en la memoria. El valor en Ci no importa pues no es seleccionado el contador de programa.

BC: Fijamos So y S1 a X. Si la variable condicional X es afirmada obtenemos un brinco a la dirección especificada, en caso contrario se activa el contador de programa que al tener un cero en Ci realiza un efecto de retención al no incrementar su valor.

C/B: Fijamos So y S1 a X. Si la variable condicional X es afirmada se activa la entrada directa ocasionando un brinco, si es negada obtendremos una cuenta al activarse el contador de programa y tener un uno en Ci.

4.9.5. Formato de Control

Nuestro formato debe incluir las entradas de control necesarias. Si observamos el cuadro de instrucciones nos damos cuenta que las terminales Ci, RE, ZE, SE y P/P presentan un mismo código binario para cualquier instrucción, por lo cual pueden fijarse directamente a un nivel de voltaje, no requiriendo ser incluidas en el formato de control.

Así el formato de control se ve reducido y tendrá la forma siguiente:

So	S1	Ci	DIR BRINCO	SALIDAS DE CONTROL

- So y S1: Terminales para selección de fuente de dirección.
- Ci: Acarreo de entrada.

- Dir Brinco: Indica la localidad de memoria a la que será transferido el contador de programa, en caso de que la instrucción a procesar involucre un brinco.
- Salidas de Control: Indica para cada salida del controlador si estará o no activa en el estado presente.

4.9.6. Metodología

a) Verificación de Restricciones y Asignación de Estados

Comprobar que cada estado del autómata cumple con las restricciones mencionadas anteriormente. En caso contrario se deberá modificar el diagrama original en aquellos estados problemáticos.

Hechos los cambios necesarios se reasignan los estados nuevamente.

b) Asignación de Acciones Inter Estados

Para cada estado se debe asignar una instrucción adecuada que permita el cambio al estado o estados siguientes. Al mismo tiempo se llevará a cabo el llenado de la Tabla de Acción.

c) Tabla de Entradas a Mux

Para dirigir la entrada correcta a las terminales So y S1 del MC2909 se requiere el uso de dos multiplexores. Cada estado requiere una entrada especifica de acuerdo a la instrucción a ejecutarse y se coloca en una tabla tomando en consideración lo especificado en la fig. 4.40.

d) Tabla de Microcódigo

La Tabla de Microcódigo estará formada por todos los bits de nuestra memoria separados en campos y mostrando su contenido (1 o 0) en cada uno de los estados de nuestro controlador. La tabla contiene cinco columnas fijas (Estado, So, S1, Ci, Dir Brinco y Salidas) más una columna por cada memoria a utilizar (HexN a Hex1). El número de renglones de la tabla será igual al número de estados de nuestro controlador.

ESTADO	So	S1	Ci	DIR BRINCO	SALIDAS	HEXN	.. .	HEX1
a								
b								
...								

TABLA 4.14. ESTRUCTURA DE LA TABLA DE MICROCODIGO UTILIZADA PARA EL METODO MC2909

La primera columna (Estado) contiene el nemónico correspondiente al estado del controlador.

Como sabemos a cada estado corresponde una acción. El llenado de la Tabla de Microcódigo depende de la Tabla de Acción y del diagrama implementado. Para cada estado las columnas So, S1, Ci y Dir Brinco se llenan de acuerdo a la instrucción.

1er. caso: Si la instrucción es CC (Cuenta Condicional)

So: Colocar la variable condicional en forma negada.

S1: Colocar un cero.

Ci: Colocar un uno.

Dir Brinco: Puede colocarse cualquier código puesto que no habrá brinco. Para nuestro método lo llenaremos con ceros.

2do. caso: Si la instrucción es CI (Cuenta Incondicional)

So: Colocar un cero.

S1: Colocar un cero.

Ci: Colocar un uno.

Dir Brinco: Llenar con ceros.

3er. caso: Si la instrucción es BC (Brinco Condicional)

So: Colocar la variable condicional en forma afirmada.

S1: Colocar la variable condicional en forma afirmada

Ci: Colocar un cero.

Dir Brinco: Colocar en binario la dirección a la cual se transferirá el control del programa.

4to. caso: Si la instrucción es BI (Brinco Incondicional)

So: Colocar un uno.

S1: Colocar un uno.

Ci: Colocar un cero.

Dir Brinco: Colocar en binario la dirección a la cual se transferirá el control del programa.

5to. caso: Si la instrucción es C/B (Cuenta Brinco)

So: Colocar la variable condicional en forma afirmada.

S1: Colocar la variable condicional en forma afirmada.

Ci: Colocar un uno.

Dir Brinco: Colocar en binario la dirección a la cual se transferirá el control del programa.

El campo Salidas contiene un bit para cada salida, todas las salidas que sean generadas en el estado presente serán asignadas con un 1, las restantes con un 0.

Las columnas restantes Hex1 a HexN dependerán tanto en número de ellas como en su contenido de las columnas anteriores. Su número se calcula con las fórmula: Num_Memorias = INT (Y/X)+1 referida anteriormente.

- Agrupar del bit menos significativo al más significativo en secuencias de X bits (longitud de palabra de la memoria), los campos de nuestro formato de control. Cada grupo corresponderá a cada memoria.
- Convertir a código hexadecimal el contenido de cada grupo binario para el grabado posterior de las memorias.
- Insertar las memorias en el alambrado del circuito.

e) Dibujo de Arquitectura

El esquema lógico de nuestra arquitectura siempre será similar en aspecto a la fig. 4.41. Obsérvese que la mayoría de las entradas de control del MC2909 han sido fijadas a un nivel de voltaje. Cada terminal de cada multiplexor es asociada a la variable o voltaje requerido de acuerdo a la Tabla de entradas a Mux.

También es importante destacar la presencia de un buffer entre el MC209 y las memorias utilizadas, lo hemos incluido en esta metodología aunque en realidad se aconseja su uso siempre que se utilicen memorias.

4.9.7. Solución al Cajero Automático

El autómata que cumple con las restricciones mencionadas y se apega al conjunto de instrucciones que hemos configurado es idéntico al que planteamos por el método microprogramado.

La solución presentada a continuación es por Introducción Tabular Normal, pero también es válida la Introducción Invertida.

Sólo resta decir, que el método propuesto aquí ha sido apegado al conjunto de instrucciones que hemos manejado, pero que las características del circuito MC2909 son capaces de producir instrucciones aún más diversas, sobre todo mediante el uso de su pila interna, que puede generar subrutinas y anidación de las mismas.

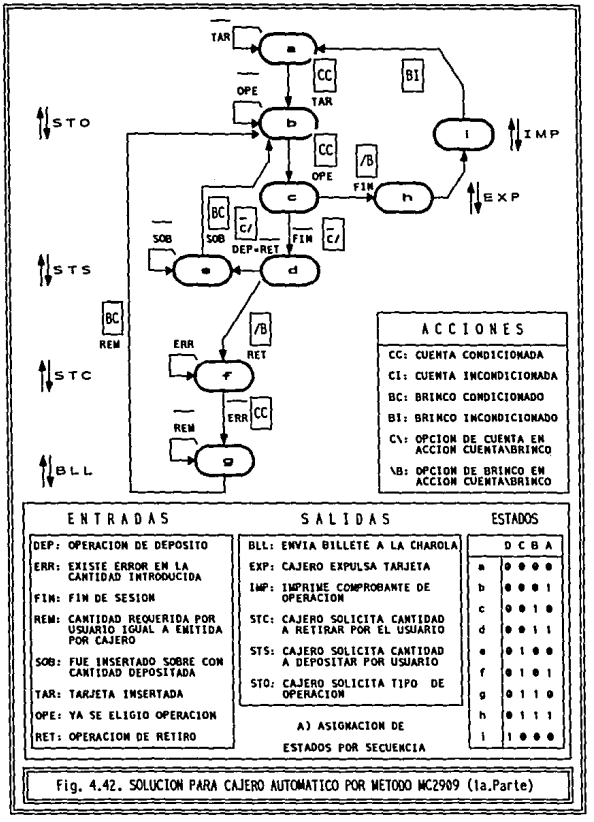


Fig. 4.42. SOLUCION PARA CAJERO AUTOMATICO POR METODO MC2909 (1a.Parte)

ESTADO	ACCION	
a	CC (TAR)	
b	CC (OPE)	<STO>
c	C/B (FIN) [0111]	
d	C/B (RET) [0101]	
e	BC (SOB) [0001]	<STS>
f	CC (ERR)	<STC>
g	BC (REM) [0001]	<BLL>
h	CI	<EXP>
i	BI [0000]	<IMP>

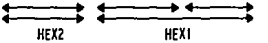
B) TABLA DE ACCIONES

TERMINAL	MUX So	MUX S1
0	$\overline{TAR}$	0
1	$\overline{OPE}$	0
2	FIN	FIN
3	RET	RET
4	SOB	SOB
5	ERR	0
6	REM	REM
7	0	0
8	1	1

