UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

SISTEMA PORTÁTIL RECOLECTOR DE DATOS (HAND HELD) POR MEDIO DE CÓDIGOS DE BARRAS EN BASE A UN MICROCONTROLADOR M68HC11

T E S I S

Que para obtener el Título de

INGENIERO ELÉCTRICO ELECTRÓNICO

P r e s e n t a:

JORGE ALFREDO PÉREZ RODRÍGUEZ

DIRECTOR DE TESIS: M. en I. ALBERTO FUENTES MAYA

México, D.F. Septiembre 2003

A la ***Universidad Nacional Autónoma de México*** *por brindarme la mejor formación y educación a través de mi paso por ella, todas las experiencias tan diversas, inolvidables e incomparables a lo largo de estos seis años; formación integral que sólo la Máxima Casa de Estudios puede ofrecer.*

A la ***Facultad de Ingeniería****, por abrirme las puertas y brindarme la oportunidad de estudiar en sus aulas; mi Alma Máter, con la que siempre estaré agradecido e intentaré representar con orgullo y coraje.*

Al Programa de Alto Rendimiento Académico, en especial a Lidia Delgado por los constantes esfuerzos que realiza día a día en beneficio de sus estudiantes.

A todos los maestros que han dejado un recuerdo en mí y ayudado en mi formación tanto dentro como fuera de las aulas, así en mi paso por la UNAM como por el Colegio Humboldt de Puebla.

Al profesor Alberto Fuentes Maya por dirigir este proyecto de tesis brindándome todas las facilidades necesarias para su realización.

Al profesor Lorenzo Juárez Palafox, Sergio Atayde del Moral y José Antonio Vila García por su importante y valiosa ayuda en la comprensión, implementación y realización de este proyecto.

A mi familia:

A mis padres, quienes han formado todo mi ser, mi carácter, mis anhelos, sentimientos, y sueños; quienes me han otorgado el privilegio de vivir y me han dado siempre la oportunidad de elegir el rumbo de mi vida con un apoyo incomparable.

A mi hermana Jeannette y mi hermano Severino por compartir sus infancias conmigo y vivir su juventud a mi lado, tal vez por caminos, etapas y tiempos muy diversos, pero siempre con apoyo y cariño.

A mi Tabita y tíos Severino y Toño, quienes ayudaron a mi educación y formación desde niño y que han sido gran ejemplo a seguir y motivación desde entonces.

Espero que esto sea una manera de retribuir en forma simbólica todo lo que me han dado siempre. Los amo a todos.
"En el sudor de mis ojos pueden sentir el reflejo de sus almas en mi corazón."

A mis amigos de la Facultad, de los que puedo presumir tengo muchos y que a pesar de que cada día nos veremos menos, estoy seguro de que seguiremos compartiendo momentos juntos a lo largo de nuestras vidas. Gracias por hacer de nuestra coincidencia una realidad presente y futura.
A las personas y amigos que siempre han estado conmigo apoyándome y regañándome en su momento; quienes me escuchan, aconsejan y comparten su vida conmigo.

A la mujer que me enseñó a amar y que ha compartido conmigo un sentimiento tan grande como el que vivimos, brindándome lo mejor de su ser durante todo este tiempo juntos.
Gracias por tu apoyo y tu amor Mara. Te amo.

ÍNDICE GENERAL.

INTRODUCCIÓN.

Entre la electrónica y la computación existe una relación muy fuerte; manteniendo no solamente el vínculo entre los componentes electrónicos, que son la base del funcionamiento de las computadoras, y su funcionamiento interno y programación, sino obligando a conocer las dos ramas para profundizar o investigar, si es que no se conoce al respecto, en temas comunes e íntimamente ligados; por lo que las personas dedicadas a la electrónica deben conocer cómo utilizar los componentes destinados a la informática y poder hacer uso de ellos de forma no común, es decir, poder usar los recursos computacionales a su favor y no necesitar por lo tanto de expertos en el tema. Lo mismo pasa con la gente de computación. Ésta debe también conocer los conceptos básicos de electrónica y los diferentes dispositivos que se utilizan comúnmente y que existen en el mercado para poder auxiliarse de ellos en las aplicaciones o proyectos que estén desarrollando.

Una de las labores más interesantes dentro del estudio en Ingeniería electrónica es el desarrollo de interfases con dispositivos y arreglos ya existentes para aplicaciones de interés específico o el de crear nuevos instrumentos utilizando los diferentes desarrollos existentes.

A lo largo del siglo XX, la construcción de sistemas electrónicos ha exhibido un ritmo de crecimiento único; sus logros resultan impensables en cualquier otra rama de la tecnología. Una de las claves de este desarrollo ha sido la estandarización, miniaturización y abstracción de sus componentes. Este proceso comenzó con la radiodifusión comercial, continúa con la aparición de la televisión y recibe el impulso definitivo durante la Segunda Guerra Mundial. El tiempo transcurrido entre los bulbos, relés, válvulas, transistores, circuitos impresos hasta llegar a los circuitos integrados de densidades y tecnologías diversas actuales es tan breve, que aún hoy es posible encontrar personas que, a lo largo de su carrera profesional, han creado productos electrónicos sorprendentes con cada uno de estos dispositivos.

Uno de los componentes estándares más famosos han sido los pertenecientes a la familia TTL. Estos dispositivos han ofrecido durante años una variedad de útiles funciones estandarizadas, magníficamente implementadas e ingeniosamente divididas en porciones de 4, 8 o más bits. Sin embargo, ya en la década de los setenta el estilo de diseño TTL había entrado en crisis. En efecto, si bien la agrupación de estos componentes permitía al diseñador materializar cualquier producto imaginable, sus características finales en tamaño, consumo, fiabilidad y precio limitaban notablemente la posibilidad de ganar dinero mediante su comercialización. Adicionalmente, dado que cada producto debía diseñarse de manera artesanal, incluso la cantidad de ingenieros disponibles por aquella época para realizar estas tareas comenzó a ser insuficiente, constituyendo otro freno a las posibilidades de negocio.

Algunas personas comenzaron a preguntarse cómo salir de esta crisis. Por un lado, los avances en el proceso de integración permitían aumentar la complejidad de los circuitos integrados. Pero agregar complejidad a los dispositivos implicaba también aceptar que el campo de aplicación sería más restringido. Otra vez aparecían los aspectos económicos. La pregunta que flotaba en el aire era: ¿cómo hacer un dispositivo lo suficientemente "grande" como para permitir la construcción de sistemas electrónicos lo suficientemente complejos, pero a la vez que resultara lo suficientemente estándar para que su precio fuese lo suficientemente bajo?. La respuesta a esta encrucijada la encontró Ted Hoff y la materializó Federico Fagin. Aparecía un componente paradigmático: el microprocesador.

Hooff y Fagin trabajaban en 1971 en una pequeña empresa dedicada al prometedor campo de la electrónica integrada, dos palabras cuyos apócopes daban nombre a la compañía: INTEL. Habían recibido el encargo de realizar un conjunto de chips para una calculadora electrónica. Pronto se dieron cuenta que con la falta de disponibilidad de ingenieros de diseño no podrían terminar el trabajo a tiempo. Sólo había una posibilidad de éxito: diseñar un único circuito, que fuera programable a la manera de los grandes computadores de la época. Así, el mismo chip podría ser usado en las diferentes tareas de la calculadora: leer el teclado, realizar las operaciones, exhibir los resultados, etc. con sólo modificar su programa.

El microprocesador surge como el primer circuito integrado altamente complejo, totalmente estándar y relativamente fácil de utilizar. La combinación complejidad-programable se ha extendido a dos famosos derivados del microprocesador: el procesador digital de señales y el microcontrolador.

El microcontrolador es uno de los componentes actuales más versátil, económico y de mayor campo de aplicación. Por un bajo costo se puede conseguir un chip que integra una CPU, varios temporizadores, puertos de entrada-salida, bloques de comunicación e incluso convertidores analógico-digital. En lo que respecta al software de programación, es sencillo, potente, gratis y prácticamente ilimitado en la World Wide Web, así como con sus mismos fabricantes y distribuidores. El objetivo esencial del dispositivo es la construcción de productos electrónicos fiables en el mínimo tiempo posible.[1]

En la Facultad de Ingeniería, dentro de las asignaturas correspondientes a las carreras de Ingeniería Eléctrica y Electrónica, Ingeniería en Computación e Ingeniería en Telecomunicaciones, existen cursos dedicados al estudio de los microcontroladores, así como de algunas de sus aplicaciones.
El microcontrolador de Motorota 68HC11 es uno de los más populares y favoritos dentro de esta institución y sus aplicaciones prácticamente infinitas, restringidas

[1] Doblado Alcázar Cristina, González Gómez Juan, Prieto-Moreno Andrés y San Martín Juan José, *Microcontrolador MC68HC11, Fundamentos, Recursos y Programación,* U.A.M., España 1997.

únicamente por su propia estructura interna y el conocimiento de su manejo por parte de quien lo utilice.

Este proyecto de tesis tiene como objetivo fundamental crear un sistema portátil que sea capaz de realizar lecturas de códigos de barras en general a través del microcontrolador 68HC11, almacenarlos en memoria para posteriormente trasmitirlos a una computadora personal que será la encargada de procesar dicha información y, dependiendo de la aplicación, se puede ajustar la lectura de los diferentes códigos de barras. Obteniendo una aplicación práctica y funcional que sea adaptable a cualquier requerimiento relacionado: control de accesos, inventarios, etc., con la ventaja de utilizar el mismo dispositivo de almacenamiento para diferentes aplicaciones, de esta manera se tendrá un sistema portátil, sencillo y económico en las aplicaciones de lectura de códigos de barras.

En la actualidad los sistemas portátiles son cada vez más utilizados. Un ejemplo de esto es el desarrollo de los teléfonos celulares. Antiguamente se utilizaban grandes aparatos telefónicos, resultando su manejo un poco engorroso debido a su tamaño. Hoy en día dichos teléfonos se encuentran en el mercado en un tamaño mucho menor a la mitad y con más funciones que los primeros teléfonos celulares. Esto nos indica un desarrollo increíble de estos sistemas en un tiempo muy corto. Estos sistemas portátiles suelen ser conocidos con el nombre *"hand-held"*, aunque dicho nombre se relaciona más no con teléfonos celulares, sino con dispositivos o computadoras lo suficientemente pequeñas para ser sostenidas en las manos y poder llevarlas fácilmente a cualquier lugar.

El sistema desarrollado en este proyecto cuenta con la facilidad de realizar la función específica de recolección de datos para control de accesos de una forma más eficiente que los sistemas existentes, ya que dicho control se realizará al entrar en cualquier lugar en dónde se necesite registrar la entrada no en la recepción de la dependencia o empresa, sino en la misma entrada principal de ella, por lo que se ahorrará tiempo en la ejecución del registro.

Por otro lado, en relación directa con el objetivo de la tesis, los puertos de comunicación de la PC son de particular interés para el estudioso de la electrónica ya que le permiten utilizar una computadora personal para controlar todo tipo circuitos electrónicos, principalmente en actividades de automatización de procesos, adquisición de datos, tareas repetitivas y otras actividades que demandan precisión. Es por esto que durante el desarrollo del proyecto se describe el funcionamiento de estos puertos y también se hace uso de ellos.

El puerto serial se utiliza para realizar el traslado de información del dispositivo portátil a la computadora principal la cual se encargará de procesar los datos recibidos a través de dicho puerto.

La estructura general de esta tesis consta de introducción, cuatro capítulos y conclusiones:

La introducción da una descripción general de la problemática así como de la consecuente revisión de la estructura de la tesis para la solución de la lectura, almacenamiento y trasmisión de los códigos de barras.

En el capítulo 1, se explican los diferentes tipos de códigos de barras existentes y se proporciona un esbozo de los lectores utilizados durante el desarrollo de la tesis.

A su vez en el capítulo 2 se da una descripción de la arquitectura y funcionamiento del microcontrolador Motorola M68HC11.

En el capítulo 3 se describen los componentes básicos de una computadora detallando los periféricos de entrada / salida, así como uso el uso de ellos como una herramienta para la comunicación y control de la aplicación desarrollada en esta tesis.

En el capítulo 4, se da una descripción detallada de todo el diseño, desarrollo e implementación del proyecto. Se explica claramente los componentes utilizados y su interconexión con el software con que se cuenta para el desarrollo de proyectos con el microcontrolador.

Finalmente, se cuenta con una retroalimentación en forma de conclusiones que engloban las funciones del proyecto y las futuras correcciones o adaptaciones que se le podrían hacer al mismo para futuros usos.

CAPÍTULO 1.

CÓDIGOS DE BARRAS.

1.1. Introducción.

Los códigos de barras se han integrado en cada aspecto de nuestras vidas, se localizan en el supermercado, en tiendas departamentales, farmacias, etc. Han sido aceptados como parte de nuestra vida diaria sin saber qué es lo que representan. Las barras y espacios aparecen impresos en etiquetas de alimentos, paquetes de envío, brazaletes de pacientes, etc. Podría parecer que todas son iguales, pero no es así. Cada tipo de industria tiene una simbología que maneja como su propio estándar para el fácil manejo de sus productos. Si se instala un sistema de manejo de datos a través de códigos de barras, hay aspectos a considerar para hacer la selección correcta y adecuada a las necesidades de cada negocio.

No se requiere de gran conocimiento técnico para entenderlos, sólo son una forma diferente de codificar números y letras usando una combinación de barras y espacios en diferentes medidas. Solamente se trata de otra manera de escritura, ya que reemplazan el tecleo de datos para recolectar información. En las empresas, el uso correcto de los códigos de barras reduce la ineficiencia y mejora la productividad de la compañía hacia un crecimiento. Simplemente, los códigos de barras son una forma fácil, rápida y precisa de codificar información.

Para entender más sobre esta forma de codificación es necesario entender una serie de conceptos básicos que se describen a continuación para una mejor compresión de la aplicación de esta tecnología.

1.2. Códigos de barras.

> ***Código de barras:*** *técnica de entrada de datos por reconocimiento óptico de una clave asignada a un archivo de datos específico.*

¿Qué es un código de barras?

El Código de Barras es una disposición en paralelo de barras y espacios que contienen información codificada.[1]

Una barra predefinida y patrones de espacio o "simbologías" se usan para codificar pequeñas bandas de caracteres de datos dentro de un símbolo impreso. Los códigos de barras se pueden imaginar como si fueran la versión impresa del

[1] http://www.codigodebarras.com/que_es.htm

código Morse, con barras angostas (y espacios) representando puntos, y barras anchas que representan rayas.

La estructura básica de un código de barras consiste de zona de inicio y término en la que se incluye (ver figura 1.1): un patrón de inicio, uno o más caracteres de datos, opcionalmente unos o dos caracteres de verificación y un patrón de término.

Fig. 1.1. Estructura básica de un código de barras.

La información es leída por dispositivos ópticos los cuales envían la información a una computadora como si la información hubiese sido tecleada. ***Un símbolo*** *de código de barras es la* ***visualización*** *física* de un código de barras. Una ***simbología*** *es* ***la forma*** *en que se codifica la información en las barras y espacios del símbolo* de código de barras.

El código de barras almacena datos que pueden ser reunidos de manera rápida y con una gran precisión. Los códigos de barras ofrecen con un método simple y fácil la codificación de información de texto que puede ser leída por lectores electrónicos de bajo costo.

1.2.1. Simbología en código de barras.

La simbología es considerada un lenguaje en la tecnología de códigos de barras. Una simbología permite a un escáner y al código de barras comunicarse. Cuando un código de barras es digitalizado, su simbología permite que la información se lea de manera precisa y a su vez, permite a una impresora comprender la información que necesita ser turnada dentro de una etiqueta al instante de imprimir dicho código.

1.2.2. Tipos de simbologías para códigos de barras.

Se puede decir que los códigos de barras vienen en muchas formas o presentaciones. Muchos son familiares porque los hemos visto en las tiendas, pero existen algunos otros que son estándares en varias industrias. La industria de la salud, manufactureras, almacenes, etc. tienen terminologías únicas para su industria y que no son intercambiables. La razón de que existan muchos tipos de

códigos de barras es simplemente porque las simbologías están diseñadas para resolver problemas específicos. [2]

1.2.3. Breve historia de los códigos de barras. [3]

El primer sistema de código de barras fue patentado en Octubre 20 de 1949 por Joseph Norman Woodland y Bernard Silver. Se trataba de un "blanco" (bull's eye code) hecho mediante una serie de círculos concéntricos. Una banda transportaba los productos a ser leídos por un fotodetector.

En 1961 aparece el primer escáner fijo de códigos de barras instalado por *Sylvania General Telephone*. Este aparato leía barras de colores rojo, azul, blanco y negro identificando vagones de ferrocarriles.

Para 1967 la Asociación de Ferrocarriles de Norteamérica aplica códigos de barras para control de tránsito de embarques. El proyecto no duró mucho por falta de adecuado mantenimiento de las etiquetas conteniendo los códigos.
En ese mismo año, la sucursal de Cincinnati (Ohio, E.U.A.) de la cadena de supermercados Kroger instala el primer sistema de venta por menudeo basado en códigos de barras. Al cliente que encontraba un código que no se podía escanear correctamente se le ofrecían cupones de compra gratis.

En 1969 el láser hace su aparición. Usando luz de gas de Helio-Neón, el primer escáner fijo es instalado con un costo de 10,000 USD. Hoy por hoy el mismo tipo de escáner estaría costando menos de 2,000 USD.

A fines de los años 60 y comienzos de los 70 aparecieron las primeras aplicaciones industriales pero sólo para manejo de información. En 1969, Rust-Oleum fue el primero en interactuar un lector de códigos con una computadora. El programa ejecutaba funciones de mantenimiento de inventarios e impresión de reportes de embarque.

En 1970 aparece el primer terminal portátil de datos fabricado por Norand. Éste utilizaba un lápiz de contacto.
El código Plessey hace su aparición en Inglaterra (The Plessey Company, Dorset, Inglaterra), para control de archivos en organismos militares en 1971. Su aplicación se difundió para control de documentos en bibliotecas.

El codabar (figura 1.2) aparece en 1971 y encuentra su mayor aplicación en los bancos de sangre, donde un medio de identificación y verificación automática eran indispensables.

[2] http://www.taltech.com/TALtech_web/resources/intro_to_bc/bcbascs.htm
[3] http://www.ent.ohiou.edu/~amable/autoid/history.htm

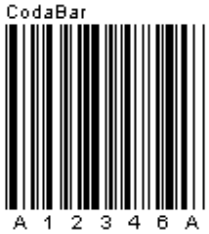


Fig. 1.2. Código de barras CodaBar.

La Buick (fábrica de automóviles) utilizó identificación automática en las operaciones de ensamble de transmisiones, también por los años 70. El sistema era utilizado para conteo de los diferentes tipos de transmisión ensamblados diariamente.

La ITF (Interleaved Two of Five, figura 1.3) marca su aparición en 1972, creado por el Dr. David Allais en Intermec.

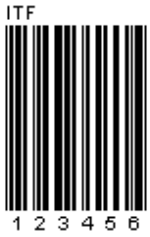


Fig. 1.3. Código de barras ITF.

En el año 1973 se anuncia el código U.P.C. (Universal Product Code) que se convertiría en el estándar de identificación de productos elaborado por George J. Laurer. De esta forma la actualización automática de inventarios permitía una mejor y más oportuna compra y reabastecimiento de bienes. Europa se hace presente con su propia versión de U.P.C. en 1976, el código EAN (European Article Number). La figura 1.4 muestra ejemplos de estos códigos.

Fig. 1.4. Código de barras UPC-A, UPC-E, EAN-8 y EAN-13.

En 1974, nuevamente el Dr. Allais conjuntamente con Ray Stevens de Intermec inventan el Código 39, el primero de tipo alfanumérico (figura 1.5).

Fig. 1.5. Código de barras Código 39.

El primer sistema patentado de verificación de códigos de barras por medio de láser aparece en el mercado en 1978.

En la figura 1.6 se muestra un símbolo de PostNet, que aparece en 1980 siendo usado por el Servicio Postal de los E.U.A.

Fig. 1.6. Código de barras PostNet.

La tecnología de CCD (Charge Coupled Device) es aplicada en un escáner a partir de 1981. En la actualidad este tipo de tecnología tiene bastante difusión en el mercado asiático, mientras que el láser domina en el mundo occidental. En ese año también aparece el código 128, de tipo alfanumérico (figura 1.7).

Fig. 1.7. Código de barras Código 128.

Aparece la norma ANSI MH10.8M que especifica las características técnicas de los códigos 39, Codabar, e ITF (Interleaved Two of Five).

En 1987 el Dr. Allais desarrolla el primer código bidimensional, el código 49. Le sigue Ted Williams (Laser Light Systems) con el código 16K (1988).

En 1990 se publica la especificación ANS X3.182, que regula la calidad de impresión de códigos de barras lineales. En ese mismo año, Symbol Technologies presenta el código bidimensional PDF417 (figura 1.8).

Fig. 1.8. Código de barras PDF 417.

1.3. Tipos de códigos de barras. [4]

Existe una gran variedad de simbologías para codificar diferentes tipos de códigos de barras, cada una de las cuales fue originalmente desarrollada para satisfacer una necesidad específica en una industria específica. Muchas de estas simbologías han sido manufacturadas dentro de los estándares que son utilizados universalmente el día de hoy en la mayoría de las industrias. Las simbologías provistas por B-Coder, el control de TAL Bar Code Active X y los TAL Bar Code DLLs son los más comúnmente utilizados a través de todas las industrias.

Las diferentes simbologías tienen diferentes capacidades de codificar datos. Por ejemplo, la simbología UPC utilizada para identificar productos al menudeo siempre contiene 12 dígitos numéricos mientras que el propósito general de las simbologías del código 39 o del código 128 pueden codificar una longitud variable de datos alfanuméricos hasta de 30 caracteres. Este tipo de códigos de barras son llamados "simbologías lineales" porque son hechos de una serie de líneas de diferente ancho. Los lectores de códigos de barras más disponibles comercialmente son capaces de leer todas las diferentes simbologías de códigos de barras lineales por lo que uno no necesita diferentes lectores para diferentes tipos de códigos de barras.

Las nuevas simbologías de códigos de barras de "dos dimensiones" como el PDF417, el Código Azteca y el Data Matriz, ahora disponibles y poseen la capacidad de codificar varios miles de bytes de datos en un simple símbolo de código de barras incluyendo texto o datos binarios. Las más recientes simbologías de códigos de barras de dos dimensiones requieren lectores especiales que han sido diseñados específicamente para leer este tipo de códigos.

El principal propósito de un código de barras es el identificar algo por el rotular el elemento con un código de barras conteniendo un único número o cadena de caracteres. Los códigos de barras son típicamente utilizados con una aplicación de bases de datos donde los datos codificados en los códigos de barras son utilizados como un índice para un registro en la base de datos que contiene información más detallada acerca del elemento o producto que está siendo leído por el lector. Por lo que, debido al uso de códigos de barras, una tienda no necesita poner una etiqueta con el precio a cada producto que es vendido en ella y puede incluso cambiar el precio de un producto en específico con la simple modificación de la base de datos central. Es posible incluso saber cuántos

[4] http://www.taltech.com/TALtech_web/resources/intro_to_bc/bcsymbol.htm

productos existen actualmente en la tienda y así poder decidir si es necesario ordenar más productos de este tipo debido a su existencia en el establecimiento.

El código de barras también provee disminución de error al introducir datos en una aplicación que esté siendo ejecutada en una computadora. Por el uso de códigos de barras, los errores potenciales por el manejo manual de datos de entrada son eliminados. Otra aplicación típica de los códigos de barras es, por la razón mencionada, el introducir datos sin tener que teclearlos. Por ejemplo se pueden codificar nombres o direcciones en códigos de barras en una insignia de identificación y entonces leer estas placas de identificación para ingresar los nombres de personas en un programa de computadora en lugar de teclear la información.

Las diferentes simbologías de códigos de barras soportan diferentes tipos y cantidades de datos para que uno normalmente escoja una simbología particular basada en el tipo y cantidad de datos que uno quiere codificar en sus códigos de barras. Uno es generalmente libre de utilizar cualquier tipo de código de barras que prefiera y codificar cualquier tipo de dato que uno necesita para sus aplicaciones en un sistema cerrado.

A continuación se realiza una descripción detallada de las simbologías de códigos de barras más utilizadas comúnmente. Todos los siguientes tipos de códigos de barras son sostenidos por los controladores B-Coder Pro, TAL Bar Code ActiveX y los TAL Bar Code DLLs contenidos normalmente en la mayoría de los lectores de códigos de barras.

1.3.1. CÓDIGO 39 (versiones Normal y ASCII).

Este código se desarrolló porque algunas industrias necesitaban codificar el alfabeto así como también números en un código de barras. Generalmente se utiliza para identificar inventarios y para propósitos de seguimiento en las industrias. Sin embargo puede producir una barra relativamente larga y no puede ser adecuada si la longitud es un factor de consideración.

El Código 39 Normal es una simbología de longitud variable que puede codificar los 44 siguientes caracteres: 1234567890ABCDEFGHIJKLMNOPQRSTU VWXYZ -.*$/+%. Este código es el más popular y es utilizado en gran medida en manufactura y en aplicaciones militares y de salud. Cada Código 39 es enmarcado con caracteres de inicio – fin representados por un asterisco (*). El asterisco es el carácter reservado para este propósito y no debe ser utilizado en el cuerpo del mensaje. El B-Coder automáticamente añade el carácter de inicio – fin a cada código de barras por lo que uno no debe incluirlos como parte del mensaje del

código de barras. Si uno selecciona la versión NORMAL del Código 39 y el texto del código de barras contiene caracteres en minúsculas, el B-Coder los convertirá automáticamente en mayúsculas. Si el mensaje del código de barras contiene caracteres inválidos, el B-Coder avisará con un mensaje de advertencia (si la opción de mensajes de caracteres inválidos es seleccionada en el menú de Preferencias).

Otra característica que el Código 39 permite es la concatenación de dos o más códigos de barras. En ocasiones ésta es una ventaja para romper largos mensajes en símbolos múltiples y más cortos.
Si el primer carácter de entrada de un símbolo de Código 39 es un espacio, algunos lectores mantendrán el dato en un buffer y no transmitirán el dato. Esta operación continúa para el resto de los símbolos de Código 39 que sean antecedidos por un espacio debido a que cada mensaje es añadido al símbolo anterior.
Cuando un mensaje sin un espacio antecesor es leído, éste es añadido a los datos previamente leídos en el buffer y el buffer completo es transmitido como un mensaje largo.

La versión ASCII del Código 39 es un modificación de la versión estándar Normal que codifica los 128 caracteres del código ASCII (incluyendo los asteriscos). La versión completa ASCII es implementada al utilizar los cuatro caracteres $/+% como caracteres de cambio para modificar el significado del resto de los caracteres en el conjunto Normal del Código 39. Debido a que la versión completa ASCII utiliza caracteres de cambio en combinación con otros caracteres estandarizados para la representación de datos que no se incluyen en el conjunto de caracteres del Código 39 Normal, cada carácter no estandarizado requiere el doble de ancho de caracteres estándares en un símbolo impreso. [5]

1.3.2. UPC-A, UPC-E y suplementos UPC.

UPC-A con suplemento UPC-E.

UPC-A es una simbología numérica de 12 dígito. Los símbolos de UPC-A consisten en 11 dígitos de datos y un dígito de revisión.
UPC-E es una simbología más pequeña de siete dígitos para sistemas numéricos. Generalmente utilizado para elementos pequeños.

[5] *Nota:* Debido a que todos los caracteres utilizados para implementar el Código 39 ASCII son parte del conjunto del Código 39 Normal, los lectores que no soportan la versión ASCII del Código 39 son capaces de leerlo. El lector introducirá caracteres de cambio como si se tratara de caracteres del Código 39 Normal.

Ambos, el UPC-A y el UPC-E permiten el suplemento de dos o cinco dígitos numéricos para ser concatenados al símbolo principal del código de barras. Este mensaje de suplemento fue diseñado para su uso en publicaciones y periódicos. Si uno ingresa un mensaje suplementario, éste debe consistir de cualesquiera dos o cinco dígitos numéricos. El suplemento es un sencillo y pequeño código de barras adicional que es añadido en el lado derecho de un símbolo UPC estándar.

Diferencias entre el tipo A y el tipo E.

UPC-E es también llamado “UPC cero suprimido” debido a que UPC-E comprime un número UPC-A de 12 dígitos normal en un código de seis dígitos a través de una supresión el sistema digital numérico, arrastrando ceros en el código de manufactura y dejando en primer lugar ceros en la parte de identificación del producto del mensaje de código de barras. Un séptimo dígito de verificación es colocado en el patrón de paridad para los principales seis dígitos. UPC-E puede entonces ser descomprimido de regreso a un UPC-A de 12 dígitos estándar.

La mayoría de los lectores de códigos de barras pueden ser configurados para convertir automáticamente números UPC-E de seis dígitos a UPC-A de doce dígitos antes de que sean transmitidos a una computadora huésped.

La mayor diferencia entre un símbolo UPC-A y uno UPC-E es el tamaño. En la figura 1.9 se muestra un símbolo de código de barras UPC-A a la izquierda y los mismos datos codificados como un símbolo UPC-E a la derecha.

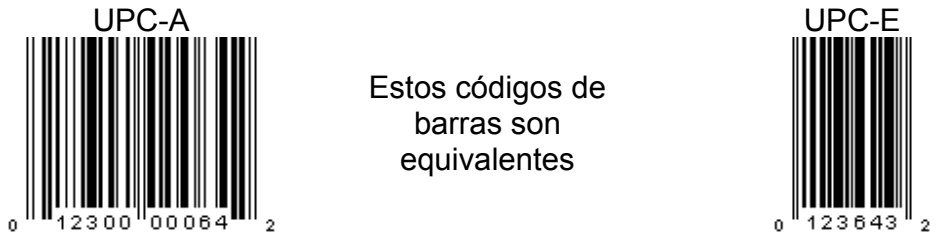


Figura 1.9. Comparación de dos símbolos de códigos de barras UPC-A y UPC-E con el mismo código.

Para convertir códigos de UPC-A y UPC-E se puede utilizar la tabla siguiente, en la cual el número 0 y cada una de las letras a, b, c, d y e representan dígitos individuales en el mensaje de código de barras y la letra X representa el dígito verificador UPC.

Número UPC-A	**Equivalente UPC-E**	**Notas:**
0ab00000cdeX	abcde0X	EL código de manufactura debe tener 2 dígitos al principio con 3 ceros que

		arrastren y el número del producto está limitado a 3 dígitos (000 a 999).
0ab10000cdeX	abcde1X	El código de manufactura debe tener 3 dígitos al inicio terminando con “1” y dos ceros siguientes. El número del producto está limitado a 3 dígitos.
0ab20000cdeX	abcde2X	El código de manufactura debe tener 3 dígitos iniciales terminando con “2” y 2 ceros que arrastren. El número del elemento está limitado a 3 dígitos.
0abc00000deX	abcde3X	El código de manufactura debe tener 3 dígitos iniciales y 2 ceros que arrastren. El número del producto está limitado a dos dígitos (00 a 99).
0abcd00000eX	abcde4X	El código de manufactura debe tener 4 dígitos de inicio con un cero de arrastre, y el número está limitado a un dígito (0 a 9).
0abcde00005X 0abcde00006X 0abcde00007X 0abcde00008X 0abcde00009X	abcde5X abcde6X abcde7X abcde8X abcde9X	El código de manufactura posee todos los 5 dígitos. El número del elemento está limitado a un dígito simple que puede ser cualquiera entre 5, 6, 7, 8 o 9.

Tabla 1.1. Conversión entre simbologías UPC-A y UPC-E.

1.3.3. EAN-8 / EAN-13, BookLand y suplementos EAN.

EAN-8, EAN-13 con suplementos (versión ISBN).

EAN o European Article Numbering (Artículo de Numeración Europeo) – también llamado JAN en Japón – es una versión europea del UPC. Éste utiliza el mismo tamaño de requerimientos y un esquema de codificación similar a los códigos UPC.

EAN-8 codifica 8 dígitos numéricos que consisten en dos como código del país, cinco dígitos de datos y un dígito de verificación. El B-Coder aceptará hasta 7 dígitos del sistema EAN-8. El B-Coder calculará automáticamente el dígito de verificación. Si uno introduce menos de 7 dígitos o si se introducen otros caracteres diferentes a los dígitos del 0 al 9, el B-Coder desplegará un mensaje de

alerta. Si la opción conocida como “Habilitador de alertas de mensajes inválidos” no está seleccionado en el menú de preferencias y no se ingresan 7 dígitos, el B-Coder introducirá ceros en el mensaje y truncará los mensajes más largos de modo que la longitud total sea de 7 dígitos.

EAN-13 es la versión europea de UPC-A. La diferencia entre EAN-13 y UPC-A es que el EAN-13 codifica un decimotercero dígito en el patrón de paridad de los seis dígitos de un símbolo UPC-A. Este decimotercero dígito, combinado con el duodécimo dígito, usualmente representa el código de un país.

Ambos, el EAN-8 y el EAN-13 soportan un número suplementario de dos o cinco dígitos para ser añadido al símbolo de código de barras principal. El suplemento está diseñado para su uso en periódicos y publicaciones. Mensajes suplementarios deben consistir en igual forma de dos o cinco dígitos y aparecerán como un código de barras pequeño y adicional en la parte derecha de un símbolo EAN estándar.

Los números de códigos de barras EAN son asignados a productos específicos y manufactureros por la organización llamada ICOF localizada en Bruselas, Bélgica.

EAN-13 ha sido adoptada como un estándar de la industria publicitaria para codificar números ISBN en libros. Un número ISBN o código de barras BookLand es simplemente un símbolo EAN-13 consistente en los primeros 9 dígitos del número ISBN precedido por los dígitos 978. El suplemento en un código ISBN es el precio de menudeo del libro en cuestión precedido por el dígito 5. Por ejemplo, si el número ISBN es 1-56276-008-4 y el precio del libro es $29.95, entonces se introducirá 978156276008 como el mensaje de código de barras y 52995 para el suplemento.

1.3.4. CODABAR.

Ésta es una simbología de longitud variable que permite la codificación de los siguientes 20 caracteres: 0123456789-$:/.+ABCD. CodaBar es comúnmente utilizada en librerías, bancos de sangre y negocios de paquetería aérea. CodaBar utiliza los caracteres A, B, C y D solamente como los caracteres de inicio y fin. Por esto, los primeros y últimos dígitos de un mensaje CodaBar deben ser A, B, C o D y el cuerpo del mensaje no deberá contener estos caracteres.

El B-Coder aceptará automáticamente cualquier longitud de un mensaje CodaBar siempre y cuando éste contenga caracteres válidos y comience y termine con los caracteres válidos de inicio – fin. Si se utilizan letras minúsculas para estos caracteres, el B-Coder los convertirá en mayúsculas.

1.3.5. ENTRELAZADO 2 DE 5.

El Entrelazado 2 de 5 es una sola simbología de alta densidad de longitud numérica variable que codifica pares de dígitos de una manera intercalada. Los dígitos en posición impar son codificados en las barras y los dígitos en posición par son codificados en los espacios. Debido a esto, los códigos de barras Entrelazados 2 de 5 deben consistir en un número par de dígitos. Incluso, debido a lecturas parciales de estos códigos, tienen una ligera oportunidad de ser decodificados como un código de barras válido pero más pequeño; los lectores usualmente están programados para leer un número para de dígitos cuando leen esta clase de códigos de barras. El número de dígitos usualmente está predefinido para una aplicación particular y todos los lectores utilizados en esta aplicación son programados para únicamente aceptar códigos de barras Entrelazados 2 de 5 de la longitud seleccionada. Datos más cortos pueden ser rellenados con ceros para ajustar en la longitud seleccionada.
La simbología intercalada 2 de 5 permite opcionalmente un módulo de 10 dígitos de ancho para situaciones especiales donde los datos de seguridad son importantes.

1.3.6. DISCRETO 2 DE 5.

Esta simbología es igualmente variable en longitud y muy similar al Entrelazado 2 de 5 excepto que en lugar de codificar datos en las barras y los espacios, los datos solamente son codificados en las barras. Debido a esto, esta simbología no es tan compacta como la anterior e inclusive, los números impares de los dígitos deben ser codificados. El uso de Discreto 2 de 5 no es muy común y pocos lectores de códigos de barras lo reconocen.

1.3.7. CÓDIGO 93.

El Código 93 es una simbología de longitud variable que puede codificar el Código ASCII completo. El Código 93 fue desarrollado como una variante pero mejorada versión del Código 39 al proveer una pero mayor capacidad en cuanto al número de caracteres que el Código 39. El Código 93 también incorpora dos dígitos de verificación así como un mensaje de seguridad. A pesar de que el

Código 93 es considerado más robusto que el Código 39, éste nunca ha alcanzado la misma popularidad que el Código 39. Los códigos de barras del Código 93 son encuadrados por un carácter especial de inicio – fin. El B-Coder añadirá automáticamente estos caracteres de inicio – fin así como los dígitos de verificación para cada Código 93 por lo que uno no debe intentar de incluirlos como parte del mensaje de código de barras.

1.3.8. CÓDIGO 128.

El Código 128 es una simbología alfanumérica de longitud variable y alta densidad. El Código 128 tiene 106 patrones diferentes de barras y espacios y cada patrón tiene uno o tres significados diferentes, dependiendo de cuál de los tres diferentes conjuntos de caracteres se esté utilizando. Caracteres especiales de inicio le dicen al lector cuál de estos conjuntos es utilizado y tres especiales códigos de cambio permiten ir cambiando el conjunto de caracteres usado en un mismo símbolo. Un conjunto de caracteres codifica todas las mayúsculas y caracteres de control del código ASCII, otro codifica todas las mayúsculas y minúsculas y el tercero codifica parejas de dígitos desde el 00 hasta el 99. Este tercer conjunto de caracteres duplica la densidad de código cuando los datos numéricos son impresos. El Código 128 también utiliza un dígito de verificación para seguridad de los datos. En suma al código ASCII, el Código 128 además permite la codificación de cuatro códigos de funciones especiales (FNC1 – FNC4).

La principal función de los códigos FNC1 y FNC4 fue originalmente dejada abierta para propósitos específicos. Recientemente un acuerdo fue realizado por la *Automatic Identification Manufacturers Assoc. (AIM)* – Asociación de Manufactureros de Identificación Automática – y la *European Article Numbering Assoc. (EAN)* – Asociación Europea de Numeración de Artículos – con el propósito de reservar FNC1 para ser utilizado en aplicaciones de la EAN. FNC4 se mantiene disponible para su uso en aplicaciones cercanas de sistemas. FNC2 es utilizado para instruir un lector de código de barras para concatenar mensajes en un símbolo de código de barras con un mensaje en el siguiente símbolo. FNC3 es utilizado para instruir un lector de códigos de barras para realizar una aplicación de inicio. Cuando FNC3 es codificado en cualquier lugar en un símbolo, cualquier dato incluso contenido en este símbolo es descartado.