C) TABLA DE ENTRADAS A MUX

FORMATO DE CONTROL

ESTADO	CI D_{10}	DIR BRINCO D_9 D_8 D_7 D_6				SALIDAS D_5 D_4 D_3 D_2 D_1 D_0						HEX2	HEX1
a	1	0	0	0	0	0	0	0	0	0	0	04	00
b	1	0	0	0	0	1	0	0	0	0	0	04	20
c	1	0	1	1	1	0	0	0	0	0	0	05	C0
d	1	0	1	0	1	0	0	0	0	0	0	05	40
e	0	0	0	0	1	0	1	0	0	0	0	00	50
f	1	0	0	0	0	0	0	1	0	0	0	04	08
g	0	0	0	0	1	0	0	0	1	0	0	00	44
h	1	0	0	0	0	0	0	0	0	1	0	04	02
i	0	0	0	0	0	0	0	0	0	0	1	00	01



D) TABLA DE MICROCODIGO EN MODO NORMAL

Fig. 4.43. SOLUCION PARA CAJERO AUTOMATICO POR METODO MC2909 (2a. Parte)

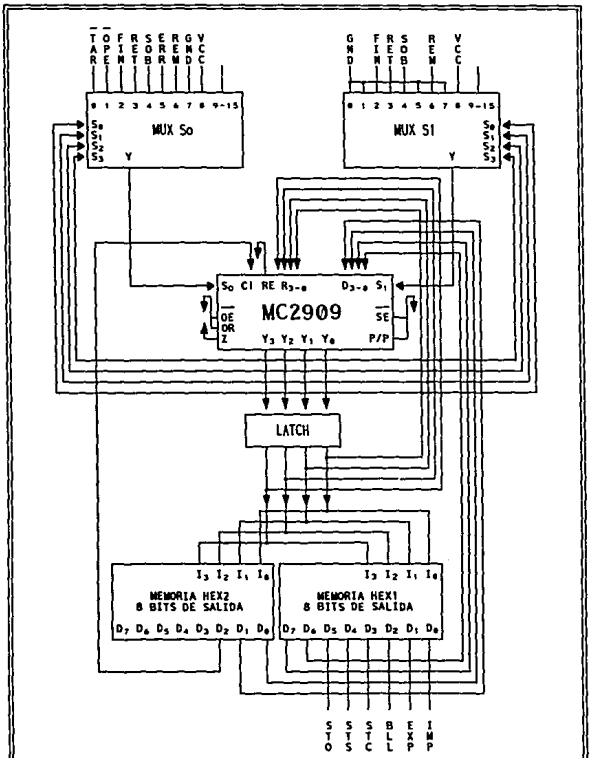


Fig. 4.44. IMPLEMENTACION PARA CAJERO AUTOMATICO POR METODO MC2909 (3a. Parte)

Capítulo Cinco

Diseño de un Sistema Automático para Desarrollo de MicroControladores

5.1. Introducción

5.2. Especificación de las Características deseadas del Sistema

5.3. Límites y Restricciones del Sistema

5.4. Modularización del Sistema

5.5. Desarrollo de Algoritmos y Flujogramas para Automatización del Sistema

5.6. Codificación del Sistema

5.7. Documentación

5.1. INTRODUCCIÓN

Ha sido concluido el primer objetivo primordial en la realización de esta tesis, es tiempo de emprender el desarrollo del sistema que nos hemos propuesto, y para ello requeriremos realizar una serie de etapas, a continuación se presentan en forma secuencial pero generalmente estando en etapas avanzadas es necesario retroceder por el camino recorrido para replantear u ordenar ideas previamente concebidas.

5.2. ESPECIFICACIÓN DE LAS CARACTERÍSTICAS DESEADAS DEL SISTEMA

Al plantear el desarrollo de una aplicación para computadora se requiere especificar anticipadamente las características esenciales que denotarán su funcionamiento interno, su acoplamiento con software ya existente y la interacción con el usuario. Al tiempo que se establece una característica es necesario estar consciente de su justificación; no es válido fijar una característica sin explicar el porque de ella. En base a este concepto el sistema propuesto adoptará las características siguientes:

a) Se ejecutará bajo un ambiente operativo gráfico multiventanas; el software actual posee esta tendencia, ya que el usuario requiere la ejecución de varias aplicaciones simultáneamente distribuyendo entre ellas los recursos disponibles. Si bien el trabajar bajo este entorno dificulta la tarea al programador, redunda en grandes beneficios para el usuario, entre los que se destaca la estandarización en el aspecto funcional de la aplicación, lo cual facilita el aprendizaje de programas que trabajan bajo los mismos conceptos de programación.

b) La aplicación contará con los conceptos CUA[1] tendientes a maximizar el grado interactivo máquina-usuario. Tales conceptos son: uso de

---

[1] El CUA (Acceso común para usuario) es parte del estándar SAA (Arquitectura para Sistemas de Aplicación) que consiste en integrar paquetes de software completamente diferentes dentro de una interface común, facilitando su aprendizaje al operador.

menús flotantes, *Cajas de Diálogo* y *Botones* para selección de respuestas, *Cuadros de Selección* para acceso a archivos, etc. Todas estas herramientas deberán ser accesibles al usuario mediante teclado o ratón según sea su preferencia. Es indudable que las aplicaciones más exitosas en el mercado mundial del software son aquellas que reúnen las características mencionadas, que dan un aspecto amigable a quien las utiliza.

c) Incluirá *Esquemas de Ayuda* dirigidos al usuario. Anteriormente los fabricantes de software al distribuir un programa lo acompañaban de un manual dirigido al usuario, si éste no lo poseía, le resultaba bastante difícil o imposible la operación de su aplicación. Actualmente todo buen programa que se precie de serlo debe incluir entre sus menús de operación uno destinado a los esquemas de ayuda que orienten al usuario sobre el objetivo de cada elemento incluido en el programa.

d) Poseerá un módulo *Tutorial* integrado. Este punto tiene un funcionamiento similar al anterior pero un fin más profundo. Cuántas veces nos hemos encontrado con programas que por su naturaleza técnica no los comprendemos adecuadamente debido a la falta de conocimientos sobre el tema que aborda. Realmente todo programa de este tipo debiera incluir entre sus opciones el uso de un tutorial dirigido hacia aquellas personas no familiarizadas con los conceptos y terminología utilizados. Esto da pauta a que una persona no especializada en áreas específicas pueda utilizar en forma sencilla el software que requiere.

e) Habrá un espacio para *Rutinas de Validación.* Realmente es decepcionante el uso de programas que no manejen adecuadamente la introducción de errores. Muchos programas al aceptar estos errores y dirigirlos hacia sus diferentes módulos causan diversos problemas en tiempo de ejecución que en la mayoría de los casos conduce a un aborto de la aplicación o a una interrupción indefinida en el sistema, lo cual puede ocasionar pérdidas en la información que se procesa.

5.3. Limites y restricciones del Sistema

Conociendo la estructura del problema es necesario demarcar los límites y restricciones del sistema, el primer concepto es fijado por el diseñador en base a los aspectos específicos que necesita o desea resolver, las restricciones son impuestas por la naturaleza propia del problema.

5.3.1. Límites

Ahora indicaremos los límites permisibles del sistema, es decir, aquello que aceptará y en consecuencia aquello que debe rechazar. Para cada opción limítrofe debe existir una rutina de validación que verifique en forma eficaz la legalidad de la acción realizada.

- Permitirá la solución de un autómata dado, mediante tres metodologías diferentes.
- El sistema aceptará la introducción de un máximo de 16 estados.
- Cada estado poseerá un máximo de 5 salidas y de acuerdo a la base teórica el conjunto de ellas será representada por un sólo bit.
- Cada estado presente tendrá un máximo de 5 estados siguientes en su trazo, pero el número de ellos estará restringido por la metodología empleada. Esto significa que al momento de introducir el autómata se permite indicar hasta 5 enlaces (incluyendo el de retención) para un estado, pero su solución será realizada sólo si elige la metodología adecuada.
- El nemónico para cada salida debe poseer exactamente tres caracteres alfanuméricos, esto para evitar la confusión entre identificadores para estados y salidas.

5.3.2. Restricciones

- No se permite que un nemónico sea asociado a más de un estado.
- Todo estado debe tener como mínimo un estado siguiente. No se permite la introducción de un estado en forma aislada.

- El usuario podrá indicar la instrucción de transición entre dos estados, sin embargo si no es factible su utilización el sistema cancelará su uso y especificará la instrucción adecuada.

5.4. MODULARIZACIÓN DEL SISTEMA

Es ya casi una práctica universal solucionar sistemas que impliquen programación mediante el proceso de modularización, esto se debe principalmente a las características propias de los ordenadores, su accionar siempre es metódico y cada proceso a efectuar se apoya en un reducido grupo de estructuras lógicas. Cada función realizada por una computadora puede ser desglosada en una serie de instrucciones ejecutadas mediante un control adecuado, así mismo, lo contrario también resulta cierto, mediante la integración apropiada de instrucciones podemos modelar el funcionamiento de prácticamente cualquier situación o problema, que deseemos o tengamos necesidad de resolver.

El proceso de modularización tal vez parezca muy simple en teoría, pero no lo es totalmente en la práctica. Un buen número de personas dedicadas a la programación de sistemas confunde la modularización con la segmentación, no basta agrupar el contenido de un programa en secciones secuenciales. Un módulo para poderlo llamar así requiere varias características más, primeramente debe quedar claro que un módulo debe poseer una función bien específica dentro del programa, la característica general a todo módulo es el procesamiento de datos lo cual origina la obtención de resultados, tanto éstos como los primeros constituyen información (recibida y emitida), se debe especificar en forma clara la naturaleza de esta información y su transformación interna si es que la hay, para ello es recomendable recurrir al uso de parámetros y argumentos.