1.3.9. EAN / UCC 128.

La simbología EAN / UCC 128 es una variante de la simbología original del Código 128 diseñado primariamente para su uso en aplicaciones de identificación de productos. Las especificaciones del Código EAN / UCC 128 utiliza el mismo conjunto de códigos de caracteres que el Código 128 excepto que no permite funciones con los Códigos FNC2 – FNC4 para ser usados en un símbolo y FNC1 usado como parte del código de inicio en el símbolo. El dígito de verificación en los símbolos EAN / UCC 128 es también calculado ligeramente diferente que en el Código 128.

1.3.10. POSTNET.

POSTNET (POSTal Numeric Encoding Technique) – Técnica de Codificación Numérica Postal – es una simbología de códigos de barras de únicamente 5, 9 u 11 dígitos utilizada en el Servicio Postal de E.U.A. para codificar información del Código ZIP (postal) para una automática repartición del correo a través del Código ZIP. El código de barras debe representar un Código ZIP de 5 dígitos (32 barras), un ZIP de nueve dígitos + 4 dígitos (52 barras) u 11 dígitos de un código de entrega (62 barras).

Esta simbología es diferente a todas las anteriores debido a que los datos son codificados en la altura de las barras en lugar del ancho de las barras y espacios. La mayoría de los lectores de códigos de barras estándares no pueden decodificar POSTNET. Esta simbología fue escogida por el Servicio Postal principalmente porque es extremadamente fácil su impresión en prácticamente cualquier tipo de impresora. POSTNET es una simbología arreglada en la dimensión de la altura de las barras, ancho y espacio de cada barra debe estar dentro de tolerancias exactas.
Algunas aplicaciones populares como Microsoft Access y Word poseen herramientas para poder generar dicho código.

Patrones Postales FIM.

Los patrones *FIM (Facing Identification Mark)* son otro tipo de código de barras postal utilizado en correo automatizado en el Servicio Postal de E.U.A. Estos patrones son usados para cancelar automáticamente correo que no contiene estampilla o algún otro requerimiento que impida su entrega. También provee humildes respuestas de cortesía y negocios de otras cartas. Tres patrones FIM son utilizados actualmente. FIM-A es usado en respuestas de cortesía que han sido impresas anteriormente con códigos de barras Postnet. FIM-B es usado en respuestas de negocios, penalidades y correo gubernamental en el que no se ha impreso código de barras Postnet. FIM-C es usado en respuestas de negocios, y correo que ha sido impreso anteriormente con Postnet. Los patrones FIM son colocados en la esquina superior derecha a lo largo del límite superior y dos pulgadas del límite derecho de letras y postales.

1.3.11. PDF 417.

PDF417 es una simbología bidimensional de alta densidad que esencialmente consiste en conjuntos de códigos de barras más pequeños. Esta simbología es capaz de codificar los 255 caracteres del Código ASCII. PDF se refiere a *Portable Data File* – Archivo de datos portátiles debido a que puede codificas tantos como 2,725 caracteres en un solo código de barras. Las especificaciones completas de PDF417 proveen muchas opciones de codificación incluyendo opciones de compactación de datos, detección y corrección de errores y tamaño variable y símbolos de diferente aspecto. La simbología fue publicada por Symbol Technologies, Inc. para completar la necesidad de códigos de barras de mayor densidad. La estructura del nivel inferior de un símbolo PDF417 consiste en un arreglo de palabras codificadas (barra pequeña y patrones de espacio) que son agrupados juntos y apilados en el superior de cada otro para producir el símbolo completo impreso. Una palabra individual codificada consiste en un patrón de 17 módulos de barras y espacios. El usuario puede especificar el ancho del módulo, su altura y el radio en general (altura y ancho) del símbolo completo. Un símbolo PDF417 consiste por lo menos de 3 hileras hasta de 30 palabras codificadas y suelen contener hasta 90 palabras codificadas en hileras por símbolo con un máximo de 928 palabras codificadas por símbolo.

Las palabras codificadas en un símbolo PDF417 son generadas utilizando uno de cada tres datos modos de compactación definidos actualmente en las especificaciones de la simbología. Esto permite más de un carácter para ser codificado en una simple palabra codificada. Debido que diferentes algoritmos de compactación de datos pueden ser utilizados, es posible para diferentes símbolos impresos l ser creados por los mismos datos de entrada. La simbología además permite para variaciones de grados de seguridad de datos o detección y corrección de errores. Nueve diferentes niveles de corrección de errores están disponibles con cada nivel superior adicionando una revisión del símbolo impreso.

1.3.12. BPO 4 CÓDIGOS DE ESTADO (Oficina Postal Británica, Código Postal Real).

BPO*, British Post Office,* – Oficina Postal Británica – Código de 4 Estados es una nueva simbología de códigos de barras postal que ha sido desarrollada por la Oficina Postal Británica para codificar las zonas postales europeas de forma similar a la que la simbología Postnet de E.U.A. codifica los datos del Código ZIP. El objetivo del BPO Código de 4 Estados es el proveer a los países europeos con un esquema de codificación de barras eficiente y simple .

La simbología Postnet de E.U.A. codifica caracteres numéricos en un patrón de cuatro barras por carácter con cada barra pudiendo ser alta o baja (dos posibles estados para cada barra). Esta técnica permite el uso de hasta 16 posibles patrones de barras diferentes para cada conjunto de cuatro barras y está adecuado para codificar diez dígitos (0 a 9).

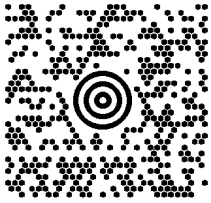
Debido a que los códigos postales europeos contienen caracteres alfanuméricos, lo que requiere un mínimo de 36 posibles patrones de barras diferentes conteniendo de la A a la Z y del 0 al 9, cada carácter en el Código de 4 Estados BPO es codificado en cuatro barras en las que cada una de ellas tiene 4 posibles estados. Estos estados son: barras altas, barras chicas, barras de altura media extendidas hacia arriba desde la parte media del símbolo y barras de altura media extendidas hacia abajo desde la parte media del símbolo.

En teoría, el Código de 4 Estados BPO es capaz de codificar hasta 128 caracteres diferentes siempre y cuando solamente caracteres de la A a la Z y del 0 al 9 sean asignados como patrones únicos de barras.
El Código de 4 Estados BPO es una simbología de dimensión ajustada, queriendo con esto decir que la altura, ancho y espaciamiento de todas las barras deben entrar entre tolerancias exactas.

1.3.13. MATRIZ DE DATOS.

La Matriz de Datos es una simbología de códigos de barras bidimensional de alta densidad estilo matriz que puede codificar hasta 3116 caracteres desde la entera tabla de los 256 bytes del Código ASCII.
El símbolo es construido en una rejilla arreglada con un patrón de rastreo alrededor del perímetro del símbolo del código de barras.
Existen dos tipos de símbolos de Matriz de Datos, cada uno usa diferente esquema de detección y corrección de errores (ECC). Los diferentes tipos de símbolos de Matriz de Datos son identificados utilizando la terminología “ECC” seguida por un número representando el tipo de corrección de error que es utilizado por el software de codificación. ECC 000 a ECC 140 son los símbolos originales de la Matriz de Datos y ahora son considerados obsoletos. La versión más nueva de la Matriz de Datos es llamada ECC 200 y es recomendada para todas las nuevas aplicaciones de Matriz de Datos. La versión ECC 200 de Matriz de Datos usa un algoritmo de codificación de datos en un símbolo mucho más eficiente así como un esquema mucho más avanzado de detección y corrección de errores.

1.3.14. MAXICODE.



MaxiCode es una simbología ajustada a un tamaño de matriz el cual está hecho de un desnivel en las hileras de los módulos hexagonales arreglados alrededor de un blanco como patrón único. Cada símbolo de MaxiCode tiene 884 módulos hexagonales arreglados en 33 hileras cada una conteniendo hasta 30 módulos. La capacidad máxima de datos de un símbolo MaxiCode es de 93 caracteres alfanuméricos o de 138 caracteres numéricos. La simbología fue diseñada por el *United Parcel Service* – Servicio de Paquetería Unida – para aplicaciones de rastreo de paquetes. El diseño de la simbología MaxiCode fue escogida porque es bien ajustable a lectura de alta velocidad y orientación independiente. A pesar de que la capacidad de un símbolo MaxiCode no es tan alta como otras simbologías de códigos de barras matriciales, ésta fue diseñada primeramente para codificar direcciones de datos que raramente requerían más de 80 caracteres. Los símbolos MaxiCode en realidad codifican dos mensajes separados – un mensaje primario y un mensaje secundario. El mensaje primario normalmente codifica un código postal, un código de país de 3 dígitos y un número de servicio de 3 dígitos. El mensaje secundario normalmente codifica datos de direcciones y cualquier otra información requerida.

1.3.15. CÓDIGO AZTECA.

El Código Azteca es una simbología bidimensional de alta densidad estilo matriz que puede codificar hasta 3,750 caracteres del conjunto de 256 bytes del Código ASCII. El símbolo es construido en una rejilla cuadrada con un patrón de blanco en el centro. Los datos son codificados en una serie de capas que circundan alrededor del patrón de blanco. Cada capa adicional completamente circunda la capa anterior causando que el símbolo crezca en tamaño con cada dato codificado hasta que el símbolo quede cuadrado. Las características primarias de esta simbología son: un rango ancho de tamaños permitiendo codificar mensajes pequeños y grandes, lectura orientada independiente y un mecanismo de corrección de errores seleccionado por el usuario.

El elemento más pequeño de un símbolo Azteca es llamado “módulo” (por ejemplo un punto cuadrado). El tamaño del módulo y el conjunto de corrección de errores son las únicas “dimensiones” que pueden ser especificadas por un símbolo Azteca y ambos pueden ser seleccionados por el usuario. Es recomendable que el tamaño del módulo puede estar en el rango de 15 a 30 mils para poder ser leído por la mayoría de los lectores que actualmente están disponibles.

El tamaño del resto de un símbolo Azteca es dependiente del tamaño de los módulos, el conjunto total de datos codificados e incluso del nivel de capacidad de corrección de errores escogido por el usuario. El símbolo Azteca más pequeño es de 15 módulos cuadrados y puede codificar hasta 14 dígitos con una corrección de errores del 40%. El símbolo más grande es uno de 151 módulos cuadrados y puede codificar 3,000 caracteres o 3,750 dígitos numéricos con una corrección de errores del 25%.

1.3.16. MSI / PLESSEY.

MSI – PLESSEY es una simbología de longitud variable pero únicamente numérica. Esta simbología es una de las más recientes simbologías desarrolladas hasta el momento y está basada en un esquema numérico binario de 4 bits. Cada símbolo está enmarcado por un patrón de inicio – fin y contiene un carácter de verificación que es calculado a través de los valores de cada uno de los dígitos de datos codificados. MSI-Plesey es raramente usado en alguna otra aplicación que no esté relacionada con las mercancías de estantes. De hecho, los más modernos lectores de códigos de barras no poseen todavía capacidad de leer esta simbología.

1.4. Exactitud de diferentes simbologías.

Es comúnmente conocido que el operador introductor de datos mejor entrenado cometerá un error de teclado cada 300 movimientos. Cada uno de estas pulsaciones representan y cometen errores en los datos de construcción de los códigos. Esto conduce a la pérdida de tiempo, capital mal utilizado y pérdidas económicas. Es por esta razón que las compañías adoptan tecnologías que eviten estos errores.
Estudios conducidos por la Universidad de Ohio han examinado las simbologías de códigos de barras con el objetivo de determinar la exactitud real de ellas. La peor exactitud en códigos de barras de los examinados probó ser uno de los más comunes – el UPC. El UPC tiene una tasa de error de 1 en 394K caracteres. La mejor simbología probada resultó ser la Matriz de Datos y PDF417, con una tasa de error de 1 en 10.5 M caracteres. Más resultados otorgados por este estudio en la Universidad de Ohio son listados a continuación. [6]

[6] http://www.taltech.com/TALtech_web/resources/intro_to_bc/bcsymbol.htm

Simbología	Peor Caso	Mejor Caso
DataMatrix	1 error en 10.5M	1 error en 612.9M
PDF417	1 error en 10.5M	1 error en 612.4M
Code 128	1 error en 2.8M	1 error en 37M
Code 39	1 error en 1.7M	1 error en 4.5M
UPC	1 error en 394K	1 error en 800K

Tabla 1.2. Tasas de error de simbologías de códigos de barras. [7]

1.5. Cómo se lee un código de barras.

Los códigos de barras se leen mediante un rayo de luz a través de la simbología del código impresa. Lo que sucede mientras solamente se observa una línea roja emitida del escáner láser es que el rayo de luz está siendo absorbido por la oscuridad de las barras y reflejado por los espacios de luz. Un dispositivo, un fotodiodo, en el escáner toma la luz reflejada y la convierte en una señal eléctrica que es exactamente igual al patrón impreso como código de barras.

La fuente de luz del escáner láser comienza a leer el código en el espacio blanco (la zona neutral) y luego la primera barra y continúa pasando por las demás hasta llegar al final en el espacio blanco que le sigue.

Un código no puede leerse si el barrido se desvía fuera del área del símbolo, la altura de las barras se seleccionan para hacer fácil el barrido dentro del área. Lo largo de la información para ser leída es el largo de la información para ser codificada. En tanto que la longitud se incrementa, así también la altura de las barras y espacios para ser leídos.

Debido al diseño de la mayoría de las simbologías de códigos de barras, no existe diferencia alguna si se lee el código de derecha a izquierda o de izquierda a derecha.

Actualmente existen cuatro diferentes tipos de lectores de códigos de barras o escáner que son los que realizan la función descrita anteriormente. Cada uno utiliza ligeramente diferente tecnología para leer y decodificar un código de barras. Existen los lectores tipo pluma (bar code wands), escáner láser, lectores CCD y lectores basados en cámara. [8]

1.5.1. Lectores tipo pluma y escáner láser.

Los lectores tipo pluma consisten en una fuente de luz y un fotodiodo que son colocados cerca uno del otro en la punta de la pluma o de la varilla. Para leer

[7] ibid.
[8] http://www.taltech.com/TALtech_web/resources/intro_to_bc/bcpwork.htm

el código de barras se debe arrastrar la punta de la pluma a través de todas las barras en un movimiento firme. El fotodiodo mide la intensidad de luz reflejada de regreso desde la fuente de luz y genera una forma de onda que es utilizada para medir el ancho de las barras y espacios en un código de barras. Las barras oscuras en el código de barras absorben la luz y los espacios en blanco reflejan la luz así que el voltaje en la forma de onda generada por el fotodiodo es un duplicado exacto de el patrón de barras y espacios en el código de barras. Esta forma de onda es decodificada por el escáner de manera similar a la forma en que se decodifican los puntos y rayas del código Morse.

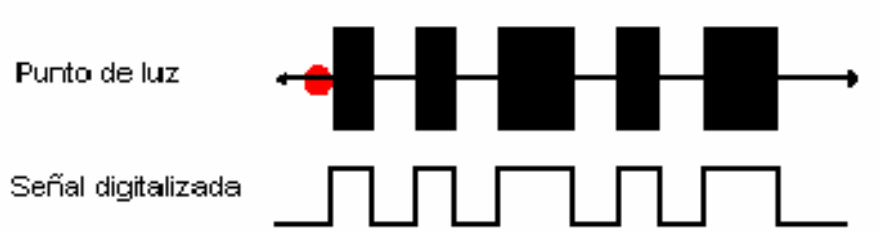


Fig. 1.10. Simulación de lectura de un código de barras y su señal digitalizada.

El escáner láser trabaja de la misma manera que un lector tipo pluma con excepción de que utiliza un rayo láser como la fuente de luz y típicamente emplean un espejo reflector o un prisma rotatorio para leer el rayo láser de regreso y por delante del código de barras. En ambos, el lector de tipo pluma y el escáner láser, la luz emitida por el lector es regresada a una frecuencia específica y un fotodiodo es designado para detectar únicamente este tipo de frecuencia luminosa.

El lector tipo pluma y el escáner láser pueden ser adquiridos con diferente resolución para seleccionarlos para leer códigos de barras de diferentes tamaños. La resolución del escáner es medido por el tamaño del punto de luz emitido por el lector. El punto de luz debe ser igual o ligeramente más pequeño que el elemento de más angosto (dimensión “X”). Si el punto es más ancho que la más delgada barra o espacio, entonces el punto podrá sobreponer dos o más barras al mismo tiempo causando que el escáner no tenga la capacidad de distinguir transiciones limpias entre barras y espacios. Si el punto es demasiado pequeño, entonces cualquier punto o blanco en las barras podrían ser interpretados en forma incorrecta como áreas de luz inclusive pudiendo hacer al código de barras no legible. La dimensión X más utilizada comúnmente es de 13 mils (prácticamente 4 puntos de impresión en una impresora 300 DPI). Debido a que esta dimensión X es tan pequeña, es extremadamente importante que el código de barras sea creado con un programa que cree gráficos de alta resolución (como el B-Coder). [9]

[9] Ibid.

1.5.2. Lectores CCD.

Los lectores CCD (Charge Coupled Device) utilizan un arreglo de cientos de pequeños sensores de luz alineados en una fila en la cabeza del lector. Cada sensor puede ser idealizado como un simple fotodiodo que mide la intensidad de luz que se encuentre exactamente enfrente de él. Cada sensor de luz en el lector CCD es extremadamente pequeño y debido a que hay cientos de sensores alineados en una fila, un patrón de voltaje idéntico al patrón en el código de barras es generado por la medición secuencial de voltajes a través de cada sensor en la fila. La diferencia importante entre un lector CCD y un lector de pluma o láser es que el lector CCD está midiendo la luz ambiente emitida desde el código de barras mientras que los otros dos están midiendo la luz reflejada de una frecuencia específica originada del escáner mismo. [10]

1.5.3. Lectores basados en cámaras.

El cuarto y más reciente tipo de lector de códigos de barras actualmente disponible son los lectores basados en cámaras que utilizan una pequeña cámara de video que captura una imagen de un código de barras. El lector utiliza entonces sofisticadas técnicas de procesamiento de imágenes para decodificar este código de barras. Las cámaras de video utilizan tecnología CCD como en un lector de códigos de barras CCD excepto que en lugar de tener simples filas de sensores una cámara de video tiene cientos de hileras de sensores acomodados en un arreglo de dos dimensiones para que pueden generar una imagen.
Los factores que hacen un código de barras legible son: un adecuado contraste de impresión entre las barras oscuras y blancas teniendo todas las barras dimensiones dentro de las tolerancias de su simbología. Es también de gran ayuda tener límites bien definidos, pocos o ningún punto en blanco o rayas, una superficie lisa y limpia con márgenes bien delineados en un símbolo impreso. [11]

1.6. El lector Hewlett – Packard HBCS-A300.

El lector de códigos de barras tipo lápiz o pluma Hewlett-Packard HBCS-A300 tiene una salida serial por tres conductores que consiste en dos líneas de polarización y una línea por donde viaja la señal. Cuando se polariza con +5 Vdc y tierra, el conductor de la señal puede ser levantada con una resistencia del riel positivo para que se vuelva una salida compatible con la lógica TTL. Al mover el lápiz sobre un patrón de código de barras, la señal de salida del lápiz es alta para las barras blancas y baja para las barras negras. [12]

[10] Ibid.
[11] Ibid.
[12] http://www.linuxdevices.com/articles/AT2281789509.html

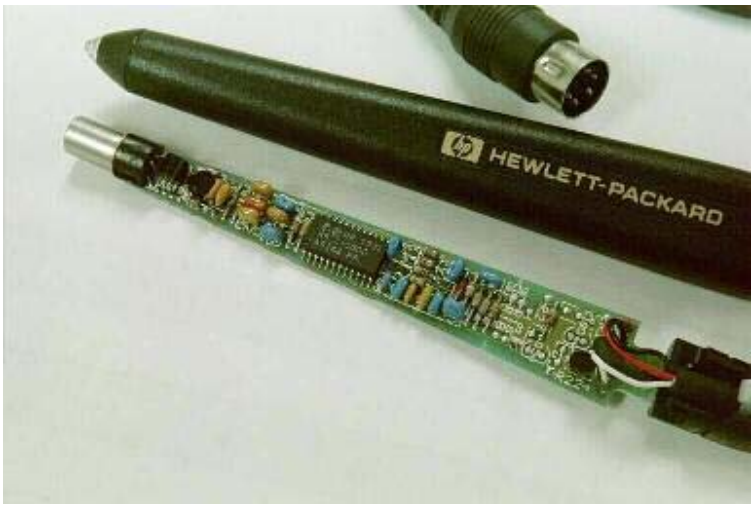


Figura 1.11. El lector de códigos de barras tipo lápiz Hewlett-Packard HBCS-A300.

El cable del lector está equipado con un conector DIN que Hewlett-Packard describe como un conector de 5 pines. Se trata realmente de un conector DIN de 6 pines con el sexto pin central omitido. Existe otro conector de 5 pines utilizado para los teclados PC/AT en el cual los pines se encuentran en un arreglo en forma de arco de 180 grados, mientras que el conector de el lector de códigos de barras posee los pines colocados en un arreglo de igual forma pero de 240 grados.

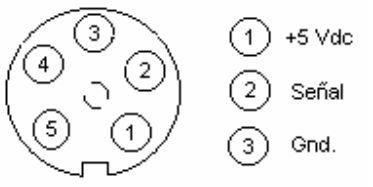


Fig. 1.12. Vista transversal del conector del HBCS-A300

1.7. El lector Voyager MS-9500 de Metrologic instruments

Este lector posee una forma de leer datos que es muy versátil y útil para cualquier tipo de aplicación como control de inventarios, escaneo de menús, códigos de ventas o procesamiento de documentos, por ejemplo, o el que nos ocupa, control de accesos.

Con tan sólo exponer un código de barras frente al lector se activa el láser que permite al usuario seleccionar fácilmente el código a ser escaneado y los datos son transmitidos al sistema host.

Fig. 1.13. El lector de códigos de barras Metrologic Voyager MS-9500.

Este lector puede ser utilizado de dos formas: en forma fija o tipo “hand held” que permite llevar manualmente al lector por dónde se necesite. Esta doble función permite bien conectar el lector a una PC y activar la aplicación necesaria para procesar la información en forma directa; o bien, tomar el lector y llevarlo a donde se necesite tomar la lectura. Esto permite escanear códigos de barras que se encuentren el lugares de difícil acceso como podría ser un código de barras colocado en la parte posterior de un mueble de gran tamaño o de gran peso. [13]

Al igual que el lector de códigos de barras mencionado en la sección anterior tipo pluma Hewlett-Packard HBCS-A300, el lector Metrologic Voyager MS-9500 transmite los códigos leídos a través de un conector DIN, lo cual envía la información de igual manera a como lo realiza un teclado de cualquier computadora personal. De hecho, posee la facilidad de contar con dos conectores de tal forma que el teclado se puede conectar al lector de códigos de barras y éste a la computadora, y así, tener ambos dispositivos conectados y utilizarlos en forma indiferente.

En este proyecto se han utilizado ambos lectores, usando el primero para conocer su funcionamiento y simular la trasmisión de información, y el segundo el utilizado en forma definitiva en el proyecto final, por lo que la presentación final y el estudio en sí se realiza sobre el lector Metrologic Voyager MS-9500.

[13] Metrologic’s MS-9500 Voyager series datasheet; http://www.metrologic.com

CAPÍTULO 2.

EL MICROCONTROLADOR MOTOROLA M68HC11.

2.1. Introducción.

En el mercado existen diferentes microcontroladores que pueden resolver el problema planteado en esta tesis, un ejemplo son los microcontroladores COPSAXX de National o los 80XXX de Philips, los cuales utilizan una arquitectura similar a los microcontroladores de Motorola, (Arquitectura Harvard). También existen los microcontroladores de Microchip, denominados PICs. (Arquitectura Von Neumann) los cuales tiene como características principales que los datos y las instrucciones se almacenan en una sola memoria de lectura – escritura, los contenidos de esta memoria se direcciona indicando la posición, sin considerar el tipo de dato contenido en la misma, de esta forma, la ejecución se produce siguiendo una secuencia de instrucción tras instrucción. Este microcontrolador utiliza tecnología RISC.

Podría resultar muy atractivo el uso de cualquiera de estos microcontroladores, sin embargo, en la Facultad de Ingeniería de la UNAM se imparte un curso de *microcontroladores y microprocesadores* donde se estudia el funcionamiento del microcontrolador M68HC11. Es cierto también mencionar que dentro de la asignatura de *diseño de sistemas con microprocesadores* se estudia el funcionamiento de los PICs de Microchip, sin embargo, se contaba con un tarjeta de desarrollo basada en el M68HC11 y es de interés nuestro crear una contribución que permita fomentar el uso de este microcontrolador.

En este capítulo se realiza una explicación del funcionamiento del microcontrolador de Motorola M68HC11 en su versión E9. Esta descripción se basa principalmente en la interfase de comunicación asíncrona del microcontrolador, sus registros, así como de la memoria EEPROM que es utilizada para albergar el programa fuente de este proyecto. Para una consulta más detallada tanto de la arquitectura como de la estructura interna del microcontrolador ver el apéndice **A** o en su defecto el manual de usuario editado por la misma compañía manufacturera del microcontrolador.

Salvo algunas pequeñas diferencias, la familia entera de estos microcontroladores, poseen las mismas características. Esto permite que sea muy uniforme el uso de estos dispositivos y que esta explicación resulte práctica para casi cualquier miembro de esta familia.

Estas pequeñas diferencias consisten, por ejemplo, en el tamaño de la memoria EEPROM, que en la versión E2 es de 2048 bytes y en la mayoría de los demás es de 512 bytes, cuando cuentan con este recurso.
Cabe mencionar también que la mayoría de estos microcontroladores posee 5 puertos de entrada / salida para las diferentes aplicaciones que se realicen,

mientras que la versión F1 posee 7 puertos que le dan un poco más de versatilidad al momento de buscar implementaciones complejas.

En este proyecto se ha utilizado un microcontrolador versión MC68HC11E9 por lo que una descripción general de este dispositivo se vuelve necesaria e imprescindible.

2.2. ¿Qué es un microcontrolador?

Un microcontrolador (MCU) es un circuito integrado que incorpora una unidad central de proceso (CPU) y cuenta con una serie de recursos internos. La CPU permite que el microcontrolador pueda ejecutar instrucciones almacenadas en una memoria. Sus recursos internos son: memoria RAM, memoria ROM, memoria EEPROM, puerto serie, puertos de entrada – salida, temporizadores, comparadores, etc.

Se puede decir que el MCU es una evolución del microprocesador, con la diferencia de añadirle a éste último las funciones que antes era necesario situar externamente con otros circuitos. El ejemplo típico está en los puertos de entrada - salida y en la memoria RAM, en los sistemas con microprocesadores es necesario desarrollar una lógica de control y unos circuitos para implementar las funciones anteriores, con un microcontrolador no hace falta tal desarrollo porque lo lleva todo incorporado, además en el caso de tener que ampliar el sistema ya ofrece recursos que facilitara su aplicación.

En resumen, un microcontrolador es un circuito integrado, el cual facilita la tarea de diseño y reducción del espacio ya que integra la CPU y todos los periféricos necesarios, traduciéndose todo a tener una aplicación final más económica y fiable. [15]

2.3. El microcontrolador M68HC11 de Motorola.

La familia de microcontroladores de Motorola 68HC11E combina todas las capacidades del CPU M68HC11 con los mismos periféricos. La serie E comprende varias y diferentes configuraciones de RAM, ROM o EPROM y EEPROM. Con excepción de diferencias pequeñas, pero las operación de los microcontroladores de la serie E es idéntica.

[15] Doblado Alcázar Cristina, González Gómez Juan, Prieto-Moreno Andrés y San Martín Juan José, *Microcontrolador MC68HC11, Fundamentos, Recursos y Programación,* U.A.M., España 1997.

Características del M68HC11E9:

- CPU M68HC11
- Capacidad para dispositivos de voltaje bajo (3.0 a 5.5 Vdc)
- 512 bytes de memoria RAM en el mismo chip
- 512 bytes de memoria EEPROM en el mismo chip
- 12 Kbytes de memoria ROM en el mismo chip
- Interfaz de comunicación serial asíncrona de no-retorno a cero (NRZ) – non return to zero – (SCI)
- Interfaz periférica serial síncrona (SPI)
- Convertidor analógico – digital (A/D) de 8 canales de 8 bits
- Acumulador de pulsos de 8 bits
- Circuito de interrupción en tiempo real [16]

2.4. Transmisión serial asíncrona, SCI (Serial Communications Interface).

El microcontrolador 68HC11 dispone de una unidad de comunicaciones serie (SCI) que permite realizar comunicaciones asíncronas a distintas velocidades y con paquetes de 8 y 9 bits de datos con un formato de transmisión NRZ (non return to zero) y cada paquete acompañado de un bit de inicio y un bit de final.

El SCI está formado por una unidad de transmisión y una unidad de recepción que son totalmente independientes, lo que permite que las comunicaciones sean bilaterales, es decir, se puede transmitir y recibir a la vez (modo Full-duplex).

La unidad de transmisión está formada por un registro de desplazamiento con carga en paralelo llamado “registro de transmisión”. Al introducir un valor en este registro, comienza a desplazarse hacia la derecha el contenido del registro, enviando los bits por la línea serie, a una velocidad configurable por el usuario.

De la misma manera la unidad de recepción dispone de otro registro, “registro de recepción” que recibe los bits en serie y los va desplazando hasta obtener un dato en paralelo que puede ser leído.

Tanto el registro de transmisión como el de recepción están mapeados en la misma dirección de memoria. Al escribir en esa dirección de memoria, el dato se cargará en el registro de transmisión. Al efectuar una lectura, el dato se leerá del registro de recepción. Ambos registros comparten la misma dirección física de memoria pero se trata de dos registros diferentes. Puesto que ambos registros comparten la misma dirección física, se les ha asignado un único nombre: registro de datos (SCDR). [17]

[16] Motorola, *M68HC11 Reference Manual*, Rev. 3, 1996.
[17] Ibid.

2.4.1. Registros del SCI. [18]

El SCI dispone de 5 registros mapeados en memoria. Estos registros son los siguientes:
Cuatro registros de control y estado:

- SCCR1 – Serial communications control register 1
- SCCR2 – Serial communications control register 2
- BAUD – Baud rate register
- SCSR – Serial communications status register

Y un registro de datos:

- SCDR – Serial communications data register

2.4.1.1. Registro de control 1. SCCR1.

El registro de control SCCR1 provee los bits de control que determinan la longitud de la palabra y seleccionan el método usado para la capacidad de wake up.

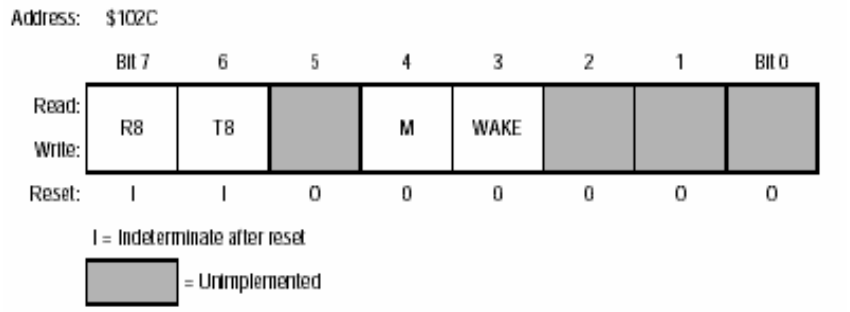


Figura 2.1. Registro 1 de control de comunicación serial.
(Serial Communications Control Register 1, SCCR1).

El bit 4 (M) permite configurar el SCI para utilizar 8 ó 9 bits de datos. Si M=1 implica 9 bits de datos, si M=0 implica 8 bits de datos. Lo normal es utilizar 8 bits de datos. Para comunicarse con una PC el bit se deberá poner a cero puesto que la PC no permite más de 8 bits de datos. Por defecto está en cero. Si se quieren transmitir 9 bits de datos, el noveno bit se escribe en el bit 6 del SCCR1 (T8) y los 8 bits menos significativos se escriben en el registro de datos. Análogamente, si se quiere recibir un dato de 9 bits, el bit más significativo (el noveno) se sitúa en el bit 7 del registro SCCR1 (R8) y los 8 bits restantes en el registro de datos.

El SCI tiene un modo de funcionamiento especial (modo Wake Up) para aumentar la eficiencia en sistemas multireceptores. Es el modo en le cual el receptor se queda con las interrupciones inhibidas esperando un evento hardware externo, (asociado a la línea de recepción), que le devuelva al estado activo con interrupciones. El evento externo puede ser de dos tipos y se selecciona con el bit 3 (Wake). Si Wake=1 entonces se espera hasta detectar una marca de dirección,

[18] Op. Cit. Motorola, *M68HC11 Reference Manual*.

si por el contrario Wake=0 se espera hasta detectar que la línea de recepción esté vacía.
Los bits 0, 1, 2 y 5 no se usan y permanecen siempre en cero.

2.4.1.2. Registro de control 2. SCCR2.

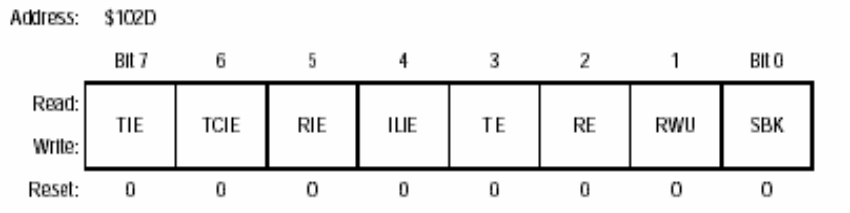


Figura 2.2. Registro 2 de control de comunicación serial.
Serial Communications Control Register 2 (SCCR2).

Este registro de control 2 es el principal, con él se configura la salida al exterior del circuito de comunicaciones serie. Esta salida se corresponde con los pines emparejados a los bits 0 y 1 del puerto D. Por tanto, esos pines tiene dos funciones: ser utilizados como bits 0 y 1 del puerto D o como señales Tx y Rx para comunicaciones serie. Mediante el bit 3 del registro de control 2 (TE) se activa o desactiva el transmisor del bit correspondiente. Si TE=1 (Transmitter enable) el transmisor está activo y el bit 1 del puerto D se desactiva. No se puede utilizar el bit PD1 del puerto D. Si TE=0 el transmisor se desconecta y no se mandan datos serie. El pin funciona como bit 1 del puerto D.

Análogamente, con el bit 2 del registro de control 2 (RE) se activa o desactiva el receptor, que comparte pin con el bit 0 del puerto D. Si RE=1 (Receive enable) el pin funciona para recibir datos serie del exterior. El bit 0 del puerto D deja de funcionar. Si RE=0 se desconecta el receptor y el pin funciona como bit 0 del puerto D.

En resumen, siempre que se quieran transmitir y recibir datos es imprescindible activar los bits RE y TE del registro de control “. Una vez activados estos bits, cualquier lectura – escritura sobre el puerto D no se reflejará en los pines 0 y 1 de puesto D, como si no existiesen.

Los bits 4, 5, 6 y 7 (ILIE, RIE, TCIE, TIE) son máscaras que permiten que ciertas interrupciones ocurran o no. TCIE y Tie corresponden a dos interrupciones del transmisor e ILIE y RIE a dos interrupciones del receptor. En el transmisor, cada vez que el registro de transmisión se vacía, es decir, cada vez que se termina de mandar un dato y por tanto se puede mandar un dato nuevo, se genera una interrupción.

Esta interrupción se enmascara con el bit TIE. Si TIE=1 la interrupción de transmisión está permitida y para TIE=0 está desactivada. El bit de estado asociado es TDRE, cuando TDRE=1 indica que se ha recibido un dato y si el bit TIE está a 1 entonces se produce la interrupción. También hay otra interrupción

que indica cuándo se ha quedado vacía la línea de transmisión después de mandar el dato, ésta se puede enmascarar con el bit TCIE. Si TCIE=1 la interrupción queda permitida, si TCIE=0 está desactivada. El bit de estado asociado es TC y el funcionamiento es análogo al anterior.

En el receptor también existe una interrupción que aparece cada vez que se ha recibido un dato, ésta se enmascara con el bit RIE del registro de control 2. Con RIE=1 se permite y con RIE=0 no. El bit de estado asociado es RDRF. La otra interrupción se activa cuando la línea de recepción está vacía, el bit ILIE se encarga de enmascararla. Igual que antes, ILIE=1 la permite e ILIE=0 la desactiva. El bit de estado asociado es IDLE.

Cuando se activa el bit 1 (SBK=1), se mandan señales de BREAK indefinidamente hasta que el bit se desactive. Las señales de BREAK se caracterizan porque se envía todo ceros por la línea serie, no sólo son cero los bits de datos, sino también se hace cero el bit de stop que siempre vale 1.

Si el bit 1 (RWU) es igual a 1 significa que el modo especial Wake Up está activo. En este modo el receptor está aletargado con las interrupciones inhibidas, esperando una condición hardware que lo despierte. Dicha condición depende del valor que tenga el bit 3 (WAKE) del registro de control 1 (CSR1); generalmente es el programa quien pone este bit a 1 y la CPU quien lo desactiva.

2.4.1.3. Registro de velocidad. BAUD.

Address: $102B

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	TCLR	SCP2	SCP1	SCP0	RCKB	SCR2	SCR1	SCR0
Reset:	0	0	0	0	0	U	U	U

U = Unaffected

Figura 2.3. Registro de rango de baud.
(Baud Rate Register, BAUD).

El registro de velocidad permite configurar la velocidad (en baudios) de la comunicación. Los bits 4 y 5 (SCP0 y SCP1) determinan la máxima velocidad en baudios. Esta velocidad depende del cristal que se haya conectado al microcontrolador. Lo habitual es colocar un cristal de 8 MHz. Para este cristal se tiene la siguiente tabla:

SCP1	SCP0	BAUDS
0	0	125000
0	1	41667
1	0	31250
1	1	9600

Tabla 2.1. Velocidad de comunicación según los valores de SCP1 y SCP0.

Una vez determinada la velocidad máxima en baudios, con los bits 0, 1 y 2; (SCR0, SCR1 y SCR2) se divide la velocidad máxima por un valor como se muestra a continuación:

SCR2	SCR1	SCR0	DIVIDIR VELOCIDAD MÀXIMA ENTRE
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Tabla 2.2. Submúltiplos de la velocidad de comunicación máxima.