Un módulo debe ser reutilizable, pudiendo ser accesado en múltiples ocasiones sin necesidad de reescribir innecesariamente código común.

Como en todo sistema, no debe buscarse la solución a una aplicación para computadora atacándola como un todo. Debe fragmentarse en secciones menos complejas más asequibles al pensamiento humano, este proceso se denomina modularización y se le llama módulo a cada sección obtenida. El

concepto opera de la siguiente forma, dado un proceso complejo, se divide en módulos con funciones simples, se les da una solución individual, y se integran al todo mediante un marco de agrupación.

Cuando la especificación del problema ha sido comprendida y planteada en forma clara, la modularización resulta un proceso sencillo.

Generalmente cada característica formulada para una aplicación es factible englobarla como un módulo. Después de todo una aplicación se define en base a sus características. Si esto resulta cierto bastará acoplar cada característica de la sección anterior dentro de un esquema que muestre a bloques su configuración interna.

Teniendo en mente los puntos considerados se procede a la modularización de nuestro sistema. Para cumplir con los requerimientos y características que se desean en él, definiremos a sus módulos integrantes:

a) Módulo de Entrada-Salida

b) Módulo de Edición de Autómata

c) Módulo para Solución de Autómata

d) Módulo para Visualización de Resultados

e) Módulo para Impresión de Resultados

f) Módulo de Ayuda

g) Módulo Interactivo

La cantidad de módulos integrantes de un sistema está determinada por sus requerimientos y por las características inherentes a él. En primera instancia el planteamiento de funciones se asigna en la forma más general posible, evitando traslapamientos.

a) Módulo Entrada-Salida: Tendrá las siguientes funciones:

- Capacidad para almacenar el diagrama correspondiente a un autómata
- Capacidad para recuperar el diagrama de un autómata previamente almacenado.

- Permitirá la inicialización del sistema a fin de terminar el trabajo actual y comenzar uno nuevo.

b) Módulo de Edición de Autómata

Este módulo nos facilitará las herramientas necesarias para introducir el esquema de un autómata, como ya sabemos un diagrama MDS se compone de estados, enlaces y salidas generadas además de sus nemótecnicos correspondientes, por lo tanto, el sistema debe ser capaz de permitir la inserción de cada uno de estos elementos que en conjunto integrarán nuestro autómata final, además el sistema debe permitir editar a los elementos insertados ya sea mediante su modificación o por eliminación.

La edición del autómata se realiza a nivel interno y externo, el primero representa los cambios necesarios en las estructuras de datos que contienen la información del modelo planteado, el nivel externo es lo que el usuario mira en la pantalla o en la impresora, en este caso se compone de texto y gráficos comprensibles al observador y que son trazados a partir de la recuperación de los datos internamente almacenados.

Obviamente toda aplicación tendiente a solucionar un sistema requiere el uso de una gran cantidad de estructuras de datos para simulación interna del problema y su posible manipulación. El objetivo de esta sección no es describir la función de cada estructura que se pretenda emplear en el sistema, sólo interesa especificar a los elementos centrales que constituirán la base del sistema, al modelo del autómata descrito en este inciso y al modelo para su solución descrito en el inciso *c)* de esta secuencia.

Para el modelado del autómata se ha previsto el uso de una tabla matricial bidimensional que sea capaz de contener en su interior la estructura completa de un diagrama MDS, tal cual se muestra en la fig. 5.1.

c) Módulo de Impresión

Permitirá la impresión de los elementos siguientes:

- Diagrama del autómata

A CONTINUACION SE MUESTRA EL MODELO DE LA ESTRUCTURA QUE ALMACENARA AL AUTOMATA CORRESPONDIENTE A UN MICROCONTROLADOR. SI OBSERVA CUIDADOSAMENTE NOTARA QUE TODA LA INFORMACION CONCERNIENTE A UN DIAGRAMA MDS PUEDE SER OBTENIDA POR COMPLETO A PARTIR DE ESTA ESTRUCTURA.

POR LO TANTO, LA APLICACION A DESARROLLAR ESTARA BASADA EN ESTA ESTRUCTURA QUE SERA ACCESADA GLOBALMENTE POR UNA GRAN CANTIDAD DE MODULOS INTEGRANTES DEL SISTEMA.

POSX	POSY	ESTADO	ENL[i]	VENL[i]	COND [i]	SALIDAS[i]
		1				
		.				
		.				
		.				
		16				

16 RENGLONES PARA 16 ESTADOS

POSX: INDICARA LA COLUMNA DE LA REJILLA EN QUE SERA VISUALIZADO EL ESTADO.
POSY: INDICARA EL RENGLON DE LA REJILLA EN QUE SERA VISUALIZADO EL ESTADO.
ESTADO: INDICA EL NUMERO CLAVE CON QUE SERA REFERENCIADO EL ESTADO.
ENL[i]: ARREGLO DE "i" LOCALIDADES PARA ESTABLECER "i" ENLACES CON OTROS ESTADOS DEL AUTOMATA REFERENCIADOS POR SU NUMERO CLAVE.
VENL[i]: ARREGLO QUE INDICA LA VARIABLE DE TRANSICION ENTRE EL ESTADO PRESENTE Y EL ESTADO SIGUIENTE "i".
COND[i]: ARREGLO QUE INDICA LA INSTRUCCION CONDICIONAL OPCIONAL ENTRE EL ESTADO PRESENTE Y EL ESTADO SIGUIENTE "i".

SALIDAS[i]: LISTA DE SALIDAS ASOCIADAS AL ESTADO.

Fig.5.1. ESTRUCTURA DE DATOS QUE CONTENDRA AL AUTOMATA DEL MICROCONTROLADOR

- Tabla de Estados Futuros
- Tabla de Acción
- Tabla de MicroCódigo
- Diagrama Lógico correspondiente a la solución del autómata

d) Módulo de Ayuda

Tendrá dos funciones principales:

- Auxiliar al usuario en la operación del sistema explicando el uso de cada una de sus características.
- Proporcionarle ayuda sobre los conceptos y terminología empleados por el sistema.

e) Módulo para Solución de Autómata

Incluirá al conjunto de metodologías que serán soportadas por el sistema, para cada una de ellas se contará con submódulos que efectúen las acciones siguientes:

- Validación del autómata: Cada metodología, precisa restricciones particulares para su posible aplicación, de manera que si un autómata no cumple la totalidad de ellas no procederá su solución.
- Formación interna para solución del autómata: Requiere de un proceso de modelado, ampliaremos el uso de tablas bidimensionales utilizado en el modelo del autómata debido a las características enumerativas del problema
- Módulo para despliegue de resultados: Los resultados los agruparemos en dos conjuntos, el primero abarca exclusivamente al diagrama lógico y es propiamente el objetivo final del sistema, el segundo grupo incluye los pasos intermedios necesarios para obtener el diagrama lógico a partir del autómata. En ambos casos el proceso consistirá en leer la información almacenada en las tablas de datos descritas anteriormente y visualizarla con el formato apropiado.

f) Módulo Interactivo

Debe existir un módulo que coordine las funciones efectuadas por el sistema. Permitirá la selección de comandos al usuario y los dirigirá adecuadamente a los módulos restantes. Esta interacción se realizará a través de una interfaz gráfica compuesta en su base por un menú de opciones desplegable. Dada la naturaleza del módulo interactivo, será uno de los más extensos y estará formado por una serie de submódulos que realicen las funciones siguientes:

- Captura de errores en el uso del sistema.
- Manejo de rutinas de petición de datos al usuario.
- Acceso a todas las características del sistema mediante comandos.

5.5. Desarrollo de Algoritmos y Flujogramas para Automatización del Sistema

Habiendo identificado los módulos integrantes del sistema debemos darle solución a cada uno de ellos. Cada módulo se visualiza como un pequeño programa que recibe datos, los procesa mediante una lógica específica, realiza accciones en base a este proceso, y retorna resultados. Para generar un módulo se requiere especificar mediante secuencias de pasos simples la ruta apropiada que tomará la aplicación para cada situación particular, esto puede indicarse en forma escrita mediante algoritmos o en forma gráfica mediante flujogramas, ambos métodos son equivalentes.

La figura 5.2. muestra un esquema del menú que soportará la estructura general de la aplicación. Estará compuesto por seis submenús flotantes, cada uno de ellos formado por una serie de comandos que accesarán a los módulos ya definidos en la secuencia requerida.