De esta manera, si por ejemplo se selecciona 9600 baudios como velocidad máxima (SCP1=1, SCP2=1), al dividir entre 8 se obtiene una velocidad de 1200 baudios. Así existen diferentes combinaciones para seleccionar la velocidad de comunicación más adecuada.

2.4.1.4. Registro de estado. SCSR.

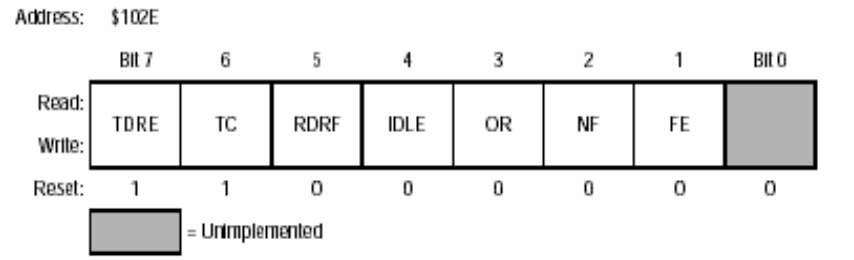


Figura 2.4. Registro de estado de comunicación serial
(Serial communications status register, SCSR).

El registro de estado es de sólo lectura y permite comprobar el estado del SCI: El bit 7 (TDRE) se pone en 1 cada vez que se ha terminado de enviar un carácter y por tanto el registro de transmisión está vacío y listo para enviar el siguiente carácter. Siempre que se vaya a transmitir un dato hay que asegurarse

que este bit está a 1, porque de lo contrario se trunca el dato que se está enviando.

El bit 6 (TC) se pone a 1 cada vez que se ha enviado un carácter y la línea de transmisión se ha quedado vacía 'IDLE'. Este bit ofrece mayores garantías cuando se quiere saber si la transmisión ha terminado completamente.

El bit 5 (RDRF) se pone a 1 cuando se ha recibido un dato nuevo en el registro de recepción.

El bit 4 (IDLE) se pone a 1 cuando detecta que la línea de recepción se ha quedado vacía (IDLE). Si el bit RWU del registro de control 2 está activo entonces este bit estará inhibido.

El bit 3 (OR) se pone a 1 cuando se ha recibido un carácter por el puerto serie y el anterior dato recibido todavía no se ha leído. Cuando ocurre este error el dato que se pierde es el que se acaba de recibir.

El bit 2 (NF=Noise Flag) se activa cuando se ha detectado un error en el dato que se acaba de recibir. La activación de este bit no produce interrupción, será el software quien se preocupe de examinar este bit después de recibir un dato para saber si es válido o no.

El bit 1 (FE) se activa cuando se ha detectado un error en la trama enviada. Al recibir un dato se comprueba que el bit de stop esté a nivel alto. Si esto no se cumple se pone este bit (FE) a nivel alto (1).
El bit 0 no se utiliza.

Todos estos bits se ponen a cero automáticamente cuando se lee el registro SCSR y a continuación se lee del registro de datos (SCDR).

2.4.1.5. Registro de datos. SCDR.

Figura 2.5. Registro de datos de comunicación serial.
(Serial communications data register, SCDR).

Este registro tiene una doble función. Si se escribe en él, el dato se manda al registro de transmisión en la unidad de transmisión del SCI para ser enviado con la velocidad y configuración establecida. Al leerlo se obtiene el valor que tiene el registro de recepción interno de la unidad de recepción del SCI.

2.5. La memoria EEPROM. [19]

En la mayoría de los miembros de la familia 68HC11 se encuentra un recurso interno bastante útil para sistemas autónomo el cual es barato y sencillo: la memoria EEPROM, con lo cual se ahorra la expansión externa de memoria.

Lo anterior tiene un inconveniente: dicha memoria EEPROM generalmente es de pequeña capacidad y la mayoría de los miembros de esta familia poseen únicamente 512 bytes de memoria. El MC68HC11E2 contiene 2048 bytes y es direccionable el inicio de la misma. Sin embargo, dentro de estos microcontroladores, prácticamente el resto poseen 512 bytes e inclusive algunos no tienen este recurso. La versión E9 posee este mismo número de bytes en memoria EEPROM, 512. Este espacio en memoria es para un gran número de aplicaciones suficiente.

La EEPROM es una memoria no volátil que mantiene los datos aún desconectando la alimentación del sistema. Su ciclo de lectura es equivalente al de la ROM interna pero en escritura es más lenta.

En la versión E9 esta memoria de 512 bytes se encuentran situados entre las direcciones $B600 - $B7FF. La programación de la EEPROM se realiza a través del registro PPROG. Sin no se va a usar se puede desactivar del mapa de memoria poniendo a cero el bit EEON del registro CONFIG.

2.5.1. Registros de control de la EEPROM. [20]

Son dos los registros que actúan directamente sobre la EEPROM, estos son los registros CONFIG y PPROG, y se describen a continuación:

2.5.1.1. Registro CONFIG.

El registro CONFIG se encarga, entre otras funciones, de activar o desactivar la EEPROM del mapa de memoria. Este registro es especial y está implementado con células EEPROM. Si el bit EEON está a ‘1’, significa que la EEPROM está activa, y si por el contrario EEON es ‘0’, entonces la EEPROM está inactiva. Este bit por defecto se encuentra en ‘1’. Este registro también controla la presencia de la ROM interna del microcontrolador.

[19] Op. Cit., *Microcontrolador MC68HC11, Fundamentos, Recursos y Programación*.
[20] Motorola, *M68HC11 Microcontrollers, M68HC11E Family Technical Data*, Rev. 4, 2002.

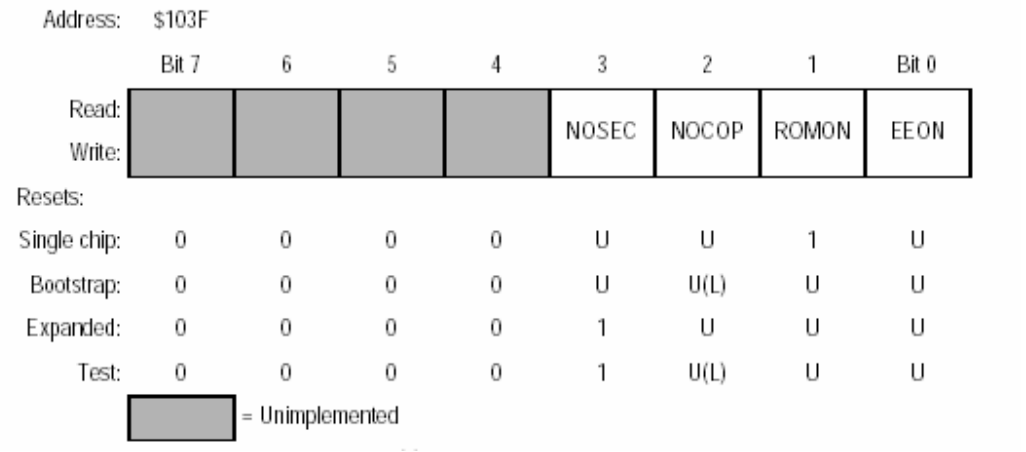


Figura 2.6. Registro de configuración de sistema.
(System Configuration Register, CONFIG).

Los bits 7, 6, 5, 4 no tienen efecto en esta versión. Son utilizados para controlar la posición de la EEPROM en otros modelos de la misma familia de microcontroladores. El resto de los bits de este registro se describen a continuación.

- **NOSEC** – Bit de desactivación del modo de seguridad.

NOSEC es inválido a menos que la opción de enmascaramiento de seguridad se especifique antes de que el MCU sea manufacturado. Si la opción de enmascaramiento de seguridad se omite, NOSEC siempre será '1'. Cuando se quiere poner el modo de seguridad en la RAM y EEPROM el bit NOSEC se debe programar a cero. De esta forma se activa el mecanismo anti-espía en el cual se previene contra la selección del modo expandido multiplexado. Además si se da un reset al MCU operando en bootstrap y NOSEC=0, entonces la EEPROM, RAM y el registro CONFIG se borran antes de proseguir con el proceso de carga. Es decir, la seguridad limita al micro a trabajar en modo single chip.
0 = Activa seguridad
1 = Desactivada seguridad

- **NOCOP** – COP System Disable

El MCU incorpora un mecanismo contra errores que puedan aparecer al procesar instrucciones. Cuando se activa esta protección, el software se encarga de impedir que un contador interno llegue a cero. Si en algún momento ocurre esto, significa que no se están ejecutando más instrucciones y en consecuencia el MCU inicia una secuencia de reset. La frecuencia del contador del COP es ajustable, para ello hay que actuar sobre los bits CR1 y CR0 del registro OPTION. Por defecto esta protección está quitada, el bit NOCOP=1.
0 = Activa la protección WatchDog
1 = Desactiva la protección WatchDog

- **ROMON** – Activación de la ROM interna

Cuando este bit se encuentra en '0', se liberan los 8 Kbytes de ROM o EPROM, por lo que se encuentran desactivadas y esta memoria es accesible externamente. Esta opción es útil en el modo expandido. En el modo single-chip la ROM siempre está activa independientemente del estado de este bit.
0 = ROM desactivada del mapa de memoria
1 = ROM activada en el mapa de memoria

- **EEON** – Activación de la EEPROM interna.

Cuando este bit está en '0', los 512 bytes de EEPROM están desactivados, y la memoria es accesible externamente. Por defecto, en los microcontroladores que tienen EEPROM interna, esta memoria está activada por lo que EEON vale '1'.
0 = EEPROM desactivada del mapa de memoria
1 = EEPROM activada en el mapa de memoria

El registro CONFIG está implementado con células EEPROM por lo que, para escribir sobre él hay que hacer lo mismo que se hace para los bytes de la EEPROM. Para borrar este registro se realiza un ERASE BULK pero apuntando a la dirección de este registro. Cuando se borra el registro CONFIG también se borra la EEPROM, además el registro se programa estando en los modos special test o bootstrap.

Este registro controla la inicialización del MCU. Los cambios en este registro no tienen efecto hasta que no se realice un reset. Para ello, el MCU tiene un mecanismo que hace que al arrancarlo el registro CONFIG se copia en un latch; de esta forma aunque se modifique, el registro CONFIG sigue manteniendo la configuración inicial, pues está grabada en el latch. Pero si se hace un reset, el nuevo valor de CONFIG se graba en el latch y la nueva configuración tiene efecto.

2.5.1.2. Registro PPROG.

Este registro se encarga de las operaciones realizadas sobre la EEPROM. Al hacer un reset se pone a cero y la EEPROM queda configurada para sólo lectura.

Address: $103B

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	ODD	EVEN	ELAT(1)	BYTE	ROW	ERASE	EELAT	EPGM
Reset:	0	0	0	0	0	0	0	0

Figura 2.7. Registro de control de programación de EPROM y EEEPROM.
(EPROM and EEPROM ProgrammingControl Register, PPROG).

- **ODD** – Programa líneas impares de la EEPROM (sólo usado en modo Test).

- **EVEN** – Programa líneas pares de la EEPROM (sólo usado en modo Test).

- **ELAT** – Siempre es cero.

- **BYTE** – Selección de borrado de un byte (este bit tiene prioridad sobre el bit ROW).
  0 = Modo de borrado de fila (Row) o total (Bulk).
  1 = Modo de borrado de un byte.

- **ROW** – Selección de borrado ROW (borrado de una fila).
  Si el bit BYTE está en uno, este bit no tiene significado.
  0 = Selección de borrado total (Bulk Erase).
  1 = Selección de borrado de una fila (Row Erase).

- **ERASE** – Selección de la opción de borrado.
  0 = Accesos a la EEPROM para lecturas o programación.
  1 = Accesos a la EEPROM para borrado.

- **EELAT** – Control sobre el latch de la EEPROM.
  0 = La dirección de la EEPROM configurada para modo READ.
  1 = La dirección y dato configurados para programción / borrado.

- **EEPGM** – Activación o desactivación del voltaje de programación de la EEPROM.
  0 = Voltaje de programación ON.
  1 = Voltaje de programación OFF.

2.5.2. Programación de la EEPROM. [21]

La programación de la EEPROM se realiza por medio del registro PPROG. En una EEPROM el estado de un byte borrado o, en su caso, no programado tiene un valor de $FF, es decir sus bits a nivel alto. La programación permite pasar a ‘0’ (nivel bajo) los bits que están a ‘1’ (nivel alto), pero no al revés; si se quiere poner un bit a uno (estando previamente a cero) hay que borrar todo el byte primero. Si por el contrario se quier pasar de ‘1’ a ‘0’ no hace falta borrar todo el byte primero. En resumen, siempre que hay que grabar un ‘1’ en una celda EEPROM que previamente estaba a ‘0’ hay que borrarla primero.

La programación requiere una sobrecarga de tensión, aunque no hay que preocuparse de esto, puesto que la proporciona el microcontrolador de forma

[21] Op. Cit., *Microcontrolador MC68HC11, Fundamentos, Recursos y Programación*.

transparente al usuario. Lo que sí hay que tener en cuenta es que la velocidad de programación es más lenta que la velocidad de lectura, pro lo tanto, entre programación y programación hay que introducir unos retardos. Estos retardos son variables y dependen la velocidad del reloj E.

Cuanto menor sea el valor de E (frecuencia de reloj) la efectividad de la sobrecarga disminuye, pero la eficiencia de la sobrecarga disminuye con el incremento de tiempo necesario para borrar y grabar la posición de EEPROM. Por esta razón el MCU incorpora un pequeño oscilador RC interno que puede generar los tiempos de programación necesarios independientemente del reloj externo E. Se recomienda que, cuando se trabaje con E < 1 [MHz] activar el oscilador RC para programar la EEPROM. Para activar ese oscilador hay que actuar sobre el bit CSEL del registro OPTION. Si E = 2 [MHz] el tiempo de escritura / borrado es de 10 [ms]. Si E > 1 [MHz] pero < 2 [MHz], el tiempo de escritura / borrado se encuentra entre 10 y 20 [ms]. Finalmente si E < 1 [MHz] se recomienda activar el oscilador interno y mantener un tiempo de 20 [ms].

Para leer la EEPROM el bit EELAT del registro PPROG debe estar a cero. Cuando este bit está a cero, el resto de bits del registro dejan de tener significado o efecto. La lectura de la EEPROM se realiza como la de una memoria ROM normal. Al realizar un reset sobre el MCU el estado de este bit es cero, por lo que la EEPROM queda configurada para lectura.

En la programación de la EEPROM, los bits ROW y ERASE del registro PPROG no se usan, éstos sólo intervienen en el proceso de borrado.

Para borrar la EEPROM existen tres modos de borrado: borrado completo (Bulk Erase), borrado de una fila (Row Erase) y borrado de un byte (Byte Erase). Según cual sea la necesidad hay que emplear uno u otro. Por ejemplo, si se desea borrar toda la EEPROM es más efectivo realizarlo en un solo paso en vez de realizar 512 borrados de byte.

- **Bulk Erase (Borrado Total).**

  Con esta opción se pueden borrar los 512 bytes de la EEPROM a la vez. Con esta operación el registro CONFIG que también está realizado con celdas EEPROM no se ve afectado.
  A continuación se muestra una subrutina que permite hacerlo:

```
BTOTAL   LDAB  #$06
         STAB  $103B ;indicar modo de borrado total
         STAA  $B600 ;situar cualquier dato en la dirección de la EEPROM
         LDAB  #$07
         STAB  $103B ;poner a '1' EEPGM (EELAT=1). Se empieza a borrar
         JSR   RET10;llamar subrutina para esperar 10 [ms]
         CLR   $103B ;borrar PPROG, terminar borrado y pasar a modo lectura
         RTS
```

- **Row Erase (Borrado de una fila).**

  Este método permite borrar una fila de la EEPROM. La utilidad radica en que una fila contiene 16 bytes por lo que se proporciona la opción de borrar bloques de bytes rápidamente, sin hacer un borrado total. La rutina siguiente acepta como dato la dirección del primer byte de la fila a borrar.

```
        LDX   #dirección ;dirección del primer byte de la fila a borrar
        JSR   BFILA
        ...

BFILA   LDAB  #$0E
        STAB  $103B ;indicar modo de borrado de fila (Row Erase)
        STAA  $00,x  ;situar cualquier dato en la dirección de la fila a borrar
        LDAB  #$0F
        STAB  $103B ;poner a '1' EEPGM (EELAT=1). Se empieza a borrar
        JSR   RET10;llamar subrutina para esperar 10 [ms]
        CLR   $103B ;borrar PPROG, terminar borrado y pasar a modo lectura
        RTS
```

- **Byte Erase (Borrado de un byte).**

  Con esta opción se borra un solo byte, sin afectar al resto. La subrutina que se muestra acepta como parámetro la dirección del byte a borrar, como en el caso anterior.

```
        LDX   #dirección ;dirección del byte a borrar
        JSR   BBYTE
        ...

BBYTE   LDAB  #$16
        STAB  $103B ;indicar modo de borrado de byte (Byte Erase)
        STAA  $00,x  ;situar cualquier dato en la dirección del byte a borrar
        LDAB  #$17
        STAB  $103B ;poner a '1' EEPGM (EELAT=1). Se empieza a borrar
        JSR   RET10;llamar subrutina para esperar 10 [ms]
        CLR   $103B ;borrar PPROG, terminar borrado y pasar a modo lectura
        RTS
```

CAPÍTULO 3.

PERIFÉRICOS DE COMUNICACIONES .

3.1. Introducción.

En este capítulo se trata a grandes rasgos las características principales de una computadora, así como una revisión de su arquitectura y organización. Esto es con el propósito de tener una visión general de cómo opera y se estructura una computadora y a la vez ligar los dispositivos que se han utilizado en el desarrollo de esta tesis.

Una computadora está formada por dos partes claramente diferenciables, para un adecuado funcionamiento de la misma.
Una de ellas formada por el medio físico o tangible, compuesto por el equipo y sus accesorios, tales como: los circuitos, teclado, monitor, cables, etc. Y la segunda parte que comprende toda la información complementaria que el equipo necesita par su normal funcionamiento: el sistema operativo, intérpretes, compiladores, programas, etc.
La parte física se suele denominar con el término inglés *"hardware"* y la parte no física con un término derivado de aquel: *"software"*.

Partiendo de una definición de la función general de una computadora se llegará a un detalle de los elementos fundamentales de la misma: podemos decir que la función principal de una computadora es la obtención de resultados a través de la información dada por el hombre en forma de instrucciones y datos.

Cuando se describe o define una computadora, frecuentemente se distingue entre arquitectura y organización de la misma.
La arquitectura de las computadoras se refiere a los atributos de un sistema que son visibles a un programador, o para decirlo de otra manera, aquellos atributos que tienen un impacto directo en la ejecución lógica de un programa. **La organización de las computadoras** se refiere a las unidades funcionales y sus interconexiones, que materializan especificaciones arquitectónicas.

Entre los ejemplos de atributos arquitectónicos se encuentran el conjunto de instrucciones, el número de bits usados para representar varios tipos de datos, mecanismos de E/S y técnicas para direccionamiento de memoria. Entre los atributos de organización se incluyen aquellos detalles del hardware transparentes al programador, tales como señales de control, interfaces entre la computadora y los periféricos y la tecnología de memoria usada.

3.2. Estructura y función de una computadora. [22]

Una computadora es un sistema complejo; hoy en día las computadoras contienen millones de componentes electrónicos básicos. A través de un reconocimiento jerárquico de la naturaleza de la mayoría de los sistemas complejos se pueden describir clara y brevemente la estructura y función de una computadora. Un sistema jerárquico es un conjunto de sistemas interrelacionados, cada uno de los cuales se organiza en una estructura jerárquica, uno tras otro, hasta que alcanza el nivel más bajo de subsistema elemental.

La naturaleza jerárquica de los sistemas complejos es esencial tanto para su diseño como para su descripción. Así, tratando solamente con un nivel particular del sistema, se puede comprender el comportamiento de cada nivel, sus componentes e interrelaciones. En cada nivel lo que más importa es su estructura y función:

- Estructura: el modo en que los componentes están interrelacionados.
- Función: la operación de cada componente individual como parte de la estructura.

3.2.1. Función.

En forma general, las funciones básicas que una computadora puede llevar a cabo son (ver figura 3.1):

- Procesamiento de datos
- Almacenamiento de datos
- Transferencias de datos
- Control

La función principal de una computadora es el procesamiento de datos, los cuales deben ser guardados temporalmente o al menos aquellos datos con los que esté trabajando en un momento dado. De igual forma, debe tener la capacidad de transferirlos en ella misma y al mundo exterior. El entorno de operación de la computadora se compone de dispositivos que sirven bien como fuente o como destino de datos. Cuando se reciben o se llevan datos a un dispositivo que está directamente conectado con la computadora, el proceso se conoce como entrada-salida (E/S), y este dispositivo recibe el nombre de periférico. El proceso de transferir datos a largas distancias, desde o hacia un dispositivo remoto, recibe el nombre de comunicación de datos.
Finalmente, debe existir un control de estas tres funciones. Este control es ejercido por una unidad de control que gestiona los recursos de la máquina y dirige las prestaciones de sus partes funcionales.

[22] Stallings William, Organización y Arquitectura de computadores, diseño para optimizar prestaciones; 4ª edición, Prentice Hall, 1997.

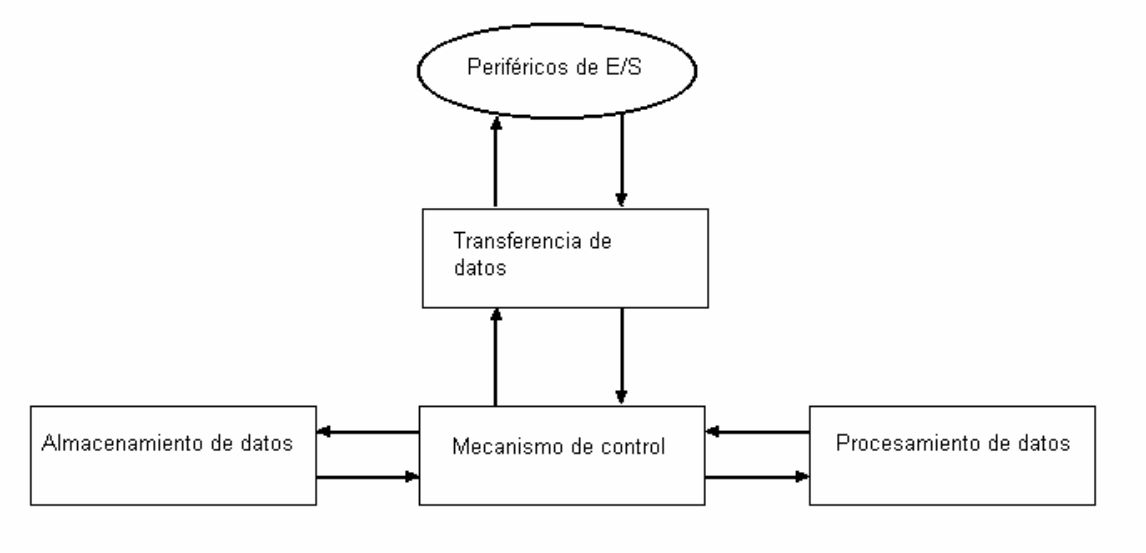


Figura 3.1. Visión funcional de una computadora.

3.2.2. Estructura.

La computadora es una entidad que interactúa de alguna manera con su entono externo. En general, todas sus conexiones con el entorno externo pueden ser clasificadas como dispositivos periféricos o líneas de comunicación. Sin embargo, la estructura central de la computadora se basa en cuatro componentes principales:

- ***Unidad Central de Proceso (CPU):*** controla el funcionamiento de la computadora y lleva acabo sus funciones de procesamiento de datos. Frecuentemente se le llama simplemente procesador.
- ***Memoria principal:*** almacena datos.
- ***Periféricos de E/S (entrada-salida):*** transfieren datos entre la computadora y el entorno externo.
- ***Sistema de interconexión:*** es un mecanismo que proporciona la comunicación entre la CPU, la memoria principal y los periféricos de E/S.

La CPU es el corazón de la máquina y está formada por la unidad de control, la unidad lógica-aritmética y la unidad de memoria.

- **Unidad de control:** es la encargada de interpretar y seleccionar las distintas instrucciones que forman el programa y que se encuentran almacenadas en la memoria. Esta unidad interpreta, de entre los datos introducidos a la memoria, aquellos códigos que significan instrucciones, generando señales que envía a las otras unidades de la máquina que son las encargadas de ejecutarlos.

- **Unidad lógica-aritmética:** es la que lleva a cabo todos los cálculos y comparaciones o decisiones que forman el programa. Durante la ejecución del mismo fluyen entre esta unidad y la memoria los diferentes datos, indicándole la unidad de control el tipo de operación a realizar con estos datos.

- **Unidad de memoria:** Es la parte de la computadora en la que se introducen y almacenan los datos e instrucciones que forman el programa. De ella extraen, la unidad de control y la unidad lógica-aritmética, la información que necesitan para ejecutar los programas y deposita los datos intermedios que se generan. Una vez finalizada la ejecución del programa, quedan almacenados en esta unidad los resultados para ser extraídos por la unidad de salida.

Los dispositivos periféricos a través de los cuales la computadora realiza todas sus conexiones con el entorno externo se dividen en unidades de entrada, unidades de salida y unidades de almacenamiento.

a) ***Unidad de entrada****:* es el sistema mediante el cual se introducen en el ordenador los datos y las instrucciones que forman el programa que se habrá de ejecutar. Son los elementos de comunicación entre el hombre y la máquina.

   Ejemplos de periféricos de entrada son:
   - Teclados
   - Tarjetas de cartón perforado o magnéticas
   - Lápices ópticos
   - Lector de códigos de barras
   - Joysticks y ratones
   - Digitalizadores y scanners
   - Convertidores analógico/digital
   - Sensores (auditivos, de movimiento, ópticos, etc.)

b) ***Unidad de salida****:* es el sistema que toma de la memoria la información codificada y la convierte en un formato que, o bien puede ser directamente interpretado por el hombre, o trasladado a otra máquina de procesamiento. En general son los instrumentos de comunicación entre la máquina y el hombre.

   Ejemplos de periféricos de salida son:
   - Monitores
   - Impresoras
   - Plotters
   - Convertidores digitales a analógicos
   - Displays de cuarzo líquido o leds
   - Bocinas

c) ***Unidades de almacenamiento****:* Son aquellas unidades que utilizando algún elemento de soporte (generalmente magnético), como memorias auxiliares de almacenamiento, permiten la entrada o salida de información de la Unidad Central de Proceso (CPU).

Ejemplos de periféricos de almacenamiento son:

- Unidades de disco magnético
- Unidades de cinta magnética
- Unidades de disco óptico
- Cintas de video

Después de haber revisado en forma general el funcionamiento y estructura de una computadora, centraremos nuestra atención en los periféricos que se utilizaron en la realización de esta tesis, siendo estos los puertos de comunicaciones serie y paralela, así como el funcionamiento del teclado.

3.3. Periféricos de Entrada – Salida (E/S) para comunicaciones de una computadora.

Diferentes asociaciones, organismos y fabricantes están tratando continuamente de definir una serie de especificaciones, de tal manera, que estando normalizadas las diferentes interfases y protocolos, sean compatibles unos equipos con otros. Entre éstos, destacan el CCITT (Comité Consultivo Internacional de Telecomunicaciones y Telefonía), el ANSI (Instituto Nacional Americano de Normalización), la ISO (Organización Internacional de Estándares) y la EIA (Asociación de Industrias Electrónicas).

La EIA enfoca su actividad principalmente en el campo de la normalización eléctrica, y entre sus logros más destacados en las interfases RS-232C y RS-449, convertidos en estándares y usados por la mayoría de fabricantes de equipo de cómputo.

Cabe definir la interfase como una frontera compartida, que está determinada mediante una serie de características eléctricas y funcionales perfectamente especificadas. Las interfases constituyen el medio por el cual los periféricos se conectan a la CPU. Existen interfases de aplicación general para distintos usos (paralelo y serie) aunque muchos periféricos requieren de interfases específicas.

Las interfases paralelas generalmente responden a una especificación estándar de IBM denominada *1284*, puerto principalmente, de impresoras y plotters. Un mismo equipo puede contar con una o varias de ellas identificándose como LPT1, LPT2, etc. Su principal característica es que la información se transmite, básicamente, por varios conductores simultáneamente (en paralelo) en un modo denominado síncrono.

Las interfases serie generalmente responden a una especificación estándar denominada RS-232C, son muy comunes y permiten la conexión de terminales, ratones, lápices ópticos, scaners, modems, sistemas de adquisición de datos, impresoras y plotters. Por el tipo de periférico que pueden admitir se deduce que son tanto de entrada como de salida. Un mismo equipo puede contar con una o varias de ellas, identificándose como COM1, COM2, etc. Su principal característica es transmitir información por un solo conductor o cable (uno para los datos de entrada, otro para los de salida y un cable común) es decir que los datos se transmiten uno a continuación del otro, en un modo denominado asíncrono en que se debe establecer la velocidad de comunicación y una convención respecto de la información que se envía o recibe. [23]

Existe recientemente un nuevo tipo de conexión para transferencia de datos que está reemplazando todos los diferentes conectores de puertos paralelo y serie, conocido como USB (Universal Serial Bus). Esta nueva forma de transmisión de datos permite conectar diferentes dispositivos en una misma computadora, como impresoras, escáners, joysticks, por mencionar algunos, en forma simultánea. La mayoría de las computadoras con esta característica poseen un par de puertos USB, y periféricos especiales llamados "USB hubs" tienen puertos adicionales que permiten conectar más dispositivos a la vez.

Esta capacidad de incrementar el número de puertos es una de las innovaciones más importantes del USB, ya que así ya no está limitado el número de dispositivos que se pueden conectar a la computadora debido a la escasa cantidad de puertos que se tienen en ella; sin embargo, no es la única innovación. También de gran importancia y fascinación es que estos dispositivos se pueden conectar y desconectar en cualquier momento y la computadora los detecta en el instante que esta acción se realice. Esto ha permitido desarrollar nuevos dispositivos, además de que los ya existentes muden a conexión USB como las impresoras, como "memory sticks" que poseen capacidades diferentes de memoria y seguramente irán reemplazando a los discos flexibles y discos compactos. [24]

3.3.1. El puerto paralelo. [25]

Puerto de comunicaciones de una PC, normalmente programado unidireccional (de fábrica). Esta interfase se distinguen por dos elementos: la parte transmisora y la parte receptora. La parte transmisora coloca la información en las líneas de datos e informa a la parte receptora que la información (los datos) está disponible; entonces la parte receptora lee la información en las líneas de datos e informa a la parte transmisora que ha tomado la información. Ambas partes sincronizan su respectivo acceso a las líneas de datos; la parte receptora no leerá

[23] Armada de México, *El Universo Digital de IBM PC, AT y PS/2;* Valladolid, España, 1997. http://fly.to/udigital
[24] http://www.usb.org/home
[25] http://www.modelo.edu.mx/univ/virtech/circuito/paralelo.htm

las líneas de datos hasta que la parte transmisora se lo indique en tanto que la parte trasmisora no colocará nueva información en las líneas de datos hasta que la parte receptora remueva la información y le indique a la parte transmisora que ya ha tomado los datos. A esta coordinación de operaciones entre la parte transmisora y la parte receptora se le llama ***handshaking***.

3.3.1.1. El Handshaking.

Para implementar el ***handshaking*** se requieren tres líneas: la línea de ***strobe***, que es la que utiliza la parte transmisora para indicarle a la parte receptora la disponibilidad de información, y la línea de ***acknowledge***, que es la que utiliza la parte receptora para indicarle a la parte transmisora que ha tomado la información y que está lista para recibir más datos. El puerto paralelo provee de una tercera línea llamada ***busy***. Esta linea es utilizada para que la parte receptora le indicarle a la parte transmisora que está ocupada y de esta manera evitar colisiones. Una típica sesión de transmisión de datos es como sigue:

Parte transmisora:

- La parte transmisora verifica la línea busy para ver si la parte receptora está ocupada. Si la línea busy está activa, la parte transmisora espera en un bucle hasta que la línea busy esté inactiva.
- La parte transmisora coloca la información en las líneas de datos.
- La parte transmisora activa la línea de strobe.
- La parte transmisora espera en un bucle hasta que la línea de acknowledge esté activa.
- La parte transmisora inactiva la línea de strobe.
- La parte transmisora espera en un bucle hasta que la línea acknowledge esté inactiva.
- La parte transmisora repite los pasos anteriores por cada byte a ser transmitido.

Parte receptora:

- La parte receptora inactiva la línea busy, asumiendo que está lista para recibir información.
- La parte receptora espera en un bucle hasta que la línea strobe esté activa.
- La parte receptora lee la información de las líneas de datos, y de ser necesario los procesa.
- La parte receptora activa la línea acknowledge.
- La parte receptora espera en un bucle hasta que esté inactiva la línea de strobe.
- La parte receptora inactiva la línea acknowledge.
- La parte receptora repite los pasos anteriores por cada byte que debe recibir.

Se debe ser muy cuidadoso al seguir estos pasos, tanto la parte transmisora como la receptora coordinan sus acciones de tal manera que la parte transmisora no intentará colocar varios bytes en las líneas de datos, en tanto que la parte receptora no debe leer más datos de los que le envíe la parte transmisora, por lo que se transmite y recibe un byte a la vez.

3.3.1.2. Conectores del puerto paralelo.

El puerto paralelo tiene diferentes tipos de conectores definidos por el estándar IEEE 1284. El objetivo de este estándar es diseñar nuevos dispositivos que sean totalmente compatibles con el puerto paralelo estándar (SPP) definido originalmente por IBM. El primero, llamado ***1284 tipo A*** es un conector hembra de 25 pines tipo D. El segundo conector se llama ***1284 tipo B*** que es un conector de 36 pines de tipo *centronics* y lo encontramos en la mayoría de las impresoras. El tercero se denomina ***1284 tipo C***; se trata de un conector similar al 1284 tipo B pero más pequeño, además se dice que tiene mejores propiedades eléctricas y mecánicas. Este conector es el recomendado para nuevos diseños.

El puerto paralelo de una PC generalmente se incluye en la tarjeta madre de ella. Este puerto utiliza un conector hembra DB25, definido por el estándar 1284 tipo A, mencionado en el párrafo anterior. De estas 25 conexiones, 17 son líneas de señales y 8 son líneas de tierra. Las líneas de señales están formadas por tres grupos: 4 líneas de control, 5 líneas de estado y 8 líneas de datos.

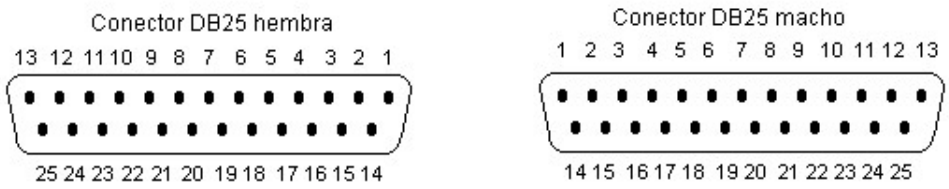


Fig. 3.2. Conectores DB25 hembra y macho para puerto paralelo.

Las líneas de control son usadas para la interfase, control e intercambio desde la PC a la impresora, que es el uso más común de este puerto. Las líneas de estado son usadas para el intercambio de mensajes, indicadores de estado desde la impresora a la PC, como son: falta de papel, impresora ocupada, error en la impresora, etc. Las líneas de datos suministran los datos de impresión de la PC hacia la impresora y solamente en esa dirección. Las nuevas implementaciones del puerto permiten una comunicación bidireccional mediante estas líneas.
La tabla 4.1 muestra la asignación de correspondiente entre los pines del conector DB25 y su función:

PIN	E/S	POLARIDAD ACTIVA	DESCRIPCIÓN
1	Salida	0	Strobe
2 ~ 9	Salida	-	Líneas de datos (bit 0/pin 2, bit 7/pin 9)

10	Entrada	0	Línea acknowledge (activa cuando el sistema remoto toma datos)
11	Entrada	0	Línea busy (si está activa, el sistema remoto no acepta datos)
12	Entrada	1	Línea Falta de papel (si está activa, falta papel en la impresora)
13	Entrada	1	Línea Select (si está activa, la impresora se ha seleccionado)
14	Salida	0	Línea Autofeed (si está activa, la impresora inserta una nueva línea por cada retorno de carro)
15	Entrada	0	Línea Error (si está activa, hay un error en la impresora)
16	Salida	0	Línea Init (Si se mantiene activa por al menos 50 micro-segundos, ésta señal autoinicializa la impresora)
17	Salida	0	Línea Select input (Cuando está inactiva, obliga a la impresora a salir de línea)
18 ~ 25	-	-	Tierra eléctrica

Tabla 3.1. Configuración del puerto paralelo estándar.

Observe que el puerto paralelo tiene 12 líneas de salida (8 líneas de datos, strobe, autofeed, init, y select input) y 5 de entrada (acknowledge, busy, falta de papel, select y error).

Cada una de estas líneas (control, estado, datos) puede ser referenciada de modo independiente mediante un registro.
Cada registro del puerto paralelo es accedido mediante una dirección. El puerto paralelo tiene tres registros: ***registro de datos***, ***registro de estado*** y el ***registro de control.***

El *registro de datos* es un puerto de lectura-escritura de ocho bits. Leer el registro de datos (en la modalidad unidireccional) retorna el último valor escrito en el registro de datos. Los registros de control y estado proveen la interfase a las otras líneas de E/S. La distribución de las diferentes señales para cada uno de los tres registros de un puerto paralelo esta dada en las siguientes tablas:

BIT #	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
PROPIEDADES	Dato 7	Dato 6	Dato 5	Dato 4	Dato 3	Dato 2	Dato 1	Dato 0

Tabla 3.2. Registro de datos.

BIT #	PROPIEDADES
Bit 7	Busy
Bit 6	Acknowledge
Bit 5	Falta de papel
Bit 4	Select In
Bit 3	Error
Bit 2	IRQ (Not)
Bit 1	Reservado
Bit 0	Reservado

Tabla 3.3. Registro de estado.

BIT #	PROPIEDADES
Bit 7	No usado
Bit 6	No usado
Bit 5	Permite puerto bidireccional
Bit 4	Permite IRQ a través de la línea acknowledge
Bit 3	Selecciona impresora
Bit 2	Inicializa impresora
Bit 1	Nueva línea automática
Bit 0	Strobe

Tabla 3.4. Registro de control.

Una PC soporta hasta tres puertos paralelo separados, por tanto puede haber hasta tres juegos de registros en un sistema en un momento dado. Existen tres **direcciones base** para el puerto paralelo asociadas con tres posibles puertos paralelo: 0x3BCh, 0x378h y 0x278h, nos referimos a éstas como las direcciones

base para el puerto **LPT1**, **LPT2** y **LPT3**, respectivamente. El registro de datos se localiza siempre en la dirección base de un puerto paralelo, el registro de estado aparece en la dirección base + 1, y el registro de control aparece en la dirección base + 2. Por ejemplo, para un puerto LPT2 localizado en 0x378h, ésta es la dirección del registro de datos, al registro de estado le corresponde la dirección 0x379h y su respectivo registro de control está en la dirección 0x37Ah.

3.3.2. El puerto serie.

3.3.2.1. Comunicación Serial.

Una de las formas de comunicación bidireccional que proporcionan las computadoras personales (PC's) se realiza a través del puerto asíncrono, puerto serie o "puerto COM". Debido a que este puerto opera bajo el estándar RS-232C de EIA (Electronics Industry Association).

Estos puertos convierten los ocho o más bits en una cadena de pulsos y transfieren la información en series largas de bits. A esta forma de intercambio de datos se le llama transferencia serial.

La transferencia de información en forma serial se puede efectuar básicamente por dos métodos:

- Transmisión síncrona
- Transmisión asíncrona

En la comunicación síncrona, los sistemas transmisor y receptor son sincronizados de tal forma que la conexión lógica entre sistemas se realice al mismo tiempo, así se permite que los caracteres sean enviados uno tras otro, pero incluye caracteres especiales al principio de cada mensaje y caracteres de relleno que son enviados cuando no se está enviando información. Por esto, los caracteres deben estar espaciados.

La otra alternativa es marcar en el tren de datos para indicar el inicio y término del bloque de datos. El sistema receptor puede detectar el comienzo de la información del bloque de datos, este método le permite evitar confusiones en la sincronización. Este tipo de transmisión se denomina asíncrona y es la base de operación en los puertos seriales de las computadoras personales. La velocidad en que se transmite la información sincronía es mayor a la transmisión asíncrona debido a que ésta requiere de bits extra con cada carácter.

En la mayoría de los sistemas asíncronos, los datos son divididos en pequeños bloques de bits, a cada uno de estos bloques se le llama "palabra" (*word* en inglés), y puede consistir de cinco a ocho bits. La longitud de palabra comúnmente utilizada es de siete u ocho bits, debido a que las palabras formadas se asemejan a la descripción de caracteres del código ASCII.

Para el protocolo de transmisión serial de datos, los bits de una palabra son enviados de uno en uno a través del canal de comunicación. Cuando no se transmite ningún carácter, la línea está en alto y cuando cambia a bajo, se sabe que se transmiten datos. Por convención, primero se envía el bit menos significativo de la palabra, el resto de los bits son enviados uno tras otro en orden creciente de significación.

Además de los bits de datos se tiene un pulso de doble longitud llamado bit de inicio *(start bit)*, el cual indica el inicio de la palabra de datos. También se tiene otro bit para indicar el fin de la palabra *(stop bit)*. Entre el último bit de la palabra y el bit de fin se tiene un bit más llamado bit de paridad *(parity bit)*; este bit se utiliza para verificar la integridad de los datos. De esta forma, los bits de datos, los bits de inicio y fin, y el bit de paridad forman un frame de datos.

En la comunicación serial, se pueden utilizar cinco tipos de bits de paridad, dos de los cuales ofrecen detección de errores. La detección del error se efectúa contando el número de bits en la palabra de datos para determinar si el resultado es par o non. En la paridad non, el bit de paridad es inicializado cuando el número de bits en la palabra es non. La paridad par enciende el bit de paridad cuando en la palabra el número de bits es par.

En la paridad por marca *(mark parity)*, el bit de paridad siempre está encendido. En la paridad espacio *(space parity)* siempre deja apagado el bit de paridad. Por último, No paridad, no incluye bit de paridad en el frame.

Las señales seriales también se describen por la velocidad nominal a la cual son transmitidos los trenes de bits. La forma estándar de medición es la cantidad de bits que son enviados en una unidad de segundo o bits por segundo - *bps*. La velocidad mínima más común es de 300 bps, sin embargo, existen también submúltiplos de esta velocidad que son: 50, 100 y 150 bps. Por el contrario, las velocidades de transmisión mayores a 300 bps son 600, 1200, 2400, 4800 9600 hasta 19200 bps. [26]

3.3.2.2. Tipos de transmisión.

Los diferentes tipos de transmisión utilizados comúnmente para comunicar dos puntos, operan bajo los siguientes esquemas:

a) Transmisión Simplex. La comunicación se efectúa solamente en un sentido: del transmisor al receptor.

b) Transmisión Half-duplex. Es la comunicación electrónica en dos sentidos, la cual se efectúa en un sólo sentido a la vez. Por lo que debe existir algún

[26] Sargent III Murray, Shoemaker Richard L., *The IBM PC from the incide out,* Addison-Wesley Publishing Company 1986.

protocolo para cambiar el modo de transmisión – recepción de la línea de transmisión utilizada. Por ejemplo, la comunicación entre la gente es usualmente half-fuplex, uno escucha mientras el otro habla.

c) Transmisión Full-duplex. Este tipo de comunicación también se efectúa en ambos sentidos, aunque la principal diferencia con el anterior es que aquí se lleva a cabo en ambos sentidos al mismo tiempo. En consecuencia, se utilizan líneas diferentes para las dos direcciones de la señal. [27]

3.3.2.3. Estándar RS-232C.

En la actualidad, la gama de protocolos y estándares de comunicaciones existentes es muy amplia, aunque en la práctica se ha extendido el uso de dos bien conocidos: la interfase serie, que asociamos al RS-232C y la interfase paralela.

Las interfases tipo serie se caracterizan por el envío de la información bit a bit de una manera seriada o secuencial; en las interfases paralelo, la información se transmite en grupo, enviando una serie de bits simultáneamente.

Cada uno de ellos presenta sus ventajas e inconvenientes, estando condicionado el uso de uno y otro por el tipo de equipo o por el costo de los mismos. Mientras que en una interfase tipo serie el costo del mismo es muy pequeño por emplear pocos circuitos, en uno de tipo paralelo se necesita un circuito por cada bit del carácter, necesitándose de esta manera más lógica; por el contrario su rapidez y fiabilidad es mayor, siendo ideal para la interconexión en distancias cortas.

Sin embargo, el método de comunicaciones más empleado para interconectar computadoras personales o algún otro dispositivo de forma sencilla es caracterizada por el estándar RS-232C. Este estándar representa los 1's entre –3 hasta –20 volts y los 0's entre +3 y +20 volts, siendo los valores de voltaje más utilizados para este propósito –12V y +12V, respectivamente. Esto ofrece un intervalo mayor de voltaje entre cada valor diferente así como un cruce por cero que brindan una inmunidad más grande al ruido que los niveles de voltaje TTL. [28]

3.3.2.4. Conectores para comunicación serial.

Este estándar también posee un conector especial para la comunicación serial llamado DB-25, que es un conector de 25 pines. Este conector es similar al utilizado en el puerto paralelo. Empero, generalmente se utilizan un máximo de 9 pines para realizar esta función, lo que crea un conector llamado DB-9. Dichos

[27] Olmedo A. Oscar, Buenabad C. Jorge, Vega G. Andrés, *Programación de microprocesadores 8086/88 e interfases;* Centro de Investigación y de Estudios Avanzados del IPN, Departamento de Ingeniería Eléctrica – Sección de Computación.
[28] Op. Cit., *El Universo Digital de IBM PC, AT y PS/2*

conectores se muestran el la figura 4.3. Pero si se trata de aplicaciones sencillas, la conexión para la comunicación serial puede ser reducida a solamente 2 pines que serán los conductores necesarios para dicho propósito: uno para transmitir y otro para recibir información. Por lo anterior solamente 3 conductores son necesarios: los dos mencionados anteriormente y uno más para establecer una referencia o tierra común entre los dispositivos.

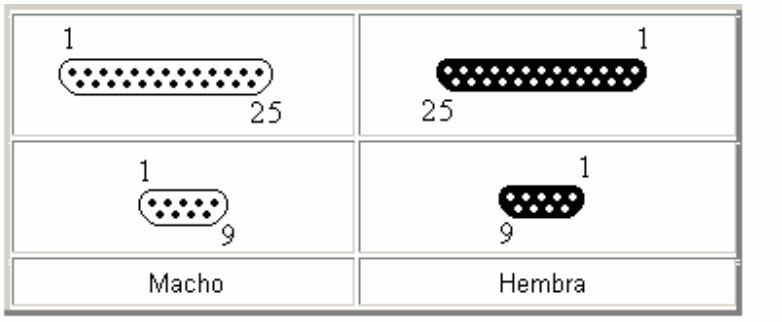


Figura 3.3. Conectores DB25 y DB9 macho y hembra.

Figura 3.4. Fotografía de un conector DB9 macho.

Generalmente, en las computadoras se emplea un conector DB9 macho, que se muestra en la figura 3.4. A la computadora en este caso se le denomina DTE (Data Terminal Equipment) que, a través de un cable es conectado a un periférico que posee un conector hembra denominado DCE (Data Communications Equipment). En dichos conectores, las señales generadas son las siguientes:

Pin #	Señal
1	Data Carrier Detect (DCD)
2	Received Data (RxD)
3	Transmited Data (TxD)
4	Data Terminal Ready (DTR)
5	Ground (Gnd)
6	Data Set Ready (DSR)

7	Request to Send (RTS)
8	Clear to Send (CTS)
9	Ring Indicator

Tabla 3.5. Descripción de los pines del conector DB9.

Los datos son recibidos y transmitidos por los pines 2 y 3 respectivamente. La señal DSR del pin 6 es un indicador de que los datos están listos. De forma similar, DCD, pin 1, indica que los datos son recibidos correctamente. Los pines 7 (RTS) y 8 (CTS) son usados para control. En la mayoría de las situaciones, RTS y CTS se encuentran encendidos durante toda la sesión de comunicación.

Existen en el mercado diferentes cables con conectores asociados de tal forma que se puedan utilizar para diferentes dispositivos. Ejemplos de estos son cables DB9 macho – DB9 hembra, DB9 macho – DB25 hembra, en la siguiente tabla se muestra cómo se interconectan las terminales de estos conectores para un cable DB9 macho a un DB25 hembra:

DESCRIPCIÓN	SEÑAL	9-PIN DTE	25-PIN DCE
Carrier Detect	CD	1	8
Receive Data	RD	2	3
Transmit Data	TD	3	2
Data Terminal Ready	DTR	4	20
Signal Ground	SG	5	7
Data Set Ready	DSR	6	6
Request to Send	RTS	7	4
Clear to Send	CTS	8	5
Ring Indicator	RI	9	22

Tabla 3.6. Descripción de la interconexión de conectores DB9 – DB25.

Sin embargo, cómo ya se ha mencionado, estos cables existen en el mercado y son de bajo costo. En nuestro caso, utilizaremos un cable de estos para realizar la interfase serial entre el microcontrolador y la computadora.

A continuación describiremos el funcionamiento general de otro periférico de entrada necesario para el desarrollo de este proyecto: El lector Metrologic Voyager MS-9500 descrito en el capítulo dos. Este lector de códigos de barras entrega los datos leídos de forma idéntica a como funciona un teclado e inclusive posee la capacidad de conectar directamente el teclado al lector y éste a la computadora, de tal forma que funcionen en forma independiente y los datos puedan ser ingresados mediante teclado o mediante el lector utilizado.

3.4. El Teclado.

Los teclados son actualmente el medio de entrada. A pesar de que en éstos el principio de funcionamiento se basa en interruptores y decodificadores, de esta manera el CPU obtiene la información de la posición que este interruptor (tecla) ocupa dentro del teclado.

Existen teclados decimales, hexadecimales y alfanuméricos. La disposición de las teclas en los teclados alfanuméricos es parecida a los de las máquinas de escribir, existiendo dos configuraciones principales: la conocida como QWERTY (la más popular) y la conocida como ASERTY, correspondiendo las siglas a las primeras cinco teclas de la segunda fila de caracteres.

Las principales características que definen un teclado para su utilización en informática son:

+ *Tipo de Tecla:* ésta es una de las características más importantes ya que determina la profesionalidad y el precio del teclado. Existen modelos denominados Soft-touch (toque suave) en las que los interruptores son láminas metálicas blandas recubiertas de una membrana plástica. Su utilización es incómoda pero son muy baratos y se emplean en sistemas de seguridad ya que al ser impermeables y herméticos no corren riesgos (por ejemplo: derramamientos accidentales sobre el teclado, polvo, etc.). A continuación de los teclados de toque suave, viene los tipos calculadora, que representan una pequeña mejora respecto de los anteriores y luego de éstos, los teclados de teclas móviles o de gran recorrido, que son los preferidos para aplicaciones profesionales, tratamientos de textos, etc.

+ *Cantidad de teclas:* en este punto lo importante es la cantidad de teclas con que cuenta el teclado, además de las alfanuméricas correspondientes, es decir teclas de funciones, teclado numérico separado también llamado Key-Pad, teclas especiales para otros lenguajes como acentos y “ñ”, recordando que el teclado universal de computación es de sistema anglosajón. El primitivo teclado de la IBM/PC contaba con 83 teclas; los más modernos tienen 101 y hasta 103 teclas y se denominan “enhanced” (mejorados) ya que tienen mayor cantidad de teclas de función y teclas separadas para el control del cursor. Estos teclados se impusieron con la aparición de las IBM/AT. [29]

3.4.1. Teoría del teclado de una PC.

El teclado de IBM más popular, envía código scan a la computadora. El código scan le informa al Bios del teclado qué teclas han sido presionadas o soltadas. Si pensamos en un ejemplo, la letra ‘A’. Esta letra tiene un código de 1C en hexadecimal. Cuando se presiona la ‘A’ el teclado mandará 1C a través de su línea serial. Si la tecla todavía es presionada, es decir, no se ha dejado de

[29] http://www.beyondlogic.org/keyboard/keybrd.htm

presionar por más de su retardo, se volverá a mandar 1C a través de su línea serial. Si se sigue presionando, volverá a mandarse otra vez 1C a través de la línea serial. Esto ocurre hasta que la tecla sea soltada o hasta que otra tecla sea presionada.

Sin embargo, el teclado también mandará otro código cuando la tecla es soltada. Si seguimos con el mismo ejemplo de la letra 'A', una vez que sea soltada, el teclado mandará F0 a través de su línea serial para indicar que la tecla con el código posterior ha sido soltada, así se volverá a mandar 1C para saber que esta tecla se ha soltado. Así, al presionar y soltar la letra 'A', el teclado mandará el siguiente código: "1CF01C".
El teclado solamente tiene un código para cada tecla. No importa si la tecla de shift ha sido presionada. Mandará de igual forma el mismo código. Es el Bios del teclado el que se encarga de determinar esto y de tomar la acción apropiada. El teclado tampoco procesa la tecla de Num Lock, Caps Lock y Scroll Lock. Cuando algunas de estas teclas son presionadas, el teclado mandará el código scan de esta tecla. Posteriormente, nuevamente el Bios del teclado es el encargado de mandar un código al teclado para prender el LED que identifica a estas teclas.

Ahora, existen 101 teclas y se sabe que 8 bits generan 256 combinaciones diferentes, así, podría pensarse que solamente se necesita de un byte para cada tecla. Sin embargo, desafortunadamente un grupo completo de teclas que se encuentran en el teclado son teclas extendidas, y esto requiere dos códigos scan. Estas teclas son precedidas por un E0 en hexadecimal. Pero esto no se detiene después de dos códigos scan. Por ejemplo, también se podría pensar que el código E1, 14, 77, E1, F0, 14, F0, 77 no podría mandarse, sin embargo, esto sucede cuando se presiona la tecla Pause/break.
Cuando una tecla extendida ha sido soltada, se espera que una F0 se envíe para decir o indicar que una tecla ha sido soltada. Entonces se espera un E0, indicando que era una tecla extendida seguida del código scan para la tecla presionada. Sin embargo, éste no es el caso. E0 es enviado primero, seguido de F0 cuando una tecla extendida ha sido soltada. [30]

3.4.2. Códigos del teclado. [31]

El diagrama muestra el código asignado a cada tecla. El código es mostrado en la parte baja de cada tecla. Por ejemplo, el código para tecla ESC es 76; el código es mostrado en hexadecimal (ver figura 3.5).

[30] Ibid.
[31] Ibid.

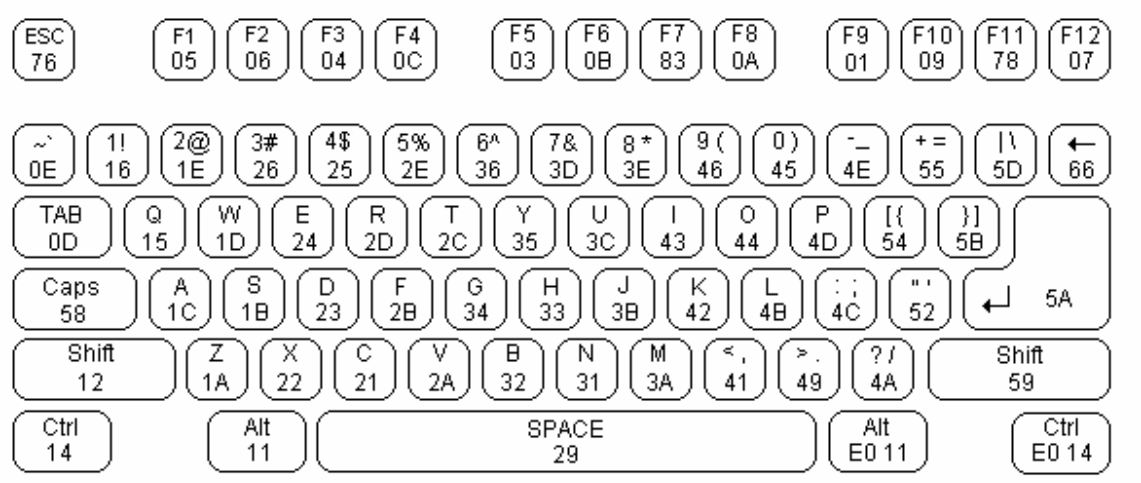


Fig. 3.5. Tabla de equivalencia y descripción de las teclas de un teclado y su código scan.

Como se puede observar, la asignación del código es realmente aleatorio, como podría pensarse también es el lugar ocupado por cada tecla. En muchos casos la forma más fácil de convertir este código a ASCII es el utilizar una tabla.

En la figura 3.6 se muestra el código para el teclado numérico y el extendido:

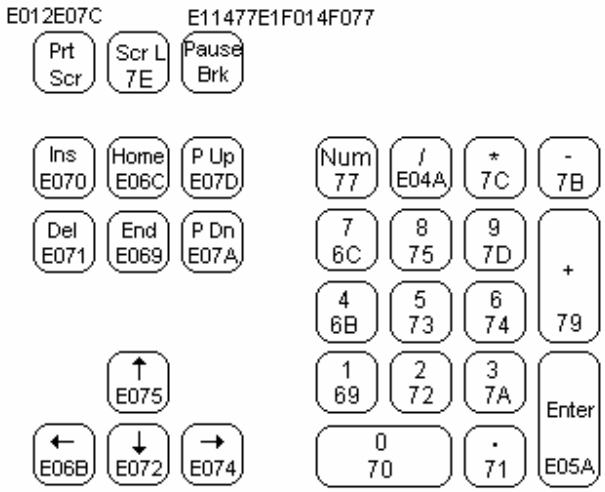
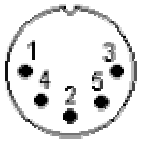


Fig. 3.6. Código de equivalencia par el teclado numérico y extendido.

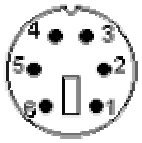
3.4.3. El conector del teclado. [32]

El teclado es conectado a equipo externo utilizando solamente cuatro conductores (Ver figura 3.7) Estos conductores se muestran para los modelos *"5 Pin Din Male Plug"* y "*PS/2 Plug"*.



5 Pin DIN

1. KBD Clock
2. KBD Data
3. N/C
4. GND
5. +5V (VCC)



PS/2

1. KBD Clock
2. GND
3. KBD Data
4. N/C
5. +5V (VCC)
6. N/C

Fig. 3.7. Descripción de los pines y conectores DIN y PS/2.

En ocasiones se puede encontrar un conector de cinco conductores. Este modelo fue hace tiempo implementado como el Reset del teclado, pero ahora ha sido descontinuado en los teclados AT. Ambos, los KBD clock y KBD data son líneas bi-direccionables de entrada salida a colector abierto. Si se desea, el Host puede hablar al teclado utilizando estas líneas. La mayoría de los teclados son especificados con un máximo de corriente de 300 mA a través de sus conductores.

3.4.4. El protocolo del teclado / lector de código de barras. [33]

Del teclado al host.

El teclado de una PC, utiliza comunicación bidireccional. El teclado puede enviar datos al Host y el Host puede enviar datos al teclado. El Host tiene la última prioridad sobre la dirección del envió de información. Es decir puede a cualquier tiempo enviar un comando al teclado.

El teclado es libre de enviar datos al Host solo cuando las líneas (ambos), el KBD Data y el KBD Clock se encuentran en alto. La línea del KBD Clock puede ser utilizada como una línea limpia para poder transmitir. Si el Host tiene la línea del KBD Clock en bajo, el teclado pasará cualquier dato hasta que el KBD Clock es liberada (se va a alto). Entonces, la línea de KBD Data irá a bajo, entonces el teclado se preparará para aceptar un comando del host.

[32] Ibid.
[33] Ibid.

La transmisión de datos en dirección del teclado al host es hecha con 11 bits. El primer bit es de inicio (lógica 0) seguido de 8 bits de datos (siendo el LSB el primero), una paridad de bit (odd parity) y un bit de alto (lógica 1). Cada bit puede ser leído en la bajada del reloj.

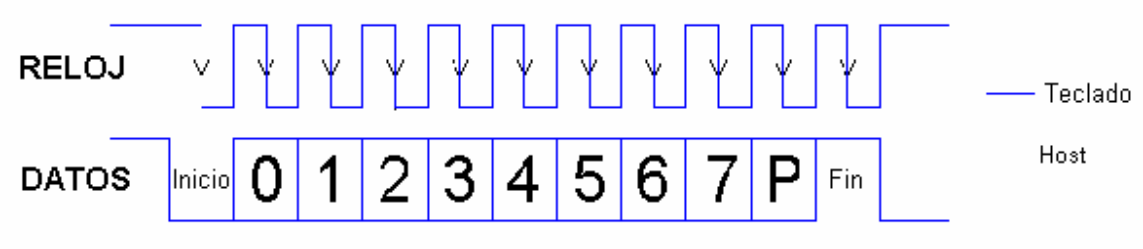


Fig. 4.8. Transmisión de datos del teclado al host.

La forma de onda que se muestra en la figura 4.8, representa una transmisión de un byte desde el teclado. El teclado generalmente no cambiará sus líneas de datos en el ciclo de subida del reloj como es mostrado en el diagrama. La línea de datos solamente debe ser válida en el ciclo de bajada del reloj. El teclado generará el reloj. La frecuencia de la señal de reloj se encuentra en un rango de 20 a 30 kHz. El bit menos significativo es enviado siempre al inicio.

Del host al teclado.

El protocolo del host al teclado es inicializado al tomar la línea del KBD Data en bajo. Sin embargo, para prevenir de enviar datos al mismo tiempo que se pretende se mande datos al teclado, es muy común el tomar la línea del KBD Clock en bajo por más de 60μs. Este es más que un bit de longitud. Entonces la línea de KBD Data es puesta en bajo, mientras que la línea del KBD Clock es liberada.

El teclado comenzará a generar una señal de reloj en la línea del KBD Clock. Este proceso puede tomarse hasta 10 ms. Después de que el ciclo de bajada es detectado se puede cargar el primer dato en la línea del KBD Data. Este bit será leído en el teclado en el siguiente ciclo de bajada, después del cual se puede posicionar el siguiente bit del dato. Este proceso es repetido por cada 8 bits de dato. Después de los datos de bits viene un bit de paridad odd.

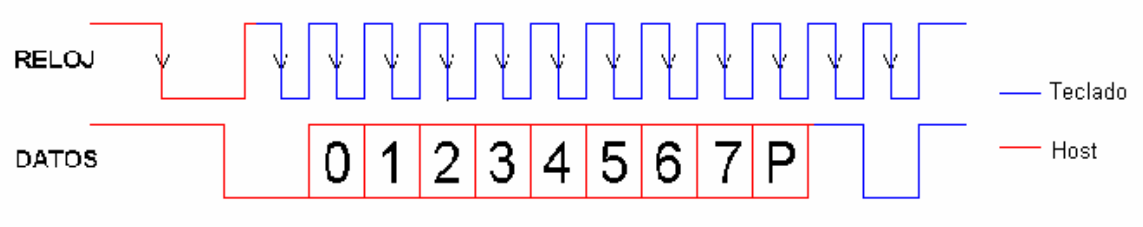


Fig. 3.9. Transmisión de datos del teclado al host.

Una vez que el bit de paridad ha sido enviado a la línea del KBD Data está en un estado alto para el siguiente ciclo de reloj, el teclado podría atender la recepción de nuevos datos. El teclado hace esto al tomar la línea del KBD Data en bajo para el siguiente ciclo de reloj en transmisión. Si la línea del KBD Data no está en este estado después del décimo bit (inicio, 8 bits de datos y uno de paridad), el teclado continuará por enviar una señal al KBD Clock hasta que la línea de KBD cambie de estado.

CAPÍTULO 4.

DISEÑO, DESARROLLO E IMPLEMENTACIÓN DEL PROYECTO.

4.1. Electrónica y programas de desarrollo para el "hand-held".

Durante la planeación y la realización de esta tesis se tuvo la necesidad de utilizar diferentes recursos que permitieran el desarrollo y la realización de la misma. En un principio, se requiere una tarjeta de desarrollo que esté basada en un microcontrolador M68HC11. En el mercado existen diferentes opciones para poder conseguir una tarjeta de este tipo, e inclusive uno mismo puede desarrollarla. Inicialmente se contó con una tarjeta de desarrollo ***M68HC11EVBU Universal Evaluation Board***, propiedad de Motorola muy limitada pero confiable, donde se desarrolló modularmente cada componente de la aplicación. En la Facultad de Ingeniería de la UNAM existen profesores que se han dedicado a fabricar este tipo de tarjetas, se ha decidido utilizar para el diseño final una tarjeta ya fabricada en lugar de crear una nueva que realmente se basaría en las ya existentes.

Para realizar cualquier aplicación sobre un microcontrolador M68HC11 es necesaria la utilización de diferentes programas que nos ayuden a cumplir este propósito. Básicamente se requieren un compilador, un simulador, un programa de monitoreo, que nos permita acceder al microcontrolador con programas y comandos simples, así como un programa que nos permita de igual forma ejecutar programas grabados en la memoria RAM o en la EEPROM del microcontrolador.

Es por esto que los siguientes programas fueron utilizados durante el desarrollo de este proyecto: el compilador ***IASM11.EXE***, el simulador ***AVSIM11.EXE***, los programas de monitoreo ***PCBUG11.EXE*** y ***WINBUG11.EXE*** y el programa ***ASBO.EXE*** que permitió grabar y ejecutar programas en las memorias del microcontrolador. Cada uno de ellos se describe breve y básicamente a continuación.[34]

Un compilador traduce programas en ensamblador a código de máquina. La velocidad de ejecución y el tamaño del código de máquina dependen del compilador. Algunos compiladores son rápidos pero gastan mucha memoria. Otros utilizan menos memoria pero tardan un poco más en compilar los programas. Sin embargo, para nuestros propósitos no es necesaria la forma más eficiente de compilación ya que el tiempo o espacio en memoria que nos podría ahorrar resulta verdaderamente insignificante.

El compilador que se utilizó es el ***IASM11.EXE*** (toma su nombre de The Integrated Assembler). IASM11 permite la edición de programas así como su

[34] NOTA: Una copia del software utilizado se podrá encontrar en el disco adjunto a este trabajo.

compilación creando archivos con extensión ***.S19,*** formato que utiliza el microcontrolador de Motorola. Los archivos ejecutables que contienen el código máquina de las instrucciones en ensamblador pueden estar en varios formatos. El formato mencionado de Motorola .S19 y el de Intel .HEX son los formatos principales. El formato .S19 es el más habitual y es el que generan los compiladores para microcontroladores M68HC11. El compilador IASM11 permite la creación de estos archivos y viene acompañado como software de apoyo para la realización de proyectos en estos microcontroladores.

Los archivos en formato .S19 están formados por líneas de dígitos hexadecimales. Es importante mencionar que cada línea del archivo comienza con los caracteres Sx, donde x puede ser el dígito 1 o 9. Existen por tanto dos tipos de líneas, las que comienzan con S1 y las que comienzan con S9. Cada tipo de línea tiene campos diferentes:

Línea del tipo S1:

Número de bytes línea	Dirección de comienzo	Campo de datos	Checksum
1 byte	2 bytes	X bytes	1 byte

Estas líneas constan de 4 campos. El primer campo consta de 2 dígitos hexadecimales (un byte) e indica el número de bytes que existen en la línea, sin contar el byte de este campo ni tampoco el código S1. El segundo campo, constituido por 4 dígitos hexadecimales (2 bytes), indica la dirección de comienzo a partir de la cual se debe situar en memoria el campo de datos. El tercer campo denominado campo de datos contiene los códigos máquina de las instrucciones y los datos a almacenar en memoria. Su longitud no está determinada. Finalmente, el último campo está constituido por un byte de checksum para detectar errores en la transmisión.

Líneas de tipo S9:

Número de bytes en la línea	4 dígitos de relleno	Checksum
1 byte	2 bytes	1 byte

Este tipo de líneas sólo aparecen para indicar el final de un archivo en formato .S19. El primer campo tiene la misma función que el de las líneas S1: indica el número de bytes de la línea sin contar el byte de este campo ni los dígitos S9. El siguiente campo contiene 4 dígitos de relleno, que normalmente son ceros. El último es un byte de checksum.

A continuación se presenta un ejemplo de un archivo en formato .S19 formado a partir de uno de los programas que se utilizó durante el desarrollo del proyecto:

S113B600CE10008630A72B4FA72C860CA72D01CE79
S113B610000018CE1000181F2E20FB18A62FA7001C
S113B620088C001026F02000CE000018CE10001860
S113B6301F2E80FBA60018A72F088C001026F020D0
S106B640CD20FE18
S9030000FC

Si se descomprimen las líneas en campos obtendremos:

S1	13	B600	CE10008630A72B4FA72C860CA72D01CE	21
S1	13	B610	000018CE1000181F2E20FB18A62FA700	1C
S1	13	B620	088C001026F02000CE000018CE100018	60
S1	13	B630	1F2E80FBA60018A72F088C001026F020	D0
S1	06	B640	CD20FE	18
S9	03	0000	FC	

El programa ***AVSIM11.EXE*** se empleó como una herramienta de trabajo para el desarrollo del programa del microcontrolador ya que permite realizar la simulación de los microcontroladores M68HC11 de Motorola. Simula la arquitectura del CPU incluyendo los temporizadores, las interrupciones y los puertos de entrada –salida. También permite depurar fácilmente programas.

El simulador despliega las condiciones de todos los registros, banderas, puertos y uso de áreas de memoria seleccionada durante la ejecución de los programas. La capacidad de depuración incluye el uso de puntos de interrupción dinámicos mediante condiciones, y ejecución paso a paso.

Se puede realizar la simulación directa de entradas y salidas al cambiar los valores de los puertos o se pueden usar archivos previamente creados que se cargan para simular las entradas al circuito. La salida del programa se puede salvar en un archivo para un análisis posterior.

Cuando el AVSIM11 se ejecuta, simula en memoria todo el hardware del M68HC11, incluyendo los registros, puertos y memoria. Al realizar una instrucción se puede observar mientras se ejecuta, el estado de la CPU y el efecto que se tiene sobre los registros, puertos y direcciones de memoria del microcontrolador simulado.

El AVSIM11 está provisto de un depurador simbólico en el que se pueden establecer puntos de interrupción, depuración paso a paso, comando “undo” y cambio de registros, memoria y banderas en cada operación. El AVSIM11 maneja la pantalla de despliegue como lo hace un editor de texto, permitiendo el movimiento hacia casi cualquier parte del circuito y modificar su contenido, para posteriormente correr la modificación y ver su efecto. También se puede configurar

para salvar automáticamente fragmentos de programa en un archivo, liberando al usuario de tener que recordar la modificación exacta que se realizó.

Con el fin de organizar las opciones disponibles, el AVSIM11 trabaja en dos modos:

- Modo Display: En modo “display”, el simulador funciona como un editor de textos, es posible mover el cursor a cualquier parte de la CPU, como son los registros, puertos o memoria y modificar el contenido de estas áreas.

- Modo Command: En el modo “command”, se tiene acceso a un menú de comandos localizado en la parte inferior de la pantalla. Esto permite cargar archivos, examinar la memoria, introducir código de programa, etc.

Para realizar el intercambio de modos, se debe presionar la tecla de “**ESC**“ (Escape). El simulador también está provisto de una serie de funciones que permiten controlar la manera de correr el programa. Estas funciones permiten establecer una operación a intervalos continuos o bien, paso a paso, deshacer operaciones, seleccionar la velocidad de simulación y las características de despliegue, establecer puntos de interrupción dinámicos, etc.

La pantalla del AVSIM11 funciona como una “CPU visual”, simula cada una de las características de la CPU y despliega su contenido. Como resultado, se tiene una pantalla repleta de información, en la que se deben distinguir dos áreas importantes: una que posee la información de registros y aquella en la que se encuentran los valores de la CPU. Dentro de cada región, cada objeto, ya sea registro o pin, se trata como una ventana por separado. Con las teclas de movimiento de cursor se puede desplazar de una ventana a la otra.

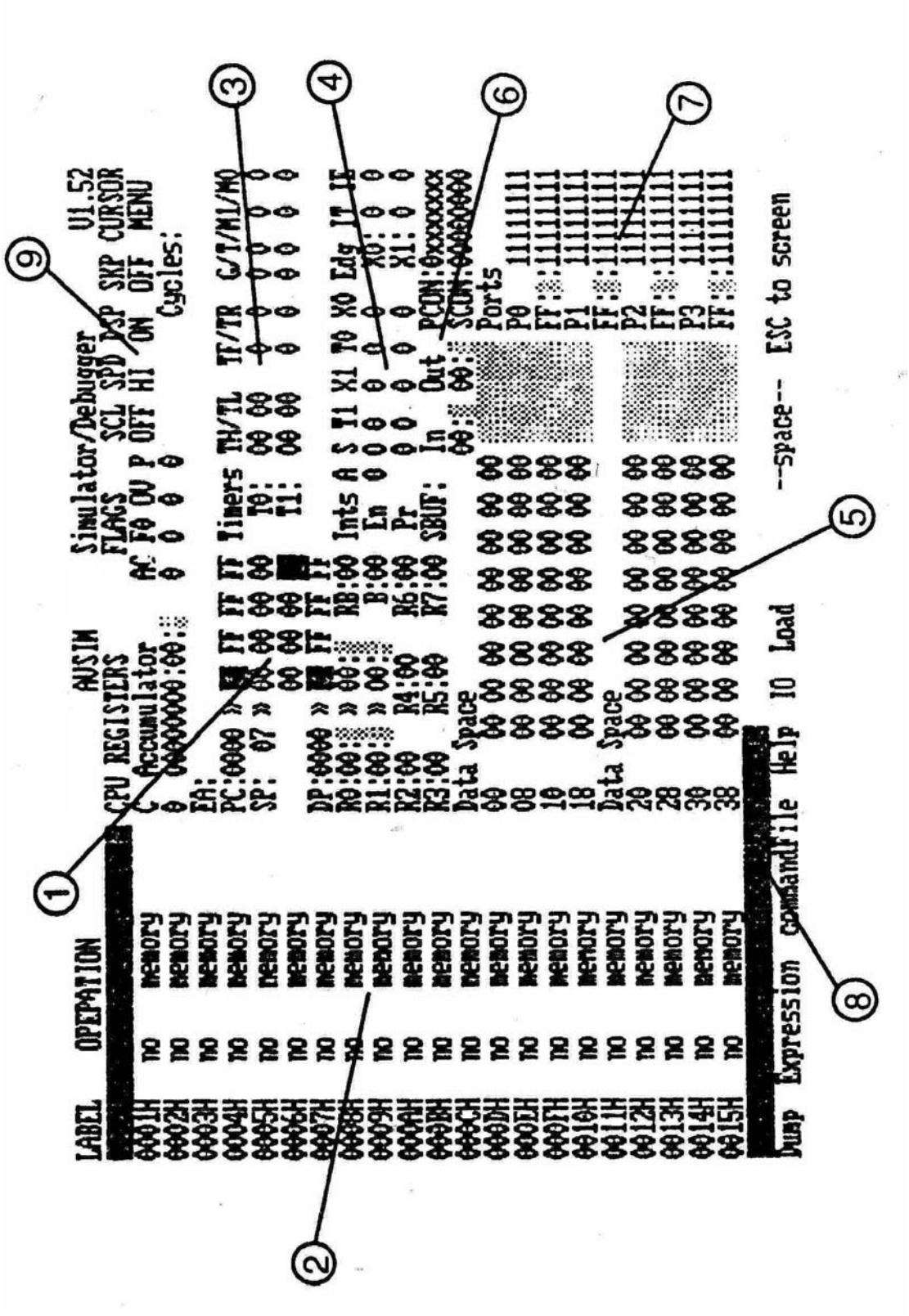


Figura 4.1. Pantalla del simulador AVSIM11.

Donde:

1. *Registros y Banderas.* Esta región despliega el contenido de cada registro asociado a la CPU y presenta el estado de cada bandera en la CPU.

2. *Programa.* Esta área despliega la dirección, etiqueta, codificación y argumentos de la porción del programa que está siendo ejecutado. Mediante la combinación ALT + F5 se intercambia el despliegue para mostrar los símbolos o sus valores numéricos.

3. *Temporizadores.* Esta área muestra las características y el contenido de cada uno de los temporizadores.

4. *Interrupciones.* Esta área muestra las características de las interrupciones.

5. *Áreas de memoria.* Estas áreas se usan mediante el comando “Dump” para desplegar el contenido de memoria en hexadecimal y caracteres ASCII.

6. *Puerto Serial.* Esta área muestra el contenido de los buffers de recepción y transmisión del puerto serial en hexadecimal y ASCII.