El haber modularizado el sistema para integrarlo bajo un menú general nos permitirá desarrollarlo de manera muy conveniente. Ahora podemos enfocar nuestra atención en procesos más simples, tomando cada módulo e implementándolo de la manera que juzguemos más apropiada apegándonos a los objetivos planteados. Antes de pretender escribir directamente las instrucciones mediante un lenguaje de programación se debe considerar el uso de instrumentos intermedios que faciliten la comprensión de la solución que

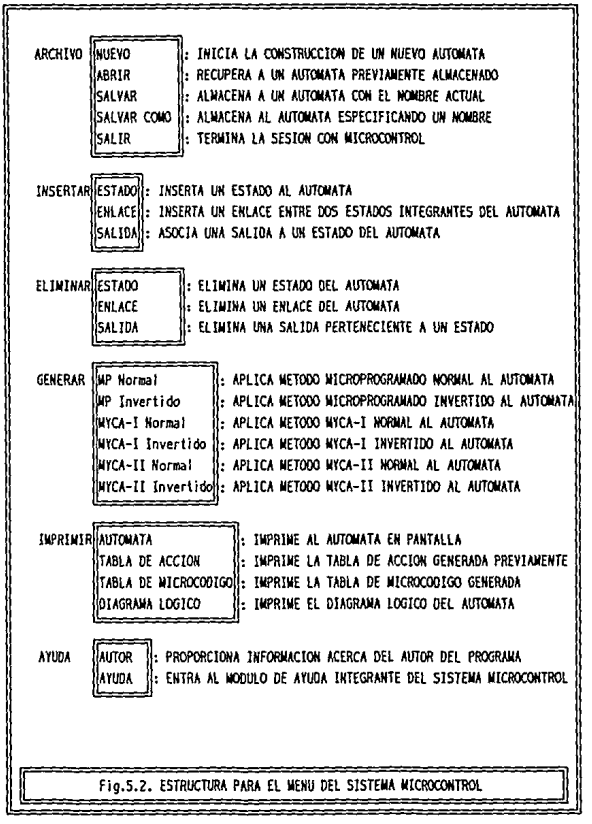


Fig.5.2. ESTRUCTURA PARA EL MENU DEL SISTEMA MICROCONTROL

deseamos implementar, este paso reducirá significativamente la complejidad al momento de codificar el programa constituyendo a la vez un mejoramiento a la documentación del sistema, los instrumentos empleados más comúnmente comprende algoritmos, pseudocódigos y diagramas de flujo, su uso puede ser indistinto y de acuerdo al gusto del programador, ya que la función de todos ellos es similar.

En las páginas siguientes abordaremos el funcionamiento de los comandos integrantes del sistema en forma individual utilizando para ello algoritmos

Los algoritmos descritos a continuación son bastante generales, debido a los límites de nuestro espacio, sólo tenga en cuenta que lo que aquí se escribe en unas decenas de líneas al programarlas mediante un lenguaje de computadora se requerirá de miles de líneas de código fuente y por supuesto de una gran cantidad de tiempo.

A pesar de todo los algoritmos aquí incluidos son una guía fundamental para entender el funcionamiento del sistema y constituyen una buena fuente de documentación.

Algoritmo para el comando: ARCHIVO NUEVO

Existe un autómata en la rejilla ?

Verdadero:

Ha sido modificado sin haberlo almacenado ?

Verdadero:

- Pregunta al usuario si desea salvarlo.

Lo desea salvar ?

Verdadero:

- Ejecuta el comando: **Archivo Salvar**

Falso:

- Continua el proceso

Falso:

- Continua el proceso

- Elimina todo el contenido de la rejilla, borrando cualquier elemento que exista en ella.

Falso:

- Continua el proceso

- Inicializa a sus valores originales el contenido de todas las estructuras de datos empleadas por el programa.

Algoritmo para el comando: ARCHIVO ABRIR

Ejecuta el comando: **Archivo Nuevo**

- Muestra al usuario los autómatas almacenados en el directorio actual.

- Solicita al usuario el nombre y ruta del archivo que contiene al autómata deseado.

Existe archivo indicado ?

Verdadero:

Contiene a un autómata ?

Verdadero:

- La información almacenada en el archivo la copia a estructuras de datos en RAM.

- Decodifica la información leída visualizando en pantalla al autómata resultante.

Falso:

- Mensaje de Error: El archivo no contiene a ningún autómata.

Falso:

- Mensaje de Error: Archivo indicado no existe.

- Cancela comando.

Algoritmo para el comando: ARCHIVO SALVAR

El autómata contenido en la rejilla tiene nombre ?

Verdadero:

- Continua proceso.

Falso:

- Solicita nombre al usuario.

- Almacena el contenido de la matriz que contiene al autómata en un archivo de disco aplicando un formato específico que posteriormente reconozca el sistema.

Algoritmo para el comando: ARCHIVO SALVAR COMO

- Solicita nombre al usuario para el autómata contenido en la rejilla.

- Almacena el contenido de la matriz que contiene al autómata en un archivo de disco aplicando un formato específico que posteriormente reconozca el sistema.

Algoritmo para el comando: ARCHIVO SALIR

Existe un autómata en la rejilla ?

Verdadero:

Ha sido modificado sin haberlo almacenado ?

Verdadero:

- Pregunta al usuario si desea salvarlo.

Lo desea salvar ?

Verdadero:

- Ejecuta el comando: **Archivo Salvar**

Falso:

- Continua el proceso

Falso:

- Continua el proceso

- Elimina toda estructura dinámica de memoria y restaura valores iniciales modificados por el sistema.

Algoritmo para el comando: INSERTA ESTADO

La casilla indicada está vacía (Inserción de Estado) ?

Verdadero:

- Solicita nemotécnico del estado
- Marca la casilla como ocupada
- Incrementa número de estados
- Almacena información del estado (columna,
- renglón, nemónico asociado).
- Habilita los comandos: **Eliminar Estado** e **Imprimir Autómata**
- Dibuja el estado en la rejilla

Falso: (Es decir la casilla indicada esta ocupada (Edición de Estado)

- Indica el nemónico actual del estado y solicita uno nuevo.
- Modifica información del estado y actualiza nemónico
  en pantalla.

Algoritmo para el comando: INSERTA SALIDA

La casilla indicada está vacía ?

Verdadero:

- Mensaje de Error: Una casilla vacía no puede tener salidas.

Falso:

- Cuenta número actual de salidas que posee el estado ubicado en la casilla.

Tiene espacio libre para más salidas ?

Verdadero:

- Solicita nemónico para la salida a insertar.
- Almacena información en la tabla de estados.
- Incrementa contador de salidas asociadas al estado.

Falso:

- Mensaje de Error: Exceso de salidas en el estado indicado.

Algoritmo para el comando: INSERTA ENLACE

Este comando requiere la selección de dos casillas, la que contiene al estado presente y la que contiene al estado siguiente.

Alguna o las dos casillas indicadas están vacias ?

Verdadero:

- Mensaje de Error: Una casilla vacía no puede tener enlaces.

Falso:

El estado presente tiene espacio para más enlaces ?

Verdadero:

- Solicita al usuario los datos de enlace: variable de acción e instrucción asociada.
- La información recibida la valida y la inserta en la estructura que contiene al autómata.
- Une gráficamente en pantalla a los estados indicados por medio de segmentos rectilineos.
- Incrementa el contador de enlaces asociado al estado.

Falso:

- Mensaje de Error: Exceso de enlaces en el estado indicado.

Algoritmo para el comando: ELIMINA ESTADO

La casilla indicada está vacía ?

Verdadero:

- Mensaje de Error: No existe estado que pueda ser eliminado.

Falso:

- Elimina información del estado (columna, renglón, nemónico asociado y enlaces)
- Borra el estado de la rejilla, así como los enlaces que tengan su origen en él.
- Marca la casilla como vacía.
- Decrementa el contador de estados.

Existe al menos un estado en la rejilla ?

Verdadero:

- Continua el proceso.

Falso:

- Deshabilita los comandos: Eliminar Estado e Imprimir Autómata.

Algoritmo para el comando: ELIMINAR ENLACE

Este comando requiere la selección de dos casillas, la que contiene al estado presente y la que contiene al estado siguiente.

Alguna o las dos casillas indicadas están vacías ?

Verdadero:

- Mensaje de Error: Una casilla vacía no puede tener enlaces.

Falso:

- Busca el enlace en la estructura del estado presente, eliminando la información anterior y marcando el espacio libre para la inserción de un nuevo enlace.
- Borra de pantalla el enlace indicado.
- Decrementa el contador de enlaces asociado al estado presente.

Algoritmo para el comando: ELIMINA SALIDA

La casilla indicada está vacía ?

Verdadero:

- Mensaje de Error: No existe estado al cual asociarle una salida.

Falso:

- Muestra al usuario las salidas que posee actualmente el estado indicado y le solicita aquella que desea eliminar.

El usuario eligió alguna salida ?

Verdadero:

- Elimina la salida elegida, marcando la localidad en que estaba almacenada.
- Decrementa el contador de salidas asociado al estado indicado.

Falso:

- Finaliza el proceso.

Algoritmo para el submenú: GENERAR

El usuario podrá aplicar una metodología al autómata contenido en la rejilla seleccionándola a partir del menú desplegable denominado *Generar*. Cada metodología constituye por sí sola un módulo, pero todas ellas realizan un proceso similar.

- Verifica que el autómata cumpla con las restricciones generales aplicables a la construcción de diagramas MDS.
- Verifica que el autómata cumpla con las restricciones específicas impuestas por la metodología seleccionada.

Existe algún error en la construcción del autómata ?

Verdadero:

- Mensaje de Error: Despliega error y causa posible.
- Cancela la aplicación de la metodología.

Falso:

- Aplica la metodología. Este punto implica un gran número de pasos y basa su lógica en los procesos discutidos en el capítulo cuatro. Recuerde que cada metodología fue explicada en detalle mediante un conjunto de incisos, la totalidad de ellos forma la estructura requerida por cada método en particular.
- Todos los datos intermedios se almacenan en tablas para su proceso por los módulos necesarios y/o por los comandos seleccionados por el usuario.
- Crea una ventana que contenga el diagrama lógico correspondiente al autómata de acuerdo al método seleccionado.
- Crea una ventana que contenga el desarrollo necesario para llegar al diagrama lógico del autómata.

Algoritmo para el submenú: IMPRIMIR

El usuario podrá elegir la impresión de datos y resultados a partir del menú desplegable denominado *Imprimir*. Cada proceso de impresión constituye por sí solo un módulo, pero todos ellos realizan un procedimiento similar.

- Recupera los datos a imprimir a partir de las estructuras de datos que los contienen.
- Verifica las características de la impresora
- Aplica los factores de escala y formatos de presentación requeridos por los datos a imprimir.
- Inhabilita los comandos de impresión y salida del sistema. para evitar errores en tiempo de proceso.
- Inicia proceso de impresión.
- Habilita comandos de impresión y salida del sistema.

Algoritmo para el submenú: AYUDA

Propiamente el esquema de ayuda consiste de un conjunto de pantallas con información específica sobre un tema, el algoritmo simplemente debe permitir el cambio de una pantalla por otra de acuerdo a la selección del usuario. La ayuda parte de un índice general con una serie de capítulos, la navegación entre ellos se realiza en relación a su contenido y bastará seleccionar el tema adecuado señalándolo con el cursor de la aplicación.

En la figura 5.2. se incluye el comando *Autor*, con el fin de orientar al usuario sobre quien diseño el sistema y por tanto a quien puede acudir para solucionar problemas relacionados con la aplicación y a quien dirigir sus sugerencias.

5.6. Codificación del Sistema

Habiendo estructurado a los módulos integrantes del sistema, es necesario abocarse a la extenuante tarea que implica la codificación de cada módulo.

La solución mediante algoritmos y flujogramas a un sistema es independiente al equipo de cómputo destinado a ejecutar la aplicación final, es por esto que nos hemos dedicado a establecer un resultado conceptual, es decir, se ha procurado plantear la idea de como implementar el sistema en detalle pero sin llegar a su codificación mediante un lenguaje de programación específico.

Ahora bien uno de nuestros objetivos originales fue la completa implementación del sistema para desarrollar microcontroladores de manera que para la codificación y ejecución de la aplicación debemos elegir un lenguaje de programación y un entorno operativo apropiado.

He optado por utilizar como entorno operativo al sistema *Windows Microsoft*, aunque actualmente existen otros productos mejores; la razón de ello es que *Windows* presenta una tendencia a mejorar continuamente, lo cual en poco tiempo lo hará superior a sus competidores, si a esto agregamos que es el estándar multiventanas más popular y con mayor soporte en nuestro país no hay duda que la elección es buena.

En cuanto a lenguaje de programación he seleccionado a *Pascal* en su versión *Turbo Pascal*[2], por las características que posee.

a) Esta orientado a los conceptos de programación estructurada y modularización. Ambos empleados en la aplicación a desarrollar..

b) Utiliza OOPS (Programación Orientada a Objetos), imprescindible para desarrollar aplicaciones bajo Windows Microsoft.

c) La lógica del código fuente se lee con relativa facilidad, considerando que la programación Windows es eventual más que secuencial.

---

[2] Turbo Pascal es una marca registrada por Borland International.

d) Turbo Pascal es capaz de producir programas compactos, con un buen grado de eficiencia.

e) Las características operativas de *Turbo Pascal* permiten un considerable ahorro de tiempo en procesos de edición y compilación.

5.7. Documentación

La documentación del sistema se debe realizar en tres niveles, la totalidad de ellos ha sido cubierta por nuestra aplicación.

1. El código fuente correspondiente al programa debe contener suficientes comentarios que indiquen el funcionamiento de los módulos que lo integran.

2. Se debe tratar que el sistema sea auto documentado, lo cual se logra mediante un esquema de ayuda lo suficientemente extenso.

3. Por escrito deben especificarse las bases teóricas del sistema, aquí lo hemos hecho a través del capítulo cuatro. El uso de algoritmos, flujogramas, especificaciones y limitaciones (todos ellos contenidos en este capítulo) constituyen en conjunto el centro de la documentación esencial que debe acompañar a un sistema.

Las obras de consulta incluidas en la sección bibliográfica también representan una fuente de documentación importante.

Capítulo Seis

Aplicación del Sistema en la realización de un MicroControlador

6.1. Características Finales del Sistema

6.2. Aplicación del Sistema

6.3. Futuro del Sistema

6.1. CARACTERÍSTICAS FINALES DEL SISTEMA

Este capítulo se ha escrito en forma posterior al desarrollo del sistema, al momento de escribir estas líneas se cuenta ya con la aplicación software completamente estructurada, a continuación se listan las características que posee, como se podrá apreciar cubre por completo las especificaciones planteadas en el capítulo anterior e incorpora algunos conceptos que se decidieron al momento de su desarrollo:

a) Permite trazar a un autómata que incluya un máximo de 16 estados.

b) Permite obtener el diagrama lógico correspondiente a un microcontrolador a partir del trazado de su autómata.

c) La operación del sistema hace uso frecuente del *mouse*

d) La selección de comandos para su ejecución se realiza a partir de un menú general.

e) Las metodologías soportadas por el sistema son:

- Método Microprogramado
- Método MYCA-I
- Método MYCA-II

Todas ellas en sus formas *normal* e *invertida*.

f) Permite almacenar en disco a los autómatas trazados para su recuperación posterior.

g) Genera automáticamente los microprogramas que deben almacenarse en las memorias incluidas en los diagramas lógicos.

h) Incluye un sistema completo de ayuda dirigida al usuario para operar adecuadamente el sistema.

i) Utiliza ampliamente las características soportadas por el estándar *Windows*: ventanas, cajas de diálogo, etc.

Para que el sistema pueda operar correctamente se requiere el equipo siguiente:

a) Una computadora PC tipo AT, con un mínimo de 1 megabyte en RAM y 300 kilobytes de espacio en disco duro.

b) Debe tener instalado *Windows 3.1* o versión superior y contar con un *mouse* para su uso.

Antes de instalar al sistema consulte el Apéndice A: Instalación del Sistema MicroControl.

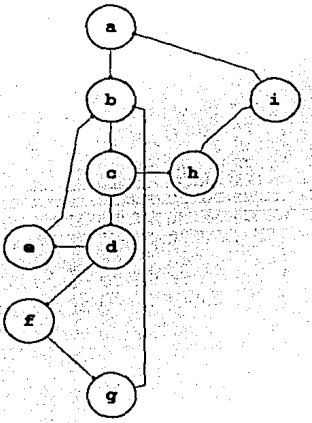
6.2. Aplicación del Sistema

Para evaluar la aplicabilidad del sistema se utilizaron los autómatas ilustrados en las figuras 4.22. y 4.36. correspondientes al problema del cajero automático y que hemos utilizado como ejemplo en el desarrollo de todas las metodologías discutidas en este trabajo. El proceso a realizar consiste en introducir el diagrama MDS en la computadora con ayuda del sistema MicroControl y aplicarle mediante los comandos apropiados cada una de las metodologías soportadas por él. Los resultados obtenidos en pantalla y posteriormente en forma impresa deben ser comparados con los obtenidos a lo largo del capítulo cuatro, en las siguientes páginas se muestran los resultados obtenidos tal y como fueron impresos. Se podrá observar que los resultados son equivalentes por completo, cabe señalar que aunque en este espacio sólo ha sido incluida la solución al cajero automático, el uso y los resultados obtenidos a partir del sistema se ha extendido a una gran cantidad de autómatas, todos ellos han sido resueltos en forma manual y en forma automática mediante el sistema, en todos los casos el sistema ha demostrado consistencia mediante la generación de resultados enteramente satisfactorios, por lo cual considero que el sistema cumple en su totalidad sus objetivos primarios.

MicroControl MicroControl

AUTOMATA:E:\TPW\PAS\CAJERO1

ESTADO	CODIGO	ESTADOS FUTUROS	SALIDAS GENERADAS
a	0000	b:TAR	
b	0001	c:OPE	<STO>
c	0010	d:FIN h:FIN	
d	0011	e:DEP f:RET	
e	0100	b:SOB	<STS>
f	0101	g:ERR	<STC>
g	0110	b:REM	<BLL>
h	0111	i:	<EXP>
i	1000	a:	<IMP>



MicroControl MicroControl
AUTOMATA:E:\TPW\PAS\CAJERO1

Metodología:MicroProgramado Normal

Los microprogramas correspondientes a cada memoria han sido creados en el directorio de su autómata.