7. *Puertos.* Esta área muestra el contenido de cada uno de los puertos de la CPU. Las características del “latch” se encuentren al lado del nombre del puerto. El contenido del puerto se presenta debajo de las características del “latch” hexadecimal, caracteres ASCII y bits.

8. *Área de comandos.* Esta área despliega los menús utilizados para controlar el simulador. Se encuentra localizada en las dos últimas líneas de la pantalla.

9. *Status del simulador.* Esta área despliega el status del simulador como lo es la velocidad de simulación. El contador de ciclos es un pseudo-registro que se puede editar como cualquier otro registro. Incluso se pueden establecer puntos de interrupción en el contador. Para limpiar el contador, se debe mover al pseudo-registro de ciclos con CTRL y presionar CTRL HOME.

Las funciones de las teclas más importantes son:

NOMBRE	TECLA	DESCRIPCIÓN
GO	F1	Inicia y termina la simulación de la CPU
BKUP	F2	Mueve el breakpoint una instrucción arriba en el programa
BKST	F3	Establece un breakpoint dinámico en el lugar en que se encuentra el cursor
BKDN	F4	Mueve el breakpoint una línea abajo.

SPD	F5	Selecciona la velocidad de ejecución de la simulación (HI, Médium, LOW)
DSP	F6	Inicia y detiene la actualización continua del despliegue
WMD	F7	Establece el tipo de ventana (BINaria, HEX o ASCII) a la cual se moverá el cursor cuando la ventana muestre datos en más de una base
SKP	F8	Activa o desactiva (ON, OFF) el interruptor de salto en subrutinas. Si se está operando en modo paso a paso y el interruptor está en ON no ejecutará la subrutina paso a paso. Cuando está en OFF sí la ejecutará paso a paso.}
UNDO	F9	Deshace el último comando
STEP	F10	Ejecuta una instrucción cada vez que se presiona esta tecla, es decir, ejecuta el programa paso a paso. No toma en cuenta los *breakpoints*
TRACE	ALT+F6	Activa o desactiva la actualización de las ventanas sin importar el estado del apuntador de la pila
SCLTOG	ALT+PgUp	Activa y desactiva el modo “Srcool”. Cuando está desactivado (OFF) las teclas de movimiento de cursor se mueven libremente en la pantalla. Cuando está activado (ON), el cursor se mueve únicamente en la ventana actual

Tabla 4.1. Descripción de las funciones más importantes del AVSIM11.

El programa M68HC11 ***PCBUG11.EXE*** es un paquete de software utilizado para acceder de forma fácil y experimentar con los microcontroladores M68HC11. PCBUG11 permite programar cualquier miembro de la familia de microcontroladores M68HC11 y examinar el comportamiento de periféricos internos bajo condiciones específicas. Además, permite correr cualquier programa en el microcontrolador, procesar breakpoints y traces o pistas.

Para correr el PCBUG11 se debe primeramente polarizar la tarjeta que contenga nuestro microcontrolador y posteriormente conectar la tarjeta al puerto de comunicaciones predilecto de la computadora. Es entonces cuando se puede ejecutar el programa de monitoreo. PCBUG11 permite diferentes opciones para su uso: es posible escoger el puerto de comunicaciones, el cristal utilizado en la tarjeta, así como utilizar cualquier macro que determina cuáles opciones están permitidas para un microcontrolador determinado. Para correr el software es recomendable utilizar la computadora en modo MS-Dos para no crear problemas con el sistema operativo.

El estado del hardware aparece dentro de la pantalla de la computadora. Esta pantalla consiste en cuatro áreas principales:

a) Ventana principal: Esta ventana es la parte superior y ocupa la mitad de la pantalla. Tiene fondo color azul y el texto aparece en color blanco. Esta

pantalla muestra información de la operación de PCbug11: resultados de comandos, contenidos de memoria, códigos de ensamblaje y demás operaciones que se puedan realizar y de las que se pueda mostrar información.

b) Ventana de registros: Esta ventana se encuentra en el centro de la pantalla. Tiene fondo color rojo y texto en color blanco o amarillo. Esta ventana muestra el último contenido grabado de los registros del procesador. Cabe notar que los valores de la ventana de registros se actualizan solamente después de iniciar el programa o por petición del usuario. Los comandos del programa de monitoreo permiten modificar el contenido de los registros.

c) Ventana de estado: Esta ventana se encuentra en la parte central derecha de la pantalla. Tiene un fondo color púrpura y texto blanco. Esta ventana muestra el microcontrolador en uso, el estado del microcontrolador (corriendo, detenido, etc.); el estado de la línea de comunicaciones y el conjunto actual del usuario de los vectores de interrupciones.

d) Ventana de comandos: Esta ventana se encuentra en la parte inferior izquierda de la pantalla. El fondo de esta ventana es de color negro con texto blanco. Se utiliza esta ventana para introducir y leer comandos para PCbug11. El cursor de comandos (carácter >>) se puede localizar en la parte inferior de esta ventana; los comandos que el usuario ingresa aparecen después de este carácter. Los comandos previos y el último mensaje de error también suelen aparecer en esta ventana. Si esta ventana no se muestra, el programa no se está ejecutando correctamente. Un mensaje de error de DOS o de PCbug11 indican normalmente el problema.

Existen dos ventanas adicionales, pero su diferencia es que son temporales y aparecen superpuestas sobre la ventana principal:

1. Ventana de error: Esta ventana indica cualquier error o cualquier problema de operación o de comunicación con el microcontrolador. Esta ventana tiene fondo de color negro y texto en rojo. Para borrar inmediatamente esta ventana se presiona cualquier tecla o se espera un promedio de cinco segundos y la ventana se cierra sola.

2. Ventana de ayuda: Esta ventana despliega información general pedida a través del comando HELP. Normalmente tiene un fondo color negro y texto en blanco. Para moverse a través de este archivo de información se utilizan las teclas Re-Pág y Av-Pág o simplemente las flechas que se encuentran a la izquierda del teclado numérico. Para cerrar la ventana de ayuda, presione la tecla ESC.

El programa de monitoreo ***WINBUG11.EXE*** es muy parecido a PCbug11 pero crea una interfase más amigable y menos tosca que este último. Winbug11 permite una interfase visual a través de ventanas y está basado y diseñado para trabajar en ambiente Windows.

Al tener este mismo tipo de plataforma, las opciones de abrir archivos, editar y depuración de los programas se realiza con la barra de menú que se encuentra en la parte superior de la ventana principal. Es muy semejante su uso a cualquier software utilizado en la arquitectura de Windows y su aprendizaje es muy fácil con tan sólo explorar entre las opciones existentes, por lo que no se abundará más al respecto.

El programa ***ASBO.EXE***, desarrollado en la Facultad de Ingeniería de la UNAM, permite conectar al microcontrolador M68HC11 a la PC en modo bootstrap. Mediante este programa es posible realizar funciones de carga y ejecución de programas tanto en la RAM como en la EEPROM del microcontrolador, así como revisión y modificación de localidades de memoria. Dicho programa utiliza un pequeño programa monitor llamado PUMMA también desarrollado en la Facultad de Ingeniería por el profesor Antonio Salvá Calleja.

Es este programa el que ha sido más utilizado en el desarrollo de este proyecto debido a su fácil interfase con el microcontrolador y la practicidad que demuestra su manejo y configuración, así como también la muy práctica y fácil programación de la EEPROM del M68HC11.

4.2. Desarrollo del proyecto.

El proyecto consiste en un dispositivo de adquisición de datos que sea capaz de mantenerlos en memoria para posteriormente enviarlos a una computadora personal. Esto conlleva a conformarlo en dos partes. Lo primero es la forma de adquisición de datos y lo segundo el traslado de los datos de memoria a la PC. El problema de lectura de los datos se controlara mediante el lector de códigos de barras. Este dispositivo envía los datos en la misma forma que un teclado a la computadora, es decir, el lector manda los datos en la misma forma que el teclado; así, tenemos que su conector es equivalente y funciona de la misma manera, por lo que se debe plantear cómo capturar estos datos dentro del microcontrolador y almacenarlos en la memoria de éste, mientras que para la segunda parte del proyecto se tiene que el microcontrolador envía los datos almacenados en memoria a la PC, lo que implica utilizar un programa de comunicación vía puerto serial entre el microcontrolador y la PC. Esto es, el microcontrolador posee una interfase de comunicación serial asíncrona que es muy funcional y tradicional en el uso de computadoras por lo que es ésta la que se utiliza y la PC tendrá que recibir los datos por el puerto serial. Para este proceso se utiliza programación en **DELPHI** para la interfase de comunicación de la PC.

Todo el desarrollo se llevó a cabo mediante diferentes pruebas aplicadas al MCU usando programas diseñados para probar parte por parte las necesidades del proyecto. Es por esto que, antes de continuar con la especificación de dicho procedimiento, se realiza una breve explicación de cómo está estructurado un programa escrito en lenguaje ensamblador y las diferentes formas de ejecutar este programa ya sea en memoria RAM o desde la memoria EEPROM del MCU que permite realizar funciones del MCU en forma automática una vez alimentada la tarjeta de desarrollo sin la necesidad de volver a transferir el programa al MCU, hecho que resulta obvio al conocer las características de dichas memorias.

Posteriormente se describirán poco a poco y con ejemplos sencillos explicados en detalle la forma en que las fases de desarrollo se han realizado empezando por la interfase serial, continuando con la lectura del teclado y de esa forma la lectura de datos del lector de códigos de barras, mostrando la interfase con un LCD (Liquid Crystal Display) para visualizar los dato del código de barras y finalmente la integración para tener la aplicación general.

4.2.1. Carga y ejecución autónoma de programas en el M68HC11.

Es necesario describir a grandes rasgos un programa .asm sencillo escrito en lenguaje ensamblador que sirva de ejemplo para explicar cómo se pueden realizar pruebas y cargas de dicho programa en la memoria del MCU.

El código del programa se muestra a continuación:

```
        ORG   $0000
INICIO  LDAA  #$00
LOOP    STAA  $1004
        INCA
        BSR   TIME
        BRA   LOOP
TIME    LDX   #$00
MALLA   INX
        BNE   MALLA
        RTS
```

El programa simplemente es un contador cuya salida puede ser vista en el puerto B del MCU. Normalmente se le conectan a los bits de salida unos leds para poder hacer la visualización más práctica y fácil. El MCU encenderá el bit menos significativo del puerto B e irá incrementando sin parar y desbordando el bit más significativo al llegar al final para volver a empezar la cuenta.

Las instrucciones utilizadas forman parte del juego de instrucciones del MCU la cuales se pueden consultar en el manual de referencia del mismo, así como los modos de direccionamiento que se han descrito en el apéndice correspondiente a la estructura del microcontrolador M68HC11 y las direcciones utilizadas en el programa, sabiendo solamente que la dirección $1004 corresponde al puerto B.

Sin embargo, existe un término del cual es muy importante su análisis: la instrucción **ORG $0000**. Esta instrucción lo único que realiza es informar al MCU a partir de qué dirección de la memoria colocar este programa. Se ha visto en este capítulo la necesidad de utilizar un compilador para estos programas, así, el compilador creará un archivo objeto .S19 que es el que se le transfiere al MCU a través de un programa de monitoreo. Es importante mencionar que el 0 después de la instrucción ORG significa que el programa se cargue en la memoria del MCU a partir de la dirección $0000 de éste. En este ejemplo, el programa se escribirá en la memoria RAM a partir del inicio de ésta, en la dirección $0000. Sin embargo, existe un inconveniente. Si se utiliza como programa de monitoreo el PCbug11 o el Winbug11, se necesita dejar espacio libre en memoria RAM para que dichos programas puedan interactuar con el MCU. Ambos requieren la instalación y ejecución de un pequeño programa denominado ***"Talker"***, que es el que permite la comunicación entre la PC y el MCU. Por lo anterior, es recomendable, si se utilizan estos programas de monitoreo, direccionar el inicio del programa a partir de la dirección $100 de la memoria RAM del MCU. Se puede pensar que esto no es funcional, ya que la mayoría de los MCU de esta familia poseen 512 bytes de memoria RAM y 100 en base hexadecimal representa 256 en base decimal, lo cual deja solamente al MCU la posibilidad de manejar 256 bytes en esta memoria, sin embargo es el mismo *"Talker"* quien se encarga de hacer el arreglo correspondiente y por lo tanto es transparente para el usuario.

Se puede deducir entonces, que es posible direccionar un programa a cualquier localidad de memoria del MCU, es recomendable hacerlo en la memoria RAM si se trata de programas pequeños que no rebasen de 256 bytes. Si se requiere más espacio en memoria para el programa en cuestión o si se requiere que el MCU opere en forma automática sin la necesidad de comunicarse con la PC, entonces se puede direccionar la escritura del programa a partir de la dirección $B600 que corresponde al inicio de la EEPROM de la mayoría de los miembros de la familia M68HC11, y que es la que se utiliza en el desarrollo de este proyecto al utilizar la versión E9 del M68HC11.

En general, se ha utilizado más que los programas de monitoreo el programa ASBO.EXE descrito en la sección anterior. Dicho programa permite tanto la ejecución de un programa originado desde la dirección inicial de la RAM ($0000) del MCU como la ejecución y programación de la EEPROM cuya dirección ya mencionada es la $B600. De esta forma es posible realizar pruebas básicas en memoria RAM para desarrollar diferentes aplicaciones programadas en el MCU y también es posible el programar la EEPROM del MCU para que pueda funcionar nuestra aplicación de manera autónoma. Para hacer esto, es muy importante tomar en cuenta que una vez programada la EEPROM es necesario realizar un cortocircuito en los pines PD0 (Rx) y PD1 (Tx) del MCU para que se redireccione el inicio del programa a la dirección inicial de la EEPROM, debiendo ésta estar activa según el registro CONFIG del MCU. Dicho cortocircuito consiste solamente en juntar estos dos pines mediante un *"jumper"* . Sin embargo, debe tomarse en cuenta que esto depende de la aplicación que se realice. Si en el programa se utilizan estos pines para la comunicación serial no es posible mantenerlos en

cortocircuito debido a que no se podrá realizar ninguna transmisión o recepción de datos, por lo que cabe aclarar que el MCU solamente necesita de un instante para direccionar la ejecución de un programa a la EEPROM, por lo que aplicando un corto momentáneo a estos dos pines es más que suficiente para la ejecución autónoma de programas.

Por lo anterior, con solamente modificar el código del programa del contador en su primera línea por **ORG $B600**, la carga y ejecución de este programa se realizará a partir de la programación de la EEPROM. Si se requiere la ejecución autónoma de este programa, basta con seguir lo mencionado en el párrafo anterior.

Ahora se comenzará con la descripción de las fases de desarrollo del proyecto en forma de bloques.

4.2.2. Diagrama general del proyecto.

Antes de describir los diferentes programas, así como las interfases utilizadas en el desarrollo de este trabajo, es necesario explicar en forma esquemática las interconexiones de los diferentes componentes que conforman al sistema **"HAND HELD**". La descripción se realiza de la siguiente manera: un diagrama que representa la interfase de entrada (figura 4.2.a), otro para la interfase de salida (figura 4.2.b), uno para esquematizar como es el sistema completo (figura 4.2.c), y finalmente un diagrama de conexiones entre el lector, el MCU y la PC (figura 4.2.d).

Como se ha mencionado, en la figura 4.2.a se muestra el sistema de bloques representando al lector de códigos de barras y su comunicación con el MCU que se utilizó en el proyecto, señalando la interfase realizada con el microcontrolador y su display que muestra el código leído. Es necesario mencionar que existe un programa para utilizar la memoria del MCU el cual almacena el código. Este módulo representa parte del sistema portátil del proyecto donde se efectúa la fase de lectura o adquisición de datos.

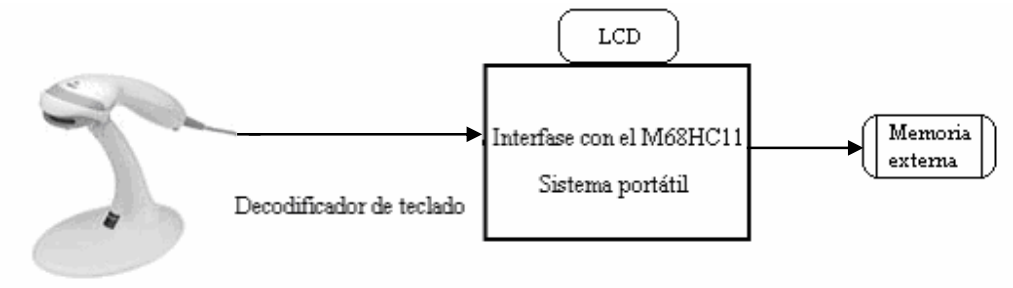


Figura 4.2.a. Diagrama de bloques del sistema portátil. (Fase lectura / adquisición).

En la figura 4.2.b se observa la segunda parte del proyecto que representa la fase de transferencia de datos que entrega los datos guardados en la memoria del microcontrolador a la PC, esto se realiza mediante la interfase de comunicación serial asíncrona del microcontrolador y recibido por el puerto serial Com1 de la PC. Los datos son los códigos de barras leídos los cuales son recibidos mediante un programa en la PC utilizando programación en Delphi, la forma en que realiza la captura de los datos es mediante el puerto serie. Posteriormente, estos datos se procesarán a través de este mismo programa para que se actualice la base de datos de la aplicación.

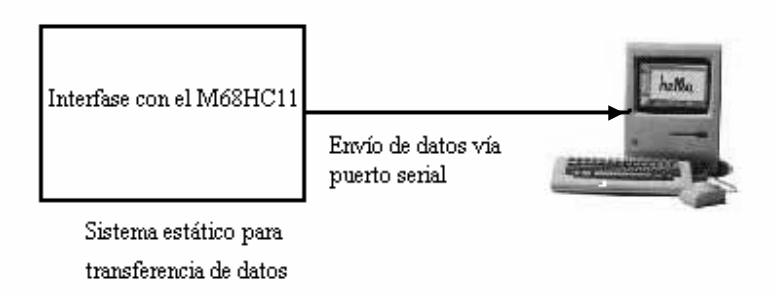


Figura 4.2.b. Diagrama de bloques del sistema de transferencia de datos vía puerto serial. (Fase transferencia).

La figura 4.3.c esquematiza estas dos fases interconectadas en forma simultánea que es otra forma de utilizar este sistema.

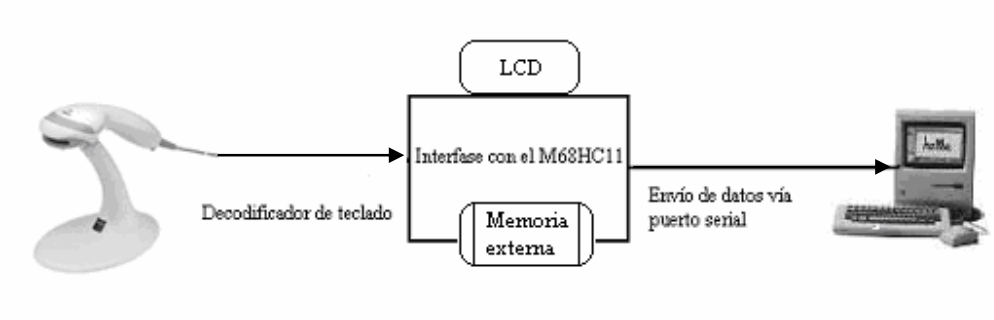


Figura 4.2.c. Diagrama de bloques completo de ambas interfases.

En la figura 4.2.d, se muestra el diagrama de interconexiones que se ha implementado para poder realizar el proyecto final como es: la lectura del código de barras mediante el lector hand held y la comunicación serial por medio de un cable DB25 – DB9. Como se puede observar una de las principales interfases a utilizar es el puerto serial de los diferentes puertos del MCU y de la computadora por lo que se empezaría con la descripción de éste.

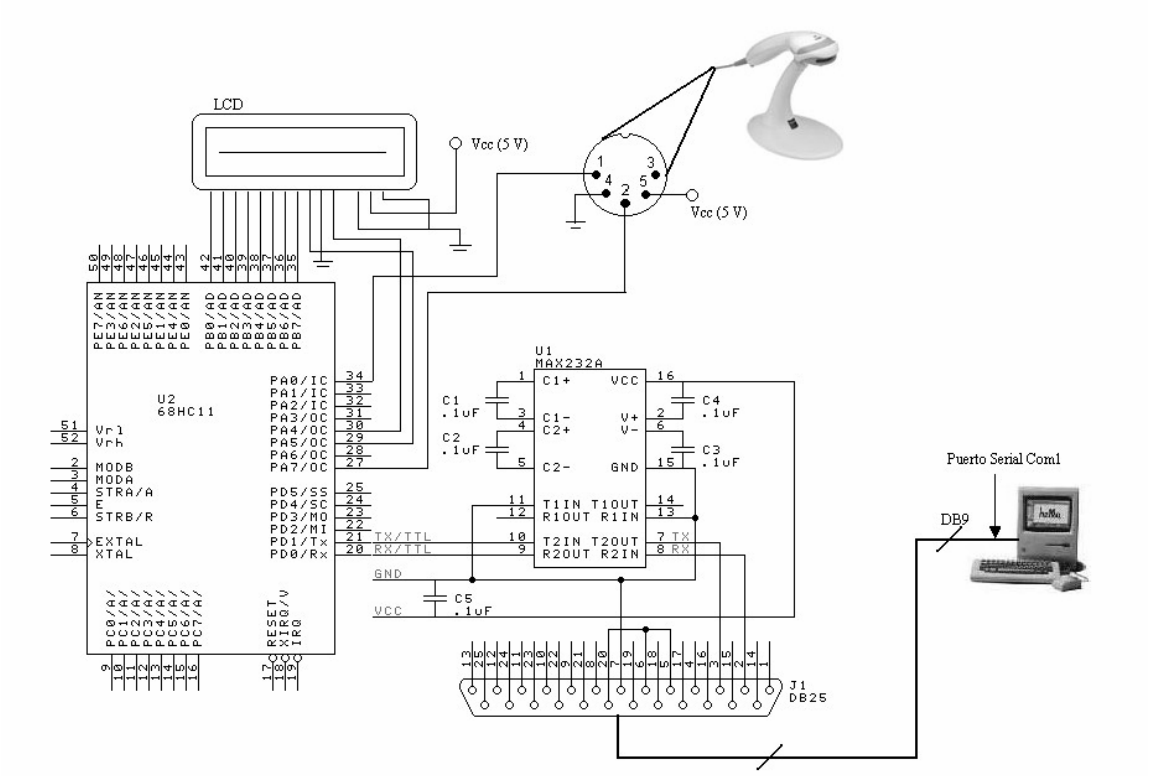


Figura 4.2.d. Diagrama de interconexiones entre el microcontrolador y la PC.

4.2.3. Interfase serie.

En el capítulo dos se describen las formas en que el microcontrolador MC68HC11 se comunica con el exterior, por lo que, con la aplicación a desarrollar, se mencionarán los conceptos relacionados y nombres directos de los registros del microcontrolador en forma continua. Como ya se ha mencionado, se utiliza la interfase de comunicaciones serial asíncrona del MCU (SCI).

Se comenzó por realizar pruebas de comunicación de puertos serial para conocer el funcionamiento del MCU en este aspecto. Para ayudar en estas pruebas se ha hecho uso de un programa que el sistema operativo “Windows”, el más común en la mayoría de las computadoras personales, que ofrece desde sus primeras versiones, el programa HyperTerminal.

La HyperTerminal es un producto pequeño y fácil de utilizar, diseñado para satisfacer las comunicaciones mediante terminal básicas que pueda necesitar. Hilgraeve Inc. escribió HyperTerminal para Microsoft, y se encuentra incluido en la instalación de “Windows” dentro de las comunicaciones ofrecidas como accesorios. Este programa permite realizar comunicaciones directamente desde los puertos Com1, Com2, Com3, Com4 o directamente a cualquier fax – modem que se encuentre instalado en una PC. Es posible configurar los puertos a diferentes velocidades así como el protocolo de comunicaciones que implica el

número de bits de datos en una palabra, la paridad, número de bits de fin y control de fluyo de los mismos. Se puede enviar y recibir archivos completos e incluso capturarlos en impresora.
Para nuestro propósito, HyperTerminal se ha utilizado en los puertos seriales Com1 y Com2.

Para la comunicación serial asíncrona de la PC se sigue el estándar RS-232 descrito en la sección 3.3.2.3 de esta tesis. Este estándar representa los 1's entre –3 hasta –20 volts y los 0's entre +3 y +20 volts, siendo los valores de voltaje más utilizados para este propósito –12 [V] y +12 [V], respectivamente. Esto ofrece un intervalo mayor de voltaje entre cada valor diferente así como un cruce por cero que brindan una inmunidad más grande al ruido que los niveles de voltaje TTL. Sin embargo, los niveles de voltaje que el MCU maneja son TTL, es decir, representa los unos con 5 [V] y los ceros con 0 [V]. Por lo anterior, es necesario cambiar estos niveles de voltaje para que sean compatibles con el estándar RS-232 a través de un circuito de adaptación que realice esta función.

Se puede realizar un arreglo de comparadores de voltaje en base a amplificadores operacionales para solucionar este problema, sin embargo, existe un circuito integrado que realiza esta adaptación. Este circuito es el MAX232 que es el más comúnmente utilizado, aunque también se puede usar el MAX233 que realiza la misma función sin la necesidad de elementos externos como capacitores. Dicho circuito es un convertidor de voltajes RS-232 a TTL y posee como una de sus grandes ventajas que no hace falta el uso de alimentaciones externas, diferentes a las habituales para el uso del MCU que generalmente es de 5 [V], evitando de esta manera aumentar la complejidad de los sistemas de alimentación.

La figura 4.3 muestra un diagrama de la estructura interna del MAX232, así como de la conexión necesaria para realizar este propósito. Los capacitores mostrados son, para el MAX232, de 1 [μF] y, para el MAX232A, de 0.1 [μF].

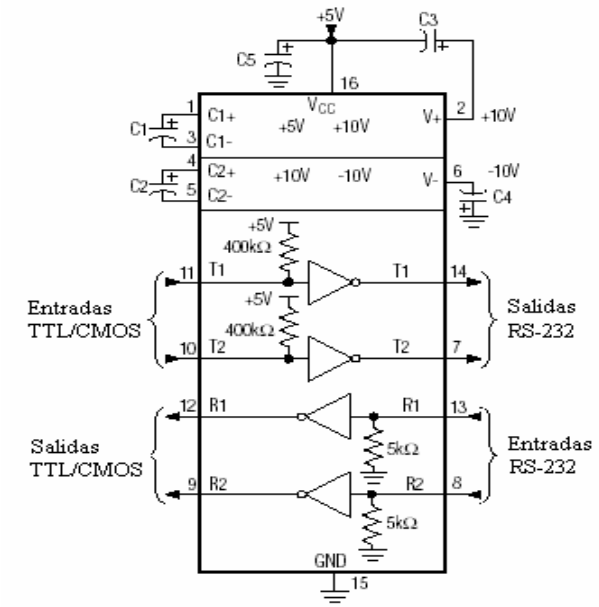


Figura 4.3. Configuración de pines y circuito armado típico del MAX232 y MAX232A.

La figura 4.4 muestra cómo se conecta este dispositivo a un microcontrolador M68HC11 y a un conector DB25 que se puede utilizar para la conexión serial con la computadora. Dicho conector puede ser cambiado por un conector DB9 hembra que es mucho más común en el uso de los puertos seriales de una PC.

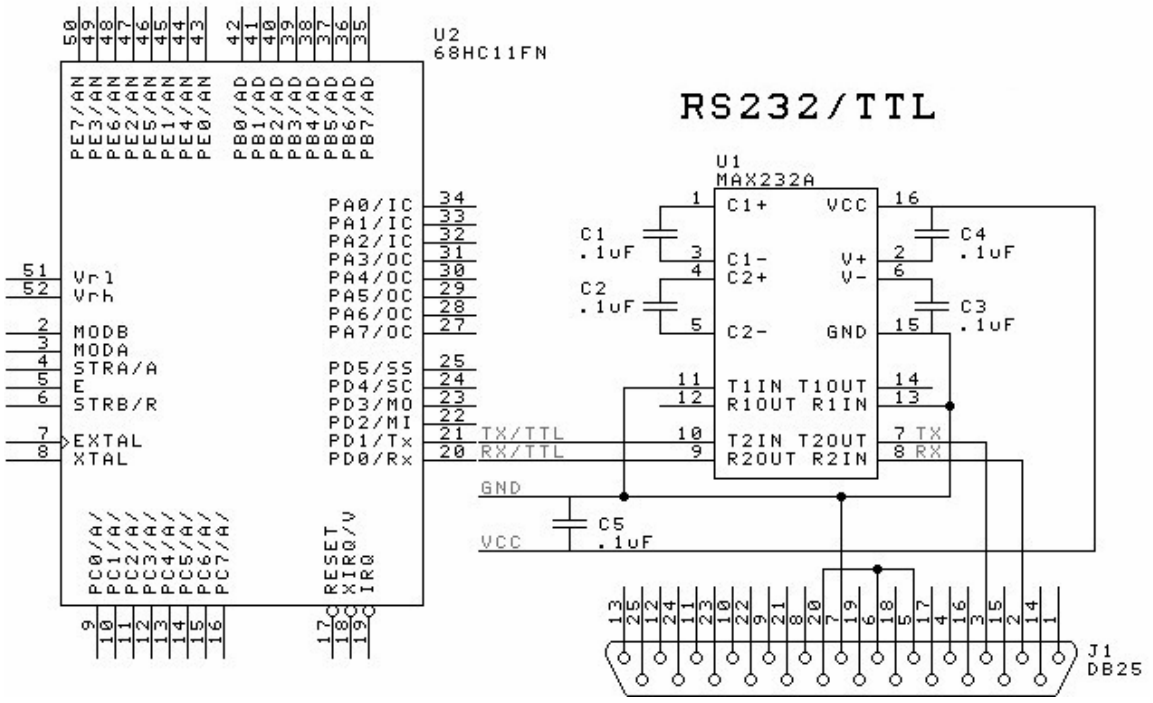


Figura 4.4. Diagrama de conexiones entre un M68HC11, un MAX232A y un conector DB25.

Considerando lo anterior, se podría deducir que se debe crear esta interfase a través de los pines de transmisión serial asíncrona que posee el MCU que se encuentran en el puerto D y son PD0 (Rx) y PD1 (Tx). Sin embargo, esto no es necesario del todo, ya que las tarjetas de desarrollo normalmente cuentan con este arreglo para permitir la comunicación entre el MCU y una PC común. Esto significa que la interfase que permite al MCU interactuar con los programas de desarrollo como el PCbug11 utiliza este medio para hacerlo. Por lo anterior, en cuanto a hardware, no se necesita crear nuevos arreglos electrónicos para poder comunicarse con la PC.

Contando entonces con este diseño ya instalado en la tarjeta, se puede comenzar a crear programas que nos permitan comunicarnos con la PC y ver los resultados de la transmisión o recepción de datos en la pantalla de HyperTerminal.

Mediante un programa básico de comunicación para el MCU, el cual permita únicamente probar esta interfase. Este programa simplemente Hará la transmisión serial de la palabra 'HOLA' una infinidad de veces.
Programa de Transmisión serial de la palabra 'HOLA' una infinidad de veces.

```
                ORG   $0000

REGBAS          EQU   $1000
BAUD            EQU   $2B
SCCR1           EQU   $2C
SCCR2           EQU   $2D
SCSR            EQU   $2E
SCDR            EQU   $2F

                LDX    #REGBAS          ;X=$1000
                LDAA  #$30
                STAA  BAUD,X            ;programa la velocidad de trans. = 9600 baud
                CLRA
                STAA  SCCR1,X           ;longitud de palabra: 1 sb, 8 db, 1 sb
                LDAA  #$0C
                STAA  SCCR2,X           ;Tx = Rx = 1 => SCI ACTIVADO

                LDX    #$50
                LDAA  #$48              ;'H'=$48
                STAA  $0,X
                INX
                LDAA  #$4F              ;'O'=$4F
                STAA  $0,X
                INX
                LDAA  #$4C              ;'L'=$4C
                STAA  $0,X
                INX
                LDAA  #$41              ;'A'=$41
                STAA  $0,X

                LDX    #$50
                LDY    #REGBAS
```

```
AQUI        BRCLR SCSR,Y,$80,AQUI
            LDAA  $0,X
            STAA  SCDR,Y
            INX
            CPX   #$54
            BNE   AQUI
            LDX   #$50
            BRA   AQUI
```

El primer bloque de instrucciones de este programa solamente establece las direcciones base que se utilizan en él. REGBAS para hacer uso del direccionamiento indexado en el código y controlar las direcciones de los registros utilizados en el mismo.

Después se encuentra un pequeño bloque de instrucciones que programan la comunicación serial asíncrona del MCU y de esta manera el puerto D funciona para este propósito. Se selecciona la velocidad de transmisión, la longitud de la palabra, su paridad y bits de inicio y fin y finalmente se activa la función de comunicación asíncrona.

Posteriormente se graba en la memoria RAM a partir de la dirección $50 y hasta la dirección $54 la palabra “HOLA” y se envían estos datos escribiéndolos carácter por carácter en el registro SCDR para su envío por comunicación serial asíncrona.

Con esto, se ha probado una parte de la comunicación serial, la transmisión. A continuación se presenta otro pequeño ejemplo que permite comprobar el funcionamiento de la recepción de datos y para saber que se han recibido de manera correcta se vuelven a enviar por la misma interfase.
La diferencia con el programa anterior, que también caracterizará a los ejemplos posteriores presentados en este trabajo, es que se graba y ejecuta dicho programa desde la EEPROM a partir de la dirección $B600.

```
            ORG   $B600

REGBAS      EQU   $1000
BAUD        EQU   $2B
SCCR1       EQU   $2C
SCCR2       EQU   $2D
SCSR        EQU   $2E
SCDR        EQU   $2F

            LDX    #REGBAS          ;X=$1000
            LDAA  #$30
            STAA  BAUD,X            ;programa la velocidad de trans. = 9600 baud
            CLRA
            STAA  SCCR1,X           ;longitud de palabra: 1 sb, 8 db, 1 sb
            LDAA  #$0C
            STAA  SCCR2,X           ;Tx = Rx = 1 => SCI ACTIVADO

REC         LDX   #$0000
```

```
                LDY   #REGBAS

RECIBE          BRCLR SCSR,Y,$20,RECIBE
                LDAA  SCDR,Y
                STAA  $0,X
                INX
                CPX   #$0010
                BNE   RECIBE
                BRA   ENVIO

ENVIO           LDX   #$0000
                LDY   #REGBAS
MANDA           BRCLR SCSR,Y,$80,MANDA
                LDAA  $0,X
                STAA  SCDR,Y
                INX
                CPX   #$0010
                BNE   MANDA
                BRA   REC
```

La primera parte de este programa es idéntica a la programación anterior, se trata simplemente de la asignación de direcciones y programación de la comunicación serial asíncrona.

La segunda parte a partir de la etiqueta REC y hasta antes de la etiqueta ENVIO especifica en primer lugar las direcciones de importancia en el programa y la recepción de datos desde el registro SCDR y guardándolos a partir de la dirección $00 y hasta la dirección $10 que cuando se ocupa el programa continúa su flujo y entonces empieza a enviar los datos a partir de la dirección $00 y hasta la dirección $10, retornando nuevamente a la espera de recepción de datos. Con esto, se puede entender que este programa lee o recibe 16 caracteres por la interfase serial asíncrona del MCU guardándolos en memoria RAM a partir de $00 y posteriormente los envía a través de la misma interfase para volver a esperar la recepción de datos.

Con estos pequeños pero muy didácticos programas se puede comprender y probar de manera fácil la interfase de comunicación serial asíncrona del M68HC11. Obviamente, los resultados y los datos enviados al microcontrolador se realizan mediante una PC a través del programa ya descrito HyperTerminal, con la característica de no incluir el eco local de la terminal en su configuración para poder apreciar el funcionamiento de los programas.

4.2.4. Lectura del teclado.

El funcionamiento del teclado ha sido ya descrito en el capítulo anterior, por lo que en esta sección se revisarán únicamente los programas que han permitido reconocer su protocolo de envío de datos.

El lector de códigos de barras Metrologic Voyager MS-9500 transmite los códigos leídos a través de un conector DIN, lo cual envía la información de igual

manera a como lo realiza un teclado de cualquier computadora personal. Un esquema de este conector se muestra en la figura 4.5.

1. KBD Clock
2. KBD Data
3. N/C
4. GND
5. +5V (VCC)

Figura 4.5. Diagrama de conexiones de un conecto DIN.

Como se puede ver en la descripción del diagrama de este tipo de conectores a pesar de que se tienen cinco conductores, solamente cuatro son necesarios y de estos cuatro, dos son utilizados para la polarización del dispositivo que se toma directamente de la fuente de la PC. Los otros dos conductores son los que interesan. El pin 1 es un reloj que el mismo dispositivo genera para sincronizar los datos enviados. En un teclado, esta línea siempre está en alto, y baja a partir que es oprimida una tecla. Los datos o código generado al oprimir esta tecla se transmite por el pin 2.

Para la aplicación que se pretende realizar, no es importante considerar todas las teclas que un teclado estándar posee, sino solamente las alfanuméricas. Esto se debe a que el lector no tiene registrados estos códigos, como podrían ser CTRL-ALT-SUPR, que es un comando directo a la PC para reiniciar. Entonces, al utilizar también códigos de barras que solamente poseen números y letras, se facilita en cierta medida el estudio y descifrado del protocolo del teclado.

En el capítulo anterior se explica que al oprimir una tecla común el teclado envía un código único para cada tecla. Esto es más de fácil comprender con un ejemplo. Si se oprime la tecla ‘R’ lo que sucede es que, en principio, se genera un reloj en la línea correspondiente por lo que la línea ‘KBD Clock’ cambia de estado y comienza a oscilar a una frecuencia aproximada de 20 [KHz] durante la transmisión de datos. Cuando esta transmisión termina, la línea regresa al estado inicial. Mientras tanto, por la línea de datos se genera y envía el código correspondiente a la letra ‘R’ que es $2D_H$. Esta transmisión se realiza con 11 bits, uno de inicio, 8 de datos, uno de paridad y uno de final. Los ocho bits de datos se envían iniciando con el bit menos significativo del código seleccionado. En este caso, $0010\ 1101_B$ es el código binario de $2D_H$ y se transmitirá al revés, es decir que se transmite en el siguiente orden: 1011 0100 después del bit de inicio. Posteriormente se envía un código $F0_H$ al soltar la tecla ‘R’ y nuevamente se envía el código de la tecla liberada $2D_H$. Esto significa que al oprimir una sola vez la tecla ‘R’ lo que el teclado transmite por su línea de datos es 2D F0 2D en base

hexadecimal con sus correspondientes bits de inicio, paridad y fin, por lo que al oprimir y liberar una tecla del teclado se transmiten en sí 33 bits.