H1: E:\TPW\PAS\CAJERO1.MN1
H2: E:\TPW\PAS\CAJERO1.MN2

TABLA DE ACCION

ESTADO	ACCION
a	C.C (TAR)
b	C.C (OPE) <STO>
c	C/B (FIN) (FIN) [0111]
d	C/B (DEP) (RET) [0101]
e	B.C (SOB) [0001] <STS>
f	C.C (ERR) <STC>
g	B.C (REM) [0001] <BLL>
h	C.I <EXP>
i	B.I [0000] <IMP>

MicroControl MicroControl
AUTOMATA:E:\TPW\PAS\CAJERO1

Metodología:MicroProgramado Normal

TABLA DE MICROCODIGO

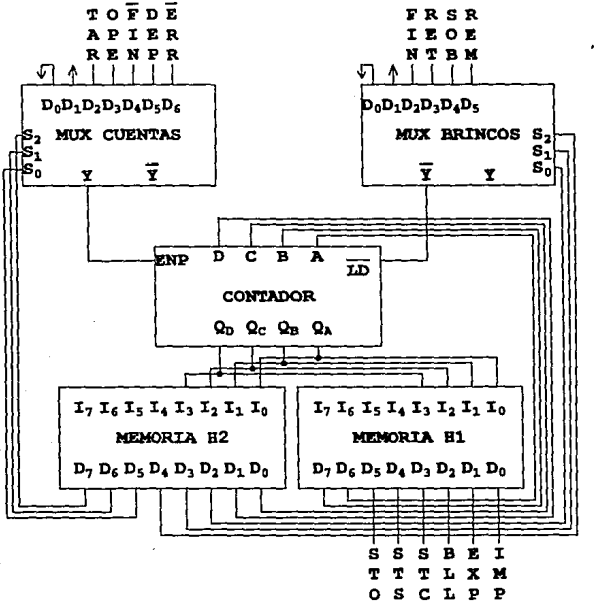
ESTADO	MUX CUENTAS $D_{15}D_{14}D_{13}$	MUX BRINCOS $D_{12}D_{11}D_{10}$	DIR BRINCO $D_9\ D_8\ D_7\ D_6$
a	0 1 0	0 0 0	0 0 0 0
b	0 1 1	0 0 0	0 0 0 0
c	1 0 0	0 1 0	0 1 1 1
d	1 0 1	0 1 1	0 1 0 1
e	0 0 0	1 0 0	0 0 0 1
f	1 1 0	0 0 0	0 0 0 0
g	0 0 0	1 0 1	0 0 0 1
h	0 0 1	0 0 0	0 0 0 0
i	0 0 0	0 0 1	0 0 0 0

ESTADO	SALIDAS $D_5\ D_4\ D_3\ D_2\ D_1\ D_0$	H2	H1
a	0 0 0 0 0 0	40	00
b	1 0 0 0 0 0	60	20
c	0 0 0 0 0 0	89	C0
d	0 0 0 0 0 0	AD	40
e	0 1 0 0 0 0	10	50
f	0 0 1 0 0 0	C0	08
g	0 0 0 1 0 0	14	44
h	0 0 0 0 1 0	20	02
i	0 0 0 0 0 1	04	01

MicroControl MicroControl
AUTOMATA:E:\TPW\PAS\CAJERO1

Metodología:MicroProgramado Normal

DIAGRAMA LOGICO



MicroControl MicroControl
AUTOMATA:E:\TPW\PAS\CAJERO1

Metodología:MicroProgramado Invertido

Los microprogramas correspondientes a cada memoria han sido creados en el directorio de su autómata.

H1: E:\TPW\PAS\CAJERO1.MI1
H2: E:\TPW\PAS\CAJERO1.MI2

TABLA DE ACCION

ESTADO	ACCION
a	C.C (TAR)
b	C.C (OPE) <STO>
c	C/B (FIN) (FIN) [0111]
d	C/B (DEP) (RET) [0101]
e	B.C (SOB) [0001]<STS>
f	C.C (ERR) <STC>
g	B.C (REM) [0001]<BLL>
h	C.I <EXP>
i	B.I [0000]<IMP>

MicroControl MicroControl
AUTOMATA:E:\TPW\PAS\CAJERO1

Metodología:MicroPrograma do Invertido

TABLA DE MICROCODIGO

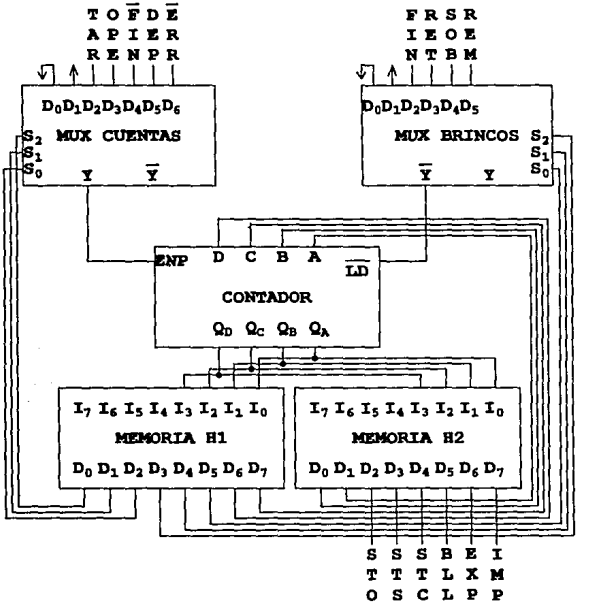
ESTADO	MUX CUENTAS D_0 D_1 D_2	MUX BRINCOS D_3 D_4 D_5	DIR BRINCO D_6 D_7 D_8 D_9
a	0 1 0	0 0 0	0 0 0 0
b	1 1 0	0 0 0	0 0 0 0
c	0 0 1	0 1 0	1 1 1 0
d	1 0 1	1 1 0	1 0 1 0
e	0 0 0	0 0 1	1 0 0 0
f	0 1 1	0 0 0	0 0 0 0
g	0 0 0	1 0 1	1 0 0 0
h	1 0 0	0 0 0	0 0 0 0
i	0 0 0	1 0 0	0 0 0 0

ESTADO	SALIDAS D_{10} D_{11} D_{12} D_{13} D_{14} D_{15} STO STS STC BLL RED INC	H1	H2
a	0 0 0 0 0 0	02	00
b	1 0 0 0 0 0	03	04
c	0 0 0 0 0 0	D4	01
d	0 0 0 0 0 0	5D	01
e	0 1 0 0 0 0	60	08
f	0 0 1 0 0 0	06	10
g	0 0 0 1 0 0	68	20
h	0 0 0 0 1 0	01	40
i	0 0 0 0 0 1	08	80

MicroControl MicroControl
AUTOMATA:E:\TPW\PAS\CAJERO1

Metodología:MicroProgramado Invertido

DIAGRAMA LOGICO



MicroControl MicroControl

AUTOMATA:E:\TPW\PAS\CAJERO1

Metodología:MYCA-I Normal

Los microprogramas correspondientes a cada memoria han sido creados en el directorio de su autómata.

H1: E:\TPW\PAS\CAJERO1.1N1
H2: E:\TPW\PAS\CAJERO1.1N2

TABLA DE ACCION

ESTADO	ACCION				
a	C.C		(TAR)		
b	C.C	_____	(OPE)		<STO>
c	C/B	(FIN)	(FIN)	[0111]	
d	C/B	(DEP)	(RET)	[0101]	
e	B.C		(SOB)	[0001]	<STS>
f	C.C		(ERR)		<STC>
g	B.C		(REM)	[0001]	<BLL>
h	C.I				<EXP>
i	B.I			[0000]	<IMP>

MicroControl MicroControl
AUTOMATA:E:\TPM\PAS\CAJERO1

Metodología:MYCA-I Normal

TABLA DE MICROCODIGO

ESTADO	OPCODE $D_{15}D_{14}D_{13}$	DIR MUX $D_{12}D_{11}D_{10}$	DIR BRINCO $D_9\ D_8\ D_7\ D_6$
a	0 0 0	0 0 0	0 0 0 0
b	0 0 0	0 0 1	0 0 0 0
c	1 0 0	0 1 0	0 1 1 1
d	1 0 0	0 1 1	0 1 0 1
e	0 1 0	1 0 0	0 0 0 1
f	0 0 0	1 0 1	0 0 0 0
g	0 1 0	1 1 0	0 0 0 1
h	0 0 1	0 0 0	0 0 0 0
i	0 1 1	0 0 0	0 0 0 0

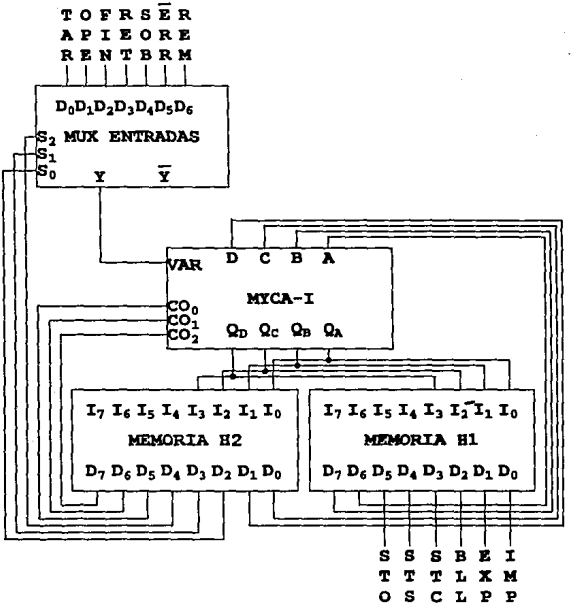
ESTADO	SALIDAS $D_5\ D_4\ D_3\ D_2\ D_1\ D_0$	H2	H1
a	0 0 0 0 0 0	00	00
b	1 0 0 0 0 0	04	20
c	0 0 0 0 0 0	89	C0
d	0 0 0 0 0 0	8D	40
e	0 1 0 0 0 0	50	50
f	0 0 1 0 0 0	14	08
g	0 0 0 1 0 0	58	44
h	0 0 0 0 1 0	20	02
i	0 0 0 0 0 1	60	01