Para hacer la lectura de estos datos, no se necesita tomar en cuenta la frecuencia de oscilación del reloj generado, sino simplemente se debe considerar su estado. Esto es, mientras el reloj esté en alto, no hay datos enviados. Entonces, cuando esta línea cambia de estado se empieza a transmitir, por lo que comienza también la lectura del código enviado. Así también, no es necesario considerar los bits de inicio, paridad y fin sino solamente los datos y es importante tomar en cuenta que la lectura se debe hacer en el estado bajo del reloj para garantizar que el dato ya está estable en la línea correspondiente. Esta acción y lectura es muy rápida, pero la velocidad de ejecución del MCU permite leer cada estado sin problemas y ejecutar las instrucciones correctamente.

A continuación se muestra el código de otro programa realizado para poder leer el teclado mediante el MCU y desplegar los dados en la pantalla de HyperTerminal.

```
                ORG    $B600

REGBAS          EQU    $1000
PORTA           EQU    $00
PORTC           EQU    $03
PORTB           EQU    $04
DDRC            EQU    $07
DATO            EQU    $00
CONT            EQU    $01
BAUD            EQU    $2B
SCCR1           EQU    $2C
SCCR2           EQU    $2D
SCSR            EQU    $2E
SCDR            EQU    $2F

                LDX    #REGBAS
                CLRA
                STAA   DDRC,X              ;programa al puerto C como entradas.

                LDAA   #$30
                STAA   BAUD,X              ;programa la velocidad de trans. = 9600 baud
                CLRA
                STAA   SCCR1,X             ;longitud de palabra: 1 sb, 8 db, 1 sb
                LDAA   #$0C
                STAA   SCCR2,X             ;Tx = Rx = 1 => SCI activado

INICIO          LDY    #$00                ;registro Y es el contador de pulsos

AQUI            BRSET  PORTC,X,$80,AQUI    ;datos del teclado leödo por el bit 7 del puerto C

VERIF1          LDAA   PORTC,X             ;reloj bit 0 del mismo puerto
                ANDA   #$01                ;este ciclo se mantiene mientras el reloj siga en
                CMPA   #$01                ;estado alto. Una vez que el reloj comienza a
                BEQ    VERIF1              ;oscilar, el flujo continua
```

```
                CPY    #$00                ;si es el primer pulso => bit inicio
                BEQ    VERIF0              ;salta a verificar el estado bajo del reloj

                CPY    #$09                ;si el contador ≥ 9 => bits no relevantes
                BHS    VERIF0              ;salta a verificar el estado bajo del reloj

                LDAA   PORTC,X             ;si no es ni 0 ni 9 entonces es dato
                ANDA   #$80                ;enmascara el bit leído

                CPY    #$01                ;si el contador = 1 no debe haber corrimiento
                BEQ    N_SHIFT             ;salta a guardar bit sin corrimiento

S_SHIFT         LSR    DATO                ;recorre el dato en memoria a la izquierda
                ORAA   DATO                ;se realiza una función lógica OR con el bit recibido
                STAA   DATO                ;se guarda el dato nuevo en memoria
                BRA    VERIF0              ;salta a verificar el estado bajo del reloj
N_SHIFT         STAA   DATO                ;guarda el dato en memoria sin corrimiento
                BRA    VERIF0              ;salta a verificar el estado bajo del reloj

VERIF0          LDAA   PORTC,X             ;reloj bit 0 del puerto C
                ANDA   #$01                ;este ciclo se mantiene mientras el reloj siga en
                CMPA   #$00                ;estado bajo. Una vez que el reloj sube de nivel,
                BEQ    VERIF0              ;el flujo continua

                INY                        ;incrementa contador
                CPY    #$21                ;compara con 33 que son los bits enviados por dato
                BEQ    ETIQ                ;salta a etiqueta
                BRA    VERIF1              ;si no, regresa otra vez

ETIQ            LDX    #$1000              ;carga X con $1000
                LDY    #TABLA              ;carga Y con la dirección de la tabla
                LDAB   #$00                ;acumulador B = 0
MANDA           BRCLR SCSR,X,$80,MANDA     ;mientras el bit 7 del registro scsr esté limpio
                                           ;sigue en este ciclo
BUSCA           LDAA   $00,Y               ;carga A con el valor de la tabla
                CMPA   DATO                ;compara A con el dato en memoria
                BEQ    ENVIA               ;si es igual lo envía por com. serial
                INY                        ;incrementa valor de la tabla
                INCB                       ;incrementa valor de B
                CMPB   #$2B                ;compara B con 43 (# de datos en la tabla)
                BEQ    INICIO              ;salta a inicio de haber leído todos
                BRA    BUSCA               ;si no ha acabado vuelve a buscar

ENVIA           ADDB   #$30                ;aumenta B + 48 para cambiar a código ASCII
                STAB   SCDR,X              ;lo escribe en el registro de envio
                BRA    INICIO              ;salta al inicio

TABLA: FCB $45,$16,$1E,$26,$25,$2E,$36,$3D,$3E,$46,$4C,$4C,$41,$55,$49,$4A,$1E
       FCB $1C,$32,$21,$23,$24,$2B,$34,$33,$43,$3B,$42,$4B,$3A,$31,$44,$4D,$15,$2D,
       $1B,$2C,$3C,$2A,$1D,$22,$35,$1A
```

Resulta interesante analizar la estructura de este programa. La primera parte es la asignación de direcciones y la programación de la interfase de comunicación serial asíncrona del MCU. Posteriormente desde la etiqueta INICIO y hasta la etiqueta ETIQ se realiza la lectura de un sólo carácter enviado por el teclado, llevando a cabo el algoritmo ya mencionado párrafos arriba. Se lee la línea de reloj del teclado o lector de códigos de barras por medio del bit 0 del puerto C. Cuando este bit baja su nivel entonces se continúa con el flujo del programa, haciendo un análisis primero del número de bit que se recibe para poder discriminar el bit de inicio y el resto de los 33 bits enviados a partir del bit 9 recibido.

Posteriormente se analiza el bit 7 del puerto C que es en donde se leen los datos enviados por el teclado o lector de códigos de barras. Se enmascara con #$80 para quedarse solamente con el bit de interés y se recorre hacia la izquierda y realiza una operación lógica OR con el dato guardado en memoria RAM en la dirección $00. El corrimiento no se realiza si se trata del primer bit de importancia recibido y se almacena en memoria. Entonces se salta a un pequeño ciclo que verifica el estado del nivel del reloj, mientras este nivel sea bajo se queda en el ciclo para cuando cambie leer el siguiente carácter. Se incrementa el contador y compara con 33 para saber si ya se han leído todos los bits de un carácter o no. De ya haberlo hecho se tiene almacenado el código correspondiente a la tecla oprimida en la dirección $00 pero este código es aquel asignado al teclado que es completamente aleatorio.

Es por esta razón que se incluye al final del programa una tabla, que resulta ser el método más fácil de comparación entre el código del teclado y el código ASCII, para poder buscar de qué carácter se trata. Como se puede observar en la tabla, el incremento o decremento entre cada valor no está determinado por ninguna función pero corresponde al código de teclado de los dígitos del ‘0’ al ‘9’ en forma creciente y continúa con el código del alfabeto de la ‘A’ a la ‘Z’.

A partir de la etiqueta ETIQ comienza la búsqueda del carácter recibido. El registro índice Y apunta al inicio de la tabla y el acumulador A guarda el dato que se tiene en la dirección $00, mientras que el acumulador B será el encargado de encontrar el carácter en la tabla. Se compara el dato del acumulador A con el correspondiente valor de la tabla y se encierra en un ciclo hasta que estos valores sean iguales. Entonces se enviará el correspondiente valor ASCII, dado por el acumulador B al sumarle 48 para corresponder con dicho código, por la interfase de comunicaciones serial asíncrona del MCU y se regresa al inicio del programa para leer el siguiente carácter. Si se encuentra con un carácter no considerado en la tabla, simplemente se ignora y continúa con el siguiente. Esto está hecho con el fin de únicamente reconocer códigos alfanuméricos. Además, no se considera el hecho de poder escribir minúsculas, sino solamente mayúsculas para la escritura y lectura de los códigos.

Con esto queda resuelto el problema de leer o adquirir datos del teclado o en este caso del lector de códigos de barras. A continuación se analiza una

interfase realizada para poder observar el código leído en un LCD (Liquid Crystal Display).

4.2.5. Interfase con LCD.

Esta interfase se realiza utilizando un LCD de 16 caracteres en dos líneas y se utilizan los puertos B y D para su manejo. El uso de un LCD es muy general y debido a que mantienen un estándar en cuanto al código utilizado para escribir en él, así como los comandos de control para el manejo del mismo y que este mismo estándar permite unificar prácticamente cualquier dispositivo de este tipo con otros dispositivos TTL, se vuelve un recurso fácil de implementar a bajo costo y brinda confiabilidad para mostrar datos en procesos de control y monitoreo.

Un LCD posee generalmente 14 o 16 pines, ocho de los cuales se utilizan para enviar los datos deseados a escribir y el resto son líneas de control para el mismo dispositivo. El LCD utilizado en este proyecto es de 14 pines que se describen a continuación el la tabla 4.2.

Número de pin	Función
1	Tierra
2	V_{cc} 5 [V]
3	Control de intensidad
4	RS
5	R / W
6	Enable
7	DB0
8	DB1
9	DB2
10	DB3
11	DB4
12	DB5
13	DB6
14	DB7

Tabla 4.2. Descripción de los pines de un LCD.

Los datos manejados en el LCD se codifican en ASCII y se escriben en la memoria del LCD en forma secuencial. A través de estas mismas señales pueden escribirse comandos. Por lo anterior, se entiende que hay dos tipos de información: datos y comandos que pueden ser interpretados por el LCD. Los pines del 7 al 14 son los utilizados para recibir los datos, siento el pin 14 el más significativo. Los pines 1 y 2 se utilizan para la polarización del LCD con tierra y 5 [V] respectivamente. El pin 3 es el control de intensidad de la pantalla del LCD, mientras más se acerque este valor a tierra se tendrá la mayor intensidad; se

puede utilizar un potenciómetro para este propósito. Los pines 4, 5 y 6 son en sí los que controlan al LCD. El pin 5 generalmente se coloca a tierra, ya que esto significa escribir sobre el LCD, mientras que el valor de 5 [V] permite leer las memorias del LCD. El pin 4 indica, de ser '0' que el dato enviado se trata de un comando y de ser '1' que se trata de un carácter a escribir en la pantalla. El pin 6 es el conocido 'enable' del sistema que indica el momento de lectura o adquisición de datos o comandos. Mientras este pine esté en '1' no se realiza ninguna lectura, sino hasta que cambie a '0'. Es muy común manejarlo siempre en nivel alto hasta que se envíe un dato o comando. Esto es útil ya que el LCD no tiene una respuesta tan rápida como un MCU. Se necesita de un tiempo de retardo para leer todos los datos o comandos en forma correcta.

Además, es necesario siempre inicializar un LCD antes de mandar dato o comando alguno. Esta inicialización consiste en enviar algunos comandos para seleccionar el funcionamiento del LCD, como el tipo de letra el número de líneas a utilizar, encendido, apagado o limpieza de la pantalla, por ejemplo.

Esta inicialización se ha convertido por la misma estandarización de estos dispositivos en un algoritmo ya establecido y funcional para prácticamente todos estos dispositivos cuyo diagrama de flujo se muestra en la figura 4.6.

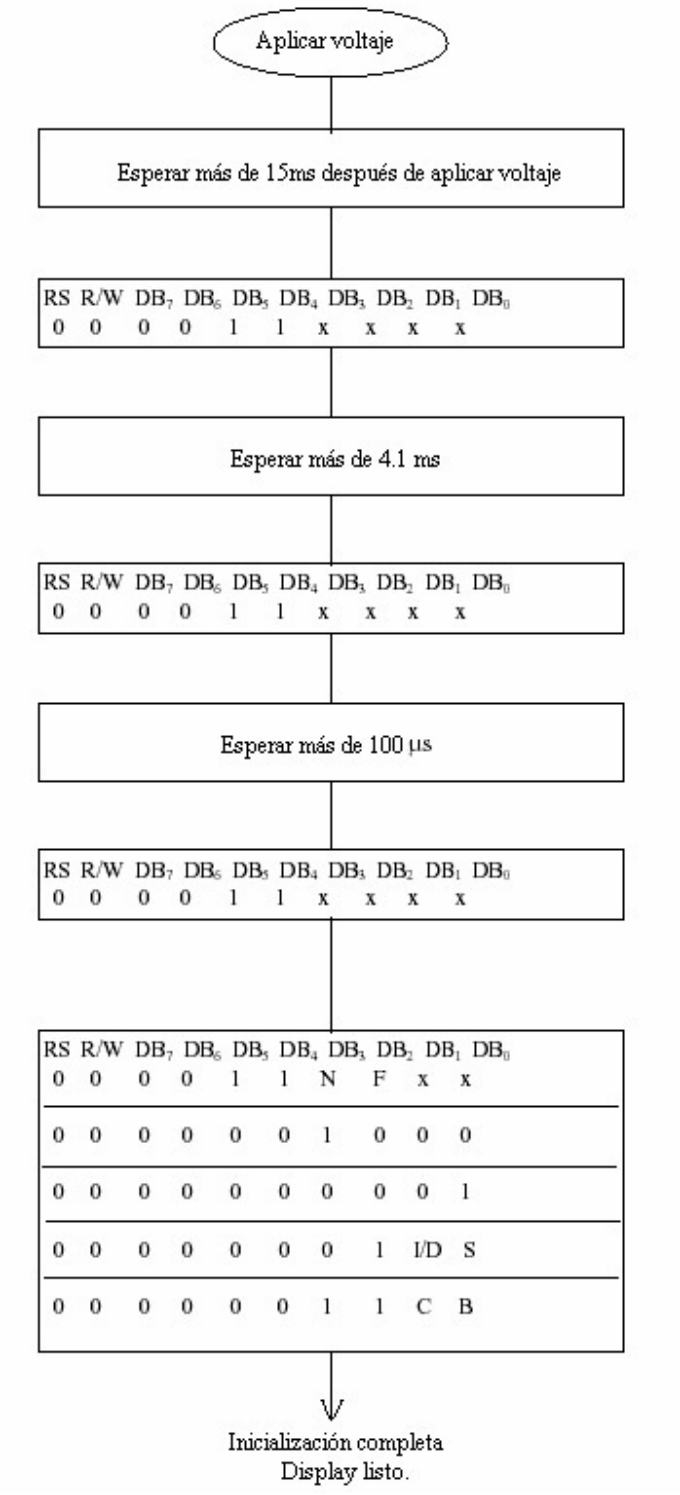


Fig. 4.6. Diagrama de flujo de la inicialización de un LCD para 8 bits.

En dicho diagrama los ‘0’s representan 0 [V] y los ‘1’s 5[V], las x son valores que no importan *(don’t care)*, y durante los espacios de espera o retardos, no se debe enviar ningún dato o comando. El primer valor representa el tipo de comandos que se enviarán al LCD. Después de enviarlo tres veces y esperar los retardos correspondientes, se puede seleccionar el número de bits a utilizar y el tipo de letra, se apaga el LCD, se limpia, se introduce el modo de uso para saber hacia qué lado incrementará el cursor y posteriormente el LCD está listo para funcionar. Mayor referencia sobre el uso de estos dispositivos se puede encontrar en los manuales correspondientes, aunque con esta pequeña explicación es suficiente para poder utilizarlo.

Para entender un poco mejor el uso de este recurso, se lista a continuación otro programa fuente que está basado en el último anterior y muestra cómo se escriben los datos recibidos en el MCU sobre el LCD mediante el uso de pequeñas subrutinas que permiten esta acción.

```
                ORG   $B600

REGBAS          EQU   $1000
PORTA           EQU   $00
PORTC           EQU   $03
PORTB           EQU   $04
DDRC            EQU   $07
DDRD            EQU   $09
DATO            EQU   $00
Y_ALTO          EQU   $50
Y_BAJO          EQU   $51

                LDX   #REGBAS
                LDAA  #$10
                STAA  $1028         ;desactiva SCI y libera puerto D para propósito general
                LDAA  #$FE
                STAA  DDRD,X        ;configura el puerto D como salidas de PD1 a PD5
                CLRA
                STAA  DDRC,X        ;configura el puerto C como entradas

                JSR   RET10MS       ;salta a subrutina retardo de 10 ms
                LDAA  #$30          ;carga A con #$30
                JSR   ENVCOM        ;salta a subrutina Envía Comando
                JSR   RET10MS       ;salta a subrutina retardo de 10 ms
                JSR   ENVCOM        ;salta a subrutina Envía Comando
                JSR   RET100        ;salta a subrutina retardo 100 μs
                LDAA  #$38
                JSR   ENVCOM        ;inicializa dos renglones y caracteres de 5x7
                LDAA  #$0C
                JSR   ENVCOM        ;enciende display
                LDAA  #$01
                JSR   ENVCOM        ;limpia display
                LDAA  #$06
                JSR   ENVCOM        ;escribe mod set para mover cursor a la derecha sin
                                    ;desplazamiento del display.
```

```
INI_REAL    JSR     RET                 ;salto a rutina de retardo
            JSR     RET
            JSR     RET

            LDX   #REGBAS               ;X = #$1000
            CLRA
            STAA  Y_ALTO                ;Y_ALTO es una variable para encontrar los datos
            LDAA  #$01                  ;inicia con valor igual a '1'
            STAA  Y_BAJO

INICIO      LDY    #$00                 ;registro Y es el contador de pulsos

AQUÍ        BRSET  PORTC,X,$80,AQUÍ     ;datos del teclado leödo por el bit 7 del puerto C

VERIF1      LDAA   PORTC,X              ;reloj bit 0 del mismo puerto
            ANDA   #$01                 ;este ciclo se mantiene mientras el reloj siga en
            CMPA   #$01                 ;estado alto. Una vez que el reloj comienza a
            BEQ    VERIF1               ;oscilar, el flujo continua

            CPY    #$00                 ;si es el primer pulso => bit inicio
            BEQ    VERIF0               ;salta a verificar el estado bajo del reloj

            CPY    #$09                 ;si el contador ≥ 9 => bits no relevantes
            BHS    VERIF0               ;salta a verificar el estado bajo del reloj

            LDAA   PORTC,X              ;si no es ni 0 ni 9 entonces es dato
            ANDA   #$80                 ;enmascara el bit leído

            CPY    #$01                 ;si el contador = 1 no debe haber corrimiento
            BEQ    N_SHIFT              ;salta a guardar bit sin corrimiento

S_SHIFT     LSR    DATO                 ;recorre el dato en memoria a la izquierda
            ORAA DATO                   ;se realiza una función lógica OR con el bit recibido
            STAA  DATO                  ;se guarda el dato nuevo en memoria
            BRA   VERIF0                ;salta a verificar el estado bajo del reloj

N_SHIFT     STAA  DATO                  ;guarda el dato en memoria sin corrimiento
            BRA   VERIF0                ;salta a verificar el estado bajo del reloj

VERIF0      LDAA   PORTC,X              ;reloj bit 0 del puerto C
            ANDA   #$01                 ;este ciclo se mantiene mientras el reloj siga en
            CMPA   #$00                 ;estado bajo. Una vez que el reloj sube de nivel,
            BEQ     VERIF0              ;el flujo continua

            INY                         ;incrementa contador
            CPY    #$21                 ;compara con 33 que son los bits enviados por dato
            BEQ    ETIQ                 ;salta a etiqueta
            BRA    VERIF1               ;si no, regresa otra vez

INI_RE      BRA    INI_REAL

ETIQ        LDY    Y_ALTO               ;carga Y con lo que tenga Y_ALTO
            LDAA  DATO                  ;carga A con el dato
            STAA  $00,Y                 ;lo guarda en Y_ALTO
            INY                         ;incrementa Y
            CPY    #$0E                 ;compara con #$0E, número de caracteres
```

```
            BEQ   DISPLAY          ;salta a etiqueta Display
            STY    Y_ALTO          ;guarda el valor de Y
            BRA    INICIO          ;salta al inicio

DISPLAY     LDAA   #$80            ;se limpia lcd y se apunta al inicio de ram
            JSR    ENVCOM          ;salta a subrutina Envía comando
            LDX    #$00            ;carga X con $00
OTRO        LDY    #TABLA          ;carga Y con la dirección de la tabla
            LDAB   #$00            ;acumulador B = 0
            INX                    ;incrementa X
            CPX    #$0E            ;compara X con #$0E
            BEQ    INI_RE          ;salta si es igual al inicio real

BUSCA       LDAA   $00,Y           ;carga A con el valor de la tabla
            CMPA   $00,X           ;compara A con el dato en memoria
            BEQ    ENVIA           ;si es igual lo envía al LCD
            INY                    ;incrementa valor de la tabla
            INCB                   ;incrementa valor de B
            CMPB   #$2B            ;compara B con 43 (# de datos en la tabla)
            BEQ    OTRO            ;salta a OTRO para seguir con la cadena
            BRA    BUSCA           ;si no ha acabado vuelve a buscar

ENVIA       TBA                    ;copia el contenido de B en A
            ADDA   #$30            ;se suma 48 para estar en código ASCII
            JSR    ENVDAT          ;salta a subrutina Envía Dato
            BRA    OTRO            ;regresa a OTRO

*******************************************

RET         PSHX                   ;subrutina de retardo
            LDX    #$FFFF
FGHH        NOP
            DEX
            BNE    FGHH
            PULX
            RTS

RET100      PSHX                   ;subrutina de retardo de 100 μs
            LDX    #$0015
NOPAL       NOP
            DEX
            BNE    NOPAL
            PULX
            RTS

RET10MS     PSHA                   ;subrutina de retardo de 10 ms
            LDAA   #$64
NOPON       BSR    RET100
            DECA
            BNE    NOPON
            PULA
            RTS

DELAY       PSHA                   ;subrutina de delay o retardo
            LDAA   #$02
NOPOL       BSR    RET100
```

```
            DECA
            BNE   NOPOL
            PULA
            RTS

*****************************************
                                    ;función Envía Comando
ENVCOM      PSHA                    ;guarda A en el stack
            STAA  $1004             ;escribe en puerto B, LCD
            BSR   DELAY             ;salto a subrutina delay
            LDAA  #$20              ;E=1, RS=0, modo comando
            STAA  $1008             ;escribe en puerto D
            BSR   DELAY             ;salto a subrutina delay
            LDAA  #$00              ;E=0, RS=0
            STAA  $1008             ;escribe en puerto D
            BSR   DELAY             ;salto a subrutina delay
            LDAA   #$20             ;E=1, RS=0
            STAA   $1008            ;escribe en puerto D
            BSR    DELAY            ;salto a subrutina delay
            PULA                    ;recupera el valor de A
            RTS                     ;regresa de subrutina

*****************************************

ENVDAT      PSHA                    ;guarda A en el stack
            STAA  $1004             ;escribe en puerto B, LCD
            BSR   DELAY             ;salta a subrutina delay
            LDAA  #$30              ;E=1, RS=1, modo datos
            STAA  $1008             ;escribe en puerto D, Enable
            BSR   DELAY             ;salta a subrutina delay
            LDAA  #$10              ;E=0, RS=1
            STAA  $1008             ;escribe en puerto D
            BSR   DELAY             ;salta a subrutina delay
            LDAA  #$30              ;E=1, RS=1
            STAA  $1008             ;escribe en puerto D
            BSR   DELAY             ;salta a subrutina delay
            PULA                    ;recupera el valor de A
            RTS                     ;regresa de subrutina

*****************************************

TABLA: FCB $45,$16,$1E,$26,$25,$2E,$36,$3D,$3E,$46,$4C,$4C,$41,$55,$49,$4A,$1E
   FCB $1C,$32,$21,$23,$24,$2B,$34,$33,$43,$3B,$42,$4B,$3A,$31,$44,$4D,$15,$2D,
   $1B,$2C,$3C,$2A,$1D,$22,$35,$1A
```

Este programa fue desarrollado para poder ver sobre el LCD los datos escritos por medio de un teclado o del lector de códigos de barras. Como se ha mencionado, es muy similar en estructura y algoritmo al programa anterior. En su primera parte lo que se realiza es la inicialización del LCD y en unas pocas líneas se puede configurar el display con tan sólo enviarle los datos escribiéndolos sobre el puerto B y controlando su flujo por el puerto D. Se realizaron subrutinas de retardos específicos para poder controlar el LCD que simplemente son contadores que consumen tiempo de ejecución equivalente al tiempo requerido.

Las funciones de envío de datos y de comandos son las que se deben explicar más a fondo. En ellas se realiza el envío y escritura de los datos al LCD y en sí son muy semejantes, solamente cambian en el valor asignado a RS que se maneja por el bit 4 del puerto D, mientras que el Enable se maneja por el bit 5 del mismo puerto. Al iniciar la rutina se guarda el valor del acumulador A para no perderlo simplemente por si se requiere enviar nuevamente no tener que volver a cargarlo con el valor deseado. Posteriormente se escribe en el puerto B y entonces viene una rutina de retardo necesaria para la respuesta del LCD. Después se escribe en el puerto D el valor correspondiente si se trata de comando o de dato tras otro retardo para enseguida dar el pulso al bit de Enable que permite que el dato sea transferido y leído por el LCD. Un retardo también es necesario para que el LCD pueda responder con tiempo y finalmente se regresa el Enable a nivel alto para evitar que se escriba el mismo dato o comando más de una vez.

Este programa puede leer cualquier dato alfanumérico y escribirlo en el LCD; sin embargo, existe ya la necesidad de adecuar estos programas al objetivo del proyecto, por lo que a continuación se describen las especificaciones requeridas.

4.2.6. Necesidades del proyecto.

Este proyecto requiere detectar códigos de barras de 12 caracteres alfanuméricos codificados según el RFC de una persona y realizados según las normas del código 128 descrito en el primer capítulo de la tesis. Un ejemplo de dichos códigos representan el siguiente arreglo: 00REGA801026. En el desarrollo que se ha seguido hasta la sección anterior se puede leer cualquier tipo de códigos alfanuméricos, sin embargo se deben poder discriminar los códigos de barras que no correspondan al arreglo mostrado. Analizando el arreglo anterior, el protocolo que se sigue es de iniciar con un par de ceros seguidos de cuatro letras y de seis números más. Los dos primeros dígitos son invariables y el resto puede cambiar, es por esto que ellos mismos son un delimitador importante para discriminar al resto de los códigos. Así también, al final del código el lector de códigos de barras manda un símbolo de *"enter o carry return"* el cual nos ayuda también a delimitar los códigos si se considera que consiste en 12 dígitos y un símbolo de *"enter o carry return"* en la posición número trece.

Además, es necesario ya considerar que los códigos de barras se guardarán en memoria externa al MCU y no en la memoria RAM debido a su reducido espacio. La memoria RAM es de 256 bytes y si se ocupan trece bytes para cada código de barras en el mejor de los casos podremos leer únicamente19 códigos de barras, sin embargo también es necesario utilizar memoria RAM para guardar variables y datos auxiliares que permiten la lectura y adquisición de datos. Por lo anterior es necesario utilizar el MCU en modo expandido para poder tener más espacio en memoria. Usar el MCU en modo expandido implica tener 56 kbytes de uso libre. Sin embargo no son bytes que se encuentren continuos en la

memoria expandida por lo que se utilizarán únicamente 42433 bytes que se encuentran entre el espacio reservado para los registros de control y el inicio de la memoria EEPROM. Esto se puede apreciar en el mapa de memoria del MCU en modo extendido. A pesar de esto, se iniciará a partir de la dirección $2000 y hasta la dirección $B5FF para evitar problemas de configuración. Este rango de memoria nos da 38,400 bytes que es una cifra más fácil de recordar y que nos da la posibilidad de guardar 2953 códigos de barras, lo cual es más que suficiente para esta aplicación.

En modo expandido el MCU puede acceder al espacio completo de direcciones de 64 kbytes. Cabe mencionar también, que para poder utilizar el MCU en modo expandido es realizado a través de los puertos B y C de forma multiplexada para que se generen los buses de direcciones de 16 bits y de datos de 8 bits, así como las señales de control AS (address - strobe) y R/W (read – write). Estas señales de control (AS y R/W) permiten que los ocho bits menos significativos del bus de direcciones y los ocho bits de datos sean multiplexados en los mismos pines. Durante el primer medio cada ciclo del bus de direcciones la información está presente. Durante el segundo medio de cada ciclo de bus los pines se convierten en un bus bidireccional de datos. La señal AS se encuentra en un estado alto en un latch para una dirección externa. La información de direcciones es permitida a través de un latch transparente mientras esta señal se encuentra en alto y posteriormente cambia de nivel.

La señales de dirección, R/W y AS se encuentran activas y son válidas para todos los ciclos de los buses, incluyendo acceso a localidades de memoria interna. El reloj E es utilizado para activar dispositivos externos para llevar los datos hace el bus de datos durante la segunda mitad de un ciclo de lectura de un bus que ocurre cuando el reloj se encuentra en alto. R/W controla la transferencia de dirección de datos. Esta señal se encuentra en nivel bajo cuando los datos son escritos al bus interno de datos. R/W permanecerá en bajo durante los ciclos de escritura consecutiva del bus de datos.

La figura 4.7 muestra un diagrama de cómo se conecta una memoria externa par utilizar el MCU en modo expandido.

Debido a lo anterior, también se tendrá que cambiar el puerto de lectura del lector de códigos de barras. Hasta el momento se ha utilizado el puerto C, pero a partir de ahora se utilizará el puerto A para la lectura de los códigos de barras. Así también se implementará memoria extendida en la tarjeta de desarrollo utilizada.

A continuación se muestra entonces el código del programa último corregido. Reconociendo y discriminando los códigos de barras que no cumplan con las características mencionadas, además de utilizar el puerto A como puerto de lectura de los códigos.

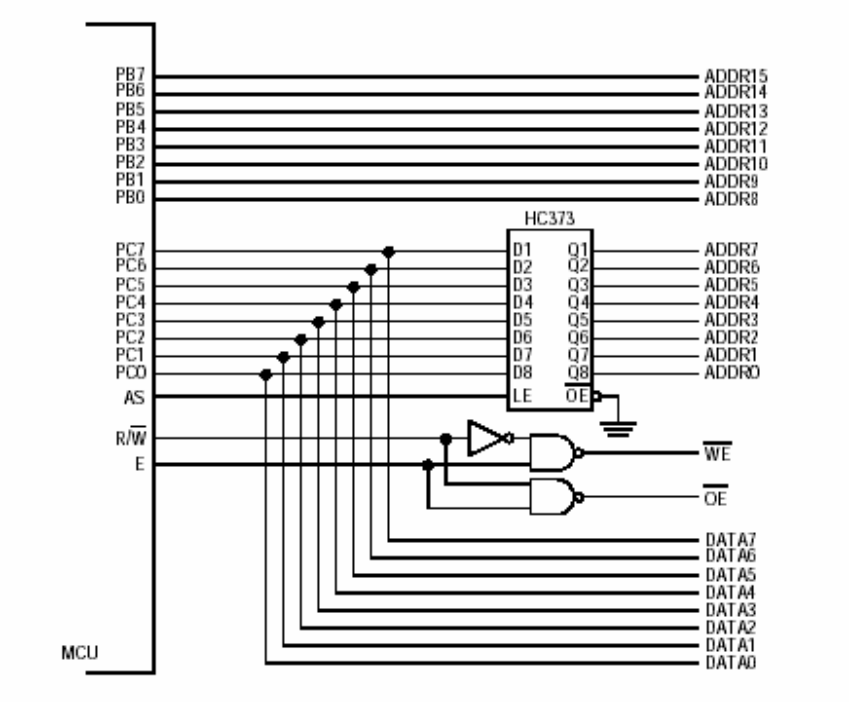


Fig. 4.7. Multiplexeo de los buses de datos y direcciones.

```
                ORG    $B600

REGBAS          EQU    $1000
PORTA           EQU    $00
PORTC           EQU    $03
PORTB           EQU    $04
DDRC            EQU    $07
DDRD            EQU    $09
PACTL           EQU    $26
DATO            EQU    $00
CONT            EQU    $01
Y_ALTO          EQU    $50
Y_BAJO          EQU    $51

                LDX    #REGBAS
                LDAA   #$10
                STAA   $1028          ;desactiva SCI y libera puerto D para propósito general
                LDAA   #$FE
                STAA   DDRD,X         ;configura el puerto D como salidas de PD1 a PD5
                CLRA
                STAA   DDRC,X         ;configura el puerto C como entradas
                CLRA
                STAA   PACTL,X
                STAA   CONT

                JSR    RET10MS        ;salta a subrutina retardo de 10 ms
                LDAA   #$30           ;carga A con #$30
                JSR    ENVCOM         ;salta a subrutina Envía Comando
                JSR    RET10MS        ;salta a subrutina retardo de 10 ms
```

```
                JSR   ENVCOM        ;salta a subrutina Envía Comando
                JSR   RET100        ;salta a subrutina retardo 100 μs
                LDAA  #$38
                JSR   ENVCOM        ;inicializa dos renglones y caracteres de 5x7
                LDAA  #$0C
                JSR   ENVCOM        ;enciende display
                LDAA  #$01
                JSR   ENVCOM        ;limpia display
                LDAA  #$06
                JSR   ENVCOM        ;escribe mod set para mover cursor a la derecha sin
                                    ;desplazamiento del display.

INI_REAL        JSR    RET                  ;salto a rutina de retardo
                JSR    RET
                JSR    RET

                LDX   #REGBAS               ;X = #$1000
                CLRA
                STAA  Y_ALTO                ;Y_ALTO es una variable para encontrar los datos
                LDAA  #$01                  ;inicia con valor igual a '1'
                STAA  Y_BAJO

INICIO          LDY    #$00                 ;registro Y es el contador de pulsos

AQUÍ            BRSET  PORTA,X,$80,AQUÍ     ;datos del teclado leödo por el bit 7 del puerto A

VERIF1          LDAA   PORTA,X              ;reloj bit 0 del mismo puerto
                ANDA   #$01                 ;este ciclo se mantiene mientras el reloj siga en
                CMPA   #$01                 ;estado alto. Una vez que el reloj comienza a
                BEQ    VERIF1               ;oscilar, el flujo continua

                CPY    #$00                 ;si es el primer pulso => bit inicio
                BEQ    VERIF0               ;salta a verificar el estado bajo del reloj

                CPY    #$09                 ;si el contador ≥ 9 => bits no relevantes
                BHS    VERIF0               ;salta a verificar el estado bajo del reloj

                LDAA   PORTA,X              ;si no es ni 0 ni 9 entonces es dato
                ANDA   #$80                 ;enmascara el bit leído

                CPY    #$01                 ;si el contador = 1 no debe haber corrimiento
                BEQ    N_SHIFT              ;salta a guardar bit sin corrimiento

S_SHIFT         LSR    DATO                 ;recorre el dato en memoria a la izquierda
                ORAA DATO                   ;se realiza una función lógica OR con el bit recibido
                STAA   DATO                 ;se guarda el dato nuevo en memoria
                BRA    VERIF0               ;salta a verificar el estado bajo del reloj

N_SHIFT         STAA   DATO                 ;guarda el dato en memoria sin corrimiento
                BRA    VERIF0               ;salta a verificar el estado bajo del reloj

VERIF0          LDAA   PORTA,X              ;reloj bit 0 del puerto A
                ANDA   #$01                 ;este ciclo se mantiene mientras el reloj siga en
                CMPA   #$00                 ;estado bajo. Una vez que el reloj sube de nivel,
                BEQ    VERIF0               ;el flujo continua
```

```
            INY                 ;incrementa contador
            CPY   #$21          ;compara con 33 que son los bits enviados por dato
            BEQ   ETIQ          ;salta a etiqueta
            BRA   VERIF1        ;si no, regresa otra vez

INI_RE      BRA    INI_REAL

ETIQ        LDY    Y_ALTO       ;carga Y con lo que tenga Y_ALTO
            LDAA   DATO         ;carga A con el dato
            STAA   $00,Y        ;lo guarda en Y_ALTO
            CMPA   #$5A         ;compara dato con el código del “enter” ‘5A’
            BEQ    REV_Y        ;salta a rev_y
            INY                 ;incrementa Y
            STY    Y_ALTO       ;guarda el valor de Y
            BRA    INICIO       ;salta al inicio

REV_Y       CPY    #$0E         ;compara Y con #$0E
            BEQ    REVISA       ;si es igual salta a revisa
            BRA    N_VALID      ;si no es igual es código inválido, salta a n_valid

REVISA      LDY    #$0E         ;carga Y con la dirección #$0E
            LDAA   $00,Y        ;carga A con el valor que está en esa dirección
            CMPA   #$5A         ;compara A con el código del “enter” ‘5A’
            BEQ    REVISA2      ;si es igual salta a revisa2
            BRA    N_VALID      ;si no es igual es código inválido, salta a n_valid

REVISA2     LDY    #$03         ;carga Y con la dirección #$03
            LDAA   $00,Y        ;carga A con el valor que está en esa dirección
            CMPA   #$12         ;compara A con el código del “shift” ‘12’
            BEQ    DISPLAY      ;si es igual salta a display, LCD
            BRA    N_VALID      ;si no es igual es código inválido, salta a n_valid

DISPLAY     LDAA   #$80         ;se limpia lcd y se apunta al inicio de ram
            JSR    ENVCOM       ;salta a subrutina Envía comando
            LDX    #$00         ;carga X con $00
OTRO        LDY    #TABLA       ;carga Y con la dirección de la tabla
            LDAB   #$00         ;acumulador B = 0
            INX                 ;incrementa X
            CPX    #$0E         ;compara X con #$0E
            BEQ    S_VALID      ;salta si es igual al inicio real

BUSCA       LDAA   $00,Y        ;carga A con el valor de la tabla
            CMPA   $00,X        ;compara A con el dato en memoria
            BEQ    ENVIA        ;si es igual lo envía al LCD
            INY                 ;incrementa valor de la tabla
            INCB                ;incrementa valor de B
            CMPB   #$2B         ;compara B con 43 (# de datos en la tabla)
            BEQ    OTRO         ;salta a OTRO para seguir con la cadena
            BRA    BUSCA        ;si no ha acabado vuelve a buscar

ENVIA       TBA                 ;copia el contenido de B en A
            ADDA   #$30         ;se suma 48 para estar en código ASCII
            JSR    ENVDAT       ;salta a subrutina Envía Dato
            BRA    OTRO         ;regresa a OTRO

INI_RE1     BRA    INI_RE
```

```
S_VALID     LDAA  #$1F          ;código válido por lo que se
            JSR   ENVDAT        ; envía un espacio
            LDAA  #$4F          ;y 'OK' después del código leído y escrito en LCD
            JSR   ENVDAT
            LDAA  #$4B
            JSR   ENVDAT
            BRA   INI_RE1       ;regresa al inicio

N_VALID     LDAA  #$80          ;se limpia lcd y se apunta al inicio de ram
            JSR   ENVCOM        ;se envía comando
            LDAA  #$4E          ;se envía el letrero "NO VALIDO" al LCD
            JSR   ENVDAT
            LDAA  #$4F          :'0'
            JSR   ENVDAT
            LDAA  #$1F          ;' '
            JSR   ENVDAT
            LDAA  #$56          ;'V'
            JSR   ENVDAT
            LDAA  #$41          ;'A'
            JSR   ENVDAT
            LDAA  #$4C          ;'L'
            JSR   ENVDAT
            LDAA  #$49          ;'I'
            JSR   ENVDAT
            LDAA  #$44          ;D'
            JSR   ENVDAT
            LDAA  #$4F          ;'O'
            JSR   ENVDAT
            BRA   INI_RE1       ;regresa al inicio del programa

*****************************************

RET         PSHX                ;subrutina de retardo
            LDX   #$FFFF
FGHH        NOP
            DEX
            BNE   FGHH
            PULX
            RTS

RET100      PSHX                ;subrutina de retardo de 100 μs
            LDX   #$0015
NOPAL       NOP
            DEX
            BNE   NOPAL
            PULX
            RTS

RET10MS     PSHA                ;subrutina de retardo de 10 ms
            LDAA  #$64
NOPON       BSR   RET100
            DECA
            BNE   NOPON
            PULA
            RTS
```

```
DELAY       PSHA                    ;subrutina de delay o retardo
            LDAA  #$02
NOPOL       BSR   RET100
            DECA
            BNE   NOPOL
            PULA
            RTS

*****************************************
                                    ;función Envía Comando
ENVCOM      PSHA                    ;guarda A en el stack
            STAA  $1004             ;escribe en puerto B, LCD
            BSR   DELAY             ;salto a subrutina delay
            LDAA  #$20              ;E=1, RS=0, modo comando
            STAA  $1008             ;escribe en puerto D
            BSR   DELAY             ;salto a subrutina delay
            LDAA  #$00              ;E=0, RS=0
            STAA  $1008             ;escribe en puerto D
            BSR   DELAY             ;salto a subrutina delay
            LDAA   #$20             ;E=1, RS=0
            STAA   $1008            ;escribe en puerto D
            BSR    DELAY            ;salto a subrutina delay
            PULA                    ;recupera el valor de A
            RTS                     ;regresa de subrutina

*****************************************

ENVDAT      PSHA                    ;guarda A en el stack
            STAA  $1004             ;escribe en puerto B, LCD
            BSR   DELAY             ;salta a subrutina delay
            LDAA  #$30              ;E=1, RS=1, modo datos
            STAA  $1008             ;escribe en puerto D, Enable
            BSR   DELAY             ;salta a subrutina delay
            LDAA  #$10              ;E=0, RS=1
            STAA  $1008             ;escribe en puerto D
            BSR   DELAY             ;salta a subrutina delay
            LDAA  #$30              ;E=1, RS=1
            STAA  $1008             ;escribe en puerto D
            BSR   DELAY             ;salta a subrutina delay
            PULA                    ;recupera el valor de A
            RTS                     ;regresa de subrutina

*****************************************

TABLA: FCB $45,$16,$1E,$26,$25,$2E,$36,$3D,$3E,$46,$4C,$4C,$41,$55,$49,$4A,$1E
   FCB $1C,$32,$21,$23,$24,$2B,$34,$33,$43,$3B,$42,$4B,$3A,$31,$44,$4D,$15,$2D,
   $1B,$2C,$3C,$2A,$1D,$22,$35,$1A
```

No existe gran diferencia con el programa anterior más que lo especificado anteriormente. En las etiquetas REV_Y, REVISA Y REVISA2 se revisa si el código de barras es válido y de serlo, salta a S_VALID en donde se coloca 'OK' después del código; de no ser válido, salta a la etiqueta N_VALID que se encarga de desplegar el letrero 'NO VALIDO' en el display.