MicroControl MicroControl

AUTOMATA:E:\TPW\PAS\CAJERO1

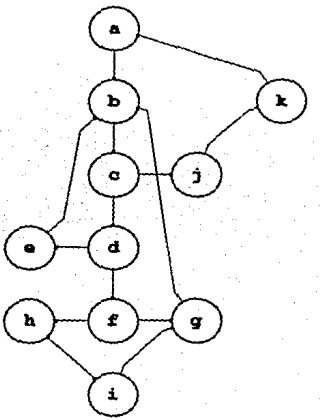
Metodología:MYCA-I Normal

DIAGRAMA LOGICO



AUTOMATA:C:\MICROX\CAJERO2

ESTADO	CODIGO	ESTADOS FUTUROS	SALIDAS GENERADAS
a	0000	b:TAR	
b	0001	c:OPE	<STO>
c	0010	d:$\overline{FIN}$ j:FIN	
d	0011	e:DEP f:$\overline{DEP}$	
e	0100	b:SOB	<STS>
f	0101	h:MAN g:FIJ	
g	0110	b:REM	<BLL>
h	0111	i:ENT	<STC>
i	1000	h:ERR g:$\overline{ERR}$	
j	1001	k:	<EXP>
k	1010	a:	<IMP>



MicroControl MicroControl

AUTOMATA:C:\MICROX\CAJERO2

Metodología:MYCA-II Normal

Los microprogramas correspondientes a cada memoria han sido creados en el directorio de su autómata.

H1: C:\MICROX\CAJERO2.2N1
H2: C:\MICROX\CAJERO2.2N2

TABLA DE ACCION

ESTADO	ACCION		
a	H/CC	(TAR)	
b	H/CC	(OPE)	<STO>
c	C/BC (FIN)	(FIN)	[1001]
d	C/BC (DEP)	(DEP)	[0101]
e	H/BC	(SOB)	[0001]<STS>
f	C/JS (FIJ)	(MAN)	[0111]
g	H/BC	(REM)	[0001]<BLL>
h	C.I		<STC>
i	B/RT (ERR)	(ERR)	[0111]
j	C.I		<EXP>
k	B.I		[0000]<IMP>

MicroControl MicroControl
AUTOMATA: C:\MICROX\CAJERO2

Metodología:MYCA-II Normal

TABLA DE MICROCODIGO

ESTADO	OPCODE $D_{15}D_{14}D_{13}$	DIR MUX $D_{12}D_{11}D_{10}$	DIR BRINCO D_9 D_8 D_7 D_6
a	0 0 0	0 0 0	0 0 0 0
b	0 0 0	0 0 1	0 0 0 0
c	1 0 0	0 1 0	1 0 0 1
d	1 0 0	0 1 1	0 1 0 1
e	0 1 0	1 0 0	0 0 0 1
f	1 0 1	1 0 1	0 1 1 1
g	0 1 0	1 1 0	0 0 0 1
h	0 0 1	0 0 0	0 0 0 0
i	1 1 0	1 1 1	0 1 1 1
j	0 0 1	0 0 0	0 0 0 0
k	0 1 1	0 0 0	0 0 0 0

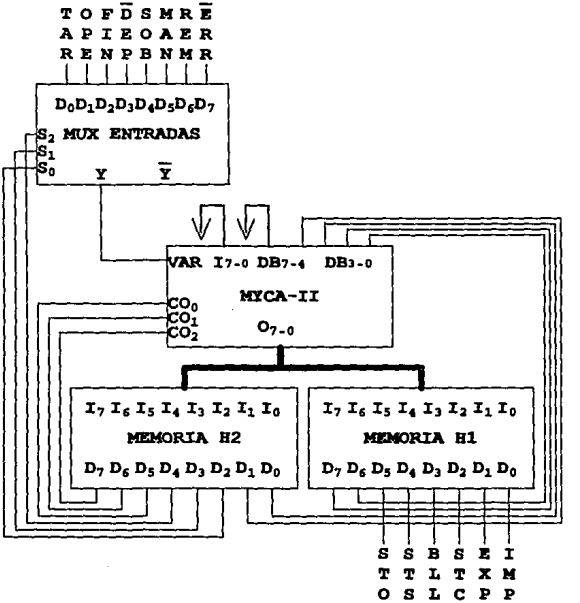
ESTADO	SALIDAS D_5 D_4 D_3 D_2 D_1 D_0 STO STS RLL STC REP INP	H2	H1
a	0 0 0 0 0 0	00	00
b	1 0 0 0 0 0	04	20
c	0 0 0 0 0 0	8A	40
d	0 0 0 0 0 0	8D	40
e	0 1 0 0 0 0	50	50
f	0 0 0 0 0 0	B5	C0
g	0 0 1 0 0 0	58	48
h	0 0 0 1 0 0	20	04
i	0 0 0 0 0 0	DD	C0
j	0 0 0 0 1 0	20	02
k	0 0 0 0 0 1	60	01

MicroControl MicroControl

AUTOMATA:C:\MICROX\CAJERO2

Metodología:MYCA-II Normal

DIAGRAMA LOGICO



6.3. Futuro del Sistema

Al desarrollar un sistema para computadora no debemos olvidar dos consideraciones importantes:

1a. **No existen los sistemas perfectos.** Al momento de implementar un sistema, por más pruebas que se le hagan durante su desarrollo nunca será perfecto, tarde o temprano aparecerán detalles que corregir,

2a. **Ningún sistema es eterno**: Todo sistema tiene un lapso de vida útil, el cual comienza en el momento de su implementación y termina cuando aparece un sistema mejor o cuando las características del problema han rebasado las características del sistema.

Si deseamos que un sistema tenga éxito debemos pensar en lo anterior y evitar en lo posible su estancamiento, para ello es recomendable tomar las siguientes actitudes:

1a. El sistema debe ser puesto a consideración a un gran número de personas principalmente a aquellas que constituirán a nuestro grupo de usuarios, recuerde que nadie conoce mejor el problema por solucionar que quien lo padece, en este aspecto nuestro grupo tal vez no sepa exactamente con que resolver su problema pero sí como les gustaría fuera resuelto,

2a. Deben considerarse las sugerencias aportadas al sistema, la incorporación de ellas puede enriquecer su funcionamiento y características facilitando su operación y permitiendo nuevas posibilidades al usuario.

3a. Si alguien más se interesa en el sistema, a fin de mejorar su calidad y prestaciones, no debe desecharse su ayuda, podría ser el eslabón necesario para darle continuidad al sistema evitando que la idea original se pierda en el tiempo.

Deseando que el sistema que he propuesto no sea de utilidad pasajera, acompañando a este libro se encuentra un disco flexible para computadora, el cual contiene el sistema MicroControl en forma integra así como el código fuente para su análisis y posible modificación. Seguramente usted después de utilizarlo tendrá sugerencias para mejorarlo e incluso deseos de modificarlo.

utilizarlo tendrá sugerencias para mejorarlo e incluso deseos de modificarlo. Como dije ningún, sistema es perfecto y una razón para ello es el usuario, que siempre requiere resolver problemas, un sistema deposita en sus manos una posible solución, más no es la única ni necesariamente la mejor, para que esto llegue a ser cierto se requiere una retroalimentación continua entre diseñador y usuario del sistema.

En este momento al haber concluido el desarrollo del sistema y habiéndolo utilizado constantemente han surgido en mi mente nuevas ideas tendientes a mejorarlo, entre ellas podría mencionar las siguientes:

a) Incrementar el número de metodologías para desarrollo de microcontroladores soportadas por el sistema.

b) Incrementar el número de estados permisible para estructurar a un autómata mediante el sistema.

c) Mejorar los métodos de dibujo que utiliza MicroControl para trazar autómatas en su rejilla.

d) A medida que se incrementen las características del sistema, puede ser necesario incluir el uso de iconos que permitan al usuario seleccionar en forma directa los comandos más frecuentes.

La integración al sistema de éstas nuevas ideas obviamente requiere aportar un tiempo mayor para su desarrollo. Sin embargo si decidiera implementarlas, estoy totalmente seguro que al final de ello, un nuevo cúmulo de pensamientos tendientes a mejorar el sistema estarían esperando su implementación, por ello creo prudente finalizar de manera temporal el desarrollo del sistema, concluyendo por ahora este trabajo y esperando que alguien más en algún momento futuro prosiga con el mismo si así lo cree conveniente.

Capítulo Siete

Conclusiones

7.1. Conclusiones

7.2. Sugerencias

7.1. CONCLUSIONES

He llegado al final de este trabajo y leyendo una vez más los objetivos planteados en su inicio, he de expresar que me siento realmente satisfecho con los resultados. Cada punto siento haberlo cumplido favorablemente.

Creo yo que en este momento, la persona más indicada para establecer sus conclusiones, es precisamente usted, que página a página me ha acompañado en este libro.

Mi mayor ilusión después del tiempo invertido en este trabajo es que toda persona que tenga a bien consultarlo primeramente obtenga un beneficio inmediato con los conocimientos en el estructurados y posteriormente tenga la oportunidad para despertar en ella la inquietud por proseguir la propuesta fundamental aquí plasmada, la búsqueda de la sistematización de procesos. Uno de los objetivos primordiales en el hombre es su desarrollo intelectual, el cual se da mediante la adquisición de conocimientos y la aplicación de ellos. Mientras el hombre no abandone todos aquellos procesos tediosos que consumen rutinariamente su existencia, no podrá liberar su mente, no utilizará jamás su poder innovativo, latente en todo momento, encerrado y exigiendo aflorar para crear alternativas hacia un mundo mejor.

7.2. SUGERENCIAS

A primera vista las partes integrantes de esta tesis (texto y programa) pueden ser aprovechadas con fines didácticos.

- El texto de esta tesis podría ser utilizado como material de apoyo para los cursos de Circuitos Lógicos y materias a fines que forman parte del plan académico de las carreras de Ingeniería.
- Sería útil la incorporación de los ejemplos resueltos en esta tesis al esquema de prácticas de laboratorio correspondiente a Circuitos Lógicos.
- Un buen proyecto a desarrollar en forma conjunta por alumnos de Ingeniería Electrónica e Ingeniería en Computación coordinados por sus profesores, sería el mejorar significativamente las funciones del sistema MicroControl, realizando las modificaciones apropiadas. Incluso se podría mejorar el sistema, teniendo en mente no solo fines educativos sino también lucrativos mediante la comercialización del software desarrollado. Es tiempo de que haya un acercamiento entre empresas e instituciones educativas, ya no es válido argumentar que el estudiante carece de experiencia. Mientras no se le dé la oportunidad y el apoyo debido, este hecho seguirá persistiendo y seguiremos dependiendo inevitablemente de soluciones parciales a problemas que nos aquejan y que nadie conoce mejor que nosotros mismos.

Por que no pensar en el desarrollo de una **Coordinación Estudiantil para Desarrollo de Sistemas**, bien estructurada e integrada por alumnos no solamente de Ingeniería, sino de otras áreas requeridas para su funcionamiento. Esta coordinación con el apoyo de las autoridades educativas podría realizar convenios con empresas públicas y privadas para solucionar los problemas que las afecten. Tanto empresas como estudiantes se verían altamente beneficiados.

Ventajas para las empresas:

a) El costo de los sistemas desarrollados sería significativamente menor al costo que tendrían que pagar a empresas privadas dedicadas a ello.

b) Posteriormente mediante la contratación de egresados pertenecientes a la Coordinación contarían con personal que conoce los problemas existentes en sus organizaciones.

Ventajas para la escuela y sus estudiantes:

a) Los estudiantes estarían en disposición para adquirir bastante experiencia al tener la oportunidad de conocer problemas prácticos. Al finalizar sus estudios contarían con mayores oportunidades de conseguir un buen trabajo.

b) Algunas de las asignaturas cursadas por los alumnos bien podrían ser evaluadas a través de los sistemas desarrollados por la coordinación.

c) Las autoridades podrían establecer convenios con empresas a fin de intercambiar asesoría técnica por donaciones de equipo o materiales requeridos por la escuela.

Finalmente creo que la creación de una coordinación de este tipo fomentaría la vinculación entre estudiante y empresa, propósito fundamental de toda institución educativa.

Anexos

Bibliografía

Apéndice A.

Instalación de MicroControl

BIBLIOGRAFÍA

APROACH TO DIGITAL SYSTEMS
William Fletcher

FAST AND LS TTL DATA
Motorola 4a.Ed.
1989

LÓGICA DIGITAL Y DISEÑO DE COMPUTADORAS
M. Morris Mano
Edit. Prentice Hall
México, 1989

TURBO PASCAL FOR WINDOWS
Gunther Farber & Martin Pauly
Edit. Abacus
USA, 1992

TURBO PASCAL 7 The Complete Reference
Stephen K.O'Brien & Steve Nameroff
Edit. Osborne McGraw-Hill
USA, 1993

WINDOWS 3.1 SECRETS
Brian Livingston
Edit. IDG WorldWide
USA, 1992

APÉNDICE A: INSTALACIÓN DEL SISTEMA MICROCONTROL

Adjunto a esta tesis se encuentra un disco para computadora que contiene los archivos necesarios para ejecutar al sistema que he denominado *MicroControl* y que ha sido desarrollado como parte integral de este trabajo.

Para ejecutar satisfactoriamente este programa se requiere mínimamente de los recursos siguientes:

- Computadora PC tipo AT con 1 MegaByte en RAM, y 300 KiloBytes de espacio disponible en su disco duro.
- *Windows Microsoft 3.1*[1] o posterior previamente instalado.

A continuación se describe detalladamente el proceso necesario para su instalación.

a) Encienda su computadora y cargue el sistema *Windows Microsoft*.

b) Active al ADMINISTRADOR DE PROGRAMAS.

c) Elija el submenu desplegable correspondiente a la opción ARCHIVO localizado en el ADMINISTRADOR DE PROGRAMAS.

d) Seleccione el comando EJECUTAR. En este punto será abierta una caja de diálogo titulada EJECUTAR.

e) Presione el botón EXAMINAR localizado en la caja de diálogo. Se abrirá una nueva caja de diálogo titulada EXAMINAR y sobrepuesta a la anterior titulada EJECUTAR.

f) Inserte el disco original del Sistema MicroControl que acompaña a esta tesis en la unidad de disco adecuada a su computadora.

---

[1] Windows Microsoft es una marca registrada por Microsoft Corporation.

g) Indique la unidad lógica de disco flexible en que insertó el disco de *MicroControl*. Al hacer esto, aparecerá el archivo INSTALAR.EXE en el cuadro NOMBRES DE ARCHIVO.

h) Haga un doble clic sobre el archivo INSTALAR.EXE. La caja de diálogo titulada EXAMINAR desaparecerá de la pantalla y se retornará el control a la caja titulada EJECUTAR.

i) Elija el botón ACEPTAR. La caja de diálogo será cerrada y comenzará la ejecución del instalador localizado en el disco de *MicroControl*.

j) Indique la ruta del subdirectorio en que está instalado *Windows Microsoft* en su disco duro y presione el botón CONTINUAR. Por omisión el instalador buscará en la ruta C:\WINDOWS.

k) Indique la unidad lógica de disco flexible en que está ubicado el disco que contiene al sistema *MicroControl* y presione el botón CONTINUAR.

l) Si los parámetros anteriores no son introducidos correctamente aparecerá un mensaje de error y se solicitarán nuevamente al usuario, o bien en su caso la instalación podrá ser suspendida eligiendo el botón CANCELAR.

m) Si los parámetros fueron correctamente escritos, el instalador inicializa al sistema *MicroControl* en su sistema *Windows Microsoft*. Este proceso es indicado en pantalla y toma unos cuantos segundos. Si no hay problema aparecerá un recuadro que asi se lo hará saber.

n) Para que *Windows Microsoft* reconozca al sistema *MicroControl* es necesario que abandone *Windows* y nuevamente entre en él.

o) Hecho el paso anterior notará que fue creado un grupo denominado *MicroControl* en su ADMINISTRADOR DE PROGRAMAS. Si no lo ha localizado debido a solapamiento de ventanas, utilice el comando MAS VENTANAS del submenú VENTANAS ubicado en el ADMINISTRADOR DE PROGRAMAS.

p) En el interior de este grupo se encuentra localizado un sólo icono correspondiente al sistema *MicroControl*, para ejecutarlo simplemente haga un doble clic sobre él.

q) A entrado a nuestro sistema y ahora puede hacer uso de él.

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
ARAGÓN
DIRECCION

UNIVERSIDAD NACIONAL AVANZADA DE MEXICO

MIGUEL ANGEL LERA LOPEZ
P r e s e n t e.

En referencia a su atento escrito de fecha 1o. de julio del año en curso, por el que solicita autorización para cambio de titulo de su trabajo de Tesis, mismo que propone se denomine "DESARROLLO AUTOMATICO DE MICROCONTROLADORES", dirigido por el profesor, Ing. DONACIANO JIMENEZ VAZQUEZ, con fundamento en el punto 6 y siguientes del Reglamento para Exámenes Profesionales en esta Escuela, y toda vez que la documentación presentada por usted reúne los requisitos que establece el precitado Reglamento; me permito comunicarle que ha sido aprobada su solicitud.

Aprovecho la ocasión para reiterarle mi distinguida consideración.

ATENTAMENTE
"POR MI RAZA HABLARA EL ESPIRITU"
San Juan de Aragón, Edo. de Méx., agosto 2 de 1993
EL DIRECTOR

M en I CLAUDIO C. MERRIFIELD CASTRO

c c p Lic. Alberto Ibarra Rosas, Jefe de la Unidad Académica.
c c p Ing. Juan Gastaldi Pérez, Jefe de Carrera de Ingeniería en Computación.
c c p Ing. Manuel Martínez Ortiz, Jefe del Departamento de Servicios Escolares.
c c p Ing. Donaciano Jiménez Vázquez, Asesor de Tesis.

CCMC/AIR/vr