La figura 4.8 representa las interconexiones de cada pin de los diferentes puertos utilizados. Cabe mencionar que el LCD está conectado al puerto de salida B del MCU y es este mismo puerto el que se ocupa para poder agregar la memoria física externa para que el MCU funcione en modo extendido. Se han multiplexado de igual forma los datos que se escriben a este puerto con el fin de poder utilizar el puerto para la memoria extendida y el LCD. Sin embargo, debido a que el código del programa no se ha podido reducir de tal manera que se optimice la ejecución del mismo y no rebase el número máximo de bytes dedicados a la EEPROM en donde se alberga el programa fuente, dicha aplicación no es posible utilizarla en forma conjunta a pesar de que se ha logrado su funcionamiento en forma modular.

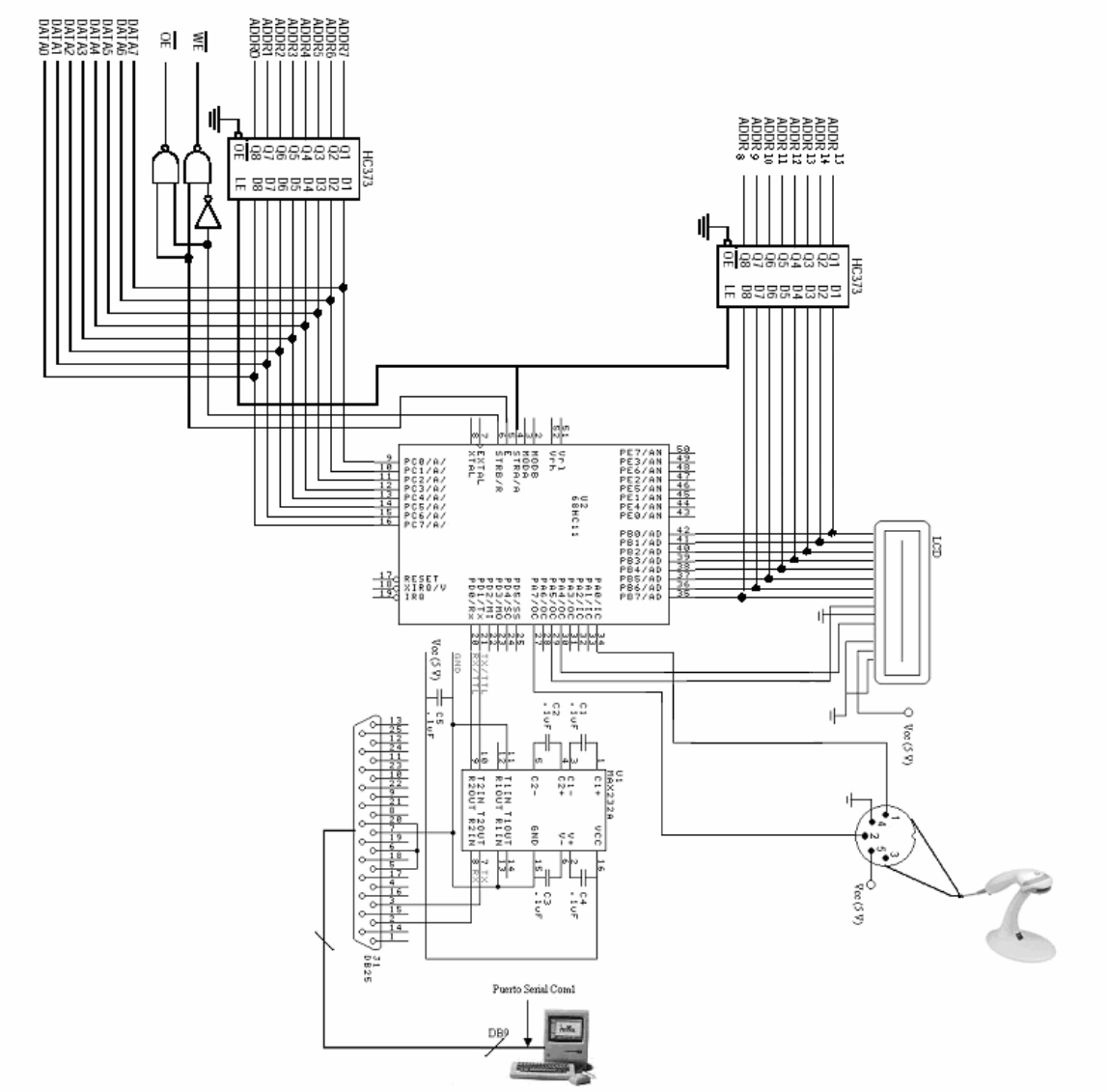


Fig. 4.8. Diagrama de interconexiones total entre la memoria externa y los dispositivos de entrada – salida utilizados en el proyecto.

4.2.7. Comunicación del HAND HELD con la PC.

Es necesario mencionar que la aplicación desarrollada será utilizada en los diferentes programas tales como control de accesos, inventarios, etc. por lo que se desarrolló un programa modular para que interconexión sea de una manera fácil.
Esta aplicación se ha realizado mediante una interfase visual programada en Delphi. Delphi es un lenguaje de programación de cuarta generación orientado a objetos y se basa en una plataforma conocida como *"Object Pascal"* que soporta tanto el diseño de programación estructurada como la orientada a objetos.

Algunas de las mayores ventajas que ofrece este lenguaje de programación es la rápida compilación de los módulos involucrados, el uso de programación modular y por supuesto la estructura visual que contiene su editor y su fácil y rápido acceso a las propiedades y eventos característicos de cada objeto introducido a las formas en donde se realiza generalmente la visualización de la pantalla final del programa ejecutable.

El programa realizado consiste simplemente en una pantalla receptora de datos vía serial, la cual procesa la información recibida y la despliega en pantalla. Dicho objeto está relacionado con las propiedades y programación de otro objeto que liga al puerto serial; en dicho objeto se reciben los datos y se procesan para desplegarlos en la pantalla receptora. Es importante mencionar que esta interfase permite visualizar los datos y se podrían cambiar algunos errores de trasmisión.

Tres botones de ejecución acompañan a esta aplicación. Uno de ellos, *"Limpia Datos"*, tiene únicamente la función de limpiar la pantalla principal en donde se encuentran los datos. El siguiente botón, *"Procesa Información"*, es el más complejo y en el que radica la programación de esta aplicación. La facilidad que tiene el lenguaje de programación Delphi y que es una propiedad de dicho lenguaje, es que permite interconectar los propios objetos con bases de datos existentes. Así, lo que realiza el procedimiento ligado a este objeto es el reconocer los códigos escritos en la pantalla receptora y ligarlos a su base de datos correspondiente con lo que se permite desplegar en pantalla los datos correspondientes al código recibido. El último botón, *"Salir"*, únicamente cierra la aplicación desactivando previamente el puerto serial. Todos estos botones se activan únicamente al seleccionarlos con el *mouse* de la PC.

El resto de los módulos mostrados en esta aplicación son de carácter informativo. La leyenda de Personal en la parte superior de la pantalla y el escudo de la Secretaría de Marina en la esquina superior izquierda le dan presentación a esta aplicación. En la parte media un campo en donde se muestra la fotografía de la persona a la que corresponde al código en turno y abajo su nombre y situación que igualmente corresponde al mismo código.

En la figura 4.9 se muestra la pantalla principal de dicha aplicación y en el anexo C el código fuente de dicho programa.

Fig. 4.8. Pantalla principal de la aplicación final.

CONCLUSIONES.

Durante el desarrollo de este proyecto de tesis se realizaron diferentes aplicaciones de forma modular que fueron vitales para la comprensión del funcionamiento del microcontrolador de Motorola **M68HC11**. Es importante mencionar que durante la creación de dichas aplicaciones se han comprendido y mejorado conceptos y conocimientos sobre el mismo microcontrolador dando un panorama más general de utilización en muchas aplicaciones no tanto en el ámbito de ingeniería sino en aplicaciones de otras ramas de la ciencia en general.

En general, las aplicaciones que se pueden llevar a cabo con este microcontrolador son muy numerosas y existe la posibilidad de acoplarlo a diferentes sistemas ya existentes debido a que tiene una gran facilidad de interconexión; comprendiendo la versatilidad de uso de este microcontrolador utilizado en aplicaciones sencillas y precisas en cualquier establecimiento o dependencia que posea requerimientos de control automatizado.

Es por esto que, para el sistema desarrollado en esta tesis se puede decir que se alcanzaron metas importantes ya que al construir un sistema portátil, fácil de utilizar y a bajo costo utilizando el microcontrolador como parte del corazón de nuestro sistema dejaremos poco a poco la dependencia tecnológica del exterior. Finalmente, es importante mencionar las de conclusiones mas relevantes de este trabajo:

- Los códigos de barras son actualmente una forma muy eficiente de codificación y registro de datos que permite hacer más eficiente la captura de dicha información disminuyendo en gran medida la posibilidad de cometer errores al momento mismo de capturarla.
- El microcontrolador de Motorota M68HC11 es un dispositivo que, por su estructura versátil y recursos internos, acepta el desarrollo de gran número de aplicaciones.

- El desarrollo de este proyecto implica un conocimiento preciso de muchos de los recursos que tiene este microcontrolador.
- La primera plataforma desarrollada se basa en la versión E9 del microcontrolador, sin embargo es posible mudar de versión y reacondicionarla en la versión F1 del mismo microcontrolador, con el objetivo de que se cuente con más recursos para poder ampliar la versatilidad del sistema.
- Para aplicaciones grandes, es recomendable utilizar el microcontrolador en modo extendido para disponer del mapa de memoria completo de 64 Kbytes.
- El sistema desarrollado se puede extender no sólo a control de personal, sino también a otras aplicaciones como realización de inventarios inmobiliarios o bibliotecarios, por mencionar un ejemplo.
- El sistema mencionado para control de personal cuenta con varios filtros de seguridad para evitar la captura de códigos inválidos.
- Es importante el desarrollo de un modelo y diseño exclusivos que involucre una reducción en tamaño y forma del sistema físico para la aplicación que se realizó en este proyecto.
- La tarjeta diseñada exclusivamente para la aplicación desarrollada puede ser bastante más pequeña por lo que se compactaría considerablemente el tamaño del dispositivo de adquisición de datos.
- Una aplicación inmediata posterior puede ser el de diseño o acoplamiento de un teclado - serial para uso de encriptores. Esto es debido a que el sistema realizado se basó en un decodificador de señales de teclado, por lo que el uso de esto podría utilizarse para aplicaciones que necesiten codificación de mensajes o información.

BIBLIOGRAFÍA.

- Doblado Alcázar Cristina, González Gómez Juan, Prieto-Moreno Andrés y San Martín Juan José, ***Microcontrolador MC68HC11, Fundamentos, Recursos y Programación,*** U.A.M., España 1997.

- Spasov Peter, ***Microcontroller Technology The 68HC11*****,** Prentice Hall 1996. Clasif: TJ223.M53 S64 1996.

- Tocci Ronald J., Ambrosio Frank J., Laskowski Lester P., ***Microprocessors and Microcomputers Hardware and Software*****,** Prentice Hall 1997.

- Wray William C., Greenfield Joseph D., ***Using microprocessors and microcomputers : the Motorola family*****,** Prentice Hall, 1994.
  Clasif : QA76.8M67 G73 1994.

- Hall Douglas V., ***Microprocessors and interfacing : Programming and hardware*****,** McGraw-Hill 1986.
  Clasif : QA76.6 H315.

- Stallings William, ***Organización y Arquitectura de computadores, diseño para optimizar prestaciones;*** 4ª edición, Prentice Hall, 1997.

- Armada de México, ***El Universo Digital de IBM PC, AT y PS/2;*** Valladolid, España, 1997. http://fly.to/udigital

- Olmedo A. Oscar, Buenabad C. Jorge, Vega G. Andrés, ***Programación de microprocesadores 8086/88 e interfases;*** Centro de Investigación y de Estudios Avanzados del IPN, Departamento de Ingeniería Eléctrica – Sección de Computación.

- Sargent III Murray, Shoemaker Richard L., ***The IBM PC from the incide out,*** Addison-Wesley Publishing Company 1986.

- Aldape Alonso Marcela, García Álvarez José Ramón, Sánchez Sabbagh Alejandro, Molinero Torres Miguel Ángel, Rivas Ortiz Faustino Alberto; *Tesis:* ***Microcontroladores: desarrollo de un sistema de adquisición de datos con base en un microcontrolador 8751.*** Tesis Licenciatura (Ingeniero Mecánico-Electricista), Universidad La Salle, Escuela de Ingeniería. México 1994.

- Motorola, ***M68HC11 Reference Manual,*** Rev. 3, 1996.

- Motorola, ***M68HC11 Pcbug11 User's Manual,*** Rev. 3, 1996.

- Motorola, ***M68HC11 Microcontrollers, M68HC11E Family Technical Data,*** Rev. 4, 2002.

- Motorola, ***M68HC11EVBU Universal Evaluation Board, User's Manual,*** Rev. 3, 1997.

- Motorola, ***Programming MC68HC711E9 Devices with PCbug11and the M68HC11EVB,*** Motorola Semiconductor Engineering Bulletin EB187, 1998.

- Motorola, ***M68HC11 EEPROM Programming from a Personal Computer – Application Note AN1010/D,*** Rev. 1.5, 2002.

- Motorola, ***M68HC11 Bootstrap Mode – Application Note AN1060/D,*** Rev. 1.0, 1999.

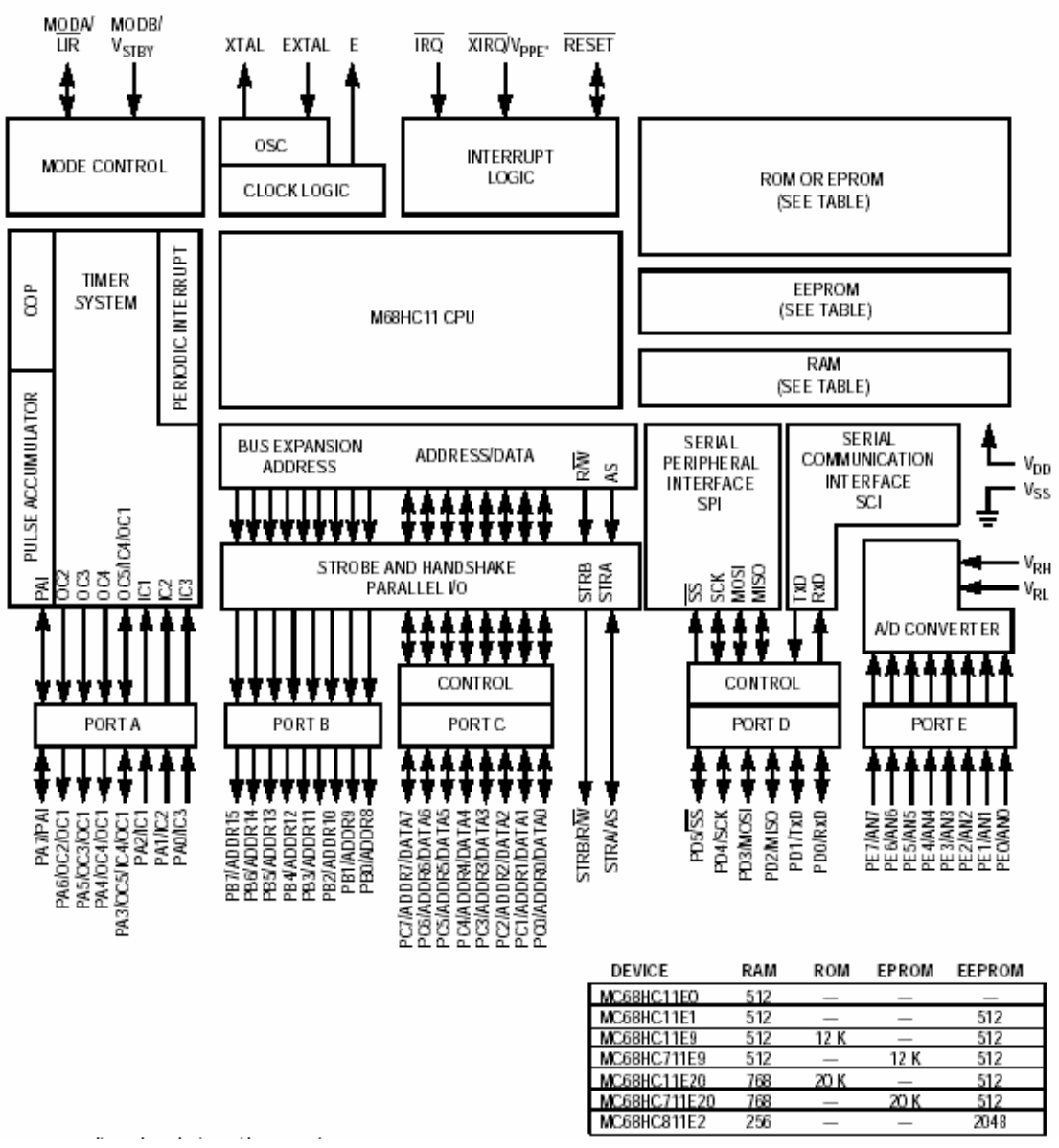
REFERENCIAS ELECTRÓNICAS:

- http://www.codigodebarras.com/que_es.htm
- http://www.radiouniversidad.org/secciones/reportajes/tecnologia/El%20lenguaje%20de%20las%2 0l%EDneas.html
- http://www.ent.ohiou.edu/~amable/autoid/history.htm
- http://www.taltech.com/TALtech_web/resources/intro_to_bc/bcsymbol.htm
- http://www.taltech.com/TALtech_web/resources/intro_to_bc/bcbascs.htm
- http://www.taltech.com/TALtech_web/resources/intro_to_bc/bcpwork.htm
- http://www.linuxdevices.com/articles/AT2281789509.html
- Metrologic's MS-9500 Voyager series datasheet; http://www.metrologic.com
- http://www.beyondlogic.org/keyboard/keybrd.htm
- http://www.seattlerobotics.org/encoder/aug97/cable.html
- http://www.arcelect.com/rs232.htm

Apéndice A.

Descripción general del microcontrolador M68HC11.

A.1. Estructura general del M68HC11.



DEVICE	RAM	ROM	EPROM	EEPROM
MC68HC11E0	512	—	—	—
MC68HC11E1	512	—	—	512
MC68HC11E9	512	12 K	—	512
MC68HC711E9	512	—	12 K	512
MC68HC11E20	768	20 K	—	512
MC68HC711E20	768	—	20 K	512
MC68HC811E2	256	—	—	2048

Figura A.1. Diagrama de bloques de la serie E del M68HC11. [35]

[35] Motorola, *M68HC11 Reference Manual*, Rev. 3, 1996.

A.2. Asignación de pines del M68HC11.

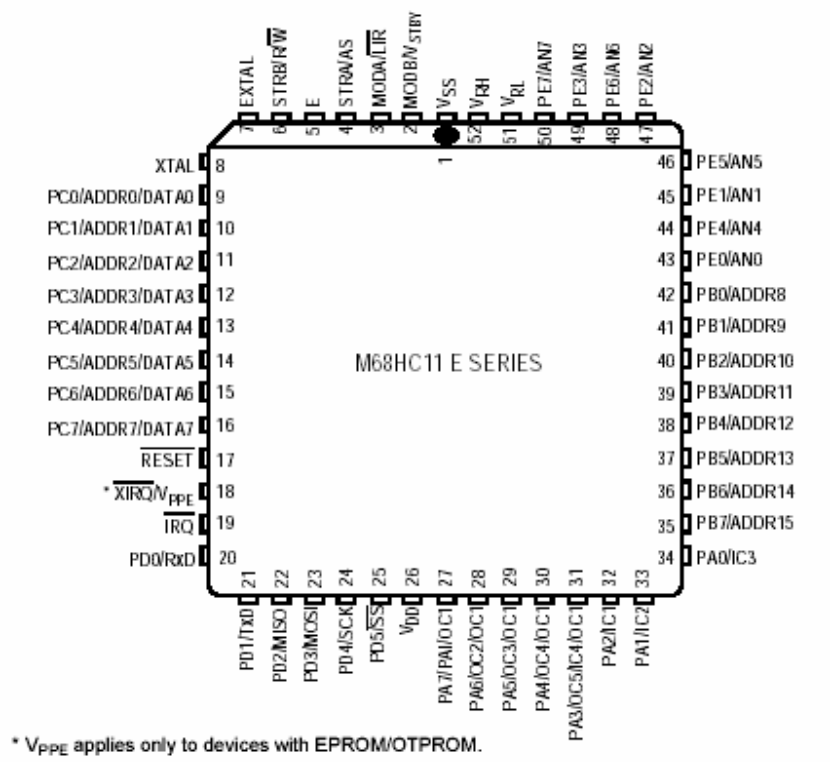


Figura A.2. Asignación de pines para el M68HC11 de 52 pines. [36]

A.3. Puertos de Entrada – Salida. [37]

A.3.1. Puerto A.

El puerto A dispone de tres pines de entrada, 4 pines de salida y uno configurable como entrada o salida. Se encuentra mapeado en memoria en la dirección $1000. Los pines del puerto A están compartidos por otros recursos: comparadores, acumulador de pulsos y capturadores.

Por defecto los recursos internos asociados a los pines del puerto A están desconectados. El puerto A se comporta como un puerto normal en el que si se escribe un valor en la dirección $1000 se reflejará en los correspondientes pines de salida y si se lee un valor, se hará de los pines de entrada.

[36] Ibid.
[37] Doblado Alcázar Cristina, González Gómez Juan, Prieto-Moreno Andrés y San Martín Juan José, *Microcontrolador MC68HC11, Fundamentos, Recursos y Programación,* U.A.M., España 1997.

El pin 7 se puede configurar tanto para entrada como para salida cambiando el bit 7 del registro PACTL ($1026). Un cero en este bit indica entrada y un uno salida. Por defecto está configurado como entrada.

A.3.2. Puerto B.

Sus ocho bits son de salida. En el modo no expandido del MCU se comporta como un puerto de salida (PBx). En el modo expandido se utiliza para mandar el byte alto del bus de direcciones (Ax). Su dirección es la $1004.

A.3.3. Puerto C.

Es un puerto de entrada – salida. En el modo no expandido sus ocho bits pueden actuar como entradas o salidas independientes, según cómo se configuren los bits en el registro DDRC. Un cero configura el pin correspondiente para entrada, un uno lo hace para salida. La dirección del puerto C es $1003.

Si el microcontrolador está funcionando en modo expandido, el puerto C actúa como parte baja del bus de direcciones multiplexada con el bus de datos.

A.3.4. Puerto D.

Igual que el puerto C, es un puerto de entrada / salida en el que se pueden configurar sus bits independientemente para entrada o salida con la diferencia de que este puerto es de solamente 6 bits. El registro de configuración DDRD se encuentra en la dirección $1009. Unos corresponden con salidas y ceros con entradas. El puerto está mapeado en la dirección $1008.

Los pines PD5 – PD0 se pueden utilizar para señales de propósito general de entrada – salida. Estos pines sirven para las señales usadas en la interfaz de comunicación serial (SCI) y en la interfaz periférica serial (SPI) cuando se utilizan estos sistemas.

- PD0 es la señal de entrada de datos recibidos (RxD) para la SCI.
- PD1 es la señal de salida de datos transmitidos (TxD) para la SCI.
- PD5 – PD2 están dedicados a la SPI:
  - PD2 es la señal de entrada maestro – salida esclavo, *master in – slave out*, (MISO).
  - PD3 es la señal de salida maestro - entrada esclavo, *master out – slave in*, (MOSI).
  - PD4 es la señal de reloj serial, *serial clock*, (SCK).
  - PD5 es la entrada de selección de esclavo, *slave select*, (SS).

A.3.5. Puerto E.

Es un puerto de 8 bits de entrada. Está situado en la dirección $100A. Comparte pines con los 8 canales del convertidor analógico – digital. Para poder utilizar el puerto E como un puerto normal es preciso que el convertidor A/D esté desconectado. Por defecto lo está.

A.4. Registros de la CPU. [38]

La CPU del M68HC11 posee siete registros que se muestran en la siguiente figura.

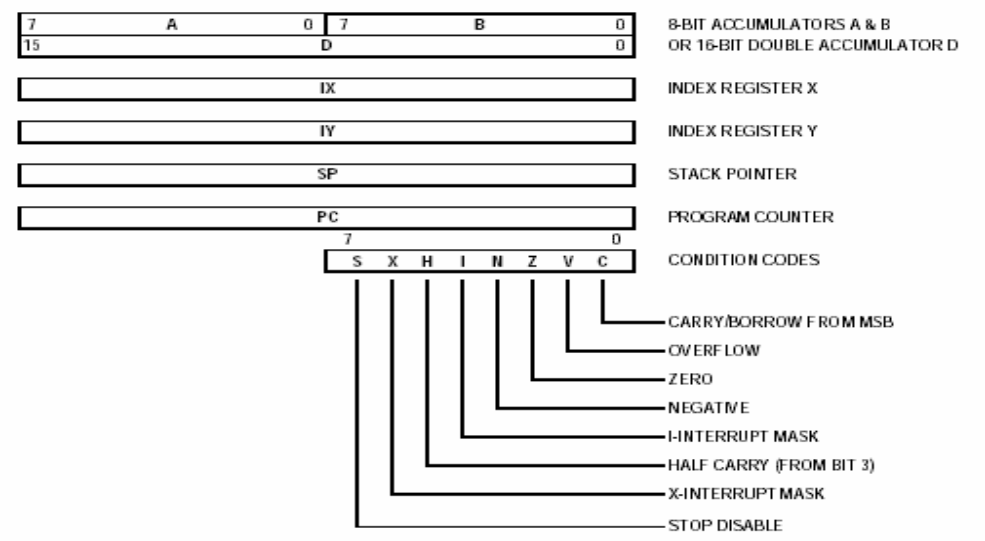


Figura A.3. Modelo de los registros del M68HC11.

A.4.1. Registros acumuladores A, B y D.

Los acumuladores A y B son registros de ocho bits de propósito general que guardan el operando, el resultado de los cálculos aritméticos o manipulación de datos. Para algunas instrucciones, estos dos acumuladores son tratados como un acumulador doble, es decir, de 16 bits que se conoce como acumulador D. La parte alta de este nuevo registro es el acumulador A y la parte baja el acumulador B.

[38] Op. Cit. Motorola, *M68HC11 Reference Manual*.

A.4.2. Registro índice X.

El registro índice X provee un valor indexado de 16 bits que puede ser añadido a un offset de 8 bits provisto en una instrucción para crear la dirección efectiva. Esto significa que el registro índice X se utiliza para guardar direcciones de memoria, no datos, de tal forma que esa dirección pueda ser aumentada gracias al offset de 8 bits. El registro índice X puede incluso ser utilizado como un contador o un registro de almacenamiento temporal.

A.4.3. Registro índice Y.

Este registro índice realiza la misma función que el registro índice X. Alguna diferencia consiste solamente en que unas instrucciones que utilizan este registro necesitan un byte extra de código de máquina y un ciclo de tiempo de ejecución extra.

A.4.4. Stack Pointer (SP).

La CPU del M68HC11 tiene una pila programada automática. Esta pila puede ser localizada en cualquier dirección y puede ser de cualquier tamaño obviamente hasta el máximo de la memoria disponible. Normalmente el SP es inicializado por una de las primeras instrucciones en un programa de aplicación. La pila es configurada como una estructura de datos que decrece desde la memoria alta a la memoria baja. Cada ocasión que un nuevo byte es introducido a la pila, el SP es decrementado y cada vez que un byte es sacado de la pila, el SP es incrementado.
Este registro permite la interacción entre subrutinas apuntando a la pila que almacena la dirección de la próxima operación de la subrutina.

A.4.5. Program Counter.

Este registro de 16 bits se va incrementando según se van ejecutando las instrucciones. Por tanto, los programas se ejecutan desde direcciones bajas a altas y la pila crece de direcciones altas a bajas. Es importante dar un valor seguro a este registro de tal manera que la pila no se solape con el código, si es que el código se encuentra en la memoria RAM.

A.4.6. Condition Code Register (CCR).

Es un registro de estados de 8 bits que indica el resultado de la última operación efectuada por la ALU, son las llamadas banderas del microcontrolador.

Estos 8 bits están repartidos como 5 indicadores de estado (C, V, Z, N y H), dos bits de interrupción mascarable (IRQ y XIRQ), y un bit para deshabilitar el alto o stop (S).

Dichos bits del registro CCR se describen a continuación:

A.4.6.1. Carry / Borrow (C).

Este bit se enciende si la unidad lógica aritmética (ALU) produce un bit de acarreo o lo toma prestado durante una operación aritmética. También actúa como una bandera de error en caso de operaciones de multiplicación y división.

A.4.6.2. Overflow (V).

Esta bandera se enciende si una operación produce un overflow aritmético. En caso contrario, este bit será limpiado.

A.4.6.3. Zero (Z).

El bit Z se enciende si el resultado de una operación aritmética, lógica o de manipulación de datos es cero. En caso contrario, este bit se encontrará en cero. Algunas instrucciones solamente afectan a este bit del registro de estados y a ningún otro, por lo que para estas operaciones pueden ser determinadas estas condiciones.

A.4.6.4. Negative (N).

Este bit es encendido si el resultado de una operación aritmética, lógica o de manipulación de datos es negativa (MSB=1). En caso contrario el bit N es limpiado. Se dice que un resultado es negativo si su bit más significativo (MSB) es uno.

A.4.6.5. Interrupt Mask (I).

La máscara de petición de interrupción (IRQ) es una máscara global que desactiva todos los recursos mascarables de interrupción. Mientras este bit esté activado, las interrupciones se pueden volver pendientes, pero la operación de la CPU continúa sin interrumpirse hasta que este bit sea limpiado. Después de cualquier reset, el bit I es encendido por defecto y puede solamente ser limpiado por una instrucción de software. Normalmente este bit es cero después de una instrucción de ejecución de retorno de interrupción.

A.4.6.6. Half Carry (H).

Este bit es encendido cuando un acarreo ocurre entre los bits 3 y 4 de la unidad aritmética lógica durante alguna de las instrucciones ADD, ABA o ADC. En otro caso contrario este bit es limpiado. Este bit es utilizado durante operaciones con código BCD.

A.4.6.7. Interrupt Mask (X).

La máscara XIRQ (X) deshabilita interrupciones del pin XIRQ. Después de cualquier reset, X es encendido por defecto y debe ser limpiado por una instrucción de software. Cuando una interrupción XIRQ es reconocida, los bits X e I son encendidos después de que los registros son puestos en la pila pero primero los vectores de interrupción son reconocidos. Posteriormente los bits son limpiados de nueva cuenta cuando se realiza el retorno de interrupción.

A.4.6.8. Stop Disable (S).

Poner este bit en alto, previene la instrucción de alto en poner el M68HC11 en una condición de alto a baja potencia. Si la instrucción STOP es encontrada por la CPU mientras el bit S está encendido, dicha instrucción se interpretará como una instrucción de no operación (NOP), y el proceso continúa con la siguiente instrucción. S es encendido por reset, por lo que STOP es deshabilitado por defecto.

A.5. Tipos de datos.

La CPU del M68HC11 soporta cuatro tipos de datos: datos de un bit, enteros signados y sin signo de 8 y 16 bits, fracciones no signadas de 16 bits y direcciones de 16 bits.
Un byte es reconocido como un conjunto de 8 bits, una palabra se compone de dos bytes consecutivos con el byte más significativo en la dirección de valor bajo. Debido a que el M68HC11 es una CPU de 8 bits no existen requerimientos especiales para alinear instrucciones u operandos. [39]

A.6. Modos de direccionamiento. [40]

Los modos de direccionamiento son distintas formas que tiene la CPU de acceder a los datos que están en memoria. Existen seis modos de direccionamiento distintos que se describen a continuación:

A.6.1. Direccionamiento inmediato.

Este direccionamiento se usa cuando el dato al que se hace referencia se encuentra dentro de la instrucción, no es necesario acceder a memoria. El dato puede ser de 1 o 2 bytes, pero se requieren 2 para su código de operación, uno para el operador y otro para el operando. Este modo de direccionamiento se indica mediante el signo #.
Ejemplo: LDAA #08.

[39] Ibid.
[40] Motorola, *M68HC11 Microcontrollers, M68HC11E Family Technical Data,* Rev. 4, 2002.

A.6.2. Direccionamiento extendido.

En este tipo de direccionamiento, el dato se encuentra en la dirección de memoria especificada. El dato puede estar en cualquier posición de la memoria por lo que la dirección ocupa 2 bytes.
Ejemplo: LDAA $FC00.

A.6.3. Direccionamiento directo.

Este modo de direccionamiento es exactamente igual al anterior con la salvedad de que sólo actúa con direcciones comprendidas entre $00 y $FF, que son los primeros 256 bytes de la memoria). La utilidad de este modo es que sólo necesita 1 byte para especificar la dirección del dato, con lo que se ahorra espacio y tiempo de operación y ejecución.
La conclusión importante al comparar estos los dos últimos modos de direccionamiento es que es conveniente utilizar el direccionamiento directo, es decir, situar las variables en las direcciones bajas de memoria, en el espacio $00 a $FF. De esta manera, todas las instrucciones que hagan referencia a variables utilizarán direccionamiento directo y se ahorrarán muchos bytes de memoria.
Ejemplo: LDAB $05.

A.6.4. Direccionamiento indexado.

Este direccionamiento hace uso de los registros índices X e Y. En este modo, es posible sumar un offset de 8 bits al valor contenido en los registros índices de tal forma que la suma es la dirección a la que se refiere. Este tipo de direccionamiento permite referirse a cualquier localidad de memoria en los 64 kbytes de espacio libre. Este tipo de instrucciones puede variar de un tamaño de 2 a 5 bytes dependiendo de lo que se necesite realizar.
Ejemplo: STAB $2C,X.

A.6.5. Direccionamiento relativo.

Este modo de direccionamiento sólo se utiliza con las instrucciones de bifurcación. Éstas indican a la CPU que realice un salto de tantos bytes hacia delante o hacia atrás. El desplazamiento tiene signo y es de un byte por lo que las bifurcaciones sólo se pueden hacer de 128 bytes hacia atrás ó 127 bytes hacia delante. El salto a la ramificación (branch) puede ser incondicionado o condicionado.
Ejemplo: BRA SIGUE.

A.6.6. Direccionamiento inherente.

En este modo de direccionamiento, toda la información necesaria para la ejecución de la instrucción está contenida en ella misma y por lo tanto dentro de los registros de la CPU, por lo que no se requieren operandos.
Ejemplo: CLRA.

A.7. Modos de operación. [41]

El M68HC11 puede funcionar en cuatro modos de operación diferentes, cada uno de los cuales es seleccionado con las entradas MODA y MODB durante el reset del sistema, es decir, cuando se inicializa el funcionamiento del microcontrolador. Estos valores se muestran en la tabla A.1.
Los diferentes modos de operación del M68HC11 son: single chip, expanded, bootstrap y special test. Cada modo de funcionamiento dispone de un mapa de memoria diferente como lo muestra la figura A.4.

- **Single chip:** En este modo de operación, el microcontrolador funciona como tal, es decir, con toda la disponibilidad de sus puertos y con un alcance para el usuario definido por su memoria interna. Por lo anterior, el mapa de memoria del microcontrolador está constituido por la memoria RAM, la memoria EEPROM, los registros de control y la memoria ROM, de tal manera que al arrancar se comience a ejecutar el programa indicado por los vectores de interrupción que se encuentra en la memoria ROM.

- **Expanded:** Además del mapa de memoria del modo single chip, es posible acceder al resto de las posiciones de memoria conectando memorias externas. El precio a pagar es que se pierden dos puertos de entrada – salida, los puertos B y C, que se utilizarán como bus de datos y direcciones. En este modo se puede utilizar la memoria ROM interna, pero también es posible deshabilitar esta ROM, y acceder a memoria externa y con ello a los vectores de interrupción que se encuentren en esa memoria externa. Al perder los puertos B y C, quedan 16 pines disponibles para direccionar una memoria exterior hasta de 64 kbytes.

- **Bootstrap:** Este modo difiere del single chip en que los vectores de interrupción no se encuentran en la memoria ROM común sino que se encuentran en otra memoria ROM, llamada ROM de arranque. Al arrancar en este modo, automáticamente comienza a ejecutarse el programa *"boot-loader"* que se encuentra en la memoria ROM *"bootstrap"*.
  En este modo, el microcontrolador, a través del programa *"boot-loader"*, habilita la interfase de comunicación serial asíncrona y permite que el usuario descargue programas de propósito específico en la memoria RAM interna del microcontrolador.

[41] Ibid.

- **Special test:** Este modo de operación es igual al modo bootstrap con la salvedad de que se puede acceder a memoria externa. Este modo se utiliza para realizar pruebas de fábrica, En este modo especial se tiene acceso a determinados registros de control que en los otros modos están protegidos.

MODB	MODA	MODO DE OPERACIÓN
1	0	Single chip
1	1	Expanded
0	0	Bootstrap
0	1	Special Test

Tabla A.1. Selección de modos de operación.

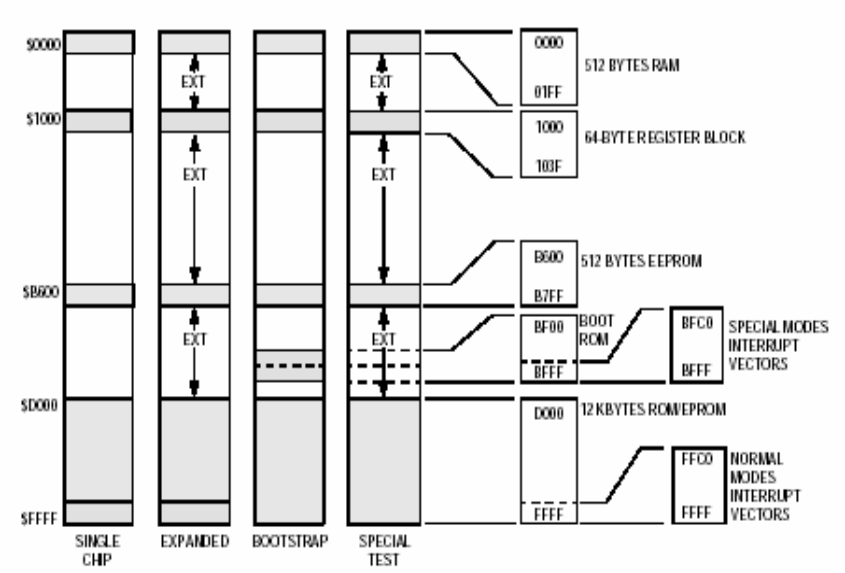


Figura A.4. Mapa de memoria del M68HC11(7)E9.

Apéndice B.

Código Fuente en el microcontrolador.

```
                ORG   $B600

REGBAS          EQU   $1000
PORTA           EQU   $00
PORTC           EQU   $03
PORTB           EQU   $04
DDRC            EQU   $07
DDRD            EQU   $09
PACTL           EQU   $26
BAUD            EQU   $2B
SCCR1           EQU   $2C
SCCR2           EQU   $2D
SCSR            EQU   $2E
SCDR            EQU   $2F

DATO            EQU   $00
CONT            EQU   $01
Y_ALTO          EQU   $20
Y_BAJO          EQU   $21
X_ALTO          EQU   $22
X_BAJO          EQU   $23
Y_EXTH          EQU   $24
Y_EXTL          EQU   $25
Y_LCDH          EQU   $26
Y_LCDL          EQU   $27


                LDX   #REGBAS
                CLRA
                STAA  CONT              ;limpia el contador
                STAA  LCDL              ;limpia apuntador bajo de LCD
                STAA  LCDH              ;limpia apuntador alto de LCD
                STAA  SCCR1,X           ;longitud de palabra: 1 sb, 8 db, 1 sb
                LDAA  #$30
                STAA  BAUD,X            ;programa la velocidad de trans. = 9600 baud
                LDAA  #$0C
                STAA  SCCR2,X           ;Tx = Rx = 1 => SCI activado

                JSR   RET10MS           ;salta a subrutina retardo de 10 ms
                LDAA  #$30              ;carga A con #$30
                JSR   ENVCOM            ;salta a subrutina Envía Comando
                JSR   RET10MS           ;salta a subrutina retardo de 10 ms
                JSR   ENVCOM            ;salta a subrutina Envía Comando
                JSR   RET100            ;salta a subrutina retardo 100 μs
                LDAA  #$38
                JSR   ENVCOM            ;inicializa dos renglones y caracteres de 5x7
                LDAA  #$0C
                JSR   ENVCOM            ;enciende display
                LDAA  #$01
                JSR   ENVCOM            ;limpia display
                LDAA  #$06
```

```
                JSR   ENVCOM              ;escribe mod set para mover cursor a la derecha
                                          ;sin desplazamiento del display.
                CLRA
                STAA  Y_EXTH              ;Y_EXTH es una variable para encontrar los datos
                LDAA  #$30
                STAA  Y_EXTL              ;Y_EXTL = #$0030

INI_REAL        LDAA  $1000               ;acumulador A = #$1000
                ANDA  #$02                ;enmascara el segundo bit
                CMPA  #$02                ;compara A con #$02
                BEQ   CHIN                ;salta si es igual a etiqueta chin
                BRA   LEE                 ;salta a etiqueta lee
CHIN            JSR   SERIE               ;salta a subrutina serie
                CLRA                      ;limpia acumulador A
                STAA  $0001               ;guarda A en $0001 (dato)

LEE             LDX   #REGBAS             ;X = #$1000
                CLRA                      ;limpia acumulador A
                STAA  X_ALTO              ;limpia X_ALTO
                STAA  Y_ALTO              ;limpia Y_ALTO
                STAA  Y_EXTH              ;limpia Y_EXTH
                LDAA  #$01                ;acumulador A = #$01
                STAA  X_BAJO              ;X_BAJO = $0001
                LDAA  #$02                ;acumulador A = #$02
                STAA  Y_BAJO              ;Y_BAJO = $0002
                LDAA  #$30                ;acumulador A = #$30
                STAA  X_EXTL              ;X_EXTL = $0030

INICIO          LDY   #$00                ;registro Y es el contador de pulsos

AQUÍ            BRSET PORTA,X,$80,AQUÍ    ;datos del teclado leödo por el bit 7 del puerto A

VERIF1          LDAA  PORTA,X             ;reloj bit 0 del mismo puerto
                ANDA  #$01                ;este ciclo se mantiene mientras el reloj siga en
                CMPA  #$01                ;estado alto. Una vez que el reloj comienza a
                BEQ   VERIF1              ;oscilar, el flujo continua

                CPY   #$00                ;si es el primer pulso => bit inicio
                BEQ   VERIF0              ;salta a verificar el estado bajo del reloj

                CPY   #$09                ;si el contador ≥ 9 => bits no relevantes
                BHS   VERIF0              ;salta a verificar el estado bajo del reloj

                LDAA  PORTA,X             ;si no es ni 0 ni 9 entonces es dato
                ANDA  #$80                ;enmascara el bit leído

                CPY   #$01                ;si el contador = 1 no debe haber corrimiento
                BEQ   N_SHIFT             ;salta a guardar bit sin corrimiento

S_SHIFT         LSR   DATO                ;recorre el dato en memoria a la izquierda
                ORAA  DATO                ;se realiza una función lógica OR con el bit recibido
                STAA  DATO                ;se guarda el dato nuevo en memoria
                BRA   VERIF0              ;salta a verificar el estado bajo del reloj
```

```
N_SHIFT     STAA  DATO          ;guarda el dato en memoria sin corrimiento
            BRA   VERIF0        ;salta a verificar el estado bajo del reloj

VERIF0      LDAA   PORTA,X      ;reloj bit 0 del puerto A
            ANDA   #$01         ;este ciclo se mantiene mientras el reloj siga en
            CMPA   #$00         ;estado bajo. Una vez que el reloj sube de nivel
            BEQ     VERIF0      ;el flujo continua

            INY                 ;incrementa contador
            CPY    #$21         ;compara con 33 que son los bits enviados por dato
            BEQ    G_RAM        ;salta a etiqueta
            BRA    VERIF1       ;si no, regresa otra vez

INI_1       BRA     INI_REAL

G_RAM       LDY    Y_ALTO       ;carga Y con lo que tenga Y_ALTO
            LDAA   DATO         ;carga A con el dato
            STAA   $00,Y        ;lo guarda en Y_ALTO
            CMPA   #$5A         ;compara A con #$ 5A (enter)
            BEQ     REVISA      ;salta si es igual a etiqueta revisa
            INY                 ;incrementa Y
            STY     Y_ALTO      ;guarda Y en Y_ALTO
            BRA     INICIO      ;salta al inicio

REVISA      CPY    #$0E         ;compara Y con #$0E
            BNE     N_VALID1    ;salta si no es igual a N_VALID
            LDAA    $00,Y       ;carga A con el valor que está en esa dirección
            CMPA   #$5A         ;compara A con el código del “enter” ‘5A’
            BNE     N_VALID     ;salta si no es igual a N_VALID
            LDY     #$04        ;carga Y con la dirección #$04
            LDAA    $00,Y       ;carga A con el valor que está en esa dirección
            CMPA   #$12         ;compara A con el código del “shift” ‘12’
            BNE     N_VALID     ;salta si no es igual a N_VALID
            BRA     CUENTA      ;salta a cuenta

CUENTA      LDY     #$00        ;carga Y con #$00
            LDAA    CONT,Y      ;carga A con el valor del contador
            INCA                ;incrementa A
            STAA    CONT,Y      ;guarda contador actualizado
            BRA     DISPLAY     ;salta a display

INI_2       BRA     INI_1
N_VALID1    BRA     N_VALID

DISPLAY     LDX     X_ALTO      ;carga X con X_ALTO
OTRO        LDY     #TABLA      ;carga Y con la dirección de la tabla
            LDAB    #$00        ;acumulador B = 0
            INX                 ;incrementa X
            CPX     #$0F        ;compara X con #$0F
            BEQ     ENV_ENT     ;salta si no es igual a ENV_ENT
            STX     X_ALTO      ;guarda X_ALTO
BUSCA       LDAA    $00,Y       ;carga A con el valor de la tabla
            CMPA    $00,X       ;compara A con el dato en memoria
            BEQ     ENVIA       ;si es igual salta a envia
            INY                 ;incrementa valor de la tabla
            INCB                ;incrementa valor de B
```

```
                CMPB  #$2B              ;compara B con 43 (# de datos en la tabla)
                BEQ   OTRO              ;salta a OTRO para seguir con la cadena
                BRA   BUSCA             ;si no ha acabado vuelve a buscar

ENVIA           LDY   Y_EXTH            ;Y = Y_EXTH
                ADDB  #$30              ;se suma 48 para estar en código ASCII
                STAB  $00,Y             ;guarda B en Y_EXTH
                INY                     ;incrementa Y
                STY   Y_EXTH            ;guarda Y_EXTH actualizado
                BRA   DISPLAY           ;salta a display

ENV_ENT         JSR   LCD               ;salta a subrutina LCD
                BRA   INI_2             ;salta a inicio_real

N_VALID         NOP
                BRA   INI_2             ;salta a inicio_real

*******************************************

LCD             LDAA  #$80              ;apunta al inicio de la RAM del LCD
                JSR   ENVCOM            ;salta a subrutina envcom
                LDAA  Y_EXTL            ;carga A con Y_EXTL
                SUBA  #$0C              ;resta a A #$0C
                STAA  Y_LCDL            ;guarda A en Y_LCDL
                LDY   Y_LCDH            ;carga Y con LCDH
MANDA           LDAA  $00,Y             ;carga A con Y_LCDH
                JSR   ENVDAT            ;salta a subrutina envdat
                INY                     ;incrementa Y
                CPY   Y_EXTH            ;compara Y con Y_EXTH
                BNE   MANDA             ;salta si no es igual a manda
                RTS                     ;regresa de subrutina

*******************************************

SERIE           LDX   #REGBAS           ;guarda A en el stack
                LDY   #$30
ENO             CPY   Y_EXTH
                BEQ   FINAL
LLENO           BRCLR SCSR,X,$80,LLENO
                LDAB  $00,Y
                STAB  SCDR,X
                INY
                BRA   ENO
FINAL           BRCLR SCSR,X,$80,FINAL
                LDAB  #$0D
                STAB  SCDR,X
                RTS                     ;regresa de subrutina

*******************************************

*******************************************

RET             PSHX                    ;subrutina de retardo
                LDX   #$FFFF
FGHH            NOP
                DEX
```

```
                BNE   FGHH
                PULX
                RTS

RET100          PSHX                    ;subrutina de retardo de 100 μs
                LDX   #$0015
NOPAL           NOP
                DEX
                BNE   NOPAL
                PULX
                RTS

RET10MS         PSHA                    ;subrutina de retardo de 10 ms
                LDAA  #$64
NOPON           BSR   RET100
                DECA
                BNE   NOPON
                PULA
                RTS

DELAY           PSHA                    ;subrutina de delay o retardo
                LDAA  #$02
NOPOL           BSR   RET100
                DECA
                BNE  NOPOL
                PULA
                RTS

*****************************************
                                        ;función Envía Comando
ENVCOM          PSHA                    ;guarda A en el stack
                STAA  $1004             ;escribe en puerto B, LCD
                BSR    DELAY            ;salto a subrutina delay
                LDAA  #$20              ;E=1, RS=0, modo comando
                STAA  $1000             ;escribe en puerto A
                BSR    DELAY            ;salto a subrutina delay
                LDAA  #$00              ;E=0, RS=0
                STAA  $1000             ;escribe en puerto A
                BSR    DELAY            ;salto a subrutina delay
                LDAA   #$20             ;E=1, RS=0
                STAA   $1000            ;escribe en puerto A
                BSR     DELAY           ;salto a subrutina delay
                PULA                    ;recupera el valor de A
                RTS                     ;regresa de subrutina

*****************************************
*****************************************

ENVDAT          PSHA                    ;guarda A en el stack
                STAA  $1004             ;escribe en puerto B, LCD
                BSR    DELAY            ;salta a subrutina delay
                LDAA  #$30              ;E=1, RS=1, modo datos
                STAA  $1000             ;escribe en puerto A, Enable
                BSR    DELAY            ;salta a subrutina delay
                LDAA  #$10              ;E=0, RS=1
                STAA  $1000             ;escribe en puerto A
```

```
        BSR    DELAY            ;salta a subrutina delay
        LDAA   #$30             ;E=1, RS=1
        STAA   $1000            ;escribe en puerto A
        BSR    DELAY            ;salta a subrutina delay
        PULA                    ;recupera el valor de A
        RTS                     ;regresa de subrutina

*******************************************

TABLA: FCB $45,$16,$1E,$26,$25,$2E,$36,$3D,$3E,$46,$4C,$4C,$41,$55,$49,$4A,$1E
FCB $1C,$32,$21,$23,$24,$2B,$34,$33,$43,$3B,$42,$4B,$3A,$31,$44,$4D,$15,$2D,
$1B,$2C,$3C,$2A,$1D,$22,$35,$1A
```

Apéndice C.

Código Fuente de la aplicación final en Delphi.

```
unit USerial;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  CommInt, StdCtrls, Buttons, ExtCtrls, Db, DBTables, DBCtrls, MPlayer;

type
  TFPrincipal = class(TForm)
    Comm1: TComm;
    Memo1: TMemo;
    Panel1: TPanel;
    LDatos: TStaticText;
    Table1: TTable;
    STFecha: TStaticText;
    STHora: TStaticText;
    STGrado: TStaticText;
    Label1: TLabel;
    Label2: TLabel;
    STNombre: TStaticText;
    Foto: TDBImage;
    STSituacion: TStaticText;
    Label3: TLabel;
    MediaPlayer1: TMediaPlayer;
    IEscudo: TImage;
    MediaPlayer2: TMediaPlayer;
    DataSource1: TDataSource;
    LimpiaDatos: TBitBtn;
    ProcesaInformacion: TBitBtn;
    BitBtn1: TBitBtn;
    procedure Comm1RxChar(Sender: TObject; Count: Integer);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormShow(Sender: TObject);
    procedure ProcesaInformacionClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure LimpiaDatosClick(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
  private
    LineData,
    Linea,
    Directorio : string;
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FPrincipal: TFPrincipal;

implementation
```

```
{$R *.DFM}
procedure TFPrincipal.Comm1RxChar(Sender: TObject; Count: Integer);
  type CharBuf = array[0..9999] of Char;
  var Buffer: ^CharBuf;
     Bytes, P, i, j : Integer;
  begin
    GetMem(Buffer, Comm1.ReadBufSize);
    try
       Fillchar(Buffer^, Comm1.ReadBufSize, 0);
       Bytes := Comm1.Read(Buffer^, Count);
       for P := 0 to Bytes - 1 do
         begin
           case Buffer^[P] of
              #0, #10:;
              #13: begin
                    Linea := '';
                    j     := 0;
                    For i := 1 to Length(LineData) do
                      Begin
                        Linea := Linea +  LineData[i];
                        Inc(j);
                        If j = 12
                          Then Begin
                                 Memo1.Lines.Add(Linea);
                                 j     := 0;
                                 Linea := '';
                               End;
                      End;
                    LineData := '';
                  End;
             else LineData := LineData + CharBuf(Buffer^)[P];
           end; //case
         end; //for do
       Application.ProcessMessages;
       finally
         FreeMem(Buffer);
     end;
  end;

procedure TFPrincipal.FormCreate(Sender: TObject);
  begin
    GetDir(0, Directorio);
    Directorio := Directorio + '\Bases';
    STFecha.Caption     := DateToStr(Date);
    STHora.Caption      := TimeToStr(Time);
  end;

procedure TFPrincipal.FormShow(Sender: TObject);
  begin
    Comm1.Open;
  end;
```

```
procedure TFPrincipal.FormClose(Sender: TObject; var Action: TCloseAction);
  begin

    Comm1.Close;
  end;

procedure TFPrincipal.ProcesaInformacionClick(Sender: TObject);
    Var  //Longitud  : Integer;
         Code          : Integer;
         ClaveNum      : Integer;
         ClaveText     : String[2];
         RFC           : String[10];
         OkBusca       : Boolean;
         MyErrorString,
         Texto,
         Codigo        : string;
         Situacion     : string[7];
         Tono          : String[3];
         Fecha         : String[10];
         Hora          : String[8];
         Lineas,
         i             : Integer;
    Begin
      Lineas := Memo1.Lines.Count;
      For i := 0 to Lineas - 1 do
        Begin
          Sleep(1000);
          Codigo := Memo1.Lines.Strings[i];
          ClaveText   := Copy(Codigo, 1, 2);
          Val(ClaveText, ClaveNum, Code);
          RFC := Copy(Codigo, 3, 10);
          Table1.Active       := False;
          Table1.DatabaseName := Directorio;
          Case ClaveNum of
             0 : Begin
                   Table1.TableName := '00.db';
             Texto           := 'Planta';
            end;
        1 : Begin
             Table1.TableName := '01.db';
             Texto           := 'Mando Superior';
            end;
        2 : Begin
             Table1.TableName := '02.db';
             Texto           := 'Estado Mayor General';
            end;
        3 : Begin
             Table1.TableName := '03.db';
             Texto           := 'Mando Naval';
            end;
        4 : Begin
             Table1.TableName := '04.db';
             Texto           := 'Logistica Basica';
            end;
        5 : Begin
             Table1.TableName := '05.db';
```

```
      Texto          := 'Informatica';
    end;
 6 : Begin
      Table1.TableName := '06.db';
      Texto          := 'Ingles';
    end;
 7 : Begin
      Table1.TableName := '07.db';
      Texto          := 'Frances';
    end;
 8 : Begin
      Table1.TableName := '08.db';
      Texto          := 'Aleman';
    end;
 9 : Begin
      Table1.TableName := '09.db';
      Texto          := 'Logistica Avanzada';
    end;
10 : Begin
      Table1.TableName := '10.db';
      Texto          := 'Logistica Intermedia';
    end;
11 : Begin
      Table1.TableName := '11.db';
      Texto          := 'Comunicaciones';
    end;
12 : Begin
      Table1.TableName := '12.db';
      Texto          := 'Sistemas de Armas';
    end;
13 : Begin
      Table1.TableName := '13.db';
      Texto          := 'Inteligencia';
    end;
14 : Begin
      Table1.TableName := '14.db';
      Texto          := 'Chino';
    end;
15 : Begin
      Table1.TableName := '15.db';
      Texto          := 'Coreano';
    end;
16 : Begin
      Table1.TableName := '16.db';
      Texto          := 'Ruso';
    end;
17 : Begin
      Table1.TableName := '17.db';
      Texto          := 'Cursos Especiales';
    end;
     End;
    Table1.Active := True;
    Table1.Refresh;
    OkBusca := False;
    While Not(Table1.EOF) And (OkBusca = False) do
     Try
```

```
          If Trim(UpperCase(RFC)) = Trim(UpperCase(Table1.FieldByName('RFC').AsString))
            Then OkBusca := True
            Else Table1.Next;
          except
            MyErrorString := 'Error en la base de datos: ' + Table1.TableName + #13#10
                        + 'en el registro ' + Table1.FieldByName('Nombre').AsString + #13#10;
            MessageDlg(MyErrorString, mtError, [mbOk], 0);
          End;
        If OkBusca = True
          Then Begin
               If Table1.FieldByName('Clave Situacion').AsInteger = 0
                 Then Begin
                      //MediaPlayer1.AutoOpen := False;
                      //MediaPlayer1.FileName := Directorio + '\Entrada.WAV';
                      Tono := 'Uno';
                      //MediaPlayer1.Play;
                      //MediaPlayer1.AutoOpen := True;
                      Table1.Edit;
                      Table1.FieldByName('Clave Situacion').AsInteger := 1;
                      Table1.FieldByName('Fecha de Entrada').AsDateTime :=
StrToDateTime(STFecha.Caption);
                      Table1.FieldByName('Hora de Entrada').AsDateTime :=
StrToDateTime(STHora.Caption);
                      Table1.Post;
                      Situacion := 'Entrada';
                     End
                 Else Begin
                      //Table1.FieldByName('Presente') <> 'AUSENTE' Then
                      //MediaPlayer1.AutoOpen := False;
                      //MediaPlayer1.FileName := Directorio + '\Salida.WAV';
                      //MediaPlayer1.AutoOpen := True;
                      Tono := 'Dos';
                      //MediaPlayer2.Play;
                      Table1.Edit;
                      Table1.FieldByName('Clave Situacion').AsInteger  := 0;
                      Table1.FieldByName('Fecha de Salida').AsDateTime :=
StrToDateTime(STFecha.Caption);
                      Table1.FieldByName('Hora de Salida').AsDateTime :=
StrToDateTime(STHora.Caption);
                      Table1.Post;
                      Situacion := ' Salida';
                     End;

               Fecha  := DateToStr(Date);
               Hora   := TimeToStr(Time);
               STFecha.Caption     := Fecha;
               STHora.Caption      := Hora;
               STGrado.Caption     := Table1.FieldByName('Grado').AsString;
               STNombre.Caption    := Table1.FieldByName('Nombre').AsString;
//               Foto.Picture.Bitmap := Table1.FieldByName('Fotografia').AsVariant;
               Foto.DataField   := 'Fotografia';
               Foto.Update;
               STSituacion.Caption := Situacion;
               If Tono = 'Uno'
                 Then MediaPlayer1.Play
                 Else MediaPlayer2.Play;
```

```
//                ETRFC.Text := ''
                End
            Else Begin
                LDatos.Caption := 'Personal';
//                IEscudo.Picture.LoadFromFile(Directorio + '\Planta.BMP');
                STFecha.Caption     := DateToStr(Date);
                STHora.Caption      := TimeToStr(Time);
                STGrado.Caption     := '';
                STNombre.Caption    := '';
                STSituacion.Caption := '';
//                ETRFC.Text          := '';
                Foto.DataField      := '';
                End;
            End;
            ProcesaInformacion.UpDate;
          End;

procedure TFPrincipal.LimpiaDatosClick(Sender: TObject);
  begin
    Memo1.Clear;
  end;

procedure TFPrincipal.BitBtn1Click(Sender: TObject);
  begin
    Comm1.Close;
    Application.Terminate;
  end;

end.
```

Apéndice D.

Tabla ASCII.

ASCII	HEX	SÍMBOLO
0	0	NUL
1	1	SOH
2	2	STX
3	3	ETX
4	4	EOT
5	5	ENQ
6	6	ACK
7	7	BEL
8	8	BS
9	9	TAB
10	A	LF
11	B	VT
12	C	FF
13	D	CR
14	E	SO
15	F	SI
16	10	DLE
17	11	DC1
18	12	DC2
19	13	DC3
20	14	DC4
21	15	NAK
22	16	SYN
23	17	ETB
24	18	CAN
25	19	EM
26	1ª	SUB
27	1B	ESC
28	1C	FS
29	1D	GS
30	1E	RS
31	1F	US
32	20	(space)
33	21	!
34	22	"
35	23	#

ASCII	HEX	SÍMBOLO
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G

ASCII	HEX	SÍMBOLO
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_
96	60	`
97	61	a
98	62	b
99	63	c

ASCII	HEX	SÍMBOLO
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	□

Apéndice E.

Diagramas esquemáticos de la tarjeta M68HC11EVBU.

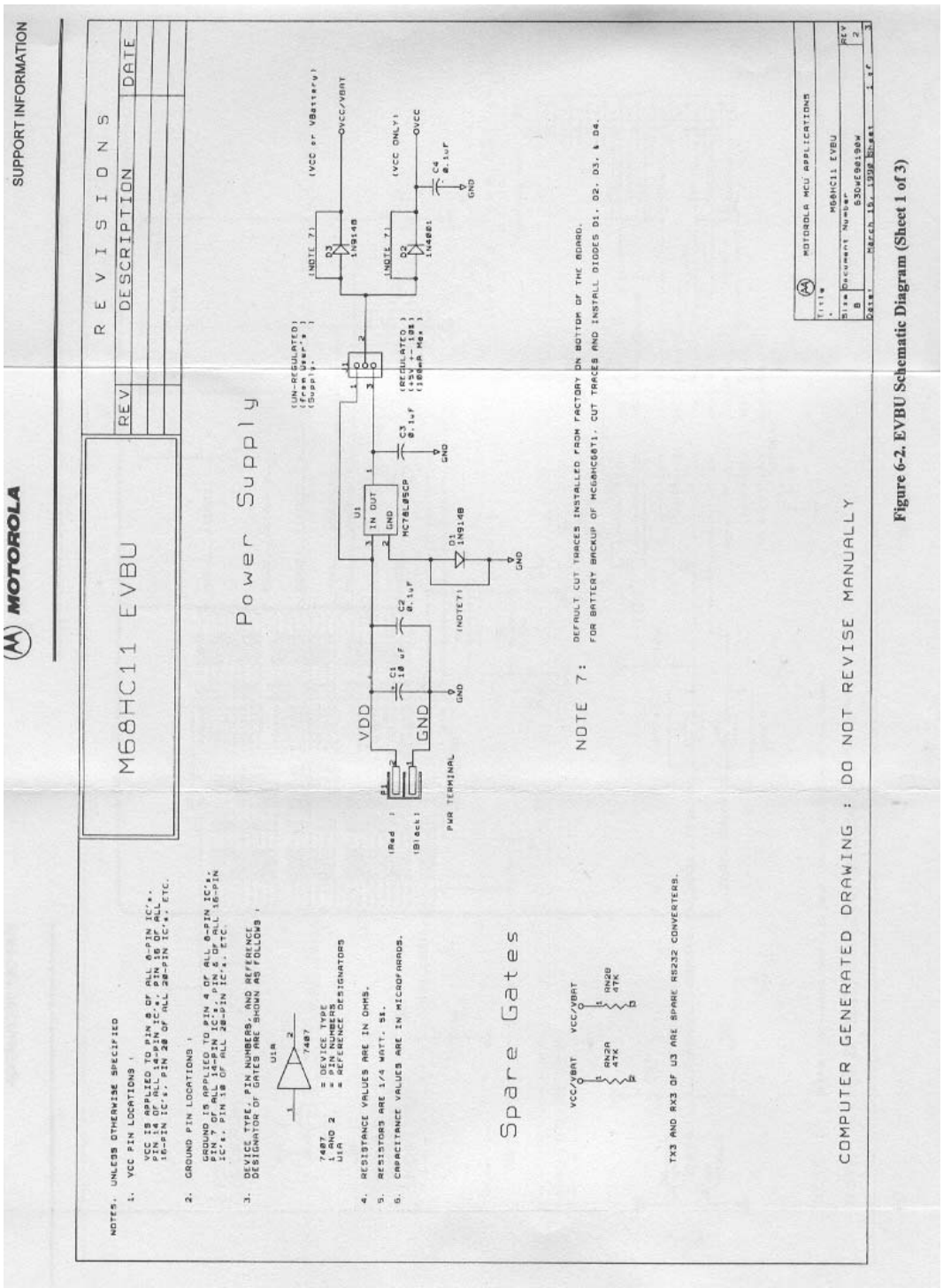


Figure 6-2. EVBU Schematic Diagram (Sheet 1 of 3)

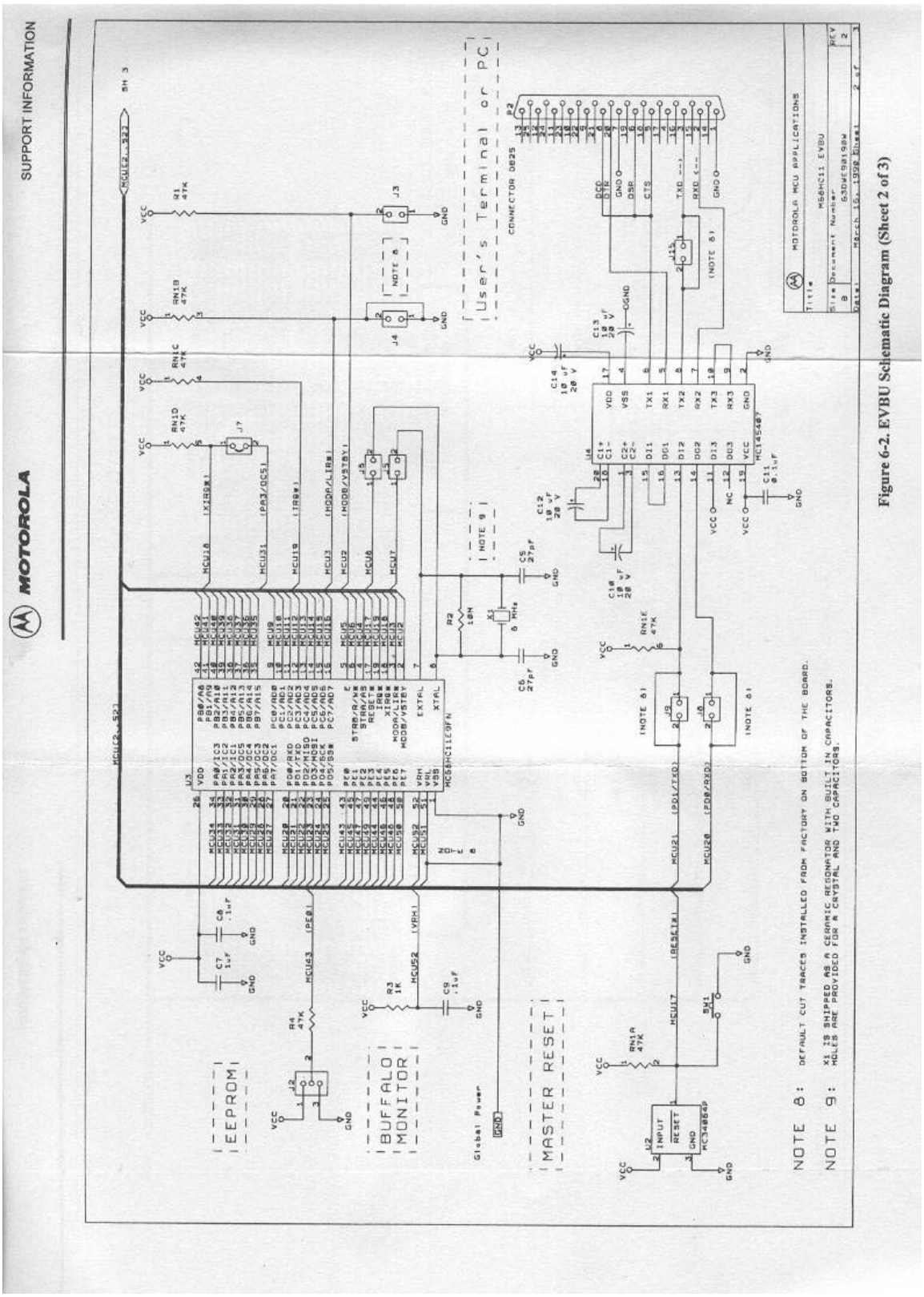


Figure 6-2. EVBU Schematic Diagram (Sheet 2 of 3)

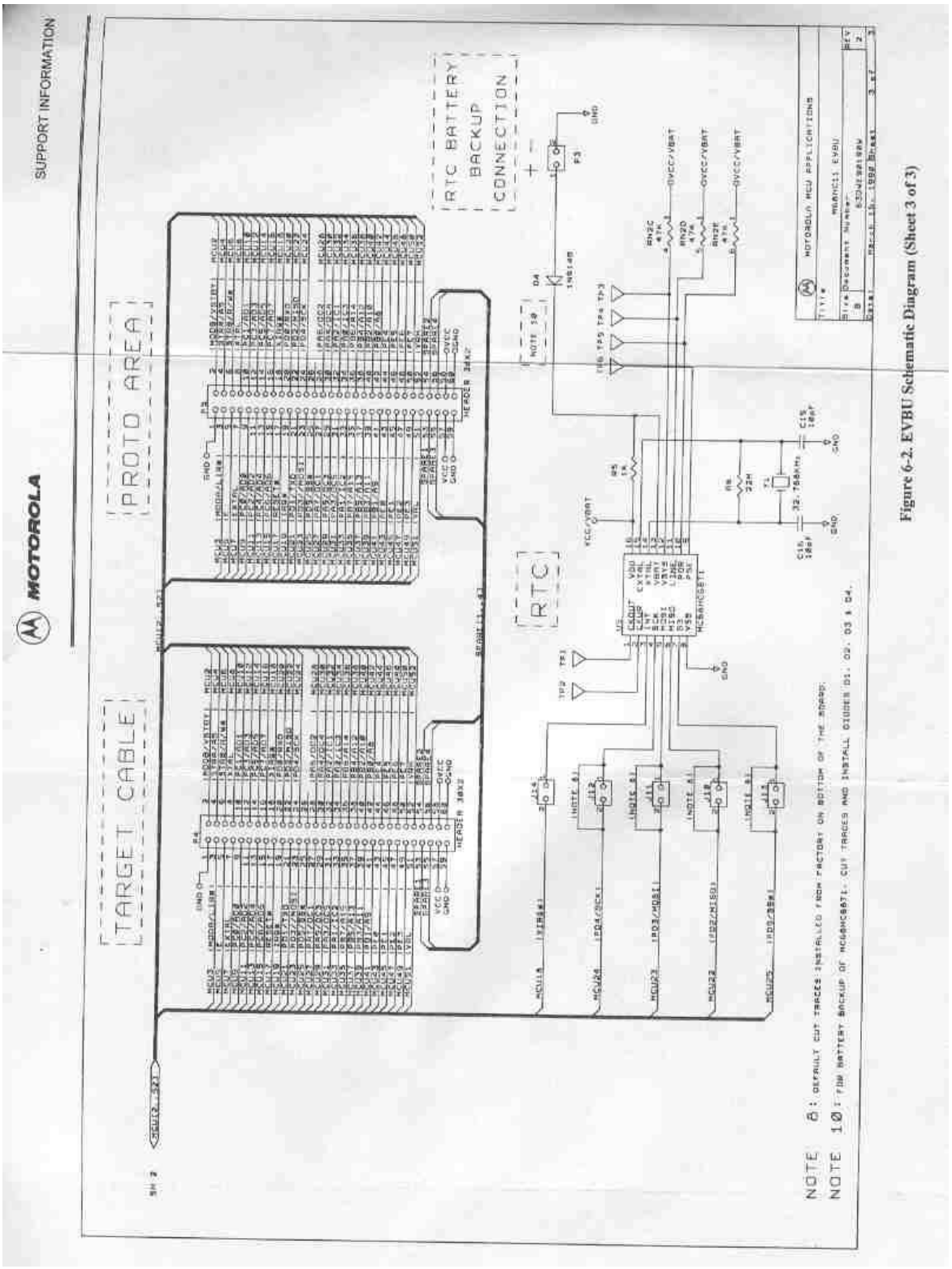


Figure 6-2. EVBU Schematic Diagram (Sheet 3 of 3)